



University of Wales
Trinity Saint David
LONDON

Department of Cyber Security and Digital Forensics Master's Programme

Module: ACCA7027 – Cloud Engineering Principles and Practice

**Assignment Title: Cloud Computing , Docker Containerisation and
serverless functions.**

Submitted by :	Shahriar Badhan
Submitted To:	Renuga Jayakumar
Student ID :	2505521
Submission Date :	18 December 2025

Table of Contents

1. Introduction	1
2. Principles of Virtualisation	3
2.1 Virtualisation of Shared Hardware	4
3. Principles and Benefits of Containerisation	6
3.1 Benefits of Containerisation in the Cloud	7

4. Constructing a Docker Image Using a Dockerfile	9
4.1 Dockerfile and Application Setup	11
4.2 Docker Volumes	14
4.3 Container Networking	17
5. Load Balancing, Scalability, and Application Gateways	20
5.1 Load Balancing	21
5.2 Scalability	22
5.3 Application Gateways	24
6. Relationship Between Load Balancing and Scalability	26
7. Benefits of Combining Load Balancing, Scalability, and Application Gateways	28
8. Cloud Providers and Container-Based Applications	30
9. Summary of Key Concepts	32
10. Motivational Factors for Cloud Computing Growth.....	19
11.	34
10.1 Cost Efficiency	35
10.2 Flexibility and Agility	37
10.3 Reliability and Global Reach	39
12. Serverless Computing (Functions as a Service)	41
11.1 Benefits of Serverless Computing	42
13. Comparison of Serverless Computing and Containerisation	44
12.1 Containerisation	45
12.2 Serverless Computing	47
12.3 Identifying the Driving Forces	49
14. Challenges and Limitations of Containerisation in Cloud Environments	51
15. Critical Discussion	54
16.Dockerhub Demonstrate video link:.....	25
17. Conclusion	57
18. References	59

1.Introduction:

A popular model of computation on the Internet is Cloud computing, which is increasingly becoming an essential part of modern IT environments²⁹. With the growing need for cloud platforms, organisations/healers are using these platforms to host applications, store data and

provide services in a scalable and efficient way. On-Premises Limitations On-premises solutions of the past lack the flexibility, speed and cost-savings expected today. For this reason, cloud-based methods are substituting much of the legacy infrastructure.

One of the most important innovations for cloud computing is containerization, in particular Docker. Application containerisation enables the adoption of a packaging mechanism where an application and all its dependencies are isolated from other parts of the system. It addresses the problem of deployments being made incorrectly, not scalable and adheres to current software practices, including DevOps and continuous integration.

This article examines the importance of containerisation as it applies to cloud computing and explains why many believe it to be indispensable technology when deploying applications in a cloud. We start with the fundamentals of virtualisation, then cover containerisation and Docker. The whole work shows examples of Dockerfile construction, images manipulation, volumes and networks with practical exercises performed in the classroom. These are they that illustrate the economic delivery of applications in a cloudbased world using containerisation.

2. Principles of Virtualisation

Principles of Virtualisation Thus it is the very foundation upon which cloud computing is built. That is, abstracting physical computing resources enables multiple virtual systems to operate simultaneously on a single physical machine. This abstraction is created using a software layer known as a hypervisor, which manages access to hardware resources such as CPU, memory and storage.

In a virtualised environment, each virtual machine (VM) operates independently of other virtual machines on the same host and can therefore run its own operating system and applications. This separation improves security and reliability; it also enables better use of physical resources. Virtualisation makes possible also enables fast creation, modification or deletion of systems--a requirement for cloud-based platforms where demand can alter with little warning [1].

2.1 Virtualisation of Shared Hardware

While virtualisation offers many benefits, one of its great advantages is the ability to divide hardware resources effectively among VMs-shared workloads. Instead of having a separate physical server for every application you run, several virtual machines can use the same hardware. VM acts as if it had exclusive access to the system but in fact resources are shared among users behind the scene.

In cloud data centers, this approach reduces operational costs by making the most out of hardware utilization. It allows cloud provides to offer services like pay-as-you-use where a customer only pays for what resources he uses. Nevertheless, despite these benefits virtual machines are resource-intensive as every VM needs to run a full operating system. This in turn contributed to the birth of containerization, which offers a less ponderous alternative [2].

3. Principles and Benefits of Containerisation

Containerisation is a form of virtualisation that operates at operating system level rather than hardware level. Instead of running multiple operating systems, containers share the host operating system kernel while maintaining isolation between applications. Docker is the most widely adopted platform for containerisation.

Containers bundle an application and its libraries, dependencies, configuration files, upfiles. This ensures that the application behaves the same regardless of where it is deployed on the network. Therefore, developers may build any number of applications locally and sign them up for services without concern for compatibility issues between different instances.

3.1 Benefits of Containerisation in the Cloud

Advantages Containerisation has some advantages, which make it well-suited to cloud computing:

Efficiency: Containers consume less system resources than virtual machines (VMs) do, because they do not have an operating system of their own.

Portability: It is easily transportable to different environments and cloud types.

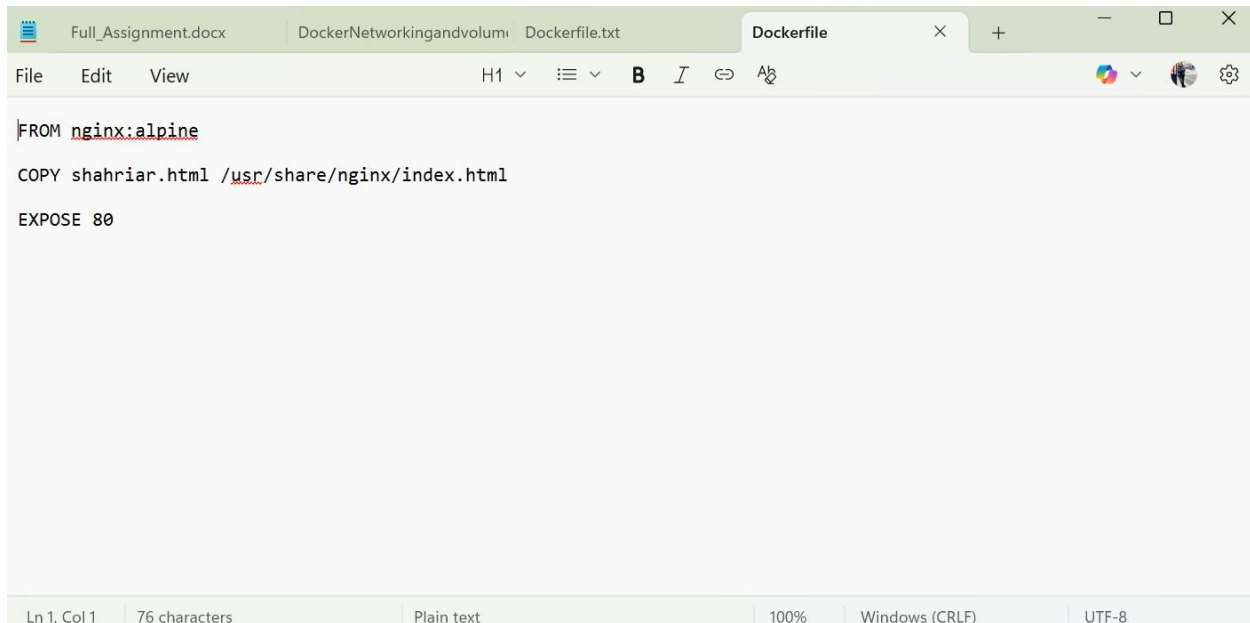
Scalability: Containers can be created and destroyed very quickly, which leads to rapid scalability.

