Deepak Mane / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com
Vol. 3, Issue 4, Jul-Aug 2013, pp.269-277

Building A High Availability - Openstack

Deepak Mane
GCP – IT Performance Management Tata Consultancy Services Hadapsar, Pune INDIA

ABSTRACT

High Availability is an important component to provide Zero downtime , zero outage coverage for Openstack components . it also provides automated failover and switchover .

Openstack is a private cloud platform has lack of inbuilt high availability functions this can be implemented by opensource solution by Pacemaker and Corosync software.

This paper provided clear concepts of High availability requirements, solution approach , Openstack components , detailed implementation approach using PCS shell and LCMC. It also summarizes various tools to manage pacemaker and Corosync clusters

KEYWORDS: Cloud Computing , Openstack, High Availability, Corosync , Pacemaker , PCS, keystone, nova , glance, LCMC

.

I. Requirement for High Availability

High Availability systems, fundamentally, seek to minimize two things:

- System downtime — the unavailability of a user-facing service beyond a specified maximum amount of time, and
- Data loss — the accidental deletion or destruction of data.
- Budget - Each high availability solution has an associated cost. The cost for the solution must be compared to the benefit achieved for your business. When asked about a high availability solution, most customers will say that they want continuous availability with zero downtime. While this is technically possible, the cost of the protection offered by the solution may be too great.
- Uptime requirements- Up-time requirements refers to the total amount of time that the system is available for end-use applications. The value is stated as a percent of total scheduled working hours.
- Outage coverage -What kind of outage is the business trying to protect against? Backup window reduction, planned maintenance, unplanned outages, or site disasters are events to consider when choosing a high availability solution.
- Resilience requirements -The business must identify what it is that needs to be protected when the system hosting the application experiences an outage. The resilience requirements are the set of applications, data and system environments required to be preserved across an outage of the production system. These entities remain available through a failover even when the system currently hosting them experiences an outage.
- Automated failover and switchover -The business must define how much control is given up to automation during unplanned outages
- Distance requirements -Distance between systems, or geographic dispersion, has benefits but is gated by physical and practical limits. For a disaster recovery solution, there are always benefits in having geographic dispersion between the systems. Typically, the greater the distance between the systems, the greater the protection you will have from area wide disasters. However, this distance will come with application environment impacts.
- Number of backup systems -Different data resilience technologies offer differing numbers of possible backup systems and copies of application data.
- Access to a secondary copy of the data - Different data resilience technologies have different restrictions to the backup data set. Access to the backup data set requirements indicates the level of access that is required to secondary copies of the data for other work activity off-loaded from primary copies, such as saves and queries/reports. You should consider the frequency, duration, and what type of access is needed for the backup copy of the data.
- System performance - Implementing high availability may have performance implications. The requirements of the business may determine what data resilience technology is required.
- Data resilience method comparison - This table provides a brief description of the major characteristics of the solution that generates a copy of the data onto auxiliary storage.

It is important to understand that most high availability systems can guarantee protection against these issues only in the face of a single failure event. They are also expected to protect against cascading failures, where an originally singular failure deteriorates into a series of consequential failures High-availability systems typically achieve uptime of 99.99% or more, meaning less than roughly an hour of cumulative downtime per year. From this, it follows that highly-available systems are generally expected to keep recovery times in the face of a failure on the order of 1-2 minutes, sometimes significantly less.

II. High Availability Solution – Openstack

Openstack infrastructure high availability relies on the Pacemaker cluster stack, the stateof-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 and 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 Pacemaker RAs (such as those managing MySQL databases or virtual IP addresses), existing third-party RAs (such as for RabbitMQ), and native Openstack RAs (such as those managing the Openstack Identity and Image

We will provide detailed solution approach for High availability – Openstack

III. Openstack – Introduction

Openstack is a collection of open source software projects that enterprises/service providers can use to setup and run their cloud compute and storage infrastructure. Rackspace and NASA are the key initial contributors to the stack. Rackspace contributed their "Cloud Files" platform (code) to power the Object Storage part of the Openstack, while NASA contributed their "Nebula" platform (code) to power the Compute part. Openstack consortium has managed to have more than 150 members including Canonical, Dell, Citrix etc.

