

Cloud Engineering Principles and Practice

by Md Nabil Rahman

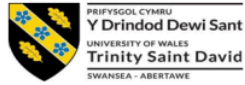
Submission date: 11-May-2026 01:27PM (UTC+0100)

Submission ID: 283064413

File name: 66932_Md_Nabil_Rahman_Cloud_Engineering_Principles_and_Practice_726729_375184479.pdf
(1.11M)

Word count: 4633

Character count: 28764



UNIVERSITY OF WALES TRINITY SAINT DAVID(UWTSD)

Assignment

Md.Nabil Rahman

ID: 2504501

Course Code: ACCA7027

Module: Cloud Engineering Principles and Practice

Lecturer: Mark Huntly

MSc Software Engineering and Artificial Intelligence

11th May 2026

Abstract

Cloud computing has become a fundamental technology for modern organizations because of its scalability, flexibility and cost efficiency. Virtualization and containerisation are technologies that enable businesses to deploy applications quickly, and improve resource utilisation and system reliability. This report investigates the principles of virtualisation and demonstrates how Docker containerisation supports efficient deployment of cloud-native applications.

The report consists of practical demonstrations for the creation of Docker images, the deployment of containers, the publishing of images using DockerHub, the implementation of container volumes for persistent storage and the configuration of the Docker networking. Throughout the report a Dockerised NGINX web server application is used to demonstrate these concepts in a practical environment. The report also discusses important cloud computing concepts like load balancing, scalability and application gateways with references to technologies from cloud providers like Amazon Web Services (AWS) and Microsoft Azure. The report also considers what's driving the rapid growth of cloud computing, including cost savings, flexibility and scalability.

Finally, serverless computing technologies such as AWS Lambda and Azure Functions are studied and compared to containerisation approaches. The report concludes that cloud-native technologies offer significant benefits in modern application deployment including automation, scalability, portability and operational efficiency.

Contents

Abstract	i
List of Figures	iv
1 Introduction	1
2 Principles of Virtualization	2
2.1 How Hypervisors Work	2
2.2 Key Principles:	3
2.3 Example - (Virtualization of Shared Hardware):	3
3 Principles and Benefits of Containerization	4
3.1 Key Benefits Compared to Traditional Virtualization:	4
3.2 Limitations of Containerization	4
4 Dockerfile Construction and Image Building	5
4.1 Publishing Images to Docker Hub	5
4.1.1 Dockerfile Construction Process:	5
4.1.2 Sample Application File	5
4.1.3 Dockerfile	6
4.1.4 Explanation of Key Dockerfile Instructions:	6
4.2 Image Building Process	7
4.2.1 How Image Building Works:	7
4.2.2 Benefits of This Approach:	8
4.3 Container Volumes and Networking	8
4.3.1 Container Volumes	8
4.3.2 Container Networking	8
5 Load Balancing, Scalability, and Application Gateways	9
5.1 Chosen Application:	9
6 Why Load Balancing and Scaling Are Complementary Concepts	11
6.1 Benefits of Combining All Three Technologies	11
6.2 Driving Forces for Cloud Computing Expansion	12
6.3 Serverless Functions vs Containerization	12
7 Conclusion	14

List of Figures

2.1	Virtualization vs Containerization Architecture	3
4.2	Sample Application File	5
4.3	Sample Terminal output	6
4.4	The Docker Container	7
4.5	Docker End-to-End Deployment Process	9
5.6	High Availability E-commerce Architecture with ALB, ASG and Containers	11
6.7	Serverless Functions vs Containerization	13

