Systematic Approach to Docker

University of New Haven
Tagliatela College of Engineering

A Research Project Paper
Submitted by
Group – 9
CSCI – 6643 : Operating Systems

Professor. Vyshampa Maringati

Project Team
Yashwanth Yarram
Kamal Siddharth Teki
Srihari Chitikela
Bhaskar Yannam
R Ram Kumar

December 2021

Abstract

Docker provides services and some key features which are essential for building and developing applications in a portable environment with the help of Docker Engine.

In this paper, we have discussed on how docker containers create vast economies of large scales unlike remaining virtual machines and allows developers to build, implement, run, update, and stop containers from using simple commands through a single API. Virtual machines, on the other hand, enclose a whole operating system with variable executable codes on an abstracted layer of physical hardware resources. This paper also focusses on various features provided by the docker engine that are utilized for several application services which will give us an idea on how a container technology like docker enables us to deploy the products in various physical servers as well as data servers and in various cloud platforms replacing virtual machines in an efficient manner.

Keywords:

Docker, Virtualization, Virtual Machine, Container, Implementation.

Introduction

It is an open-source software platform that allows the user to design and implement virtualized application containers on a single OS using a set of tools. Docker creates, manages and operates containers. A container is a standard software unit that encapsulates code and all its dependencies, allowing the program to be quickly and reliably transported from one computing environment to another. A Docker container image is a tiny, stand-alone software package that includes the code, runtime, system tools, system libraries, and settings needed to run a program. Container images become containers at runtime, and images become containers when Docker containers run on Docker Engine. Containerized software, which is accessible for both Linux and Windows-based applications, runs consistently regardless of the infrastructure. To run several containers on the same OS, Docker leverages resource isolation in the OS kernel.

It was originally made to run on the Linux OS, but it has now been expanded to incorporate support for the non-Linux based operating systems such as Windows and Mac OS. The Docker Community Edition is open - source software, which is a collection of components and tools that aid in the creation, verification, and management of containers. The Docker Engine oversees the tasks and workflows that go into creating container-based applications. This engine creates a server-side process that generally hosts pictures, containers, networks, and the storage volumes, as well as the client-side command-line interface (CLI) for users to relate with the Docker API and docker files are the containers built by Docker to assemble the images.

Technical Specifications

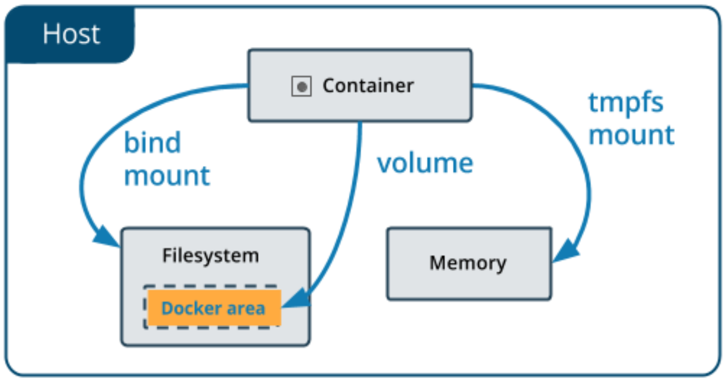
Primary Tool used : Docker Engine
Web Browser used : Google Chrome
Hypervisor used : Virtualbox
Operating system used : MacOS

Course of Action

It is essential to know the key features such as Docker Storage, Dockerfile, Docker Swarm and Docker Compose as it would make it easy for further understanding of concepts that are involved in the implementation of the containers with the help of Docker.

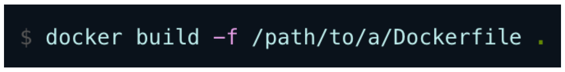
Docker Storage

Containerization is storing the data of the programs that operate within the container. When a container has been running for a long time and another container requires the data from this halted container, data persistence might be lost. If a container is removed or crashes due to internal issues, data might be lost altogether. Docker gives us a way to keep the data alive even if the containers are stopped or removed. Docker creates and manages volumes, which are kept as part of the host file system and should not be modified by non-docker processes. One can be able to build a volume directly using the docker volume create command, or docker will create one for the user automatically when the user creates a container.



