



ACCA7033

Ethical Hacking

Lecturer: Renuga Jayakumar

Project Name: Cloud Engineering Principles and Practice

Submitted by -

MD. Mahmudul Hassan

ID - 2504534

Submission Deadline: Thursday, 18<sup>th</sup> December 2025

## Contents

|                                                                                |    |
|--------------------------------------------------------------------------------|----|
| 1. The Principles of Virtualisation.....                                       | 4  |
| 1.1 Definition.....                                                            | 4  |
| 1.2 Core Principles.....                                                       | 4  |
| B. Isolation.....                                                              | 4  |
| C. Resource Sharing .....                                                      | 4  |
| D. Encapsulation .....                                                         | 4  |
| E. Portability (Live Migration).....                                           | 4  |
| Example: Virtualisation of Shared Hardware .....                               | 5  |
| Benefits .....                                                                 | 5  |
| 2.1 Transportable Isolation .....                                              | 6  |
| 2.2 Image-Based Implementation .....                                           | 6  |
| 2.3 Dependency Packaging .....                                                 | 6  |
| 2.4 Cross-Environment Portability .....                                        | 6  |
| 2.5 Scalability with Microservices .....                                       | 7  |
| Figure: Docker file to Build an Image .....                                    | 8  |
| 3.2 Publishing Docker Images .....                                             | 9  |
| Use Cases .....                                                                | 11 |
| Part: 02 .....                                                                 | 23 |
| 2.1 Load Balancing .....                                                       | 23 |
| 2.2 Scalability.....                                                           | 23 |
| 2.3 Application Gateways.....                                                  | 24 |
| 2. Complementary Nature of Load Balancing and Scalability .....                | 24 |
| 3. Advantages of Integrating All Three Technologies .....                      | 25 |
| 4. Motivators for the Growth of Cloud Computing (Cost, Flexibility, etc.)..... | 25 |
| 4.1 Cost-Effectiveness .....                                                   | 25 |
| 4.2 Flexibility & Rapid Deployment .....                                       | 25 |
| 4.3 Disaster Recovery & Reliability .....                                      | 25 |
| 4.4 Global Reach .....                                                         | 25 |
| 5.1 Serverless Functions.....                                                  | 25 |
| 5.2 Containerization.....                                                      | 26 |
| 5.3 Comparison of Driving Forces .....                                         | 26 |
| 6. Conclusion.....                                                             | 27 |

|                  |    |
|------------------|----|
| References ..... | 28 |
|------------------|----|

## Part 01:

### **1. The Principles of Virtualisation**

#### **1.1 Definition**

Virtualization is a technology that permits numerous autonomous operating systems—known as virtual machines (VMs) to run simultaneously on a single physical machine. This is achieved through a software layer called a hypervisor, which manages and allocates hardware resources to each VM while maintaining isolation between them [1].

#### **1.2 Core Principles**

##### **A. Abstraction**

The hypervisor abstracts the underlying physical hardware and presents each VM with its own virtual CPU, memory, storage and network interface. Through this abstraction, VMs operate independently of the physical hardware configuration, enabling a high degree of flexibility and hardware independence [2].

##### **B. Isolation**

Each VM runs in complete isolation from the others. A failure, crash or security breach within one VM does not affect other VMs or the host system. This isolation ensures security, stability and controlled resource access across virtualised environments [1], [3].

##### **C. Resource Sharing**

Multiple VMs share the same physical hardware resources dynamically. Idle resources can be reassigned to busy VMs, resulting in efficient utilisation of server hardware, improved scalability and reduced operational cost. This principle underpins server consolidation in cloud infrastructures [2], [4].

##### **D. Encapsulation**

A VM exists as a collection of files—including virtual disk images and configuration data. This encapsulation makes VMs portable, easy to back up, clone and restore, supporting disaster recovery and simplified system management [1], [5].

##### **E. Portability (Live Migration)**

Virtual machines can be moved between physical hosts with minimal or no downtime, a process known as live migration. Portability supports load balancing, maintenance and high availability without service interruption [3] [5].

### Example: Virtualisation of Shared Hardware

A single high-performance server with:

- 32 CPU cores
- 128 GB RAM
- 2 TB SSD

can run 10 virtual machines, each configured with:

- 4 vCPUs
- 12 GB RAM
- 200 GB storage

Instead of purchasing 10 separate physical servers, the organisation can host multiple virtual servers on shared hardware through a hypervisor. This approach enables efficient resource sharing and workload isolation [5], [6].

### Benefits

- Lower cost: Fewer physical servers are required, reducing hardware and operational expenses [5].
- Better utilisation: Resource abstraction and dynamic allocation improve hardware efficiency [6].
- Easier management: Centralised administration, snapshots, and standardised deployments simplify management [7].

## 2. The Fundamentals and Advantages of Containerization: An Illustration of a Dockerfile Definition

Containerization creates a lightweight, portable, and isolated environment known as a container, which bundles an application with all its dependencies. Containers, unlike virtual machines, share the host operating system (OS) kernel, making them notably more efficient in terms of performance and start up time.

### 2.1 Transportable Isolation

Containers run isolated user-space processes while sharing the host OS kernel. This enables strong separation of applications while greatly reducing overhead compared to virtual machines. Containers typically start within milliseconds due to their lightweight architecture [8].

- Minimal resource consumption
- Rapid startup time (milliseconds)

### 2.2 Image-Based Implementation

Applications are packaged into immutable container images. These images ensure predictable and consistent deployments across environments, reducing configuration drift and improving reliability [10].

- Uniform deployments
- Repeatable infrastructure behaviour

### 2.3 Dependency Packaging

Each image contains all the necessary runtime components, such as libraries, executables, and configurations this avoids the usual issue of "works on my machine," which provides environment consistency for development, testing, and production.[9].

- Images include all libraries, code, and runtime
- Removes environment-related execution problems

### 2.4 Cross-Environment Portability

A container image can run on:

- A developer workstation
- Any cloud platform supporting Docker or Kubernetes
- On-premise servers

Because the container format is standardized, applications become highly portable across heterogeneous infrastructure

## 2.5 Scalability with Microservices

Containers are well-suited to small service architecture. Every service function in its own container, enabling independent deployment and scaling. This design improves fault isolation and supports modern, distributed applications

- Each microservice operates independently
- Services can scale individually

### Advantages of Containerization

| Advantage         | Explanation                                                                                   |
|-------------------|-----------------------------------------------------------------------------------------------|
| Speed             | Containers start in milliseconds or seconds, enabling rapid scaling [8].                      |
| Efficiency        | Containers consume significantly fewer resources than VMs due to OS-level virtualization [9]. |
| Consistency       | The same image runs uniformly across environments [10].                                       |
| Flexibility       | Well-suited for Kubernetes auto-scaling and orchestration [11].                               |
| Isolation         | Issues in one container don't affect others. [12].                                            |
| Streamlined CI/CD | Supports automated builds, testing, and deployments, simplifying DevOps pipelines [10].       |

The screenshot shows the Visual Studio Code interface with a project named 'BABOR'. The Explorer sidebar on the left shows a file named 'dockerfile' selected. The main editor displays the content of 'dockerfile', which contains the following instructions:

```
1
2 FROM nginx:alpine (last pushed 1 month ago)
3 COPY index.html /usr/share/nginx/html/index.html
4
```

