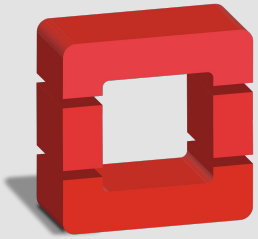
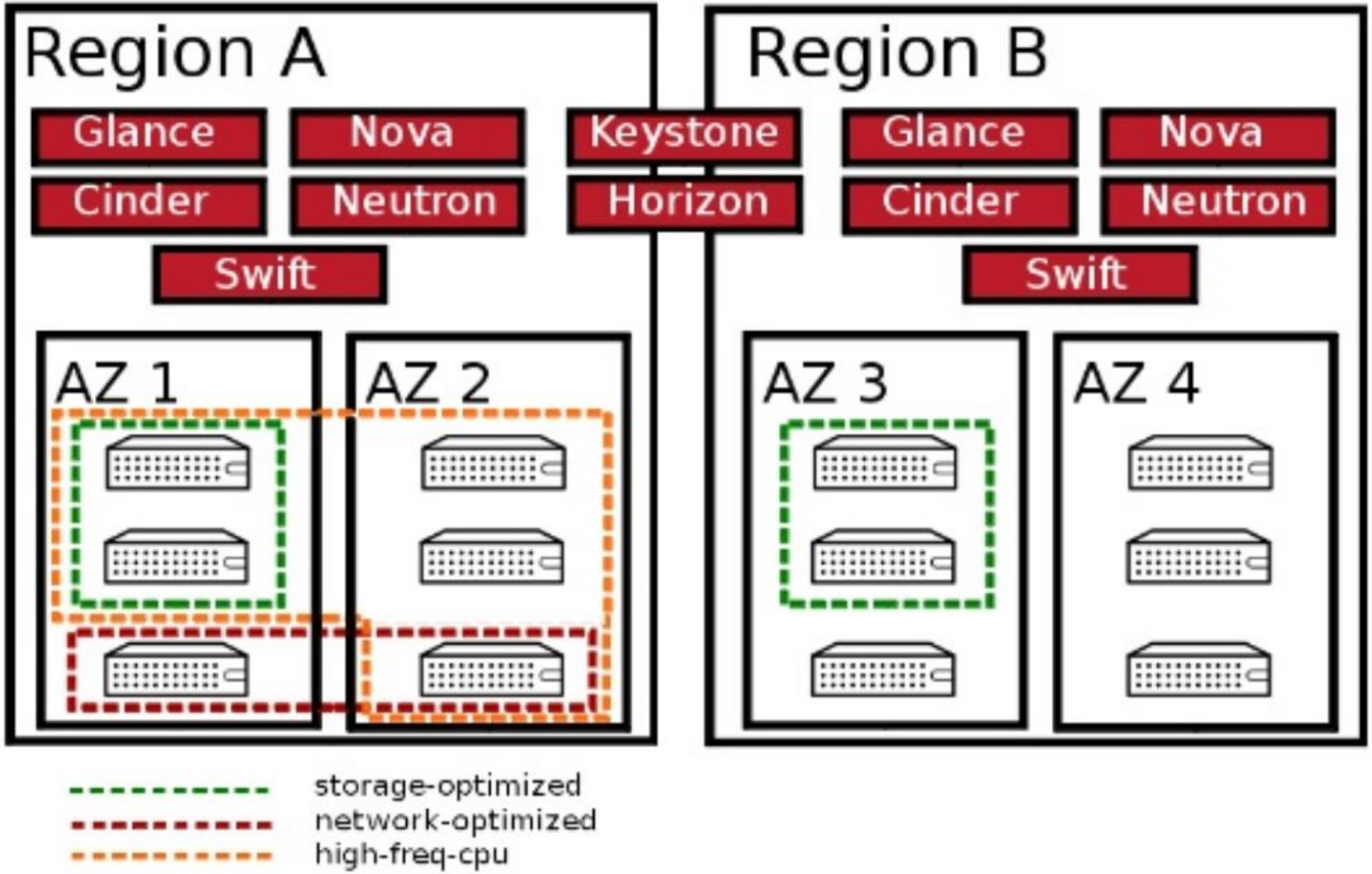
OpenStack distributed deployments and service HA

Openstack Administration 101



openstack
CLOUD SOFTWARE

Stefano Stalio - LNGS


Region A
Glance
Nova
Cinder
Neutron
Swift
Keystone
Horizon
Region B
Glance
Nova
Cinder
Neutron
Swift
AZ 1
AZ 2
AZ 3
AZ 4
storage-optimized
network-optimized
high-freq-cpu

Host aggregates

- Host aggregates are a mechanism for partitioning hosts in an OpenStack cloud, or a region of an OpenStack cloud, based on arbitrary characteristics. Examples where an administrator may want to do this include where a group of hosts have additional hardware or performance characteristics.
- Host aggregates are not explicitly exposed to users. Instead **administrators map flavors to host aggregates**. Administrators do this by setting metadata on a host aggregate, and matching flavor extra specifications. The scheduler then endeavors to match user requests for instance of the given flavor to a host aggregate with the same key-value pair in its metadata. Compute nodes can be in more than one host aggregate.
- Administrators are able to optionally expose a host aggregate as an availability zone. Availability zones are different from host aggregates in that they are explicitly exposed to the user, and hosts can only be in a single availability zone. Administrators can configure a default availability zone where instances will be scheduled when the user fails to specify one.