- Nova - Compute Service
- Swift - Storage Service
- Glance - Imaging Service
- Keystone - Identity Service
- Horizon - UI Service

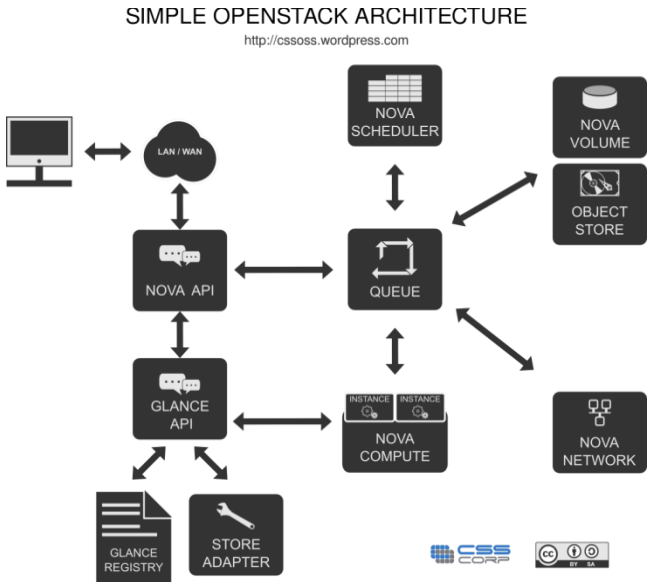

Figure 1: Simple Openstack Architecture

3.1 Openstack Compute Service (Nova)

Nova is the Computing Fabric controller for the Openstack Cloud. All activities needed to support the life cycle of instances within the Openstack cloud are handled by Nova. This makes Nova a Management Platform that manages compute resources, networking, authorization, and scalability needs of the Openstack cloud. But, Nova does not provide any virtualization capabil- ities by itself; instead, it uses libvirt API to interact with supported hypervisors. Nova exposes all its capabilities through a web services API that is compatible with the EC2 API of Amazon Web Services.

Functions and features of Nova are

• Instance life cycle management
• Management of compute resources
• Networking and Authorization
• REST-based API
•Asynchronous eventually consistent communication
• Hypervisor agnostic: support for Xen, XenServer/XCP, KVM, UML, VMware vSphere and Hyper-V

3.2 Openstack Image Service (Glance)

Openstack Imaging Service is a lookup and retrieval system for virtual machine images. It can be configured to use any one of the following storage backbends:

• Local filesystem (default)
• Openstack Object Store to store images
• S3 storage directly

• S3 storage with Object Store as the intermediate for S3 access.
• HTTP (read-only)

Functions and features of Glance are

• Provides imaging service

3.2 Openstack Storage requirement (Swift)

Openstack Imaging Service is a lookup and retrieval system for virtual machine images. It can be configured to use any one of the following storage backbends:

• Local filesystem (default)
• Openstack Object Store to store images
• S3 storage directly
• S3 storage with Object Store as the intermediate for S3 access.
• HTTP (read-only)
• Provides imaging service

Swift provides a distributed, eventually consistent virtual object store for Openstack. It is analogous to Amazon Web Services - Simple Storage Service (S3). Swift is capable of storing billions of objects distributed across nodes. Swift has built-in redundancy and failover management and is capable of archiving and media streaming. It is extremely scalable in terms of both size (several petabytes) and capacity (number of objects).

Functions and Features

• Storage of large number of objects
• Storage of large sized objects
• Data Redundancy
• Archival capabilities - Work with large datasets
• Data container for virtual machines and cloud apps
• Media Streaming capabilities
• Secure storage of objects
• Backup and archival
• Extreme scalability

3.3 Openstack Identity Service (Keystone)

Keystone provides identity and access policy services for all components in the Openstack family. It implements its own REST based API (Identity API). It provides authentication and authorization for all components of Openstack including (but not limited to) Swift, Glance, Nova. Authentication verifies that a request actually comes from who it says it does. Authorization is verifying whether the authenticated user has access to the services he/she is requesting for.

