Performance Comparison between Container-based and VM-based Services

Tasneem Salah, M. Jamal Zemerly, Chan Yeob Yeun, Mahmoud Al-Qutayri, Yousof Al-Hammadi
Department of Electrical and Computer Engineering
Khalifa University of Science Technology and Research
PO Box 127788 Abu Dhabi, UAE
{tasneem.salah, jamal, cyeun, mqutayri, yousof.alhammadi}@kustar.ac.ae

Abstract—**These days, microservice architecture is widely used in the design and development of many real-time, critical, and large-scale online services. These services are typically deployed using Docker containers on cloud platforms. Container technology supports the deployment of these services with high portability, scalability, and performance, when compared to deploying them using virtual machines (i.e. VM-based services). It is widely known fact that container-based services give better performance than VM-based services. However, we show in this paper that services deployed using Amazon AWS ECS (EC2 Container Service) surprisingly perform significantly worse when compared with services deployed using Amazon EC2 VMs. We study and quantify the performance difference in terms of throughput, response time and CPU utilization considering different deployment scenarios.**

Keywords— Cloud computing, Docker containers, Virtual Machines, Microservices, Performance Evaluation.

I. Introduction

The cloud has become the preferred platform for deploying applications and services designed with microservice architecture. The aim of microservice architecture is to break down the application into a set of smaller independent services [1]. This architecture enables application resilience, scalability, fast software delivery and the use of minimal resources. Such features can enhance the development method followed in complex systems which require continuously changing algorithms. Such algorithms can be found in real-time services as those found in recommender systems and stock market predictions.

Docker containers are commonly used to support the deployment of services developed with microservice architecture [2]. Docker containers are lightweight, highly portable and scalable [3]. Such features become attractive to develop containerized services (or microservices). In the literature, container-based services are reported to always outperform VM-based services. For example, it was shown in [4-8] that containers outperform virtual machines in terms of execution time, latency, throughput, power consumption, CPU utilization and memory usage. However, according to Amazon AWS documentation [9], it was reported that Docker containers are deployed on top of EC2 (Elastic Compute Cloud) virtual machines (VMs), and not on bare-metal hardware. This contradicts the common practice of container deployment which is widely adopted in the literature of deploying container on bare-metal hardware. This was the primary motivation for our research work in which we aim to investigate and assess the performance difference resulting from deploying services using VMs (virtual machines) and using Docker containers.

In our experimental study, we chose Amazon Web Services (AWS) cloud platform, as Amazon cloud has been thus far the most popular Infrastructure as a Service (IaaS) cloud provider. More specifically, and in Amazon AWS taxonomy, we focus to evaluate the performance of web services using: (1) AWS EC2 Container Service for container-based deployment, and (2) AWS Elastic Cloud Compute (EC2) for VM-based deployment.

The main contributions of this paper can be summarized as follows:

- We provide a methodology and experimental setup that can be used in general to assess the performance of container-based and VM-based services to be deployed on any cloud environment.
- We present a comparative performance study and results considering different type of deployment scenarios for a web service
- We show that the performance of VM-based web services is surprisingly and substantially superior when compared to container-based series.

The remainder of the paper is organized as follows. Section II provides a brief background and related work on microservices, virtual machines, Docker containers and their expected performances. Section III describes our methodology including the test environment setup, tools, and the different test scenarios of EC2 and ECS. Section IV presents and provides interpretations for the experimental performance measurements obtained from the three test scenarios. The measurements are reported in terms of throughput, CPU utilization and response time. Finally, Section V concludes our performance study.

II. Backgruond and Related Work

In this section, we provide a brief background about microservices architecture, Docker containers, and virtual machines. We also described related work reported in the literature with regards to performance of Docker containers when compared to the performance of VMs.

Microservices architecture is often compared in the context of the traditional SOA (Service Oriented Architecture)

architecture. SOA allows services to be loosely coupled, reusable and dynamically assembled which supports the changing business environment [10]. SOA supports explicit boundaries between autonomous services located on different servers to fulfil application requirements. The most crucial drawback that called the need for enhancing SOA is its centralized integration [11]. Some prefer referring to microservices as an approach to build SOA. Others stated that SOA architecture can be referred to as the monolithic representation of an application or as a compressed view of microservices. There are strong motivations to consider splitting a monolith application some of them would be to ease up service changing, fast delivery of features and functions, elastic scaling and fault tolerance [12][13].