Availability zones

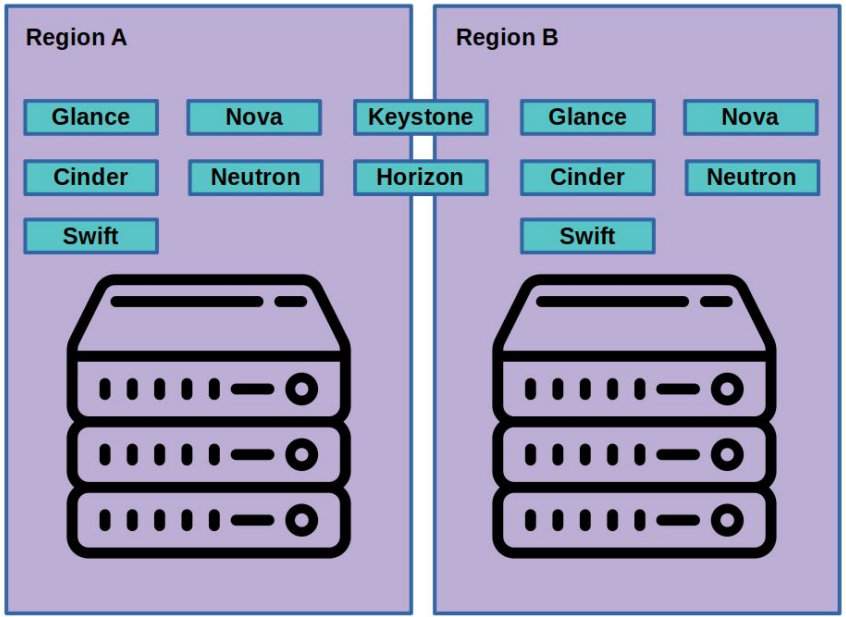
Availability Zones are the end-user visible logical abstraction for partitioning a cloud without knowing the physical infrastructure. That abstraction doesn't come up in Nova with an actual database model since the availability zone is actually a specific metadata information attached to an aggregate. Adding that specific metadata to an aggregate makes the aggregate visible from an end-user perspective and consequently allows to schedule upon a specific set of hosts (the ones belonging to the aggregate).

That said, there are a few rules to know that diverge from an API perspective between aggregates and availability zones:

- one host can be in multiple aggregates, but it can only be in one availability zone
- by default a host is part of a default availability zone even if it doesn't belong to an aggregate (the configuration option is named **default_availability_zone**)

OpenStack Regions

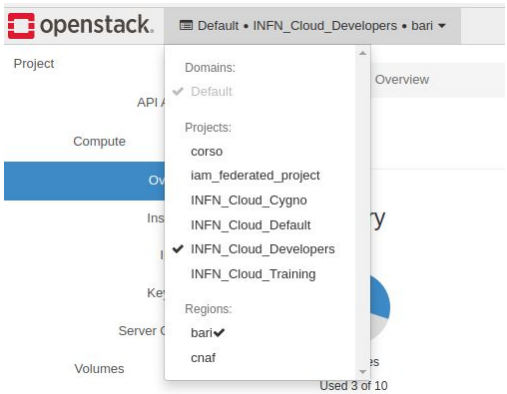
- **Multi region OpenStack deployments are separate Openstack instances**, often running in different data centres hundreds of kilometers apart, **that share a subset of services**, at least Keystone and Horizon.
- This approach is similar to that of AWS, where a user can decide where to deploy its own workloads.
- Besides Keystone and Horizon, other OpenStack services can be shared among multiple regions, whereas some services are not fit for a shared implementation: think how bad would a Cinder volume perform if attached to a VM running hundreds of km and tens of ms apart!!!

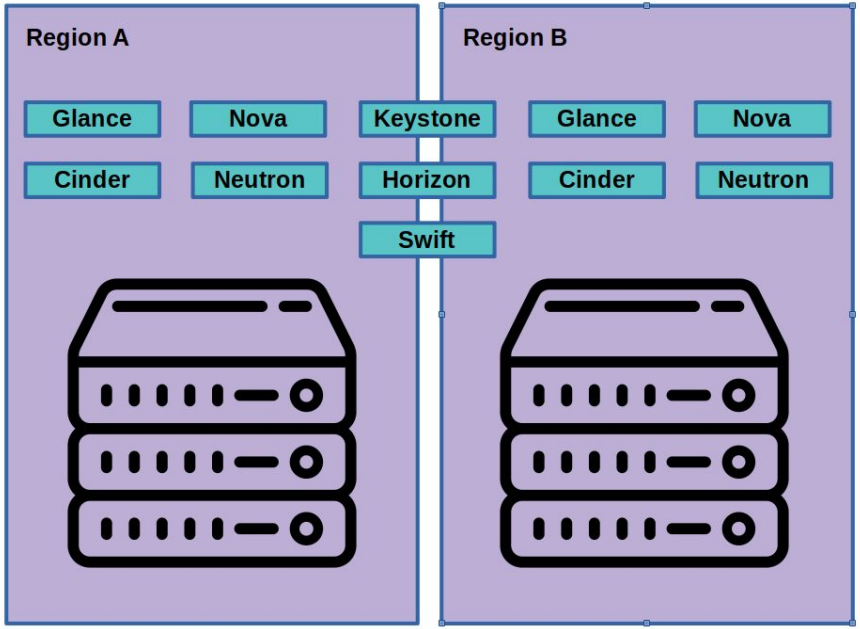


OpenStack Regions

Sharing the Object Storage service in a multi-region OpenStack deployment has interesting advantages:

- Swift can be deployed as a distributed infrastructure
- a distributed Object Storage infrastructure increases user data availability and throughput
    - data is replicated over multiple data centers
    - data is accessed through multiple, independent endpoints

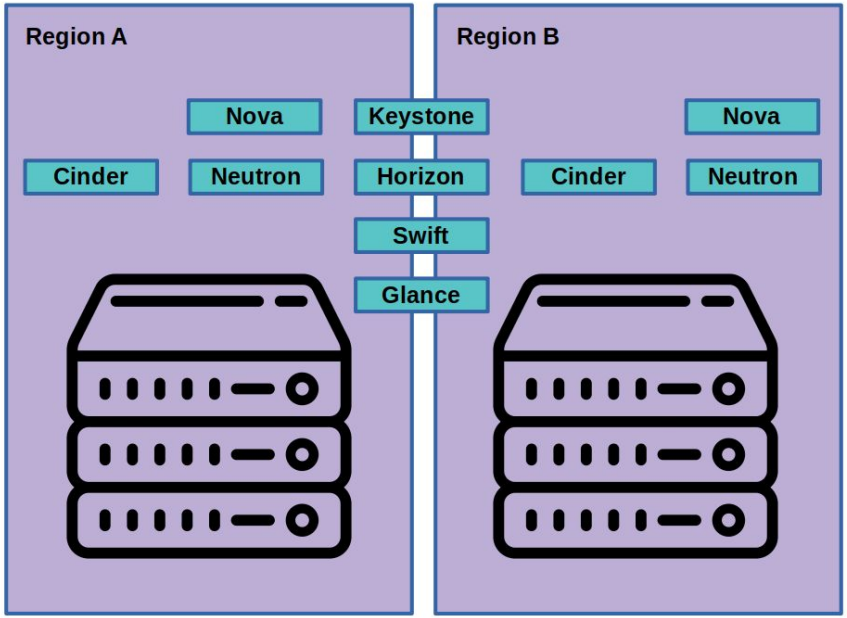




OpenStack Regions

Also **the image service can be shared among two or more different regions**:

