openstack.

Architecture Design Guide

Release Version: 15.0.0

OpenStack contributors

Jun 18, 2017

CONTENTS

Note: This guide is a work in progress. Contributions are welcome.

ABSTRACT

The Architecture Design Guide provides information on planning and designing an OpenStack cloud. It explains core concepts, cloud architecture design requirements, and the design criteria of key components and services in an OpenStack cloud. The guide also describes five common cloud use cases.

Before reading this book, we recommend:

- Prior knowledge of cloud architecture and principles.
- Linux and virtualization experience.
- A basic understanding of networking principles and protocols.

For information about deploying and operating OpenStack, see the Installation Tutorials and Guides, Deployment Guides, and the OpenStack Operations Guide.

Conventions

The OpenStack documentation uses several typesetting conventions.

Notices

Notices take these forms:

Note: A comment with additional information that explains a part of the text.

Important: Something you must be aware of before proceeding.

Tip: An extra but helpful piece of practical advice.

Caution: Helpful information that prevents the user from making mistakes.

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

Command prompts

```
$ command
```

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

```
# command
```

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.

Architecture requirements

This chapter describes the enterprise and operational factors that impacts the design of an OpenStack cloud.

Enterprise requirements

The following sections describe business, usage, and performance considerations for customers which will impact cloud architecture design.

Cost

Financial factors are a primary concern for any organization. Cost considerations may influence the type of cloud that you build. For example, a general purpose cloud is unlikely to be the most cost-effective environment for specialized applications. Unless business needs dictate that cost is a critical factor, cost should not be the sole consideration when choosing or designing a cloud.

As a general guideline, increasing the complexity of a cloud architecture increases the cost of building and maintaining it. For example, a hybrid or multi-site cloud architecture involving multiple vendors and technical architectures may require higher setup and operational costs because of the need for more sophisticated orchestration and brokerage tools than in other architectures. However, overall operational costs might be lower by virtue of using a cloud brokerage tool to deploy the workloads to the most cost effective platform.

Consider the following costs categories when designing a cloud:

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

It is also important to consider how costs will increase as your cloud scales. Choices that have a negligible impact in small systems may considerably increase costs in large systems. In these cases, it is important to minimize capital expenditure (CapEx) 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 Open Compute (more information available in the Open Compute Project) provide additional information.

Time-to-market

The ability to deliver services or products within a flexible time frame is a common business factor when building a cloud. Allowing users to self-provision and gain access to compute, network, and storage resources on-demand may decrease time-to-market for new products and applications.

You must balance the time required to build a new cloud platform against the time saved by migrating users away from legacy platforms. In some cases, existing infrastructure may influence your architecture choices. For example, using multiple cloud platforms may be a good option when there is an existing investment in several applications, as it could be faster to tie the investments together rather than migrating the components and refactoring them to a single platform.

Revenue opportunity

Revenue opportunities vary based on the intent and use case of the cloud. The requirements of a commercial, customer-facing product are often very different from an internal, private cloud. You must consider what features make your design most attractive to your users.

Capacity planning and scalability

Capacity and the placement of workloads are key design considerations for clouds. A long-term capacity plan for these designs must incorporate growth over time to prevent permanent consumption of more expensive external clouds. To avoid this scenario, account for future applications' capacity requirements and plan growth appropriately.

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 use. It is possible to define application requirements in terms of vCPU, RAM, bandwidth, or other resources and plan appropriately. However, other clouds might not use the same meter or even the same oversubscription rates.

Oversubscription is a method to emulate more capacity than 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 instances do not concurrently use 2 full gigabytes, this arrangement works well. However, some hosts take oversubscription to extremes and, as a result, performance can be inconsistent. If at all possible, determine what the oversubscription rates of each host are and plan capacity accordingly.

Performance

Performance is a critical consideration when designing any cloud, and becomes increasingly important as size and complexity grow. While single-site, private clouds can be closely controlled, multi-site and hybrid deployments require more careful planning to reduce problems such as network latency between sites.

For example, you should consider the time required to run a workload in different clouds and methods for reducing this time. This may require moving data closer to applications or applications closer to the data they process, and grouping functionality so that connections that require low latency take place over a single cloud rather than spanning clouds.

This may also require a CMP that can determine which cloud can most efficiently run which types of workloads.

Using native OpenStack tools can help improve performance. For example, you can use Telemetry to measure performance and the Orchestration service (heat) to react to changes in demand.

Note: Orchestration requires special client configurations to integrate with Amazon Web Services. For other types of clouds, use CMP features.

Cloud resource deployment The cloud user expects repeatable, dependable, and deterministic processes for launching and deploying cloud resources. You could deliver this through a web-based interface or publicly available API endpoints. All appropriate options for requesting cloud resources must be available through some type of user interface, a command-line interface (CLI), or API endpoints.

Consumption model Cloud users expect a fully self-service and on-demand consumption model. When an OpenStack cloud reaches the massively scalable size, expect consumption as a service in each and every way.

- Everything must be capable of automation. For example, everything from compute hardware, storage hardware, networking hardware, to the installation and configuration of the supporting software. Manual processes are impractical in a massively scalable OpenStack design architecture.
- 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 enable operations to act on the meters provided by the metering and monitoring solutions. 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.

Location 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.

Input-Output requirements Input-Output performance requirements require researching and modeling before deciding on a final storage framework. Running benchmarks for Input-Output performance provides a baseline for expected performance levels. If these tests include details, then the resulting data can help model behavior and results during different workloads. Running scripted smaller benchmarks during the lifecycle of the architecture helps record the system health at different points in time. The data from these scripted benchmarks assist in future scoping and gaining a deeper understanding of an organization’s needs.

Scale Scaling storage solutions in a storage-focused OpenStack architecture design is driven by initial requirements, including *IOPS*, capacity, bandwidth, and future needs. Planning capacity based on projected needs over the course of a budget cycle is important for a design. The architecture should balance cost and capacity, while also allowing flexibility to implement new technologies and methods as they become available.

Network

It is important to consider the functionality, security, scalability, availability, and testability of the network when choosing a CMP and cloud provider.

- Decide on a network framework and design minimum functionality tests. This ensures testing and functionality persists during and after upgrades.
- Scalability across multiple cloud providers may dictate which underlying network framework you choose in different cloud providers. It is important to present the network API functions 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. Development of high availability and test frameworks is necessary to insure understanding of functionality and limitations.
- Consider the security of data between the client and the endpoint, and of traffic that traverses the multiple clouds.

For example, degraded video streams and low quality VoIP sessions negatively impact user experience and may lead to productivity and economic loss.

Network misconfigurations Configuring incorrect IP addresses, VLANs, and routers can cause outages to areas of the network or, in the worst-case scenario, the entire cloud infrastructure. Automate network configurations to minimize the opportunity for operator error as it can cause disruptive problems.

Capacity planning Cloud networks require management for capacity and growth over time. Capacity planning includes the purchase of network circuits and hardware that can potentially have lead times measured in months or years.

Network tuning Configure cloud networks to minimize link loss, packet loss, packet storms, broadcast storms, and loops.

Single Point Of Failure (SPOF) Consider high availability 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 can become unavoidable.

Complexity An overly complex network design can be difficult to maintain and troubleshoot. While device-level configuration can ease maintenance concerns and automated tools can handle overlay networks, avoid or document non-traditional interconnects between functions and specialized hardware 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) 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 implementation 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.

Dynamic resource expansion or bursting An application that requires additional resources may suit a multiple cloud architecture. For example, a retailer needs additional resources during the holiday season, but does not want to add private cloud resources to meet the peak demand. The user can accommodate the increased load by bursting to a public cloud for these peak load periods. These bursts could be for long or short cycles ranging from hourly to yearly.

Compliance and geo-location

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

Compliance considerations are particularly important for multi-site clouds. Considerations include:

- federal legal requirements
- local jurisdictional legal and compliance requirements
- image consistency and availability
- storage replication and availability (both block and file/object storage)
- authentication, authorization, and auditing (AAA)

Geographical considerations may also impact the cost of building or leasing data centers. Considerations include:

- floor space
- floor weight
- rack height and type

- environmental considerations
- power usage and power usage efficiency (PUE)
- physical security

Auditing

A well-considered auditing plan is essential for quickly finding 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. For more details on auditing, see the Compliance chapter in the OpenStack Security Guide.

Security

The importance of security varies based on the type of organization using a cloud. For example, government and financial institutions often have very high security requirements. Security should be implemented according to asset, threat, and vulnerability risk assessment matrices. See *security-requirements*.

Service level agreements

Service level agreements (SLA) must be developed in conjunction with business, technical, and legal input. Small, private clouds may operate under an informal SLA, but hybrid or public clouds generally require more formal agreements with their users.

For a user of a massively scalable OpenStack public cloud, there are no expectations for control over security, performance, or availability. Users expect only SLAs related to uptime of API services, and very basic SLAs for services offered. It is the user's 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.

High performance systems have SLA requirements for a minimum quality of service 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.

Hybrid cloud designs must accommodate differences in SLAs between providers, and consider their enforceability.

Application readiness

Some applications are tolerant of a lack of synchronized object storage, while others may need those objects to be replicated and available across regions. Understanding 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 or rewritten for an infrastructure with little to no redundancy, or with the cloud in mind.

Application momentum Businesses with existing applications may find that it is more cost effective to integrate applications on multiple cloud platforms than migrating them to a single platform.

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.

Authentication

It is recommended to have a single authentication domain rather than a separate implementation for each and every site. This requires an authentication mechanism that is highly available and distributed to ensure continuous operation. Authentication server locality might be required and should be planned for.

Migration, availability, site loss and recovery

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

- The deployed applications need to continue to function and, more importantly, you must consider the impact on the performance and reliability of the application when a site is unavailable.
- It is important to understand what happens to the replication of objects and data between the sites when a site goes down. If this causes queues to start building up, consider how long these queues can safely exist until an error occurs.
- After an outage, ensure the method for resuming proper operations of a site is implemented when it comes back online. We recommend you architect the recovery to avoid race conditions.

Disaster recovery and business continuity Cheaper storage makes the public cloud suitable for maintaining backup applications.

Migration scenarios Hybrid cloud architecture enables the migration of applications between different clouds.

Provider availability or implementation details Business changes can affect provider availability. Likewise, changes in a provider's service can disrupt a hybrid cloud environment or increase costs.

Provider API changes Consumers of external clouds rarely have control over provider changes to APIs, and changes can break compatibility. Using only the most common and basic APIs can minimize potential conflicts.

Image portability As of the Kilo release, there is no common image format that is usable by all clouds. Conversion or recreation of images is necessary if migrating between clouds. To simplify deployment, use the smallest and simplest images feasible, install only what is necessary, and use a deployment manager such as Chef or Puppet. Do not use golden images to speed up the process unless you repeatedly deploy the same images on the same cloud.

API differences Avoid using a hybrid cloud deployment with more than just OpenStack (or with different versions of OpenStack) as API changes can cause compatibility issues.

Business or technical diversity Organizations leveraging cloud-based services can embrace business diversity and utilize a hybrid cloud design to spread their workloads across multiple cloud providers. This ensures that no single cloud provider is the sole host for an application.

Operational requirements

This section describes operational factors affecting the design of an OpenStack cloud.

Network design

The network design for an OpenStack cluster includes decisions regarding the interconnect needs within the cluster, the need to allow clients to access their resources, and the access requirements for operators to administrate the cluster. You should consider the bandwidth, latency, and reliability of these networks.

Consider additional design decisions about monitoring and alarming. If you are using an external provider, service level agreements (SLAs) are typically defined in your contract. Operational considerations such as bandwidth, latency, and jitter can be part of the SLA.

As demand for network resources increase, make sure your network design accommodates expansion and upgrades. Operators add additional IP address blocks and add additional bandwidth capacity. In addition, consider managing hardware and software lifecycle events, for example upgrades, decommissioning, and outages, while avoiding service interruptions for tenants.

Factor maintainability 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 you may need to change all of the IP addresses on a network, a process known as renumbering, then the design must support this function.

Address network-focused applications when considering certain operational realities. For example, consider the impending exhaustion of IPv4 addresses, the migration to IPv6, and the use 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. We recommend you avoid relying on IPv4 features that did not carry over to the IPv6 protocol or have differences in implementation.

To segregate traffic, allow applications to create a private tenant network for database and storage network traffic. Use a public network for services that require direct client access from the Internet. Upon segregating the traffic, consider *quality of service (QoS)* and security to ensure each network has the required level of service.

Also consider the routing of network traffic. For some applications, develop a complex policy framework for routing. To create a routing policy that satisfies business requirements, consider the economic cost of transmitting traffic over expensive links versus cheaper links, in addition to bandwidth, latency, and jitter requirements.

Finally, consider how to respond to network events. How load transfers from one link to another during a failure scenario could be a factor in the design. If you do not plan network capacity 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 all network traffic stops.

SLA considerations

Service-level agreements (SLAs) define the levels of availability that will impact the design of an OpenStack cloud to provide redundancy and high availability.

SLA terms that affect the design include:

- API availability guarantees implying multiple infrastructure services and highly available load balancers.
- Network uptime guarantees affecting switch design, which might require redundant switching and power.
- Networking security policy requirements.

In any environment larger than just a few hosts, there are two areas that might be subject to a SLA:

- Data Plane - services that provide virtualization, networking, and storage. Customers usually require these services to be continuously available.
- Control Plane - ancillary services such as API endpoints, and services that control CRUD operations. The services in this category are usually subject to a different SLA expectation and may be better suited on separate hardware or containers from the Data Plane services.

To effectively run cloud installations, initial downtime planning includes creating processes and architectures that support planned maintenance and unplanned system faults.

It is important to determine as part of the SLA negotiation which party is responsible for monitoring and starting up the Compute service instances if an outage occurs.

Upgrading, patching, and changing configuration items may require downtime for some services. Stopping services that form the Control Plane may not impact the Data Plane. Live-migration of Compute instances may be required to perform any actions that require downtime to Data Plane components.

There are many services outside the realms of pure OpenStack code which affects the ability of a cloud design to meet SLAs, including:

- Database services, such as `MySQL` or `PostgreSQL`.
- Services providing RPC, such as `RabbitMQ`.
- External network attachments.
- Physical constraints such as power, rack space, network cabling, etc.
- Shared storage including SAN based arrays, storage clusters such as `Ceph`, and/or NFS services.

Depending on the design, some network service functions may fall into both the Control and Data Plane categories. For example, the neutron L3 Agent service may be considered a Control Plane component, but the routers themselves would be a Data Plane component.

In a design with multiple regions, the SLA would also need to take into consideration the use of shared services such as the Identity service and Dashboard.

Any SLA negotiation must also take into account the reliance on third parties for critical aspects of the design. For example, if there is an existing SLA on a component such as a storage system, the SLA must take into account this limitation. If the required SLA for the cloud exceeds the agreed uptime levels of the cloud components, additional redundancy would be required. This consideration is critical in a hybrid cloud design, where there are multiple third parties involved.

Support and maintenance

An operations staff supports, manages, and maintains an OpenStack environment. Their skills may be specialized or varied depending on the size and purpose of the installation.

The maintenance function of an operator should be taken into consideration:

Maintenance tasks Operating system patching, hardware/firmware upgrades, and datacenter related changes, as well as minor and release upgrades to OpenStack components are all ongoing operational tasks. The six monthly release cycle of the OpenStack projects needs to be considered as part of the cost of ongoing maintenance. The solution should take into account storage and network maintenance and the impact on underlying workloads.

Reliability and availability Reliability and availability depend on the many supporting components' availability and on the level of precautions taken by the service provider. This includes network, storage systems, datacenter, and operating systems.

For more information on managing and maintaining your OpenStack environment, see the OpenStack Operations Guide.

Logging and monitoring

OpenStack clouds require appropriate monitoring platforms to identify and manage errors.

Note: We recommend leveraging existing monitoring systems to see if they are able to effectively monitor an OpenStack environment.

Specific meters that are critically important to capture include:

- Image disk utilization
- Response time to the Compute API

Logging and monitoring does not significantly differ for a multi-site OpenStack cloud. The tools described in the Logging and monitoring in the Operations Guide remain applicable. Logging and monitoring can be provided 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 the load placed on the inter-site networking links

Management software

Management software providing clustering, logging, monitoring, and alerting details for a cloud environment is often used. This impacts and affects the overall OpenStack cloud design, and must account for the additional resource consumption such as CPU, RAM, storage, and network bandwidth.

The inclusion of clustering software, such as Corosync or Pacemaker, 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.

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

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

MySQL is the default database for OpenStack, but other compatible databases are available.

Note: Telemetry uses MongoDB.

The chosen high availability database solution changes according to the selected database. MySQL, for example, provides several options. Use a replication technology such as Galera for active-active clustering. For active-passive use some form of shared storage. Each of these potential solutions has an impact on the design:

- Solutions that employ Galera/MariaDB require at least three MySQL nodes.
- MongoDB has its own design considerations for high availability.
- OpenStack design, generally, does not include shared storage. However, for some high availability designs, certain components might require it depending on the specific implementation.

Operator access to systems

There is a trend for cloud operations systems being hosted within the cloud environment. Operators require access to these systems to resolve a major incident.

Ensure that the network structure connects all clouds to form an integrated system. Also consider the state of handoffs which must be reliable and have minimal latency for optimal performance of the system.

If a significant portion of the cloud is on externally managed systems, prepare for situations where it may not be possible to make changes. Additionally, cloud providers may differ on how infrastructure must be managed and exposed. This can lead to delays in root cause analysis where a provider insists the blame lies with the other provider.

High availability

Data plane and control plane

When designing an OpenStack cloud, it is important to consider the needs dictated by the *Service Level Agreement (SLA)*. This includes the core services required to maintain availability of running Compute service instances, networks, storage, and additional services running on top of those resources. These services are often referred to as the Data Plane services, and are generally expected to be available all the time.

The remaining services, responsible for create, read, update and delete (CRUD) operations, metering, monitoring, and so on, are often referred to as the Control Plane. The SLA is likely to dictate a lower uptime requirement for these services.

The services comprising an OpenStack cloud have a number of requirements that you need to understand in order to be able to meet SLA terms. For example, in order to provide the Compute service a minimum of storage, message queueing and database services are necessary as well as the networking between them.

Ongoing maintenance operations are made much simpler if there is logical and physical separation of Data Plane and Control Plane systems. It then becomes possible to, for example, reboot a controller without affecting customers. If one service failure affects the operation of an entire server (`noisy neighbor`), the separation between Control and Data Planes enables rapid maintenance with a limited effect on customer operations.

Eliminating single points of failure within each site

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.

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

There are a small number of OpenStack services which are intended to only run in one place at a time (for example, the `ceilometer-agent-central` service) . In order to prevent these services from becoming a single point of failure, they can be controlled by clustering software such as `Pacemaker`.

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.

Care must be taken when deciding network functionality. Currently, OpenStack supports both the legacy networking (nova-network) system and the newer, extensible OpenStack Networking (neutron). OpenStack Networking and legacy networking 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.

When using the Networking service, the OpenStack controller servers or separate Networking hosts handle routing unless the dynamic virtual routers pattern for routing is selected. Running routing directly on the controller servers mixes the Data and Control Planes and can cause complex issues with performance and troubleshooting. It is possible to use third party software and external appliances that help maintain highly available layer three routes. Doing so allows for common application endpoints 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 layer three routing.

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 if a compute host fails, that instance and any data local to that instance will be deleted. However, when providing an expectation to users that instances have a high-level of uptime guaranteed, the infrastructure must be deployed in a way that eliminates any single point of failure if 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.

If using a storage design that includes shared access to centralized storage, ensure that this is also designed without single points of failure and the SLA for the solution matches or exceeds the expected SLA for the Data Plane.

Eliminating single points of failure in a multi-region design

Some services are commonly shared between multiple regions, including the Identity service and the Dashboard. In this case, it is necessary to ensure that the databases backing the services are replicated, and that access to multiple workers across each site can be maintained in the event of losing a single region.

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: If the data store is highly changeable, the network requirements could have a significant effect on the operational cost of maintaining the sites.

If the design incorporates more than one site, the ability to maintain object availability in both sites has significant implications on the Object Storage design and implementation. It also has a significant impact on the WAN network design between the sites.

If applications running in a cloud are not cloud-aware, there should be clear measures and expectations to define what the infrastructure can and cannot support. 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.

Connecting more than two sites increases the challenges and adds more complexity to the design considerations. Multi-site implementations require planning to address the additional topology used for internal and external connectivity. Some options include full mesh topology, hub spoke, spine leaf, and 3D Torus.

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

Site loss and recovery

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

- The deployed applications need to continue to function and, more importantly, you must consider the impact on the performance and reliability of the application if a site is unavailable.
- It is important to understand what happens to the replication of objects and data between the sites when a site goes down. If this causes queues to start building up, consider how long these queues can safely exist until an error occurs.
- After an outage, ensure that operations of a site are resumed when it comes back online. We recommend that you architect the recovery to avoid race conditions.

Replicating inter-site data

Traditionally, replication has been the best method of protecting object store implementations. A variety of replication methods exist in storage architectures, for example synchronous and asynchronous mirroring. Most object stores and back-end storage systems implement methods for replication at the storage subsystem layer. Object stores also tailor replication techniques to fit a cloud's requirements.

Organizations must find the right balance between data integrity and data availability. Replication strategy may also influence disaster recovery methods.

Replication across different racks, data centers, and geographical regions increases focus on 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.

Note: When running embedded object store methods, ensure that you do not instigate extra data replication as this may cause performance issues.

Design

Designing an OpenStack cloud requires a understanding of the cloud user's requirements and needs to determine the best possible configuration. This chapter provides guidance on the decisions you need to make during the design process.

To design, deploy, and configure OpenStack, administrators must understand the logical architecture. OpenStack modules are one of the following types:

Daemon Runs as a background process. On Linux platforms, a daemon is usually installed as a service.

Script Installs a virtual environment and runs tests.

Command-line interface (CLI) Enables users to submit API calls to OpenStack services through commands.

OpenStack Logical Architecture shows one example of the most common integrated services within OpenStack and how they interact with each other. End users can interact through the dashboard, CLIs, and APIs. All services authenticate through a common Identity service, and individual services interact with each other through public APIs, except where privileged administrator commands are necessary.

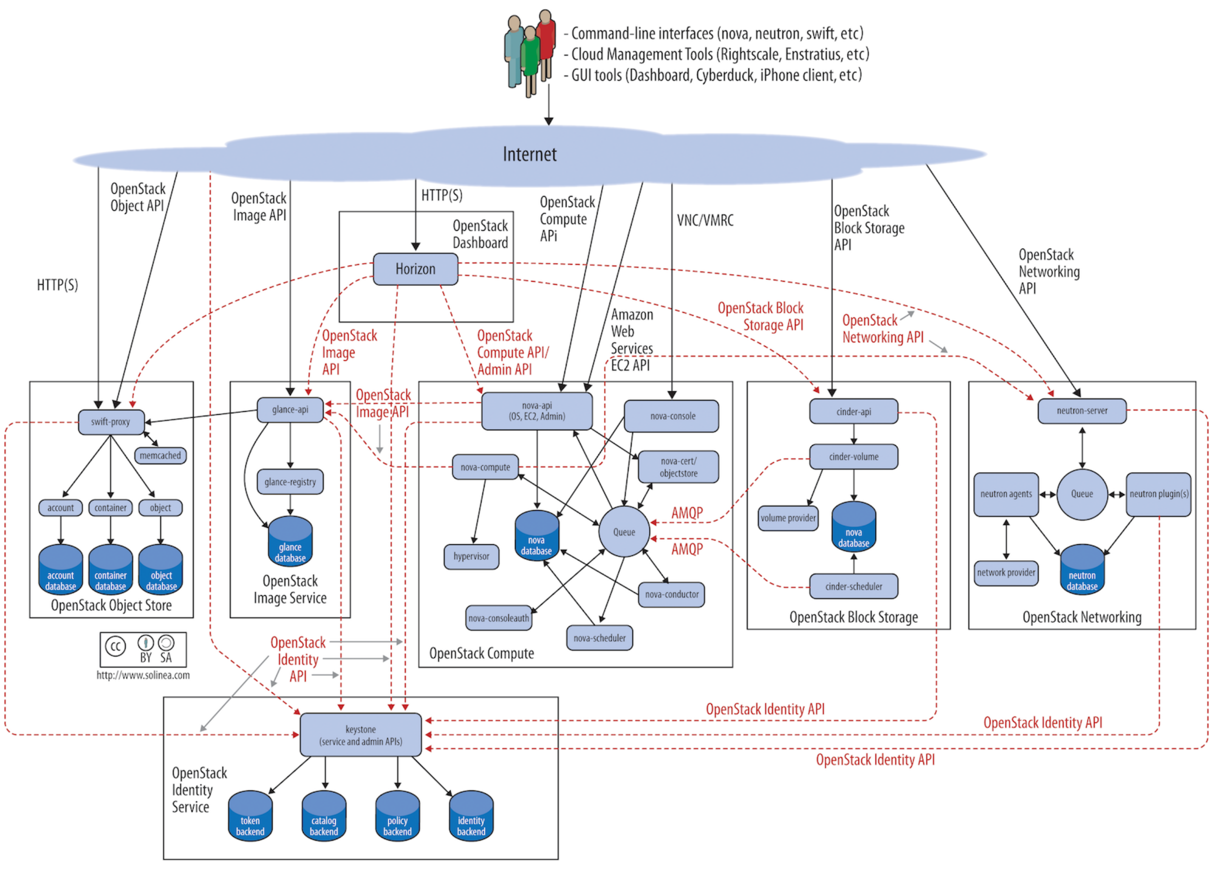


Fig. 1: OpenStack Logical Architecture

Compute node design

Compute server architecture overview

When designing compute resource pools, consider the number of processors, amount of memory, network requirements, the quantity of storage required for each hypervisor, and any requirements for bare metal hosts provisioned through ironic.

