iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 8 **Issue:** V **Month of publication:** May 2020

DOI: http://doi.org/10.22214/ijraset.2020.5216

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 8 Issue V May 2020- Available at www.ijraset.com

A Study of Containerization as a Micro service Deployment Model

Nikhila Dharmaji[1], Deepika H C[2], Prof. Geetha V[3], Dr. G N Srinivasan[4]
[1, 2] *Student,* [3, 4]*Professor, Department of Information Science and Engineering, R V College of Engineering, Bengaluru*

Abstract: ***Containerization is the concept of packaging the entirety of the runtime of an application or service into an isolated, lightweight, platform-agnostic image which can then be deployed and run across various hardware platforms. It has emerged as a lightweight and cost-effective alternative to the existing technology of full machine virtualization. The shift towards microservice architectures has also resulted in significant changes in the way applications are deployed. Containers have, in the recent past, become a strong contender in the runtime options for microservices. Containerization of microservices has, in a short span of time evolved into a mature deployment model emerging with full force in many tech spaces. The introduction of tooling such as the Docker Platform, Amazon Elastic Container Services and Kubernetes have improved confidence in the use of container technology and have accelerated its adoption. Docker Platform, for instance, has a multitude of users littered all over the industry - Netflix, JpMorgan Chase, etc. We review the background of microservice containerization, motivation for its emergence, performance studies and its impacts on the industry. This paper seeks to review some of the efforts and literature surrounding the containerization of microservices.***

Keywords: Containerization, Microservices, Virtualization, Docker, PaaS

I. INTRODUCTION

The microservice model of developing software has been hugely popular and has been adopted widely. It has almost become the definitive way in which software development is done. The increasing popularity of this model has consequently resulted in bare metal hardware platforms becoming less and less popular as an option for running services. This is because bare metal options run the risk of conflicting application components, as the lack of isolation between services could mean one service affecting the availability of another. The solution to this problem lay in isolating services using Virtual Machines, which allowed multiple run times to exist and function on a single server without affecting each other. However, virtual machines come with their shortcomings when it comes to running microservices - they can be costly and prone to performance overheads and penalties. These shortcomings are due to the nature of VM's - they offer virtualization and isolation of operating systems as well as hardware, to emulate a fully functioning, self sufficient machine. The virtualization of hardware is associated with costs that are heavier than the benefits in the case of microservices. It is for this reason that the industry has been enthusiastic in the container approach. Containers, in contrast with VMs, do not virtualize or isolate the underlying hardware. Containers offer isolation at the operating system level only. This approach has various benefits - improved efficiency, reduced costs and finer-grained execution environments. It is due to these factors that containers are the preferred runtime option for a microservices architecture.

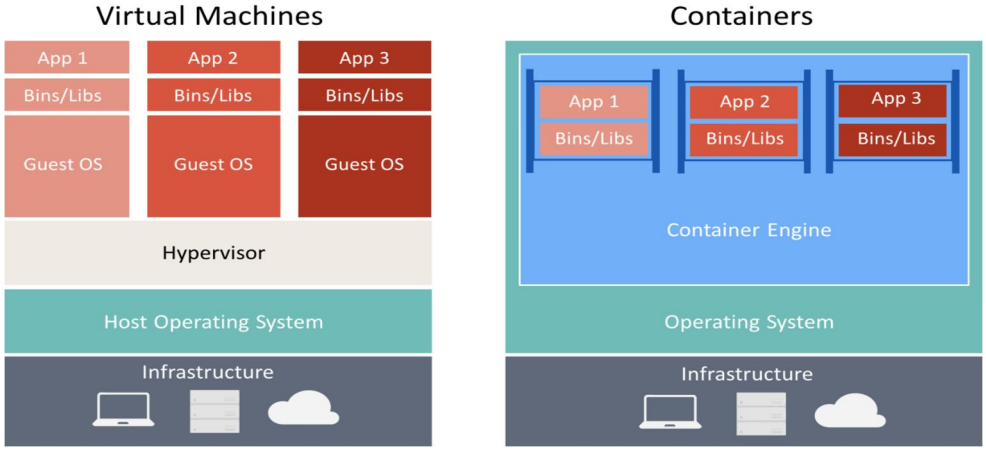


Fig 1.1 Virtual Machines vs Containers

II. BACKGROUND

Hypervisor-based virtualization has been around for several decades now, widely used in the domain of cloud computing. These systems enabled the running of multiple operating systems concurrently the same server. Examples of these systems include Hyper-V, XenServer, kernel virtual machines (KVM) [1-2].

In the early 2000s, FreeBSD introduced the concept of 'jail' into its operating systems. The concept was simple - a 'jailed' process had restrictions on its runtime. It could not affect the run time of other processes or make certain system calls. Jail was mostly used for the purpose of security - to isolate certain processes from the others so as to prevent them from inflicting any damage on other processes or the underlying system.

