



## Enterprise IaaS: High-availability Implementation Guide

---

## Informazioni deliverable

|  |   |
|--|---|
| Titolo del deliverable   |   |
| Sigla di classificazione del documento   |   |
| Responsabile del Deliverable   | INFN  |
| Obiettivo Realizzativo   | OR 1: Studio e realizzazione di Piattaforma cloud |
| Attività relativa  |   |
| Autori   | INFN  |
| Approvazione   | Reply, INFN, Sielte                               |
| Autorizzazione   |   |
| <p>Questo documento è un Report Tecnico che descrive gli step di installazione e configurazione dell'high-availability nella IAAS.</p> |   |

I contenuti del documento sono da intendersi “confidenziali” di proprietà esclusiva delle parti coinvolte nel progetto “PRISMA”.

Né il documento né le sue parti possono essere pubblicate, riprodotte, copiate o comunque divulgate senza autorizzazione scritta delle parti coinvolte nel progetto oltre quanto previsto dalla lista di distribuzione

### Lista di distribuzione

## Stato del deliverable

| Ver. | Data       | Autore della modifica | Note          | Validazione |
|------|------------|-----------------------|---------------|-------------|
| 1.0  | 14/07/2014 | Marica Antonacci      | Prima Stesura |             |
|      |            |                       |               |             |
|      |            |                       |               |             |

## INDICE DEGLI ARGOMENTI

|  |           |
|--|-----------|
| <b>ACRONIMI .....</b>  | <b>5</b>  |
| <b>1. INTRODUCTION .....</b>                                     | <b>6</b>  |
| 1.1 Active/passive and active/active configurations .....        | 7         |
| 1.2 High-availability implementation strategy .....              | 8         |
| 1.2.1 Notes .....  | 10        |
| 1.3 Software versions.....                                       | 11        |
| <b>2. CONFIGURING MYSQL AND GALERA .....</b>                     | <b>11</b> |
| 2.1 Node Preparation.....  | 13        |
| 2.2 Install HAProxy .....  | 14        |
| 2.3 Configuration of the database cluster for OpenStack .....    | 17        |
| <b>3. INSTALL RABBITMQ CLUSTER .....</b>                         | <b>20</b> |
| <b>4. GLUSTERFS INSTALLATION .....</b>                           | <b>23</b> |
| <b>5. INSTALLING AND SETTING UP PACEMAKER AND COROSYNC .....</b> | <b>24</b> |
| 5.1 Starting the Pacemaker and Corosync services .....           | 25        |
| 5.2 Configure Openstack Services .....                           | 27        |
| 5.2.1 Configure the Cluster Virtual IPs (VIP).....               | 27        |
| 5.2.2 Keystone .....   | 28        |
| 5.2.3 Glance .....   | 30        |
| 5.2.4 Nova .....   | 36        |
| 5.2.5 Neutron.....   | 40        |
| 5.2.6 Horizon .....  | 43        |
| <b>6. RIFERIMENTI.....</b>                                       | <b>44</b> |

## Acronimi

|        |   |
|--------|---|
| HA     | High-Availability                                     |
| PRISMA | PiattafoRme cloud Interoperabili per SMArt-government |
| RA     | Resource Agent  |
| REST   | REpresentational State Transfer                       |
| SLA    | Service Level Agreement                               |
| SPOF   | Single Point Of Failure                               |
| VA     | Virtual Appliance                                     |
| VM     | <b>V</b> irtual <b>M</b> achine                       |

## 1. Introduction

High availability (HA) is a *must have* for **enterprise** deployment of OpenStack. High-availability systems seek to minimize two things:

- **System downtime** — occurs when a user-facing service is unavailable beyond a specified maximum amount of time, and
- **Data loss** — accidental deletion or destruction of data.

A crucial aspect is the elimination of *single points of failure* (SPOFs). A SPOF is an individual piece of equipment or software that will cause system downtime or data loss if it fails.

In order to eliminate SPOFs, redundancy mechanisms have to be set-up for:

- Network components, such as switches and routers
- Applications and automatic service migration
- Storage components
- Facility services such as power, air conditioning, and fire protection

Preventing single points of failure can depend on whether or not a service is stateless.

A stateless service is one 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.

A stateful service is one 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, so simply providing additional instances and load balancing will not solve the problem. For example, if the Horizon user interface reset itself every time you went to a new page, it wouldn't 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.

## 1.1 Active/passive and active/active configurations

In an active/passive configuration, systems are set up to bring additional resources online to replace those that have failed. For example, OpenStack would write to the main database while maintaining a disaster recovery database that can be brought online in the event that the main database fails.

Typically, an active/passive installation for a stateless service would maintain a redundant instance that can be brought online when required. Requests are load balanced using a virtual IP address and a load balancer such as HAProxy.

A typical active/passive installation for a stateful service maintains a replacement resource that can be brought online when required. A separate application (such as Pacemaker or Corosync) monitors these services, bringing the backup online as necessary.

In an active/active configuration, systems also use a backup but will manage both the main and redundant systems concurrently. This way, if there is a failure the user is unlikely to notice. The backup system is already online, and takes on increased load while the main system is fixed and brought back online.

Typically, an active/active installation for a stateless service would maintain a redundant instance, and requests are load balanced using a virtual IP address and a load balancer such as HAProxy.

A typical active/active installation for a stateful service would include redundant services with all instances having an identical state. For example, updates to one instance of a database would also update all other instances. This way a request to one instance is the same as a request to any other. A load balancer manages the traffic to these systems, ensuring that operational systems always handle the request.

These are some of the more common ways to implement these high availability architectures, but they are by no means the only ways to do it. The important thing is to make sure that services are redundant, and available.

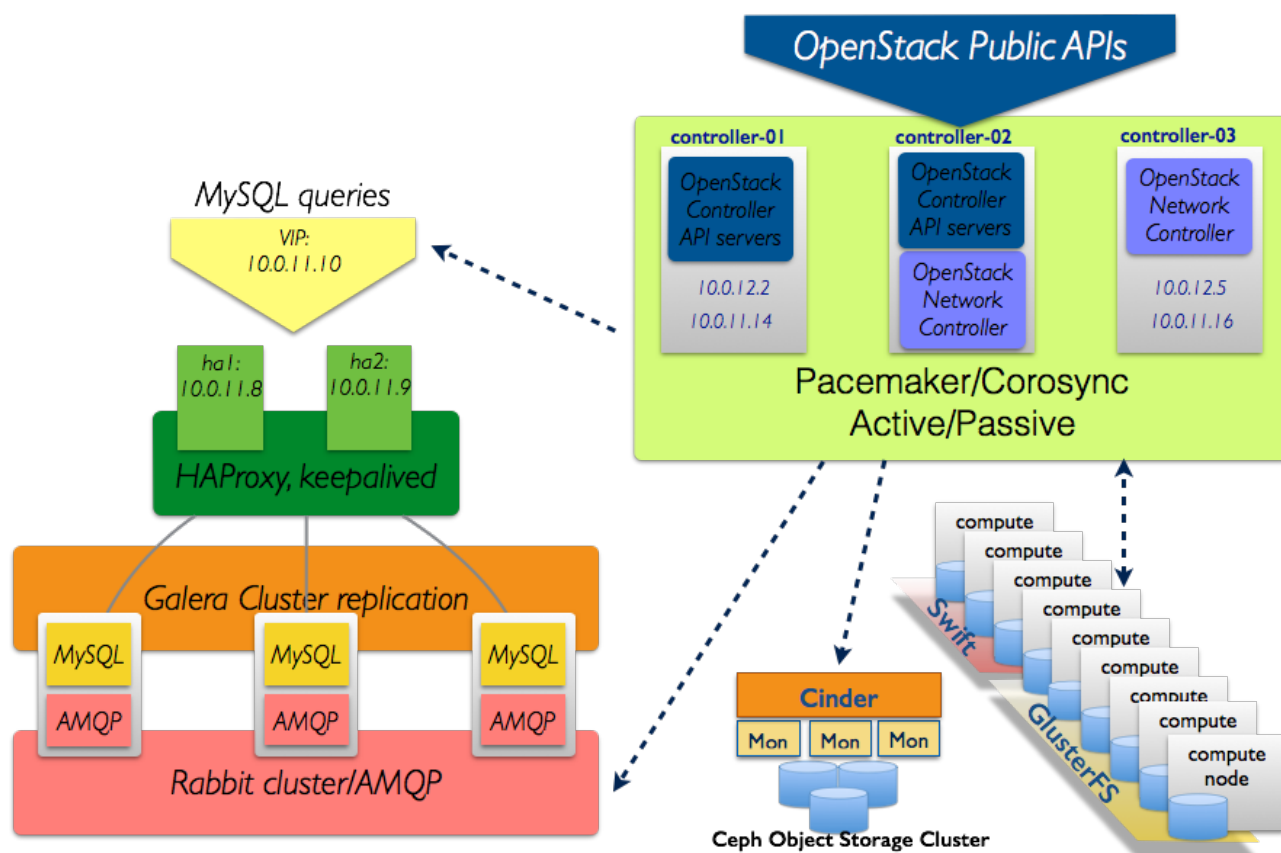
## 1.2 High-availability implementation strategy

As already explained in the previous section, there are many ways to implement the high-availability and what is described in this document is just one of the possible choices, but has the advantage of being simple to install/deploy, using only one software solution to control and manage all the Openstack services.