When architecting an OpenStack cloud, as part of the planning process, you must not only determine what hardware to utilize but whether compute resources will be provided in a single pool or in multiple pools or availability zones. You should consider if the cloud will provide distinctly different profiles for compute.

For example, CPU, memory or local storage based compute nodes. For NFV or HPC based clouds, there may even be specific network configurations that should be reserved for those specific workloads on specific compute nodes. This method of designing specific resources into groups or zones of compute can be referred to as bin packing.

Note: In a bin packing design, each independent resource pool provides service for specific flavors. Since instances are scheduled onto compute hypervisors, each independent node's resources will be allocated to efficiently use the available hardware. While bin packing can separate workload specific resources onto individual servers, bin packing also requires a common hardware design, with all hardware nodes within a compute resource pool sharing a common processor, memory, and storage layout. This makes it easier to deploy, support, and maintain nodes throughout their lifecycle.

Increasing the size of the supporting compute environment increases the network traffic and messages, adding load to the controllers and administrative services used to support the OpenStack cloud or networking nodes. When considering hardware for controller nodes, whether using the monolithic controller design, where all of the controller services live on one or more physical hardware nodes, or in any of the newer shared nothing control plane models, adequate resources must be allocated and scaled to meet scale requirements. Effective monitoring of the environment will help with capacity decisions on scaling. Proper planning will help avoid bottlenecks and network oversubscription as the cloud scales.

Compute nodes automatically attach to OpenStack clouds, resulting in a horizontally scaling process when adding extra compute capacity to an OpenStack cloud. To further group compute nodes and place nodes into appropriate availability zones and host aggregates, additional work is required. It is necessary to plan rack capacity and network switches as scaling out compute hosts directly affects data center infrastructure resources as would any other infrastructure expansion.

While not as common in large enterprises, compute host components can also be upgraded to account for increases in demand, 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. Since OpenStack schedules workload placement based on capacity and technical requirements, removing compute nodes from availability and upgrading them using a rolling upgrade design.

When selecting a processor, compare features and performance characteristics. Some processors include features specific to virtualized compute hosts, such as hardware-assisted virtualization, and technology related to memory paging (also known as EPT shadowing). These types of features can have a significant impact on the performance of your virtual machine.

The number of processor cores and threads impacts the number of worker threads which can be run on a resource node. Design decisions must relate directly to the service being run on it, as well as provide a balanced infrastructure for all services.

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. This ratio is configurable for CPU and memory. The default CPU overcommit ratio is 16:1, and the default memory overcommit ratio is 1.5:1. Determining the tuning of the overcommit ratios during the design phase is important as it has a direct impact on the hardware layout of your compute nodes.

Note: Changing the CPU overcommit ratio can have a detrimental effect and cause a potential increase in a noisy neighbor.

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 includes adding disk shelves to enterprise storage systems or installing additional Block Storage nodes. Upgrading directly attached storage installed in Compute hosts, and adding capacity to the shared storage for additional ephemeral storage to instances, may be necessary.

Consider the Compute requirements of non-hypervisor nodes (also referred to as resource nodes). This includes controller, Object Storage nodes, Block Storage nodes, and networking services.

The ability to create pools or availability zones for unpredictable workloads should be considered. In some cases, the demand for certain instance types or flavors may not justify individual hardware design. Allocate hardware designs that are capable of servicing the most common instance requests. Adding hardware to the overall architecture can be done later.

Choosing a CPU

The type of CPU in your compute node is a very important decision. You must ensure that the CPU supports virtualization by way of *VT-x* for Intel chips and *AMD-v* for AMD chips.

Tip: Consult the vendor documentation to check for virtualization support. For Intel CPUs, see Does my processor support Intel® Virtualization Technology?. For AMD CPUs, see AMD Virtualization. Your CPU may support virtualization but it may be disabled. Consult your BIOS documentation for how to enable CPU features.

The number of cores that the CPU has also affects your decision. It is common for current CPUs to have up to 24 cores. Additionally, if an Intel CPU supports hyper-threading, those 24 cores are doubled to 48 cores. If you purchase a server that supports multiple CPUs, the number of cores is further multiplied.

As of the Kilo release, key enhancements have been added to the OpenStack code to improve guest performance. These improvements allow the Compute service to take advantage of greater insight into a compute host's physical layout and therefore make smarter decisions regarding workload placement. Administrators can use this functionality to enable smarter planning choices for use cases like NFV (Network Function Virtualization) and HPC (High Performance Computing).

Considering non-uniform memory access (NUMA) is important when selecting CPU sizes and types, as there are use cases that use NUMA pinning to reserve host cores for operating system processes. These reduce the available CPU for workloads and protects the operating system.

Tip: When CPU pinning is requested for for a guest, it is assumed there is no overcommit (or, an overcommit ratio of 1.0). When dedicated resourcing is not requested for a workload, the normal overcommit ratios are applied.

Therefore, we recommend that host aggregates are used to separate not only bare metal hosts, but hosts that will provide resources for workloads that require dedicated resources. This said, when workloads are provisioned to NUMA host aggregates, NUMA nodes are chosen at random and vCPUs can float across NUMA nodes on a host. If workloads require SR-IOV or DPDK, they should be assigned to a NUMA node aggregate with hosts that supply the functionality. More importantly, the workload or vCPUs that are executing processes for a workload should be on the same NUMA node due to the limited amount of cross-node memory bandwidth. In all cases, the `NUMATopologyFilter` must be enabled for `nova-scheduler`.

Additionally, CPU selection may not be one-size-fits-all across enterprises, but more of a list of SKUs that are tuned for the enterprise workloads.

For more information about NUMA, see CPU topologies in the Administrator Guide.

In order to take advantage of these new enhancements in the Compute service, compute hosts must be using NUMA capable CPUs.

Tip: Multithread Considerations

Hyper-Threading is Intel's proprietary simultaneous multithreading implementation used to improve parallelization on their CPUs. You might consider enabling Hyper-Threading to improve the performance of multithreaded applications.

Whether you should enable Hyper-Threading on your CPUs depends upon your use case. For example, disabling Hyper-Threading can be beneficial in intense computing environments. We recommend performance testing with your local workload with both Hyper-Threading on and off to determine what is more appropriate in your case.

In most cases, hyper-threading CPUs can provide a 1.3x to 2.0x performance benefit over non-hyper-threaded CPUs depending on types of workload.

Choosing a hypervisor

A hypervisor provides software to manage virtual machine access to the underlying hardware. The hypervisor creates, manages, and monitors virtual machines. OpenStack Compute (nova) supports many hypervisors to various degrees, including:

- KVM
- LXC
- QEMU
- VMware ESX/ESXi
- Xen
- Hyper-V
- Docker

An important factor in your choice of hypervisor is your current organization's hypervisor usage or experience. Also important is the hypervisor's feature parity, documentation, and the level of community experience.

As per the recent OpenStack user survey, KVM is the most widely adopted hypervisor in the OpenStack community. Besides KVM, there are many deployments that run other hypervisors such as LXC, VMware, Xen, and

Hyper-V. However, these hypervisors are either less used, are niche hypervisors, or have limited functionality compared to more commonly used hypervisors.

Note: It is also possible to run multiple hypervisors in a single deployment using host aggregates or cells. However, an individual compute node can run only a single hypervisor at a time.

For more information about feature support for hypervisors as well as ironic and Virtuozzo (formerly Parallels), see Hypervisor Support Matrix and Hypervisors in the Configuration Reference.

Choosing server hardware

Consider the following factors when selecting compute server hardware:

- **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 delivers.
- **Expandability** The number of additional resources you can add to a server before it reaches capacity.
- Cost The relative cost of the hardware weighed against the total amount of capacity available on the hardware based on predetermined requirements.

Weigh these considerations against each other to determine the best design for the desired purpose. For example, increasing server density means sacrificing resource capacity or expandability. It also can decrease availability and increase the chance of noisy neighbor issues. Increasing resource capacity and expandability can increase cost but decrease server density. Decreasing cost often means decreasing supportability, availability, server density, resource capacity, and expandability.

Determine the requirements for the cloud prior to constructing the cloud, and plan for hardware lifecycles, and expansion and new features that may require different hardware.

If the cloud is initially built with near end of life, but cost effective hardware, then the performance and capacity demand of new workloads will drive the purchase of more modern hardware. With individual hardware components changing over time, you may prefer to manage configurations as stock keeping units (SKU)s. This method provides an enterprise with a standard configuration unit of compute (server) that can be placed in any IT service manager or vendor supplied ordering system that can be triggered manually or through advanced operational automations. This simplifies ordering, provisioning, and activating additional compute resources. For example, there are plug-ins for several commercial service management tools that enable integration with hardware APIs. These configure and activate new compute resources from standby hardware based on a standard configurations. Using this methodology, spare hardware can be ordered for a datacenter and provisioned based on capacity data derived from OpenStack Telemetry.

Compute capacity (CPU cores and RAM capacity) is a secondary consideration for selecting server hardware. The required server hardware must supply adequate CPU sockets, additional CPU cores, and adequate RA. For more information, see *Choosing a CPU*.

In compute server architecture design, you must also consider network and storage requirements. For more information on network considerations, see *Networking*.

Considerations when choosing hardware

Here are some other factors to consider when selecting hardware for your compute servers.

Instance density

More hosts are required to support the anticipated scale if the design architecture uses dual-socket hardware designs.

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

Host density

Another option to address the higher host count is to use a quad-socket platform. Taking this approach decreases host density which also increases rack count. This configuration affects the number of power connections and also impacts network and cooling requirements.

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 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.

Data centers have a specified amount of power fed to a given rack or set of racks. Older data centers may have power densities as low as 20A per rack, and current data centers can be designed to support power densities as high as 120A per rack. The selected server hardware must take power density into account.

Selecting hardware form factor

Consider the following in selecting server hardware form factor suited for your OpenStack design architecture:

- Most blade servers can support dual-socket multi-core CPUs. To avoid this CPU limit, select `full width` or `full height` blades. Be aware, however, that this also decreases server density. For example, high density blade servers such as HP BladeSystem or Dell PowerEdge M1000e support up to 16 servers in only ten rack units. Using half-height blades is twice as dense as using full-height blades, which results in only eight servers per ten rack units.
- 1U rack-mounted servers have the ability to offer greater server density than a blade server solution, but are often limited to dual-socket, multi-core CPU configurations. It is possible to place forty 1U servers in a rack, providing space for the top of rack (ToR) switches, compared to 32 full width blade servers.

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.

Warning: 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 that 1U rack-mounted servers offer).
- 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 are often more expensive.
- `Sled servers` are rack-mounted servers that support multiple independent servers in a single 2U or 3U enclosure. These deliver higher 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 eight CPU sockets in 2U.

Scaling your cloud

When designing a OpenStack cloud compute server architecture, you must decide whether you intend to scale up or scale out. Selecting a 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. Typically, the scale out model has been popular for OpenStack because it reduces the number of possible failure domains by spreading workloads across more infrastructure. However, the downside is the cost of additional servers and the datacenter resources needed to power, network, and cool the servers.

Overcommitting CPU and RAM

OpenStack allows you to overcommit CPU and RAM on compute nodes. This allows you to increase the number of instances running on your cloud at the cost of reducing the performance of the instances. The Compute service 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.

The formula for the number of virtual instances on a compute node is `(OR*PC)/VC`, where:

OR CPU overcommit ratio (virtual cores per physical core)

PC Number of physical cores

VC Number of virtual cores per instance

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).

Note: Regardless of the overcommit ratio, an instance can not be placed on any physical node with fewer raw (pre-overcommit) resources than the instance flavor requires.

You must select the appropriate CPU and RAM allocation ratio for your particular use case.

Instance storage solutions

As part of the architecture design for a compute cluster, you must specify storage for the disk on which the instantiated instance runs. There are three main approaches to providing temporary storage:

- Off compute node storage—shared file system
- On compute node storage—shared file system
- On compute node storage—nonshared file system

In general, the questions you should ask when selecting storage are as follows:

- What are my workloads?
- Do my workloads have IOPS requirements?
- Are there read, write, or random access performance requirements?
- What is my forecast for the scaling of storage for compute?
- What storage is my enterprise currently using? Can it be re-purposed?
- How do I manage the storage operationally?

Many operators use separate compute and storage hosts instead of a hyperconverged solution. Compute services and storage services have different requirements, and compute hosts typically require more CPU and RAM than storage hosts. Therefore, for a fixed budget, it makes sense to have different configurations for your compute nodes and your storage nodes. Compute nodes will be invested in CPU and RAM, and storage nodes will be invested in block storage.

However, if you are more restricted in the number of physical hosts you have available for creating your cloud and you want to be able to dedicate as many of your hosts as possible to running instances, it makes sense to run compute and storage on the same machines or use an existing storage array that is available.

The three main approaches to instance storage are provided in the next few sections.

Non-compute node based shared file system

In this option, the disks storing the running instances are hosted in servers outside of the compute nodes.

If you use separate compute and storage hosts, you can treat your compute hosts as "stateless". As long as you do not have any instances currently running on a compute host, you can take it offline or wipe it completely without having any effect on the rest of your cloud. This simplifies maintenance for the compute hosts.

There are several advantages to this approach:

- If a compute node fails, instances are usually easily recoverable.

- Running a dedicated storage system can be operationally simpler.
- You can scale to any number of spindles.
- It may be possible to share the external storage for other purposes.

The main disadvantages to this approach are:

- Depending on design, heavy I/O usage from some instances can affect unrelated instances.
- Use of the network can decrease performance.
- Scalability can be affected by network architecture.

On compute node storage—shared file system

In this option, each compute node is specified with a significant amount of disk space, but a distributed file system ties the disks from each compute node into a single mount.

The main advantage of this option is that it scales to external storage when you require additional storage.

However, this option has several disadvantages:

- Running a distributed file system can make you lose your data locality compared with nonshared storage.
- Recovery of instances is complicated by depending on multiple hosts.
- The chassis size of the compute node can limit the number of spindles able to be used in a compute node.
- Use of the network can decrease performance.
- Loss of compute nodes decreases storage availability for all hosts.

On compute node storage—nonshared file system

In this option, each compute node is specified with enough disks to store the instances it hosts.

There are two main advantages:

- Heavy I/O usage on one compute node does not affect instances on other compute nodes. Direct I/O access can increase performance.
- Each host can have different storage profiles for hosts aggregation and availability zones.

There are several disadvantages:

- If a compute node fails, the data associated with the instances running on that node is lost.
- The chassis size of the compute node can limit the number of spindles able to be used in a compute node.
- Migrations of instances from one node to another are more complicated and rely on features that may not continue to be developed.
- If additional storage is required, this option does not scale.

Running a shared file system on a storage system apart from the compute nodes is ideal for clouds where reliability and scalability are the most important factors. Running a shared file system on the compute nodes themselves may be best in a scenario where you have to deploy to pre-existing servers for which you have little to no control over their specifications or have specific storage performance needs but do not have a need for persistent storage.

Issues with live migration

Live migration is an integral part of the operations of the cloud. This feature provides the ability to seamlessly move instances from one physical host to another, a necessity for performing upgrades that require reboots of the compute hosts, but only works well with shared storage.

Live migration can also be done with non-shared storage, using a feature known as *KVM live block migration*. While an earlier implementation of block-based migration in KVM and QEMU was considered unreliable, there is a newer, more reliable implementation of block-based live migration as of the Mitaka release.

Live migration and block migration still have some issues:

- Error reporting has received some attention in Mitaka and Newton but there are improvements needed.
- Live migration resource tracking issues.
- Live migration of rescued images.

Choice of file system

If you want to support shared-storage live migration, you need to configure a distributed file system.

Possible options include:

- NFS (default for Linux)
- GlusterFS
- MooseFS
- Lustre

We recommend that you choose the option operators are most familiar with. NFS is the easiest to set up and there is extensive community knowledge about it.

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.

The selection of form factors or architectures affects the selection of server hardware. 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. Similarly, the network architecture impacts the server hardware selection and vice versa.

While each enterprise install is different, the following networks with their proposed bandwidth is highly recommended for a basic production OpenStack install.

Install or OOB network - Typically used by most distributions and provisioning tools as the network for deploying base software to the OpenStack compute nodes. This network should be connected at a minimum of 1Gb and no routing is usually needed.

Internal or Management network - Used as the internal communication network between OpenStack compute and control nodes. Can also be used as a network for iSCSI communication between the compute and iSCSI storage nodes. Again, this should be a minimum of a 1Gb NIC and should be a non-routed network. This interface should be redundant for high availability (HA).

Tenant network - A private network that enables communication between each tenant's instances. If using flat networking and provider networks, this network is optional. This network should also be isolated from all other networks for security compliance. A 1Gb interface should be sufficient and redundant for HA.

Storage network - A private network which could be connected to the Ceph frontend or other shared storage. For HA purposes this should be a redundant configuration with suggested 10Gb NICs. This network isolates the storage for the instances away from other networks. Under load, this storage traffic could overwhelm other networks and cause outages on other OpenStack services.

(Optional) External or Public network - This network is used to communicate externally from the VMs to the public network space. These addresses are typically handled by the neutron agent on the controller nodes and can also be handled by a SDN other than neutron. However, when using neutron DVR with OVS, this network must be present on the compute node since north and south traffic will not be handled by the controller nodes, but by the compute node itself. For more information on DVR with OVS and compute nodes, see Open vSwitch: High availability using DVR

Compute server logging

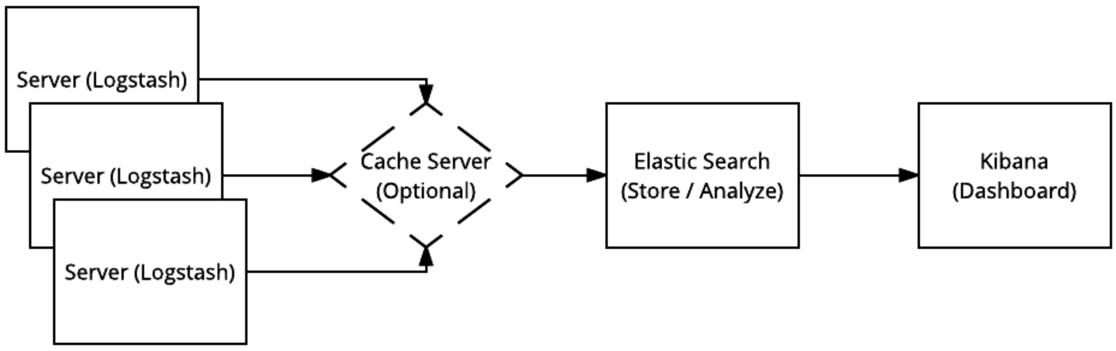
The logs on the compute nodes, or any server running nova-compute (for example in a hyperconverged architecture), are the primary points for troubleshooting issues with the hypervisor and compute services. Additionally, operating system logs can also provide useful information.

As the cloud environment grows, the amount of log data increases exponentially. Enabling debugging on either the OpenStack services or the operating system further compounds the data issues.

Logging is described in more detail in the Operations Guide. However, it is an important design consideration to take into account before commencing operations of your cloud.

OpenStack produces a great deal of useful logging information, but for the information to be useful for operations purposes, you should consider having a central logging server to send logs to, and a log parsing/analysis system such as Elastic Stack [formerly known as ELK].

Elastic Stack consists of mainly three components: Elasticsearch (log search and analysis), Logstash (log intake, processing and output) and Kibana (log dashboard service).



Due to the amount of logs being sent from servers in the OpenStack environment, an optional in-memory data structure store can be used. Common examples are Redis and Memcached. In newer versions of Elastic Stack, a file buffer called Filebeat is used for a similar purpose but adds a "backpressure-sensitive" protocol when sending data to Logstash or Elasticsearch.

Log analysis often requires disparate logs of differing formats. Elastic Stack (namely Logstash) was created to take many different log inputs and transform them into a consistent format that Elasticsearch can catalog and analyze. As seen in the image above, the process of ingestion starts on the servers by Logstash, is forwarded to the Elasticsearch server for storage and searching, and then displayed through Kibana for visual analysis and interaction.

For instructions on installing Logstash, Elasticsearch and Kibana, see the Elasticsearch reference.

There are some specific configuration parameters that are needed to configure Logstash for OpenStack. For example, in order to get Logstash to collect, parse, and send the correct portions of log files to the Elasticsearch server, you need to format the configuration file properly. There are input, output and filter configurations. Input configurations tell Logstash where to recieve data from (log files/forwarders/filebeats/StdIn/Eventlog), output configurations specify where to put the data, and filter configurations define the input contents to forward to the output.

The Logstash filter performs intermediary processing on each event. Conditional filters are applied based on the characteristics of the input and the event. Some examples of filtering are:

- grok
- date
- csv
- json

There are also output filters available that send event data to many different destinations. Some examples are:

- csv
- redis
- elasticsearch
- file
- jira
- nagios
- pagerduty
- stdout

Additionally there are several codecs that can be used to change the data representation of events such as:

- collectd
- graphite
- json
- plan
- rubydebug

These input, output and filter configurations are typically stored in `/etc/logstash/conf.d` but may vary by linux distribution. Separate configuration files should be created for different logging systems such as syslog, Apache, and OpenStack.

General examples and configuration guides can be found on the Elastic Logstash Configuration page.

OpenStack input, output and filter examples can be found at https://github.com/sorantis/elkstack/tree/master/elk/logstash.

Once a configuration is complete, Kibana can be used as a visualization tool for OpenStack and system logging. This will allow operators to configure custom dashboards for performance, monitoring and security.

This section describes some of the choices you need to consider when designing and building your compute nodes. Compute nodes form the resource core of the OpenStack Compute cloud, providing the processing, memory, network and storage resources to run instances.

Storage design

Storage is found in many parts of the OpenStack cloud environment. This chapter describes storage type, design considerations and options when selecting persistent storage options for your cloud environment.

Storage concepts

Storage is found in many parts of the OpenStack cloud environment. It is important to understand the distinction between *ephemeral* storage and *persistent* storage:

- Ephemeral storage - If you only deploy OpenStack *Compute service (nova)*, by default your users do not have access to any form of persistent storage. The disks associated with VMs are ephemeral, meaning that from the user's point of view they disappear when a virtual machine is terminated.
- Persistent storage - Persistent storage means that the storage resource outlives any other resource and is always available, regardless of the state of a running instance.

OpenStack clouds explicitly support three types of persistent storage: *Object Storage*, *Block Storage*, and *File-based storage*.

Object storage

Object storage is implemented in OpenStack by the Object Storage service (swift). Users access binary objects through a REST API. If your intended users need to archive or manage large datasets, you should provide them with Object Storage service. Additional benefits include:

- OpenStack can store your virtual machine (VM) images inside of an Object Storage system, as an alternative to storing the images on a file system.
- Integration with OpenStack Identity, and works with the OpenStack Dashboard.
- Better support for distributed deployments across multiple datacenters through support for asynchronous eventual consistency replication.

You should consider using the OpenStack Object Storage service if you eventually plan on distributing your storage cluster across multiple data centers, if you need unified accounts for your users for both compute and object storage, or if you want to control your object storage with the OpenStack Dashboard. For more information, see the Swift project page.

Block storage

Block storage is implemented in OpenStack by the Block Storage service (cinder). Because these volumes are persistent, they can be detached from one instance and re-attached to another instance and the data remains intact.

The Block Storage service supports multiple back ends in the form of drivers. Your choice of a storage back end must be supported by a block storage driver.

Most block storage drivers allow the instance to have direct access to the underlying storage hardware's block device. This helps increase the overall read/write IO. However, support for utilizing files as volumes is also well established, with full support for NFS, GlusterFS and others.

These drivers work a little differently than a traditional block storage driver. On an NFS or GlusterFS file system, a single file is created and then mapped as a virtual volume into the instance. This mapping and translation is similar to how OpenStack utilizes QEMU's file-based virtual machines stored in `/var/lib/nova/instances`.

File-based storage

In multi-tenant OpenStack cloud environment, the Shared File Systems service (manila) provides a set of services for management of shared file systems. The Shared File Systems service supports multiple back-ends in the form of drivers, and can be configured to provision shares from one or more back-ends. Share servers are virtual machines that export file shares using different file system protocols such as NFS, CIFS, GlusterFS, or HDFS.

The Shared File Systems service is persistent storage and can be mounted to any number of client machines. It can also be detached from one instance and attached to another instance without data loss. During this process the data are safe unless the Shared File Systems service itself is changed or removed.

Users interact with the Shared File Systems service by mounting remote file systems on their instances with the following usage of those systems for file storing and exchange. The Shared File Systems service provides shares which is a remote, mountable file system. You can mount a share and access a share from several hosts by several users at a time. With shares, you can also:

- Create a share specifying its size, shared file system protocol, visibility level.
- Create a share on either a share server or standalone, depending on the selected back-end mode, with or without using a share network.
- Specify access rules and security services for existing shares.
- Combine several shares in groups to keep data consistency inside the groups for the following safe group operations.
- Create a snapshot of a selected share or a share group for storing the existing shares consistently or creating new shares from that snapshot in a consistent way.
- Create a share from a snapshot.
- Set rate limits and quotas for specific shares and snapshots.
- View usage of share resources.
- Remove shares.

Differences between storage types

Table. OpenStack storage explains the differences between Openstack storage types.