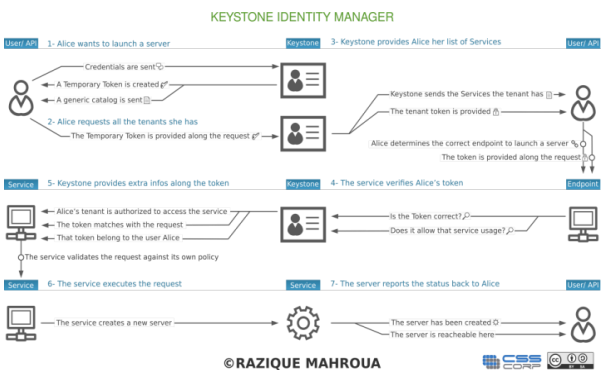

Figure 2: Keystone process

Keystone provides two ways of authentication. One is username/password based and the other is token based. Apart from that, keystone provides the following services:

• Token Service (that carries authorization information about an authenticated user)

• Catalog Service (that contains a list of available services at the users' disposal)

• Policy Service (that let's keystone manage access to specific services by specific users or groups).

3.4 Openstack Administrative Web-Interface (Horizon)

Horizon the web based dashboard can be used to manage /administer Openstack services. It can be used to manage instances and images, create keypairs, attach volumes to instances, manipulate Swift containers etc. Apart from this, dashboard even gives the user access to instance console and can connect to an instance through VNC. Overall, Horizon features the following:

• Instance Management - Create or terminate instance, view console logs and connect through VNC, Attaching volumes, etc.
• Access and Security Management - Create security groups, manage keypairs, assign floating IPs, etc.
• Flavor Management - Manage different flavors or instance virtual hardware templates.
• Image Management - Edit or delete images.
• View service catalog.
• Manage users, quotas and usage for projects.
• User Management - Create user, etc.
• Volume Management - Creating Volumes and snapshots.
• Object Store Manipulation - Create, delete containers and objects.
• Downloading environment variables for a project.

IV. High Availability Solution Components

4.1 Pacemaker

Pacemaker is a cluster resource manager. It achieves maximum availability for your cluster services (aka. resources) by detecting and recovering from node and resource-level failures by making use of the messaging and membership capabilities provided by your preferred cluster infrastructure (either Corosync or Heartbeat).

Pacemaker's key features include:

- Detection and recovery of node and service-level failures
- Storage agnostic, no requirement for shared storage

- Resource agnostic, anything that can be scripted can be clustered
- Supports STONITH for ensuring data integrity
- Supports large and small clusters
- Supports both quorate and resource driven clusters
- Supports practically any redundancy configuration
- Automatically replicated configuration that can be updated from any node
- Ability to specify cluster-wide service ordering, colocation and anti-colocation
- Support for advanced service types
  - Clones: for services which need to be active on multiple nodes
  - Multi-state: for services with multiple modes (eg. master/slave, primary/secondary)
- Unified, scriptable, cluster management tools.

4.1.1 Pacemaker - Internal components

Pacemaker itself is composed of four key components (illustrated below in the same color scheme as the previous diagram):

- CIB (aka. Cluster Information Base)
- CRMd (aka. Cluster Resource Management daemon)
- PEngine (aka. PE or Policy Engine)
- STONITHd

The CIB uses XML to represent both the cluster's configuration and current state of all resources in the cluster. The contents of the CIB are automatically kept in sync across the entire cluster and are used by the PEngine to compute the ideal state of the cluster and how it should be achieved.

This list of instructions is then fed to the DC (Designated Co-coordinator). Pacemaker centralizes all cluster decision making by electing one of the CRMd instances to act as a master. Should the elected CRMd process, or the node it is on, fail… a new one is quickly established.

The DC carries out the PEngine's instructions in the required order by passing them to either the LRMd (Local Resource Management daemon) or CRMd peers on other nodes via the cluster messaging infrastructure (which in turn passes them on to their LRMd process).

