

CLOUD ENGINEERING PRINCIPLES AND PRACTICE

Virtualization, Docker Containerization and Cloud-Native Service Architecture

A critical report supported by an end-to-end Apache web deployment

Student name	Md Eyamin Molla
Student ID	2425028
Programme	MSc Computing Portfolio
Module	Cloud Engineering Principles and Practice
Module code	ACCA7027
Lecturer	Mark Huntly
Submission status	Second-attempt submission

Table Of Contents

Executive Summary	3
1. Introduction.....	3
2. Virtualization and Containerization Foundations	4
2.1 Principles of virtualization	4
2.2 Principles of containerization	4
2.3 Container image lifecycle and security	6
3. Practical Docker Implementation	6
3.1 Aim, artefact and evidence strategy	6
3.2 End-to-end workflow	7
3.3 Building and verifying the image	7
3.4 Running the container and publishing the port	8
3.5 Operational verification through logs	10
3.6 Docker Hub publication.....	11
3.7 Named volume and persistence.....	12
3.8 User-defined network and service discovery.....	13
3.9 Requirement coverage and critical evaluation	15
4. Load Balancing, Scalability and Application Gateways.....	15
4.1 Application scenario and provider selection	15
4.2 Application Gateway.....	16
4.3 Load balancing.....	16
4.4 Scalability and elasticity	17
4.5 Why load balancing and scaling are complementary.....	17
4.6 Benefits and limitations of combining all three	17
5. Driving Forces Behind Cloud Expansion and Serverless Computing	18
5.1 Economic and operational driving forces	18
5.2 Serverless functions	18
5.3 Containerization versus serverless functions.....	19
5.4 Recommended hybrid design	19
5.5 Practical limitations and production improvements	20
6. Conclusion	20
References	20
Appendix A. Command Sequence Used in the Practical.....	22
Appendix B. Video demonstration.....	23

Executive Summary

This report compares virtualization, the use of containers, and the design of web applications using the cloud-native approach, using a practical example of an Apache web application. The image was created in the Dockerfile, the container was mapped with a host-to-container port mapping, the logs contained an "HTTP 200" response, and the image was pushed to Docker Hub. A named volume retained the page after the first container was removed, and a user-defined network enabled the web container to communicate with the Alpine container by name. These results include the areas of required practice.

The report also proposes to analyze load balancing, scalability and application gateways of a proposed clinic service on Microsoft Azure. Application Gateway is capable of Layer-7 routing and supports TLS termination and optionally WAF protection, and autoscaling can adjust the number of healthy replicas according to the demand. Azure Load Balancer is only provided if a dedicated private Layer-4 TCP/UDP service is needed. Lastly, the report looks at cost, flexibility, rapid provisioning, elasticity, and managed services as factors that are fueling the growth of cloud. The web and API tiers are recommended to be deployed on containers, and serverless functions are recommended for short, background tasks that are triggered by events.

1. Introduction

By using cloud computing, organizations can only use the resources that they need when they require them and remove them when they no longer require them. NIST defines it as "on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service" [1]. So cloud engineering isn't just about cloud hosting. It is a mix of abstraction, automation, repeatable deployment, and controlled consumption. Virtualization is a good solution to this model, which would otherwise require multiple physical servers to be installed. Along with packaging comes operating-system-level packaging with containerization, meaning that it is possible to deliver an application and dependencies in one image. As of yet, containers are generally lighter than full virtual machines, but they also need to be secure, have controlled networking connectivity, persistent storage, logging, and patching [2], [3]. The purpose of this report is to have three goals. First, it's a description of virtualization and containerization. Second, the evaluation of the practicality of Docker is carried out. Thirdly, it scales, balances, and utilizes gateways and serverless computing for a service that is customer-facing. It is used in a limited manner in practice. The HTML page is an educational dashboard and is not used to process patients' data. Claims of monitoring, TLS, role-based access control, and scanning are listed as proposed controls as the screenshots don't demonstrate their configuration.

2. Virtualization and Containerization Foundations

2.1 Principles of virtualization

Virtualization is a technique that makes physical systems separate from logical systems. A hypervisor allocates CPU, memory, storage, and networking resources to the virtual machines. Guests OS are installed on each virtual machine. Type 1 and Type 2 hypervisors are located in the hardware and above the host OS, respectively. This helps with the utilization and isolation but also introduces operating system, storage, and patching overheads [2].

Figure 1 illustrates that multiple virtual machines are running on one physical server with the help of a hypervisor. This is a well- suited model for cloud resource pooling due to logically separated workloads. The hypervisor is included in the trusted platform, though. Hence, there is a need to limit management access and patch and monitor the virtualization layer [2].

