Automated Service Deployment and Management in OpenStack

https://github.com/kimt22/NSO$_{Project1.git}$

Madhukar Putta
Telecommunication Systems
Blekinge Institute of Technology
Karlskrona, Sweden
mapu22@student.bth.se

Kishore Mothukuru Pushparaj
Telecommunication Systems
Blekinge Institute of Technology
Karlskrona, Sweden
kimt22@student.bth.se

RajaVishnu Dev Venna
Telecommunication Systems
Blekinge Institute of Technology
Karlskrona, Sweden
rave22@student.bth.se

Venkata Bhamidipati
Telecommunication Systems
Blekinge Institute of Technology
Karlskrona, Sweden
vebh22@student.bth.se

***Abstract*—This report specifies the detailed path for the implementation of deployment, operating, and managing services required in the open-stack cloud environment which provides scalability, availability, and automation. The system architecture consists of the deployment and operation of a bastion host which helps control and secure access of the ports, dual proxy ports configured consist of HAProxy with keepalived for the load balancing for handling the dynamic workload. This system also consists of service node integration of custom applications and SNMPd, which is crucial in reliability and ensures continuous service availability. This also consists of automation scripts that manage the deployment, operation, and clean-up phases. The clean-up phase helps in clearing the environment as per the requirement. Performance analysis and stability of the system services are validated through Apache benchmark that examines the system response under variable load for improvement of response time as the number of nodes increases. This report helps in the construction of a dynamic, resilient, and flexible cloud environment with the requirements of the users.**

***Index Terms*—OpenStack Cloud, Ansible, Automation, HAProxy, Keepalived, Deployment, Operations, Load Balancing, Dynamic scaling, Clean up, Apache Benchmark, SNMPd.**

I. Introduction

With the high capacity network, virtualization, and low-cost computers, cloud computing provides users access to a virtually unlimited number of resources, a rapidly elastically and pay-per-use model, meeting the growing business demands [1] [2]. OpenStack is one of the complete open-source cloud platforms that offers Infrastructure as a Service (IaaS) and helps in creating and managing virtual machines on the available resources. The main characteristics of OpenStack such as scalability, flexibility, and open source make it more interesting than other cloud platforms [3].

This report mainly focuses on solving the issues related to the operational cost and enhancing the resilience of the system in deploying and maintaining the system with efficient usage of resources. This also reduces the manual work on the users with the change in requirements. To emphasize the above we have worked in the three stages such as installation, operation, and clean-up. In the install stage we have deployed the required nodes, servers and these are managed efficiently in the next stage the operation which checks the requirements of a client and manages efficiently followed by clean where we clean the environment for the next usage of the resources to not get corrupt and safety of the environment. This series of stages is also made to be automated in such a way as to reduce manual intervention and fast flexibility as per the load on the system. By following this way we ensure that the system is designed to make it a more dynamic, resilient, and flexible cloud environment with the requirements of the users.

II. Architecture Overview

To achieve our goals the architecture is constructed in such a way a private network consists of the deployment of the following key components:

- Bastion host: This serves as a secure entry for SSH access to the internal network and acts as a monitoring guide.
- Proxy nodes: HAProxy with Keepalived provides load balancing and availability of service nodes.
- Floating IP's: These are assigned for external access of nodes.
- service nodes: these are responsible for the workload management.

  The internet users or clients are the external network to our system and service nodes, Haproxy, subnets, and networks are our internal network and bastion acts as a mediator to enter into the internal network from the external network

III. Deployment Stage

As mentioned in the above section the building of the architecture starts through the deployment of the requirements in the Open stack. The following sequence is followed through the automation to the deployment of precise architecture.

A. Keypair

Secure access to the virtual network is created through this key pair.

B. Network, Subnet, Router

This helps in making a pavement to the data as per requirement.

C. Security group

This helps in managing the traffic to created instances.

D. HAProxy nodes

This helps in the load balancing of the service nodes for efficient usage of the resources

E. Bastion node

The secure monitoring of nodes is done through the configuration of the bastion config file.

F. Service nodes

These functions as per the configured through .py file.

G. SSH configuration file

The script generates an SSH configuration file and a host file to simplify SSH access to the deployed nodes.

H. Run Ansible playbook

It consists of a set of instructions, configurations, and tasks that must be done in the automation of the network.

IV. Operation Stage

The main functionality of this stage mainly focuses on the monitoring and scaling of the service nodes as per requirements.

A. Monitoring and Scaling

Manages the status of the service nodes. if something node fails new one is created immediately and in this phase by executing the operate.py file. Within this file, the desired quantity of nodes is read from the servers.conf file checks whether service nodes are working as per the desired performance as specified in service.conf and adjust nodes as per load to ensure resource utilization.

