

**Cloud Engineering Principles and Practice: An
Analytical Study of Virtualisation, Containerisation,
and Docker-Based Application Deployment**

Student Name:	MD Monsuruzzaman
Student ID:	2504658
Programme:	MSc Cybersecurity and Digital Forensics
Institution:	University of Wales Trinity Saint David London
Module:	Cloud Engineering Principles and Practice
Module Code:	ACCA7027
Lecturer:	Renuga Jayakumar
Submission Date:	28 April 2026

Table of Contents

Abstract	3
1. Introduction	4
2. Principles of Virtualisation	5
3. Containerisation and Docker Implementation	7
3.1 Introduction to Containerisation	7
3.2 Docker Architecture	7
3.3 Dockerfile Implementation	9
3.4 Building Docker Image	12
3.5 Running Docker Container	13
3.6 Container Output	14
3.7 Container Management	15
3.8 Publishing Image to Docker Hub	16
3.9 Container Volumes	17
3.10 Container Networking	17
3.11 Critical Evaluation	18
3.12 Security in Containerisation	19
3.13 Container Orchestration Kubernetes	20
4. Cloud Computing Concepts	20
5. Driving Forces and Serverless Computing	23
6. Video Demonstration	25
7. Future Trends in Cloud Computing	25
7. Conclusion	26
8. References	27

Abstract

Cloud computing now plays a central role in modern IT systems by allowing organisations to access resources over the internet instead of relying on local infrastructure [1]. This shift has significantly improved flexibility, scalability, and cost efficiency in IT operations. This report explores key concepts in cloud engineering, focusing on virtualisation, containerisation, and practical implementation using Docker. A practical implementation using Docker is presented to demonstrate how applications can be containerised, deployed, and managed effectively in a cloud environment. The implementation includes creating a Dockerfile, building an image, running containers, managing volumes, configuring networking, and publishing images to Docker Hub. In addition, this report examines key cloud computing concepts, including load balancing, scalability, and application gateways, supported by practical examples from major cloud providers such as Amazon Web Services (AWS) and Microsoft Azure [2], [6], [7]. The relationship between these technologies is critically evaluated to demonstrate how they contribute to system performance, reliability, and security. Furthermore, the report explores the main driving forces behind the growth of cloud computing, including cost efficiency and flexibility, and examines serverless computing as a modern alternative to containerisation. A comparison between these approaches highlights their advantages, limitations, and appropriate use cases.

Overall, the findings demonstrate that containerisation plays a critical role in modern cloud environments by improving deployment efficiency, portability, and scalability, while serverless computing provides an alternative model focused on automation and reduced operational overhead. Additionally, Modern organisations increasingly rely on cloud-based systems to improve scalability, flexibility, and operational efficiency [1]. With the increasing demand for high-performance applications, containerisation technologies such as Docker provide efficient solutions for deploying applications consistently across multiple environments. This report not only focuses on theoretical concepts but also demonstrates practical implementation using Docker, highlighting real-world applicability.

This report provides both theoretical analysis and practical implementation to evaluate modern cloud engineering practices. It demonstrates how containerisation enhances deployment efficiency, portability, and scalability, while serverless computing offers an alternative model focused on automation and reduced operational overhead. The findings highlight the importance of selecting appropriate cloud architectures based on performance, cost, and security requirements.

1. Introduction

In recent years, organisations have shifted from traditional infrastructure to cloud-based solutions, enabling faster deployment and improved scalability. Instead of relying on physical infrastructure, organisations can now access computing resources over the internet on demand [1]. This model enables organisations to rapidly scale their resources, lower operational expenses, and enhance overall service availability. Cloud environments are typically categorised into three deployment models: public, private, and hybrid clouds, each providing varying degrees of control and flexibility. Public cloud platforms, such as Amazon Web Services (AWS) and Microsoft Azure, offer shared infrastructure that can be accessed by multiple users, delivering high scalability and cost efficiency [2]. In contrast, private cloud environments are designed for a single organisation, ensuring greater control over resources and improved security. Hybrid cloud solutions combine both public and private approaches, allowing organisations to achieve a balance between flexibility and security requirements [4].

In addition to deployment models, cloud computing is structured into service models including Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). These models define different levels of control and responsibility between the cloud provider and the user. This report focuses on the technical foundations of cloud engineering, particularly virtualisation and containerisation. A practical Docker-based implementation is used to demonstrate how applications can be efficiently deployed in a cloud environment. Furthermore, the report evaluates essential cloud concepts such as load balancing, scalability, and application gateways, and analyses modern approaches such as serverless computing. Over the past decade, cloud computing has evolved from a simple storage solution into a comprehensive platform for application development and deployment. Organisations such as Amazon, Netflix, and Google rely heavily on cloud infrastructure to deliver scalable and highly available services to millions of users worldwide [1]. One of the key advantages of cloud computing is its ability to provide on-demand resources. This allows organisations to avoid large upfront investments in hardware and instead adopt a pay-as-you-go model. This flexibility is particularly beneficial for startups and small businesses.

Another important aspect is high availability. Cloud providers distribute applications across multiple data centres, ensuring that services remain operational even in the event of hardware failure. This level of reliability is difficult to achieve in traditional on-premises systems. Cloud computing is also a key enabler of digital transformation across industries such as healthcare, finance, and e-commerce. Organisations increasingly rely on cloud-native technologies to deliver scalable and resilient services. For example, platforms such as Netflix utilise distributed cloud infrastructure to stream content globally with minimal latency.

Furthermore, the rise of DevOps practices has increased the importance of automation and continuous deployment, where containerisation plays a central role. Technologies such as Docker allow developers to package applications and deploy them consistently across multiple environments. This reduces system errors and improves software delivery speed. This report not only explores theoretical concepts but also reflects on practical implementation, providing a

comprehensive understanding of modern cloud engineering practices. In addition, cloud computing supports innovation by enabling rapid development and deployment of applications. Developers can use cloud platforms to experiment, test, and deploy applications without requiring significant infrastructure investment. This accelerates time-to-market and encourages innovation in industries such as fintech, healthcare, and artificial intelligence. Cloud computing also supports remote collaboration and global accessibility [1]. Teams can access systems and data from anywhere in the world, improving productivity and enabling modern work environments. This became particularly important during global events such as the COVID-19 pandemic, where remote working became essential.

Furthermore, cloud platforms provide built-in security, monitoring, and compliance tools, allowing organisations to maintain high levels of data protection and regulatory compliance. These capabilities make cloud computing a critical component of modern digital infrastructure. In this project, I implemented a Docker-based web application, which helped me understand how containerisation simplifies deployment and improves scalability in real-world environments.

2. Principles of Virtualisation

Virtualisation allows a single physical machine to run multiple environments [5]. This is achieved using a hypervisor, which manages and allocates hardware resources among multiple virtual environments [10].

Hypervisors are generally divided into two types: those that run directly on hardware and those that operate on top of an operating system [5].

Virtualisation provides several benefits:

- ❖ Efficient resource utilisation
- ❖ Isolation between applications
- ❖ Improved scalability
- ❖ Reduced hardware costs

For example, in enterprise environments, a single physical server can host multiple VMs running different operating systems and applications. This significantly reduces infrastructure costs while improving system flexibility.

