International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2013): 4.438

An Efficient Failover Enabling Mechanism in OpenStack

S. Ramakrishnan

M.Tech Cloud Computing
Department of Information Technology
SRM University, SRM Nagar, Kattankulathur, Tamil Nadu, India, 603203

Abstract: ***This Paper actually focuses on an efficient approach to enable failover in OpenStack an Open Source Cloud operating system that plays an important role in business continuity and Disaster Recovery. We have setup a simulated environment wherein Single node OpenStack Deployment has been installed on two different hypervisors and Efficient Scripting has been done in the Compute node i.e. Nova as it takes control of the Hypervisors Controlling the OpenStack installed on top of the Host machines. We have demonstrated the Failover in the OpenStack.The goal of this work is to create a framework that will enable protecting applications and services (VMs, images, volumes, etc) from disaster. Determining and selecting what application and services to protect is the responsibility of the user, while handling the logistics of protecting is up to the cloud (and its operator).***

Keywords: OpenStack, Disaster Recovery, Failover, Hypervisors

1. Introduction

OpenStack Compute (Nova) is a cloud computing fabric controller, which is the main part of an IaaS system. It is designed to manage and automate pools of computer resources and can work with widely available virtualization technologies, as well as bare metal and high-performance computing (HPC) configurations. KVM, VMware, and Xen are available choices for hypervisor technology, together with Hyper-V and Linux container technology such as LXC.It is written in Python and uses many external libraries such as Eventlet (for concurrent programming), Kombu (for AMQP communication), and SQL Alchemy (for database access).Compute's architecture is designed to scale horizontally on standard hardware with no proprietary hardware or software requirements and provide the ability to integrate with legacy systems and third-party technologies.[1]

The proposed system is developed using OpenStack. OpenStack is open source software which the project developers and cloud computing technologist can use to setup and run the cloud. Its services can be accessed via APIs. The important components of OpenStack are Nova, Swift, Keystone and Glance, Keystone and Horizon.

Disaster Recovery (DR) for OpenStack is an umbrella topic that describes what needs to be done for applications and services (generally referred to as workload) running in an OpenStack cloud to survive a large scale disaster. Providing DR for a workload is a complex task involving infrastructure, software and an understanding of the workload. To enable recovery following a disaster, the administrator needs to execute a complex set of provisioning operations that will mimic the day-to-day setup in a different environment. Enabling DR for OpenStack hosted workloads requires enablement (APIs) in OpenStack components (e.g., Cinder) and tools which may be outside of OpenStack (e.g., scripts) to invoke, orchestrate and leverage the component specific APIs.

2. Design Procedure

Disaster Recovery should include support for:

- Capturing the metadata of the cloud management stack, relevant for the protected workloads/resources: either as point-in-time snapshots of the metadata, or as continuous replication of the metadata.
- Making available the VM images needed to run the hosted workload on the target cloud.
- Replication of the workload data using storage replication, application level replication, or backup/restore.

We note that metadata changes are less frequent than application data changes, and different mechanisms can handle replication of different portions of the metadata and data (volumes, images, etc.)[2]

The approach is built around:

1. Identify required enablement and missing features in OpenStack projects
2. Create enablement in specific OpenStack projects
3. Create orchestration scripts to demonstrate DR

When resources to be protected are logically associated with a workload (or a set of inter-related workloads), both the replication and the recovery processes should be able to incorporate hooks to ensure consistency of the replicated data & metadata, as well as to enable customization (automated or manual) of the individual workload components at recovery site. Heat can be used to represent such workloads, as well as to automate the above processes.

