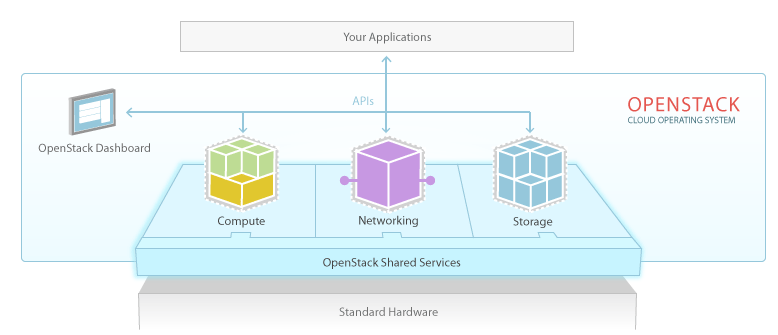
1. **INTRODUCTION**

OpenStack is a set of software tools for building and managing cloud computing platforms for public and private clouds. Backed by some of the biggest companies in software development and hosting, as well as thousands of individual community members, many think that OpenStack is the future of cloud computing. OpenStack is managed by the OpenStack Foundation, a non-profit that oversees both development and community-building around the project.

OpenStack lets users deploy virtual machines and other instances that handle different tasks for managing a cloud environment on the fly. It makes horizontal scaling easy, which means that tasks that benefit from running concurrently can easily serve more or fewer users on the fly by just spinning up more instances. For example, a mobile application that needs to communicate with a remote server might be able to divide the work of communicating with each user across many different instances, all communicating with one another but scaling quickly and easily as the application gains more users.

And most importantly, OpenStack is open source software, which means that anyone who chooses to can access the source code, make any changes or modifications they need, and freely share these changes back out to the community at large. It also means that OpenStack has the benefit of thousands of developers all over the world working in tandem to develop the strongest, most robust, and most secure product that they can.

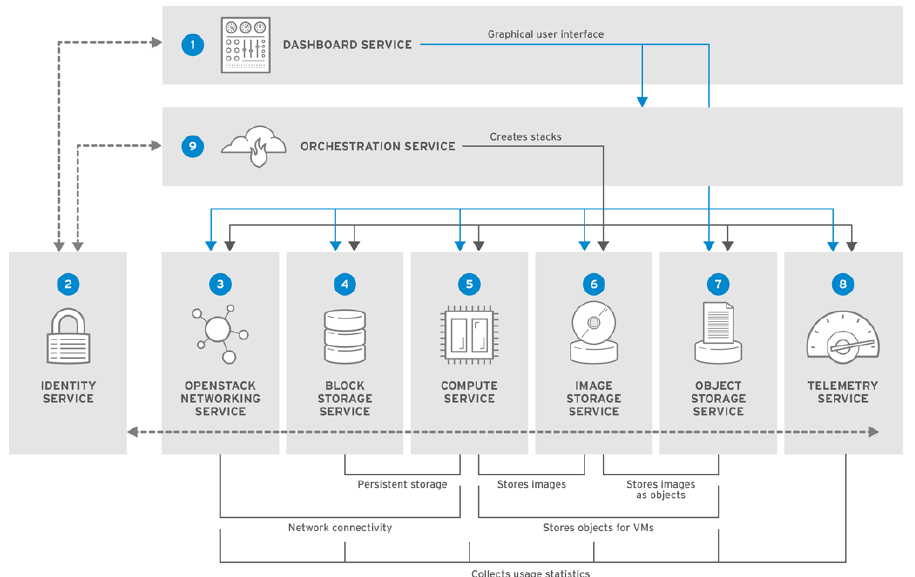
****

**Fig.1. 1 Architecture of OpenStack**

The cloud is all about providing computing for end users in a remote environment, where the actual software runs as a service on reliable and scalable servers rather than on each end-user's computer. Cloud computing can refer to a lot of different things, but typically the industry talks about running different items "as a service"—software, platforms, and infrastructure. OpenStack falls into the latter category and is considered Infrastructure as a Service (IaaS). Providing infrastructure means that OpenStack makes it easy for users to quickly add new instance, upon which other cloud components can run. Typically, the infrastructure then runs a "platform" upon which a developer can create software applications that are delivered to the end users.

1. **COMPONENTS OF OPENSTACK**

The Red Hat Enterprise Linux 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.