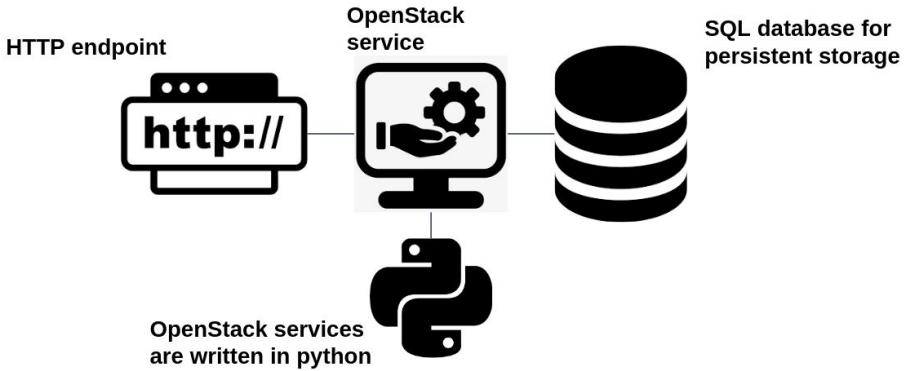
- the same images, both public and private, are available in all data centers
- instance snapshots, that are managed inside glance, are available in all data centers
    - this means that **users have an easy way of cold migrating their VMs for one data center to the other**
    - this can also be an easy way of replicating the same services over different data centers
- if we use Swift as a storage backend for the Glance image service
    - data is replicated over multiple data centers
    - data is accessed through multiple, independent endpoints



Stateless versus stateful services

OpenStack components can be divided into three categories:

- **OpenStack APIs**: APIs that are HTTP(s) stateless services written in python, easy to duplicate and mostly easy to load balance.
- The **SQL relational database** server provides stateful type consumed by other components. Making the SQL database redundant is complex.
- **Advanced Message Queuing Protocol** (AMQP) provides OpenStack internal stateful communication service.



Stateless versus stateful services

Stateless services

A service that provides a response after your request and then requires no further attention. To make a stateless service highly available, you need to provide redundant instances and load balance them.

OpenStack services that are stateless include **nova-api**, **nova-conductor**, **glance-api**, **keystone-api**, **neutron-api**, and **nova-scheduler**.

Stateful services

A service where subsequent requests to the service depend on the results of the first request. Stateful services are more difficult to manage because a single action typically involves more than one request. Providing additional instances and load balancing does not solve the problem. For example, if the horizon user interface reset itself every time you went to a new page, it would not be very useful. **OpenStack services that are stateful include the OpenStack database and message queue**. Making stateful services highly available can depend on whether you choose an active/passive or active/active configuration.

Configuring HA for stateless services

HA for stateless services can be configured using a **load-balancer**, like haproxy or nginx or traefik.

HAProxy provides a fast and reliable HTTP reverse proxy and load balancer for TCP or HTTP applications. It is particularly suited for web crawling under very high loads while needing persistence or Layer 7 processing. It realistically supports tens of thousands of connections with recent hardware.

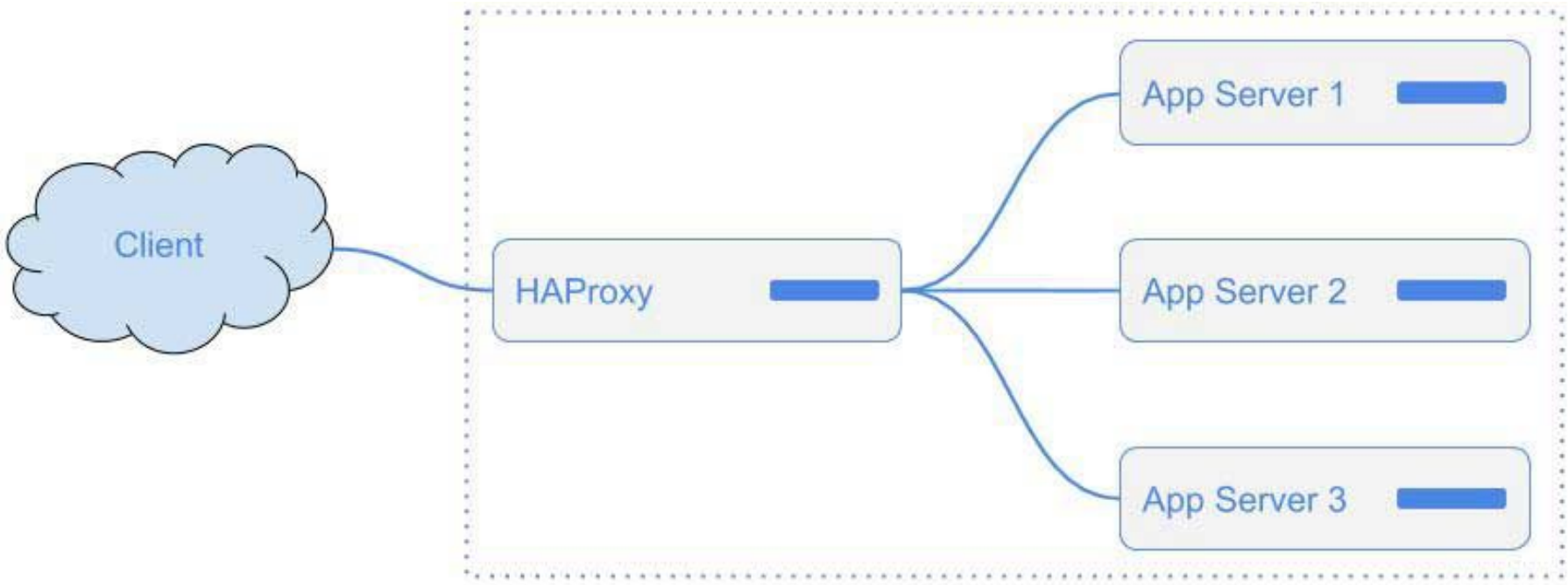
Each instance of HAProxy configures its front end to accept connections only to the virtual IP (VIP) address. The HAProxy back end (termination point) is a list of all the IP addresses of instances for load balancing.

