Red Hat OpenStack Platform 10

Architecture Guide

Introduction to the product, components, and architectural examples

Last Updated: 2019-08-19

Red Hat OpenStack Platform 10 Architecture Guide

Introduction to the product, components, and architectural examples

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Abstract

This guide introduces the OpenStack cloud components and provides design guidelines and architecture examples to help you design your own OpenStack cloud.

Table of Contents

PREFACE

Red Hat OpenStack Platform provides the foundation to build a private or public Infrastructure-as-a-Service (IaaS) cloud on top of Red Hat Enterprise Linux. It offers a highly scalable, fault-tolerant platform for the development of cloud-enabled workloads.

Red Hat OpenStack Platform is packaged so that available physical hardware can be turned into a private, public, or hybrid cloud platform that includes:

- Fully distributed object storage
- Persistent block-level storage
- Virtual machine provisioning engine and image storage
- Authentication and authorization mechanisms
- Integrated networking
- Web browser-based interface accessible to users and administrators

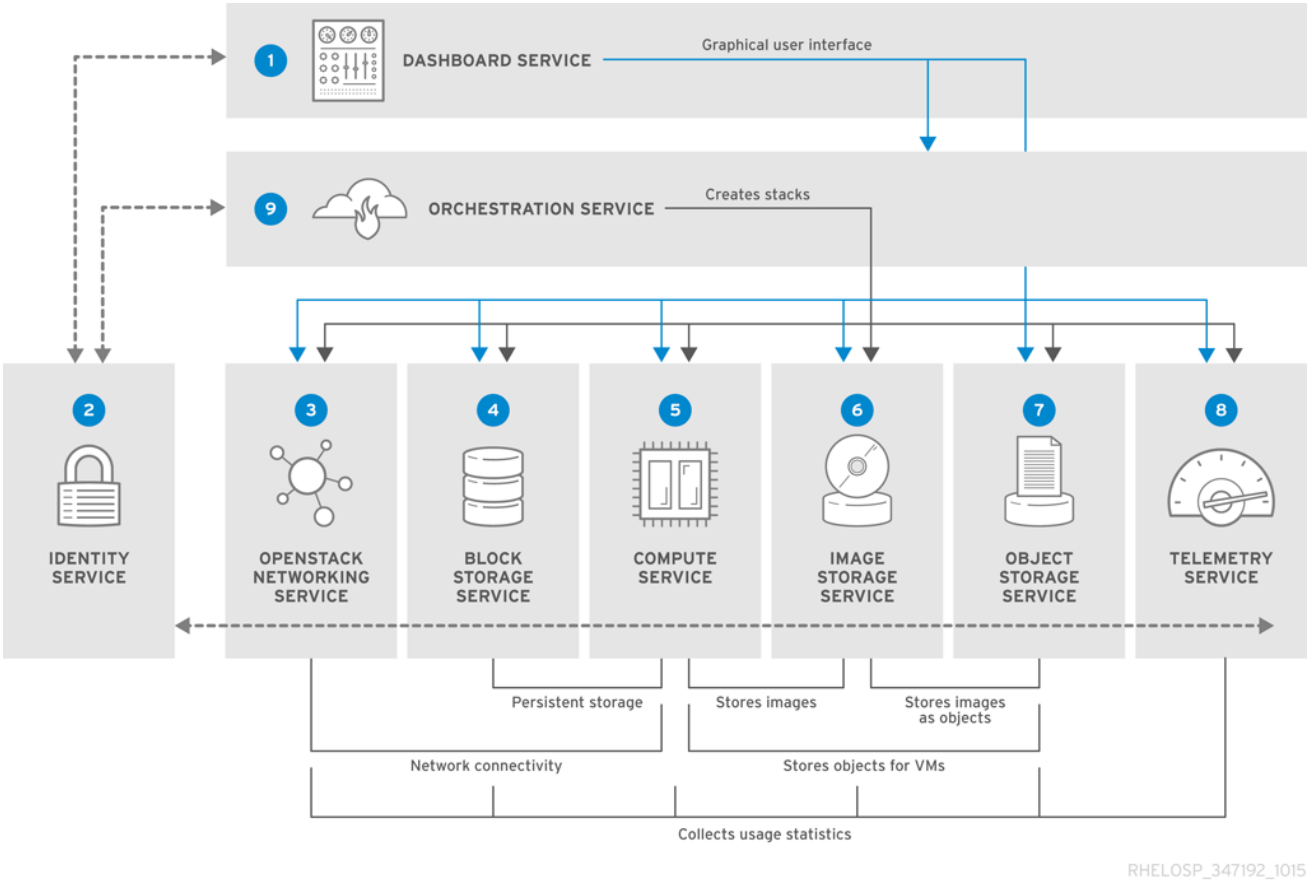
NOTE

- For reference information about the components mentioned in this guide, see Chapter 5, *Deployment Information*.
- For the complete Red Hat OpenStack Platform documentation suite, see Red Hat OpenStack Platform Documentation Suite.

CHAPTER 1. COMPONENTS

The Red Hat OpenStack Platform IaaS cloud is implemented as a collection of interacting services that control compute, storage, and networking resources. The cloud can be managed with a web-based dashboard or command-line clients, which allow administrators to control, provision, and automate OpenStack resources. OpenStack also has an extensive API, which is also available to all cloud users.

The following diagram provides a high-level overview of the OpenStack core services and their relationship with each other.



The following table describes each component shown in the diagram and provides links for the component documentation section.

Table 1.1. Core services

	Service	Code	Description	Location
1	Dashboard	horizon	Web browser-based dashboard that you use to manage OpenStack services.	Section 1.5.1, "OpenStack Dashboard (horizon)"
2	Identity	keystone	Centralized service for authentication and authorization of OpenStack services and for managing users, projects, and roles.	Section 1.4.1, "OpenStack Identity (keystone)"

	Service	Code	Description	Location
3	OpenStack Networking	neutron	Provides connectivity between the interfaces of OpenStack services.	Section 1.1.1, "OpenStack Networking (neutron)"
4	Block Storage	cinder	Manages persistent block storage volumes for virtual machines.	Section 1.2.1, "OpenStack Block Storage (cinder)"
5	Compute	nova	Manages and provisions virtual machines running on hypervisor nodes.	Section 1.3.1, "OpenStack Compute (nova)"
6	Image	glance	Registry service that you use to store resources such as virtual machine images and volume snapshots.	Section 1.3.3, "OpenStack Image (glance)"
7	Object Storage	swift	Allows users to store and retrieve files and arbitrary data.	Section 1.2.2, "OpenStack Object Storage (swift)"
8	Telemetry	ceilometer	Provides measurements of cloud resources.	Section 1.5.2, "OpenStack Telemetry (ceilometer)"
9	Orchestration	heat	Template-based orchestration engine that supports automatic creation of resource stacks.	Section 1.3.4, "OpenStack Orchestration (heat)"

Each OpenStack service contains a functional group of Linux services and other components. For example, the **glance-api** and **glance-registry** Linux services, together with a MariaDB database, implement the Image service. For information about third-party components included in OpenStack services, see Section 1.6.1, "Third-party Components".

Additional services are:

- Section 1.3.2, "OpenStack Bare Metal Provisioning (ironic)" - Enables users to provision physical machines (bare metal) with a variety of hardware vendors.
- Section 1.3.5, "OpenStack Data Processing (sahara)" - Enables users to provision and manage Hadoop clusters on OpenStack.

1.1. NETWORKING

1.1.1. OpenStack Networking (neutron)

OpenStack Networking handles creation and management of a virtual networking infrastructure in the OpenStack cloud. Infrastructure elements include networks, subnets, and routers. You can also deploy advanced services such as firewalls or virtual private networks (VPN).

OpenStack Networking provides cloud administrators with flexibility to decide which individual services to run on which physical systems. All service daemons can be run on a single physical host for evaluation purposes. Alternatively, each service can have a unique physical host or replicated across multiple hosts to provide redundancy.

Because OpenStack Networking is software-defined, it can react in real-time to changing network needs, such as creation and assignment of new IP addresses.

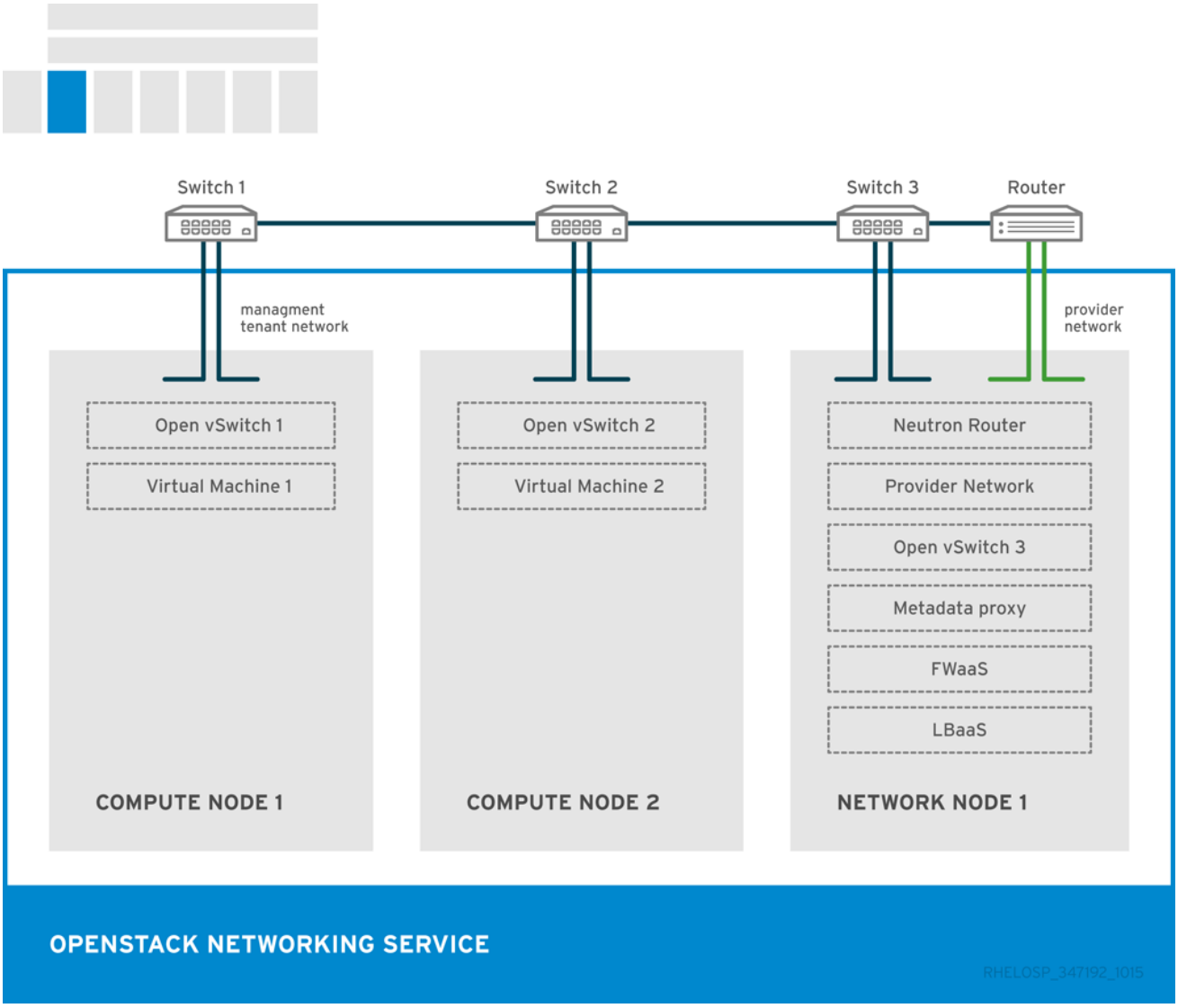
OpenStack Networking advantages include:

- Users can create networks, control traffic, and connect servers and devices to one or more networks.
- Flexible networking models can adapt to the network volume and tenancy.
- IP addresses can be dedicated or floating, where floating IPs can be used for dynamic traffic rerouting.
- If using VLAN networking, you can use a maximum of 4094 VLANs (4094 networks), where 4094 = 2^12 (minus 2 unusable) network addresses, which is imposed by the 12-bit header limitation.
- If using VXLAN tunnel-based networks, the VNI (Virtual Network Identifier) can use a 24-bit header, which will essentially allow around 16 million unique addresses/networks.

Table 1.2. OpenStack Networking components

Component	Description
Network agent	Service that runs on each OpenStack node to perform local networking configuration for the node virtual machines and for networking services such as Open vSwitch.
neutron-dhcp-agent	Agent that provides DHCP services to tenant networks.
neutron-ml2	Plug-in that manages network drivers and provides routing and switching services for networking services such as Open vSwitch or Ryu networks.
neutron-server	Python daemon that manages user requests and exposes the Networking API. The default server configuration uses a plug-in with a specific set of networking mechanisms to implement the Networking API. Certain plug-ins, such as the **openvswitch** and **linuxbridge** plug-ins, use native Linux networking mechanisms, while other plug-ins interface with external devices or SDN controllers.
neutron	Command-line client to access the API.

The placement of OpenStack Networking services and agents depends on the network requirements. The following diagram shows an example of a common deployment model without a controller. This model utilizes a dedicated OpenStack Networking node and tenant networks.



The example shows the following Networking service configuration:

- Two Compute nodes run the Open vSwitch (ovs-agent), and one OpenStack Networking node performs the following network functions:
  - L3 routing
  - DHCP
  - NAT including services such as FWaaS and LBaaS
- The compute nodes have two physical network cards each. One card handles tenant traffic, and the other card manages connectivity.
- The OpenStack Networking node has a third network card dedicated to provider traffic.

1.2. STORAGE

Section 1.2.1, “OpenStack Block Storage (cinder)”

Section 1.2.2, “OpenStack Object Storage (swift)”

1.2.1. OpenStack Block Storage (cinder)

OpenStack Block Storage provides persistent block storage management for virtual hard drives. Block Storage enables the user to create and delete block devices, and to manage attachment of block devices to servers.

The actual attachment and detachment of devices is handled through integration with the Compute service. You can use regions and zones to handle distributed block storage hosts.

You can use Block Storage in performance-sensitive scenarios, such as database storage or expandable file systems. You can also use it as a server with access to raw block-level storage. Additionally, you can take volume snapshots to restore data or to create new block storage volumes. Snapshots are dependent on driver support.

OpenStack Block Storage advantages include:

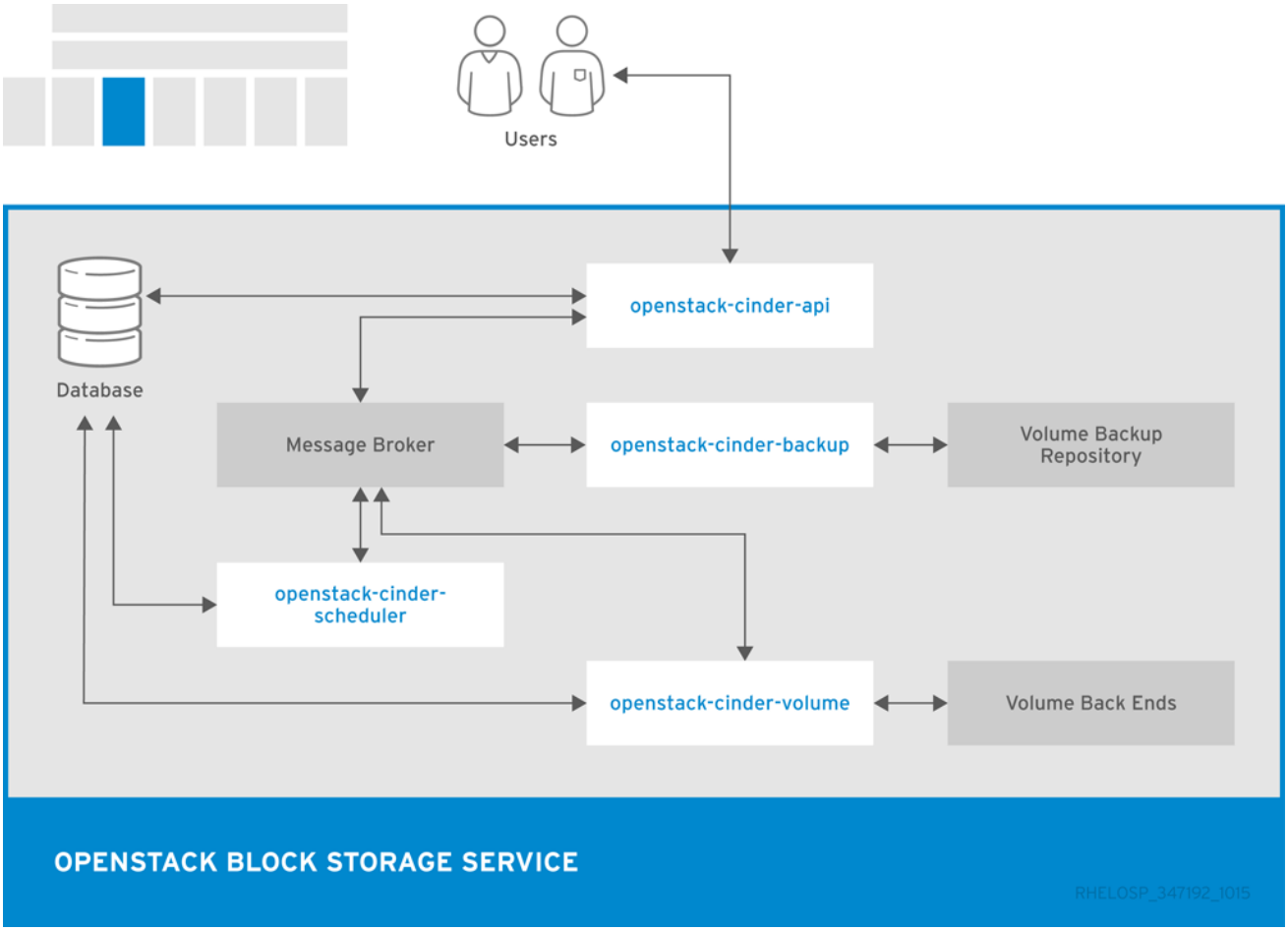
- Creating, listing and deleting volumes and snapshots.
- Attaching and detaching volumes to running virtual machines.

Although the main Block Storage services, such as volume, scheduler, API, can be co-located in a production environment, it is more common to deploy multiple instances of the volume service along one or more instances of the API and scheduler services to manage them.

Table 1.3. Block Storage components

Component	Description
openstack-cinder-api	Responds to requests and places them in the message queue. When a request is received, the API service verifies that identity requirements are met and translates the request into a message that includes the required block storage action. The message is then sent to the message broker for processing by the other Block Storage services.
openstack-cinder-backup	Backs up a Block Storage volume to an external storage repository. By default, OpenStack uses the Object Storage service to store the backup. You can also use Ceph or NFS back ends as storage repositories for backups.
openstack-cinder-scheduler	Assigns tasks to the queue and determines the provisioning volume server. The scheduler service reads requests from the message queue and determines on which block storage host to perform the requested action. The scheduler then communicates with the openstack-cinder-volume service on the selected host to process the request.
openstack-cinder-volume	Designates storage for virtual machines. The volume service manages the interaction with the block-storage devices. When requests arrive from the scheduler, the volume service can create, modify, or remove volumes. The volume service includes several drivers to interact with the block-storage devices, such as NFS, Red Hat Storage, or Dell EqualLogic.
cinder	Command-line client to access the Block Storage API.

The following diagram shows the relationship between the Block Storage API, the scheduler, the volume services, and other OpenStack components.



1.2.2. OpenStack Object Storage (swift)

Object Storage provides an HTTP-accessible storage system for large amounts of data, including static entities such as videos, images, email messages, files, or VM images. Objects are stored as binaries on the underlying file system along with metadata stored in the extended attributes of each file.

The Object Storage distributed architecture supports horizontal scaling as well as failover redundancy with software-based data replication. Because the service supports asynchronous and eventual consistency replication, you can use it in a multiple data-center deployment.

OpenStack Object Storage advantages include:

- Storage replicas maintain the state of objects in case of outage. A minimum of three replicas is recommended.
- Storage zones host replicas. Zones ensure that each replica of a given object can be stored separately. A zone might represent an individual disk drive, an array, a server, a server rack, or even an entire data center.
- Storage regions can group zones by location. Regions can include servers or server farms that are usually located in the same geographical area. Regions have a separate API endpoint for each Object Storage service installation, which allows for a discrete separation of services.

Object Storage uses ring **.gz** files, which serve as database and configuration files. These files contain details of all the storage devices and mappings of stored entities to the physical location of each file. Therefore, you can use ring files to determine the location of specific data. Each object, account, and

container server has a unique ring file.

The Object Storage service relies on other OpenStack services and components to perform actions. For example, the Identity Service (keystone), the rsync daemon, and a load balancer are all required.