Figure 1 shows the deployment of the core services and underlying infrastructure services (database and message broker) that will be described in this document. The presented solution exploits mature industry proven open source technologies, as detailed hereafter:

- **MySQL Galera** cluster + HA Proxy load balancing for the database deployment;
- Rabbitmq messaging servers cluster with “**active-active mirrored queues**” configuration;
- **Pacemaker/corosync** cluster to manage all Openstack services (APIs, schedulers, etc.) in **active/passive** configuration;
- **Swift cluster** (replica 3) with redundant proxy servers;
- **Shared Filesystem**: glusterFS in replica 3 to host the virtual images (as default backend of the Glance server) and the running VM images (as backend for Nova enabling the VM live migration).





**Figure 1 Openstack deployment implementing high-availability**

Database queries are load balanced through clustered HAProxy in front of the database instances: the database endpoint is exposed through the virtual IP (VIP) managed by Keepalived.

Galera is used to synchronize the persistent data layer across the running database nodes. Galera is a synchronous multi-master cluster for MySQL database, handling synchronous replication. This enables an Active/Active scale-out of the database layer without requiring shared storage.

Concerning the Openstack services, they are typically grouped in two main nodes: the controller node (hosting the management services and the APIs servers) and the network node.

In our deployment we have two controller nodes (one active and one in stand-by) and two network nodes (one active and one in stand-by) distributed on three physical hosts as follows:

| Host name | Role |
|-----------|------|
|-----------|------|

| Host name           | Role  |
|---------------------|---|
| controller-01       | Controller active node                            |
| controller-02       | Network stand-by node<br>Controller stand-by node |
| controller-03       | Network active node                               |
| vm-01, vm-02, vm-03 | Galera & RabbitMQ cluster                         |

Moreover, the Galera MySQL cluster is installed on three separate nodes used also for the RabbitMQ cluster.

We will start by setting each control node up as we would a non-HA control node. Once each control node is installed without being aware of each other the nodes will be clustered together using Pacemaker.

Pacemaker implements the state-of-the-art high availability and load balancing stack for the Linux platform. Pacemaker is storage and application-agnostic, and is in no way specific to OpenStack.

Pacemaker relies on the Corosync messaging layer for reliable cluster communications. Corosync implements the Totem single-ring ordering and membership protocol. It also provides UDP and InfiniBand based messaging, quorum, and cluster membership to Pacemaker.

Pacemaker interacts with applications through resource agents (RAs), of which it supports over 70 natively. Pacemaker can also easily use third-party RAs. An OpenStack high-availability configuration uses existing native OpenStack RAs (such as those managing the OpenStack Identity and Image Services).

### 1.2.1 Notes

The scope of this document is to outline how to achieve Openstack services high-availability.

As the load and demand on OpenStack services the reader might be interested in the ability to add nodes and scale-out the controller plane. Building a scale-out controller would require setting all services and infrastructure components (database and message broker) in Active/Active

configuration and confirming that they are capable to add more nodes to the cluster as load grows, and balancing the API requests load between the nodes. In this case, you have also to implement the monitoring of the services to check their status (this is automatically done by pacemaker resource agents in the HA configuration described in this document).

### 1.3 Software versions

The following table summarizes the software packages and their versions used for the deployment described in this document:

| Package                      | Version                  |
|------------------------------|--------------------------|
| Openstack                    | Icehouse                 |
| SeveralNines Cluster Control | 2.8                      |
| MySQL                        | 5.5 (with galera 24.2.x) |
| RabbitMq                     | 2.7.1                    |
| HAProxy                      | 1.4.18                   |

The installation steps described in this guide have been tested using Ubuntu 12.04 servers.

## 2. Configuring MySQL and Galera

1. We first use a Web browser from our desktop and head over to <http://www.severalnines.com/galera-configurator/>, where we will input some information about our environment to produce the script required to install our Galera-based MySQL cluster.
2. The first screen asks for the **Vendor**. Select **Codership (based on MySQL 5.5)** as shown in the following screenshot:

Vendor

Select the vendor you want to use:  
☒ [Codership](#) (based on MySQL 5.5, galera 24.2.x) - requires internet access from Controller  
☐ [Percona XtraDb Cluster](#) (latest yum/apt repos or internet-less install optional)  
☐ [MariaDB Cluster](#) (latest yum/apt repositories are used)

Attention: Packages may be removed during the automatic installation process. Use clean/dedicated servers.

Next

3. The next screen asks for general settings, as follows:

```
Infrastructure: none/on-premise
Operating System: Ubuntu 12.04
Platform: Linux 64-bit (x86_64)
Number of Galera Servers: 3+1
MySQL PortNumber: 3306
Galera PortNumber: 4567
Galera SST PortNumber: 4444
SSH PortNumber: 22
OS User: galera
MySQL Server Password (root user): openstack
CMON DB password (cmon user): cmon
Firewall (iptables): Disabled
```

4. Next, we will configure server properties (configure as appropriate):

```
System Memory (MySQL Servers): [at least 512Mb]
WAN: no
Skip DNS Resolve: yes
Database Size: [Select the option that comes closest to the anticipated size of your database]
Galera Cache (gache): [The gcache can speed up recovery significantly if set large enough. Optimal is to set it to the same size as the database size if possible]
MySQL Usage: [e.g. Medium write/high read]
Number of cores: <N> #Specify how many cores the computer has
Max connections per server: 200
Innodb_buffer_pool_size: X Mb
Innodb_file_per_table: checked
```

5. On the next screen, we'll configure the nodes and addresses. The first section asks for details about our ClusterControl Server running Cmon, as follows:

```
ClusterControl Server: 10.0.11.7
System Memory: (at least 512Mb)
Datadir: <same as for mysql>
Installdir: /usr/local

Web server(apache) settings
```

```
Apache User: www-data
WWWROOT: /var/www/
```

6. Further down the screen, we can now configure the Galera nodes. The following table lists the IP address, data directory, and installation directory for the servers.

Config Directory: /etc/mysql

| Server-id | IP-address | Datadir                   | InstalDir                 |
|-----------|------------|---------------------------|---------------------------|
| 1         | 10.0.11.4  | /var/lib/mysql/           | /usr/local/               |
| 2         | 10.0.11.5  | same as mentioned earlier | same as mentioned earlier |
| 3         | 10.0.11.6  | same as mentioned earlier | same as mentioned earlier |

7. The final step asks which e-mail address the configuration and deployment script should be sent to. Once a valid e-mail address has been entered, press the Generate Deployment Scripts button. You will be taken to a summary screen where you will be presented with an API key. You will require this key to complete the installation.

## 2.1 Node Preparation

1. Add user galera, change its password, ensure the galera user can execute commands using sudo without being asked for a password. To do this, we execute the following on all nodes:

```
# useradd -m -s /bin/bash galera
# passwd galera
# echo "galera ALL=(ALL:ALL) NOPASSWD:ALL" | tee -a /etc/sudoers.d/galera
# chmod 0440 /etc/sudoers.d/galera
```

- Each node is configured such that the user used to run the setup routine (the OS user as configured in step 2 in the previous section) can SSH to each node—including itself—and run commands through sudo without being asked for a password. To do this, we first create the user's SSH key as follows:

```
# ssh-keygen -t rsa -N ""
```

- We now need to copy this to each of our nodes, including the node we are on now (so that it can SSH to itself).

## 2.2 Install HAProxy

Now we will install and configure the load balancer (HAProxy) using the cluster control UI.

You can use two physical hosts or two virtual machines (running on separate hosts) to run the load balancers in high-availability fashion.

### Pre-requisites:

You need to allocate a floating IP that will be used for load-balancing: all Openstack services will be configured to establish connections to the MySQL databases using this virtual IP.

