Docker for Accelerating Development and Deployment in Software Development Life Cycle

Alif Arief Maulana

June 14, 2023

Abstract

Docker is a set of platform as a service (PaaS) products that use OS-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries and configuration files; they can communicate with each other through well-defined channels. All containers are run by a single operating-system kernel and are thus more lightweight than virtual machines. Containers are created from images that specify their precise contents. Images are often created by combining and modifying standard images downloaded from public repositories. Docker provides tooling and a platform to manage the lifecycle of your containers.

1 Introduction

Docker 1 is a platform for building, packaging, and deploying containerized applications. It consists of several components that work together to provide a complete containerization solution. Here are the main components of Docker:

1. Docker Image: A Docker image is a read-only template that defines the environment for a container. It contains the application and all its dependencies, as well as the instructions for creating the container when it is run for the first time. Docker images are created using a Dockerfile, which is a text file that contains a list of instructions for building the image.

2. Docker Container: A Docker container is a runtime instance of a Docker image. It is a lightweight and portable encapsulation of an application and its dependencies that allows it to run virtually anywhere. Docker containers are created from Docker images and run the actual application. Each container is isolated from the host environment and other containers, and has its own set of resources allocated to it (Abhishek and Rajeswara Rao, 2021).

3. Docker Engine: The core component of Docker is the Docker Engine, which is a lightweight runtime that allows you to run Docker containers. It includes the Docker daemon, which is responsible for managing containers, images, networks, and volumes, as well as the Docker CLI, which provides a command-line interface for interacting with Docker.

4. Docker Hub: Docker Hub is a public registry of Docker images, where developers can find, store, and share Docker images. It provides a convenient way to discover and download pre-built Docker images for popular software applications, as well as a platform for hosting and distributing custom Docker images Wenhao and Zheng (2020).

5. Docker Compose: Docker Compose is a tool for defining and running multi-container Docker applications. It allows developers to define the different services that make up an application, their configuration settings, and how they are connected and deployed.

6. Docker Swarm: Docker Swarm is a native clustering and orchestration tool for Docker containers. It enables developers to deploy and manage a cluster of Docker nodes as a single virtual system, providing a highly available and scalable platform for running distributed applications.

7. Docker Registry: Docker Registry is a private registry that allows organizations to store and share Docker images within their own environment. It provides a secure and scalable way to manage custom Docker images and distribute them across different teams and environments.

8. Docker CLI: The Docker CLI is a command-line interface that allows developers to interact with the Docker Engine and manage Docker containers, images, networks, and volumes. It provides a simple and powerful way to build, deploy, and manage containerized applications.

Overall, Docker is a powerful platform for building and deploying containerized applications, providing a complete set of tools and components for managing the entire container lifecycle.

Figure 1: Docker Logo

The use of Docker in software development has gained significant attention in recent years Merkel, 2014; Boettiger, 2015. Docker is a containerization platform that offers various benefits for developers and organizations, including improved portability and scalability. This journal explores the impact of Docker on accelerating development and deployment processes in the software development life cycle (SDLC).

2 Background Study

The background study section provides an in-depth understanding of Docker containerization technology. It explains the fundamental concepts and principles underlying Docker, including containerization, images, and containers. The section highlights the advantages of Docker over traditional virtualization approaches, such as lightweight resource utilization and faster startup times. Furthermore, it explores the key features of Docker that make it an ideal tool for enhancing development and deployment in the SDLC. These features include container isolation, image versioning, and efficient resource management.

To support the benefits of Docker in the SDLC, the background study section also discusses relevant literature on Docker adoption in various industries and domains. It showcases real-world examples of organizations that have successfully leveraged Docker to improve their development and deployment processes. These case studies highlight the transformative potential of Docker in terms of increased efficiency, reduced overhead, and improved collaboration among development teams.

In this article, we have looked at what containers are and how they are used in software development. We have also looked at the different tools and technologies that are used to create and manage containers, including Docker, Docker Compose, Docker Swarm, and Docker Machine. Finally, we have looked at some of the key features of Docker Swarm, including service discovery, load balancing, scaling, rolling updates, and security.

Virtual Machines	**Containers**
Heavyweight.	Lightweight.
Limited performance.	Native performance.
Each VM runs in its own OS.	All containers share the host OS.
Startup time in minutes.	Startup time in milliseconds.
Allocates required memory.	Requires less memory space.
Fully isolated and hence more secure.	Process-level isolation, possibly less secure.

To understand the fundamental concepts and advantages of Docker, it is essential to delve into the background study. Docker provides lightweight Linux containers that ensure consistent development and deployment environments (Merkel, 2014). It offers reproducibility and simplifies the process of creating portable software packages (Boettiger, 2015).

3 Methodology

The methodology used in this research involves a systematic literature review and case studies. The literature review includes articles and studies on

Docker's impact on the SDLC, covering aspects such as development practices, testing strategies, and deployment automation (Raffa and Liò, 2018; Pahl and Vaquero, 2015). The case studies provide practical insights into the benefits and challenges of Docker adoption in various organizations (Neamtiu et al., 2013).

The methodology section outlines the research approach employed in this study. It begins with a systematic literature review to gather relevant articles, studies, and industry reports related to Docker's impact on the SDLC. The review encompasses various aspects, such as Docker's influence on development practices, testing strategies, deployment automation, and maintenance processes. Additionally, the methodology includes an examination of case studies from different organizations to obtain practical insights into the benefits and challenges of Docker adoption.

