Docker - The Future of Virtualization

Asst. Prof. Mrs. Namrata Khandhar
Computer Department
SAL College of Engineering
(of Affiliation GTU)
Ahmedabad, India

Mr. Sagar Shah
Senior Software Developer
Green Apex Technolabs LLP
Ahmedabad, India

Abstract— ***Docker is a latest technology that allows development teams to build, manage, and secure apps anywhere. Docker makes it possible to get far more apps running on the same old servers and it also makes it very easy to package and ship programs. It's not possible to explain what Docker is without explaining what containers are. Containers, however, use shared operating systems. This means they are much more efficient than hypervisors in system resource terms. Instead of virtualizing hardware, containers rest on top of a single Linux instance. This means you can "leave behind the useless 99.9 percent VM junk, leaving you with a small, neat capsule containing your application.***

Keywords— docker, containers, hypervisors virtual machines, docker daemon, virtualization

I. INTRODUCTION

Docker is a platform for developers and sysadmins to develop, ship, and run applications, anywhere. Docker is popular because it has revolutionized development. Docker and the containers have revolutionized the software industry and in five short years their popularity as a tool and platform has skyrocketed. The main reason is that containers create vast economies of scale. Systems that used to require expensive, dedicated hardware resources can now share hardware with other systems. Containers are self-contained and portable. If a container works on one host, it will work just as well on any other, as long as that host provides a compatible runtime.

II. WHY DOCKER?

i. Ease of use: Docker is open-source, so all you need to get started is a computer with an operating system that supports Virtualbox, Docker for Mac/Windows, or supports containers natively, such as Linux.

ii. Faster scaling of systems: Containers allow much more work to be done by far less computing hardware.

iii. Better software delivery: Software delivery using containers can also be more efficient. Containers are portable.[3] The software dependencies (libraries, runtimes, etc.) ship with the container. If a container works on your machine, it will run the same way in a Development, Staging, and Production environment. Containers can eliminate the configuration variance problems common when deploying binaries or raw code.

iv. Flexibility: Operating containerized applications is more flexible and resilient than that of non-containerized applications

v. Docker supports software-defined networking. The Docker CLI and Engine allow operators to define isolated networks for containers, without having to touch a single router. Developers and operators can design systems with complex network topologies and define the networks in configuration files.

vi. Docker enables developers to easily pack, ship, and run any application as a lightweight, portable, self-sufficient containers, which can run virtually anywhere.

III. DOCKER ARCHITECTURE

Docker uses a client-server architecture. The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface.

A.The Docker daemon

The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.[2]

B.The Docker client

The Docker client (docker) is the primary way that many Docker users interact with Docker. When you use commands such as docker run, the client sends these commands to dockerd, which carries them out. The docker command uses the Docker API. The Docker client can communicate with more than one daemon.

C.Docker registries

A Docker registry stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry. If you use Docker Datacenter (DDC), it includes Docker Trusted Registry (DTR).

When you use the docker pull or docker run commands, the required images are pulled from your configured registry. When you use the docker push command, your image is pushed to your configured registry.[1]

D.Docker objects

When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects. This section is a brief overview of some of those objects.

IMAGES

An image is a read-only template with instructions for creating a Docker container. Often, an image is based on another image, with some additional customization. For example, you may build an image which is based on the ubuntu image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run.

You might create your own images or you might only use those created by others and published in a registry. To build your own image, you create a Dockerfile with a simple syntax for defining the steps needed to create the image and run it. Each instruction in a Dockerfile creates a layer in the image. When you change the Dockerfile and rebuild the image, only those layers which have changed are rebuilt. This is part of what makes images so lightweight, small, and fast, when compared to other virtualization technologies.

CONTAINERS

A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.

By default, a container is relatively well isolated from other containers and its host machine. You can control how isolated a container's network, storage, or other underlying subsystems are from other containers or from the host machine.

A container is defined by its image as well as any configuration options you provide to it when you create or start it. When a container is removed, any changes to its state that are not stored in persistent storage disappear.

IV DOCKER CONTAINER VS VIRTUAL MACHINE

i. There's a lot less moving parts with Docker. We don't need to run any type of hypervisor or virtual machine.

ii. Instead of having to wait a minute for a virtual machine to boot up, you can start a docker container in a few milliseconds.

iii. You also save a ton of disk space and other system resources due to not needing to lug around a bulky guest OS for each application that you run. There's also no virtualization needed with Docker since it runs directly on the host OS.[4]

iv. Virtual machines are very good at isolating system resources and entire working environments. For example, if you owned a web hosting company you would likely use virtual machines to separate each customer.

v. On the flip side, Docker's philosophy is to isolate individual applications, not entire systems. A perfect example of this would be breaking up a bunch of web application services into their own Docker images.

vi. Virtual machines have a full OS with its own memory management installed with the associated overhead of virtual device drivers. In a virtual machine, valuable resources are emulated for the guest OS and hypervisor, which makes it possible to run many instances of one or more operating systems in parallel on a single machine (or host)[5]. Every guest OS runs as an individual entity from the host system.

vii. On the other hand Docker containers are executed with the Docker engine rather than the hypervisor. Containers are therefore smaller than Virtual Machines and enable faster start up with better performance, less isolation and greater compatibility possible due to sharing of the host's kernel.

