UPGRADING CLOUD INFRASTRUCTURE – CHALLENGES AND SOLUTIONS

Andrei Petrescu and Mihai Carabas

University POLITEHNICA of Bucharest, Splaiul Independentei 313, Bucharest, Romania

ABSTRACT

In today's fast-moving world, advances in technology occur at an alarming rate. Keeping up is difficult, but mandatory, and we must find solutions that will make the process easy. Out of all these technologies, cloud computing is one that is evolving the quickest. We will explore the tools which will help us help us reach our goal and talk about the main subject of our paper, namely keeping up to date with the latest releases in OpenStack private cloud technology. We will also talk about the results and how we found the best solution for the context in which this paper lies.

KEYWORDS

cloud, openstack, cinder, nova, keystone, glance, heat

1. INTRODUCTION

The term "cloud computing" has been around since the 1990s, but the first depiction was observed in a 1977 drawing of a multi-network diagram that described connections between ARPANET, SATNET and Packet Radio net. It was only in 2006 when the term cloud computing [1] was used in the context that we know today, and the technology came to reality when Amazon launched Amazon Web Services and offered S3 (Simple Storage Service) for cloud storage, EC2 (Elastic Compute Cloud) for infrastructure and SQS (Simple Queue Service) for messaging queues.

The problem arising from the context which we described earlier is that institutions and companies have problems keeping up with new releases of said software and cannot benefit from the advances and fixes that they bring. The problem of keeping up with new releases arises from the fact that development is done in an agile way in Openstack community [9]: there is a new version once every six months. In the case of the Faculty of Computer Science and Information Technology, their OpenStack cluster has been stuck to the same release since 2015, when Openstack Liberty was developed. They did not upgrade the version of Openstack due to the fact that there was not present any clean methodology to do the upgrade without breaking any production services. Because there have been 5 releases since Liberty, the cluster suffers from a lack of features and stability which new features provide [15]. Another problem is that the Liberty release has reached its EOL (end of life) status, meaning that it will not receive any more updates. As time passes by, more problems will arise because the difference between the versions will grow, and it will become even harder to do upgrades without causing loss of data or increased downtime.

This paper proposes a methodology on how to upgrade Openstack from Liberty to Queens version without breaking anything in production. Thus, our main objective is to provide a way

Natarajan Meghanathan et al. (Eds) : CSEIT, CMLA, NeTCOM, CIoT, SPM, NCS, WiMoNe, Graph-hoc - 2019
pp. 73-82, 2019. © CS & IT-CSCP 2019 DOI: 10.5121/csit.2019.91306

to upgrade a cloud infrastructure based on OpenStack framework to the latest release, which at the time of writing this paper is Queens, from earlier releases.

The paper is structured as follows: Chapter 2 presents an overview of different cloud computing technologies, chapter 3 shows how to do configuration management which is a trivial step in managing cloud frameworks. Chapter 4 presents the proposed solution regarding the cloud infrastructure upgrade and chapter 5 is doing evalution related to the services upgraded in Openstack.

2. CLOUD COMPUTING TECHNOLOGIES

Cloud computing [6] is a modern concept that enables users to abstract the hardware layer by using resources in a dynamic way and based on their needs, in a much faster way than using standard baremetal servers [16]. This concept is based around virtual machines, which can share physical resources and can be created and destroyed very fast. It all began with the concept of the GRID architecture [7] that describes the close coupling of computational resources to act like a single machine [2]. We can refer to a SMP (Symmetric Multiprocessor) as a grid of many colocated CPUs that do the same work, and to an MPP (Massively Parallel Processor) as a grid of SMPs interconnected by very fast busses. A cluster is a group of computers that share the same purpose [10]. A very well-known example is SETI@Home, that comprised of 400000 CPUs which belonged to computers all over the world, all serving the purpose of finding extra-terrestrial life [2]. In recent years, the term as-a-service has been coined, which describes the types of services that the cloud can offer [12]. The three big kinds of services, are as follows:

Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS). Another type of service that appeared recently, where focus is shifted on the function that resources do, and not on the resources themselves is called Function as a Service (FaaS).