However, virtualisation has limitations. Each virtual machine requires a full operating system, which increases memory and storage usage. This results in slower performance and higher resource consumption compared to lightweight alternatives. Virtualisation can also be categorised into server virtualisation, storage virtualisation, and network virtualisation. Server virtualisation enables multiple virtual machines to run on a single physical server, improving hardware utilisation. Storage virtualisation combines multiple storage devices into a single logical system, improving data management and accessibility. Network virtualisation allows the creation of

virtual networks independent of physical infrastructure, improving flexibility and security. From a cloud provider perspective, virtualisation is essential for multi-tenancy, where multiple customers share the same physical resources securely. Hypervisors ensure isolation between different virtual machines, preventing interference between users.

However, virtualisation introduces performance overhead due to the need for multiple operating systems. This increases memory usage and startup time. In contrast, containerisation eliminates this overhead by sharing the host operating system, making it more efficient for cloud-native applications.

Therefore, while virtualisation provides strong isolation and flexibility, it is increasingly complemented or replaced by containerisation in modern cloud environments.

Table 1: Comparison between Virtual Machines and Containers

Feature	Virtual Machines	Containers
OS	Full OS per VM	Shared OS
Size	Large (GBs)	Small (MBs)
Startup Time	Slow	Fast
Performance	Lower	Higher
Resource Usage	High	Low

The comparison demonstrates that virtual machines offer stronger isolation, while containers focus more on performance and efficiency [3], [10].

Virtualisation can also be classified into several categories, including server virtualisation, storage virtualisation, and network virtualisation [10]. Server virtualisation enables multiple virtual machines to run on a single physical server, leading to more efficient use of resources and reduced hardware costs. Storage virtualisation integrates several physical storage units into one logical system, making data management easier and more flexible. Similarly, network virtualisation allows the creation of independent virtual networks that are not tied directly to physical infrastructure, improving both control and scalability. From the perspective of cloud providers, virtualisation is essential for supporting multi-tenancy, where multiple users can securely share the same physical resources. This is achieved through the use of hypervisors, which maintain isolation between virtual machines and prevent interference. Such isolation is particularly critical in public cloud environments, where infrastructure is shared among different customers.

However, despite its advantages, virtualisation introduces certain limitations. Each virtual machine requires its own operating system, which increases memory consumption and storage

requirements. This results in slower startup times and reduced performance compared to lightweight solutions such as containers. Additionally, managing large numbers of virtual machines can become complex and resource-intensive. From a critical perspective, virtualisation is essential for traditional cloud infrastructure but is gradually being complemented by containerisation technologies. Containers provide a more efficient alternative by reducing overhead and improving deployment speed. Therefore, modern cloud architectures often combine both virtualisation and containerisation to achieve optimal performance and scalability.

According to the National Institute of Standards and Technology (NIST), virtualisation is a key enabler of cloud computing, supporting essential characteristics such as resource pooling and rapid elasticity [5]. However, recent industry trends indicate a shift towards container-based systems due to their lightweight and flexible nature. This evolution highlights the importance of understanding both technologies in modern cloud engineering [3], [10].

3. Containerisation and Docker Implementation

3.1 Introduction to Containerisation

Containerisation involves packaging applications along with all their required dependencies, ensuring they can run consistently across different environments. In contrast to virtual machines, containers share the host operating system, which makes them more lightweight, efficient, and quicker to deploy [13].

Docker is one of the most commonly used platforms for managing containers due to its simplicity and efficiency [3].

3.2 Docker Architecture

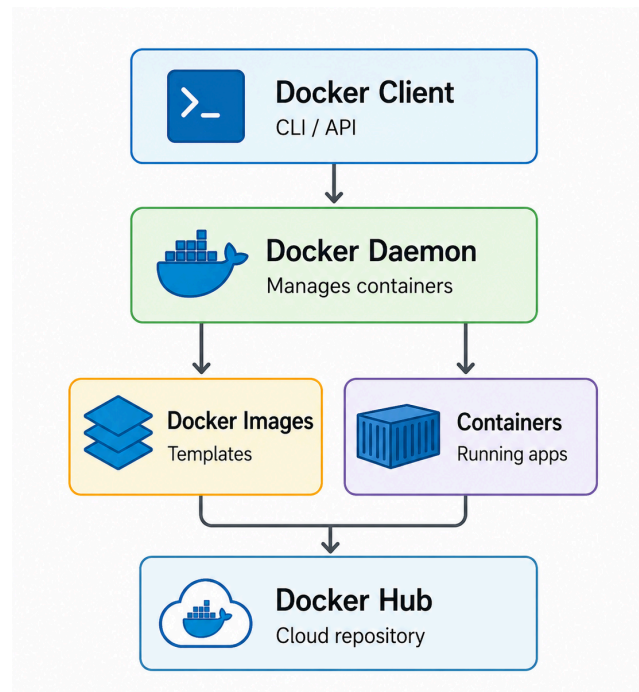
Docker architecture follows a client-server model that supports the efficient development, deployment, and management of containerised applications. It is composed of several core components, each contributing to different stages of the container lifecycle [3].

The Docker Client acts as the main interface through which users interact with the Docker platform. It enables the execution of commands such as building images, running containers, and managing resources using command-line tools (e.g., `docker build` and `docker run`). The client communicates directly with the Docker Daemon to carry out these operations. The Docker Daemon serves as the central component responsible for managing Docker objects, including images, containers, networks, and volumes. It processes requests received from the Docker Client and performs tasks such as building images, running containers, and managing their lifecycle [3].

Docker Images function as read-only templates used to create containers. They include all necessary components required to run an application, such as the code, runtime, libraries, and

dependencies. Images are typically created using Dockerfiles and can be reused across multiple environments, ensuring consistency and reliability in application deployment.

Figure 1: illustrates how the Docker client communicates with the Docker daemon to manage images and containers, while Docker Hub is used as a remote repository.



Docker Containers are running instances of Docker images [3]. They provide an isolated environment where applications can execute independently of the host system. Containers are lightweight, fast to start, and share the host operating system kernel, making them more efficient than traditional virtual machines. Docker Hub is a cloud-based repository that allows users to store, share, and distribute Docker images. It supports version control and collaboration, enabling developers to publish their images and access pre-built images created by others. This facilitates faster development and deployment processes [3].

In modern cloud environments, Docker is often integrated with orchestration tools such as Kubernetes, which automate container deployment, scaling, networking, and availability. This integration enhances system reliability and supports large-scale applications. Overall, Docker architecture provides a flexible and efficient framework for containerised application development, enabling organisations to achieve portability, scalability, and consistency across different environments.

3.3 Dockerfile Implementation

A Dockerfile contains a set of instructions used to automatically build a container image. It defines the environment, dependencies, and commands required to run an application within a container. Dockerfiles play a crucial role in ensuring consistency and reproducibility across different environments [3]. The Dockerfile was used to create a simple web application using an Nginx web server. The following Dockerfile was implemented:

```
FROM nginx:latest
```

```
COPY index.html /usr/share/nginx/html/index.html
```

The FROM command defines the base image that the container will use. This simplifies the setup process and reduces the need for manual configuration.

The "COPY" instruction copies the local index.html file from the project directory into the container's file system, specifically into the default Nginx web directory (/usr/share/nginx/html/). This ensures that when the container runs, the web server will display the custom HTML page created for this assignment.

After creating the Dockerfile, the image can be built using the following command:

```
docker build -t my-web-app .
```

The "docker build" command reads the Dockerfile and creates an image based on the defined instructions. The "-t" flag assigns a name (tag) to the image, making it easier to identify and manage. The "." indicates that the current directory is used as the build context [3].

