



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

ACCA7027_LON ASSESSMENT

**Cloud Engineering Principles and
Practice**

Habibur Rahman | 2428138

To
Renuga Jayakumar

Submission Date: 18/12/2025

Table of Contents

List of Tables	4
Table of Figures and Diagrams	4
1. Abstract	5
2. Introduction	5
3. Literature Review	5
4. Virtualization	5
4.1. Physical machine (server/computer)	6
4.2. Virtual machine (VMs)	6
4.3. Hypervisor	6
4.3.1. Type-1 or native or bare-metal hypervisors	6
4.3.2. Type-2 or hosted hypervisors	6
5. Containerization	7
6. Docker	7
6.1. Docker Architecture	8
6.2. Docker Daemon	8
6.3. Docker Client	8
6.4. Docker registries	8
6.5. Docker Objects	9
6.5.1. Docker Images	9
6.5.2. Docker Containers	9
6.5.3. Docker Volumes	9
6.5.4. Docker Networks	9
7. A Practical Demonstration: From Dockerfile to Docker Hub	9
7.1. Project Implementation Video (YouTube)	20
8. Cloud Computing Concepts	20
8.1.1. Application Selection: Brave Browser	20
8.1.2. Load Balancing in Brave Browser	20
8.1.3. Scalability in Brave Browser	20
8.1.4. Application Gateway in Brave Browser	20
8.2. Relationship Between Load Balancing and Scalability	21
8.3. Benefits of Combining Load Balancing, Scalability, and Application Gateways	21
8.4. Driving Forces Behind Cloud Computing Expansion	21

8.4.1.	Cost Efficiency	21
8.4.2.	Scalability and Flexibility	21
8.4.3.	Accessibility and Global Reach	21
8.4.4.	Maintenance and Operational Simplification	21
8.4.5.	Innovation Enablement	22
8.5.	Serverless Functions (Functions as a Service)	22
8.5.1.	Characteristics	22
8.5.2.	Benefits	22
8.6.	Serverless Functions vs Containerization	22
9.	Conclusion	22
10.	References	23

List of Tables

Table 1: Comparison between Virtual Machines (VMs) and Containers	7
Table 2: Comparison between serverless and containerization	22

Table of Figures and Diagrams

Figure 1: Types of Hypervisors	7
Figure 2: Docker Architecture	8
Figure 3: Dockerfile	10
Figure 4: First page HTML source code view of the index.html file (.html extension)	11
Figure 5: HTML source code view of the second page, thankyou.html	11
Figure 6: Building a Docker image	12
Figure 7: Running the Docker container using the 'docker run' command	13
Figure 8: Listing running Docker containers using 'docker ps'	13
Figure 9: Pushing a Docker image to Docker Hub	14
Figure 10: Listing Docker images in the Docker Hub account	15
Figure 11: Accessing HTML files through Docker-mapped port.	15
Figure 12: First HTML page view (index.html)	16
Figure 13: Second HTML page view (thankyou.html)	17
Figure 14: Docker Volume command	17
Figure 15: Docker Network command	18
Figure 16: Output of 'docker network inspect' command - 1	19
Figure 17: Output of 'docker network inspect' command - 2	19
Figure 18: Output of 'docker network inspect' command - 3	19

1. Abstract

This review paper provides a very informative review of critical technologies underlying modern cloud-based systems: virtualization, containerization, Docker, and load balancing. The discussion covers their principles, practical use, and roles in enhancing the scalability, security, and performance of the system. By reviewing related literature and industry practices, this paper presents evidence showing how these technologies contribute to good resource utilization, flexibility in deployment, and high availability in distributed environments.

2. Introduction

Privacy also is one of the basic human rights used in computer ethics [1] . Personal privacy is then the ability of individuals to control their personal information and resist unwanted attention and mishandling [1] . Privacy has emerged as an important issue due to the abundance of personal information collected, stored, and analyzed across the globe in digital form [1]. Because of this, a system of ethical principles and guidelines has been developed to ensure the protection of individual privacy rights [1]. On the other hand, Cloud computing is a key shift in the levels of computing [2]. The "cloud" networked computers and servers available over the Internet to share resources, instead of relying on local desktop or laptop computers for applications [2]. Cloud computing allows access to apps and data anywhere in the world, promoting worldwide teamwork [2] . Cloud computing vendors, such as Amazon, Google, Salesforce.com, IBM, Microsoft, and Sun Microsystems (since acquired by Oracle Corporation), are building out redundancy in their data centers across the world to reduce instances of site failure [2]. Flexibility is a requirement of cloud service providers to serve various users while protecting user traffic from the infrastructure [2]. These factors make it crucial to comprehend the ideas behind cloud storage and its function in contemporary computing [2]. In today's digital world, this information serves as the foundation for investigating related ideas that facilitate effective data management and application deployment [2].

3. Literature Review

In this paper, the discussion will be about virtualization, containerization, Docker, and Load balancing. Examining their roles and their impact on scalability, security, and performance.

4. Virtualization

Virtualization is a method that allows you to create several virtual machines (VMs) on a single physical hardware platform, each of which can run its own operating system [2] . The essential principle of cloud computing is the abstraction of physical resources, which allows for efficient resource use, scalability, and task separation [2] . Virtualization has several advantages, including cheaper costs and faster server deployment to handle shifting online service demand [2]. It is an essential component of cloud computing because of these advantages [2]. However, there are limitations to virtualization, particularly in multi-tenant environments where virtual PCs share hardware [2] . According to [3] Virtualization has various advantages for both on-premises and cloud-based data centers that support IT operations, including:

- Resource efficiency
- Easier management
- Minimal downtime
- Faster provisioning
- Disaster recovery (DR)

- Cost-effectiveness

Also, in [3], It shows that the main components of virtualization are:

- Physical machine (server/computer)
- Virtual machine (VMs)
- Hypervisor

4.1. Physical machine (server/computer)

The physical machine, also called a 'host machine', is the server or computer that provides CPU, memory, storage, and network resources to the virtual machines [3].

4.2. Virtual machine (VMs)

A virtual machine (VM) is software like a physical computing device [3]. VMs are commonly called 'guests'; one or more "guest" machines can run on a host machine [3]. Virtual machines are typically composed of multiple files, which include a configuration file as well as the storage for its virtual hard drive and anything else it relies on [3]. System resources are shared among virtual machines and, as a result, the technology provides scalability on demand as well as cost savings in terms of efficiency [3].

4.3. Hypervisor

A hypervisor, also known as a Virtual Machine Monitor (VMM), is a software layer that virtualizes a physical machine's resources and allows for the operation of numerous virtual machines (VMs) [4]. It is a connection point between the VM and the actual physical hardware, ensuring all of them have access to the physical resources that they need to run [3]. It also ensures the VMs don't step on each other in terms of memory space or compute cycles [3]. According to, [3],[4],[5] There are two types of VMMs:

1. Type-1 or native or bare-metal hypervisors
2. Type-2 or hosted hypervisors

4.3.1. Type-1 or native or bare-metal hypervisors

Type 1 or "bare-metal" hypervisors have direct access to physical hardware, without having to depend on a host OS. They are also seen in Virtual Server situations where a virtual server is created by splitting up a physical machine into several self-contained pieces, each capable of running its own operating system and applications [3]. For example: VMware ESXi, Microsoft Hyper-V, Xen Server, KVM. Also, as shown in Figure 1: Types of Hypervisors [5]

4.3.2. Type-2 or hosted hypervisors

Type 2 hypervisors operate as a software program on an existing operating system. Mostly used on end hosts where they host guest OS instances, they have a performance overhead, as access and coordination of the underlying hardware is done through the host OS [3]. For example: VMware Workstation, Oracle VirtualBox, Parallels Desktop. Also, as shown in Figure 1 [5]

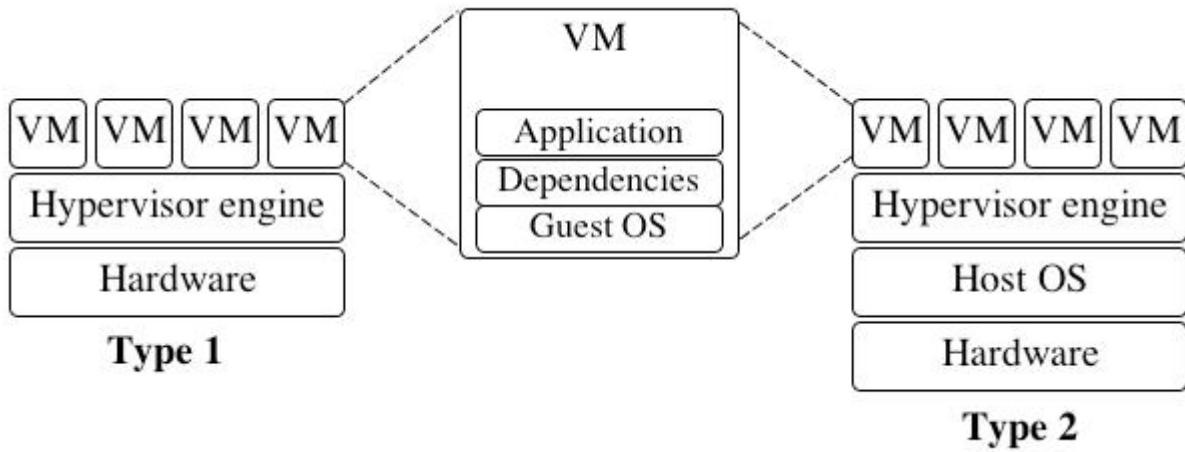


Figure 1: Types of Hypervisors

5. Containerization

A container is a lightweight operating system that runs native instructions on the host CPU, removing the requirement for instruction-level emulation or just-in-time compilation [6]. Containers reduce resource use without the burden of virtualization, while still offering isolation [6]. According to [6], the comparison between Virtual Machines (VMs) and Containers is given below: Table 1

Features	Virtual Machines (VMs)	Containers
Guest OS	Full separate OS per VM	Share the host OS kernel
Isolation	Strong (hardware-level)	Process-level (Weaker)
Performance	Slower (Hypervisor overhead)	Near-Native (faster)
Startup Time	Minutes	Seconds
Storage	Large (entire OS + apps)	Small (app + dependencies)
Security	Strong (hardware isolation)	Depends on host OS controls
Communication	By virtual network, such as Ethernet	IPC like pipes, sockets, signals

Table 1: Comparison between Virtual Machines (VMs) and Containers.

6. Docker

Containerization is a method for packaging a program and its dependencies, including system libraries, inside a container [7]. The apps, which are built and organized, can be run and deployed as containers. That platform is called Docker, and it guarantees that the application will run as intended on any infrastructure [7]. Docker is an open platform for creating, delivering, and operating applications [8]. Users may separate apps from the underlying infrastructure and control the infrastructure as if it were an application [8]. In a containerized environment in which the programs are executed and virtualized, Docker puts another deployment engine on top of it [7]. Docker is used to create a minimal running environment for the code [7]. The following discussion addresses Docker architecture, the Docker daemon, and Docker objects.