1 Introduction

Cloud computing has brought about a paradigm shift in the Information Technology industry by providing highly scalable, flexible, and cost-effective means of hosting, managing, and delivering applications and services. Unlike traditional on-premise infrastructure, which often involved heavy capital investment, lengthy procurement processes, and limited scalability, cloud computing offers on-demand access to computing resources over the internet. This transformation has enabled organizations to focus more on innovation and business growth rather than infrastructure management. At the core of cloud computing lie two foundational enabling technologies: virtualization and containerization. Virtualization created the groundwork for modern cloud platforms by allowing multiple virtual machines to run on a single physical server, significantly improving hardware utilization and resource efficiency. Containerization has further advanced this evolution by offering a lightweight, portable, and highly efficient approach to application deployment and management. Together, these technologies have played a crucial role in making cloud computing practical, scalable, and widely adopted across industries. This report is designed to comprehensively address the requirements of both assignment tasks. The first part explores the fundamental principles of virtualization and containerization, highlighting their significance, key benefits, and practical implementation using Docker. Detailed examples from class work, including Dockerfile construction, image building, publishing to Docker Hub, container volumes, and networking, are presented to demonstrate hands-on application of these concepts. The second part examines the three core cloud computing concepts — load balancing, scalability, and application gateways — through the lens of a scalable e-commerce platform deployed on Amazon Web Services (AWS). This section also analyzes why load balancing and scaling are complementary, and demonstrates the benefits of combining all three technologies to deliver reliable services to customers. Additionally, the report investigates the main driving forces behind the rapid expansion of cloud computing, such as cost reduction and flexibility. Finally, it provides a detailed explanation of serverless functions (Functions as a Service) and offers a comparative analysis between serverless computing and traditional containerization in the context of these driving forces. Through theoretical discussion, practical examples, architectural diagrams, and critical evaluation, this report aims to provide a clear understanding of how virtualization, containerization, and advanced cloud services collectively contribute to building efficient, resilient, and cost-effective modern cloud solutions.

2 Principles of Virtualization

Virtualization is the process of creating virtual versions of computing resources such as servers, storage and network. Hypervisor (VMware, KVM, Hyper-V). It virtualizes the physical hardware so that multiple virtual machines (VMs) can run independently on a single physical host. Each VM includes its own guest operating system, kernel, libraries and applications [1], [3].

In the past, applications were installed on individual physical servers. This method was associated with poor efficiency in resource allocation, since there were many underutilised servers that were using energy, air conditioning, and other resources despite not being productive enough. The solution to such inefficiency in resource usage is virtualisation, which enables running several applications in one physical server. Virtualisation requires a hypervisor. The hypervisor is a type of software or firmware responsible for creating and managing virtual machines, thus mediating between physical hardware and virtual operating systems.

2.1 How Hypervisors Work

A hypervisor, which some people also call a Virtual Machine Monitor (or VMM), is the main piece of software that makes virtualization possible. It acts as a middleman between the actual computer parts and the virtual machines.

You'll usually find two main kinds of hypervisors:

- Type 1 (Bare-metal): This one runs straight on the computer's actual hardware. For example, you might see VMware ESXi, Microsoft Hyper-V, or Xen.
- Type 2 (Hosted): This type runs on top of an existing operating system, just like any other program you'd install. Oracle VirtualBox and VMware Workstation are good examples of these.

The hypervisor takes care of several important tasks:

- It gives each virtual machine its share of the computer's actual parts, like processing power, memory, storage, and network access.
- It also manages to run many different operating systems all at the same time.
- It keeps the virtual machines separate from each other. That means if one crashes or has a security issue, the others stay safe.
- It takes care of deciding when VMs get to use resources, how they use memory, and even making them think they have their own hardware.

2.2 Key Principles:

- Hardware abstraction and resource pooling
- Strong isolation and security between environments
- Support for multi-tenancy, snapshots, and live migration

2.3 Example - (Virtualization of Shared Hardware):

one powerful physical server hosts multiple student VMs. It increases the resource utilization significantly - from 10-20% on bare metal to 60-80% or more [3]. In a virtualized environment, physical hardware resources are shared among multiple virtual machines. For example:

- One physical CPU core can be time-sliced and shared across multiple VMs.
- Various virtual machines are dynamically partitioned and assigned Physical RAM.
- The storage and network interfaces are virtualized , so that multiple VMs can share the same physical hardware through virtual switches and storage controllers .