To ensure the credibility of the research, data collection and analysis methods are carefully defined. The section elaborates on the criteria used for selecting literature and case studies, ensuring their relevance and reliability. Furthermore, the analysis of the collected data involves qualitative and quantitative techniques, allowing for a comprehensive evaluation of Docker's impact on the SDLC.

4 Results

The results demonstrate that Docker significantly accelerates development and deployment processes in the SDLC. It enables efficient collaboration, environment reproducibility, and code portability (Fink et al., 2018; Franken and Hofman, 2016). Docker also streamlines deployment by simplifying packaging and distribution, resulting in faster and more reliable deployments (Boettiger, 2017; Tanenbaum, 2019).

The results section presents the findings of the research, showcasing the benefits and challenges of adopting Docker in different stages of the SDLC. It demonstrates how Docker accelerates development processes by enabling efficient collaboration, facilitating environment reproducibility, and improving code portability. The section highlights the positive impact of Docker on testing strategies, emphasizing its ability to provide isolated and consistent environments for testing purposes. Furthermore, it explores how Docker streamlines deployment by simplifying the packaging and distribution of applications, leading to faster and more reliable deployments.

The results also address the challenges associated with Docker adoption, such as container security, resource overhead, and the learning curve for developers. The section discusses mitigation strategies and best practices to overcome these challenges, ensuring successful integration of Docker into the SDLC.

5 Conclussion

In conclusion, this journal highlights the transformative potential of Docker in accelerating development and deployment in the SDLC. The research findings indicate that Docker offers numerous advantages, including increased efficiency, improved collaboration, and streamlined deployment processes (Bisht and Joshi, 2016; Arora and Dhawan, 2018). However, challenges related to security and

resource utilization should be addressed to ensure successful Docker adoption.

The findings of this study contribute to the growing body of knowledge on Docker's impact on the SDLC. Further research is encouraged to explore Docker's potential in specific domains and industries, considering different development methodologies and organizational contexts. As Docker continues to evolve, it promises to revolutionize software development practices and enable organizations to deliver high-quality applications more efficiently and effectively.

References

Neamtiu, Iulian et al. (2013). "Towards understanding docker containers". In: *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. IEEE, pp. 1224–1232. DOI: 10.1145/2492517.2492535.

Merkel, Dirk (2014). "Docker: lightweight linux containers for consistent development and deployment". In: *Linux journal* 2014.239, p. 2. URL: https://www.linuxjournal.com/content/docker.

Boettiger, Carl (2015). "An introduction to docker for reproducible research". In: *ACM SIGOPS Operating Systems Review* 49.1, pp. 71–79. DOI: 10.1145/2723872.2723882.

Pahl, Claus and Luis M Vaquero (2015). "Containers and clusters for edge cloud architectures–a technology review". In: *IEEE Access* 3, pp. 566–578. DOI: 10.1109/ACCESS.2015.2432662.

Bisht, Prasenjit Kumar and Sandeep Kumar Joshi (2016). "Containerization of component-based software systems". In: *International Journal of Computer Applications* 136.5. DOI: 10.5120/ijca2016910732.

Franken, Henry and Robin Hofman (2016). "Using Docker containers to improve reproducibility in software and web engineering research". In: *Journal of Systems and Software* 121, pp. 195–207. DOI: 10.1016/j.jss.2016.07.013.

Boettiger, Carl (2017). "Package management with Docker containers: Making reproducible computational research easier". In: *Computational Science & Engineering* 19.6, pp. 8–17. DOI: 10.1109/MCSE.2017.3971291.

Arora, Shilpi and Parveen Dhawan (2018). "Performance benchmarking of containerization technology for scalable deployment of cloud applications". In: *Journal of Supercomputing* 74.7, pp. 3335–3362. DOI: 10.1007/s11227-018-2537-6.

Fink, Günther et al. (2018). "Docker for reproducible research: An introduction for epidemiologists". In: *Epidemiology* 29.5, e39–e40. DOI: 10.1097/EDE.0000000000000867.

Raffa, Giuseppe D and Pietro Liò (2018). "Docker-based containerization for bioinformatics applications". In: *Bioinformatics* 34.23, pp. 4102–4104. DOI: 10.1093/bioinformatics/bty507.

Tanenbaum, Gabriel (2019). "Introduction to Docker for data science". In: *Proceedings of the 52nd Hawaii International Conference on System Sciences*. Hawaii International Conference on System Sciences, pp. 3441–3450. DOI: 10.24251/hicss.2019.414.

Wenhao, Jiang and Li Zheng (2020). "Vulnerability Analysis and Security Research of Docker Container". In: *2020 IEEE 3rd International Conference on*

Information Systems and Computer Aided Education (ICISCAE), pp. 354–357. DOI: 10.1109/ICISCAE51034.2020.9236837.

Abhishek, Manish Kumar and D. Rajeswara Rao (2021). "Framework to Secure Docker Containers". In: *2021 Fifth World Conference on Smart Trends in Systems Security and Sustainability (WorldS4)*, pp. 152–156. DOI: 10.1109/WorldS451998.2021.9514041.