Docker File

Dockerfile is best defined as infrastructure as code, in which one may script it to construct the images required for building production environments, and these images can then be used to spin up numerous containers for repeatable setups. It is merely a collection of commands that will be performed in the sequence they were scripted, resulting in the 42 image that contains these instructions. Dockerfile is generally a set of command line interface instructions that it will execute to put all the images together. These images could be stored on Docker Hub making it simple to share them with other individuals within the organization. If we wish to erase an image from the localhost, We have first stop and delete all of the images related containers before we can erase the file.



```
$ docker build -f /path/to/a/Dockerfile .
```

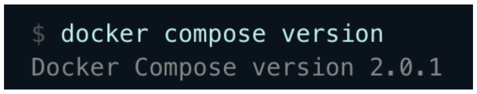
Docker Swarm

It is a container orchestration tool that allows the user to manage multiple containers on various host machines. The container orchestration system takes care of these issues by distributing apps over several clusters of nodes or virtual machines based on the traffic they generate inside the container. This system should also be able to do health checks on the nodes where the containers are executing, as well as route traffic away from nodes where the health checks have failed. Apart from that, it should be able to load balance traffic and perform rolling updates on the cluster's services.

Docker swarm employs the Raft consensus mechanism to record the configuration of all nodes, making it easier for the management server to schedule activities on master node. To keep the swarm synchronized, once a job or service is scheduled on the nodes, the task state and swarm state are updated on all the manager nodes. When a primary manager goes down, the newly chosen manager may quickly take over the swarm's responsibilities because it has already obtained the swarm's status. It is usually a best practice to have at least three managers for higher fault tolerance but increasing the number of managers may reduce cluster performance since data transfer across the multiple number of manager nodes would take more time.

Docker Compose

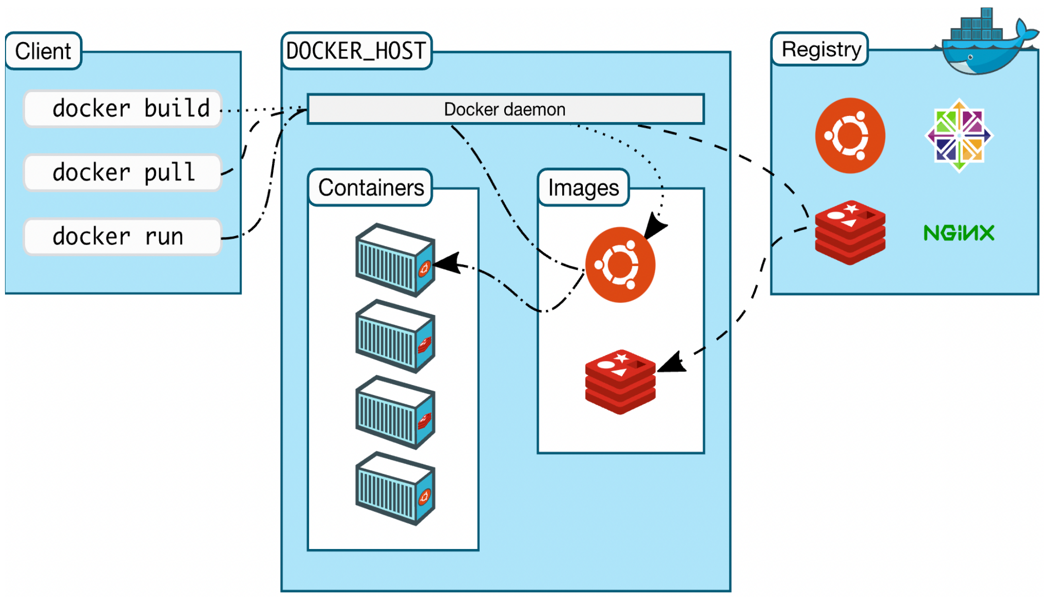
When your application requires several isolated containers, Docker compose comes in useful for constructing, operating, and linking those containers, and the full setup may be completed on a single host. Using docker compose, one may quickly set up a development environment on their local desktop, which can then be shared on source control repositories, allowing other crew members to contribute and benefit from the rapid building of an environment without having to install any tools locally.



```
$ docker compose version
Docker Compose version 2.0.1
```