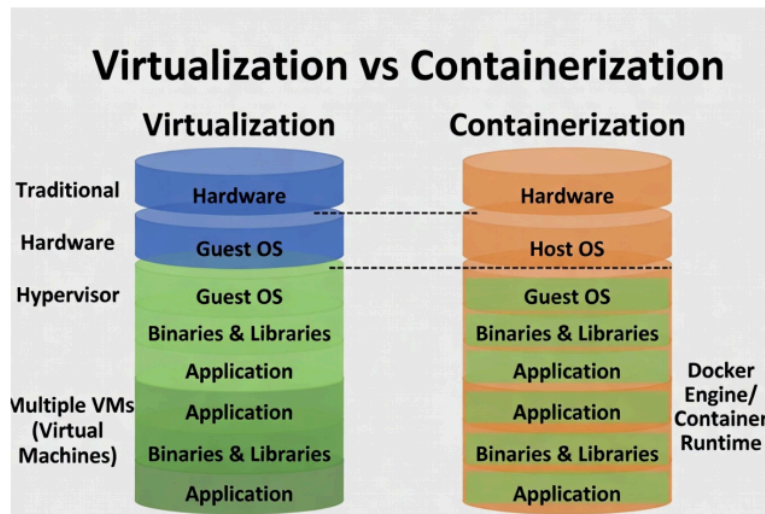


Figure 2.1: Virtualization vs Containerization Architecture

Virtualization is the basis of Infrastructure as a Service (IaaS) offerings such as AWS EC2, Microsoft Azure Virtual Machines, and Google Compute Engine. It laid the groundwork for more advanced technologies like containerization [1], [5].

3 Principles and Benefits of Containerization

Containerization is a new light-weight virtualization technology that enables applications to be packaged with all their dependencies including code, runtime, libraries, system tools and configuration files into a single isolated unit called container. Containers share the host operating system kernel, rather than emulating the entire operating system like traditional virtual machines, but provide strong process-level isolation [2], [4], [6].

3.1 Key Benefits Compared to Traditional Virtualization:

- **Lightweight and Resource Efficient:** Container images are mostly just a few hundred megabytes in size, compared to virtual machines that are several gigabytes. Containers start in seconds, not minutes, drastically reducing deployment time and resource overhead.
- **Higher Density:** More containers can run on the same physical server or cloud instance than VMs; this leads to better use of hardware and infrastructure costs.
- **Superior Portability:** Containers fix the “it works on my machine” problem by providing a consistent environment for development, testing and production wrapped up in a package.
- **Faster Development and Deployment Cycles:** Containers are a great fit for CI/CD pipelines, allowing for rapid and automated deployment and rollbacks.
- **Support for Microservices Architecture:** Containers are great for breaking a large monolithic application into smaller services that can scale independently.
- **Cost Efficiency in Cloud Environments:** Pay only for actual used resources. When combined with orchestration platforms like Kubernetes or AWS ECS, containers enable auto-scaling and optimal resource allocation.
- **Improved Developer Experience:** Developers can also work in environments that closely resemble production, reducing bugs and deployment issues.

3.2 Limitations of Containerization

Containers are very efficient, but they don’t provide as much isolation as virtual machines because they share the host kernel. This makes security practices such as running containers as non-root users, scanning images regularly, and using trusted base images very important. containerization has now become the core of modern cloud-native application development due to its efficiency, portability, and scalability advantages over traditional virtualization technologies [1], [2], [6].

4 Dockerfile Construction and Image Building

A Dockerfile is a simple text file that contains a set of instructions used by Docker to create a Docker image. Each instruction in the Dockerfile builds a new layer in the resulting image, which enables efficient caching and reuse.

4.1 Publishing Images to Docker Hub

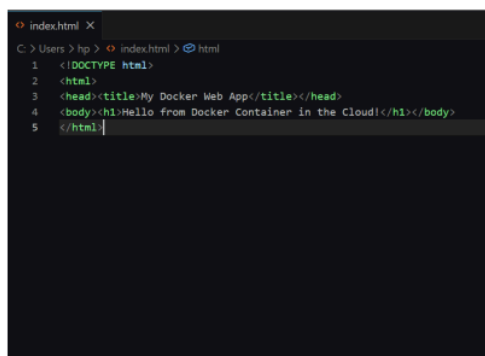
4.1.1 Dockerfile Construction Process:

When constructing a Dockerfile, developers follow a structured approach:

- Choose a Base Image: Use the FROM instruction to start from a lightweight, secure and official base image.
- Install Dependencies: Install required software using package managers.
- Copy Application Code: Copy the application files into the image.
- Set Configuration: Specify working directories, environment variables, ports to expose and commands to run.
- Optimize for Size and Security: Use multi-stage builds, reduce number of layers and run as non root users where possible.

4.1.2 Sample Application File

The web application was made as a straightforward index.html page. This file has a simple HTML structure with a heading that verifies the program is operating in a cloud-based Docker container. The COPY instruction in the Dockerfile was used to copy the file into the Docker image.