V DOCKER STATISTICS AND FACTS

i. 2/3 of companies that try using Docker, adopt it. Most companies who will adopt have already done so within 30 days of initial production usage, and almost all the remaining adopters convert within 60 days.

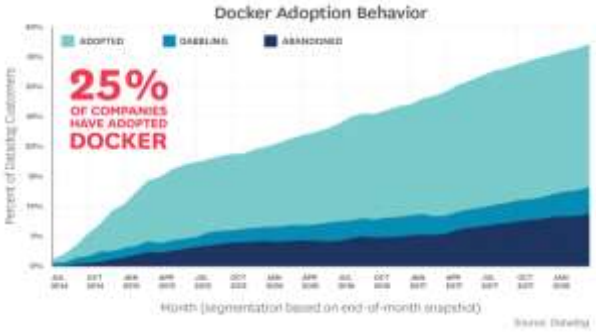

Fig.1. Month(Segmentation based on end-of-month snapshot)

ii. Docker adoption is up 30% in the last year. Adopters multiply their containers by five. Docker adopters approximately quintuple the average number of running containers they have in production between their first and

tenth month of usage.

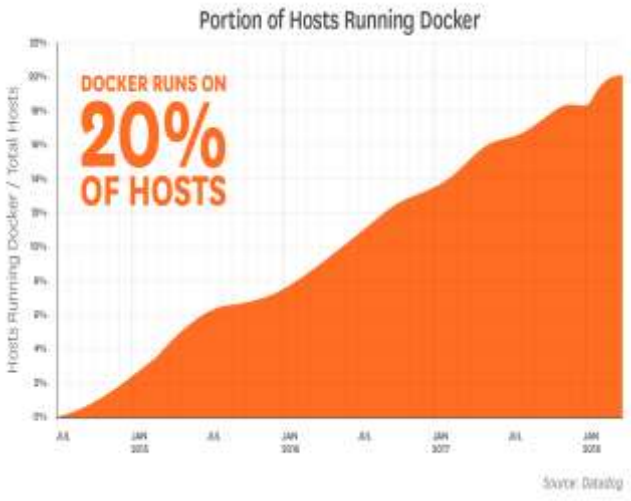

Fig. 2. Portion of hosts running docker

iii. PHP, Ruby, Java, and Node are the main programming frameworks used in containers.

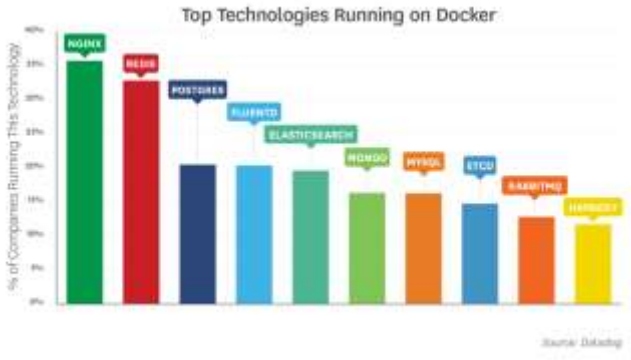

Fig. 3. Top technologies running on docker

VI BENEFITS OF DOCKER

Why do large companies like ING, Paypal, ADP, and Spotify keep using Docker? Why is Docker adoption growing that fast? Let's cover the top advantages of docker to better understand it.

A.Return on Investment and Cost Savings

The first advantage of using docker is ROI. The biggest driver of most management decisions when selecting a new product is the return on investment. The more a solution can drive down costs while raising profits, the better a solution it is, especially for large, established companies, which need to generate steady revenue over the long term.

In this sense, Docker can help facilitate this type of savings by dramatically reducing infrastructure resources. The nature of Docker is that fewer resources are necessary to run the same application. Because of the reduced infrastructure requirements Docker has, organizations are able to save on everything from server costs to the employees needed to maintain them. Docker allows engineering teams to be smaller and more effective.[6]

B. Standardization and Productivity

Docker containers ensure consistency across multiple development and release cycles, standardizing your environment. One of the biggest advantages to a Docker-based architecture is actually standardization.

Docker provides repeatable development, build, test, and production environments. Standardizing service infrastructure across the entire pipeline allows every team member to work in a production parity environment. By doing this, engineers are more equipped to efficiently analyze and fix bugs within the application. This reduces the amount of time wasted on defects and increases the amount of time available for feature development.

As we mentioned, Docker containers allow you to commit changes to your Docker images and version control them. For example, if you perform a component upgrade that breaks your whole environment, it is very easy to rollback to a previous version of your Docker image. This whole process can be tested in a few minutes. Docker is fast, allowing you to quickly make replications and achieve redundancy. Also, launching Docker images is as fast as running a machine process.