OpenStack is an IaaS [1] solution for private clouds, and one of the most popular among them, occupying second place, behind VMWare vSphere, and has been adopted by more than 1200 companies, including Best Buy, Comcast, PayPal, Walmart and Wells Fargo. It is an open source project that was initiated in 2010 and is now backed by more than 1300 active contributors.

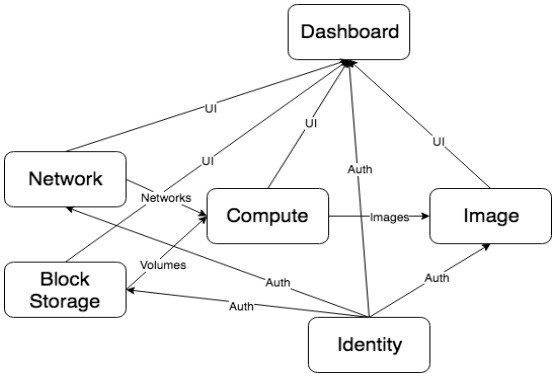

Figure 1. Flow of data in the OpenStack architecture

Its architecture is based mainly around controller and compute nodes but can also have optional nodes such as networking and storage nodes [8]. The controller nodes are the control plane of the cluster by holding the APIs of the services and performs authentication and scheduling of resources. They can also hold resources that are shared between components all over the cluster such as the database or the messaging queue [14]. The compute nodes are the ones that hold the

virtual machines and are usually in greater number than controller nodes and also can provide virtual machine live migration services [13]. The networking nodes are responsible for DHCP (Dynamic Host Configuration Protocol), VLANs (Virtual Local Area Networks), tunneling, routing and also for the flow of traffic in the cluster. The storage nodes are assigned the role of providing block storage volumes and are typically backed by LVM (Logical Volume Manager). There can also be nodes dedicated to generic object storage and image storage.

In this project we will focus mainly on a handful of important services but will briefly discuss others too. The most important services that are found in OpenStack are Keystone, Nova, Neutron, Cinder, Glance, Heat and Horizon.

Much of the current literature which describes the upgrade of OpenStack focuses on upgrading from release N to release N+1 by doing rolling updates with no downtime. This is not helpful in the context of the problem that this project will solve, because the difference of releases that will be covered will be 5, from Liberty to Queens.

3. CONFIGURATION MANAGEMENT

This section is focused on describing the notion of configuration management. The most important role of configuration management is to provide an easy and fast way to provision servers by using automation, thus eliminating human error. Before con-figuration management, the setup of servers was mainly done by hand, or by the use of bash scripts. The main problem with the old way of configuring servers is that it was error prone and it was not modular. Since the invention of configuration management, the term Infrastructure as Code has been brought up.

Infrastructure as code enables infrastructure to be treated as application code and be edited, reviewed and version controlled. System administrators could now track errors in their code and treat them as bugs, have repositories for their code and have different branches for testing and production. There are two types of Infrastructure as Code software on the market right now. The first type focuses on creating infra-structure, i.e. virtual machines, networks, IP allocation and so on. Examples of soft-ware that are specialized for these kinds of tasks are Terraform and Salt-cloud and work by accessing the APIs of the cloud platforms that they target. The second kind is the one that focuses on configuring servers by installing software, managing con- figuration files and ensuring that certain services are up and running. This is the type of Infrastructure as Code software that we will focus on and that we have used in my project.

Puppet is a configuration management software written in Ruby that is developed by Puppet Labs and is one of the first modern CM software, being launched in 2005. Puppet is based around a master-slave architecture, where the code resides on the master and the agents pull the code and run it locally.