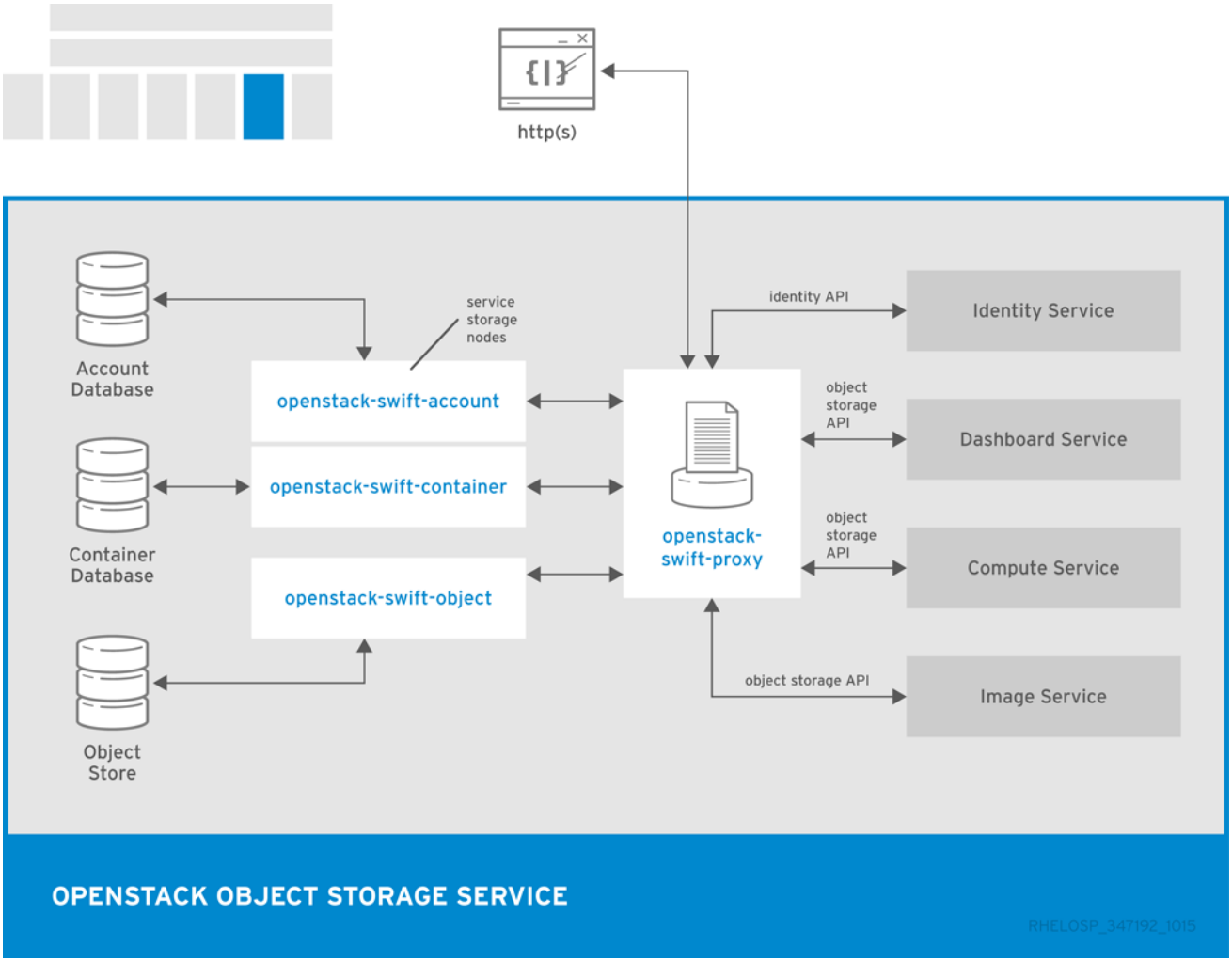
Table 1.4. Object Storage components

Component	Description
openstack-swift-account	Handles listings of containers with the account database.
openstack-swift-container	Handles listings of objects that are included in a specific container with the container database.
openstack-swift-object	Stores, retrieves, and deletes objects.
openstack-swift-proxy	Exposes the public API, provides authentication, and routes requests. Objects are streamed through the proxy server to the user without spooling.
swift	Command-line client to access the Object Storage API.

Table 1.5. Object Storage housekeeping components

Housekeeping	Components	Description
Auditing	• openstack-swift-account-auditor • openstack-swift-container-auditor • openstack-swift-object-auditor	Verifies the integrity of Object Storage accounts, containers, and objects, and helps to protect against data corruption.
Replication	• openstack-swift-account-replicator • openstack-swift-container-replicator • openstack-swift-object-replicator	Ensures consistent and available replication throughout the Object Storage cluster, including garbage collection.
Updating	• openstack-swift-account-updater • openstack-swift-container-updater • openstack-swift-object-updater	Identifies and retries failed updates.

The following diagram shows the main interfaces that the Object Storage uses to interact with other OpenStack services, databases, and brokers.



1.3. VIRTUAL MACHINES, IMAGES, AND TEMPLATES

Section 1.3.1, “OpenStack Compute (nova)”

Section 1.3.2, “OpenStack Bare Metal Provisioning (ironic)”

Section 1.3.3, “OpenStack Image (glance)”

Section 1.3.4, “OpenStack Orchestration (heat)”

Section 1.3.5, “OpenStack Data Processing (sahara)”

1.3.1. OpenStack Compute (nova)

OpenStack Compute serves as the core of the OpenStack cloud by providing virtual machines on demand. Compute schedules virtual machines to run on a set of nodes by defining drivers that interact with underlying virtualization mechanisms, and by exposing the functionality to the other OpenStack components.

Compute supports the libvirt driver **libvirtd** that uses KVM as the hypervisor. The hypervisor creates virtual machines and enables live migration from node to node. To provision bare metal machines, you can also use Section 1.3.2, “OpenStack Bare Metal Provisioning (ironic)” .

Compute interacts with the Identity service to authenticate instance and database access, with the Image service to access images and launch instances, and with the dashboard service to provide user and administrative interface.

You can restrict access to images by project and by user, and specify project and user quota, such as the number of instances that can be created by a single user.

When you deploy a Red Hat OpenStack Platform cloud, you can break down the cloud according to different categories:

Regions

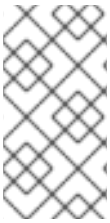
Each service cataloged in the Identity service is identified by the service region, which typically represents a geographical location, and the service endpoint. In a cloud with multiple compute nodes, regions enable discrete separation of services.
You can also use regions to share infrastructure between Compute installations while maintaining a high degree of failure tolerance.

Cells (Technology Preview)

The Compute hosts can be partitioned into groups called cells to handle large deployments or geographically separate installations. Cells are configured in a tree, where the top-level cell, called the API cell, runs the nova-api service but no nova-compute services.
Each child cell in the tree runs all other typical nova-* services but not the nova-api service. Each cell has a separate message queue and database service, and also runs the nova-cells service that manages the communication between the API cell and the child cells.

The benefits of cells include:

- You can use a single API server to control access to multiple Compute installations.
- An additional level of scheduling at the cell level is available that, unlike host scheduling, provides greater flexibility and control when you run virtual machines.



NOTE

This feature is available in this release as a **Technology Preview**, and therefore is not fully supported by Red Hat. It should only be used for testing, and should not be deployed in a production environment. For more information about Technology Preview features, see Scope of Coverage Details.

Host Aggregates and Availability Zones

A single Compute deployment can be partitioned into logical groups. You can create multiple groups of hosts that share common resources such as storage and network, or groups that share a special property such as trusted computing hardware.
To administrators, the group is presented as a Host Aggregate with assigned compute nodes and associated metadata. The Host Aggregate metadata is commonly used to provide information for openstack-nova-scheduler actions, such as limiting specific flavors or images to a subset of hosts.

To users, the group is presented as an Availability Zone. The user cannot view the group metadata or see the list of hosts in the zone.

The benefits of aggregates, or zones, include:

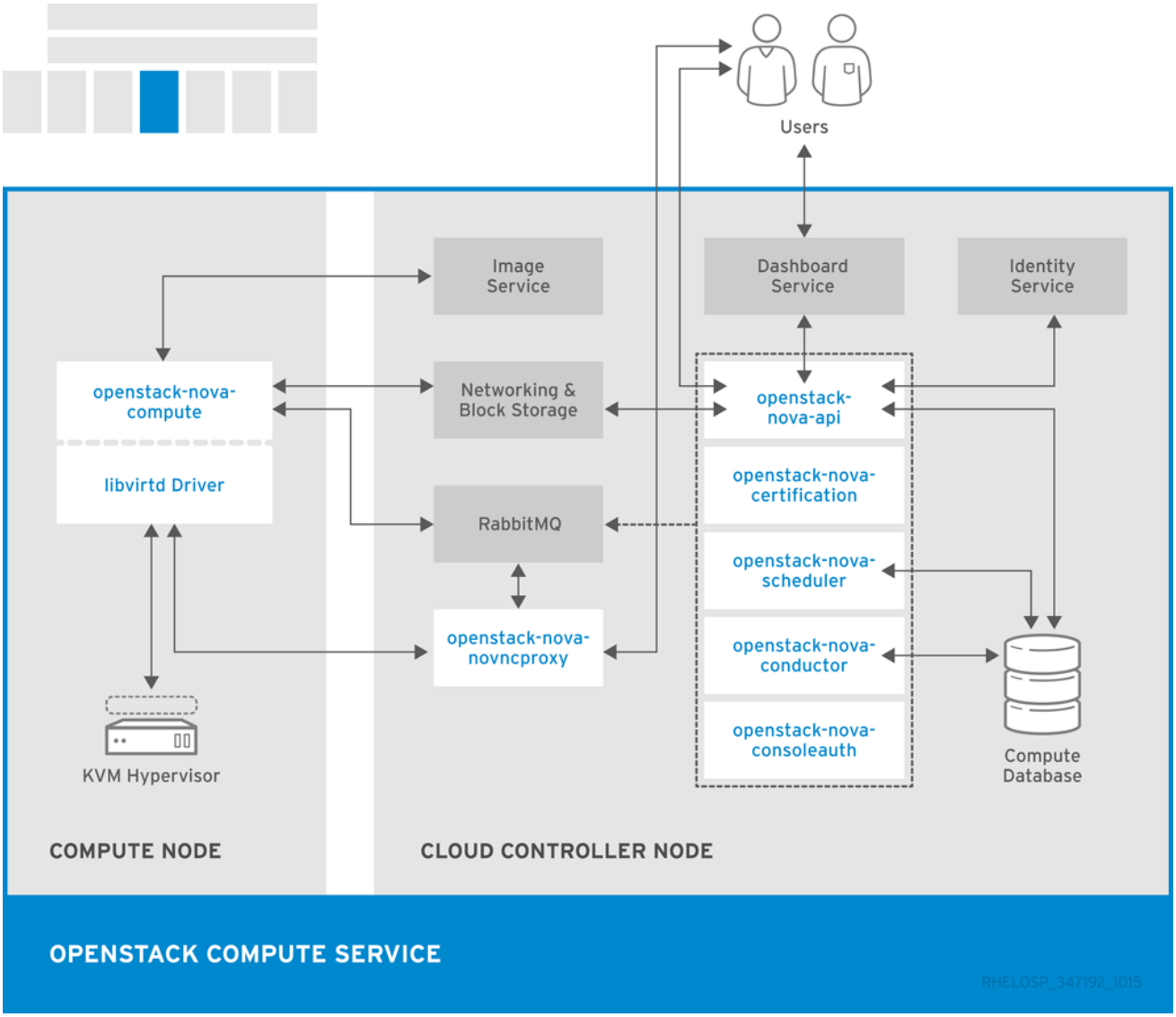
- Load balancing and instance distribution.
- Physical isolation and redundancy between zones, implemented with a separate power supply or network equipment.
- Labeling for groups of servers that have common attributes.

- Separation of different classes of hardware.

Table 1.6. Compute components

Component	Description
openstack-nova-api	Handles requests and provides access to the Compute services, such as booting an instance.
openstack-nova-cert	Provides the certificate manager.
openstack-nova-compute	Runs on each node to create and terminate virtual instances. The compute service interacts with the hypervisor to launch new instances, and ensures that the instance state is maintained in the Compute database.
openstack-nova-conductor	Provides database-access support for compute nodes to reduce security risks.
openstack-nova-consoleauth	Handles console authentication.
openstack-nova-novncproxy	Provides a VNC proxy for browsers to enable VNC consoles to access virtual machines.
openstack-nova-scheduler	Dispatches requests for new virtual machines to the correct node based on configured weights and filters.
nova	Command-line client to access the Compute API.

The following diagram shows the relationship between the Compute services and other OpenStack components.



1.3.2. OpenStack Bare Metal Provisioning (ironic)

OpenStack Bare Metal Provisioning enables the user to provision physical, or bare metal machines, for a variety of hardware vendors with hardware-specific drivers. Bare Metal Provisioning integrates with the Compute service to provision the bare metal machines in the same way that virtual machines are provisioned, and provides a solution for the *bare-metal-to-trusted-tenant* use case.

OpenStack Baremetal Provisioning advantages include:

- Hadoop clusters can be deployed on bare metal machines.
- Hyperscale and high-performance computing (HPC) clusters can be deployed.
- Database hosting for applications that are sensitive to virtual machines can be used.

Bare Metal Provisioning uses the Compute service for scheduling and quota management, and uses the Identity service for authentication. Instance images must be configured to support Bare Metal Provisioning instead of KVM.

The following diagram shows how Ironic and the other OpenStack services interact when a physical server is being provisioned:

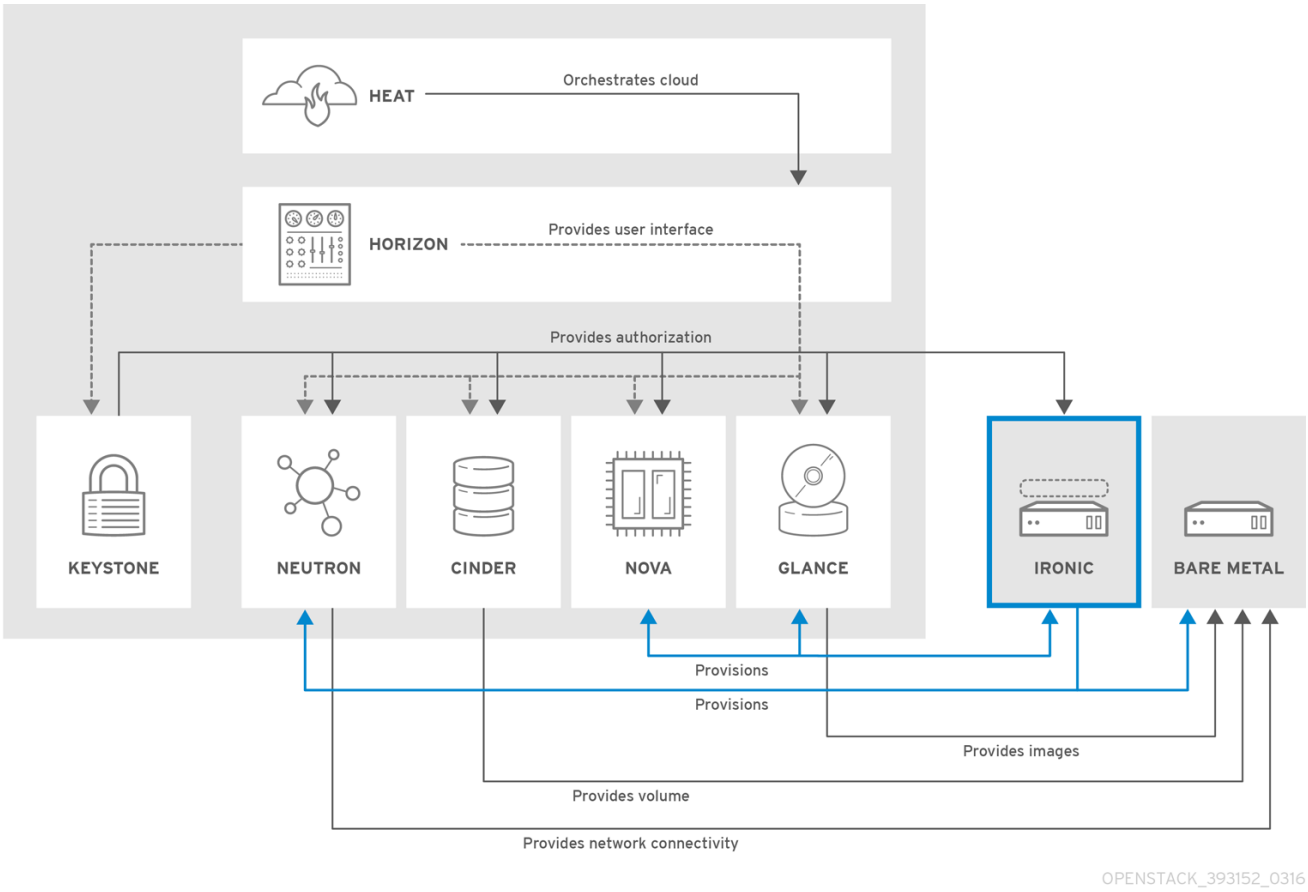
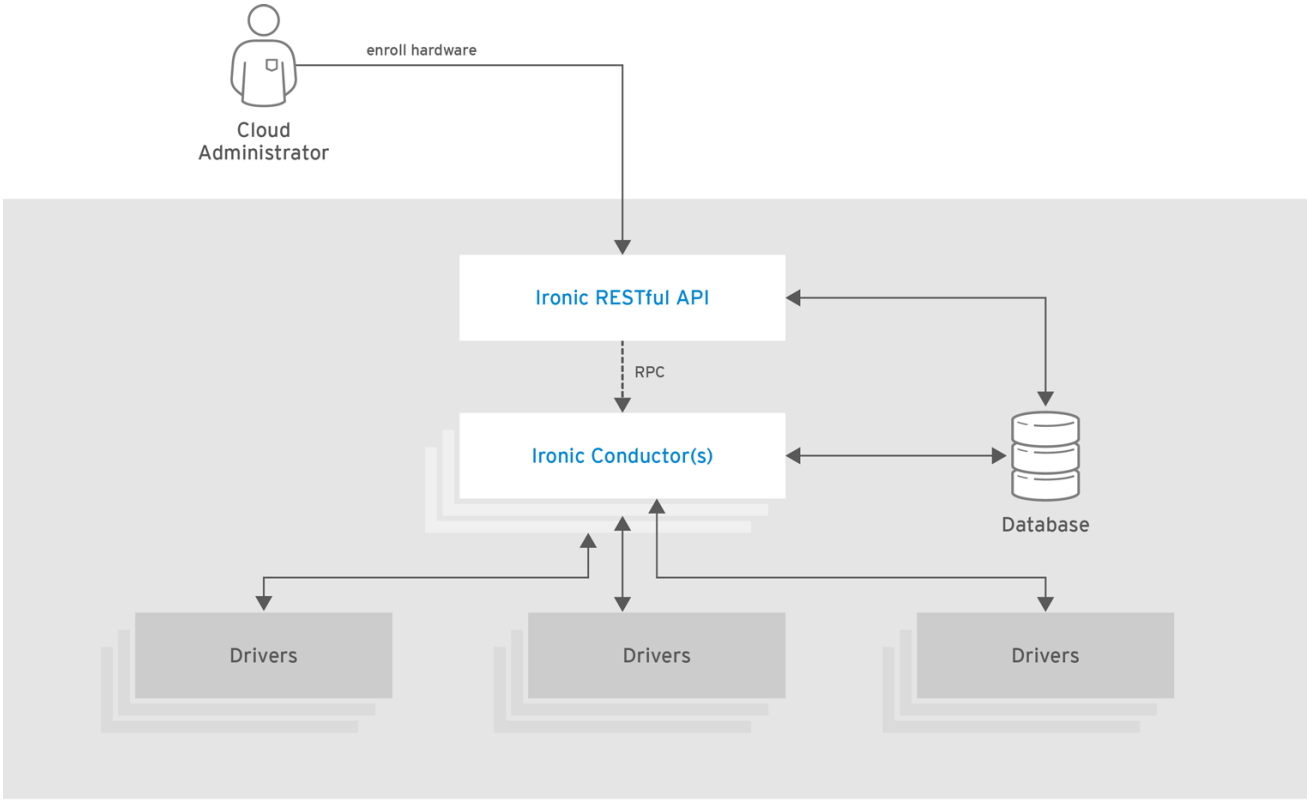


Table 1.7. Bare Metal Provisioning components

Component	Description
openstack-ironic-api	Handles requests and provides access to Compute resources on the bare metal node.
openstack-ironic-conductor	Interacts directly with hardware and ironic databases, and handles requested and periodic actions. You can create multiple conductors to interact with different hardware drivers.
ironic	Command-line client to access the Bare Metal Provisioning API.

The Ironic API is illustrated in following diagram:



1.3.3. OpenStack Image (glance)

OpenStack Image acts as a registry for virtual disk images. Users can add new images or take a snapshot of an existing server for immediate storage. You can use the snapshots for backup or as templates for new servers.

Registered images can be stored in the Object Storage service or in other locations, such as simple file systems or external Web servers.

The following image disk formats are supported:

- aki/ami/ari (Amazon kernel, ramdisk, or machine image)
- iso (archive format for optical discs, such as CDs)
- qcow2 (Qemu/KVM, supports Copy on Write)
- raw (unstructured format)
- vhd (Hyper-V, common for virtual machine monitors from vendors such as VMware, Xen, Microsoft, and VirtualBox)
- vdi (Qemu/VirtualBox)
- vmdk (VMware)

Container formats can also be registered by the Image service. The container format determines the type and detail level of the virtual machine metadata to store in the image.

The following container formats are supported:

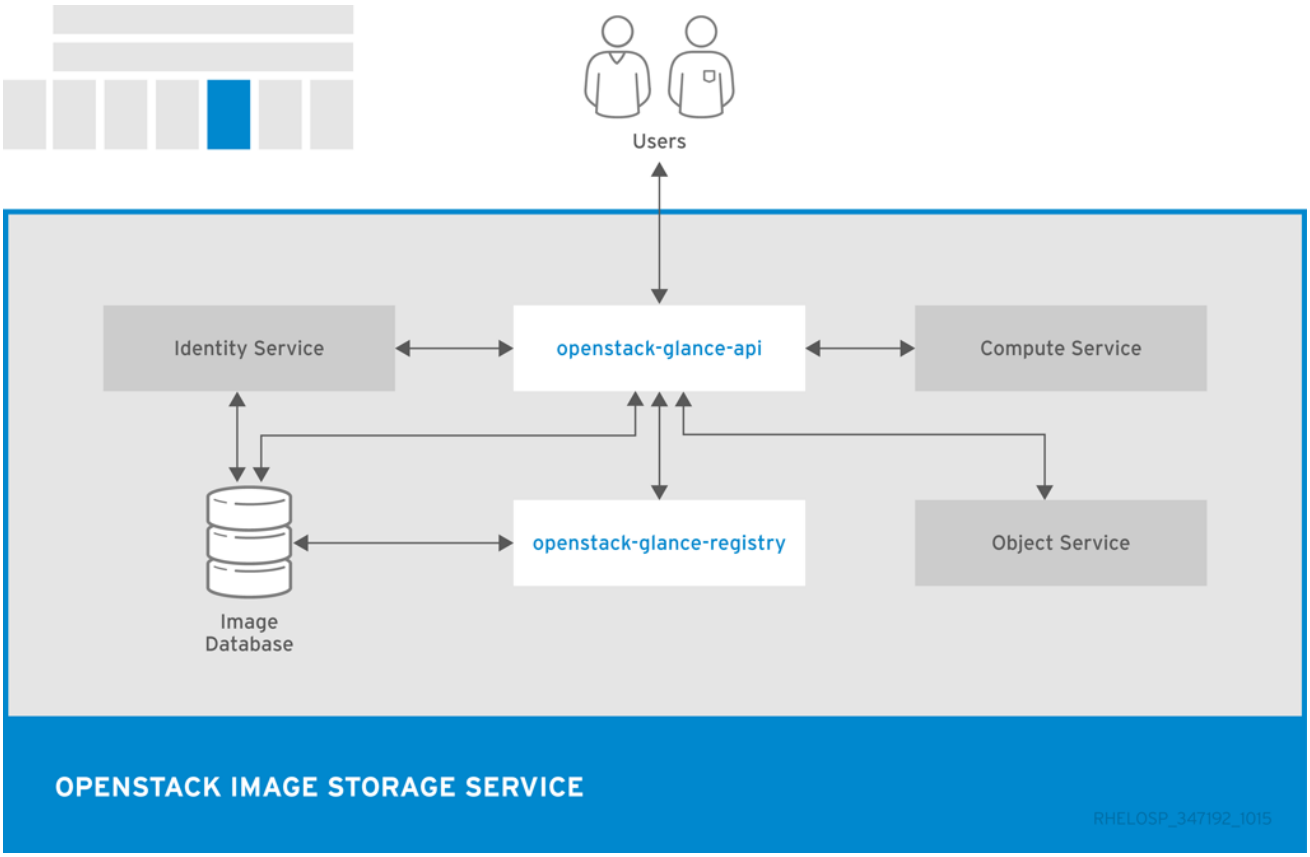
- bare (no metadata)

- ova (OVA tar archive)
- ovf (OVF format)
- aki/ami/ari (Amazon kernel, ramdisk, or machine image)

Table 1.8. Image components

Component	Description
openstack-glance-api	Interacts with storage back ends to handle requests for image retrieval and storage. The API uses openstack-glance-registry to retrieve image information. You must not access the registry service directly.
openstack-glance-registry	Manages all metadata for each image.
glance	Command-line client to access the Image API.