For scalability and ease of replications and migration, microservices get deployed in a cloud environment and run within a virtual machine. Virtualization is the process of splitting up the physical machine into several virtual components in which each can host different software to run within. The authors in [3] presented an example on how to build a custom VM image. Some examples on virtualization might include VMWare and AWS (Amazon Web Services) virtual machine technologies. Unfortunately building them takes a long time and some images might result with a large size. Transferring those large images across the network will result in a complex activity to be performed. The overhead is caused by the Hypervisor layer as illustrated in [3] which is responsible for performing virtualizing and managing physical resources. These drawbacks entail the need for container technologies such as Linux Containers [14]. Containers provide a separate space for the processes without the need for a hypervisor to control the machines. Moreover, Docker platform contributed on building lightweight and portable containers enabling to run on any infrastructure.

Docker is an open source platform focuses on building, shipping, and running distributed applications within containers to be located on either on local machines or the cloud. In fact, one Virtual Machine (VM) can host multiple containerized microservices. Containers provide a very appealing feature which is packing up the service with its dependencies into a single image also referred to as code portability [15]. Fig. 1 illustrates the difference in the architecture approach for virtual machines and Docker container technology. The structure of the containers architecture approach seen in Fig. 1 provides portability and efficiency and less overhead as the hypervisor layer is eliminated compared to virtual machines.

Many performance comparisons were reported in the literature on virtual machines and containers especially Docker containers. In most of the comparisons, Docker containers performance has been reported to be superior when compared with virtual machines. Services deployed using containers are expected to take less execution time thus resulting in less latency over virtual machines as in [5]. In addition, containers were shown to introduce lower power consumption over virtualized technologies [4].

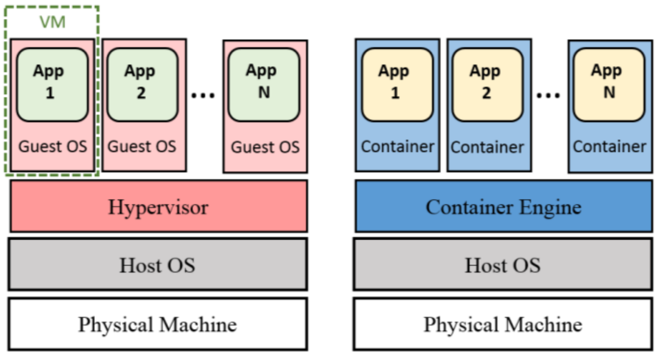


Fig. 1: Common approach of hosting of VMs vs. Containers

Moreover, in [6][7], authors showed that services deployed using Docker containers outperform VMs in terms of overall throughput. Significant performance overhead was also reported for real-time applications when deployed using virtual machines, which was evident in the resulting CPU utilization and memory usage [8]. Furthermore, according to Amazon [9], Docker containers are deployed on top of EC2 virtual machine, which is against the common practice of deployment as depicted in Fig. 1. Fig. 2 shows the container deployment approach adopted by Amazon cloud.

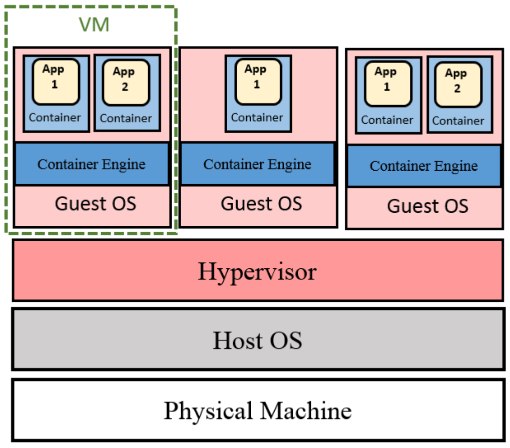


Fig. 2: Hosting of Containers in Amazon cloud

As shown in Fig. 2, an overhead is clearly introduced which will impact the overall performance of the running services within containers. Amazon AWS may have adopted such approach for good reasons. First, it is less costly for a service provider to make more use of an existing infrastructure (i.e. EC2 VMs) and leverage to provide additional service (i.e., container ECS). Second, customers will be able to easily manage their container instances as ECS offers high management and auto-scaling capabilities which are basically the features and capabilities of EC2 VMs. Third, Amazon argues that deploying containers using EC2 VMs would ease management and increase deployment speed while maintaining flexibility [9].