Portability: Sames container image can be running all the environment such as dev, test and prod [3].

These advantages of containerisation position it as a key component in today's cloud-native application development.

4. Constructing a Docker Image Using a Dockerfile

Figure-1



```
FROM nginx:alpine
COPY shahriar.html /usr/share/nginx/index.html
EXPOSE 80
```

The screenshot shows a code editor window titled 'Dockerfile'. The editor has a menu bar with 'File', 'Edit', and 'View'. Below the menu bar is a toolbar with various icons for text formatting and editing. The main text area contains the following Dockerfile instructions: `FROM nginx:alpine`, `COPY shahriar.html /usr/share/nginx/index.html`, and `EXPOSE 80`. At the bottom of the editor, there is a status bar showing 'Ln 1, Col 1', '76 characters', 'Plain text', '100%', 'Windows (CRLF)', and 'UTF-8'.

A **Dockerfile** is a text file that contains a set of instructions used to build a Docker image. Docker images serve as templates for creating containers. By defining the environment in code, Dockerfiles support repeatable and automated deployments.

Line 1: FROM nginx:alpine

The base image and so synthesis the Docker image will be the specifies one by the FROM instruction. Here, Nginx has been used as the Web Server. The alpine tag is a light weight Linux distribution that helps to keep the overall image size down and increases performance. This is by no means an unusual combination when it comes to Nginx and Alpine; people tend to use the two together for lean, fast web server containers.

Line 2: COPY shahriar.html /usr/share/nginx/html/index.html

The web application file is copied from the local project directory into the container using COPY instruction. The file shahriar.html is the custom HTML web page I made for this project. It is copied into /usr/share/nginx/html/index.html" - the Nginx document root directory. Renaming the file to index.html: Allows loading into a web page as the server is contacted with a browser.

After we told it what to do, save that in to a file called Dockerfile with no extension. Docker then uses this file to build the image for the web server.

Figure-2 Docker File

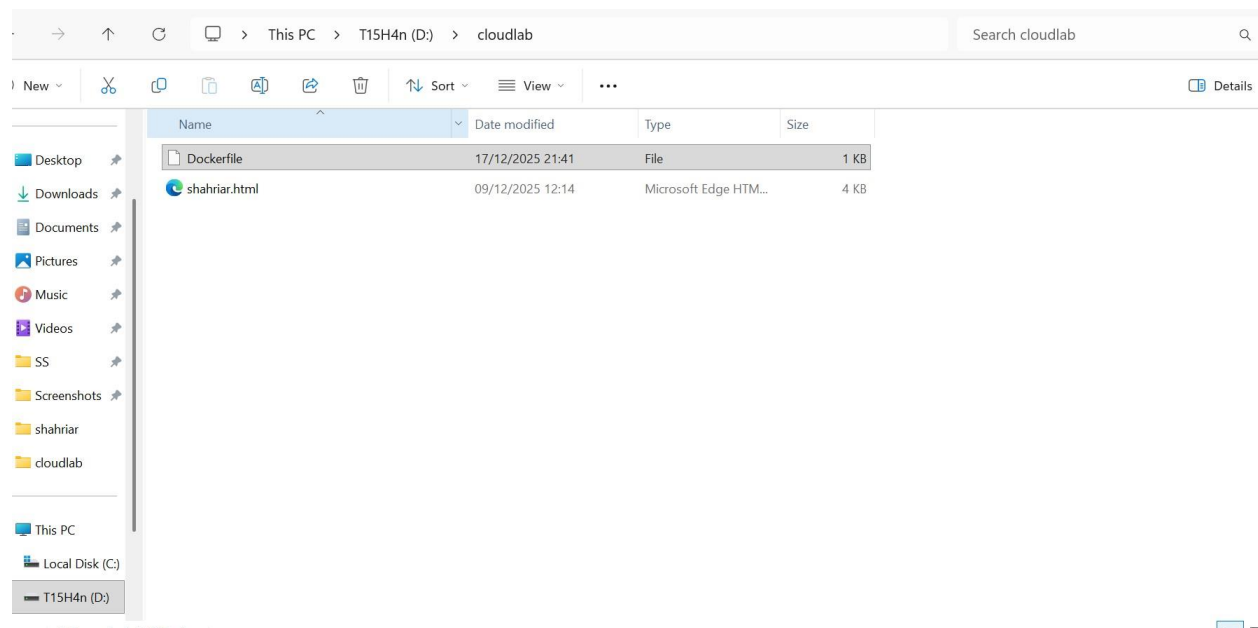
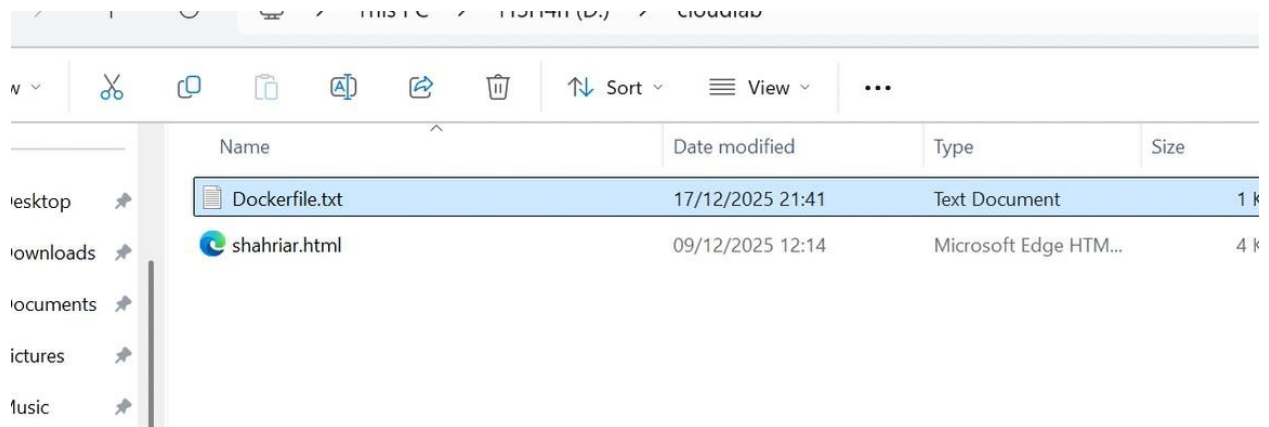