Table 1: Table. OpenStack storage

	Ephemeral storage	Block storage	Object storage	Shared File System storage
Application	Run operating system and scratch space	Add additional persistent storage to a virtual machine (VM)	Store data, including VM images	Add additional persistent storage to a virtual machine
Accessed through…	A file system	A block device that can be partitioned, formatted, and mounted (such as, /dev/vdc)	The REST API	A Shared File Systems service share (either manila managed or an external one registered in manila) that can be partitioned, formatted and mounted (such as /dev/vdc)
Accessible from…	Within a VM	Within a VM	Anywhere	Within a VM
Managed by…	OpenStack Compute (nova)	OpenStack Block Storage (cinder)	OpenStack Object Storage (swift)	OpenStack Shared File System Storage (manila)
Persists until…	VM is terminated	Deleted by user	Deleted by user	Deleted by user
Sizing determined by…	Administrator configuration of size settings, known as *flavors*	User specification in initial request	Amount of available physical storage	• User specification in initial request • Requests for extension • Available user-level quotes • Limitations applied by Administrator
Encryption configuration	Parameter in `nova.conf`	Admin establishing encrypted volume type, then user selecting encrypted volume	Not yet available	Shared File Systems service does not apply any additional encryption above what the share's back-end storage provides
Example of typical usage…	10 GB first disk, 30 GB second disk	1 TB disk	10s of TBs of dataset storage	Depends completely on the size of back-end storage specified when a share was being created. In case of thin provisioning it can be partial space reservation (for more details

Note: File-level storage for live migration

With file-level storage, users access stored data using the operating system's file system interface. Most users who have used a network storage solution before have encountered this form of networked storage. The most common file system protocol for Unix is NFS, and for Windows, CIFS (previously, SMB).

OpenStack clouds do not present file-level storage to end users. However, it is important to consider file-level storage for storing instances under `/var/lib/nova/instances` when designing your cloud, since you must have a shared file system if you want to support live migration.

Commodity storage technologies

There are various commodity storage back end technologies available. Depending on your cloud user's needs, you can implement one or many of these technologies in different combinations.

Ceph

Ceph is a scalable storage solution that replicates data across commodity storage nodes.

Ceph utilises and object storage mechanism for data storage and exposes the data via different types of storage interfaces to the end user it supports interfaces for: - Object storage - Block storage - File-system interfaces

Ceph provides support for the same Object Storage API as swift and can be used as a back end for the Block Storage service (cinder) as well as back-end storage for glance images.

Ceph supports thin provisioning implemented using copy-on-write. This can be useful when booting from volume because a new volume can be provisioned very quickly. Ceph also supports keystone-based authentication (as of version 0.56), so it can be a seamless swap in for the default OpenStack swift implementation.

Ceph's advantages include:

- The administrator has more fine-grained control over data distribution and replication strategies.
- Consolidation of object storage and block storage.
- Fast provisioning of boot-from-volume instances using thin provisioning.
- Support for the distributed file-system interface CephFS.

You should consider Ceph if you want to manage your object and block storage within a single system, or if you want to support fast boot-from-volume.

Gluster

A distributed shared file system. As of Gluster version 3.3, you can use Gluster to consolidate your object storage and file storage into one unified file and object storage solution, which is called Gluster For OpenStack (GFO). GFO uses a customized version of swift that enables Gluster to be used as the back-end storage.

The main reason to use GFO rather than swift is if you also want to support a distributed file system, either to support shared storage live migration or to provide it as a separate service to your end users. If you want to manage your object and file storage within a single system, you should consider GFO.

LVM

The Logical Volume Manager (LVM) is a Linux-based system that provides an abstraction layer on top of physical disks to expose logical volumes to the operating system. The LVM back-end implements block storage as LVM logical partitions.

On each host that will house block storage, an administrator must initially create a volume group dedicated to Block Storage volumes. Blocks are created from LVM logical volumes.

Note: LVM does *not* provide any replication. Typically, administrators configure RAID on nodes that use LVM as block storage to protect against failures of individual hard drives. However, RAID does not protect against a failure of the entire host.

iSCSI

Internet Small Computer Systems Interface (iSCSI) is a network protocol that operates on top of the Transport Control Protocol (TCP) for linking data storage devices. It transports data between an iSCSI initiator on a server and iSCSI target on a storage device.

iSCSI is suitable for cloud environments with Block Storage service to support applications or for file sharing systems. Network connectivity can be achieved at a lower cost compared to other storage back end technologies since iSCSI does not require host bus adaptors (HBA) or storage-specific network devices.

NFS

Network File System (NFS) is a file system protocol that allows a user or administrator to mount a file system on a server. File clients can access mounted file systems through Remote Procedure Calls (RPC).

The benefits of NFS is low implementation cost due to shared NICs and traditional network components, and a simpler configuration and setup process.

For more information on configuring Block Storage to use NFS storage, see Configure an NFS storage back end in the OpenStack Administrator Guide.

Sheepdog

Sheepdog is a userspace distributed storage system. Sheepdog scales to several hundred nodes, and has powerful virtual disk management features like snapshot, cloning, rollback and thin provisioning.

It is essentially an object storage system that manages disks and aggregates the space and performance of disks linearly in hyper scale on commodity hardware in a smart way. On top of its object store, Sheepdog provides elastic volume service and http service. Sheepdog does require a specific kernel version and can work nicely with xattr-supported file systems.

ZFS

The Solaris iSCSI driver for OpenStack Block Storage implements blocks as ZFS entities. ZFS is a file system that also has the functionality of a volume manager. This is unlike on a Linux system, where there is a separation of volume manager (LVM) and file system (such as, ext3, ext4, xfs, and btrfs). ZFS has a number of advantages over ext4, including improved data-integrity checking.

The ZFS back end for OpenStack Block Storage supports only Solaris-based systems, such as Illumos. While there is a Linux port of ZFS, it is not included in any of the standard Linux distributions, and it has not been tested with OpenStack Block Storage. As with LVM, ZFS does not provide replication across hosts on its own, you need to add a replication solution on top of ZFS if your cloud needs to be able to handle storage-node failures.

Storage architecture

There are many different storage architectures available when designing an OpenStack cloud. The convergence of orchestration and automation within the OpenStack platform enables rapid storage provisioning without the hassle of the traditional manual processes like volume creation and attachment.

However, before choosing a storage architecture, a few generic questions should be answered:

- Will the storage architecture scale linearly as the cloud grows and what are its limits?
- What is the desired attachment method: NFS, iSCSI, FC, or other?
- Is the storage proven with the OpenStack platform?
- What is the level of support provided by the vendor within the community?
- What OpenStack features and enhancements does the cinder driver enable?
- Does it include tools to help troubleshoot and resolve performance issues?
- Is it interoperable with all of the projects you are planning on using in your cloud?

Choosing storage back ends

Users will indicate different needs for their cloud architecture. Some may need fast access to many objects that do not change often, or want to set a time-to-live (TTL) value on a file. Others may access only storage that is mounted with the file system itself, but want it to be replicated instantly when starting a new instance. For other systems, ephemeral storage is the preferred choice. When you select *storage back ends*, consider the following questions from user's perspective:

First and foremost:

- Do I need block storage?
- Do I need object storage?
- Do I need file-based storage?

Next answer the following:

- Do I need to support live migration?
- Should my persistent storage drives be contained in my compute nodes, or should I use external storage?
- What type of performance do I need in regards to IOPS? Total IOPS and IOPS per instance? Do I have applications with IOPS SLAs?
- Are my storage needs mostly read, or write, or mixed?
- Which storage choices result in the best cost-performance scenario I am aiming for?

- How do I manage the storage operationally?
- How redundant and distributed is the storage? What happens if a storage node fails? To what extent can it mitigate my data-loss disaster scenarios?
- What is my company currently using and can I use it with OpenStack?
- Do I need more than one storage choice? Do I need tiered performance storage?

While this is not a definitive list of all the questions possible, the list above will hopefully help narrow the list of possible storage choices down.

A wide variety of use case requirements dictate the nature of the storage back end. Examples of such requirements are as follows:

- Public, private, or a hybrid cloud (performance profiles, shared storage, replication options)
- Storage-intensive use cases like HPC and Big Data clouds
- Web-scale or development clouds where storage is typically ephemeral in nature

Data security recommendations:

- We recommend that data be encrypted both in transit and at-rest. To this end, carefully select disks, appliances, and software. Do not assume these features are included with all storage solutions.
- Determine the security policy of your organization and understand the data sovereignty of your cloud geography and plan accordingly.

If you plan to use live migration, we highly recommend a shared storage configuration. This allows the operating system and application volumes for instances to reside outside of the compute nodes and adds significant performance increases when live migrating.

To deploy your storage by using only commodity hardware, you can use a number of open-source packages, as described in *Persistent file-based storage support*.

Table 2: Persistent file-based storage support

	Object	Block	File-level
Swift	✓		
LVM		✓	
Ceph	✓	✓	Experimental
Gluster	✓	✓	✓
NFS		✓	✓
ZFS		✓	
Sheep-dog	✓	✓	

This list of open source file-level shared storage solutions is not exhaustive. Your organization may already have deployed a file-level shared storage solution that you can use.

Note: Storage driver support

In addition to the open source technologies, there are a number of proprietary solutions that are officially supported by OpenStack Block Storage. You can find a matrix of the functionality provided by all of the supported Block Storage drivers on the CinderSupportMatrix wiki.

Also, you need to decide whether you want to support object storage in your cloud. The two common use cases for providing object storage in a compute cloud are to provide:

- Users with a persistent storage mechanism for objects like images and video.
- A scalable, reliable data store for OpenStack virtual machine images.
- An API driven S3 compatible object store for application use.

Selecting storage hardware

Storage hardware architecture is determined by selecting specific storage architecture. Determine the selection of storage architecture by evaluating possible solutions against the critical factors, the user requirements, technical considerations, and operational considerations. Consider the following factors when selecting storage hardware:

Cost Storage can be a significant portion of the overall system cost. For an organization that is concerned with vendor support, a commercial storage solution is advisable, although it 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 Performance of block based storage is typically measured in the maximum read and write operations to non-contiguous storage locations per second. This measurement typically applies to SAN, hard drives, and solid state drives. While IOPS can be broadly measured and is not an official benchmark, many vectors like to be used by vendors to communicate performance levels. Since there are no real standards for measuring IOPS, vendor test results may vary, sometimes wildly. However, along with transfer rate which measures the speed that data can be transferred to contiguous storage locations, IOPS can be used in a performance evaluation. Typically, transfer rate is represented by a bytes per second calculation but IOPS is measured by an integer.

To calculate IOPS for a single drive you could use: IOPS = 1 / (AverageLatency + AverageSeekTime) For example: Average Latency for Single Disk = 2.99ms or .00299 seconds Average Seek Time for Single Disk = 4.7ms or .0047 seconds IOPS = 1/(.00299 + .0047) IOPS = 130

To calculate maximum IOPS for a disk array: Maximum Read IOPS: In order to accurately calculate maximum read IOPS for a disk array, multiply the IOPS for each disk by the maximum read or write IOPS per disk. maxReadIOPS = nDisks * diskMaxIOPS For example, 15 10K Spinning Disks would be measured the following way: maxReadIOPS = 15 * 130 maxReadIOPS = 1950

Maximum write IOPS per array: Determining the maximum *write* IOPS is a little different because most administrators configure disk replication using RAID and since the RAID controller requires IOPS itself, there is a write penalty. The severity of the write penalty is determined by the type of RAID used.

Raid Type	Penalty
1	2
5	4
10	2

Note: Raid 5 has the worst penalty (has the most cross disk writes.) Therefore, when using the above examples, a 15 disk array using RAID 5 is capable of 1950 read IOPS however, we need to add the penalty when determining the *write* IOPS:

```
maxWriteIOPS = 1950 / 4
maxWriteIOPS = 487.5
```

A RAID 5 array only has 25% of the write IOPS of the read IOPS while a RAID 1 array in this case would produce a maximum of 975 IOPS.

What about SSD? DRAM SSD? In an HDD, data transfer is sequential. The actual read/write head "seeks" a point in the hard drive to execute the operation. Seek time is significant. Transfer rate can also be influenced by file system fragmentation and the layout. Finally, the mechanical nature of hard disks also has certain performance limitations.

In an SSD, data transfer is *not* sequential; it is random so it is faster. There is consistent read performance because the physical location of data is irrelevant because SSDs have no read/write heads and thus no delays due to head motion (seeking).

Note: Some basic benchmarks for small read/writes:

- **HDDs**: Small reads – 175 IOPs, Small writes – 280 IOPs
- **Flash SSDs**: Small reads – 1075 IOPs (6x), Small writes – 21 IOPs (0.1x)

- **DRAM SSDs**: Small reads – 4091 IOPs (23x), Small writes – 4184 IOPs (14x)

Scalability 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 as there are no established usage patterns for a general purpose cloud. It might become necessary to expand the initial deployment in order to accommodate growth and user demand.

Expandability Expandability is a major architecture factor for storage solutions with general purpose OpenStack cloud. A storage solution that expands to 50 PB is considered more expandable than a solution that only scales to 10 PB. This meter is related to scalability, which is the measure of a solution's performance as it expands.

Implementing Block Storage

Configure Block Storage resource nodes with advanced RAID controllers and high-performance disks to provide fault tolerance at the hardware level.

We recommend deploying high performing storage solutions such as SSD drives or flash storage systems for applications requiring additional performance out of Block Storage devices.

In environments that place substantial demands on Block Storage, we recommend using 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. Ideally, ensure that applications can schedule volumes in multiple regions, each with their own network, power, and cooling infrastructure. This will give tenants the option of building fault-tolerant applications that are 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. We recommend designing 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. It is imperative that a highly available database cluster is used to store the Block Storage metadata.

In a cloud with significant demands on Block Storage, the network architecture should take into account the amount of East-West bandwidth required for instances to make use of the available storage resources. The selected network devices should support jumbo frames for transferring large blocks of data, and utilize a dedicated network for providing connectivity between instances and Block Storage.

Implementing Object Storage

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 the implemented system meets those goals. The OpenStack Object Storage service places a specific number of data replicas as objects on resource nodes. Replicas are distributed throughout the cluster, based on a consistent hash ring also stored on each node in the cluster.

When designing your cluster, you must consider durability and availability which is dependent on the spread and placement of your data, rather than the reliability of the hardware.

Consider the default value of the number of replicas, which is three. This means that before an object is marked as having been written, at least two copies exist in case a single server fails to write, the third copy may or may not yet exist when the write operation initially returns. Altering this number increases the robustness of your data, but reduces the amount of storage you have available. Look at the placement of your servers. Consider spreading them widely throughout your data center's network and power-failure zones. Is a zone a rack, a server, or a disk?

Consider these main traffic flows for an Object Storage network:

- Among *object*, *container*, and *account servers*
- Between servers and the proxies
- Between the proxies and your users

Object Storage frequent communicates among servers hosting data. Even a small cluster generates megabytes per second of traffic.

Consider the scenario where an entire server fails and 24 TB of data needs to be transferred "immediately" to remain at three copies — this can put significant load on the network.

Another consideration is when a new file is being uploaded, the proxy server must write out as many streams as there are replicas, multiplying network traffic. For a three-replica cluster, 10 Gbps in means 30 Gbps out. Combining this with the previous high bandwidth bandwidth private versus public network recommendations demands of replication is what results in the recommendation that your private network be of significantly higher bandwidth than your public network requires. OpenStack Object Storage communicates internally with unencrypted, unauthenticated rsync for performance, so the private network is required.

The remaining point on bandwidth is the public-facing portion. The `swift-proxy` service is stateless, which means that you can easily add more and use HTTP load-balancing methods to share bandwidth and availability between them. More proxies means more bandwidth.

You should consider designing the Object Storage system with a sufficient number of zones to provide quorum for the number of replicas defined. For 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 five. 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 you properly account for the number of zones in the Object Storage cluster.

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, a pool of Object Storage proxy servers providing access to data stored on the object nodes should service each availability zone. Object proxies in each region should leverage local read and write affinity so that local storage resources facilitate access to objects wherever possible. We recommend deploying upstream load balancing to ensure that proxy services are 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. Any of the following may represent a zone:

- A disk within a single node
- One zone per node
- Zone per collection of nodes
- Multiple racks
- Multiple data centers

Selecting the proper zone design is crucial for allowing the Object Storage cluster to scale while providing an available and redundant storage system. 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.

Planning and scaling storage capacity

An important consideration in running a cloud over time is projecting growth and utilization trends in order to plan capital expenditures for the short and long term. Gather utilization meters for compute, network, and storage, along with historical records of these meters. While securing major anchor tenants can lead to rapid jumps in the utilization of resources, the average rate of adoption of cloud services through normal usage also needs to be carefully monitored.

Scaling Block Storage

You can upgrade Block Storage pools to add storage capacity without interrupting the overall Block Storage service. Add nodes to the pool by installing and configuring the appropriate hardware and software and then allowing that node to report in to the proper storage pool through the message bus. Block Storage nodes generally report into the scheduler service advertising their availability. As a result, after the node is online and available, tenants can make use of those storage resources instantly.

In some cases, the demand on Block Storage may exhaust the available network bandwidth. As a result, design network infrastructure that services Block Storage resources in such a way that you can add capacity and bandwidth easily. This often involves the use of dynamic routing protocols or advanced networking solutions to add capacity 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.

Note: Sufficient monitoring and data collection should be in-place from the start, such that timely decisions regarding capacity, input/output metrics (IOPS) or storage-associated bandwidth can be made.

Scaling Object Storage

Adding back-end storage capacity to an Object Storage cluster requires careful planning and forethought. 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 has 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 you add back-end storage capacity to the system, the partition maps redistribute data amongst the storage nodes. In some cases, this involves replication of extremely large data sets. In these cases, we recommend using back-end replication links that do not contend with tenants' access to data.

As more tenants begin to access data within the cluster and their data sets grow, it is 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 tenants use to gain access to the data, along with the high availability solutions that enable easy scaling of the proxy layer. We recommend designing 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, you must add bandwidth and capacity to the network resources servicing requests between proxy servers and storage nodes. For this reason, the network architecture used for access to storage nodes and proxy servers should make use of a design which is scalable.

Redundancy

Replication

Replicas in Object Storage function independently, and clients only require a majority of nodes to respond to a request in order for an operation to be considered successful. Thus, transient failures like network partitions can quickly cause replicas to diverge. Fix These differences are eventually reconciled by asynchronous, peer-to-peer replicator processes. The replicator processes traverse their local filesystems, concurrently performing operations in a manner that balances load across physical disks.

Replication uses a push model, with records and files generally only being copied from local to remote replicas. This is important because data on the node may not belong there (as in the case of handoffs and ring changes), and a replicator can not know what data exists elsewhere in the cluster that it should pull in. It is the duty of any node that contains data to ensure that data gets to where it belongs. Replica placement is handled by the ring.

Every deleted record or file in the system is marked by a tombstone, so that deletions can be replicated alongside creations. The replication process cleans up tombstones after a time period known as the consistency window. The consistency window encompasses replication duration and the length of time a transient failure can remove a node from the cluster. Tombstone cleanup must be tied to replication to reach replica convergence.

If a replicator detects that a remote drive has failed, the replicator uses the `get_more_nodes` interface for the ring to choose an alternative node with which to synchronize. The replicator can maintain desired levels of replication in the face of disk failures, though some replicas may not be in an immediately usable location.

Note: The replicator does not maintain desired levels of replication when other failures occur, such as entire node failures, because most failures are transient.

Replication is an area of active development, andimplementation details are likely to change over time.

There are two major classes of replicator: the db replicator, which replicates accounts and containers, and the object replicator, which replicates object data.

For more information, please see the Swift replication page.

Networking

Networking concepts

A cloud environment fundamentally changes the ways that networking is provided and consumed. Understanding the following concepts and decisions is imperative when making architectural decisions. For detailed information on networking concepts, see the OpenStack Networking Guide.

Network zones

The cloud networks are divided into a number of logical zones that support the network traffic flow requirements. We recommend defining at the least four distinct network zones.

Underlay

The underlay zone is defined as the physical network switching infrastructure that connects the storage, compute and control platforms. There are a large number of potential underlay options available.

Overlay

The overlay zone is defined as any L3 connectivity between the cloud components and could take the form of SDN solutions such as the neutron overlay solution or 3rd Party SDN solutions.

Edge

The edge zone is where network traffic transitions from the cloud overlay or SDN networks into the traditional network environments.

External

The external network is defined as the configuration and components that are required to provide access to cloud resources and workloads, the external network is defined as all the components outside of the cloud edge gateways.

Traffic flow

There are two primary types of traffic flow within a cloud infrastructure, the choice of networking technologies is influenced by the expected loads.

East/West - The internal traffic flow between workload within the cloud as well as the traffic flow between the compute nodes and storage nodes falls into the East/West category. Generally this is the heaviest traffic flow and due to the need to cater for storage access needs to cater for a minimum of hops and low latency.

North/South - The flow of traffic between the workload and all external networks, including clients and remote services. This traffic flow is highly dependant on the workload within the cloud and the type of network services being offered.

Layer networking choices

There are several factors to take into consideration when deciding on whether to use Layer 2 networking architecture or a layer 3 networking architecture. For more information about OpenStack networking concepts, see the OpenStack Networking section in the OpenStack Networking Guide.

Benefits using a Layer-2 network

There are several reasons a network designed on layer-2 protocols is selected over a network 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 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.
- Adding more layers to the Ethernet frame only slows the networking process down. This is known as nodal processing delay.
- You can add adjunct networking features, for example class of service (CoS) or multicasting, 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, and video. The concept is that the network will benefit more from the advantages of Ethernet if the transfer of information from a source to a destination is in the form of Ethernet frames.

Although 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 additional benefits over layer-3 IP network usage:

- Speed
- Reduced overhead of the IP hierarchy.
- No need to keep track of address configuration as systems move 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 addresses 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

Layer-2 network architectures have some limitations that become noticeable when used outside of traditional data centers.

- Number of VLANs is limited to 4096.
- The number of MACs stored in switch tables is limited.
- You must accommodate the need to maintain a set of layer-4 devices to handle traffic control.
- 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 can be complicated on a 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 start and stop.
- Migrating MACs (instance migration) to different physical locations are a potential problem if you do not set ARP table timeouts properly.

It is important to know that layer-2 has a very limited set of network management tools. It is difficult to control traffic as it does not have mechanisms to manage the network or shape the traffic. Network troubleshooting is also troublesome, in part because network devices have no IP addresses. As a result, there is no reasonable way to check network delay.

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 starts or stops. Because of this, there is far too much churn in the MAC tables on the backbone switches.

Furthermore, on large layer-2 networks, configuring ARP learning can be complicated. The setting for the MAC address timer on switches is critical and, if set incorrectly, can cause significant performance problems. So when migrating MACs to different physical locations to support instance migration, problems may arise. As an example, the Cisco default MAC address timer is extremely long. As such, the network information maintained in the switches could be out of sync with the new location of the instance.

Benefits using a Layer-3 network

In layer-3 networking, routing takes instance MAC and IP addresses out of the network core, reducing state churn. The only time there would be a routing state change is 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.
- You can configure layer-3 to use Border Gateway Protocol (BGP) confederation for scalability. This way core routers have state proportional to the number of racks, not to the number of servers or instances.
- There are a variety of well tested tools, such as Internet Control Message Protocol (ICMP) to monitor and manage traffic.
- Layer-3 architectures enable the use of *quality of service (QoS)* to manage network performance.

Layer-3 architecture limitations

The main limitation of layer-3 networking 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 is on the same subnet as its physical host, making migration out of the subnet difficult. For these reasons, network virtualization needs to use IP encapsulation and software at the end hosts. This is for isolation and the separation of the addressing in the virtual layer from the addressing in the physical layer. Other potential disadvantages of layer-3 networking include the need to design an IP addressing scheme rather than relying on the switches to keep track of the MAC addresses automatically, and to configure the interior gateway routing protocol in the switches.

Networking service (neutron)

OpenStack Networking (neutron) is the component of OpenStack that provides the Networking service API and a reference architecture that implements a Software Defined Network (SDN) solution.

The Networking service provides full control over creation of virtual network resources to tenants. This is often accomplished in the form of tunneling protocols that establish encapsulated communication paths over existing network infrastructure in order to segment tenant traffic. This method varies depending on the specific implementation, but some of the more common methods include tunneling over GRE, encapsulating with VXLAN, and VLAN tags.

Designing an OpenStack network

There are many reasons an OpenStack network has complex requirements. One main factor is that many components interact at different levels of the system stack. Data flows are also complex.

Data in an OpenStack cloud moves between instances across the network (known as east-west traffic), as well as in and out of the system (known as north-south traffic). Physical server nodes have network requirements that are independent of instance network requirements and must be isolated to account for scalability. We recommend separating the networks for security purposes and tuning performance through traffic shaping.

You must consider a number of important technical and business requirements when planning and designing an OpenStack network:

- Avoid hardware or software vendor lock-in. The design should not rely on specific features of a vendor's network router or switch.
- Massively scale the ecosystem to support millions of end users.
- Support an indeterminate variety of platforms and applications.
- Design for cost efficient operations to take advantage of massive scale.
- Ensure that there is no single point of failure in the cloud ecosystem.
- High availability architecture to meet customer SLA requirements.
- Tolerant to rack level failure.
- Maximize flexibility to architect future production environments.

Considering these requirements, we recommend the following:

- Design a Layer-3 network architecture rather than a layer-2 network architecture.
- 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 a 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 External Border Gateway Protocol (eBGP) to connect to the Internet up-link.
- Use Internal Border Gateway Protocol (iBGP) to flatten the internal traffic on the layer-3 mesh.
- Determine the most effective configuration for block storage network.

