A Comparative Analysis of AWS Cloud-Native Application Deployment Model

Chapter · October 2022

DOI: 10.1007/978-981-19-2445-3_29

2 authors:

Khandakar Razoan Ahmed
Therap BD Ltd.
2 PUBLICATIONS **5** CITATIONS
SEE PROFILE

Md. Motaharul Islam, PhD
United International University
103 PUBLICATIONS **905** CITATIONS
SEE PROFILE

A Comparative Analysis of AWS Cloud Native Application Deployment Model

Khandakar Razoan Ahmed[1][0000−0001−6086−8723] and Md. Motaharul Islam[2][0000−0002−8030−3225]

[1] Associate System Administrator
Therap BD Limited, Dhaka, Bangladesh
razoanahmed421@gmail.com
[2] Dept. of Computer Science and Engineering
United International University, Dhaka, Bangladesh
motaharul@cse.uiu.ac.bd

Abstract. Application deployment in cloud computing has become a widely used system where we follow some specific models. For a long time, we have deployed our application in virtual machines. Nevertheless, nowadays the Docker container model is gaining popularity due to its important features. Therefore, we need to know which model is better in terms of request-response, which model is error-prone and gives better throughput. In our paper, we have done a comparative study between these two native cloud application deployment models. The main parameters of this study are HTTP response in a fixed time, error percentage, and throughput. We have used Amazon EC2 service for the Virtual Machine model and Amazon ECS service for the Docker Container model in the implementation part and Apache JMeter tool in the test part. In our study, we have found that the docker container has performed better in terms of total and average responses to the HTTP request and throughput.

Keywords: Virtual Machine · Docker Container · Amazon EC2 · Amazon ECS · Cloud · AWS · Apache JMeter.

1 Introduction

Cloud computing depends on shared computing resources through the internet instead of having local machines or servers. Cloud computing is the most ideal choice for storing, handling, and maintaining millions of data [1]. Now we can store and retrieve our data from anywhere at any time [3]. A cloud can be private or public. There are some service providers available who give us the public cloud service, for example, Amazon Web Services (AWS), Google Cloud Platform, and Microsoft Azure [2]. Among them, AWS is the better public cloud service provider in light of four advantages like fault-tolerant, high scalability, resilience, and availability zone. Fault tolerance is the property that empowers a framework to keep on working appropriately in case of the disappointment of a portion of its segments. Scalability is the attribute of a system that depicts its capacity to adapt and perform well under an expanding workload likewise will have the option to increment or lessen its level of performance

according to the demands. Resilience is the ability to recoup from troublesome and harder conditions. AWS gives us the resources we need as well as gives us the highest fault-tolerant advantage, high scalability, and best resilience facilities. AWS also has the most availability zone 48 with the 20 regions all over the world. So, we can choose any zone from any region where we want to deploy our application. Also, our data will be stored in another two data centers for backup, so under any misfortune or pandemic, our information will be protected.

AWS provides us more than 90+ services like Amazon Elastic Compute Cloud (Amazon EC2), Amazon Elastic Container Service (Amazon ECS), AWS Lambda, Amazon Simple Storage Service (Amazon S3), AWS IoT Greengrass, AWS CloudFormation, virtual private cloud (VPC), AWS Identity and Access Management (IAM), etc [11][8]. We have utilized a large number of them in our project. Amazon EC2 is a web service that gives us a secure, resizable process limit in the cloud [11]. EC2 is intended to make web-scale [10] distributed computing simpler for developers. It does not just give us the unlimited authority of computing resources yet, it lets us run on Amazon's proven computing environment. Amazon VPC gives us serious security features, for example, security groups and network access control lists, to empower inbound and outbound filtering at the instance and subnet level [11]. Amazon S3 is a scalable, rapid, web-based cloud storage service intended for online backup and archiving of information and applications on AWS. Additionally, it is an object storage administration that offers industry-driving scalability, data accessibility, security, and performance. ECS is an Amazon container administration service. AWS ECS is an exceptionally adaptable, super-quick, and fault-tolerant service help that makes it easy to create, launch, stop, and maintain our containers on the cluster[14]. We can run our tasks on a serverless infrastructure (managed by AWS Fargate) for full command over our infrastructure. We additionally can run our services on a cluster of EC2 instances that we can oversee.