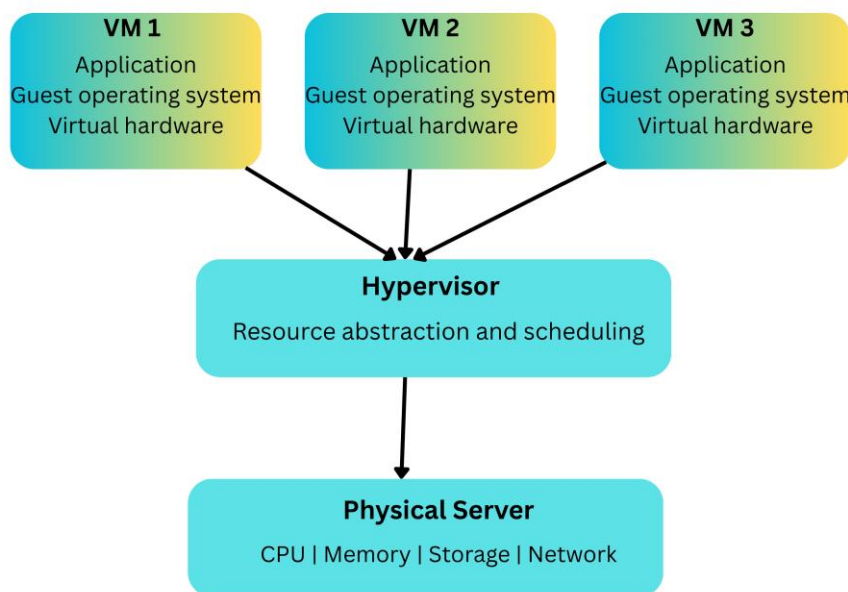


Figure 1. Virtualization of shared physical hardware (Source: Author's illustration based on [2]).

2.2 Principles of containerization

Containerization bundles an application and the application's user-space requirements and shares the host kernel. The image is a template (a layer) and a container is a running instance of the image. Namespaces isolate processes, networks, and mount points, and control groups bound resources. This allows containers to get started sooner and require fewer resources than a full guest operating system [4].

The primary advantage is portability, and repeatability and deployment times are also key. The image can be deployed on any host that is compatible with Docker; the steps used to create the image are documented in the Dockerfile. This helps to facilitate automation, CI, and horizontal scaling. But this portability is not complete, as CPU architecture, kernel capabilities, external services, secrets, and storage have some impact on the deployment [4], [26].

The two models are compared to each other in Figure 2. Each virtual machine will typically have its own guest kernel and thus a better boundary of isolation. The containers share the host kernel and, therefore, are lighter. In reality, both can be combined, for example, containers that are deployed inside virtual machines, on a managed cloud platform.

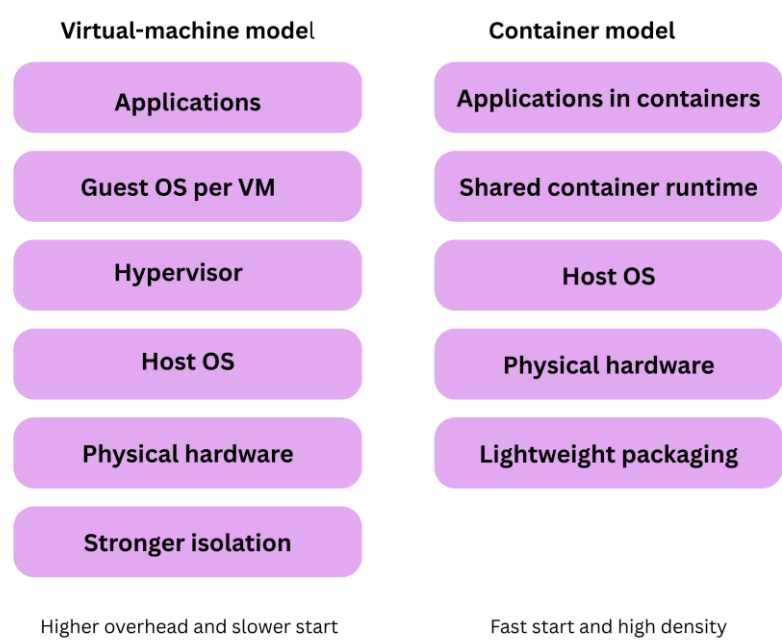


Figure 2. Architectural comparison of virtual machines and containers (Source: Author's illustration based on [3], [4]).

Table 1 summarizes the main operational differences. The better choice depends on isolation, operating-system needs, workload type and deployment speed.

