DevOps with Docker-An Overview

Mr.G.Kanagaraj[1], Ms.T.Primya[2], M.Kirthika[3]
[1,] Kumaraguru College of Technology Coimbatore
[2, 3] Dr.N.G.P.Institute of Technology,Coimbatore

I. INTRODUCTION

Docker is the world's leading software container platform. Developers use Docker to eliminate "works on my machine" problems when collaborating on code with co-workers. Operators use Docker to run and manage apps side-by-side in isolated containers to get better compute density. Enterprises use Docker to build agile software delivery pipelines to ship new features faster, more securely and with confidence for both Linux and Windows Server apps.

Docker is a tool that can package an application and its dependencies in a virtual container that can run on any Linux server. This helps enable flexibility and portability on where the application can run, whether on premises, public cloud, private cloud, bare metal, etc.

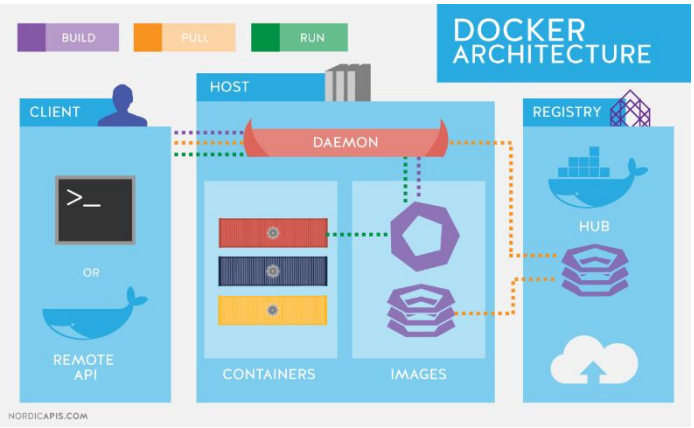

Figure 1. Docker Architecture

Docker provides an additional layer of abstraction and automation of virtualization at the operating system level on Linux. Docker provides an additional layer of abstraction and automation of operating-system-level virtualization on Windows and Linux. Docker uses the resource isolation features of the Linux kernel such as cgroups and kernel namespaces, and a union-capable file system such as OverlayFS and others to allow independent 'containers' to run within a single Linux instance.

This helps in avoiding the overhead of starting and maintaining virtual machines, which boosts its performance and also reduces the size of the application. The Linux kernel's support for namespaces mostly isolates an application's view of the operating environment, including process trees, network, user IDs and mounted file systems, while the kernel's cgroups provide resource limiting, including the CPU, memory, block I/O and network.

Docker containers are so lightweight, a single server or virtual machine can run several containers simultaneously. By using containers, resources can be isolated, services restricted, and processes provisioned to have an almost completely private view of the operating system with their own process ID space, file system structure, and network interfaces. Multiple containers share the same kernel, but each container can be constrained to only use a defined amount of resources such as CPU, memory and I/O.

Using Docker to create and manage containers may simplify the creation of highly distributed systems by allowing multiple applications, worker tasks and other processes to run autonomously on a single physical machine or across multiple virtual machines. This allows the deployment of nodes to be performed as the resources become available or when more nodes are needed, allowing a platform as a service (PaaS)-style of deployment and scaling for systems like Apache Cassandra, MongoDB or Riak. Docker also helps in simplifying the creation and operation of task or workload queues and other distributed systems

II. CONTAINER

Using containers, everything required to make a piece of software run is packaged into isolated containers. Unlike VMs, containers do not bundle a full operating system - only libraries and settings required to make the software work are needed. This makes for efficient, lightweight, self-contained systems that guarantees software will always run the same, regardless of where it's deployed.