Additional network design considerations

There are several other considerations when designing a network-focused OpenStack cloud.

Redundant networking

You should conduct a high availability risk analysis to determine whether to use redundant switches such as Top of Rack (ToR) switches. In most cases, it is much more economical to use single switches 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 tolerate rack level outages without affecting normal operations since network and compute resources are easily provisioned and plentiful.

Research indicates 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.

Providing IPv6 support

One of the most important networking topics today is the exhaustion of IPv4 addresses. As of late 2015, ICANN announced that the final IPv4 address blocks have been fully assigned. Because of this, IPv6 protocol has become the future of network focused applications. IPv6 increases the address space significantly, fixes long standing issues in the IPv4 protocol, and will become essential for network focused applications in the future.

OpenStack Networking, when configured for it, supports IPv6. To enable IPv6, create an IPv6 subnet in Networking and use IPv6 prefixes when creating security groups.

Supporting asymmetric links

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

Optimizing network 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. When latency and jitter are issues, certain applications may require tuning of QoS parameters and network device queues to ensure that they immediately queue for transmitting or guarantee minimum bandwidth. Since OpenStack currently does not support these functions, consider carefully your selected network plug-in.

The location of a service may also impact the application or consumer experience. If an application serves differing content to different users, it must properly direct connections to those specific locations. Where appropriate, use a multi-site installation for these situations.

You can implement networking in two separate ways. 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 (L3) agent using the multi-host configuration. The Networking service (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 must be is based on the number 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 a decision based on the greatest port density often comes with a drawback. Aggregation switches and routers have not all kept pace with ToR 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.

Using tunable networking components

Consider configurable networking components related to an OpenStack architecture design when designing for network intensive workloads that include MTU and QoS. Some workloads require a larger MTU than normal due to the transfer of large blocks of data. When providing network service for applications such as video streaming or storage replication, we recommend that you configure both OpenStack hardware nodes and the supporting network equipment for jumbo frames where possible. This allows for better use of available bandwidth. Configure jumbo frames across the complete path the packets traverse. If one network component is not capable of handling jumbo frames then the entire path reverts to the default MTU.

Quality of Service (QoS) also has a great impact on network intensive workloads as it provides instant service to packets which have a higher priority due to the impact of poor network performance. In applications such as Voice over IP (VoIP), differentiated services code points are a near requirement for proper operation. You can also use QoS 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.

Choosing network hardware

The network architecture determines which network hardware will be used. Networking software is determined by the selected networking hardware.

There are more subtle design impacts that need to be considered. The selection of certain networking hardware (and the networking software) affects 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 there are instances where the relationship between networking hardware and networking software are not as tightly defined.

Some of the key considerations 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 higher port density is preferred, as it leaves more rack space for compute or storage components. This can also lead into considerations about fault domains and power density. 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 User requirements for high availability and cost considerations influence the level of network hardware redundancy. Network redundancy can be achieved by adding redundant power supplies or paired switches.

Note: Hardware must support network redundancy.

Power requirements Ensure that the physical data center provides the necessary power for the selected network hardware.

Note: This is not an issue for top of rack (ToR) switches. This 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 more performance out of a single storage system by using specialized network technologies such as RDMA, SRP, iSER and SCST. The specifics of using these technologies is beyond the scope of this book.

There is no single best practice architecture for the networking hardware supporting an OpenStack cloud. Some of the key factors that will have a major influence on selection of networking hardware include:

Connectivity All nodes within an OpenStack cloud require 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 network design should encompass a physical and logical network design that can be easily expanded upon. Network hardware should offer the appropriate types of interfaces and speeds that are required by the hardware nodes.

Availability To ensure access to nodes within the cloud is not interrupted, we recommend that the network architecture identifies any single points of failure and provides some level of redundancy or fault tolerance. The network infrastructure often involves use of networking protocols such as LACP, VRRP or others to achieve a highly available network connection. It is also important to consider the networking implications on API availability. We recommend a load balancing solution is designed within the network architecture to ensure that the APIs and potentially other services in the cloud are highly available.

Choosing networking software

OpenStack Networking (neutron) provides a wide variety of networking services for instances. There are many additional networking software packages that can be useful when managing OpenStack components. Some examples include:

- Software to provide load balancing
- Network redundancy protocols
- Routing daemons.

Additional networking services

OpenStack, like any network application, has a number of standard services to consider, such as NTP and DNS.

NTP

Time synchronization is a critical element to ensure continued operation of OpenStack components. Ensuring that all components have the correct time is necessary to avoid errors in instance scheduling, replication of objects in the object store, and matching log timestamps for debugging.

All servers running OpenStack components should be able to access an appropriate NTP server. You may decide to set up one locally or use the public pools available from the Network Time Protocol project.

DNS

OpenStack does not currently provide DNS services, aside from the dnsmasq daemon, which resides on `nova-network` hosts. You could consider providing a dynamic DNS service to allow instances to update a DNS entry with new IP addresses. You can also consider making a generic forward and reverse DNS mapping for instances' IP addresses, such as `vm-203-0-113-123.example.com`.

DHCP

LBaaS

OpenStack provides a rich networking environment. This chapter details the requirements and options to consider when designing your cloud. This includes examples of network implementations to consider, information about some OpenStack network layouts and networking services that are essential for stable operation.

Warning: If this is the first time you are deploying a cloud infrastructure in your organization, your first conversations should be with your networking team. Network usage in a running cloud is vastly different from traditional network deployments and has the potential to be disruptive at both a connectivity and a policy level.

For example, you must plan the number of IP addresses that you need for both your guest instances as well as management infrastructure. Additionally, you must research and discuss cloud network connectivity through proxy servers and firewalls.

Identity

Images

Control Plane

OpenStack is designed to be massively horizontally scalable, which allows all services to be distributed widely. However, to simplify this guide, we have decided to discuss services of a more central nature, using the concept of a *cloud controller*. A cloud controller is a conceptual simplification. In the real world, you design an architecture for your cloud controller that enables high availability so that if any node fails, another can take over the required tasks. In reality, cloud controller tasks are spread out across more than a single node.

The cloud controller provides the central management system for OpenStack deployments. Typically, the cloud controller manages authentication and sends messaging to all the systems through a message queue.

For many deployments, the cloud controller is a single node. However, to have high availability, you have to take a few considerations into account, which we'll cover in this chapter.

The cloud controller manages the following services for the cloud:

Databases Tracks current information about users and instances, for example, in a database, typically one database instance managed per service

Message queue services All *Advanced Message Queuing Protocol (AMQP)* messages for services are received and sent according to the queue broker

Conductor services Proxy requests to a database

Authentication and authorization for identity management Indicates which users can do what actions on certain cloud resources; quota management is spread out among services, howeverauthentication

Image-management services Stores and serves images with metadata on each, for launching in the cloud

Scheduling services Indicates which resources to use first; for example, spreading out where instances are launched based on an algorithm

User dashboard Provides a web-based front end for users to consume OpenStack cloud services

API endpoints Offers each service's REST API access, where the API endpoint catalog is managed by the Identity service

For our example, the cloud controller has a collection of `nova-*` components that represent the global state of the cloud; talks to services such as authentication; maintains information about the cloud in a database; communicates to all compute nodes and storage *workers* through a queue; and provides API access. Each service running on a designated cloud controller may be broken out into separate nodes for scalability or availability.

As another example, you could use pairs of servers for a collective cloud controller—one active, one standby—for redundant nodes providing a given set of related services, such as:

- Front end web for API requests, the scheduler for choosing which compute node to boot an instance on, Identity services, and the dashboard
- Database and message queue server (such as MySQL, RabbitMQ)
- Image service for the image management

Now that you see the myriad designs for controlling your cloud, read more about the further considerations to help with your design decisions.

Hardware Considerations

A cloud controller's hardware can be the same as a compute node, though you may want to further specify based on the size and type of cloud that you run.

It's also possible to use virtual machines for all or some of the services that the cloud controller manages, such as the message queuing. In this guide, we assume that all services are running directly on the cloud controller.

Table. Cloud controller hardware sizing considerations contains common considerations to review when sizing hardware for the cloud controller design.

Table 3: Table. Cloud controller hardware sizing considerations

Consideration	Ramification
How many instances will run at once?	Size your database server accordingly, and scale out beyond one cloud controller if many instances will report status at the same time and scheduling where a new instance starts up needs computing power.
How many compute nodes will run at once?	Ensure that your messaging queue handles requests successfully and size accordingly.
How many users will access the API?	If many users will make multiple requests, make sure that the CPU load for the cloud controller can handle it.
How many users will access the dashboard versus the REST API directly?	The dashboard makes many requests, even more than the API access, so add even more CPU if your dashboard is the main interface for your users.
How many `nova-api` services do you run at once for your cloud?	You need to size the controller with a core per service.
How long does a single instance run?	Starting instances and deleting instances is demanding on the compute node but also demanding on the controller node because of all the API queries and scheduling needs.
Does your authentication system also verify externally?	External systems such as *LDAP* or *Active Directory* require network connectivity between the cloud controller and an external authentication system. Also ensure that the cloud controller has the CPU power to keep up with requests.

Separation of Services

While our example contains all central services in a single location, it is possible and indeed often a good idea to separate services onto different physical servers. *Table. Deployment scenarios* is a list of deployment scenarios we've seen and their justifications.

Table 4: Table. Deployment scenarios

Scenario	Justification
Run `glance-*` servers on the `swift-proxy` server.	This deployment felt that the spare I/O on the Object Storage proxy server was sufficient and that the Image Delivery portion of glance benefited from being on physical hardware and having good connectivity to the Object Storage back end it was using.
Run a central dedicated database server.	This deployment used a central dedicated server to provide the databases for all services. This approach simplified operations by isolating database server updates and allowed for the simple creation of slave database servers for failover.
Run one VM per service.	This deployment ran central services on a set of servers running KVM. A dedicated VM was created for each service (`nova-scheduler`, rabbitmq, database, etc). This assisted the deployment with scaling because administrators could tune the resources given to each virtual machine based on the load it received (something that was not well understood during installation).
Use an external load balancer.	This deployment had an expensive hardware load balancer in its organization. It ran multiple `nova-api` and `swift-proxy` servers on different physical servers and used the load balancer to switch between them.

One choice that always comes up is whether to virtualize. Some services, such as `nova-compute`, `swift-proxy` and `swift-object` servers, should not be virtualized. However, control servers can often be happily virtualized—the performance penalty can usually be offset by simply running more of the service.

Database

OpenStack Compute uses an SQL database to store and retrieve stateful information. MySQL is the popular database choice in the OpenStack community.

Loss of the database leads to errors. As a result, we recommend that you cluster your database to make it failure tolerant. Configuring and maintaining a database cluster is done outside OpenStack and is determined by the database software you choose to use in your cloud environment. MySQL/Galera is a popular option for MySQL-based databases.

Message Queue

Most OpenStack services communicate with each other using the *message queue*. For example, Compute communicates to block storage services and networking services through the message queue. Also, you can optionally enable notifications for any service. RabbitMQ, Qpid, and Zeromq are all popular choices for a message-queue service. In general, if the message queue fails or becomes inaccessible, the cluster grinds to a halt and ends up in a read-only state, with information stuck at the point where the last message was sent. Accordingly, we recommend that you cluster the message queue. Be aware that clustered message queues can be a pain point for many OpenStack deployments. While RabbitMQ has native clustering support, there have been reports of issues when running it at a large scale. While other queuing solutions are available, such as Zeromq and Qpid, Zeromq does not offer stateful queues. Qpid is the messaging system of choice for Red Hat and its derivatives. Qpid does not have native clustering capabilities and requires a supplemental service, such as Pacemaker or Corsync. For your message queue, you need to determine what level of data loss you are comfortable with and whether to use an OpenStack project's ability to retry multiple MQ hosts in the event of a failure, such as using Compute's ability to do so.

Conductor Services

In the previous version of OpenStack, all `nova-compute` services required direct access to the database hosted on the cloud controller. This was problematic for two reasons: security and performance. With regard to security, if a compute node is compromised, the attacker inherently has access to the database. With regard to performance, `nova-compute` calls to the database are single-threaded and blocking. This creates a performance bottleneck because database requests are fulfilled serially rather than in parallel.

The conductor service resolves both of these issues by acting as a proxy for the `nova-compute` service. Now, instead of `nova-compute` directly accessing the database, it contacts the `nova-conductor` service, and `nova-conductor` accesses the database on `nova-compute`'s behalf. Since `nova-compute` no longer has direct access to the database, the security issue is resolved. Additionally, `nova-conductor` is a nonblocking service, so requests from all compute nodes are fulfilled in parallel.

Note: If you are using `nova-network` and multi-host networking in your cloud environment, `nova-compute` still requires direct access to the database.

The `nova-conductor` service is horizontally scalable. To make `nova-conductor` highly available and fault tolerant, just launch more instances of the `nova-conductor` process, either on the same server or across multiple servers.

Application Programming Interface (API)

All public access, whether direct, through a command-line client, or through the web-based dashboard, uses the API service. Find the API reference at Development resources for OpenStack clouds.

You must choose whether you want to support the Amazon EC2 compatibility APIs, or just the OpenStack APIs. One issue you might encounter when running both APIs is an inconsistent experience when referring to images and instances.

For example, the EC2 API refers to instances using IDs that contain hexadecimal, whereas the OpenStack API uses names and digits. Similarly, the EC2 API tends to rely on DNS aliases for contacting virtual machines, as opposed to OpenStack, which typically lists IP addresses.

If OpenStack is not set up in the right way, it is simple to have scenarios in which users are unable to contact their instances due to having only an incorrect DNS alias. Despite this, EC2 compatibility can assist users migrating to your cloud.

As with databases and message queues, having more than one *API server* is a good thing. Traditional HTTP load-balancing techniques can be used to achieve a highly available `nova-api` service.

Extensions

The API Specifications define the core actions, capabilities, and mediatypes of the OpenStack API. A client can always depend on the availability of this core API, and implementers are always required to support it in its entirety. Requiring strict adherence to the core API allows clients to rely upon a minimal level of functionality when interacting with multiple implementations of the same API.

The OpenStack Compute API is extensible. An extension adds capabilities to an API beyond those defined in the core. The introduction of new features, MIME types, actions, states, headers, parameters, and resources can all be accomplished by means of extensions to the core API. This allows the introduction of new features in the API without requiring a version change and allows the introduction of vendor-specific niche functionality.

Scheduling

The scheduling services are responsible for determining the compute or storage node where a virtual machine or block storage volume should be created. The scheduling services receive creation requests for these resources from the message queue and then begin the process of determining the appropriate node where the resource should reside. This process is done by applying a series of user-configurable filters against the available collection of nodes.

There are currently two schedulers: `nova-scheduler` for virtual machines and `cinder-scheduler` for block storage volumes. Both schedulers are able to scale horizontally, so for high-availability purposes, or for very large or high-schedule-frequency installations, you should consider running multiple instances of each scheduler. The schedulers all listen to the shared message queue, so no special load balancing is required.

Images

The OpenStack Image service consists of two parts: `glance-api` and `glance-registry`. The former is responsible for the delivery of images; the compute node uses it to download images from the back end. The latter maintains the metadata information associated with virtual machine images and requires a database.

The `glance-api` part is an abstraction layer that allows a choice of back end. Currently, it supports:

OpenStack Object Storage Allows you to store images as objects.

File system Uses any traditional file system to store the images as files.

S3 Allows you to fetch images from Amazon S3.

HTTP Allows you to fetch images from a web server. You cannot write images by using this mode.

If you have an OpenStack Object Storage service, we recommend using this as a scalable place to store your images. You can also use a file system with sufficient performance or Amazon S3—unless you do not need the ability to upload new images through OpenStack.

Dashboard

The OpenStack dashboard (horizon) provides a web-based user interface to the various OpenStack components. The dashboard includes an end-user area for users to manage their virtual infrastructure and an admin area for cloud operators to manage the OpenStack environment as a whole.

The dashboard is implemented as a Python web application that normally runs in *Apache* `httpd`. Therefore, you may treat it the same as any other web application, provided it can reach the API servers (including their admin endpoints) over the network.

Authentication and Authorization

The concepts supporting OpenStack's authentication and authorization are derived from well-understood and widely used systems of a similar nature. Users have credentials they can use to authenticate, and they can be a member of one or more groups (known as projects or tenants, interchangeably).

For example, a cloud administrator might be able to list all instances in the cloud, whereas a user can see only those in his current group. Resources quotas, such as the number of cores that can be used, disk space, and so on, are associated with a project.

OpenStack Identity provides authentication decisions and user attribute information, which is then used by the other OpenStack services to perform authorization. The policy is set in the `policy.json` file. For information on how to configure these, see Managing Projects and Users in the OpenStack Operations Guide.

OpenStack Identity supports different plug-ins for authentication decisions and identity storage. Examples of these plug-ins include:

- In-memory key-value Store (a simplified internal storage structure)
- SQL database (such as MySQL or PostgreSQL)
- Memcached (a distributed memory object caching system)
- LDAP (such as OpenLDAP or Microsoft's Active Directory)

Many deployments use the SQL database; however, LDAP is also a popular choice for those with existing authentication infrastructure that needs to be integrated.

Network Considerations

Because the cloud controller handles so many different services, it must be able to handle the amount of traffic that hits it. For example, if you choose to host the OpenStack Image service on the cloud controller, the cloud controller should be able to support the transferring of the images at an acceptable speed.

As another example, if you choose to use single-host networking where the cloud controller is the network gateway for all instances, then the cloud controller must support the total amount of traffic that travels between your cloud and the public Internet.

We recommend that you use a fast NIC, such as 10 GB. You can also choose to use two 10 GB NICs and bond them together. While you might not be able to get a full bonded 20 GB speed, different transmission streams use different NICs. For example, if the cloud controller transfers two images, each image uses a different NIC and gets a full 10 GB of bandwidth.

Cloud management platform tools

Complex clouds, in particular hybrid clouds, may require tools to facilitate working across multiple clouds.

Broker between clouds Brokering software evaluates relative costs between different cloud platforms. Cloud Management Platforms (CMP) allow the designer to determine the right location for the workload based on predetermined criteria.

Facilitate orchestration across the clouds CMPs simplify the migration of application workloads between public, private, and hybrid cloud platforms.

We recommend using cloud orchestration tools for managing a diverse portfolio of systems and applications across multiple cloud platforms.

Technical details

Capacity and scale

High availability

Operator requirements

Deployment considerations

Maintenance considerations

Use cases

Development cloud

Design model

Requirements

Component block diagram

General compute cloud

Design model

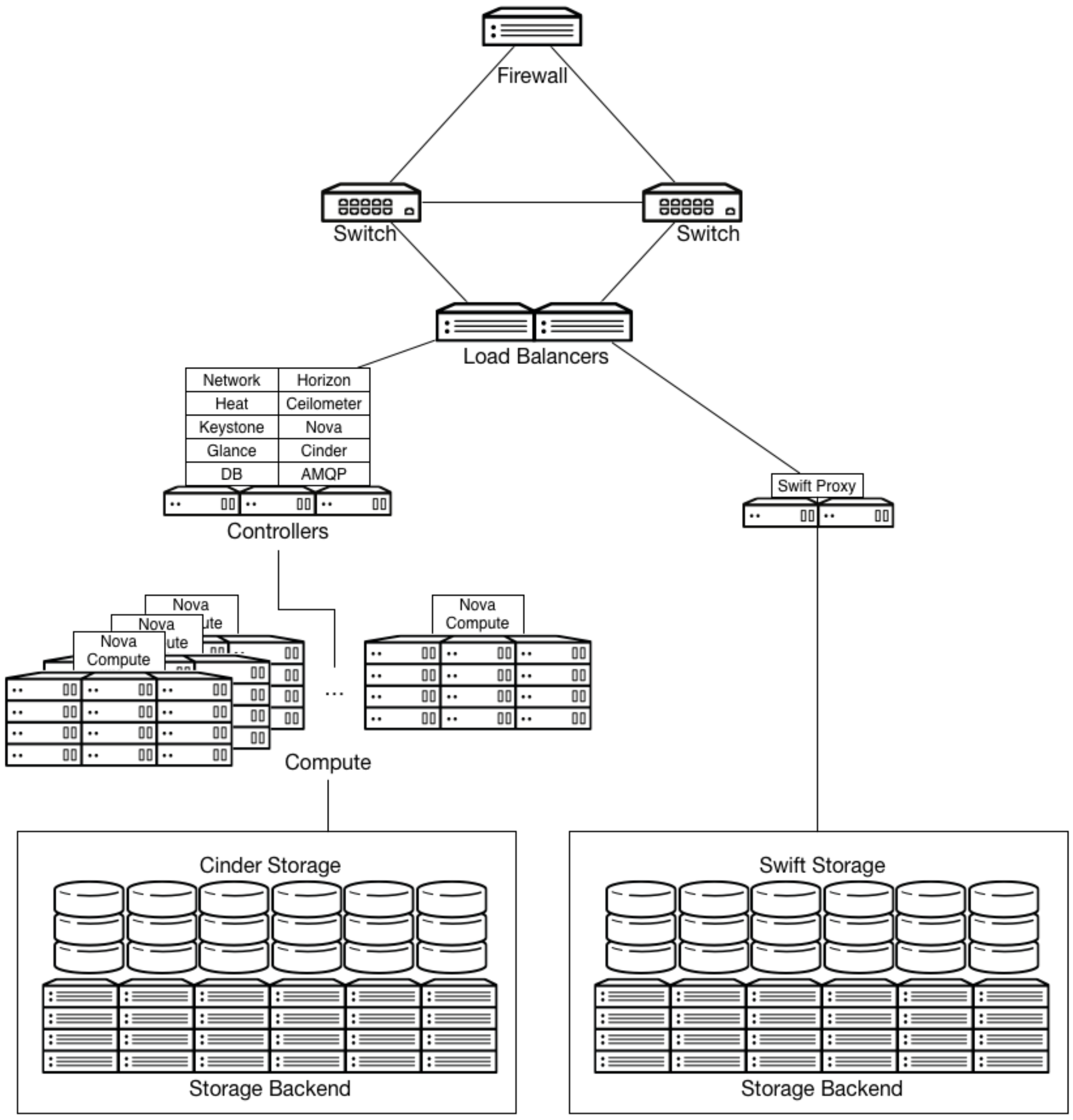
An online classified advertising company wants to run web applications consisting of Tomcat, Nginx, and MariaDB in a private cloud. To meet the policy requirements, the cloud infrastructure will run in their own data center. The company has predictable load requirements but requires scaling to cope with nightly increases in demand. Their current environment does not have the flexibility to align with their goal of running an open source API environment. The 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 of RAM

The company runs hardware load balancers and multiple web applications serving their websites and orchestrates environments using combinations of scripts and Puppet. The website generates large amounts of log data daily that requires archiving.

The solution would consist of the following OpenStack components:

- A firewall, switches and load balancers on the public facing network connections.
- OpenStack Controller service running Image service, Identity service, Networking service, combined with support services such as MariaDB and RabbitMQ, configured for high availability on at least three controller nodes.
- OpenStack compute nodes running the KVM hypervisor.
- OpenStack Block Storage for use by compute instances, requiring 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 of 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, meaning 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 processing and archiving.

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: Increasing OpenStack Object Storage means network bandwidth needs to be taken into consideration. Running OpenStack Object Storage with network connections offering 10 GbE or better connectivity is advised.

Leveraging Orchestration and Telemetry services is also a potential issue when providing auto-scaling, orchestrated web application environments. Defining the web applications in a *Heat Orchestration Template (HOT)* negates the reliance on the current scripted Puppet solution.

OpenStack Networking can be used to control hardware load balancers through the use of plug-ins and the Networking API. This allows users 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.

Requirements

Storage requirements

Using a scale-out storage solution with direct-attached storage (DAS) in the servers is well suited for a general purpose OpenStack cloud. Cloud services requirements determine your choice of scale-out solution. You need to determine if a single, highly expandable and highly vertical, scalable, centralized storage array is suitable for your design. After determining an approach, select the storage hardware based on this criteria.

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 If storage protocols other than Ethernet are part of the storage solution, ensure the appropriate hardware has been selected. 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.

Instance and image locations Where instances and images will be stored will influence the architecture.

Server hardware If the solution is a scale-out storage architecture that includes DAS, it 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.

A general purpose OpenStack cloud has multiple options. 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. Defining the initial requirements and ensuring the design can support adding capacity is important. Hardware nodes selected for object storage should be capable of support a large number of inexpensive disks with no reliance on RAID controller cards. Hardware nodes selected for block storage should be capable of supporting high speed storage solutions and RAID controller cards to provide performance and redundancy to storage at a hardware level. Selecting hardware RAID controllers that automatically repair damaged arrays will assist with the replacement and repair of degraded or deleted storage devices.

Performance Disks selected for object storage services do not need to be fast performing disks. We recommend that object storage nodes take advantage of the best cost per terabyte available for storage. Contrastingly, disks chosen for block storage services should take advantage of performance boosting features that may entail the use of SSDs or flash storage to provide high performance block storage pools. Storage performance of ephemeral disks used for instances should also be taken into consideration.

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 hardware 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.

Network hardware requirements

For a compute-focus architecture, we recommend designing the network architecture using a scalable network model that makes it easy to add capacity and bandwidth. A good example of such a model is the leaf-spine model. In this type of network design, you can add additional bandwidth as well as scale out to additional racks of gear. It is important to select network hardware that supports port count, port speed, and port density while allowing for future growth as workload demands increase. In the network architecture, it is also important to evaluate where to provide redundancy.