To deploy applications, the cloud is the briskest, reliable, and useful approach. From the very beginning, people deploy applications on-premises basis. For quite a while, this was the main method of deployment. On-premises always require a greater expense in getting off the ground as hardware and installation are necessary for the housing of on-site servers. In addition, maintaining the infrastructure and wasting resources are some big disadvantages of on-premises [13]. To diminish these issues Cloud computing is very much helpful. As every year, many applications are deployed so it is necessary to look for smooth computation for large data and performance of those applications. To deploy any kind of application or software we have four kinds of deployment models such as Public Cloud, Private Cloud, Community Cloud, and Hybrid Cloud. Among them, the public cloud is the best model [13]. Some of the advantages are unsophisticated installation and use, easy access to data, cost-effectiveness, continuous operation time, eliminated need for software, and simple admittance to data, cost-effectiveness, and scalability [13].

A virtual machine (VM) is an image file that is overseen by the hypervisor [1]. Hypervisor permits the computing resources (RAM, CPU, Memory, Storage, etc.) of the server to every single Virtual Machine [12], and keeps them isolated to avoid interference. VM represents the behavior of a separate machine and the capacity to perform

tasks as a separate machine. In our proposed project, we have built the infrastructure for deploying our web application in the VM. AWS provides us the VM's services by the Amazon EC2 service and we have utilized EC2 instance for the execution part.

Docker containers are primarily used to deploy our applications in microservice-based architecture [9]. We divide the whole service into small parts that we call microservices [4]. The benefits of Docker containers are fast speed, lightweight, compactness, quick conveyance, and scalability [2][7]. Because of these advantages, it becomes attractive to develop containerized services nowadays. Docker container is less costly than the virtual machine [9]. In a container environment, we virtualized the applications and executed them [6]. Containers are running in an isolated way on top of the operating system's kernel [7]. Docker container guarantees that our application is going to work consistently in every environment. In our proposed project, we are going to build the infrastructure for deploying our web application by using the Docker container model. AWS provides us the Amazon ECS services and we have used Cluster and Task Definition of the Amazon ECS services for the implementation part [7]. The healthcare application we have deployed has been customized by us. This website is a healthcare website that we have named WeCare Hospital. We have deployed this website in both models and afterward, we need to see how they perform.

In our paper, we have proposed a comparative study between two native cloud application deployment models. Cloud-native application deployment implies where we build the whole infrastructure in the cloud with the help of services of cloud service providers like AWS. And afterward, we deploy our application in that infrastructure for quick and easy deployment, superfast performance, and simplicity to oversee. In this deployment, we follow some models like Virtual Machine, Docker Container, etc.

This paper aims to solve two challenges of the native cloud application deployment. First, our paper helps to present the accurate way of deploying applications in the Virtual Machine and Docker Container models which will also guide about deploying applications using Amazon EC2 and Amazon ECS services. Consequently, it will give proper knowledge about which models perform better in which performance test. There are lot of work we have done to build up the infrastructure and deploy our application but some major contributions are:

- We have used AWS services to implement this comparative study such as Amazon EC2, Amazon ECS, CloudFormation, Amazon S3, VPC, and IAM.
- To build the infrastructure in AWS we have utilized the CloudFormation service by using the automation concept.
- The healthcare website template that we have deployed in the models is customized by us.
- We have used the Apache JMeter tool in the performance comparison test.
- We have composed a realistic performance test plan and divided it into three-part such as Linear Load, Step Load, and Peak Load.

The rest of the paper is organized into different sections. Section 2 discusses the previous research works. In section 3. A and 3. B, we describe our system architecture. Section 4. A and 4. B, we implement and describe our realistic test plan using Apache

JMeter. Section 5 is about experimental evaluations. Finally, the last section puts a conclusion to our research work.

2 Related Works

In [1], authors have analyzed the performance of VM and containers with the same hardware and software configuration based on performance efficiency, security, storage, isolation, and networking.