Puppet code is based around the idea of modules, each serving a different purpose. In turn, modules are composed of three folders: files, manifests and templates. The files folder is used for static files, the templates are used for dynamically generated files, for example a template that generates a MySQL configuration file and the IP that it listens on is determined at run-time, and finally, there are manifests. Mani-fests are the core of Puppet and contain the actual code. They are made up of classes, each doing a specific task. Classes can build on other classes and modules can build on other modules. In OpenStack installation, there is a module named Open- Stack, which builds on two other modules, OpenStack Controller and OpenStack Compute, and installs either one depending on the type of computer that is executing the code. OpenStack Controller builds on Puppet modules that install RabbitMQ, Memcached, Apache, MySQL, Keystone, and parts of Neutron and Nova that be-long on a controller node. OpenStack

Compute builds on two Puppet modules that install the Neutron OpenVSwitch agent and the Nova Compute service.

4. PROPOSED SOLUTION AND IMPLEMENTATION DETAILS

As stated in the introduction, the purpose of this project is to upgrade the current version of OpenStack, which is Liberty, to the latest version, Queens. Because these two versions are 5 releases apart, the two main goals which we will achieve are to execute the upgrade fast and to preserve all the data. To be able to do this, we pro-pose a solution where there are as few upgrades between services as possible. Traditional OpenStack upgrades are executed on every service that the cluster is running. In the case of the cluster that is running on the servers of the faculty, there are 7 ser-vices, 3 of which also run on different nodes than the controller node.

Cinder runs on two servers, the controller and the storage node, Nova runs on the controller and each compute node and Neutron runs on the controller, the network node and on each compute node. Suppose we have 1 storage node, 1 network node and 4 compute nodes. There would be 35 upgrades on the controller, 40 upgrades on the compute nodes, 5 on the storage node and 5 on the network node. In total, there would be 85 upgrades. The solution that we propose will introduce downtime, but it will preserve data and it will be relatively fast for getting through 5 releases.

Nodes other than the controller nodes, and the dashboard service, will be upgraded directly from Liberty to Queens because the data in the database is not modified by them. The data in the database is modified by the services which run on the controller node and synchronize the database on each release. Synchronization does two things, it either upgrades schemas or migrates data. Sometimes columns are renamed, or their type is changed, and not synchronizing the database would cause the new version of the service to not start at all. Database version are called migrations levels and they need to be sequential.

By doing upgrades this way, using the same scenario as before, we will have 18 upgrades on the controller, 1 on the storage node, 1 on the network node and 2 on each compute node, so 22 in total, compared to 85.

Another proposition is using Puppet for managing the upgrades, as it can speed up the upgrade process and assure that every execution of the code will produce the same results. Even if we have narrowed down the updates to 22, the upgrade still needs to be done with great care. Because of this, the Puppet classes must do the least amount of work and be run manually so as to make sure that errors have not occurred. There should be classes for each service upgrade and the code should be copied on each node, so that it can be run using “puppet apply”. The connection strings in the configuration files must also be changed so that the services can successfully connect to the database. The user has updated the API endpoints or create new end- points if necessary, which is the case for Cinder and Nova, because of the Placement API that is introduced in the Ocata release. Lastly, and most importantly, before synchronizing the database in Ocata, the user must create a database named “nova_cell0”, map cell0 and create a cell named “cell1”. This is mandatory in Ocata, as Nova has switched to this new, more scalable, system of managing compute nodes.

Lastly, the organization of the Puppet modules must be done so they cover all the possible deployments of OpenStack cluster. Some examples of OpenStack deployments are presented in Figure 2.