3. Implementation

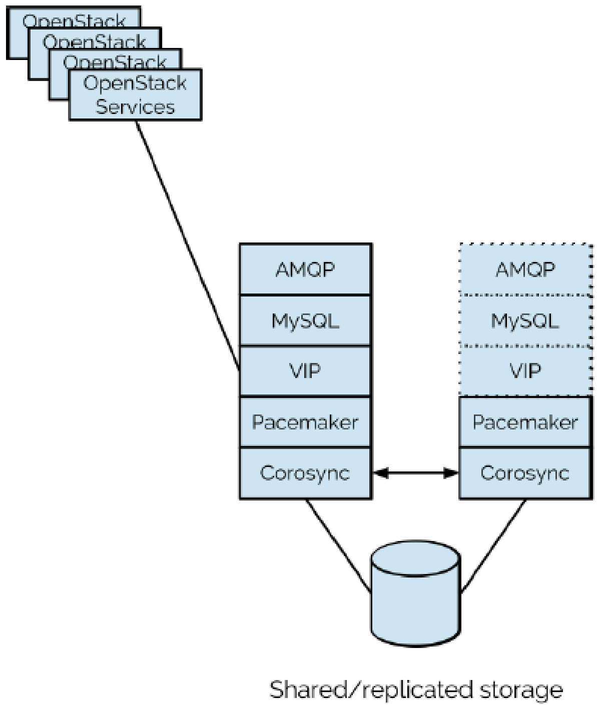


Figure 1: A Basic Implementation of the Concept

1. Design Tenets [4]
1.1 The DR is between a primary cloud and a target cloud - independently managed.
1.2 The approach should enable a hybrid deployment between private and public cloud.
1.3 Note that some of the work related to DR may be relevant to enabling high-availability between regions, availability zones or cells which do share some of the OpenStack services.
1.4 Ideally (but not as an immediate step) one of the clouds (primary or target) could be non-OpenStack or even non-cloud bare-metal environments.
1.5 The primary and target cloud interact through a "mediator" - a DR Middleware or gateway to make sure the clouds are decoupled.
1.6 The DR scheme will protect a set of VMs and related resources (VM images, persistent storage, network definitions, metadata, etc). The resources would be typically associated with a workload or a set of workloads owned by a tenant.
1.7 Allow flexibility in choice of Recovery Point Objective (RPO) and Recovery Time Objective (RTO).

The thing we concentrate upon here is to control the Nova component as controlling and Mastering Nova commands and deep understanding of the same will lead to efficient failover Enabling mechanism which will lead to better disaster Recovery Techniques and Better business continuity processes.

A. OpenStack Architecture

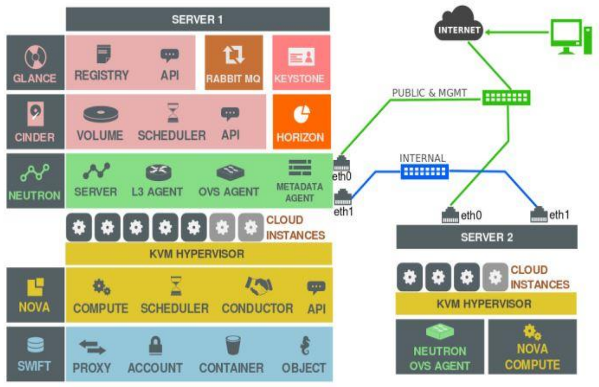


Figure 2: OpenStack Architecture (Single Node)

OpenStack version used here for the Design of the project is Juno Update on top of Ubuntu 14.04(Trusty Tahr).[3]
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, and Citrix etc.

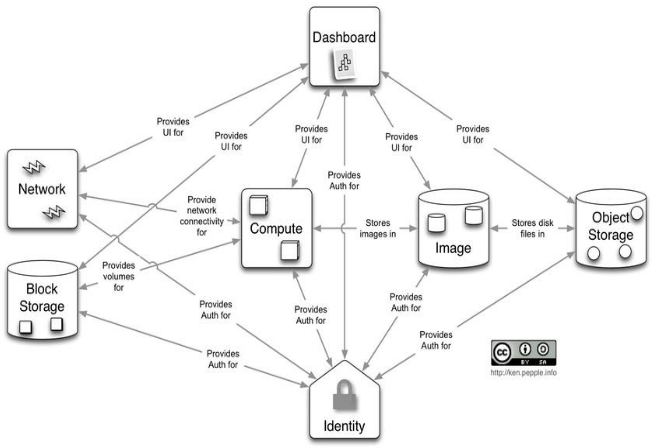


Figure 3: Data Flow in OpenStack

There are 7 main service families in OpenStack Juno:

- Nova - Compute Service
- Swift - Storage Service
- Glance - Imaging Service
- Keystone - Identity Service
- Neutron - Networking service
- Cinder - Volume Service
- Horizon - Web UI Service

We will see Nova that is intended for this paper.