Criterion	Virtual machines	Containers	Critical implication
-----------	------------------	------------	----------------------

Isolation	Separate guest kernel	Shared host kernel with namespaces and cgroups	VMs often give stronger default isolation; containers need hardened runtime controls.
Start-up	Full guest OS; higher overhead	Starts from an image; lower overhead	Containers suit rapid deployment and replication.
OS flexibility	Can run different guest operating systems	Needs a compatible host kernel	VMs suit legacy or mixed operating-system needs.
Packaging	OS and application in a machine image	Application and user-space dependencies in an image	Container images are easier to distribute, but base images still need governance.
Persistence	Virtual disks can follow the VM lifecycle	Writable layer is disposable; volumes hold state	Durable state should be kept outside replaceable containers.
Typical use	Legacy systems, full OS control, stronger isolation	Microservices, CI/CD and portable delivery	A cloud platform may combine both models.

Table 1. Critical comparison of virtual machines and containers (Source: Author's synthesis based on [2]–[4], [26]).

2.3 Container image lifecycle and security

The Dockerfile contains instructions in order to build an image. Apache base image was chosen with the FROM directive, the index.html file was placed in the document root of the Apache base image with the COPY directive, and the EXPOSE directive was used to describe the Apache base image service port. However, 'EXPOSE' does not publish the port to the host, and access to the host is created at runtime using the '-p' option [5], [8], [9].

An image tag is a human-readable reference, and a digest identifies the exact image content. For production, teams should employ trusted base images, scan for vulnerabilities, rebuild after security updates, and limit access to the registry. Other recommendations for image trust, least privilege, and host hardening and continuous monitoring are made by NIST. The practical is for creating and publishing images and does not show image scanning or signing.

3. Practical Docker Implementation

3.1 Aim, artefact and evidence strategy

The goal of the practical was to package up a static HTML page in Apache, expose it to a local browser, and publish the image to Docker Hub. There are two files in the project folder: 'Dockerfile' and 'index.html'. Each of the source files is associated with the build instructions as shown in Figure 3.



Figure 3. Project structure and Apache-based Dockerfile used for the build (Source: Author's own practical work, 2026).

FROM httpd:2.4-alpine

COPY index.html /usr/local/apache2/htdocs/index.html

EXPOSE 80

Since the official image for httpd already has Apache installed [6] the Dockerfile is short. The custom layer is just to add the page, so it can be repeated easily. Alpine also maintains a base image that is as small as possible. Patches are still necessary for small images, though. Also, the page fetches Tailwind CSS from a public CDN, which should be replaced by the necessary CSS in a production version, and a content-security policy should be used.

3.2 End-to-end workflow

The process began with the HTML page along with the Dockerfile. We created a reusable image using Docker build, and a container was created from the image using the command docker run. The host port 8091 was redirected to port 80 of Apache, and the page was browsed. Then, once verified, the same image was tagged and uploaded to Docker Hub. So, it was the same image that was the stable deployment unit during the entire practical.

3.3 Building and verifying the image

The image has been created using the following command: `docker build -t eyamin-cloud-ops-apache:v1 .`. The last dot selected was the current directory that was used as the build context. Docker downloaded the Apache base image, copied the HTML file, and created the image named v1. The successful build is shown in Figure 4(a), and the check of the image in the local store is shown in Figure 4(b).

```
Windows PowerShell
PS E:\Assignment\Eyamin\clinic-cloud-dashboard> docker build -t eyamin-cloud-ops-apache:v1 .
[+] Building 17.5s (8/8) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 127B
=> [internal] load metadata for docker.io/library/httpd:2.4-alpine
=> [auth] library/httpd:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/2] FROM docker.io/library/httpd:2.4-alpine@sha256:968c8b4098fcec473762b45f6c541a
=> => resolve docker.io/library/httpd:2.4-alpine@sha256:968c8b4098fcec473762b45f6c541a
=> sha256:b989da80e2107a23042852f28ec7a4b2558251f2e4846a22ee054968 11.07MB / 11.07MB
=> sha256:4f4fb708ef54461cf82571ae8db9a0dc1e0c8b557708a6d75e68dc38e8acc1 32B / 32B
=> sha256:7b3c5d639c1194129f70ca528c7c5aa08d1ada08fc8d8c16423b51de3e 5.90MB / 5.90MB
=> sha256:4ca6f6797e45f445f90851d91732e15b07e8990f66fd0aee8106ace2cebb37 286B / 286B
=> sha256:719e56c8c985fd8fb6194fe05949486f3e41afc09e04ab9abaffcbbb5df9c 144B / 144B
=> sha256:41c15f5ba5843d4eed3ceeb41954062982e98c04a43aa6783d25dc27d4c194 930B / 930B
=> extracting sha256:41c15f5ba5843d4eed3ceeb41954062982e98c04a43aa6783d25dc27d4c194
=> extracting sha256:719e56c8c985fd8fb6194fe05949486f3e41afc09e04ab9abaffcbbb5df9c
=> extracting sha256:4f4fb708ef54461cf82571ae8db9a0dc1e0c8b557708a6d75e68dc38e8acc
=> extracting sha256:b989da80e2107a23042852f28ec7a4b2558251f2e4846a22ee054968e6362c2
=> extracting sha256:7b3c5d639c1194129f70ca528c7c5aa08d1ada08fc8d8c16423b51de3e979ac
=> extracting sha256:4ca6f6797e45f445f90851d91732e15b07e8990f66fd0aee8106ace2cebb37e
=> [internal] load build context
=> => transferring context: 11.95kB
=> [2/2] COPY index.html /usr/local/apache2/htdocs/index.html
=> exporting layers
=> exporting manifest sha256:f9af4dd4fb5903d7545fb7c2e5da8e55ff73e7934595a98ed1d25c49
=> exporting config sha256:8b5d944711e1249794f38b454c17444f23f77e9cf984add7aa3073a17
=> exporting attestation manifest sha256:3dd0aebfb48483768312c099238083693ab5f1b175
=> exporting manifest list sha256:fa0b33ffb3c3a97bd1268d2ff208d4de38cb2b7ee129bd6afa
=> naming to docker.io/library/eyamin-cloud-ops-apache:v1
=> unpacking to docker.io/library/eyamin-cloud-ops-apache:v1
PS E:\Assignment\Eyamin\clinic-cloud-dashboard>
```