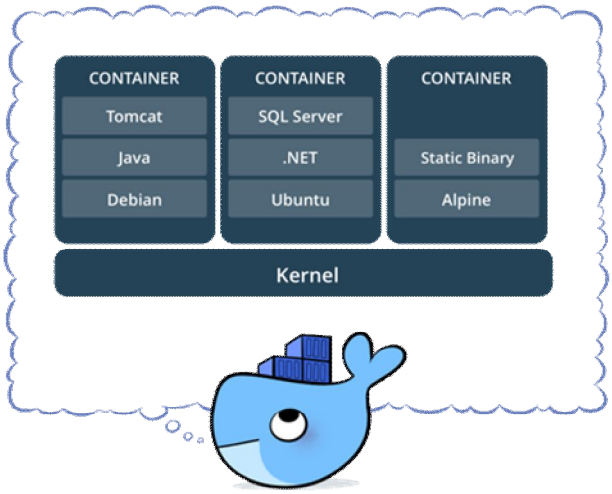

Figure 2. CONTAINER

A container image is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it i.e.,code, runtime, system tools, system libraries, settings. Available for both Linux and Windows based apps, containerized software will always run the same, regardless of the environment. Containers isolate software from its surroundings, for example differences between development and staging environments and help reduce conflicts between teams running different software on the same infrastructure.

Containers are easily packaged and designed to run anywhere. There can be multiple containers deployed in a single VM. It may take several minutes for VM's to boot up their operating systems and then begin running the applications they host, whereas containerised applications can be started almost instantly.

III. COMPARING CONTAINERS AND VIRTUAL MACHINES

Containers and virtual machines have similar resource isolation and allocation benefits, but function differently because containers virtualize the operating system instead of hardware, containers are more portable and efficient.

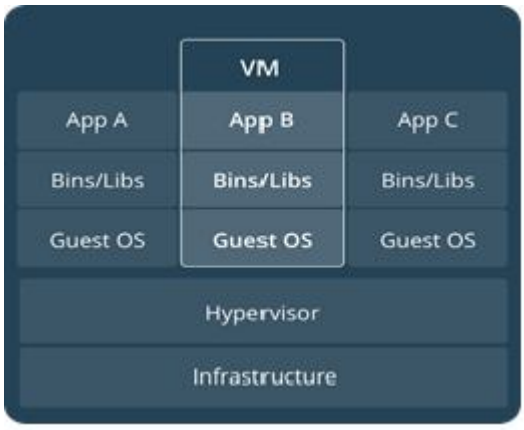

Figure 3. Virtual Machine

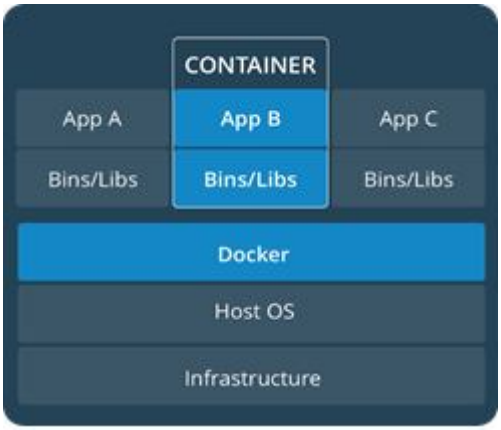

Figure 4. Container

Container:

Containers are an abstraction at the app layer that packages code and dependencies together. Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space. Containers take up less space than VMs (container images are typically tens of MBs in size), and start almost instantly.

Virtual machines

Virtual machines (VMs) are an abstraction of physical hardware turning one server into many servers. The hypervisor allows multiple VMs to run on a single machine. Each VM includes a full copy of an operating system, one or more apps, necessary binaries and libraries - taking up tens of GBs. VMs can also be slow to boot.

Tools that hosted with Docker:

Figure 5.Docker Hosting

Docker can be integrated into various infrastructure tools, including Amazon Web Services, Ansible, Chef, CFEngine or Google cloud platform, which can be easily integrated with Docker. It also interacts with tools like Jenkins, IBM Bluemix, Microsoft Azure, Openstack Nova, Salt, Vagrant and VMware Vsphere Integrated containers.