Docker Architecture

Docker is built on a client-server model. The Docker client communicates with the Docker daemon, which handles the construction, execution, and distribution of the Docker containers. One can execute the Docker client and daemon on the same machine or link a Docker client to a Docker daemon that is located elsewhere. Docker Compose, which we previously described, is another Docker client that allows the users to deal with applications made up of several containers.



Docker Daemon

The Docker daemon handles Docker objects such as images, containers, networks, and volumes by listening to the Rest API. To manage Docker services, a daemon can communicate with other daemons.

Docker Client

Many users can interact with Docker primarily through the Docker client. When we use commands like docker run, the client delivers them to daemon, then executes them. The docker command makes use of the Docker API, which allows a client to connect with multiple daemons.

Docker Registry

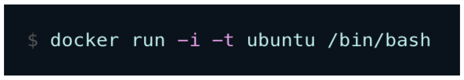
The images are stored in the Docker registry, and Docker Hub is a public registry that anybody may use to utilize the images required for the application. So basically, Docker is set up to look for images on Docker Hub by default. But one can also run a own private registry instead of Docker Hub.

For example, docker pull is a command used to pull the required images from docker hub or a private registry and docker push is a command used to push or insert the images to the registry.

Containers

The Docker API may be used to build, start, stop, move, or remove a container, which is a virtual instance of an image. A container can be connected to one or more networks, have storage attached to it, or even be used to produce a new image based on its existing state. The image of a container, as well as any configuration parameters the user provides during the build defines it. Any modifications to a container's state which aren't saved in file systems will be erased when the container is withdrawn.

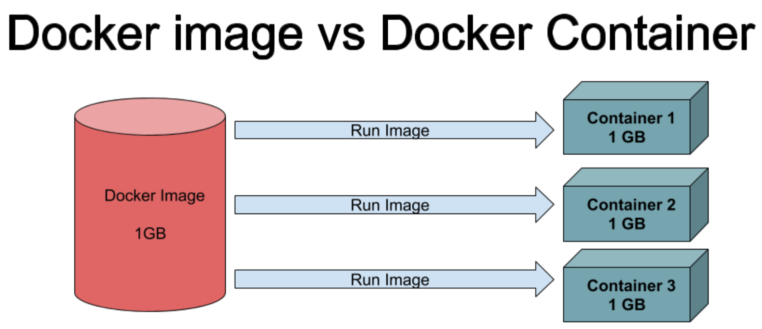
For example, docker run is a command which is used to run a specific container and attaches it to the local command line-session.



```
$ docker run -i -t ubuntu /bin/bash
```

Images

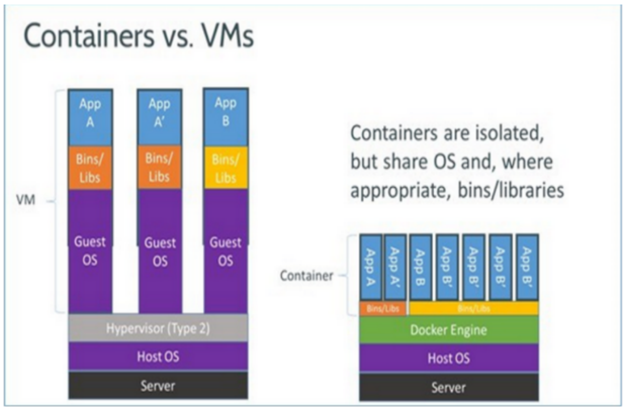
A Docker image is a read-only file with certain specifications for building a container. An image is created from a source image with some specific modifications. For example, one can be able to create an image based on the Ubuntu image but installing an Apache server is essential to make the application run.