The following table contains the IPs that will be used in the following sections to describe the set-up of the load-balancers:

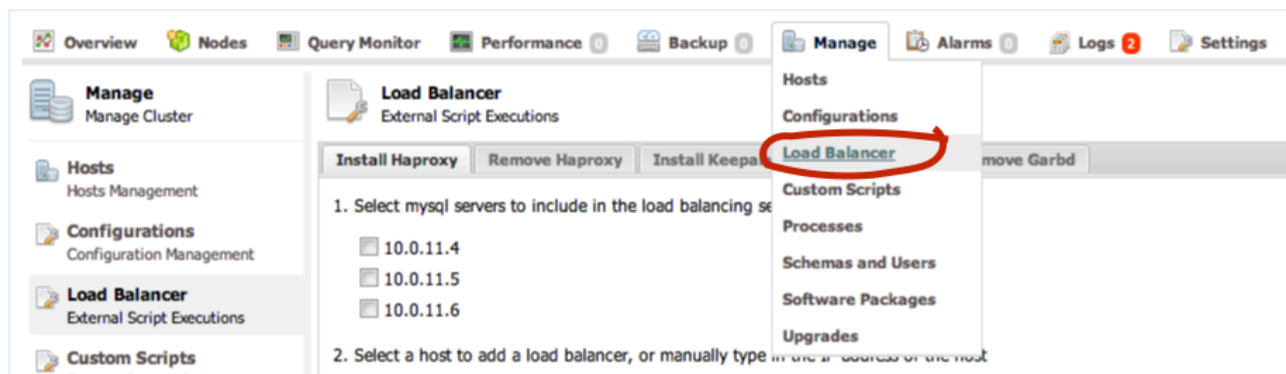
|   |                                    |
|---|------------------------------------|
| Ubuntu 12.04 servers  |                                    |
| Load balancers IPs  | 10.0.11.8, 10.0.11.9               |
| Floating IP to be used by keepalived for the Virtual IP (VIP) | 10.0.11.10                         |
| MySQL servers   | 10.0.11.4, 10.0.11.5,<br>10.0.11.6 |

Note that, since the installation is performed by the same linux user used for the installation of the galera cluster, we will create the user 'galera' also on these two servers and grant the access via ssh without password copying the keys across the new servers.

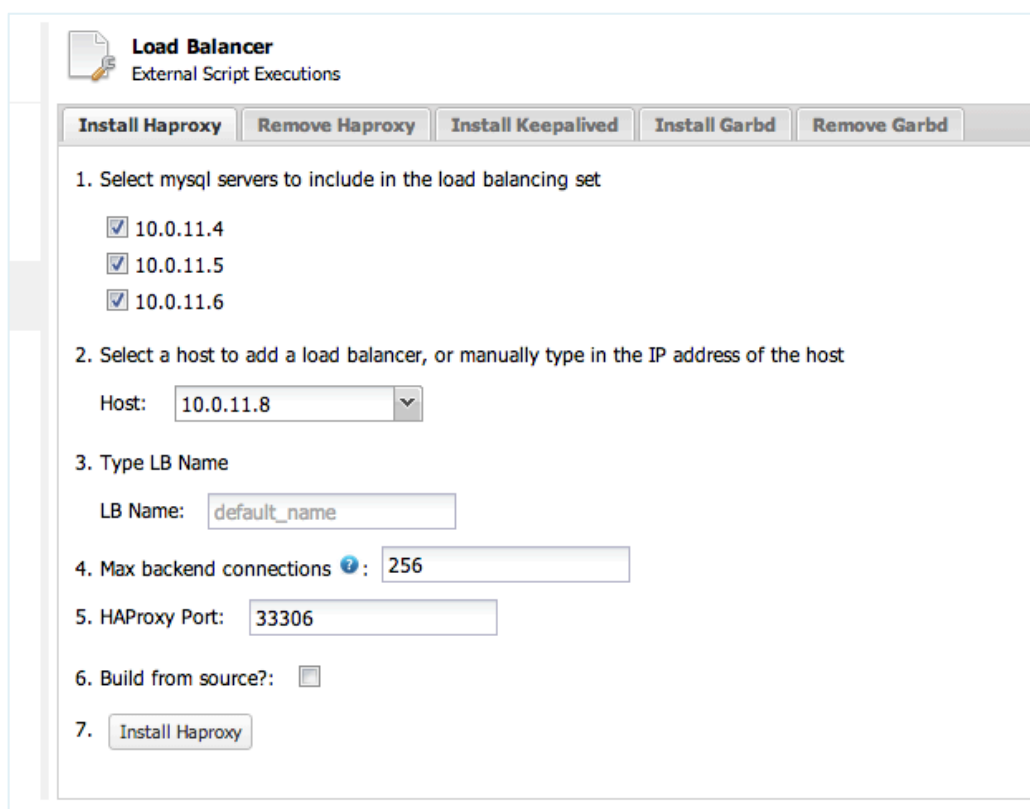
### Installation steps:

Then, we can start the installation following the steps below:

1. From menu “Manage” choose “Load Balancer”, then “Install HA Proxy”



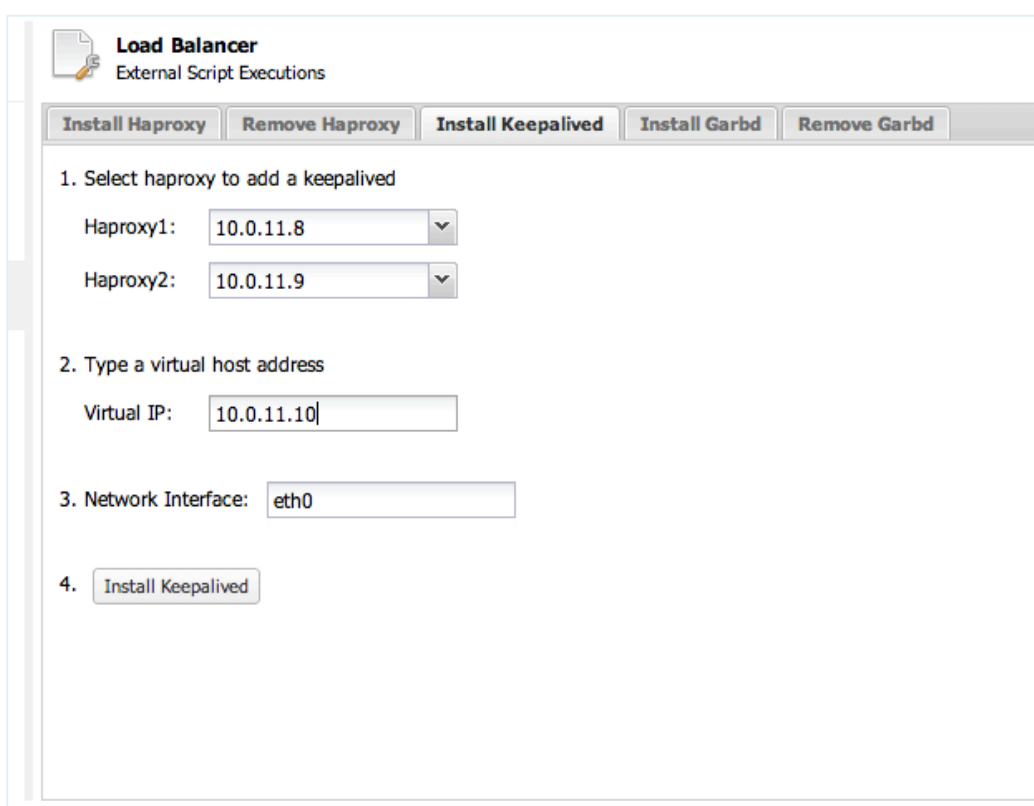
2. Select mysql servers to include in the load balancing pool
3. Insert the IP of the host where the load balancer will be installed
4. Click the button “Install Haproxy”

A screenshot of the 'Load Balancer' configuration page in the PRISMA web interface. The page has a title 'Load Balancer' and a subtitle 'External Script Executions'. Below the title are several tabs: 'Install Haproxy', 'Remove Haproxy', 'Install Keepalived', 'Install Garbd', and 'Remove Garbd'. The 'Install Haproxy' tab is active. The page contains a list of MySQL servers to include in the load balancing set, with checkboxes next to each IP address: 10.0.11.4, 10.0.11.5, and 10.0.11.6. Below this, there is a section for selecting a host to add a load balancer, with a dropdown menu showing '10.0.11.8'. There is also a section for typing the LB Name, with a text input field containing 'default\_name'. Below that, there is a section for setting the Max backend connections, with a text input field containing '256'. There is also a section for setting the HAProxy Port, with a text input field containing '33306'. At the bottom, there is a checkbox for 'Build from source?' and a button labeled 'Install Haproxy'.

Repeat steps from 1 to 4 in order to install the second load balancer.

Then install Keepalived:

1. Select the two load balancers
2. Insert the floating IP to be used as Virtual IP
3. Click the button “Install Keepalived”



### **Important:**

Galera cluster has known limitations, one of them is that it uses cluster-wide optimistic locking. This may cause some transactions to rollback:

```
2014-07-08 09:38:00.744 3127 CRITICAL keystone [-] OperationalError:
(OperationalError) (1213, 'Deadlock found when trying to get lock; try
restarting transaction') None None
```

### **Workaround:**



Send conflicting updates to one node only. When deploying HAProxy via ClusterControl, HAProxy will load balance all requests across the load balancing set. Modify HAProxy configuration file located at `/etc/haproxy/haproxy.cfg` to redirect to only one Galera node at a time (see last three lines):

```
listen s9sl_33306_default_LB
    bind *:33306
    mode tcp
    timeout client 60000ms
    timeout server 60000ms
    balance leastconn
    option httpchk
    option allbackups
    default-server port 9200 inter 2s downinter 5s rise 3 fall 2 slowstart
    60s maxconn 256 maxqueue 128 weight 100
    server 10.0.11.4 10.0.11.4:3306 check
    server 10.0.11.5 10.0.11.5:3306 check backup
    server 10.0.11.6 10.0.11.6:3306 check backup
```