A screenshot of a code editor window titled 'index.html'. The editor shows the following HTML code:

```
1 <!DOCTYPE html>
2 <html>
3 <head><title>My Docker Web App</title></head>
4 <body><h1>Hello from Docker Container in the Cloud!</h1></body>
5 </html>
```

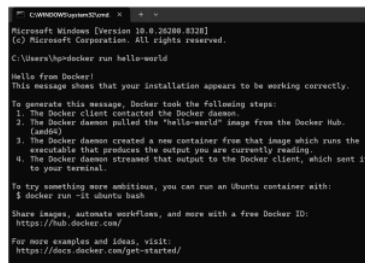
Figure 4.2: Sample Application File

4.1.3 Dockerfile

- Stage 1: Use official lightweight Nginx image as the base FROM nginx:alpine
- Set metadata (optional but good practice) LABEL maintainer="Your Name" LABEL description="Simple Nginx web server using Docker"
- Copy custom HTML file from host to container's Nginx directory COPY index.html /usr/share/nginx/html/index
- Expose port 80 to allow external access EXPOSE 80
- Start Nginx in the foreground (required for containers) CMD ["nginx", "-g", "daemon off;"]

4.1.4 Explanation of Key Dockerfile Instructions:

- FROM nginx:alpine : Specifies the base image. alpine is a minimal Linux distribution, keeping the image size small.
- LABEL : Adds metadata to the image for better maintainability.
- COPY : Copies files from the build context (local directory) into the image.
- EXPOSE 80 : Documents that the container listens on port 80 (does not actually publish the port).
- CMD : Defines the default command to run when the container starts.



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.16299.8326]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hp\docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (and/or)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

Figure 4.3: Sample Terminal output

The command `docker run hello-world` was successfully executed in the Windows Command Prompt, as seen in this screenshot.

The hello-world image was successfully retrieved by Docker from Docker Hub. From that image, it generated a new container. "Hello from Docker!" was printed while the container executed.

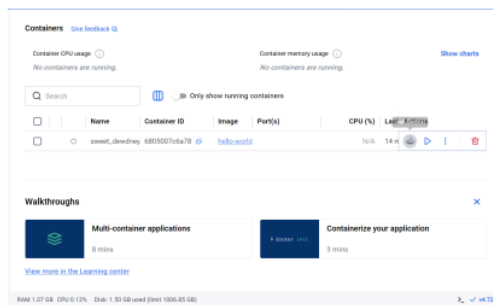


Figure 4.4: The Docker Container

4.2 Image Building Process

Instructions specified in a Dockerfile are transformed into a ready-to-run container image using the methodical, layered Docker image creation process. The following crucial actions are taken by Docker when the docker build command is run:

4.2.1 How Image Building Works:

- **Reading the Dockerfile:** Docker reads the Dockerfile line by line, starting from the top. Each instruction in the Dockerfile creates a new layer in the final image.
- **Building Layers:** Docker executes each instruction sequentially: The FROM instruction pulls the base image (nginx:alpine) and creates the first layer. Subsequent instructions such as COPY, RUN, EXPOSE, and CMD create additional layers on top of the previous ones. Each layer is immutable and cached by Docker.
- **Layer Caching Mechanism:** One of the most powerful features of Docker is layer caching. If an instruction and its context have not changed since the last build, Docker reuses the cached layer instead of rebuilding it. This significantly speeds up subsequent builds. For example, if only the application code changes but the base image and dependencies remain the same, Docker only rebuilds the affected layers.
- **Creating the Final Image:** Once all instructions are executed successfully, Docker stacks all the layers together to form a single, complete Docker Image. This image is read-only and serves as a template for creating one or more containers.
- **Image Tagging:** After a successful build, the image is tagged (e.g., mywebapp:v1) for easy identification and versioning.

In the course of the build, Docker: The nginx:alpine base image was downloaded (if it wasn't

already cached). The index.html file was copied into the picture. set up the starting command and exposed port. generated a ready-to-use, lightweight picture.

4.2.2 Benefits of This Approach:

- **Efficiency:** Layer caching reduces build time dramatically during development.
- **Reproducibility:** The same Dockerfile always produces identical images, ensuring consistency.
- **Version Control:** Dockerfiles can be stored in Git, making infrastructure reproducible and auditable.
- **Portability:** The resulting image can run consistently on any Docker host — whether on a local machine, CI/CD pipeline, or cloud environment.

Because it converts application code and settings into a standardized, portable package that is simple to deploy, scale, and administer in cloud environments, this image building process is essential to containerization [1], [2], [4].