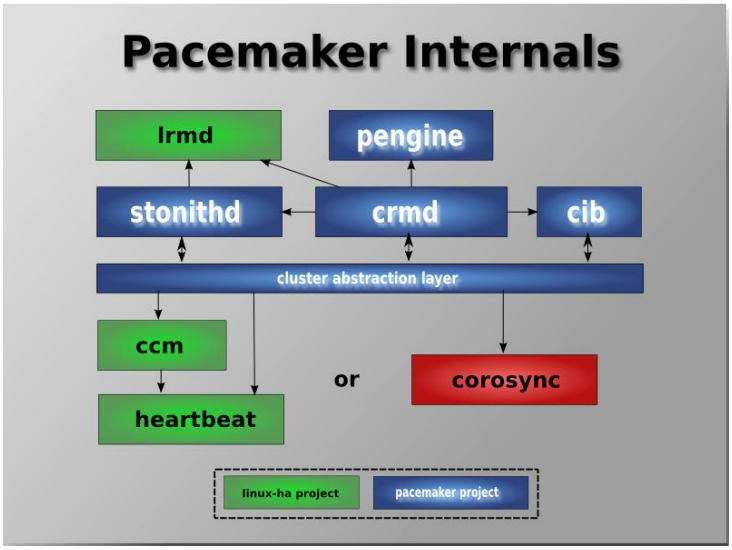

Figure 3: Pacemaker internals

4.2 Corosync – Introduction

The **Corosync Cluster Engine** is a group communication system with additional features for implementing high availability within applications. The project provides four C programming interfaces features:

- A closed process group communication model with virtual synchrony guarantees for creating replicated state machines.
- A simple availability manager that restarts the application process when it has failed.
- A configuration and statistics in-memory database that provide the ability to set, retrieve, and receive change notifications of information.
- A quorum system that notifies applications when quorum is achieved or lost.

4.2.1 Corosync Architecture

The software is composed of an executive binary which uses a client-server communication model between libraries and service engines. Loadable modules, called service engines, are loaded into the Corosync Cluster Engine and use the services provided by the Corosync Service Engine internal API.

The services provided by the Corosync Service Engine internal API are:

- An implementation of the Totem Single Ring Ordering and Membership [1] protocol providing the Extended Virtual Synchrony model [2] for messaging and membership.
- The croup high performance shared memory IPC system.[3]
- An object database that implements the in memory database model.

- Systems to route IPC and Totem messages to the correct service engines.

Additionally Corosync provides several default service engines that are used via C Application Programming Interfaces:

- cpg - Closed Process Group
- sam - Simple Availability Manager
- confdb - Configuration and Statistics database
- quorum - Provides notifications of gain or loss of quorum

4.3 Resource Agents

A resource agent is a standardized interface for a cluster resource. In translates a standard set of operations into steps specific to the resource or application, and interprets their results as success or failure.

Resource Agents have been managed as a separate Linux-HA sub-project since their 1.0 release, which coincided with the Heartbeat 2.99 release. Previously, they were a part of the then-monolithic Heartbeat project, and had no collective name. Later, the Linux-HA Resource Agents and the RHCS Resource Agents sub-projects have been merged. The joint upstream repository is now https://github.com/ClusterLabs/resource-agents

Pacemaker supports three types of Resource Agents,

- LSB Resource Agents,
- OCF Resource Agents,
- legacy Heartbeat Resource Agents

4.3.1 Supported Operations

Operations which a resource agent my perform on a resource instance include:

- *start*: enable or start the given resource
- *stop*: disable or stop the given resource
- *monitor*: check whether the given resource is running (and/or doing useful work), return status as *running* or *not running*
- *validate-all*: validate the resource's configuration
- *meta-data*: return information about the resource agent itself (used by GUIs and other management utilities, and documentation tools)

V. High Availability – Architecture type

The most common size for an HA cluster is a two-node a cluster, since that is the minimum required to provide redundancy, but many clusters consist of many more, sometimes dozens of nodes. Such configurations can sometimes be categorized into one of the following models:

- Active/active — Traffic intended for the failed node is either passed onto an existing node or load balanced across the remaining nodes. This is usually only possible when the nodes utilize a homogeneous software configuration.
- Active/passive — provides a fully redundant instance of each node, which is only brought online when its associated primary node fails. This configuration typically requires the most extra hardware.
- N+1 — provides a single extra node that is brought online to take over the role of the node that has failed. In the case of heterogeneous software configuration on each primary node, the extra node must be universally capable of assuming any of the roles of the primary nodes it is responsible for. This normally refers to clusters which have multiple services running simultaneously; in the single service case, this degenerates to active/passive.
- N+M — in cases where a single cluster is managing many services, having only one dedicated failover node may not offer sufficient redundancy. In such cases, more than one (M) standby servers are included and available. The number of standby servers is a tradeoff between cost and reliability requirements.
- N-to-1 — allows the failover standby node to become the active one temporarily, until the original node can be restored or brought back online, at which point the services or instances must be failed-back to it in order to restore high availability.
- N-to-N — A combination of active/active and N+M clusters, N to N clusters redistribute the services, instances or connections from the failed node among the remaining active nodes, thus eliminating

