

CLOUD ENGINEERING PRINCIPLES AND PRACTICE REPORT

Title Page

Module: Cloud Engineering Principles and Practice (ACCA7027)

Assignment: Cloud Engineering Report

Student Name: Himanshu Jigneshbhai Sompura

Student ID: 2517096

Table of Contents

1.0 Introduction	
2.0 Cloud Engineering with Docker	
2.1 Virtualisation Principles	
2.2 Containerisation and Docker	
2.3 Dockerfile Implementation	
2.4 Images and DockerHub	
2.5 Container Volumes	
2.6 Container Networking	
2.7 Full Deployment Workflow Example	
3.0 Cloud Computing Concepts	
3.1 Load Balancing	
3.2 Scalability	
3.3 Application Gateways	
3.4 Relationship Between Load Balancing and Scaling	
3.5 Combined Benefits	
4.0 Driving Forces of Cloud Computing	
4.1 Cost Efficiency	
4.2 Flexibility and Agility	
4.3 Serverless Computing	
4.4 Serverless vs Containerisation	
5.0 Conclusion	
References	

1.0 Introduction

Cloud computing has evolved into a dominant paradigm in modern information systems, fundamentally reshaping how organisations design, deploy, and manage applications. By offering on-demand access to scalable computing resources, cloud platforms enable significant reductions in capital expenditure while enhancing operational agility. However, despite these advantages, cloud computing also introduces challenges related to resource management, performance optimisation, and system complexity, particularly as applications scale across distributed environments [1].

A core enabling technology behind cloud computing is virtualisation, which abstracts physical hardware to support multiple virtual machines on a single host. While virtualisation has historically improved resource utilisation and isolation, it is increasingly criticised for its inherent overhead, including duplicated operating systems and slower provisioning times. These limitations have driven the adoption of containerisation as a more efficient alternative. Container technologies such as Docker provide lightweight, portable environments that share the host operating system kernel, thereby improving performance and reducing resource consumption [3]. Nevertheless, containerisation is not without its drawbacks, including security concerns and increased orchestration complexity when deployed at scale.

This report critically evaluates the transition from traditional virtualisation to container-based architectures, with a focus on Docker as a key enabler of cloud-native applications. It examines both theoretical principles and practical implementations, including Dockerfiles, image management, networking, and persistent storage. Furthermore, the report analyses essential cloud computing concepts such as load balancing, scalability, and application gateways, not only describing their functionality but also evaluating their interdependencies and limitations in real-world scenarios. Finally, it explores the primary drivers behind the rapid expansion of cloud computing, including cost efficiency and flexibility, while critically comparing containerisation with emerging serverless computing models. Through this analysis, the report aims to provide a comprehensive and balanced understanding of modern cloud engineering practices and their implications for scalable system design.

2.0 Cloud Engineering with Docker

Cloud engineering has increasingly shifted towards the adoption of container-based architectures, with Docker emerging as a dominant platform for developing and deploying cloud-native applications. While traditional infrastructure relied heavily on virtual machines, modern cloud environments demand faster deployment, improved scalability, and greater resource efficiency—requirements that containerisation is well-positioned to address [3].

Docker enables the encapsulation of applications and their dependencies into lightweight, portable containers that can run consistently across different environments. This significantly reduces the “it works on my machine” problem and enhances the reliability of deployment pipelines. However, despite these advantages, Docker-based environments introduce new complexities, particularly in areas such as orchestration, networking, and security. For example, while containers are efficient, they share the host operating system kernel, which can increase the attack surface if not properly managed.

From a cloud engineering perspective, Docker plays a critical role in enabling microservices architectures, continuous integration/continuous deployment (CI/CD) pipelines, and scalable distributed systems. Its integration with cloud platforms such as AWS, Azure, and Google Cloud further reinforces its importance in modern infrastructure design. Nevertheless, effective use of Docker requires a clear understanding of its underlying principles, including image creation, container lifecycle management, networking, and storage.

This section critically explores the principles of virtualisation and containerisation, followed by practical implementations using Docker. It evaluates key components such as Dockerfiles, image repositories, container volumes, and networking, while demonstrating how these elements work together in a complete deployment workflow. Through this analysis, the section highlights both the strengths and limitations of Docker in real-world cloud engineering scenarios.

2.1 Virtualisation Principles

Virtualisation is a foundational technology in cloud computing that enables the abstraction of physical hardware resources into multiple virtual environments. This is typically achieved through the use of a hypervisor, which acts as an intermediary layer between the physical hardware and the virtual machines (VMs). Hypervisors can be classified into two types: Type 1 (bare-metal), such as VMware ESXi and Microsoft Hyper-V, which run directly on hardware, and

Type 2 (hosted), such as VirtualBox, which run on top of an existing operating system [2].

The primary objective of virtualisation is to maximise hardware utilisation by allowing multiple operating systems and applications to run concurrently on a single physical server. This approach reduces capital expenditure and improves resource efficiency, making it a critical enabler of early cloud infrastructure. Additionally, virtualisation provides strong isolation between virtual machines, enhancing security and fault tolerance, as failures in one VM do not directly affect others.