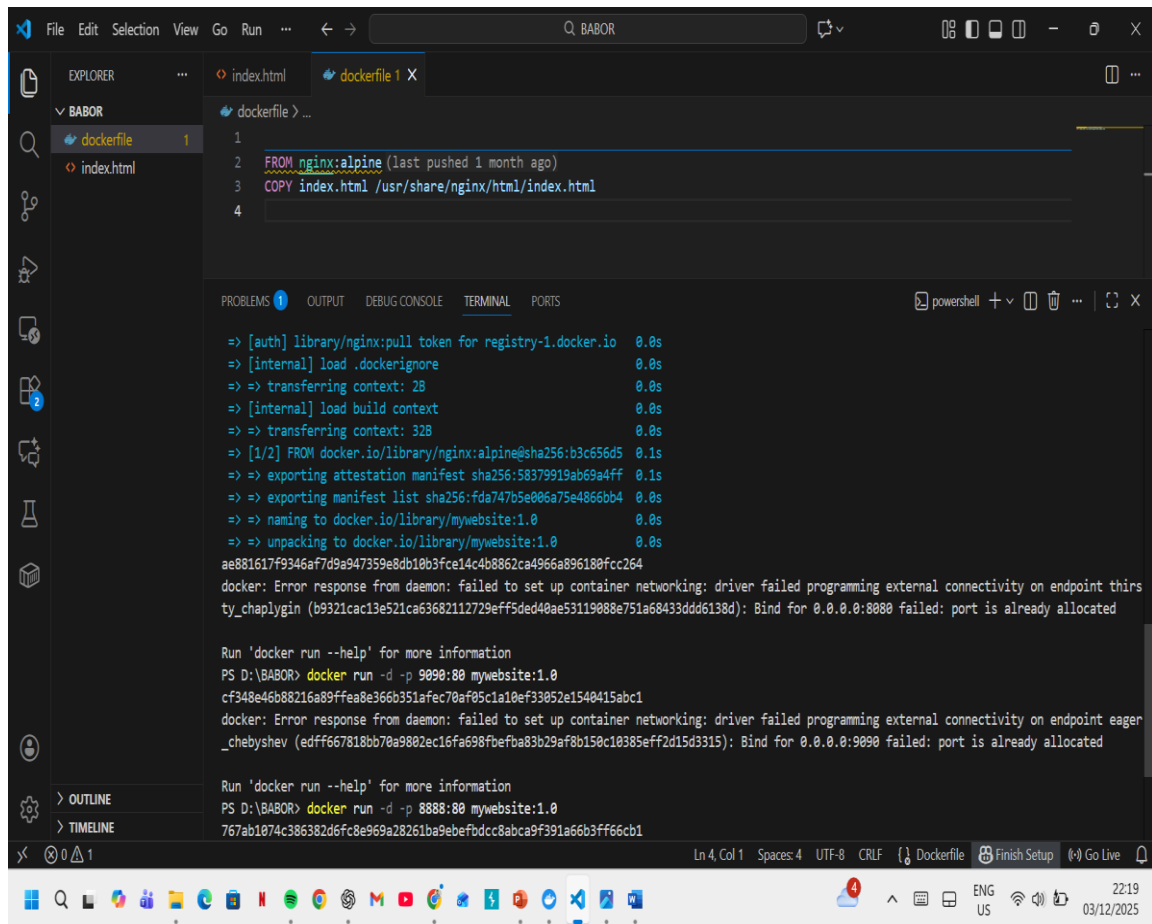
The bottom panel shows the 'TERMINAL' tab with the output of the Docker build process. The commands executed are:

```
PS D:\BABOR> docker build -t mywebsite:1.0 .
[+] Building 2.7s (8/8) FINISHED    docker:desktop-linux
=> [internal] load build definition from dockerfile    0.0s
=> => transferring dockerfile: 100B    0.0s
=> [internal] load metadata for docker.io/library/nginx:alpi 2.0s
=> [auth] library/nginx:pull token for registry-1.docker.io 0.0s
=> [internal] load .dockerignore    0.0s
=> => transferring context: 2B    0.0s
=> [internal] load build context    0.0s
=> => transferring context: 32B    0.0s
=> [1/2] FROM docker.io/library/nginx:alpine@sha256:b3c656d5 0.1s
=> => exporting attestation manifest sha256:58379919ab69a4ff 0.1s
=> => exporting manifest list sha256:fda747b5e006a75e4866bb4 0.0s
=> => naming to docker.io/library/mywebsite:1.0    0.0s
=> => unpacking to docker.io/library/mywebsite:1.0    0.0s
ae881617f9346af7d9a947359e8db18b3fce14c4b8862ca4966a896180fcc264
docker: Error response from daemon: failed to set up container networking: driver failed programming external connectivity on endpoint thirs
ty_chaplygin (b9321cac13e521ca63682112729eff5ded40ae53119088e751a68433ddd6138d): Bind for 0.0.0.0:8080 failed: port is already allocated

Run 'docker run --help' for more information
PS D:\BABOR> docker run -d -p 9090:80 mywebsite:1.0
cf348e46b88216a89ffea8e366b351afec70af05c1a10ef33052e1540415abc1
docker: Error response from daemon: failed to set up container networking: driver failed programming external connectivity on endpoint eager
```

The status bar at the bottom indicates the file is 'Ln 4, Col 1', uses 'Spaces: 4', 'UTF-8' encoding, and 'CRLF' line endings. The system clock shows 22:18 on 03/12/2025.

Figure: Docker file to Build an Image



The screenshot shows the Visual Studio Code interface with a file explorer on the left showing 'index.html' and 'dockerfile 1'. The main editor displays the content of 'dockerfile 1' with the following lines:

```
1 FROM nginx:alpine (last pushed 1 month ago)
2 COPY index.html /usr/share/nginx/html/index.html
3
4
```

The terminal at the bottom shows the output of running 'docker build' and 'docker run' commands. The 'docker build' command successfully builds the image 'mywebsite:1.0' with the following output:

```
=> [auth] library/nginx:pull token for registry-1.docker.io 0.0s
=> [internal] load .dockerignore 0.0s
=> => transferring context: 28 0.0s
=> [internal] load build context 0.0s
=> => transferring context: 328 0.0s
=> [1/2] FROM docker.io/library/nginx:alpine@sha256:b3c656d5 0.1s
=> => exporting attestation manifest sha256:58379919ab69a4ff 0.1s
=> => exporting manifest list sha256:fda747b5e006a75e4866bb4 0.0s
=> => naming to docker.io/library/mywebsite:1.0 0.0s
=> => unpacking to docker.io/library/mywebsite:1.0 0.0s
ae881617f9346af7d9a947359e8db10b3fca14c4b8862ca4966a896180fcc264
```

The 'docker run' command fails with the following error:

```
docker: Error response from daemon: failed to set up container networking: driver failed programming external connectivity on endpoint thirs
ty_chaplygin (b9321cac13e521ca63682112729eff5ded40ae53119088e751a68433ddd6138d): Bind for 0.0.0.0:8080 failed: port is already allocated
```

The terminal also shows the command 'Run "docker run --help" for more information' and the command 'PS D:\BATOR> docker run -d -p 9090:80 mywebsite:1.0' which fails with the same error.

### 3. Images and Publishing

#### 3.1 Understanding Docker Images

A Docker image is a packaged application that contains the application code, runtime environment, system tools, libraries, and dependencies required for execution. Images are built from layered file systems, making them lightweight, modular, and easy to distribute across environments [13], [14]. Because each layer is immutable, images support reproducible deployments and efficient storage.

#### 3.2 Publishing Docker Images

"Publishing" an image entails uploading the Docker image to a Docker registry. A registry—such as Docker Hub, Amazon ECR, GitHub Container Registry, or a private company registry—is a centralized storage system for images [15]. Once published, photos can be downloaded ("pulled") by other users, automated workflows, or production systems, facilitating broad reuse and consistent publication [16]. This strategy is critical for collaborative development, CI/CD workflows, and cloud-native deployment of apps.

```
File Edit Selection View Go Run ... Q BABOR
EXPLORER
  BAVOR
    dockerfile 1
    index.html
  OUTLINE
  TIMELINE dockerfile
    File Saved 21 mins
    File Saved 24 mins
    File Saved 30 mins
    File Saved 43 mins
  dockerfile > ...
  1 FROM nginx:alpine (last pushed 1 month ago)
  2 COPY index.html /usr/share/nginx/html/index.html
  3

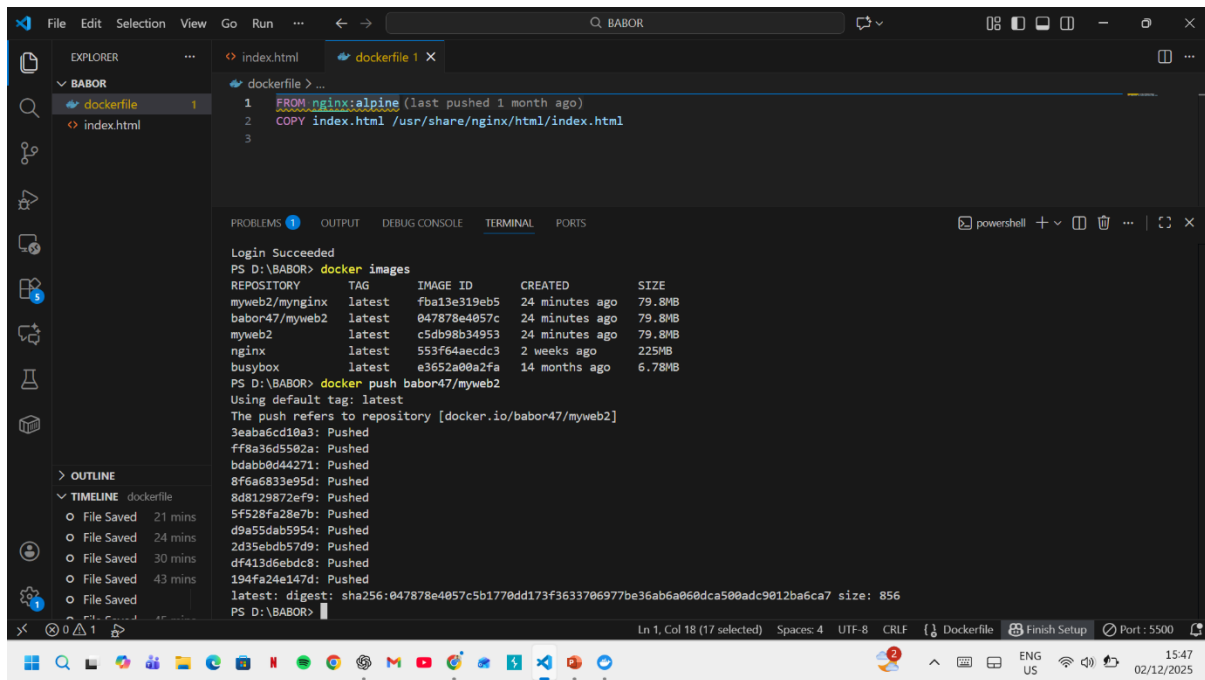
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\BABOR> docker build -t babor47/myweb2 .
[+] Building 2.1s (8/8) FINISHED
=> [internal] load build definition from dockerfile
=> => transferring dockerfile: 106B
=> [internal] load metadata for docker.io/library/nginx:alpine
=> [auth] library/nginx:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load build context
=> => transferring context: 32B
=> [1/2] FROM docker.io/library/nginx:alpine@sha256:b3c656d55d7ad751196f21b7fd2e8d4da9cb430e32f646adcf92441b72f82b14
=> => resolve docker.io/library/nginx:alpine@sha256:b3c656d55d7ad751196f21b7fd2e8d4da9cb430e32f646adcf92441b72f82b14
=> CACHED [2/2] COPY index.html /usr/share/nginx/html/index.html
=> => exporting to image
=> => exporting layers
=> => exporting manifest sha256:edc085cf0e3074258a374414ccdb9846a0d19bcf9f79cccc628cfd5636fd5d79
=> => exporting config sha256:7c94413d89218fac85918811438c3c06185a5ff5b0fe909310c3ceca38ce4a8a
=> => exporting attestation manifest sha256:ea96110517c5dd1880f0eaa0986d8901630d0a8f84bfc219a65b93b3436818cc
=> => exporting manifest list sha256:047878e4057c5b1770dd173f3633706977be36ab6a060dca500adc9e12ba6ca7
=> => naming to docker.io/babor47/myweb2:latest
=> => unpacking to docker.io/babor47/myweb2:latest
PS D:\BABOR> docker login
Authenticating with existing credentials... [Username: babor47]
```

```
File Edit Selection View Go Run ... Q BABOR
EXPLORER
  BAVOR
    dockerfile 1
    index.html
  OUTLINE
  TIMELINE dockerfile
    File Saved 21 mins
    File Saved 24 mins
    File Saved 30 mins
    File Saved 43 mins
  dockerfile > ...
  1 FROM nginx:alpine (last pushed 1 month ago)
  2 COPY index.html /usr/share/nginx/html/index.html
  3

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\BABOR> docker login
Authenticating with existing credentials... [Username: babor47]

Info -> To Login with a different account, run 'docker Logout' followed by 'docker Login'

Login Succeeded
PS D:\BABOR> docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
myweb2/myngx latest fba13e319eb5 24 minutes ago 79.8MB
babor47/myweb2 latest 047878e4057c 24 minutes ago 79.8MB
myweb2 latest c5db90b34953 24 minutes ago 79.8MB
nginx latest 553f64aecd3 2 weeks ago 225MB
busybox latest e3652a00a2fa 14 months ago 6.78MB
PS D:\BABOR> docker push babor47/myweb2
Using default tag: latest
The push refers to repository [docker.io/babor47/myweb2]
3eaba6cd10a3: Pushed
ff8a36d5502a: Pushed
bdabb0d44271: Pushed
8f6a6833e95d: Pushed
8d8129872ef9: Pushed
5f528fa28e7b: Pushed
```