VI. Configuration tools

To configure Pacemaker and Corosync there are 2 types of configuration tools supported

- Command line tools
- Gui Tools

6.1 Command line interfaces

crmsh – The original configuration shell for Pacemaker. Written and maintained by SUSE, it may be used either as an interactive shell with tab completion, for single commands directly on the shell's commands line or as batch mode scripting tool.

pcs – An alternate vision for a full cluster lifecycle configuration shell and web based GUI. Handles everything from cluster installation through to resource configuration and status.

6.2 GUI tools

Pygui – The original GUI for Pacemaker written in Python by IBM China. Mostly deprecated on SLES in favor of Hawk

Hawk – Hawk is a web-based GUI for managing and monitoring Pacemaker HA clusters. It is generally intended to be run on every node in the cluster, so that you can just

point your web browser at any node to access it. It is documented as part of the SUSE Linux Enterprise High Availability Extension documentation

LCMC – The Linux Cluster Management Console (LCMC) is a GUI with an innovative approach for representing the status of and relationships between cluster services. It uses SSH to let you install, configure and manage clusters from your desktop.

pcs – An alternate vision for a full cluster lifecycle configuration shell and web based GUI. Handles everything from cluster installation through to resource configuration and status.

6.3 Others Add-on

booth – The Booth cluster ticket manager extends Pacemaker to support geographically distributed clustering. It does this by managing the granting and revoking of 'tickets' which authorizes one of the cluster sites, potentially located in geographically dispersed locations, to run certain resources.

VII. Implementation High Availability using PCS

PCS Shell - PCS will continue the tradition of having a regression test suite and discoverable 'ip'-like hierarchical "menu" structure, however unlike the shell we may end up not adding interactivity.

Both projects are far from complete, but so far PCS can:
- Create Corosync/pacemaker clusters from scratch
- Add simple resources and add constraints
- Create/Remove resource groups
- Set most pacemaker configuration options
- Start/Stop pacemaker/Corosync
- Get basic cluster status

7.1 Adding Openstack resources in Corosync and pacemaker resources

7.1.1 PCS -installation

$ yum install -y pcs

7.1.2 Setup

Start and enable the daemon by issuing the following commands on each node.

systemctl start pcsd.service
systemctl enable pcsd.service

While pcs can be used locally without setting up these user accounts, this tutorial will make use of these remote access commands, so we will set a password for the *hacluster* user. Its probably best if password is consistent across all the nodes.

As *root*, run:

passwd hacluster
password:

7.1.3 Configuring Corosync

Using pcs with the pcs daemon greatly simplifies this process by generating *corosync.conf* across all the nodes in the cluster with a single command. The only thing required to achieve this is to authenticate as the pcs user *hacluster* on one of the nodes in the cluster

Execute this commands on node-1 and node-2
pcs cluster auth node-1 node-2
Username: hacluster
Password:
node-1: Authorized
node-2: Authorized

pcs cluster setup mycluster node-1 node-2
node-1: Succeeded
node-2: Succeeded

7.1.4 Start the Cluster

Now that corosync is configured, it is time to start the cluster. The command below will start corosync and pacemaker on both nodes in the cluster. If you are issuing the start command
pcs cluster start --all
node-1: Starting Cluster...
node-2: Starting Cluster..
or
An alternative to using the *pcs cluster startall* command is to issue either of the below commands on each node in the cluster by hand.

pcs cluster start
Starting Cluster...
or
systemctl start corosync.service
systemctl start pacemaker.service

7.1.5 Adding keystone resource to pacemaker

You may now proceed with adding the Pacemaker configuration for Keystone resource.Connect to the Pacemaker cluster with pcs resource create and add the following cluster resources:

pcs resource create p_keystone ocf:openstack:keystone \
params config="/etc/keystone/keystone.conf" os_password="secret"
os_username="admin" os_tenant_name="admin"
os_auth_url="http://192.168.42.
103:5000/v2.0/" \
op monitor interval="30s" timeout="30s