From a practical perspective, Dockerfiles provide a standardised method for packaging applications, ensuring that they run consistently regardless of the deployment environment. This eliminates the common issue of "it works on my machine" and improves collaboration among developers. However, Dockerfiles must be written carefully to avoid inefficiencies and security risks. For example, using unnecessarily large base images can increase image size and slow down deployment. Therefore, best practices such as using lightweight images and minimising layers should be followed. Overall, Dockerfiles are a fundamental component of containerisation, enabling automated, consistent, and efficient application deployment in cloud environments.

Figure 2: *Dockerfile configuration for Nginx web server*

The Dockerfile used in this project is as follows: The Docker workflow demonstrates how a Dockerfile is used to build an image, which is then executed as a container and accessed through a web browser.

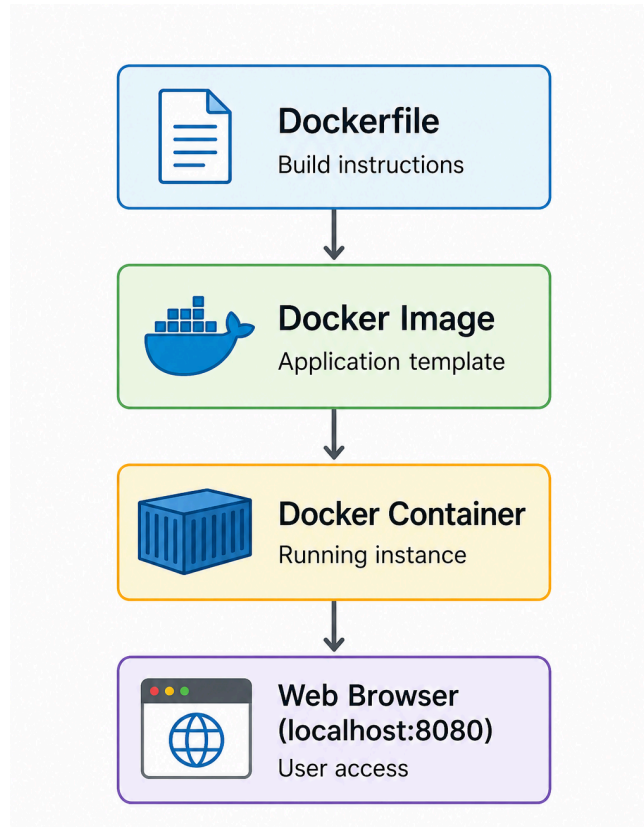
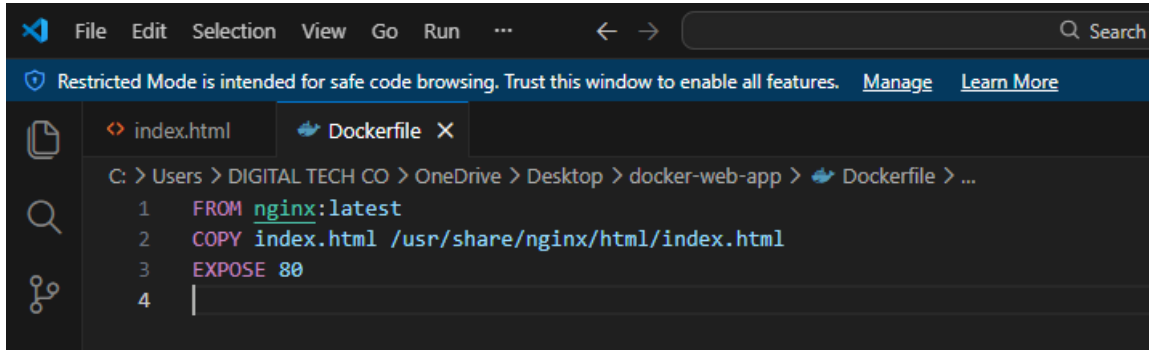
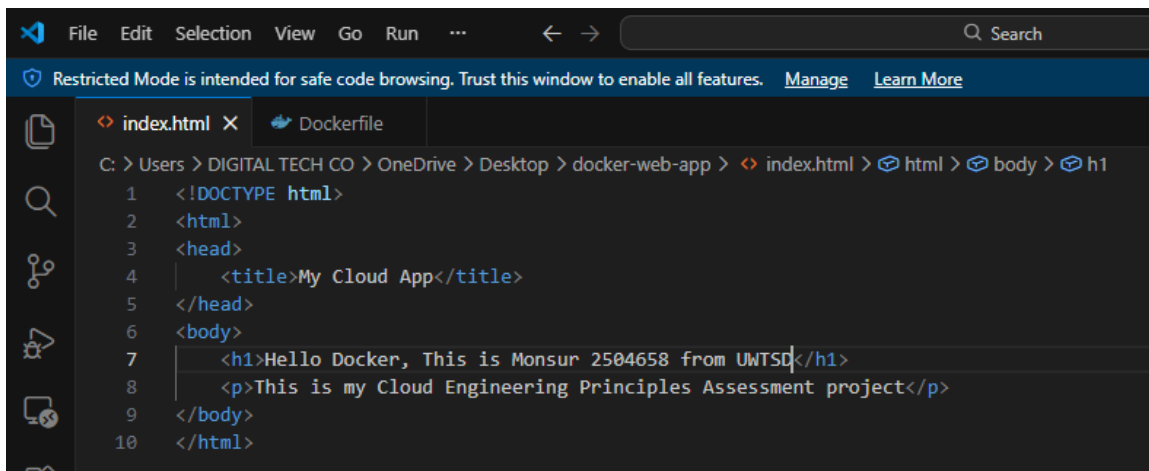


Figure 3: Dockerfile used to build the Nginx web container



```
1 FROM nginx:latest
2 COPY index.html /usr/share/nginx/html/index.html
3 EXPOSE 80
4 |
```

Figure 4: index.html file displaying content for the containerised web application



```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>My Cloud App</title>
5 </head>
6 <body>
7   <h1>Hello Docker, This is Monsur 2504658 from UWTSD</h1>
8   <p>This is my Cloud Engineering Principles Assessment project</p>
9 </body>
10 </html>
```

The implementation is illustrated in Figure 3 and 4.

