# Availability in Openstack

# Availability in Openstack: The bunny that killed the cloud

No Author Given

**Abstract** The use of cloud is on the rise and is forecast to keep increasing exponentially in the coming years. Openstack is one of the major contributors to the cloud space and there are lot of providers using Openstack for providing cloud solutions. Many organizations run their entire network in the cloud and the availability of the cloud is of paramount importance. Openstack is the popular choice of implementation by many organizations. Openstack is an integration of many projects that make up the platform. There are many advantages to open source modularization and many essential services required to run the infrastructure. One such service is AMQP message broker service and the default one for Openstack is RabbitMQ. Our experimentation shows that it is possible to inject random messages to the queue of RabbitMQ bottling the resource of the main controller. This eventually leads to the entire cloud infrastructure crashing.

## 1 Introduction

Cloud Computing is an amalgamation of a range of different technologies and about 94% of all enterprises use cloud in one way or another [1]. One of the biggest names of open source cloud Computing infrastructure software is Openstack [2]. Openstack is made up of several projects that work together to provide a seamless experience to the user. Availability is one of the most important features of the cloud, since many critical applications and services of the organizations run in the cloud.

Generally, the most common reason to denying service to a legitimate users is Distributed Denial of Service (DDoS) that involves creating huge amount of traffic or greater speed in the flow of traffic [3]. But to deny service sometimes, all that needs to be done is to take down one essential service that holds the infrastructure together. Such a similar adhesive to the cloud is RabbitMQ. This message broker is based on the Advance Message Queueing Protocol (AMQP) which is one of the

essential components in setting up Openstack. It is critical and its setup is one of the major steps performed in the setting of the cloud [4].

## 2 Openstack Architecture

### 2.1 Core Services

Openstack is an open source cloud Operating System that acts as an enabler to an organization to create an IaaS based cloud. Openstack pools all the hardware resources to be provisioned and provides a simple web interface to control the cloud [5]. The ability to deploy Openstack to bare metal, virtual machines or containers makes it desirable due to the variety of options possible as shown in Figure 1

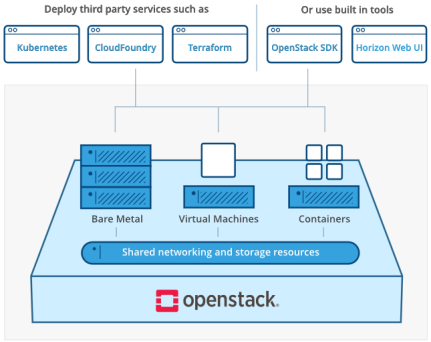


**Fig. 1** High level Openstack Architecture of Openstack as portrayed in the documentation [5]

Openstack is broken down to smaller sub-projects whereby each of them form the cloud itself. There are four core services that are essential to running a base Openstack environment. Table 1 provides a description of the core services required to run Openstack.

**Table 1** List and details of common Openstack Component [6]

| Services | Roles | Description | Requirement |
|---|---|---|---|
| Nova | Compute | This service provides scalable, on demand access to compute resources. Some of the responsibilities include spawning, scheduling and decommissioning VMs on demand. | Mandatory |
| Neutron | Networking | Neutron works as SDN project that provides connectivity between the hardware as well as the other Openstack services. | Mandatory |
| Keystone | Identity Service | It essentially works as the security layer of Openstack. The service handles the authentication and authorization of the Openstack services. | Mandatory |
| Glance | Image Service | This service stores the virtual images of the operating systems that helps nova to provision VMs. | Mandatory |
| Cinder | Block Storage | It enables the creation, deletion and management of black storage for the VMs. | Optional |
| Swift | Object Storage | A good solution for storing unstructured data that can grow without a limit. | Optional |
| Horizon | Dashboard | A web based UI that helps in administration of Openstack. | Optional |

### 2.1.1 Compute Service

The compute service is referred to as Nova and it implements services and associated libraries to provide massively scalable, on demand, self service access to compute resources. The resources that Nova tries to access include containers, virtual machines and bare metals [7].