7.1.6 Adding glance resources to pacemaker

pcs resource create p_glance-api ocf:openstack:glance-api \

params config="/etc/glance/glance-api.conf" os_password="secrete"
os_username="admin" os_tenant_name="admin" os_auth_url="http://192.168.42.103:5000/v2.0/" \
op monitor interval="30s" timeout="30s"
pcs resource create p_glance-registry ocf:openstack:glance-registry \
params config="/etc/glance/glance-registry.conf" os_password="secrete"
os_username="admin" os_tenant_name="admin" os_auth_url="http://192.168.42.103:5000/v2.0/" \
op monitor interval="30s" timeout="30s

7.1.7 Adding Cinder resources to pacemaker

pcs resource create p_cinder-api ocf:openstack:cinder-api \
params config="/etc/cinder/api-paste.conf" os_password="secrete"
os_username="admin" os_tenant_name="admin" os_auth_url="http://192.168.42.103:5000/v2.0/" \
op monitor interval="30s" timeout="30s"
pcs resource create p_cinder-volume ocf:openstack:cinder-volume \
params config="/etc/cinder/cinder.conf" os_password="secrete"
os_username="admin" os_tenant_name="admin" os_auth_url="http://192.168.42.103:5000/v2.0/" \
op monitor interval="30s" timeout="30s
pcs resource create p_cinder-schedule ocf:openstack:cinder-schedule \
params config="/etc/glance/cinder-api.conf" os_password="secrete"
os_username="admin" os_tenant_name="admin" os_auth_url="http://192.168.42.103:5000/v2.0/" \
op monitor interval="30s" timeout="30s

7.1.8 Adding Nova resources to pacemaker

Pcs resource create p_nova_api ocf:openstack :nova-api \ params config="/etc/nova/nova.conf" \ op monitor interval="5s" timeout="5s" pcs resource create p_scheduler ocf:openstack:nova-scheduler \params config= "/ etc/nova/nova.conf" \ op monitor interval="30s" timeout="30s"

pcs resource create p_novnc ocf:openstack:nova-vnc \
params config="/etc/nova/nova.conf" \ op monitor interval="30s" timeout="30s"
pcs resource create p_nova-cert ocf:openstack:nova-cert \
params config="/etc/nova/nova.conf" \
op monitor interval="30s" timeout="30s"
pcs resource create p_nova-consoleauth ocf:openstack:nova-consoleauth params config="/etc/nova/nova.conf" \
op monitor interval="30s" timeout="30s"

pcs resource create p_novn-conductor ocf:openstack:nova-conductor \
params config="/etc/nova/nova.conf" \
op monitor interval="30s" timeout="30s"

pcs resource create p_nova-network ocf:openstack:nova-network \
params config="/etc/nova/nova.conf" \
op monitor interval="30s" timeout="30s"

7.1.9 Adding Quantum resources to pacemaker

pcs resource create ocf:openstack:quantum-server \
os_password="secrete" os_username="admin" os_tenant_name="admin" \
keystone_get_token_url="http://192.168.42.103:5000/v2.0/" \
op monitor interval="30s" timeout="30s"