Figure-3 Docker txt



4.1 Dockerfile and Application Setup

Figure 03 shows the final appearance of the Dockerfile after the .txt extension has been removed. The file must be saved simply as **Dockerfile** with no extension in order for Docker to recognise it correctly.

It is important that both the Dockerfile and the web application file are stored in the **same folder**. If the files are placed in different directories, the Docker build process will fail because Docker will not be able to locate the required application file during image creation.

Once the Dockerfile and application file are correctly placed, the project is ready for image creation. Before executing the docker build command, it is necessary to ensure that Docker Desktop is operating within the correct working directory. If the command prompt is not already pointing to the project folder, the directory must be changed using the appropriate command. This step ensures that Docker can successfully access all required files and generate the image without errors.

Figure-4

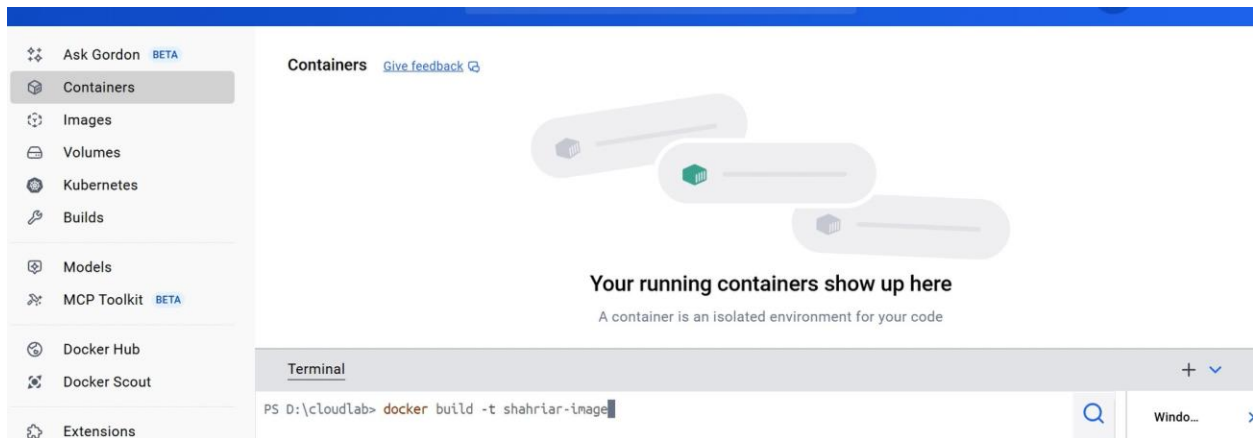
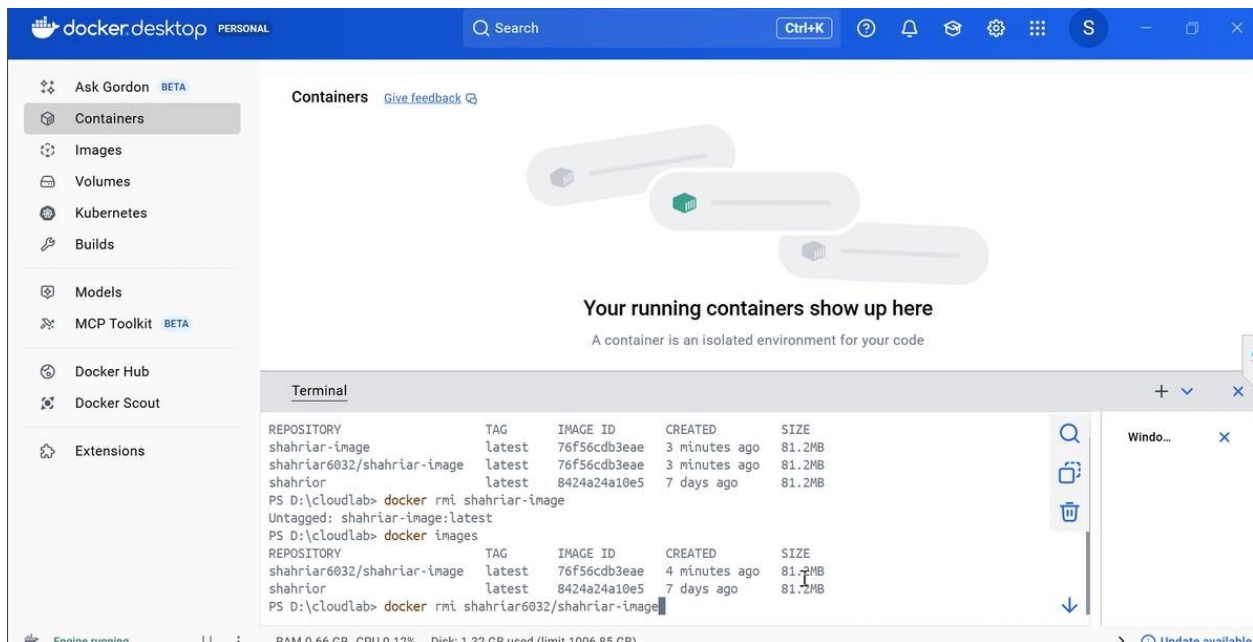