> **Note**
> Ensure your HAProxy installation is not a single point of failure, it is advisable to have multiple HAProxy instances running. You can also ensure the availability by other means, using Keepalived or Pacemaker.

Alternatively, you can use a commercial load balancer, which is hardware or software.

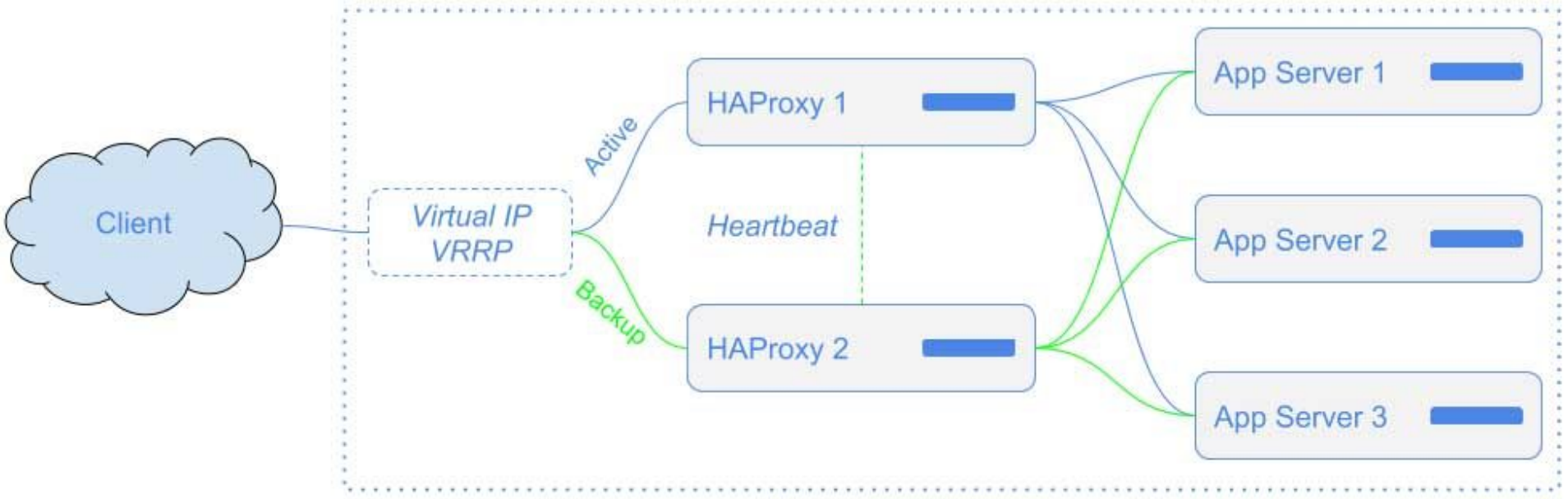
HAProxy deployment

In the HAProxy load balancing setup shown in below diagram the **HAProxy is the single point of failure**, which may cause downtime / service unavailability. To make this a true High Availability setup, we need to **make the HAProxy load balancer redundant** with two or more HAProxy nodes using Keepalived and VRRP.



HAProxy deployment

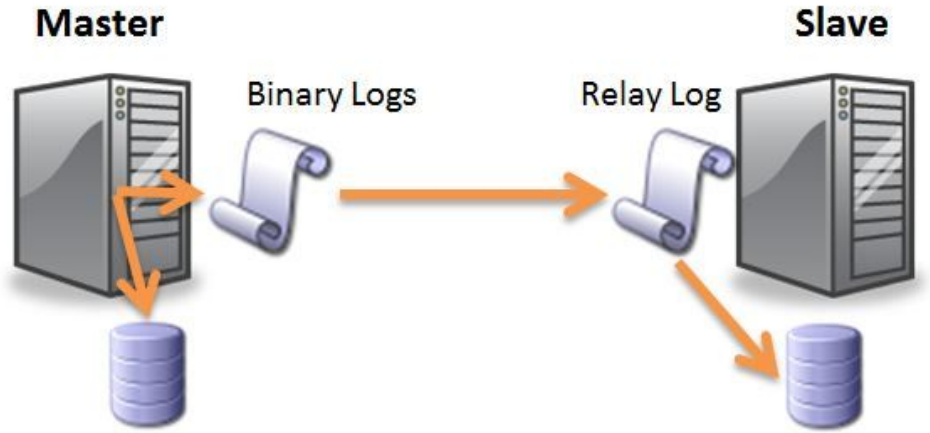
The example shown in the below diagram uses a virtual ip address which is currently assigned to the Active HAProxy node and in case of failure can be reassigned to the HAProxy Backup node.



Configuring HA for databases

Mysql master slave

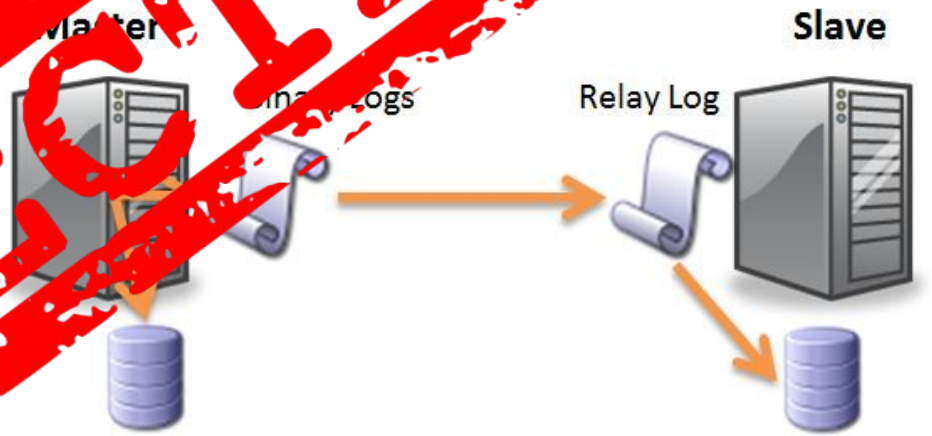
- The master-slave replication process enables database administrators to replicate or copy data stored in more than one server simultaneously.
- This helps the database administrator to create a live backup of the database all the time. During some situations, when the master-slave is down to any issues, they can instantly switch over the slave database and keep the application up and running.
- The replication process ensures that your application doesn't face any kind of downtime at all.