7.1.10 Results – PCS Shell

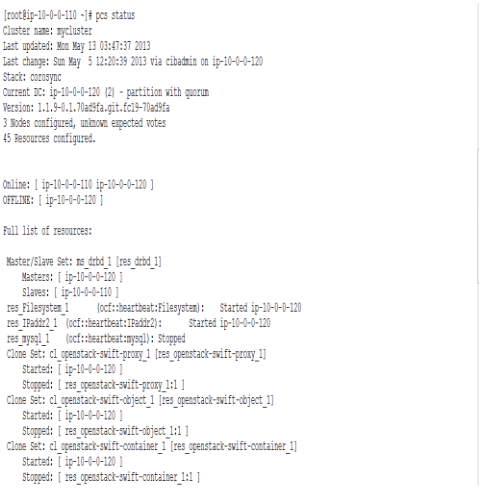

Figure 4 : PCs status

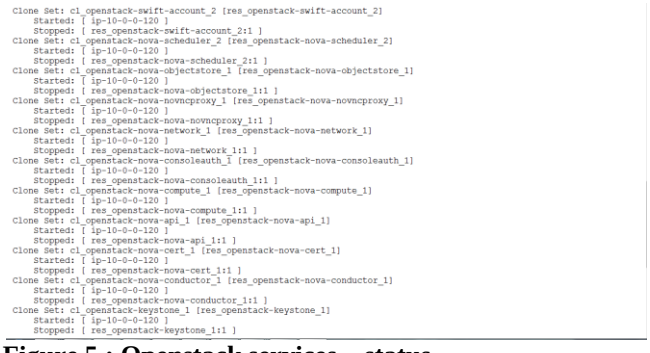

Figure 5 : Openstack services – status

VIII. Implementation High Availability using LCMC

The LCMC is a GUI application that configures, manages and visualizes high-availability clusters. Specifically it manages clusters that use one or more of these components: Pacemaker, Corosync, Heartbeat, DRBD, KVM, XEN and LVM

LCMC (Linux Cluster Management Console) by Rasto Levrinc is a Pacemaker GUI written in Java. Formerly known as DRBD MC. It uses SSH to connect to the Linux cluster from your desktop computer, or it can be embedded in a web-page as an applet. See the website for more information

8.1 Check Corosync and Pacemaker status

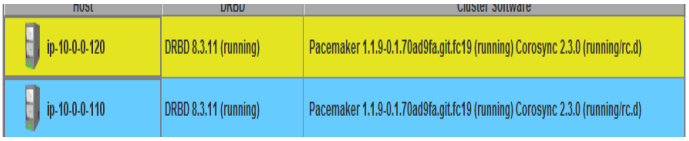

Figure 6: Pacemaker and Corosync status

8.2 Openstack Services- Availability

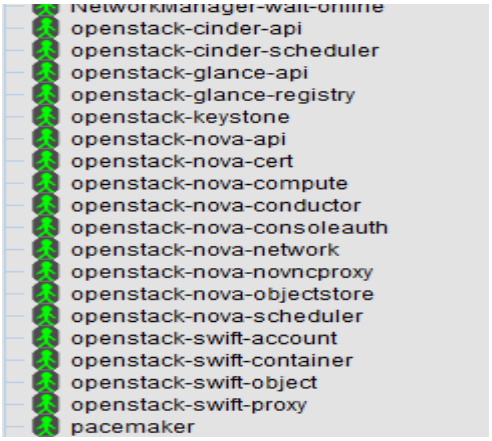

Figure 7 : Availability of Openstack services

8.3 Resource types in Pacemaker/Corosync

8.3.1 Primitive

First, is this Primitive resources of the most commonly used. It is the basis for all resource definitions. A resource to be used in the Act-Standby normal configuration, this can move a node in one place somewhere. Therefore, you can use the resources that it is sufficient to move in only one node of a cluster-wide resource if failure and to fail over to another node. You will want to define this resource usually if you want to HA clustering something like a mail server or database.

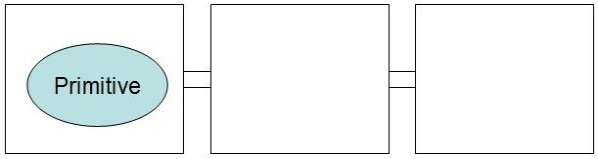

Figure 8 : Primitive resource

8.3.2 Clone

The Clone resource is used when you want to operate with multiple nodes Primitive resources. Definition method is the flow to define the Primitive first, is to Clone of it for that.

If you want to run on more than one node at a certain application, when you try to achieve in only Primitive, you must be defined by the number of nodes that you want to move, but it can be moved by simply defining the Clone one resource in the case of Clone .

Use as a typical example, there is a (following RA) pingd resource agent to be used for monitoring of communication networks. In Act-Standby configuration that provides a service over the network, Primitive resources Act has failed and failing it over to the Standby node, it does not make sense network of Standby node you are off. When monitoring the network at the node in order to avoid a situation in which these, is allowed to operate on all nodes pingd RA, does not provide the service, the network has gone out of order, the service does not fail over to that node Here is how to use the normal to like.

It should be noted that the implementation of RA, because there is no difference Primitive, the Clone, as a separate, RA that can be defined in Primitive can Clone of behavior comes is guaranteed.