The following diagram shows the main interfaces that the Image service uses to register and retrieve images from the Image database.



1.3.4. OpenStack Orchestration (heat)

OpenStack Orchestration provides templates to create and manage cloud resources such as storage, networking, instances, or applications. Templates are used to create stacks, which are collections of resources.

For example, you can create templates for instances, floating IPs, volumes, security groups, or users. Orchestration offers access to all OpenStack core services with a single modular template, as well as capabilities such as auto-scaling and basic high availability.

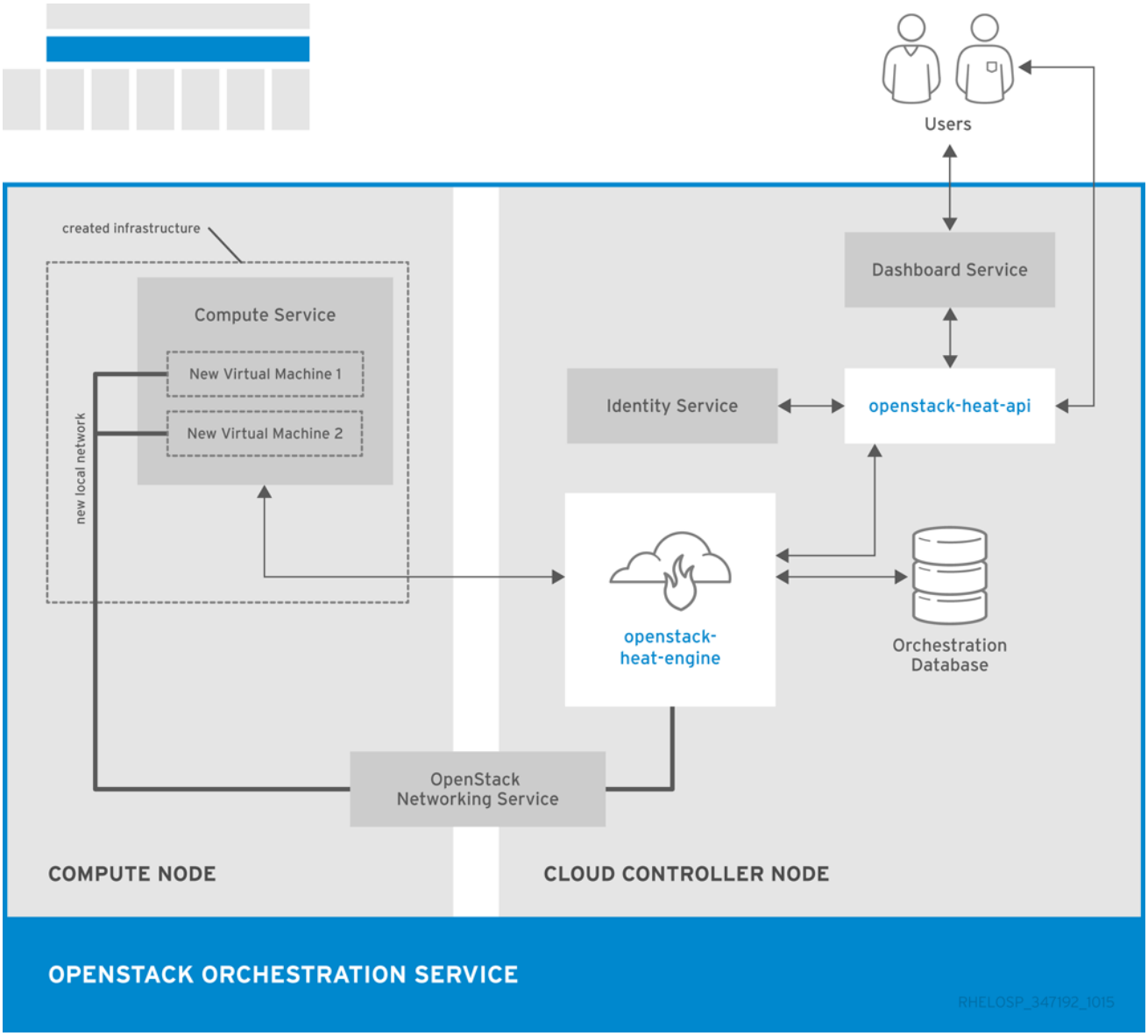
OpenStack Orchestration advantages include:

- A single template provides access to all underlying service APIs.
- Templates are modular and resource-oriented.
- Templates can be recursively defined and reusable, such as nested stacks. The cloud infrastructure can then be defined and reused in a modular way.
- Resource implementation is pluggable, which allows for custom resources.
- Resources can be auto-scaled, and therefore added or removed from the cluster based on usage.
- Basic high availability functionality is available.

Table 1.9. Orchestration components

Component	Description
openstack-heat-api	OpenStack-native REST API that processes API requests by sending the requests to the openstack-heat-engine service over RPC.
openstack-heat-api-cfn	Optional AWS-Query API compatible with AWS CloudFormation that processes API requests by sending the requests to the openstack-heat-engine service over RPC.
openstack-heat-engine	Orchestrates template launch and generates events for the API consumer.
openstack-heat-cfntools	Package of helper scripts such as cfn-hup, which handle updates to metadata and execute custom hooks.
heat	Command-line tool that communicates with the Orchestration API to execute AWS CloudFormation APIs.

The following diagram shows the main interfaces that the Orchestration service uses to create a new stack of two new instances and a local network.



1.3.5. OpenStack Data Processing (sahara)

OpenStack Data Processing enables the provisioning and management of Hadoop clusters on OpenStack. Hadoop stores and analyze large amounts of unstructured and structured data in clusters.

Hadoop clusters are groups of servers that can act as storage servers running the Hadoop Distributed File System (HDFS), compute servers running Hadoop's MapReduce (MR) framework, or both.

The servers in a Hadoop cluster need to reside in the same network, but they do not need to share memory or disks. Therefore, you can add or remove servers and clusters without affecting compatibility of the existing servers.

The Hadoop compute and storage servers are co-located, which enables high-speed analysis of stored data. All tasks are divided across the servers and utilizes the local server resources.

OpenStack Data Processing advantages include:

- Identity service can authenticate users and provide user security in the Hadoop cluster.
- Compute service can provision cluster instances.

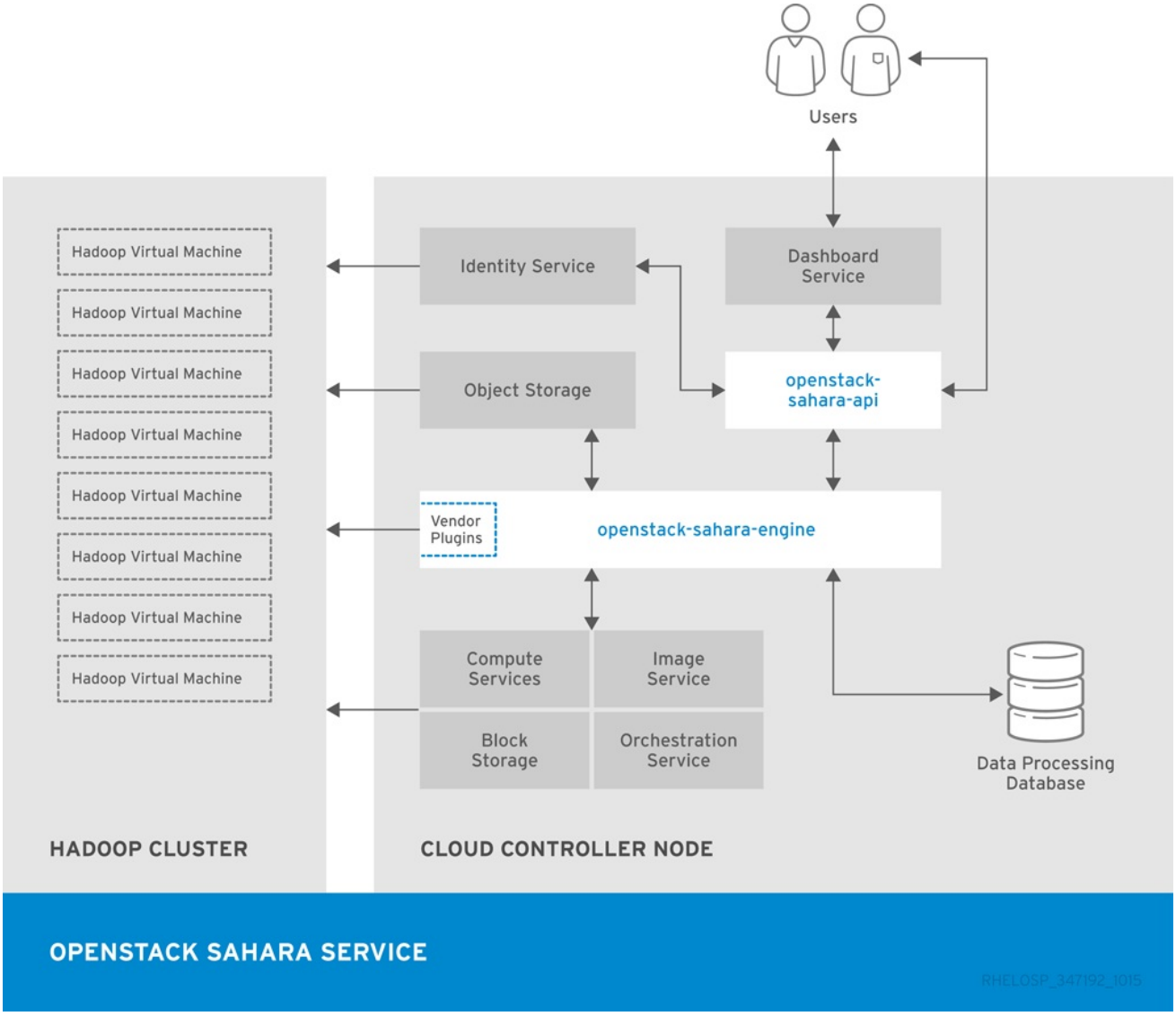
- Image service can store cluster instances, where each instance contains an operating system and HDFS.
- Object Storage service can be used to store data that Hadoop jobs process.
- Templates can be used to create and configure clusters. Users can change configuration parameters by creating custom templates or overriding parameters during cluster creation. Nodes are grouped together using a Node Group template, and cluster templates combine Node Groups.
- Jobs can be used to execute tasks on Hadoop clusters. Job binaries store executable code, and data sources store input or output locations and any necessary credentials.

Data Processing supports the Cloudera (CDH) plug-in as well as vendor-specific management tools, such as Apache Ambari. You can use the OpenStack dashboard or the command-line tool to provision and manage clusters.

Table 1.10. Sahara components

Component	Description
openstack-sahara-all	Legacy package that handles API and engine services.
openstack-sahara-api	Handles API requests and provides access to the Data Processing services.
openstack-sahara-engine	Provisioning engine that handles cluster requests and data delivery.
sahara	Command-line client to access the Data Processing API.

The following diagram shows the main interfaces that the Data Processing service uses to provision and manage a Hadoop cluster.



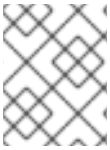
1.4. IDENTITY MANAGEMENT

1.4.1. OpenStack Identity (keystone)

OpenStack Identity provides user authentication and authorization to all OpenStack components. Identity supports multiple authentication mechanisms, including user name and password credentials, token-based systems, and AWS-style log-ins.

By default, the Identity service uses a MariaDB back end for token, catalog, policy, and identity information. This back end is recommended for development environments or to authenticate smaller user sets. You can also use multiple identity back ends concurrently, such as LDAP and SQL. You can also use memcache or Redis for token persistence.

Identity supports Federation with SAML. Federated Identity establishes trust between Identity Providers (IdP) and the services that Identity provides to the end user.



NOTE

Federated Identity and concurrent multiple back ends require Identity API v3 and Apache HTTPD deployment instead of Eventlet deployment.

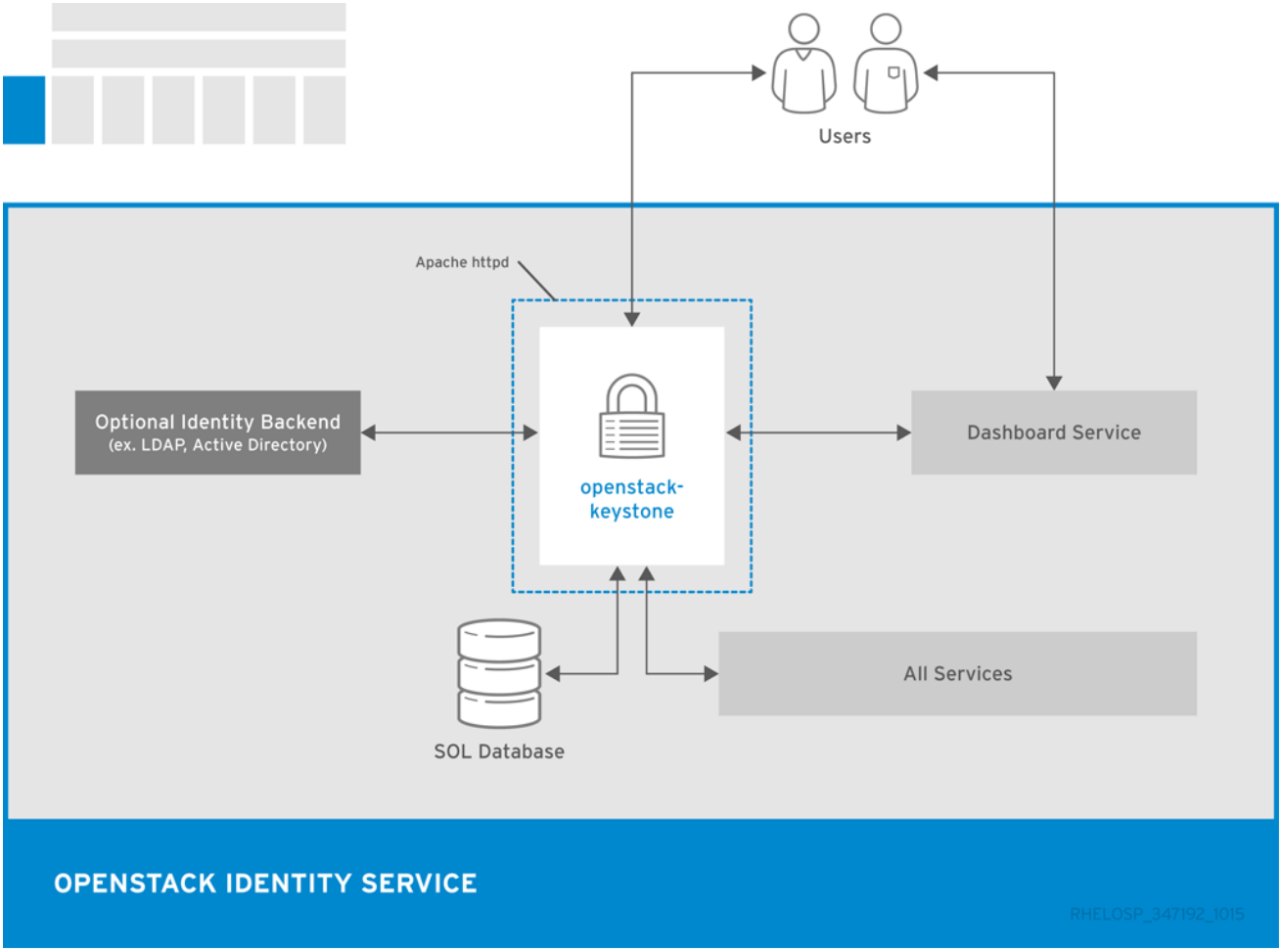
OpenStack Identity advantages include:

- User account management, including associated information such as a name and password. In addition to custom users, a user must be defined for each cataloged service. For example, the *glance* user must be defined for the Image service.
- Tenant, or project, management. Tenants can be the user group, project, or organization.
- Role management. Roles determine the user permissions. For example, a role might differentiate between permissions for a sales rep and permissions for a manager.
- Domain management. Domains determine the administrative boundaries of Identity service entities, and support multi-tenancy, where a domain represents a grouping of users, groups, and tenants. A domain can have more than one tenant, and if you use multiple concurrent Identity providers, each provider has one domain.

Table 1.11. Identity components

Component	Description
openstack-keystone	Provides Identity services, together with the administrative and public APIs. Both Identity API v2 and API v3 are supported.
keystone	Command-line client to access the Identity API.

The following diagram shows the basic authentication flow that Identity uses to authenticate users with other OpenStack components.



1.5. USER INTERFACES

Section 1.5.1, “OpenStack Dashboard (horizon)”

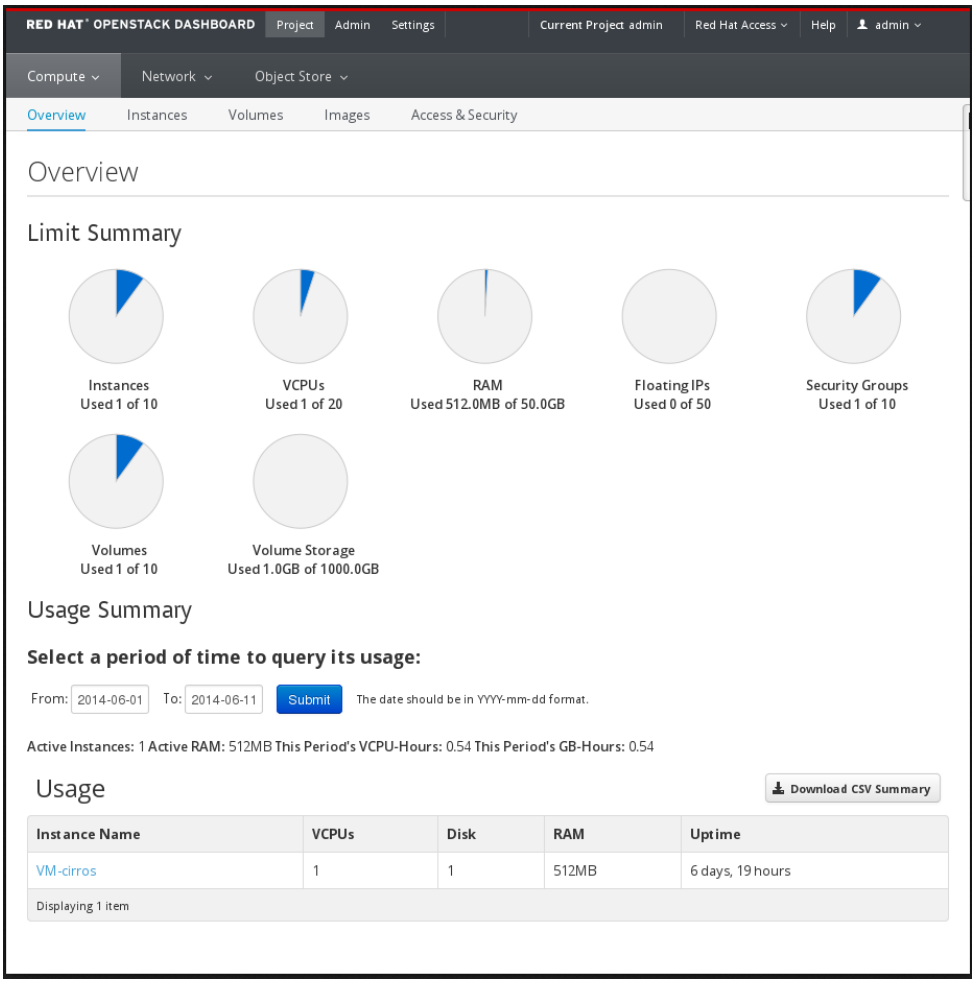
Section 1.5.2, “OpenStack Telemetry (ceilometer)”

1.5.1. OpenStack Dashboard (horizon)

OpenStack Dashboard provides a graphical user interface for users and administrators to perform operations such as creating and launching instances, managing networking, and setting access control.

The Dashboard service provides the Project, Admin, and Settings default dashboards. The modular design enables the dashboard to interface with other products such as billing, monitoring, and additional management tools.

The following image shows an example of the Compute panel in the Admin dashboard.