6.1. Docker Architecture

As reported in [8] It says Docker's architecture is client-server. The Docker Daemon, which interacts with the Docker client, constructs your Docker container (and they run as well as get disseminated across them). The Docker client and daemon communicate on a socket-based based, which could connect to a remote Docker daemon or work on the local machine. The Docker client can communicate with the daemon through a network interface, UNIX Domain sockets, or a REST API. Docker Compose is another Docker client used when you are working with applications (that consist of multiple containers). Which is shown on Figure 2

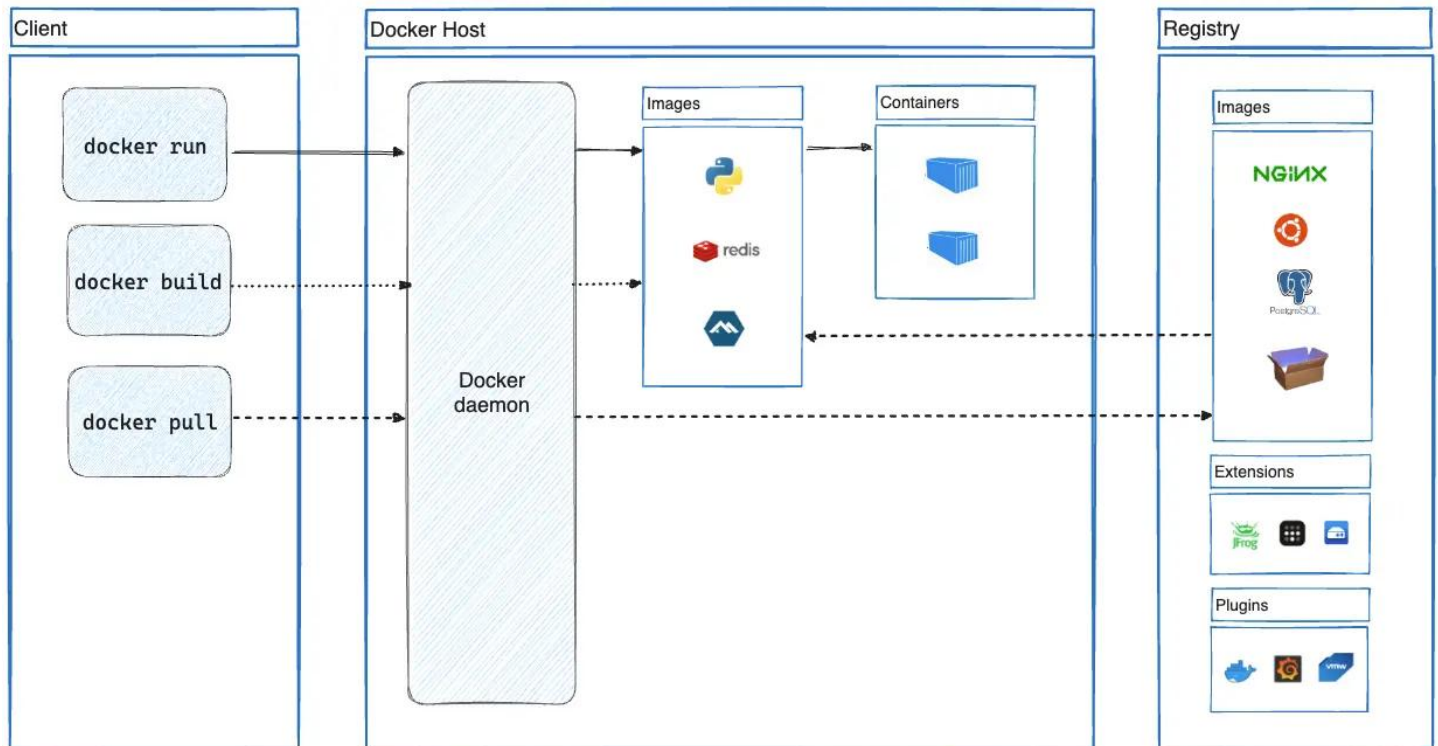


Figure 2: Docker Architecture

6.2. Docker Daemon

The Docker daemon (`'dockerd'` command) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes [7][8]. A daemon is also able to communicate with other daemons to manage Docker services [8].

6.3. Docker Client

Docker clients are the most obvious way that many Docker users interact with Docker. Every time after typing a command like `'docker run'`, the CLI sends that command to `'dockerd'` for processing. The Docker command uses Docker's API. So multiple daemons can be used from the Docker client [8].

6.4. Docker registries

Docker registry is a place to store images. Docker Hub is a public registry that anyone can use, and Docker looks for images on it by default. Anyone can run their own private register [8] . After running the `'docker pull'` or `'docker run'` command, these images are pulled from the personal registry. Along with the `'docker push'` command, Docker pushes the image to the registry that a user can specify [8].

6.5. Docker Objects

Docker enables the creation and management of images, containers, networks, and volumes. This chapter offers only a brief version of a selection of those items.

6.5.1. Docker Images

As outlined in [8] , A Docker image functions as a read-only template for container creation. It is always built by expanding a base image with changes, such as adding the Apache web server and application code to an Ubuntu image, in addition to the settings. The user can upload their own image and build a Dockerfile with a straightforward syntax for specifying the procedures required to generate and operate the image. Each instruction in a Dockerfile adds a layer to the image. After updating the Dockerfile and rebuilding the image, only the layers that have changed are rebuilt. This is part of the reason why images are lightweight, small, and fast.

6.5.2. Docker Containers

According to [7] , Docker images generate Docker containers. To operate the program in a constrained space, the container should hold all necessary kits. Container images can be produced based on the application or software's service requirements. Assume Dockerfile should include an application that includes the Ubuntu operating system and the Nginx web server. Using the command "docker run," a container with the Ubuntu OS image and Nginx server is built and started.

6.5.3. Docker Volumes

As discussed in [8] , Docker generates and manages volumes, which are permanent data repositories for containers. A volume can be created directly using the 'docker volume create' command, or it can be created by Docker during container or service creation. After building a volume, it is saved in a directory on the Docker host. When the volume is mounted inside a container, this directory is used. This is like how bind mounts operate, except that volumes are handled by Docker and segregated from the host machine's fundamental functionality.

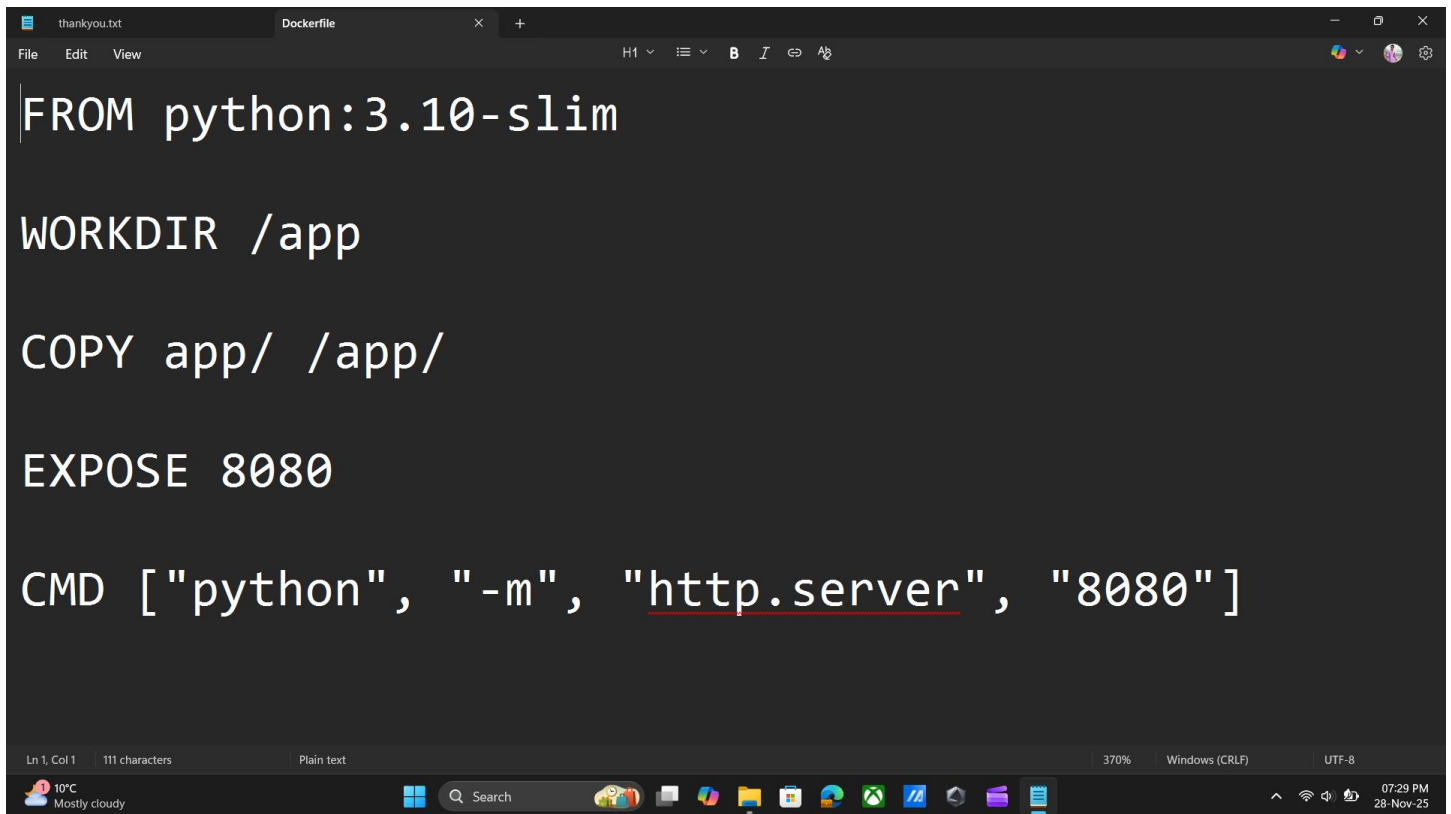
6.5.4. Docker Networks

As reported in [8] , Container networking refers to a container's ability to connect to and communicate with other containers as well as non-Docker network services. By default, containers support networking and may establish outbound connections. A container has no idea what type of network it is connected to or if its network neighbors are likewise Docker containers. A container only sees the network interface, which contains an IP address, a gateway, a routing table, DNS services, and other networking information.

7. A Practical Demonstration: From Dockerfile to Docker Hub

As an example, a complete procedure is demonstrated, illustrating the stages of pushing a web server to display a custom HTML page. The process illustrates building an image, running a container, creating a network, and pushing the resulting output to Docker Hub.

