ET 2598 – Group Project

https://github.com/suve19/Project_GroupNSO

Group Name: group NSO

Students

Sai Chaya Mounika Mudragada

Surya Teja Kodukulla

Sushma Vendrapu

Introduction

There is a greater demand for online services that can manage enormous levels of traffic due to the internet's fast expansion. We suggest using the OpenStack cloud computing platform as the basis for creating a scalable web service to address these problems. A load balancer, proxy nodes, and application nodes make up the architecture. We utilised Apache Benchmark (ab) to test the request performance for 1, 2, 3, 4, and 5 application nodes in order to assess the performance of our solution. We also examined potential changes to our architecture to manage increased traffic levels and assessed the number of servers needed to handle various amounts of demand.

Design Description

Our solution is a scalable web service using OpenStack cloud infrastructure. The service is composed of a load balancer, proxy nodes, and application nodes. The proxy nodes forward requests to the application nodes once the load balancer divides up incoming traffic among them. The proxy nodes offer access to the service via a public IP address and act as a proxy for external resources. We chose OpenStack to serve our online service because it offers a scalable, affordable, and simple-to-manage infrastructure. We chose OpenStack as well since it offers a variety of tools and APIs for developers to manage and customise their cloud infrastructure.

The deployment information for our servers is shown in Table 1. While we installed the application nodes on one to five machines, we deployed the load balancer (HAProxy) and proxy servers (Nginx) on three nodes.

Table 1. Server Deployment Details

Server Type	*Name of Severs*
Load Balancer	1
Proxy Servers	3
Application Nodes	1-5

We employed HAProxy, a trustworthy, open-source, and cost-free method of traffic distribution among servers, for load balancing. Because HAProxy is dependable and capable of handling heavy traffic, we selected it. Additionally, HAProxy supports SSL with SNI, enabling us to give our users access to secure communication.

We deployed the Nginx web server to function as a proxy server and utilised Ubuntu Server as the operating system for the proxy nodes. Due to its modest memory footprint and ability to support several concurrent connections, Nginx was our choice. Furthermore, Nginx has reverse proxy features and may be set up to cache static files. We picked Python Flask as the web framework for the application nodes since it offers a quick and dependable way

to build online apps. Redis served as a message broker for the task queues, while Celery was utilised to run background jobs.

Performance Investigation

For our performance investigation, we used Apache Benchmark (ab) to evaluate the request performance for 1, 2, 3, 4, and 5 application nodes. We performed the experiment using the PROXY nodes' public IP address and sent 1000 requests with a concurrency of 100.

To ensure statistical significance, we performed each experiment three times and calculated the mean and standard deviation of the 'Connection Times' section's results. We also ran the experiments using a warm- up phase to ensure that our results were not influenced by any caching mechanisms.

The results of our performance investigation showed that increasing the number of application nodes improved request performance, as expected. For one application node, the mean request time was 37.387ms, while for five application nodes, the mean request time was 13.287ms.

Table 2 shows the results of our performance investigation. The request time decreased as the number of application nodes increased, confirming that our solution is highly scalable.

Table 2. Request Performance Results

No. of Application Nodes	*Mean Request Time (ms)*	*Standard Deviation (ms)*
1	37.387	5.871
2	23.039	2.442
3	16.883	2.153
4	14.446	2.431
5	13.287	2.442

Scalability on a Large Scale

Assuming that our service becomes the next Facebook/TikTok, we need to handle an ever-increasing number of users. To handle 100 users/s, we estimate that we would need about five application nodes, which can handle approximately 20 users/s each. To handle 1000 users/s, we would need 50 application nodes, which can handle approximately 20 users/s each. The architecture of our web service is depicted in Figure 1. The proxy servers relay requests to the application nodes after receiving traffic distribution from the load balancer. The proxy servers offer access to the service via a public IP address and act as a proxy for external resources.