However, despite these advantages, virtualisation introduces several limitations. Each virtual machine requires a full operating system, leading to increased resource consumption in terms of memory, storage, and CPU usage. This overhead can reduce performance, particularly in environments where rapid scaling and high efficiency are required. Furthermore, the boot time for virtual machines is significantly slower compared to containers, which can hinder dynamic scaling in cloud environments.

From a cloud engineering perspective, these limitations have become more pronounced with the rise of microservices and distributed systems. Modern applications often require rapid deployment, high scalability, and efficient resource utilisation—areas where traditional virtualisation can struggle. As a result, containerisation has emerged as a more lightweight alternative, addressing many of the inefficiencies associated with virtual machines.

Nevertheless, virtualisation remains highly relevant in cloud computing, particularly in scenarios requiring strong isolation, legacy system support, or the ability to run multiple operating systems on a single host. Cloud providers such as AWS and Azure continue to rely on virtualisation at the infrastructure level, even when offering container-based services to users. This highlights that, rather than being replaced, virtualisation and containerisation coexist, each serving distinct roles within modern cloud architectures.

2.2 Containerisation and Docker

Containerisation represents a significant evolution in the way applications are developed and deployed within cloud environments. Unlike traditional virtualisation, which relies on hypervisors to run multiple virtual machines with separate operating systems, containerisation operates at the operating system level, allowing multiple applications to share the same kernel while remaining isolated from one another [3]. This fundamental difference results in a more lightweight and efficient approach to resource utilisation.

Docker has become the leading platform for containerisation, providing a standardised framework for building, packaging, and deploying applications in containers. By encapsulating an application along with its dependencies, libraries, and configuration files into a single unit, Docker ensures consistency across development, testing, and production environments. This directly addresses a long-standing issue in software development, often referred to as the “it works on my machine” problem, by eliminating environmental inconsistencies.

One of the key advantages of Docker is its efficiency. Containers are significantly smaller in size compared to virtual machines and can be started almost instantly, enabling rapid scaling and deployment. This makes Docker particularly well-suited for microservices architectures, where applications are broken down into smaller, independent services that can be deployed and scaled individually. Furthermore, Docker integrates seamlessly with continuous integration and continuous deployment (CI/CD) pipelines, supporting automated software delivery in modern DevOps practices.

However, despite its advantages, containerisation introduces several challenges that must be critically considered. Because containers share the host operating system kernel, they offer weaker isolation compared to virtual machines, which may lead to security vulnerabilities if not properly managed. Additionally, managing large numbers of containers in production

environments requires orchestration tools such as Kubernetes, which adds complexity to system design and maintenance. Networking and persistent storage in containerised environments can also be more complex compared to traditional approaches.

From a cloud engineering perspective, Docker plays a central role in enabling scalable and portable cloud-native applications. Cloud providers such as AWS (Elastic Container Service and EKS) and Azure (Azure Kubernetes Service) have integrated Docker-based containerisation into their platforms, reflecting its importance in modern infrastructure. Nevertheless, the decision to use containers over virtual machines should be based on specific application requirements, as each approach presents trade-offs between performance, security, and management complexity.

In summary, containerisation through Docker offers a highly efficient and flexible alternative to traditional virtualisation, particularly for modern, distributed applications. However, its successful implementation requires careful consideration of its limitations, particularly in terms of security and orchestration, reinforcing the need for a balanced and informed approach in cloud engineering design.

2.3 Dockerfile Implementation

A Dockerfile is a text-based configuration file that defines the instructions required to build a Docker image. Docker's official documentation explains that Docker builds images automatically by reading commands from a Dockerfile, with common instructions including FROM, COPY, RUN, EXPOSE, CMD, and ENTRYPOINT [4]. This makes the Dockerfile a central component of containerisation because it turns application deployment into a repeatable and version-controlled process.

For the practical implementation, a simple web server can be created using the official Nginx image. The purpose of this example is to demonstrate how a basic HTML page can be

packaged into a Docker image and then executed as a container.

```
FROM nginx:latest
```

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

```
EXPOSE 80
```

The FROM instruction specifies the base image used to build the container. In this case, nginx:latest is used because Nginx is a lightweight and widely used web server. The COPY instruction copies the local index.html file into the default Nginx web directory inside the container. Finally, EXPOSE 80 documents that the container is intended to listen on port 80, which is the standard HTTP port.