Network software requirements

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

Component block diagram

Web scale cloud

Design model

Requirements

Component block diagram

Storage cloud

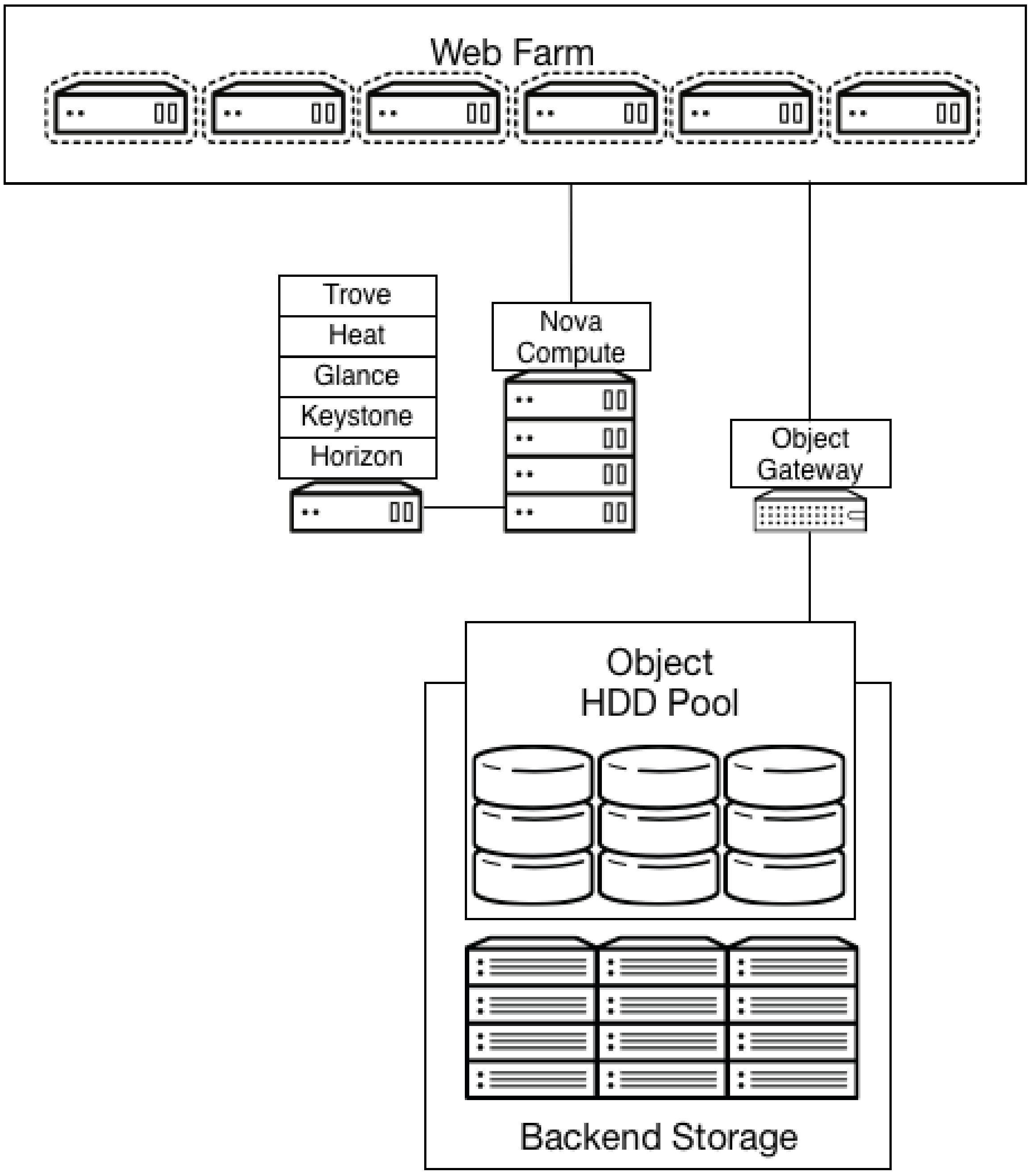
Design model

Storage-focused architecture depends on specific use cases. This section discusses three example use cases:

- An object store with a RESTful interface
- Compute analytics with parallel file systems
- High performance database

An object store with a RESTful interface

The example below shows a REST interface without a high performance requirement. The following diagram depicts the example architecture:



The example REST interface, presented as a traditional Object Store running on traditional spindles, does not require a high performance caching tier.

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 equaling 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

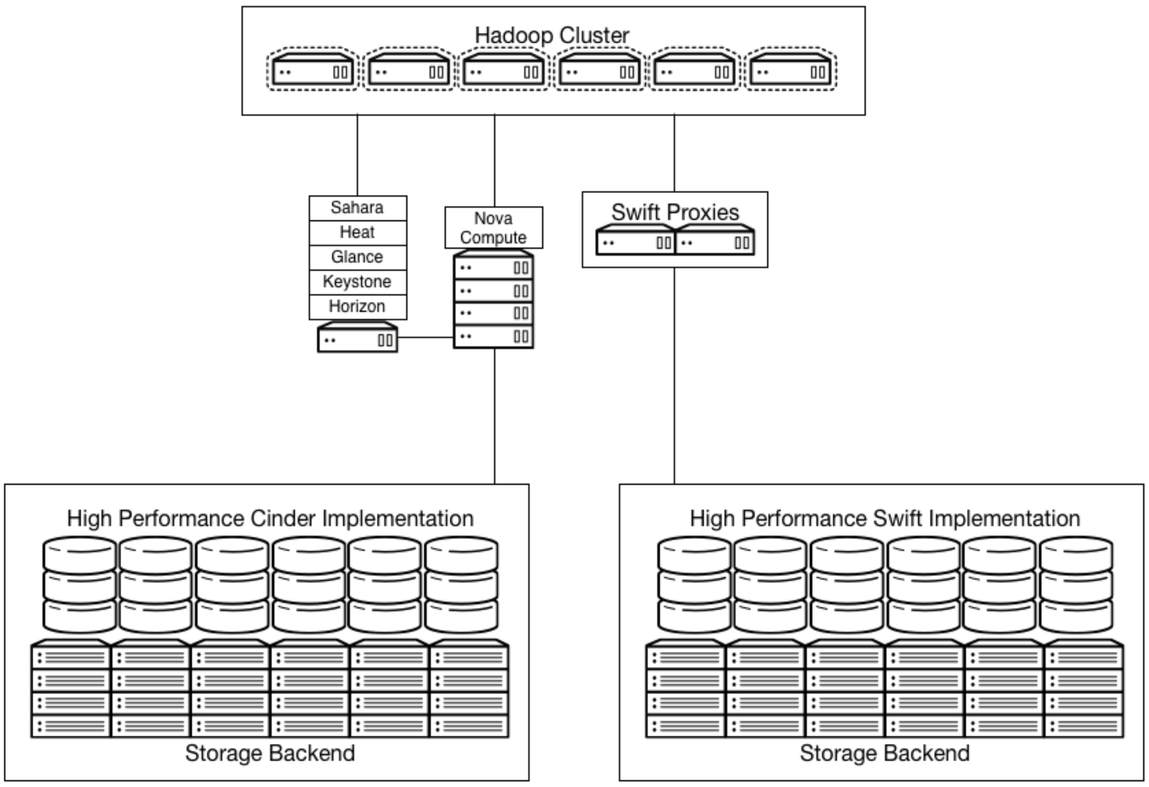
Note: It may be necessary to implement a third party caching layer for some applications to achieve suitable performance.

Compute analytics with data processing service

Analytics of large data sets are dependent on the performance of the storage system. Clouds using storage systems such as Hadoop Distributed File System (HDFS) have inefficiencies which can cause performance issues.

One potential solution to this problem is the implementation of storage systems designed for performance. Parallel file systems have previously filled this need in the HPC space and are suitable for large scale performance-orientated systems.

OpenStack has integration with Hadoop to manage the Hadoop cluster within the cloud. The following diagram shows an OpenStack store with a high performance requirement:



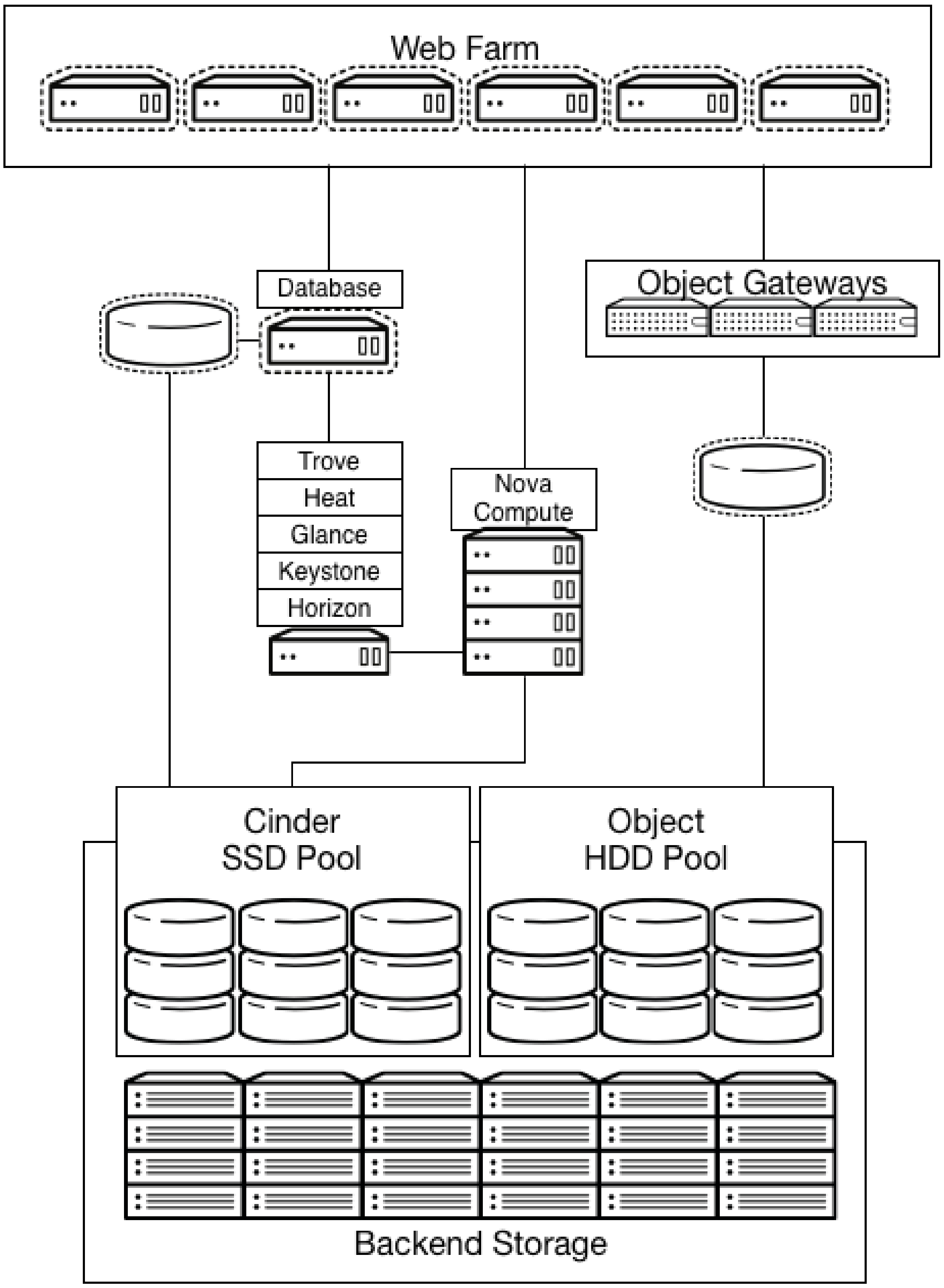
The hardware requirements and configuration are similar to those of the High Performance Database example below. In this case, the architecture uses Ceph's Swift-compatible REST interface, features that allow for connecting a caching pool to allow for acceleration of the presented pool.

High performance database with Database service

Databases are a common workload that benefit from high performance storage back ends. Although enterprise storage is not a requirement, many environments have existing storage that OpenStack cloud can use as back ends. You can create a storage pool to provide block devices with OpenStack Block Storage for instances as well as object interfaces. In this example, the database I-O requirements are high and demand storage presented from a fast SSD pool.

A storage system presents a LUN 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 system can provide additional performance. 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 inline SSD cache layer accelerates the REST interface.



In this example, Ceph presents 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 enable reduced cost of operations such as self healing and auto balancing. Using erasure coded pools are a suitable way of maximizing the amount of usable space.

Note: There are special considerations around erasure coded pools. For example, higher computational requirements and limitations on the operations allowed on an object; erasure coded pools do not support partial writes.

Using Ceph as an applicable example, a potential architecture would have the following requirements:

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

Using an SSD cache layer, you can present block devices directly to hypervisors or instances. The REST interface can also use the SSD cache systems as an inline cache.

Requirements

Storage requirements

Storage-focused OpenStack clouds must address I/O intensive workloads. These workloads are not CPU intensive, nor are they consistently network intensive. The network may be heavily utilized to transfer storage, but they are not otherwise network intensive.

The selection of storage hardware determines the overall performance and scalability of a storage-focused OpenStack design architecture. Several factors impact the design process, including:

Latency A key consideration in a storage-focused OpenStack cloud is latency. Using solid-state disks (SSDs) to minimize latency and, to reduce CPU delays caused by waiting for the storage, increases performance. Use RAID controller cards in compute hosts to improve the performance of the underlying disk subsystem.

Scale-out solutions Depending on the storage architecture, you can adopt a scale-out solution, or use a highly expandable and scalable centralized storage array. If a centralized storage array meets your requirements, then the array vendor determines the hardware selection. It is possible to build a storage array using commodity hardware with Open Source software, but requires 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. This requires configuration of the server hardware to support the storage solution.

Considerations affecting storage architecture (and corresponding storage hardware) of a Storage-focused OpenStack cloud include:

Connectivity Ensure the connectivity matches the storage solution requirements. We recommend confirming that the network characteristics minimize latency to boost the overall performance of the design.

Latency Determine if the use case has consistent or highly variable latency.

Throughput Ensure that the storage solution throughput is optimized for your 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.

Component block diagram

Network virtual function cloud

Design model

Requirements

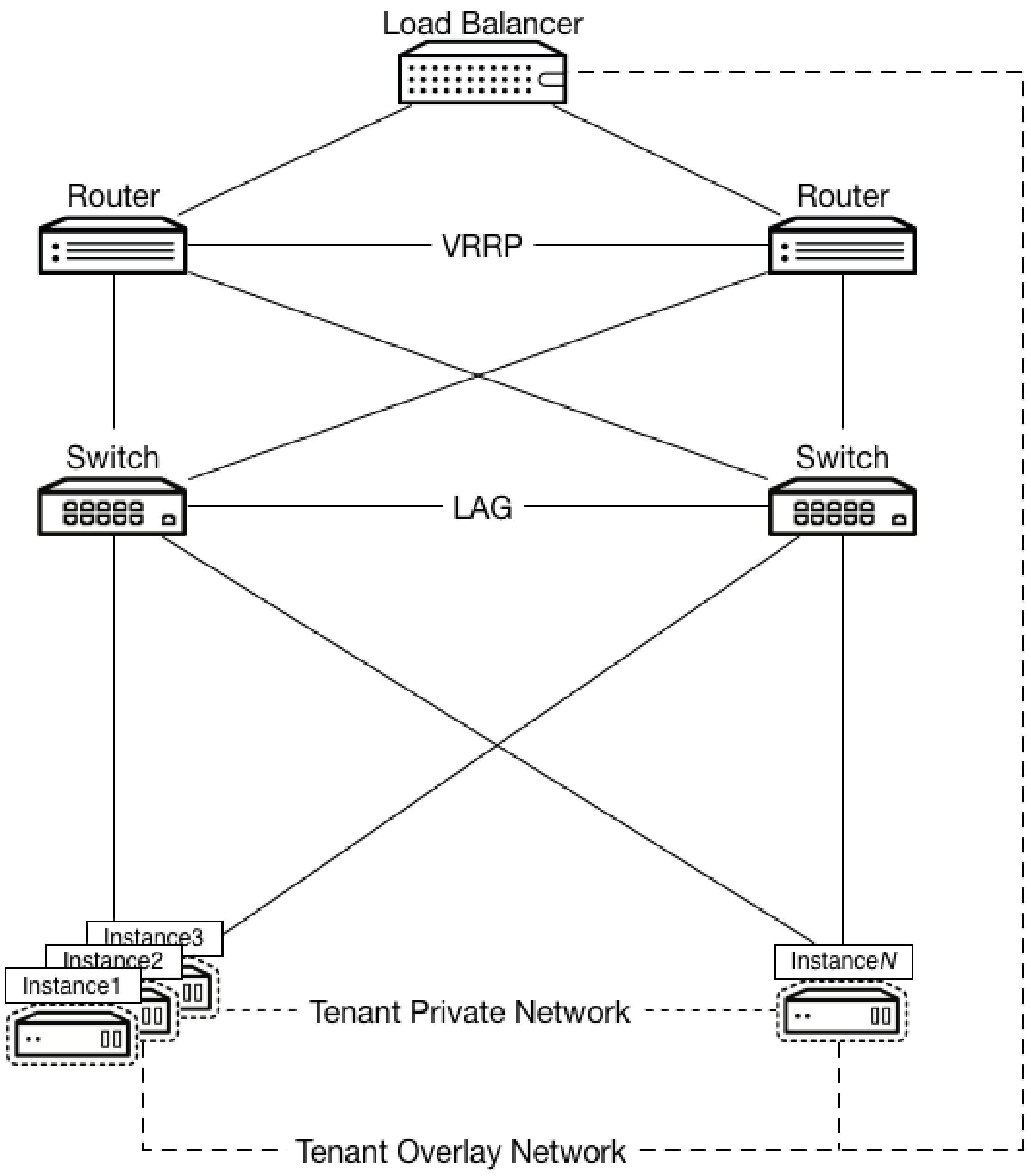
Component block diagram

Network-focused cloud examples

An organization designs a large scale cloud-based web application. The application scales horizontally in a bursting behavior and generates a high instance count. The application requires an SSL connection to secure data and must not lose connection state to individual servers.

The figure below depicts an example design for this workload. In this example, a hardware load balancer provides SSL offload functionality and connects to tenant networks in order to reduce address consumption. This load balancer links to the routing architecture as it services the VIP for the application. The router and load balancer use the GRE tunnel ID of the application's tenant network and 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 closed, the routing and switching architecture provides high availability. Switches mesh to each hypervisor and each other, and also provide an MLAG implementation to ensure that layer-2 connectivity does not fail. Routers use VRRP and fully mesh with switches to ensure layer-3 connectivity. Since GRE provides an overlay network, Networking is present and uses the Open vSwitch agent in GRE tunnel mode. This ensures all devices can reach all other devices and that you can create tenant networks 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. A few key components, however, need to be in place to handle the nature of most web-scale workloads. You require the following components:

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

Beyond the normal Identity service, Compute service, Image service, and Object Storage components, we recommend the Orchestration service component to handle the proper scaling of workloads to adjust to demand. Due to the requirement for auto-scaling, the design includes the Telemetry service. 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 works 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 spreads requests across multiple instances. This workload scales well horizontally across large numbers of instances. This enables instances to run without publicly routed IP addresses and instead to rely on the load balancer to provide a globally reachable service. 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 pair with VRRP, and switches pair with an implementation of MLAG 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)* applies to these workloads for a middle ground Class Selector depending on existing policies. It is 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, you can optimize bandwidth utilization 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. Use a multi-site approach 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 therefore we do not recommend them.

QoS is desirable 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. Configure rules in upstream devices to apply a higher Class Selector to DNS to ensure faster delivery or a better spot in queuing algorithms.

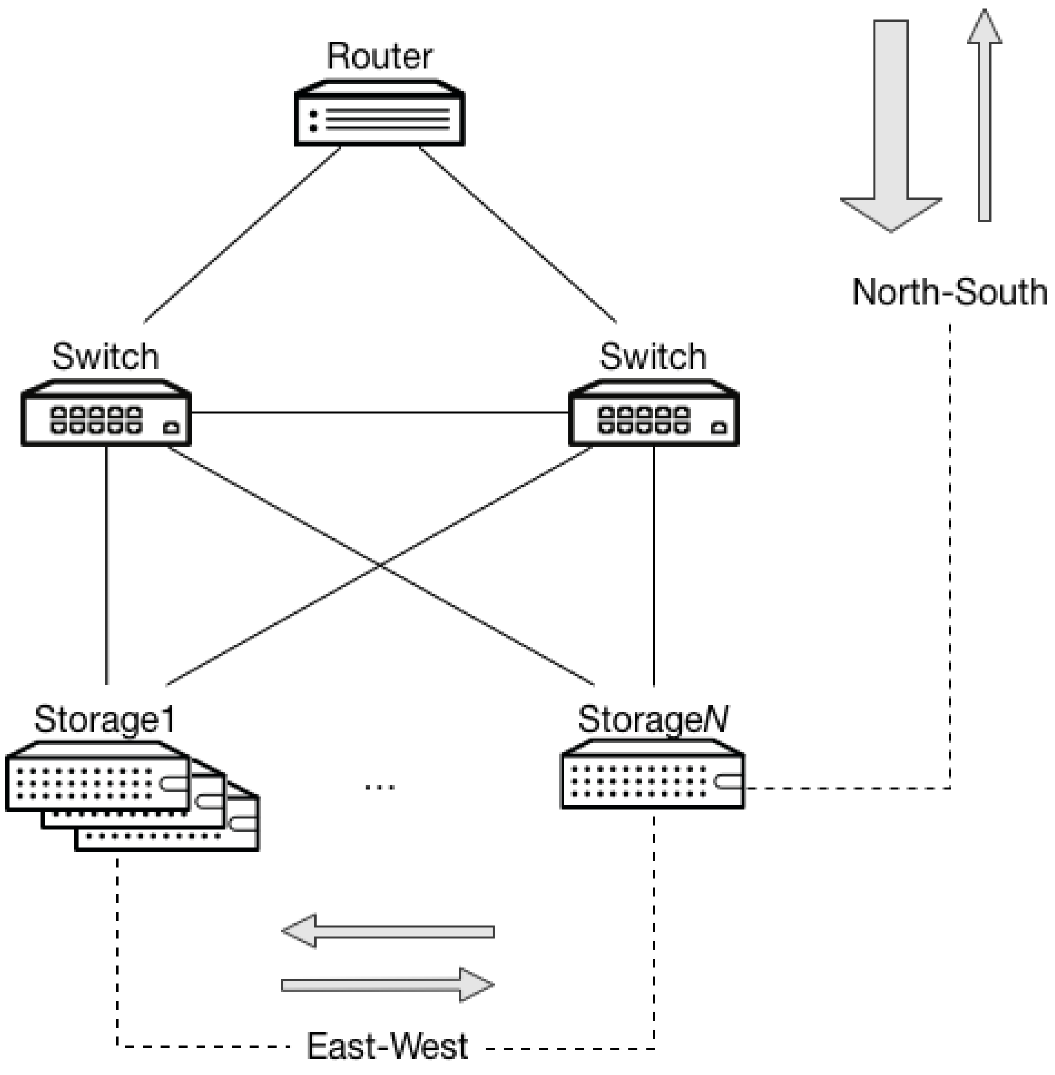
Cloud storage

Another common use case for OpenStack environments is providing 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:

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 out from the OpenStack installation. Because this service operates primarily as a backup, most of the traffic moves southbound 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 dependent on availability and more dependent on being able to handle high bandwidth. As a direct result, it is beneficial to forgo 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.

Appendix

Community support

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, join our IRC channel `#openstack-doc` on the freenode IRC network, or report a bug.

The following books explain how to install an OpenStack cloud and its associated components:

- Installation Tutorial for openSUSE Leap 42.2 and SUSE Linux Enterprise Server 12 SP2
- Installation Tutorial for Red Hat Enterprise Linux 7 and CentOS 7
- Installation Tutorial for Ubuntu 16.04 (LTS)

The following books explain how to configure and run an OpenStack cloud:

- Architecture Design Guide
- Administrator Guide
- Configuration Reference
- Operations Guide
- Networking Guide
- High Availability Guide
- Security Guide
- Virtual Machine Image Guide

The following books explain how to use the OpenStack Dashboard and command-line clients:

- End User Guide
- Command-Line Interface Reference

The following documentation provides reference and guidance information for the OpenStack APIs:

- API Guide

The following guide provides how to contribute to OpenStack documentation:

- Documentation Contributor Guide

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 Ask OpenStack 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 the general OpenStack mailing list. If you are interested in the other mailing lists for specific projects or development, refer to Mailing Lists.

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 OpenStack Compute, 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. 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, `"Kilo 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: Application catalog (murano)
- Bugs: Bare metal service (ironic)
- Bugs: Clustering service (senlin)
- Bugs: Container Infrastructure Management service (magnum)
- Bugs: Data processing service (sahara)
- Bugs: Database service (trove)
- Bugs: Deployment service (fuel)
- Bugs: DNS service (designate)
- Bugs: Key Manager Service (barbican)
- Bugs: Monitoring (monasca)
- Bugs: Orchestration (heat)
- Bugs: Rating (cloudkitty)
- Bugs: Shared file systems (manila)
- Bugs: Telemetry (ceilometer)
- Bugs: Telemetry v3 (gnocchi)
- Bugs: Workflow service (mistral)
- Bugs: Messaging service (zaqar)
- 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 https://webchat.freenode.net/. You can also use Colloquy (Mac OS X), mIRC (Windows), 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 Paste. 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 on the IRC page on the wiki.