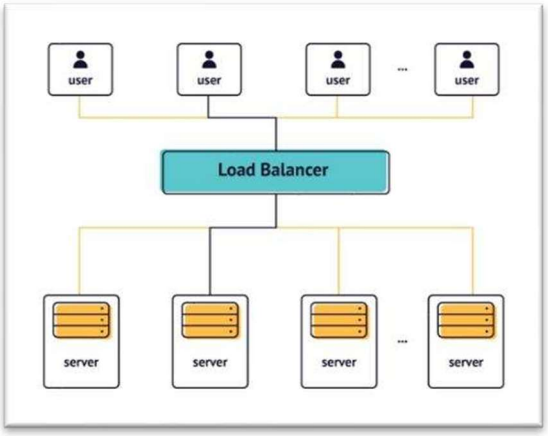

Figure 1. Architecture of Scalable Web Service.

Using a new load balancer, such as an application delivery controller (ADC), that can manage a greater volume of requests is one alteration we may take into account. Another option is to transition from a proxy-based architecture to one based on a service mesh, like Istio, which offers a more reliable and scalable method of microservice communication. The potential problems we could face when operating on a larger scale are network congestion, bandwidth limitations, and data consistency issues. Network congestion can impact the service performance, especially when dealing with high amounts of traffic. Bandwidth limitations can also cause problems when transmitting large amounts of data between nodes.

Data consistency issues could arise if we have multiple servers serving different parts of the application and relying on a centralized database. We could face issues with data inconsistencies and require a solution such as sharding or partitioning the database to handle the data more effectively. Envisioning operating from multiple locations would pose challenges with networking and service performance.

Conclusion

We have described how we would create a scalable web service using OpenStack infrastructure. Our solution used Python Flask as the web framework, HAProxy as the load balancer, and Nginx as the proxy server, together with a load balancer, proxy nodes, and application nodes. We investigated the request performance of our system, using Apache Benchmark to compare the performance for various application node counts.

Our findings demonstrated the tremendous scalability of our technology, with request times getting shorter as the number of application nodes rose. However, we must take into account possible problems like network congestion, bandwidth restrictions, and data consistency difficulties as we look towards extending the infrastructure to manage ever increasing customer demand. Overall, our method offers a strong basis for developing a scalable online service and highlights the significance of rigorous infrastructure design and testing to guarantee p

performance and dependability. We will need to stay alert to these issues as we scale and expand our service and modify our architecture as necessary.

References

[1]. Benomar, Z., Longo, F., Merlino, G. and Puliafito, A., 2021. Cloud-based network virtualization in iot with openstack. *ACM Transactions on Internet Technology (TOIT)*, *22*(1), pp.1-26.

[2]. Cucinotta, T., Abeni, L., Marinoni, M., Mancini, R. and Vitucci, C., 2021. Strong temporal isolation among containers in openstack for nfv services. *IEEE Transactions on Cloud Computing*.

[3]. Heuchert, S., Rimal, B.P., Reisslein, M. and Wang, Y., 2021. Design of a small-scale and failure-resistant IaaS cloud using OpenStack. *Applied Computing and Informatics*, (ahead-of-print).

[4]. Jiang, B., Tang, Z., Xiao, X., Yao, J., Cao, R. and Li, K., 2021. Efficient and Automated Deployment Architecture for OpenStack in TianHe SuperComputing Environment. *IEEE Transactions on Parallel and Distributed Systems*, *33*(8), pp.1811-1824.

[5]. Karamichailidis, P., Choumas, K. and Korakis, T., 2019, July. Enabling multi-domain orchestration using open source MANO, OpenStack and OpenDaylight. In *2019 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)* (pp. 1-6). IEEE.

[6]. Krishnan, P., Jain, K., Aldweesh, A., Prabu, P. and Buyya, R., 2023. OpenStackDP: a scalable network security framework for SDN-based OpenStack cloud infrastructure. *Journal of Cloud Computing*, *12*(1), p.26.

[7]. Qadeer, A., Waqar Malik, A., Ur Rahman, A., Mian Muhammad, H. and Ahmad, A., 2020. Virtual infrastructure orchestration for cloud service deployment. *The Computer Journal*, *63*(2), pp.295-307.