In [2], a performance comparison has been shown between Linux containers and VM where the author gave the reasons why the container is more useful and attractive to software developers and infrastructure administrators. And later conducted a comparison on the basis of performance and scalability. In [9], authors have described Docker System, Docker's applications, and advantages in cloud Computing through an application instance.

In [5], the authors proposed a comparative study between VM and Docker Containers. They discussed some basics of VM, Docker containers, virtualization techniques, and their advantages and disadvantage. Lastly compared them based on booting time, standardization, portability, security, low redundancy, resource distribution, etc.

In [6], authors have operated a comparative study between Xen and Docker. They have used one extensive micro-benchmarks to compare virtualization upward of distinct elements on a host system between these two virtualization technologies. They have also added another two macro-benchmarks called HPL and YCSB where HPL is for a case of high-performance computing applications and YCSB for a case of online transaction processing applications, to understand the virtualization impact on real-world application.

Though in [7], authors have pointed that services that have been deployed by AWS ECS perform worse in comparison with services that have been deployed by Amazon EC2 VMs. They have conducted the performance test based on throughput, response time, and CPU utilization. However, in our paper, we have tried to conduct the performance test with a proper testing plan and strategy where we have conducted real-life scenario-based performance tests.

In [8], the authors presented a performance study to examine the proper way to utilize containers from different viewpoints. They have led some experiments to measure performance variations between application containers and system containers. Furthermore, they have inspected the service quality of ECS and Google Container Engine.

3 System Architecture

3.1 EC2 based System Architecture

The EC2 based cloud-native application system architecture is introduced in Fig. 1. We have deployed a sample healthcare website application and analyzed this web application's performance by Apache JMeter from our local machine. We need to do

this performance test through the Internet. So we have to set up Apache JMeter in our local machine and then we can send an HTTP request through the internet to our web address. Here we need Amazon Route 53 service. Amazon Route 53 is a highly accessible and scalable cloud DNS web service. Amazon Route 53 connects user requests to the infrastructure successfully. Those infrastructures run in AWS – such as Amazon EC2 instances, Elastic load balancers, or Amazon S3. Amazon Route 53 service connects our HTTP request to our infrastructure so that we can access the hospital website. When our HTTP request tries to access the hospital application hosted in EC2 Webserver, HTTP requests are served through the Internet gateway attached in the VPC Network. Internet gateway sends the HTTP request to the EC2 instance or machine which is working as a web server.

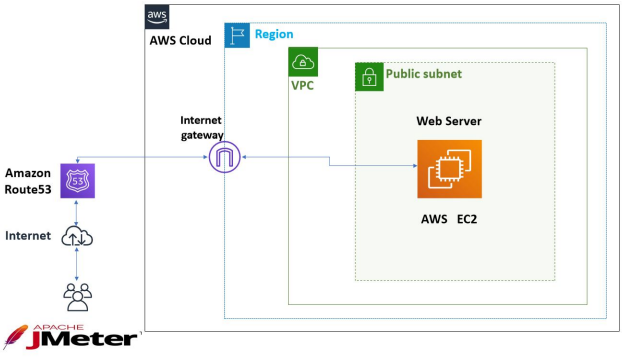

Fig. 1. EC2 System Architecture.

In Fig. 1. the peripheral box signifies the AWS cloud. The immediate inner box signifies a region that implies we are working in a specific region of the AWS cloud. We have deployed EC2 instance in the closest location which is in Mumbai(India) and the short name of the region is ap-south-1, Mumbai. This region has two Availability zone which implies there are two data centers in this region and correspondingly AWS has more than 48 availability zone where we can deploy our application. As Fig. 1. shows we have built a VPC network for the network requirements of the application. There is a public subnet in the VPC to isolate network traffics and the EC2 box is placed in the public subnet. Our hospital website application is running on the Apache HTTPD service on our web server.

3.2 ECS based System Architecture

The ECS-based cloud-native application system architecture is introduced in Fig. 2. We have deployed our web-application and analyzed the performance by the Apache JMeter from our local machine. The main difference between Fig. 1. and Fig. 2. is the ECS service in the EC2 infrastructure. Here, we have used ECS service in the EC2 based infrastructure to implement the Docker container model.