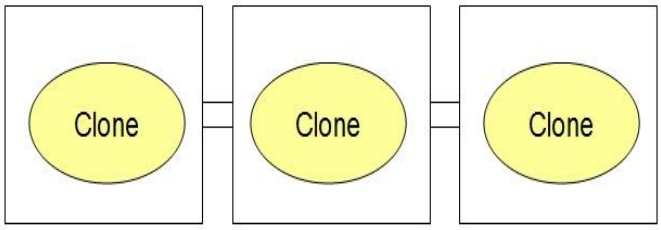

Figure 9 : Clone resources

8.3.3 Master / Slave

The Master / Slave resource, in which further development of the Clone resource, use the resources that a parent-child relationship in the Clone resources. Definition method is the flow to define the Primitive first, is to Master / Slave of it.

The typical use, there is a RA of DRBD to use for a replication of data. In DRBD, there is a condition called Secondary and Primary, reading and writing of data, receive data for replication from the Primary In Secondary in Primary, and writes it to disk. Therefore, it is necessary to running on multiple nodes DRBD, necessary to distinguish the status of the Secondary and Primary further comes out. I realize using the Master / Slave of Pacemaker this.

The operation by adding start and stop operation of resources in the Master / Slave, is mounted on the RA Primitive, of Clone for, start, that stop, was promoted to the parent promote, that demote, demote to children in RA Because that must be implemented, it is not possible to Master / Slave of as an RA for the Clone and Primitive.

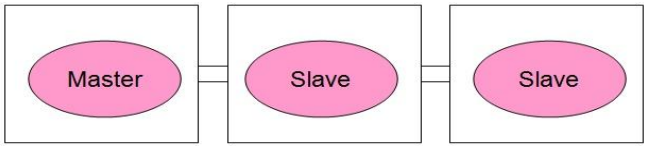

Figure 10 : Master-Slave resources

Screenshots of HA Openstack using LCMC

8.3.4 HA for MySQL

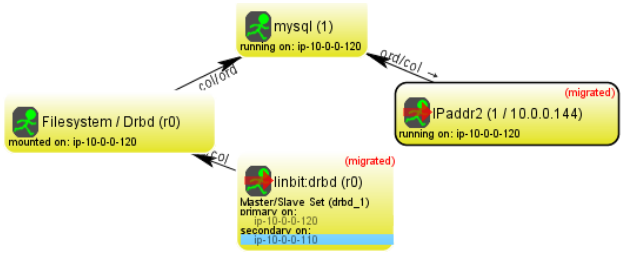

Figure 11 : MySQL HA Availability

8.3.5 HA for NOVA components

Figure 12 : Nova high Availability

8.3.6 HA for Swift components

Figure 13 : Swift High Availability

8.3.7 HA for Cinder Components

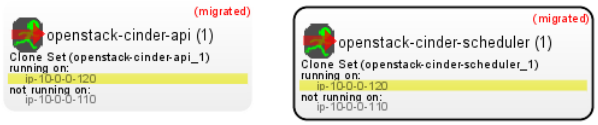

Figure 14 : Cinder – High Availability

IX. Conclusion

While Pacemaker and Corosync is designed to provide High availability for all components of Openstack on physical servers and virtual servers also

This solution is zero investment since it is available from opensource community Corosync and pacemaker is lightweight process with respect to performance since it is consumes 1 or 2% CPU utilization. It also supports automated failover and switchover within 2 seconds .data and system environments preserved across an outage of the production system. These entities remain available through a failover even when the system currently hosting them experiences an outage

References

[1] http://www.sebastien-han.fr/blog/2012/07/02/openstack-nova-components-ha/
[2] https://github.com/mseknibilel/OpenStack-Grizzly-Install-Guide/blob/master/OpenStack_Grizzly_Install_Guide.rst
[3] http://clusterlabs.org/doc/en-US/Pacemaker/1.1-pcs/html/Clusters_from_Scratch/index.html
[4] http://www.6tech.org/2013/03/linux-firewall-cluster-with-pacemaker-and-corosync/
[5] http://www.linux-ha.org/wiki/Resource_agents
[6] https://github.com/madkiss/openstack-resource-agents/tree/master/ocf
[7] https://wiki.openstack.org/wiki/Main_Page
[8] http://docs.openstack.org/trunk/openstack-ha/content/ch-intro.html
[9] http://blog.scottlowe.org/2013/04/17/openstack-summit-2013-openstack-high-availability-in-grizzly-and-beyond/