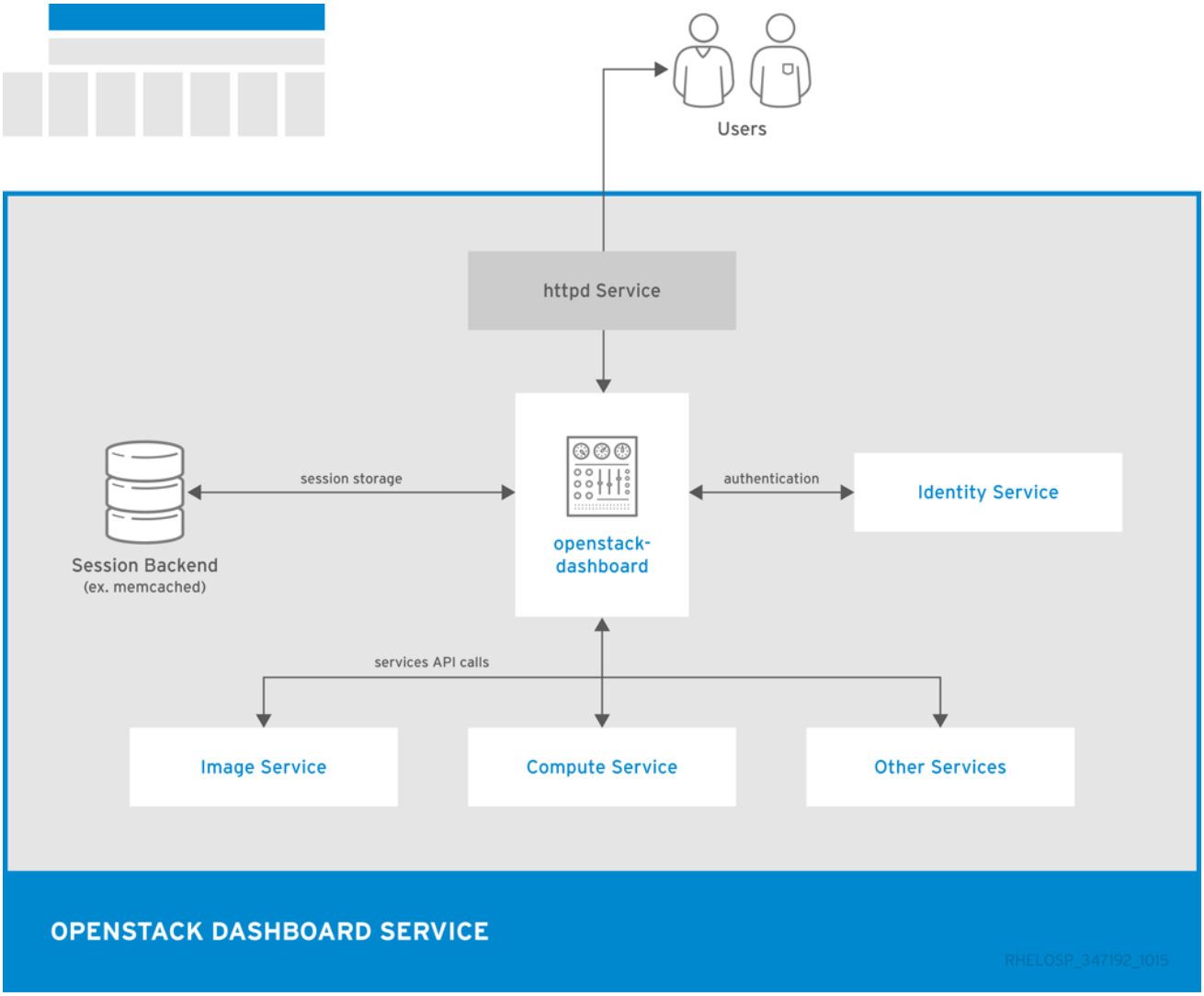
The role of the user that logs in to the dashboard determines which dashboards and panels are available.

Table 1.12. Dashboard components

Component	Description
openstack-dashboard	Django Web application that provides access to the dashboard from any Web browser.

Component	Description
Apache HTTP server (httpd service)	Hosts the application.

The following diagram shows an overview of the dashboard architecture.



The example shows the following interaction:

- The OpenStack Identity service authenticates and authorizes users
- The session back end provides database services
- The httpd service hosts the Web application and all other OpenStack services for API calls

1.5.2. OpenStack Telemetry (ceilometer)

OpenStack Telemetry provides user-level usage data for OpenStack-based clouds. The data can be used for customer billing, system monitoring, or alerts. Telemetry can collect data from notifications sent by existing OpenStack components such as Compute usage events, or by polling OpenStack infrastructure resources such as libvirt.

Telemetry includes a storage daemon that communicates with authenticated agents through a trusted messaging system to collect and aggregate data. Additionally, the service uses a plug-in system that

you can use to add new monitors. You can deploy the API Server, central agent, data store service, and collector agent on different hosts.

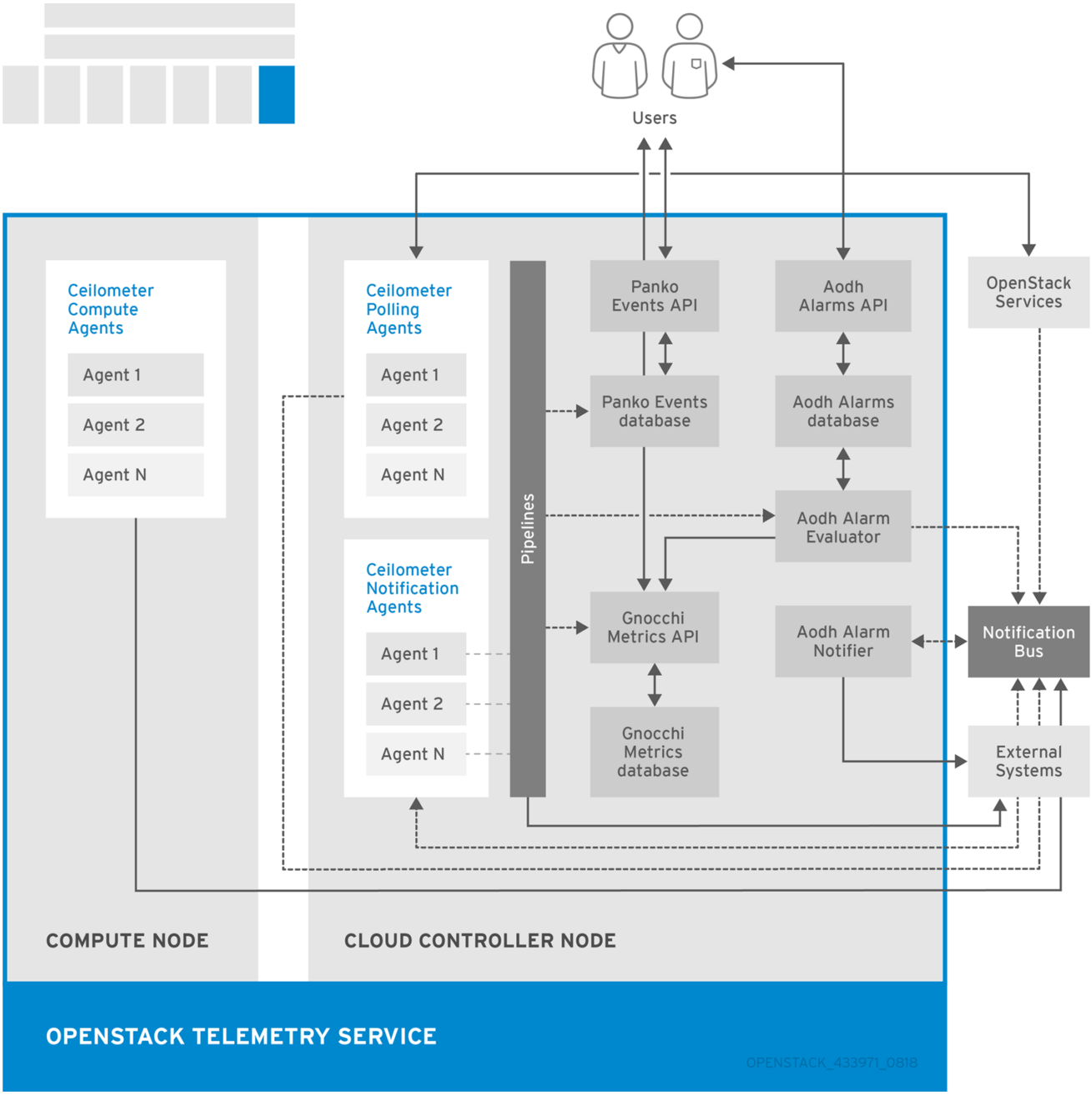
The service uses a MongoDB database to store collected data. Only the collector agents and the API server have access to the database.

The alarms and notifications are newly handled and controlled by the **aodh** service.

Table 1.13. Telemetry components

Component	Description
openstack-aodh-api	Provides access to the alarm information stored in the data store.
openstack-aodh-alarm-evaluator	Determines when alarms fire due to the associated statistic trend crossing a threshold over a sliding time window.
openstack-aodh-alarm-notifier	Executes actions when alarms are triggered.
openstack-aodh-alarm-listener	Emits an alarm when a pre-defined event pattern occurs.
openstack-ceilometer-api	Runs on one or more central management servers to provide access to data in the database.
openstack-ceilometer-central	Runs on a central management server to poll for utilization statistics about resources independent from instances or Compute nodes. The agent cannot be horizontally scaled, so you can run only a single instance of this service at a time.
openstack-ceilometer-collector	Runs on one or more central management servers to monitor the message queues. Each collector processes and translates notification messages to Telemetry messages, and sends the messages back to the message bus with the relevant topic. Telemetry messages are written to the data store without modification. You can choose where to run these agents, because all intra-agent communication is based on AMQP or REST calls to the ceilometer-api service, similar to the ceilometer-alarm-evaluator service.
openstack-ceilometer-compute	Runs on each Compute node to poll for resource utilization statistics. Each nova-compute node must have a ceilometer-compute agent deployed and running.
openstack-ceilometer-notification	Pushes metrics to the collector service from various OpenStack services.
ceilometer	Command-line client to access the Telemetry API.

The following diagram shows the interfaces used by the Telemetry service.



1.6. THIRD-PARTY COMPONENTS

1.6.1. Third-party Components

Some Red Hat OpenStack Platform components use third-party databases, services, and tools.

1.6.1.1. Databases

- MariaDB is the default database that is shipped with Red Hat Enterprise Linux. MariaDB enables Red Hat to fully support open source community-developed software. Each OpenStack component except Telemetry requires a running MariaDB service. Therefore, you need to deploy MariaDB before you deploy a full OpenStack cloud service or before you install any standalone OpenStack component.

- The Telemetry service uses a MongoDB database to store collected usage data from collector agents. Only the collector agents and the API server have access to the database.

1.6.1.2. Messaging

RabbitMQ is a robust open-source messaging system based on the AMQP standard. RabbitMQ is a high-performance message broker used in many enterprise systems with widespread commercial support. In Red Hat OpenStack Platform, RabbitMQ is the default and recommended message broker.

RabbitMQ manages OpenStack transactions including queuing, distribution, security, management, clustering, and federation. It also serves a key role in high availability and clustering scenarios.

1.6.1.3. External Caching

External applications for caching, such as memcached or Redis, offer persistence and shared storage and speed up dynamic web applications by reducing the database load. External caching is used by various OpenStack components, for example:

- The Object Storage service uses memcached to cache authenticated clients, instead of requiring each client to re-authorize each interaction.
- By default, the dashboard uses memcached for session storage.
- The Identity service uses Redis or memcached for token persistence.

CHAPTER 2. NETWORKING IN-DEPTH

2.1. HOW BASIC NETWORKING WORKS

Networking consists of moving information from one computer to another. At the most basic level, this is performed by running a cable between two machines, each with a network interface card (NIC) installed. If you ever studied the OSI networking model, this is layer 1.

When you want to involve more than two computers in the conversation, you need to scale out this configuration by adding a device called a switch. Switches are dedicated devices with multiple Ethernet ports to which you connect additional machines. This configuration is called a Local Area Network (LAN).

Switches move up the OSI model to layer 2 and apply more intelligence than the lower layer 1. Each NIC has a unique MAC address that is assigned to the hardware, and this number allows machines that are plugged in to the same switch find each other.

The switch maintains a list of which MAC addresses are plugged into which ports, so that when one computer attempts to send data to another computer, the switch knows where each NIC is located and adjusts the circuitry to direct the network traffic to the correct destination.

2.1.1. Connecting multiple LANs

If you use two LANs on two separate switches, you can connect them to share information with each other in the following ways:

Trunk cable

You can connect the two switches directly with a physical cable, called a trunk cable. In this configuration, you plug each end of the trunk cable into a port on each switch, and then define these ports as trunk ports. Now the two switches act as one big logical switch, and the connected computers can successfully find each other. This option is not very scalable, and overhead becomes an issue the more switches you link directly.

Router

You can use a device called a router to plug in cables from each switch. As a result, the router is aware of the networks that are configured on both switches. Each switch that you plug into the router becomes an interface and is assigned an IP address, known as the default gateway for that network. The "default" in default gateway means that this is the destination where traffic will be sent if it is clear that the destination computer is not on the same LAN as the source of the data transfer. After you set this default gateway on each of your computers, they do not need to be aware of all of the other computers on the other networks to send traffic to them. The traffic is just sent to the default gateway and the router handles it from there. Since the router is aware of which networks reside on which interface, it can send the packets on to their intended destinations. Routing works at layer 3 of the OSI model, and utilizes familiar concepts like IP addresses and subnets.

NOTE

This concept is how the Internet itself works. Many separate networks run by different organizations are all inter-connected using switches and routers. Keep following the correct default gateways and your traffic will eventually get to where it needs to go.

2.1.2. VLANs

Virtual Local Area Networks (VLANs) allow you to segment network traffic for computers that run on

the same switch. You can logically divide your switch by configuring the ports to be members of different networks. This configuration turns the ports into mini-LANs that allow you to separate traffic for security purposes.

For example, if your switch has 24 ports, you can define ports 1-6 to belong to VLAN200, and ports 7-18 belong to VLAN201. Computers that are plugged into VLAN200 are completely separate from computers on VLAN201, and they can no longer communicate directly. All traffic between the two VLANs must now pass through the router as if they were two separate physical switches. You can also enhance the security with firewalls to determine which VLANs can communicate with each other.

2.1.3. Firewalls

Firewalls operate at the same OSI layer as IP routing. They are often located in the same network segments as the routers, where they govern the traffic between all the networks. Firewalls use a predefined set of rules that prescribe which traffic can or cannot enter a network. These rules can be very granular. For example, you can define a rule where servers on VLAN 200 can only communicate with computers on VLAN201, and only on a Thursday afternoon, and only if the traffic is Web (HTTP) and moves in one direction.

To help enforce these rules, some firewalls also perform Stateful Packet Inspection (SPI), where they examine the contents of packets to ensure that they are what they claim to be. Hackers are known to exfiltrate data by sending traffic that masquerades as something else, and SPI is one method that can help mitigate that threat.

2.1.4. Bridges

Network bridges are switches that operate at the same level 2 of the OSI model, but their only function is to connect separate networks together, similar to routers.

2.2. NETWORKING IN OPENSTACK

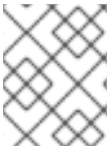
All of the basic networking concepts in an OpenStack cloud, except that they are defined by services and configuration. This is known as Software-Defined Networking (SDN). Virtual switches (Open vSwitch) and routers (l3-agent) allow your instances to communicate with each other, and allow them to communicate externally using the physical network. The Open vSwitch bridge allocates virtual ports to instances and spans across to the physical network to allow incoming and outgoing traffic.

2.3. ADVANCED OPENSTACK NETWORKING CONCEPTS

2.3.1. Layer 3 High Availability

OpenStack Networking hosts virtual routers on a centralized Network node, which is a physical server that is dedicated to the function of hosting the virtual networking components. These virtual routers direct traffic to and from virtual machines, and are vital to the continued connectivity of your environment. Since physical servers might experience outages due to many reasons, your virtual machines might be vulnerable to outages when the Network node becomes unavailable.

OpenStack Networking uses Layer 3 High Availability to help mitigate this vulnerability, implementing the industry standard VRRP to protect virtual routers and floating IP addresses. With Layer 3 High Availability, the virtual routers of the tenant are randomly distributed across multiple physical Network nodes, with one router designated as the active router, and the other routers on standby, ready to take over if the Network node that hosts the active router experiences an outage.



NOTE

"Layer 3" refers to the section of the OSI model where this feature functions, and means that it can protect routing and IP addressing.

For more information, see the "Layer 3 High Availability" section in the *Networking Guide*.

2.3.2. Load Balancing-as-a-Service (LBaaS)

Load Balancing-as-a-Service (LBaaS) enables OpenStack Networking to distribute incoming network requests equally between designated instances. This distribution ensures the workload is shared among instances and helps to use system resources more effectively. Incoming requests are distributed using one of the following load balancing methods:

Round robin

Rotates requests evenly between multiple instances.

Source IP

Requests from a unique source IP address are always directed to the same instance.

Least connections

Allocates requests to the instance with the lowest number of active connections.

For more information, see the "Configuring Load Balancing-as-a-Service" section in the *Networking Guide*.

2.3.3. IPv6

OpenStack Networking supports IPv6 addresses in tenant networks, so you can dynamically assign IPv6 addresses to virtual machines. OpenStack Networking can also integrate with SLAAC on your physical routers, so that virtual machines can receive IPv6 addresses from your existing DHCP infrastructure.

For more information, see the "IPv6" section in the *Networking Guide*.

CHAPTER 3. DESIGN

This section describes technical and operational considerations to take when you design your Red Hat OpenStack Platform deployment.

NOTE

All architecture examples in this guide assume that you deploy OpenStack Platform on Red Hat Enterprise Linux 7.3 with the KVM hypervisor.

3.1. PLANNING MODELS

When you design a Red Hat OpenStack Platform deployment, the duration of the project can affect the configuration and resource allocation of the deployment. Each planning model might aim to achieve different goals, and therefore requires different considerations.

3.1.1. Short-term model (3 months)

To perform short-term capacity planning and forecasting, consider capturing a record of the following metrics:

- Total vCPU number
- Total vRAM allocation
- I/O mean latency
- Network traffic
- Compute load
- Storage allocation

The vCPU, vRAM, and latency metrics are the most accessible for capacity planning. With these details, you can apply a standard second-order regression and receive a usable capacity estimate covering the following three months. Use this estimate to determine whether you need to deploy additional hardware.

3.1.2. Middle-term model (6 months)

This model requires review iterations and you must estimate deviations from the forecasting trends and the actual usage. You can analyze this information with standard statistical tools or with specialized analysis models such as Nash-Sutcliffe. Trends can also be calculated using second-order regression.

NOTE

In deployments with multiple instance flavors, you can correlate vRAM and vCPU usage more easily if you treat the vCPU and vRAM metrics as a single metric.

3.1.3. Long-term model (1 year)

Total resource usage can vary during one year, and deviations normally occur from the original long-term capacity estimate. Therefore, second-order regression might be an insufficient measure for capacity forecasting, especially in cases where usage is cyclical.

When planning for a long-term deployment, a capacity-planning model based on data that spans over a year must fit at least the first derivative. Depending on the usage pattern, frequency analysis might also be required.

3.2. COMPUTE RESOURCES

Compute resources are the core of the OpenStack cloud. Therefore, it is recommended to consider physical and virtual resource allocation, distribution, failover, and additional devices, when you design your Red Hat OpenStack Platform deployment.

3.2.1. General considerations

Number of processors, memory, and storage in each hypervisor

The number of processor cores and threads directly affects the number of worker threads that can run on a Compute node. Therefore, you must determine the design based on the service and based on a balanced infrastructure for all services.
Depending on the workload profile, additional Compute resource pools can be added to the cloud later. In some cases, the demand on certain instance flavors might not justify individual hardware design, with preference instead given to commoditized systems.

In either case, initiate the design by allocating hardware resources that can service common instances requests. If you want to add additional hardware designs to the overall architecture, this can be done at a later time.

Processor type

Processor selection is an extremely important consideration in hardware design, especially when comparing the features and performance characteristics of different processors.
Processors can include features specifically for virtualized compute hosts, such as hardware-assisted virtualization and memory paging, or EPT shadowing technology. These features can have a significant impact on the performance of your cloud VMs.

Resource nodes

You must take into account Compute requirements of non-hypervisor resource nodes in the cloud. Resource nodes include the controller node and nodes that run Object Storage, Block Storage, and Networking services.

Resource pools

Use a Compute design that allocates multiple pools of resources to be provided on-demand. This design maximizes application resource usage in the cloud. Each resource pool should service specific flavors of instances or groups of flavors.
Designing multiple resource pools helps to ensure that whenever instances are scheduled to Compute hypervisors, each set of node resources is allocated to maximize usage of available hardware. This is commonly referred to as bin packing.

Using a consistent hardware design across nodes in a resource pool also helps to support bin packing. Hardware nodes selected for a Compute resource pool should share a common processor, memory, and storage layout. Choosing a common hardware design helps easier deployment, support and node lifecycle maintenance.

Over-commit ratios

OpenStack enables users to over-commit CPU and RAM on Compute nodes, which helps to increase the number of instances that run in the cloud. However, over-committing can reduce the performance of the instances.
The over-commit ratio is the ratio of available virtual resources compared to the available physical resources.

- The default CPU allocation ratio of 16:1 means that the scheduler allocates up to 16 virtual cores for every physical core. For example, if a physical node has 12 cores, the scheduler can allocate up to 192 virtual cores. With typical flavor definitions of 4 virtual cores per instance, this ratio can provide 48 instances on the physical node.

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

Tuning the over-commit ratios for CPU and memory during the design phase is important because it has a direct impact on the hardware layout of your Compute nodes. When designing a hardware node as a Compute resource pool to service instances, consider the number of processor cores available on the node, as well as the required disk and memory to service instances running at full capacity.

For example, an **m1.small** instance uses 1 vCPU, 20 GB of ephemeral storage, and 2,048 MB of RAM. For a server with 2 CPUs of 10 cores each, with hyperthreading turned on:

- The default CPU overcommit ratio of 16:1 allows for 640 (2 × 10 × 2 × 16) total **m1.small** instances.

- The default memory over-commit ratio of 1.5:1 means that the server needs at least 853 GB (640 × 2,048 MB / 1.5) of RAM.

