**Cinder API High availability and load balancing**

**1)What?**

High availability of cinder API(c-api) service;

To cater to high load and withstand failures and be available throughout, we need a HA solution for SBS’ various components;

To begin with, we need the cinder API service of SBS to be highly available.

**2)Goals:**

* ability to withstand partial failures(process issue, hardware issue, network issue)
* ability to cater to high load and scale out if need be.

TBD - HA goal numbers

**3)Current design:**

/- ( LDAP )  
 [ Auth Manager ] ---  
 | \- ( DB )  
 |  
 |  
 cinderclient |  
 / \ |  
[ Web Dashboard ]- -[ api ] -- < AMQP > -- [ scheduler ] -- [ volume ] -- ( iSCSI )  
 \ / |  
 novaclient |  
 |  
 |  
 |  
 < REST >

* Web Dashboard: potential external component that talks to the api
* api: component that receives http requests, converts commands and communicates with other components via the queue or http
* Auth Manager: component responsible for users/projects/and roles. Can backend to DB or LDAP. This is not a separate binary, but rather a python class that is used by most components in the system.
* scheduler: decides which host gets each volume
* volume: manages dynamically attachable block devices.

Source:

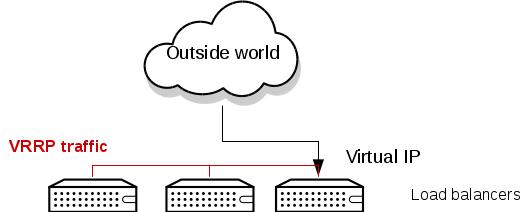
http://docs.openstack.org/developer/cinder/devref/architecture.html

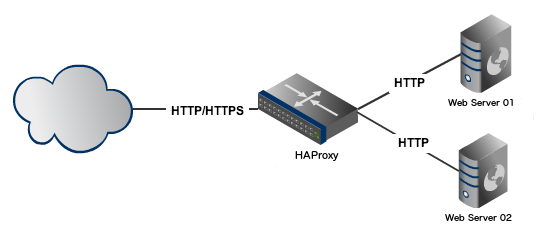
**4)Problems**

\* The c-api service is a single point of failure(SPOF) and has no failure handling capabilities.

\* The c-api service has no load-balancing capabilities built-in.

**5)Proposed design:**





Source:

<https://docs.google.com/document/d/1wQfdNrEJKsHGATn5H_1kxmQcsieSjM473kjUFeKtHv4/edit?ts=5680c4ac>

Add redundancy by having multiple API service nodes to cater to high load and withstand failures.

To begin with, have two c-api nodes. This would server two purposes:

a)share the load

b)in case a node fails through a process issue, hardware issue or network issue, the other node would ensure availability of our SBS service.

To distribute the load across these API nodes, we need a load balancing(LB) proxy;

To prevent the proxy from being a single point of failure, we shall have a standby proxy.

We’ll have a Virtual IP(VIP) associated with an active proxy and when it fails, the VIP will be reassociated with the standby proxy;

**6)Failovers:**

* If a c-api node fails, the LB proxy would route all requests to the available node. There is no failover downtime involved here.
* If the LB proxy fails, the next second the VIP would be associated with the standby proxy. So, there is no failover downtime involved here as well.

**7)Scale-out:**

If need be, we could add a new c-api node to cater to increased load requirements. There is no failover downtime involved here as well.

**8)Reasoning behind choices:**

Status quo:

* LB proxy - HAProxy - begin with haproxy as it's used across the company

Requirements:

We need a proxy that distributes the load in round robin fashion to multiple application servers(c-api service nodes) that sit behind the proxy.

Alternative that could be considered in future - Nginx

* LB proxy failover - Keepalived -

Requirements:

We need a service to switch the virtual IP from the active LB proxy to the standby LB proxy when the former fails.

Choices considered:

Heartbeat/Corosync:

Pros:

\* Supports two node active/passive configuration

Cons:

\* was developed to maintain cluster resources and by virtue of that, has sophisticated functionalities like resource restart/migration and resource management with interdependencies and constraints; Serves as a good fit when you want a true active/standby cluster setup (where only one node is actually "up.") eg. when you share filesystems; So, not necessary to have such a tool for a simple redundant proxy setup requirement as ours.

\* also, in case of a failure say network partition, a cluster-based product may very well end up with none of the nodes offering the service, to ensure that the shared resource is never corrupted by concurrent accesses. Keepalived/ucarp may end up with the IP present on both nodes, resulting in the service being available on both of them.

Ucarp:

Pros:

\* Supports automatic virtual IP failover

Cons:

\* Not much documentation online on ucarp and not much people seem to be using it.

\* no active development in ucarp. Latest version is one released in Feb,2010.

Keepalived:

Pros:

\* Supports automatic virtual IP failover.

\* simpler to setup (one daemon and config) and lower CPU overhead.

\* Active development. Latest version is one released in July, 2015.

\* Good documentation and lot of people seem to be using it.

Cons:

\* In case of network partition, two nodes could hold the same vip

**9)Monitoring - Out of scope of this document. Taken care in monitoring activity.**

We need monitoring scripts that would check the health of the three processes namely keepalived, haproxy, c-api and restart them in case of failures; The restart attempts would be limited and if the processes aren’t up within that, then manual intervention would be required. In either case, notifications would be sent about failure details.