Clearly, in Amazon cloud, the performance of containerized services has not been considered a top concern [9]. The deployment of containerized services within virtualized machines is expected to result in an overhead leading to a significant increase in response time and degradation in the overall performance of the containerized service.

We believe it is important to quantify the performance difference. This can be key to the deployment real-time applications and services. To the best of our knowledge, no experimental work has been conducted on measuring the performance of VM-based services and container-based services on Amazon cloud. The closest to our work is that of [6] in which the authors compared the performance of containers deployed on a privately owned bare-metal hosts against Amazon EC2 VMs. In our work, we focus on measuring the performance of container-based services against VM-based services when both services are deployed on the same Amazon cloud.

III. EXPERIMENTAL EVALUATION

This section describes our experimental methodology to evaluate the performance of container-based and VM-based services. We use a web service as an example to assess and compare the performance.

A. *AWS Environment Setup*

Amazon was selected to be the cloud platform to study and measure the performance of container-based services against VM-based services. Amazon Web Services (AWS) provides powerful features and fine-grained deployment capabilities as it is an Infrastructure as a Service and not a Platform as a Service (PaaS) in which more control over the resources can be achieved. The EC2 Container Service (ECS) provided by Amazon will be used to deploy Docker containers and the EC2 instances as the virtual machines.

1) EC2 Container Service (ECS): The ECS service first allows the developer to create a cluster of resources. These resources can be spread among different data centers. The cluster then can be filled with EC2 instances. In our case the cluster is created in Tokyo region. Each instance has an ECS Agent which is responsible to register the instance to the cluster. The ECS agent can talk via API using the Agent communication service offered by the container service. Within each Instance multiple services (tasks) with different ports can run on the same instance. Each of these tasks can have multiple related containers. In order to connect containers of the same task, a task definition has to be created and described using JSON Syntax. A Simple task definition representing a webpage will be relied on in this experiment [16]. The task will only require one container which will be using an httpd:2.4 image (which is the image for Apache Hypertext Transfer Protocol Server). The task basically displays a web page based on the Html command parameter. The web page will be accessed using the DNS Name or IP Address of the Container Instance the task is running on. In order to test the ECS performance against virtual machines, different schenarios need to be considered to support the performance analysis.

2) Evaluation Scenarios

We consider three evaluation scenarios with different number of web services deployed for each.

a) Scenario 1: In this scenario, we consider one web service running on one container instance. A similar EC2 based structure was set in which a single web service was created which uses port 80. Apache server had to be installed on the EC2 instance besides ECS where the containers only use httpd images. Both instances are of size t2.small (small instance size of 1 virtual CPUs and 2 GB of memory).

b) Scenario 2: The second scenario is related to the way ECS scales a given task. Tasks can simply be distributed among different container instances in which each container instance would usually host multiple tasks or containers. We run two tasks representing web services each using one container running on different ports. The first service will use port 80 and the other will use port 8080. The requests will be sent to both services at the same time. Two similar web services will be hosted on an EC2 instance to be compared using ports 80 and 8080. For EC2, httpd had to be re-configured in order to listen to another port.

c) Scenario 3: In real cases, a container instance would host multiple services running at the same time. We demonstrate a third scenaio to monitor the performance where more than two web services are running on the same container instance. We use three web services running on ports 80, port 8080 and port 81. Another httpd port was configures in order to add a web service using port 8181. The requests will be sent to the three services at the same time.

B. *Performance metrics*

In this section, we outline the set of key performance metrics to assess and compare the performance of containerized web services and VM-based web services. In order to effectively analyze the performance of the container-based and VM-based web services, we consider the following metrics are used in our study:

1) Server Throughput (requests/sec): It is important to measure server performance with respect of throughput at different request rates. Since we are dealing with web services, we focus on the rate at which we are able to retrieve the web documents with the GET method of HTTP requests sent.

