International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 07 Issue: 07 | July 2020 www.irjet.net p-ISSN: 2395-0072

A Survey on Docker Container and its Use Cases

Bellishree P[1], Dr. Deepamala. N [2]

[1] *Department of CSE, RVCE, Bengaluru*
[2]*Associate Professor, Department of CSE, RVCE, Bengaluru*

Abstract - *Docker is an open platform used for development, shipping, and running applications. It facilitates in delivering software quickly a*

s it separates applications from the infrastructure. Docker methodologies promote quick shipping, testing, and deploying code anywhere, which reduces the delay between code development and its deployment in production. It has a wider scope of benefits for both Developers and System Administrators by allowing Developers to write code without having to worry about the system, that it will ultimately be running on. Also, it potentially reduces the number of systems and offers flexibility for the operations staff. This paper discusses the basics of Docker containers as well as related work that is being carried out in this area. Further, the paper includes key use cases along with the benefits and challenges involved in its usage.

Keywords - Docker; Container; Virtual Machine; Docker Image; Docker use case

I. INTRODUCTION

Years ago, before Docker containers came into existence, big companies like Walmart, Target, Chase Bank that relied on technology traditionally used servers and added many of them which lead to over-allocation, this was to handle the increasing number of requests from the users. The downside of over-allocating severs was that it was extremely expensive and failing to scale well the servers would crash and burn, and the business would be dead.

After a few years, the company named VMware came up with the concept called as Virtualization that allowed to run multiple operating systems on the same host which essentially meant running any application in isolation on the same server and same infrastructure, so it seemed like running an entirely separate computer on the same computer which was a game-changer for the many industries.

Although the concept of Virtualization came with privileges, it was very expensive. To begin with, there are multiple kernels for each guest operating system that will run on the infrastructure, besides, for each guest operating system added to the infrastructure, resources must be assigned. Further the reason behind virtualization being expensive is that even though there is no physical hardware there is virtual hardware that takes up resources, the guest operating system space, and the RAM allocation required for this operating system [6].

Advancing towards modern-day technology known as Containerization which is characterized as a type of operating system (OS) virtualization, employing which applications are run in isolated user spaces, using the same shared OS. Dockers being a containerization platform combines all the application and its dependencies in the form of Docker Container to ensure that the application works seamlessly in any environment. Docker provides far denser server consolidation than one can get with VMs, with the capability to share extra usable memory across the instances.[9] The advantages are that it facilitates Rapid Deployment; It is fast, lightweight as it doesn't boot a separate OS per VM making it fast to start/stop, it requires less disk space due to sharing of common layers across images, and due to the image layering incremental deployments of new app versions are smaller and therefore faster than VMs, it shares a kernel across containers and therefore uses less memory. It is portable as Docker files can be used to instantaneously configure any machine. Dockers coming with no hypervisor come as a greater advantage as it does not need a separate kernel and it still utilizes the same resources as the host OS and it exploits namespaces and control groups to tightly use these resources more efficiently.

II. LITERATURE SURVEY

The paper titled **"OpenStack and Docker: building a high-performance IaaS platform for interactive social media applications" [1]** describes about the Nova-Docker plugin which enables the fast and efficient provisioning of computing resources which can run as a Hypervisor that helps to manage the growth of application users. This is built using an OpenStack IaaS which enables to control data centres for cloud computing. OpenStack standard architecture contains three important roles: Nova, that

manages the computation, storage resources are managed by Cinder. The entire networking resources are managed by Neutron across multiple data centre. NUBOMEDIA is another approach which enables (PaaS) interactive social media through cloud. The major technologies adopted are Kurento Media Server (KMS) which provides interactive communications through WebRTC media server. OpenBaton which manages the lifecycle of media server capabilities using Docker containers. In order to host applications which consumes media server capabilities, OpenShift Origin is enabled. Developers and Administrators are interested more in Docker container than Kernel-based Virtual machine mainly for its Fast Boot time, Direct Access to containers, it can be run on any hardware that supports Linux based OS. Docker containers are lightweight, minimizing the bandwidth needed for deployment using required resources [8].