Configuring HA for databases

Mysql master slave

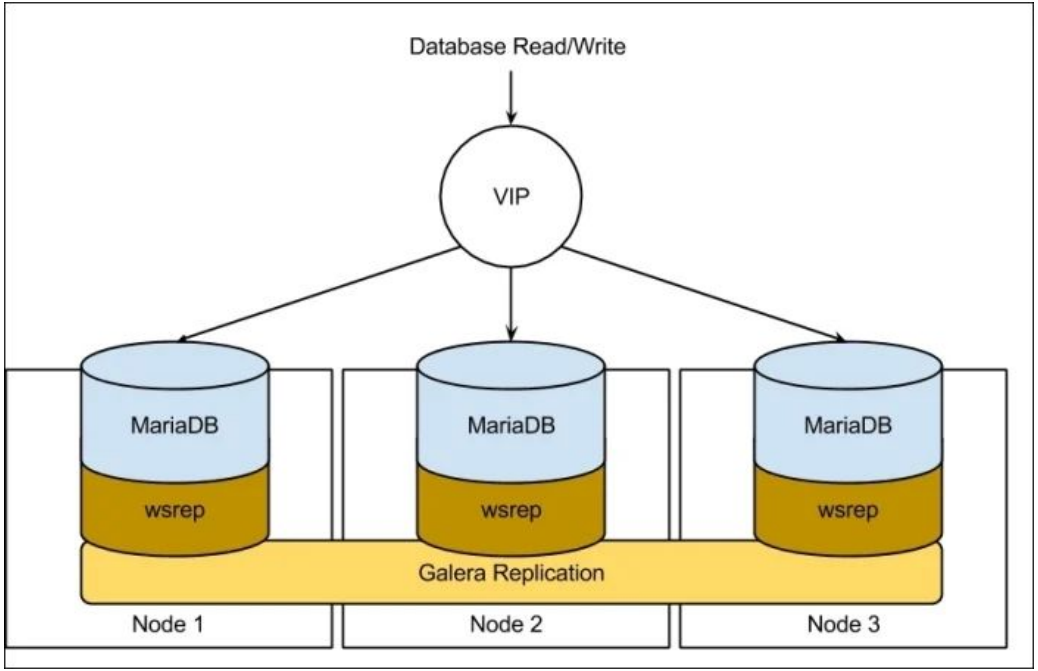
- The master-slave replication process enables database administrators to replicate or copy data stored in more than one server simultaneously.
- This helps the database administrator to create a live backup of the database all the time. During some situations when the master-slave is down to any issues, the can instantly switch over the slave database and keep the application up and running.
- The replication process ensures that your application does not face any kind of downtime at



Configuring HA for databases

Galera Cluster

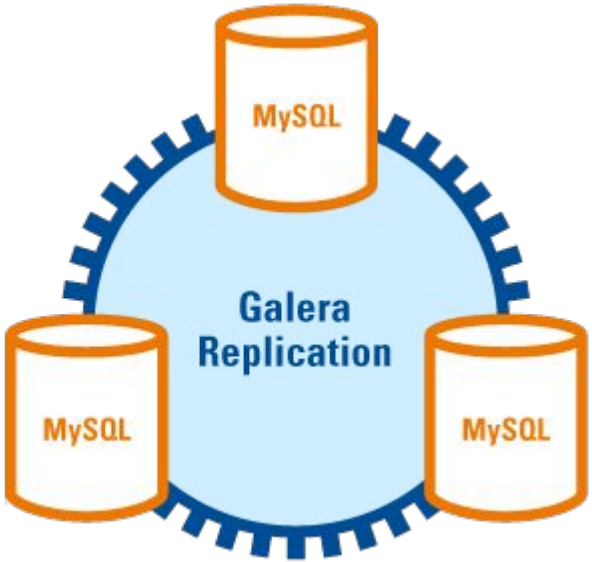
- Galera is a multimaster cluster for MariaDB, which is based on synchronous replication between all cluster nodes.
- Galera treats a cluster of MariaDB nodes as one single master node that reads and writes to all nodes.
- Galera replication happens at transaction commit time, by broadcasting the transaction write set to the cluster for application.
- Client connects directly to the DBMS and experiences close to the native DBMS behavior. write set replication (wsrep) API defines the interface between Galera replication and the DBMS



Configuring HA for databases

Galera clusters can be implemented as geographic clusters, at least for services that do not write intensively to the database (e.g. keystone, glance).