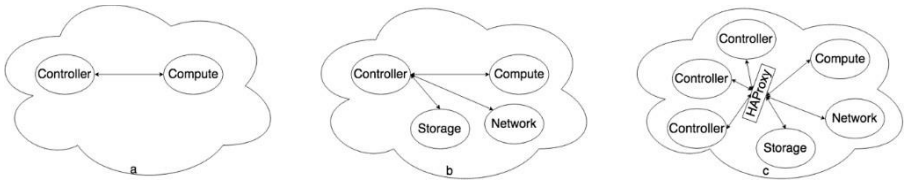

Figure 2. The figure presents three OpenStack deployment examples

In Figure 2a, OpenStack is installed on only two nodes, on what we call a proof of concept cluster. In Figure 2b, the Network and Storage nodes are separated from the Controller to provide better resource management. Figure 2c presented a high availability OpenStack deployment where there is more than one Controller node. Requests to the controller nodes go through a proxy for the reason of simplifying access and ensuring equal load on the nodes.

The first thing which we needed to create was an automated, repeatable and fast deployment mechanism in order start from scratch every time we have done a mistake that would cost us more time to fix than to start over again. To do this, we made use of the existing OpenStack cluster and created two virtual machines, with 4GB RAM each, which is the minimum recommended for a proof of concept deployment of OpenStack Queens. Because we needed to start from scratch every time, we made use of the rebuild feature that OpenStack Nova provides. This enabled the reinitialization of the instance with a fresh install of CentOS in a short period of time, so we could start over again and change the approach that we took to creating the up- grade process. Below, in table 1, the times for rebuilding and bootstrapping the Liberty deployment on the two nodes is presented.

Table 1. Deployment time benchmark.

Rebuild Time (s)	Deployment on controller (s)	Deployment on compute (s)
71	601.67	565.11

Information about upgrades between multiple releases is hard to find, so we chose to test the upgrade of each service from Liberty to Queens. This would often fail, and the main indication was that the synchronization of the database would fail.

Further, we will describe some interesting database errors that we have come across when upgrading Nova and the usual errors which appears when trying to upgrade the database schema of other services between releases which are too far apart. We will also present some packaging errors and some bugs that we have found in the OpenStack code.

OpenStack Mitaka introduces an important change in the Nova database, more specifically, in the compute_nodes table. There is one column that is added, named uuid, and because of this, it is important to re-register the compute nodes after upgrading. Figure 3 is outputted from a 2 select operations done on the compute_nodes table and describe how the uuid entry is filled after a compute node reconnects to the Nova API.

```
              host: openstack-agent
ram_allocation_ratio: 0
cpu_allocation_ratio: 0
              uuid: NULL
```

Figure 3. Difference in output before and after the node registered

If this step is skipped, when trying to upgrade to the Newton release, the upgrade script will output an error message, as shown in Figure 4, and will not continue until all the entries in the table with the value of NULL in the uuid column will be deleted.

```
error: There are still 1 unmigrated records in the compute_nodes table. Migration cannot continue until all records have been migrated.
```

Figure 4. Error outputed if nodes are not re-registered

When trying to upgrade to Ocata, it is also important to go through the Newton release, because there are database entries which need to be migrated into the nova_api database. Figure 5 shows the error message that the upgrade script outputs when data is not migrated.

```
ValidationError: Migration cannot continue until all these have been migrated to the api database. Please run `nova-manage db online_migrations' on Newton code before continuing.
```