## 2.3 Configuration of the database cluster for OpenStack

### Character Set

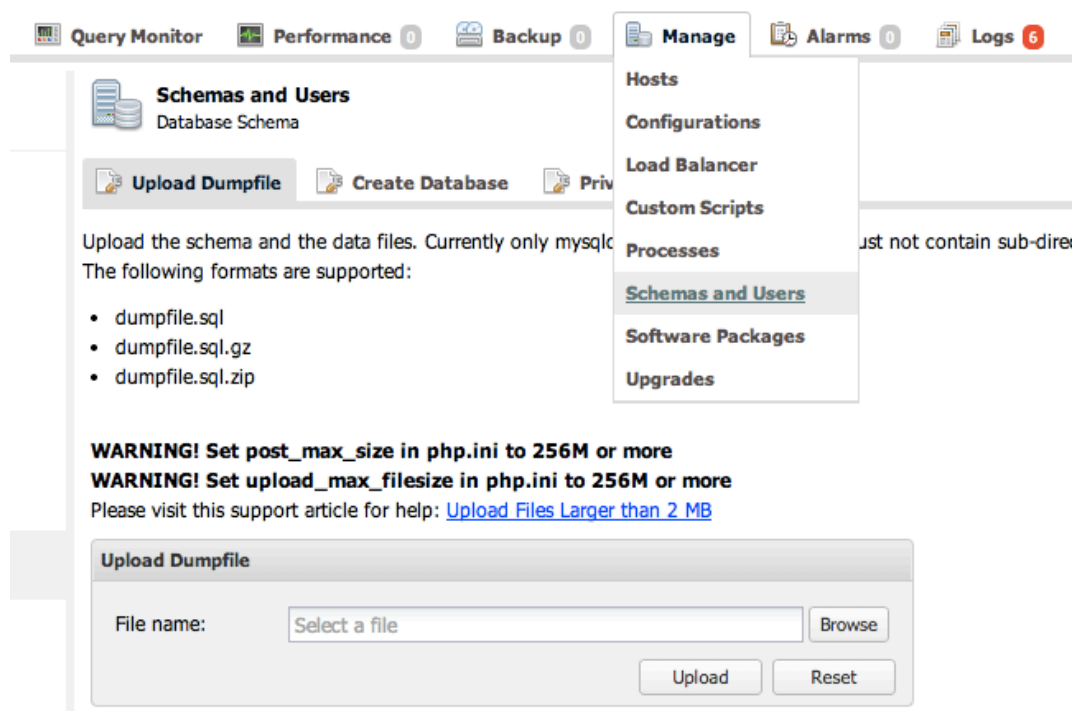
Edit the configuration file `/etc/mysql/my.cnf`: under the `[mysqld]` section, set the following keys to enable UTF-8 character set, and UTF-8 collation by default:

```
[mysqld]
...

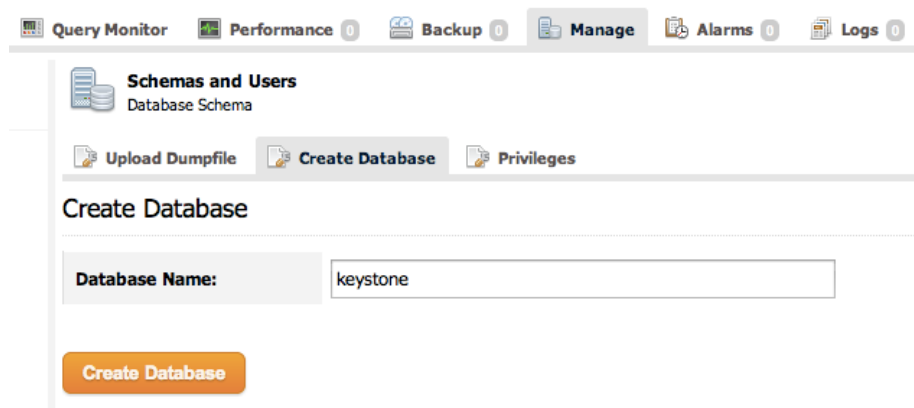
collation-server = utf8_general_ci
init-connect = 'SET NAMES utf8'
character-set-server = utf8
```

Then execute the following steps:

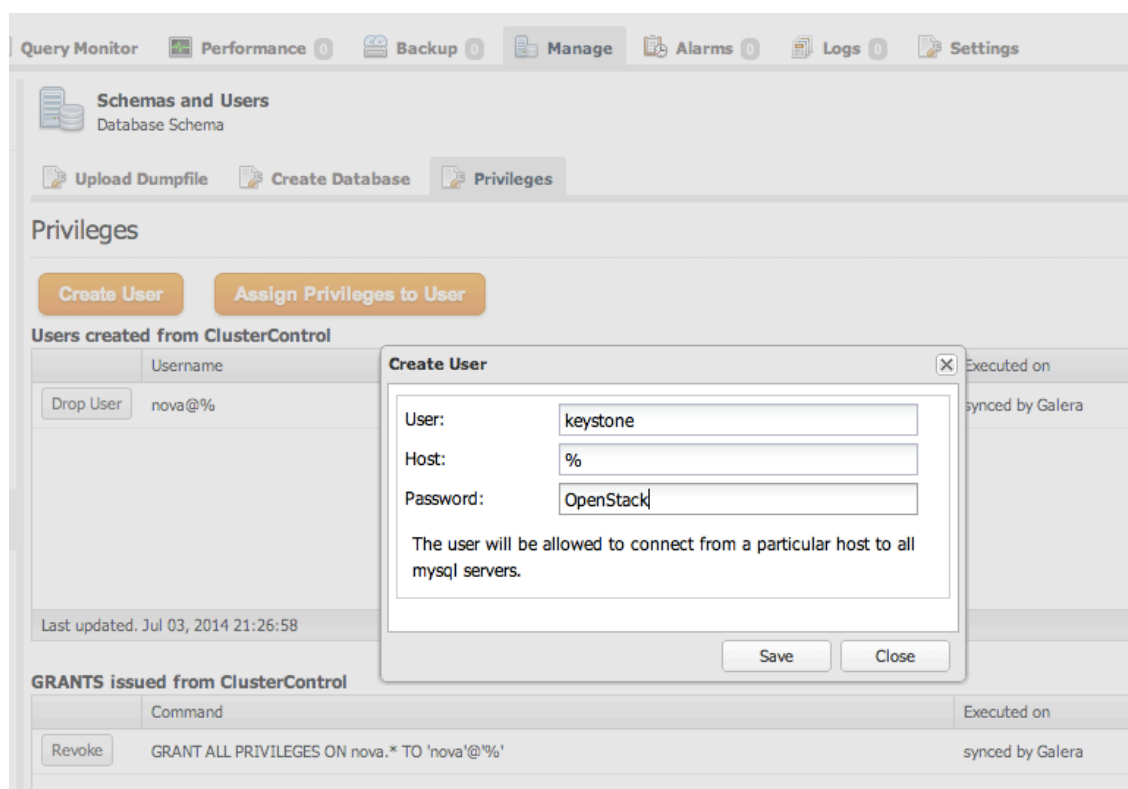
1. Once the cluster has been set up, we can now create the databases, users, and privileges required for our OpenStack environment, as we would do for any other OpenStack installation. To do this, we click on the Manage link



2. From this screen, choose the Manage menu, and select the Schemas and Users menu option as shown in the screenshot above.
3. Under Schema and Users, we can create and drop databases, create and delete users, and grant and revoke privileges. For OpenStack, we need to create five users and the five databases, with appropriate privileges, that relate to our OpenStack installation. These are nova, keystone, glance, quantum (used by Neutron), and cinder. First, we create the keystone database. To do this, click on the Create Database button.



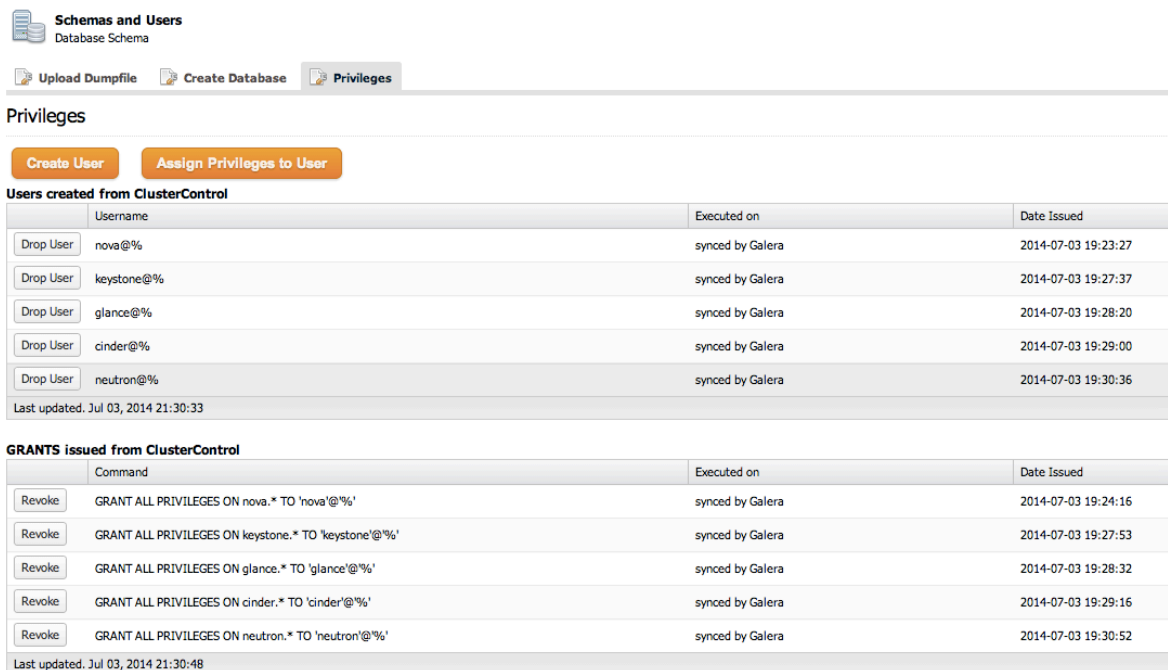
4. Once entered, click on the Create Database button and a popup will acknowledge the request.
5. Repeat the process to create the nova, glance, neutron and cinder databases.
6. Once done, we can now create our users. To do this we click on the Privileges button
7. To create a user called keystone, that we will use to connect to our keystone database, click on the Create User button and fill in the details:



8. Repeat this step for each of the required usernames for our other database, which we will call the same name for ease of administration: glance, keystone, neutron, and cinder.
9. With the users created, we assign their privileges to the corresponding databases. We will create a user named nova, which is allowed to access our database cluster from any host (using the MySQL wildcard character %).

```
Check ALL PRIVILEGES
Database: keystone
User Host: 'keystone'@'%'
```

10. Repeating this step for the other users gives us the required privileges for us to utilize our new cluster for OpenStack



**Schemas and Users**  
Database Schema

Upload Dumpfile Create Database Privileges

**Privileges**

Create User Assign Privileges to User

**Users created from ClusterControl**

|           | Username   | Executed on      | Date Issued         |
|-----------|------------|------------------|---------------------|
| Drop User | nova@%     | synced by Galera | 2014-07-03 19:23:27 |
| Drop User | keystone@% | synced by Galera | 2014-07-03 19:27:37 |
| Drop User | glance@%   | synced by Galera | 2014-07-03 19:28:20 |
| Drop User | cinder@%   | synced by Galera | 2014-07-03 19:29:00 |
| Drop User | neutron@%  | synced by Galera | 2014-07-03 19:30:36 |

Last updated: Jul 03, 2014 21:30:33

**GRANTS issued from ClusterControl**

|        | Command  | Executed on      | Date Issued         |
|--------|--|------------------|---------------------|
| Revoke | GRANT ALL PRIVILEGES ON nova.* TO 'nova'@'%'         | synced by Galera | 2014-07-03 19:24:16 |
| Revoke | GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'%' | synced by Galera | 2014-07-03 19:27:53 |
| Revoke | GRANT ALL PRIVILEGES ON glance.* TO 'glance'@'%'     | synced by Galera | 2014-07-03 19:28:32 |
| Revoke | GRANT ALL PRIVILEGES ON cinder.* TO 'cinder'@'%'     | synced by Galera | 2014-07-03 19:29:16 |
| Revoke | GRANT ALL PRIVILEGES ON neutron.* TO 'neutron'@'%'   | synced by Galera | 2014-07-03 19:30:52 |

Last updated: Jul 03, 2014 21:30:48

### 3. Install RabbitMQ Cluster

We install RabbitMQ on the three nodes:

```
# apt-get install ntp rabbitmq-server
```

After the install has completed, we should stop rabbitmq on all nodes:

```
# service rabbitmq stop
```

Then we copy the erlang cookie from node 1 to node 2. You may need to enable root ssh login on node 2 for this step.

```
[10.0.11.4]# scp /var/lib/rabbitmq/.erlang.cookie  
10.0.11.5:/var/lib/rabbitmq/.erlang.cookie
```

```
[10.0.11.4]# scp /var/lib/rabbitmq/.erlang.cookie  
10.0.11.6:/var/lib/rabbitmq/.erlang.cookie
```

On node vm-1 start the server in detached mode:

```
root@vm-1:~# rabbitmq-server -detached  
Activating RabbitMQ plugins ...  
0 plugins activated:
```

On vm-2 perform the following operations to form the cluster:

```
# rabbitmq-server -detached  
# rabbitmqctl stop_app  
# rabbitmqctl reset  
# rabbitmqctl cluster rabbit@vm-1 rabbit@vm-2  
# rabbitmqctl start_app
```

Hereafter the expected output of the commands:

```
root@vm-2:~# rabbitmq-server -detached  
Activating RabbitMQ plugins ...  
0 plugins activated:  
root@vm-2:~# rabbitmqctl stop_app  
Stopping node 'rabbit@vm-2' ...  
...done.  
root@vm-2:~# rabbitmqctl reset  
Resetting node 'rabbit@vm-2' ...  
...done.  
root@vm-2:~# rabbitmqctl cluster rabbit@vm-1 rabbit@vm-2  
Clustering node 'rabbit@vm-2' with ['rabbit@vm-1','rabbit@vm-2'] ...  
...done.  
root@vm-2:~# rabbitmqctl start_app  
Starting node 'rabbit@vm-2' ...  
...done.
```

On node vm-3 do similar operations:

```
# rabbitmq-server -detached
# rabbitmqctl stop_app
# rabbitmqctl reset
# rabbitmqctl cluster rabbit@vm-1 rabbit@vm-3
# rabbitmqctl start_app
```

Use the command `cluster_status` to see the nodes in the cluster:

```
root@vm-1:~# rabbitmqctl cluster_status
Cluster status of node 'rabbit@vm-1' ...
[{nodes,[{disc,['rabbit@vm-3','rabbit@vm-2','rabbit@vm-1']}]},
 {running_nodes,['rabbit@vm-3','rabbit@vm-2','rabbit@vm-1']}]
...done.
```

## Troubleshooting:

In case you get the following error stopping one of the nodes:

```
# service rabbitmq-server stop
Stopping rabbitmq-server: No process in pidfile '/var/run/rabbitmq/pid' found
running; none killed.
FAILED - check /var/log/rabbitmq/shutdown_log, _err
rabbitmq-server.
```

Then execute the following operations:

1. kill all the running processing belonging to user `rabbitmq`:

```
# killall -u rabbitmq
```

2. Remove the content of `mnesia` folder:

```
# rm -rf /var/lib/rabbitmq/mnesia/*
```

Then stop the node again.

## 4. GlusterFS installation

Recommended configuration: replica 3

1. Install the packages on all the three nodes:

```
# apt-get -y install python-software-properties  
# add-apt-repository ppa:semiosis/ubuntu-glusterfs-3.5  
# apt-get update  
# apt-get -y install glusterfs-server glusterfs-common glusterfs-client xfsprogs
```

2. From one node of the cluster, add the other ones using the “*peer probe*” command:

```
# gluster peer probe 10.0.11.12  
# gluster peer probe 10.0.11.13
```

Check the list of nodes in the cluster:

```
# gluster peer status
```

Prepare the disks (formatting them as xfs) and then add them to the cluster as follows:

```
# gluster volume create gfile replica 3 transport tcp 10.0.11.11:/disk/sdb/  
10.0.11.12:/disk/sdb/ 10.0.11.13:/disk/sdb/ force
```

where /disk/vdb are the mount point where the device has been mounted.

Then start the volume:

```
# gluster volume start gfile
```

You can get information about the glusterFS volume with the following command:

```
# gluster volume info
```

To mount the glusterFS volume on the client, you can do the following:

```
# mount -t glusterfs <server_ip>:/gfile <mount_point>
```

where `server_ip` is one of the nodes address

```
# mount -t glusterfs 10.0.11.12:/gfile /gfile
```

## 5. Installing and setting up Pacemaker and Corosync

It's important that our nodes know each other by address and hostname, so enter their details in `/etc/hosts` to avoid DNS lookups, as follows:

```
10.0.12.14 controller-01  
10.0.12.15 controller-02  
10.0.12.16 controller-03
```

Install the required packages on all nodes:

```
# apt-get update  
# apt-get -y install pacemaker corosync
```

Edit the `/etc/corosync/corosync.conf` file so the interface section matches the following:

```
interface {  
    # The following values need to be set based on your environment  
    ringnumber: 0  
    bindnetaddr: 10.0.12.0  
    mcastaddr: 226.94.1.1  
    mcastport: 5405  
}
```

By default, the `corosync` service is not set to start. To ensure it starts, edit the `/etc/default/corosync` service and set `START=yes`, as follows:

```
# sed -i 's/^START=no/START=yes/g' /etc/default/corosync
```



We now need to generate an authorization key to secure the communication between our hosts:  
on one of the nodes do the following:

```
# corosync-keygen
```

You will be asked to generate some random entropy by typing using the keyboard. If you are using an SSH session, rather than a console connection, you won't be able to generate the entropy using a keyboard. To do this remotely, launch a new SSH session, and in that new session, while the corosync-keygen command is waiting for entropy, run the following:

```
while /bin/true; do  
    dd if=/dev/urandom of=/tmp/100 bs=1024  
    count=100000;  
    for i in {1..10}; do  
        cp /tmp/100 /tmp/tmp_$i_$RANDOM;  
    done;  
    rm -f /tmp/tmp_* /tmp/100;  
done
```

When the corosync-keygen command has finished running and an authkey file has been generated, simply press *Ctrl* + *C* to stop this random entropy creation loop.