The paper titled **"Evaluation of Docker as Edge Computing Platform" [2]** describes about how to overcome problems such as High latency, network bottleneck and network congestion. We can achieve this from moving centralized to decentralized paradigm, Edge computing will be able to reduce application response time for better user experience. Edge computing is enabled with Docker, a platform of container-based technology that has more advantages over VM based Edge computing. This paper mainly evaluates the fundamental requirement for EC that are 1) Deployment and Termination which mainly describes the platform that provides an easy way to manage, install and configure services to deploy the low-end devices. 2) Resource and Service Management that allows users to use the services even when the resources are out of limit. 3) Fault Tolerance which relies on the High availability and reliability to the user. 4) Caching allows the user to experience better performance where Docker images can cache at the Edge. One such that enables Docker concept which was applied on Hadoop Streaming which reduces the setup time and configuration errors. Overall, there are areas of improvement yet, it provides elasticity and good performance.

The paper titled **"Model-Driven Management of Docker Containers" [3]** focuses on management of docker containers, mainly where users finds low-level system issues and it describes how modelling Docker containers helps to achieve sustainable deployment and management of Docker containers. Indeed, Docker system has more advantages than cloud-computing like Azure, Amazon; the Docker containers attains drawbacks in synchronizing between deployed and designed containers. This paper provides a model driven approach to manage not just to design the containers architecture but also represents the deployed containers in target systems. The motivation for this model-driven approach is that present docker containers lacks verification and resource management. The overview of the architecture described in this paper tells us about the three components Docker Model, Connector and Execution Environment. The abstraction of the containers is given by Docker Model. Connector provides the link between Docker model and Execution environment, which provides tools to operate Docker model efficiently by generating certain artifacts with regards to changes in Execution environment. This paper has presented a model-driven approach for verification and synchronization and its future work will investigate atomic changes performed in managed containers architecture.

The paper titled **"Docker container-based analytics at IoT edge" [4]** deals with internet connecting devices which use sensors and eventually generate high load of data. This becomes a challenging task to compute and process the data. In a workaround, this paper mainly describes about simplifying deployment of application at IoT end using Docker- lightweight virtualization mechanism. The research mainly depicts the use case for the video surveillance feed established in UK in order to analyse and detect the incidents that are about to occur using Deep learning. The implementation for the above use case, mainly involves components like Raspberry Pi 3 which is used as a gateway and obtains the video feed from the CCTV. Docker Swarm is used to orchestrate the frames obtained in the surveillance and multiple deep learning frameworks are used to analyse the feed and MQTT client is used to notify the cloud-based servers about the occurrence of thread in the frames. This implementation also sets a benchmark, in order to achieve efficient processing high resolution frames. This study shows Deep learning can be incorporated with Docker containers on single based computers which gives negligible overhead CPU processing when compared to Bare metal deployment.

The paper titled **"Workload-aware Resource Management for Energy Efficient Heterogeneous Docker Containers" [5]** propose Workload aware Energy Efficient Container (WEEC) brokering system to align with energy consumed by running container applications in multiple cloud servers. The proposed system shows how efficiently energy consumption can be reduced. Basically,

the system classifies input requests based on the containers utilization pattern of the resources into multiple server racks. The WEEC brokering system mainly comprises of 4 sub-components 1) Power Consumption Per Application (ppA) Table Manager which is responsible for quantifying the initial energy efficiency consumed by the heterogenous server in containers. 2) Exponential Weighted Moving Average (EWMA) predicts the workload for the future demands of input requests. 3) Dynamic Right Sizing (DRS) helps to manage the number of active servers. 4) Request Allocation Manager manages allocation of input request to each specified server in Docker containers. The study sets a benchmark by measuring power and performance of several heterogeneity Docker containers using benchmark applications with power measuring devices called Yocto-Watt.

III. KEY DOCKER USECASES [10]

A. ***Simplifying Configuration*** - This is a basic use case; the Docker configurations can be used several times in a diversity of environments. VMs hold a plus when it comes to running any platform with its configurations, the same is included by the Dockers, but without any Virtual Machine overhead, and allows to put into the code your configurations and environment and deploy the code. The infrastructure requirements are disassociated from the application environment. When it comes to the real-life use case, it enables in accelerating the project setup in the organizations by allowing them to dive into development directly by skipping the repetitive job of environment settings and configuration procedures.

B. ***Code Pipeline Management*** - The prior use case majorly influences the code pipeline management. By the time the code that is written in a developer environment passes through various stages(each of different platforms/environments) and approaches the production stage, a few minor differences could be observed; However, Dockers provide a consistent environment along all the stages from development to production, facilitating an easy development and deployment pipeline. Also, the stable nature of the Docker image and the ease with which it can be started can add up in achieving the aforementioned pipeline management.