The screenshot shows the Visual Studio Code interface. The Explorer pane on the left shows a project named 'BABOR' with files 'dockerfile' and 'index.html'. The main editor shows the 'dockerfile' with three lines: 'FROM nginx:alpine (last pushed 1 month ago)', 'COPY index.html /usr/share/nginx/html/index.html', and an empty third line. The Terminal pane at the bottom shows the output of Docker commands. It starts with 'Login Succeeded', then 'PS D:\BABOR> docker images' which lists several images including 'myweb2/mynginx', 'babor47/myweb2', 'myweb2', 'nginx', and 'busybox'. Then 'PS D:\BABOR> docker push babor47/myweb2' is executed, showing the push progress for the 'latest' tag and the resulting digest: 'latest: digest: sha256:047878e4057c5b1770dd173f3633706977be36ab6a060dca500adc9012ba6ca7 size: 856'. The status bar at the bottom indicates 'Ln 1, Col 18 (17 selected)', 'Spaces: 4', 'UTF-8', 'CRLF', and 'Port: 5500'.

## 4. Explanation of Container Volumes

Container volumes provide a persistent storage mechanism for Docker containers. Unlike the container’s internal filesystem—which is deleted when the container is stopped, removed, or recreated—volumes store data independently of the container lifecycle [17]. This ensures that critical information is preserved across updates, failures, and deployments.

Volumes are managed by the Docker engine and can be mounted into one or more containers simultaneously. This design supports reliable data sharing, backup processes, and stateful application workloads [18].

### Use Cases

Volumes are especially beneficial for applications requiring persistent or shared data, including [19]:

- Databases: storing tables, indexes, and transaction logs
- Application logs: retaining logs for debugging, monitoring, and auditing
- Configuration files and state data: ensuring continuity across container restarts

By separating persistent data from the ephemeral nature of containers, volumes enable more robust, maintainable, and production-ready containerized systems.

File Edit Selection View Go Run ... Q BABOR

EXPLORER

- BABOR
  - dockerfile 1
  - index.html

dockerfile > ...

```

1
2 FROM nginx:alpine (last pushed 1 month ago)
3 COPY index.html /usr/share/nginx/html/index.html
4

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

| ID             | Service      | Command                 | Time               | Status            | Ports                                   | Image |
|----------------|--------------|-------------------------|--------------------|-------------------|-----------------------------------------|-------|
| 99310af4ab5d   | nginx        | "/docker-entrypoint..." | About a minute ago | Up About a minute | 0.0.0.0:9090->80/tcp, [::]:9090->80/tcp | my    |
| nginx2         | busybox      | "sh"                    | 14 minutes ago     | Up 14 minutes     |                                         | ap    |
| p2             | nginx        | "/docker-entrypoint..." | 14 minutes ago     | Up 14 minutes     | 80/tcp                                  | ap    |
| a6bdc58bf6d    | nginx        | "/docker-entrypoint..." | 18 minutes ago     | Up 18 minutes     | 80/tcp                                  | bu    |
| p1             | nginx        | "/docker-entrypoint..." | 18 minutes ago     | Up 18 minutes     | 80/tcp                                  | bu    |
| al8b6d7c4728   | nginx        | "/docker-entrypoint..." | 18 minutes ago     | Up 18 minutes     | 80/tcp                                  | bu    |
| sy_tu          | nginx        | "/docker-entrypoint..." | 18 minutes ago     | Up 18 minutes     | 80/tcp                                  | bu    |
| fa3a01d5a1c8   | alpine       | "tail -f /dev/null"     | 25 minutes ago     | Up 25 minutes     |                                         | my    |
| container      | nginx:latest | "/docker-entrypoint..." | 4 hours ago        | Up 4 hours        | 80/tcp                                  | di    |
| ca04e759c38b   | nginx:latest | "/docker-entrypoint..." | 4 hours ago        | Up 4 hours        | 80/tcp                                  | di    |
| stracted_black | nginx        | "/docker-entrypoint..." | 4 hours ago        | Up 4 hours        | 0.0.0.0:8080->80/tcp, [::]:8080->80/tcp | ba    |
| 7fd69612396a   | nginx        | "/docker-entrypoint..." | 4 hours ago        | Up 4 hours        | 0.0.0.0:8080->80/tcp, [::]:8080->80/tcp | ba    |
| bor            | nginx        | "/docker-entrypoint..." | 6 days ago         | Up 6 days         | 80/tcp                                  | ap    |
| 0421f09dc10b   | nginx        | "/docker-entrypoint..." | 6 days ago         | Up 6 days         | 80/tcp                                  | ap    |
| p5             | nginx        | "/docker-entrypoint..." | 6 days ago         | Up 6 days         | 80/tcp                                  | ap    |

PS D:\BABOR> docker exec -it mynginx2 bash  
 root@99310af4ab5d:/# cd /usr/share/nginx/html  
 root@99310af4ab5d:/usr/share/nginx/html#  
 root@99310af4ab5d:/usr/share/nginx/html# echo <h1>Hello Broh </h1>>"index.html  
 root@99310af4ab5d:/usr/share/nginx/html# ls  
 50x.html index.html  
 root@99310af4ab5d:/usr/share/nginx/html#

Ln 4, Col 1 Spaces: 4 UTF-8 CRLF Dockerfile Finish Setup Go Live

File Edit Selection View Go Run ... Q BABOR

EXPLORER

- BABOR
  - dockerfile 1
  - index.html

dockerfile > ...