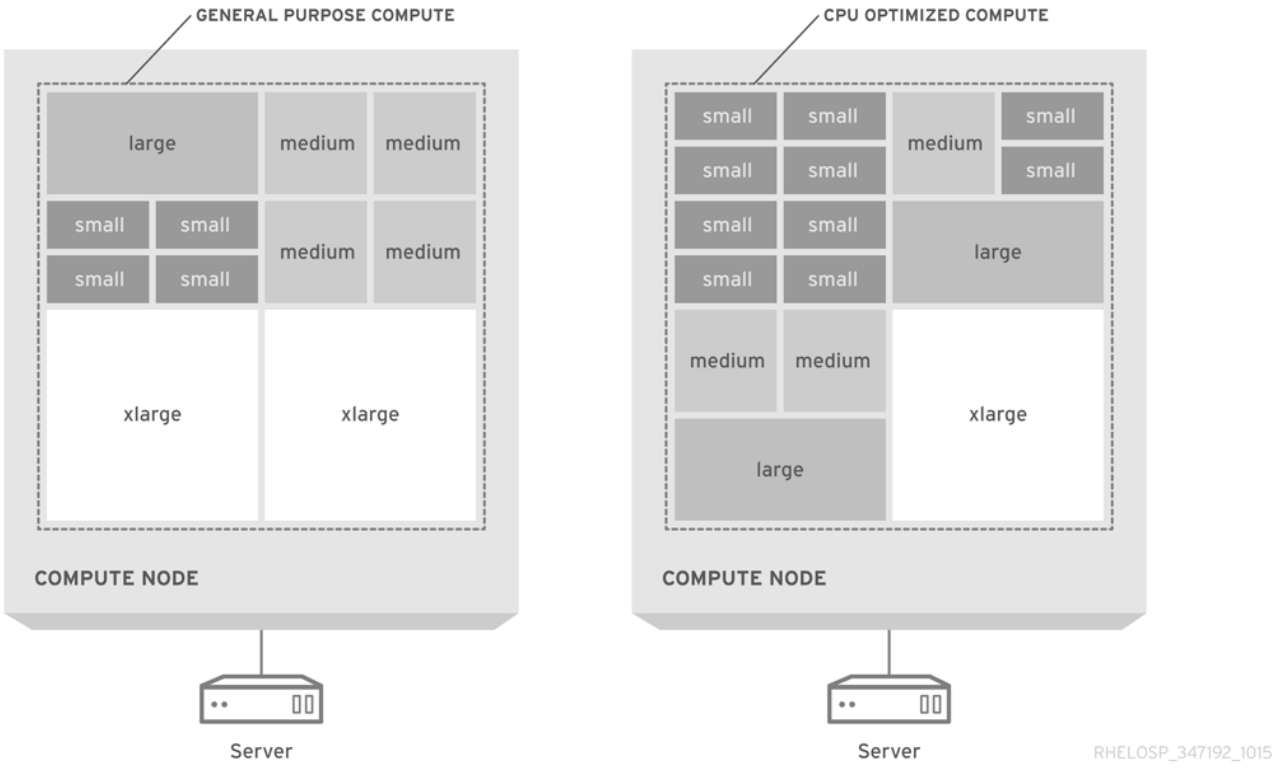
When sizing nodes for memory, it is also important to consider the additional memory required to service operating system and service needs.

3.2.2. Flavors

Each created instance is given a flavor, or resource template, which determines the instance size and capacity. Flavors can also specify secondary ephemeral storage, swap disk, metadata to restrict usage, or special project access. Default flavors do not have these additional attributes defined. Instance flavors allow to measure capacity forecasting, because common use cases are predictably sized and not sized ad-hoc.

To facilitate packing virtual machines to physical hosts, the default selection of flavors provides a second largest flavor half the size of the largest flavor in every dimension. The flavor has half the vCPUs, half the vRAM, and half the ephemeral disk space. Each subsequent largest flavor is half the size of the previous flavor.

The following diagram shows a visual representation of flavor assignments in a general-purpose computing design and for a CPU-optimized, packed server:



The default flavors are recommended for typical configurations of commodity server hardware. To maximize utilization, you might need to customize the flavors or to create new flavors to align instance sizes to available hardware.

If possible, limit your flavors to one vCPU for each flavor. It is important to note that Type 1 hypervisors can schedule CPU time more easily to VMs that are configured with one vCPU. For example, a hypervisor that schedules CPU time to a VM that is configured with 4 vCPUs must wait until four physical cores are available, even if the task to perform requires only one vCPU.

Workload characteristics can also influence hardware choices and flavor configuration, especially where the tasks present different ratios of CPU, RAM, or HDD requirements. For information about flavors, see Managing Flavors in the *Instances and Images Guide* .

3.2.3. vCPU-to-physical CPU core ratio

The default allocation ratio in Red Hat OpenStack Platform is 16 vCPUs per physical, or hyperthreaded, core.

The following table lists the maximum number of VMs that can be suitably run on a physical host based on the total available memory, including 4GB reserved for the system:

Total RAM	VMs	Total vCPU
64GB	14	56

Total RAM	VMs	Total vCPU
96GB	20	80
128GB	29	116

For example, planning an initial greenfields standup of 60 instances requires 3+1 compute nodes. Usually, memory bottlenecks are more common than CPU bottlenecks. However, the allocation ratio can be lowered to 8 vCPUs per physical core if needed.

3.2.4. Memory overhead

The KVM hypervisor requires a small amount of VM memory overhead, including non-shareable memory. Shareable memory for QEMU/KVM systems can be rounded to 200 MB for each hypervisor.

vRAM	Physical memory usage (average)
256	310
512	610
1024	1080
2048	2120
4096	4180

Typically, you can estimate hypervisor overhead of 100mb per VM.

3.2.5. Over-subscription

Memory is a limiting factor for hypervisor deployment. The number of VMs that you can run on each physical host is limited by the amount of memory that the host can access. For example, deploying a quad-core CPU with 256GB of RAM and more than 200 1GB instances leads to poor performance. Therefore, you must carefully determine the optimal ratio of CPU cores and memory to distribute across the instances.

3.2.6. Density

Instance density

In a compute-focused architecture, instance density is lower, which means that CPU and RAM over-subscription ratios are also lower. You might require more hosts to support the anticipated scale if instance density is lower, especially if the design uses dual-socket hardware designs.

Host density

You can address the higher host count of dual-socket designs by using a quad-socket platform. This platform decreases host density, which then increases the rack count. This configuration can affect the network requirements, the number of power connections, and might also impact the cooling requirements.

Power and cooling density

Reducing power and cooling density is an important consideration for data centers with older infrastructure. For example, the power and cooling density requirements for 2U, 3U, or even 4U server designs, might be lower than for blade, sled, or 1U server designs due to lower host density.

3.2.7. Compute hardware

Blade servers

Most blade servers can support dual-socket, multi-core CPUs. To avoid exceeding the CPU limit, select **full-width** or **full-height** blades. These blade types can also decrease 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. Half-height blades are twice as dense as full-height blades, which results in only eight servers per ten rack units.

1U servers

1U rack-mounted servers that occupy only a single rack unit might offer higher server density than a blade server solution. You can use 40 units of 1U servers in one rack to provide space for the top of rack (ToR) switches. In comparison, you can only use 32 full-width blade servers in one rack. However, 1U servers from major vendors have only dual-socket, multi-core CPU configurations. To support a higher CPU configuration in a 1U rack-mount form factor, purchase systems from original design manufacturers (ODMs) or second-tier manufacturers.

2U servers

2U rack-mounted servers provide quad-socket, multi-core CPU support, but with a corresponding decrease in server density. 2U rack-mounted servers offer half of the density that 1U rack-mounted servers offer.

Larger servers

Larger rack-mounted servers, such as 4U servers, often provide higher CPU capacity and typically support four or even eight CPU sockets. These servers have greater expandability, but have much lower server density and are often more expensive.

Sled servers

Sled servers are rack-mounted servers that support multiple independent servers in a single 2U or 3U enclosure. These servers deliver higher density than 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. However, the dual-socket limitation on individual nodes might not be sufficient to offset the additional cost and configuration complexity.

3.2.8. Additional devices

You might consider the following additional devices for Compute nodes:

- Graphics processing units (GPUs) for high-performance computing jobs.
- Hardware-based random number generators to avoid entropy starvation for cryptographic routines. A random number generator device can be added to an instance using with the instance image properties. **/dev/random** is the default entropy source.
- SSDs for ephemeral storage to maximize read/write time for database management systems.
- Host aggregates work by grouping together hosts that share similar characteristics, such as hardware similarities. The addition of specialized hardware to a cloud deployment might add to the cost of each node, so consider whether the additional customization is needed for all

Compute nodes, or just a subset.

3.3. STORAGE RESOURCES

When you design your cloud, the storage solution you choose impacts critical aspects of the deployment, such as performance, capacity, availability, and interoperability.

Consider the following factors when you choose your storage solution:

3.3.1. General Considerations

Applications

Applications should be aware of underlying storage sub-systems to use cloud storage solutions effectively. If natively-available replication is not available, operations personnel must be able to configure the application to provide replication service.
An application that can detect underlying storage systems can function in a wide variety of infrastructures and still have the same basic behavior regardless of the differences in the underlying infrastructure.

I/O

Benchmarks for input-output performance provide a baseline for expected performance levels. The benchmark results data can help to model behavior under different loads, and help you to design a suitable architecture.
Smaller, scripted benchmarks during the lifecycle of the architecture can help to record the system health at different times. The data from the scripted benchmarks can assist to scope and gain a deeper understanding of the organization needs.

Interoperability

Ensure that any hardware or storage management platform that you select is interoperable with OpenStack components, such as the KVM hypervisor, which affects whether you can use it for short-term instance storage.

Security

Data security design can focus on different aspects based on SLAs, legal requirements, industry regulations, and required certifications for systems or personnel. Consider compliance with HIPPA, ISO9000, or SOX based on the type of data. For certain organizations, access control levels should also be considered.

3.3.2. OpenStack Object Storage (swift)

Availability

Design your object storage resource pools to provide the level of availability that you need for your object data. Consider rack-level and zone-level designs to accommodate the number of necessary replicas. The defult number of replicas is three. Each replica of data should exist in a separate availability zone with independent power, cooling, and network resources that service the specific zone.
The OpenStack Object Storage service places a specific number of data replicas as objects on resource nodes. These replicas are distributed across the cluster based on a consistent hash ring, which exists on all nodes in the cluster. In addition, a pool of Object Storage proxy servers that provide access to data stored on the object nodes should service each availability zone.

Design the Object Storage system with a sufficient number of zones to provide the minimum required successful responses for the number of replicas. For example, if you configure three replicas

in the Swift cluster, the recommended number of zones to configure inside the Object Storage cluster is five.

Although you can deploy a solution with fewer zones, some data may not be available and API requests to some objects stored in the cluster might fail. Therefore, ensure you account for the number of zones in the Object Storage cluster.

Object proxies in each region should leverage local read and write affinity so that local storage resources facilitate access to objects wherever possible. You should deploy upstream load balancing to ensure that proxy services are distributed across multiple zones. In some cases, you might need third-party solutions to assist with the geographical distribution of services.

A zone within an Object Storage cluster is a logical division, and can be comprised of a single disk, a node, a collection of nodes, multiple racks, or multiple DCs. You must allow the Object Storage cluster to scale while providing an available and redundant storage system. You might need to configure storage policies with different requirements for replicas, retention, and other factors that could affect the design of storage in a specific zone.

Node storage

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

The consistency and partition tolerance characteristics of OpenStack Object Storage ensure that data stays current and survives hardware faults without requiring specialized data-replication devices.

Performance

Object storage nodes should be designed so that the number of requests does not hinder the performance of the cluster. The object storage service is a chatty protocol. Therefore, using multiple processors with higher core counts ensures that the IO requests do not inundate the server.

Weighting and cost

OpenStack Object Storage provides the ability to mix and match drives with weighting inside the swift ring. When designing your swift storage cluster, you can use most cost-effective storage solution.

Many server chassis can hold 60 or more drives in 4U of rack space. Therefore, you can maximize the amount of storage for each rack unit at the best cost per terabyte. However, it is not recommended to use RAID controllers in an object storage node.

Scaling

When you design your storage solution, you must determine the maximum partition power required by the Object Storage service, which then determines the maximum number of partitions that you can create. Object Storage distributes data across the entire storage cluster, but each partition cannot span more than one disk. Therefore, the maximum number of partitions cannot exceed the number of disks.

For example, a system with an initial single disk and a partition power of three can hold eight (2^3) partitions. Adding a second disk means that each disk can hold four partitions. The one-disk-per-partition limit means that this system cannot have more than eight disks and limits its scalability.

However, a system with an initial single disk and a partition power of 10 can have up to 1024 (2^{10}) partitions.

Whenever you increase the system back-end storage capacity, the partition maps redistribute data across the storage nodes. In some cases, this replication consists of extremely large data sets. In those cases, you should use back-end replication links that do not conflict with tenant access to data.

If more tenants begin to access data in the cluster and the data sets grow, you must add front-end bandwidth to service data access requests. Adding front-end bandwidth to an Object Storage cluster requires designing Object Storage proxies that tenants can use to gain access to the data, along with the high availability solutions that enable scaling of the proxy layer.

You should design a front-end load balancing layer that tenants and consumers use to gain access to data stored within the cluster. This load balancing layer can be distributed across zones, regions or even across geographic boundaries.

In some cases, you must add bandwidth and capacity to the network resources that service requests between proxy servers and storage nodes. Therefore, the network architecture that provides access to storage nodes and proxy servers should be scalable.

3.3.3. OpenStack Block Storage (cinder)

Availability and Redundancy

The input-output per second (IOPS) demand of your application determines whether you should use a RAID controller and which RAID level is required. For redundancy, you should use a redundant RAID configuration, such as RAID 5 or RAID 6. Some specialized features, such as automated replication of block storage volumes, might require third-party plug-ins or enterprise block storage solutions to handle the higher demand.
In environments with extreme demand on Block Storage, you should use multiple storage pools. Each device pool should have a similar hardware design and disk configuration across all hardware nodes in that pool. This design provides applications with access to a wide variety of Block Storage pools with various redundancy, availability, and performance characteristics.

The network architecture should also take into account the amount of East-West bandwidth required for instances to use available storage resources. The selected network devices should support jumbo frames to transfer large blocks of data. In some cases, you might need to create an additional dedicated back end storage network to provide connectivity between instances and Block Storage resources to reduce load on network resources.

When you deploy multiple storage pools, you must consider the impact on the Block Storage scheduler, which provisions storage across resource nodes. Ensure that applications can schedule volumes in multiple regions with specific network, power, and cooling infrastructure. This design allows tenants to build fault-tolerant applications distributed across multiple availability zones.

In addition to the Block Storage resource nodes, it is important to design for high availability and redundancy of APIs and related services that are responsible for provisioning and providing access to the storage nodes. You should design a layer of hardware or software load balancers to achieve high availability of the REST API services to provide uninterrupted service.

In some cases, you might need to deploy an additional load balancing layer to provide access to back-end database services that are responsible for servicing and storing the state of Block Storage volumes. You should design a highly-available database solution to store the Block Storage databases, such as MariaDB and Galera.

Attached storage

The Block Storage service can take advantage of enterprise storage solutions using a plug-in driver developed by the hardware vendor. A large number of enterprise plug-ins ship out-of-the-box with OpenStack Block Storage, and others are available through third-party channels.
General-purpose clouds typically use directly-attached storage in the majority of Block Storage nodes. Therefore, you might need to provide additional levels of service to tenants. These levels might only be provided by enterprise storage solutions.

Performance

If higher performance is needed, you can use high-performance RAID volumes. For extreme performance, you can use high-speed solid-state drive (SSD) disks.

Pools

Block Storage pools should allow tenants to choose appropriate storage solutions for their applications. By creating multiple storage pools of different types and configuring an advanced storage scheduler for the Block Storage service, you can provide tenants a large catalog of storage services with a variety of performance levels and redundancy options.

Scaling

You can upgrade Block Storage pools to add storage capacity without interruption to the overall Block Storage service. Add nodes to the pool by installing and configuring the appropriate hardware and software. You can then configure the new nodes to report to the proper storage pool with the message bus.
Because Block Storage nodes report the node availability to the scheduler service, when a new node is online and available, tenants can use the new storage resources immediately.

In some cases, the demand on Block Storage from instances might exhaust the available network bandwidth. Therefore, you should design the network infrastructure to service Block Storage resources to allow you to add capacity and bandwidth seamlessly.

This often involves dynamic routing protocols or advanced networking solutions to add capacity to downstream devices. The front-end and back-end storage network designs should include the ability to quickly and easily add capacity and bandwidth.

3.3.4. Storage Hardware

Capacity

Node hardware should support enough storage for the cloud services, and should ensure that capacity can be added after deployment. Hardware nodes should support a large number of inexpensive disks with no reliance on RAID controller cards.
Hardware nodes should also be capable of supporting high-speed storage solutions and RAID controller cards to provide hardware-based storage performance and redundancy. Selecting hardware RAID controllers that automatically repair damaged arrays assists with the replacement and repair of degraded or destroyed storage devices.

Connectivity

If you use non-Ethernet storage protocols in the storage solution, ensure that the hardware can handle these protocols. If you select a centralized storage array, ensure that the hypervisor can connect to that storage array for image storage.

Cost

Storage can be a significant portion of the overall system cost. If you need vendor support, a commercial storage solution is recommended but incurs a bigger expense. If you need to minimize initial financial investment, you can design a system based on commodity hardware. However, the initial saving might lead to increased running support costs and higher incompatibility risks.

Directly-Attached Storage

Directly-attached storage (DAS) impacts the server hardware choice and affects host density, instance density, power density, OS-hypervisor, and management tools.

Scalability

Scalability is a major consideration in any OpenStack cloud. It can sometimes be difficult to predict the final intended size of the implementation; consider expanding the initial deployment in order to accommodate growth and user demand.

Expandability

Expandability is a major architecture factor for storage solutions. A storage solution that expands to 50 PB is considered more expandable than a solution that only expands to 10 PB. This metric is different from scalability, which is a measure of the solution's performance as its workload increases. For example, the storage architecture for a development platform cloud might not require the same expandability and scalability as a commercial product cloud.

Fault tolerance

Object Storage resource nodes do not require hardware fault tolerance or RAID controllers. You do not need to plan for fault tolerance in the Object Storage hardware, because the Object Storage service provides replication between zones by default.

Block Storage nodes, Compute nodes, and cloud controllers should have fault tolerance built-in at the hardware level with hardware RAID controllers and varying levels of RAID configuration. The level of RAID should be consistent with the performance and availability requirements of the cloud.

Location

The geographical location of instance and image storage might impact your architecture design.

Performance

Disks that run Object Storage services do not need to be fast-performing disks. You can therefore maximize the cost efficiency per terabyte for storage. However, disks that run Block Storage services should use performance-boosting features that might require SSDs or flash storage to provide high-performance Block Storage pools.

The storage performance of short-term disks that you use for instances should also be considered. If Compute pools need high utilization of short-term storage, or requires very high performance, you should deploy similar hardware solutions the solutions you deploy for Block Storage.

Server type

Scaled-out storage architecture that includes DAS affects the server hardware selection. This architecture can also affect host density, instance density, power density, OS-hypervisor, management tools, and so on.

3.3.5. Ceph Storage

If you consider Ceph for your external storage, the Ceph cluster back-end must be sized to handle the expected number of concurrent VMs with reasonable latency. An acceptable service level can maintain 99% of I/O operations in under 20ms for write operations and in under 10ms for read operations.

You can isolate I/O spikes from other VMs by configuring the maximum bandwidth for each Rados Block Device (RBD) or by setting a minimum guaranteed commitment.

3.4. NETWORK RESOURCES

Network availability is critical to the hypervisors in your cloud deployment. For example, if the hypervisors support only a few virtual machines (VMs) for each node and your applications do not

require high-speed networking, then you can use one or two 1GB ethernet links. However, if your applications require high-speed networking or your hypervisors support many VMs for each node, one or two 10GB ethernet links are recommended.

A typical cloud deployment uses more peer-to-peer communication than a traditional core network topology normally requires. Although VMs are provisioned randomly across the cluster, these VMs need to communicate with each other as if they are on the same network. This requirement might slow down the network and cause packet loss on traditional core network topologies, due to oversubscribed links between the edges and the core of the network.

3.4.1. Segregate Your Services

OpenStack clouds traditionally have multiple network segments. Each segment provides access to resources in the cloud to operators and tenants. The network services also require network communication paths separated from the other networks. Segregating services to separate networks helps to secure sensitive data and protects against unauthorized access to services.

The minimum recommended segragation involves the following network segments:

- A public network segment used by tenants and operators to access the cloud REST APIs. Normally, only the controller nodes and swift proxies in the cloud are required to connect to this network segment. In some cases, this network segment might also be serviced by hardware load balancers and other network devices.

- An administrative network segment used by cloud administrators to manage hardware resources and by configuration management tools to deploy software and services to new hardware. In some cases, this network segment might also be used for internal services, including the message bus and database services that need to communicate with each other.
  Due to the security requirements for this network segment, it is recommended to secure this network from unauthorized access. This network segment usually needs to communicate with every hardware node in the cloud.

- An application network segment used by applications and consumers to provide access to the physical network and by users to access applications running in the cloud. This network needs to be segregated from the public network segment and should not communicate directly with the hardware resources in the cloud.
  This network segment can be used for communication by Compute resource nodes and network gateway services that transfer application data to the physical network outside of the cloud.

3.4.2. General Considerations

Security

Ensure that you segregate your network services and that traffic flows to the correct destinations without crossing through unnecessary locations.
Consider the following example factors:

- Firewalls

- Overlay interconnects for joining separated tenant networks

- Routing through or avoiding specific networks

The way that networks attach to hypervisors can expose security vulnerabilities. To mitigate against exploiting hypervisor breakouts, separate networks from other systems and schedule instances for the network to dedicated Compute nodes. This separation prevents attackers from gaining access to

the networks from a compromised instance.

Capacity planning

Cloud networks require capacity and growth management. Capacity planning can include the purchase of network circuits and hardware with lead times that are measurable in months or years.

Complexity

An complex network design can be difficult to maintain and troubleshoot. Although 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.

Configuration errors

Configuring incorrect IP addresses, VLANs, or routers can cause outages in areas of the network or even in the entire cloud infrastructure. Automate network configurations to minimize the operator error that can disrupt the network availability.

Non-standard features

Configuring the cloud network to take advantage of vendor-specific features might create additional risks.
For example, you might use multi-link aggregation (MLAG) to provide redundancy at the aggregator switch level of the network. MLAG is not a standard aggregation format and each vendor implements a proprietary flavor of the feature. MLAG architectures are not interoperable across switch vendors, which leads to vendor lock-in and can cause delays or problems when you upgrade network components.

Single Point of Failure

If your network has a Single Point Of Failure (SPOF) due to only one upstream link or only one power supply, you might experience a network outage in the event of failure.

Tuning

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

3.4.3. Networking Hardware

There is no single best-practice architecture for networking hardware to support an OpenStack cloud that you can apply to all implementations. Key considerations for the selection of networking hardware include:

Availability

To ensure uninterrupted cloud node access, the network architecture should identify any single points of failure and provide adequate redundancy or fault-tolerance:

- Network redundancy can be achieved by adding redundant power supplies or paired switches.
- For the network infrastructure, networking protocols such as LACP, VRRP or similar can be used to achieve a highly available network connection.
- To ensure that the OpenStack APIs and any other services in the cloud are highly available, you should design a load-balancing solution within the network architecture.

Connectivity

All nodes within an OpenStack cloud require network connectivity. In some cases, nodes require access to multiple network segments. The design must include sufficient network capacity and bandwidth to ensure that all north-south and east-west traffic in the cloud have sufficient resources.