Solaris and Google also made significant efforts in this area through their contributions of Solaris Zones and cgroups. When cgroups were added to the Linux Kernel, it resulted in the technology we now know as Linux Containers (LXC).

Docker was born out of this search for a comprehensive and open-source container technology platform. Docker initially drew from LXC, but later replaced it with 'libcontainer', which also used Linux namespaces.

Ever since the introduction of Docker in 2013, there has been no looking back. Many tools to be used in conjunction with Docker have come to the fore since - Kubernetes, ECS, etc. These factors greatly contributed to the growth of the container ecosystem - and with it the change in the way microservices were deployed and run. Many actors in the industry saw a value add in moving from virtual machines to container technology, since the latter was a more reliable option for the microservice models.

III. MOTIVATION

As mentioned in [3], some companies with microservices architectures have adopted the twelve-factor app methodology to achieve the characteristics and goals of a true microservice architecture. The source also states that many of the goals of the twelve-factor app are embodied by container technology, for instance -

1) *Dev/Prod Parity:* Keeping development, staging, and production as similar as possible,
2) *Dependencies:* Dependencies are self-contained within the container and not shared with other services.
3) *Disposability:* Disposability is leveraged and satisfied by containers that are easily pulled from a repository and discarded when they stop running
4) *Concurrency:* Concurrency consists of tasks or pods (made of containers working together) that can be auto scaled in a memory- and CPU-efficient manner.

Along with meeting the criteria of the twelve factor app, the container methodology has shown itself to support the overarching software-development goals of modularity, agility and continuous integration and delivery, as mentioned in [4].

IV. PERFORMANCE ANALYSIS

A comprehensive evaluation of microservices architectures using containers is done in [5]. The evaluation is performed over five different kinds of experiments where the performance of containers in a microservice setup are compared with bare metal machines and virtual machines. The experiments relevant for the purposes of this paper are summarised as follows:

1) *CPU Performance:* In terms of CPU performance, the source reports that all systems evaluated - bare metal, VMs and containers perform equivalently. All systems report greater overhead when more instances of the service are run at once, due to increased context switching.
2) *Overhead of Virtual Container Creation:* A hosting entity is a container/VM running a dummy benchmark application. The time taken to create between 1 to 64 concurrent hosting entities is measured. It is noted that both types of containers (regular and nested) perform better than virtual machines.
3) *Network Performance:* Network throughput and latency were measured for many combinations of platforms - host-host, host-container and host-VM. The source reports that containers match bare metal under certain configurations. Virtual machines fall behind.

A comparative analysis between a VM-run microservice setup and a container-run microservice is done in [6]. Amazon ECS, a managed container orchestration service is used to launch Docker containers and EC2 is used to launch virtual machines for the experimental evaluation.

[7] also does a comparison between a VM and container setups for microservices, in which they report that two factors - deployment and updates were significantly faster in containers as compared to virtual machines.

These factors are especially relevant to microservice architectures because an important characteristic of microservice-based models is that they are usually developed using agile practices, leading to frequent revisions. In case of updates, the authors report that the

time taken to roll out updates in the case of VMs was just upwards of 3 times that of containers. They conclude that this is due to the fact that since containers share most of their runtime with the host OS, there is no need for these to be updated every time a revision is rolled out, in contrast to VMs which do not share any libraries with the host OS.

They also compare the sizes of the images of the VM and the container and report that VM images are at least 3 times as heavy as container images. This is due to the inherent nature of virtual machines - they contain an entire, fully functioning, self sufficient OS in their images. Containers are significantly smaller in size on account of them sharing their OS with the host machine.