C. ***Docker for Development Productivity*** - Docker facilitates achieving the 2 goals in a developer environment - First is to keep a developer close to production, this is made achievable by the Docker as it has no or low overhead when it comes to working remotely. The second is to have a development environment active for interactive use; Docker facilitates this by making the application code accessible to the container from the container's host OS by its shared volumes. This fetches benefits like - enabling the developer to modify the source code from the platform of his choice and also observe the same.

D. ***Multi-Tenancy*** - Docker helps to prevent major application rewrites and hence is used in multi-tenant applications. Multi-tenant applications' codebases are considerably more complex, rigid, and obstinate to manage. Redesigning an application consumes a surplus of time and money. Docker usage eases the creation of confined environments to running multiple instances of applications for each tenant and makes the process economical. The easy-to use API provided by the Docker's enables to spin up containers programmatically.

E. ***Debugging Capabilities*** - Docker provides various tools that work skilfully with containers concept. To name a few of its provisions, one being the ability to checkpoint containers and its versions, to differentiate two containers, hence readily fixing applications whenever necessary.

F. ***Better disaster recovery*** - A Docker image, alias snapshot, can be backed up at a certain point in time and retrieved to overcome any emergency issues. With Docker, a file can be replicated to new, different hardware. Docker image allows switching between the two different versions of the same software.

G. ***Improvement in the adoption of DevOps*** - DevOps community has developed bases from Docker, to standardize contained deployment. CI/CD determines the correlation linking the Docker and DevOps. Docker holds consistency among the testing environment and the production environment. Docker systematizes the configuration interface and simplifies the machine setup. Docker can be employed to introduce improvements in the DevOps of the company.

H. ***Rapid Deployment*** - Before Virtual Machines, the creation of new hardware resources took a days-together worth of task, which was later reduced to only a few minutes by Virtualization. On the other hand, Docker further reduces it to seconds by supporting just the creation of a container for the process, without the need of booting up an OS. It facilitates creating and destroying resources having to not despair about the cost of bringing it up again. The reduction in the cost of bringing up a new instance permits a more dynamic way of resource allocation.

IV. DOCKER USAGE

A. WHEN TO USE?

i. ***Discovering the latest technologies:*** Docker provides a disposable and isolated environment in order to begin with a new tool even without having to spend much time on installations and configurations. Several projects maintain docker images with the applications previously installed and configured.

ii. ***Basic use cases:*** Docker Hub is a registry service on the cloud which sanctions to download Docker images built by other communities, this service provides a good solution for pulling images for either Basic or Standard application.

iii. ***App Isolation:*** Managing each application component by keeping it in a separate container will discard dependencies when running various applications on a distinct server.

iv. ***Developer Teams:*** Docker provides a close match between the local development environment and production environment for developers working in a different setup.

B. WHEN NOT TO USE?

Docker cannot be the best solution always; few cases are as described below [10] -

i. ***Complicated Applications:*** Unlike basic applications, using pre-obtained docker file or pulling images from Docker Hub will be in-sufficient for complicated applications as the task of editing, building, and managing the requests and responses among multiple containers on various servers is very time-consuming.

ii. ***Performance-critical Applications:*** Docker holds an upper hand in comparison to VMs when it comes to the performance because the Containers share host kernel and emulate a full Operating system. In order to obtain the best possible performance out of the server, dockers can be avoided because the processes running on a native OS will be faster than a process within a container.

iii. ***Upgrade hassles:*** Docker as an emerging technology, is still under development due to which frequent updates are required in order to experience new features.

iv. ***Security is a critical factor:*** When it comes to more complicated applications, docker containerization's approach involves several vulnerabilities like Kernel level threats, inconsistent update, and patching of docker containers, un-verified docker images, unsecured communication, and unrestricted network traffic.

v. ***Multiple Operating System:*** As the docker containers share the operating system of host computer, it would need to use VM's to test or run the same applications on the different OS.