(a) Successful image build from the Dockerfile

```
Windows PowerShell
PS E:\Assignment\Eyamin\clinic-cloud-dashboard> docker images
IMAGE ID DISK USAGE CONTENT SIZE EXTRA
eyamin-cloud-ops-apache:v1 fa0b33ffb3c3 96.7MB 20.8MB
PS E:\Assignment\Eyamin\clinic-cloud-dashboard>
```

(b) Verification of the local image and v1 tag

Figure 4. Successful image build and verification of the local v1 image (Source: Author's own practical work, 2026).

docker build -t eyamin-cloud-ops-apache:v1.

docker images

The output also connects the Dockerfile with the resulting image as it contains the base-image step, build context and the 'COPY' instruction as well as the image name. A production pipeline would introduce automated tests, a software bill of materials and an immutable release tag, as well as vulnerability scanning. The following controls were not included in the practical.

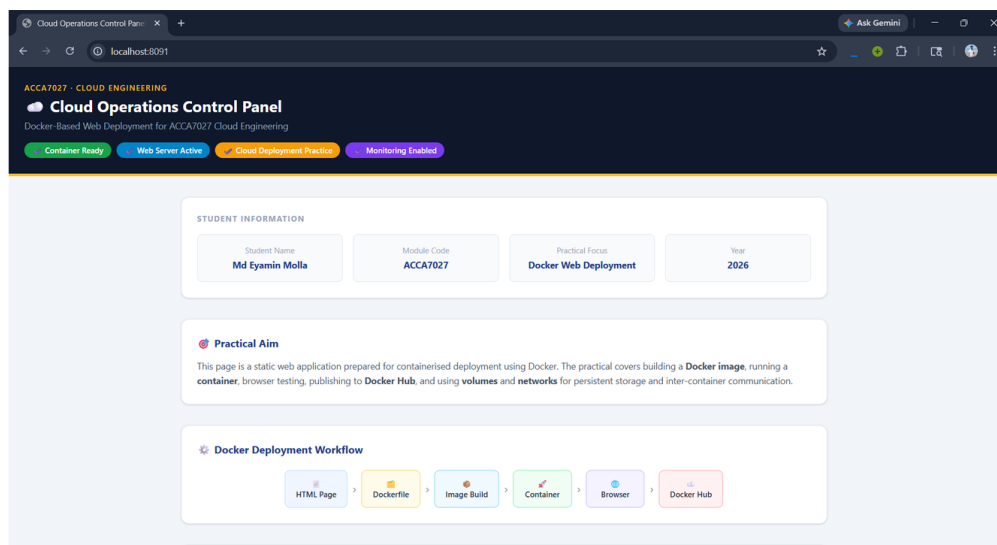
3.4 Running the container and publishing the port

Docker was run in the background with the command: 'docker run -d --name eyamin-apache-web -p 8091:80 eyamin-cloud-ops-apache:v1'. The -p 8091:80 option allowed us to map the

host port 8091 to the container port 80 while the -d option was used to run Apache in the background. This was confirmed by Figure 5(a), while Figure 5(b) shows the page at 'http://localhost:8091' [9].

```
Windows PowerShell
PS E:\Assignment\Eyamin\clinic-cloud-dashboard> docker run -d --name eyamin-apache-web -p 8091:80 eyamin-cloud-ops-apache:v1
43d6f279e9e107b05bd1d52173100f8f5602f80e37af11de73137b45063c9daf
PS E:\Assignment\Eyamin\clinic-cloud-dashboard> docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS
PORTS         NAMES
43d6f279e9e1   eyamin-cloud-ops-apache:v1         "httpd-foreground"     20 seconds ago Up 18 seconds
0.0.0.0:8091->80/tcp, [::]:8091->80/tcp   eyamin-apache-web
PS E:\Assignment\Eyamin\clinic-cloud-dashboard>
```

(a) Container running with host-to-container port mapping



(b) Web application served at http://localhost:8091

Figure 5. Running container with host-to-container port mapping and successful browser output (Source: Author's own practical work, 2026).

```
docker run -d --name eyamin-apache-web -p 8091:80 eyamin-cloud-ops-apache:v1
docker ps
```

Browser test successfully confirmed that the image started, Apache served the requested file, and the port mapping was successful. It's stronger than the previous Live Server preview as it has a documented port 8091 that links to Docker. But `localhost` does not make an endpoint for public cloud deployment.

3.5 Operational verification through logs

Once the browser made a request, the status of the root page returned by `docker logs eyamin-apache-web` was 200, as shown in Figure 6(a). The 'favicon.ico' file request returned 404, as there was no favicon. There was also a server-name warning, but it didn't stop Apache from working. This warning can be eliminated by setting up a small Apache configuration file.

Then, the image was tagged as `mdeyaminmolla/eyamin-cloud-ops-apache:v1`, and uploaded to Docker Hub. As seen in Figure 6(b) and 6(c), both tags had the same image ID and recorded the same layers and returned digest. This indicates that the image has been accepted to the registry.

```

Windows PowerShell
PS E:\Assignment\Eyamin\clinic-cloud-dashboard> docker logs eyamin-apache-web
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 17
2.17.0.2. Set the 'ServerName' directive globally to suppress this message
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 17
2.17.0.2. Set the 'ServerName' directive globally to suppress this message
[Sun May 03 21:07:41.940414 2026] [mpm_event:notice] [pid 1:tid 1] AH00489: Apache/2.4.66 (Unix
) configured -- resuming normal operations
[Sun May 03 21:07:41.940870 2026] [core:notice] [pid 1:tid 1] AH00094: Command line: 'httpd -D
FOREGROUND'
172.17.0.1 - - [03/May/2026:21:09:09 +0000] "GET / HTTP/1.1" 200 11913
172.17.0.1 - - [03/May/2026:21:09:11 +0000] "GET /favicon.ico HTTP/1.1" 404 236
172.17.0.1 - - [03/May/2026:21:10:01 +0000] "-" 408 -
PS E:\Assignment\Eyamin\clinic-cloud-dashboard>

```

(a) Apache log showing a successful HTTP 200 request

```

Windows PowerShell
PS E:\Assignment\Eyamin\clinic-cloud-dashboard> docker images

```

IMAGE	ID	DISK USAGE	CONTENT SIZE	EXTRA
eyamin-cloud-ops-apache:v1	fa0b33ffb3c3	96.7MB	20.8MB	U In Use
mdeyaminmolla/eyamin-cloud-ops-apache:v1	fa0b33ffb3c3	96.7MB	20.8MB	U

```

PS E:\Assignment\Eyamin\clinic-cloud-dashboard>

```

(b) Local image tagged with the Docker Hub namespace

```

Windows PowerShell
PS E:\Assignment\Eyamin\clinic-cloud-dashboard> docker push mdeyaminmolla/eyamin-cloud-ops-apac
he:v1
The push refers to repository [docker.io/mdeyaminmolla/eyamin-cloud-ops-apache]
719e56c8c985: Pushed
4f4fb70ef54: Pushed
7b3c5d639c11: Pushed
e1a4731f5772: Pushed
b989da80e210: Pushed
352f015f534a: Pushed
4ca6f6797e45: Pushed
6a0ac1617861: Mounted from mdimranbiswas/imran-cloud-assignment-nginx
41c15f5ba584: Pushed
v1: digest: sha256:fa0b33ffb3c3497bd1268d2ff208d4de38cb2b7ee129bd6afaa24781c2198072 size: 856
PS E:\Assignment\Eyamin\clinic-cloud-dashboard>

```

(c) Image layers pushed and digest returned by Docker Hub

Figure 6. Apache request logs, Docker Hub image tag and successful registry push (Source: Author's own practical work, 2026).

docker logs eyamin-apache-web

docker tag eyamin-cloud-ops-apache:v1 mdeyaminmolla/eyamin-cloud-ops-apache:v1

docker push mdeyaminmolla/eyamin-cloud-ops-apache:v1

3.6 Docker Hub publication

The public Docker Hub repository appears as shown in Figure 7 after it was published. The same image can be pulled and run by another Docker host, enabling repeatable deployments and collaboration. Nevertheless, secrets or patient information should not be stored in a public

repository, with release tags backed by a description and vulnerability scanning and/or retention rules [7].

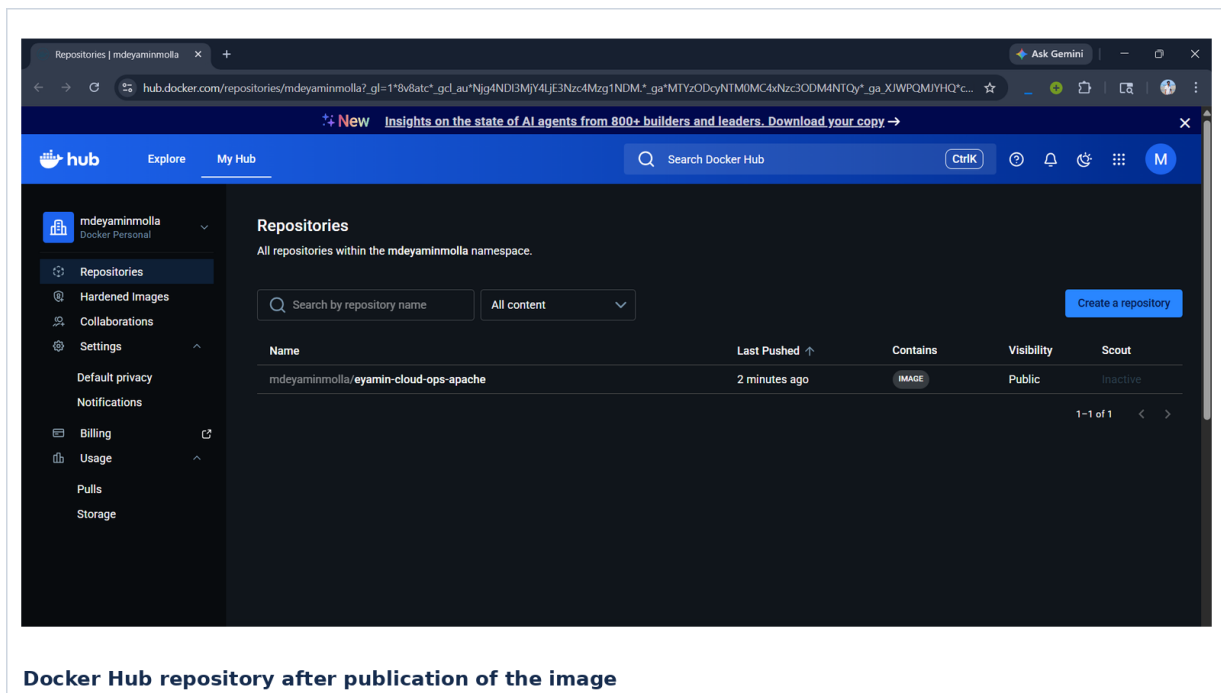


Figure 7. Published Docker Hub repository containing the Apache web image (Source: Author's own practical work, 2026).

3.7 Named volume and persistence

If there is no volume under Apache's document root, initially the image content at this path is hidden by mounting an empty volume. Hence, the volume has been populated using the `docker cp` command. The page would stay in the immutable image; volumes are more appropriate for uploaded files, generated content, or other data that should not be deleted once the container is deleted.

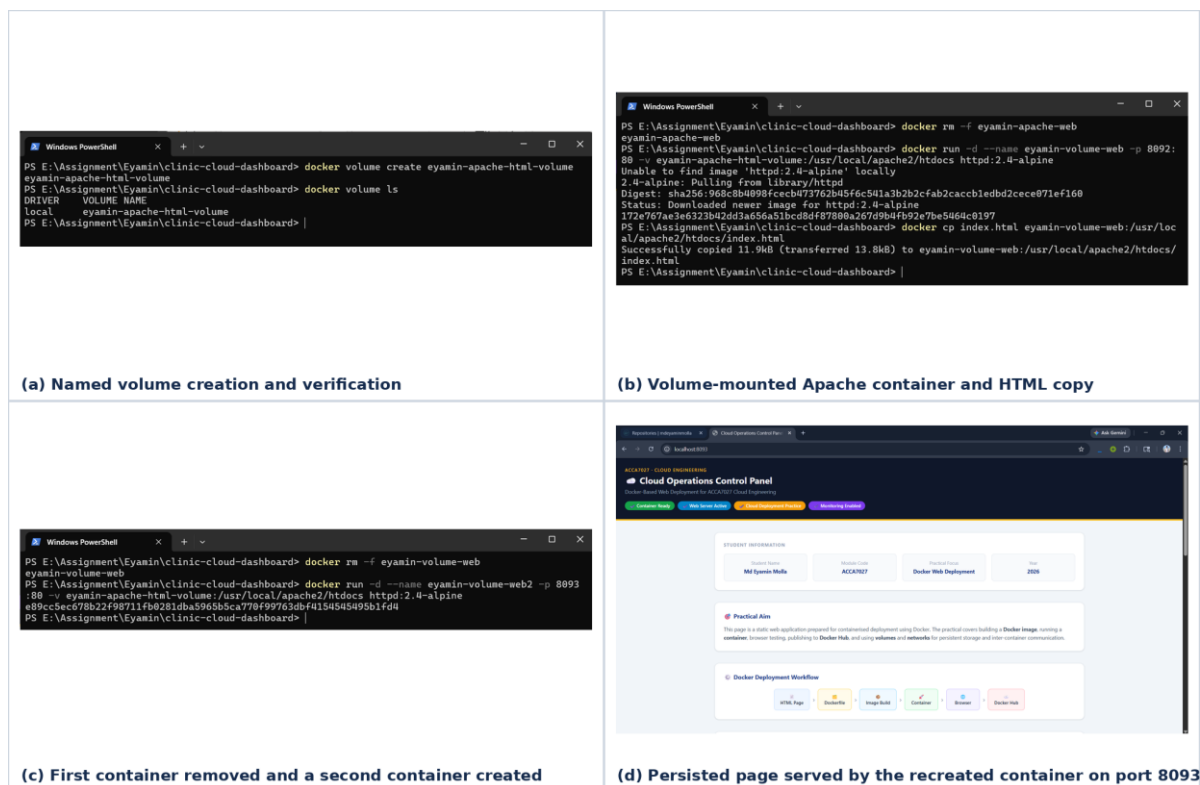


Figure 8. Named-volume creation, mounted content and persistence across container recreation (Source: Author's own practical work, 2026).

```
docker volume create eyamin-apache-html-volume
docker run -d --name eyamin-volume-web -p 8092:80 -v eyamin-apache-html-volume:/usr/local/apache2/htdocs httpd:2.4-alpine
docker cp index.html eyamin-volume-web:/usr/local/apache2/htdocs/index.html
docker rm -f eyamin-volume-web
docker run -d --name eyamin-volume-web2 -p 8093:80 -v eyamin-apache-html-volume:/usr/local/apache2/htdocs httpd:2.4-alpine
```

If the image is not put into a volume and the image is mounted over Apache's document root, the image content initially is hidden at that path. The latter 'docker cp' command filled the volume. The page would be static and would typically stay within the immutable image. The uploads, other files created, or data for which the duration of life is higher than a container is better suited for a volume.

3.8 User-defined network and service discovery

The practical has created a user-defined bridge network called 'eyamin-cloud-network'. An Apache container and a temporary Alpine container were connected in the same network. This HTML page was returned from the Alpine container by issuing the command `wget -qO-`

http://eyamin-network-web. Hence, Figure 9 verifies the creation of the network and the name-based communication between the containers [11]. A name for the container is more secure than an IP address that changes. This is a bridge network that can only be used on one Docker host, though: Multi-host communication would require an overlay network or cloud orchestrator.

```
Windows PowerShell
PS E:\Assignment\Eyamin\clinic-cloud-dashboard> docker network create eyamin-cloud-network
249425b414accfa76d5ebe4b13c26509191fb05c555bcb5b190905a45091194d
PS E:\Assignment\Eyamin\clinic-cloud-dashboard> docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
69cdcc0745c9        bridge              bridge              local
249425b414ac        eyamin-cloud-network bridge              local
43e56c7b8bdb        host                host                local
4de3af798eb4        none                null                local
PS E:\Assignment\Eyamin\clinic-cloud-dashboard>
```