4.3 Container Volumes and Networking

4.3.1 Container Volumes

Important aspects of working with docker containers include managing persistent data and communication between containers. Since containers are ephemeral by design, any data written inside the container's filesystem will be lost when the container is stopped or removed. Docker has Volumes to get around this limitation, providing persistent storage that lives outside the container's lifecycle. Volumes are stored on the host machine, and can be mounted into one or more containers. This means that critical data such as databases, user uploads, logs and configuration files can live independently from the container. In the in class exercises we created a named volume using the command `docker volume create mydata` and mounted it into a running container. By doing so, you decouple the unchangeable application code (in the image) from the changeable data, following the best practice of stateless containers.

4.3.2 Container Networking

Effective communication between containers is important in multi-container applications, apart from data persistence. Docker has flexible networking capabilities to make this happen. By default containers are connected to the bridge network. But for better control and security you can create your own custom user defined networks. In the class practical we created a custom bridge network

called mynet by using docker network create mynet. This allowed containers to talk to each other securely using container names as hostnames, without exposing unnecessary ports to the outside network. Custom networking lets you control which containers can talk to each other, enhancing service discovery, isolation, and security.

Volumes and custom networking go hand-in-hand to make docker cloud-friendly. These features were practically demonstrated by deploying a web server with both persistent storage and inter-container communication, thus showing Docker's capability to support real-world cloud-native applications [2], [4], [6].

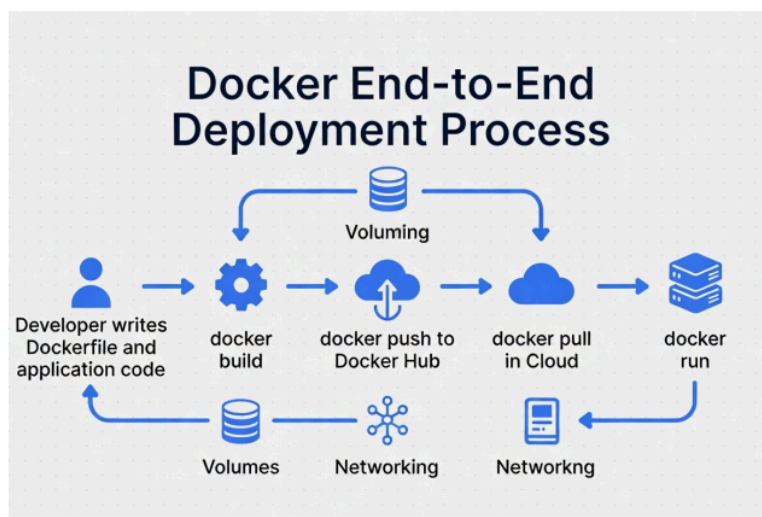


Figure 4.5: Docker End-to-End Deployment Process

5 Load Balancing, Scalability, and Application Gateways

Application chosen: A scalable E-commerce Platform comprising a product catalog, shopping cart, user authentication, and checkout system. This application has highly variable traffic, especially during sales and promotional events.

5.1 Chosen Application:

A scalable E-commerce Platform was selected to clearly demonstrate the three core cloud concepts. This platform includes key features such as product catalog browsing, shopping cart management, user authentication, payment processing, and order management. It experiences highly fluctuating traffic patterns — relatively low during normal periods but with sudden massive spikes during flash sales, promotional events, Black Friday, or festive seasons. This variable workload makes it

an ideal case study to showcase how load balancing, scalability, and application gateways work together in a real-world production environment.