****

**Fig.2.1 OpenStack core services and their relationship with each other.**

THE FOLLOWING ARE THE COMPONENTS OF OPENSTACK :-

* **Nova** is the primary computing engine behind OpenStack. It is used for deploying and managing large numbers of virtual machines and other instances to handle computing tasks.
* **Swift** is a storage system for objects and files. Rather than the traditional idea of a referring to files by their location on a disk drive, developers can instead refer to a unique identifier referring to the file or piece of information and let OpenStack decide where to store this information. This makes scaling easy, as developers don’t have the worry about the capacity on a single system behind the software. It also allows the system, rather than the developer, to worry about how best to make sure that data is backed up in case of the failure of a machine or network connection.
* **Cinder** is a block storage component, which is more analogous to the traditional notion of a computer being able to access specific locations on a disk drive. This more traditional way of accessing files might be important in scenarios in which data access speed is the most important consideration.
* **Neutron** provides the networking capability for OpenStack. It helps to ensure that each of the components of an OpenStack deployment can communicate with one another quickly and efficiently.
* **Horizon** is the dashboard behind OpenStack. It is the only graphical interface to OpenStack, so for users wanting to give OpenStack a try, this may be the first component they actually “see.” Developers can access all of the components of OpenStack individually through an application programming interface (API), but the dashboard provides system administrators a look at what is going on in the cloud, and to manage it as needed.
* **Keystone** provides identity services for OpenStack. It is essentially a central list of all of the users of the OpenStack cloud, mapped against all of the services provided by the cloud, which they have permission to use. It provides multiple means of access, meaning developers can easily map their existing user access methods against Keystone.
* **Glance** provides image services to OpenStack. In this case, "images" refers to images (or virtual copies) of hard disks. Glance allows these images to be used as templates when deploying new virtual machine instances.
* **Ceilometer** provides telemetry services, which allow the cloud to provide billing services to individual users of the cloud. It also keeps a verifiable count of each user’s system usage of each of the various components of an OpenStack cloud. Think metering and usage reporting.
* **Heat** is the orchestration component of OpenStack, which allows developers to store the requirements of a cloud application in a file that defines what resources are necessary for that application. In this way, it helps to manage the infrastructure needed for a cloud service to run.

1. **NETWORKING**
   1. **OpenStack Networking (neutron)**

**Neutron** provides the networking capability for OpenStack. It helps to ensure that each of the components of an OpenStack deployment can communicate with one another quickly and efficiently.

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

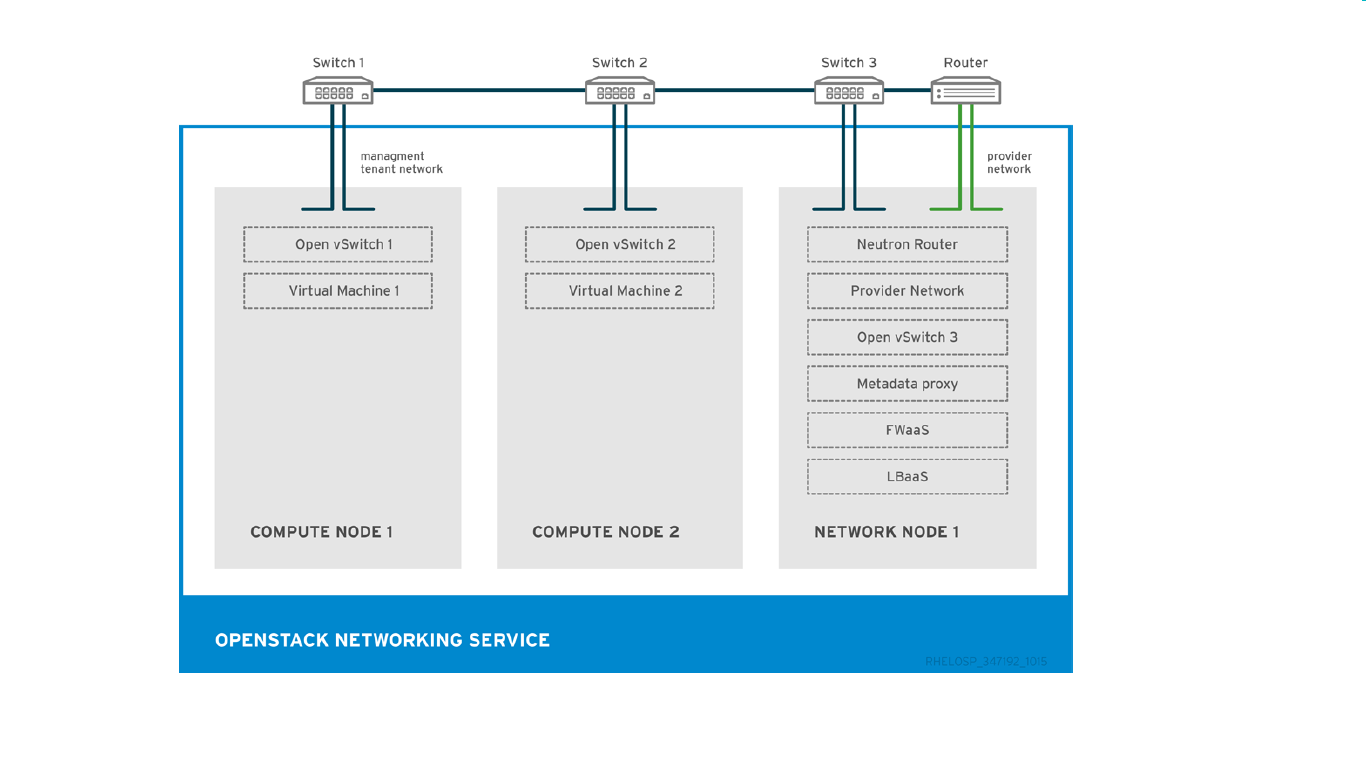
header, which will essentially allow around 16 million unique addresses/networks.