With the help of [8] , The setup procedure begins with PowerShell's 'wsl' command, which enables virtualization functionality. Following this, Docker Desktop is installed. Once the environment is set up, a new Dockerfile (without a file extension) and the required HTML pages are produced in a project directory.



```
FROM python:3.10-slim

WORKDIR /app

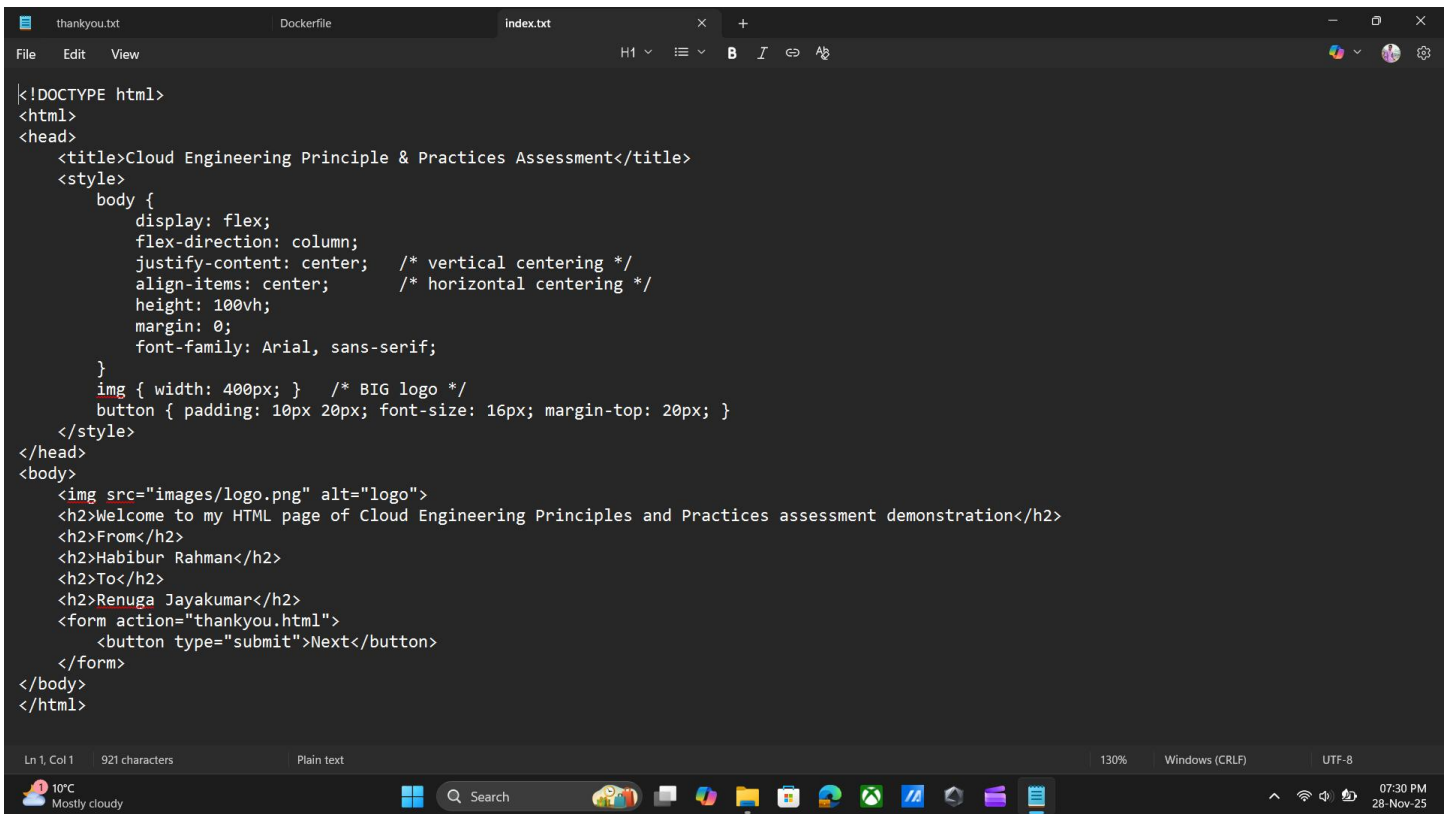
COPY app/ /app/

EXPOSE 8080

CMD ["python", "-m", "http.server", "8080"]
```

Figure 3: Dockerfile

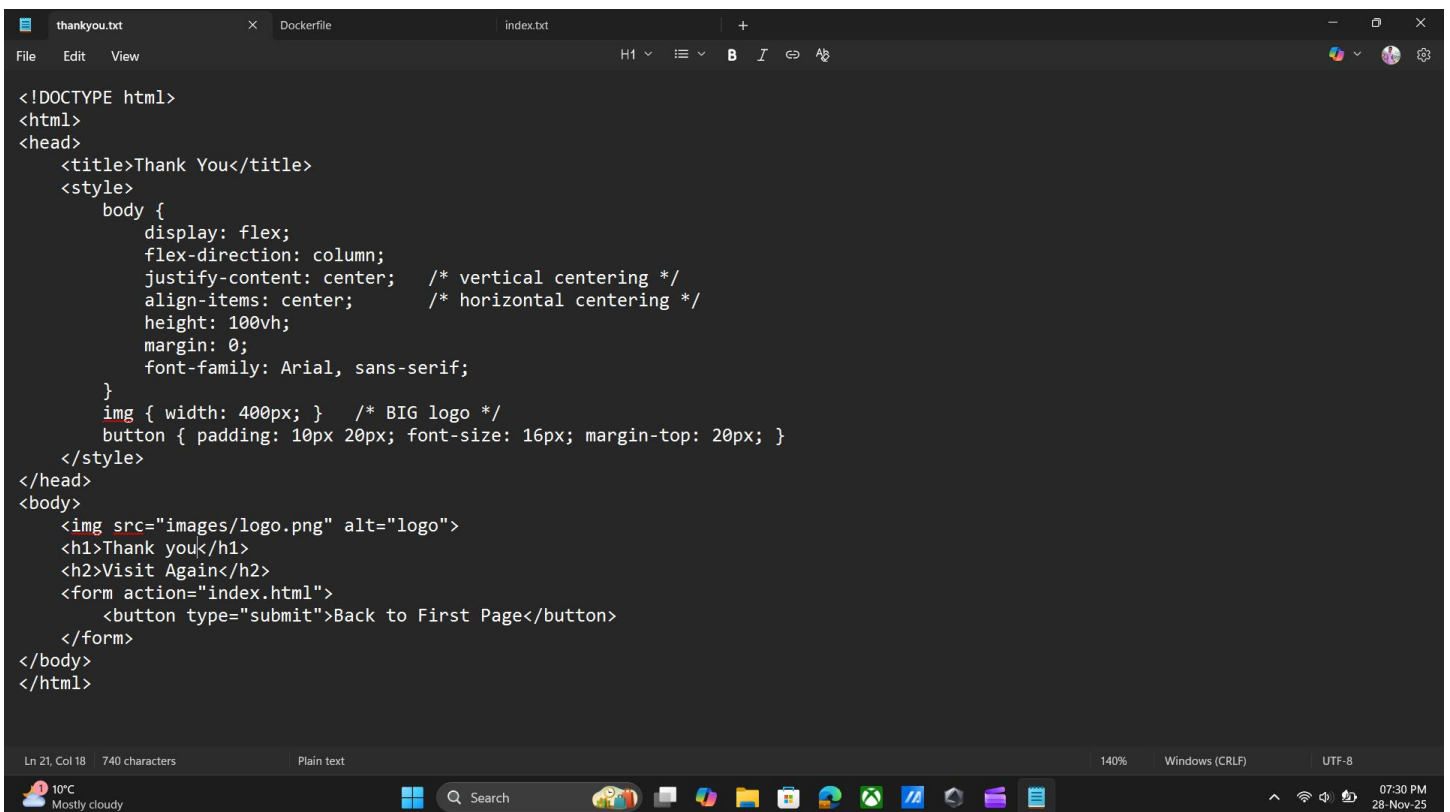
To demonstrate Figure 3, The command 'FROM python:3.10-slim' tells Docker to use the official Python 3.10 as the base for building the new image. The command 'WORKDIR /app' sets /app as a working directory inside the container. The command 'COPY app/ /app/' copies the contents of the local app/ directory into the container's /app/ directory. The 'EXPOSE 8080' command informs Docker that the container listens on network port 8080 at runtime. This declaration documents the port on which the application (in this case, a Python HTTP server) is configured to run. The 'CMD ["python", "-m", "http.server", "8080"]' command tells the default command to run when the container starts. It directs Python to start its built-in HTTP server module, which serves files from the current working directory and listens on port 8080.



```
<!DOCTYPE html>
<html>
<head>
  <title>Cloud Engineering Principle & Practices Assessment</title>
  <style>
    body {
      display: flex;
      flex-direction: column;
      justify-content: center; /* vertical centering */
      align-items: center; /* horizontal centering */
      height: 100vh;
      margin: 0;
      font-family: Arial, sans-serif;
    }
    img { width: 400px; } /* BIG logo */
    button { padding: 10px 20px; font-size: 16px; margin-top: 20px; }
  </style>
</head>
<body>
  
  <h2>Welcome to my HTML page of Cloud Engineering Principles and Practices assessment demonstration</h2>
  <h2>From</h2>
  <h2>Habibur Rahman</h2>
  <h2>To</h2>
  <h2>Renuga Jayakumar</h2>
  <form action="thankyou.html">
    <button type="submit">Next</button>
  </form>
</body>
</html>
```

Figure 4: First page HTML source code view of the index.html file (.html extension).

Figure 4, shows the HTML source code of index.html, the main page of the website. It highlights the basic HTML structure, including the head section and body content used to build and display the web page.



```
<!DOCTYPE html>
<html>
<head>
  <title>Thank You</title>
  <style>
    body {
      display: flex;
      flex-direction: column;
      justify-content: center; /* vertical centering */
      align-items: center; /* horizontal centering */
      height: 100vh;
      margin: 0;
      font-family: Arial, sans-serif;
    }
    img { width: 400px; } /* BIG logo */
    button { padding: 10px 20px; font-size: 16px; margin-top: 20px; }
  </style>
</head>
<body>
  
  <h1>Thank you</h1>
  <h2>Visit Again</h2>
  <form action="index.html">
    <button type="submit">Back to First Page</button>
  </form>
</body>
</html>
```

Figure 5: HTML source code view of the second page, thankyou.html

Figure 5 as seen in the HTML code of `thankyou.html`, the second page of the website. It includes the head for page info, and the body for the content users see after completing an action.

