**How Docker changed the world for the better**

Before 2013, software and DevOps engineers found it increasingly challenging to test, deploy, update and maintain software on multiple servers with multiple hardware configurations. Even when the server hardware was identical, the software running on them, like the operating system or the version of PHP or Node.js or Python, was slightly different, which caused countless issues and headaches. Especially when the software being deployed uses the Microservices architecture.

Before the deployment stage, giving the software for testing or development purposes to a co-worker with a slightly different software or hardware configuration would cause a disaster. These issues plagued engineers for years, and something had to be done. The same issue was facing my team and me in 2020 at Alshaya Group.

**The company**

Alshaya Group (Alshaya Group About Us, 2022) is a large corporation that has many branches of large brands open in the middle east. Some of these brands are Starbucks, H&M, Bath & Body Works, Foot Locker and many more. I was hired by Alshaya back in 2020 as a senior React Native engineer to work on the front-end side of the many brand apps that Alshaya manages. While I worked on the front-end side of the app, I also worked closely with the backend lead engineer to help deploy the many services and backend software the company needs to make their apps work reliably and efficiently.

**The problem**

During the development of the backend application and services, we faced many issues in the testing and deployment of those apps. The problem was that our backend services would work perfectly fine on our testing machines. However, when we would deploy these services on a final testing production server, we would receive a difficult hard to decipher error messages from applications. We spent countless hours figuring out what was the issue and what was causing it. After many hours of testing and debugging, we discovered that the Node.js version installed on the production test machine was older than what the developers used to develop and test their applications.

A year or so went by, and the team needed to scale the backend services onto multiple servers to accommodate for the increased network traffic from customer usage. So the team went ahead and set up and configured a few new virtual machines and deployed our services on them. However, shortly after running our services, all of them crashed. Again, a new undecipherable message was printed out on the console; however, this time it was not caused by Node.js since the team made sure all the servers were running the same exact Node.js version as the development and testing machines. Another few hours of debugging later, the cause was found when we discovered that one of the software packages was updated automatically and had caused a bug in our code.

That was the final straw. Not only did this issue happen twice, causing delays in deploying our products, but the fact that we had to set up a new virtual machine every single time we needed to scale our backend was frustrating and time-consuming. There had to be a better way to do this.

**The solution**

Enters Docker (Docker overview, 2022). Docker's initial release was back in 2013. Since then, Docker has grown tremendously, and many tech companies have adopted it into their day-to-day work. The entire idea behind Docker is that software or DevOps engineers do not have to worry about the target machine hardware, operating system, or what software version is installed. Instead, the software engineer creates a container image and packages their applications alongside the environment needed to run them. Then, all the DevOps engineer has to do is mount the image on the target machine and run it. Simple as that. There is no need to spend hours installing the correct operating system version or the development environment.

Docker has many other benefits, like the fact that it is very lightweight and can run multiple containers simultaneously, taking advantage of the target hardware resources. Unlike a traditional virtual machine which is very heavyweight and does not utilize the target hardware resources as efficiently as possible. Traditional virtual machines have a hypervisor that the guest operating systems run on. That adds a layer of complexity, slows down the machine, and takes a tremendous amount of resources. On the other hand, Docker runs a Docker demon service that is very lightweight and fast and can start in less than 15 milliseconds. In contrast, traditional virtual machines take seconds or even minutes to start. Furthermore, Docker has developed a new way to manage multiple containers across multiple virtual machines or multiple physical machines all at once. This technology is called Docker Swarm (What is Docker Swarm? | Sumo Logic, 2022).

After researching Docker for a few days, it seemed like the perfect solution for the problems the team was facing. Not only will we be able to package our software's into multiple or a single container and deploy them without having to install tens if not hundreds of different packages but we will be able to do that at scale. We will be able to scale up our services by simply creating a new virtual machine, installing Docker, running our containers, and managing all of them using Docker Swarm.

After informing my team lead of the results of my research, he was happy and approved going in that direction.

**The plan**

Before converting all of our software to use Docker or what has otherwise known as Dockerizing an application which is the process or packaging, deploying, and running an application using a Docker container, we had to first test it and make sure it actually works.