**Table 3.1. 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.



**Fig.3.1. OpenStack Networking node and tenant networks.**

The example shows the following Networking service configuration:

1. 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

1. The compute nodes have two physical network cards each. One card handles tenant traffic, and the other card manages connectivity.
2. The OpenStack Networking node has a third network card dedicated to provider traffic.
3. **STORAGE**
   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.

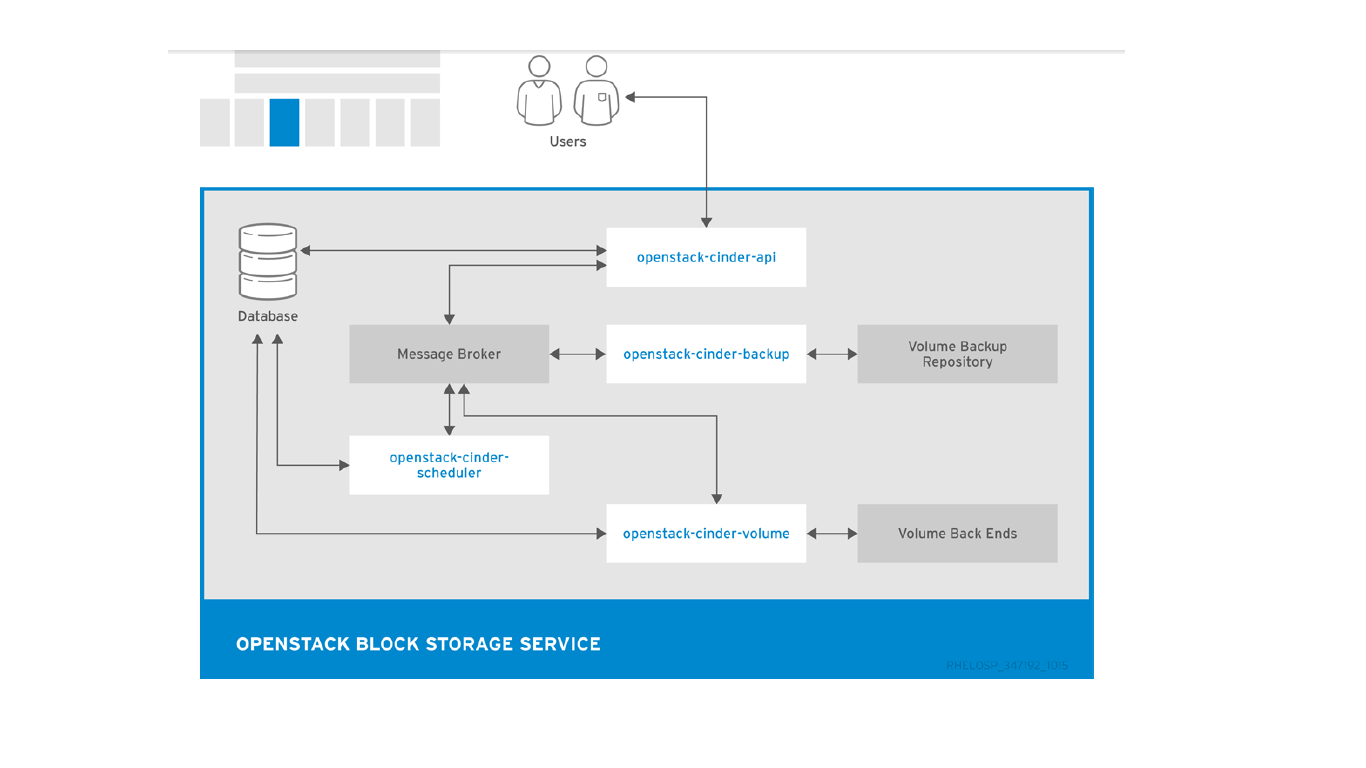
OpenStack Block Storage advantages include:

* 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 4.1.. 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, NFS, or GlusterFS 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-cindervolume  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. |



**Fig.4.1 Relationship between the Block Storage API, the scheduler, the volume services and other OpenStack components.**