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:** https://wiki.debian.org/OpenStack
- **CentOS, Fedora, and Red Hat Enterprise Linux:** https://www.rdoproject.org/
- **openSUSE and SUSE Linux Enterprise Server:** https://en.opensuse.org/Portal:OpenStack
- **Ubuntu:** https://wiki.ubuntu.com/ServerTeam/CloudArchive

Glossary

This glossary offers a list of terms and definitions to define a vocabulary for OpenStack-related concepts.

To add to OpenStack glossary, clone the openstack/openstack-manuals repository and update the source file `doc/common/glossary.rst` through the OpenStack contribution process.

0-9

6to4 A mechanism that allows IPv6 packets to be transmitted over an IPv4 network, providing a strategy for migrating to IPv6.

A

absolute limit Impassable limits for guest VMs. Settings include total RAM size, maximum number of vCPUs, and maximum disk size.

access control list (ACL) A list of permissions attached to an object. An ACL specifies which users or system processes have access to objects. It also defines which operations can be performed on specified objects. Each entry in a typical ACL specifies a subject and an operation. For instance, the ACL entry `(Alice, delete)` for a file gives Alice permission to delete the file.

access key Alternative term for an Amazon EC2 access key. See EC2 access key.

account The Object Storage context of an account. Do not confuse with a user account from an authentication service, such as Active Directory, /etc/passwd, OpenLDAP, OpenStack Identity, and so on.

account auditor Checks for missing replicas and incorrect or corrupted objects in a specified Object Storage account by running queries against the back-end SQLite database.

account database A SQLite database that contains Object Storage accounts and related metadata and that the accounts server accesses.

account reaper An Object Storage worker that scans for and deletes account databases and that the account server has marked for deletion.

account server Lists containers in Object Storage and stores container information in the account database.

account service An Object Storage component that provides account services such as list, create, modify, and audit. Do not confuse with OpenStack Identity service, OpenLDAP, or similar user-account services.

accounting The Compute service provides accounting information through the event notification and system usage data facilities.

Active Directory Authentication and identity service by Microsoft, based on LDAP. Supported in OpenStack.

active/active configuration In a high-availability setup with an active/active configuration, several systems share the load together and if one fails, the load is distributed to the remaining systems.

active/passive configuration In a high-availability setup with an active/passive configuration, systems are set up to bring additional resources online to replace those that have failed.

address pool A group of fixed and/or floating IP addresses that are assigned to a project and can be used by or assigned to the VM instances in a project.

Address Resolution Protocol (ARP) The protocol by which layer-3 IP addresses are resolved into layer-2 link local addresses.

admin API A subset of API calls that are accessible to authorized administrators and are generally not accessible to end users or the public Internet. They can exist as a separate service (keystone) or can be a subset of another API (nova).

admin server In the context of the Identity service, the worker process that provides access to the admin API.

administrator The person responsible for installing, configuring, and managing an OpenStack cloud.

Advanced Message Queuing Protocol (AMQP) The open standard messaging protocol used by OpenStack components for intra-service communications, provided by RabbitMQ, Qpid, or ZeroMQ.

Advanced RISC Machine (ARM) Lower power consumption CPU often found in mobile and embedded devices. Supported by OpenStack.

alert The Compute service can send alerts through its notification system, which includes a facility to create custom notification drivers. Alerts can be sent to and displayed on the dashboard.

allocate The process of taking a floating IP address from the address pool so it can be associated with a fixed IP on a guest VM instance.

Amazon Kernel Image (AKI) Both a VM container format and disk format. Supported by Image service.

Amazon Machine Image (AMI) Both a VM container format and disk format. Supported by Image service.

Amazon Ramdisk Image (ARI) Both a VM container format and disk format. Supported by Image service.

Anvil A project that ports the shell script-based project named DevStack to Python.

aodh Part of the OpenStack *Telemetry service*; provides alarming functionality.

Apache The Apache Software Foundation supports the Apache community of open-source software projects. These projects provide software products for the public good.

Apache License 2.0 All OpenStack core projects are provided under the terms of the Apache License 2.0 license.

Apache Web Server The most common web server software currently used on the Internet.

API endpoint The daemon, worker, or service that a client communicates with to access an API. API endpoints can provide any number of services, such as authentication, sales data, performance meters, Compute VM commands, census data, and so on.

API extension Custom modules that extend some OpenStack core APIs.

API extension plug-in Alternative term for a Networking plug-in or Networking API extension.

API key Alternative term for an API token.

API server Any node running a daemon or worker that provides an API endpoint.

API token Passed to API requests and used by OpenStack to verify that the client is authorized to run the requested operation.

API version In OpenStack, the API version for a project is part of the URL. For example, `example.com/nova/v1/foobar`.

applet A Java program that can be embedded into a web page.

Application Catalog service (murano) The project that provides an application catalog service so that users can compose and deploy composite environments on an application abstraction level while managing the application lifecycle.

Application Programming Interface (API) A collection of specifications used to access a service, application, or program. Includes service calls, required parameters for each call, and the expected return values.

application server A piece of software that makes available another piece of software over a network.

Application Service Provider (ASP) Companies that rent specialized applications that help businesses and organizations provide additional services with lower cost.

arptables Tool used for maintaining Address Resolution Protocol packet filter rules in the Linux kernel firewall modules. Used along with iptables, ebtables, and ip6tables in Compute to provide firewall services for VMs.

associate The process associating a Compute floating IP address with a fixed IP address.

Asynchronous JavaScript and XML (AJAX) A group of interrelated web development techniques used on the client-side to create asynchronous web applications. Used extensively in horizon.

ATA over Ethernet (AoE) A disk storage protocol tunneled within Ethernet.

attach The process of connecting a VIF or vNIC to a L2 network in Networking. In the context of Compute, this process connects a storage volume to an instance.

attachment (network) Association of an interface ID to a logical port. Plugs an interface into a port.

auditing Provided in Compute through the system usage data facility.

auditor A worker process that verifies the integrity of Object Storage objects, containers, and accounts. Auditors is the collective term for the Object Storage account auditor, container auditor, and object auditor.

Austin The code name for the initial release of OpenStack. The first design summit took place in Austin, Texas, US.

auth node Alternative term for an Object Storage authorization node.

authentication The process that confirms that the user, process, or client is really who they say they are through private key, secret token, password, fingerprint, or similar method.

authentication token A string of text provided to the client after authentication. Must be provided by the user or process in subsequent requests to the API endpoint.

AuthN The Identity service component that provides authentication services.

authorization The act of verifying that a user, process, or client is authorized to perform an action.

authorization node An Object Storage node that provides authorization services.

AuthZ The Identity component that provides high-level authorization services.

Auto ACK Configuration setting within RabbitMQ that enables or disables message acknowledgment. Enabled by default.

auto declare A Compute RabbitMQ setting that determines whether a message exchange is automatically created when the program starts.

availability zone An Amazon EC2 concept of an isolated area that is used for fault tolerance. Do not confuse with an OpenStack Compute zone or cell.

AWS CloudFormation template AWS CloudFormation allows Amazon Web Services (AWS) users to create and manage a collection of related resources. The Orchestration service supports a CloudFormation-compatible format (CFN).

B

back end Interactions and processes that are obfuscated from the user, such as Compute volume mount, data transmission to an iSCSI target by a daemon, or Object Storage object integrity checks.

back-end catalog The storage method used by the Identity service catalog service to store and retrieve information about API endpoints that are available to the client. Examples include an SQL database, LDAP database, or KVS back end.

back-end store The persistent data store used to save and retrieve information for a service, such as lists of Object Storage objects, current state of guest VMs, lists of user names, and so on. Also, the method that the Image service uses to get and store VM images. Options include Object Storage, locally mounted file system, RADOS block devices, VMware datastore, and HTTP.

Backup, Restore, and Disaster Recovery service (freezer) The project that provides integrated tooling for backing up, restoring, and recovering file systems, instances, or database backups.

bandwidth The amount of available data used by communication resources, such as the Internet. Represents the amount of data that is used to download things or the amount of data available to download.

barbican Code name of the *Key Manager service*.

bare An Image service container format that indicates that no container exists for the VM image.

Bare Metal service (ironic) The OpenStack service that provides a service and associated libraries capable of managing and provisioning physical machines in a security-aware and fault-tolerant manner.

base image An OpenStack-provided image.

Bell-LaPadula model A security model that focuses on data confidentiality and controlled access to classified information. This model divides the entities into subjects and objects. The clearance of a subject is compared to the classification of the object to determine if the subject is authorized for the specific access mode. The clearance or classification scheme is expressed in terms of a lattice.

Benchmark service (rally) OpenStack project that provides a framework for performance analysis and benchmarking of individual OpenStack components as well as full production OpenStack cloud deployments.

Bexar A grouped release of projects related to OpenStack that came out in February of 2011. It included only Compute (nova) and Object Storage (swift). Bexar is the code name for the second release of OpenStack. The design summit took place in San Antonio, Texas, US, which is the county seat for Bexar county.

binary Information that consists solely of ones and zeroes, which is the language of computers.

bit A bit is a single digit number that is in base of 2 (either a zero or one). Bandwidth usage is measured in bits per second.

bits per second (BPS) The universal measurement of how quickly data is transferred from place to place.

block device A device that moves data in the form of blocks. These device nodes interface the devices, such as hard disks, CD-ROM drives, flash drives, and other addressable regions of memory.

block migration A method of VM live migration used by KVM to evacuate instances from one host to another with very little downtime during a user-initiated switchover. Does not require shared storage. Supported by Compute.

Block Storage API An API on a separate endpoint for attaching, detaching, and creating block storage for compute VMs.

Block Storage service (cinder) The OpenStack service that implement services and libraries to provide on-demand, self-service access to Block Storage resources via abstraction and automation on top of other block storage devices.

BMC (Baseboard Management Controller) The intelligence in the IPMI architecture, which is a specialized micro-controller that is embedded on the motherboard of a computer and acts as a server. Manages the interface between system management software and platform hardware.

bootable disk image A type of VM image that exists as a single, bootable file.

Bootstrap Protocol (BOOTP) A network protocol used by a network client to obtain an IP address from a configuration server. Provided in Compute through the dnsmasq daemon when using either the FlatDHCP manager or VLAN manager network manager.

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.

browser Any client software that enables a computer or device to access the Internet.

builder file Contains configuration information that Object Storage uses to reconfigure a ring or to re-create it from scratch after a serious failure.

bursting The practice of utilizing a secondary environment to elastically build instances on-demand when the primary environment is resource constrained.

button class A group of related button types within horizon. Buttons to start, stop, and suspend VMs are in one class. Buttons to associate and disassociate floating IP addresses are in another class, and so on.

byte Set of bits that make up a single character; there are usually 8 bits to a byte.

C

cache pruner A program that keeps the Image service VM image cache at or below its configured maximum size.

Cactus An OpenStack grouped release of projects that came out in the spring of 2011. It included Compute (nova), Object Storage (swift), and the Image service (glance). Cactus is a city in Texas, US and is the code name for the third release of OpenStack. When OpenStack releases went from three to six months long, the code name of the release changed to match a geography nearest the previous summit.

CALL One of the RPC primitives used by the OpenStack message queue software. Sends a message and waits for a response.

capability Defines resources for a cell, including CPU, storage, and networking. Can apply to the specific services within a cell or a whole cell.

capacity cache A Compute back-end database table that contains the current workload, amount of free RAM, and number of VMs running on each host. Used to determine on which host a VM starts.

capacity updater A notification driver that monitors VM instances and updates the capacity cache as needed.

CAST One of the RPC primitives used by the OpenStack message queue software. Sends a message and does not wait for a response.

catalog A list of API endpoints that are available to a user after authentication with the Identity service.

catalog service An Identity service that lists API endpoints that are available to a user after authentication with the Identity service.

ceilometer Part of the OpenStack *Telemetry service*; gathers and stores metrics from other OpenStack services.

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.

cell forwarding A Compute option that enables parent cells to pass resource requests to child cells if the parent cannot provide the requested resource.

cell manager The Compute component that contains a list of the current capabilities of each host within the cell and routes requests as appropriate.

CentOS A Linux distribution that is compatible with OpenStack.

Ceph Massively scalable distributed storage system that consists of an object store, block store, and POSIX-compatible distributed file system. Compatible with OpenStack.

CephFS The POSIX-compliant file system provided by Ceph.

certificate authority (CA) In cryptography, an entity that issues digital certificates. The digital certificate certifies the ownership of a public key by the named subject of the certificate. This enables others (relying parties) to rely upon signatures or assertions made by the private key that corresponds to the certified public key. In this model of trust relationships, a CA is a trusted third party for both the subject (owner) of the certificate and the party relying upon the certificate. CAs are characteristic of many public key infrastructure (PKI) schemes. In OpenStack, a simple certificate authority is provided by Compute for cloudpipe VPNs and VM image decryption.

Challenge-Handshake Authentication Protocol (CHAP) An iSCSI authentication method supported by Compute.

chance scheduler A scheduling method used by Compute that randomly chooses an available host from the pool.

changes since A Compute API parameter that downloads changes to the requested item since your last request, instead of downloading a new, fresh set of data and comparing it against the old data.

Chef An operating system configuration management tool supporting OpenStack deployments.

child cell If a requested resource such as CPU time, disk storage, or memory is not available in the parent cell, the request is forwarded to its associated child cells. If the child cell can fulfill the request, it does. Otherwise, it attempts to pass the request to any of its children.

cinder Codename for *Block Storage service*.

CirrOS A minimal Linux distribution designed for use as a test image on clouds such as OpenStack.

Cisco neutron plug-in A Networking plug-in for Cisco devices and technologies, including UCS and Nexus.

cloud architect A person who plans, designs, and oversees the creation of clouds.

Cloud Auditing Data Federation (CADF) Cloud Auditing Data Federation (CADF) is a specification for audit event data. CADF is supported by OpenStack Identity.

cloud computing A model that enables access to a shared pool of configurable computing resources, such as networks, servers, storage, applications, and services, that can be rapidly provisioned and released with minimal management effort or service provider interaction.

cloud controller Collection of Compute components that represent the global state of the cloud; talks to services, such as Identity authentication, Object Storage, and node/storage workers through a queue.

cloud controller node A node that runs network, volume, API, scheduler, and image services. Each service may be broken out into separate nodes for scalability or availability.

Cloud Data Management Interface (CDMI) SINA standard that defines a RESTful API for managing objects in the cloud, currently unsupported in OpenStack.

Cloud Infrastructure Management Interface (CIMI) An in-progress specification for cloud management. Currently unsupported in OpenStack.

cloud-init A package commonly installed in VM images that performs initialization of an instance after boot using information that it retrieves from the metadata service, such as the SSH public key and user data.

cloudadmin One of the default roles in the Compute RBAC system. Grants complete system access.

Cloudbase-Init A Windows project providing guest initialization features, similar to cloud-init.

cloudpipe A compute service that creates VPNs on a per-project basis.

cloudpipe image A pre-made VM image that serves as a cloudpipe server. Essentially, OpenVPN running on Linux.

Clustering service (senlin) The project that implements clustering services and libraries for the management of groups of homogeneous objects exposed by other OpenStack services.

command filter Lists allowed commands within the Compute rootwrap facility.

Common Internet File System (CIFS) A file sharing protocol. It is a public or open variation of the original Server Message Block (SMB) protocol developed and used by Microsoft. Like the SMB protocol, CIFS runs at a higher level and uses the TCP/IP protocol.

Common Libraries (oslo) The project that produces a set of python libraries containing code shared by OpenStack projects. The APIs provided by these libraries should be high quality, stable, consistent, documented and generally applicable.

community project A project that is not officially endorsed by the OpenStack Foundation. If the project is successful enough, it might be elevated to an incubated project and then to a core project, or it might be merged with the main code trunk.

compression Reducing the size of files by special encoding, the file can be decompressed again to its original content. OpenStack supports compression at the Linux file system level but does not support compression for things such as Object Storage objects or Image service VM images.

Compute API (Nova API) The nova-api daemon provides access to nova services. Can communicate with other APIs, such as the Amazon EC2 API.

compute controller The Compute component that chooses suitable hosts on which to start VM instances.

compute host Physical host dedicated to running compute nodes.

compute node A node that runs the nova-compute daemon that manages VM instances that provide a wide range of services, such as web applications and analytics.

Compute service (nova) The OpenStack core project that implements services and associated libraries to provide massively-scalable, on-demand, self-service access to compute resources, including bare metal, virtual machines, and containers.

compute worker The Compute component that runs on each compute node and manages the VM instance lifecycle, including run, reboot, terminate, attach/detach volumes, and so on. Provided by the nova-compute daemon.

concatenated object A set of segment objects that Object Storage combines and sends to the client.

conductor In Compute, conductor is the process that proxies database requests from the compute process. Using conductor improves security because compute nodes do not need direct access to the database.

congress Code name for the *Governance service*.

consistency window The amount of time it takes for a new Object Storage object to become accessible to all clients.

console log Contains the output from a Linux VM console in Compute.

container Organizes and stores objects in Object Storage. Similar to the concept of a Linux directory but cannot be nested. Alternative term for an Image service container format.

container auditor Checks for missing replicas or incorrect objects in specified Object Storage containers through queries to the SQLite back-end database.

container database A SQLite database that stores Object Storage containers and container metadata. The container server accesses this database.

container format A wrapper used by the Image service that contains a VM image and its associated metadata, such as machine state, OS disk size, and so on.

Container Infrastructure Management service (magnum) The project which provides a set of services for provisioning, scaling, and managing container orchestration engines.

container server An Object Storage server that manages containers.

container service The Object Storage component that provides container services, such as create, delete, list, and so on.

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.

controller node Alternative term for a cloud controller node.

core API Depending on context, the core API is either the OpenStack API or the main API of a specific core project, such as Compute, Networking, Image service, and so on.

core service An official OpenStack service defined as core by DefCore Committee. Currently, consists of Block Storage service (cinder), Compute service (nova), Identity service (keystone), Image service (glance), Networking service (neutron), and Object Storage service (swift).

cost Under the Compute distributed scheduler, this is calculated by looking at the capabilities of each host relative to the flavor of the VM instance being requested.

credentials Data that is only known to or accessible by a user and used to verify that the user is who he says he is. Credentials are presented to the server during authentication. Examples include a password, secret key, digital certificate, and fingerprint.

CRL A Certificate Revocation List (CRL) in a PKI model is a list of certificates that have been revoked. End entities presenting these certificates should not be trusted.

Cross-Origin Resource Sharing (CORS) A mechanism that allows many resources (for example, fonts, JavaScript) on a web page to be requested from another domain outside the domain from which the resource originated. In particular, JavaScript's AJAX calls can use the XMLHttpRequest mechanism.

Crowbar An open source community project by SUSE that aims to provide all necessary services to quickly deploy and manage clouds.

current workload An element of the Compute capacity cache that is calculated based on the number of build, snapshot, migrate, and resize operations currently in progress on a given host.

customer Alternative term for project.

customization module A user-created Python module that is loaded by horizon to change the look and feel of the dashboard.

D

daemon A process that runs in the background and waits for requests. May or may not listen on a TCP or UDP port. Do not confuse with a worker.

Dashboard (horizon) OpenStack project which provides an extensible, unified, web-based user interface for all OpenStack services.

data encryption Both Image service and Compute support encrypted virtual machine (VM) images (but not instances). In-transit data encryption is supported in OpenStack using technologies such as HTTPS, SSL, TLS, and SSH. Object Storage does not support object encryption at the application level but may support storage that uses disk encryption.

Data loss prevention (DLP) software Software programs used to protect sensitive information and prevent it from leaking outside a network boundary through the detection and denying of the data transportation.

Data Processing service (sahara) OpenStack project that provides a scalable data-processing stack and associated management interfaces.

data store A database engine supported by the Database service.

database ID A unique ID given to each replica of an Object Storage database.

database replicator An Object Storage component that copies changes in the account, container, and object databases to other nodes.

Database service (trove) An integrated project that provides scalable and reliable Cloud Database-as-a-Service functionality for both relational and non-relational database engines.

deallocate The process of removing the association between a floating IP address and a fixed IP address. Once this association is removed, the floating IP returns to the address pool.

Debian A Linux distribution that is compatible with OpenStack.

deduplication The process of finding duplicate data at the disk block, file, and/or object level to minimize storage use—currently unsupported within OpenStack.

default panel The default panel that is displayed when a user accesses the dashboard.

default project New users are assigned to this project if no project is specified when a user is created.

default token An Identity service token that is not associated with a specific project and is exchanged for a scoped token.

delayed delete An option within Image service so that an image is deleted after a predefined number of seconds instead of immediately.

delivery mode Setting for the Compute RabbitMQ message delivery mode; can be set to either transient or persistent.

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.

deprecated auth An option within Compute that enables administrators to create and manage users through the `nova-manage` command as opposed to using the Identity service.

designate Code name for the *DNS service*.

Desktop-as-a-Service A platform that provides a suite of desktop environments that users access to receive a desktop experience from any location. This may provide general use, development, or even homogeneous testing environments.

developer One of the default roles in the Compute RBAC system and the default role assigned to a new user.

device ID Maps Object Storage partitions to physical storage devices.

device weight Distributes partitions proportionately across Object Storage devices based on the storage capacity of each device.

DevStack Community project that uses shell scripts to quickly build complete OpenStack development environments.

DHCP agent OpenStack Networking agent that provides DHCP services for virtual networks.

Diablo A grouped release of projects related to OpenStack that came out in the fall of 2011, the fourth release of OpenStack. It included Compute (nova 2011.3), Object Storage (swift 1.4.3), and the Image service (glance). Diablo is the code name for the fourth release of OpenStack. The design summit took place in the Bay Area near Santa Clara, California, US and Diablo is a nearby city.

direct consumer An element of the Compute RabbitMQ that comes to life when a RPC call is executed. It connects to a direct exchange through a unique exclusive queue, sends the message, and terminates.

direct exchange A routing table that is created within the Compute RabbitMQ during RPC calls; one is created for each RPC call that is invoked.

direct publisher Element of RabbitMQ that provides a response to an incoming MQ message.

disassociate The process of removing the association between a floating IP address and fixed IP and thus returning the floating IP address to the address pool.

Discretionary Access Control (DAC) Governs the ability of subjects to access objects, while enabling users to make policy decisions and assign security attributes. The traditional UNIX system of users, groups, and read-write-execute permissions is an example of DAC.

disk encryption The ability to encrypt data at the file system, disk partition, or whole-disk level. Supported within Compute VMs.

disk format The underlying format that a disk image for a VM is stored as within the Image service back-end store. For example, AMI, ISO, QCOW2, VMDK, and so on.

dispersion In Object Storage, tools to test and ensure dispersion of objects and containers to ensure fault tolerance.

distributed virtual router (DVR) Mechanism for highly available multi-host routing when using OpenStack Networking (neutron).

Django A web framework used extensively in horizon.

DNS record A record that specifies information about a particular domain and belongs to the domain.

DNS service (designate) OpenStack project that provides scalable, on demand, self service access to authoritative DNS services, in a technology-agnostic manner.

dnsmasq Daemon that provides DNS, DHCP, BOOTP, and TFTP services for virtual networks.

domain An Identity API v3 entity. Represents a collection of projects, groups and users that defines administrative boundaries for managing OpenStack Identity entities. On the Internet, separates a website from other sites. Often, the domain name has two or more parts that are separated by dots. For example, yahoo.com, usa.gov, harvard.edu, or mail.yahoo.com. Also, a domain is an entity or container of all DNS-related information containing one or more records.

Domain Name System (DNS) A system by which Internet domain name-to-address and address-to-name resolutions are determined. DNS helps navigate the Internet by translating the IP address into an address that is easier to remember. For example, translating 111.111.111.1 into www.yahoo.com. All domains and their components, such as mail servers, utilize DNS to resolve to the appropriate locations. DNS servers are usually set up in a master-slave relationship such that failure of the master invokes the slave. DNS servers might also be clustered or replicated such that changes made to one DNS server are automatically propagated to other active servers. In Compute, the support that enables associating DNS entries with floating IP addresses, nodes, or cells so that hostnames are consistent across reboots.

download The transfer of data, usually in the form of files, from one computer to another.

durable exchange The Compute RabbitMQ message exchange that remains active when the server restarts.

durable queue A Compute RabbitMQ message queue that remains active when the server restarts.

Dynamic Host Configuration Protocol (DHCP) A network protocol that configures devices that are connected to a network so that they can communicate on that network by using the Internet Protocol (IP). The protocol is implemented in a client-server model where DHCP clients request configuration data, such as an IP address, a default route, and one or more DNS server addresses from a DHCP server. A method to automatically configure networking for a host at boot time. Provided by both Networking and Compute.

Dynamic HyperText Markup Language (DHTML) Pages that use HTML, JavaScript, and Cascading Style Sheets to enable users to interact with a web page or show simple animation.

E

east-west traffic Network traffic between servers in the same cloud or data center. See also north-south traffic.

EBS boot volume An Amazon EBS storage volume that contains a bootable VM image, currently unsupported in OpenStack.

ebtables Filtering tool for a Linux bridging firewall, enabling filtering of network traffic passing through a Linux bridge. Used in Compute along with arptables, iptables, and ip6tables to ensure isolation of network communications.

EC2 The Amazon commercial compute product, similar to Compute.

EC2 access key Used along with an EC2 secret key to access the Compute EC2 API.

EC2 API OpenStack supports accessing the Amazon EC2 API through Compute.

EC2 Compatibility API A Compute component that enables OpenStack to communicate with Amazon EC2.