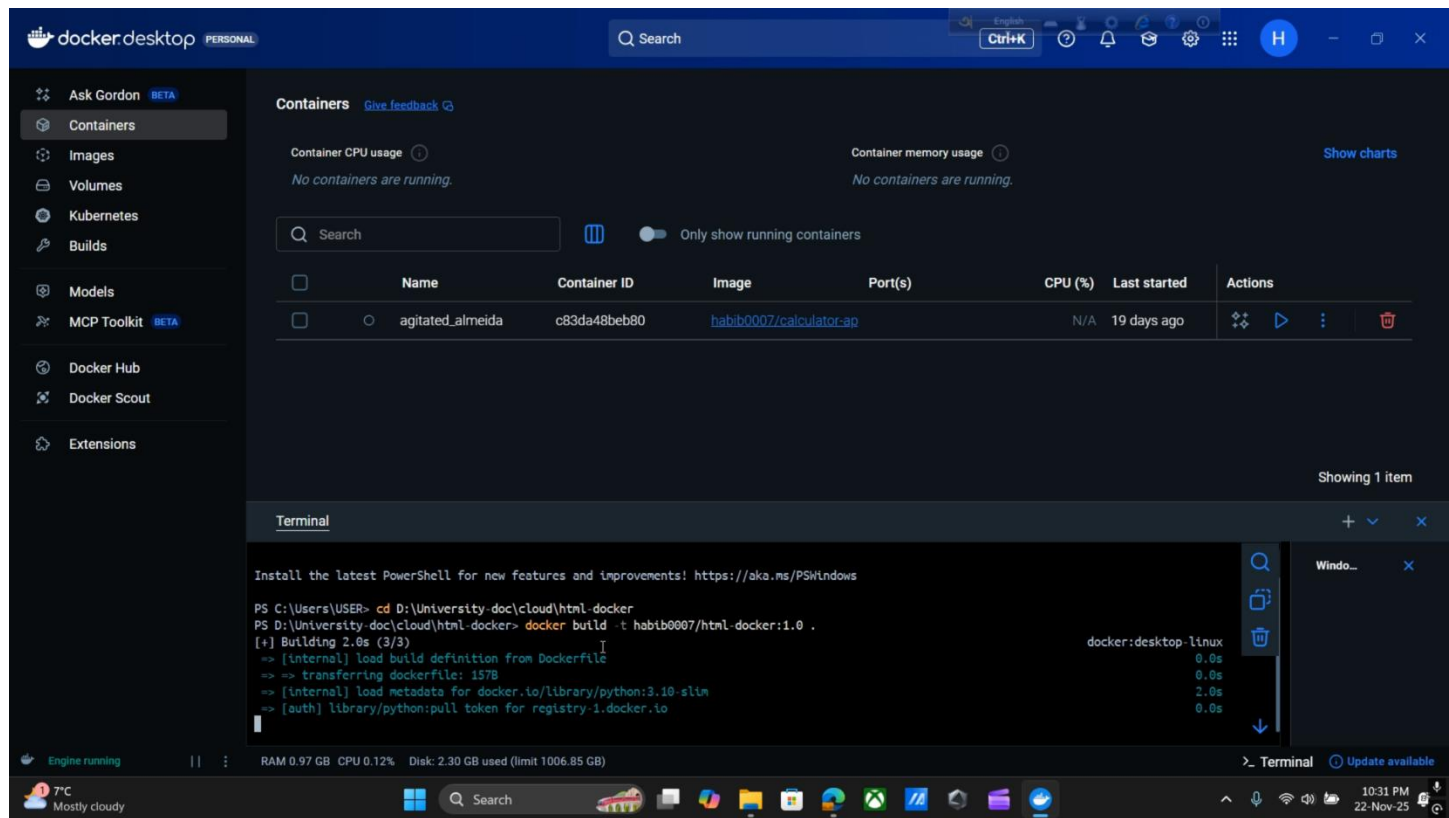


Figure 6: Building a Docker image

Looking at Figure 6, The process of creating a Docker image from a Dockerfile. It illustrates how Docker reads the instructions in the file to build an image that can later be used to run containers.

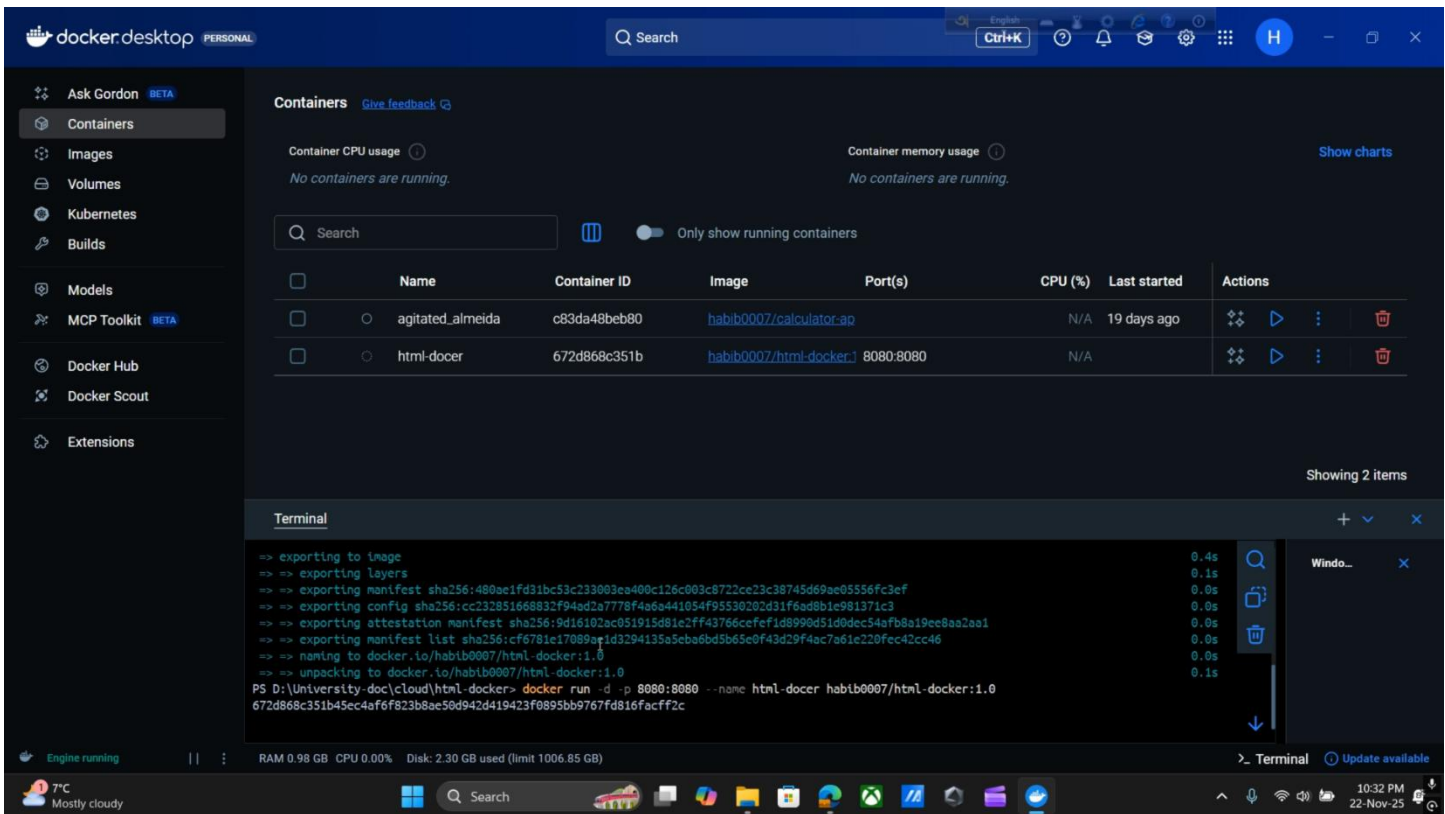


Figure 7: Running the Docker container using the 'docker run' command

Figure 7 Demonstrates the command, which starts a container named 'html-docker' from the image. The flag runs in detached mode, and the container port is mapped to the host port so the application can be accessed from the browser.

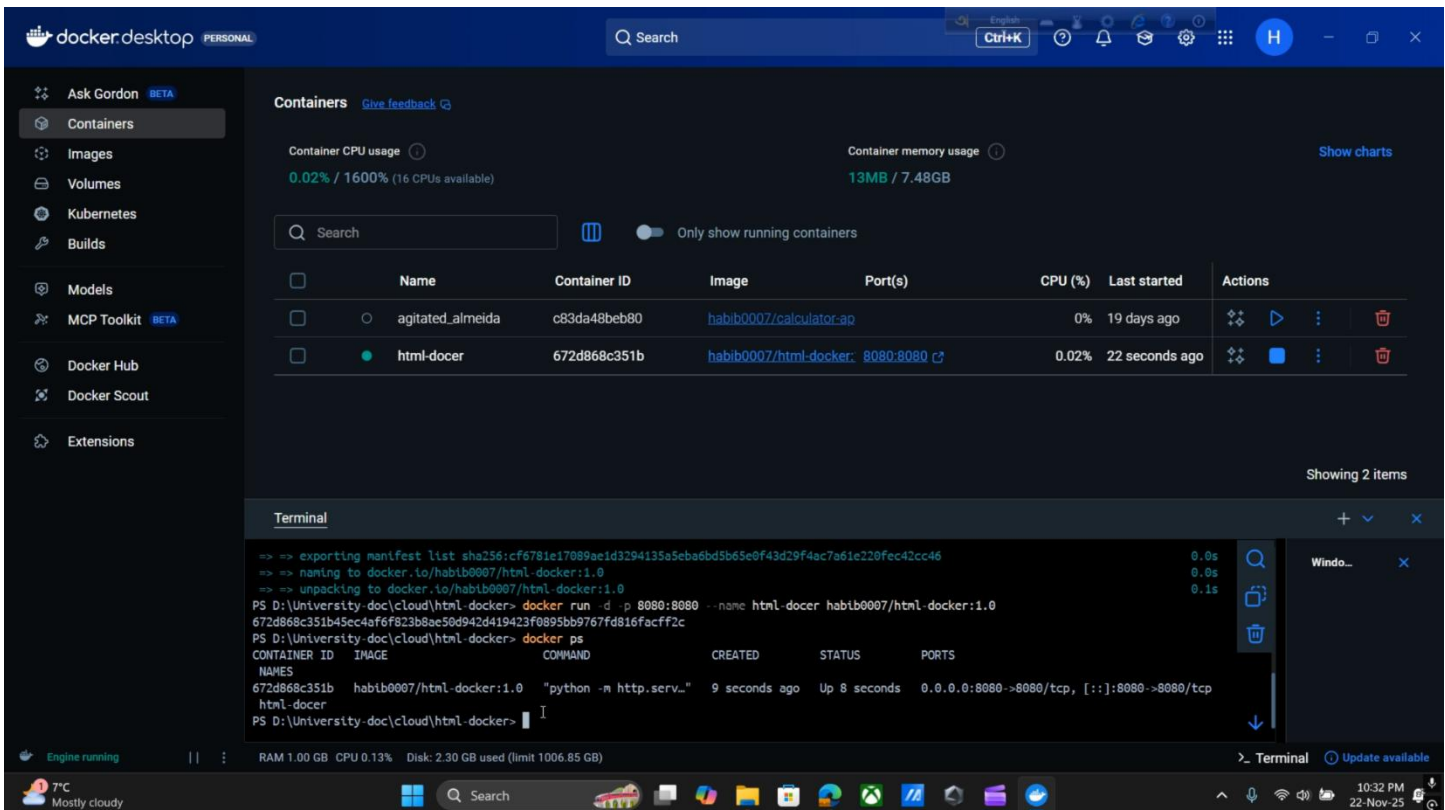


Figure 8: Listing running Docker containers using 'docker ps'

Figure 8 illustrates the ‘docker ps’ command, which lists all currently running containers. This helps verify if a container is up and running and provides details like container ID, name, status, and the image it’s using.