**4.2.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.

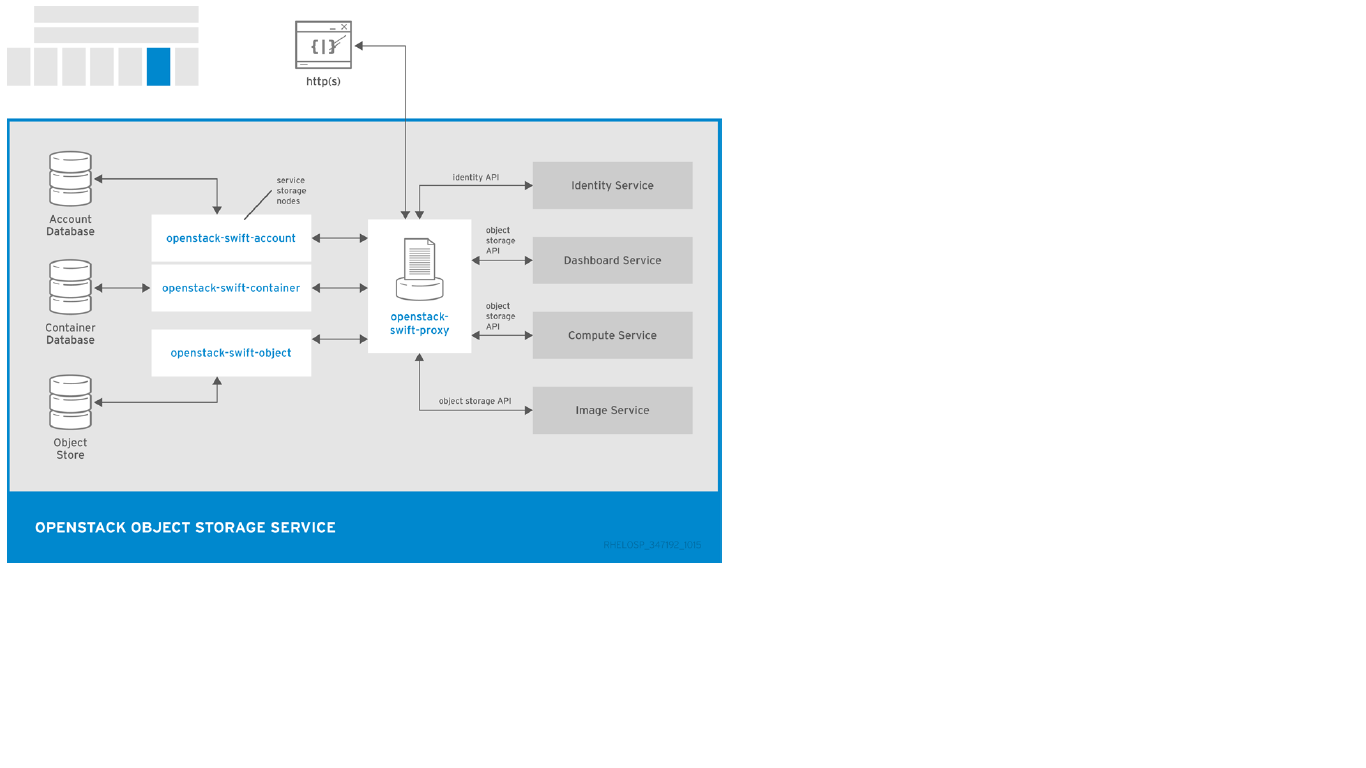
The Object Storage service relies on other OpenStack services and components to perform actions. For example, the Identity Service (keystone), the rsync daemon, and a load balancer are all required.

**Table 4.2.1 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 4.2.2 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. |

****

**Fig 4.2. The Object Storage uses**

**4.3 OpenStack Database-as-a-Service (trove)**

OpenStack Database-as-a-Service allows users to select, provision, and operate a variety of

relational and non-relational databases and handles more complex database administration tasks

out-of-the-box.

OpenStack Database-as-a-Service advantages include:

* Users and database administrators can provision and manage multiple database instances in the cloud.
* High-performance resource isolation while automating complex administrative tasks such as deployment, configuration, patching, backup, restore, and monitoring.

**Table 4.3 Trove components**

|  |  |
| --- | --- |
| Component | Description |
| openstack-trove-api | RESTful API that supports JSON and XML to provision and manage Databaseas-a-Service instances. |
| openstack-troveconductor | Runs on the host to receive messages from guest instances with requests to update information on the host. The requests can include the status of an instance or the current status of a backup.  With openstack-trove-conductor, guest instances do not need a direct connection to the host database. The service listens for RPC messages through the message bus and performs the requested operation. |
| openstack-troveguestagent | Runs on the guest instance to manage and perform operations on the host database directly. The openstack-trove-guestagent listens for RPC messages through the message bus and performs the requested operation. |
| openstack-trovetaskmanager | Responsible for tasks such as provisioning instances, managing the lifecycle of instances, and operations on the database instance. |
| trove | Command-line client to access the Database-as-a-Service API. |

1. **VIRTUAL MACHINES, IMAGES, AND TEMPLATES**

**5.1 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.

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 RHEL 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 withmultiple 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 novacells 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.