Figure 5. Ocata - migrations were not done beforehand in Newton

An example of the data migration talked about earlier is presented in Figure 6. It shows the output after the migration is done because, before that, the table was empty.

```
MariaDB [nova_api]> select * from flavors;
+---------------------+------------+-----------+----+-------------+-------+------+-------------+----------+--------------+--------+-----------+-----------+-----------+
| created_at          | updated_at | name      | id | memory_mb | vcpus | swap | vcpu_weight | flavorid | rxtx_factor | root_gb | ephemeral_gb | disabled | is_public |
+---------------------+------------+-----------+----+-------------+-------+------+-------------+----------+--------------+--------+-----------+-----------+-----------+
| 2018-06-18 17:13:13 | NULL       | m1.medium |  1 |        4096 |     2 |    0 |           0 | 3        |          1.0 |     40 |         0 |         0 |         1 |
| 2018-06-18 17:13:13 | NULL       | m1.tiny   |  2 |         512 |     1 |    0 |           0 | 1        |          1.0 |      1 |         0 |         0 |         1 |
| 2018-06-18 17:13:12 | NULL       | m1.large  |  3 |        8192 |     4 |    0 |           0 | 4        |          1.0 |     80 |         0 |         0 |         1 |
| 2018-06-18 17:13:13 | NULL       | m1.xlarge |  4 |       16384 |     8 |    0 |           0 | 5        |          1.0 |    160 |         0 |         0 |         1 |
| 2018-06-18 17:13:13 | NULL       | m1.small  |  5 |        2048 |     1 |    0 |           0 | 2        |          1.0 |     20 |         0 |         0 |         1 |
+---------------------+------------+-----------+----+-------------+-------+------+-------------+----------+--------------+--------+-----------+-----------+-----------+
```

Figure 6. Select operation after data has been migrated in the Newton release

Another important aspect when taking into consideration the upgrade to Ocata, is to create the nova_cell0 and the cell mappings, as they are mandatory from that release on. If these steps are not done before the synchronization database, the upgrade script will out an error as show as in Figure 7.

```
ValidationError: Cell mappings are not created, but required for Ocata. Please run nova-manage cell_v2 simple_cell_setup before continuing.
```

Figure 7. Error of the Ocata upgrade - cell mapping was not done beforehand

Other common errors are related to packages which the package manager does not upgrade automatically, and the services either fail to start or the database migration fails because of them. Below we describe one of them and the process that we went through to fix it and others which were related.

When trying to migrate Cinder from Liberty to Newton, because the package manager sometimes does not update all dependencies. This error can be resolved by upgrading the

python2-os-brick package. Other errors like these can be resolved the same. We looked at packages that were imported by the Python module and searched what version is installed by using the command “yum list installed” and then piping the output to grep.
Upgrading services too far will cause errors to be outputted by the management script. One example is shown in Figure 8 and checking the current migration level in the database is shown in Figure 9.

```
[root@openstack-master openstack_upgrade]# /bin/heat-manage db_sync
ERROR: "Database schema file with version 66 doesn't exist."
```

Figure 8. Error output if Heat is upgraded too far

```
MariaDB [heat]> select * from migrate_version;
+---------------+-----------------------------------------------------------------+---------+
| repository_id | repository_path                                                 | version |
+---------------+-----------------------------------------------------------------+---------+
| heat          | /usr/lib/python2.7/site-packages/heat/db/sqlalchemy/migrate_repo |      65 |
+---------------+-----------------------------------------------------------------+---------+
```

Figure 9. Finding the current migration level of the database

The most interesting kind of errors were those where all the imported packages in the Python modules were up to date and changes in the code were needed. These occurred when upgrading Cinder to the Newton release or Nova to the Ocata release.

These errors were caused by small bugs in the api.py module from the "sqlalchemy" database upgrade packages. Because they were looking for a profiler group in the configuration files of both services, and because those did not exist, the synchronization of the database would fail. Fixing the errors was done by adding a check for profiler attribute in the CONF object.

Each step of the process is done automatically by the classes from the Puppet module that we have developed. To satisfy the constraint of modularity, where the kind of deployment does not matter, we have created 4 types of classes, those with "api" in the name are applied to the controller nodes, those with "comp" in the name are applied to the compute nodes, those with "net" in the name are applied to network nodes and those with "store" in the node are applied to the storage nodes. It does not matter which type is upgraded first, but it is important that all the types of nodes are running OpenStack Queens when starting all the services. The recommended order of upgrading for the controller nodes is presented in Figure 10. The other types of nodes can be safely upgraded directly from Liberty to Queens because they do not store data anywhere.