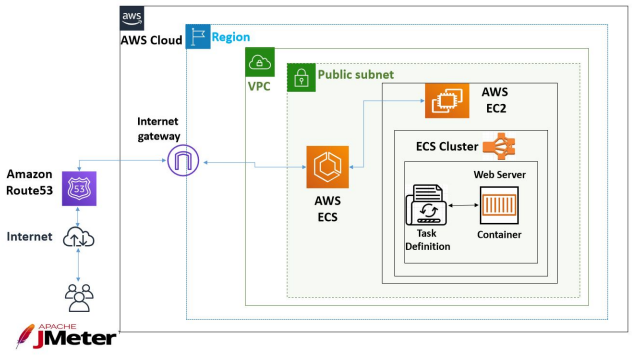

Fig. 2. ECS System Architecture.

To deploy our web application in the ECS service, first, we need an Amazon EC2 instance where we can create an ECS Cluster. Amazon ECS cluster is a logical gathering of tasks or services [14]. We can use one or more EC2 instances as container instances with the cluster to run the task definition. As we know cluster manages a group of services, so to prepare our application to run on ECS, we need to create a task definition which we can call a blueprint for our application. Task definition is a text file, in JSON format. It depicts one or more containers, up to a limit of ten, that structure the application. We can determine various parameters in the Task definitions for our application. The parameters that we need to determine are which containers to utilize, how much data volumes ought to be utilized, and which ports ought to be opened for the application. The particular parameters for the task definition rely upon the requirements of our application. So this cluster will manage the task definition that we have created and register the EC2 instance with the container where our webserver is running.

4 Implementation Details

4.1 Infrastructure Build

For the implementation part, we have utilized some AWS services. First of all, to construct the network part, we have used VPC service where we have set the subnets, route tables, internet gateway, Elastic IPs. To store our data and file we have utilized the S3 bucket where we need to create our bucket and store the data. We have used AWS IAM which is a web service. By IAM we obtain secure control access to our AWS resources. The complete infrastructure is built using the automation concept and the automation is done by using an AWS service called CloudFormation. This service is considered as Infrastructure as a Code (IAAC) service. We have made a YML layout document for building the EC2 and ECS administration. In the format document, we have referenced and selected everything that we have to do manually for making these services. In table 1, we can see, we have chosen the T2.Nano type instance for

EC2 service which consists of 1 vCPU and 500 MB Memory. For our project, this type of instance is a decent choice. But for the ECS service, we have chosen a T2.2xlarge type instance to consist of 8 vCPUs and 32 GB Memory [14]. To support the microservices we need to choose this big instance yet we have restricted the instance to create 1 container consisting of 1vCPU and 500 MB memory so we can play out the test with a similar sort of condition.

Table 1. Environment and Tools

AWS Services	AWS IAM
	Amazon ECS
	Amazon VPC
	Amazon EC2
	Cloudformation
Tools	Apache JMeter
	Visual Studio Code
Network performance	EC2: Low to Moderate
	ECS: Moderate
Instance Type	EC2: T2.Nano: 1 vCPU and 500MB
	ECS: T2.2xlarge: 8 vCPU and 32 GB

4.2 Performance Test

We have stored the healthcare website template in the S3 bucket and run it on the webserver. The website has been customized by us for the test to create a real-life scenario-based comparison. For the comparison test part, we have utilized Apache JMeter [15]. Apache JMeter is an adaptable and smart tool where we can test the performance of both static and dynamic web applications. We expect to lead a wise and realistic test plan. Our customized website has run on the webserver on both models and we have tested on both models. In the deployment part, we have selected the ap-south-1(Mumbai) region which is the closest to our destination. However, at the test period, we have selected the us-east-1(North Virginia) region in the JMeter with the goal that Latency can be there and make this test realistic. To lead this test, we have to pick and select some particular ideas and plans [15]. We have set the test on three steps to take the test more realistic and real-life scenario-based. The three steps are Linear Load, Step Load, and Peak Load. Native JMeter does not support three complex load tests like this so we have used a third-party plugin called Ultimate Thread Group. This plugin helps to set the thread group parameters and examine the system behavior after every new group of users is added. JMeter does not offer a native listener to identify the number of active virtual users for a thread group at every given point of time, so we have utilized another third-party plug-in called Active Threads Over Time. It presents this number on some simple yet concise graphs.