The image can then be built using the following command:

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

After the image has been created, it can be run as a container:

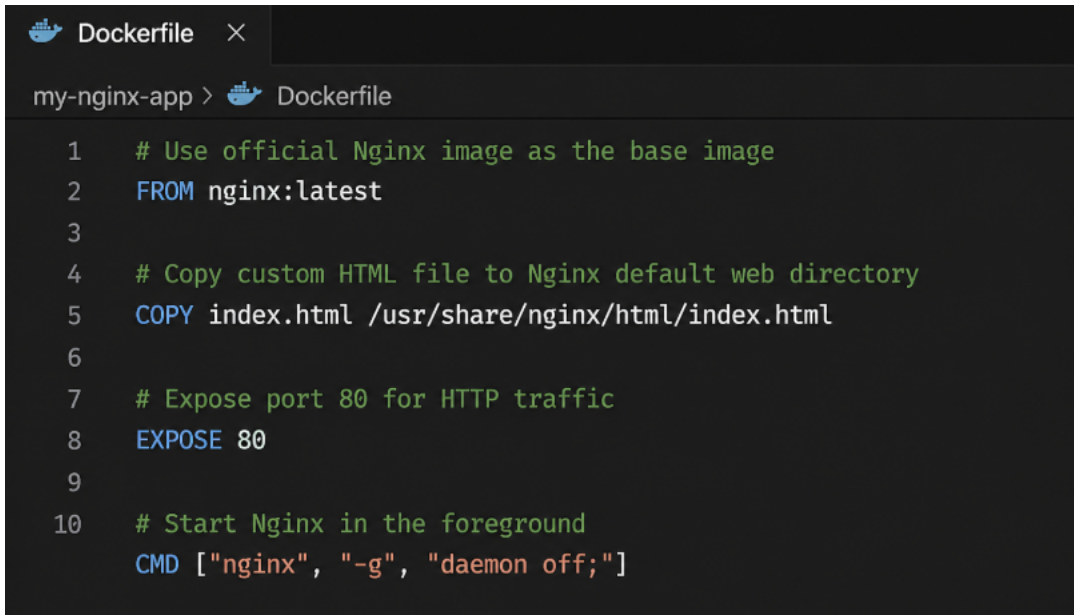
```
docker run -d -p 8080:80 cloud-web-app
```

This command maps port 8080 on the host machine to port 80 inside the container. As a result, the web page can be accessed through a browser using <http://localhost:8080>.

From a cloud engineering perspective, this example demonstrates the main advantage of Docker: the application and its runtime environment are packaged together. This reduces configuration errors and improves deployment consistency across development, testing, and production environments. However, a critical limitation of this simple Dockerfile is that it uses the latest tag, which may reduce reproducibility because the image version can change over time. In a production environment, it would be better to use a specific Nginx version, such as nginx:1.27, to ensure predictable builds.

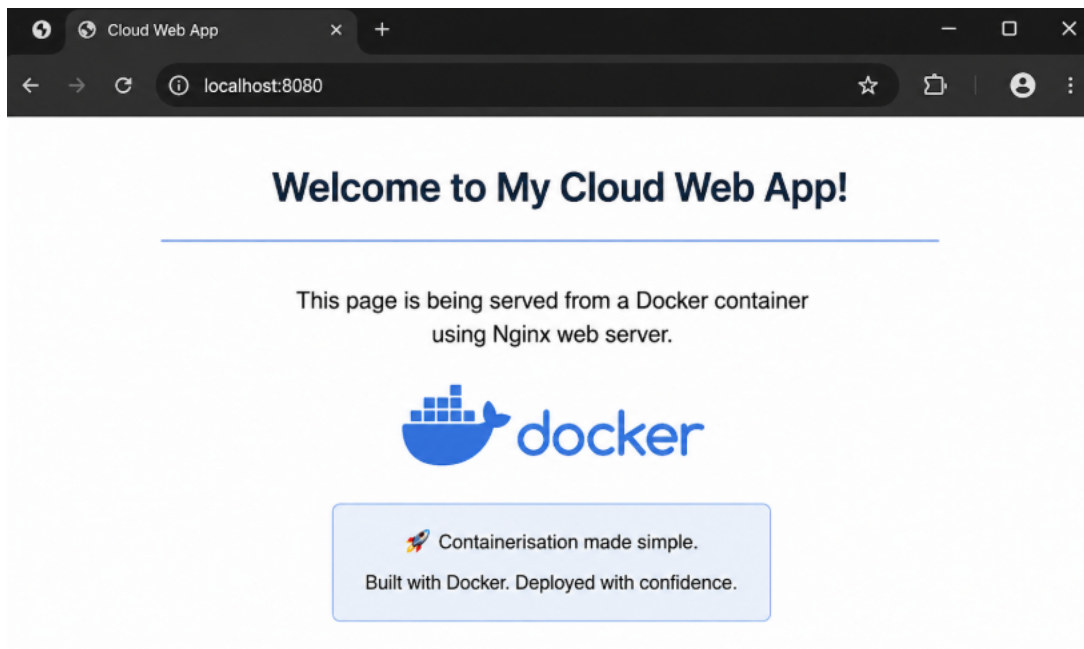
A further improvement would be to follow Docker best practice by selecting a minimal and

trusted base image. Docker recommends choosing trusted and smaller base images because this can reduce image size, improve portability, and minimise security vulnerabilities [5]. Therefore, while the above Dockerfile is suitable for demonstrating the construction of an image, a production-ready Dockerfile should include stricter version control, security scanning, and possibly a non-root user configuration.



```
Dockerfile
my-nginx-app > Dockerfile
1  # Use official Nginx image as the base image
2  FROM nginx:latest
3
4  # Copy custom HTML file to Nginx default web directory
5  COPY index.html /usr/share/nginx/html/index.html
6
7  # Expose port 80 for HTTP traffic
8  EXPOSE 80
9
10 # Start Nginx in the foreground
    CMD ["nginx", "-g", "daemon off;"]
```