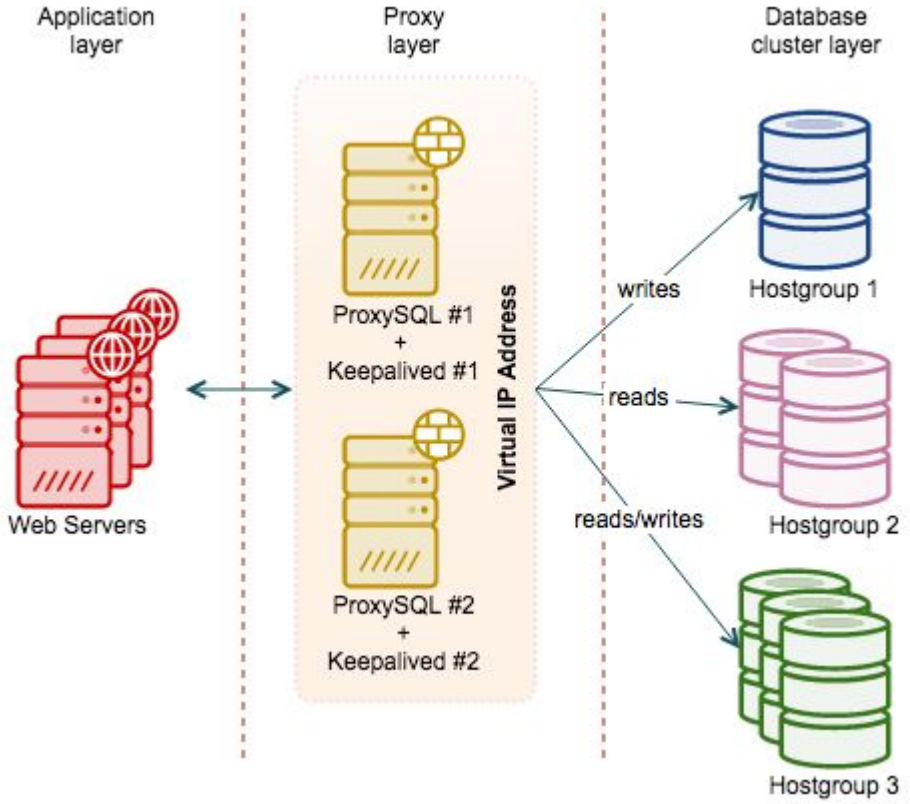
This makes the implementation of multi-region clouds easier.



Configuring HA for databases

ProxySQL

ProxySQL can act for SQL as HAProxy does for HTTP, implementing a level 7 highly configurable load balancer.



Configuring HA for the messaging service

RabbitMQ

- An AMQP (Advanced Message Queueing Protocol) compliant message bus is required for most OpenStack components in order to coordinate the execution of jobs entered into the system.
- The most popular AMQP implementation used in OpenStack installations is RabbitMQ.
- RabbitMQ can be deployed on a single host or in a cluster. In this case nodes fail over on the application and the infrastructure layers.
- The application layer is controlled by the **oslo.messaging** configuration options for multiple AMQP hosts. If the AMQP node fails, the application reconnects to the next one configured within the specified reconnect interval. The specified reconnect interval constitutes its SLA.
- Making the RabbitMQ service highly available involves the following steps:
    a. Install RabbitMQ
    b. Configure RabbitMQ for HA queues
    c. Configure OpenStack services to use RabbitMQ HA queues

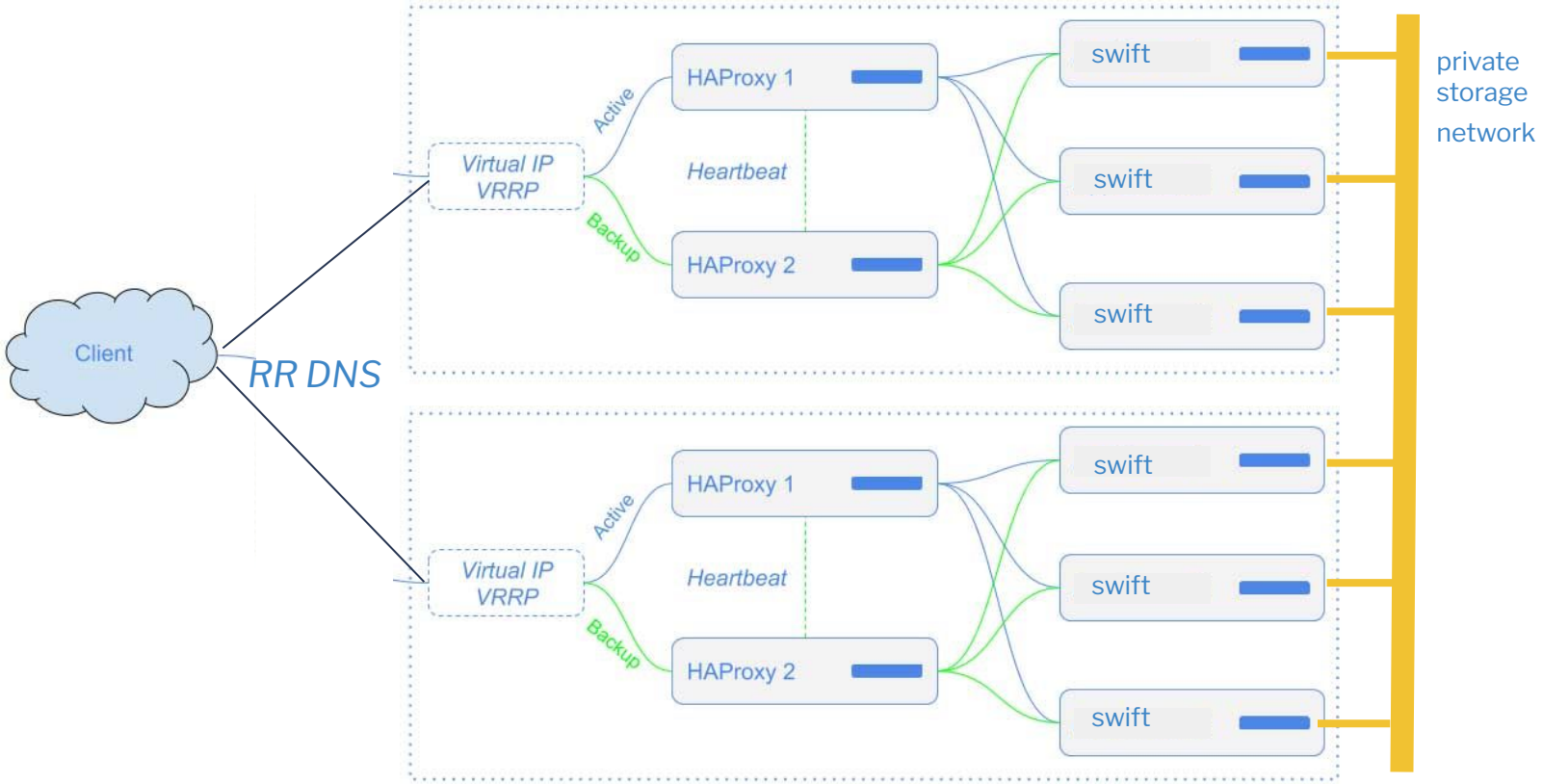
HA for georeplicated services

In multi region OpenStack deployments some services can be deployed in multiple data centers. Implementing HA in such environments represents a different challenge with respect to services that are deployed locally.