3.4 Building Docker Image

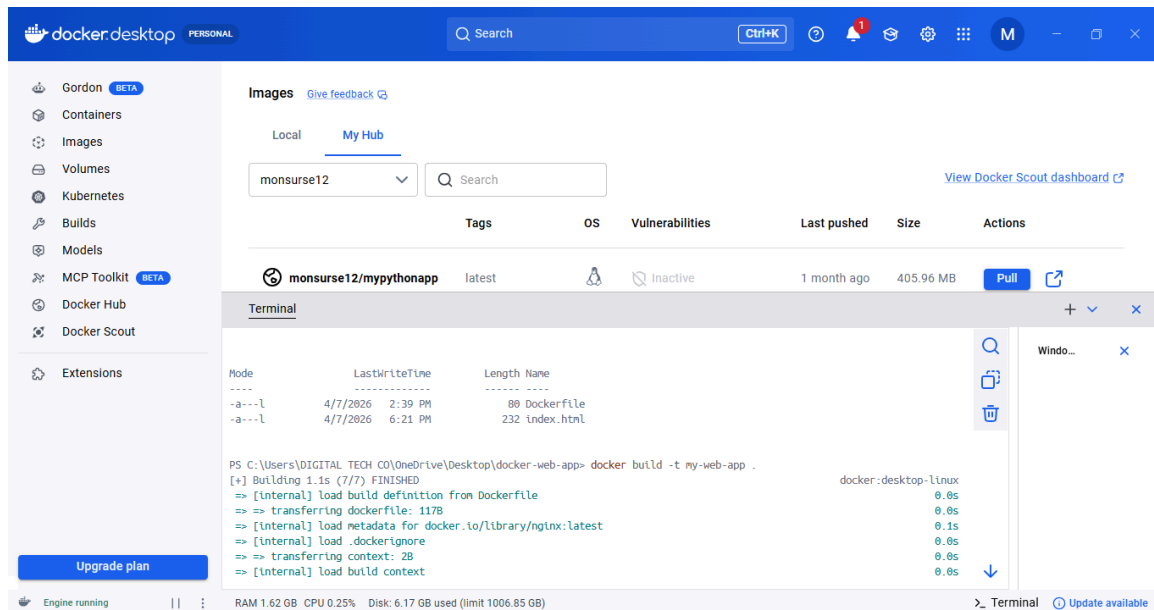
The following command was used to build the Docker image:

```
docker build -t my-web-app .
```

In this command, the "-t" option assigns a name (tag) to the image, and the dot (.) represents the current directory where the Dockerfile is located. From my practical experience, this step is important because it converts the application into a reusable image that can be deployed anywhere. This command creates an image named *my-web-app* based on the Dockerfile [3].

Figure 5: Docker image build process using Docker CLI

The process of building the Docker image is illustrated in Figure 2.



3.5 Running Docker Container

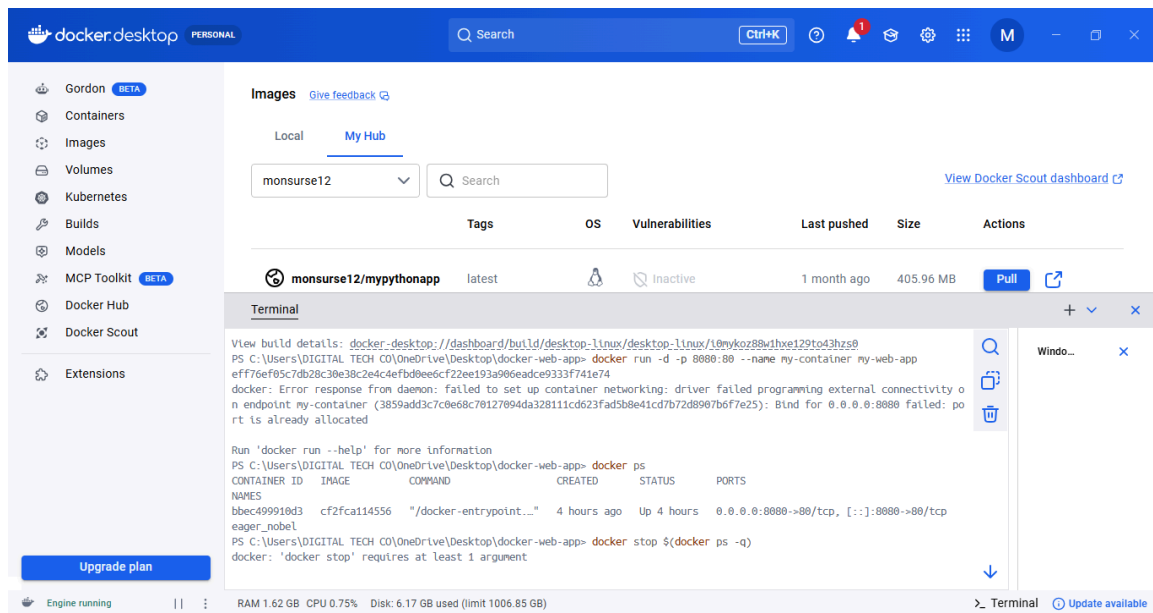
The container was executed using:

```
docker run -d -p 8080:80 --name my-container my-web-app
```

This command runs the container in detached mode (-d), maps port 8080 of the host machine to port 80 of the container, and assigns a name to the container. I used this to access my application through the browser using localhost [3].

Figure 6: Running Docker container with port mapping

The container execution with port mapping is shown in Figure 3.



3.6 Container Output

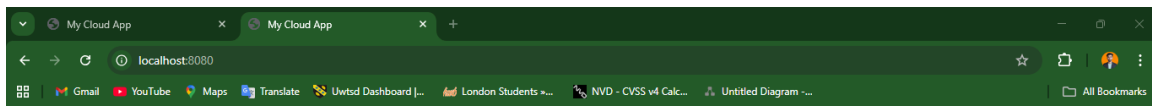
The application was accessed through a web browser using:

<http://localhost:8080>

This demonstrates how containerised applications can be deployed locally and accessed through a browser.

Figure 7: Output of the containerised web application in browser

The output of the containerised application is presented in Figure 4.



Hello Docker, This is Monsur 2504658 from UWTSD

This is my Cloud Engineering Principles Assessment project

3.7 Container Management

Running containers were checked using:

```
docker ps
```

Containers can be stopped using:

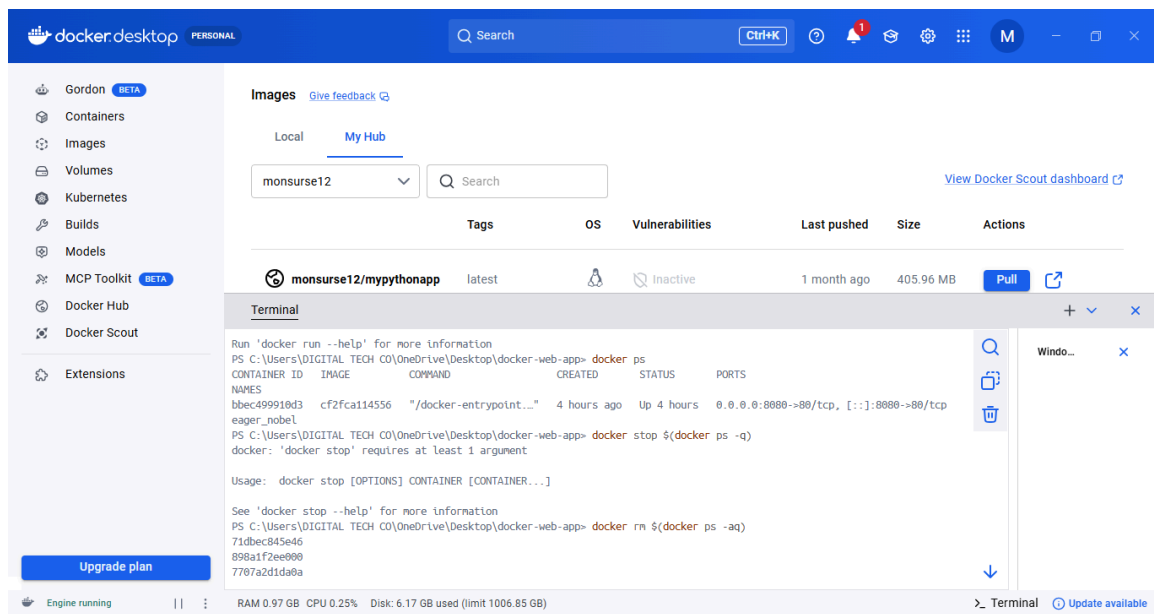
```
docker stop <container_id>
```

Like (docker stop bbec499910d3)

These commands allow monitoring and controlling containers.