V. IMPACTS

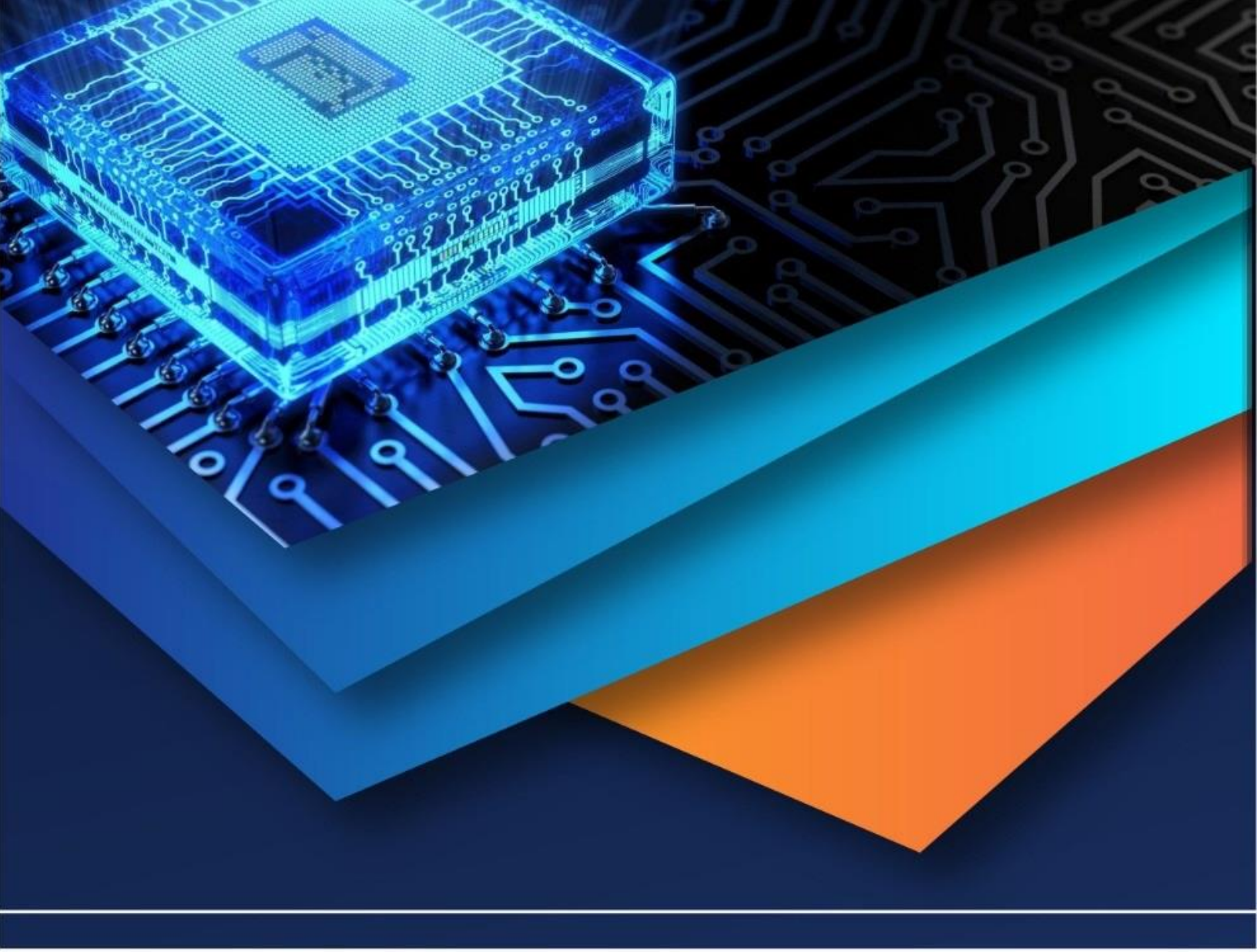
The term "container" has almost become the industry standard in the deployment of microservices. As more and more organizations move away from monolithic systems to microservice architectures, they are increasingly investing in container tooling like AWS ECS, Kubernetes, etc. This trend is also accelerated by the increased adoption of cloud-based deployment models. More and more organizations in tech spaces are gravitating towards the container promise of "build-ship-run". An increasing amount of literature dedicated to the move from monolithic services that run on VMs to containerized microservices also verifies these conclusions [8-10].

VI. CONCLUSION

An ever-expanding amount of literature suggests that containers are the best runtime option for microservices. While there is no one-size-fits-all model that guarantees that a certain model is the most suitable for a particular microservice, containers have proven themselves to be one of the stronger contenders in the race. Containers not only enable builders of services to move closer towards the principles embodied by the twelve-factor methodology, they also embody broader principles that have been established by software engineering as effective and superior - agility, modularity and continuous integration and delivery.

REFERENCES

[1] Sogand Shirinbab, Lars Lundberg, Emiliano Casalicchio, Performance evaluation of containers and virtual machines when running Cassandra workload concurrently (2020)
[2] Stephen Soltesz, Herbert Pötzl, Marc E. Fiuczynski, Andy Bavier, Larry Peterson, Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors, European conference on computer systems (2007)
[3] Somya garg, Satvik garg, Automated Cloud Infrastructure, Continuous Integration and Continuous Delivery using Docker with Robust Container Security (2019)
[4] Asif Khan, Pierre Steckmeyer, Nathan Peck, Running containerized microservices on AWS (2017)
[5] M. Amaral, J. Polo, D. Carrera, I. Mohomed, M. Unuvar and M. Steinder, Performance Evaluation of Microservices Architectures Using Containers, IEEE 14th International Symposium on Network Computing and Applications (2015)
[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, 20th Conference on Innovations in Clouds, Internet and Networks (ICIN) (2015)
[7] V. Singh and S. K. Peddoju, Container-based microservice architecture for cloud applications, International Conference on Computing, Communication and Automation (ICCCA) (2017)
[8] Joonseok Park, Daeho Kim, Keunhyuk Yeom, An Approach for Reconstructing Applications to Develop Container-Based Microservices (2020)
[9] S. Sarkar, G. Vashi and P. P. Abdulla, Towards Transforming an Industrial Automation System from Monolithic to Microservices, IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA) (2018)
[10] C. Pahl, A. Brogi, J. Soldani and P. Jamshidi, Cloud Container Technologies: A State-of-the-Art Review, IEEE Transactions on Cloud Computing (2019)



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY