



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

Cloud Engineering Principles and Practice

ACCA7027

Student Name : Jay Shaileshbhai Lad

Student ID:2504388

Program: MSc Cybersecurity and Digital Forensics

Submission To: Renuga Jayakumar

Table of Contents

1. Introduction.....	3
2. Cloud Technology (Docker) and Examples.....	3
2.1 Principles of Virtualisation	3
2.2 Principles and Benefits of Containerisation.....	4
2.3 Docker Images and the Process of Publishing Images	6
2.4 Docker Volumes and Persistent Storage.....	8
2.5 Container Networking and Service Communication	10
2.6 Combined Example: A Complete End-to-End Container Workflow	15
2.7 Video Demo	17
3. Cloud Computing Concepts.....	17
3.1 Load Balancing in Cloud Computing	17
3.2 Scalability in Cloud Infrastructure.....	18
3.3 Application Gateways and Intelligent Traffic Management.....	19
3.4 Why Load Balancing and Scalability Are Complementary Concepts.....	20
3.5 Benefits of Combining Load Balancers, Scalability and Application Gateways	20
4. Main Driving Forces Leading to the Expansion of Cloud.....	21
4.1 Cost Efficiency and Reduction of Capital Expenditure	21
4.2 Flexibility, Scalability and Rapid Provisioning.....	22
4.3 Innovation Through Automation and DevOps Practices	22
4.4 Reliability, Availability and Global Reach.....	23
4.5 Security, Compliance and Managed Services.....	23
5. Conclusion	24
References.....	25

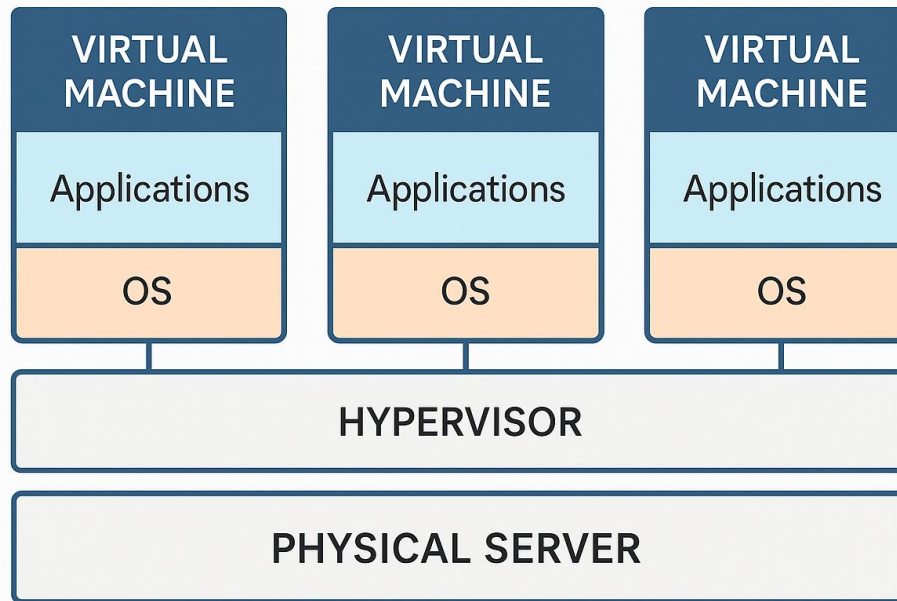
1. Introduction

The design, deployment and management of current digital services have been changed by cloud computing to present efficient, scaling and highly comprehensive computing solutions. With organisations moving workloads to cloud environments more often, portability, support and automation-supported technologies and operational consistency are now a necessity. Docker, which is a popular containerisation platform, is at the centre of this change as it allows applications to be deployed in a more reliable manner across multiple computing environments. Docker is used to distribute software via light and isolated containers and removes much of the complexity that is believed to be inherent with the use of virtual machines. In the provision of responsive, resilient and secure services, along with containerisation, cloud platforms are based on important architectural principles, such as load balancing, scalability, and application gateways. All these mechanisms improve performance and ensure system overload is avoided, as well as intelligent traffic routing among distributed systems. Economic and operational reasons such as lowering infrastructure costs, flexibility in resource assignment and quick deployment are even further motivating the development of cloud computing. The recent development covers serverless computing, where functions are done on demand, and you need not worry about operating the servers. The analysis of serverless models and container-based deployment shows the dynamic nature of cloud-native engineering and the solutions that can be offered to various application needs. These technologies are collectively characterised as the practices that constitute cloud engineering in the present day.

2. Cloud Technology (Docker) and Examples

2.1 Principles of Virtualisation

The overriding principle under cloud computing is virtualisation, which makes cloud computing work effectively. It also enables the physical hardware to be abstracted into a number of isolated environments, in that a single physical server may contain lots of independent virtual machines. The virtual machines have their own operating system, memory, virtualised computer and virtualised storage resources, though these resources are anchored on a single physical host. It is an abstraction designed by a hypervisor, a software layer charged with the task of creating, administering, as well as isolating several virtual machines on the same hardware platform [1].



Virtualization

Practically, the benefits of virtualisation are better hardware revision, lower operating expenses and is ability to roll out computing capacities. Instead of allocating a specific physical server to a certain application, virtualisation can have multiple applications safely run in separate virtual machines with no harm. The model has allowed data centres and cloud providers to scale their services by dramatic factors since servers can be used more effectively, and heterogeneous workloads can be hosted in the same place.

Class examples usually consist of trying to illustrate how virtual machines are allocated by the cloud provider on demand. An example of this is where a student who is testing a Linux server on a cloud computing system is running a virtual machine which is sharing hardware with numerous other clients. This is seen as a dedicated environment to the user in spite of the fact that the underlying physical resources are shared [2]. This degree of abstraction is important in explaining the subsequent emergence of Docker: despite having eliminated important constraints on the degree of hardware utilisation, virtual machines created overhead due to complete emulation of an operating system. The alteration of solutions toward being more efficient led to containerisation.

2.2 Principles and Benefits of Containerisation

Containerisation is an effort based on virtualisation, but in a fundamentally different manner. Rather than emulating a whole operating system to put it within each environment, containers

use the same kernel of the host system and provide isolation at the application level. This causes containers to be much lighter and quicker to initialise, in addition to being more resource efficient, as compared to conventional virtual machines. Docker has created an open-source ecosystem of containerisation, which includes application building, packaging, and running tools in isolated settings.

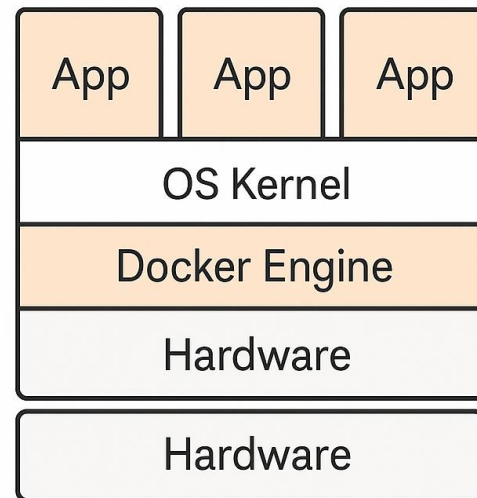
Reproducibility is a major concept of containerisation. Since a container contains its application files, dependencies and runtime configuration, it will act in the same manner regardless of the environment it is run in [3]. The consistency removes historical problems in software development, including environment mismatch, dependency conflicts and configuration drift. A common example in a classroom situation is where students find out that an application that works on one system can crash on another based on the absence of dependencies or different operating system setups. These variables are captured in the container environment with containerisation, and as such, these variables exhibit homogeneous behaviour.

There is also an enhancement of scalability through containerisation. The containers are rapidly replicable, hence they are ideal for load-balanced or distributed cloud applications. They come online nearly instantly, allowing the ability of cloud orchestration platforms to scale capacity down to a fine-grained level. Besides, containers facilitate the microservices architecture where large applications are segmented into discrete, modular services that can be developed and deployed independently, and scaled independently. The model is very common in the current cloud platforms, and it is also part of the skill set required in cloud engineering [4].

Virtual Machines



Containers



On the workflow side, the creation of a container image starts with the definition of the base environment and specification of the files and configuration of the applications. Though the process is defined with the help of a configuration file in practice, its conceptual framework can be defined without any references to code. The base environment is selected, application files are added, and documentation of how to set up the runtime environment is established. When such a setup is run, Docker generates a containerised image that can run anywhere.

2.3 Docker Images and the Process of Publishing Images

A Docker image is a packaged, self-contained plan of execution of an application. It contains all that is needed to run the application: runtime environment, libraries, settings and code of the application. Images have a layered architecture whereby a new read-only layer is added with each instruction in the build process [5]. This hierarchical method maximises the storage, facilitates effective caching and eases the distribution. An illustration is when different pictures have a similar base environment, then a single copy of such a layer is required to be stored.

When performing classroom exercises, image building usually commences with a description of the application environment and an explanation of how the application is to be prepared to be executed. After creating an image, it is a portable object and can be executed the same on any device or service that supports Docker. It is one of the most important strengths of Docker,

and, therefore, it comes in especially handy with cloud engineering, where applications can frequently be transferred between development, testing, and production environments.

In cloud processes, it is vital to publish an image to a store-like DockerHub. A registry is a centralised database in which images can be stored, revised and shared [6]. In a classroom exercise, the student, after generating an image locally, authenticates to a registry service, where they give the image a tag to distinguish it and submit the image to the far-off storehouse. When it is published, the image may be downloaded and run on any other system of choice. This feature facilitates collaborative development due to the ability of teams to develop a similar environment and automated deployment pipelines under continuous integration and continuous delivery (CI/CD). In cases of cloud deployments, registries can also guarantee the constant availability of images to the tools used in orchestrating a deployment to enable a seamless deployment across pools or distributed environments.

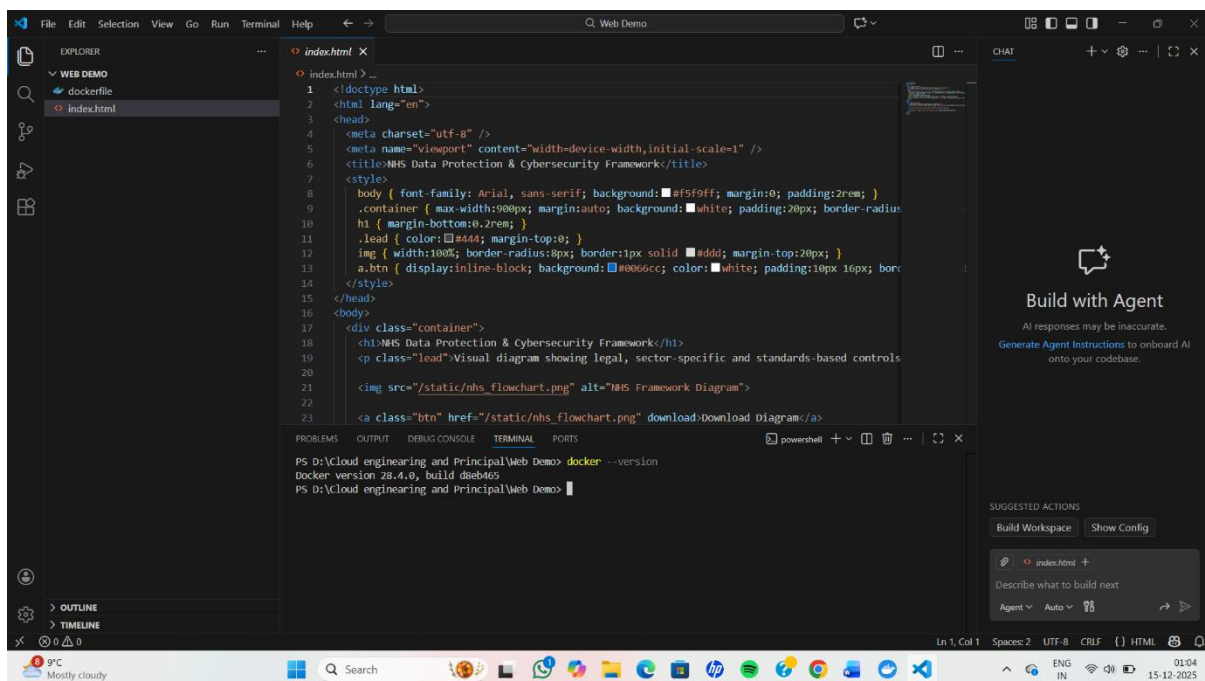


Figure 1. Docker Image

1. Build Image: `docker build -t jay18082005/docker-web-demo .`

```

PS D:\cloud engineering and Principal\Web Demo> docker build -t jay18082005/docker-web-demo .
[*] Building 476.6s (8/8) FINISHED
=> [internal] load build definition from dockerfile
=> transferring dockerfile: 119B
=> WARN: ConsistentInstructionCasing: Command 'FROM' should match the case of the command majority (lowercase) (line 1)
=> [internal] load metadata for docker.io/library/python:3.11-slim
=> [internal] load .dockerignore
=> transferring context: 2B
=> [internal] load build context
=> transferring context: 32B
=> [1/2] FROM docker.io/library/python:3.11-slim@sha256:158caf0e880e2cd74ef2879ed3c4e697792ee5251c8208b7af56683c32e0ac
=> resolve docker.io/library/python:3.11-slim@sha256:158caf0e880e2cd74ef2879ed3c4e697792ee5251c8208b7af56683c32e0ac
=> sha256:1733a4cd59540b3470f7a90963bcd5a53279dd8bdaef02d7883fda221e5 29.70MB / 29.70MB
=> sha256:72cf4c3b83019e176aba979aba19d35f56576bbcfaf7240a1ab1dd536730b 1.29MB / 1.29MB
=> sha256:4d5cfcf3663813d0c369bcd532b89f41cf07b5d95887ef686538370a747c 14.36MB / 14.36MB
=> sha256:3f8dcba744e7bdc0effeda73b9148829b64309925992954a314ba203f56e99 249B / 249B
=> sha256:1733a4cd59540b3470f7a90963bcd5a53279dd8bdaef02d7883fda221e5
=> extracting sha256:72cf4c3b83019e176aba979aba19d35f56576bbcfaf7240a1ab1dd536730b
=> extracting sha256:4d5cfcf3663813d0c369bcd532b89f41cf07b5d95887ef686538370a747c
=> extracting sha256:3f8dcba744e7bdc0effeda73b9148829b64309925992954a314ba203f56e99
=> [auth] library/python:pull token for registry-1.docker.io
=> [2/2] COPY index.html /url/share/nginx/html/index.html
=> exporting to image
=> exporting layers
=> exporting manifest sha256:f98baedf657f354ef05ed809dce874d9bd7a5a3ach33f03f1728deaab470bb6f
=> exporting config sha256:4d5d8a724604357e364fba5bae5400ae73f94251c0b09032e0f884ef96a6eb
=> exporting attestation manifest sha256:47f6a3a117ceaf720ef4d50c1908c1240d4248b8f657a09d831129e8427
=> exporting manifest list sha256:c956b6c0142719f5c1ef3c79542e1c8aef3d571d50c31a7b5163791f4aa098
=> naming to docker.io/jay18082005/docker-web-demo:latest
=> unpacking to docker.io/jay18082005/docker-web-demo:latest

1 warning found (use docker --debug to expand):
- ConsistentInstructionCasing: Command 'FROM' should match the case of the command majority (lowercase) (line 1)
PS D:\cloud engineering and Principal\Web Demo>

```

Figure 2. Docker Image Build.

2. Push to DockerHub: docker push jay18082005/docker-web-demo

```

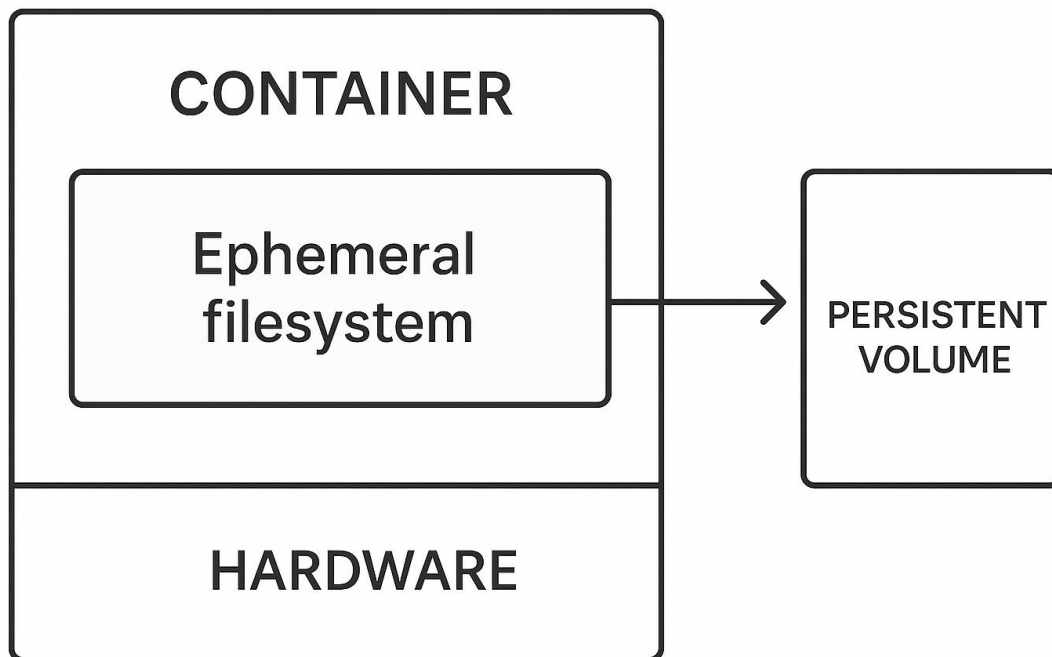
PS D:\cloud engineering and Principal\Web Demo> docker push jay18082005/docker-web-demo
Using default tag: latest
The push refers to repository [docker.io/jay18082005/docker-web-demo]
180667ace1a7: Pushed
65308d834b56: Pushed
1733a4cd5954: Pushed
4d5cfcf366: Pushed
3f8dcba744e: Pushed
72cf4c3b8301: Pushed
latest: digest: sha256:c956b6c0142719f5c1ef3c79542e1c8aef3d571d50c31a7b5163791f4aa098 size: 856
PS D:\cloud engineering and Principal\Web Demo>

```

2.4 Docker Volumes and Persistent Storage

Data persistence is one of the key factors that are considered in containerised environments. Containers are intended to be tamper-evident in order to be discarded, and internal file systems are momentary [7]. Any data written in a container will be lost when a container is stopped or

removed. Docker volumes overcome this fact and allow persistent storage, which lives beyond the life cycle of containers.



A Docker volume is a controlled volume that is managed by Docker. Volumes exist on the outside of the internal file system of a container, allowing information to persist on a restart or replacement of the container. They are specifically relevant in use in applications like databases, content management systems and logging services where data survival is important. Students can see that the crucialness of volumes in the context of classroom exercises is evident by utilising an application container to store data, pause it, and then resume without losing the information provided in the container.

Also, volumes can improve the process of data interchange among several containers. To give an example, one container may produce data that is analysed in another container, yet be referencing the same underlying volume [8]. Such a design accommodates modular workflows and is based on the microservice interactions in reality. In addition, volumes minimise the threat of corruption of data that might have happened had there been application data that were stored on volatile container file systems. Since it is often the case that cloud environments require scaling of recovery, rejects of external storage will guarantee data consistency irrespective of the number of times an application container is re-created.

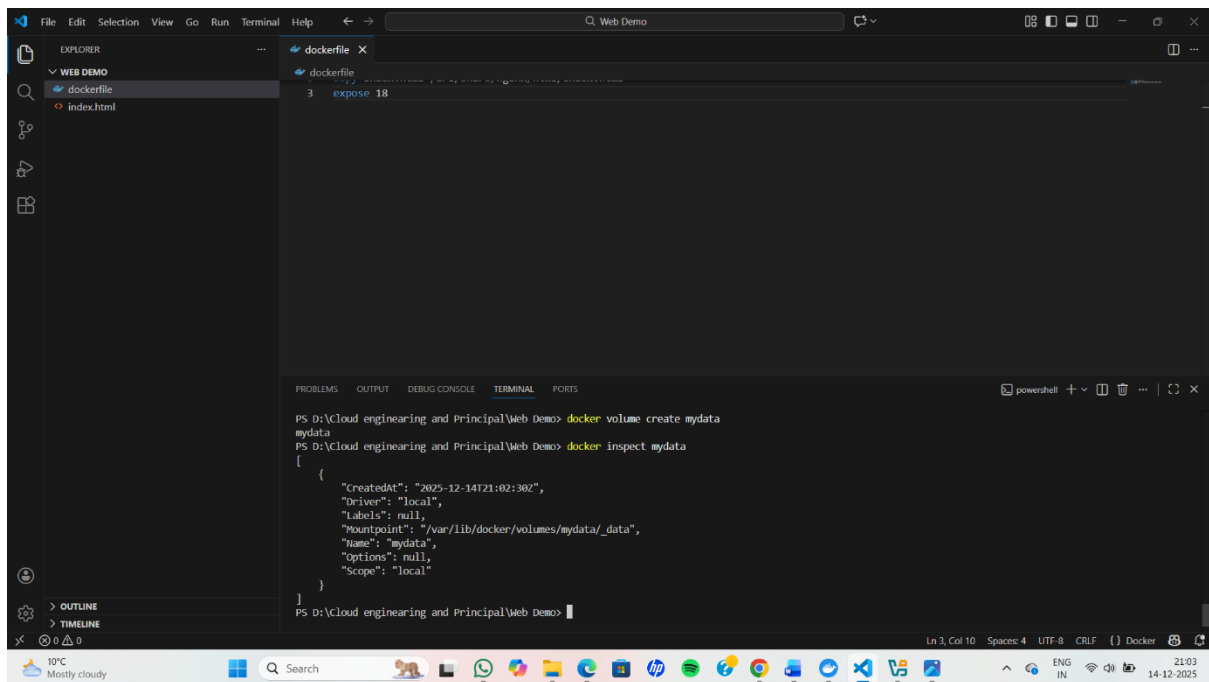


Figure 4. Docker Volume Creation

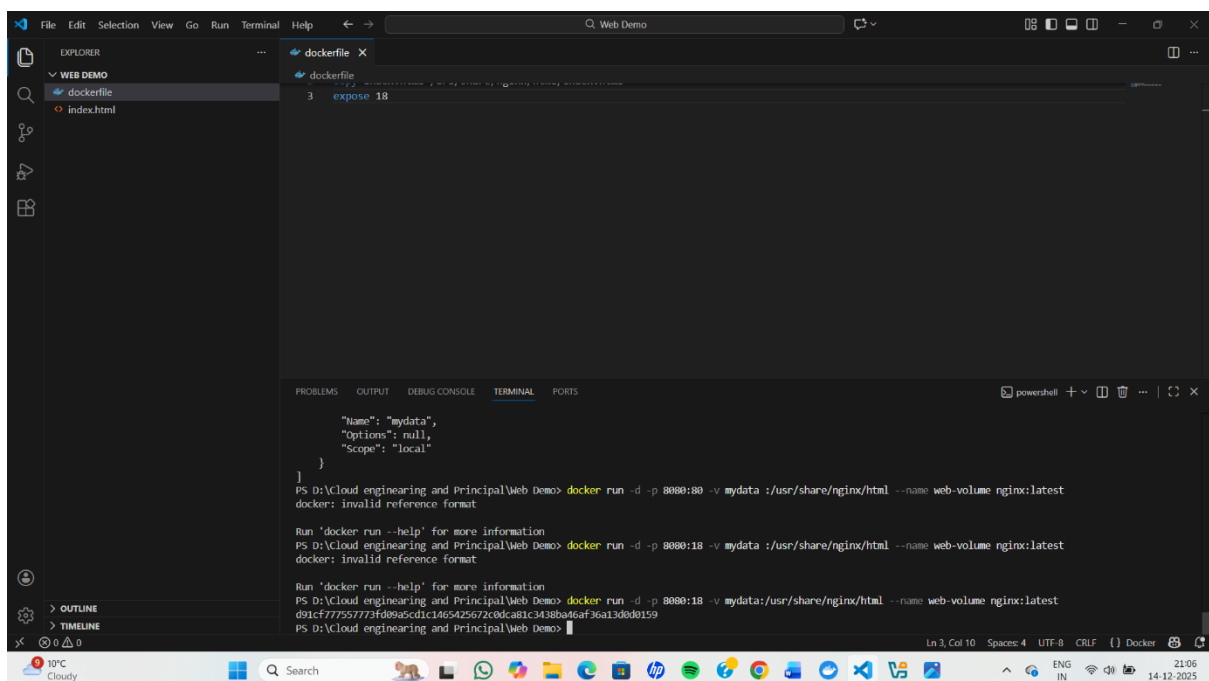


Figure 5. Volume Attached to a Container

2.5 Container Networking and Service Communication

The other fundamental characteristic of Docker is networking, which allows containers to interact securely and predictably. In default mode, Docker develops suspended networks, which enable the containers to define and engage one another without being revealed to external systems unless motivated otherwise. This behaviour embraces microservice

architectures, in which the various parts of an application are processes operating inside different containers [9].

Docker offers a number of networking modes, each applicable to its use case. A bridge network is the most widely utilised model, and it provides the isolated internal routing of containers on the same host. Placing such containers over a network can communicate by the help of predefined hostnames, making service discovery easier. Largely, classroom demonstrations involve forming a bridge network, placing several containers on it and watching the interaction between them as an integral system, e.g. an application container connecting to a database container.

Networking plays a critical role in the implementation of cloud deployment, as distributed systems rely on regulated communication channels. Service mesh or proxies, in their use, containers that are serving behind load balancers are based on known networking settings to ensure the security of their operation and performance. The implementation of these interactions is offered by the networking system provided by Docker, which makes containerised environments possible [10].

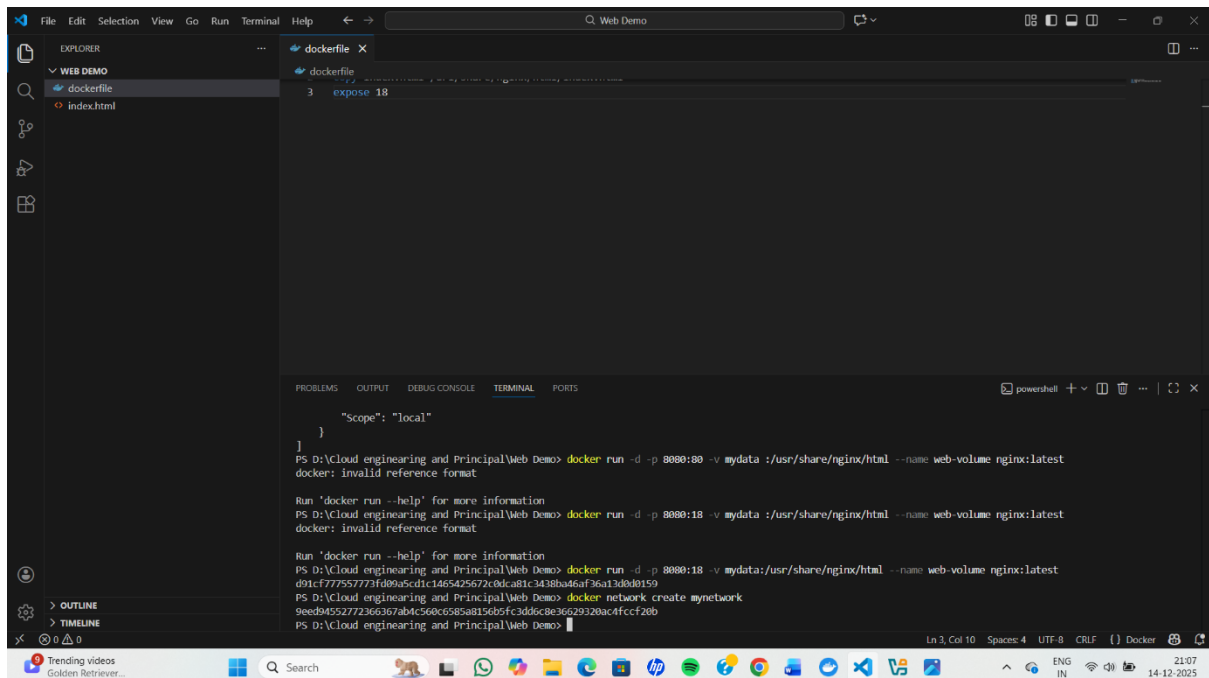


Figure 6. Docker Network Creation

```

dockerfile
3 expose 18

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell + - - - - -

docker: invalid reference format

Run 'docker run --help' for more information
PS D:\cloud engineering and Principal\Web Demo> docker run -d -p 8080:18 -v mydata:/usr/share/nginx/html --name web-volume nginx:latest
docker: invalid reference format

Run 'docker run --help' for more information
PS D:\cloud engineering and Principal\Web Demo> docker run -d -p 8080:18 -v mydata:/usr/share/nginx/html --name web-volume nginx:latest
d91cf7755773f80a5cd1c146542672c0dc811c3430ba4af36a13da9d159
PS D:\cloud engineering and Principal\Web Demo> docker network create mynetwork
9eed94552772366367abdc560c6585a815b5fc3ddc8e36629320ac4fccf20b
PS D:\cloud engineering and Principal\Web Demo> docker run -d --network mynetwork -p 8081:80 --name web1
PS D:\cloud engineering and Principal\Web Demo> docker run -d --network mynetwork -p 8081:80 --name web1 jay18082005/docker-web-demo
b6b704d08363113ef94f7c49392f0b05cfcfde3b97cfb86f91bdc3cbbdf272
PS D:\cloud engineering and Principal\Web Demo>
  
```

Figure 7. Docker Network Run

```

dockerfile
3 expose 18

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell + - - - - -

PS D:\cloud engineering and Principal\Web Demo> docker run -d --network mynetwork -p 8081:80 --name web1 jay18082005/docker-web-demo
b6b704d08363113ef94f7c49392f0b05cfcfde3b97cfb86f91bdc3cbbdf272
PS D:\cloud engineering and Principal\Web Demo> docker run -d --network mynetwork --name db -e MYSQL_ROOT_PASSWORD=Password123 mysql:latest
Unable to find image 'mysql:latest' locally
latest: Pulling from library/mysql
ac5a2aa7003a: Pull complete
ba83a91434a1: Pull complete
aff72f8f4e08: Pull complete
57bd38a2d740: Pull complete
7a5e1e917526: Pull complete
ba8d858f8b56: Pull complete
98d18218a1bb: Pull complete
f6a972aa365b: Pull complete
55f85a70891e: Pull complete
edf376e2970b1: Pull complete
Digest: sha256:f0036967257bf11aab7184e371920c5b629f0dd36cbe9dfaccd2ae18cc900dbd
Status: Downloaded newer image for mysql:latest
6e7c589a6dc5a1457e63fe151e12706c86b1cecc1cfe5dada3792ca2a5f9db603
PS D:\cloud engineering and Principal\Web Demo>
  
```

Figure 8. Docker Custom Network

The screenshot shows the Visual Studio Code interface with the Explorer view on the left showing a project named 'WEB DEMO' with files 'dockerfile' and 'index.html'. The main editor area shows the 'dockerfile' file with the content '3 expose 18'. The bottom panel shows the 'TERMINAL' view with the command 'PS D:\cloud engineering and PrincipalWeb Demo> docker network ls' and its output:

NETWORK ID	NAME	DRIVER	SCOPE
d1e6c448b46c	\	bridge	local
e6e5ac20b4de	bridge	bridge	local
7580dc2b42b5	host	host	local
54ca34d883d0	my_custom_network	bridge	local
c0e89c9c526e	mybridge	bridge	local
0d4d901b229	myjay	bridge	local
9eed94552772	mynetwork	bridge	local
ad88786310a6	none	null	local
52db9096ffe1	subnet-10.10.10	bridge	local

The terminal prompt is 'PS D:\cloud engineering and PrincipalWeb Demo>'.

Figure 9. Docker Networks List

Example: Static Web Application Image

The screenshot shows the Visual Studio Code interface with the Explorer view on the left showing a project named 'WEB DEMO' with files 'dockerfile' and 'index.html'. The main editor area shows the 'index.html' file with the following content:

```

1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8" />
5   <title>Docker web Demo - waleed</title>
6   <meta name="viewport" content="width=device-width,initial-scale=1.0" />
7 </head>
8 <body>
9   <div class="container">
10     <h1>NHS Data Protection & Cybersecurity Framework</h1>
11     <p class="lead">Visual diagram showing legal, sector-specific and standards-based controls.</p>
12     
13     <a class="btn" href="/static/nhs_flowchart.png" download>Download Diagram</a>
14   </div>
15 </body>
16 </html>

```

The bottom panel shows the 'TERMINAL' view with the command 'PS D:\cloud engineering and PrincipalWeb Demo> docker push jay18082005/docker-web-demo .' and its output:

```

docker: 'docker push' requires 1 argument
PS D:\cloud engineering and PrincipalWeb Demo> docker push jay18082005/docker-web-demo .
docker: 'docker push' requires 1 argument

```

The terminal prompt is 'PS D:\cloud engineering and PrincipalWeb Demo>'.

Figure 10. Docker index file

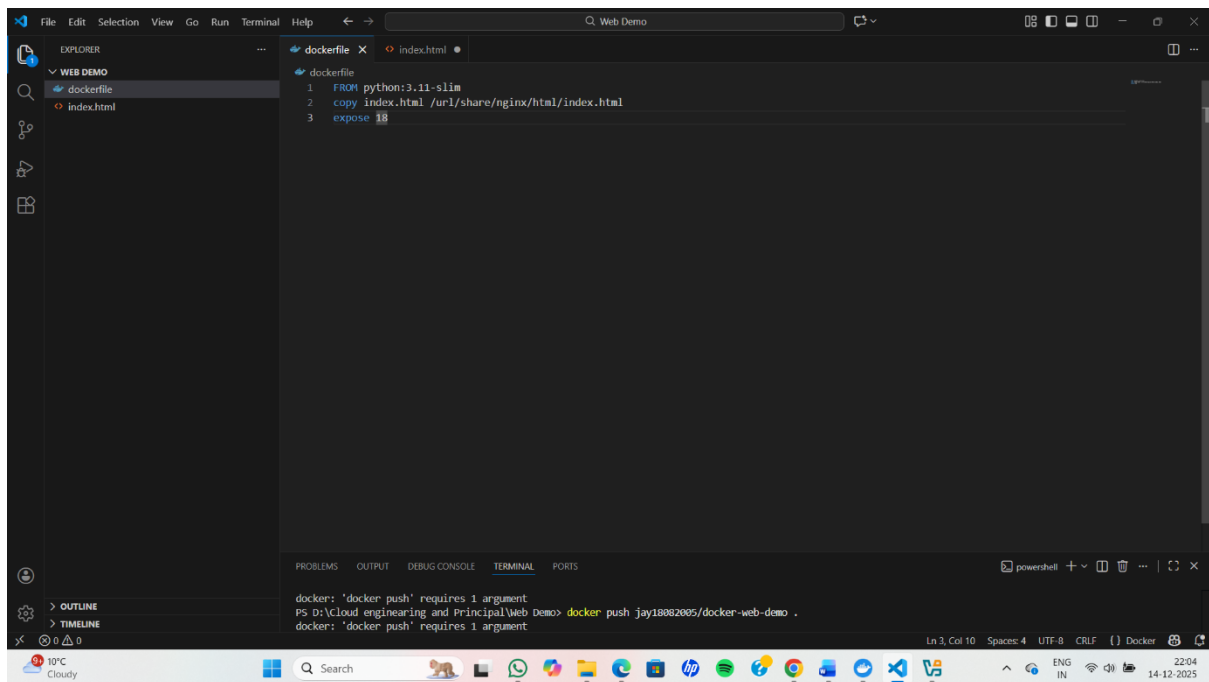


Figure 11. Dockerfile

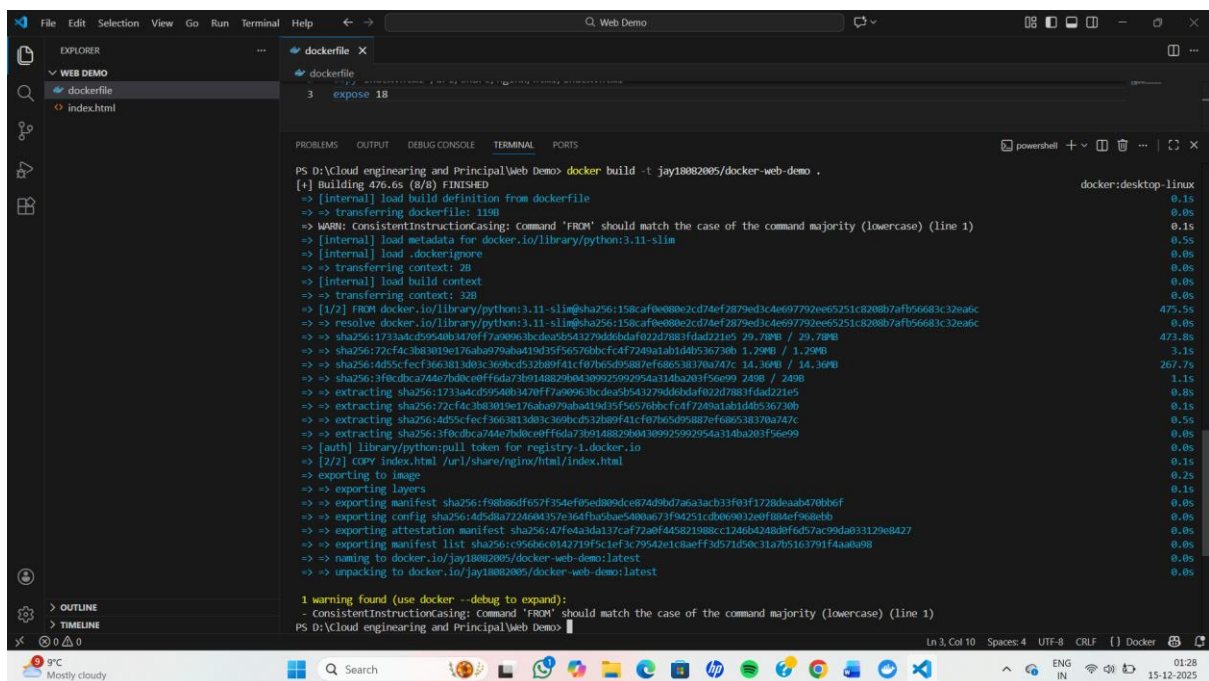


Figure 12. Docker Image Build

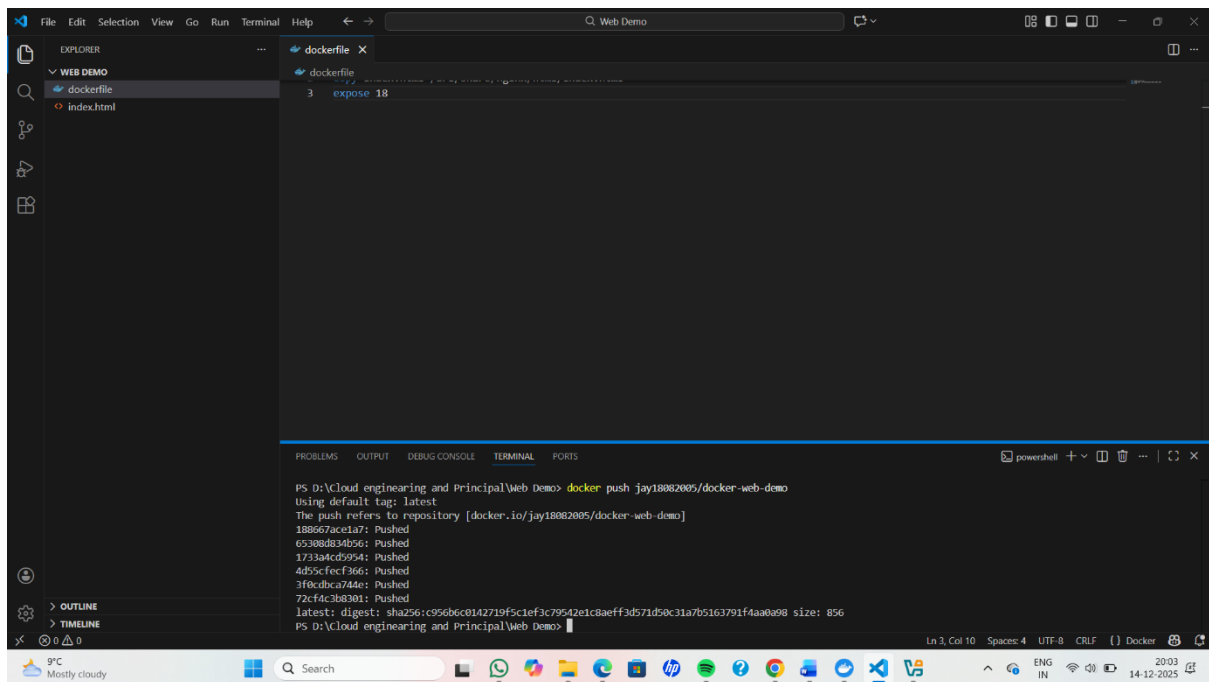


Figure 13. Docker Image push

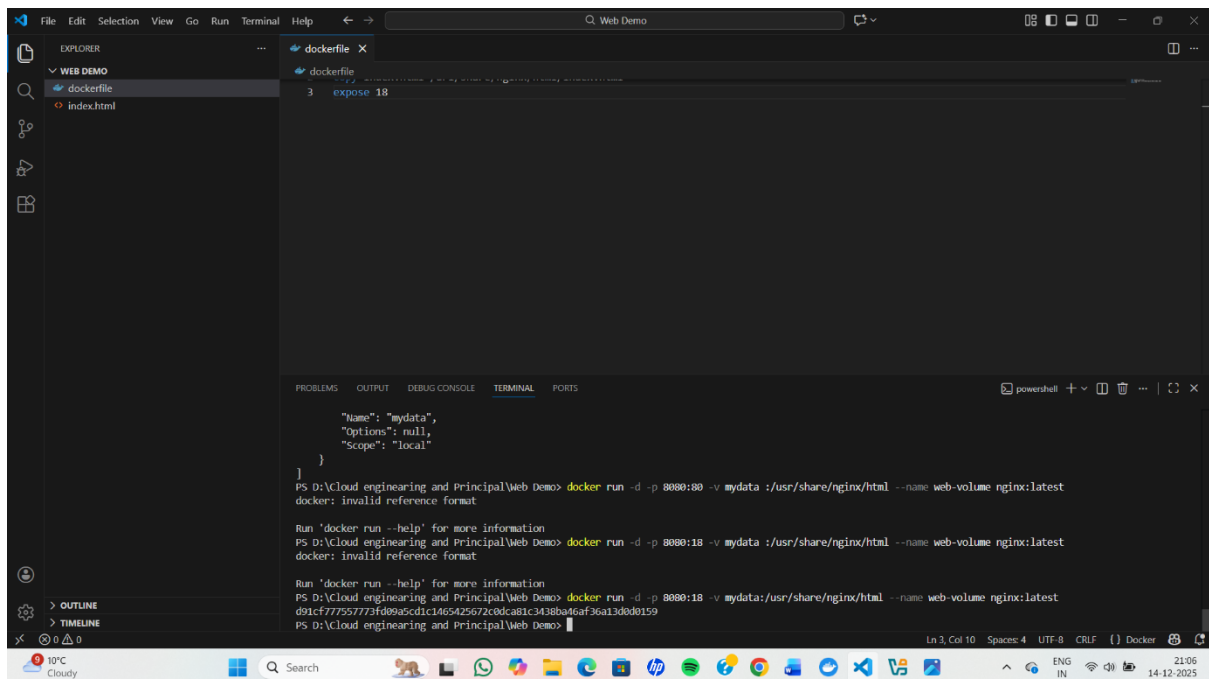


Figure 14. Docker Container Run

2.6 Combined Example: A Complete End-to-End Container Workflow

The combinations of these parts, like virtualisation, containerisation, images, storage and networking, can be explained by giving one entire process in full. Even though the steps

commonly entail the execution of commands, the conceptual workflow can be summarised in a set of descriptive paragraphs.

This is by starting to define a simple web application, using which a server has to show a custom HTML page. Applying a configuration file that defines the base server software and the files that are desired to be included in the end environment specifies the application environment. Upon execution of the build process, an image consisting of the web server software together with the HTML is built by Docker. The image can be used as a mobile embodiment of the app [11].

After the image has been constructed, it is executed as a container. This container operates in an isolated environment, though it is designed to expose the web server to the host system or external users. Through initialising the container and proxying its internal port of communication to an external interface, the web page is accessible on a local computer or open-source cloud computer. Here we can see that containerisation is able to wrap application logic and yet allow controlled access to the services.

A Docker volume can be mounted on the container to give it persistent storage, in case the web server will (e.g. by logging or storing user uploads) or will be used to hold configuration data. This allows important data to be maintained in case of container stoppage, restart and replacement.

Custom network characteristics come next, enabling the containerised web server to communicate with other services [12]. As an example, one classroom activity could be building an inner network and setting up two other containers: a database and an application layer. Every container is allowed to interact using set hostnames, and it shows how microservices perform communication in isolated network environments.

Lastly, when the image has been locally tested, it may be published to DockerHub. The steps at this level include authentication to the registry, assigning a version tag and uploading an image. By publishing the image, the application can be deployed on any cloud platform that is Docker-enabled. Image can be pulled via cloud orchestration tools directly out of the registry, and therefore, deployments in production can consistently and reproducibly be done.

This start-to-stop workflow shows the usefulness and theoretical strength of Docker. It demonstrates the packaging, execution, network and persistence, and distribution of an application with a toolset built in [13]. All of them, such as images, volumes, networks and

registries, add up to a cloud-ready deployment pipeline, which is associated with the sense of the contemporary industry best practices.

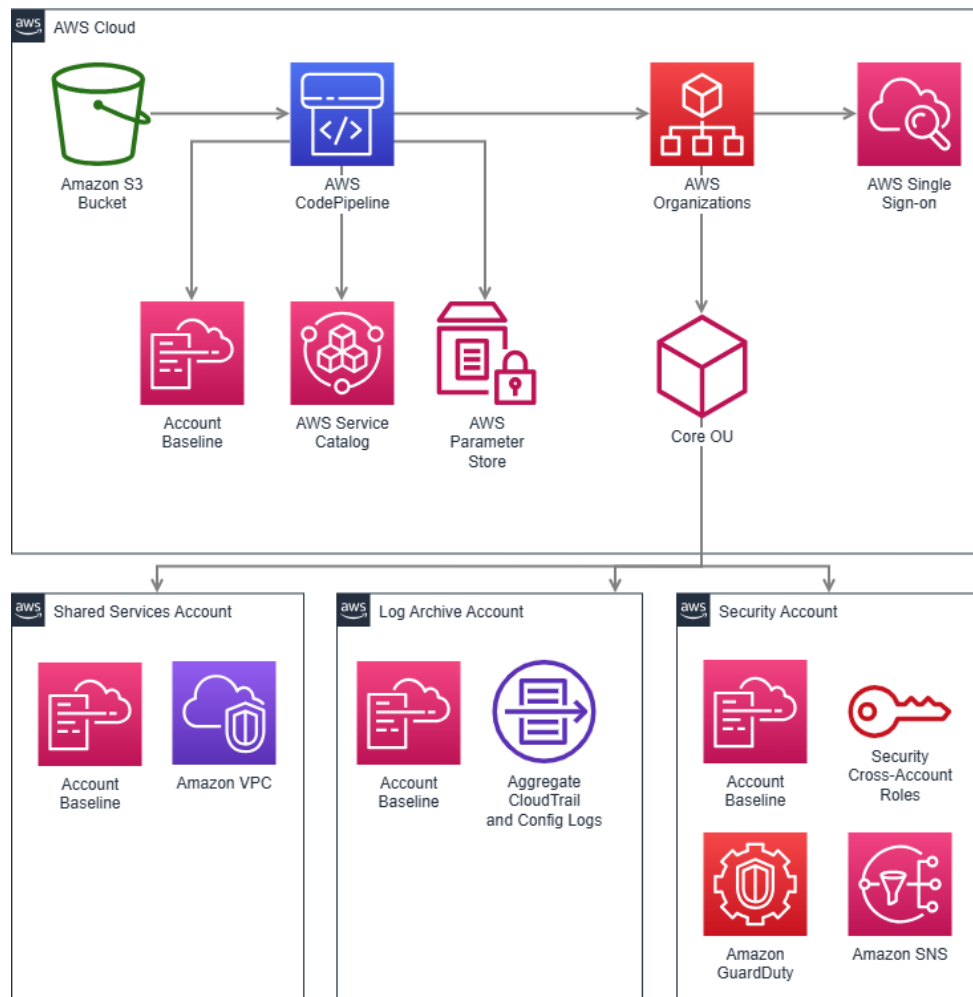
2.7 Video Demo

<https://www.youtube.com/watch?v=pF8r1WqEiVE>

<https://www.youtube.com/watch?v=pF8r1WqEiVE>

3. Cloud Computing Concepts

3.1 Load Balancing in Cloud Computing



Load balancing refers to the procedure of redistributing the incoming traffic or computation workload to a variety of servers or service instances in networks. In the absence of load balancing, a single server can be overloaded and hence experience slow response time, service failure or even total system crash. Load balancing is necessary in cloud environments, where a large and changing number of users might be involved with applications.

A load balancer is a device between clients and server backends. It accepts requests and chooses the server most suitable to serve that request out of such factors as server health, current

workload, geographic proximity or routing policy. This allocation makes sure that there is no server congestion with others not used effectively [14].

The cloud providers have provided managed load balancing services, where customers will not have to configure or maintain the infrastructure. As an illustration, one of the leading providers has introduced an application-level balancer that can deploy both HTTP and HTTPS traffic through smart routing guidelines. One of them offers a network-level balancer that is workload optimised for high-performance and low latency. These services are usually provided with health checks built to identify failed servers and automatically redirect traffic to the surviving instances, which helps the system remain resilient.

High availability can also be achieved through load balancing because the service will not be interrupted even when specific components fail. The load balancer monitors the failure of one of the servers and redirects the traffic to other healthy servers. This robotic reaction is capable of avoiding downtime and promoting cloud principles of resilience and elasticity.

3.2 Scalability in Cloud Infrastructure

Scalability means the ability of a system to respond to fluctuations in demand by adding or removing computing resources. This is one of the key benefits of cloud computing since an organisation can deal with times of high demand without necessarily investing in more hardware permanently [15].

Scalability can take two main aspects:

- **Vertical Scalability:** Vertical scalability is a type of scalability where the area capacity of a single server is expanded, e.g. by adding more memory or CPU resources. This solution is not complicated but has physical constraints, since hardware may be scaled up to a given extent.
- **Horizontal Scalability:** Horizontal scalability refers to adding the number of service instances or servers to an application. New machines are created instead of upgrading a single one to take away the workload. This style is already in proximity to distributed architecture and is highly convergent with containerisation and microservices.

Scalability is carried out by automated processes by the cloud providers. Auto-scaling groups are one approach that is commonly used. These groups observe demand dynamically, e.g. CPU utilisation, memory usage or request rate, and dynamically add or delete instances accordingly. Given that traffic may be high or low, the system is scaled to increase or decrease the number

of servers, respectively [16]. This active adaptation guarantees the stable performance and consumption of the resources.

Scalability is also commonly accomplished by adjusting the number of containers that are running in the environment where a container orchestration platform is applied. Orchards automatically measure metrics and procure more containers when needed. This fast and light-scaled characteristic is especially important to the microservices that have to scale to single components regardless of the other components.

Scalability has advantages more than performance. It also promotes the effectiveness of finances since clients do not pay for resources that they do not utilise. Scalable cloud systems enable systems to run at the optimum capacity without the costly capital outlay that was common with the ownership of dedicated hardware.

3.3 Application Gateways and Intelligent Traffic Management

Application gateways are used at the application layer within the network stack, which allows intricate routing and security capabilities as well as traffic inspection. In contrast to a simple load balancer, which passes traffic without inspecting the content, an application gateway can inspect traffic at a more sophisticated level, and then the system may apply rules to use URL paths, hostnames, protocols or particular application actions [17].

Application-level routing is considered one of the main purposes of an application gateway. Indicatively, an organisation could have multiple microservices that are behind one gateway. The requests to a login page could be sent to an authentication service, and requests to product information could be sent to a product catalogue service. The decision-making process of the routing decisions in the gateway is governed by predefined rules, which make the application architectures modular and well-structured.

The application gateways also have strong security features. Web Application Firewall (WAF) features are offered by most cloud providers as a part of the gateway service. The WAF looks into the request patterns and blocks threats like SQL injection, cross-scripting and malicious bot traffic. This security level is particularly necessary in applications that are open to the network for the general public.

Some other advantage is that of terminating the connections, where the gateway takes care of encryption and decryption of the incoming connections. By transferring this task to the backend servers, the server processing load will be reduced and simplify the certificate management [18].

The application gateway is a managed service that is implemented by cloud providers to ensure that high amounts of traffic in the application layer are automatically scaled. This provides reliability even during peak load conditions and also gives developers a chance to concentrate on application logic instead of the issue of traffic management.

3.4 Why Load Balancing and Scalability Are Complementary Concepts

Load balancing and scalability are terms that are used in tandem, as each one of them fills the shortcomings of the other. They create a vicious cycle that is crucial to the sustainability of performance and availability in the cloud space.

Load balancing will never guarantee a performance of a sufficient level should there not be enough backup servers. With distributed routing, there is still a finite number of servers that can be overloaded because of sudden demand spikes. Scalability will handle this drawback by allowing the number of servers to automatically increase as the demands are met (Patel, 2024). On the other hand, load balancing would be useless for scalability. In the case of the addition of new instances, and yet the requests keep coming to a single server, then the system would be prone to bottlenecks or failure. Load balancers make sure that the newly added resources get their quota of incoming traffic to enable the system to make the most out of the expanded capacity.

These two mechanisms combined form an elastic architecture in which:

- The demand increases the resources.
- Traffic is spread among the available resources.
- It is an automatic system that is self-adjusting.
- The performance and availability are consistent with the changes in the workload.

This balanced interplay is the core of cloud-native design, allowing applications to manage unpredictable levels of traffic with resilience and efficiency.

3.5 Benefits of Combining Load Balancers, Scalability and Application Gateways

Scalability and application gateways have been integrated when load balancing, they offer a fully infrastructure solution that covers all issues of performance, reliability, security and cost efficiency at the same time [20]. Their aggregate value is more than the added value of their capabilities.

Enhanced Performance and Reliability:

Load balancers are used to make sure that the requests are distributed evenly so that one server does not become overloaded. Scalability makes sure that supplementary resources can be made

when required. Application gateways optimise traffic in consideration of application-specific policies and improve routing information. Collectively, these technologies minimise the chances of slowdowns or outages by a significant factor, even at the peak times of traffic.

Improved Security and Compliance:

Application gateways are provided with newer security capabilities, including WAF protection, termination of the SSL protocol and protocol inspection. There is the protection of the backend services that would be directly exposed to the internet, which limits the attack surface. The balance load balancers and auto-scaling also contribute to enhancing security due to their ability to offer stability to the system and minimise chances of failure or vulnerability [21].

Cost Efficiency and Resource Optimisation:

Scalability saves unwarranted expenditures through dynamically allocating resources. Load balancers activate the fact that resources are exhausted by the employees before new ones are established. Application gateways also simplify routing as well as minimise redundant traffic to back-end systems. These efficiencies lead to a reduction in operational costs and predictable expenditure trends.

Support for Modern Application Architectures:

These three technologies are critical in microservices, serverless functions and container orchestration platforms. Application gateways provide routing to various independent services, load balancers distribute hits to duplicated instances of service and scale to provide each service part to grow based on its individual workload needs. The alignment of cloud-native applications promotes large degrees of modularity, maintainability and performance [22].

4. Main Driving Forces Leading to the Expansion of Cloud

4.1 Cost Efficiency and Reduction of Capital Expenditure

Reduction in cost is one of the best incentives to use the cloud. Conventional On-premises computing involves organisations spending a lot of money on the physical servers, network devices, air conditioning devices, security measures and data centre warehouse. These are mainly capital expenditures, which are initial items and the hardware frequently is not fully used since it will have to be sized to meet peak demand even when that peak is not very frequent.

Cloud computing substitutes this model with a consumption-based pricing system. The resources (billed) are based on the actual usage, and hence organisations can scale down and scale up when it is necessary. This saves the cost of buying hardware that is costly and the

maintenance costs that are incurred on day-to-day operations, including power usage, repairs, upgrades and maintenance staff [23]. Capital expenditure is replaced with operational expenditure; hence, investment businesses will manage budgets at reduced financial risk.

Moreover, scalability realised by the cloud providers is through the operation of huge distributed data centres. These economies of scale are conveyed into the hands of the customers, and even small organisations can enjoy the benefits of having high-performance computing at a small fraction of the cost they used to pay. As companies strive to become leaner in operations and gain more predictability in their financial processes, one of the primary sources of cloud computing development is the economic appeal of cloud computing.

4.2 Flexibility, Scalability and Rapid Provisioning

Flexibility provided by means of on-demand resource provisioning is another important aspect of cloud growth. The cloud systems give organisations the ability to launch computing resources within minutes instead of days or weeks. This is critical in fast-paced business organisations where the needs often vary, and organisations are required to take responsive actions to new opportunities or challenges [24].

This flexibility is based on scalability. The cloud services are automatic by adjusting the level of resources to adapt to changing workload demands. Organisations can no longer plan that their systems can manage a maximum load, but use the mechanisms of elastic scaling, which add more resources when the need arises. This guarantees maximum performance with no over-provisioning, which is wasteful.

The flexibility is also demonstrated in the broad availability of the various services provided by the cloud platform. In addition to virtual machines, providers also provide managed databases, analytics solutions, artificial intelligence, messaging, and monitoring and others. These services can be incorporated into the applications of the organisations without necessarily developing or maintaining them separately.

The freedom in this scale not only fast-tracks development but also assists in experimentation and innovation. New ideas can be experimented with at a rapid pace, prototype systems can be developed, or temporary settings put in place without significant investment. Consequently, flexibility in the clouds has become a formidable force that keeps it growing in industries [25].

4.3 Innovation Through Automation and DevOps Practices

Cloud computing has transformed the nature of software development, deployment and maintenance. This transformation focuses on automation. Such practices as continuous

integration, continuous delivery and infrastructure as code are supported by cloud environments and help organisations to automate deployment pipelines, configuration processes and scaling operations.

The automation helps a great deal in the way the operations are run and lowers the chances of human error. Rather than manually deploying servers of their own, groups specify what they require in their infrastructure as declarative, interpreted, and executed uniformly by cloud providers. This enables the deployment of applications in more than one environment predictively.

Cloud is also very compatible with the DevOps practice, which focuses on the cooperation between operations and development teams. Containers, orchestration systems and automated workflow tools are more effective in a cloud environment and allow quick release cycles and continuous improvement of the system [26].

With the emphasis on speed and reliability in software delivery continuing to be critical concerns of any business, the automation and DevOps approaches supported by the cloud are also a significant impetus in adopting the cloud.

4.4 Reliability, Availability and Global Reach

Cloud vendors also design systems with great degrees of reliability and availability that would otherwise be impractical for individual organisations on their own systems. There are centres that are spread through various geographical areas with redundancy and failover services that reduce and limit service interruptions. In case one region goes offline, the traffic can be diverted to the other regions, and as a result, the applications may be made available.

Other facilities embedded in cloud environments are monitoring, backup systems, distributed storage facilities, as well as disaster recovery tools. These characteristics promote the continuity of the business and lessen the risk in business. The international nature of cloud data centres provides organisations with a chance to bring applications nearer to users, enhancing the latency, as well as overall performance [27]. It is crucial for companies operating in foreign markets or providing services to customers in different locations.

As organisations increasingly depend on 24/7 service provision, the cloud's capacity to provide credible, distributed infrastructure worldwide has encouraged its use.

4.5 Security, Compliance and Managed Services

Cloud providers have made a lot of investment in the security infrastructure using sophisticated methods like encrypted data storage, identity management, network segmentation and constant

monitoring. Most organisations have realised that cloud environments are more secure as compared to on-premises systems, especially where internal security management resources are meagre.

Moreover, cloud providers enable adherence to the key regulatory frameworks, and the tools and certifications that help organisations adhere to the industry standards are provided. Intelligence of managed security services, including firewalls, intrusion detection systems and threat intelligence platforms, makes it easier to protect digital assets, which is a complex undertaking [28].

One of the reasons why cloud computing would be so appealing is the aspect of outsourcing security to trusted cloud providers, so that small and medium-sized enterprises that do not have the resources to support complex security infrastructures can still adopt the cloud computing model.

5. Conclusion

Cloud computing has transformed the new infrastructures in the digital world by providing flexible, affordable and scalable solutions that meet the changing requirements of the organisations. The technologies that have been at the centre of this change are Docker, which allows the application to operate reliably in various environments, as well as improve performance through lightweight containerisation. Docker can be used to apply powerful cloud-native development and deployment cultures through the principles of virtualisation, container networking, persistent storage and image distribution. Of no less significance are the capital architecture concepts that cloud systems are based upon. With load balancing, scalability, and an application gateway, the services can offer performance and resilience to changing demands with security and availability. Their combined use comes as a trend to elastic, automated, and smart infrastructures of the cloud. Cost reduction, operation flexibility, global coverage, enhanced reliability and strong security are some of the factors that have triggered the increased adoption of the cloud. The recent emergence of new paradigms, such as serverless computing, allows this trend to increase even further, as it minimises the task of providing infrastructure control and allows extremely efficient execution patterns. Comparing serverless functions with containerisation, the variety of possible options to develop cloud-native applications is both noticeable and clear. Collectively, these technologies represent the ever-present innovation that forms the future of cloud engineering and the provision of digital services.

References

- [1] D. C. Marinescu, *Cloud Computing: Theory and Practice*. Morgan Kaufmann, 2022.
- [2] D. Comer, *The Cloud Computing Book: The Future of Computing Explained*. Chapman and Hall/CRC, 2021.
- [3] G. Shyam, *Cloud Computing: Concepts and Technologies*. CRC Press, 2021.
- [4] N. Brooks, C. Vance, and D. Ames, “Cloud computing: A review of evolution, challenges, and emerging trends,” *Journal of Computer Science and Software Applications*, vol. 5, no. 4, 2025.
- [5] S. Agrawal and D. Singh, “Study containerization technologies like Docker and Kubernetes and their role in modern cloud deployments,” in *Proc. 9th IEEE Int. Conf. for Convergence in Technology (I2CT)*, Apr. 2024, pp. 1–5.
- [6] D. Santana and A. Malik, *Cloud Computing Demystified for Aspiring Professionals: Hone Your Skills in AWS, Azure, and Google Cloud Computing*. Packt Publishing Ltd., 2023.
- [7] S. P. Bandaru, *Cloud Computing for Software Engineers: Building Serverless Applications*, 2023.
- [8] S. S. Sangapu, D. Panyam, and J. Marston, *The Definitive Guide to Modernizing Applications on Google Cloud*. Packt Publishing Ltd., 2022.
- [9] N. B. Ruparelia, *Cloud Computing*. MIT Press, 2023.
- [10] A. Sapkal and S. S. Kusi, “Evolution of cloud computing: Milestones, innovations, and adoption trends,” 2024.
- [11] D. Gupta, *The Cloud Computing Journey: Design and Deploy Resilient and Secure Multi-Cloud Systems*. Packt Publishing Ltd., 2024.
- [12] J. S. Murthy, G. M. Siddesh, and K. G. Srinivasa, Eds., *Cloud Security: Concepts, Applications and Practices*. CRC Press, 2024.
- [13] D. Temgire, V. Thorat, V. Raut, A. Babar, and S. Yadav, “Cloud-based solution architecture for placement cell web-based application,” *IJSAT–International Journal on Science and Technology*, vol. 16, no. 2, 2025.
- [14] G. Agarwal, *Modern DevOps Practices: Implement, Secure, and Manage Applications on the Public Cloud*. Packt Publishing Ltd., 2024.
- [15] R. V. S. Praveen, *Data Engineering for Modern Applications*. Addition Publishing House, 2024.

- [16] B. R. Cherukuri, “Containerization in cloud computing: Comparing Docker and Kubernetes for scalable web applications,” *International Journal of Scientific Research and Application*, vol. 13, no. 1, pp. 3302–3315, 2024.
- [17] V. U. Ugwueze, “Cloud native application development: Best practices and challenges,” *International Journal of Research Publication and Reviews*, vol. 5, no. 12, pp. 2399–2412, 2024.
- [18] J. Khan, W. Liang, B. J. Mary, F. Hamzah, B. John, and M. Blessing, “The evolution of ETL processes in the age of cloud computing and microservices: From traditional batch loading to serverless data integration,” 2025.
- [19] D. Patel, “The role of Amazon Web Services in modern cloud architecture: Key strategies for scalable deployment and integration,” *Asian Journal of Computer Science Engineering*, vol. 9, no. 4, pp. 1–9, 2024.
- [20] M. Beshley, M. Klymash, I. Scherm, H. Beshley, and Y. Shkoropad, “Emerging network technologies for digital transformation: 5G/6G, IoT, SDN/IBN, cloud computing, and blockchain,” in *Proc. IEEE Int. Conf. on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering*, Feb. 2022, pp. 1–20.
- [21] A. M. Ahmed and A. H. Fatima, “The hybrid role: Bridging cloud engineering and security practices for enhanced protection,” *International Journal of Trend in Scientific Research and Development*, vol. 6, no. 1, pp. 1590–1598, 2022.
- [22] M. Saqib, S. Malhotra, D. Mehta, J. Jangid, F. Yashu, and S. Dixit, “Optimizing spot instance reliability and security using cloud-native data and tools,” *arXiv preprint arXiv:2502.01966*, 2025.
- [23] S. Gajula, “A decade of cloud-native technologies: Multi-cloud strategies, Docker, and CI/CD pipelines,” *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 16, no. 4, pp. 75–81, 2025.
- [24] J. Krško, A. Kanáliková, P. Kovačovič, G. Gašpar, V. Šimák, and P. Glajza, “Cloud services and security,” in *Proc. 26th Int. Carpathian Control Conf. (ICCC)*, May 2025, pp. 1–6.
- [25] N. Agrawal, R. Kumar, and S. Tapaswi, *Cloud Computing Security: Strategies and Best Practices*. CRC Press, 2025.
- [26] R. Srivastava, *Cloud Native Microservices with Spring and Kubernetes*. BPB Publications, 2021.

- [27] A. Elnur, “How cloud-native architectures revolutionize software development,” *Universum: Technical Sciences*, vol. 5, no. 3 (132), pp. 56–62, 2025.
- [28] H. Rehan, “Revolutionizing America’s cloud computing: The pivotal role of AI in driving innovation and security,” *Journal of Artificial Intelligence General Science*, vol. 2, no. 1, pp. 239–240, 2024.