```

1
2 FROM nginx:alpine (last pushed 1 month ago)
3 COPY index.html /usr/share/nginx/html/index.html
4

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS D:\BABOR> docker volume create mydatavol  
 mydatavol  
 PS D:\BABOR> docker volume inspect mydatavol  
 [
 {
 "CreatedAt": "2025-12-02T17:01:00Z",
 "Driver": "local",
 "Labels": null,
 "Mountpoint": "/var/lib/docker/volumes/mydatavol/\_data",
 "Name": "mydatavol",
 "Options": null,
 "Scope": "local"
 }
 ]

PS D:\BABOR> docker run -d --name mynginx -p 8080:80 -v mydatavol:/usr/share/nginx/html nginx  
 17283cb48db4872934394c6255702900f63adb26baf914390e63b6bb0eb65a8  
 docker: Error response from daemon: failed to set up container networking: driver failed programming external connectivity on endpoint myngi  
 nx (8a44fc742eb22d017317d739c9a565b2daaaa92320a330bf6adb46be5abd7cf4): Bind for 0.0.0.0:8080 failed: port is already allocated

Run 'docker run --help' for more information  
 PS D:\BABOR> docker run -d --name mynginx -p 9090:80 -v mydatavol:/usr/share/nginx/html nginx  
 docker: Error response from daemon: Conflict. The container name "/mynginx" is already in use by container "17283cb48db4872934394c6255702900  
 f63adb26baf914390e63b6bb0eb65a8". You have to remove (or rename) that container to be able to reuse that name.

Ln 4, Col 1 Spaces: 4 UTF-8 CRLF Dockerfile Finish Setup Go Live

Visual Studio Code interface showing a Dockerfile and a terminal window. The Dockerfile contains the following content:

```
1
2 FROM nginx:alpine (last pushed 1 month ago)
3 COPY index.html /usr/share/nginx/html/index.html
4
```

The terminal window displays the output of the `docker run` command, showing the container ID, image, command, and status. The output is as follows:

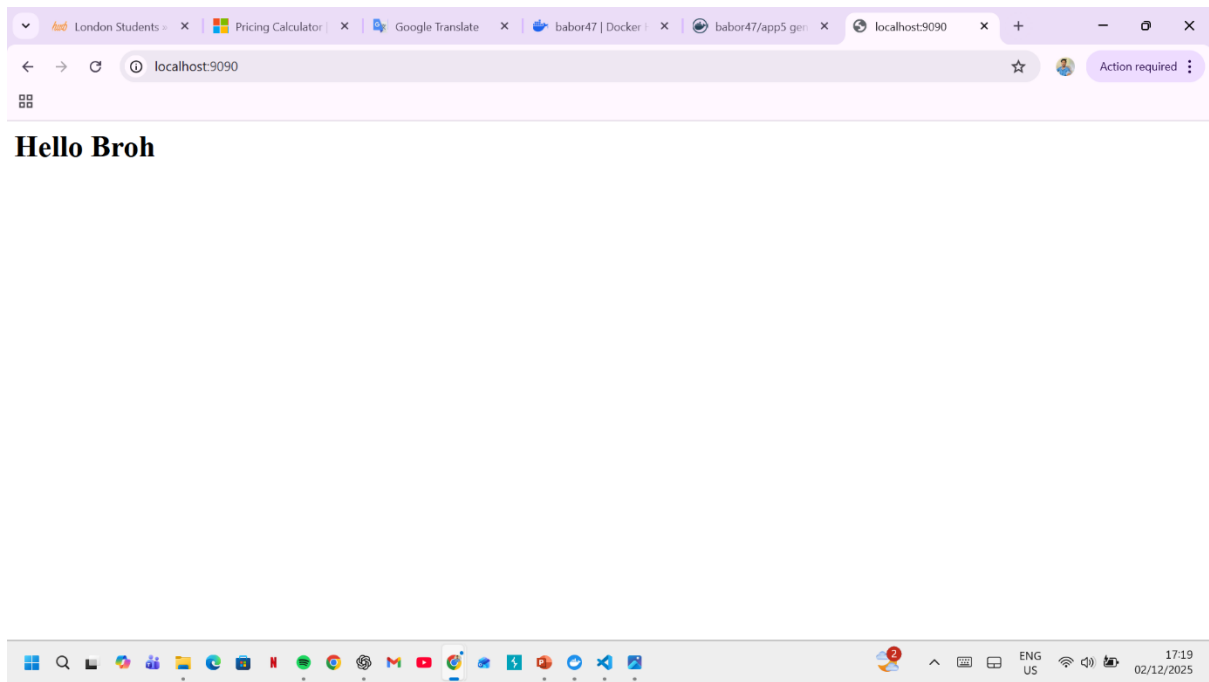
```
PS D:\BABOR> docker run -d --name mynginx2 -p 9090:80 -v mydatavol:/usr/share/nginx/html nginx
99310af4ab5dae48cbd4a7db565f57da33de875b0163c561b0a2f76e25cffe09
PS D:\BABOR> docker ps
```

| CONTAINER ID   | IMAGE        | COMMAND                 | CREATED            | STATUS            | PORTS                                   | NAME |
|----------------|--------------|-------------------------|--------------------|-------------------|-----------------------------------------|------|
| MES            |              |                         |                    |                   |                                         |      |
| 99310af4ab5d   | nginx        | "/docker-entrypoint..." | About a minute ago | Up About a minute | 0.0.0.0:9090->80/tcp                    | my   |
| nginx2         |              |                         |                    |                   |                                         |      |
| 90ff1941f781   | busybox      | "sh"                    | 14 minutes ago     | Up 14 minutes     |                                         | ap   |
| p2             |              |                         |                    |                   |                                         |      |
| a6bdcb58bfd6   | nginx        | "/docker-entrypoint..." | 14 minutes ago     | Up 14 minutes     | 80/tcp                                  | ap   |
| p1             |              |                         |                    |                   |                                         |      |
| alb86d7c4728   | nginx        | "/docker-entrypoint..." | 18 minutes ago     | Up 18 minutes     | 80/tcp                                  | bu   |
| sy_tu          |              |                         |                    |                   |                                         |      |
| fa3a01d5a1c8   | alpine       | "tail -f /dev/null"     | 25 minutes ago     | Up 25 minutes     |                                         | my   |
| container      |              |                         |                    |                   |                                         |      |
| ca04e759c38b   | nginx:latest | "/docker-entrypoint..." | 4 hours ago        | Up 4 hours        | 80/tcp                                  | di   |
| stracted_black |              |                         |                    |                   |                                         |      |
| 7fd69612396a   | nginx        | "/docker-entrypoint..." | 4 hours ago        | Up 4 hours        | 0.0.0.0:8080->80/tcp, [::]:8080->80/tcp | ba   |
| bor            |              |                         |                    |                   |                                         |      |
| 0421f09dc10b   | nginx        | "/docker-entrypoint..." | 6 days ago         | Up 6 days         | 80/tcp                                  | ap   |

The browser window shows the local host `localhost:9090` with a status bar indicating "Action required".

Hello Renuga mam,I am your fav student!

Windows taskbar showing the system clock at 17:20 on 02/12/2025.