### 2.1.2 Networking Service

Neutron (See Table1) allows for the creation and attachment of interface devices created and managed by Openstack to other networks. The Physical Network Infrastructure (PNI) and Virtual Network Infrastructure (VNI) are both managed by Neutron. Neutron provides a lot of flexibility to the cloud administrators to create and manage a lot of virtual services in the network like firewall, load-balancers and VPN. The Openstack Network has at least one 'external network', which represents the physical network. On that external network we could have at least one or more 'internal network', that are SDN based virtual networks. This would allow to isolate having smaller clouds within the infrastructure with strong isolation. In terms of security, Openstack allow for the isolation of part of the network, providing for smaller clouds within the infrastructure with strong isolation. Thus, every VM could be a part of one or more Security Groups. Based on the concepts mentioned above, we have two options for creating the virtual network:

1. Networking Option 1: Provider networks: It is the simplest form of networking option in Openstack where it bridges virtual network to physical networks. This option deploys bridging and VLAN segmentation of networks [6].

2. Networking Option 2: Self-service networks: This options routes virtual networks into physical networking using NAT (Network Address Translation). This option allows the customers of the cloud to create virtual interfaces with the involvement of the administrators with help of VXLAN [6].

#### 2.1.3 Identity Service

Keystone service helps to manage credentials in Openstack. Keystone is a single point of integration that uses Openstack's Identity API for client authentication, service discovery and multi-tenant authorization. Openstack maintains a service catalogue that lists all the active services (both mandatory and optional) used by Openstack. These services can have one or many end-points. There are three types of end-points: admin, internal and public. The idea of these different end-points is to provide isolation. For instance, the public end-point can be on a separate network that can be accessed by users through the Internet. This can be further categorization into 'Regions' in Openstack providing scalability. So each Region can have an admin, internal and public end-point providing separation and isolation for better security [8].

#### 2.1.4 Image Service

Glance helps Openstack create, manage and retrieve virtual images. It uses REST API to query metadata of images stored in the database. This query can be used to retrieve the image and launch an instance based on the image. The images can be stored in a simple file system to block based object storage as available in Openstack. There are replication services that runs to ensure that these images are available within the cluster [9].

### 2.2 Environment Essentials

In order to run the core services listed above, it is important that the following environment components are installed:

- Network Time Protocol : It is important that the services running in controller node and the compute nodes are synchronized. Chrony [10] is recommended as the NTP implementation to be used with Openstack. The suggestion is to use the controller as the reference point for all the other nodes [11].
- Database: Most of the services that run on Openstack use a database for storing information. Generally this service is installed on the controller node. Openstack supports a variety of databases like MYSQL, MariaDB and PostgreSQL [12].

- Caching: Openstack uses memcached to cache the tokens used in the Identity Service. This helps to quicken the process of interaction between the controller and compute nodes [13].
- Key-Value Store: Etcd is used by Openstack for a distributed key locking, storing configuration, keeping track of service live-ness and other scenarios [14].
- Message Broker: RabbitMQ is the deafult message broker used in Openstack. We have a detailed discussion on RabbitMQ in the section below.

Message Broker

The message broken that Openstack uses is RabbitMQ by default for most distributions. RabbitMQ is installed on the controller nodes and sits between the compute nodes to help them to coordinate operations and status information among services. RabbitMQ uses Remote Procedural Calls (RPC) to achieve this. Figure 2 gives a deeper look into the architectural setup of the message broker in Openstack [4].

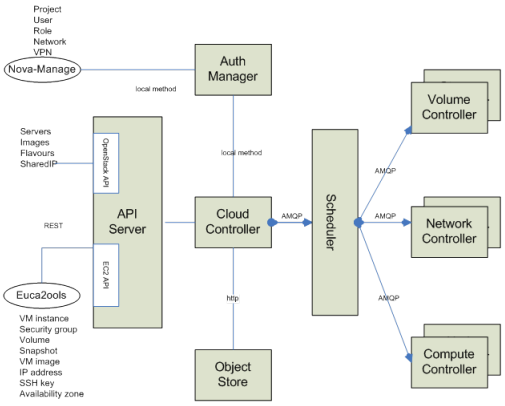


**Fig. 2** AMQP and Nova Architecture for interaction [4]

Openstack groups and ungroups RPCs into function calls using an adapter. Each nova service creates two queues: one for one which accepts messages with routing keys *NODE-TYPE.NODE-ID* (for example compute.hostname) and another, which accepts messages with routing keys as generic *NODE-TYPE*

## 3 Our Experimental Setup

We created an Openstack setup using five machines (Intel Core i5, 8GB RAM) and each machine having two Network Interface Cards. We created two networks: Provider Network and Management Network. We have one machine acting as the controller, two compute nodes, one object storage and one block storage.