Docker and Containers

Containers and virtual machines distinguish in that a way that hypervisor abstracts the entire device, whereas containers merely abstract the kernel of the operating system. A hypervisor is a class of software that is the heart of a virtual machine. In the cloud, a hypervisor enables to isolate a single virtual machine which is much more than just a separator for the virtual machines. Hypervisor will always mirror the hardware components of a standard operating system in addition to establishing an impenetrable virtual barrier among various operating systems.

In Hypervisor, virtualized copies of hardware resources like as CPU, I/O, memory, and others will be available. The distinction between virtual machines and containers is frequently blurred. Container is an another component of the virtual environment, but unlike hypervisor, they are not based on it. It is essentially a virtual OS without the virtual hardware components of a full VM. So, containers can run inside virtual machines and reside under a hypervisor.



Challenges faced & Measures taken to overcome

- One of the major challenges that we have faced during our project is that Docker is not supported for most of the versions of Windows OS as the Native Docker support has been officially out for Windows OS. There is nothing we can do but to use an alternative operating system such as Mac OS or Ubuntu and wait for the production ready version of native Docker for Windows OS.

- Another challenge was to monitor the Docker. There are two ways to monitor, one is pull monitoring where one can have access to the monitoring server and get into the docker logs to check the stability. The other is push monitoring, where a monitoring agent pushes its status to the monitoring server. So, this has become a challenge when we are dealing with multiple hosts. We can overcome this challenge by using container monitoring tools such as Sematext Agent as they will be aware of the container deployment state.

Advantages

- Applications created within Docker containers are incredibly portable, and as a result, they may be basically moved as an individual part whilst holding on to the same level of performance.
- Containers can be quickly transported from the cloud location to the local host and vice versa. Even the scale can be customized by the user to meet his needs.
- Docker always makes better use of resources as it doesn't utilize a hypervisor and for this reason the user will be able to run a greater number of containers on a single host compared to virtual machines.

Disadvantages

- Docker does not provide complete virtualization since it relies mostly on the Linux kernel, and that is given by the local host.
- It was created to make it easier to deploy server applications that don't need a graphical user interface. As a result, applications with a graphical user interface perform poorly.
- Docker still cannot be operated on older machines. 64-bit local machines are supported.

Conclusion

In this paper we conclude that Docker containers have simplified application development in all settings, including development, testing, and production. It is evident that Docker can be utilized to quickly recreate environments on our local desktops or remote servers to test and deploy our application without the need to install additional software that would drain the system's resources. This paper also indicates that application downtime may be reduced by rapidly increasing the number of containers where the application is deployed.

Also, we have primarily focused on containers and its unique characteristics which are used for a rapid delivery and containerization of enterprise applications. In the future, we would also intend to focus on various third-party cluster tools like Apache Mesos and Kubernetes that are much more complex and advanced than the Docker Swarm which we have discussed previously.

References

[1] https://www.docker.com

[2] https://www.ibm.com/cloud/learn/docker

[3] https://docs.docker.com/get-started/overview/

[4] https://sematext.com/blog/docker-container-management/

[5] https://www.infoq.com/news/2015/12/dockercon-docker-monitoring/

[6] Turnbull, J. (2014). The Docker Book: Containerization is the new virtualization https://www.oreilly.com/library/view/the-docker-book/9780988820203/

[7] Scheepers, M. J. (2014). Virtualization and containerization of application infrastructure: A comparison. https://thijs.ai/papers/scheepers-virtualization-containerization.pdf

[8] IJCSNS International Journal of Computer Science and Network Security, VOL.17 No.3, March 2017. http://paper.ijcsns.org/07_book/201703/20170327.pdf