Stack has complete support for automatically performing builds inside Docker, using the user ID and volume mounts switching to make it mostly seamless. FP Complete provides images for use with Stack that also include other tools, GHC and optionally, have all of the Stackage LTS packages which are pre-installed in the global package database. The main purpose for using Stack/Docker this way is to ensure that all developers build in a consistent environment without any of the team members needing to deal with Docker on their own.

Docker Swarm

Docker Engine 1.12 includes swarm mode for natively managing a cluster of Docker Engines called a swarm.

Docker Swarm is native clustering for Docker. It turns a pool of Docker hosts into a single, virtual Docker host. Because Docker Swarm serves the standard Docker API, any tool that already communicates with a Docker daemon can use Swarm to transparently scale to multiple hosts. Supported tools include, but are not limited to, the following:

- Dokku
- Docker Compose

Cluster management integrated with Docker Engine: Use the Docker Engine CLI to create a swarm of Docker Engines where you can deploy application services. No need of additional orchestration software to create or manage a swarm.

Decentralized design: Instead of handling differentiation between node roles at deployment time, the Docker Engine handles any specialization at runtime. Deploying both kinds of nodes, managers and workers, using the Docker Engine is possible. Thus entire swarm can build from a single disk image.

Service discovery: Swarm manager nodes assign each service in the swarm a unique DNS name and load balances running containers. It is possible to query every container running in the swarm through a DNS server embedded in the swarm.

Dockers for developers:

Docker is a tool that is designed to benefit both developers and system administrators, making it a part of many DevOps (developers + operations) toolchains. For developers, it means that they can focus on writing code without worrying about the system that it will ultimately be running on. It also allows them to get a head start by using one of the thousands of programs already designed to run in a Docker container as a part of their application.

Docker runs behind a virtual machine both on Windows and Mac. use docker-machine ls for checking our docker-machine/s.

```
$ docker-machine ls
NAME ACTIVE DRIVER STATE URL SWARM DOCKER ERRORS
default - virtualbox Stopped Unknown
```

The "default" docker-machine is created as a result of the installation process. In order to drop it, command is.

```
$ docker-machine rm default
About to remove default
Are you sure? (y/n): y
Successfully removed default
```

Clustering using Docker Swarm 0.2.0:

One of the key updates as part of Docker 1.6 is Docker Swarm 0.2.0. Docker Swarm solves one of the fundamental limitations of Docker where the containers could only run on a single Docker host. Docker Swarm is native clustering for Docker. It turns a pool of Docker hosts into a single, virtual host.

Swarm Manager: Docker Swarm has a Master or Manager, that is a pre-defined Docker Host, and is a single

point for all administration. Currently only a single instance of manager is allowed in the cluster. This is a SPOF for high availability architectures and additional managers will be allowed in a future version of Swarm with #598.

Swarm Nodes: The containers are deployed on Nodes that are additional Docker Hosts. Each Swarm Node must be accessible by the manager, each node must listen to the same network interface (TCP port). Each node runs a node agent that registers the referenced Docker daemon, monitors it, and updates the discovery backend with the node's status. The containers run on a node.

IV. BENEFITS OF DOCKER

The benefits of Docker is Version Control, Portability, Isolation and Security.

- Rapid application deployment – containers include the minimal runtime requirements of the application, reducing their size and allowing them to be deployed quickly.

- Portability across machines – an application and all its dependencies can be bundled into a single container that is independent from the host version of Linux kernel, platform distribution, or deployment model. This container can be transferred to another machine that runs Docker, and executed there without compatibility issues.

- Version control and component reuse – you can track successive versions of a container, inspect differences, or roll-back to previous versions. Containers reuse components from the preceding layers, which makes them noticeably lightweight.

- Sharing – you can use a remote repository to share your container with others. Red Hat provides a registry for this purpose, and it is also possible to configure your own private repository.

- Lightweight footprint and minimal overhead – Docker images are typically very small, which facilitates rapid delivery and reduces the time to deploy new application containers.

- Simplified maintenance – Docker reduces effort and risk of problems with application dependencies.

REFERENCES

[1] http://dzone.com

[2] http://opensourceforu.com

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

[4] https://access.redhat.com