EC2 secret key Used along with an EC2 access key when communicating with the Compute EC2 API; used to digitally sign each request.

Elastic Block Storage (EBS) The Amazon commercial block storage product.

encapsulation The practice of placing one packet type within another for the purposes of abstracting or securing data. Examples include GRE, MPLS, or IPsec.

encryption OpenStack supports encryption technologies such as HTTPS, SSH, SSL, TLS, digital certificates, and data encryption.

endpoint See API endpoint.

endpoint registry Alternative term for an Identity service catalog.

endpoint template A list of URL and port number endpoints that indicate where a service, such as Object Storage, Compute, Identity, and so on, can be accessed.

entity Any piece of hardware or software that wants to connect to the network services provided by Networking, the network connectivity service. An entity can make use of Networking by implementing a VIF.

ephemeral image A VM image that does not save changes made to its volumes and reverts them to their original state after the instance is terminated.

ephemeral volume Volume that does not save the changes made to it and reverts to its original state when the current user relinquishes control.

Essex A grouped release of projects related to OpenStack that came out in April 2012, the fifth release of OpenStack. It included Compute (nova 2012.1), Object Storage (swift 1.4.8), Image (glance), Identity (keystone), and Dashboard (horizon). Essex is the code name for the fifth release of OpenStack. The design summit took place in Boston, Massachusetts, US and Essex is a nearby city.

ESXi An OpenStack-supported hypervisor.

ETag MD5 hash of an object within Object Storage, used to ensure data integrity.

euca2ools A collection of command-line tools for administering VMs; most are compatible with OpenStack.

Eucalyptus Kernel Image (EKI) Used along with an ERI to create an EMI.

Eucalyptus Machine Image (EMI) VM image container format supported by Image service.

Eucalyptus Ramdisk Image (ERI) Used along with an EKI to create an EMI.

evacuate The process of migrating one or all virtual machine (VM) instances from one host to another, compatible with both shared storage live migration and block migration.

exchange Alternative term for a RabbitMQ message exchange.

exchange type A routing algorithm in the Compute RabbitMQ.

exclusive queue Connected to by a direct consumer in RabbitMQ—Compute, the message can be consumed only by the current connection.

extended attributes (xattr) File system option that enables storage of additional information beyond owner, group, permissions, modification time, and so on. The underlying Object Storage file system must support extended attributes.

extension Alternative term for an API extension or plug-in. In the context of Identity service, this is a call that is specific to the implementation, such as adding support for OpenID.

external network A network segment typically used for instance Internet access.

extra specs Specifies additional requirements when Compute determines where to start a new instance. Examples include a minimum amount of network bandwidth or a GPU.

F

FakeLDAP An easy method to create a local LDAP directory for testing Identity and Compute. Requires Redis.

fan-out exchange Within RabbitMQ and Compute, it is the messaging interface that is used by the scheduler service to receive capability messages from the compute, volume, and network nodes.

federated identity A method to establish trusts between identity providers and the OpenStack cloud.

Fedora A Linux distribution compatible with OpenStack.

Fibre Channel Storage protocol similar in concept to TCP/IP; encapsulates SCSI commands and data.

Fibre Channel over Ethernet (FCoE) The fibre channel protocol tunneled within Ethernet.

fill-first scheduler The Compute scheduling method that attempts to fill a host with VMs rather than starting new VMs on a variety of hosts.

filter The step in the Compute scheduling process when hosts that cannot run VMs are eliminated and not chosen.

firewall Used to restrict communications between hosts and/or nodes, implemented in Compute using iptables, arptables, ip6tables, and ebtables.

FireWall-as-a-Service (FWaaS) A Networking extension that provides perimeter firewall functionality.

fixed IP address An IP address that is associated with the same instance each time that instance boots, is generally not accessible to end users or the public Internet, and is used for management of the instance.

Flat Manager The Compute component that gives IP addresses to authorized nodes and assumes DHCP, DNS, and routing configuration and services are provided by something else.

flat mode injection A Compute networking method where the OS network configuration information is injected into the VM image before the instance starts.

flat network Virtual network type that uses neither VLANs nor tunnels to segregate project traffic. Each flat network typically requires a separate underlying physical interface defined by bridge mappings. However, a flat network can contain multiple subnets.

FlatDHCP Manager The Compute component that provides dnsmasq (DHCP, DNS, BOOTP, TFTP) and radvd (routing) services.

flavor Alternative term for a VM instance type.

flavor ID UUID for each Compute or Image service VM flavor or instance type.

floating IP address An IP address that a project can associate with a VM so that the instance has the same public IP address each time that it boots. You create a pool of floating IP addresses and assign them to instances as they are launched to maintain a consistent IP address for maintaining DNS assignment.

Folsom A grouped release of projects related to OpenStack that came out in the fall of 2012, the sixth release of OpenStack. It includes Compute (nova), Object Storage (swift), Identity (keystone), Networking (neutron), Image service (glance), and Volumes or Block Storage (cinder). Folsom is the code name for the sixth release of OpenStack. The design summit took place in San Francisco, California, US and Folsom is a nearby city.

FormPost Object Storage middleware that uploads (posts) an image through a form on a web page.

freezer Code name for the *Backup, Restore, and Disaster Recovery service*.

front end The point where a user interacts with a service; can be an API endpoint, the dashboard, or a command-line tool.

G

gateway An IP address, typically assigned to a router, that passes network traffic between different networks.

generic receive offload (GRO) Feature of certain network interface drivers that combines many smaller received packets into a large packet before delivery to the kernel IP stack.

generic routing encapsulation (GRE) Protocol that encapsulates a wide variety of network layer protocols inside virtual point-to-point links.

glance Codename for the *Image service*.

glance API server Alternative name for the *Image API*.

glance registry Alternative term for the Image service *image registry*.

global endpoint template The Identity service endpoint template that contains services available to all projects.

GlusterFS A file system designed to aggregate NAS hosts, compatible with OpenStack.

gnocchi Part of the OpenStack *Telemetry service*; provides an indexer and time-series database.

golden image A method of operating system installation where a finalized disk image is created and then used by all nodes without modification.

Governance service (congress) The project that provides Governance-as-a-Service across any collection of cloud services in order to monitor, enforce, and audit policy over dynamic infrastructure.

Graphic Interchange Format (GIF) A type of image file that is commonly used for animated images on web pages.

Graphics Processing Unit (GPU) Choosing a host based on the existence of a GPU is currently unsupported in OpenStack.

Green Threads The cooperative threading model used by Python; reduces race conditions and only context switches when specific library calls are made. Each OpenStack service is its own thread.

Grizzly The code name for the seventh release of OpenStack. The design summit took place in San Diego, California, US and Grizzly is an element of the state flag of California.

Group An Identity v3 API entity. Represents a collection of users that is owned by a specific domain.

guest OS An operating system instance running under the control of a hypervisor.

H

Hadoop Apache Hadoop is an open source software framework that supports data-intensive distributed applications.

Hadoop Distributed File System (HDFS) A distributed, highly fault-tolerant file system designed to run on low-cost commodity hardware.

handover An object state in Object Storage where a new replica of the object is automatically created due to a drive failure.

HAProxy Provides a load balancer for TCP and HTTP-based applications that spreads requests across multiple servers.

hard reboot A type of reboot where a physical or virtual power button is pressed as opposed to a graceful, proper shutdown of the operating system.

Havana The code name for the eighth release of OpenStack. The design summit took place in Portland, Oregon, US and Havana is an unincorporated community in Oregon.

health monitor Determines whether back-end members of a VIP pool can process a request. A pool can have several health monitors associated with it. When a pool has several monitors associated with it, all monitors check each member of the pool. All monitors must declare a member to be healthy for it to stay active.

heat Codename for the *Orchestration service*.

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 seek to minimize system downtime and data loss.

horizon Codename for the *Dashboard*.

horizon plug-in A plug-in for the OpenStack Dashboard (horizon).

host A physical computer, not a VM instance (node).

host aggregate A method to further subdivide availability zones into hypervisor pools, a collection of common hosts.

Host Bus Adapter (HBA) Device plugged into a PCI slot, such as a fibre channel or network card.

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.

Hyper-V One of the hypervisors supported by OpenStack.

hyperlink Any kind of text that contains a link to some other site, commonly found in documents where clicking on a word or words opens up a different website.

Hypertext Transfer Protocol (HTTP) An application protocol for distributed, collaborative, hypermedia information systems. It is the foundation of data communication for the World Wide Web. Hypertext is structured text that uses logical links (hyperlinks) between nodes containing text. HTTP is the protocol to exchange or transfer hypertext.

Hypertext Transfer Protocol Secure (HTTPS) An encrypted communications protocol for secure communication over a computer network, with especially wide deployment on the Internet. Technically, it is not a protocol in and of itself; rather, it is the result of simply layering the Hypertext Transfer Protocol (HTTP) on top of the TLS or SSL protocol, thus adding the security capabilities of TLS or SSL to standard HTTP communications. Most OpenStack API endpoints and many inter-component communications support HTTPS communication.

hypervisor Software that arbitrates and controls VM access to the actual underlying hardware.

hypervisor pool A collection of hypervisors grouped together through host aggregates.

I

Icehouse The code name for the ninth release of OpenStack. The design summit took place in Hong Kong and Ice House is a street in that city.

ID number Unique numeric ID associated with each user in Identity, conceptually similar to a Linux or LDAP UID.

Identity API Alternative term for the Identity service API.

Identity back end The source used by Identity service to retrieve user information; an OpenLDAP server, for example.

identity provider A directory service, which allows users to login with a user name and password. It is a typical source of authentication tokens.

Identity service (keystone) The project that facilitates API client authentication, service discovery, distributed multi-project authorization, and auditing. It provides a central directory of users mapped to the OpenStack services they can access. It also registers endpoints for OpenStack services and acts as a common authentication system.

Identity service API The API used to access the OpenStack Identity service provided through keystone.

IETF Internet Engineering Task Force (IETF) is an open standards organization that develops Internet standards, particularly the standards pertaining to TCP/IP.

image A collection of files for a specific operating system (OS) that you use to create or rebuild a server. OpenStack provides pre-built images. You can also create custom images, or snapshots, from servers that you have launched. Custom images can be used for data backups or as "gold" images for additional servers.

Image API The Image service API endpoint for management of VM images. Processes client requests for VMs, updates Image service metadata on the registry server, and communicates with the store adapter to upload VM images from the back-end store.

image cache Used by Image service to obtain images on the local host rather than re-downloading them from the image server each time one is requested.

image ID Combination of a URI and UUID used to access Image service VM images through the image API.

image membership A list of projects that can access a given VM image within Image service.

image owner The project who owns an Image service virtual machine image.

image registry A list of VM images that are available through Image service.

Image service (glance) The OpenStack service that provide services and associated libraries to store, browse, share, distribute and manage bootable disk images, other data closely associated with initializing compute resources, and metadata definitions.

image status The current status of a VM image in Image service, not to be confused with the status of a running instance.

image store The back-end store used by Image service to store VM images, options include Object Storage, locally mounted file system, RADOS block devices, VMware datastore, or HTTP.

image UUID UUID used by Image service to uniquely identify each VM image.

incubated project A community project may be elevated to this status and is then promoted to a core project.

Infrastructure Optimization service (watcher) OpenStack project that aims to provide a flexible and scalable resource optimization service for multi-project OpenStack-based clouds.

Infrastructure-as-a-Service (IaaS) 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.

ingress filtering The process of filtering incoming network traffic. Supported by Compute.

INI format The OpenStack configuration files use an INI format to describe options and their values. It consists of sections and key value pairs.

injection The process of putting a file into a virtual machine image before the instance is started.

Input/Output Operations Per Second (IOPS) IOPS are a common performance measurement used to benchmark computer storage devices like hard disk drives, solid state drives, and storage area networks.

instance A running VM, or a VM in a known state such as suspended, that can be used like a hardware server.

instance ID Alternative term for instance UUID.

instance state The current state of a guest VM image.

instance tunnels network A network segment used for instance traffic tunnels between compute nodes and the network node.

instance type Describes the parameters of the various virtual machine images that are available to users; includes parameters such as CPU, storage, and memory. Alternative term for flavor.

instance type ID Alternative term for a flavor ID.

instance UUID Unique ID assigned to each guest VM instance.

Intelligent Platform Management Interface (IPMI) IPMI is a standardized computer system interface used by system administrators for out-of-band management of computer systems and monitoring of their operation. In layman's terms, it is a way to manage a computer using a direct network connection, whether it is turned on or not; connecting to the hardware rather than an operating system or login shell.

interface A physical or virtual device that provides connectivity to another device or medium.

interface ID Unique ID for a Networking VIF or vNIC in the form of a UUID.

Internet Control Message Protocol (ICMP) A network protocol used by network devices for control messages. For example, `ping` uses ICMP to test connectivity.

Internet protocol (IP) Principal communications protocol in the internet protocol suite for relaying datagrams across network boundaries.

Internet Service Provider (ISP) Any business that provides Internet access to individuals or businesses.

Internet Small Computer System Interface (iSCSI) Storage protocol that encapsulates SCSI frames for transport over IP networks. Supported by Compute, Object Storage, and Image service.

IP address Number that is unique to every computer system on the Internet. Two versions of the Internet Protocol (IP) are in use for addresses: IPv4 and IPv6.

IP Address Management (IPAM) The process of automating IP address allocation, deallocation, and management. Currently provided by Compute, melange, and Networking.

ip6tables Tool used to set up, maintain, and inspect the tables of IPv6 packet filter rules in the Linux kernel. In OpenStack Compute, ip6tables is used along with arptables, ebtables, and iptables to create firewalls for both nodes and VMs.

ipset Extension to iptables that allows creation of firewall rules that match entire "sets" of IP addresses simultaneously. These sets reside in indexed data structures to increase efficiency, particularly on systems with a large quantity of rules.

iptables Used along with arptables and ebtables, iptables create firewalls in Compute. iptables are the tables provided by the Linux kernel firewall (implemented as different Netfilter modules) and the chains and rules it stores. Different kernel modules and programs are currently used for different protocols: iptables applies to IPv4, ip6tables to IPv6, arptables to ARP, and ebtables to Ethernet frames. Requires root privilege to manipulate.

ironic Codename for the *Bare Metal service*.

iSCSI Qualified Name (IQN) IQN is the format most commonly used for iSCSI names, which uniquely identify nodes in an iSCSI network. All IQNs follow the pattern iqn.yyyy-mm.domain:identifier, where 'yyyy-mm' is the year and month in which the domain was registered, 'domain' is the reversed domain name of the issuing organization, and 'identifier' is an optional string which makes each IQN under the same domain unique. For example, 'iqn.2015-10.org.openstack.408ae959bce1'.

ISO9660 One of the VM image disk formats supported by Image service.

itsec A default role in the Compute RBAC system that can quarantine an instance in any project.

J

Java A programming language that is used to create systems that involve more than one computer by way of a network.

JavaScript A scripting language that is used to build web pages.

JavaScript Object Notation (JSON) One of the supported response formats in OpenStack.

jumbo frame Feature in modern Ethernet networks that supports frames up to approximately 9000 bytes.

Juno The code name for the tenth release of OpenStack. The design summit took place in Atlanta, Georgia, US and Juno is an unincorporated community in Georgia.

K

Kerberos A network authentication protocol which works on the basis of tickets. Kerberos allows nodes communication over a non-secure network, and allows nodes to prove their identity to one another in a secure manner.

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.

Key Manager service (barbican) The project that produces a secret storage and generation system capable of providing key management for services wishing to enable encryption features.

keystone Codename of the *Identity service*.

Kickstart A tool to automate system configuration and installation on Red Hat, Fedora, and CentOS-based Linux distributions.

Kilo The code name for the eleventh release of OpenStack. The design summit took place in Paris, France. Due to delays in the name selection, the release was known only as K. Because `k` is the unit symbol for kilo and the kilogram reference artifact is stored near Paris in the Pavillon de Breteuil in Sèvres, the community chose Kilo as the release name.

L

large object An object within Object Storage that is larger than 5 GB.

Launchpad The collaboration site for OpenStack.

Layer-2 (L2) agent OpenStack Networking agent that provides layer-2 connectivity for virtual networks.

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 errors that may occur in the physical layer.

Layer-3 (L3) agent OpenStack Networking agent that provides layer-3 (routing) services for virtual networks.

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.

Liberty The code name for the twelfth release of OpenStack. The design summit took place in Vancouver, Canada and Liberty is the name of a village in the Canadian province of Saskatchewan.

libvirt Virtualization API library used by OpenStack to interact with many of its supported hypervisors.

Lightweight Directory Access Protocol (LDAP) An application protocol for accessing and maintaining distributed directory information services over an IP network.

Linux Unix-like computer operating system assembled under the model of free and open-source software development and distribution.

Linux bridge Software that enables multiple VMs to share a single physical NIC within Compute.

Linux Bridge neutron plug-in Enables a Linux bridge to understand a Networking port, interface attachment, and other abstractions.

Linux containers (LXC) An OpenStack-supported hypervisor.

live migration The ability within Compute to move running virtual machine instances from one host to another with only a small service interruption during switchover.

load balancer A load balancer is a logical device that belongs to a cloud account. It is used to distribute workloads between multiple back-end systems or services, based on the criteria defined as part of its configuration.

load balancing The process of spreading client requests between two or more nodes to improve performance and availability.

Load-Balancer-as-a-Service (LBaaS) Enables Networking to distribute incoming requests evenly between designated instances.

Load-balancing service (octavia) The project that aims to provide scalable, on demand, self service access to load-balancer services, in technology-agnostic manner.

Logical Volume Manager (LVM) Provides a method of allocating space on mass-storage devices that is more flexible than conventional partitioning schemes.

M

magnum Code name for the *Containers Infrastructure Management service*.

management API Alternative term for an admin API.

management network A network segment used for administration, not accessible to the public Internet.

manager Logical groupings of related code, such as the Block Storage volume manager or network manager.

manifest Used to track segments of a large object within Object Storage.

manifest object A special Object Storage object that contains the manifest for a large object.

manila Codename for OpenStack *Shared File Systems service*.

manila-share Responsible for managing Shared File System Service devices, specifically the back-end devices.

maximum transmission unit (MTU) Maximum frame or packet size for a particular network medium. Typically 1500 bytes for Ethernet networks.

mechanism driver A driver for the Modular Layer 2 (ML2) neutron plug-in that provides layer-2 connectivity for virtual instances. A single OpenStack installation can use multiple mechanism drivers.

melange Project name for OpenStack Network Information Service. To be merged with Networking.

membership The association between an Image service VM image and a project. Enables images to be shared with specified projects.

membership list A list of projects that can access a given VM image within Image service.

memcached A distributed memory object caching system that is used by Object Storage for caching.

memory overcommit The ability to start new VM instances based on the actual memory usage of a host, as opposed to basing the decision on the amount of RAM each running instance thinks it has available. Also known as RAM overcommit.

message broker The software package used to provide AMQP messaging capabilities within Compute. Default package is RabbitMQ.

message bus The main virtual communication line used by all AMQP messages for inter-cloud communications within Compute.

message queue Passes requests from clients to the appropriate workers and returns the output to the client after the job completes.

Message service (zaqar) The project that provides a messaging service that affords a variety of distributed application patterns in an efficient, scalable and highly available manner, and to create and maintain associated Python libraries and documentation.

Meta-Data Server (MDS) Stores CephFS metadata.

Metadata agent OpenStack Networking agent that provides metadata services for instances.

migration The process of moving a VM instance from one host to another.

mistral Code name for *Workflow service*.

Mitaka The code name for the thirteenth release of OpenStack. The design summit took place in Tokyo, Japan. Mitaka is a city in Tokyo.

Modular Layer 2 (ML2) neutron plug-in Can concurrently use multiple layer-2 networking technologies, such as 802.1Q and VXLAN, in Networking.

monasca Codename for OpenStack *Monitoring*.

Monitor (LBaaS) LBaaS feature that provides availability monitoring using the `ping` command, TCP, and HTTP/HTTPS GET.

Monitor (Mon) A Ceph component that communicates with external clients, checks data state and consistency, and performs quorum functions.

Monitoring (monasca) The OpenStack service that provides a multi-project, highly scalable, performant, fault-tolerant monitoring-as-a-service solution for metrics, complex event processing and logging. To build an extensible platform for advanced monitoring services that can be used by both operators and projects to gain operational insight and visibility, ensuring availability and stability.

multi-factor authentication Authentication method that uses two or more credentials, such as a password and a private key. Currently not supported in Identity.

multi-host High-availability mode for legacy (nova) networking. Each compute node handles NAT and DHCP and acts as a gateway for all of the VMs on it. A networking failure on one compute node doesn't affect VMs on other compute nodes.

multinic Facility in Compute that allows each virtual machine instance to have more than one VIF connected to it.

murano Codename for the *Application Catalog service*.

N

Nebula Released as open source by NASA in 2010 and is the basis for Compute.

netadmin One of the default roles in the Compute RBAC system. Enables the user to allocate publicly accessible IP addresses to instances and change firewall rules.

NetApp volume driver Enables Compute to communicate with NetApp storage devices through the NetApp OnCommand Provisioning Manager.

network A virtual network that provides connectivity between entities. For example, a collection of virtual ports that share network connectivity. In Networking terminology, a network is always a layer-2 network.

Network Address Translation (NAT) Process of modifying IP address information while in transit. Supported by Compute and Networking.

network controller A Compute daemon that orchestrates the network configuration of nodes, including IP addresses, VLANs, and bridging. Also manages routing for both public and private networks.

Network File System (NFS) A method for making file systems available over the network. Supported by OpenStack.

network ID Unique ID assigned to each network segment within Networking. Same as network UUID.

network manager The Compute component that manages various network components, such as firewall rules, IP address allocation, and so on.

network namespace Linux kernel feature that provides independent virtual networking instances on a single host with separate routing tables and interfaces. Similar to virtual routing and forwarding (VRF) services on physical network equipment.

network node Any compute node that runs the network worker daemon.

network segment Represents a virtual, isolated OSI layer-2 subnet in Networking.

Network Service Header (NSH) Provides a mechanism for metadata exchange along the instantiated service path.

Network Time Protocol (NTP) Method of keeping a clock for a host or node correct via communication with a trusted, accurate time source.

network UUID Unique ID for a Networking network segment.

network worker The `nova-network` worker daemon; provides services such as giving an IP address to a booting nova instance.

Networking API (Neutron API) API used to access OpenStack Networking. Provides an extensible architecture to enable custom plug-in creation.

Networking service (neutron) The OpenStack project which implements services and associated libraries to provide on-demand, scalable, and technology-agnostic network abstraction.

neutron Codename for OpenStack *Networking service*.

neutron API An alternative name for *Networking API*.

neutron manager Enables Compute and Networking integration, which enables Networking to perform network management for guest VMs.

neutron plug-in Interface within Networking that enables organizations to create custom plug-ins for advanced features, such as QoS, ACLs, or IDS.

Newton The code name for the fourteenth release of OpenStack. The design summit took place in Austin, Texas, US. The release is named after "Newton House" which is located at 1013 E. Ninth St., Austin, TX. which is listed on the National Register of Historic Places.

Nexenta volume driver Provides support for NexentaStor devices in Compute.

NFV Orchestration Service (tacker) OpenStack service that aims to implement Network Function Virtualization (NFV) orchestration services and libraries for end-to-end life-cycle management of network services and Virtual Network Functions (VNFs).

Nginx An HTTP and reverse proxy server, a mail proxy server, and a generic TCP/UDP proxy server.

No ACK Disables server-side message acknowledgment in the Compute RabbitMQ. Increases performance but decreases reliability.

node A VM instance that runs on a host.

non-durable exchange Message exchange that is cleared when the service restarts. Its data is not written to persistent storage.

non-durable queue Message queue that is cleared when the service restarts. Its data is not written to persistent storage.

non-persistent volume Alternative term for an ephemeral volume.

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 Codename for OpenStack *Compute service*.

Nova API Alternative term for the *Compute API*.

nova-network A Compute component that manages IP address allocation, firewalls, and other network-related tasks. This is the legacy networking option and an alternative to Networking.

O

object A BLOB of data held by Object Storage; can be in any format.

object auditor Opens all objects for an object server and verifies the MD5 hash, size, and metadata for each object.

object expiration A configurable option within Object Storage to automatically delete objects after a specified amount of time has passed or a certain date is reached.

object hash Unique ID for an Object Storage object.

object path hash Used by Object Storage to determine the location of an object in the ring. Maps objects to partitions.

object replicator An Object Storage component that copies an object to remote partitions for fault tolerance.

object server An Object Storage component that is responsible for managing objects.

Object Storage API API used to access OpenStack *Object Storage*.

Object Storage Device (OSD) The Ceph storage daemon.

Object Storage service (swift) The OpenStack core project that provides eventually consistent and redundant storage and retrieval of fixed digital content.

object versioning Allows a user to set a flag on an *Object Storage* container so that all objects within the container are versioned.

Ocata The code name for the fifteenth release of OpenStack. The design summit will take place in Barcelona, Spain. Ocata is a beach north of Barcelona.

Octavia Code name for the *Load-balancing service*.

Oldie Term for an *Object Storage* process that runs for a long time. Can indicate a hung process.

Open Cloud Computing Interface (OCCI) A standardized interface for managing compute, data, and network resources, currently unsupported in OpenStack.

Open Virtualization Format (OVF) Standard for packaging VM images. Supported in OpenStack.

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).

Open vSwitch (OVS) agent Provides an interface to the underlying Open vSwitch service for the Networking plug-in.

Open vSwitch neutron plug-in Provides support for Open vSwitch in Networking.