Ports

Any design requires networking hardware that has the required ports:

- Ensure you have the physical space required to provide the ports. A higher port density is preferred, as it leaves more rack space for Compute or storage components. Adequate port availability also prevents fault domains and assists with power density. Higher density switches are more expensive and should also be considered, as it is important to not to over-design the network if it is not required.
- The networking hardware must support the proposed network speed. For example: 1 GbE, 10 GbE, or 40 GbE (or even 100 GbE).

Power

Ensure that the physical data center provides the necessary power for the selected network hardware. For example, spine switches in a leaf-and-spine fabric or end of row (EoR) switches might not provide sufficient power.

Scalability

The network design should include a scalable physical and logical network design. Network hardware should offer the types of interfaces and speeds that are required by the hardware nodes.

3.5. PERFORMANCE

The performance of an OpenStack deployment depends on multiple factors that are related to the infrastructure and controller services. User requirements can be divided to general network performance, Compute resource performance, and storage systems performance.

Ensure that you retain a historical performance baseline of your systems, even when these systems perform consistently with no slow-downs. Available baseline information is a useful reference when you encounter performance issues and require data for comparison purposes.

In addition to Section 1.5.2, “OpenStack Telemetry (ceilometer)”, external software can also be used to track performance. The Operational Tools repository for Red Hat OpenStack Platform includes the following tools:

- collectd
- Graphite-web
- InfluxDB
- Grafana

3.5.1. Network Performance

The network requirements help to determine performance capabilities. For example, smaller deployments might employ 1 Gigabit Ethernet (GbE) networking, and larger installations that serve multiple departments or many users should use 10 GbE networking.

The performance of running instances might be limited by these network speeds. You can design OpenStack environments that run a mix of networking capabilities. By utilizing the different interface speeds, the users of the OpenStack environment can choose networks that fit their purposes.

For example, web application instances can run on a public network with OpenStack Networking with 1 GbE capability, and the back end database can use an OpenStack Networking network with 10 GbE capability to replicate its data. In some cases, the design can incorporate link aggregation to increase throughput.

Network performance can be boosted by implementing hardware load balancers that provide front-end services to the cloud APIs. The hardware load balancers can also perform SSL termination if necessary. When implementing SSL offloading, it is important to verify the SSL offloading capabilities of the selected devices.

3.5.2. Compute Nodes Performance

Hardware specifications used in compute nodes including CPU, memory, and disk type, directly affect the performance of the instances. Tunable parameters in the OpenStack services can also directly affect performance.

For example, the default over-commit ratio for OpenStack Compute is 16:1 for CPU and 1.5 for memory. These high ratios can lead to an increase in "noisy-neighbor" activity. You must carefully size your Compute environment to avoid this scenario and ensure that you monitor your environment when usage increases.

3.5.3. Block Storage Hosts Performance

Block Storage can use enterprise back end systems such as NetApp or EMC, scale-out storage such as Ceph, or utilize the capabilities of directly-attached storage in the Block Storage nodes.

Block Storage can be deployed to enable traffic to traverse the host network, which could affect, and be adversely affected by, the front-side API traffic performance. Therefore, consider using a dedicated data storage network with dedicated interfaces on the controller and Compute hosts.

3.5.4. Object Storage Hosts Performance

Users typically access Object Storage through the proxy services, which run behind hardware load balancers. By default, highly resilient storage system replicate stored data, which can affect the overall system performance. In this case, 10 GbE or higher networking capacity is recommended across the storage network architecture.

3.5.5. Controller Nodes

Controller nodes provide management services to the end-user and provide services internally for the cloud operation. It is important to carefully design the hardware that is used to run the controller infrastructure.

The controllers run message-queuing services for system messaging between the services. Performance issues in messaging can lead to delays in operational functions such as spinning up and deleting instances, provisioning new storage volumes, and managing network resources. These delays can also adversely affect the ability of the application to react to some conditions, especially when using auto-scaling features.

You also need to ensure that controller nodes can handle the workload of multiple concurrent users. Ensure that the APIs and Horizon services are load-tested to improve service reliability for your customers.

It is important to consider the OpenStack Identity Service (keystone), which provides authentication and authorization for all services, internally to OpenStack and to end-users. This service can lead to a degradation of overall performance if it is not sized appropriately.

Metrics that are critically important to monitor include:

- Image disk utilization
- Response time to the Compute API

3.6. MAINTENANCE AND SUPPORT

To support and maintain an installation, OpenStack cloud management requires the operations staff to understand the architecture design. The skill level and role separation between the operations and engineering staff depends on the size and purpose of the installation.

- Large cloud service providers, or telecom providers, are more likely to be managed by specially-trained, dedicated operations organization.
- Smaller implementations are more likely to rely on support staff that need to take on combined engineering, design and operations functions.

If you incorporate features that reduce the operations overhead in the design, you might be able to automate some operational functions.

Your design is also directly affected by terms of Service Level Agreements (SLAs). SLAs define levels of service availability and usually include penalties if you do not meet contractual obligations. SLA terms that affect the design include:

- API availability guarantees that imply multiple infrastructure services and highly available load balancers.
- Network uptime guarantees that affect switch design and might require redundant switching and power.
- Network security policy requirements that imply network segregation or additional security mechanisms.

3.6.1. Backups

Your design might be affected by your backup and restore strategy, data valuation or hierarchical storage management, retention strategy, data placement, and workflow automation.

3.6.2. Downtime

An effective cloud architecture should support the following:

- Planned downtime (maintenance)
- Unplanned downtime (system faults)

For example, if a compute host fails, instances might be restored from a snapshot or by re-spawning an instance. However, for high availability you might need to deploy additional support services such as shared storage or design reliable migration paths.

3.7. AVAILABILITY

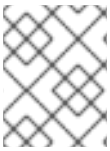
OpenStack can provide a highly-available deployment when you use at least two servers. The servers can run all services from the RabbitMQ message queuing service and the MariaDB database service.

When you scale services in the cloud, back end services also need to scale. Monitoring and reporting server utilization and response times, as well as load testing your systems, can help determine scaling decisions.

- To avoid a single point of failure, OpenStack services should be deployed across multiple servers. API availability can be achieved by placing these services behind highly-available load balancers with multiple OpenStack servers as members.
- Ensure that your deployment has adequate backup capabilities. For example, in a deployment with two infrastructure controller nodes using high availability, if you lose one controller you can still run the cloud services from the other.
- OpenStack infrastructure is integral to provide services and should always be available, especially when operating with SLAs. Consider the number of switches, routes and redundancies of power that are necessary for the core infrastructure, as well as the associated bonding of networks to provide diverse routes to a highly available switch infrastructure.
- If you do not configure your Compute hosts for live-migration and a Compute host fails, the Compute instance and any data stored on that instance might be lost. To do ensure the uptime of your Compute hosts, you can use shared file systems on enterprise storage or OpenStack Block storage.

External software can be used to check service availability or threshold limits and to set appropriate alarms. The Operational Tools repository for Red Hat OpenStack Platform includes:

- Sensu
- Uchiwa dashboard



NOTE

For a reference architecture that uses high availability in OpenStack, see: Deploying Highly Available Red Hat OpenStack Platform 6 with Ceph Storage

3.8. SECURITY

A security domain includes users, applications, servers, or networks that share common trust requirements and expectations in a single system. Security domains typically use the same authentication and authorization requirements and users.

Typical security domain categories are Public, Guest, Management, and Data. The domains can be mapped to an OpenStack deployment individually or combined. For example, some deployment topologies combine guest and data domains in one physical network, and in other cases the networks are physically separated. Each case requires the cloud operator to be aware of the relevant security concerns.

Security domains should be mapped to your specific OpenStack deployment topology. The domains and the domain trust requirements depend on whether the cloud instance is public, private, or hybrid.

Public domain

Entirely untrusted area of the cloud infrastructure. A public domain can refer to the Internet as a whole or networks over which you have no authority. This domain should always be considered untrusted.

Guest domain

Typically used for Compute instance-to-instance traffic and handles Compute data generated by instances on the cloud, but not by services that support the operation of the cloud, such as API calls. Public cloud providers and private cloud providers that do not have strict controls on instance usage or that allow unrestricted Internet access to instances, should consider this domain untrusted. Private cloud providers might consider this network internal and therefore trusted only with controls that assert trust in instances and all cloud tenants.

Management domain

The domain where services interact with each other. This domain is sometimes known as the *control plane*. Networks in this domain transfer confidential data, such as configuration parameters, user names, and passwords. In most deployments, this domain is considered trusted.

Data domain

The domain where storage services transfer data. Most data that crosses this domain has high integrity and confidentiality requirements and, depending on the type of deployment, might also have strong availability requirements. The trust level of this network is dependent on other deployment decisions.

When you deploy OpenStack in an enterprise as a private cloud, the deployment is usually behind a firewall and inside the trusted network with existing systems. Users of the cloud are typically employees that are bound by the security requirements that the company defines. This deployment implies that most of the security domains can be trusted.

However, when you deploy OpenStack in a public facing role, no assumptions can be made regarding the trust level of the domains, and the attack vectors significantly increase. For example, the API endpoints and the underlying software become vulnerable to bad actors that want to gain unauthorized access or prevent access to services. These attacks might lead to loss data, functionality, and reputation. These services must be protected using auditing and appropriate filtering.

You must exercise caution also when you manage users of the system for both public and private clouds. The Identity service can use external identity back ends such as LDAP, which can ease user management inside OpenStack. Because user authentication requests include sensitive information such as user names, passwords, and authentication tokens, you should place the API services behind hardware that performs SSL termination.

3.9. ADDITIONAL SOFTWARE

A typical OpenStack deployment includes OpenStack-specific components and Section 1.6.1, “Third-party Components”. Supplemental software can include software for clustering, logging, monitoring, and alerting. The deployment design must therefore account for additional resource consumption, such as CPU, RAM, storage, and network bandwidth.

When you design your cloud, consider the following factors:

Databases and messaging

The underlying message queue provider might affect the required number of controller services, as well as the technology to provide highly resilient database functionality. For example, if you use MariaDB with Galera, the replication of services relies on quorum. Therefore, the underlying database should consist of at least three nodes to account for the recovery of a failed Galera node.

When you increase the number of nodes to support a software feature, consider both rack space and switch port density.

External caching

Memcached is a distributed memory object caching system, and Redis is a key-value store. Both systems can be deployed in the cloud to reduce load on the Identity service. For example, the memcached service caches tokens, and uses a distributed caching system to help reduce some bottlenecks from the underlying authentication system.
Using memcached or Redis does not affect the overall design of your architecture, because these services are typically deployed to the infrastructure nodes that provide the OpenStack services.

Load balancing

Although many general-purpose deployments use hardware load balancers to provide highly available API access and SSL termination, software solutions such as HAProxy can also be considered. You must ensure that software-defined load balancing implementations are also highly available.
You can configure software-defined high availability with solutions such as Keepalived or Pacemaker with Corosync. Pacemaker and Corosync can provide active-active or active-passive highly available configuration based on the specific service in the OpenStack environment.

These applications might affect the design because they require a deployment with at least two nodes, where one of the controller nodes can run services in standby mode.

Logging and monitoring

Logs should be stored in a centralized location to make analytics easier. Log data analytics engines can also provide automation and issue notification with mechanisms to alert and fix common issues.
You can use external logging or monitoring software in addition to the basic OpenStack logs, as long as the tools support existing software and hardware in your architectural design. The Operational Tools repository for Red Hat OpenStack Platform includes the following tools:

- Fluentd
- ElasticSearch
- Kibana

3.10. PLANNING TOOL

The Cloud Resource Calculator tool can help you calculate capacity requirements.

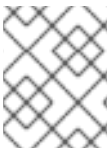
To use the tool, enter your hardware details into the spreadsheet. The tool then shows a calculated estimate of the number of instances available to you, including flavor variations.

IMPORTANT

This tool is provided only for your convenience. It is not officially supported by Red Hat.

CHAPTER 4. ARCHITECTURE EXAMPLES

This chapter contains references to architecture examples of Red Hat OpenStack Platform deployments.



NOTE

All architecture examples in this guide assume that you deploy OpenStack Platform on Red Hat Enterprise Linux 7.3 with the KVM hypervisor.

4.1. OVERVIEW

Typically, deployments are based on performance or functionality. Deployments can also be based on deployed infrastructure.

Table 4.1. Deployments based on functionality or performance

Example	Description
Section 4.2, "General-Purpose Architecture"	General high-availability cloud to use if you are unsure of specific technical or environmental needs. This architecture type is flexible, does not emphasize any single OpenStack component, and is not restricted to specific environments.
Section 4.3, "Compute-Focused Architecture"	Compute-focused cloud specifically that supports compute-intensive workloads. Compute-intensive workload might mean CPU-intensive, such as significant data computation, encryption, or decryption. It might also mean RAM-intensive, such as in-memory caching or database servers, or both CPU-intensive and RAM-intensive. It does not normally mean storage-intensive or network-intensive. You can choose this architecture type if you require high performance Compute resources.
Section 4.4.2, "Data Analytics Architecture"	Performance-focused storage system designed for management and analysis of large data sets, such as Hadoop clusters. In this architecture type, OpenStack integrates with Hadoop to manage the Hadoop cluster with Ceph as the storage back-end.
Section 4.4.3, "High-Performance Database Architecture"	High-performance storage system that assumes increased database IO requirements and utilizes a solid-state drive (SSD) to process data. You can use this architecture type for existing storage environments.
Section 4.5.2, "Cloud Storage and Backup Architecture"	Cloud-based file storage and sharing service, commonly used in OpenStack deployments. This architecture type uses a cloud backup application, where incoming data to the cloud traffic is higher than the outgoing data.
Section 4.5.3, "Large-Scale Web-Application Architecture"	Hardware-based load balancing cluster for a large-scale Web application. This architecture type provides SSL-offload functionality and connects to tenant networks to reduce address consumption and scale the Web application horizontally.

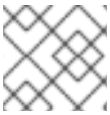
4.2. GENERAL-PURPOSE ARCHITECTURE

You can deploy a general high availability cloud if you are unsure of specific technical or environmental needs. This flexible architecture type does not emphasize any single OpenStack component, and it is not restricted to particular environments.

This architecture type covers 80% of potential use cases, including:

- Simple database
- Web application runtime environment
- Shared application development environment
- Test environment
- Environment requiring scale-out additions instead of than scale-up additions

This architecture type is not recommended for cloud domains that require increased security.



NOTE

For installation and deployment documentation, see Chapter 5, *Deployment Information*.

4.2.1. Example Use Case

An online classified advertising company wants to run web applications that include Apache Tomcat, Nginx and MariaDB in a private cloud. To meet policy requirements, the cloud infrastructure will run inside the company data center.

The company has predictable load requirements, but requires scaling to cope with nightly increases in demand. The current environment does not have the flexibility to align with the company goal of running an open-source API environment.

The current environment consists of the following components:

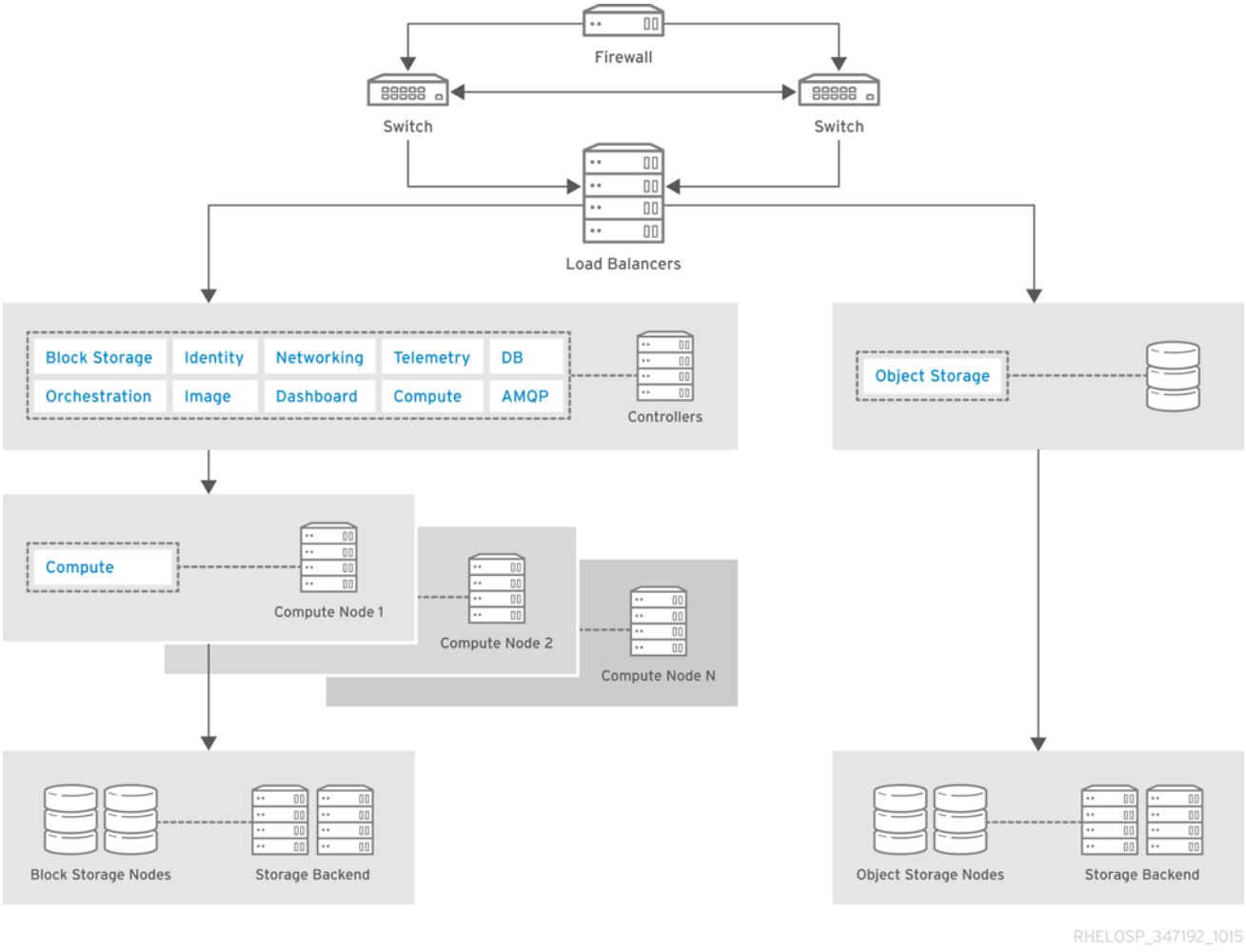
- Between 120 and 140 installations of Nginx and Tomcat, each with 2 vCPUs and 4 GB of RAM
- A three-node MariaDB and Galera cluster, each with 4 vCPUs and 8 GB RAM. Pacemaker is used to manage the Galera nodes.

The company runs hardware load balancers and multiple web applications that serve the websites. Environment orchestration uses combinations of scripts and Puppet. The website generates large amounts of log data every day that need to be archived.

4.2.2. About the Design

The architecture for this example includes three controller nodes and at least eight Compute nodes. It uses OpenStack Object Storage for static objects and OpenStack Block Storage for all other storage needs.

To ensure that the OpenStack infrastructure components are highly available, nodes use the Pacemaker add-on for Red Hat Enterprise Linux together with HAProxy.



The architecture includes the following components:

- Firewall, switches, and hardware load balancers for the public-facing network connections.
- OpenStack controller service that run Image, Identity, and Networking, combined with the support services MariaDB and RabbitMQ. These services are configured for high availability on at least three controller nodes.
- Cloud nodes are configured for high availability with the Pacemaker add-on for Red Hat Enterprise Linux.
- Compute nodes use OpenStack Block Storage for instances that need persistent storage.
- OpenStack Object Storage to serve static objects, such as images.

4.2.3. Architecture Components

Component	Description
Block Storage	Persistant storage for instances.
Compute controller services	Compute management and scheduling services that run on the controller.
Dashboard	Web console for OpenStack management.

Component	Description
Identity	Basic authentication and authorization for users and tenants.
Image	Stores images to be used for booting instances and managing snapshots.
MariaDB	Database for all OpenStack components. MariaDB server instances store data on shared enterprise storage, such as NetApp or Solidfire. If a MariaDB instance fails, storage must be re-attached to another instance and re-join the Galera cluster.
Networking	Controls hardware load balancers with plug-ins and the Networking API. If you increase OpenStack Object Storage, you must consider network bandwidth requirements. It is recommended to run OpenStack Object Storage on network connections with 10 GbE or higher.
Object Storage	Processes and archives logs from the web application servers. You can also use Object Storage to move static web content from OpenStack Object Storage containers or to back up images that are managed by OpenStack Image.
Telemetry	Monitoring and reporting for other OpenStack services.

4.2.4. Compute Node Requirements

The Compute service is installed on each of the Compute nodes.

This general-purpose architecture can run up to 140 web instances, and the small number of MariaDB instances requires 292 vCPUs and 584 GB RAM. On a typical 1U server with dual-socket hex-core Intel CPUs with Hyperthreading, and assuming 2:1 CPU over-commit ratio, this architecture requires eight Compute nodes.

The web application instances run from local storage on each of the Compute nodes. The web application instances are stateless, so in case one of the instances fails, the application can continue to run.

4.2.5. Storage Requirements

For storage, use a scaled-out solution with directly-attached storage in the servers. For example, you can populate storage in the Compute hosts in a way that is similar to a grid-computing solution, or in dedicated hosts that provide block storage exclusively.

If you deploy storage in the Compute hosts, ensure that the hardware can handle the storage and compute services.

4.3. COMPUTE-FOCUSED ARCHITECTURE

NOTE

Cells are available in this release as a **Technology Preview**, and therefore are not fully supported by Red Hat. They should only be used for testing, and should not be deployed in a production environment. For more information about Technology Preview features, see Scope of Coverage Details.

A compute-focused cloud supports CPU-intensive workloads such as data computation or encryption and decryption, RAM-intensive workloads such as in-memory caching or database servers, or both. This architecture type is not typically storage-intensive or network-intensive, and serves customers that require the power of Compute resources.

Compute-focused workloads include the following use cases:

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

A compute-focused OpenStack cloud does not typically use raw block storage services because the cloud does not generally host applications that require persistent block storage. Infrastructure components are not shared, so that the workloads can consume as many available resources as needed. Infrastructure components need to also be highly available. This design uses load balancers. You can also use HAProxy.

NOTE

For installation and deployment documentation, see Chapter 5, *Deployment Information*.