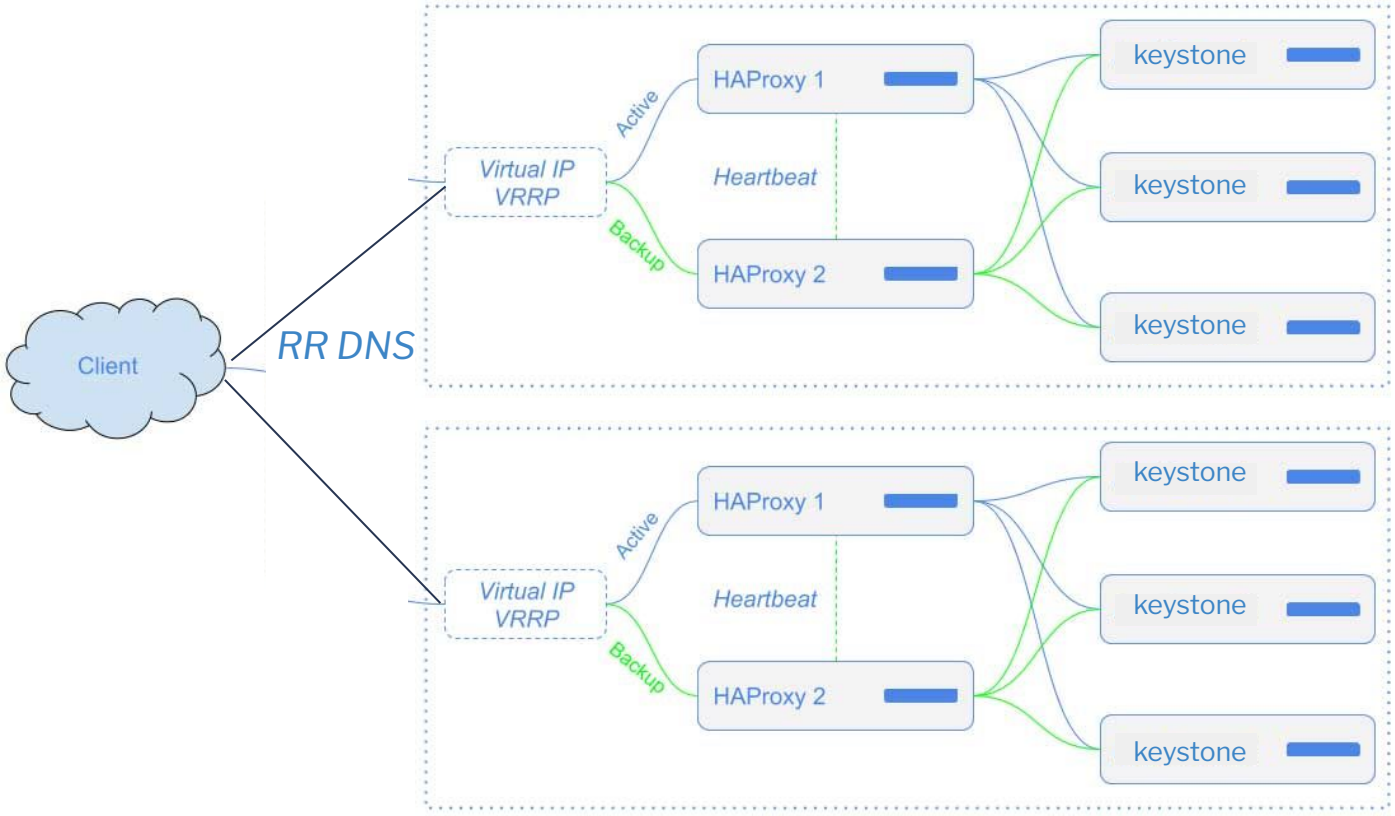
As tools like keepalived can not cross the boundary of a LAN, different approaches must be used for making a service endpoint redundant:

- Dynamic DNS
- global IPs (?)
- ???

