Ansible-Openstack Project

https://github.com/Manideep45/NSOopenstackproject.git

Manideep Gunturu
Department of Telecommunications
Blekinge Institute of Technology
Karlskrona, Sweden
magw21@student.bth.se

Partha Sarathy Singh Samanta
Department of Telecommunications
Blekinge Institute of Technology
Karlskrona, Sweden
pasb21@student.bth.se

Varun Duvva
Department of Telecommunications
Blekinge Institute of Technology
Karlskrona, Sweden
vadv21@student.bth.se

***Abstract*—This paper focuses on the Network Design, Service Deployment and Monitoring with the use of Openstack and various forms of Automation . It involves deployment and operating of services within an openstack cloud. The implementation involves three operating modes they are deployment, operation and cleanup. Deployment involves setting up of network infrastructure, routers,nodes and other components. The operation mode handles the number of nodes that present in the webserver which are requested and available. The cleanup mode releases all cloud resources that were in use.**
Deployment is done through using Ansible, Nginx, HAproxy, Prometheus. We provide a Monitoring solution to get the status of all nodes in the network.

***Index Terms*—Openstack, Monitoring solution, Ansible, Nginx, HAproxy, Prometheus**

I. Introduction

OpenStack is an open-source cloud computing platform that provides a set of software tools for building and managing both public and private clouds. They are available to every user for free of charge and the code is open-source. The software can be modified for any individual needs as it allows users to access virtual servers and other resources in both public and private clouds [2].
It enables organizations to create and manage scalable infrastructure as a service (IaaS) environments, giving them the flexibility and control over their cloud deployments. OpenStack offers a flexible and modular architecture, allowing users to choose the components they need and customize their cloud environment accordingly [4]. It provides a robust set of APIs, command-line tools, and web interfaces for managing and interacting with the cloud resources. The advantages of using OpenStack include scalability, cost-effectiveness, vendor-neutrality, and the ability to leverage open-source innovation. It enables organizations to build and manage their private or public clouds, offering a high level of control and customization over their infrastructure.
OpenStack architecture is built using three main components. They are OpenStack Compute, Image and Object [1]

1) OpenStack Compute: OpenStack Compute, also known as Nova, is a management platform that controls the infrastructure to control IaaS clouds. It provides an administrative interface and an API needed for the orchestration of the Cloud. It includes instances management for servers, networks and access control.

2) OpenStack Imaging Service: Imaging Service (project Glance) provides storage services, recording and distributing the images to virtual machine disks. It also provides an API compatible with the REST architecture to perform queries for information on the images hosted on different storage systems.

3) OpenStack Object Storage: Object Storage (Swift project) is used to create a storage space redundant and scalable for storing multiple petabytes of data. It's not really a file system but is especially designed for long term storage of large volumes. It uses a distributed architecture with multiple access points to avoid SPOF (Single Point of Failure).

II. SECTION I

A. System Architecture

The system consists of nodes working as webserver, proxy servers and a bastion host. The initial setup of webserver contains three servers. These nodes provide two services: service.py and snmp daemon.
Service.py contains a flask application handling Internet Control Message Protocol(ICMP) requests from the users. And the Simple Network Management Protocol(SNMP) daemon also known as SNMP agent is responsible for implementing SNMP functionality on a server.

B. Proxy servers

Proxy servers are used to handle the traffic between the webserves and the clients in the internet. The traffic is balanced using load balancing solutions such as HAproxy and Nginx. We have used HAproxy software to implement this functionality in proxy servers. The incoming requests are handled through two ports defined for each of the services:

1. service.py is handled through PROXY port TCP/5000.

2. SNMP agent is used through PROXY port UDP/6000.

We used two proxy servers to handle the traffic to the webservers. These two servers share the same public IP address and physical IP address. So the incoming requests are handled through two proxy servers that are sharing a single floating IP.

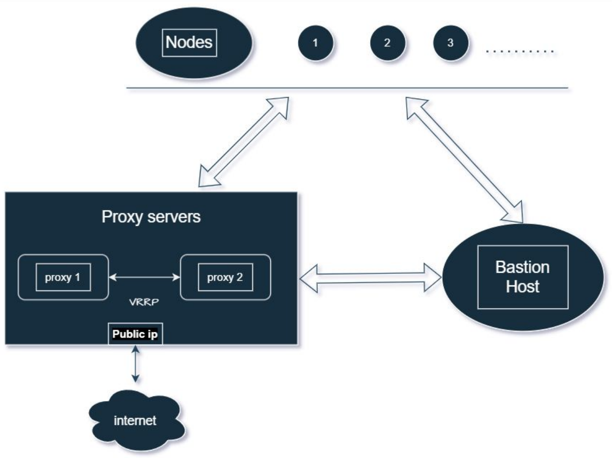


Fig. 1. Architecture

1) Keepalived Package: Keepalived is an open-source software package that provides high availability and load balancing solutions for Linux-based systems. We use keepalived to divide the proxy server into primary and secondary servers. Keepalived achieves high availability by implementing the Virtual Router Redundancy Protocol (VRRP) and the Health Check Notification (HCN) protocols. VRRP enables a group of routers or servers to act as a virtual router with a shared IP address. The primary server within the group assumes the role of the active router, while the other servers remain in standby mode. If the primary server fails or becomes unavailable, one of the standby servers takes over as the active router, ensuring uninterrupted service.