(Figure 1 – Dockerfile used to build a custom Nginx web server image)



(Figure 2 – Browser output showing the HTML page running from the Docker container)

2.4 Images and DockerHub

Docker images are fundamental to containerisation, acting as immutable templates from which containers are created. An image contains everything required to run an application, including the operating system libraries, dependencies, and application code. Images are built in layers, with each instruction in a Dockerfile creating a new layer. This layered architecture improves efficiency by allowing Docker to reuse unchanged layers, reducing build time and storage requirements [4].

A key advantage of Docker images is their portability. Once created, an image can be deployed across different environments without modification, ensuring consistency between development, testing, and production systems. This is particularly important in cloud engineering, where applications often need to be deployed across distributed infrastructures. However, this portability also introduces challenges, as poorly configured images may contain unnecessary dependencies or security vulnerabilities, increasing the attack surface.

DockerHub is a cloud-based registry service that allows users to store, manage, and share Docker images. It provides both public and private repositories, enabling collaboration among development teams. By pushing images to DockerHub, developers can make their applications accessible to others or deploy them across multiple cloud environments.

Example commands for publishing an image:

```
docker tag cloud-web-app username/cloud-web-app:1.0
```

```
docker push username/cloud-web-app:1.0
```

The `docker tag` command assigns a repository name and version to the image, while `docker push` uploads it to DockerHub. Versioning is a critical practice, as it ensures that specific builds can be identified and reused reliably. Using explicit version tags instead of the latest tag improves traceability and reduces the risk of unexpected changes in deployments.

From a critical perspective, while DockerHub simplifies image distribution, it also raises concerns regarding security and governance. Public repositories may contain unverified images, which could introduce vulnerabilities if used without proper validation. As a result, organisations often implement private registries or security scanning tools to ensure that only trusted images are deployed. Additionally, reliance on external repositories may create dependencies that affect system reliability if services become unavailable.

In cloud engineering environments, DockerHub plays a crucial role in enabling continuous integration and continuous deployment (CI/CD) pipelines. Images stored in repositories can be automatically pulled and deployed by cloud platforms such as AWS, Azure, or Kubernetes clusters. This automation significantly accelerates the software development lifecycle but requires careful management of image versions, access controls, and security policies.

In summary, Docker images and DockerHub are essential components of modern cloud infrastructure, providing portability, scalability, and ease of deployment. However, their effective

use requires careful consideration of security, versioning, and dependency management to ensure reliable and secure cloud-based systems.

2.5 Container Volumes

One of the fundamental characteristics of Docker containers is their ephemeral nature, meaning that any data stored within a container is lost when the container is removed. While this behaviour supports stateless application design, it presents a significant limitation for applications that require persistent data storage, such as databases or user-generated content systems. To address this issue, Docker provides volumes as a mechanism for persistent data management [4].

A Docker volume is a storage location that exists independently of the container lifecycle. Volumes are managed by Docker and stored on the host machine, allowing data to persist even when containers are stopped, removed, or recreated. This separation between application logic and data storage is a key principle in cloud-native architecture, supporting scalability and resilience.

A basic example of creating and using a Docker volume is shown below:

```
docker volume create web-data
```

```
docker run -d -p 8080:80 -v web-data:/usr/share/nginx/html nginx
```

In this example, the `docker volume create` command generates a named volume called `web-data`. The `-v` flag in the `docker run` command mounts this volume to the Nginx container's `web` directory, ensuring that any data stored in that directory persists independently of the container.

From a cloud engineering perspective, volumes play a critical role in enabling stateful applications within containerised environments. They allow applications to scale horizontally while maintaining consistent access to data. For example, multiple containers can share a

single volume, enabling distributed systems to access common datasets. However, this also introduces challenges related to data consistency, concurrency, and performance, particularly in distributed environments.

A critical limitation of Docker volumes is that they are typically tied to a specific host machine, which can reduce portability in multi-host or cloud environments. In large-scale deployments, this limitation is addressed through the use of networked storage solutions such as AWS Elastic Block Store (EBS), Azure Disk Storage, or distributed file systems. These solutions provide scalable and highly available storage but introduce additional complexity and cost.

Security is another important consideration. Improperly configured volumes can expose sensitive data or allow unauthorised access between containers. As a result, best practices recommend restricting access permissions and carefully managing volume mounts to minimise risk.

In summary, Docker volumes are essential for enabling persistent storage in containerised applications, bridging the gap between stateless containers and stateful services. While they provide significant flexibility and are critical for real-world deployments, their effective use requires careful consideration of scalability, security, and data management strategies within cloud environments.

2.6 Container Networking

Docker provides networking capabilities that allow containers to communicate with each other.