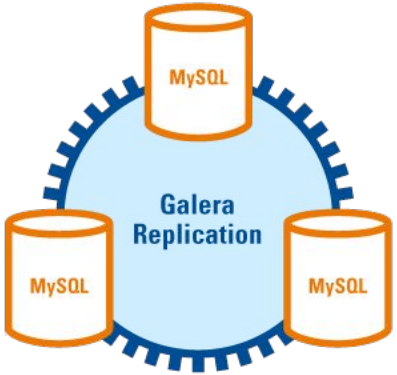
HA for georeplicated service / Swift



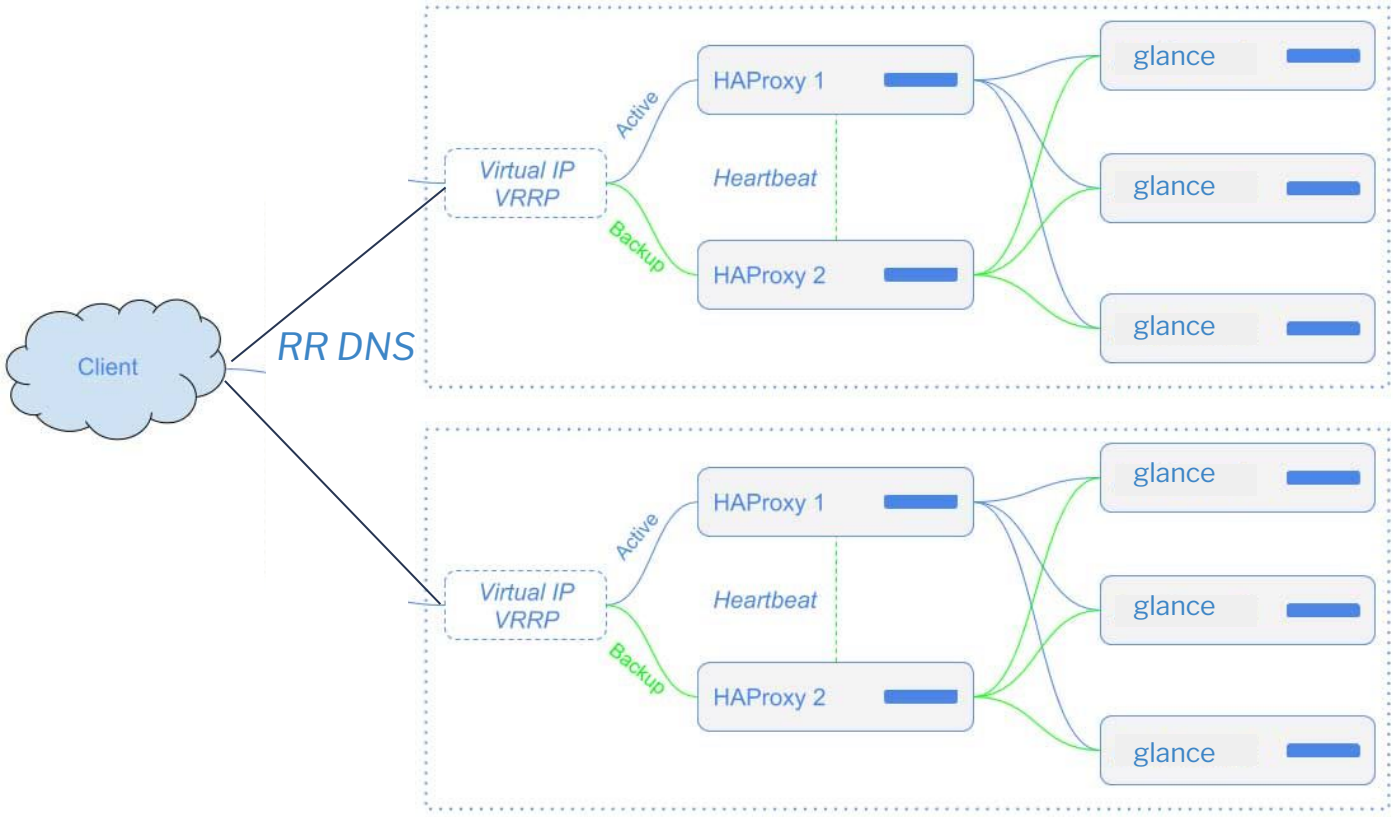
HA for georeplicated services / Keystone



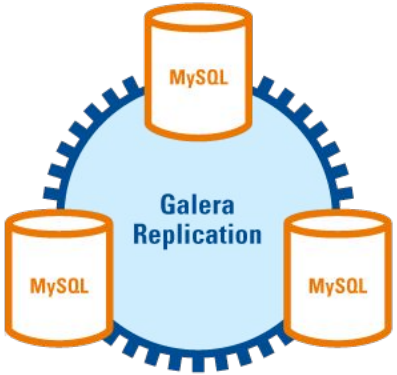
distributed
SQL cluster



HA for georeplicated services / Glance



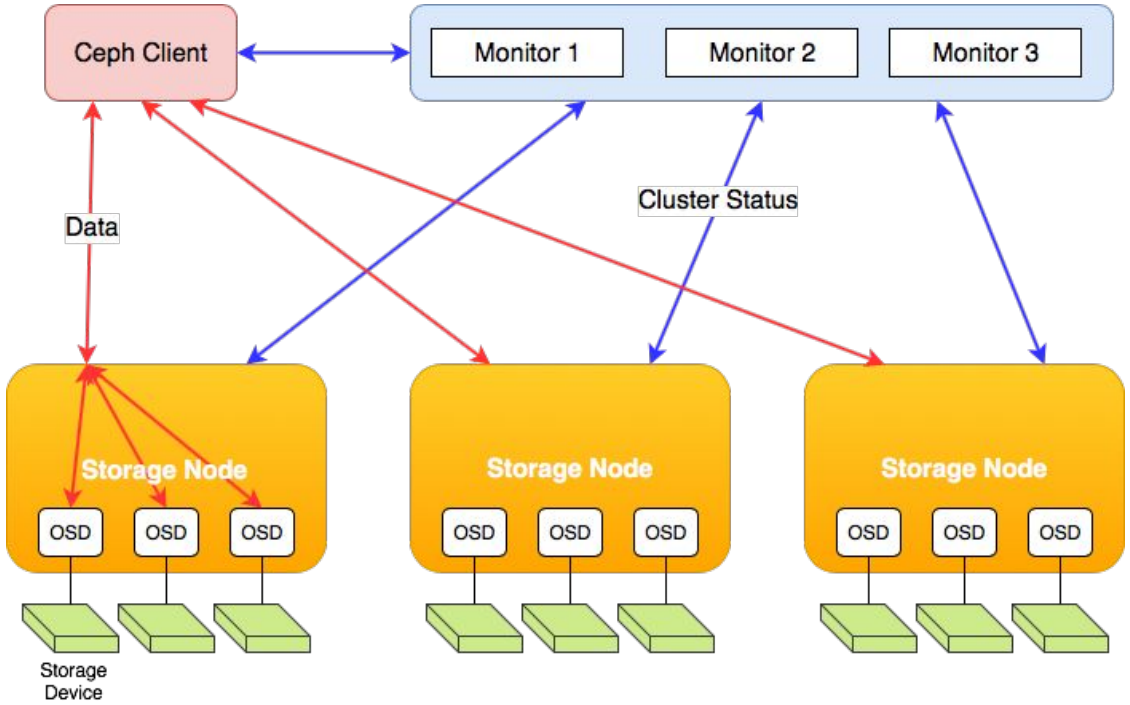
distributed
SQL cluster



Storage backend high availability

- One important component of any cloud service is the block storage backend
- High availability of the storage backend is not built within OpenStack but at the storage level
- Many storage products or architectures that can be used as Cinder/Nova storage backends implement HA, in different ways
- Ceph is presently one of the most used backends for OpenStack block devices and has an extremely solid fully redundant architecture that suffers no single point of failure in well designed implementation

Ceph high availability



Ceph high availability

- Ceph block devices support QEMU/KVM. You can use Ceph block devices with software that interfaces with libvirt. The following stack diagram illustrates how libvirt and QEMU use Ceph block devices via librbd.
- The most common libvirt use case involves providing Ceph block devices to cloud solutions like OpenStack or CloudStack. The cloud solution uses libvirt to interact with QEMU/KVM, and QEMU/KVM interacts with Ceph block devices via librbd.