Virtualization and Containerized Development Workspaces

Pratyush Paliwal
IIIT Bhubaneswar,
Bhubaneswar, India
Email: b417026@iiit-bh.ac.in

***Abstract*— Virtualization is the process where we can divide and distribute a machine's resources among different environments, with an aim of increased efficiency with mostly shared hardware components. Virtual Machines (VMs) when talking about abstraction with isolating an entire machine powered by hypervisors, and Containers when specific applications with all its dependencies are isolated using OS-level virtualization, are two major virtualization techniques.

Virtual Machines which use hypervisor-based virtualization are made up of user space and boot up their own kernel. This is supported by hypervisor which allows multiple Operating systems to share a single host.
With increased priority for privacy, complete isolation and abstraction from the server Virtual Machines are used.

Containers which use Operating System level Virtualization and are isolated from one another and bundle their own software, libraries and configuration files. Containers use fewer resources than virtual machines as they run on a single OS kernel.

Containerized Dev Workspaces are widely used for deploying and running distributed applications or software without launching an entire Virtual machine.
Containerized Dev Workspaces are packages of all the required software, environment variables, and all other dependencies required to properly run the application on which developers are working, which results in eased deployment, increased efficiency and productivity.

Docker uses the OS-Level virtualization to deliver software in packages which are called containers. It is a set of Platform as a Service (PaaS) products.

In this paper we will discuss virtualization, creating dockerized images for containerized dev workspaces and its significance in DevOps. We highlight the performance of application containers. Containers are replacing VMs all over the world and are expected to play a very significant role in microservice applications.

Keywords— Containers, Docker, Hypervisor, Virtualization, Ansible, Container dev workspaces, DevOps.

I. Introduction

Virtualization is the root concept behind virtual machines and containers. Virtual machines are based on hypervisor based virtualization and containers use OS-level virtualization.

In Hypervisor based virtualization there is a layer between Virtual Machine and underlying host hardware. Hypervisor also known as Virtual Machine Monitor (VMM). Hypervisor is a software which allows to run multiple VMs per host hardware.
A virtual Machine is created by isolating the essential resources like disk, memory, networking and CPU. VMs creates their own kernel above the base OS and only one OS operates at a time. Hyper-V and VMware ESX server are hypervisors for windows and Linux kernel respectively.

Mainly hypervisors can be classified into two types, i.e., Type1 hypervisor and type2 hypervisor.
In type1, the hypervisor layer is placed on direct hardware, on CPU, disk, RAM, with no software or OS in between. VMs are then emulated above the hypervisor. That's why they are also termed as metal/ native hypervisors Examples of type1 hypervisors are KVM and VMware ESXi. (as shown in Fig. 1)

Type2 hypervisors layer is placed on the base OS and the VMs are then emulated over the hypervisors, these VMs reserve their physical resources and run their own Guest OS. They are also termed as hosted hypervisors. (as shown in Fig. 2).
VMs serve as Infrastructure as a Service (IaaS).

Container based virtualization i.e. OS-level virtualization supports encapsulation of operating systems and their dependencies into packages which are managed by the base OS kernel, these packages are called containers. The OS-level Virtualization mechanism is to run multiple isolated systems (containers) on a single OS kernel.

Mainly we can classify the containers into system containers and application containers. System containers encapsulate a complete system and support each type of system commands. LXC, Linux containers are system containers.
Application containers packages complete applications, docker-hub has one of the largest collections of application

containers, with containers for various software applications.
These serve as Platform as a Service (PaaS).

Fig. 1. (Type1 Hypervisor)

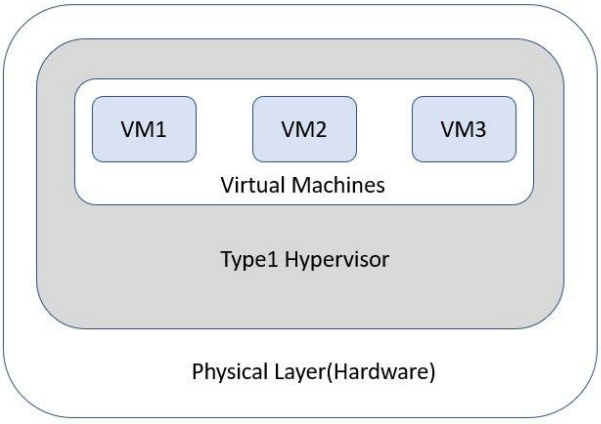
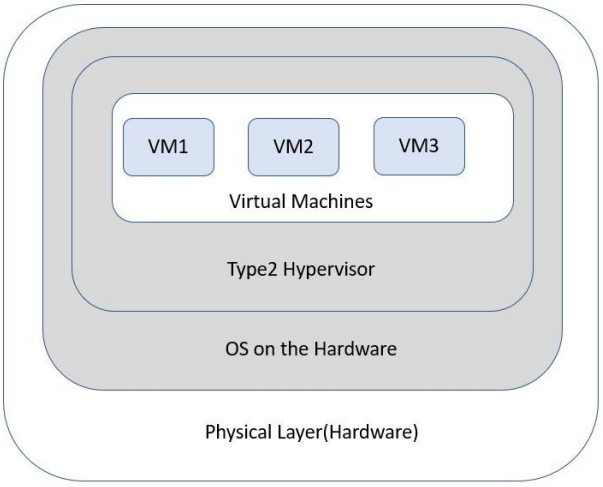