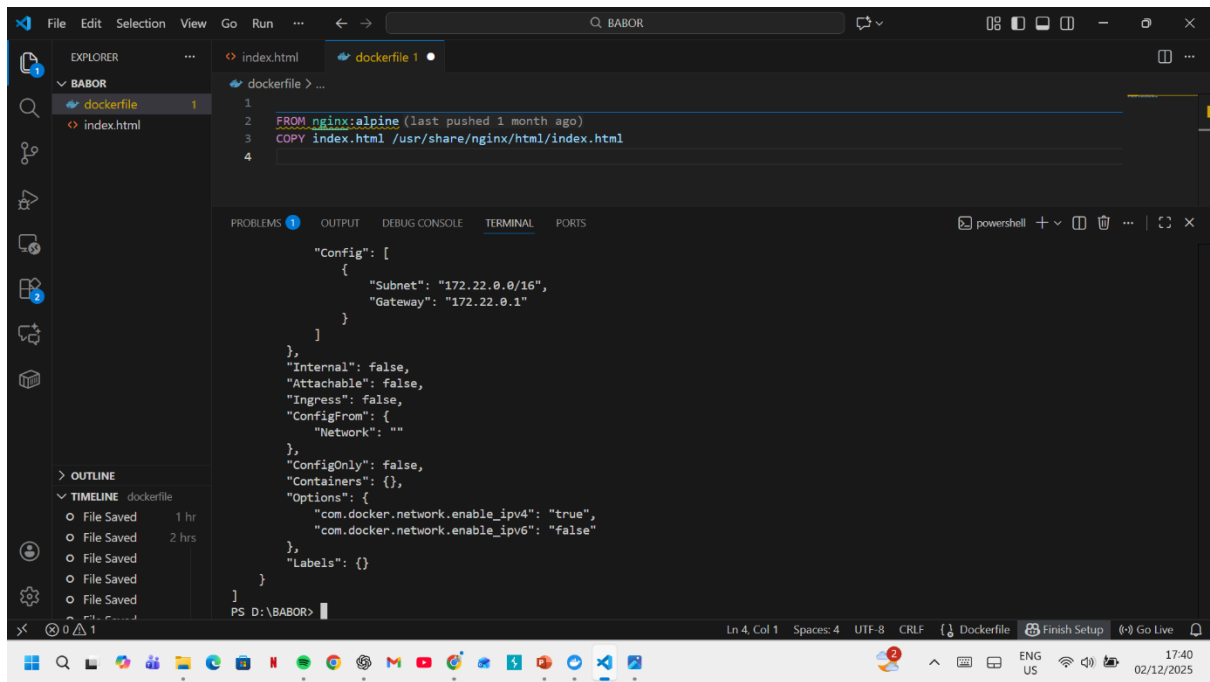


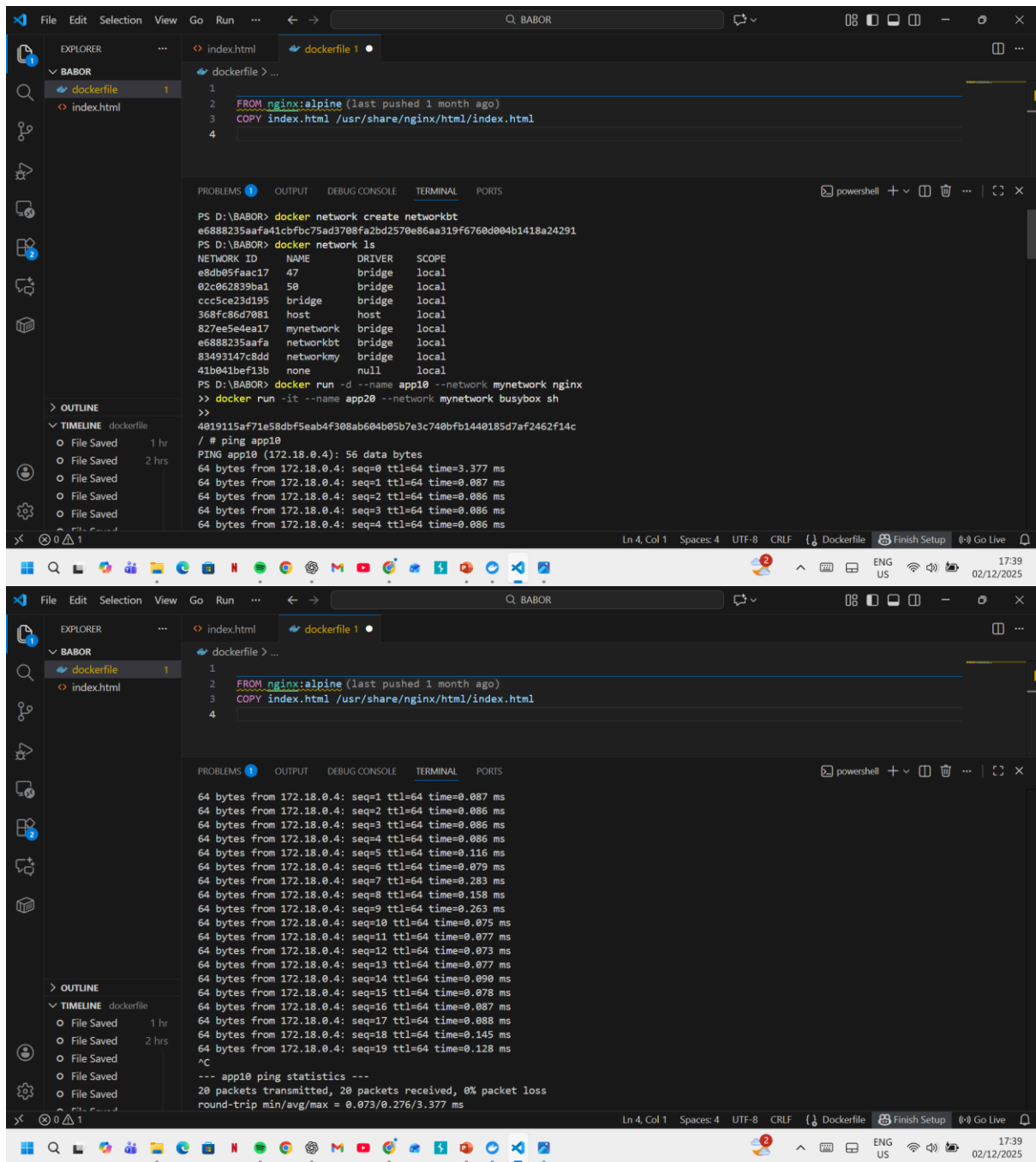
## 5. Definition of Container Networking

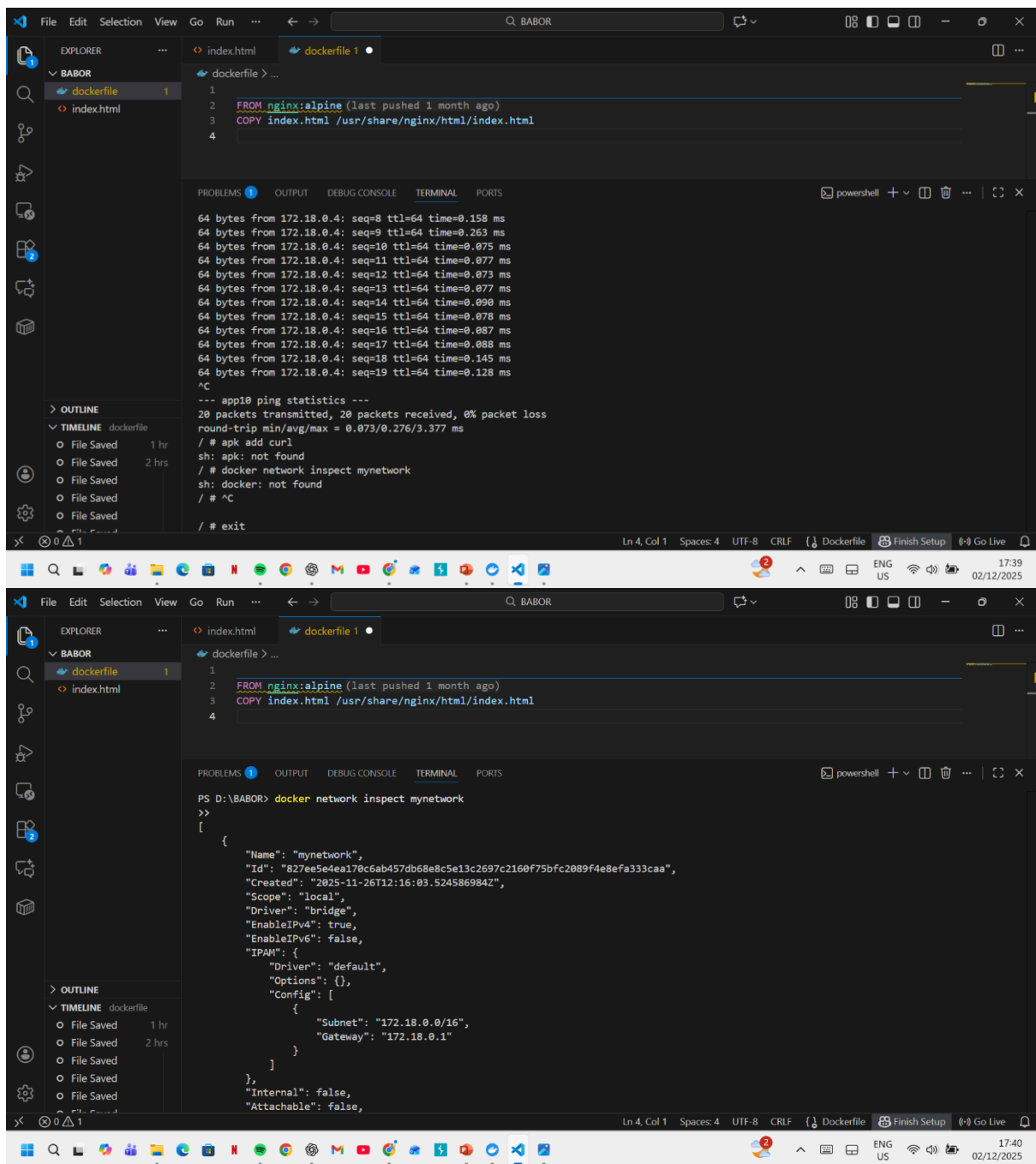
Container networking refers to the set of technologies and mechanisms that allow containers to communicate securely, efficiently, and reliably with each other, as well as with external systems. It enables isolated applications running in separate containers to exchange data through virtual networks, providing connectivity while maintaining secure boundaries [20], [21].