Then we copy the generated *authkey* file on the other nodes in our cluster, as follows:

```
scp /etc/corosync/authkey <node>:/etc/corosync/authkey
```

## 5.1 Starting the Pacemaker and Corosync services

We are now ready to start the services. On both nodes, issue the following commands:

```
# service corosync start  
# service pacemaker start
```

Pacemaker will probably fail: check the log (/etc/log/syslog) to find out that you have to configure some or disable STONITH (Shoot The Other Node In The Head) with the stonith-enabled option:

```
crm configure property stonith-enabled=false
```

Trying to start again pacemaker will fail again: in the log you will find the following message:

```
ERROR: read_config: We can only start Pacemaker from init if using version 1 of the  
Pacemaker plugin for Corosync. Terminating.
```

The error message is self-descriptive: edit (on all nodes) the corosync configuration file (/etc/corosync/corosync.conf) and change the pacemaker plugin version to 1, as follows:

```
service {  
    # Load the Pacemaker Cluster Resource Manager  
    ver:      1  
    name:     pacemaker  
}
```

Then restart the corosync service and start the pacemaker service:

```
# service corosync restart  
# service pacemaker start
```

To check that our services have started fine and our cluster is working, we can use the `crm_mon` command to query the cluster status, as follows:

```
# crm_mon -l
```

This will return output similar to the following where the important information includes the number of nodes configured, the expected number of nodes, and a list of our nodes that are online:

```
root@controller-01:~# crm_mon -l  
=====  
Last updated: Mon Jul 7 20:55:56 2014
```

```
Last change: Mon Jul  7 20:36:59 2014 via cibadmin on controller-01
Stack: openais
Current DC: controller-01 - partition with quorum
Version: 1.1.6-9971ebba4494012a93c03b40a2c58ec0eb60f50c
3 Nodes configured, 3 expected votes
0 Resources configured.
=====

Online: [ controller-02 controller-01 controller-03 ]
```

We can validate the configuration using the `crm_verify` command, as follows:

```
# crm_verify -L
```

## 5.2 Configure Openstack Services

### 5.2.1 Configure the Cluster Virtual IPs (VIP)

On the first node, we can now configure our services and set up the floating addresses that will be shared between the servers (controller nodes and network nodes).

In the following command, we've chosen 10.0.11.20 as the floating IP address for the controller nodes and 10.0.11.21 as the floating IP address for the network nodes.

To do this, we use the `crm` command again to configure these floating IP addresses:

```
# crm configure primitive Controller_VIP \
    ocf:heartbeat:IPaddr2 params ip=10.0.11.20 \
    cidr_netmask="255.255.255.0" op monitor interval=20s
```

```
# crm configure primitive Network_VIP \
    ocf:heartbeat:IPaddr2 params ip=10.0.11.21 \
```

```
cidr_netmask="255.255.255.0" op monitor interval="20s"
```

We can check the status of our cluster resources using `crm_mon`:

```
# crm_mon -l
=====
Last updated: Tue Jul  8 12:31:02 2014
Last change: Tue Jul  8 12:25:37 2014 via cibadmin on controller-01
Stack: openais
Current DC: controller-02 - partition with quorum
Version: 1.1.6-9971ebba4494012a93c03b40a2c58ec0eb60f50c
3 Nodes configured, 3 expected votes
2 Resources configured.
=====

Online: [ controller-02 controller-01 controller-03 ]

Controller_VIP      (ocf::heartbeat:IPaddr2): Started controller-01
Network_VIP (ocf::heartbeat:IPaddr2): Started controller-02
```

## 5.2.2 Keystone

Install Keystone and configure it appropriately, as if we are configuring a single host, taking care of the following aspects:

1. Edit the configuration file `/etc/keystone/keystone.conf` to ensure that Keystone uses the Galera MySQL cluster as backend database:

```
[database]
# The SQLAlchemy connection string used to connect to the database
connection = mysql://keystone:PASSWD@<VIP_GaleraCluster>:33306/keystone
```

2. Use the cluster floating IP (`Controller_VIP`) that you configured in 5.2.1 to create the Keystone service endpoint:

```
keystone endpoint-create --service-id=$(keystone service-list | awk '/ identity / {print $2}') --publicurl=http://<Controller_VIP>:5000/v2.0 --internalurl=http://<Controller_VIP>:5000/v2.0 --adminurl=http:// <Controller_VIP>:35357/v2.0
```

With Keystone running on this host, we should be able to query Keystone using both its own IP address and the floating IP from a client that has access to the OpenStack environment.

```
# export OS_USERNAME=admin  
# export OS_PASSWORD=PASSWORD  
# export OS_TENANT_NAME=admin  
# export OS_AUTH_URL=http://<Controller_VIP>:5000/v2.0/  
# keystone user-list
```

On the second node, controller2, install and configure Keystone; configured such that Keystone is pointing at the same database backend.

```
# apt-get update  
# apt-get install keystone python-mysqldb
```

Copy over the /etc/keystone/keystone.conf file from the first host, put it in place on the second node, and then restart the Keystone service. There is no further work required, as the database has already been populated with endpoints and users when the install was completed on the first node. Restart the service to connect to the database.

## Configure Pacemaker

With Keystone running on both nodes, we can now configure Pacemaker to take control of this service, so that we can ensure Keystone is running on the appropriate node when the other node fails. To do this, we first disable the upstart jobs for controlling Keystone services. To do this, we create upstart override files for this service (on both nodes).

Create `/etc/init/keystone.override` with just the keyword, manual, in and stop the Keystone service on both nodes:

```
# echo "manual" >> /etc/init/keystone.override
# service keystone stop
```

We now grab the **OCF (Open Cluster Format)** resource agent that is shell script or pieces of code that are able to control our Keystone service.

We must do this on both our nodes:

```
# mkdir -p /usr/lib/ocf/resource.d/openstack
# cd /usr/lib/ocf/resource.d/openstack
# wget https://raw.githubusercontent.com/madkiss/openstack-resource-agents/master/ocf/keystone
# chmod a+rx *
```

We can now configure Pacemaker to use this agent to control our Keystone service. To do this, we run the following command:

```
# crm configure primitive p_keystone ocf:openstack:keystone \
    params config="/etc/keystone/keystone.conf" \
    op monitor interval="30s" timeout="30s"
```

We can verify that we have our Pacemaker configured correctly, by issuing the following command: **crm\_mon -1**

We now have Keystone running on two separate nodes, where a node can fail and the service will still be available.

### 5.2.3 Glance

Install Glance and configure it appropriately, as if we are configuring a single host, taking care of the following aspects:

1. Edit the configuration file `/etc/glance/glance-api.conf` and `/etc/glance/glance-registry.conf` to ensure that Glance uses the Galera MySQL cluster as backend database:

```
[database]
# The SQLAlchemy connection string used to connect to the database
connection = mysql://glance:PASSWD@<VIP_GaleraCluster>:33306/glance
```

2. Configure the rabbitMQ cluster as messaging server in `/etc/glance/glance-registry.conf` as in the following example:

```
[DEFAULT]
...
notifier_strategy = rabbit
rpc_backend = rabbit
rabbit_hosts = 10.0.11.4:5672, 10.0.11.5:5672, 10.0.11.6:5672
rabbit_ha_queues = True
rabbit_user = guest
rabbit_retry_interval = 1
rabbit_retry_backoff = 2
rabbit_max_retries = 0
rabbit_password = PASSWD
rabbit_notification_exchange = glance
rabbit_notification_topic = notifications
```

3. Use the cluster floating IP (Controller\_VIP) that you configured in 5.2.1 to create the Glance service endpoint:

```
keystone endpoint-create --service-id=$(keystone service-list | awk '/ image / {print $2}') --publicurl=http://<Controller_VIP>:9292 --internalurl=http:// <Controller_VIP>:9292 --adminurl=http:// <Controller_VIP>:9292
```

With Glance running on this host, we should be able to query Glance using both its own IP address and the floating IP from a client that has access to the OpenStack environment.

```
# export OS_USERNAME=admin
# export OS_PASSWORD=PASSWORD
```

```
# export OS_TENANT_NAME=admin  
  
# export OS_AUTH_URL=http://<Controller_VIP>:5000/v2.0/  
  
# glance image-list
```

On the second node, controller-02, install and configure Glance:

```
# apt-get update  
  
# apt-get install glance python-glanceclient
```

Copy over the `/etc/glance/glance-api.conf` and `/etc/glance/glance-registry.conf` files from the first host, put it in place on the second node, and then restart the Glance service. There is no further work required, as the database has already been populated with endpoints and users when the install was completed on the first node. Restart the service to connect to the database.