Fig. 2. (Type2 Hypervisor)

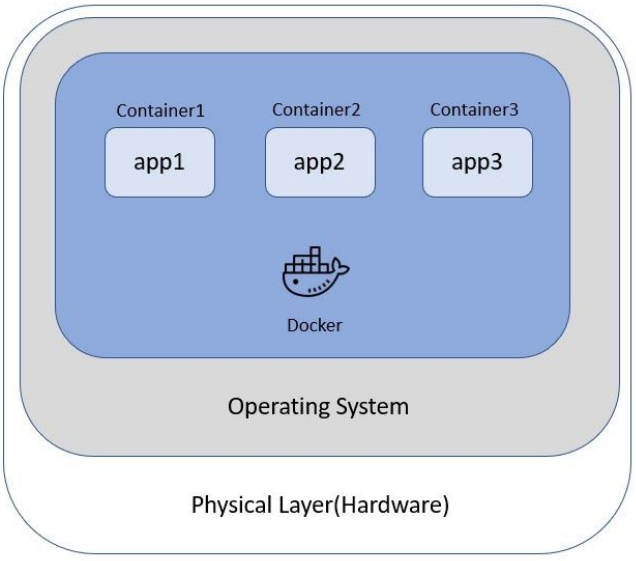


Containerized Dev Workspaces are containers that packages the software and their dependencies required for their proper functioning, all these software are installed with their specific versions required for particular development workspace. These packages when built provide shareable docker containers which can be shared among all the developers of the team, which enables them to work in the same development workspace. It eliminates the errors introduced due to version mismatch and other problems due to manual environment settings.

II. Docker

Docker is a set of Platform as a Service (PaaS), container technology which allows to package software applications with their dependencies and also allows to distribute it among others. Docker is the flag bearer of container movement. It is open source written in Go language. Docker API had fifteen revisions in the past one and a half years and developers and the open source community are still working on it. Docker utilizes LXC containers, fitting these container environments is difficult as they're terribly low, and here docker offers a high level tool with several functionings making it easy.
Docker containers share the underline kernel, Docker can run any particular OS distributions containers on top of it as long as they all share the same kernel. Shown in Fig. 3.
Windows based containers can not run on a docker host based on a Linux kernel. For running that we require a docker host on windows server.

Fig. 3. (Docker Container Architecture)



Docker Software Components:
Software: The *Docker daemon* manages docker containers and handles container objects. It listens to the requests from Docker engine API.
Docker CLI which is a docker client program gives a command line interface which allows users to interact with Docker daemons.

Objects: assembling applications in docker is main functions, these objects are *images, containers and services.*
Docker images are read-only templates, which are used as an original document and photocopy is used as containers of that particular image.

Docker Containers are packages created by docker images that can be managed by API or CLI.
Docker Services provides a platform for containers to scale across various docker daemons.

Registries: collection of docker images is Docker registry. Clients can use these registries for downloading or uploading the images which have been built. DockerHub is the default Docker registry. Docker cloud is another Docker registry.

Docker helps in standardizing the environment by ensuring consistency across multiple development and release cycles. It allows the same application software container from build, to test, to production, throughout the pipeline.
Docker highly supports Digital transformation. Microservices based containerized application development is one of the key factors of digital transformation. This helps in reducing the complexity of IT systems in the organization by providing the ability to build and deliver quickly.

Tech organizations like google, AWS, Intel, Tesla use container technology and docker. Google created Kubernetes which is a container orchestration tool, google provides numerous container services. This year Intel launched a new version of "clear containers" which is a tool to improve scalability of applications. This tool runs on Kubernetes.

With the increased need for automation in each field, docker encapsulates the automated orchestration of workloads, which does not leave docker as a standalone technology.
The global application container market which includes container monitoring, Data management, networking, security and orchestration platforms, is growing with an Compound Annual Growth Rate (CAGR) of 32.9%.
This is due to the fact that various container orchestration services and container security services are deployed in enterprises globally.

III. Virtual Machines and Containers

With basic understanding of Virtual Machines, containers and their architectures, various research and development are analyzed in their comparison, in order to determine which one is better to use at a specified application area.
These comparisons are done on Ubuntu 16.04. Devstack which is a Openstack dev environment, is used to create VM. Docker is used for container hosting. For comparisons based on HTTP requests, apache web server is used.

We focus on getting results which will suggest more preferable virtualization techniques, docker or VMs, on the basis of performance, scalability and user space isolation for the purpose of hosting applications.