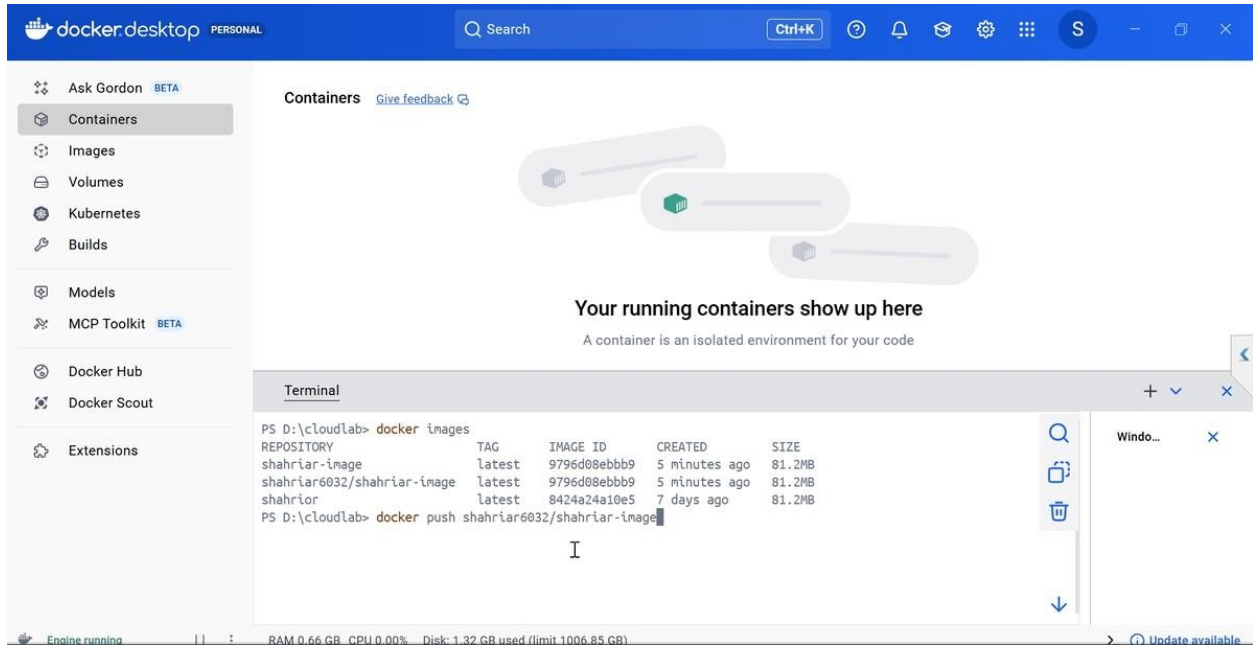
Figure-5



Docker outputs verbose process logs in the image build that can give you a nice breakdown on the steps of each stage. These messages show what Docker is doing - downloading the base image, copying your app's files and putting everything together in a final image.

This minutia of output is actually nice to have, because it lets the user see where in the build process we are and ensure that every step was completed without error. And if any failure occurs during image generation, the build output is very helpful to locate where there was anything wrong. This simplifies debugging when files are absent, not properly named or excluded from the build context.

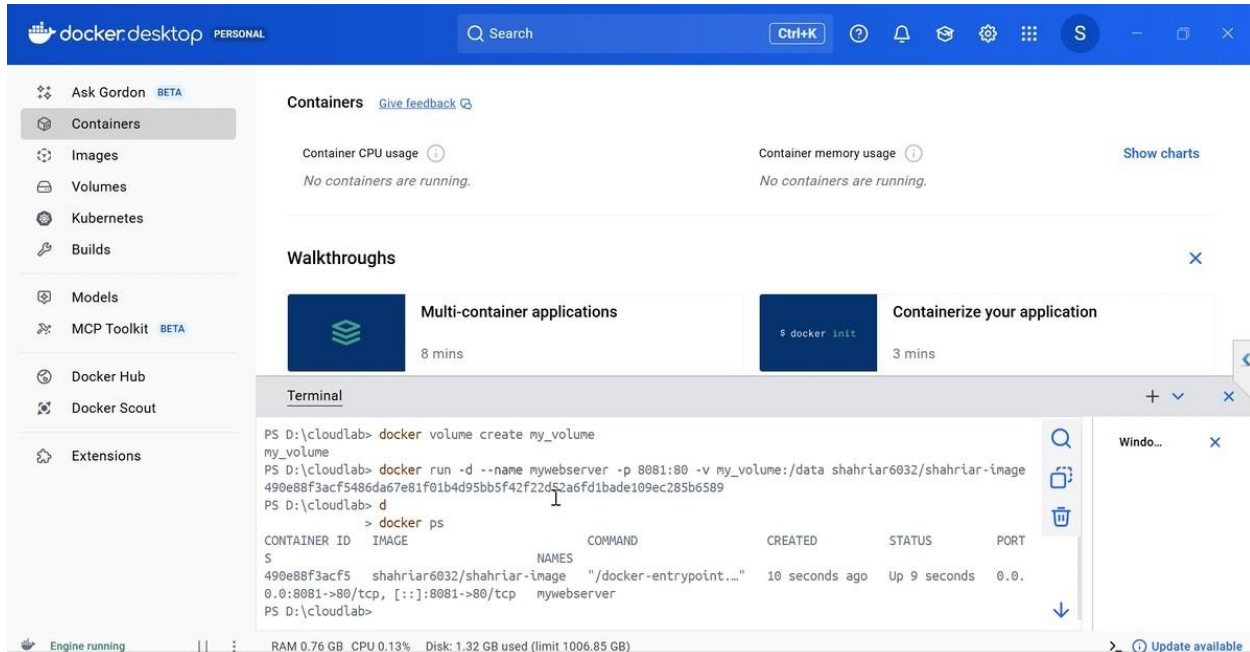
Figure-6



In containerizing solutions, images can be thought of as encapsulating a complete application and all required components. These are based on a Dockerfile and versioned so they can be deployed in a consistent way across your environments.

Once you have a successfully built image, it can be uploaded to a container registry like Docker Hub or your own private repository using `docker push`. And when you store images in a registry, it's simple to share and get applications reused across development, test and production. This technique achieves consistency that the applications work in the same way, no matter where they are deployed (configuration or behavior-wise) over its whole life-cycle.

Figure-7



4.2 Docker Volumes

Docker volumes are a way to persist data beyond the life cycle of containers. In plain english this means that any data written to a volume will not be wiped if the container is stopped, removed or even re-built. Volumes decouple application data from the container, preventing data loss and providing better, more predictable application behavior.