B. Open Stack Compute Infrastructure (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 capabilities by itself; instead, it uses libvirt [5] APIs to interact with the 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:

•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

Components of OpenStack Compute
Nova Cloud Fabric is composed of the following major components:

1. API Server (nova-api)
2. Compute Workers (nova-compute)
3. Scheduler (nova-scheduler)

1. API Server (nova-api)

The API Server provides an interface to the outside world to interact with the cloud infrastructure. API server is the only component that the outside world uses to manage the infrastructure. The management is done through web services calls using EC2 API. The API Server then, in turn, communicates with the relevant components of the cloud infrastructure through theMessage Queue. As an alternative to EC2 API, OpenStack also provides a native API called "OpenStack API".

2. Compute Worker (nova-compute)

Compute workers deal with instance management life cycle. They receive the requests for life cycle management via the Message Queue and carry out operations. There are several Compute Workers in a typical production cloud deployment. An instance is deployed on any of the available compute worker based on the scheduling algorithm used.

3. Scheduler (nova-scheduler)

The scheduler maps the nova-API calls to the appropriate OpenStack components. It runs as a daemon named nova-schedule and picks up a compute server from a pool of available resources depending upon the scheduling algorithm in place. A scheduler can base its decisions on various factors such as load, memory, physical distance of the availability zone, CPU architecture, etc. The nova scheduler implements a pluggable architecture. Currently the nova-scheduler implements a few basic scheduling algorithms:

3.1. Chance: In this method, a compute host is chosen randomly across availability zones. [6]

3.2. Availability zone: Similar to chance, but the compute host is chosen randomly from within a specified availability zone.

3.3. Simple: In this method, hosts whose load is least are chosen to run the instance. The load information may be fetched from a load balancer.

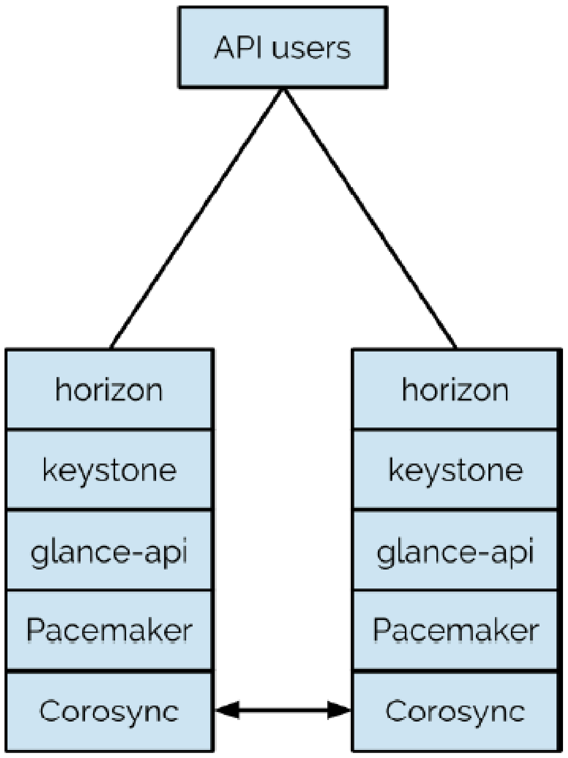


Figure 4. A modified Efficient Conceptualization of Nova API for efficient failover.

High Availability Solution Components

3.1. 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 Coro sync or Heartbeat).

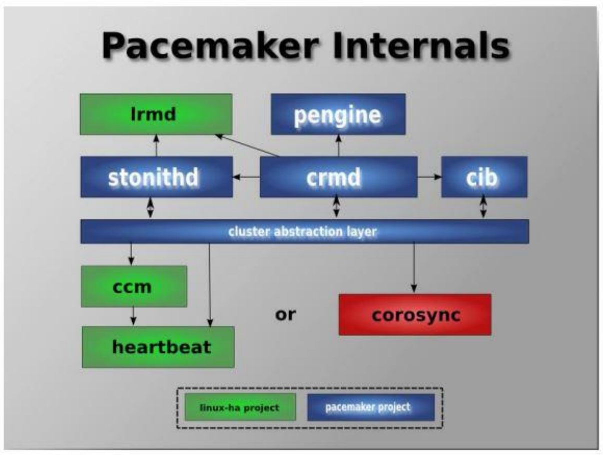
Pacemaker’s key features include:

- 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.