B. Ansible Playbook Execution

If there is a change in the number of service nodes, it triggers the execution of the Ansible playbook (site.yaml). The playbook is responsible for configuring the servers and ensuring that they are properly set up and integrated with the rest of the network environment.

V. Cleanup Stage

Followed through the operational stage the clean-up stage takes place. Here the clean-up concentrates on the clearing of the resources that are generated or used during the above stages for the construction of the network architecture. It ensures the deletion of all nodes, networks, routers, and floating IPs in such a way that it does not corrupt the next activities done in this environment with these resources.

VI. Motivation for the design

*A. **Modular design***

- Choice: Isolating components
- Reasoning:
  - The design chosen is modular, with specific components for load balancing, and servicing hosts it makes it easier to manage and scale the network as per the requirement without failure.

*B. **Scalability and Flexibility***

- Choice: Dynamic Scaling with Automation
- Reasoning:
  - The dynamic scaling scales the number of nodes based on the server configuration which returns efficient resource handling that shows flexibility to use in dynamic load on the system,

*C. **Monitoring and tagging***

- Choice: Prometheus and Grafana for Monitoring
- Reasoning:
  - It helps in real-time monitoring of system performance and all resources are tagged for easy identification and maintaining of them individually.

VII. Alternatives considered

*A. **OpenStack Built-in Load Balancer***

- Advantages: It reduces the work of configuring the external node to work as a load balancer. easy to create and manage.
- Limitations: The project is restricted to the use of an automated load balancer.

*B. **Integration of powerful tools to openstack***

- Advantages: Adding powerful tools like Kubernetes improves performance.
- Limitations: Increases setup complexity, maintenance overhead, and potential security risks due to more exposed points.

VIII. Statistical Analysis

The purpose of this analysis is to evaluate the performance, reliability, and efficiency of our system as the number of nodes increases. This is evaluated by simulating varying loads using Apache Bench, a tool designed for benchmarking and testing HTTP server performance metrics, specifically the mean and standard deviation (sd) values from the 'Connection Times' section, to determine the system's scalability and response to varying loads. By the above simulation, it could be found the optimize the number of nodes or servers needed for efficient performance, and reliability for the service.

A. *Experimental Setup*

1) Tools and Environment:

- Apache Benchmark (ab): Used to simulate HTTP requests and measure performance.
- OpenStack Cloud Environment: Deployed service using OpenStack with configurations as described in the project.
- Proxy Nodes: HAProxy with keepalived instances are used to distribute traffic to the service nodes.

2) Parameters:

- Number of Requests (-n): Total number of requests to be performed during the test.
- Concurrency Level (-c): Number of multiple requests to perform at a time.
- Test Configurations: 0 (Baseline), 1, 2, 3, 4, and 5 nodes.

3) Baseline Test: A baseline test is established to get a reference point for the system performance using only proxy nodes and keep alive. Running the baseline test without service nodes intends to reduce the complexity in the system and verify the keepalived application on the proxy nodes whether it can handle the incoming traffic. The baseline test also helps determine whether the system has reached a steady state, which is crucial for obtaining reliable and consistent performance measurements. Using the baseline test the performance metrics such as throughput, and response time of the node with different loads is calculated by sending high levels of requests and concurrency ie n is 1000 and c is 100. After reaching a steady state provides reliable and for comparing performance across different configurations.

B. *Procedure and Data Collection*

The experimental procedure involves setting up and testing the performance of the deployed service under various node configurations. The node configurations were changed based on the values provided in the servers.conf file, ranging from 1 to 5 nodes. For each test, the mean connection time and standard deviation (sd) of connection times were recorded. A series of 5 iterations were conducted for each configuration to account for variability in network conditions and other transient factors. The results were averaged to ensure statistical significance and provide a reliable measure of performance.

N=10, C=1

Nodes	Mean (ms)	Standard Deviation (ms)
1	71	8.8
2	77	23.1
3	77	23.8
4	70	0.8
5	61	2.3

N=50, C=10

Nodes	Mean (ms)	Standard Deviation (ms)
1	67	4.5
2	79	23.3
3	72	10
4	61	1.4
5	61	2.5

N=100, C=10

Nodes	Mean (ms)	Standard Deviation (ms)
1	88	23.9
2	69	6.1
3	71	9.3
4	60	1.4
5	59	1.5

C. *Analysis and Conclusion*

With the above experiment bench the analysis is made in such a way that As the number of nodes increased, we observed the following change in response times, the Response times decreased, indicating improved distribution of the load. with an increase in the load, the standard deviation also increases which requires to increase in the active nodes to make it efficient performance and reliable on the system.