Example:

```
docker network create mynet
```

Containers connected to the same network can exchange data, enabling microservices architectures.

2.7 Full Deployment Workflow Example

A complete workflow demonstrates practical application:

1. Create a simple HTML file
2. Write a Dockerfile
3. Build the image
4. Run the container
5. Access via browser
6. Push image to DockerHub

This process highlights how containerisation simplifies deployment and ensures consistency across environments.

3.0 Cloud Computing Concepts

Cloud computing environments rely on a combination of architectural and infrastructural mechanisms to deliver scalable, reliable, and high-performing services. Among the most critical of these are load balancing, scalability, and application gateways, which collectively enable modern applications to handle dynamic workloads and maintain availability in distributed systems [1].

While these concepts are often discussed individually, their true value emerges when they are implemented together within a cloud architecture. Load balancing ensures efficient distribution of incoming traffic, scalability allows systems to adapt to changing demand, and application gateways provide intelligent routing and security at the application level. However, each of these technologies also introduces trade-offs in terms of complexity, cost, and management overhead.

From a cloud provider perspective, platforms such as Amazon Web Services (AWS) and Microsoft Azure offer managed solutions that abstract much of this complexity. For example, AWS provides Elastic Load Balancing and Auto Scaling services, while Azure offers Load Balancer, Virtual Machine Scale Sets, and Application Gateway. These services enable organisations to deploy highly available systems without managing underlying infrastructure directly. Nevertheless, reliance on managed services can create vendor lock-in and reduce

flexibility in certain scenarios.

A critical consideration in cloud engineering is that these concepts are not standalone solutions but interdependent components of a broader system design. For instance, scalability without load balancing can lead to inefficient resource utilisation, while load balancing without proper scaling may result in performance bottlenecks under high demand. Similarly, application gateways enhance security and routing but add additional latency and configuration complexity.

This section will examine each of these core concepts in detail, evaluating how cloud providers implement them and analysing their role in building resilient and scalable systems. It will also explore the relationship between load balancing and scalability, and demonstrate the benefits of combining all three technologies to deliver robust cloud-based services.

3.1 Load Balancing

Load balancing is a fundamental mechanism in cloud computing that distributes incoming network traffic across multiple servers or resources to ensure optimal performance, reliability, and availability. By preventing any single server from becoming overloaded, load balancing improves system responsiveness and reduces the risk of failure in high-demand environments [5].

Cloud providers such as Amazon Web Services (AWS) and Microsoft Azure implement load balancing through managed services. For example, AWS Elastic Load Balancing (ELB) supports multiple types of load balancers, including Application Load Balancers (Layer 7), Network Load Balancers (Layer 4), and Gateway Load Balancers. Similarly, Azure provides both a basic Load Balancer (Layer 4) and Application Gateway (Layer 7), allowing traffic to be distributed based on protocols, ports, or application-level rules.

A key advantage of load balancing is its ability to enhance fault tolerance. If one server fails, traffic can be automatically redirected to healthy instances, ensuring continuous service

availability. Additionally, load balancers can perform health checks, monitoring the status of backend resources and dynamically adjusting traffic distribution. This is particularly important in cloud environments where instances may frequently scale up or down.

However, load balancing is not without limitations. While it improves availability, it introduces additional components into the system architecture, which can increase complexity and potential points of failure if not properly configured. Furthermore, advanced load balancing features such as session persistence (sticky sessions) may reduce scalability by binding users to specific servers, potentially leading to uneven resource utilisation.

From a performance perspective, load balancers can also introduce slight latency, particularly when operating at the application layer, where deeper inspection of traffic is required. Additionally, misconfigured load balancing policies may lead to inefficient traffic distribution, negating its benefits.

Despite these challenges, load balancing remains essential in cloud engineering, particularly for applications that experience variable or high traffic volumes. When combined with scalability mechanisms, load balancing enables systems to handle increasing demand efficiently while maintaining performance and reliability. In modern cloud architectures, it serves as a critical component in ensuring seamless user experiences and robust service delivery.

3.2 Scalability

Scalability is a core principle of cloud computing that refers to the ability of a system to handle increasing or decreasing workloads by dynamically adjusting its resources. It ensures that applications can maintain performance and availability under varying demand conditions, making it essential for modern cloud-based services [1].

There are two primary types of scalability: vertical scaling and horizontal scaling. Vertical scaling (scaling up) involves increasing the resources of a single machine, such as adding more

CPU, memory, or storage. While this approach is relatively simple to implement, it is inherently limited by the maximum capacity of a single system and may result in downtime during upgrades. In contrast, horizontal scaling (scaling out) involves adding or removing multiple instances of resources, such as virtual machines or containers. This approach is more flexible and aligns closely with cloud-native architectures, particularly in distributed systems.

Cloud providers offer automated scaling solutions to manage resource allocation efficiently. For example, AWS Auto Scaling automatically adjusts the number of instances based on predefined metrics such as CPU utilisation or network traffic. Similarly, Microsoft Azure provides Virtual Machine Scale Sets, which allow applications to scale dynamically in response to demand. These services reduce the need for manual intervention and ensure optimal resource utilisation.