Configure glance to use GlusterFS:

1. install the glusterFS client on both nodes:

```
# add-apt-repository ppa:semiosis/ubuntu-glusterfs-3.5  
  
# apt-get update  
  
# apt-get -y install glusterfs-common glusterfs-client
```

2. on both nodes mount the shared filesystem specifying the address of one of the glusterfs servers (in the next example the mount point is created as `/gfile`):

```
# mkdir /gfile  
  
# mount -t glusterfs 10.0.11.11:/gfile /gfile
```

3. create the directory glance and its sub-dirs in the mounted path:

```
# mkdir -p /gfile/glance/images  
  
# mkdir /gfile/glance/image-cache  
  
# mkdir /gfile/scrubber
```

4. on both nodes modify the glance configuration:



edit the file `/etc/glance/glance-api.conf` as follows:

```
[DEFAULT]

...

# ===== Filesystem Store Options =====

# Directory that the Filesystem backend store
# writes image data to
filesystem_store_datadir = /gfile/glance/images/

...
```

edit the file `/etc/glance/glance-cache.conf` as follows:

```
[DEFAULT]

...

# Directory that the Image Cache writes data to
image_cache_dir = /gfile/glance/image-cache/

...
```

edit the file `/etc/glance/glance-scrubber.conf` as follows:

```
[DEFAULT]

...

# Directory that the scrubber will use to remind itself of what to delete
# Make sure this is also set in glance-api.conf
scrubber_datadir = /gfile/glance/scrubber

...
```

With Glance running on both nodes, we can now configure Pacemaker to take control of this service, so that we can ensure Glance is running on the appropriate node when the other node fails. To do this, we first disable the upstart jobs for controlling Glance services. To do this, we create upstart override files for this service (on both nodes).

Create `/etc/init/glance-api.override` and `/etc/init/glance-registry.override` with just the keyword, manual, in and stop the Glance services on both nodes:

```
# echo "manual" >> /etc/init/glance-api.override
# echo "manual" >> /etc/init/glance-registry.override
# service glance-api stop
# service glance-registry stop
```

We now grab the **OCF (Open Cluster Format)** resource agent that is shell script or pieces of code that are able to control our Glance services.

We must do this on both our nodes:

```
# cd /usr/lib/ocf/resource.d/openstack
# wget https://raw.githubusercontent.com/madkiss/openstack-resource-agents/master/ocf/glance-registry
# wget https://raw.githubusercontent.com/madkiss/openstack-resource-agents/master/ocf/glance-api
# chmod a+rx *
```

Configure Pacemaker to use this agent to control our Glance services. To do this, we run the following command:

```
# crm configure primitive p_glance-api ocf:openstack:glance-api \
    params config="/etc/glance/glance-api.conf" os_password="OpenStack" \
    os_username="admin" os_tenant_name="admin" \
    os_auth_url="http://10.0.11.20:5000/v2.0/" \
    op monitor interval="300s" timeout="120s"
```

## Testing & Verifying

Try to upload an image.

After exporting the credentials, run the following command:

```
# glance image-create --name ubuntu-precise-server-amd64 --disk-format qcow2 --
container-format bare --copy-from https://cloud-images.ubuntu.com/precise/current/precise-server-cloudimg-amd64-disk1.img
```

you will get an output like the following:

```
+-----+-----+
| Property      | Value                                |
+-----+-----+
| checksum      | None                                |
| container_format | bare                                |
| created_at    | 2014-07-11T12:46:47                 |
| deleted       | False                               |
| deleted_at    | None                                |
| disk_format   | qcow2                               |
| id            | 487e05aa-8978-4feb-a05f-57628452c408 |
| is_public     | False                               |
| min_disk      | 0                                    |
| min_ram       | 0                                    |
| name          | ubuntu-precise-server-amd64         |
| owner         | cefedd83bd7748e391a54f9fe130fdd0    |
| protected     | False                               |
| size          | 260833792                           |
| status        | queued                              |
| updated_at    | 2014-07-11T12:46:47                 |
| virtual_size  | None                                |
+-----+-----+
```

Then you can check the status of the upload with the command “glance image-list”:

```
# glance image-list
+-----+-----+-----+-----+
+-----+-----+-----+-----+
| ID                                     | Name                                     | Disk Format |
| Container Format | Size          | Status      |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
| 487e05aa-8978-4feb-a05f-57628452c408 | ubuntu-precise-server-amd64           | qcow2       |
| bare                               | 260833792 | saving |
+-----+-----+-----+-----+
+-----+-----+-----+-----+

# glance image-list
+-----+-----+-----+-----+
+-----+-----+-----+-----+
| ID                                     | Name                                     | Disk Format |
| Container Format | Size          | Status      |
+-----+-----+-----+-----+
```

```
+-----+-----+-----+
+-----+-----+-----+
| 487e05aa-8978-4feb-a05f-57628452c408 | ubuntu-precise-server-amd64 | qcow2
| bare                | 260833792 | active |
+-----+-----+-----+
+-----+-----+-----+
```

If the installation and configuration has been correctly carried out, the image status will be active in the end and the directory `/gfile/glance/images` will be populated with the uploaded image:

```
# ll /gfile/glance/images
total 254721
drwxr-xr-x 2 glance glance      49 Jul 11 12:47 ./
drwxr-xr-x 5 glance glance      52 Jul 11 12:33 ../
-rw-r----- 1 glance glance 260833792 Jul 11 12:47 487e05aa-8978-4feb-a05f-
57628452c408
```

## 5.2.4 Nova

On controller-01 install Nova (`nova-api nova-cert nova-conductor nova-consoleauth nova-novncproxy nova-scheduler`) and configure it appropriately, as if we are configuring a single host, taking care of the following aspects:

1. Edit the configuration file `/etc/nova/nova.conf` to ensure that Nova uses the Galera MySQL cluster as backend database:

```
[database]
# The SQLAlchemy connection string used to connect to the database
connection = mysql://nova:PASSWD@<VIP_GaleraCluster>:33306/nova
```

2. Configure the rabbitMQ cluster as messaging server in `/etc/nova/nova.conf` as in the following example:

```
[DEFAULT]
...
rpc_backend = rabbit
rabbit_hosts = 10.0.11.4:5672, 10.0.11.5:5672, 10.0.11.6:5672
rabbit_ha_queues = True
rabbit_user = guest
```

```
rabbit_retry_interval = 1  
rabbit_retry_backoff = 2  
rabbit_max_retries = 0  
rabbit_password = PASSWD
```

3. Use the cluster floating IP (Controller\_VIP) that you configured in 5.2.1 to create the Nova service endpoint:

```
# keystone endpoint-create --service-id=$(keystone service-list | awk '/ compute / {print $2}') --publicurl=http://<Controller_VIP>:8774/v2/%(tenant_id)s --internalurl=http://  
<Controller_VIP>:8774/v2/%(tenant_id)s --adminurl=http://  
<Controller_VIP>:8774/v2/%(tenant_id)s
```

With Nova running on this host, we should be able to query Nova using both its own IP address and the floating IP from a client that has access to the OpenStack environment.

```
# export OS_USERNAME=admin  
# export OS_PASSWORD=PASSWORD  
# export OS_TENANT_NAME=admin  
# export OS_AUTH_URL=http://<Controller_VIP>:5000/v2.0/  
# nova list
```

On the second node, controller-02, install and configure Nova:

```
# apt-get install -y nova-api nova-cert nova-conductor nova-consoleauth nova-novncproxy  
nova-scheduler python-novaclient
```

Copy over the `/etc/nova/nova.conf` files from the first host, put it in place on the second node, and then restart the Nova services. There is no further work required, as the database has already been populated with endpoints and users when the install was completed on the first node. Restart the services to connect to the database.

With Nova running on both nodes, we can now configure Pacemaker to take control of its services, so that we can ensure Nova is running on the appropriate node when the other node fails. To do this, we first disable the upstart jobs for controlling Nova services. To do this, we create upstart override files for this service (on both nodes).

Create in /etc/init/ the files nova-api.override, nova-cert.override, nova-consoleauth.override, nova-novncproxy.override, nova-scheduler.override with just the keyword, manual, in and stop the Nova services on both nodes:

```
# echo "manual" >> /etc/init/nova-api.override
# echo "manual" >> /etc/init/nova-cert.override
# echo "manual" >> /etc/init/nova-consoleauth.override
# echo "manual" >> /etc/init/nova-novncproxy.override
# echo "manual" >> /etc/init/nova-scheduler.override
# service nova-api stop
# service nova-cert stop
# service nova-consoleauth stop
# service nova-novncproxy stop
# service nova-scheduler stop
```

Note that no resource agent is available for nova-conductor service: in order to avoid manual start/stop of this service (that can run on both nodes) you can patch the nova-scheduler resource agent as follows:

```
nova_scheduler_start() {
    ...
    # added to manage nova-conductor together with nova-scheduler
    service nova-conductor start
    return $OCF_SUCCESS
}

nova_scheduler_stop() {
    local rc
    local pid
    ...
    # added to manage nova-conductor together with nova-scheduler
    service nova-conductor stop
    return $OCF_SUCCESS
}
```

We now grab the **OCF (Open Cluster Format)** resource agent that is shell script or pieces of code that are able to control our Nova services.