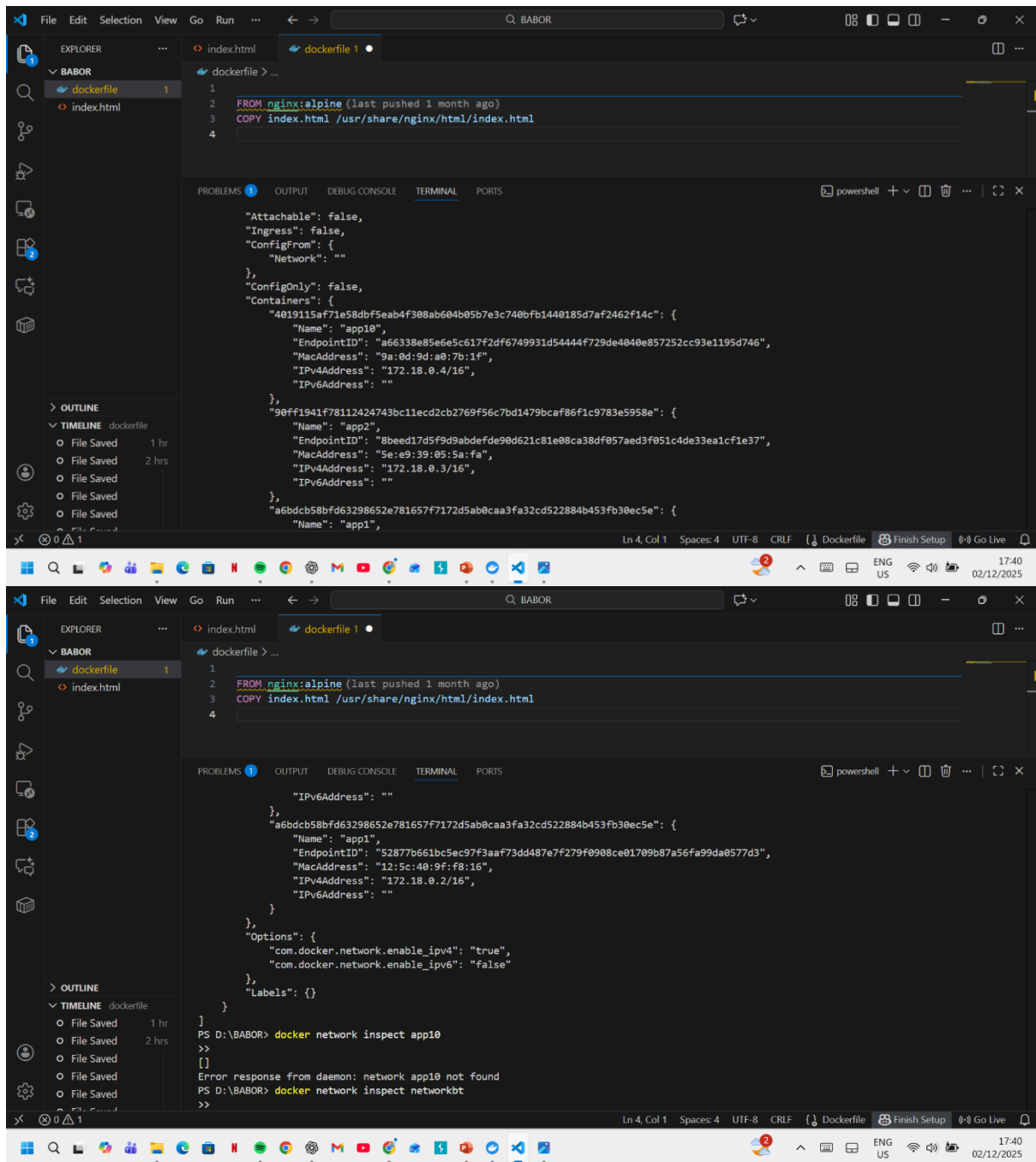
Docker and other container platforms create virtual network interfaces and use software-defined networking (SDN) techniques to manage traffic between containers. Network drivers—such as bridge, host, overlay, and macvlan—enable different communication models depending on deployment requirements [22].

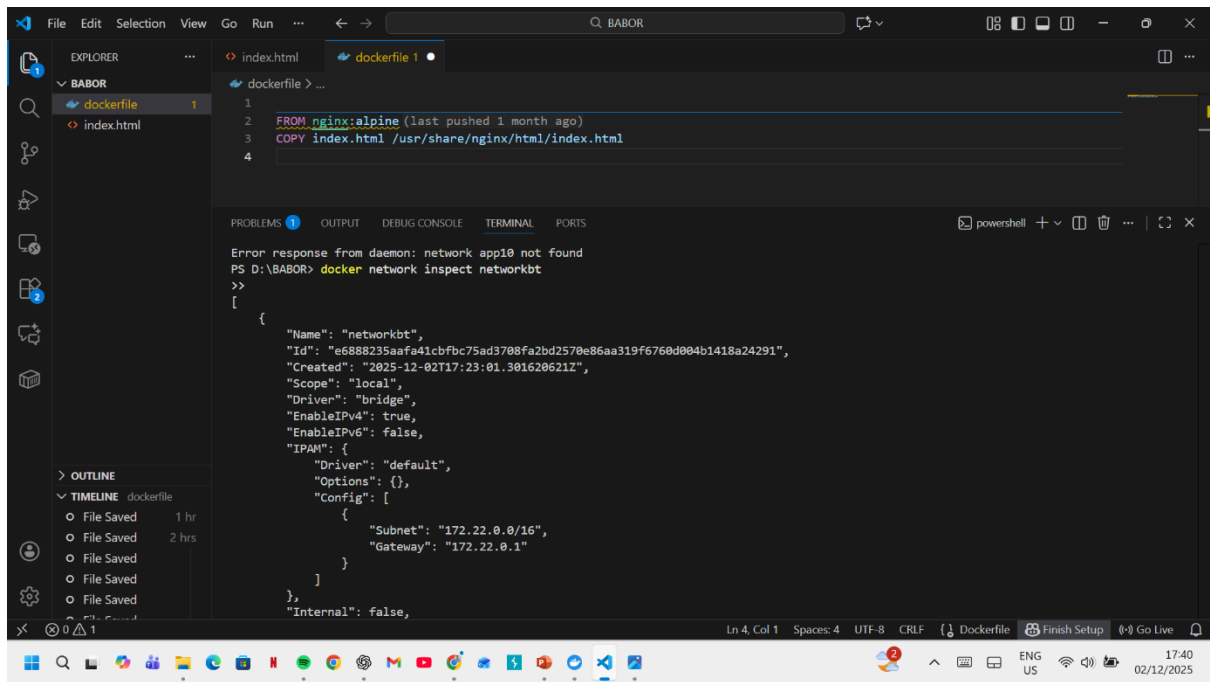
Container networking is essential for distributed applications, microservices architectures, and orchestrated environments such as Kubernetes, where services rely on predictable, stable networking to function correctly [23].











A web server and his HTML page:

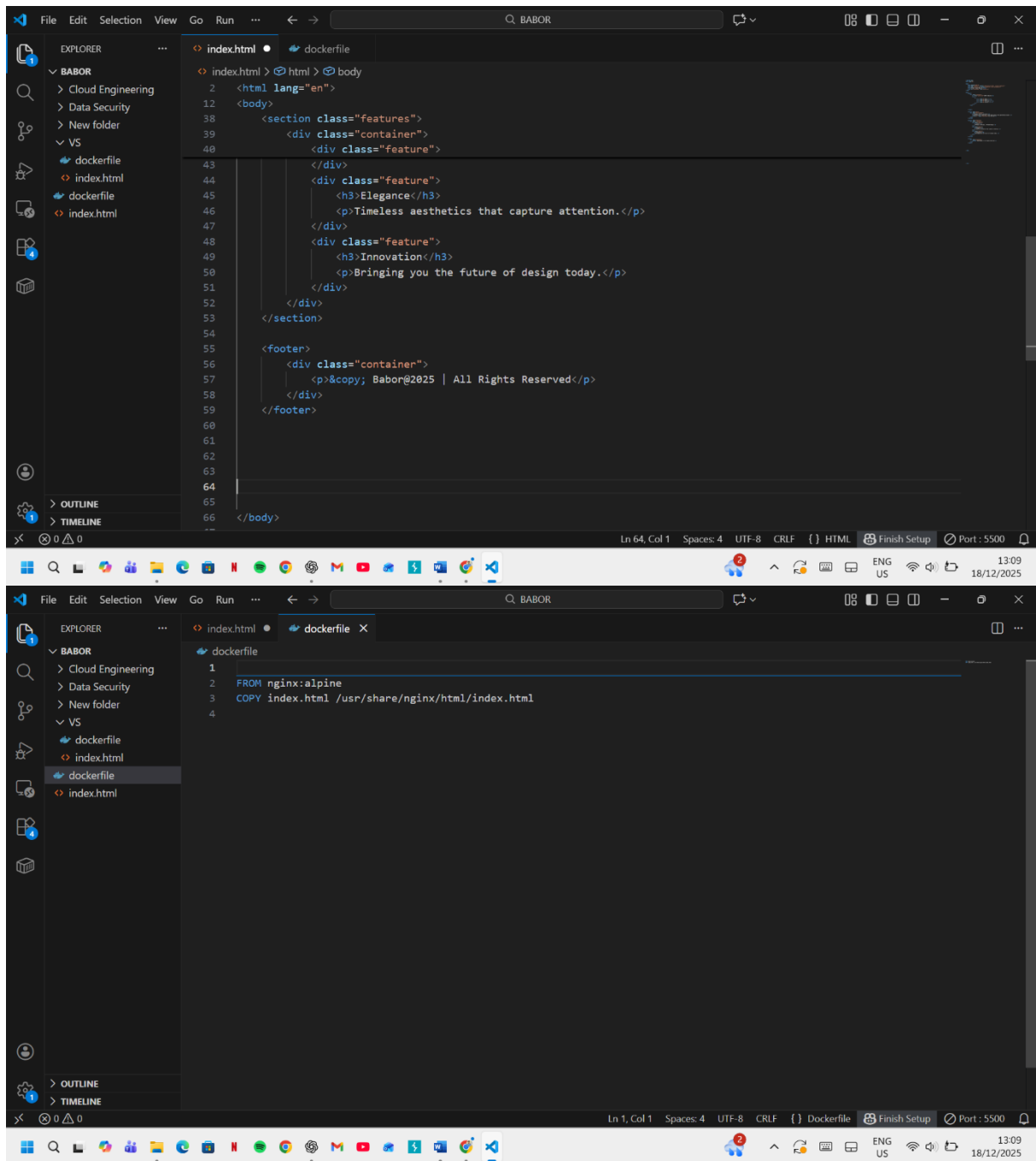
The image displays two screenshots of the Visual Studio Code editor, showing the development of an HTML page. The top screenshot shows the initial structure of the HTML document, including the DOCTYPE, charset, viewport, and title tags. The bottom screenshot shows the completed HTML code, including the hero section and the features section.