Figure 8: Listing running containers using docker ps

The running containers can be verified using the command shown in Figure 5.



3.8 Publishing Image to Docker Hub

To publish the image to Docker Hub, I used the following commands:

```
docker login
```

```
docker tag my-web-app monsurse12/my-web-app
```

```
docker push monsurse12/my-web-app
```

These steps allowed me to upload my image to Docker Hub, making it accessible from anywhere. This demonstrates how cloud-based image repositories support application deployment and sharing [3].

Figure 9: Docker image successfully pushed to Docker Hub

The Docker image was successfully pushed to Docker Hub as shown in Figure 6 and Figure 7.

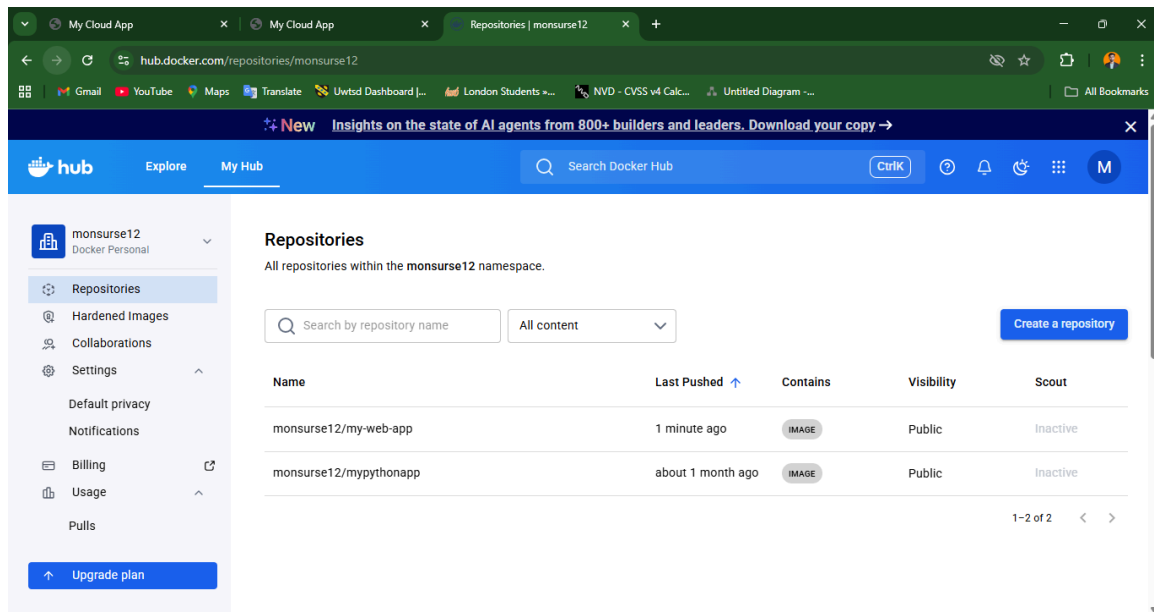
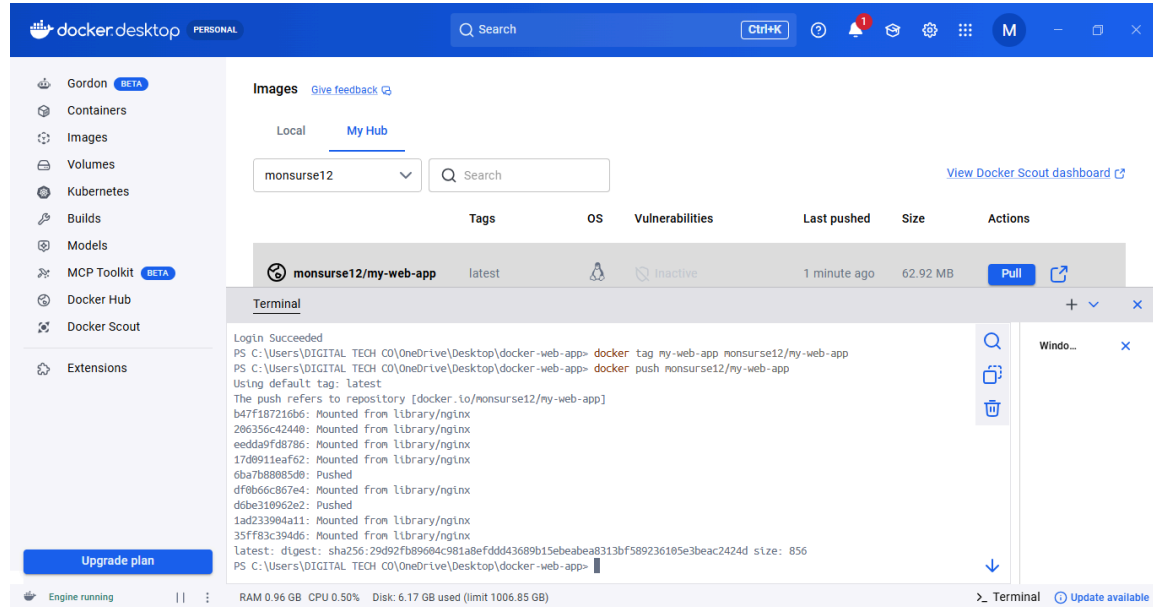


Figure 10: Docker image pushing process to Docker Hub



3.9 Container Volumes

Docker volumes are used to manage persistent data generated by containers. Unlike the default container storage, which is temporary, volumes ensure that data remains available even if a container is stopped or removed.

In Docker, a volume can be created using the following command:

```
docker volume create my-volume
```

Once created, the volume can be mounted to a container as shown below:

```
docker run -d -p 8080:80 -v my-volume:/usr/share/nginx/html nginx
```

This approach ensures that data within the container is stored persistently outside the container lifecycle. Volumes are especially important in real-world applications such as databases, where maintaining data integrity across container restarts is essential [3].

3.10 Container Networking

Docker networking enables communication between containers and external systems. By default, Docker provides a bridge network, but custom networks can be created for better control and security.

A custom network can be created using:

```
docker network create my-network
```

Containers can then be connected to this network:

```
docker run -d --network my-network --name web-container my-web-app
```

This approach enables containers to communicate with each other using container names rather than relying on IP addresses. Container networking is a key requirement for microservices-based architectures in cloud environments, as it supports efficient and flexible service communication [3].

3.11 Critical Evaluation

Container-based deployment has significantly improved how applications are developed and managed in cloud environments. Compared to traditional virtualisation, containers are lightweight because they share the host operating system kernel, which reduces resource consumption and improves startup time. This makes containerisation highly suitable for dynamic cloud environments where rapid deployment is essential [10], [16].

From my experience during this project, I observed that practical implementation helped bridge the gap between theoretical knowledge and real-world application. This ensures consistency across development, testing, and production environments, effectively eliminating configuration-related issues. Modern development practices, including continuous integration and deployment, benefit greatly from container-based systems [3]. However, despite these advantages, containerisation also introduces several challenges. One of the key concerns is security. Since containers share the host operating system, vulnerabilities in the host kernel can affect all running containers. This creates potential risks such as container escape and unauthorised access, which must be mitigated through proper isolation and security mechanisms [3].