In our test plan, the initial step is Linear Load where both of the models give their basic performance because this is one of the typical performance testings. Here

we have chosen 500 threads implies 500 virtual users for the test. We have set the parameters that this test begins with 10 seconds of initial delay and then it increases from 1 to 500 in a few seconds. After that 500 threads hit consistently and we need to see how our web server performs. The load had been generated for 600 seconds. In fig 3 you can see the active threads over time where Jmeter is visualizing how the load is generating. Jmeter also collects the data that during the test time how many requests are being responded from our website in total, how many requests are being responded in every second how many seconds this website is taking to respond to the request, and how many errors are being created by the webserver. This Linear Load test runs in both models EC2 and ECS.

Secondly, We need to conduct a step load test. Here the initial stage is similar to the Linear load test but after a particular time, one more thread group is added. In this way, a total of 6 threads is added through a loop. For example, we have set the initial parameters as the initial stage is 500 threads send the HTTP request to our webservers in every second in 70 seconds afterward one more step is added where more 500 threads start to hit with 15 seconds initial delay and hit for 220 seconds. Another 500 threads start to hit after 30 seconds with the initial delay of 60 seconds and hit for 180 seconds. In this way total, 6 thread group send the request to our website so in this Step load test our webserver has to respond to almost 3000 threads and this increase from 500 in step by step. We need to check our websites how it keeps pace and increases it with the requirement. Ultimate Thread Group plugin helps to set the thread group parameters and examine the system behavior after every new group of users is added.

Lastly, we have utilized a short time intensive load test called Peak load. It creates an unusual environment to evaluate the reaction of the server. The server needs to respond to such extreme load jumps and we can see that in a normal load test when an extreme situation takes place how our webserver faces it and also how well it recovers from the traffic burst. Both models react differently so we need to monitor their reactions. In the Peak load test, JMeter starts with the usual 500 threads with the 0 seconds initial delay. 500 users hit our website for 600 seconds and then suddenly the threads number jump from 500 to 3000 at a glance and then we perceive how the server responds in this situation and we assemble the data in the meantime.

5 Experimental Evaluations

This section describes the way we have collected data. We have applied the Apache JMeter tool for this performance test. In this section, we also examine our data, and based on the data we decide which models perform better.

In Fig. 3. we can see how it looks like when the Step load is running. In the y-axis number of active threads are from 0 to 3000 and in the x-axis elapsed time is from 0 - 12 minutes. This graph shows how many threads are hitting per second to the server and how 500 threads group joins after a specific time. In this way, a total of 3000 threads hit the server every second and it continuously hits for 12 minutes. The whole step load test has worked according to our plan just like other load tests.

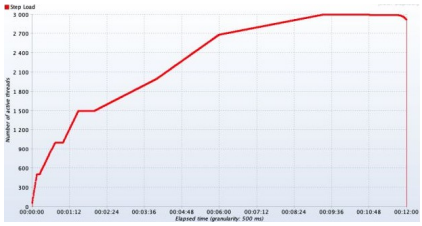

Fig. 3. Active threads over time EC2 Step load

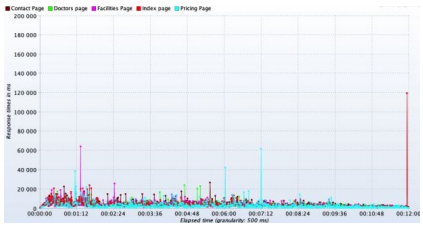

Fig. 4. Response time over time EC2 Step load

In Fig. 4. we can see the response time over time of the EC2 model in the Step Load test. In the graph, the x-axis presents elapsed time from 0 to 12 minutes and the y-axis presents response time in milliseconds. As the graph shows EC2 performs well in the Step Load test and it has taken an average of 20000 milliseconds which means an average of 2 seconds to respond to all the HTTP requests. Only in some moments when new chunks of threads have added server then it takes 4 to 6 seconds. But at one moment it takes 10 seconds to respond though it has happened only one time so we can overlook it.

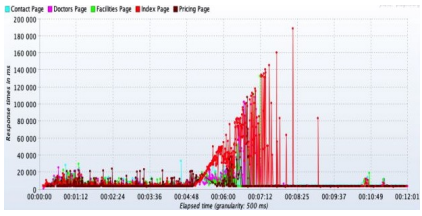

Fig. 5. Response time over time ECS Step load

In Fig. 5. ECS is surprisingly performed better than the EC2 model because as we can see ECS-based webserver has taken almost 0 seconds sometimes 2-4 seconds to respond to the HTTP request whereas EC2 has taken an average of 2 seconds sometimes 4 to 6. The threads increase on a stepped basis but still, it never reacts rather performs constantly.

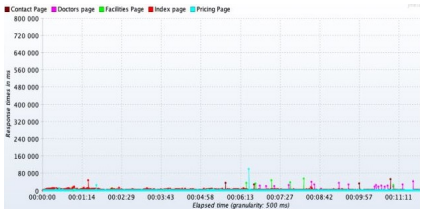

Fig. 6. Response time over time EC2 Peak load

In Fig. 6. EC2 performs well from the start but when the threads become 3000 from just 500 it doesn't react well. We can after the peak load average response increase much higher which means the server has become slow now it has responded at an average 12-14 seconds.

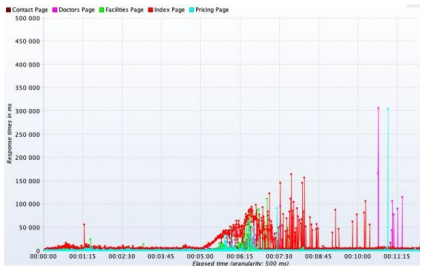

Fig. 7. Response time over time ECS Peak load

ECS performs significantly well in Fig. 7. We can see the server was responding at an average time of almost 0 seconds but when the peak started it gets slower and it was tough but still, they responded at an average of 8-10 seconds which is much lesser than EC2 but at some point, it took more than EC2 model but yet that is in very few times.

In TABLE 1 we can see ECS performs superior to EC2 as far as the total response in the entire test time. In the Linear Load test, EC2 responded to 33946 requests whereas ECS responded to 78163 requests which are practically twofold in contrast with EC2. In the Step Load, EC2 responded to 29043 requests though ECS reacted 64094 requests which are a similar practically double in contrast with EC2. In Peak, Load EC2 responded to 18337 requests while ECS responded to 34003 requests which are again almost double in comparison to EC2. In error percentage, EC2 made 0 % blunder while ECS made 1.05 % mistake in the Linear Load. In the Step Load, EC2 made a 0.01 % error which is right around 0 while ECS made a 2.22 % mistake. In the Peak Load, EC2 made a 2.93 % blunder which is nearly 3% though whereas ECS created an 8.86 % error which is almost three times then EC2. In Throughput EC2 responded to 47.77 requests per second while ECS responded to 92.10 requests per second which are twofold than EC2 in the Linear Load. In the Step Load, EC2 responded to 35.89437

The table 2 below shows the data that we get from the Performance test. In the data we have presented Samples, Average, Error and Throughput.

Table 2. Performance Analysis Table

Model	***Label***	***Samples***	***Avg***	***Error(%)***	***Throughput***
EC2	Linear Load	33946	1893	0.00 %	47.77
EC2	Step Load	29043	3365	0.01 %	35.89
EC2	Peak Load	18337	6747	2.93 %	25.41
ECS	Linear Load	78163	4945	1.05 %	92.10
ECS	Step Load	64094	1816	2.22 %	80.95
ECS	Peak Load	34003	44776	8.86 %	34.45

requests per second whereas ECS responded to 80.95 requests per second which are practically triple than EC2. In the Peak, Load EC2 responded to 25.41 requests every second whereas ECS responded to 34.46 requests every second which is way more than EC2.