Volumes can be shared and reused between containers, while data in a container's writable layer is automatically deleted when the container is deleted, making volumes an appropriate location for higher-value data. Furthermore, backup and restore operations are easier since the data is in one known location. This is especially relevant for applications that require persistent or stateful data, like databases or content management systems.

Docker Volumes for durable and portable app data management. They enable data to be moved or reused between differing containers and environments without changes. Dynamic volumes are

particularly effective in cloud deployments where containers may be created and destroyed often, but where the underlying data must survive.

On this project we mounted a Docker volume to one container running an image from the remote Docker Hub. First, Docker Desktop fetched a copy of the image, then the container started with that volume. This is just an alternative example to show how the prebuilt images can be reused while the application data will persist between the runs using Docker volumes.

Figure-8

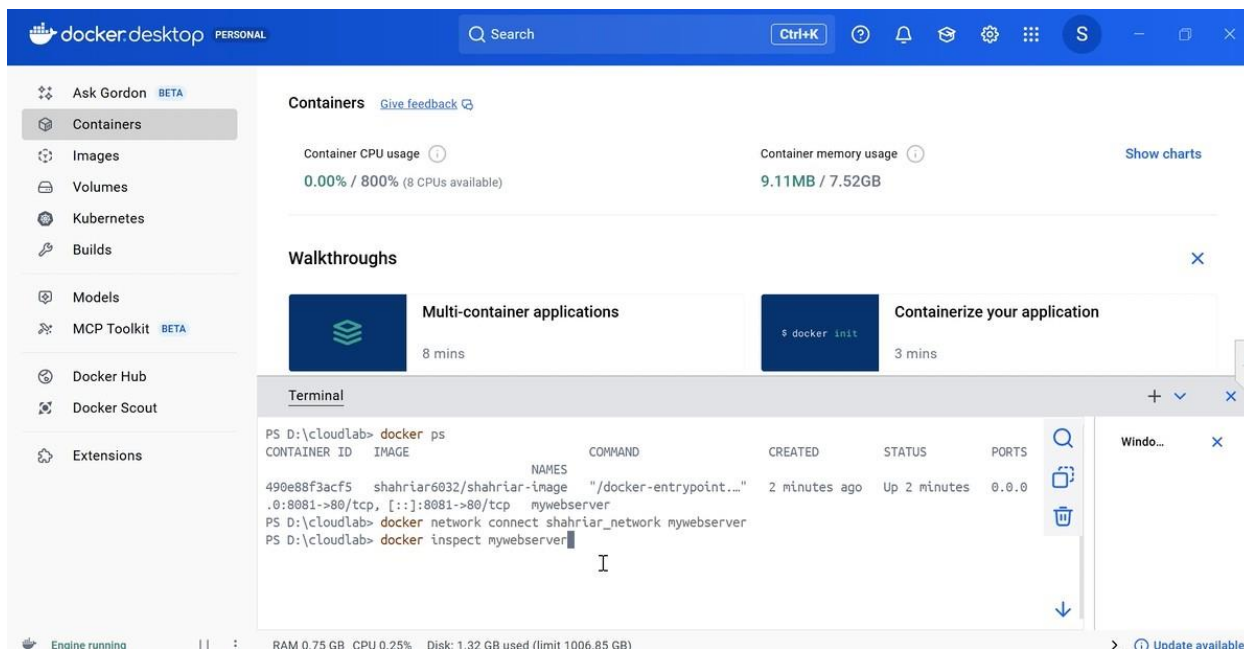
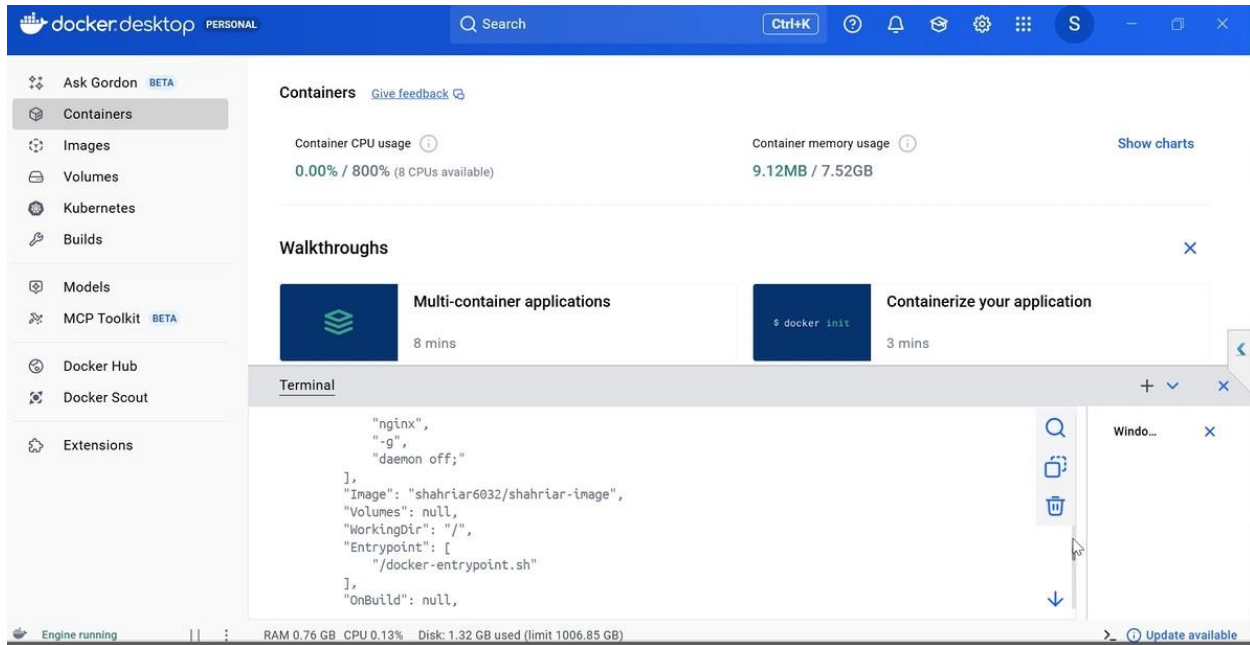


Figure-9



4.3 Container Networking

Network networking for traffic container and services where there is a need to have them communicate. By default, every container runs in a way that is isolated from all other containers — as well as the host system. Due to this isolation, networking is incredibly important when containers: have to communicate with each other, create microservices architectures, or expose applications to the end users.