V. CASE STUDY

Integrating Containers into Workflows: A Case Study Using Makeflow, Work Queue, and Docker [11] - Distributed Systems such as Clusters, Clouds, and grids are integrated to execute large scientific applications that explain the topic of Workflow systems as a widely used abstraction. Although, workflow has a hindrance when it comes to multiple environments and differences in Operating systems, data, and certain resources. To solve this problem, the case study discusses how Lightweight containers come into existence by providing a well-defined operating system by making use of Make Flow and Work Queue. Specifically, this case discusses the integration of Docker with Make Flow and Work Queue using 4 strategies; Base Architecture – The basic architecture places a container in a directory and use as a placeholder, this ensures no overhead on the execution of the application. Wrapper-Script – This explains about the script which can be framed that helps to contain local docker, pull the desired image, and assign each task to a container. In this way, each task will be isolated from one another. Worker-in-container- This provides a secondary approach by placing an entire workflow system into a

container, this allows the prevention of start-up and shutdown costs of each task. Shared-Container- In this approach, only one container is created for each environment, this allows the concept of the task run on the same container instead of deleting the container. This avoids the overhead of multiple container creation. With the creation of container technology with the Workflow system significantly, lower cost can be accomplished without any impact on Workflow on distributed environments.

IBM Control Desk existing solution: A containerization case study with Docker [13] -
The IBM Control Desk case study has pertinence to the topic discussed in this paper. The IBM Control Desk software provides IT service management that eases users' support and infrastructures. This case study purely defines building the Control Desk entirely on Cloud using Docker containers and to provides SaaS as an edge platform. This mainly follows protocols in order to build and deploy. Firstly, the Control Desk is built on an administrative workstation mode and deployed on the DB2 node. A docker image is built for the control desk under the JMS server. A network of direct communication is created among the containers. For each Docker image, one contained is processed. An IBM HTTP server is configured to route traffic channels. By implementing this, IBM achieved no impacts at the application level and grew towards a cloud-native architecture.

Spotify: A Case Study with Containerization - Spotify is a digital music service that uses microservice architecture which delivers a whole host of features like music streaming, logging in through social media credentials globally. This application keeps 60 million subscribers happy with their ultimate music streaming capability. Spotify enables microservices through containerization of a bundle of requests so that a single request will provide all the information through its interface, which helps the user not to make any repeated requests to match specific information. Spotify has been using this technology since 2014 and has been a key pillar for strategical scalability.

VI. CONCLUSION

This paper gives a review of the use cases of Docker containers which are surveyed on a wide range of methods like- Docker as Edge Computing, building IaaS platform for interactive social media applications, and integrating Docker containers with IoT and resource management for energy efficiency. This paper also gives a summary of the evolution of the Docker container, extending to its use cases and area of usage.

VII. REFERENCES

[1] Alin Calinciuc, Cristian Constantin Spoiala, Corneliu Octavian Turcu, Constantin Filote, "OpenStack and Docker: building a high-performance IaaS platform for interactive social media applications", May 19-21, 2016.
[2] Bukhary Ikhwan Ismail, Ehsan Mostajeran Goortani, Mohd Bazli Ab Karim, Wong Ming Tat, Sharipah Setapa, Jing, Yuan Luke, Ong Hong Hoe, "Evaluation of Docker as Edge Computing Platform"., 2015 Advanced Computing Lab
[3] Fawaz Paraiso, St´ephanie Challita, Yahya Al-Dhuraibi, Philippe Merle, "Model-DrivenManagement of Docker Containers"., University of Lille & Inria Lille - Nord Europe 2016.
[4] Pankaj Mendki, "Docker container-based analytics at IoT edge"., Senior Principal Engineer, Member of R&D 2018.
[5] Dong-Ki Kang, Gyu-Beom Choi, Seong-Hwan Kim, II-Sun Hwang and Chan-Hyun Youn, "Workload-aware Resource Management for Energy Efficient Heterogeneous Docker Containers"., School of Electrical Engineering.
[6] Containers vs. VMs: What's the difference? http://searchservervirtualization.techtarget.com/answer/Containers-vs-VMs-Whats-the-difference.
[7] Understanding the architecture https://docs.docker.com/engine/understanding-docker/.
[8] M. Raho, A. Spyridakis, M. Paolino, D. Raho, "KVM, Xen and Docker: a performance analysis for ARM based NFV and Cloud computing," IEEE 3rd Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE), pp. 1–8, November 2015.
[9] R. R. Yadav, E. T. G. Sousa, and G. R. A. Callou, Docker Containers Versus Virtual Machine-Based Virtualization: Proceedings of IEMIS 2018.
[10] Deploy Docker Open Source, or Enterprise for High Performing Systems, https://www.flux7.com/tech/container-technology/docker/
[11] Chao Zheng and Douglas, Integrating Containers into Workflows: A Case Study Using Makeflow, Work Queue, and Docker.
[12] Control Desk existing solution: A containerization case study with Docker https://developer.ibm.com/technologies/ containers/articles/containerization-docker-case-study.