Now, if we consider the performance the number of requests reacting more than twofold from EC2. Additionally in the throughput part, ECS performs much more than double. Be that as it may, in error percentage, ECS is creating more than EC2, particularly in Peak Load when an abrupt traffic burst happens it begins to make far more blunder than EC2. However, when we run our application first thing we investigate is what number of requests it can respond and in that manner, ECS is far better than EC2. Moreover, traffic burst is certainly not a customary thing so we need to see which method gives us better performance most of the time. Here one point is ECS is getting better network performances which are moderate. But EC2 getting low to moderate network performances. However, to support docker containers in the EC2 instance we need a better network performance. And, it is not much more than ECS. Even after getting slightly better network performances, ECS is taking much fewer resources than EC2. Furthermore, suppose if one container in one instance can give a better response than the entire EC2 instance then if we use 10 containers in the same instance it can give multiple times better performance which is the fundamental advantage of ECS. So overall the conversations, we can concur that the Docker Container model is performing superior to Virtual machines as far as HTTP reactions and throughput. But, Virtual Machine performs better than Docker in terms of error generalization.

6 Conclusions

In this paper, we have presented a comparative study between two native cloud application deployment models. We have developed the infrastructure for those models and deployed our healthcare application there. We have to lead a comparison test that how both models perform by using the Apache JMeter tool. Additionally, to make the performance test more realistic we have utilized a three-step based test plan. The parameters of our comparison test were HTTP response, error percentage, and

throughput. Docker container performed essentially better than Virtual Machine in the HTTP response and throughput part although Virtual Machine did better than Docker Container in the error percentage part.

References

1. R. K. Barik, R. K. Lenka, K. R. Rao and D. Ghose, "Performance analysis of virtual machines and containers in cloud computing," 2016 International Conference on Computing, Communication and Automation (ICCCA), Noida, 2016, pp. 1204-1210.
2. A. M. Joy, "Performance comparison between Linux containers and virtual machines," 2015 International Conference on Advances in Computer Engineering and Applications, Ghaziabad, 2015, pp. 342-346.
3. H. Zeng, B. Wang, W. Deng and W. Zhang, "Measurement and Evaluation for Docker Container Networking," 2017 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), Nanjing, 2017, pp. 105-108.
4. A. K. Yadav, M. L. Garg, Ritika, "Docker Containers Versus Virtual Machine-Based Virtualization," 2018 International Conference on Emerging Technologies in Data Mining and Information Security (IEMIS), Kolkata, 2018, pp. 141-150.
5. B. Wang, Y. Song, X. Cui and J. Cao, "Performance comparison between hypervisor- and container-based virtualizations for cloud users," 2017 4th International Conference on Systems and Informatics (ICSAI), Hangzhou, 2017, pp. 684-689.
6. T. Salah, M. J. Zemerly, C. Y. Yeun, M. Al-Qutayri and Y. Al-Hammadi, "Performance comparison between container-based and VM-based services," 2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN), Paris, 2017, pp. 185-190.
7. D. Liu and L. Zhao, "The research and implementation of cloud computing platform based on docker," 2014 11th International Computer Conference on Wavelet Actiev Media Technology and Information Processing(ICCWAMTIP), Chengdu, 2014, pp. 475-478.
8. Sutton, R.S., Barto, A.: Reinforcement Learning: An Introduction. MIT Press (1998)
9. Islam, M. M., Khan, Z., Alsaawy, Y.: An Implementation of Harmonizing Internet of Things (IoT) in Cloud. Lecture Notes of the Institute for Computer Sciences, Social Infor-matics and Telecommunications Engineering, pp.3-12. Springer, Cham (2018).
10. Islam, M.M., Khan, Z. and Alsaawy, Y. A framework for harmonizing internet of things (IoT) in cloud: analyses and implementation. Wireless Netw (2019).
11. B. Ruan, H. Huang, S. Wu and H. Jin, "A Performance Study of Containers in Cloud Environment," 2016 Asia-Pacific Services Computing Conference (APSCC), Zhangjiajie, 2016, pp. 343-356.
12. C. Qu, R. N. Calheiros and R. Buyya, "Auto-scaling web applications in clouds: A taxonomy and survey", ACM Comput. Surv., vol. 51, no. 4, Sep. 2018.
13. Medium Homepage, www.medium.com, last accessed on 30 August, 2020
14. AWS Homepage, www.aws.amazon.com, last accessed on 30 August, 2020
15. DZone Homepage, www.dzone.com, last accessed on 30 August, 2020