Efficient container networking is essential to cloud features such as scaling, load balancing and secure communication. Network policies apply for restrictions of traffic flow between containers, so only explicitly allowed connections are possible. That makes services run more performantly and security mode of operations, especially when you have multiple services cohabiting together.

Container networking additionally offers robust and reliable connectivity for distributed applications to act as a single system. They are critical in service composition as they allow containers to find and connect to each other based on rules that reflect predictable network behavior. By doing this, it means that workloads interacting between containers are efficient and stable regardless of whether they're running in the cloud or on-premises.

The sample container was running with a custom Docker network in this project. Docker will assign an IP address to a container automatically when the default network mode is used, but using a custom bridge provides more control and customization options. For that, a separate network is first created and then the container is connected to it. This makes for a better network organisation,

easier inter-service communication and is in line with best practices when working with containers.

Figure- 10

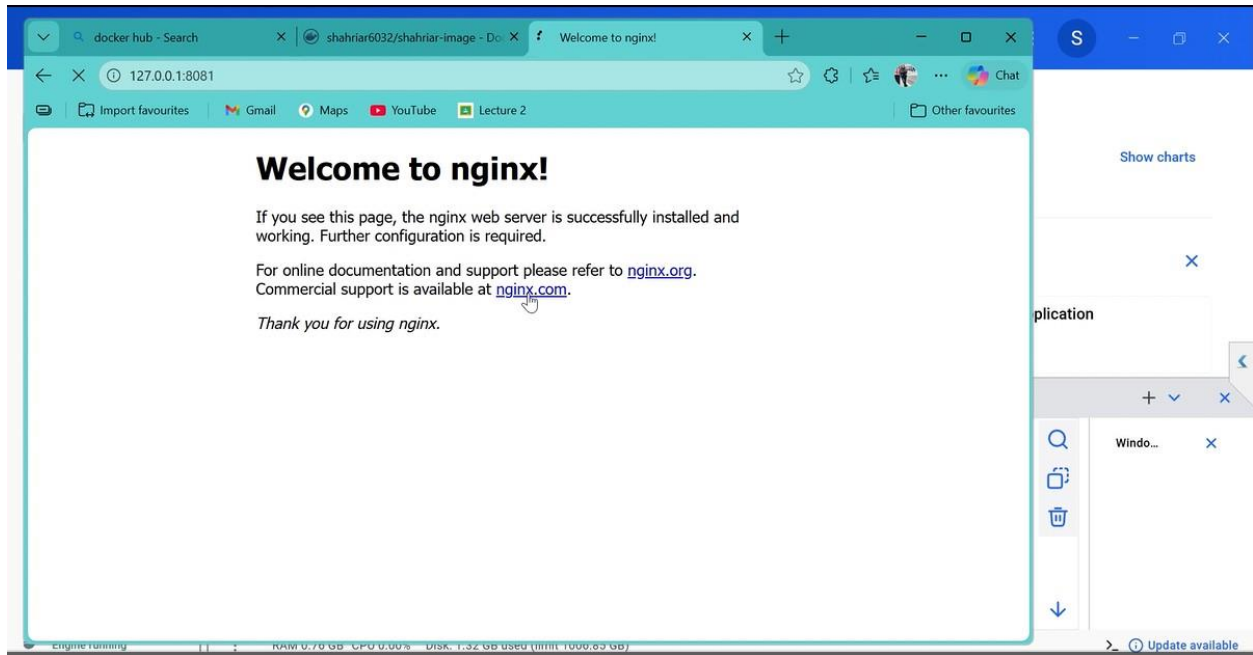
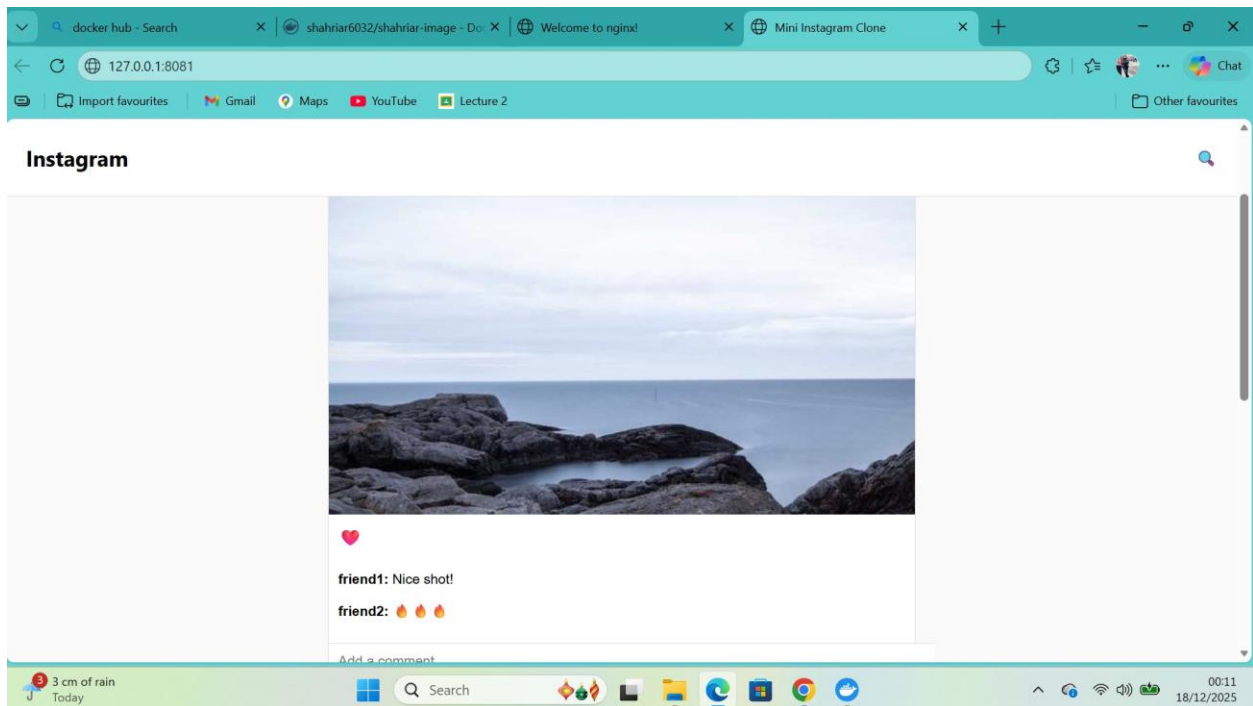


Figure-11



Finally, it presents the result of the deployment, where the sample container successfully displays the website through the local loopback address. The application is accessed using port **8081**, which was configured at the time the container was created. This confirms that the containerised web server is running correctly and that the port mapping between the host system and the container has been properly established.