2) Response Time (millisecond): It is essential to measure the time elapsed from sending the request until a response is received which is referred to as the response time. The response time is measured in milliseconds. Both response time and throughput are measured using the measurement tool JMeter described in Section III.C.

3) CPU Utilization (percent): Average CPU Utilization is usually measured to understand and amount of work and CPU usage of the virtual machine done when handling incoming number of requests. It is represented in terms of percentages out of one hundred. In our experiments, we measured the CPU utilization is by recording the CPU idle % and subtracting it from 100.

C. *Measurement Tool*

In order to test the different scenarios of deploying web services considering different scenarios, we need to measure the performance of the services by subjecting these services to a large number of HTTP requests, and subsequently, key performance metrics (for response time, throughput and CPU utilization) can be gathered and recorded at different incoming web workload. We use JMeter [17] as a generation and testing tool. JMeter comes with a graphical server performance dashboard. JMeter simulates a group of users sending requests to a target server then statistics will be provided indicating the performance of that specific target server.

JMeter will be configured to generate HTTP requests to a web service that will be running using Apache server deployed on an EC2 instance to mimic the VM-based service. For container-based service, the same EC2 image of the Apache service will be used by the ECS machines. JMeter was installed on a Windows EC2 machine locally (i.e. within AWS cloud Tokyo region and within the same availability zone) in order to minimize network overhead and variability. As shown in Fig. 3, the instance size chosen for JMeter is of t2.large type (large instance size of 2 virtual CPUs and 8 GB of memory) as we need to examine the performance under large request rate that can exceed10,000 requests/sec.

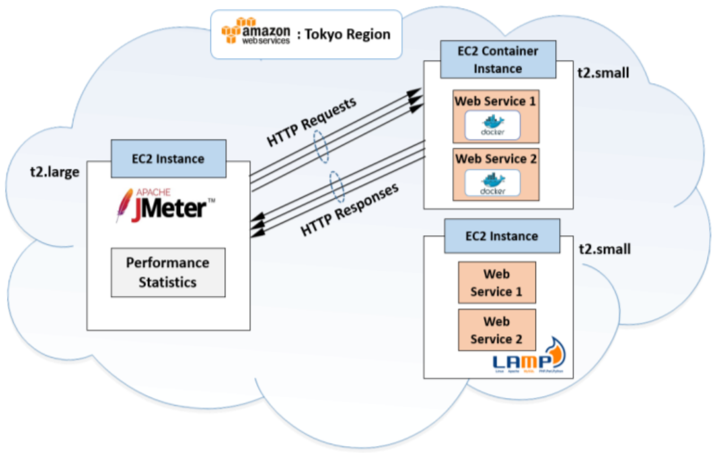


Fig. 3: Experimental setup in Amazon Cloud

We have chosen the number of users to vary from 1 to 20 virtual users (threads). Each thread will send as many requests possible in duration of 20 seconds duration. The number of requests sent by each user is indicated by how fast the target machine responses are. JMeter will send each request after the response of the previous request is received. By the end of each test JMeter will provide the performance measurements done within the 20 seconds. The throughput is returned as an average of the number of successful requests handled by the targeted server per second. JMeter measure the throughput for requests that were sent and received back successfully from the service. JMeter does not send any request unless the server accepts to open connections. JMeter records the response time as the average response times of all requests sent within the 20 seconds period. JMeter does not have the ability to measure the average CPU utilization of targeted servers. The CPU utilization was measured manually “using Linux *sar* utility” by directly connecting to the instance and measuring the average CPU Utilization in the period we are having JMeter sending the requests. Fig. 3 illustrates the basic components of our experimental setup.

All instances are launched in Tokyo region and within the same availability zone. JMeter will only send requests for one instance type at a time. The requests were sent from JMeter at midnight Tokyo time to prevent any network fluctuations and undesired overhead. Each performance metric computed for each number of users was repeated 5 times and averaged. This was necessary as the cloud environment is highly variable and fluctuating with respect to workload, hosted services, and utilized network and compute resources. This can be caused by network workload sizes variations, workload rates, number of VMs, and the type of applications running on the physical machine.