Another limitation is the complexity of managing containers at scale. While Docker is effective for running individual containers, large-scale applications require orchestration tools such as Kubernetes to manage deployment, scaling, and networking. This introduces additional complexity, requiring specialised knowledge and infrastructure management. Furthermore, container networking and storage management can become challenging in distributed environments. Misconfiguration of networks or volumes may lead to performance issues or data loss. Therefore, organisations must implement robust monitoring and management tools to maintain system reliability. From a critical perspective, containerisation should not be viewed as a complete replacement for virtualisation but rather as a complementary technology. Virtual machines provide stronger isolation and are still preferred in scenarios requiring high security and complete system separation. In contrast, containers offer efficiency and flexibility, making them ideal for cloud-native applications.

Overall, containerisation plays a significant role in enhancing application deployment and scalability within cloud environments. However, its effectiveness depends on proper security practices, the use of orchestration tools, and efficient system management. Therefore, organisations need to carefully assess their requirements and adopt a well-balanced approach

when integrating containerisation into their cloud architecture. This evaluation highlights the importance of selecting appropriate cloud technologies based on application requirements, balancing performance, scalability, security, and cost considerations. In addition to the observed benefits, it is important to evaluate containerisation in comparison with other deployment models such as virtual machines and serverless computing. Containers provide a balance between performance and control. Unlike virtual machines, containers do not require a full operating system, which significantly reduces resource consumption and improves startup time. This makes containers highly efficient for microservices-based architectures where rapid deployment is required [10].

From a performance perspective, containers outperform virtual machines due to their lightweight nature. However, serverless computing can offer even greater efficiency for short-lived tasks because it eliminates the need for continuously running instances. Despite this advantage, serverless platforms introduce latency issues known as cold starts, which can negatively impact performance in time-sensitive applications [9]. In real-world cloud environments, organisations often adopt a hybrid approach by combining containers and serverless technologies. For example, companies such as Netflix use containerised microservices for core application logic while leveraging serverless functions for background processing and event-driven tasks. This approach maximises efficiency while maintaining flexibility and control. Furthermore, cost considerations play a significant role in selecting the appropriate deployment model. Containers require continuous resource allocation, whereas serverless computing charges based on execution time. As a result, serverless is more cost-effective for intermittent workloads, while containers are more suitable for long-running services.

Therefore, containerisation should be viewed as part of a broader cloud strategy rather than a standalone solution. Organisations must evaluate performance, cost, scalability, and security requirements before selecting the most appropriate technology. This highlights the importance of architectural decision-making in modern cloud engineering. From a strategic perspective, organisations must also consider operational and financial trade-offs when adopting containerisation. While containers improve deployment efficiency, they require investment in orchestration tools, monitoring systems, and skilled personnel. Failure to manage these factors effectively can lead to increased complexity and operational overhead, reducing the overall benefits of container-based architectures.

3.12 Security in Containerisation

Security remains an important challenge in container-based systems, particularly due to shared operating system components. Unlike virtual machines, where each instance operates independently with its own operating system, containers rely on the same underlying OS, which can introduce potential vulnerabilities if not properly managed [10]. One of the primary security risks in containerisation is container escape, where a malicious process gains access to the host system. This can compromise other containers and the entire infrastructure. To reduce these risks, organisations apply security mechanisms such as namespace isolation, control groups (cgroups), and mandatory access control frameworks including SELinux and AppArmor [3]. Another important aspect is image security. Docker images must be carefully managed to ensure they do

not contain vulnerabilities or malicious code. Best practices include using trusted base images, regularly scanning images for vulnerabilities, and applying security patches. Tools such as Docker Scout and third-party vulnerability scanners are commonly used in industry to enhance image security. Network security is also essential in container environments [14]. Containers communicate through networks, which must be properly configured to prevent unauthorized access. Techniques such as network segmentation and firewall rules are used to restrict communication between containers. From a critical perspective, while containerisation improves deployment efficiency, it also introduces new security challenges that must be addressed through proper configuration and monitoring. Organisations must adopt a DevSecOps approach, integrating security practices into the development lifecycle to ensure secure container deployment [3]. Therefore, security plays a vital role in containerised environments, and failure to implement proper controls can lead to significant risks in cloud-based systems [15].

3.13 Container Orchestration Kubernetes

In modern cloud environments, managing a large number of containers manually becomes complex and inefficient. Container orchestration tools such as Kubernetes are used to automate deployment, scaling, and management of containerised applications [8]. Kubernetes is an open-source platform that enables organisations to manage container clusters efficiently.

One of the key features of Kubernetes is automatic scaling. It can increase or decrease the number of containers based on system demand, ensuring optimal resource utilisation. This is particularly important in cloud environments where workloads can change dynamically. Kubernetes also provides self-healing capabilities, automatically restarting failed containers and replacing unhealthy instances [8].

Another important feature is service discovery and load balancing. Kubernetes assigns each container a unique IP address and DNS name, allowing containers to communicate easily. It also distributes traffic across containers, ensuring high availability and performance. From a practical perspective, while Docker is used to create and run containers, Kubernetes is used to manage them at scale. In enterprise environments, organisations rarely use Docker alone; instead, they integrate it with orchestration tools to handle complex applications [13].

However, Kubernetes introduces additional complexity. It requires knowledge of cluster management, networking, and security configurations. Despite this, it is widely adopted in industry due to its ability to manage large-scale applications efficiently. Therefore, container orchestration is a critical component of modern cloud architecture, enabling organisations to deploy scalable and resilient systems.

4. Cloud Computing Concepts

Cloud computing environments rely on key architectural concepts such as load balancing, scalability, and application gateways to ensure system performance, reliability, and security.

These components are fundamental in designing modern cloud-based applications and are widely implemented by cloud providers such as AWS and Microsoft Azure [6], [7].

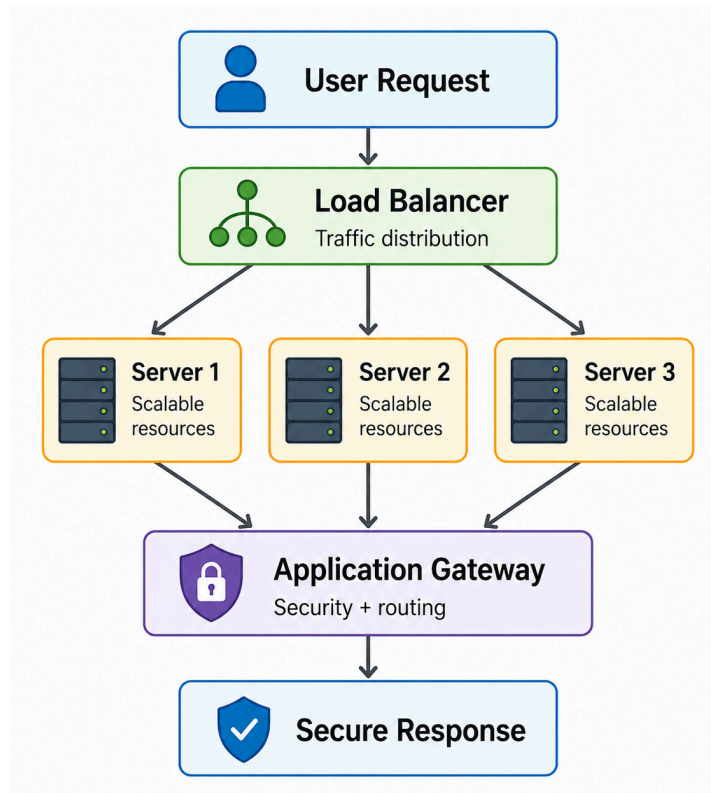
Load balancing spreads user requests across multiple servers to maintain system stability and avoid overload [6]. This ensures high availability and improves system performance by efficiently utilising available resources. Cloud providers offer managed load balancing services, such as AWS Elastic Load Balancing (ELB) and Azure Load Balancer, which automatically distribute traffic based on predefined rules [6]. From a critical perspective, ineffective load balancing can result in uneven traffic distribution, leading to server overload, increased latency, and potential service outages. This is particularly problematic in high-demand scenarios such as e-commerce platforms during peak sales events. Therefore, organisations must carefully configure load balancing algorithms and continuously monitor performance to ensure optimal resource utilisation.