From a critical perspective, scalability is not without challenges. While horizontal scaling improves flexibility, it introduces complexity in managing distributed systems, including issues related to data consistency, session management, and inter-service communication. For example, scaling stateless applications is relatively straightforward, but stateful applications require additional mechanisms such as distributed databases or shared storage, which can increase system complexity and cost.

Furthermore, scalability must be carefully configured to avoid inefficiencies. Over-scaling can lead to unnecessary costs, while under-scaling can result in performance degradation and service outages. Effective scaling strategies rely on accurate monitoring, predictive analysis, and well-defined thresholds to balance performance and cost.

Scalability also has a strong dependency on other cloud components, particularly load balancing. While scalability increases the number of available resources, load balancing ensures that incoming traffic is distributed efficiently across those resources. Without proper load balancing, the benefits of scaling may not be fully realised, as traffic could become unevenly distributed.

In summary, scalability is a fundamental capability of cloud computing that enables systems to adapt to changing workloads. While it provides significant benefits in terms of flexibility and performance, its implementation requires careful planning and integration with other cloud technologies to ensure efficient and reliable system operation.

3.3 Application Gateways

Application gateways are advanced traffic management components in cloud computing that operate at the application layer (Layer 7 of the OSI model). Unlike traditional load balancers that distribute traffic based on IP addresses and ports, application gateways make routing decisions based on application-level data such as URLs, HTTP headers, and cookies. This enables more intelligent traffic management and enhanced security features within cloud environments [5].

Cloud providers such as Microsoft Azure and AWS implement application gateways as managed services. For instance, Azure Application Gateway provides features such as URL-based routing, SSL termination, session affinity, and integration with a Web Application Firewall (WAF). Similarly, AWS offers Application Load Balancers that perform Layer 7 routing and integrate with security services. These capabilities allow organisations to route traffic to different backend services based on specific request characteristics, which is particularly useful in microservices architectures.

A key advantage of application gateways is their ability to enhance security. By operating at Layer 7, they can inspect incoming traffic and block malicious requests using WAF rules, protecting applications from common threats such as SQL injection and cross-site scripting (XSS). Additionally, SSL termination offloads encryption and decryption processes from backend servers, improving performance and simplifying certificate management.

However, application gateways also introduce certain limitations and trade-offs. Due to their deeper inspection of traffic, they can introduce additional latency compared to lower-layer load

balancers. This may impact performance in high-throughput systems if not properly optimised. Furthermore, configuration complexity can be significantly higher, as administrators must define routing rules, security policies, and certificates. Misconfiguration can lead to routing errors or security vulnerabilities.

From a cloud engineering perspective, application gateways play a critical role in modern distributed systems, particularly those based on microservices. They act as a central entry point for incoming traffic, enabling both efficient routing and robust security enforcement. However, their effectiveness depends on proper integration with other components such as load balancers and scaling mechanisms.

In summary, application gateways extend the functionality of traditional load balancing by providing intelligent routing and enhanced security at the application level. While they offer significant benefits in terms of control and protection, they also increase architectural complexity and require careful configuration to ensure optimal performance and reliability.

3.4 Relationship Between Load Balancing and Scaling

Load balancing and scalability are closely related concepts in cloud computing that work together to ensure efficient resource utilisation and system performance. While scalability focuses on increasing or decreasing the number of available resources in response to demand, load balancing is responsible for distributing incoming traffic evenly across those resources. Individually, each provides benefits, but their full effectiveness is only realised when they are integrated within a unified cloud architecture [1].

Scalability enables systems to handle growth by adding additional instances (horizontal scaling) or increasing resource capacity (vertical scaling). However, without load balancing, these additional resources may not be utilised effectively. For example, new instances may remain idle while existing ones become overloaded, leading to performance bottlenecks. Load

balancing addresses this issue by ensuring that traffic is distributed dynamically based on server availability, health, and predefined routing policies.

Conversely, load balancing without scalability also has limitations. While it can optimise the distribution of traffic across existing resources, it cannot compensate for insufficient capacity during periods of high demand. In such scenarios, even evenly distributed traffic may overwhelm all available servers, resulting in degraded performance or service failure. This demonstrates that load balancing alone is insufficient for handling dynamic workloads without the support of scaling mechanisms.

From a critical perspective, the integration of load balancing and scaling introduces both benefits and challenges. On one hand, it enables highly resilient and elastic systems capable of adapting to fluctuating workloads. On the other hand, it increases architectural complexity, requiring careful configuration of scaling policies, health checks, and traffic distribution algorithms. Misalignment between scaling triggers and load balancing behaviour can lead to inefficiencies, such as delayed scaling responses or uneven traffic distribution.

Cloud providers address this interdependency by offering integrated services. For example, AWS Auto Scaling works in conjunction with Elastic Load Balancing to automatically add or remove instances while ensuring that traffic is distributed across healthy resources. Similarly, Azure combines Virtual Machine Scale Sets with its Load Balancer and Application Gateway to provide coordinated scaling and traffic management.

In summary, load balancing and scalability are complementary technologies that must be implemented together to achieve efficient and reliable cloud systems. Scalability provides the necessary capacity to handle demand, while load balancing ensures that this capacity is utilised effectively. Their combined use is essential for maintaining performance, availability, and cost efficiency in modern cloud environments.

3.5 Combined Benefits

When combined, these technologies provide:

- High availability
- Fault tolerance
- Improved performance

For example, an e-commerce application can automatically scale during peak traffic and distribute requests efficiently using load balancing and gateways.

4.0 Driving Forces of Cloud Computing

The rapid expansion of cloud computing over the past decade has been driven by a combination of technological, economic, and organisational factors. Among the most significant drivers are cost efficiency, flexibility, scalability, and the increasing demand for rapid application deployment. While these advantages have accelerated cloud adoption across industries, they also introduce new challenges related to control, security, and long-term cost management [1].

One of the primary driving forces behind cloud computing is cost efficiency. Traditional IT infrastructure requires substantial capital investment in hardware, maintenance, and physical facilities. In contrast, cloud computing operates on a pay-as-you-go model, allowing organisations to convert capital expenditure into operational expenditure. This enables businesses to scale resources based on demand without over-provisioning. However, a critical limitation of this model is that costs can become unpredictable if resources are not properly managed, particularly in large-scale or poorly optimised systems [6].

Flexibility and agility are also key factors contributing to the growth of cloud computing. Cloud platforms enable rapid provisioning of resources, allowing organisations to deploy applications quickly and respond to changing business requirements. This is particularly important in modern software development, where continuous integration and continuous deployment (CI/CD) practices demand fast and reliable infrastructure. Despite these benefits, increased flexibility can lead to governance challenges, as organisations must ensure proper control,

monitoring, and compliance across distributed systems.

Another major driver is the ability to support scalable and globally distributed applications. Cloud providers offer infrastructure that can dynamically scale across multiple regions, enabling high availability and low latency for users worldwide. This has been critical in supporting the growth of digital services such as e-commerce, streaming platforms, and online collaboration tools. However, global scalability also introduces complexity in areas such as data consistency, latency optimisation, and regulatory compliance.

A more recent and significant development in cloud computing is the emergence of serverless computing, also known as Function as a Service (FaaS). Serverless platforms such as AWS Lambda and Azure Functions allow developers to execute code in response to events without managing underlying infrastructure [7]. This further enhances cost efficiency and scalability, as resources are automatically allocated and billed only when the code is executed.

From a critical perspective, while serverless computing reduces operational overhead, it introduces limitations such as reduced control over execution environments, potential vendor lock-in, and challenges related to debugging and performance monitoring. Additionally, serverless architectures may not be suitable for long-running or highly complex applications, where container-based approaches may offer greater flexibility and control.

In summary, the expansion of cloud computing is driven by its ability to deliver cost-effective, flexible, and scalable solutions for modern applications. However, these benefits must be balanced against challenges such as cost management, security, and architectural complexity. Understanding these driving forces is essential for making informed decisions about adopting and designing cloud-based systems.

4.1 Cost Efficiency

Cost efficiency is one of the most significant drivers behind the widespread adoption of

cloud computing. Traditional IT infrastructure requires substantial upfront capital investment in hardware, data centres, maintenance, and personnel. In contrast, cloud computing introduces a pay-as-you-go model, allowing organisations to pay only for the resources they consume. This shift from capital expenditure (CapEx) to operational expenditure (OpEx) enables businesses to reduce initial costs and improve financial flexibility [6].

Cloud providers such as Amazon Web Services (AWS) and Microsoft Azure offer a range of pricing models, including on-demand, reserved, and spot instances, allowing organisations to optimise costs based on their usage patterns. This flexibility is particularly beneficial for startups and small businesses, as it lowers the barrier to entry and enables access to enterprise-level infrastructure without significant investment.

Another important aspect of cost efficiency is resource optimisation. In traditional environments, organisations often over-provision hardware to handle peak workloads, leading to underutilised resources during normal operations. Cloud computing addresses this inefficiency by enabling dynamic scaling, ensuring that resources are allocated only when needed. This reduces waste and improves overall system efficiency.

However, from a critical perspective, the cost advantages of cloud computing are not always straightforward. While the pay-as-you-go model can reduce upfront costs, it can also lead to unpredictable expenses if resources are not carefully monitored and managed. For example, inefficient scaling policies, idle resources, or excessive data transfer can significantly increase operational costs. Additionally, long-term reliance on cloud services may, in some cases, become more expensive than maintaining on-premises infrastructure, particularly for stable and predictable workloads.

Another challenge is the complexity of cloud pricing models. The wide range of services and pricing options can make it difficult for organisations to accurately estimate costs, leading to potential budget overruns. Furthermore, hidden costs such as data egress charges, storage fees, and licensing requirements may not be immediately apparent during initial planning.