4.3.1. Example Use Case

An organization provides HPC for research projects, and needs to add a third compute center to two existing compute centers in Europe.

The following table lists the requirements for each compute center to add:

Data Center	Approximate capacity
Geneva, Switzerland	• 3.5 Mega Watts • 91000 cores • 120 PB HDD • 100 PB Tape • 310 TB Memory

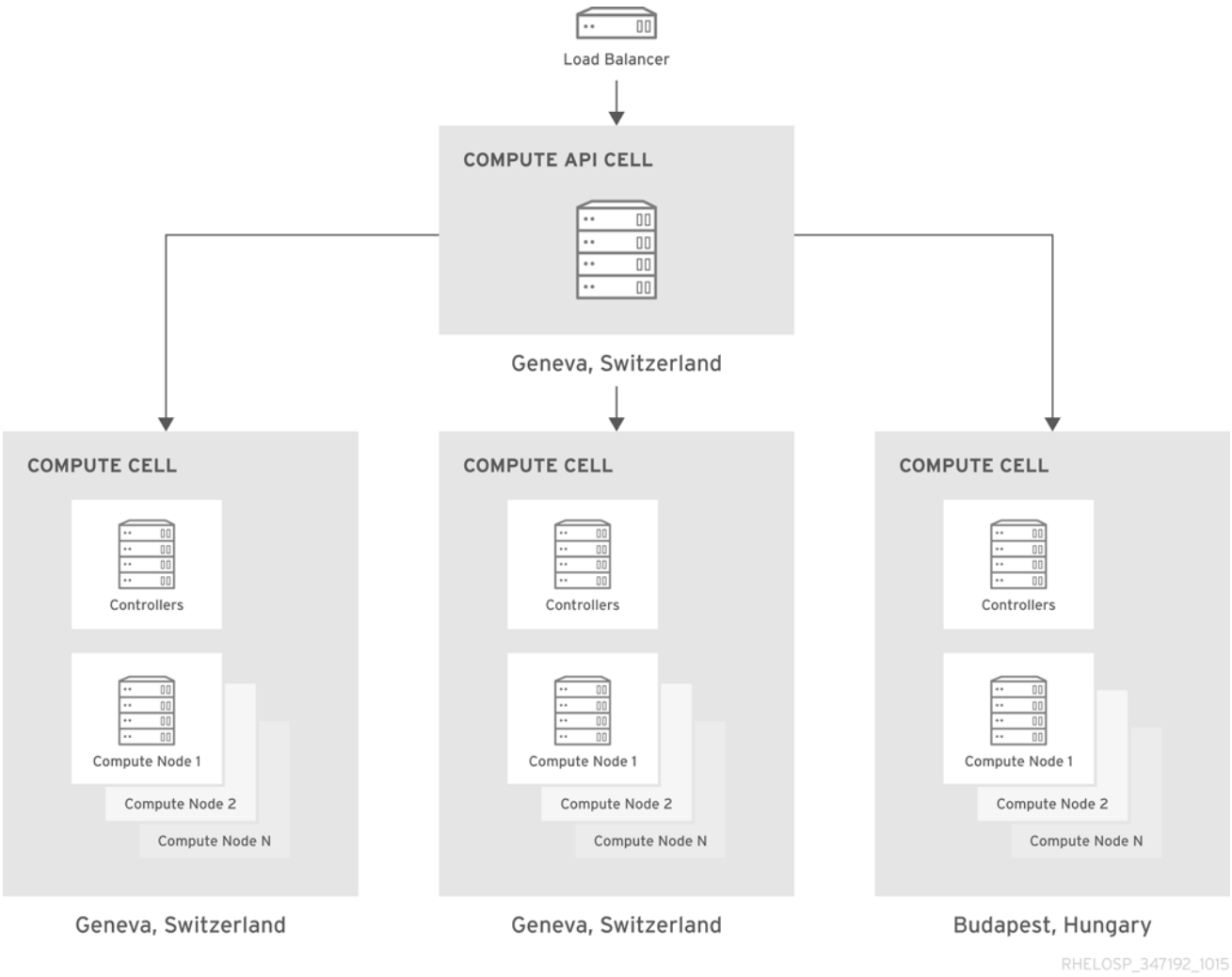
Data Center	Approximate capacity
Budapest, Hungary	• 2.5 Mega Watts • 20000 cores • 6 PB HDD

4.3.2. About the Design

This architecture uses cells for segregation of compute resources and for transparent scaling between different data centers. This decision impacts support for security groups and live migration. In addition, it requires manual replication of some configuration elements, such as flavors, across cells. However, cells provide the required scale while exposing a single public API endpoint to users.

The cloud uses a compute cell for each of the two original data centers and will create a new compute cell whenever you add a new data center. Each cell contains three availability zones to further segregate compute resources and at least three RabbitMQ message brokers configured for clustering with mirrored queues for high availability.

The API cell, which resides behind an HAProxy load balancer, is in the data center in Switzerland. The API cell directs API calls to compute cells using a customized variation of the cell scheduler. The customizations allow certain workloads to route to a specific data center or to all data centers based on cell RAM availability.



You can also use filters to customize the Compute scheduler that handles placement in the cells. For example, **ImagePropertiesFilter** provides special handling based on the operating system that the guest runs, for example Linux or Windows.

A central database team manages the SQL database server in each cell in an active/passive configuration with a NetApp storage back-end. Backups run every six hours.

4.3.3. Architecture Components

Component	Description
Compute	Compute management and scheduling services run on the controller. The Compute service also runs on each compute node.
Dashboard Service	GUI for OpenStack management.
Identity Service	Basic authentication and authorization functionality.
Image Service	Runs in the API cell and maintains a small set of Linux images, onto which orchestration tools can place applications.
Networking	Networking services. For more information about OpenStack Networking, see Chapter 2, *Networking In-Depth*.
Monitoring	Telemetry service performs metering to adjust project quotas with a sharded, replicated MongoDB back-end. To spread the API load, you must deploy instances of the **openstack-nova-api** service in the child cells that Telemetry can query. This deployment also requires configuration of supporting services, such as Identity and Image, in the child cells. Specific critical metrics to capture include Image disk utilization and response time to the Compute API.
Orchestration Service	Automatically deploys and tests new instances.
Telemetry Service	Supports the Orchestration auto-scaling feature.
Object Storage	Stores objects with a 3 PB Ceph cluster.

4.3.4. Design Considerations

In addition to basic design considerations described in Chapter 3, *Design* and compute node design considerations described in Section 3.2, "Compute Resources", the following items should be considered for a compute-intensive architecture.

Workloads

Short-lived workloads can include continuous integration and continuous deployment (CI-CD) jobs, which create large numbers of compute instances simultaneously to perform a set of compute-intensive tasks. The environment then copies the results or the artifacts from each instance to long-term storage before it terminates the instances.

Long-lived workloads, such as a Hadoop cluster or an HPC cluster, typically receive large data sets, perform the computational work on those data sets, and then push the results to long-term storage. When the computational work ends, the instances are idle until they receive another job. Environments for long-lived workloads are often larger and more complex, but you can offset the cost of building these environments by keeping them active between jobs.

Workloads in a compute-focused OpenStack cloud generally do not require persistent block storage, except some uses of Hadoop with HDFS. A shared file system or object store maintains initial data sets and serves as the destination for saving the computational results. By avoiding input-output (IO) overhead, you can significantly enhance workload performance. Depending on the size of the data sets, you might need to scale the object store or the shared file system.

Expansion Planning

Cloud users expect instant access to new resources as needed. Therefore, you must plan for typical usage and for sudden spikes in resource demand. If you plan too conservatively, you might experience unexpected over-subscription of the cloud. If you plan too aggressively, you might experience unexpected underutilization of the cloud unnecessary operations and maintenance costs. The key factor in expansion planning is analytics of trends in cloud usage over time. Measure the consistency with which you deliver services instead of the average speed or capacity of the cloud. This information can help you model capacity performance and determine the current and future capacity of the cloud.

For information about monitoring software, see Section 3.9, “Additional Software”.

CPU and RAM

Typical server offerings today include CPUs with up to 12 cores. In addition, some Intel CPUs support Hyper-Threading Technology (HTT), which doubles the core capacity. Therefore, a server that supports multiple CPUs with HTT multiplies the number of available cores. HTT is an Intel proprietary simultaneous multi-threading implementation that is used to improve parallelization on the Intel CPUs. Consider enabling HTT to improve the performance of multi-threaded applications. The decision to enable HTT on a CPU depends on the use case. For example, disabling HTT can help intense computing environments. Running performance tests of local workloads with and without HTT can help determine which option is more appropriate for a particular case.

Capacity Planning

You can add capacity to your compute environment with one or more of the following strategies:

- Horizontally scale by adding extra capacity to the cloud. You should use the same, or similar CPUs in the extra nodes to reduce the chance of breaking any live-migration features. Scaling out hypervisor hosts also affects network and other data center resources. Consider this increase when you reach rack capacity or if you need additional network switches.

NOTE

Be aware of the additional work required to place the nodes in appropriate availability zones and host aggregates.

- Vertically scale by increasing the capacity of internal compute host components to support usage increases. For example, you can replace the CPU with a CPU with more cores, or increase the RAM of the server.

- Assess your average workload, and if need be, increase the number of instances that can run in the compute environment by adjusting the over-commit ratio.

IMPORTANT

Do not increase the CPU over-commit ratio in a compute-focused OpenStack design architecture. Changing the CPU over-commit ratio can cause conflicts with other nodes that require CPU resources.

Compute Hardware

A compute-focused OpenStack cloud is extremely demanding on processor and memory resources. Therefore, you should prioritize server hardware that can offer more CPU sockets, more CPU cores, and more RAM.

Network connectivity and storage capacity are less critical to this architecture. The hardware must provide enough network connectivity and storage capacity to meet minimum user requirements, but the storage and networking components primarily load data sets to the computational cluster and do not require consistent performance.

Storage Hardware

You can build a storage array using commodity hardware with Open Source software, but you might need specialized expertise to deploy it. You can also use a scale-out storage solution with direct-attached storage in the servers, but you must ensure that the server hardware supports the storage solution.

Consider the following factors when you design your storage hardware:

- Availability. If instances must be highly available or capable of migration between hosts, use a shared storage file system for ephemeral instance data to ensure that compute services can run uninterrupted in the event of a node failure.

- Latency. Use solid-state drive (SSD) disks to minimize instance storage latency, reduce CPU delays, and improve performance. Consider using RAID controller cards in compute hosts to improve the performance of the underlying disk sub-system.

- Performance. Although a compute-focused cloud does not usually require major data I/O to and from storage, storage performance is still an important factor to consider. You can measure the storage hardware performance by observing the latency of storage I/O requests. In some compute-intensive workloads, minimizing the delays that the CPU experiences while fetching data from storage can significantly improve the overall performance of the application.

- Expandability. Decide the maximum capacity of the storage solution. For example, a solution that expands to 50 PB is more expandable than a solution that only expands to 10PB. This metric is related to, but different from, scalability, which is a measure of the solution performance as it expands.

- Connectivity. The connectivity must satisfy the storage solution requirements. If you select a centralized storage array, determine how to connect the hypervisors to the storage array. Connectivity can affect latency and performance. Therefore, ensure that the network characteristics minimize latency to boost the overall performance of the environment.

Network Hardware

In addition to basic network considerations described in Chapter 2, *Networking In-Depth*, consider the following factors:

- The required port count affects the physical space that a network design requires. For example, a switch that provides 48 ports with 10 GbE capacity for each port in a 1U server has higher port density than a switch that provides 24 ports with 10 GbE capacity for each port in

a 2U server. Higher port density allows more rack space for compute or storage components. You must also consider fault domains and power density. Although more expensive, you can also consider higher density switches as you should not design the network beyond functional requirements.

- You should design the network architecture with a scalable network model that helps to add capacity and bandwidth, such as the leaf-spline 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 the required port count, port speed, and port density, and that also allows future growth when workload demands increase. It is also important to evaluate where in the network architecture it is valuable to provide redundancy. Increased network availability and redundancy can be expensive, so you should compare the extra cost with the benefits of redundant network switches and bonded interfaces at the host level.

4.4. STORAGE-FOCUSED ARCHITECTURES

4.4.1. Storage-Focused Architecture Types

The cloud storage model stores data in logical pools on physical storage devices. This architecture is often referred to as an integrated storage cloud.

Cloud storage commonly refers to a hosted object storage service. However, the term can also include other types of data storage that are available as a service. OpenStack offers both Block Storage (cinder) and Object Storage (swift). Cloud storage typically runs on a virtual infrastructure and resembles broader cloud computing in interface accessibility, elasticity, scalability, multi-tenancy, and metered resources.

You can use cloud storage services on-premise or off-premises. Cloud storage is highly fault tolerant with redundancy and data distribution, is highly durable with versioned copies, and can perform consistent data replication.

Example cloud storage applications include:

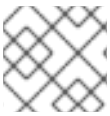
- Active archive, backups and hierarchical storage management
- General content storage and synchronization such as a private DropBox service
- Data analytics with parallel file systems
- Unstructured data store for services such as social media back-end storage
- Persistent block storage
- Operating system and application image store

- Media streaming
- Databases
- Content distribution
- Cloud storage peering

For more information about OpenStack storage services, see Section 1.2.2, "OpenStack Object Storage (swift)" and Section 1.2.1, "OpenStack Block Storage (cinder)".

4.4.2. Data Analytics Architecture

Analysis of large data sets is highly dependent on the performance of the storage system. Parallel file systems can provide high-performance data processing and are recommended for large scale performance-focused systems.

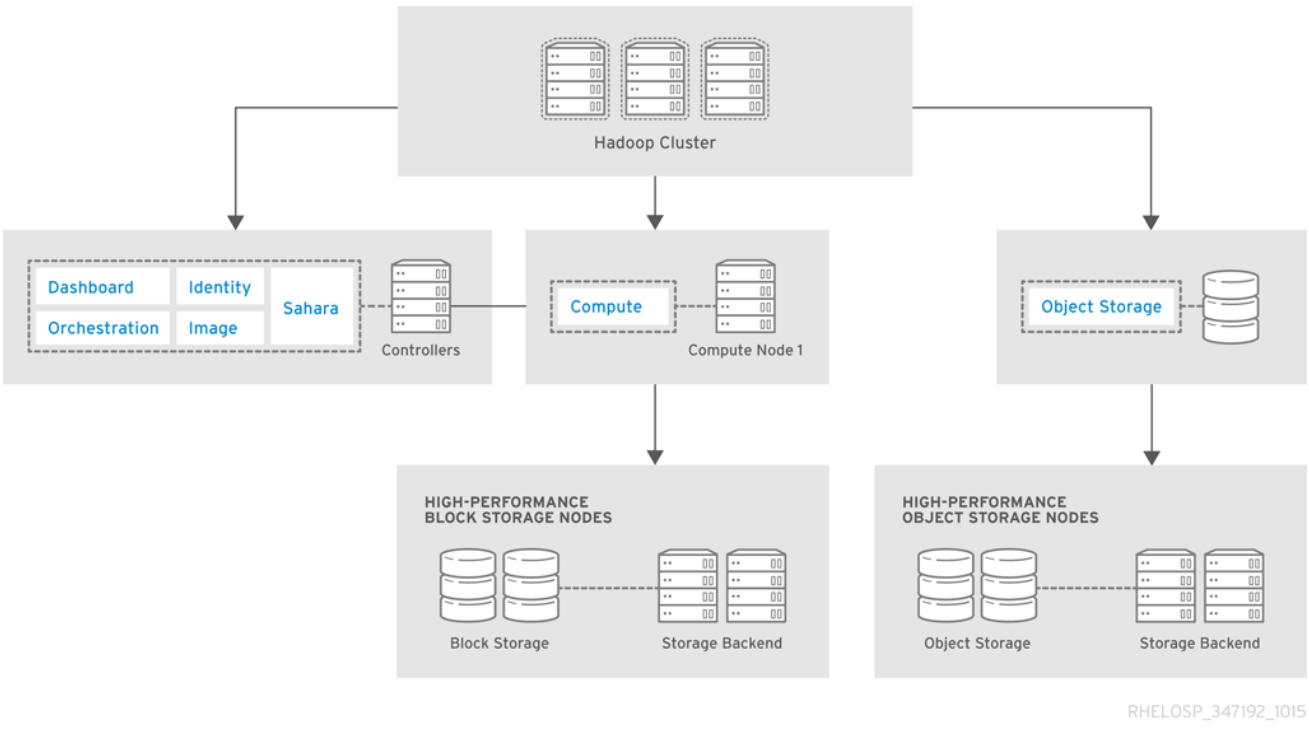


NOTE

For installation and deployment documentation, see Chapter 5, *Deployment Information*.

4.4.2.1. About the Design

OpenStack Data Processing (sahara) integrates with Hadoop to manage the Hadoop cluster inside the cloud. The following diagram shows an OpenStack store with a high-performance requirement.



The hardware requirements and configuration are similar to the high-performance architecture described in Section 4.4.3, "High-Performance Database Architecture". In this example, the architecture uses the Ceph Swift-compatible REST interface that connects to a caching pool and enables acceleration of the available pool.

4.4.2.2. Architecture Components

Component	Description
Compute	Compute management and scheduling services run on the controller. The Compute service also runs on each compute node.
Dashboard	Web console for OpenStack management.
Identity	Basic authentication and authorization functionality.
Image	Stores images to be used for booting instances and managing snapshots. This service runs on the controller and offers a small set of images.
Networking	Networking services. For more information about OpenStack Networking, see Chapter 2, *Networking In-Depth*.
Telemetry	Monitoring and reporting for other OpenStack services. Use this service to monitor instance usage and adjust project quotas.
Object Storage	Stores data with the Hadoop back-end.
Block Storage	Stores volumes with the Hadoop back-end.
Orchestration	Manages templates for instances and block storage volume. Use this service to launch additional instances for storage-intensive processing, with Telemetry for auto-scaling.

4.4.2.3. Cloud Requirements

Requirement	Description
Performance	To boost performance, you can choose specialty solutions to cache disk activity.
Security	You must protect data both in transit and at rest.
Storage proximity	In order to provide high performance or large amounts of storage space, you might need to attach the storage to each hypervisor or serve it from a central storage device.

4.4.2.4. Design Considerations

In addition to basic design considerations described in Chapter 3, *Design*, you should also follow the considerations described in Section 4.4.4, "Storage-Focused Architecture Considerations".

4.4.3. High-Performance Database Architecture

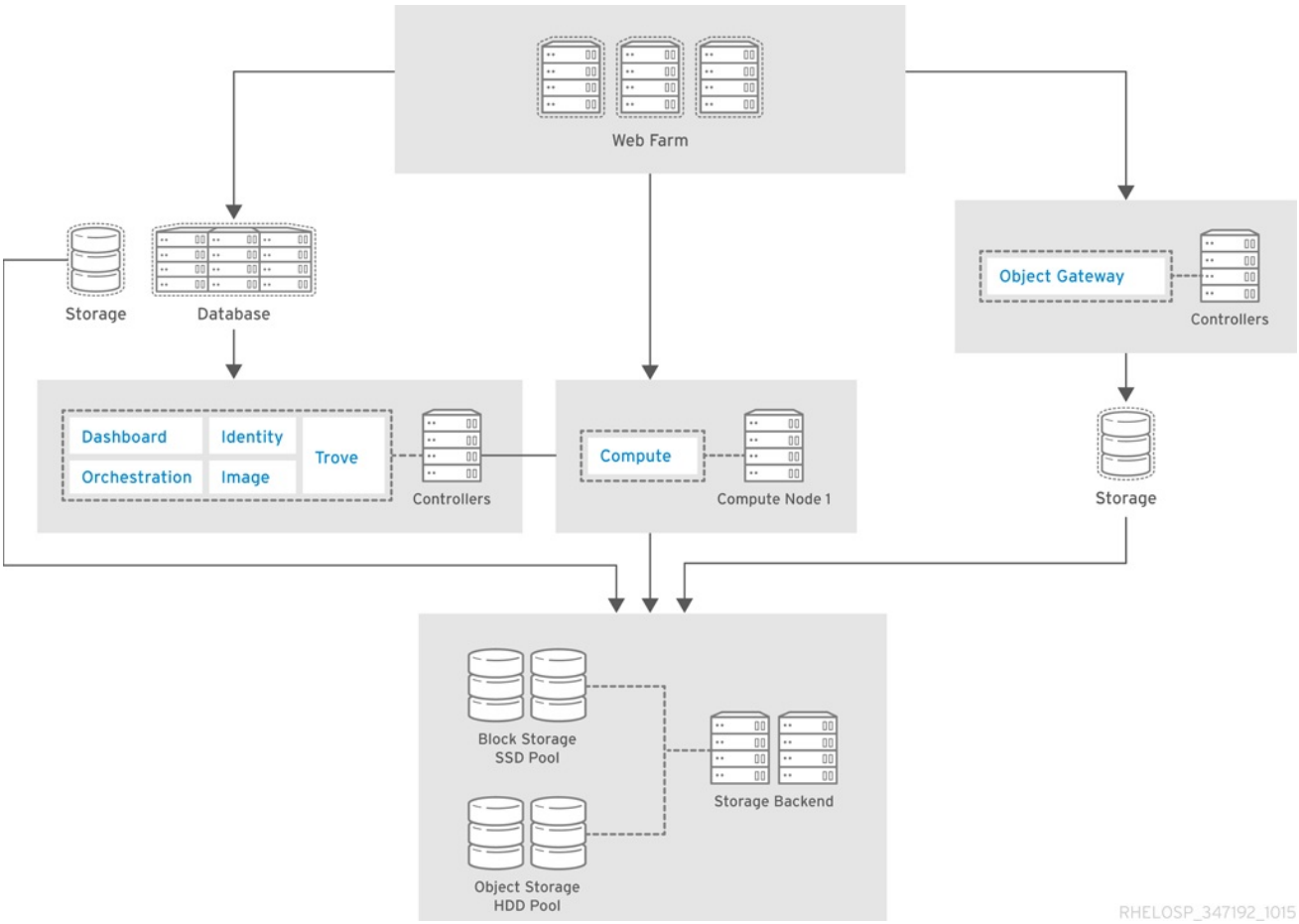
Database architectures benefit from high performance storage back-ends. Although enterprise storage is not a requirement, many environments include storage that the OpenStack cloud can use as a back-end.

You can create a storage pool to provide block devices with OpenStack Block Storage for instances and object interfaces. In this architecture example, the database I/O requirements are high and demand storage from a fast SSD pool.

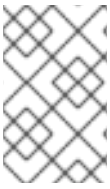
4.4.3.1. About the Design

The storage system uses a LUN backed with a set of SSDs in a traditional storage array, and uses OpenStack Block Storage integration or a storage platform such as Ceph.