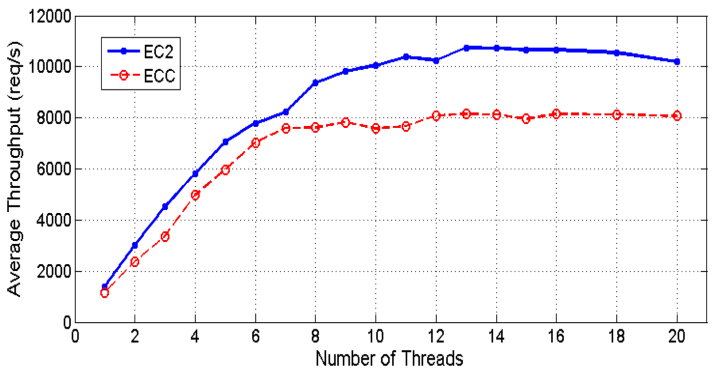
IV. EXPERIMENTAL RESULTS

In this section, we report, analyze, and compare the performance metrics for each scenario in terms of average throughput, CPU utilization and response time. Performance curves are plotted in relation to varying the number of users sending requests. The test was done during 20 seconds based on the setup described in Section III.

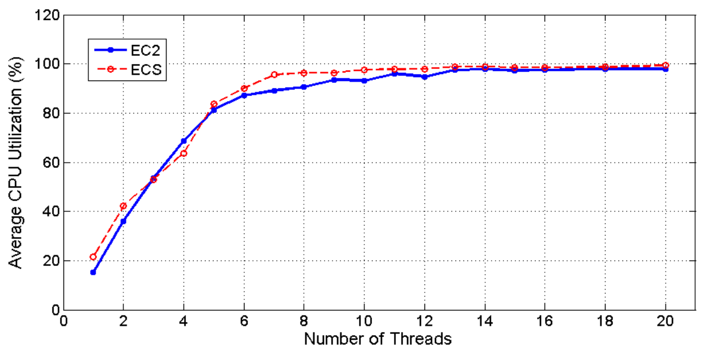
The performance measurements of the first scenario are shown in Fig. 4 for the three performance measures. Fig. 4(a) shows the average throughput curves for ECS and EC2, Fig 4(b) exhibits the corresponding CPU Utilization, and Fig. 4(c) shows the corresponding average response time. It is observed that the throughput increases with the increase of number of threads and then saturates at thread count of 10 for both ECS and EC2. Accordingly, for EC2, the throughput saturates at approximately 10,000 requests per second. Whereas for ECS, maximum throughput is obtained at approximately 8000 requests per seconds. The throughput count is affected by how fast JMeter is receiving responses and the amount of CPU utilization of the target instance. The CPU utilization in Fig. 4(b) is reaching 100% when handling requests of 10 users and above for both EC2 and ECS instance types. The saturation of the CPU utilization reflects the maximum throughout the machines are reaching to. The variance of the number of requests handled by both types of instances is indicated by the time the server takes to process incoming requests. Fig. 4(c) shows that EC2 instance has less response time than ECS which means that EC2 is faster in handling incoming requests reflecting on the amount of maximum throughput reached for each number of threads. ECS results are showing nearly 26.3% of extra time compared to EC2 at workload of 20 users and 44.4% when at workload of 10 users.

Fig. 5 exhibits the performance measures of throughput, CPU utilization and response time for both EC2 and ECS instance types resulting from Scenario 2. The throughput in this case is flattening off at 9000 req/s for EC2 and at approximately 6000 req/s for ECS which is less than the throughput observed in Fig. 4 of the first scenario. The throughput is highly reflected by the increase of the response time seen in Fig. 5(c). At workload of 10 users, ECS is resulting in 64% of extra time compared to EC2 results. The response time is reaching nearly 2 ms and 4.5 ms for EC2 and ECS respectively at workload of 20 users. Yet ECS has a significant overhead of nearly 2.5 ms which is about 125% of extra time compared to EC2. The CPU utilization is shown in Fig. 5(b) to indicate where the instance is reaching its saturation point for handling incoming workload.