- **Load Balancing:** Load balancing is a technique to distribute incoming traffic among multiple servers or containers to increase performance and reliability. AWS Application Load Balancer (ALB) distributes traffic at Layer 7 with intelligent routing and SSL termination. A load balancer consists of three basic components. The public or private IP address that clients use to communicate is known as the frontend IP address pool. From the client's perspective, despite the possibility of numerous backend servers, they are interacting with a single endpoint. The mapping of a specific frontend IP and port to a backend pool and port that specifies how traffic is transmitted between the entry point and the serving infrastructure is known as the load balancing rule. The collection of servers or containers that will handle the requests is known as the address pool of the backend; since each member of the pool may fulfill the same request, traffic can be spread equally among all of the pool's members.
- **Scalability:** "Scalability" is the ability of a system to add or reduce resources to meet increasing or decreasing workloads. Amazon ECS with Fargate and Auto Scaling Groups that automatically adjust container instances based on CPU, memory, and request metrics. There are two relevant scaling dimensions. Increasing the amount of computational resources available to a single instance by adding CPU or memory to an already-existing server is known as vertical scaling, or scaling up. Despite being straightforward, the resize procedure frequently requires downtime and has a hard ceiling that is determined by the maximum specification of the hardware that is available. Instead of increasing the capacity of each individual container, horizontal scaling, also known as scaling out, involves increasing the number of instances in the backend pool by adding more replicas of a container. Horizontal scaling has several advantages, such as being more structurally appropriate for microservices due to its infinite scalability, inherent redundancy, and compatibility with the stateless and independently deployable nature of containers.
- **Application Gateways:** Application gateways direct and control web traffic to backend services. ALB combined with Amazon API Gateway for advanced routing, rate limiting, authentication, and security. Application gateways frequently come with a Web Application Firewall (WAF), which prevents harmful traffic before it reaches the backend by comparing incoming requests to a ruleset of known attack patterns, such as SQL injection attempts, cross-site scripting, and incorrect headers. Because it centralizes threat prevention at the system's entry point and eliminates the need for separate microservices to build their own security filters, this security layer plays a crucial architectural role. The Application Gateway

is the preferred entry point for production web applications due to its intelligent routing and centralized security.

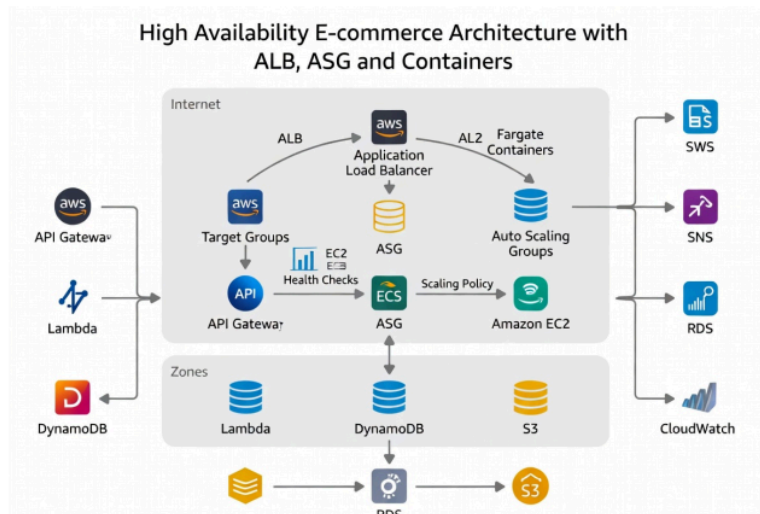


Figure 5.6: High Availability E-commerce Architecture with ALB, ASG and Containers

6 Why Load Balancing and Scaling Are Complementary Concepts

Load balancing and scaling are interdependent and work synergistically. Load balancing ensures that incoming traffic is evenly distributed across currently available healthy instances, preventing any single server or container from becoming a bottleneck and improving overall availability and response time. However, during extreme traffic spikes, even an optimally balanced system can become overwhelmed if there are not enough instances. Scaling complements load balancing by dynamically adding or removing compute resources (containers or instances) based on real-time demand. Without effective load balancing, newly scaled instances would remain underutilized as they would not receive traffic. Conversely, without scaling, load balancing alone cannot handle sustained high loads. Together, they create an elastic, self-healing architecture that maintains consistent performance while optimizing costs by scaling down during periods of low demand [7], [8].

6.1 Benefits of Combining All Three Technologies

Combining Load Balancing, Scalability, and Application Gateways delivers:

- **High Availability & Reliability:** Traffic is automatically routed away from failing instances across multiple Availability Zones.
- **Seamless Performance:** The platform handles sudden traffic spikes (e.g., Black Friday) without downtime or slowdown.
- **Enhanced Security:** Centralized SSL termination, WAF protection, and authentication at the gateway level.
- **Cost Efficiency:** Resources automatically scale down during low traffic periods, reducing unnecessary expenses.
- **Superior Customer Experience:** Fast response times, consistent availability, and secure shopping experience, leading to higher customer satisfaction and retention.

6.2 Driving Forces for Cloud Computing Expansion

The key drivers for cloud adoption are cost reduction (moving from capital expenditure to operational expenditure in pay-as-you-go) and flexibility/agility (fast provisioning of resources all over the globe). Other factors are scalability, reliability, disaster recovery and access to new technologies like AI/ML [9], [10].