We must do this on both our nodes:

```
# cd /usr/lib/ocf/resource.d/openstack
# wget https://raw.githubusercontent.com/madkiss/openstack-resource-agents/master/ocf/nova-api
# wget https://raw.githubusercontent.com/madkiss/openstack-resource-agents/master/ocf/nova-cert
# wget https://raw.githubusercontent.com/madkiss/openstack-resource-agents/master/ocf/nova-scheduler
# wget https://raw.githubusercontent.com/madkiss/openstack-resource-agents/master/ocf/nova-consoleauth
# wget https://raw.githubusercontent.com/madkiss/openstack-resource-agents/master/ocf/nova-novnc
# chmod a+rx *
```

Configure Pacemaker to use this agent to control our Nova services. To do this, we run the following command:

```
# crm configure primitive p_nova_api ocf:openstack:nova-api \
    params config="/etc/nova/nova.conf" os_password="OpenStack" \
    os_username="admin" os_tenant_name="admin" \
    keystone_get_token_url="http://10.0.11.20:5000/v2.0/tokens" \
    url="http://10.0.11.20:8774/v2/" \
    op monitor interval="10s" timeout="10s"

# crm configure primitive p_nova_cert ocf:openstack:nova-cert \
    params config="/etc/nova/nova.conf" \
    op monitor interval="10s" timeout="10s"

# crm configure primitive p_nova_consoleauth ocf:openstack:nova-consoleauth \
    params config="/etc/nova/nova.conf" \
    op monitor interval="30s" timeout="30s"

# crm configure primitive p_nova_novnc ocf:openstack:nova-novnc \
    params config="/etc/nova/nova.conf" \
    op monitor interval="30s" timeout="30s"

# crm configure primitive p_nova_scheduler ocf:openstack:nova-scheduler \
    params config="/etc/nova/nova.conf" \
```

```
op monitor interval="30s" timeout="30s"
```

### Grouping Controller resources:

We need to create a service group to ensure that virtual IP is linked to the API services resources:

```
# crm configure group g_controller \  
    Controller_VIP p_keystone p_glance-registry p_glance-api p_nova_api \  
    p_nova_cert p_nova_consoleauth p_nova_novnc p_nova_scheduler \  
    meta ordered=true
```

### Setting resources constraints:

```
# crm configure location l_controller_1 g_controller -inf: controller-03  
# crm configure location l_controller_2 g_controller 100: controller-01  
# crm configure location l_controller_3 g_controller 100: controller-02
```

## 5.2.5 Neutron

Let's proceed with the installation of Neutron services on the dedicated controller nodes: controller-03 (master), controller-02 (backup).

Install both nodes as if you would install a single node.

Download the pacemaker resource agents in the directory /usr/lib/ocf/resource.d/openstack (create it if it doesn't exist):

```
# wget https://raw.githubusercontent.com/madkiss/openstack-resource-agents/master/ocf/neutron-server  
# wget https://raw.githubusercontent.com/madkiss/openstack-resource-agents/master/ocf/neutron-agent-dhcp  
# wget https://raw.githubusercontent.com/madkiss/openstack-resource-agents/master/ocf/neutron-agent-l3  
# wget https://raw.githubusercontent.com/madkiss/openstack-resource-agents/master/ocf/neutron-metadata-agent
```

Note: the monitor function of the RA uses the command curl, therefore install it if it is not present on the node:



```
# apt-get install -y curl
```

Patch the neutron-server resource agent to fix a bug in the plugin config parameter: edit file `/usr/lib/ocf/resource.d/openstack/neutron-server` and change line 104 as follows:

```
104c104
< <parameter name="plugin config" unique="0" required="0">
---
> <parameter name="plugin_config" unique="0" required="0">
```

Then patch the l3 agent resource agent to fix a bug with the rabbitmq default port: edit file `/usr/lib/ocf/resource.d/openstack/neutron-agent-l3` and change line 40 as follows:

```
40c40
< OCF_RESKEY_neutron_server_port_default="9696"
---
> OCF_RESKEY_neutron_server_port_default="5672"
```

Change also line 98 to fix a bug with the RA parameter "plugin\_config":

```
98c98
< <parameter name="plugin config" unique="0" required="0">
---
> <parameter name="plugin_config" unique="0" required="0">
```

### **Important:**

Add the following instructions after lines 240, 246 and 312 in order to change the hostname of the node (this is **mandatory**: both nodes must have the same hostname since the Networking scheduler will be aware of one node, for example a virtual router attached to a single L3 node):

```
240a241,245
>
>     hostname network-controller
>     #Marant: temporary patch - restart plugin-openvswitch-agent
>     service plugin-openvswitch-agent restart
>
246a252,256
```

```
> # Sometimes VM instance can not access external network after pacemaker start the l3-agent
> # you should restart the neutron-openvswitch-agent after the l3-agent start for few seconds.
> sleep 10
> service plugin-openvswitch-agent restart
>
312a323,328
>
>
> # restore old hostname
> hostname $(cat /etc/hostname)
> # stop the agent ml2
> service plugin-openvswitch-agent stop
```

Then configure the services in pacemaker:

```
# crm configure primitive p_neutron-server ocf:openstack:neutron-server \
    params config="/etc/neutron/neutron.conf" \
    plugin_config="/etc/neutron/plugins/ml2/ml2_conf.ini" \
    os_password="OpenStack" os_username="admin" os_tenant_name="admin" \
    keystone_get_token_url="http://10.0.11.20:5000/v2.0/tokens" \
    url="http://10.0.11.21:9696/" \
    op monitor interval="30s" timeout="30s"
```

```
# crm configure primitive p_neutron-l3-agent ocf:openstack:neutron-agent-l3 params
config="/etc/neutron/neutron.conf" plugin_config="/etc/neutron/l3_agent.ini" op
monitor interval="30s" timeout="30s"
```

```
# crm configure primitive p_neutron-agent-dhcp ocf:openstack:neutron-agent-dhcp
params config="/etc/neutron/neutron.conf"
plugin_config="/etc/neutron/dhcp_agent.ini" op monitor interval="30s" timeout="30s"
```

```
# crm configure primitive p_neutron-metadata-agent \
    ocf:openstack:neutron-metadata-agent \
    params config="/etc/neutron/neutron.conf" \
```

```
agent_config="/etc/neutron/metadata_agent.ini" \  
op monitor interval="30s" timeout="30s"
```

Create the group `g_network` to cluster the networking services:

```
# crm configure group g_network Network_VIP p_neutron-server p_neutron-l3-agent  
p_neutron-agent-dhcp p_neutron-metadata-agent meta ordered=true
```

and add the constraints in order to locate the `g_network` group on the two nodes, `controller-02`, and `controller-03`:

```
# crm configure location l_networknode_1 g_network -inf: controller-01  
# crm configure location l_networknode_2 g_network 100: controller-02  
# crm configure location l_networknode_3 g_network 100: controller-03
```

## 5.2.6 Horizon

Install the dashboard packages on both nodes:

```
# apt-get install -y apache2 memcached libapache2-mod-wsgi openstack-dashboard
```

and edit the file `/etc/openstack-dashboard/local_settings.py` to configure the dashboard as if you configure it in a single-host installation.

Then configure the `apache2` service in `pacemaker` using the following command:

```
# crm configure primitive p_apache ocf:heartbeat:apache \  
    params configfile="/etc/apache2/apache2.conf" \  
    statusurl="http://localhost/server-status" \  
    op monitor interval="40s"
```

Insert the resource inside the `g_controller` group in order to start `apache2` with all the other controller services (remember that the order inside the group is important; therefore put the resource `p_apache` after the `Controller_VIP` resource).

To do this edit the `crm` configuration using the command **“`crm configure edit`”**: this command opens an editor section that allows to modify the configuration. Just add `p_apache` to the line starting with `“group g_controller”` as follows:

```
...  
group g_controller Controller_VIP p_apache p_keystone p_glance-registry p_glance-  
api p_nova_api p_nova_cert p_nova_consoleauth p_nova_novnc p_nova_scheduler \  
...
```

Connect to the dashboard using the Controller IP and log-in.

Then ensure that the secret key that has been generated by horizon at the first login is the same on the two controller nodes in order to avoid session closing when the dashboard starts on the second node. To do this check which node the dashboard is running on (use **`crm_mon -1`**) and copy the secret key file from this node to the other.

For example, if `apache2` is currently running on `controller-01`, copy the secret key file from `controller-01` to `controller-02` as follows:

```
controller-02# scp controller-01:/var/lib/openstack-dashboard/secret_key  
/var/lib/openstack-dashboard/secret_key  
  
controller-02# chown horizon:horizon /var/lib/openstack-dashboard/secret_key
```

## 6. Riferimenti

<http://www.gluster.org/>

<http://clusterlabs.org/>

<http://galeracluster.com/>

<https://www.rabbitmq.com/clustering.html>

<http://haproxy.1wt.eu/>