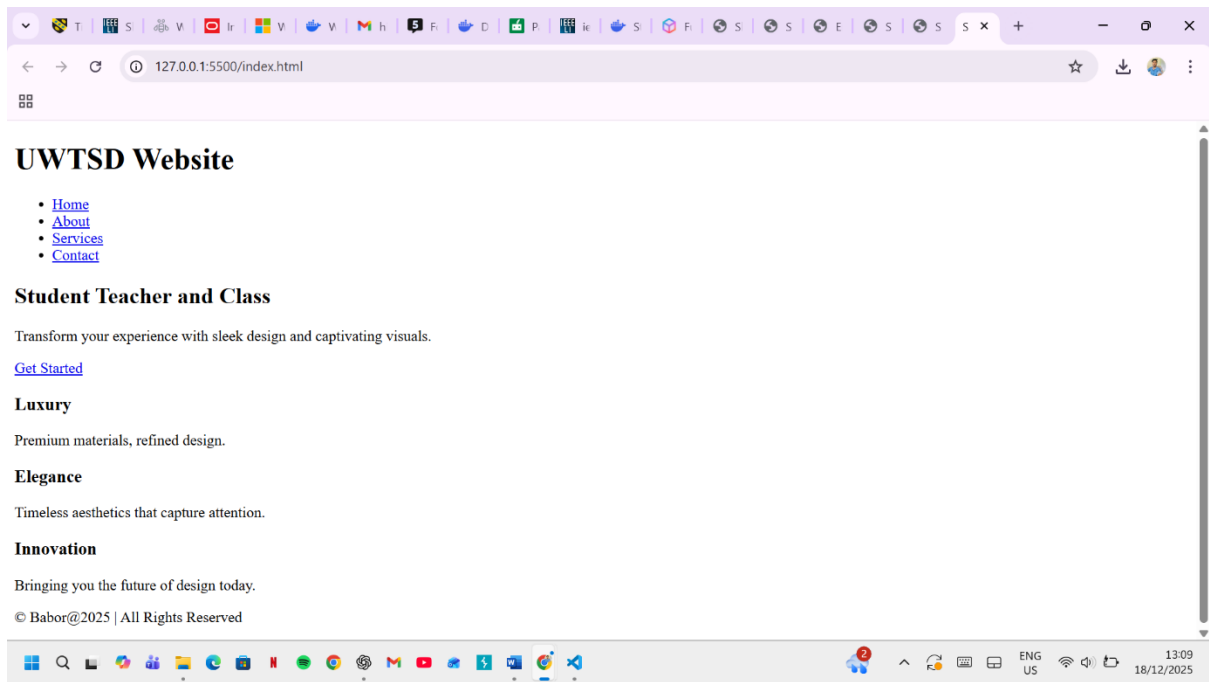
**Top Screenshot (index.html):**

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <meta name="description" content="Beautiful and Sexy Website">
8   <title>Sleek & Sexy Website</title>
9   <link rel="stylesheet" href="style.css">
10 </head>
11
12 <body>
13   <header>
14     <div class="container">
15       <h1 class="site-title">UWTSO Website</h1>
16       <nav>
17         <ul>
18           <li><a href="#">Home</a></li>
19           <li><a href="#">About</a></li>
20           <li><a href="#">Services</a></li>
21           <li><a href="#">Contact</a></li>
22         </ul>
23       </nav>
24     </div>
25   </header>
26
27   <div class="hero">
28     <div class="hero-content">
29       <h2>Student Teacher and Class</h2>
30       <p>Transform your experience with sleek design and captivating visuals.</p>
31       <a href="#" class="cta-button">Get Started</a>
32     </div>
33   </div>
34
35   <div class="features">
36     <div class="container">
37       <div class="feature">
38         <h3>Luxury</h3>
39         <p>Premium materials, refined design.</p>
40       </div>
41       <div class="feature">
42         <h3>Elegance</h3>
43         <p>Timeless aesthetics that capture attention.</p>
44       </div>
45       <div class="feature">
46         <h3>Innovation</h3>
47         <p>Bringing you the future of design today.</p>
48       </div>
49     </div>
50   </div>
51 </body>
52 </html>
```

**Bottom Screenshot (index.html):**

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <meta name="description" content="Beautiful and Sexy Website">
8   <title>Sleek & Sexy Website</title>
9   <link rel="stylesheet" href="style.css">
10 </head>
11
12 <body>
13   <header>
14     <div class="container">
15       <h1 class="site-title">UWTSO Website</h1>
16       <nav>
17         <ul>
18           <li><a href="#">Home</a></li>
19           <li><a href="#">About</a></li>
20           <li><a href="#">Services</a></li>
21           <li><a href="#">Contact</a></li>
22         </ul>
23       </nav>
24     </div>
25   </header>
26
27   <div class="hero">
28     <div class="hero-content">
29       <h2>Student Teacher and Class</h2>
30       <p>Transform your experience with sleek design and captivating visuals.</p>
31       <a href="#" class="cta-button">Get Started</a>
32     </div>
33   </div>
34
35   <div class="features">
36     <div class="container">
37       <div class="feature">
38         <h3>Luxury</h3>
39         <p>Premium materials, refined design.</p>
40       </div>
41       <div class="feature">
42         <h3>Elegance</h3>
43         <p>Timeless aesthetics that capture attention.</p>
44       </div>
45       <div class="feature">
46         <h3>Innovation</h3>
47         <p>Bringing you the future of design today.</p>
48       </div>
49     </div>
50   </div>
51 </body>
52 </html>
```





## **Part: 02**

### Cloud Computing Concepts: Load Balancing, Scalability, and Application Gateways

Application Example Demonstrating All Three Concepts an e-commerce web application hosted on a major cloud provider AWS, Azure, and Google Cloud are excellent examples of how to mix load balancing, scalability, and application gateways.

#### 2.1 Load Balancing

Load balancing is utilized by cloud providers to evenly distribute incoming traffic between many application instances, therefore improved performance and reliability. Cloud Provider Methods:

- AWS Elastic Load Balancer (ELB)
  - a) Automatically distributes HTTP/HTTPS traffic across EC2 instances, containers, or Lambda functions.
  - b) Performs health checks and reroutes traffic from unhealthy instances. Supports either Application Load Balancers (ALB) and Network Load loads (NLBs) [24].
- Azure Load Balancer
  - a) Distributes TCP/UDP traffic using Layer 4 forwarding rules.
  - b) Automatically reroutes traffic around failed instances and supports outbound NAT [25].

Illustrative Step:

Selecting a target group in AWS ALB ensures that only healthy instances receive traffic, demonstrating request filtering and instance monitoring.

#### 2.2 Scalability

Scalability allows the system to expand or contract according to demand.

Cloud Provider Methods:

- AWS Auto Scaling Groups (ASG)
- Auto start or stop EC2 instances using scaling policies prompted by memory, CPU usage, or request metrics.
  - a) Horizontal scaling adds/removes instances; vertical scaling changes instance sizes [26].
- Azure Virtual Machine Scale Sets (VMSS)
  - b) Automatically manages identical VMs and adjusts capacity according to load [27].

Illustrative Step:

Implementing a scale-out policy (e.g., add a new instance if CPU > 70% for 5 minutes) directly demonstrates reactive scalability.

## 2.3 Application Gateways

Application gateways provide Layer 7 routing, security, and intelligent traffic management.

Cloud Provider Methods:

- AWS ALB + AWS WAF
  - a) Routes traffic based on hostname, headers, or paths.
  - b) Can route /payment to one microservice and /images to another.
  - c) Provides protection against SQL injection and XSS [28].
- Azure Application Gateway
  - a) Supports URL-based routing, SSL termination, session affinity, and Web Application Firewall.
  - b) Directs user traffic to different back-end pools using application logic [29].

Illustrative Step:

Configuring path-based routing rules in Azure Application Gateway demonstrates intelligent microservice routing.

## 2. Complementary Nature of Load Balancing and Scalability

Load balancing and scalability work together to create a robust and efficient cloud service:

Load Balancing:

- Distributes traffic evenly to prevent overload.
- Ensures high availability via health checks [24].

Scalability:

- Adds instances to meet increasing demand.
- Maintains performance automatically [26].

Why They Complement Each Other:

- Without scaling, a load balancer distributes traffic over a fixed set of servers, which may still be overloaded.
- Without load balancing, newly scaled instances cannot effectively handle traffic.