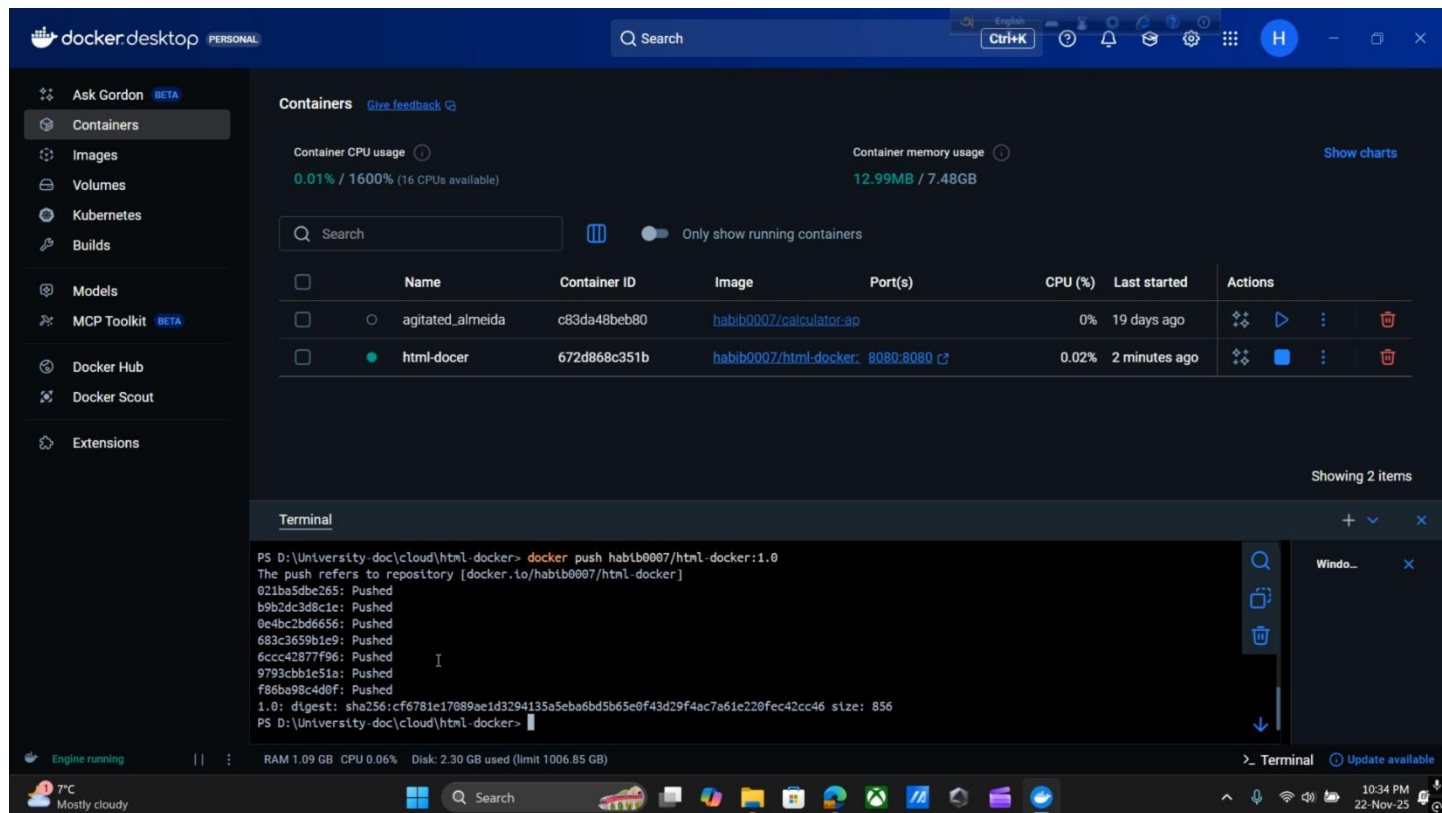


Figure 9: Pushing a Docker image to Docker Hub

Figure 9 shows the process of using the ‘docker push’ command to upload a Docker image to Docker Hub. This makes the image available remotely so it can be accessed or used on other systems.

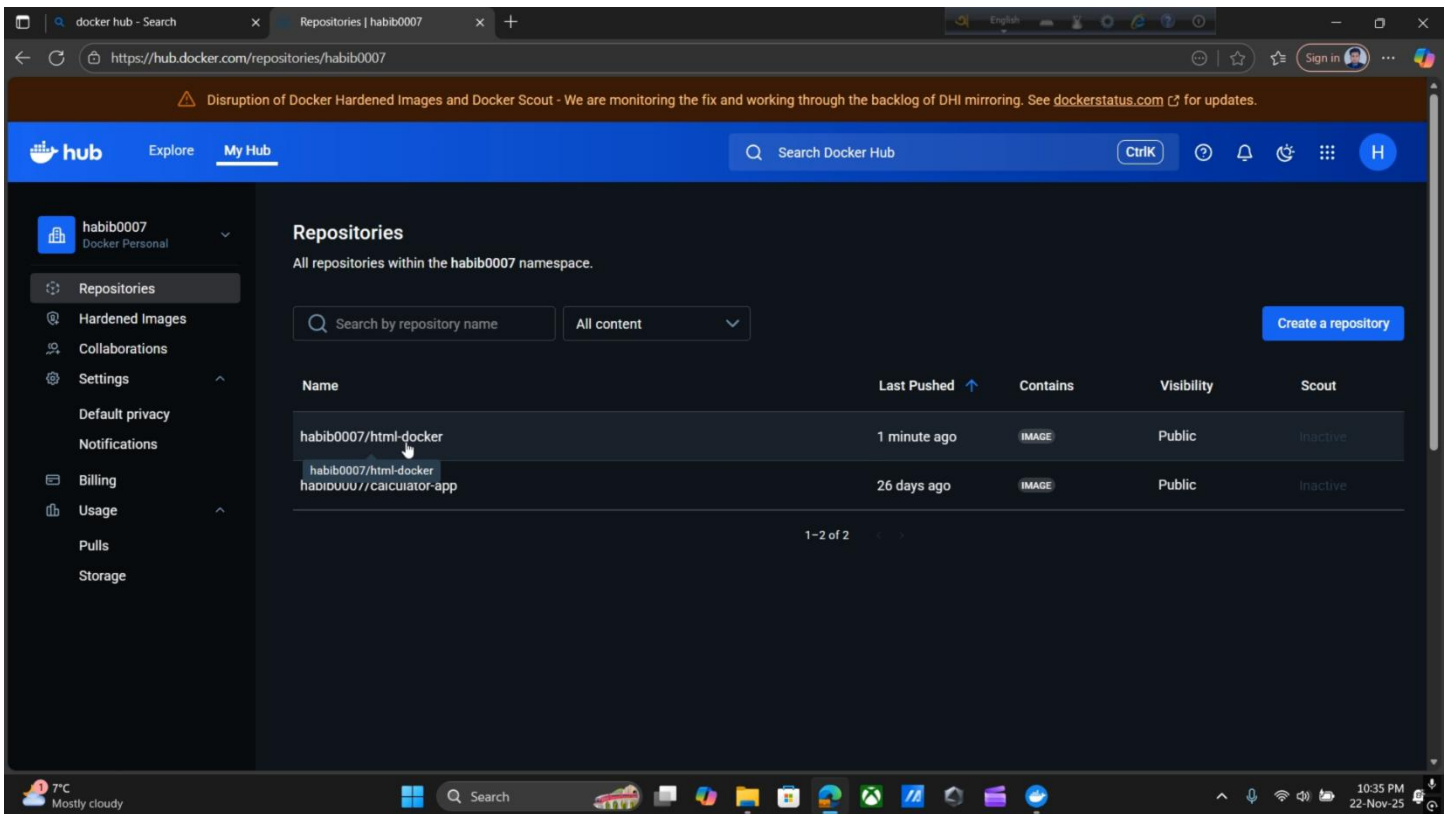


Figure 10: Listing Docker images in the Docker Hub account

According to Figure 10, The Docker Hub account is displaying the pushed image. After a successful 'docker push', the image appears in the account, making it accessible for download or deployment on other machines.

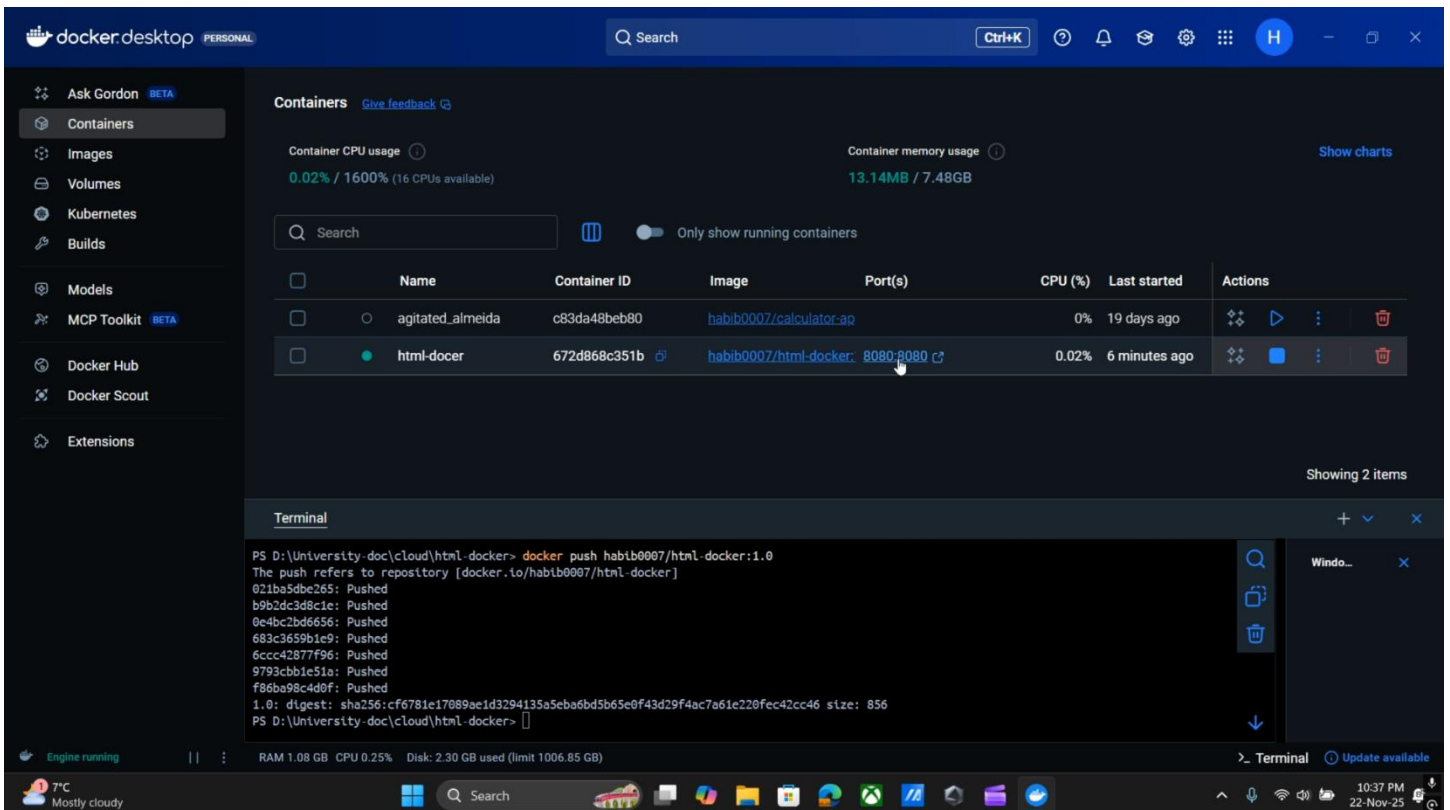


Figure 11: Accessing HTML files through Docker-mapped port.