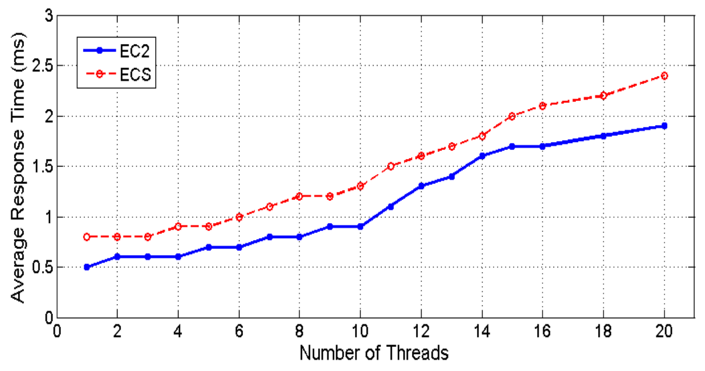
When comparing the performance measures of Scenario 1 with Scenario 2, it is clear that the performance is degraded as more services get deployed with ECS and EC2 instances. The degradation is noticeable in terms of the increase of the response time and lower throughput. For Scenario 3, another task running one container will be added to the ECS instance and similarly another web service to EC2. The corresponding performance measures are plotted in Fig. 6. The total average throughput of the three web services for both ECS and EC2 are shown in Fig. 6 (a). The throughput flattens off as the service is reaching a saturation point at approximately 7000 req/s for EC2 and 5500 req/s for ECS. Lower throughput is observed for EC2 by 1000 req/s and more response time of 1 ms compared to Fig 6. (c).



(a)

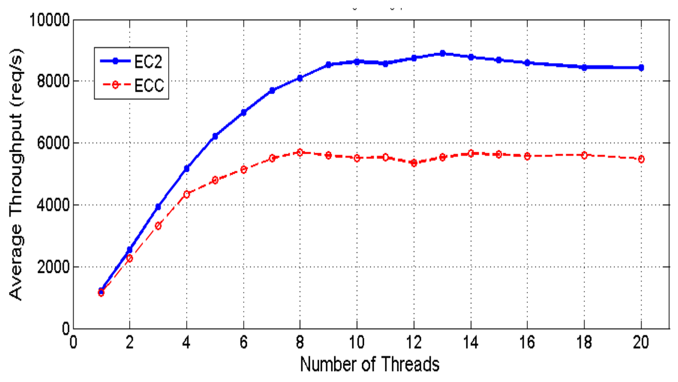


(b)

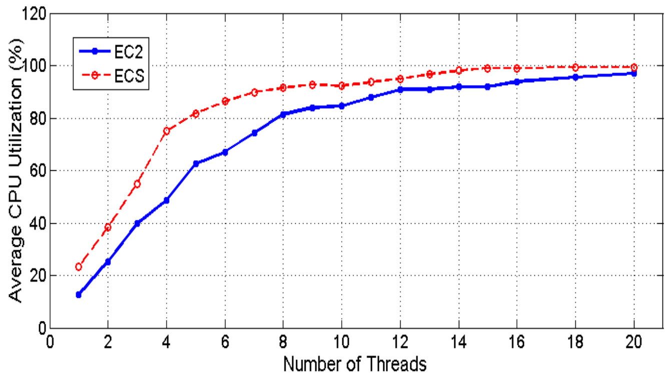


(c)

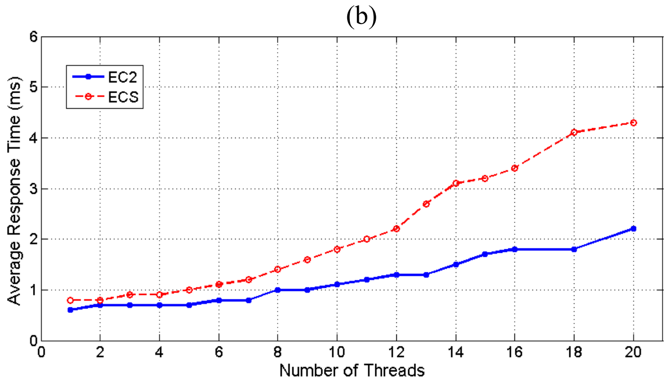
Fig. 4: Performance measures for deploying one web service (Scenario 1)



(a)



(b)



(c)

Fig. 5: Performance measures for deploying two web services (Scenario 2)

It is to be observed from Fig. 6(c) that the ECS performance in the third scenario is producing higher response time of 19% compared to EC2 at workload of 20 users and 38.4% at 10 users. Although CPU Utilization is not as crucial performance metric as throughput and response time, yet in all scenarios it shows that ECS is reaching its peak of 100% CPU utilization way ahead of EC2. ECS performance measures exhibited in Scenario 3 are very close to that of Scenario 2. Yet performance measures of EC2 are overwhelming those of ECS significantly in all scenarios.