Combined Benefit:

Enables higher throughput, resilience, cost optimization, and zero-downtime growth.

### 3. Advantages of Integrating All Three Technologies

- Combining load balancing, flexibility, and software gateways yields:
- High Availability: Only healthy instances receive traffic.
- High Performance: Scaling increases capacity as demand rises.
- Security & Intelligent Routing: Gateways provide WAF protection and smart routing [28].
- Cost Efficiency: Scaling down reduces expenses during low-demand periods.
- Smooth User Experience: End-users receive consistent response times even during peak loads.

This combination provides a secure, resilient, and scalable cloud application.

### 4. Motivators for the Growth of Cloud Computing (Cost, Flexibility, etc.)

#### 4.1 Cost-Effectiveness

- Pay-as-you-go pricing reduces capital expenses.
- Eliminates the need to maintain physical servers.
- Auto-scaling lowers costs during low traffic periods [30].

#### 4.2 Flexibility & Rapid Deployment

- Applications can be deployed globally within minutes.
- Managed services (databases, AI, analytics) accelerate development.
- Developers can focus on innovation rather than hardware [31].

#### 4.3 Disaster Recovery & Reliability

- Multi-zone redundancy ensures reliability.
- Automatic failover and backup systems protect against data loss [30].

#### 4.4 Global Reach

- Distributed data centre ensure low-latency access worldwide [31].

### 5. Containerization versus Serverless Functions

Serverless computing platforms, such as AWS Lambda, Azure Functions, and Google Cloud Functions, offer a new paradigm for cloud deployment.

#### 5.1 Serverless Functions

- Code executes only in response to events (HTTP requests, file uploads, scheduled tasks).
- Cloud provider manages scaling, load balancing, and high availability.
- Billing is based on execution time and number of requests [32].

Example: A Lambda function triggers only when a file is uploaded to Amazon S3.

## 5.2 Containerization

- - a) Packages software and components into portable boxes (e.g., Docker, Kubernetes).
  - b) Offers total control over the OS and environment.
- Well-suited for microservices architectures.
- Requires orchestration tools like Kubernetes, ECS, or AKS [33].

## 5.3 Comparison of Driving Forces

| Factor                     | Serverless                               | Containers                                     |
|----------------------------|------------------------------------------|------------------------------------------------|
| <b>Cost</b>                | Lower for intermittent workloads [32]    | Better for stable, long-running workloads [33] |
| <b>Flexibility</b>         | Limited runtime control, short execution | Full control, highly customizable              |
| <b>Scalability</b>         | Automatic, instantaneous                 | Requires orchestration                         |
| <b>Management Overhead</b> | Minimal (no servers)                     | Moderate to high (clusters, networking)        |
| <b>Performance</b>         | Cold-start delays possible               | Consistent, predictable                        |
| <b>Use Cases</b>           | Event-driven tasks, APIs, automation     | Long-running tasks, complex microservices      |

## **6. Conclusion**

Docker and containerization are crucial to contemporary cloud computing because they make it possible to deploy apps in an innovative, scalable, and portable manner. Businesses could offer reliable and trustworthy services to clients when they integrate cloud technologies like load sharing, scalability, and application gateways. These methods raise performance, lower operating costs, while encouraging quick cloud innovation.

## References

1. [1] IEEE, *Virtualisation*. Available: [https://www.ieee.li/pdf/viewgraphs/introduction\\_to\\_virtualization.pdf](https://www.ieee.li/pdf/viewgraphs/introduction_to_virtualization.pdf)
2. [2] G. J. Popek and R. P. Goldberg, "Formal Requirements for Virtualizable Third Generation Architectures," *Communications of the ACM*, vol. 17, no. 7, pp. 412–421, 1974.
3. [3] A. Kivity et al., "kvm: The Linux Virtual Machine Monitor," in *Proceedings of the Linux Symposium*, Ottawa, 2007.
4. [4] GeeksforGeeks, *Characteristics of Virtualization*. Available: <https://www.geeksforgeeks.org/characteristics-of-virtualization/>
5. [5] IBM, "What is Virtualization?" IBM Documentation, 2024. [Online]. Available: <https://www.ibm.com/topics/virtualization>
6. [6] Oracle, "Introduction to Virtualization," Oracle Documentation, 2023. [Online]. Available: [https://docs.oracle.com/cd/E20065\\_01/doc.30/e18549/intro.htm](https://docs.oracle.com/cd/E20065_01/doc.30/e18549/intro.htm)
7. [7] Amazon Web Services, Inc. (2025). AWS Documentation – Elastic Load Balancing, Auto Scaling, API Gateway. Available at: <https://aws.amazon.com/documentation/> (Accessed: 11 December 2025).
8. [8] Docker Inc., *What is a Container?*, Docker Documentation, 2024. [Online]. Available: <https://www.docker.com/resources/what-container/>
9. [9] Backblaze. (2023). 'Virtual Machines vs. Containers: A Closer Look'. Available at: <https://www.backblaze.com/blog/vm-vs-containers/> (Accessed: 11 December 2025).
10. [10] Kubernetes Authors, *Understanding Container Images*, Kubernetes Documentation, 2024. [Online]. Available: <https://kubernetes.io/docs/concepts/containers/images/>
11. [11] AWS, *Containers on AWS*, Amazon Web Services Documentation, 2024. [Online]. Available: <https://aws.amazon.com/containers/>
12. [12] Fiveable. (2025). 'Cloud Computing Architecture Layers'. Available at: <https://library.fiveable.me/> (Accessed: 11 December 2025).
13. [13] Docker Inc., *Docker Images Overview*, Docker Documentation, 2024. <https://docs.docker.com/get-started/docker-overview/#docker-images>
14. [14] Red Hat, *What is a Container Image?*, Red Hat Documentation, 2024. [Online]. <https://www.redhat.com/en/search?search=container%20images>
15. [15] Docker, Inc. (2025). Docker Documentation. Available at: <https://docs.docker.com/> (Accessed: 11 December 2025).
16. [16] Amazon Web Services, *Amazon Elastic Container Registry (ECR)*, AWS Documentation, 2024. [Online]. Available: <https://aws.amazon.com/ecr/>
17. [17] Docker Inc., *Manage Data in Docker*, Docker Documentation, 2024. [Online]. Available: <https://docs.docker.com/storage/>
18. [18] Net Solutions. (2025). 'Containers vs. Virtual Machines: Choosing the Right Virtualization Technology'. Available at: <https://www.netsolutions.com/insights/containers-vs-virtual-machines/> (Accessed: 11 December 2025).
19. [19] Kubernetes Authors, *Persistent Storage for Containers*, Kubernetes Documentation, 2024. [Online]. Available: <https://kubernetes.io/docs/concepts/storage/>
20. [20] Docker Inc., *Networking Overview*, Docker Documentation, 2024. [Online]. Available: <https://docs.docker.com/network/>

22. [21] Red Hat, *Container Networking Explained*, Red Hat Documentation, 2024. [Online]. Available: <https://www.redhat.com/en/topics/containers/container-networking>
23. Docker, Inc. (2025). Docker Documentation. Available at: <https://docs.docker.com/> (Accessed: 11 December 2025).
24. Kubernetes Authors, *Cluster Networking*, Kubernetes Documentation, 2024. [Online]. Available: <https://kubernetes.io/docs/concepts/cluster-administration/networking/>
25. Amazon Web Services, "Elastic Load Balancing," AWS Documentation, 2024. [Online]. Available: <https://docs.aws.amazon.com/elasticloadbalancing/latest/userguide/what-is-load-balancing.html>
26. Microsoft Azure, "Azure Load Balancer Overview," Azure Docs, 2024. [Online]. Available: <https://learn.microsoft.com/en-us/azure/load-balancer/load-balancer-overview>
27. Amazon Web Services, "Amazon EC2 Auto Scaling," AWS Documentation, 2024. [Online]. Available: <https://docs.aws.amazon.com/autoscaling/ec2/userguide/what-is-amazon-ec2-auto-scaling.html>
28. Microsoft Azure, "Virtual Machine Scale Sets Overview," Azure Docs, 2024. [Online]. Available: <https://learn.microsoft.com/en-us/azure/virtual-machine-scale-sets/overview>
29. Amazon Web Services, "AWS WAF and ALB Overview," AWS Docs, 2024. [Online]. Available: <https://docs.aws.amazon.com/waf/latest/developerguide/waf-chapter.html>
30. ResearchGate. (Various authors, 2023–2025). Academic diagrams and papers on virtualization, containerization, and cloud evolution. Available at: <https://www.researchgate.net/> (Accessed: 11 December 2025).
31. Gartner, "Cloud Cost Efficiency and Adoption Drivers," Gartner Report, 2023.
32. Google Cloud, "Global Cloud Infrastructure Overview," Google Cloud Docs, 2024. [Online]. Available: <https://cloud.google.com/about/locations>
33. Amazon Web Services, "AWS Lambda Developer Guide," AWS Documentation, 2024. [Online]. Available: <https://docs.aws.amazon.com/lambda/latest/dg/welcome.html>
34. Kubernetes Authors, "Kubernetes Architecture and Concepts," CNCF Documentation, 2024. [Online]. Available: <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>