As seen in the Figure 11, Clicking the mapped port '8080:8080' redirects to 'localhost:8080', allowing users to view the HTML files running inside the Docker container.

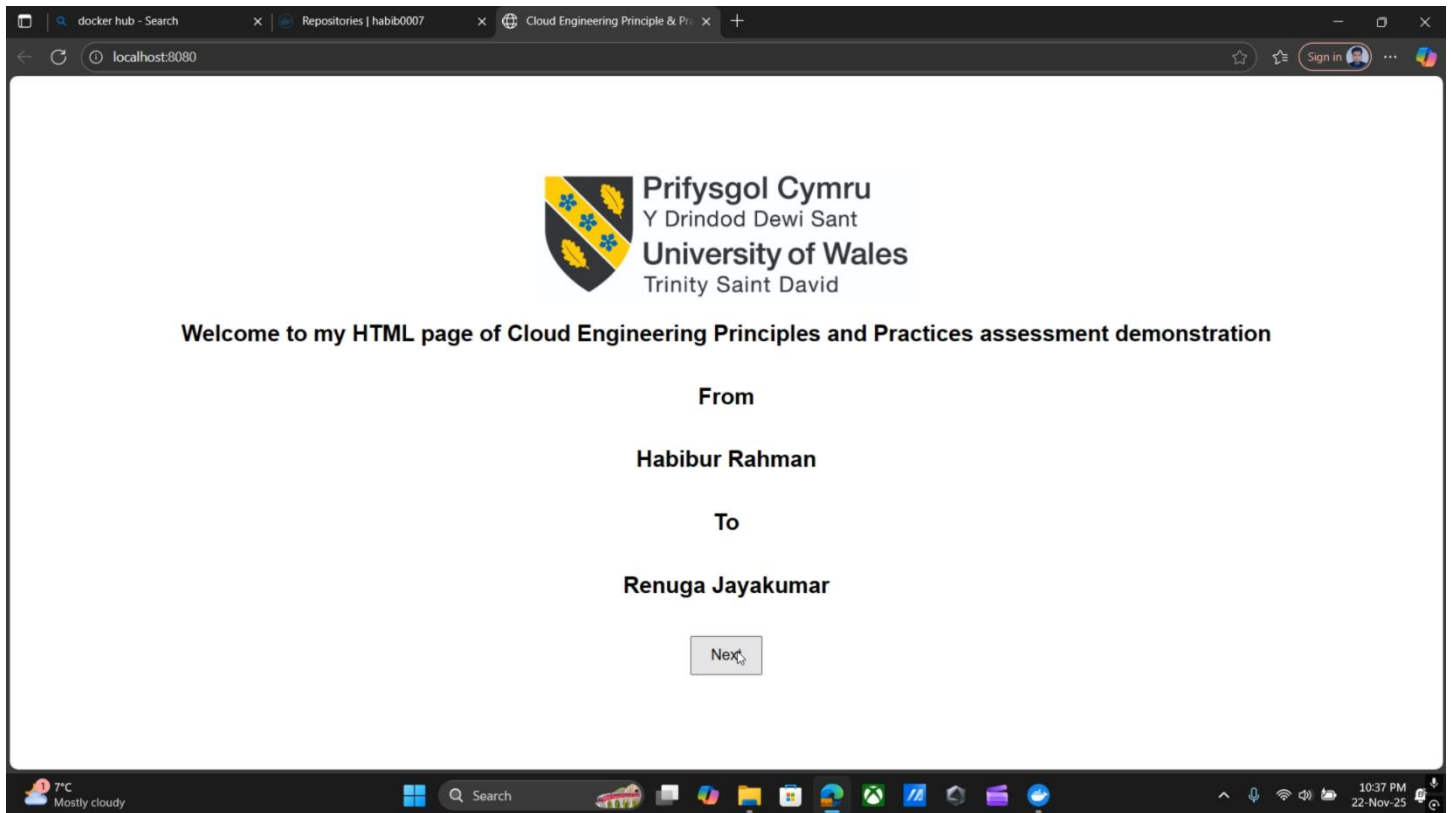


Figure 12: First HTML page view (index.html)

Figure 12 Demonstrates the output of the first HTML page 'index.html' as displayed in the browser, reflecting the content and layout defined in the HTML code.

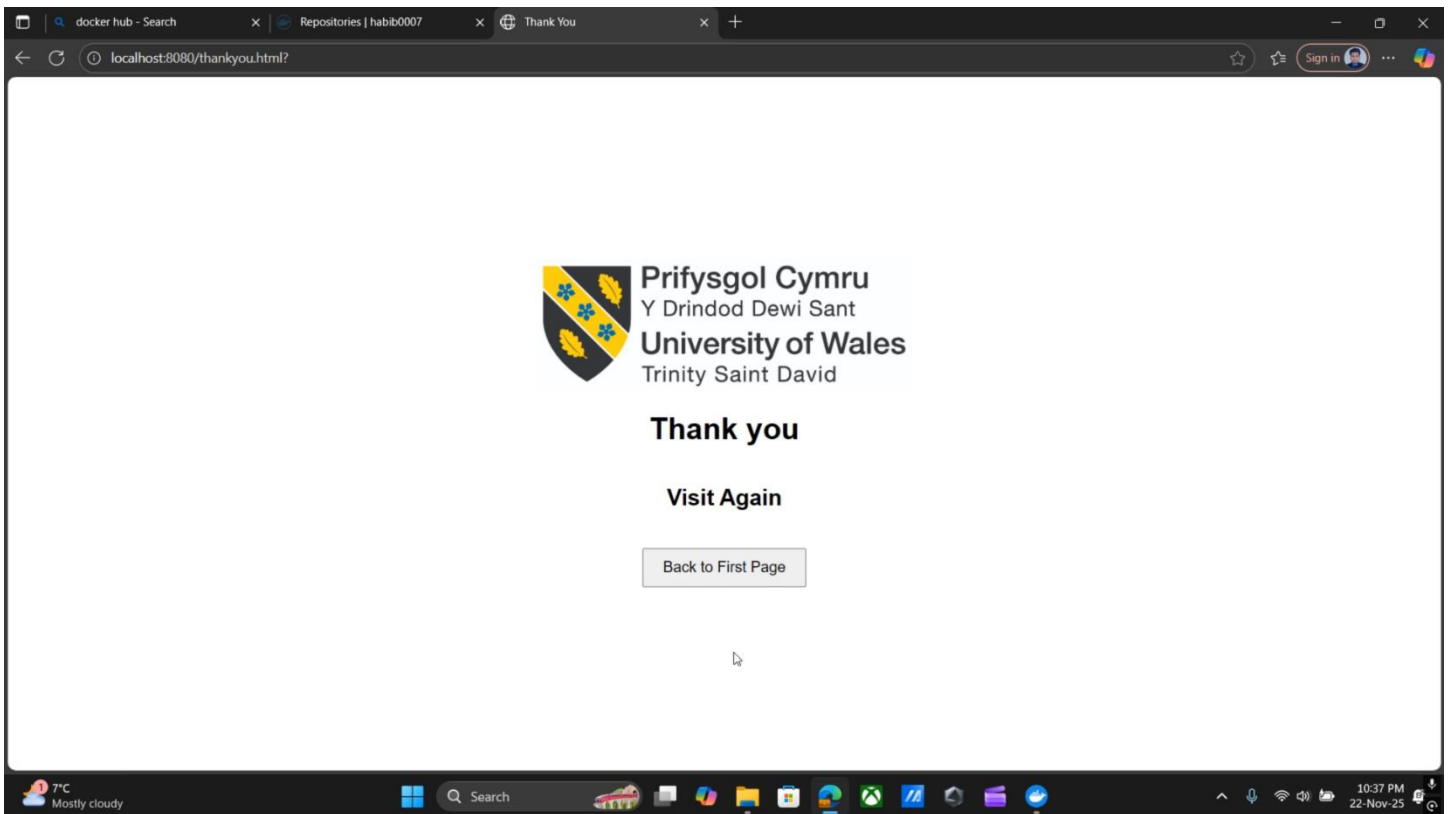


Figure 13: Second HTML page view (thankyou.html)

Figure 13 illustrates the output of the second HTML page ‘thankyou.htm’ as displayed in the browser, reflecting the content and layout defined in the HTML code.

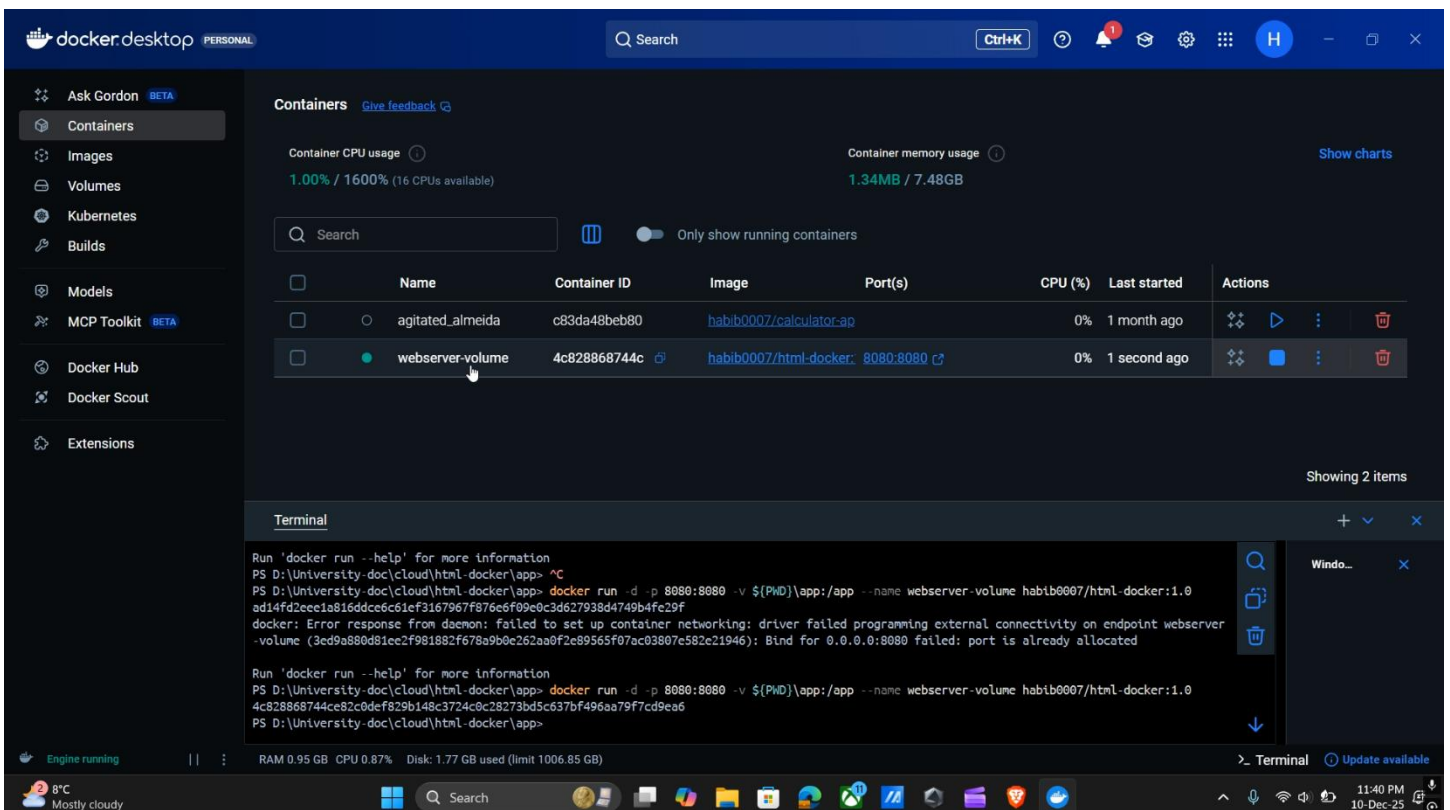


Figure 14: Docker Volume command

Figure 14, shows the command used to create and manage Docker volumes, which allow persistent storage for data used by containers.