The reason behind the degraded performance measures of the container-based services (ECS) compared with EC2 virtual machines is the deployment mechanism adopted by Amazon. The Docker containers in Amazon ECS are deployed on top of EC2 VMs instead of bare-metal as illustrated in Amazon's documentations [9]. Our experimental measures showed a significant performance overhead reflected by the existence of the hypervisor layer of the EC2 VMs in which containers are deployed on.

V. CONCLUSION

In this paper, we have conducted an experimental performance evaluation to compare and measure the performance of cloud-based services when deployed using: (1) Docker containers, and (2) Virtual Machines (VMs), on Amazon cloud environment. We considered different deployment scenarios in which web services were deployed. We measured the performance in terms of important metrics which include throughput, response time and CPU utilization while varying the incoming web request workload. Surprisingly, it was demonstrated that in general and under the three deployment scenarios, VM-based web services outperform container-based web services with respect to all performance metrics. Specifically, the performance difference has shown to be significant. For example, in terms of response time, VM-based web services can exhibit a performance gain of 125% over container-based web services. Therefore, we strongly recommend that for deploying applications with stringent performance requirements, the use of EC2 in AWS cloud should be recommended over ECS. We pointed out that the main reason behind the surprise performance degradation of container-based services when deployed on Amazon cloud was attributed to the fact that Amazon cloud runs containers on the top of EC2 VMs and not directly on bare-metal physical hosts.

In this paper, we focused on Amazon cloud platform, which by far remains the most popular cloud platform. As a future work, we plan to do similar measurements on other popular IaaS cloud platforms as that of GCE (Google Compute Engine) and Microsoft Azure. Furthermore, in our experimental study, we limited our performance evaluation for web services, but other services (e.g. database, analytics, streaming, etc.) can also be used for evaluation. The evaluation of these types of services are left for future work.

ACKNOWLEDGMENT

This research work is funded by the Information and Communication Technology Fund (ICTFund), UAE.

REFERENCES

[1] Namiot, D., & Sneps-Sneppe, M. (2014). On Micro-services, Architecture. International Journal of Open Information Technologies, 2(9), 24-27.

[2] S. Newman, Building Microservices. " O'Reilly Media, Inc.", 2015

[3] D. Jaramillo, D. V. Nguyen and R. Smart, "Leveraging microservices architecture by using Docker technology," *SoutheastCon 2016*, Norfolk, VA, 2016, pp. 1-5.

[4] R. Morabito, "Power Consumption of Virtualization Technologies: An Empirical Investigation," *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*, Limassol, 2015, pp. 522-527.

[5] J. Zhang, X. Lu and D. K. Panda, "Performance Characterization of Hypervisor-and Container-Based Virtualization for HPC on SR-IOV Enabled InfiniBand Clusters," *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, Chicago, IL, 2016, pp. 1777-1784.

[6] A. M. Joy, "Performance comparison between Linux containers and Virtual machines," *Computer Engineering and Applications (ICACEA), 2015 International Conference on Advances in*, Ghaziabad, 2015, pp. 342- 346.

[7] W. Felter, A. Ferreira, R. Rajamony and J. Rubio, "An updated performance comparison of virtual machines and Linux containers," *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium on*, Philadelphia, PA, 2015, pp. 171-172.

[8] T. Kämäräinen, Y. Shan, M. Siekkinen and A. Ylä-Jääski, "Virtual machines vs. containers in cloud gaming systems," *Network and Systems Support for Games (NetGames), 2015 International Workshop on*, Zagreb, 2015, pp. 1-6.