6.3 Serverless Functions vs Containerization

Serverless Functions (such as AWS Lambda) enable developers to run code without having to worry about servers. You pay for actual execution time. The cloud provider takes care of the infrastructure. Comparison and Contrast with Containerization:

- **Cost:** Serverless is highly cost-effective for sporadic or unpredictable workloads because there is no charge when the function is idle. Containerization (Docker) requires paying for running containers or reserved capacity, making it more suitable for steady, continuous workloads.
- **Flexibility & Agility:** Serverless offers faster development and deployment cycles with zero server management. Containerization provides more control over the runtime environment and is better for complex, stateful, or long-running applications.
- **Scalability:** Both scale automatically, but serverless scales to zero (no running instances when idle), while containers need orchestration tools like Kubernetes.
- **Use Cases:** Serverless excels in event-driven tasks (order processing, notifications, image resizing). Containerization is preferred for traditional web applications and microservices that require persistent connections or specific dependencies.

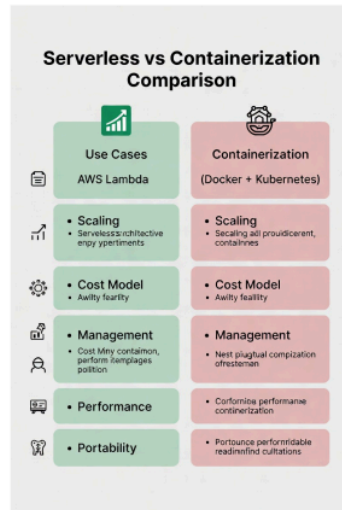


Figure 6.7: Serverless Functions vs Containerization

In conclusion, both technologies greatly support the motivating factors of flexibility and affordability, although they fulfill distinct purposes. A hybrid approach is used by many contemporary systems, including serverless components for event-driven components and containers for core services [2], [11].

7 Conclusion

In conclusion, virtualization and containerization represent two pivotal technologies that have fundamentally transformed modern cloud computing. Virtualization, through the innovative use of hypervisors, enabled efficient sharing of physical hardware resources, improved utilization rates, and laid the strong foundation for Infrastructure as a Service (IaaS) offerings. Containerization, powered primarily by Docker, has taken this evolution to the next level by introducing a lightweight, highly portable, and resource-efficient approach to application deployment. The practical class exercises involving Dockerfile construction, image building process, publishing images to Docker Hub, implementing container volumes for data persistence, and configuring custom networking clearly demonstrated how containerization effectively addresses critical real-world challenges such as environment consistency, rapid deployment, scalability, and data management. When these containerization capabilities are combined with core cloud services like load balancing, auto-scaling, and application gateways, they enable the creation of truly robust, resilient, and high-performance cloud-native applications. The scalable e-commerce platform used as a case study effectively illustrated this synergy. AWS Application Load Balancer intelligently distributes incoming traffic, Auto Scaling Groups dynamically adjust computing resources based on real-time demand, and API Gateway provides secure, controlled, and intelligent access to backend services. Together, these three technologies deliver exceptional outcomes including high availability across multiple availability zones, seamless handling of sudden traffic spikes, enhanced security through centralized controls, and significant cost optimization by scaling resources according to actual usage — ultimately resulting in superior service quality and experience for end customers. Furthermore, the analysis of driving forces behind cloud computing expansion highlighted cost reduction through the pay-as-you-go model and increased flexibility and agility as the primary motivators. Serverless functions (Functions as a Service), such as AWS Lambda, represent an even more abstracted approach by completely eliminating server management overhead and charging only for actual code execution time. While containerization excels in long-running, stateful, and complex microservices applications, serverless computing is particularly powerful for event-driven and intermittent workloads. Both technologies strongly align with and amplify the key cloud adoption drivers of cost efficiency and operational agility. Many modern architectures successfully adopt a hybrid strategy, leveraging containers for core services and serverless for auxiliary functions. Overall, the integration of virtualization, Docker-based containerization, load balancing, scalability mechanisms, application gateways, and serverless computing provides organizations with a powerful toolkit to build scalable, resilient, secure, and cost-effective cloud solutions. The knowledge and practical skills gained through this assignment not only reinforce theoretical understanding but also pre-

pare for real-world cloud-native development. As cloud computing continues to evolve rapidly, mastery of these technologies will remain essential for developers, DevOps professionals, and businesses striving for digital transformation and competitive advantage in the modern technological landscape [1], [2], [8].