There have been exploits in the past that allowed an intruder to get hold of the RabbitMQ authentication details [15]. We argue that the same vulnerability can be extended to be used by an intruder having a simple VM on the cloud. And this would allow the guestVM (currently an attacker), to have full control of the VM which in turn will allow the attacker to take down the entire cloud infrastructure. We created a Proof of Concept in the laboratory with the below assumptions:

- The attacker has been able to successfully attain the authentication credentials.
- The controller where RabbitMQ is installed has a public IP address. A simple IP scan of the range would allow the attacker to gain this information.

We enabled management plugin for RabbitMQ to enable API based access programmatically so that we could alter the variables easily. However, this is not required and this could be done with a simple written script.

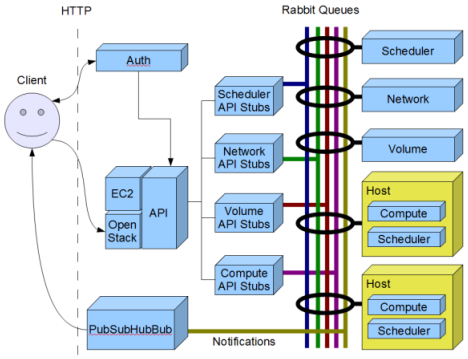


**Fig. 3** RabbitMQ in Openstack [16]

It is evident from Figure 3 that the Openstack queues and exchanges should be used for scheduler stubs API, network stubs API, volume stubs API, compute stubs API and notifications. However, We wrote a Java script (the Java API was available for RabbitMQ), where we opened a connection from a VM to the controller. We then created one exchange and multitudes of channels and queues. Through these queues we sent a random message. In order to process this request RAM and CPU resources were being utilized at great level causing the controller to not respond and eventually fail.

## 4 Results and Discussion

We observed an increase in the number of queues and channels. The message rate rose up to 239 messages/second as shown in the figure 4.

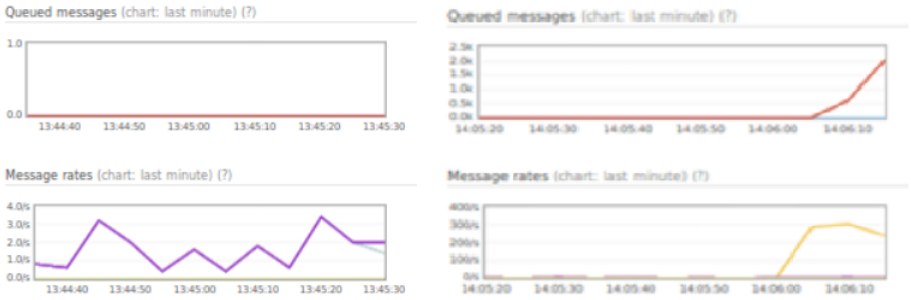


**Fig. 4** Comparison of RabbitMQ Message rate and message in queue before (left-hand-side) and during (right hand side) the attack

Table 2 shows the comparison of RabbitMQ variables before and during the attack. As evident there was a single new connection made during the attack and created 4500+ channels and 6100+ channels before eventually the service spiking the RAM and CPU usage of the controller to point of failure.

**Table 2** Comparing RabbitMQ variables before and during the attack

| | Before the Attack | During the Attack |
|---|---|---|
| Connections | 67 | 68 |
| Channels | 67 | 4618 |
| Exchanges | 34 | 34 |
| Queues | 96 | 6176 |

We actively monitored the controller's resource at the time to of attack. The results from the attack as shown in Table 3 There was an average increase in 71.3% of the CPU usage. The memory usage simply kept increasing until 99.5% with an increase of 0.1%-0.3% every second. After 99.5% the RAM started using the swap memory of 1 GB which was allowed and this was used up within 23.4 seconds causing the controller to completely crash.