**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 flavours or images to a subset of hosts.

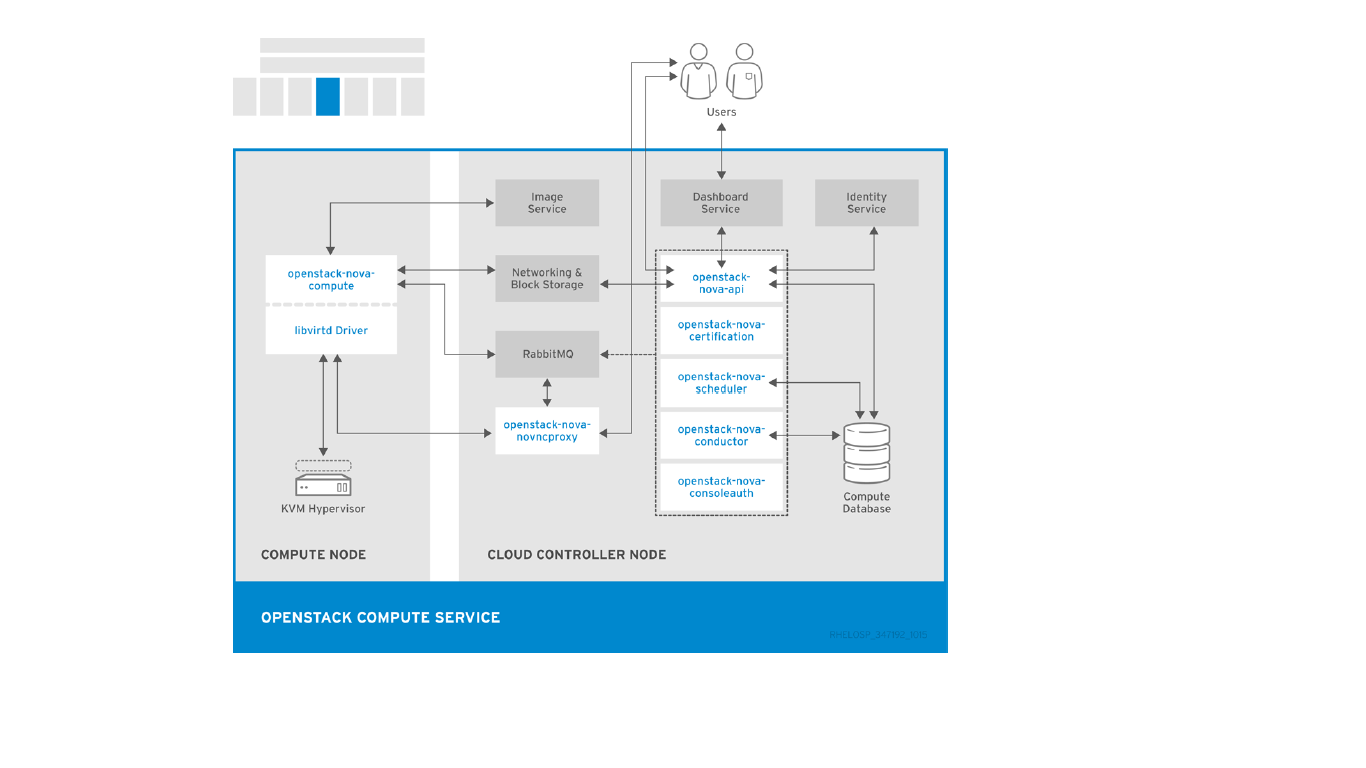
To users, the group is presented as an Availability Zone. The user cannot view the group metadata or see the list of hosts in the zone.

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 5.1.. 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-novacompute | 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-novaconductor | Provides database-access support for compute nodes to reduce security risks. |
| openstack-novaconsoleauth | Handles console authentication. |
| openstack-novanetwork | Network services that can serve as an alternative to OpenStack Networking and handle basic network traffic for private and public access. |
| openstack-novanovncproxy | Provides a VNC proxy for browsers to enable VNC consoles to access virtual machines. |
| openstack-novascheduler | 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. |



**Fig 5.1 The relationship between the Compute services and other OpenStack components.**

**5.2. 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.

**Table 5.2. Bare Metal Provisioning components**

|  |  |
| --- | --- |
| Component | Description |
| openstack-ironic-api | Handles requests and provides access to Compute resources on the bare metal node |
| openstack-ironicconductor | 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. |

**5.3. 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)

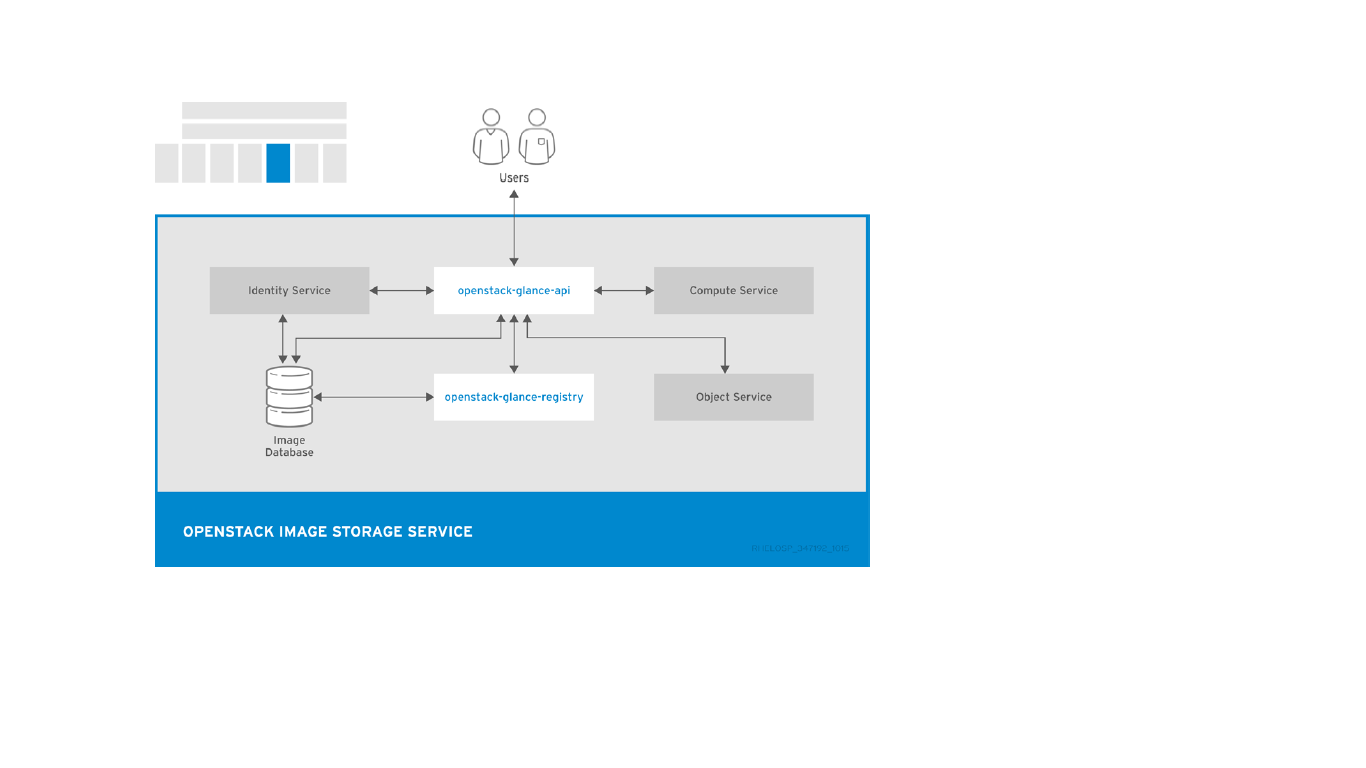
Container formats can also be registered by the Image service. The container format determines the type and detail level of the virtual machine metadata to store in the image.