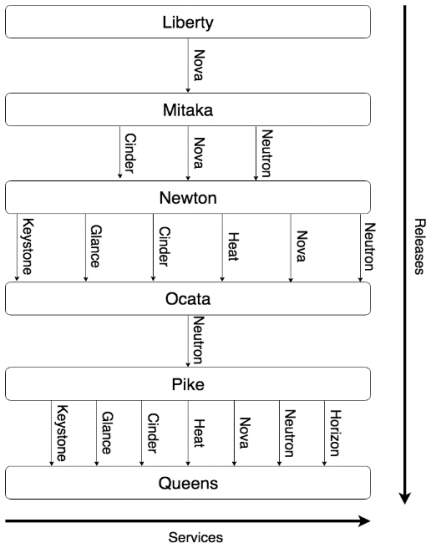

Figure 10. The figure presents the correct order of applying the classes

5. CLOUD INFRASTRUCTURE UPGRADE EVALUTION

In this section we will describe the tests which we have created to benchmark OpenStack after it was upgraded to Queens. We will focus on functional tests which will demonstrate that the cluster works as expected. To verify the functionality of the newly upgraded cluster, we decided to use 3 types of operations on 2 categories of resources, namely add, delete and edit on new and old resources. Table 2 shows what resources have been tested and the results.

Table 2. This table enumerates the types of test that we performed and the results

Name	Type	Add	Delete	Edit
Instances	Old	-	Yes	Yes
	New	Yes	Yes	Yes
Images	Old	-	Yes	Yes
	New	Yes	Yes	Yes
Key Pairs	Old	-	Yes	Yes
	New	Yes	Yes	Yes
Networks	Old	-	Yes	Yes
	New	Yes	Yes	Yes
Routers	Old	-	Yes	Yes
	New	Yes	Yes	Yes
Security Groups	Old	-	Yes	Yes
	New	Yes	Yes	Yes
Projects	Old	-	Yes	Yes
	New	Yes	Yes	Yes
Users	Old	-	Yes	Yes
	New	Yes	Yes	Yes
Roles	Old	-	Yes	Yes
	New	Yes	Yes	Yes

6. CONCLUSIONS

In the beginning of the paper we have presented a brief introduction into the notion of cloud computing and have discussed about the context in which the subject of the paper lies, the problems which arise in this context, the objectives that the paper will meet, the proposed solution to meet these objectives, and an overview of the structure and the sections. The motivating factors which led us to choose this project were mainly of technical nature and had to do with advances which would benefit the students of the faculty in area such as artificial intelligence, machine learning and container orchestration. Moreover, there were personal reasons too that related to learning these new technologies which were used and providing an open source solution that the community can build upon and add new features.

To give an overview of the state of the art we provided a detailed overview of the technologies which form the subject of this paper. The first major technology discussed was cloud computing and how it came to be, as well as the types of services it provides. There are three important types of services which are discussed: Infrastructure as a Service, Platform as a Service and Software as

a Service. We discussed the main type of cloud provider for private and public clouds, OpenStack, which is the subject of this paper. We presented its architecture and the types of services that it is composed of, like the image service, the compute service, the block storage service, the identity service and the networking service. The second technology we have talked about is configuration management (Puppet) and how it is used to create infrastructure as code.We discussed about our proposed solution and the objectives that it should meet, namely that it should be faster than normal upgrades, simple to use, and most important of all, keep the data intact. It also presents in detail on what services should be upgraded to what release and how it should suite any type of OpenStack deployment. The last section discusses benchmarking by performing functional after the cluster was upgraded to ensure that the data was not corrupted and that old resources were still usable.

In conclusion, OpenStack will not stop here, and we will see new releases every year in the future, maybe at an even faster pace than today. To keep up with the changes in technology we decided to make our project available as open source on GitHub and will write blog posts on how to use it on our faculty's blog.

ACKNOWLEDGEMENT

The work has been funded by the Operational Programme Human Capital of the Ministry of European Funds through the Financial Agreement 51675/09.07.2019, SMIS code 125125.