So my job for the next few days was to take one service from our backend and Dockerize it and see if it actually works. The way to do that is first to pick the base image on which our software will live. There are many base images such as Ubuntu, alpine, redis, and more. After doing some more research, it seemed like the alpine image was the best base image for the job. The alpine image contained a minimalistic Linux distribution that was only 5 MB in size (Docker Hub, 2022). This was the perfect image for our needs. Not only was the image 5 MB in size, which meant the footprint of our containers would be very small, but it allowed us to customize our containers the way we wanted.

Once we picked the base image, it was time to write the Dockerfile. A Dockerfile instruct Docker on which image to use as the base image and how to build and package our software into a container.

In the Dockerfile, we start by telling Docker which image is going to use as the base image and what version it is. Next, we tell Docker what files to copy into the image and where to copy them to inside the base image. Next, we tell Docker what software do we need to install onto this base image. Things like Node.js, python, etc.… We also mention the version number of each of these software. Next, we tell Docker how to run our services, and we specify the port number it should run on.

With all that done, we simply tell Docker to run the Dockerfile, and a few moments later (if everything went according to plan), we should have a Docker container ready to be run. One important thing to note is that when building a Dockerfile, the container has to be run by the same type of operating system it was built by. For example, if a Docker container was created on a Windows machine, that container image will only work on a Windows-based machine. The same goes for different operating systems like Linux and macOS. A container built on a Windows machine will not run on Linux or macOS. That is one of the downsides of Docker.

After using and testing Docker on one of our services successfully, we decided to move forward with the Dockerization of our entire backend.

**Pros and Cons of using Docker**

Docker is an amazing tool. However, all tools have advantages and disadvantages, and Docker is no exception.

**Advantages**

1. **Rapid Development**

Due to how Docker works, rapid development and testing is much easier with Docker. Docker can decree the development and testing time by a lot due to the fact that it only takes a few seconds to build a Docker container and even fewer milliseconds to start one.

1. **Simplicity in its usage and fast configuration**

Once you know and understand how Docker works, it becomes really easy almost second nature to build and deploy a Docker container. Docker is fairly easy to learn and very easy to work with.

1. **Security**

Each application that runs inside a Docker container is completely isolated and segregated from other containers and from the host machine. From a security point of view, that is great

1. **Wide use in existing cloud infrastructure**

Docker has been widely adopted by many developers, system engineers, and cloud service providers like Google, Microsoft, and Amazon. All of these platforms support Docker and allow for seamless integration.

1. **Easy to scale**

Because of how Docker works, it is easy to scale containers on multiple virtual machines. With the help of Swarm, managing all these Docker containers becomes trivial.

**Disadvantages**

1. **Staff training**

While Docker is easy to learn and use, there is a learning curve to it. As a team lead or a chief technology officer, you must account for the amount of time that it would take to transition the entire organization's software to use Docker and the amount of time it would take to train the entire staff to learn and use Docker efficiently and effectively.

1. **Runs best in command line**

Docker was designed to run and be used in the command line. While there is a user interface that exists for Docker, it is not great and does not allow you to access all of Docker's features.

1. **Lack of cross-platform compatibility**

As mentioned before, when a Docker image is built on a Linux machine, it will only run on a Linux machine and won't run on a Windows or a macOS machine.

In conclusion, Docker is an extremely useful and essential tool to learn and use in today's software engineering world. Every software engineer must learn, understand and be able to work with Docker as it becomes an essential tool in every software engineer's tool belt. More and more job postings are requiring Docker as an important skill to have.

References

1. Alshaya Group. 2022. *Alshaya Group About Us*. [online] Available at: <https://www.alshaya.com/en/about-us/> [Accessed 30 May 2022].
2. Docker Documentation. 2022. *Docker overview*. [online] Available at: <https://docs.docker.com/get-started/overview/> [Accessed 30 May 2022].
3. Sumo Logic. 2022. *What is Docker Swarm? | Sumo Logic*. [online] Available at: <https://www.sumologic.com/glossary/docker-swarm/> [Accessed 30 May 2022].
4. Hub.docker.com. 2022. Docker Hub. [online] Available at: <https://hub.docker.com/\_/alpine> [Accessed 30 May 2022].