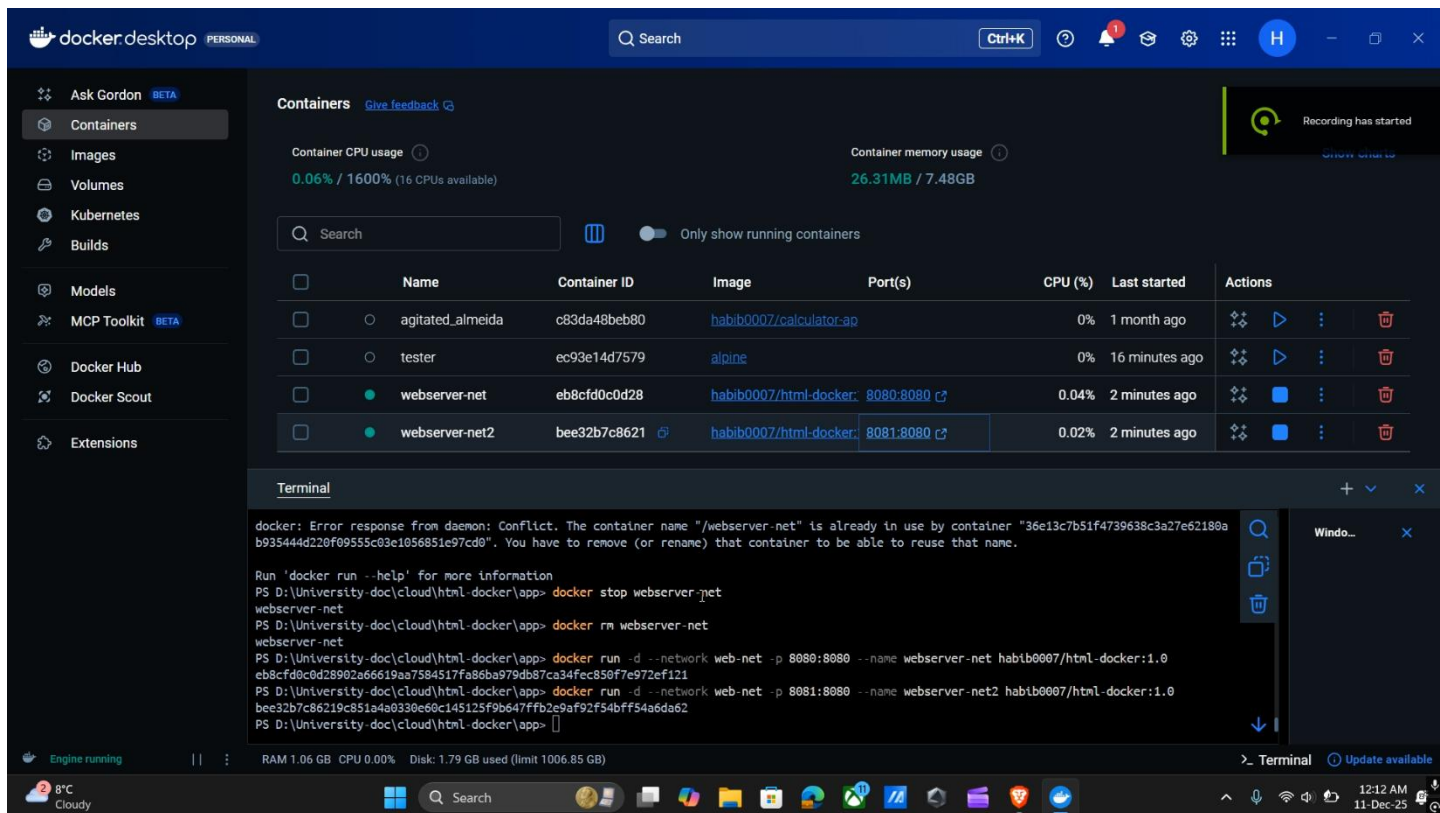


Figure 15: Docker Network command

According to Figure 15 the command used to create and manage Docker networks, enabling communication between containers in an isolated environment.

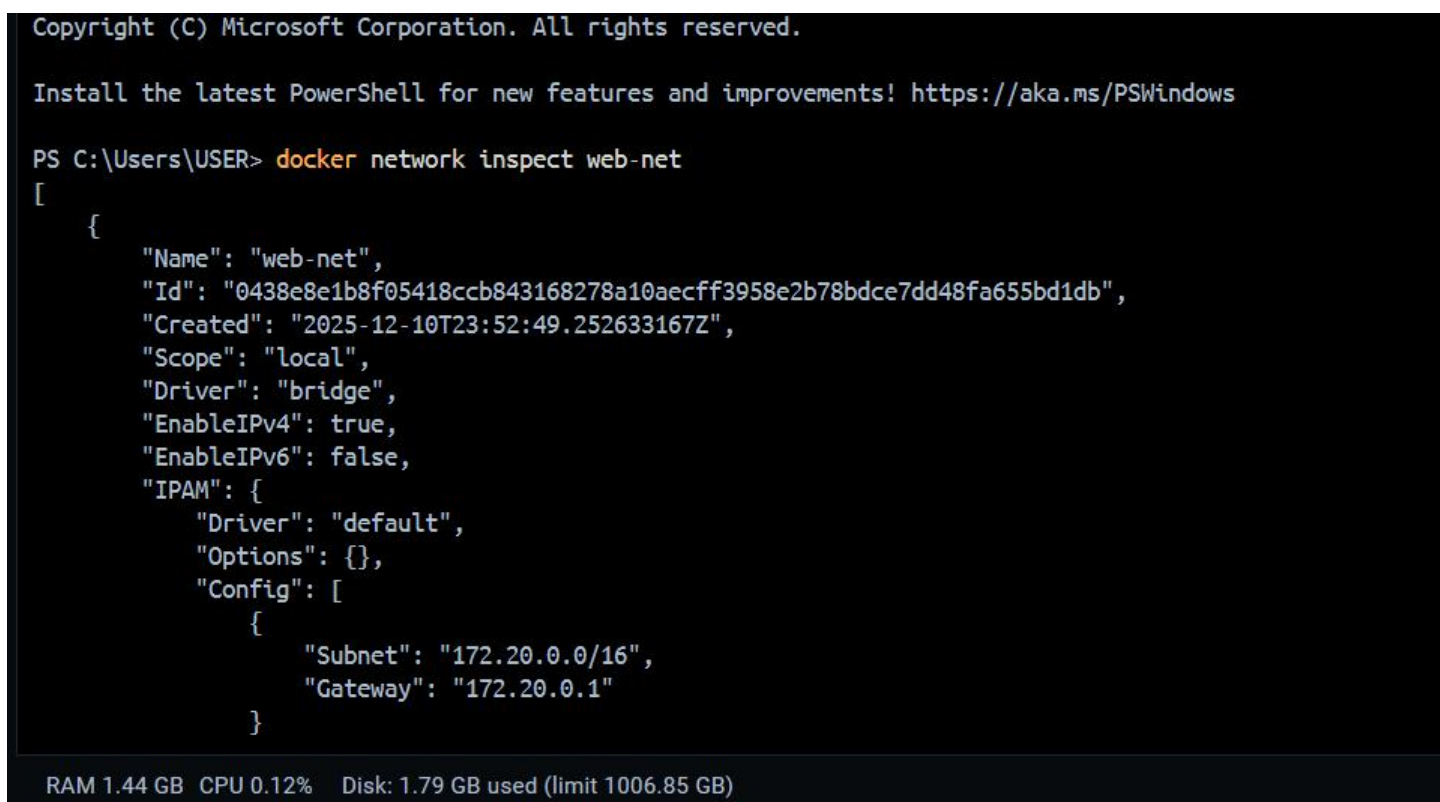


Figure 16: Output of 'docker network inspect' command - 1

```
    ],
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
        "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {
        "bee32b7c86219c851a4a0330e60c145125f9b647ffb2e9af92f54bff54a6da62": {
            "Name": "webserver-net2",
            "EndpointID": "e29dae6fa18e64c6f8185df73c4abc3ba3241ec88682aa9d5a1e3e51b3433ab9",
            "MacAddress": "f6:8c:ed:38:94:dc",
            "IPv4Address": "172.20.0.4/16",
            "IPv6Address": ""
        },
        "eb8cfd0c0d28902a66619aa7584517fa86ba979db87ca34fec850f7e972ef121": {
            "Name": "webserver-net",
            "EndpointID": "bd8a7b0a15f7a1f6a935bb1baa08a47be489521493aa9ebb1e22c44b3c7ec60b",
            "MacAddress": "02:ff:36:6e:a2:65",
            "IPv4Address": "172.20.0.3/16",
            "IPv6Address": ""
        }
    },
    "Options": {
        "com.docker.network.enable_ipv4": "true",
        "com.docker.network.enable_ipv6": "false"
    },
    "Labels": {}
}
```

1.44 GB CPU 0.06% Disk: 1.79 GB used (limit 1006.85 GB)

Figure 17: Output of 'docker network inspect' command - 2

```
    "Name": "webserver-net",
    "EndpointID": "bd8a7b0a15f7a1f6a935bb1baa08a47be489521493aa9ebb1e22c44b3c7ec60b",
    "MacAddress": "02:ff:36:6e:a2:65",
    "IPv4Address": "172.20.0.3/16",
    "IPv6Address": ""
},
"ec93e14d75792f37c32c39fef039f40cddf6c56050f3401ea78fe693ee11b31b": {
    "Name": "tester",
    "EndpointID": "188318d0f17e2942873d5a0aef6d8a5ba0ad9e00e1929d3a78fc82a3adae3a10",
    "MacAddress": "ea:c4:0a:71:a6:a5",
    "IPv4Address": "172.20.0.2/16",
    "IPv6Address": ""
},
"Options": {
    "com.docker.network.enable_ipv4": "true",
    "com.docker.network.enable_ipv6": "false"
},
"Labels": {}
}
```

C:\Users\USER>

Figure 18: Output of 'docker network inspect' command - 3

Figure 16, Figure 17 and Figure 18 show the results of running ‘docker network inspect’ on different Docker networks. This command provides detailed information about the network configuration, connected containers, and settings when the inspection is successful.