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 5.3. 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. |



**Fig.5.3. The Image service uses to register and retrieve images from the Image database**

**5.4. OpenStack Orchestration (heat)**

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

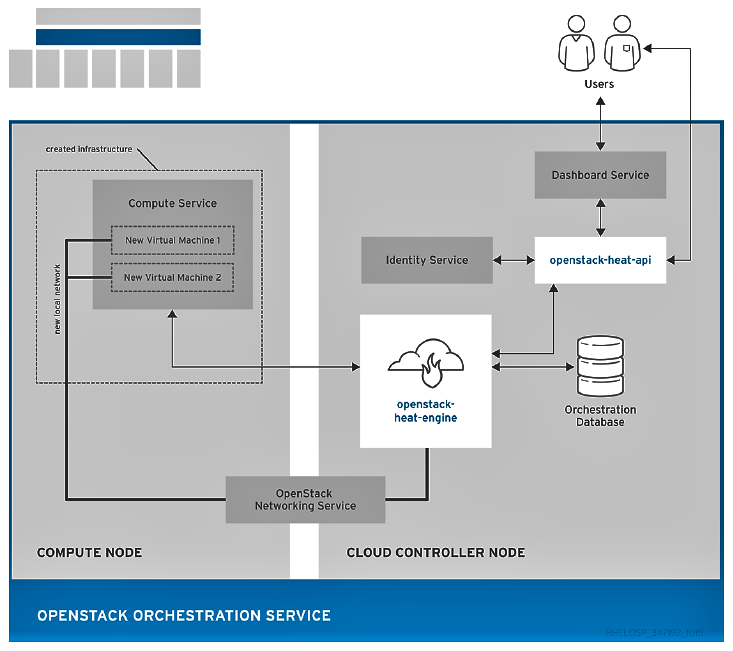
For example, you can create templates for instances, floating IPs, volumes, security groups, or users. Orchestration offers access to all OpenStack core services with a single modular template, as well as capabilities such as auto-scaling and basic high availability.

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 5.4.. 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-heatengine 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. |



**Fig 5.4 The Orchestration service uses to create a new stack of two new instances and a local network**.

**5.5.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.

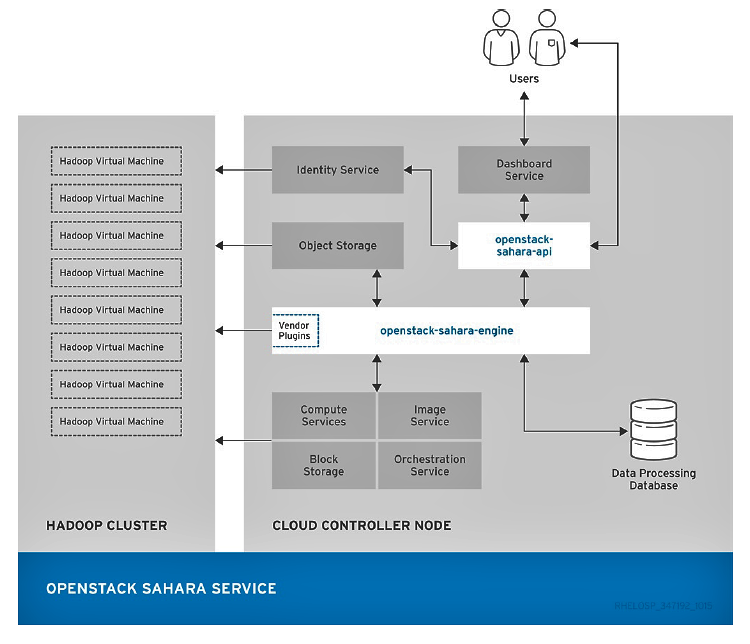
The Hadoop compute and storage servers are co-located, which enables high-speed analysis of stored data. All tasks are divided across the servers and utilizes the local server resources.