C. CI Efficiency

Docker enables you to build a container image and use that same image across every step of the deployment process. A huge benefit of this is the ability to separate non-dependent steps and run them in parallel. The length of time it takes from build to production can be sped up notably.

D. Compatibility and Maintainability

Eliminate the "it works on my machine" problem once and for all. One of the benefits that the entire team will appreciate is parity. Parity, in terms of Docker, means that your images run the same no matter which server or whose laptop they are running on. For your developers, this means less time spent setting up environments, debugging environment-specific issues, and a more portable and easy-to-set-up codebase. Parity also means your production infrastructure will be more reliable and easier to maintain.

E. Rapid Deployment

Docker manages to reduce deployment to seconds. This is due to the fact that it creates a container for every process and does not boot an OS. Data can be created and destroyed without worry that the cost to bring it up again would be higher than what is affordable.

F. Continuous Deployment and Testing

Docker ensures consistent environments from development to production. Docker containers are configured to maintain all configurations and dependencies internally; you can use the same container from development to production making sure there are no discrepancies or manual intervention.

G. Multi-Cloud Platforms

One of Docker's greatest benefits is portability. Over the last few years, all major cloud computing providers,

including Amazon Web Services (AWS) and Google Compute Platform (GCP), have embraced Docker's availability and added individual support.

Docker containers can be run inside an Amazon EC2 instance, Google Compute Engine instance, Rackspace server, or VirtualBox, provided that the host OS supports Docker. If this is the case, a container running on an Amazon EC2 instance can easily be ported between environments, for example to VirtualBox, achieving similar consistency and functionality.

Also, Docker works very well with other providers like Microsoft Azure, and OpenStack, and can be used with various configuration managers like Chef, Puppet, and Ansible, etc.

H. Isolation

Docker ensures your applications and resources are isolated and segregated. Docker makes sure each container has its own resources that are isolated from other containers. You can have various containers for separate applications running completely different stacks. Docker helps you ensure clean app removal since each application runs on its own container. If you no longer need an application, you can simply delete its container. It won't leave any temporary or configuration files on your host OS.

On top of these benefits, Docker also ensures that each application only uses resources that have been assigned to them. A particular application won't use all of your available resources, which would normally lead to performance degradation or complete downtime for other applications.

I. Security

The last of these benefits of using docker is security. From a security point of view, Docker ensures that applications that are running on containers are completely segregated and isolated from each other, granting you complete control over traffic flow and management. No Docker container can look into processes running inside another container. From an architectural point of view, each container gets its own set of resources ranging from processing to network stacks.

VII LIMITATIONS OF DOCKER

Some disadvantages of Docker are described as below:

i. Missing features

There are a ton of feature requests are under progress, like container self-registration, and self-inspects, copying files from the host to the container, and many more.

ii. Data in the container

There are times when a container goes down, so after that, it needs a backup and recovery strategy, although we have several solutions for that they are not automated or not very scalable yet.

iii. Run applications as fast as a bare-metal serve

In comparison with the virtual machines, Docker containers have less overhead but not zero overhead. If we run, an application directly on a bare-metal server we get true bare-metal speed even without using containers or virtual machines. However, Containers don't run at bare-metal speeds.

iv. Provide cross-platform compatibility

The one major issue is if an application designed to run in a Docker container on Windows, then it can't run on Linux or vice versa. However, Virtual machines are not subject to this limitation. So, this limitation makes Docker less attractive in some highly heterogeneous environments which are composed of both Windows and Linux servers.

v. Run applications with graphical interfaces

In general, Docker is designed for hosting applications which run on the command line. Though we have a few ways (like X11 forwarding) by which we can make it possible to run a graphical interface inside a Docker container, however, this is clunky. Hence we can say, for applications that require rich interfaces, Docker is not a good solution.

SUMMARY

As you can imagine, the scope of getting from full-fledged servers, to OS virtualization and then to containers is to remove the burden of building, deploying and maintaining an entire operating system when what they only need is just the application layer.With this paper, we tried to present you with some fundamental events that led us to the creation of the Docker containers.

REFERENCES

[1] Boettiger, C.: An introduction to docker for reproducible research. ACM SIGOPS Oper. Syst. Rev. **49**(1), 71–79 (2015)
[2] Gerber, A.: The state of containers and the docker ecosystem: 2015. Technical report, White paper
[3] Scheepers, Mathijs Jeroen. (2014). Virtualization and containerization of application infrastructure: A comparison. Paper presented at the 21st Twente Student Conference on IT.
[4]Jurenka, Vladimir.(2015).Virtualization using Docker Platform.https://edisciplinas.usp.br/ plug infile.php/318402/course/section/93668/thesis_3.pdf
[5]https://www.airpair.com/docker/posts/8-proven-real-world-ways-to-use-docker
[6] https://www.contino.io/insights/whos-using-docker