Scalability refers to the ability of a system to handle increasing or decreasing workloads by adjusting resources accordingly. This can be achieved through vertical scaling, which increases the capacity of a single server, or horizontal scaling, which adds additional servers to distribute the workload. In cloud environments, horizontal scaling is more commonly used due to its flexibility and ability to improve fault tolerance. Scalability enables systems to increase or decrease resources depending on demand [6]. However, scalability introduces challenges related to cost management and system complexity. Over-scaling can result in significant cost inefficiencies, particularly in cloud environments where pricing is based on resource consumption. For example, unnecessary scaling of computer instances can increase operational expenses without improving performance. Therefore, organisations must carefully balance scalability and cost optimisation through effective monitoring and auto-scaling strategies.

Application gateways operate at the application layer and deliver advanced traffic management alongside enhanced security features. These capabilities include SSL termination, URL-based routing, and Web Application Firewall (WAF) protection, which help defend applications against common threats such as SQL injection and cross-site scripting attacks [7]. For instance, Azure Application Gateway combines both load balancing and security functions, making it an important component in modern cloud infrastructures. Despite their advantages, application gateways require careful configuration and maintenance. Misconfigured security rules or routing policies can expose applications to vulnerabilities or disrupt legitimate traffic. Therefore, proper security policies, monitoring, and regular updates are essential to ensure both performance and protection.

These technologies load balancing, scalability, and application gateways work together to build highly resilient systems. Load balancing distributes incoming traffic efficiently, while scalability ensures that sufficient resources are available to meet changing demand. At the same time, application gateways provide secure routing and protect applications from external threats. In practical scenarios such as an e-commerce platform, load balancers distribute user requests across multiple servers to prevent overload, scalability manages increased traffic during peak periods like sales events, and application gateways ensure secure communication and protection against malicious activity.

Figure 11: As shown in cloud architecture uses load balancing to distribute traffic across multiple servers, while application gateways provide security and routing functions.



In addition, cloud systems are designed with fault tolerance in mind. This involves deploying applications across multiple availability zones or regions to ensure continuous operation even in the event of hardware or network failures. If one instance fails, traffic is automatically redirected to another, ensuring uninterrupted service and high availability. From a critical perspective, although load balancing, scalability, and application gateways significantly improve system performance and reliability, they also introduce architectural complexity. Misconfiguration of load balancers or scaling policies can lead to uneven traffic distribution, increased latency, or higher operational costs. Similarly, improperly configured application gateways may expose systems to security vulnerabilities. Therefore, organisations must implement proper monitoring tools, automation strategies, and security controls to manage these technologies effectively. In real-world scenarios, organisations such as Amazon and Netflix rely heavily on these technologies to maintain service availability at a global scale. For instance, Netflix uses distributed cloud infrastructure combined with load balancing and auto-scaling to deliver content to millions of users without interruption. This demonstrates the importance of integrating these technologies to achieve high performance, scalability, and reliability. Furthermore, a comparison between cloud providers highlights differences in implementation. AWS provides highly flexible

and globally scalable services such as Elastic Load Balancing, while Microsoft Azure emphasises integration with enterprise systems through services like Application Gateway and Azure Load Balancer [6], [7]. Despite these differences, both platforms offer robust solutions for managing traffic, scaling resources, and securing applications.

Overall, the effective combination of load balancing, scalability, and application gateways is essential for building modern cloud applications. While these technologies offer significant benefits, their successful implementation requires careful design, continuous monitoring, and proper configuration to ensure optimal performance and security.

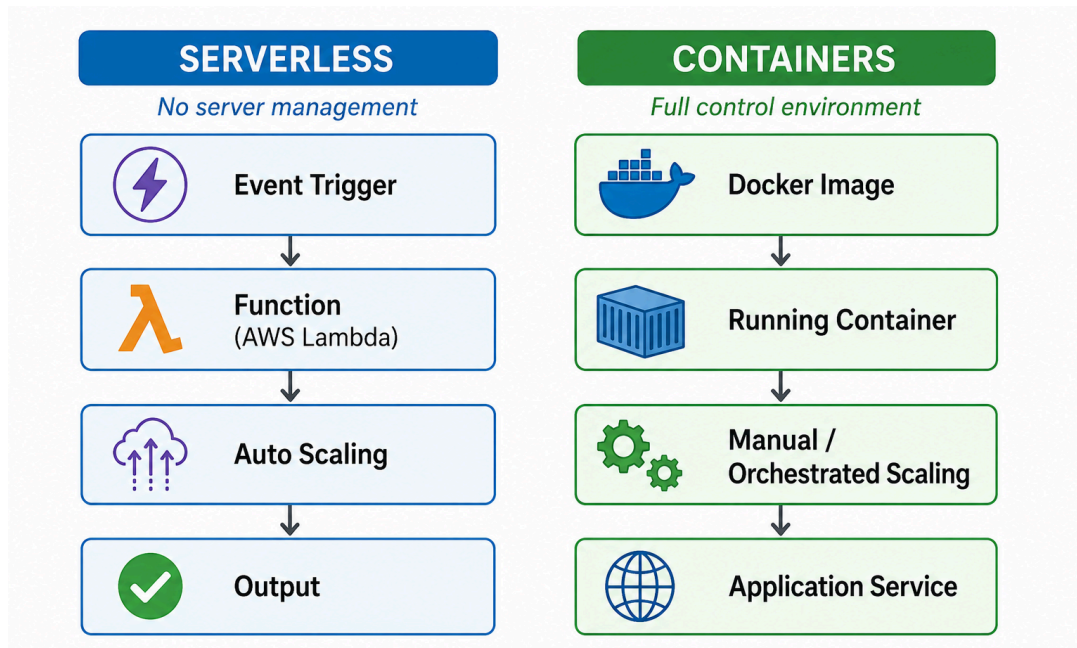
5. Driving Forces and Serverless Computing

Cloud computing growth is driven by cost efficiency, flexibility, global accessibility, and disaster recovery capabilities. The pay-as-you-go model reduces infrastructure costs and supports dynamic scaling, allowing organisations to optimise resource usage based on demand. Platforms such as AWS Lambda and Azure Functions automatically scale resources and charge based on actual execution, making them highly efficient for event-driven applications [9], [17].