7.1. Project Implementation Video (YouTube)

The full video of the “Docker project” is available on YouTube. The link is provided below: <https://youtu.be/-f5OW2PTWWM>

8. Cloud Computing Concepts

Cloud computing is that processing occurs “in the cloud literally everywhere. Specifically, it relates to accessing, using, and storing software and data in a “cloud” depiction of the Internet or other network, including utilizing related services [9]. Most cloud computing infrastructures consist of services delivered through common centers and built on servers. The cloud is like an entry point into consumer computing [9]. While cloud computing is based on multiple essential ideas, load balancing, scalability, and application gateways are three critical concepts for offering dependable, high-performance, and secure cloud services.

8.1.1. Application Selection: Brave Browser

Brave is a privacy-focused web browser that blocks trackers and ads while allowing users to browse the internet securely and efficiently. Brave also provides additional services like Brave Search, which serves millions of search requests daily. Due to fluctuating user traffic, especially during peak hours or major news events, Brave relies on cloud computing technologies such as load balancing, scalability, and application gateways to maintain fast performance, high availability, and secure access [10],[11].

8.1.2. Load Balancing in Brave Browser

For the Brave browser, load balancing is crucial, particularly for its search and content delivery services. To avoid overloading the capacity of a single server, incoming HTTP/HTTPS requests are distributed across multiple backend servers [11]. For instance, requests for pages of Brave Search results can be sent to different server pools, and requests for content delivery (images, scripts) may even be sent one-by-one. More commonly, a Layer 7 application gateway is used for routing, TLS termination, and enforcing a Web Application Firewall (WAF) to stop malicious traffic before it reaches the backend [11], [12]. Layer 4 of the OSI model that could be used to help balance TCP/UDP connections between internal services, for example, cache and API connections, for example to see to see that all your load on it is as efficient through your system.

8.1.3. Scalability in Brave Browser

Scalability enables Brave to handle variable traffic efficiently. Horizontal scaling allows the browser’s backend services to automatically add more servers during traffic rise, such as during viral news events or peak browsing hours, and reduce servers when traffic drops to save costs [13]. Kubernetes Horizontal Pod Auto scalers (HPA) or cloud auto-scaling groups monitor CPU, memory, or custom metrics (like requests per second) to dynamically adjust the number of active servers or pods [13], [11]. This ensures that users consistently receive fast search results and content delivery, even under heavy load.

8.1.4. Application Gateway in Brave Browser

The application gateway acts as a secure front door for Brave’s services, operating at the application layer (Layer 7) [12]. It manages intelligent routing, TLS termination, and security inspections using WAF rules. For example, search requests,

content requests, and API calls can be routed to separate backend pools, while traffic identified as malicious is blocked at the edge. This protects user privacy, reduces latency, and ensures that Brave's services remain reliable and secure [11], [12].

8.2. Relationship Between Load Balancing and Scalability

Load balancing and scalability work together to optimize Brave's performance. Load balancing ensures that incoming traffic is evenly distributed, while scalability ensures that there are enough servers to handle fluctuations in demand [11]. Without scaling, load balancers risk overloading servers, and without load balancing, newly added resources may remain underutilized. Together, these technologies maintain high availability, low latency, and cost-efficient resource usage [14], [13].

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

By combining load balancing, scalability, and application gateways, Brave delivers a fast, reliable, and secure browsing experience. Users can access search results and browse content without delay, even during traffic spikes. Security is enhanced as WAF rules and gateway controls block malicious traffic before it reaches the backend. Additionally, autoscaling and global load balancing help optimize operational costs while ensuring consistent performance and responsiveness for users worldwide [10], [12], [14].

8.4. Driving Forces Behind Cloud Computing Expansion

Cloud computing has gained popularity over the last decade for several reasons, including cost efficiency, flexibility, scalability, and innovation.

8.4.1. Cost Efficiency

The traditional IT infrastructure requires a significant investment in server, storage, and networking equipment upfront. but cloud computing is pay-as-you-go; they only pay for the computing power they consume. As a result, the cost associated with over-provisioning is removed, and organizations can expand and contract their capacity based on demand without significant capital investment [15].

8.4.2. Scalability and Flexibility

Cloud services enable resources to be quickly scaled up or down, depending on demand, ensuring applications are efficient. This flexibility allows companies to deploy applications quickly and to elastically maintain resources depending on demand [15].

8.4.3. Accessibility and Global Reach

Cloud applications can be accessed anywhere, enabling remote collaboration and global deployment. It is good because providers like AWS, Azure, and Google Cloud have multiple such data centers around the globe, and end users of the system experience better performance since there is less latency [16].

8.4.4. Maintenance and Operational Simplification

Cloud providers maintain characteristics such as hardware, updates, and patches for vulnerabilities, and concentrate on writing features rather than dealing with hardware. This decreases the workload of overhead operations and enhances the robustness of the service [17].

8.4.5. Innovation Enablement

Cloud computing provides other in-depth tools, including AI, ML, analytics, IoT and serverless. These cloud offerings let businesses innovate more quickly and try out new features without major investment in infrastructure [18].

8.5. Serverless Functions (Functions as a Service)

As described in [19], Serverless computing, also known as functions as a service (FaaS), is a cloud model that allows developers to run code without managing servers. The latter automatically manages resource provisioning, scaling, and execution. Instances being AWS Lambda, Azure Functions, and Google Cloud Functions.

8.5.1. Characteristics

According to [19], Serverless cloud architecture includes three fundamental characteristics:

1. **Granular billing:** The user of a serverless model is only charged when the program is running.
2. **(Almost) no operational logic;** Operational logic, such as resource management and autoscaling, is delegated to the infrastructure, making such issues the responsibility of the infrastructure operator.
3. **Event-Driven:** Serverless applications are designed for short-lived interactions, allowing the infrastructure to deploy them in response to events as needed.

8.5.2. Benefits

As mentioned earlier [19], the benefits of serverless

- Designed for building applications in a cloud-native way
- Makes it easier and faster to develop and combine services
- Lowers operational effort by handling service execution automatically

8.6. Serverless Functions vs Containerization

According to [20], Some of the differences between serverless and containerization are given below in Table 2

Aspect	Serverless (FaaS)	Containerization
Scale	Auto-scales with workload	Needs orchestration
Deploy	Simple, function-based	Complex, app-based
Cost	Pay-per-use, good for spikes	Fixed, good for steady load
Performance	Cold start possible	Predictable, always-on
Use Case	Event-driven, stateless apps	Stateful, long-running apps

Table 2: Comparison between serverless and containerization

9. Conclusion

In conclusion, the review provides a strong appreciation for how contemporary cloud systems are designed to be smarter, stronger, and more adaptable. It demonstrates how, with the combined force of base-level virtualization and

the ability to respond quickly with simplicity from containerization and tools such as Docker, we have the power to create systems that are highly effective while easily deployable. On the other hand, in cloud computing, Load balancing, scalability, and application gateways are used to provide services that are dependable, safe, and fast. Load balancing and scaling collaborate to effectively manage traffic, while application gateways enhance security and routing. The primary drivers of cloud computing growth are reduced prices and greater flexibility. Serverless computing cuts maintenance work and costs even more than containers, making cloud platforms more effective and appealing to modern applications. Combine all that technology, these aren't just flashy terminology, they are the practical tools to ensure systems optimize resources and don't spend money on idle capacity or suffer from blocked up-time. In the end, they work together to form a cloud architecture that is flexible and robust, ready to support modern digital needs.

10. References

- [1] M. Zostant and R. Chataut, "Privacy in computer ethics: Navigating the digital age," *Comput Sci Inf Technol*, vol. 4, no. 2, -07-01 2023. doi: 10.11591/csit.v4i2.pp183-190.
- [2] J. Wu, et al., "Cloud Storage as the Infrastructure of Cloud Computing," 2010-06. doi: 10.1109/icicci.2010.119.
- [3] S. Susnjara and I. Smalley, "What Is Virtualization?" IBM Think. <https://www.ibm.com/think/topics/virtualization> (accessed 8 December).
- [4] F. Rodríguez-Haro, et al., "A summary of virtualization techniques," 2012. doi: 10.1016/j.protcy.2012.03.029.
- [5] R. Morabito, J. Kjallman and M. Komu, "Hypervisors vs. Lightweight Virtualization: A Performance Comparison," 2015-03. doi: 10.1109/ic2e.2015.74.
- [6] R. Dua, A. R. Raja and D. Kakadia, "Virtualization vs Containerization to Support PaaS," 2014-03. doi: 10.1109/ic2e.2014.41.
- [7] A. M. Potdar, et al., "Performance Evaluation of Docker Container and Virtual Machine," 2020. doi: 10.1016/j.procs.2020.04.152.
- [8] "Docker Documentation." Docker Docs. <https://docs.docker.com/> (accessed 13 December).
- [9] M. Malathi, "Cloud Computing Concepts," *2011 3rd International Conference on Electronics Computer Technology (ICECTECH)*, doi: 10.1109/ICECTECH.2011.5942089.
- [10] Brave Software. "Get Started — Brave Web Search API Documentation." Brave Web Search API. <https://api-dashboard.search.brave.com/app/documentation/web-search/get-started> (accessed 16 December).
- [11] Microsoft. "Azure Application Gateway." Microsoft Learn. <https://learn.microsoft.com/en-us/azure/application-gateway/> (accessed 17 December).
- [12] Amazon Web Services (AWS), "Elastic Load Balancing," Accessed: 17 December 2025. [Online]. <https://docs.aws.amazon.com/pdfs/elasticloadbalancing/latest/userguide/elb-ug.pdf>
- [13] Kubernetes. "Horizontal Pod Autoscaling." Kubernetes Documentation. <https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/> (accessed 16 December).

- [14] K. Scarfone, et al., "Technical Guide to Information Security Testing and Assessment Recommendations of the National Institute of Standards and Technology," 2008. Accessed: 17 December 2025. [Online]. <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-115.pdf>
- [15] R. Patil, et al., "Serverless Computing and the Emergence of Function-as-a-Service," *2021 International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT)*, -08-27 2021. doi: 10.1109/rteict52294.2021.9573962.
- [16] Amazon Web Services (AWS). "Six Advantages of Cloud Computing." Overview of Amazon Web Services. <https://docs.aws.amazon.com/whitepapers/latest/aws-overview/six-advantages-of-cloud-computing.html> (accessed 17 December).
- [17] C. Gong, et al., "The Characteristics of Cloud Computing," 2010-09. doi: 10.1109/icppw.2010.45.
- [18] H. Mora, et al., "Serverless Computing at the Edge for AIoT Applications," 2022-12-29. doi: 10.1109/icaiot57170.2022.10121879.
- [19] E. Van Eyk, et al., "The SPEC cloud group's research vision on FaaS and serverless architectures," 2017-12-11. doi: 10.1145/3154847.3154848.
- [20] A. M. Giovanni, "Serverless Computing vs. Containerization: Comparative Analysis for Cloud-Native Application Deployment,"