RAM usage *(idle state)* : Virtual machine occupies way more RAM (approx 500 MBs), as compared to containers (approx 5 MBs).
Disk Usage *(post creation)* : Virtual machine uses way more disk space (approx 950 MBs), as compared to containers (only 0.5 MBs).
Performance : Both VMs and Containers showed equivalent response for executed number of operations per unit time elapsed, with container executing more number of operations in long run.
HTTP request handling : VMs handled slightly more number of HTTP requests as compared to containers on the number of requests served as a time measuring basis.
But VMs simultaneously uses more CPU while handling the requests as compared to containers. In case of RAM usage while handling HTTP requests, containers performance is way better than VMs, RAM usage goes up to 900 MBs while handling requests in case of VMs, and only upto 90 MBs RAM usage in case of containers, while handling the same number of requests.
File Transfer speed : Containers perform way better than containers in case of speed of file transfer through the network. Speed of file transfer in VMs starts with 150 MB/s and constantly drops to around 100 MB/s with increasing time in seconds (0-10s). While speed in containers starts at around 170 MB/s and drops to around 150 MB/s in the same time frame (0-10s) as used for VMs.
Scalability : when considering heavy loads, i.e. request handling at increasing number of connections VMs perform slightly better than containers, but at a cost of significant high CPU and RAM usage. Also after some timespan container performance tends to be equivalent to that of virtual machines with a very less comparative CPU and RAM usage.

Virtual Machines are seen as a single process by the underlying OS even there are various active processes executing simultaneously in VMs. This situation of not being aware of each other's processes creates almost complete abstraction between host OS and Virtual Machine. This supports the results we got for the above observations.

This justifies the use of a large number of containers on the host physical server as compared to using the VMs for the processes. If privacy is considered a deciding factor, then VM gets a higher chance of usage due to above mentioned

complete abstraction and isolation of the process details between the host and VM.
Containers use fewer resources than VM, and are faster in network supported file transfer, on the other hand for handling a higher number of HTTP requests in small timeframe VMs are better.

IV. Containerized Dev Workspaces in DevOps

DevOps is a word made by Development and Operations, with the help of containerized approach toward software development, communication gap and lack of information between teams in an organization is eliminated, DevOps combines development and operations activities with a target of reducing time for development life cycle, supporting agile development with continuous delivery at the same time not compromising on software quality.

DevOps reduces the uncertainty due to version mismatch of software, different PC settings, slightly different environment settings, wrong installation of dependencies of a software application, containerized way of development ensures that the application package in container runs same as it is developed in developer's system, on any system, without any installation, or environment setting. Docker containers run the same on each machine irrespective of which server or which machine they are running on.

Containerized Development workspaces reduce the time taken to set up development environments and dependency on local desktop resources for development.

Ansible can be used for creation of dev workspaces using docker, ansible is mostly used for automation of installation of different software packages required for given development workspace.
Ansible playbook is a YAML file, which is specified in the docker file, it tells the processor what to do, with step wise instructions for installations. These installations are automated and eliminate the need for manual package installation in container workspace.
Dockerfile contains other docker image specifications with ansible playbook, docker engine uses Dockerfile to build required docker images. With the help of docker image we can create multiple instances called docker containers.

Containerized dev workspaces in organization results in cost reduction, digital transformation, standardization of environment and increased productivity and easy management.

V. Conclusion

This paper talks about virtualization, two main approaches, hypervisor based and OS-level virtualization, Virtual Machines, containers and their comparison research based on various parameters, Docker and its growing use, containerized dev workspaces in organizations and significant role in DevOps.
Docker is still growing rapidly, Google's Kubernetes is the latest docker orchestration technology, it is open source and has a large contribution in container administration and management.

Performance comparison on the basis of different aspects between VM and Containers is shown in this paper. Container leaves behind virtual machines in performance and scalability, which makes containers more appropriate for application deployment as compared to using entire virtual machines for application deployment. Certainly, there are also some cases when virtual machines are preferred instead of containers; i.e., cases where privacy is of very high priority and the data application software deals with is confidential and critical to business.

Nested virtualization is another field of virtualization which is, virtualization of a system inside a virtual machine, that is, virtual machine on a Virtual machine.
Running a container inside a virtual machine will provide the performance of the containers with added benefit of privacy and security within Virtual machines. Large technological firms like Google and Amazon use containers inside VM, nested virtualization technology. There are still vast areas to explore in this field.

References

[1] Docker. URL https://www.docker.io/

[2] Virtual Machines vs Containers, Vestman, S., 2020. *Cloud Application Platform - Virtualization Vs Containerization : A Comparison Between Application Containers And Virtual Machines*. [online] DIVA. Available at: <https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1112069&dswid=-8369> [Accessed 9 December 2020].

[3] N. G. Bachiega, P. S. L. Souza, S. M. Bruschi and S. d. R. S. de Souza, "Container-Based Performance Evaluation: A Survey and Challenges," 2018 IEEE International Conference on Cloud Engineering (IC2E), Orlando, FL, 2018, pp. 398-403, doi: 10.1109/IC2E.2018.00075.

[4] S. Singh and N. Singh, "Containers & Docker: Emerging roles & future of Cloud technology," 2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT), Bangalore, 2016, pp. 804-807, doi: 10.1109/ICATCCT.2016.7912109.

[5] 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, doi: 10.1109/CCAA.2016.7813925.