C. Bastion Host

The bastion host is important in the aspect of accessing all the nodes in the network. It is used to enable ssh access to the service nodes. It is used as a availability checker by implementing the monitoring solution. Monitoring solution send a single request to bastion node and gets the status of all nodes. Installation of monitoring solution is done using prometheus.

1) Prometheus: Prometheus is a time-series database and monitoring system designed for collecting and storing metrics from various sources in a distributed environment. Prometheus can scrape metrics including exporters, APIs, time-series database, enabling efficient querying and analysis of historical data.It supports various service discovery mechanisms to dynamically discover and monitor targets in a changing environment. It provides a comprehensive monitoring solution for network nodes. It collects metrics from various targets, including nodes, servers, services, and applications.

D. Modes of Operation

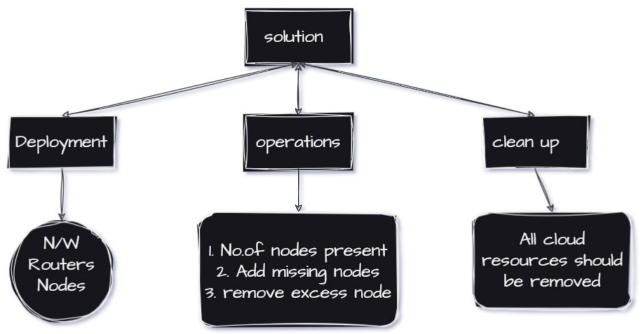


Fig. 2. Modes of Operation

1) Deployment: During the deployment, it creates the required number of networks, routers, nodes, and other required device components.

2) Operation: Initially we considered three webservers. So, in the operations mode it monitors the number of nodes present. That is to monitor how many nodes are requested and available. Adding and removing the nodes comes under the operation mode. The number of required servers is derived from the servers.conf file. If the demand is higher than the present number of nodes/servers then it deploys the required additional serves in the same way if the demand is less than the present available nodes it removes the excess nodes. In monitoring solutions a node is said to be reachable if node response to ICMP pings from bastion host. The status of the system is updated in every 30 seconds.

3) Cleanup: In the cleanup mode, the system should release and remove all cloud resources allocated like networks, routers, keypairs, security groups, webservers etc.

E. Motivation for the Design

Our scripts are written in shell programming. We chose to write in shell programming as it is easy to integrate it with a wide range of command line tools and utilities available in the operating system. Its easier to convert shell language to other formats and it has direct access to system functionalities.

One of the major advantages of using shell scripts for deployment is the flexibility they provide. Shell scripts can be easily customized and tailored to meet specific requirements, such as adapting to changes in network configurations or resource allocations. Another benefit of shell scripts is their repeatability and scalability. Once the scripts have been created and tested, they can be reused across different environments, saving time and effort.

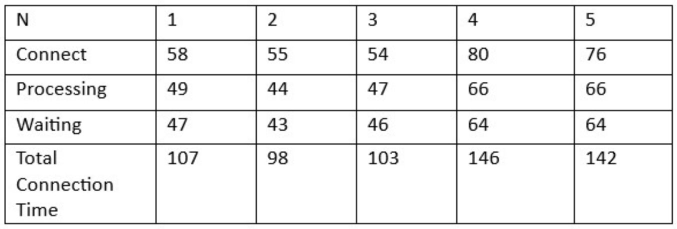
We chose prometheus to monitor our application as it actively pulls data from serves at regular intervals. Prometheus tool is more direct while telegraf is versatile and complex in terms of monitoring the given number of servers.

F. Alternatives

While choosing the language for the script a combination of shell language and python programming is chosen keeping in mind the complexity and the quantity of code. Out of these two the codes can be written in any one of them alone. While python provides its largest support to packages and tools its not an go to option in less complex and simple deployments. Telegraf can be used in combination with grafana to provide monitoring solutions. It was not used in this case as it lacks its own storage for storing the monitored data metrics.