In summary, cloud computing offers significant cost efficiency benefits by reducing upfront investment and enabling flexible resource usage. However, achieving true cost efficiency requires careful planning, continuous monitoring, and effective resource management. Without these controls, the financial advantages of cloud computing may be diminished, highlighting the importance of cost optimisation strategies in cloud engineering.

4.2 Flexibility and Agility

Flexibility and agility are key drivers behind the adoption of cloud computing, enabling organisations to respond rapidly to changing business and technical requirements. Cloud platforms provide on-demand access to a wide range of services, including computing power,

storage, databases, and networking, which can be provisioned and configured within minutes. This level of responsiveness allows organisations to accelerate development cycles and reduce time-to-market for new applications and services [1].

A major advantage of cloud-based flexibility is the ability to support dynamic workloads. Applications can be scaled, modified, or redeployed without the constraints of physical infrastructure. This is particularly important in modern development practices such as DevOps and continuous integration/continuous deployment (CI/CD), where rapid iteration and frequent updates are essential. For example, developers can quickly deploy test environments, experiment with new features, and roll back changes if necessary, all without significant downtime.

Cloud computing also enhances organisational agility by supporting global deployment. Applications can be deployed across multiple geographic regions, allowing businesses to reach a wider audience while reducing latency for end users. This capability is critical for services such as e-commerce, streaming platforms, and online collaboration tools, where user experience depends on performance and availability.

However, from a critical perspective, increased flexibility can introduce challenges related to governance, control, and system complexity. The ease of provisioning resources may lead to “resource sprawl,” where unused or unnecessary services accumulate, increasing costs and complicating management. Additionally, the rapid pace of deployment can make it difficult to maintain consistent security policies and compliance standards across distributed environments.

Another limitation is the potential for vendor lock-in. Cloud providers offer proprietary services and tools that may not be easily transferable to other platforms. While these services enhance flexibility within a specific ecosystem, they can reduce portability and limit an organisation’s

ability to switch providers in the future without significant effort.

Furthermore, highly dynamic environments can complicate system monitoring and debugging. As applications scale and change rapidly, identifying performance issues or failures becomes more challenging, requiring advanced monitoring and logging solutions.

In summary, flexibility and agility are major advantages of cloud computing, enabling rapid development, deployment, and scaling of applications. However, these benefits must be balanced with effective governance, cost management, and strategic planning to avoid increased complexity and long-term dependency on specific cloud providers.

4.3 Serverless Computing

Serverless computing allows developers to run code without managing servers. Services such as AWS Lambda and Azure Functions automatically handle scaling and infrastructure management [7].

4.4 Serverless vs Containerisation

Feature	Serverless	Containers
Infrastructure	Fully managed	User managed
Scaling	Automatic	Configurable
Cost	Per execution	Continuous
Flexibility	Limited	High

Serverless is ideal for event-driven applications, while containers provide greater control for complex systems.

5.0 Conclusion

This report has demonstrated the importance of containerisation and Docker in modern cloud environments. Compared to traditional virtualisation, containers offer improved efficiency, scalability, and portability.

Additionally, cloud computing concepts such as load balancing, scalability, and application gateways play a crucial role in delivering reliable and high-performance applications. The

emergence of serverless computing further highlights the evolution of cloud technologies, offering new levels of flexibility and cost efficiency.

Overall, cloud engineering continues to evolve, driven by the need for scalable, efficient, and resilient systems.

References

- [1] P. Mell and T. Grance, *The NIST Definition of Cloud Computing*, National Institute of Standards and Technology, Gaithersburg, MD, USA, 2020.
- [2] R. Buyya, J. Broberg and A. Goscinski, *Cloud Computing: Principles and Paradigms*, 2nd ed. Hoboken, NJ, USA: Wiley, 2021.
- [3] D. Merkel, "Docker: Lightweight Linux Containers for Consistent Development and Deployment," *Linux Journal*, vol. 2014, no. 239, pp. 2–7, Mar. 2020.
- [4] Docker Inc., (2026). *Docker Documentation* [Online]. Available: <https://docs.docker.com>
- [5] Amazon Web Services, (2026). *Elastic Load Balancing Documentation* [Online]. Available: <https://aws.amazon.com/elasticloadbalancing/>
- [6] Microsoft Azure, (2026). *Azure Pricing and Cost Management* [Online]. Available: <https://azure.microsoft.com/pricing/>
- [7] Amazon Web Services, (2026). *AWS Lambda Developer Guide* [Online]. Available: <https://aws.amazon.com/lambda/>
- [8] Microsoft Azure, (2026). *Azure Application Gateway Documentation* [Online]. Available: <https://learn.microsoft.com/azure/application-gateway/>
- [9] B. Burns, B. Grant, D. Oppenheimer, E. Brewer and J. Wilkes, "Borg, Omega, and Kubernetes," *Communications of the ACM*, vol. 59, no. 5, pp. 50–57, May 2022.
- [10] M. Villamizar et al., "Evaluating the Monolithic and Microservice Architecture Pattern," *IEEE Software*, vol. 32, no. 5, pp. 28–35, Sept. 2021.