This system can provide additional performance capabilities. In the database example, 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 provides a Swift-compatible REST interface, as well as block-level storage from a distributed storage cluster. It is highly flexible and enables reduced cost of operations with features such as self-healing and auto-balancing. Erasure coded pools are recommended to maximize the amount of usable space.



NOTE

Erasure coded pools require special considerations, such as higher computational requirements and limitations on which operations are allowed on an object. Erasure coded pools do not support partial writes.

4.4.3.2. Architecture Components

Component	Description
Compute	Compute management and scheduling services run on the controller. The Compute service also runs on each compute node.
Dashboard	Web console for OpenStack management.
Identity	Basic authentication and authorization functionality.
Image	Stores images to be used for booting instances and managing snapshots. This service runs on the controller and offers a small set of images.
Networking	Networking services. For more information about OpenStack Networking, see Chapter 2, *Networking In-Depth*.
Telemetry	Monitoring and reporting for other OpenStack services. Use this service to monitor instance usage and adjust project quotas.
Monitoring	Use the Telemetry service to perform metering for the purposes of adjusting project quotas.
Object Storage	Stores data with the Ceph back-end.
Block Storage	Stores volumes with the Ceph back-end.
Orchestration	Manages templates for instances and block storage volume. Use this service to launch additional instances for storage-intensive processing, with Telemetry for auto-scaling.

4.4.3.3. Hardware Requirements

You can use an SSD cache layer to link block devices directly to hypervisors or to instances. The REST interface can also use the SSD cache systems as an inline cache.

Component	Requirement	Network

Component	Requirement	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 with 12x4 TB disks for each server, which equals 480 TB total space with approximately 160 TB of usable space after 3 replicas

4.4.3.4. Design Considerations

In addition to basic design considerations described in Chapter 3, *Design*, you should also follow the considerations described in Section 4.4.4, "Storage-Focused Architecture Considerations".

4.4.4. Storage-Focused Architecture Considerations

In addition to basic design considerations described in Chapter 3, *Design* and to storage node design described in Section 3.3, "Storage Resources", the following items should be considered for a storage-intensive architecture.

Connectivity

Ensure the connectivity matches the storage solution requirements. If you select a centralized storage array, determine how to connect the hypervisors to the array. Connectivity can affect latency and performance. Confirm that the network characteristics minimize latency to boost the overall performance of the design.

Density

- Instance density. In a storage-focused architecture, instance density and CPU/RAM over-subscription are lower. You need more hosts to support the anticipated scale, especially if the design uses dual-socket hardware designs.
- Host density. You can address the higher host count with a quad-socket platform. This platform decreases host density and increases rack count. This configuration affects the number of power connections and also impacts network and cooling requirements.
- Power and cooling. The power and cooling density requirements might be lower with 2U, 3U, or 4U servers than with blade, sled, or 1U server designs. This configuration is recommended for data centers with older infrastructure.

Flexibility

Organizations need to have the flexibility to choose between off-premise and on-premise cloud storage options. For example, continuity of operations, disaster recovery, security, and records retention laws, regulations, and policies can impact the cost-effectiveness of the storage provider.

Latency

Solid-state drives (SSDs) can minimize latency for instance storage and reduce CPU delays that storage latency might cause. Evaluate the gains from using RAID controller cards in compute hosts to improve the performance of the underlying disk sub-system.

Monitors and alerts

Monitoring and alerting services are critical in cloud environments with high demands on storage resources. These services provide a real-time view into the health and performance of the storage systems. An integrated management console, or other dashboards that visualize SNMP data, helps to discover and resolve issues with the storage cluster.
A storage-focused cloud design should include:

- Monitoring of physical hardware and environmental resources, such as temperature and humidity.
- Monitoring of storage resources, such as available storage, memory, and CPU.
- Monitoring of advanced storage performance data to ensure that storage systems are performing as expected.
- Monitoring of network resources for service disruptions which affect access to storage.
- Centralized log collection and log-analytics capabilities.
- Ticketing system, or integration with a ticketing system, to track issues.
- Alerting and notification of responsible teams or automated systems that can resolve problems with storage as they arise.
- Network Operations Center (NOC) staffed and always available to resolve issues.

Scaling

A storage-focused OpenStack architecture should focus on scaling up instead of scaling out. You should determine whether a smaller number of larger hosts or a larger number of smaller hosts based on factors such as cost, power, cooling, physical rack and floor space, support-warranty, and manageability.

4.5. NETWORK-FOCUSED ARCHITECTURES

4.5.1. Network-Focused Architecture Types

All OpenStack deployments depend on network communication to function properly because of their service-based nature. However, in some cases the network configuration is more critical and requires additional design considerations.

The following table describes common network-focused architectures. These architectures depend on a reliable network infrastructure and on services that satisfy user and application requirements.

Architecture	Description
Big data	Clouds used for the management and collection of big data create significant demand on network resources. Big data often uses partial replicas of the data to maintain integrity over large distributed clouds. Big data applications that require a large amount of network resources include Hadoop, Cassandra, NuoDB, Riak, or other NoSQL and distributed databases.
Content delivery network (CDN)	CDNs can be used to stream video, view photographs, host web conferences, or access any distributed cloud-based data repository by a large number of end-users. Network configuration affects latency, bandwidth, and distribution of instances. Other factors that affect content deliver and performance include network throughput of back-end systems, resource locations, WAN architecture, and cache methodology.
High availability (HA)	HA environments are dependent on network sizing that maintains replication of data between sites. If one site becomes unavailable, additional sites can serve the increased load until the original site returns to service. It is important to size network capacity to handle the additional loads.
High performance computing (HPC)	HPC environments require additional consideration of traffic flows and usage patterns to address the needs of cloud clusters. HPC has high east-west traffic patterns for distributed computing within the network, but can also have substantial north-south traffic in and out of the network, depending on the application.
High-speed or high-volume transactional systems	These application types are sensitive to network jitter and latency. Example environments include financial systems, credit card transaction applications, and trading systems. These architectures must balance a high volume of east-west traffic with north-south traffic to maximize data delivery efficiency. Many of these systems must access large, high-performance database back-ends.
Network management functions	Environments that support delivery of back-end network services such as DNS, NTP, or SNMP. You can use these services for internal network management.
Network service offerings	Environments that run customer-facing network tools to support services. Examples include VPNs, MPLS private networks, and GRE tunnels.
Virtual desktop infrastructure (VDI)	VDI systems are sensitive to network congestion, latency, and jitter. VDI requires upstream and downstream traffic, and cannot rely on caching to deliver the application to the end-user.

Architecture	Description
Voice over IP (VoIP)	VoIP systems are sensitive to network congestion, latency, and jitter. VoIP system have symmetrical traffic patterns and require network quality of service (QoS) for best performance. In addition, you can implement active queue management to deliver voice and multimedia content. Users are sensitive to latency and jitter fluctuations and can detect them at very low levels.
Video or web conference	Conferencing systems are sensitive to network congestion, latency, and jitter. Video conferencing systems have symmetrical traffic pattern, but if the network is not hosted on an MPLS private network, the system cannot use network quality of service (QoS) to improve performance. Similar to VoIP, users of these system notice network performance problems even at low levels.
Web portals or services	Web servers are common applications in cloud services, and require an understanding of the network requirements. The network must scale out to meet user demand and to deliver web pages with minimum latency. Depending on the details of the portal architecture, you should consider the internal east-west and north-south network bandwidth when you plan the architecture.

4.5.2. Cloud Storage and Backup Architecture

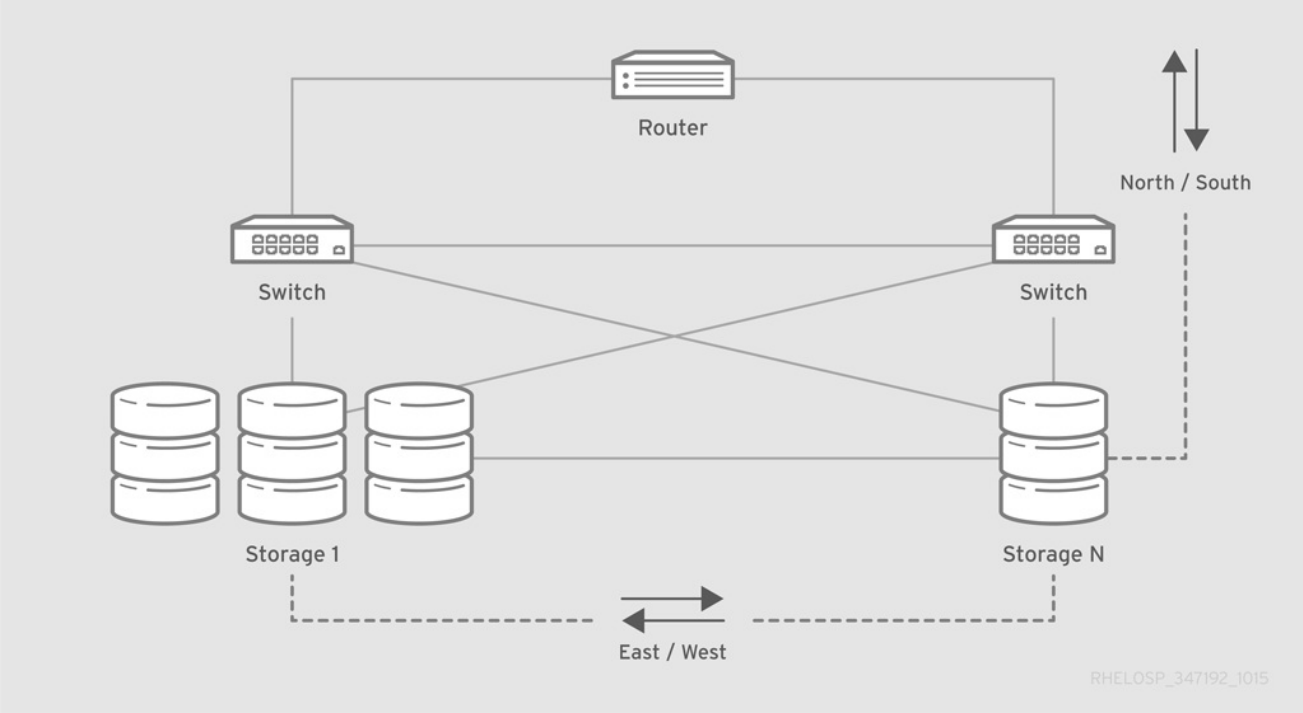
This architecture is for a cloud that provides file storage and file-sharing. You might consider this a storage-focused use case, but the network-side requirements make it a network-focused use case.

NOTE

For installation and deployment documentation, see Chapter 5, *Deployment Information*.

4.5.2.1. About the Design

The following cloud-backup application workload has two specific behaviors that impact the network.



Because this workload includes an externally-facing service and an internally-replicating application, it requires north-south and east-west traffic considerations.

North-south traffic

North-south traffic consists of data that moves in and out of the cloud. When a user uploads and stores content, that content moves southbound into the OpenStack environment. When users download content, that content moves northbound out of the OpenStack environment.

Because this service operates primarily as a backup service, most of the traffic moves southbound into the environment. In this situation, you should configure a network to be asymmetrically downstream, because the traffic that enters the OpenStack environment is greater than the traffic that leaves the environment.

East-west traffic

East-west traffic consists of data that moves inside the environment. This traffic is likely to be fully symmetric, because replication originates from any node and might target multiple nodes algorithmically. However, this traffic might interfere with north-south traffic.

4.5.2.2. Architecture Components

Component	Description
Compute	Compute management and scheduling services run on the controller. The Compute service also runs on each compute node.
Dashboard	Web console for OpenStack management.
Identity	Basic authentication and authorization functionality.
Image	Stores images to be used for booting instances and managing snapshots. This service runs on the controller and offers a small set of images.

Component	Description
Networking	Networking services. For more information about OpenStack Networking, see Chapter 2, *Networking In-Depth*.
Object Storage	Stores backup content.
Telemetry	Monitoring and reporting for other OpenStack services.

4.5.2.3. Design Considerations

In addition to basic design considerations described in Chapter 3, *Design*, you should also follow the considerations described in Section 4.5.4, "Network-Focused Architecture Considerations".

4.5.3. Large-Scale Web-Application Architecture

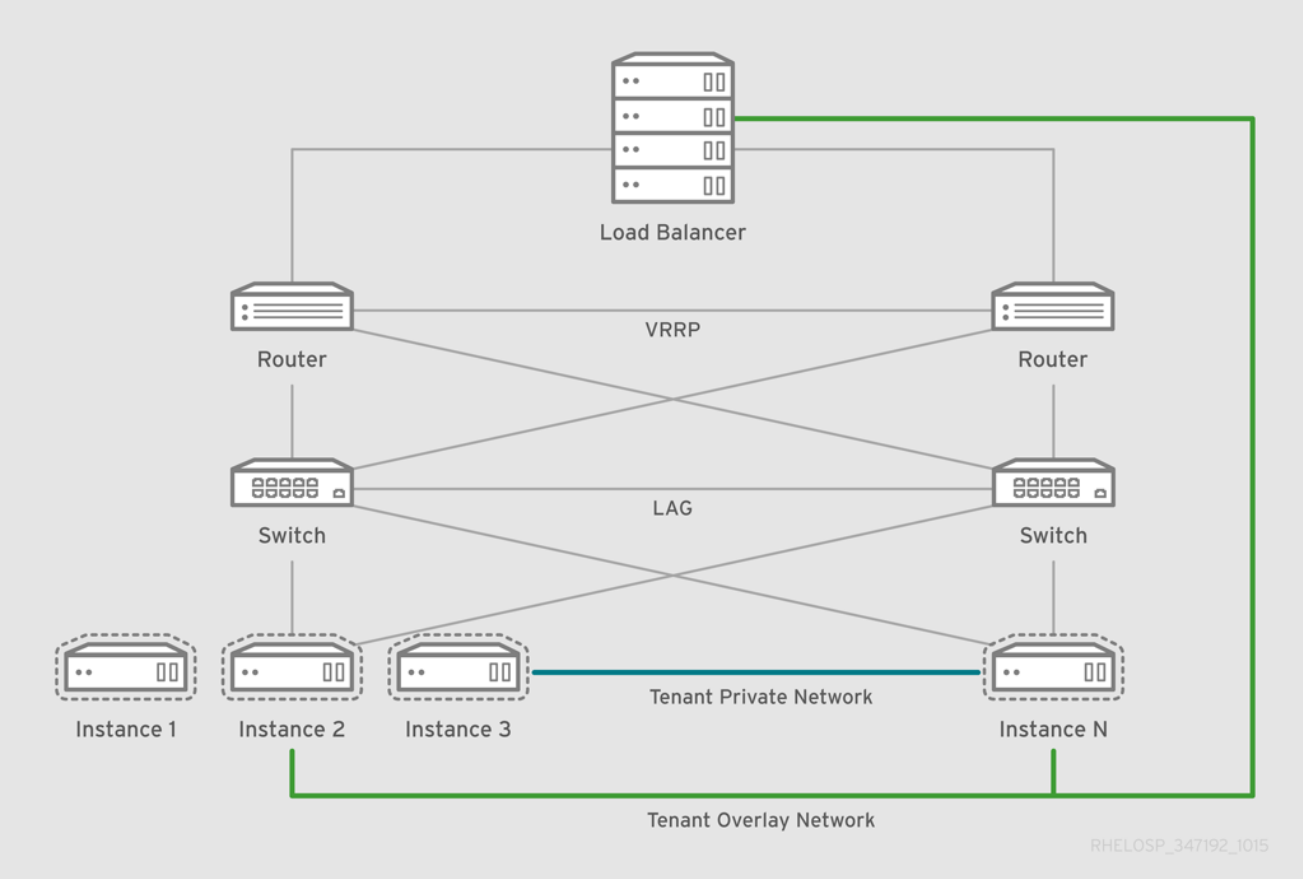
This architecture is for a large-scale web application that scales horizontally in bursts and generates a high instance count. The application requires an SSL connection to secure data and must not lose connection to individual servers.

NOTE

For installation and deployment documentation, see Chapter 5, *Deployment Information*.

4.5.3.1. About the Design

The following diagram shows an example design for this workload.



This design includes the following components and workflows:

- Hardware load balancer provides SSL offload functionality and connects to tenant networks to reduce address consumption.
- The load balancer links to the routing architecture while it services the virtual IP (VIP) for the application.
- The router and the load balancer use the GRE tunnel ID of the application tenant network, and an IP address that is located in the tenant subnet but outside of the address pool. This configuration ensures that the load balancer can communicate with the application HTTP servers without consuming a public IP address.

4.5.3.2. Architecture Components

A web service architecture can consist of many options and optional components. Therefore, this architecture can be used in multiple OpenStack designs. However, some key components must be deployed to handle most web-scale workloads.

Component	Description
Compute	Compute management and scheduling services run on the controller. The Compute service also runs on each compute node.
Dashboard	Web console for OpenStack management.
Identity	Basic authentication and authorization functionality.

Component	Description
Image	Stores images to be used for booting instances and managing snapshots. This service runs on the controller and offers a small set of images.
Networking	Networking services. A split network configuration is compatible with databases that reside on private tenant networks, because the databases do not emit a large quantity of broadcast traffic and might need to interconnect to other databases for content.
Orchestration	Manages instance templates to use when scaling out and during traffic bursts.
Telemetry	Monitoring and reporting for other OpenStack services. Use this service to monitor instance usage and invoke instance templates from the Orchestration service.
Object Storage	Stores backup content.

4.5.3.3. Design Considerations

In addition to basic design considerations described in Chapter 3, *Design*, you should also follow the considerations described in Section 4.5.4, "Network-Focused Architecture Considerations".

4.5.4. Network-Focused Architecture Considerations

In addition to basic design considerations described in Chapter 3, *Design* and to network node design described in Chapter 2, *Networking In-Depth*, the following items should be considered for a network-intensive architecture.

External dependencies

Consider using the following external network components:

- Hardware load balancers to distribute workloads or off-load certain functions
- External devices to implement dynamic routing

Although OpenStack Networking provides a tunneling feature, it is restricted to networking-managed regions. To extend a tunnel beyond the OpenStack regions to another region or to an external system, implement the tunnel outside OpenStack or use a tunnel-management system to map the tunnel or the overlay to an external tunnel.

Maximum transmission unit (MTU)

Some workloads require a larger MTU due to the transfer of large blocks of data. When providing network service for applications such as video streaming or storage replication, configure the OpenStack hardware nodes and the supporting network equipment for jumbo frames wherever possible. This configuration maximizes available bandwidth usage.
Configure jumbo frames across the entire path that the packets traverse. If one network component cannot handle jumbo frames, the entire path reverts to the default MTU.

NAT usage

If you need floating IPs instead of fixed public IPs, you must use NAT. For example, use a DHCP relay mapped to the DHCP server IP. In this case, it is easier to automate the infrastructure to apply the

target IP to a new instance, instead of reconfiguring legacy or external systems for each new instance.
The NAT for floating IPs that is managed by OpenStack Networking resides in the hypervisor. However, other versions of NAT might be running elsewhere. If there is a shortage of IPv4 addresses, you can use the following methods to mitigate the shortage outside of OpenStack:

- Run a load balancer in OpenStack as an instance or externally as a service. The OpenStack Load-Balancer-as-a-Service (LBaaS) can manage load balancing software such as HAproxy internally. This service manages the Virtual IP (VIP) addresses while a dual-homed connection from the HAproxy instance connects the public network with the tenant private network that hosts all content servers.

- Use a load balancer to serve the VIP and also connect to the tenant overlay network with external methods or private addresses.

In some cases it may be desirable to use only IPv6 addresses on instances and operate either an instance or an external service to provide a NAT-based transition technology such as NAT64 and DNS64. This configuration provides a globally-routable IPv6 address, while consuming IPv4 addresses only as necessary.

Quality of Service (QoS)

QoS impacts network-intensive workloads because it provides instant service to packets with high priority because of poor network performance. In applications such as Voice over IP (VoIP), differentiated service code points are usually required for continued operation.
You can also use QoS for mixed workloads to prevent low-priority, high-bandwidth applications such as backup services, video conferencing, or file sharing, from blocking bandwidth that is needed for the continued operation of other workloads.

You can tag file-storage traffic as lower class traffic, such as best effort or scavenger, to allow higher-priority traffic to move through the network. In cases where regions in a cloud are geographically distributed, you might also use WAN optimization to reduce latency or packet loss.

Workloads

The routing and switching architecture should accommodate workdloads that require network-level redundancy. The configuration depends on your selected network hardware, on the selected hardware performance, and on your networking model. Examples include Link Aggregation (LAG) and Hot Standby Router Protocol (HSRP).
The workload also impacta the effectiveness of overlay networks. If application network connections are small, short lived, or bursty, running a dynamic overlay can generate as much bandwidth as the packets the network carries. Overlay can also induce enough latency to cause issues with the hypervisor, which causes performance degradation on packet-per-second and connection-per-second rates.

By default, overlays include a secondary full-mesh option that depends on the workload. For example, most web services applications do not have major issues with a full-mesh overlay network, and some network monitoring tools or storage replication workloads have performance issues with throughput or excessive broadcast traffic.

CHAPTER 5. DEPLOYMENT INFORMATION

The following table includes deployment references for components mentioned in this guide.

Additional manuals for Red Hat OpenStack Platform can be found here: Red Hat OpenStack Platform Documentation Suite.

Component	Reference
Red Hat Enterprise Linux	Red Hat OpenStack Platform is supported on Red Hat Enterprise Linux 7.3. For information on installing Red Hat Enterprise Linux, see the corresponding installation guide at: Red Hat Enterprise Linux Documentation Suite
OpenStack	To install OpenStack components and their dependencies, use the Red Hat OpenStack Platform director. The director uses a basic OpenStack *undercloud*, which is then used to provision and manage the OpenStack nodes in the final *overcloud*. Be aware that you will need one extra host machine for the installation of the undercloud, in addition to the environment necessary for the deployed overcloud. For detailed instructions, see Red Hat OpenStack Platform Director Installation and Usage.
High Availability	For the configuration of additional high availability components (for example, HAProxy), see Deploying Highly Available Red Hat OpenStack Platform 6 with Ceph Storage. For information about configuring live migration, see *Managing Instances and Images*.
LBasS	To use Load Balancing-as-a-Service, see Configuring Load Balancing-as-a-Service in the *Networking Guide*.
Pacemaker	Pacemaker is integrated into Red Hat Enterprise Linux as an add-on. To set up Red Hat Enterprise Linux for high availability, see the High Availability Add-on Overview.