**Table 3** Result of Attacking the RabbitMQ service of Openstack on the controller

| | Before the Attack | During the Attack |
|---|---|---|
| CPU x 4 (Avg. in %) | 12.3 | 83.6 |
| RAM (Avg. in %) | 62.0 | 99.9 |
| Swap Memory (in Mpbs) | 0.0 | 927.6 |
| Number of Ports Open | 25 | 26 |

We noticed that the compute nodes were still functional and the VMs resources are utilized from them. But without the controller node and the other environmental essentials which runs on it, the entire range of VMs failed too. This led us to believe that even compartmentalization of the individual services would not work and RabbitMQ failure would act as a single point of failure.

## 5 Conclusion and Future Work

RabbitMQ which is the default message broker which Openstack Ubuntu distribution uses allowed injection of connections and channels. Ideally, the message broker should limit only the creation of queues and channels only pertaining to Openstack. Everything else should be disallowed. The Simple Authentication Security Layer (SASL) authentication framework does allow for enabling TLS connection. However, if the attacker is able to get hold of the authentication details from the VM as demonstrated in the exploits above, the Proof of Concept would still be effective and disastrous to the cloud. With the correct credentials we could also inject into the existing queues created by Openstack. Putting a limit to these queues and the messages that you pass through them will not be ideal for scalability. This would call for a better method to handle messaging service which essentially connects the different part of the cloud. It would be interesting to further look into those particular activities within the cloud that may be able to create a bottleneck for RabbitMQ and explore the possibility to affect availability.

## References

1. V. Sumina, 26 Cloud Computing Statistics, Facts Trends for 2021 (2021).
   URL https://www.cloudwards.net/cloud-computing-statistics/
2. Openstack, Open Source Cloud Computing Infrastructure - OpenStack (2021).
   URL https://www.openstack.org/
3. I. A. Elia, N. Antunes, N. Laranjeiro, M. Vieira, An Analysis of OpenStack Vulnerabilities (2017). doi:10.1109/EDCC.2017.29.
   URL https://bugs.launchpad.net
4. OpenStack, OpenStack Docs: AMQP and Nova (2021).
   URL https://docs.openstack.org/nova/rocky/reference/rpc.html
5. Openstack, What is OpenStack?
   URL https://www.openstack.org/software/
6. Openstack, OpenStack Docs: Install Guide Overview.
   URL https://docs.openstack.org/install-guide/overview.html
7. Openstack, OpenStack Docs: System architecture (2021).
   URL https://docs.openstack.org/nova/rocky/admin/arch.html
8. OpenStack, OpenStack Docs: Identity service overview (2021).
   URL https://docs.openstack.org/newton/install-guide-rdo/common/get-started-identity.html
9. OpenStack, OpenStack Docs: Image service overview (2021).
   URL https://docs.openstack.org/glance/rocky/install/get-started.html

10. R. Curnow, chrony – Introduction (2021).
URL https://chrony.tuxfamily.org/
11. Openstack, Network Time Protocol (NTP) — Installation Guide documentation (2021).
URL https://docs.openstack.org/install-guide/environment-ntp.html
12. Openstack, SQL database — Installation Guide documentation (2021).
URL https://docs.openstack.org/install-guide/environment-sql-database.html
13. Openstack, Memcached — Installation Guide documentation (2021).
URL https://docs.openstack.org/install-guide/environment-memcached.html
14. Openstack, Etcd — Installation Guide documentation (2021).
URL https://docs.openstack.org/install-guide/environment-etcd.html
15. Mark Kirkwood, Bug 1445295 “Guestagent config leaks rabbit password” : Bugs : OpenStack DBaaS (Trove).
URL https://bugs.launchpad.net/trove/+bug/1445295
16. Openstack, MultiClusterZones - OpenStack (2021).
URL https://wiki.openstack.org/wiki/MultiClusterZones