5. Load Balancing, Scalability, and Application Gateways

Modern cloud applications must be capable of handling varying levels of user demand while maintaining performance and availability. Three core concepts enable this capability: load balancing, scalability, and application gateways.

5.1 Load Balancing

Load balancing makes use of different resources, either servers or instances, to distribute incoming network traffic. Amazon Web Services(AWS) and Microsoft Azure offer managed load balancing services. They automatically direct traffic towards healthy resources to avoid any single server from being overloaded and increase system flexibility [9].

5.2 Scalability

Scalability is the ability of a system to increase or decrease resources to meet demand. The platform of a cloud can provide both vertical scaling (adding more resources to a single instance) and horizontal scaling (adding more instances).Via auto-scaling mechanisms that monitor traffic and resource use in real time, capacity is dynamically adjusted [10]. This makes resources more cost-effective.

5.3 Application Gateways

Application gateways act as intelligent entry points for web traffic. They operate at the application layer and provide features such as SSL termination, URL-based routing, and web application firewall protection. By managing incoming traffic securely, application gateways improve both performance and security [11].

6. Relationship Between Load Balancing and Scalability

Load Balancing and Scalability Load balancing is related to scalability, it works well in combination with it. Load balancing distributes a flow to be processed across the resources, and scalability governs whether there is enough resource capacity for processing the flows.

As new instances are started by auto-scaling when demand raises they will be automatically added to the load balancer and included in traffic. Without scalability, load balancing alone is useless in managing heavy-loads. Comparison, scaling without load balancing can be implementation imbalanced. Together, these comprise the foundation of high-availability cloud applications [12].

7. Benefits of Combining Load Balancing, Scalability, and Application Gateways

When load balancing, scalability, and application gateways are combined, cloud services achieve a high level of reliability and performance. This integrated approach provides several benefits to customers:

- ✓ Improved availability through redundancy
- ✓ Enhanced security using application-layer protections
- ✓ Reduced latency for high thread-count systems during high-traffic.
- ✓ Smooth experience even when the traffic wobbles around

The composed form of these technologies is appealing as a format that enables cloud service providers to scale out responsive and secure services on demand [13].

8. Cloud Providers and Container-Based Applications

Cloud vendors are more regularly offering containers technology as part of their infrastructure. Managed services like Amazon Elastic Container Service (ECS), Azure Kubernetes Service (AKS) and Amazon Elastic Kubernetes Service (EKS) offer organizations the ability to deploy and manage containerised applications at scale.

These are platforms which automate the scheduling, networking and scaling of containers. Developers can now focus on application logic and avoid managing infrastructure. This is also in line with the wider objectives of cloud computing: flexibility, efficiency and fast innovation [14].

9. Summary of Key Concepts

This section has discussed how Docker containerization fits in with key concepts of cloud computing. Given the pros and cons of container networking, full deployment workflows, as well as Load Balancing, scalability and applications gateways – it seems that containerization is an indispensable part of a modern cloud architecture.

The next section will discuss the overall push underlining cloud adoption and present serverless computing as an alternative cloud service model.

10. Motivational Factors for Cloud Computing Growth

The rise of cloud computing The explosive growth in cloud computing over the past ten years has been due to a convergence of technical, economic and organisational factors. Legacy on-premises solutions are expensive to purchase, have long lead times, and consume large amounts of time to maintain. Cloud computing, on the other hand, provides a more responsive and cost efficient option that is much better suited to today's business demands.

10.1 Cost Efficiency

Cost savings is one of the most compelling reasons for cloud adoption. There are economies of scale for cloud providers, which operate at massive data centers. And so they can sell computing at a lower rate than pretty much anyone else. A pay-owhileusingsothatcontentinesanisation only pays for what they use, thus reducing the capital expenditure amount and the amount of capacity not used. [15] nomy pricing model Companies should be able to pay-as-they-go with a r company can many provider.

This model reduces the threshold for small- and medium-sized enterprises, which no longer require large investments in hardware. For bigger businesses, it enables IT spending to be more closely matched with actual demand.

10.2 Flexibility and Agility

Nowhere is this more evident than in cloud computing which offers a degree of agility other infrastructure cannot as readily achieve. Resources can be added and removed in minutes, allowing enterprises to quickly adapt to changing workloads. This flexibility fosters innovations that development teams can experiment, test, and deploy applications fast without the bureaucracy of approval or procurement chain [16].

Containerisation adds additional flexibility by maintaining the portability of applications, across environments and cloud providers.

10.3 Reliability and Global Reach

Cloud providers offer built-in redundancy, backup, and disaster recovery capabilities. Applications can be deployed across multiple regions, ensuring high availability and resilience against failures. This global reach allows organisations to serve users from geographically distributed locations with minimal latency [17].

11. Serverless Computing: Functions as a Service

Serverless computing is a new way to deploy web-based applications using the cloud. Also known as Function as a Service (FaaS) This service, Azure Functions and AWS Lambda are examples In a serverless model, developers write applications as a series of independent functions. These will be initiated by events such as an HTTP request from a browser to a web server, updates made in any type of database at all or files that are uploaded onto your site via FTP.

The cloud provider is responsible for all underlying infrastructure, including server provisioning, scaling, and maintenance. Developers only pay for the time their code is actually running rather than having to maintain continuously running servers. The last sentence of this paragraph comes as an example.

11.1 Benefits of Serverless Computing

Serverless computing offers several advantages:

- No server management required
- Automatic scaling based on demand
- Reduced operational overhead
- Cost efficiency for event-driven workloads

These characteristics make serverless functions particularly suitable for microservices, APIs, and background processing tasks.

12. Comparison of Serverless Computing and Containerisation

Although both serverless computing and containerisation support modern cloud application development, they address different use cases and requirements.

12.1 Containerisation

Containerisation can give a high level of control over an application environment. Developers explicitly can define dependencies, configurations at runtime and how networking behaves. This makes containers well suited long-running services, complex applications and microservices architectures that demand predictable performance [19].

The platform as a service however must always be managed and orchestrated by an external component - often Kubernetes. This places added complications and operational responsibility on those using the platform for applications; it adds risk at installation time because not every composer will make correct installation decisions, let alone perform them with security in mind.