Serverless computing introduces a paradigm shift by removing the need to manage infrastructure. Developers can focus solely on writing code while the cloud provider handles resource allocation, scaling, and execution. This approach is particularly beneficial for applications such as APIs, real-time data processing, and automation workflows. However, serverless computing has several limitations. Cold start latency can affect performance, especially in applications requiring immediate response times. Additionally, execution time limits and reduced control over the runtime environment make serverless unsuitable for complex or long-running applications. Vendor lock-in is another critical concern, as applications may become dependent on specific cloud provider services, reducing portability. In contrast, containerisation provides greater control and flexibility. Containers allow applications to run continuously and support custom configurations, making them suitable for microservices architectures and enterprise systems. Unlike serverless, containers can be deployed across different cloud providers, reducing dependency on a single platform. From a critical perspective, serverless and containerisation should be considered complementary rather than competing technologies. Many modern architectures combine both approaches to achieve optimal performance and efficiency. For example, serverless functions can be used for lightweight event-driven tasks, while containers handle core application services [9], [12]. From a critical viewpoint, serverless computing is not a universal solution. While it reduces infrastructure management, it limits control over the execution environment and introduces dependency on specific cloud providers. This can lead to vendor lock-in, making migration between platforms complex and costly. Therefore, organisations must carefully evaluate workload requirements before adopting serverless architectures.

Therefore, the choice between serverless and containerisation depends on workload characteristics, performance requirements, and cost considerations. A hybrid approach often provides the most effective solution in modern cloud environments.

Figure 12: compares serverless and container-based architectures, highlighting differences in execution, scaling, and infrastructure management.



The following table presents a comparison between containerisation and serverless computing based on key technical and operational characteristics, including control, scalability, and cost structure. This comparison helps to evaluate how each approach aligns with different application requirements in cloud environments.

Table 2: Comparison with Containers

Feature	Serverless	Containers
Control	Low	Moderate-High
Scalability	Fully Automatic	Manual/Auto
Cost	Pay-per-use	Resource-based

As illustrated in the table, containerisation provides a higher level of control over the runtime environment, allowing developers to configure dependencies and manage long-running services effectively. This makes containers particularly suitable for complex and enterprise-level applications. In contrast, serverless computing offers fully automated scalability and a pay-per-use cost model, which is highly beneficial for event-driven and short-duration workloads. However, this convenience comes at the cost of reduced control and potential limitations such as execution time restrictions and vendor dependency. This comparison demonstrates that neither approach is universally superior. Instead, the choice depends on specific system requirements, including performance expectations, operational complexity, and cost considerations. In modern cloud architectures, organisations often use a hybrid approach by combining containerisation for core services with serverless computing for lightweight, event-driven tasks, allowing them to achieve greater efficiency and flexibility.

6. Video Demonstration

A video demonstration has been prepared to present the practical use of Docker, including building the image, running the container, and pushing it to Docker Hub.

Video Link : https://youtu.be/ndH_EWieLxQ

7. Future Trends in Cloud Computing

Cloud computing continues to evolve rapidly with advancements in artificial intelligence, machine learning, and edge computing. Edge computing allows data processing closer to users, reducing latency and improving performance for real-time applications [1], [8]. Another key trend is the increasing use of container orchestration platforms such as Kubernetes, which automate application deployment and management [8]. This improves system reliability and scalability in large-scale environments. In addition, cloud providers are integrating advanced security features, including zero-trust architectures and automated threat detection systems. These improvements help organisations protect their applications and data more effectively [1].

However, challenges such as vendor lock-in, data privacy concerns, and system complexity remain significant issues. Organisations must carefully design their cloud strategies to address these challenges. Overall, cloud computing is expected to become more intelligent, automated, and distributed, increasing its importance in modern IT infrastructure.

8. Conclusion

In conclusion, this report has explored the fundamental principles of cloud engineering, with a particular focus on virtualisation and containerisation technologies [3]. Virtualisation provides the foundation for efficient resource utilisation by enabling multiple virtual machines to operate on shared physical hardware, while containerisation offers a more lightweight and flexible approach for deploying modern applications. The practical implementation using Docker demonstrated how applications can be built, deployed, and managed using container-based technologies. The practical implementation demonstrated how Docker simplifies deployment processes and ensures consistency across different environments.

In addition, this report examined key cloud computing concepts, including load balancing, scalability, and application gateways. Together, these technologies contribute to high availability, improved performance, and enhanced security within cloud-based systems. The discussion also illustrated how cloud providers such as AWS and Microsoft Azure implement these approaches to deliver reliable and scalable services. The comparison between containerisation and serverless computing provided deeper insight into modern cloud architectures. While serverless computing reduces infrastructure management and supports event-driven applications, containerisation offers greater control and flexibility for complex and long-running applications. In practice, both approaches are often used together to achieve optimal efficiency. Looking ahead, cloud computing is expected to continue evolving with advancements in artificial intelligence, edge computing, and automation technologies. Container orchestration tools such as Kubernetes will play a key role in managing large-scale applications. From this assessment, I have developed both theoretical knowledge and practical skills in cloud engineering, which are essential for modern IT and DevOps roles.

Overall, cloud engineering principles are critical for designing scalable, secure, and efficient systems, making them a fundamental component of today's digital infrastructure. From a strategic perspective, the successful adoption of cloud technologies depends not only on technical implementation but also on effective architectural design, cost management, and security considerations.

9. References

- [1] Amazon Web Services, “What is Cloud Computing?” Available: <https://aws.amazon.com/what-is-cloud-computing/>
- [2] Microsoft Azure, “Cloud Computing Overview,” Available: <https://learn.microsoft.com/en-us/azure/architecture/>
- [3] Docker Inc., “Docker Documentation,” Available: <https://docs.docker.com/>
- [4] M. Armbrust et al., “A View of Cloud Computing,” Communications of the ACM, 2010. Available: <https://dl.acm.org/doi/10.1145/1721654.1721672>
- [5] National Institute of Standards and Technology (NIST), “The NIST Definition of Cloud Computing,” Available: <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf>
- [6] Amazon Web Services, “Elastic Load Balancing,” Available: <https://docs.aws.amazon.com/elasticloadbalancing/>
- [7] Microsoft Azure, “Application Gateway Documentation,” Available: <https://learn.microsoft.com/en-us/azure/application-gateway/>
- [8] Kubernetes, “Overview of Kubernetes,” Available: <https://kubernetes.io/docs/concepts/overview/>
- [9] Google Cloud, “Serverless Computing,” Available: <https://cloud.google.com/serverless>
- [10] IBM, “Containers vs Virtual Machines,” Available: <https://www.ibm.com/cloud/learn/containers-vs-vm>
- [11] Red Hat, “What is Containerization?” Available: <https://www.redhat.com/en/topics/containers>
- [12] Amazon Web Services, “What is Serverless?” Available: <https://aws.amazon.com/serverless/>
- [13] R. Pahl, “Containerization and the PaaS Cloud,” IEEE Cloud Computing, 2015.
- [14] P. Zhang et al., “Understanding Container Security,” IEEE, 2018.
- [15] D. Merkel, “Docker: Lightweight Linux Containers for Consistent Development and Deployment,” Linux Journal, 2014.
- [16] M. Villamizar et al., “Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud,” IEEE, 2015.

[17] P. Jamshidi et al., “Microservices: The Journey So Far and Challenges Ahead,” IEEE Software, 2018.