1.1 Pacemaker - Internal components [7]

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



1. CIB (aka. Cluster Information Base)
2. CRMd (aka. Cluster Resource Management daemon)
3. PEngine (aka. PE or Policy Engine)
4. 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 Cocoordinator). 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.[8]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).

4. Design Coding

- **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_novaconsoleauth
ocf:openstack:novaconsoleauth params
config="/etc/nova/nova.conf" \
op monitor interval="30s"
timeout="30s"
pcs resource create p_novnconductor
ocf:openstack:novaconductor \
params
config="/etc/nova/nova.conf" \
op monitor interval="30s"
timeout="30s"
pcs resource create p_novanetwork ocf:openstack:nova-
network \params
config="/etc/nova/nova.conf" \
op monitor interval="30s"
timeout="30s"
```

5. Conclusion

The above implemented setup is tested and on simulation with existing Benchmarks sets true and more efficient as this is a pure technical concept and open source and more of scripting and mastering the nova and exploring to the depth of the OpenStack and its compute node rather than concentrating upon the theoretical benchmarks and the Above formulised solution is a far more efficient solution for efficient Failover enabling mechanism and is a great solution compared to other hard coded solutions like VMware fusion and other solutions as the solution proposed by us is a pure Open Source solution.

6. Future Work

Improving reliability for OpenStack clouds is critical to facilitating broader enterprise adoption. To meet this challenge, Stratus has developed an industry exclusive set of Linux and KVM extensions to provide Unparalleled availability services for OpenStack workloads. This includes support for fully redundant, fault tolerant instances, as well as other more common high availability approaches from a single, simple set of cloud management tools. In future I will explore more of Nova and Storage (Cinder & Swift) Components and will try to make them more efficient.

References

[1] SonaliYadav, "Comparative Study on Open Source Software for Cloud Computing Platform: Eucalyptus,

Openstack and Opennebula”, International Journal Of Engineering And Science, Vol.3, Issue 10 (October 2013), pp 51-54, ISSN (e): 2278-4721, ISSN (p):2319-6483.

[2] Johnson B,YanzhenQu,“A Holistic Model for Making Cloud Migration Decision”, 10th IEEE International Symposium on Parallel and Distributed Processing with Applications, Leganes, 10-13 July 2012, pp 435-441, Print ISBN: 978-1-4673-1631-6,DOI: 10.1109/ISPA.2012.63.

[3] HHarjit Singh, “Technology Transfer Model To Migrate E-Governance To Cloud Computing”, International Journal of Advanced Technology and Engineering Research (IJATER), July 2012, ISSN No: 2250-3536.

[4] Hsu, F., Malik ,S., Ghorbani S, "Open Flow as a Service". April 21, 2014 Available:https://wiki.engr.illinois.edu/download/attachments/197298167/CloudFlow-3rd.pdf?version=6

[5] Tan, D, "Introduction to OpenStack with Dell Compellent Storage Center"April152014Available:http://commwebps3.us.dell.com/techcenter/extras/m/white_papers/20439057.aspx

[6] ITILV3- Service Asset and Configuration Management http://www.itil.org/en/vomkennen/itil/servicetransition/servicetransitionprozesse/serviceassetconfigurationmgmt.php

[7] A. Avetisyan, R. Campbell, I. Gupta, M. Heath, S. Ko,G. Ganger, M. Kozuch, D. O’Hallaron, M. Kunze, T. Kwan, K. Lai, M. Lyons, D. Milojicic, H. Y. Lee, Y. C. Soh, N. K. Ming, J.-Y. Luke, and H. Namgoong, “Open Cirrus: A GlobalCloud Computing Testbed,”Computer, vol. 43, no. 4, pp. 35–43, 2010.

[8] R. K. L. Ko, A. Y. S. Tan, and G. P. Y. Ng, “’Time’for Cloud? Design and Implementation of a Time-Based Cloud Resource Management System,” inProceedings of the 7th IEEE International Conference on Cloud Computing (CLOUD’14), 2014.

[9] OpenStack, “Openstack - Open-source software for buildingprivate and public clouds,” http://www.openstack.org/ [Accessed: 17/12/12014].

[10] C. K. Leong, “Capacity Planning for Your Virtual Data Center and Cloud –Part 3 March 2013,”https://infocus.emc.com/choong\kengleong/capacity-planning-for-your-virtual-data-center-and-cloud-part-3/ [Accessed: 18/02/2014]