OpenLDAP An open source LDAP server. Supported by both Compute and Identity.

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.

OpenStack code name Each OpenStack release has a code name. Code names ascend in alphabetical order: Austin, Bexar, Cactus, Diablo, Essex, Folsom, Grizzly, Havana, Icehouse, Juno, Kilo, Liberty, Mitaka, Newton, Ocata, Pike, Queens, and Rocky. Code names are cities or counties near where the corresponding OpenStack design summit took place. An exception, called the Waldon exception, is granted to elements of the state flag that sound especially cool. Code names are chosen by popular vote.

openSUSE A Linux distribution that is compatible with OpenStack.

operator The person responsible for planning and maintaining an OpenStack installation.

optional service An official OpenStack service defined as optional by DefCore Committee. Currently, consists of Dashboard (horizon), Telemetry service (Telemetry), Orchestration service (heat), Database service (trove), Bare Metal service (ironic), and so on.

Orchestration service (heat) The OpenStack service which orchestrates composite cloud applications using a declarative template format through an OpenStack-native REST API.

orphan In the context of Object Storage, this is a process that is not terminated after an upgrade, restart, or reload of the service.

Oslo Codename for the *Common Libraries project*.

P

panko Part of the OpenStack *Telemetry service*; provides event storage.

parent cell If a requested resource, such as CPU time, disk storage, or memory, is not available in the parent cell, the request is forwarded to associated child cells.

partition A unit of storage within Object Storage used to store objects. It exists on top of devices and is replicated for fault tolerance.

partition index Contains the locations of all Object Storage partitions within the ring.

partition shift value Used by Object Storage to determine which partition data should reside on.

path MTU discovery (PMTUD) Mechanism in IP networks to detect end-to-end MTU and adjust packet size accordingly.

pause A VM state where no changes occur (no changes in memory, network communications stop, etc); the VM is frozen but not shut down.

PCI passthrough Gives guest VMs exclusive access to a PCI device. Currently supported in OpenStack Havana and later releases.

persistent message A message that is stored both in memory and on disk. The message is not lost after a failure or restart.

persistent volume Changes to these types of disk volumes are saved.

personality file A file used to customize a Compute instance. It can be used to inject SSH keys or a specific network configuration.

Pike The code name for the sixteenth release of OpenStack. The design summit will take place in Boston, Massachusetts, US. The release is named after the Massachusetts Turnpike, abbreviated commonly as the Mass Pike, which is the easternmost stretch of Interstate 90.

Platform-as-a-Service (PaaS) Provides to the consumer an operating system and, often, a language runtime and libraries (collectively, the "platform") upon which they can run their own application code, without providing any control over the underlying infrastructure. Examples of Platform-as-a-Service providers include Cloud Foundry and OpenShift.

plug-in Software component providing the actual implementation for Networking APIs, or for Compute APIs, depending on the context.

policy service Component of Identity that provides a rule-management interface and a rule-based authorization engine.

policy-based routing (PBR) Provides a mechanism to implement packet forwarding and routing according to the policies defined by the network administrator.

pool A logical set of devices, such as web servers, that you group together to receive and process traffic. The load balancing function chooses which member of the pool handles the new requests or connections received on the VIP address. Each VIP has one pool.

pool member An application that runs on the back-end server in a load-balancing system.

port A virtual network port within Networking; VIFs / vNICs are connected to a port.

port UUID Unique ID for a Networking port.

preseed A tool to automate system configuration and installation on Debian-based Linux distributions.

private image An Image service VM image that is only available to specified projects.

private IP address An IP address used for management and administration, not available to the public Internet.

private network The Network Controller provides virtual networks to enable compute servers to interact with each other and with the public network. All machines must have a public and private network interface. A private network interface can be a flat or VLAN network interface. A flat network interface is controlled by the flat_interface with flat managers. A VLAN network interface is controlled by the `vlan_interface` option with VLAN managers.

project Projects represent the base unit of "ownership" in OpenStack, in that all resources in OpenStack should be owned by a specific project. In OpenStack Identity, a project must be owned by a specific domain.

project ID Unique ID assigned to each project by the Identity service.

project VPN Alternative term for a cloudpipe.

promiscuous mode Causes the network interface to pass all traffic it receives to the host rather than passing only the frames addressed to it.

protected property Generally, extra properties on an Image service image to which only cloud administrators have access. Limits which user roles can perform CRUD operations on that property. The cloud administrator can configure any image property as protected.

provider An administrator who has access to all hosts and instances.

proxy node A node that provides the Object Storage proxy service.

proxy server Users of Object Storage interact with the service through the proxy server, which in turn looks up the location of the requested data within the ring and returns the results to the user.

public API An API endpoint used for both service-to-service communication and end-user interactions.

public image An Image service VM image that is available to all projects.

public IP address An IP address that is accessible to end-users.

public key authentication Authentication method that uses keys rather than passwords.

public network The Network Controller provides virtual networks to enable compute servers to interact with each other and with the public network. All machines must have a public and private network interface. The public network interface is controlled by the `public_interface` option.

Puppet An operating system configuration-management tool supported by OpenStack.

Python Programming language used extensively in OpenStack.

Q

QEMU Copy On Write 2 (QCOW2) One of the VM image disk formats supported by Image service.

Qpid Message queue software supported by OpenStack; an alternative to RabbitMQ.

Quality of Service (QoS) The ability to guarantee certain network or storage requirements to satisfy a Service Level Agreement (SLA) between an application provider and end users. Typically includes performance requirements like networking bandwidth, latency, jitter correction, and reliability as well as storage performance in Input/Output Operations Per Second (IOPS), throttling agreements, and performance expectations at peak load.

quarantine If Object Storage finds objects, containers, or accounts that are corrupt, they are placed in this state, are not replicated, cannot be read by clients, and a correct copy is re-replicated.

Queens The code name for the seventeenth release of OpenStack. The design summit will take place in Sydney, Australia. The release is named after the Queens Pound river in the South Coast region of New South Wales.

Quick EMUlator (QEMU) QEMU is a generic and open source machine emulator and virtualizer. One of the hypervisors supported by OpenStack, generally used for development purposes.

quota In Compute and Block Storage, the ability to set resource limits on a per-project basis.

R

RabbitMQ The default message queue software used by OpenStack.

Rackspace Cloud Files Released as open source by Rackspace in 2010; the basis for Object Storage.

RADOS Block Device (RBD) Ceph component that enables a Linux block device to be striped over multiple distributed data stores.

radvd The router advertisement daemon, used by the Compute VLAN manager and FlatDHCP manager to provide routing services for VM instances.

rally Codename for the *Benchmark service*.

RAM filter The Compute setting that enables or disables RAM overcommitment.

RAM overcommit The ability to start new VM instances based on the actual memory usage of a host, as opposed to basing the decision on the amount of RAM each running instance thinks it has available. Also known as memory overcommit.

rate limit Configurable option within Object Storage to limit database writes on a per-account and/or per-container basis.

raw One of the VM image disk formats supported by Image service; an unstructured disk image.

rebalance The process of distributing Object Storage partitions across all drives in the ring; used during initial ring creation and after ring reconfiguration.

reboot Either a soft or hard reboot of a server. With a soft reboot, the operating system is signaled to restart, which enables a graceful shutdown of all processes. A hard reboot is the equivalent of power cycling the server. The virtualization platform should ensure that the reboot action has completed successfully, even in cases in which the underlying domain/VM is paused or halted/stopped.

rebuild Removes all data on the server and replaces it with the specified image. Server ID and IP addresses remain the same.

Recon An Object Storage component that collects meters.

record Belongs to a particular domain and is used to specify information about the domain. There are several types of DNS records. Each record type contains particular information used to describe the purpose of that record. Examples include mail exchange (MX) records, which specify the mail server for a particular domain; and name server (NS) records, which specify the authoritative name servers for a domain.

record ID A number within a database that is incremented each time a change is made. Used by Object Storage when replicating.

Red Hat Enterprise Linux (RHEL) A Linux distribution that is compatible with OpenStack.

reference architecture A recommended architecture for an OpenStack cloud.

region A discrete OpenStack environment with dedicated API endpoints that typically shares only the Identity (keystone) with other regions.

registry Alternative term for the Image service registry.

registry server An Image service that provides VM image metadata information to clients.

Reliable, Autonomic Distributed Object Store (RADOS)

A collection of components that provides object storage within Ceph. Similar to OpenStack Object Storage.

Remote Procedure Call (RPC) The method used by the Compute RabbitMQ for intra-service communications.

replica Provides data redundancy and fault tolerance by creating copies of Object Storage objects, accounts, and containers so that they are not lost when the underlying storage fails.

replica count The number of replicas of the data in an Object Storage ring.

replication The process of copying data to a separate physical device for fault tolerance and performance.

replicator The Object Storage back-end process that creates and manages object replicas.

request ID Unique ID assigned to each request sent to Compute.

rescue image A special type of VM image that is booted when an instance is placed into rescue mode. Allows an administrator to mount the file systems for an instance to correct the problem.

resize Converts an existing server to a different flavor, which scales the server up or down. The original server is saved to enable rollback if a problem occurs. All resizes must be tested and explicitly confirmed, at which time the original server is removed.

RESTful A kind of web service API that uses REST, or Representational State Transfer. REST is the style of architecture for hypermedia systems that is used for the World Wide Web.

ring An entity that maps Object Storage data to partitions. A separate ring exists for each service, such as account, object, and container.

ring builder Builds and manages rings within Object Storage, assigns partitions to devices, and pushes the configuration to other storage nodes.

Rocky The code name for the eightteenth release of OpenStack. The design summit will take place in Vancouver, Kanada. The release is named after the Rocky Mountains.

role A personality that a user assumes to perform a specific set of operations. A role includes a set of rights and privileges. A user assuming that role inherits those rights and privileges.

Role Based Access Control (RBAC) Provides a predefined list of actions that the user can perform, such as start or stop VMs, reset passwords, and so on. Supported in both Identity and Compute and can be configured using the dashboard.

role ID Alphanumeric ID assigned to each Identity service role.

Root Cause Analysis (RCA) service (Vitrage) OpenStack project that aims to organize, analyze and visualize OpenStack alarms and events, yield insights regarding the root cause of problems and deduce their existence before they are directly detected.

rootwrap A feature of Compute that allows the unprivileged "nova" user to run a specified list of commands as the Linux root user.

round-robin scheduler Type of Compute scheduler that evenly distributes instances among available hosts.

router A physical or virtual network device that passes network traffic between different networks.

routing key The Compute direct exchanges, fanout exchanges, and topic exchanges use this key to determine how to process a message; processing varies depending on exchange type.

RPC driver Modular system that allows the underlying message queue software of Compute to be changed. For example, from RabbitMQ to ZeroMQ or Qpid.

rsync Used by Object Storage to push object replicas.

RXTX cap Absolute limit on the amount of network traffic a Compute VM instance can send and receive.

RXTX quota Soft limit on the amount of network traffic a Compute VM instance can send and receive.

S

sahara Codename for the *Data Processing service*.

SAML assertion Contains information about a user as provided by the identity provider. It is an indication that a user has been authenticated.

scheduler manager A Compute component that determines where VM instances should start. Uses modular design to support a variety of scheduler types.

scoped token An Identity service API access token that is associated with a specific project.

scrubber Checks for and deletes unused VMs; the component of Image service that implements delayed delete.

secret key String of text known only by the user; used along with an access key to make requests to the Compute API.

secure boot Process whereby the system firmware validates the authenticity of the code involved in the boot process.

secure shell (SSH) Open source tool used to access remote hosts through an encrypted communications channel, SSH key injection is supported by Compute.

security group A set of network traffic filtering rules that are applied to a Compute instance.

segmented object An Object Storage large object that has been broken up into pieces. The re-assembled object is called a concatenated object.

self-service For IaaS, ability for a regular (non-privileged) account to manage a virtual infrastructure component such as networks without involving an administrator.

SELinux Linux kernel security module that provides the mechanism for supporting access control policies.

senlin Code name for the *Clustering service*.

server Computer that provides explicit services to the client software running on that system, often managing a variety of computer operations. A server is a VM instance in the Compute system. Flavor and image are requisite elements when creating a server.

server image Alternative term for a VM image.

server UUID Unique ID assigned to each guest VM instance.

service An OpenStack service, such as Compute, Object Storage, or Image service. Provides one or more endpoints through which users can access resources and perform operations.

service catalog Alternative term for the Identity service catalog.

Service Function Chain (SFC) For a given service, SFC is the abstracted view of the required service functions and the order in which they are to be applied.

service ID Unique ID assigned to each service that is available in the Identity service catalog.

Service Level Agreement (SLA) Contractual obligations that ensure the availability of a service.

service project Special project that contains all services that are listed in the catalog.

service provider A system that provides services to other system entities. In case of federated identity, OpenStack Identity is the service provider.

service registration An Identity service feature that enables services, such as Compute, to automatically register with the catalog.

service token An administrator-defined token used by Compute to communicate securely with the Identity service.

session back end The method of storage used by horizon to track client sessions, such as local memory, cookies, a database, or memcached.

session persistence A feature of the load-balancing service. It attempts to force subsequent connections to a service to be redirected to the same node as long as it is online.

session storage A horizon component that stores and tracks client session information. Implemented through the Django sessions framework.

share A remote, mountable file system in the context of the *Shared File Systems service*. You can mount a share to, and access a share from, several hosts by several users at a time.

share network An entity in the context of the *Shared File Systems service* that encapsulates interaction with the Networking service. If the driver you selected runs in the mode requiring such kind of interaction, you need to specify the share network to create a share.

Shared File Systems API A Shared File Systems service that provides a stable RESTful API. The service authenticates and routes requests throughout the Shared File Systems service. There is python-manilaclient to interact with the API.

Shared File Systems service (manila) The service that provides a set of services for management of shared file systems in a multi-project cloud environment, similar to how OpenStack provides block-based storage management through the OpenStack *Block Storage service* project. With the Shared File Systems service, you can create a remote file system and mount the file system on your instances. You can also read and write data from your instances to and from your file system.

shared IP address An IP address that can be assigned to a VM instance within the shared IP group. Public IP addresses can be shared across multiple servers for use in various high-availability scenarios. When an IP address is shared to another server, the cloud network restrictions are modified to enable each server to listen to and respond on that IP address. You can optionally specify that the target server network configuration be modified. Shared IP addresses can be used with many standard heartbeat facilities, such as keepalive, that monitor for failure and manage IP failover.

shared IP group A collection of servers that can share IPs with other members of the group. Any server in a group can share one or more public IPs with any other server in the group. With the exception of the first server in a shared IP group, servers must be launched into shared IP groups. A server may be a member of only one shared IP group.

shared storage Block storage that is simultaneously accessible by multiple clients, for example, NFS.

Sheepdog Distributed block storage system for QEMU, supported by OpenStack.

Simple Cloud Identity Management (SCIM) Specification for managing identity in the cloud, currently unsupported by OpenStack.

Simple Protocol for Independent Computing Environments (SPICE) SPICE provides remote desktop access to guest virtual machines. It is an alternative to VNC. SPICE is supported by OpenStack.

Single-root I/O Virtualization (SR-IOV) A specification that, when implemented by a physical PCIe device, enables it to appear as multiple separate PCIe devices. This enables multiple virtualized guests to share direct access to the physical device, offering improved performance over an equivalent virtual device. Currently supported in OpenStack Havana and later releases.

SmokeStack Runs automated tests against the core OpenStack API; written in Rails.

snapshot A point-in-time copy of an OpenStack storage volume or image. Use storage volume snapshots to back up volumes. Use image snapshots to back up data, or as "gold" images for additional servers.

soft reboot A controlled reboot where a VM instance is properly restarted through operating system commands.

Software Development Lifecycle Automation service (solum) OpenStack project that aims to make cloud services easier to consume and integrate with application development process by automating the source-to-image process, and simplifying app-centric deployment.

Software-defined networking (SDN) Provides an approach for network administrators to manage computer network services through abstraction of lower-level functionality.

SolidFire Volume Driver The Block Storage driver for the SolidFire iSCSI storage appliance.

solum Code name for the *Software Development Lifecycle Automation service*.

spread-first scheduler The Compute VM scheduling algorithm that attempts to start a new VM on the host with the least amount of load.

SQLAlchemy An open source SQL toolkit for Python, used in OpenStack.

SQLite A lightweight SQL database, used as the default persistent storage method in many OpenStack services.

stack A set of OpenStack resources created and managed by the Orchestration service according to a given template (either an AWS CloudFormation template or a Heat Orchestration Template (HOT)).

StackTach Community project that captures Compute AMQP communications; useful for debugging.

static IP address Alternative term for a fixed IP address.

StaticWeb WSGI middleware component of Object Storage that serves container data as a static web page.

storage back end The method that a service uses for persistent storage, such as iSCSI, NFS, or local disk.

storage manager A XenAPI component that provides a pluggable interface to support a wide variety of persistent storage back ends.

storage manager back end A persistent storage method supported by XenAPI, such as iSCSI or NFS.

storage node An Object Storage node that provides container services, account services, and object services; controls the account databases, container databases, and object storage.

storage services Collective name for the Object Storage object services, container services, and account services.

strategy Specifies the authentication source used by Image service or Identity. In the Database service, it refers to the extensions implemented for a data store.

subdomain A domain within a parent domain. Subdomains cannot be registered. Subdomains enable you to delegate domains. Subdomains can themselves have subdomains, so third-level, fourth-level, fifth-level, and deeper levels of nesting are possible.

subnet Logical subdivision of an IP network.

SUSE Linux Enterprise Server (SLES) A Linux distribution that is compatible with OpenStack.

suspend The VM instance is paused and its state is saved to disk of the host.

swap Disk-based virtual memory used by operating systems to provide more memory than is actually available on the system.

swauth An authentication and authorization service for Object Storage, implemented through WSGI middleware; uses Object Storage itself as the persistent backing store.

swift Codename for OpenStack *Object Storage service*.

swift All in One (SAIO) Creates a full Object Storage development environment within a single VM.

swift middleware Collective term for Object Storage components that provide additional functionality.

swift proxy server Acts as the gatekeeper to Object Storage and is responsible for authenticating the user.

swift storage node A node that runs Object Storage account, container, and object services.

sync point Point in time since the last container and accounts database sync among nodes within Object Storage.

sysadmin One of the default roles in the Compute RBAC system. Enables a user to add other users to a project, interact with VM images that are associated with the project, and start and stop VM instances.

system usage A Compute component that, along with the notification system, collects meters and usage information. This information can be used for billing.

T

tacker Code name for the *NFV Orchestration service*

Telemetry service (telemetry) The OpenStack project which collects measurements of the utilization of the physical and virtual resources comprising deployed clouds, persists this data for subsequent retrieval and analysis, and triggers actions when defined criteria are met.

TempAuth An authentication facility within Object Storage that enables Object Storage itself to perform authentication and authorization. Frequently used in testing and development.

Tempest Automated software test suite designed to run against the trunk of the OpenStack core project.

TempURL An Object Storage middleware component that enables creation of URLs for temporary object access.

tenant A group of users; used to isolate access to Compute resources. An alternative term for a project.

Tenant API An API that is accessible to projects.

tenant endpoint An Identity service API endpoint that is associated with one or more projects.

tenant ID An alternative term for *project ID*.

token An alpha-numeric string of text used to access OpenStack APIs and resources.

token services An Identity service component that manages and validates tokens after a user or project has been authenticated.

tombstone Used to mark Object Storage objects that have been deleted; ensures that the object is not updated on another node after it has been deleted.

topic publisher A process that is created when a RPC call is executed; used to push the message to the topic exchange.

Torpedo Community project used to run automated tests against the OpenStack API.

transaction ID Unique ID assigned to each Object Storage request; used for debugging and tracing.

transient Alternative term for non-durable.

transient exchange Alternative term for a non-durable exchange.

transient message A message that is stored in memory and is lost after the server is restarted.

transient queue Alternative term for a non-durable queue.

TripleO OpenStack-on-OpenStack program. The code name for the OpenStack Deployment program.

trove Codename for OpenStack *Database service*.

trusted platform module (TPM) Specialized microprocessor for incorporating cryptographic keys into devices for authenticating and securing a hardware platform.

U

Ubuntu A Debian-based Linux distribution.

unscoped token Alternative term for an Identity service default token.

updater Collective term for a group of Object Storage components that processes queued and failed updates for containers and objects.

user In OpenStack Identity, entities represent individual API consumers and are owned by a specific domain. In OpenStack Compute, a user can be associated with roles, projects, or both.

user data A blob of data that the user can specify when they launch an instance. The instance can access this data through the metadata service or config drive. Commonly used to pass a shell script that the instance runs on boot.

User Mode Linux (UML) An OpenStack-supported hypervisor.

V

VIF UUID Unique ID assigned to each Networking VIF.

Virtual Central Processing Unit (vCPU) Subdivides physical CPUs. Instances can then use those divisions.

Virtual Disk Image (VDI) One of the VM image disk formats supported by Image service.

Virtual Extensible LAN (VXLAN) A network virtualization technology that attempts to reduce the scalability problems associated with large cloud computing deployments. It uses a VLAN-like encapsulation technique to encapsulate Ethernet frames within UDP packets.

Virtual Hard Disk (VHD) One of the VM image disk formats supported by Image service.

virtual IP address (VIP) An Internet Protocol (IP) address configured on the load balancer for use by clients connecting to a service that is load balanced. Incoming connections are distributed to back-end nodes based on the configuration of the load balancer.

virtual machine (VM) An operating system instance that runs on top of a hypervisor. Multiple VMs can run at the same time on the same physical host.

virtual network An L2 network segment within Networking.

Virtual Network Computing (VNC) Open source GUI and CLI tools used for remote console access to VMs. Supported by Compute.

Virtual Network InterFace (VIF) An interface that is plugged into a port in a Networking network. Typically a virtual network interface belonging to a VM.

virtual networking A generic term for virtualization of network functions such as switching, routing, load balancing, and security using a combination of VMs and overlays on physical network infrastructure.

virtual port Attachment point where a virtual interface connects to a virtual network.

virtual private network (VPN) Provided by Compute in the form of cloudpipes, specialized instances that are used to create VPNs on a per-project basis.

virtual server Alternative term for a VM or guest.

virtual switch (vSwitch) Software that runs on a host or node and provides the features and functions of a hardware-based network switch.

virtual VLAN Alternative term for a virtual network.

VirtualBox An OpenStack-supported hypervisor.

Vitrage Code name for the *Root Cause Analysis service*.

VLAN manager A Compute component that provides dnsmasq and radvd and sets up forwarding to and from cloudpipe instances.

VLAN network The Network Controller provides virtual networks to enable compute servers to interact with each other and with the public network. All machines must have a public and private network interface. A VLAN network is a private network interface, which is controlled by the `vlan_interface` option with VLAN managers.

VM disk (VMDK) One of the VM image disk formats supported by Image service.

VM image Alternative term for an image.

VM Remote Control (VMRC) Method to access VM instance consoles using a web browser. Supported by Compute.

VMware API Supports interaction with VMware products in Compute.

VMware NSX Neutron plug-in Provides support for VMware NSX in Neutron.

VNC proxy A Compute component that provides users access to the consoles of their VM instances through VNC or VMRC.

volume Disk-based data storage generally represented as an iSCSI target with a file system that supports extended attributes; can be persistent or ephemeral.

Volume API Alternative name for the Block Storage API.

volume controller A Block Storage component that oversees and coordinates storage volume actions.

volume driver Alternative term for a volume plug-in.

volume ID Unique ID applied to each storage volume under the Block Storage control.

volume manager A Block Storage component that creates, attaches, and detaches persistent storage volumes.

volume node A Block Storage node that runs the cinder-volume daemon.

volume plug-in Provides support for new and specialized types of back-end storage for the Block Storage volume manager.

volume worker A cinder component that interacts with back-end storage to manage the creation and deletion of volumes and the creation of compute volumes, provided by the cinder-volume daemon.

vSphere An OpenStack-supported hypervisor.

W

Watcher Code name for the *Infrastructure Optimization service*.

weight Used by Object Storage devices to determine which storage devices are suitable for the job. Devices are weighted by size.

weighted cost The sum of each cost used when deciding where to start a new VM instance in Compute.

weighting A Compute process that determines the suitability of the VM instances for a job for a particular host. For example, not enough RAM on the host, too many CPUs on the host, and so on.

worker A daemon that listens to a queue and carries out tasks in response to messages. For example, the cinder-volume worker manages volume creation and deletion on storage arrays.

Workflow service (mistral) The OpenStack service that provides a simple YAML-based language to write workflows (tasks and transition rules) and a service that allows to upload them, modify, run them at scale and in a highly available manner, manage and monitor workflow execution state and state of individual tasks.

X

X.509 X.509 is the most widely used standard for defining digital certificates. It is a data structure that contains the subject (entity) identifiable information such as its name along with its public key. The certificate can contain a few other attributes as well depending upon the version. The most recent and standard version of X.509 is v3.

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.

Xen API The Xen administrative API, which is supported by Compute.

Xen Cloud Platform (XCP) An OpenStack-supported hypervisor.

Xen Storage Manager Volume Driver A Block Storage volume plug-in that enables communication with the Xen Storage Manager API.

XenServer An OpenStack-supported hypervisor.

XFS High-performance 64-bit file system created by Silicon Graphics. Excels in parallel I/O operations and data consistency.

Z

zaqar Codename for the *Message service*.

ZeroMQ Message queue software supported by OpenStack. An alternative to RabbitMQ. Also spelled 0MQ.

Zuul Tool used in OpenStack development to ensure correctly ordered testing of changes in parallel.

INDEX