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 Cloudera (CDH) and Hortonworks Data Platform (HDP) distributions 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.

|  |  |
| --- | --- |
| 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-saharaengine | Provisioning engine that handles cluster requests and data delivery. |
| sahara | Command-line client to access the Data Processing API. |

**Table 5.5. Sahara components**

**Fig.5.5 OpenStack Sahara services.**

**6. IDENTITY MANAGEMENT**

**6.1. 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.

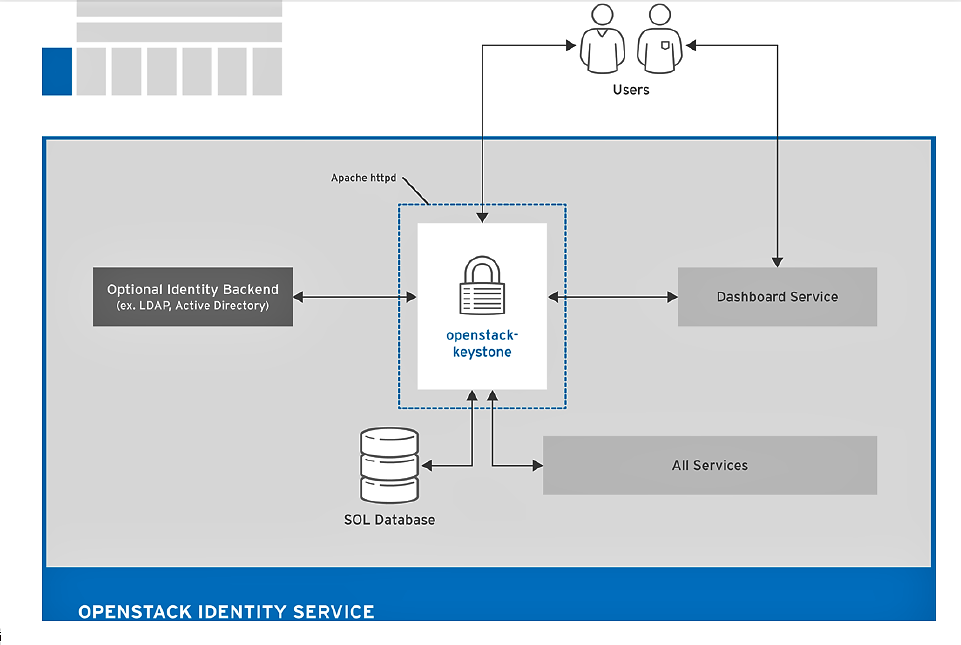
Identity supports Federation with SAML. Federated Identity establishes trust between Identity Providers (IdP) and the services that Identity provides to the end user.

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 6.1.. 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. |



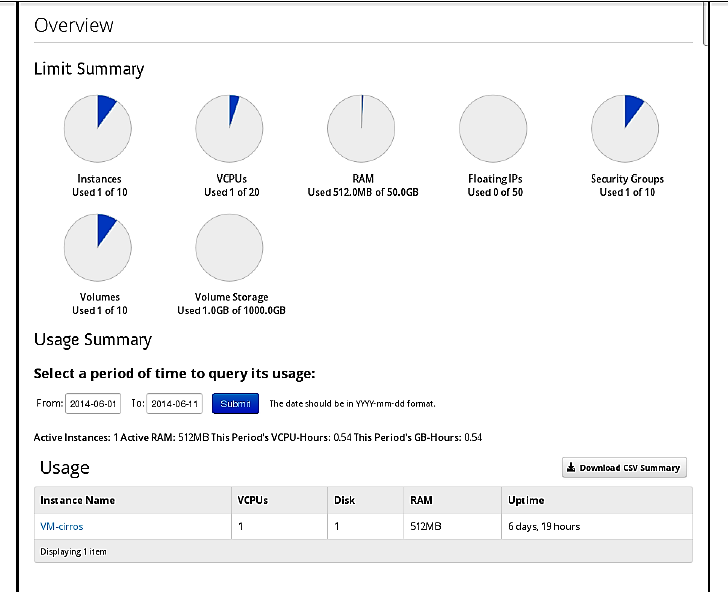
**Fig . 6.1.The Identity uses to authenticate users with other OpenStack components.**