REFERENCES

[1] Aniruddha S. Rumale, D.N.Chaudhari , „Cloud Computing: Infrastructure as a Service", International Journal of Inventive Engineering and Sciences, vol. 1, no. 3, pp 1-7, 2013

[2] Swarnpreet Singh and Tarun Jangwal , „Cost breakdown of Public Cloud Computing and Private Cloud Computing and Security Issues", International Journal of Computer Sci-ence & Information Technology, vol. 4, no. 2, pp 17-31, 2012

[3] Sumit Goyal, „Public vs Private vs Hybrid vs Community - Cloud Computing: A Criti-cal Review", I.J. Computer Network and Information Security, pp 20-29, 2014

[4] Borja Sotomayor, Ian Foster, Rubén S. Montero and Ignacio M. Llorente, „Virtual Infra-structure Management in Private and Hybrid Clouds", 2009. [Online]. Available: https://www.researchgate.net/profile/Ian_Foster/publication/224587421_Virtual_Infrastructure_Management_in_Private_and_Hybrid_Clouds/links/00b49519a475a2ad95000000/Virtual-Infrastructure-Management-in-Private-and-Hybrid-Clouds.pdf

[5] Tiago Rosado, Jorge Bernardino, „An Overview of Openstack Architecture", IDEAS '14 Proceedings of the 18th International Database Engineering & Applications Symposium, pp 366-367, 2014

[6] JoSEP, A.D., KAtz, R., KonWinSKi, A., Gunho, L.E.E., PAttERSon, D. and RABKin, A., 2010. A view of cloud computing. Communications of the ACM, 53(4).

[7] Foster, I., Zhao, Y., Raicu, I. and Lu, S., 2008. Cloud computing and grid computing 360- degree compared. arXiv preprint arXiv:0901.0131.

[8] Sefraoui, O., Aissaoui, M. and Eleuldj, M., 2012. OpenStack: toward an open-source solution for cloud computing. International Journal of Computer Applications, 55(3), pp.38-42.

[9] Kumar, R., Gupta, N., Charu, S., Jain, K. and Jangir, S.K., 2014. Open source solution for cloud computing platform using OpenStack. International Journal of Computer Science and Mobile Computing, 3(5), pp.89-98.

[10] Yadav, S., 2013. Comparative study on open source software for cloud computing platform: Eucalyptus, openstack and opennebula. International Journal Of Engineering And Science, 3(10), pp.51-54.

[11] Corradi, A., Fanelli, M. and Foschini, L., 2014. VM consolidation: A real case based on OpenStack Cloud. Future Generation Computer Systems, 32, pp.118-127.

[12] Merlino, G., Dautov, R., Distefano, S. and Bruneo, D., 2019. Enabling Workload Engineering in Edge, Fog, and Cloud Computing through OpenStack-based Middleware. ACM Transactions on Internet Technology (TOIT), 19(2), p.28.

[13] Hao, J., Ye, K. and Xu, C.Z., 2019, June. Live Migration of Virtual Machines in OpenStack: A Perspective from Reliability Evaluation. In International Conference on Cloud Computing (pp. 99-113). Springer, Cham.

[14] Balmakhtar, M., Persson, C.J. and Rajagopal, A., Sprint Communications Co LP, 2019. Secure cloud computing framework. U.S. Patent Application 10/243,959.

[15] Cotroneo, D., De Simone, L., Iannillo, A.K., Natella, R., Rosiello, S. and Bidokhti, N., 2019. Analyzing the Context of Bug-Fixing Changes in the OpenStack Cloud Computing Platform. arXiv preprint arXiv:1908.11297.

[16] Moges, F.F. and Abebe, S.L., 2019. Energy-aware VM placement algorithms for the OpenStack Neat consolidation framework. Journal of Cloud Computing, 8(1), p.2.