[9] "Amazon EC2 Container Service", aws.amazon.com/ecs/

[10] Hassan, M., Zhao, W., & Yang, J. (2010, July). Provisioning web services from resource constrained mobile devices. In Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on (pp. 490-497). IEEE.

[11] M. Vianden, H. Lichter, and A. Steffens, "Experience on a microservicebased reference architecture for measurement systems," in 2014 21st Asia-Pacific Software Engineering Conference, vol. 1. IEEE, 2014, pp. 183–190.

[12] Y. Sun, S. Nanda, and T. Jaeger, "Security-as-a-service for microservicesbased cloud applications," in 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom). IEEE, 2015, pp. 50–57.

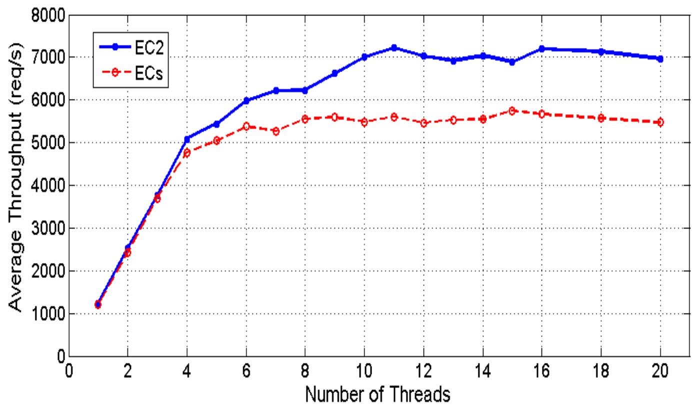
[13] B. Familiar, Microservices, IoT, and Azure. Springer, 2015.

[14] S. Nagpal and A. Shadab, "Literature review: Promises and challenges of devops."

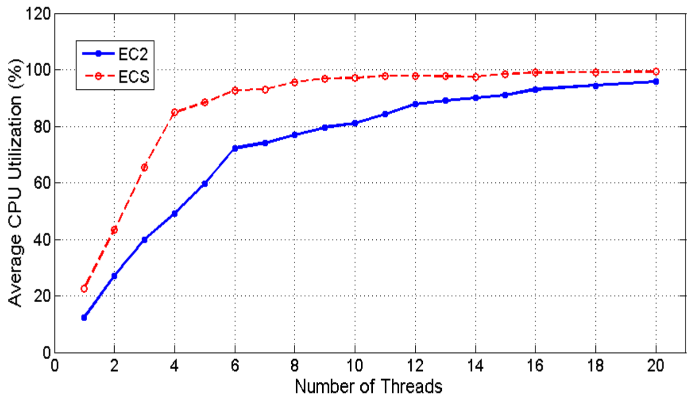
[15] H. Kang, M. Le, and S. Tao, "Container and microservice driven design for cloud infrastructure DevOps," in 2016 IEEE International Conference on Cloud Engineering (IC2E). IEEE, 2016, pp. 202–211.

[16] "Apache JMeter", jmeter.apache.org

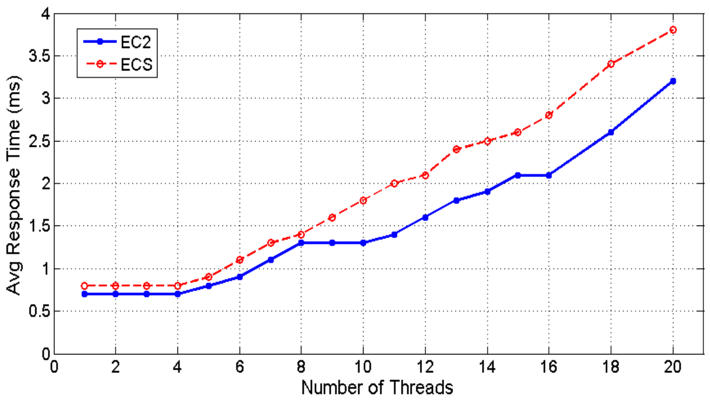
[17] Namiot, D., & Sneps-Sneppe, M. (2014). On Micro-services, Architecture. International Journal of Open Information Technologies, 2(9), 24-2



(a)



(b)



(c)

Fig. 6: Performance measures for deploying three web services (Scenario 3)