- **Effect of node configuration**:
  As the service nodes increase it reduces the load on the system reduces the mean response time and fetches the data faster and the standard deviation of the system decreases as the system performance becomes more stable due to a decrease in the variability in the load on the system.
- **Effect of c and n values**:
  As the load increases ie n, c values increase the mean response time of the service nodes increases. To stabilize the performance of the system we need to increase the nodes to decrease the load on each node and stabilize it. The service gets delayed further if the dynamical scaling does not take place, corruption of data may take place and inefficient performance of the system may arise.

IX. Scalability Analysis and Architectural Modifications

With the help of mean response time from the benchmark we can calculate the scalability of the servers required for the reliability and efficient performance of the system. The service nodes required for the system performance are calculated to

meet the desired node considering a system with 1 node when n 100,c 10 is given by:
number of servers= Target RPS / RPS per Server.
Calculating the required number of servers: Using the RPS value from n=100, c=10 (23.9 RPS)

- For 100 users/sec: (100users/s)/(23.9RPS) =5 servers
- For 1000 users/sec: (1000users/s)/(23.9RPS) = 42servers
- For 10000 users/sec: (10000users/s)/(23.9RPS) = 419 servers

A. *Architectural Modifications:*

1) *Fault Tolerance and Redundancy:*

- Add redundant proxy nodes to ensure that the system can continue to function even if one or more nodes fail. Implement regular health checks for nodes and automatic failover mechanisms to reroute traffic away from failing or unresponsive nodes.

2) *Caching and Data Management:*

- Introduce caching layers to store frequently accessed data, reducing the load on the database and improving response times

3) *Resource Optimization:*

- Use containerization to manage resources more efficiently and allow for easy deployment and scaling of services.

4) *Testing and Validation:*

- Implement CI/CD pipelines to automate testing and deployment, ensuring that new features and updates do not negatively impact system performance.

B. *Potential Problems and Their Impact on Service Performance*

1) *load balancing inefficiencies:*

- It causes the uneven distribution of traffic which causes delays in the responses. sending traffic to unhealthy nodes that cause a degradation of performance.
- This can be mitigated by regular health monitoring and using advanced algorithms to load balancing.

2) *Storage and data management issues :*

- insufficient storage space can cause loss of the data and corruption of data
- This could be mitigated through regular backup to the nodes.

3) *Resource contention:*

- Multiple services running on the same device can lead to resource contention which leads to inconsistent performance
- continuous monitoring of resource usage and making suitable changes as required.

C. *Challenges of Multi-Location Operation on Networking and Service Performance*

1) *Networking Challenges:*

- Latency:
  - distance from the data centers or data processing can impact the speed of the data transmission.
  - it can be mitigated through edge computing and setting an optimized path for the data between nodes
- Bandwidth:
  - This constraint arises if there is a narrow bandwidth between locations and a large volume of data that must be transferred.
  - data compression must be done before transmitting to down the latency of large volumes of data and prioritization is better if narrow bandwidth is present.
- Operational complexity:
  - Due to different timezones, and different kinds of network practices it is complex to service such scenarios.
  - Developing the standard practices and creating cross-channel communication for clear communication between teams to resolve issues raised.

2) *Service Performance Challenges:*

- Data consistency:
  - synchronization of data to the receiver creates some latency at the end.
  - eventual consistency models help to reduce this latency.
- Cost management:
  - operating in different locations increases cost due to resources and inefficient utilization of resources
  - regular cost analysis must be done and have to see where can we save money and efficiently utilization of resources.
- Service redundancy :
  - failure of a server at a location can cause a delay due to rerouting to another location.
  - regular failover testing and maintenance helps to regular working of server.

REFERENCES

[1] S. Saibharath and G. Geethakumari, "Design and Implementation of a forensic framework for Cloud in OpenStack cloud platform," 2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Delhi, India, 2014, pp. 645-650, doi: 10.1109/ICACCI.2014.6968451.

[2] Sefraoui, Omar Aissaoui, Mohammed Eleuldj, Mohsine. (2012). OpenStack: Toward an Open-Source Solution for Cloud Computing. International Journal of Computer Applications. 55. 38-42. 10.5120/8738-2991.

[3] R. Nasim and A. J. Kassler, "Deploying OpenStack: Virtual Infrastructure or Dedicated Hardware," 2014 IEEE 38th International Computer Software and Applications Conference Workshops, Vasteras, Sweden, 2014, pp. 84-89, doi: 10.1109/COMPSACW.2014.18

IEEE conference templates contain guidance text for composing and formatting conference papers. Please ensure that all template text is removed from your conference paper prior to submission to the conference. Failure to remove the template text from your paper may result in your paper not being published.