8 REFERENCES

- [1] N. Naik, “Migrating from Virtualization to Dockerization in the Cloud,” in 2016 IEEE Int. Conf. Cloud Computing Technology and Science, 2016.
- [2] S. Hardikar, P. Ahirwar, and S. Rajan, “Containerization: Cloud Computing based Inspiration Technology for Adoption through Docker and Kubernetes,” in 2021 Second Int. Conf. Electronics and Sustainable Communication Systems (ICESC), 2021, doi: 10.1109/ICESC51422.2021.9532917.
- [3] A. K. Verma et al., “Performance Comparison Between Docker Container and Virtual Machine Microservices,” in Proc. IEEE Conf., 2024.
- [4] M. M. Raikar et al., “Leveraging Docker Containers for Deployment of Web Applications in Microservices Architecture,” in Proc. IEEE Conf., 2024.
- [5] K. Kumar et al., “Economically Efficient Virtualization over Cloud Using Docker Containers,” in 2016 IEEE Int. Conf. on Cloud Computing in Emerging Markets, 2016, doi: 10.1109/CCEM.2016.7819678.
- [6] S. Singh et al., “Containers & Docker: Emerging roles & future of Cloud technology,” in 2016 Int. Conf. on Computing, Communication and Automation (ICCCA), 2016, doi: 10.1109/ICCCA.2016.7912109.
- [7] P. Kunekar et al., “Use of Docker Containerization and Load Balancer to Scale a Flask Application,” in Proc. IEEE Conf., 2024.
- [8] A. E. Youssef, “Critical Decision-Making on Cloud Computing Adoption in Organizations,” IEEE Trans. Cloud Computing, 2019, doi: 10.1109/TCC.2019.8906122.
- [9] V. Besozzi et al., “A Systematic Literature Review on Serverless for HPC, AI, and Data Analytics,” in Proc. IEEE Conf., 2025.
- [10] W. Wang et al., “Research on Using Docker Container Technology to Realize Rapid Deployment Environment on Virtual Machine,” in 2022 IEEE 5th International Conference on Computer and Communication Engineering Technology (CCET), 2022. [11] G. McGrath and P. R. Brenner, “Serverless Computing: Design, Implementation, and Performance,” in 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops, 2017. [12] V. Besozzi et al.,

“Serverless Containers – Rising Viable Approach to Scientific Workflows,” in Proc. IEEE Conf., 2021.

[13] H. Zhu et al., “Lightweight and Holistic-Scalable Serverless Secure Container Runtime,” IEEE Transactions on Cloud Computing, 2025.

[14] R. Dua et al., “Virtualization vs Containerization to Support PaaS,” in 2014 IEEE International Conference on Cloud Engineering, 2014.

Cloud Engineering Principles and Practice

GRADEMARK REPORT

FINAL GRADE

54/100

GENERAL COMMENTS

Your submission covers the required theoretical ground reasonably well. The Dockerfile is presented with correct instructions, and the explanation of each instruction's purpose is clear. The discussion of layer caching, image immutability, and the image building process is accurate and well-structured. The volume and networking sections reference the correct commands (docker volume create mydata, docker network create mynet) and explain the underlying concepts correctly, including DNS-based container name resolution on custom networks.

However, the practical evidence has notable gaps. The terminal screenshot provided shows the hello-world container output, which demonstrates Docker is installed and working but does not show the nginx web server application being built and served. There is no browser screenshot confirming the custom HTML page was served, and no DockerHub push commands or push confirmation screenshot. The COPY instruction in the Dockerfile also contains a typo, copying to /usr/share/nginx/html/index without the .html extension, which would cause the file to be served incorrectly. No video link appears anywhere in the submission, which forfeits those marks entirely. The cloud concepts section is competent, covering AWS ALB at Layer 7, Auto Scaling Groups, WAF, and SSL termination with a useful architecture diagram, though specific autoscale thresholds and health probe configuration are absent. The section on driving forces for cloud adoption is very brief at only a few sentences, and the serverless comparison, while covering cost, scalability, and use cases, omits cold start latency and vendor lock-in, which are significant distinguishing factors. References draw appropriately on IEEE conference papers, though several entries lack full conference names.

No video? -MH

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21