III. SECTION II

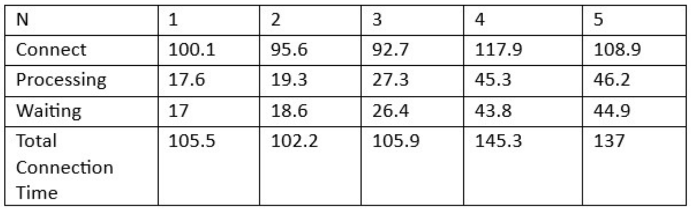
The network performance is evaluated by changing the number of nodes(N) in the webservers. We use Apache Benchmark to evaluate required performance. The number of request is fixed at 1000 and with the concurrency of 10. we are trying to investigate the connection times while sending 1000 request with 10 simultanious requests sent to the servers at the same time. We consider the mean and standard deviation values of ”connection times”. The nodes are varied between 1 to 5 that is servers= 1,2,3,4,5. The network performance is evaluated by sending request to HAproxy server at port 5000. The outputs from the investigation is tabulated in fiure [3] and [4].



N	1	2	3	4	5
Connect	58	55	54	80	76
Processing	49	44	47	66	66
Waiting	47	43	46	64	64
Total Connection Time	107	98	103	146	142

Fig. 3. Mean

The above table contains the mean values of connection time consists of time for connection + processing + waiting. It is observed that with increase in number of nodes the time taken for connection decreases upto the value of n=3 after that it shows randomness. The total connection time tends to show randomness but overall it increases between nodes 1 to 5. The average time taken for connection is 119.2 ms(milliseconds). The standard deviation of connection time varies between 105 to 145 showing its variability from mean value.
To ensure the statistical significance the data is taken after certain number of repetitions to decrease the variability of the data. The number of requests is set to a moderate value of 1000 keeping in mind the sample size. The data sets are taken by using random to avoid biasing of data. The control variables like number of requests and concurrency are fixed at 1000 and 10 respectively.



N	1	2	3	4	5
Connect	100.1	95.6	92.7	117.9	108.9
Processing	17.6	19.3	27.3	45.3	46.2
Waiting	17	18.6	26.4	43.8	44.9
Total Connection Time	105.5	102.2	105.9	145.3	137

Fig. 4. Standard Deviation

IV. SECTION III

While investigating our solution to operate on a large scale from the network perspective, the number of nodes providing service needs to be increased. The band width, throughput, latency and response time are needed to be taken care of. The mean value of connection time needs to be maintained at the lower level. In large networks the proxy servers needs to be increase so that the throughput and bandwidth remains unaffected. This also avoids packet loss and maintain optimum load balancing.
From a service perspective concurrent test need to be made and simulated in high traffic scenarios to evaluate the service response at heavy load. Since this will be a large network one can use programming like python and ansible for scripting instead of shell. We also can use telegraf instead of prometheus as telegraf is versatile to handle complex and large networks efficiently. Resource utilization in the cloud needs to be taken care of and monitor at regular intervals to prevent bottlenecks. If service is operated from different locations different servers will have different location configurations in openrc file. This will be a challange while tried to execute the opetationa scripts. Since the servers belong to different regions they are faced with different network traffics according to the regions. There are more chances of timeouts and packet loss. The way in which load balancing is handled needs to be modified in such a way that servers from particular region have enough proxy servers to avoid bottleneck. The stress testing and performance bottlenecks on service performance may result in varied and challenging due to different regions.

V. CONCLUSION

In conclusion, this project successfully maintains robust performance and availability by leveraging many nodes, load balancers, and monitoring tools. Although there may be occasional delays due to Ansible’s fact-gathering process, deployment times are optimized through the implementation of techniques such as parallel task execution and caching. The result is a solid framework that enables the rapid and successful establishment and operation of services within an OpenStack Cloud environment.

References

[1] O. Sefraoui, M. Aissaoui, and M. Eleuldj, “OpenStack: Toward an Open-Source Solution for Cloud Computing,” International Journal of Computer Applications, vol. 55, pp. 38–42, Oct. 2012, doi: 10.5120/8738-2991.

[2] K. Jackson and C. Bunch, OpenStack cloud computing cookbook: over 100 recipes to successfully set up manage your OpenStack cloud environments with complete coverage of Nova, Swift, Keystone, Glance, Horizon, Neutron, and Cinder, 2. Aufl. in Quick answers to common problems. Birmingham: Packt Publ, 2013.

[3] K. Pepple, Deploying OpenStack. O’Reilly Media, Inc., 2011.

[4] T. Rosado and J. Bernardino, “An overview of openstack architecture,” in Proceedings of the 18th International Database Engineering Applications Symposium, in IDEAS ’14. New York, NY, USA: Association for Computing Machinery, Jul. 2014, pp. 366–367. doi: 10.1145/2628194.2628195.