**7.USER INTERFACES**

**7.1. 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.

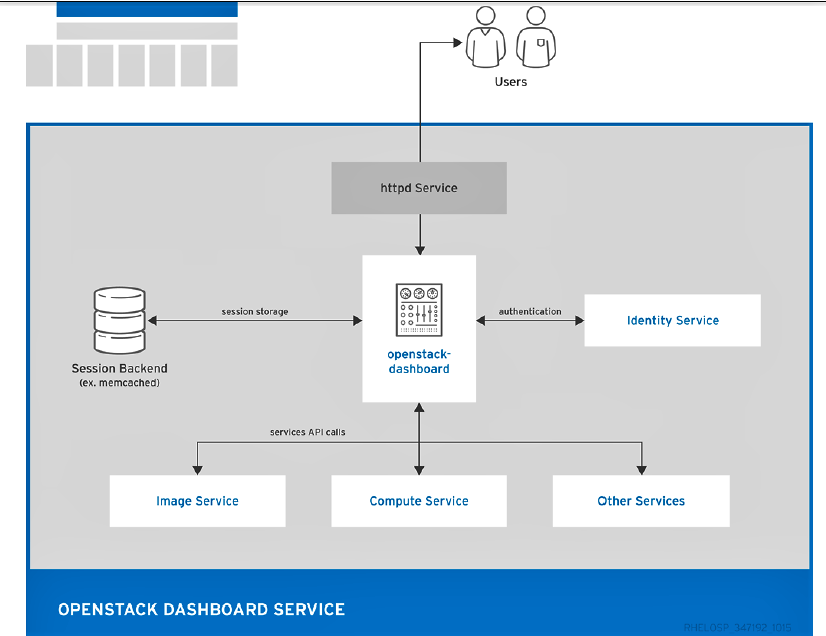
The Dashboard service provides the Project, Admin, and Settings default dashboards. The modular design enables the dashboard to interface with other products such as billing, monitoring, and additional management tools.



**Fig. 7.1.1 An example of the Compute panel in the Admin dashboard.**

**Table 7.1 Dashboard components**

|  |  |
| --- | --- |
| Component | Description |
| openstack-dashboard | Django Web application that provides access to the dashboard from any Web browser. |
| Apache HTTP server (httpd service) | Hosts the application. |



**Fig.7.1.2.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.

**7.2 OpenStack Telemetry (ceilometer)**

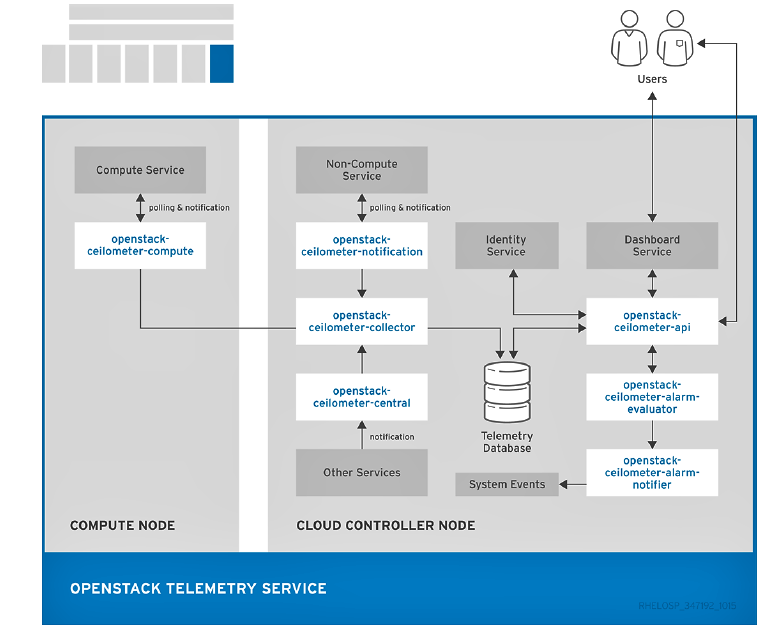
OpenStack Telemetry provides user-level usage data for OpenStack-based clouds. The data can be used for customer billing, system monitoring, or alerts. Telemetry can collect data from notifications sent by existing OpenStack components such as Compute usage events, or by polling OpenStack infrastructure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

**Table 7.2. Telemetry components**

|  |  |
| --- | --- |
| Component | Description |
| openstack-ceilometer-alarmevaluator | Triggers state transitions on alarms. |
| openstack-ceilometer-alarmnotifier | Executes actions when alarms are triggered. |
| 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-ceilometercollector | 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-ceilometercompute | 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-ceilometernotification | Pushes metrics to the collector service from various OpenStack services. |
| ceilometer | Command-line client to access the Telemetry API. |



**Fig.7.2 The interfaces used by the Telemetry service**

**8. Third-party Components**

**8.1 Third-party Components**

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.

**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 RHEL 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.

**9. CONCLUSION**OpenStack software controls large pools of compute, storage, and networking resources throughout a datacenter, managed through a dashboard or via the OpenStack API. OpenStack works with popular enterprise and open source technologies making it ideal for heterogeneous infrastructure.   
  
  
OpenStack is an early project but it is fully open source and based on open cloud standards. Hundreds of the world’s largest brands rely on OpenStack to run their businesses every day, reducing costs and helping them move faster. OpenStack has a strong ecosystem, and users seeking commercial support can choose from different OpenStack-powered products and services in the Marketplace.

**10.BIBLOGRAPHY**  
  
  
  
  
  
https://www.openstack.org  
  
<https://en.wikipedia.org/wiki/OpenStack>  
  
<https://www.redhat.com/en/insights/openstack>  
  
<https://access.redhat.com/documentation/enUS/Red_Hat_Enterprise_Linux_OpenStack_Platform/5/html/Installation_and_Configuration_Guide>  
  
https://access.redhat.com/documentation/en/red-hat-openstack-platform