12.2 Serverless Computing

Serverless computing abstracts away the entire infrastructure layer, so most of the operational burden disappears. This low entry point is particularly attractive to applications which have variable workloads or need event-driven execution patterns. Thanks to automatic scaling and

billing based on execution time (not CPU usage like with most cloud storage services), serverless solutions are very cost-effective for irregular workloads.

However, serverless computing is unable to provide the control over the execution environment and potential issues such as cold start latency.

12.3 Identifying the Driving Forces

Both approaches support the main driving forces behind cloud adoption:

- **Cost efficiency:** Reduced infrastructure costs and improved resource utilisation
- **Flexibility:** Rapid deployment and scaling
- **Operational simplicity:** Reduced management overhead

In practice, many cloud-based systems use a hybrid approach, combining containerised services with serverless functions to achieve the best balance between control and simplicity [20].

13. Challenges and Limitations of Containerisation in Cloud Environments

Benefits of containerisation in cloud computing However, it should be noted that along with advantages, containerisation also comes with a set of challenges and limitations. It is important for companies, that are planning to use container-based architecture appropriately and securely, to be aware of these problems.

One major challenge is security. Containers isolated from each other but share the host OS kernel. The common kernel model ensures that a vulnerability in the kernel could potentially affect all of the running containers. Unlike traditional virtual machines with stronger isolation provided by separate operating systems, containers need security measures such as image scanning, runtime monitoring and access control management to prevent possible risk. [21]

A further limitation is in terms of container orchestration complexity. These are some examples of the remaining inconsistencies in container management The environment where containers run: a Kubernetes deployment and 100 YouTube videos. Networking, scaling, service discovery and fault tolerance at this scale is best managed by orchestration platforms like Kubernetes. Despite

being powerful, the platforms come with a high learning curve and associated operational overhead, especially for small organisations having restricted expertise [22].

Storage management also poses problems of persistence. "Containers are stateless by nature - but most enterprise applications use at least a database and sometimes additional file storage. While professional storage solutions such as Docker volumes and cloud managed storage services solve this problem, they increase the architectural complexity and have to be designed in consideration of data consistency, performance and backup reliability [23].

In some cases, performance overhead can also be introduced. Containers are lighter comparably to VMs, but performance can be hurt by inappropriate resource limitations and bad image design. If these images are not well optimised, this will slow down the image startup times and waste resources. This reduces some of the efficiency benefits of containerisation [24].

Last but not least, vendor lock-in and ecosystem dependence may be an issue. Despite the portability of containers themselves, the major cloud vendors support their own container services that are embedded in their offerings. A move of a containerised workload between providers may still require reconfiguration, including for supporting services like networking and monitoring.

However, while these challenges are real, most limitations can be progressively mitigated through best practices, decent tooling and disciplined Architectural planning. When done right, containerization is indeed still an excellent way to deploy cloud applications that scale and remain stable.

14. Critical Discussion

Strategically, containers and serverless are two responses to the same problem that modern organizations encounter. Containerisation focuses on consistency, portability, and control and is a good fit for enterprise systems and cloud-native architectures. Serverless computing if focused on simplicity and scale, you want to build quickly and not worry about running your company like a system administrator.

The ongoing development of cloud computing is expected to be through their coexistence rather than the prevalence of one over the other. As the cloud providers improve their platforms, organisations will move to a hybrid architectures that take advantage of both.

#Dockerhub Demonstrate video link: <https://youtu.be/fYzcFCDr27g>

14. Conclusion

This report focuses on practical demonstration of the importance containerisation in cloud computing with examples using Docker and similar software. Starting with the concept of abstraction through containers, it explained how many drawbacks of traditional virtual machines are avoided altogether by containerisation. Real examples have clarified Docker image production, usage, volume, networking, and the publishing process. (Docker Image Creation, Container Execution, Volumes, Networking, Image Publishing have all been demonstrated in practise.

Cloud computing subjects load balancing, scalability and application gateways, among other things, along with how they all come together to ensure the delivery of consistently reliable but secure services. Finally, we looked at why people adopt the cloud in terms of forces, drawing a comparison between containerization and serverless computing.

After all, containerization is still an indispensable part of good modern cloud systems. In combination with other cloud-native technologies, it provides what businesses need to handle both current requirements and those visible over the horizon future. This places us just where our journey began: back at the container level.

Reference:

- [1] P. Mell and T. Grance, “The NIST Definition of Cloud Computing,” NIST Special Publication 800-145, 2011.
- [2] R. Buyya et al., *Cloud Computing: Principles and Paradigms*, Wiley, 2011.
- [3] D. Merkel, “Docker: Lightweight Linux Containers for Consistent Development and Deployment,” *Linux Journal*, vol. 2014, no. 239, 2014.

- [7] N. Dragoni et al., “Microservices: Yesterday, Today, and Tomorrow,” *Present and Ulterior Software Engineering*, Springer, 2017.
- [8] J. Turnbull, *The Docker Book*, James Turnbull, 2022.
- [9] Amazon Web Services, “Elastic Load Balancing,” 2024.
- [10] Microsoft Azure, “Autoscale Overview,” 2024.
- [11] Microsoft Azure, “Application Gateway Documentation,” 2024.
- [12] T. Erl, *Cloud Computing: Concepts, Technology and Architecture*, Prentice Hall, 2013.
- [13] A. Tanenbaum and M. Van Steen, *Distributed Systems*, Pearson, 2017.
- [14] Kubernetes Documentation, “Production-Grade Container Orchestration,” 2024.
- [15] M. Armbrust et al., “A View of Cloud Computing,” *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.
- [16] E. Evans, *Domain-Driven Design*, Addison-Wesley, 2016.
- [17] Google Cloud, “Global Infrastructure Overview,” 2024.
- [18] J. Spillner and S. Mateos, “Function as a Service,” *IEEE Internet Computing*, vol. 21, no. 2, pp. 92–95, 2017.
- [19] B. Burns, *Kubernetes: Up and Running*, O’Reilly Media, 2023.
- [20] C. Jonas et al., “Cloud Programming Simplified: A Berkeley View on Serverless Computing,” University of California, Berkeley, 2019.
- [21] S. Pahl, “Containerization and the PaaS Cloud,” *IEEE Cloud Computing*, vol. 2, no. 3, pp. 24–31, 2015.
- [22] K. Hightower, B. Burns, and J. Beda, *Kubernetes: Up and Running*, O’Reilly Media, 2023.
- [23] Docker Inc., “Docker Volumes Documentation,” 2024.
- [24] Red Hat, “Container Performance Considerations,” 2023.