(a) User-defined bridge network creation

```
Windows PowerShell
PS E:\Assignment\Eyamin\clinic-cloud-dashboard> docker run -d --name eyamin-network-web --network eyamin-cloud-network -p 8094:80 eyamin-cloud-ops-apache:v1
f9914be9c8c57204a69489d7d69c72bde77d379d2e373b415c7319cba95e19f
PS E:\Assignment\Eyamin\clinic-cloud-dashboard> docker ps
CONTAINER ID   IMAGE                COMMAND                  CREATED    STATUS    PORTS                               NAMES
f9914be9c8c5   eyamin-cloud-ops-apache:v1   "httpd-foreground"      6 seconds ago    Up 4 seconds    0.0.0.0:8094->80/tcp, [::]:8094->80/tcp    eyamin-network-web
PS E:\Assignment\Eyamin\clinic-cloud-dashboard> docker run -it --rm --network eyamin-cloud-network alpine sh
/ # wget -qO- http://eyamin-network-web
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Cloud Operations Control Panel - ACCA7027</title>
  <script src="https://cdn.tailwindcss.com"></script>
</head>
<body class="bg-slate-100 text-slate-800 min-h-screen">
  <!-- HEADER -->
  <header class="bg-slate-900 text-white px-6 py-6 border-b-4 border-amber-400">
    <p class="text-amber-400 text-xs font-semibold uppercase tracking-widest mb-1">ACCA7027 · Cloud Engineering</p>
    <h1 class="text-2xl font-bold mb-1">Cloud Operations Control Panel</h1>
    <p class="text-slate-400 text-sm mb-4">Docker-Based Web Deployment for ACCA7027 Cloud Engineering</p>
    <div class="flex flex-wrap gap-2 text-xs font-semibold">
      <span class="bg-green-600 text-white px-3 py-1 rounded-full">Container Ready</span>
      <span class="bg-sky-600 text-white px-3 py-1 rounded-full">Web Server Active</span>
      <span class="bg-amber-500 text-white px-3 py-1 rounded-full">Cloud Deployment Practices</span>
      <span class="bg-violet-600 text-white px-3 py-1 rounded-full">Monitoring Enabled</span>
    </div>
  </header>
  <main class="max-w-5xl mx-auto px-4 py-8 space-y-8">
    <!-- STUDENT INFO -->
    <section class="bg-white rounded-xl shadow p-6 border border-slate-200">
      <h2 class="text-xs font-bold uppercase tracking-widest text-slate-400 mb-4">Student Information</h2>
      <div class="grid grid-cols-2 sm:grid-cols-4 gap-4 text-center">
        <div class="bg-slate-50 rounded-lg p-4 border border-slate-200">
          <p class="text-xs text-slate-400 mb-1">Student Name</p>
          <p class="font-bold text-blue-900 text-sm">Md Eyamin Molla</p>
        </div>
        <div class="bg-slate-50 rounded-lg p-4 border border-slate-200">
          <p class="text-xs text-slate-400 mb-1">Module Code</p>
        </div>
      </div>
    </section>
  </main>
</body>
</html>
```

(b) Name-based container communication verified with wget

Figure 9. User-defined bridge network and successful container-to-container communication by service name (Source: Author's own practical work, 2026).

docker network create eyamin-cloud-network

docker run -d --name eyamin-network-web --network eyamin-cloud-network -p 8094:80 eyamin-cloud-ops-apache:v1

```
docker run -it --rm --network eyamin-cloud-network alpine sh
wget -qO- http://eyamin-network-web
```

It's safer to use a container name vs. a variable IP address. It also can be used for multi-container designs that involve linking a web service to a private API or database. This bridge network is only intended for use on a single Docker host, however. If this is a multi-host communication, then an overlay network or a cloud orchestrator would be needed.

3.9 Requirement coverage and critical evaluation

The required areas in Docker are fulfilled in the practical. The images below (Figures 3-6) illustrate the Dockerfile, creation of the image, running the container, output in a browser window, and the logs and registry push. Figure 7 proves it, Figure 8 shows it, and Figure 9 proves it! The evidence relates to commands, not to contents which are displayed inside the HTML page. It is an effective practical teaching exercise but is not set up for production. Main enhancements include adding TLS at a proven gateway, network exposure limitation, PIN and scan the image, restriction of network exposure, and centralizing logs and adding health checks. If the control is unproven, then it should be labeled as planned.

4. Load Balancing, Scalability and Application Gateways

4.1 Application scenario and provider selection

The three cloud concepts are demonstrated with an example Web service in a proposed clinic that is proposed to Microsoft Azure. Patients would have access to the information and make appointments; staff would have access in a separate authenticated area. The HTML page submitted is just a prototype for the presentation. Identity controls and protected data storage would be pertinent to a real service as well, as would be an API. Internet traffic gets to Azure Application Gateway v2 as illustrated in Figure 10. The gateway supports routing of HTTP/HTTPS traffic, TLS termination, and Web Application Firewall (WAF) inspection (optional). It sends requests to healthy replicas of the containers, and Azure Monitor adjusts the number of replicas based on demand. Only if a separate private layer-4 service needs the TCP or UDP distribution is Azure Load Balancer utilized [13]-[16].

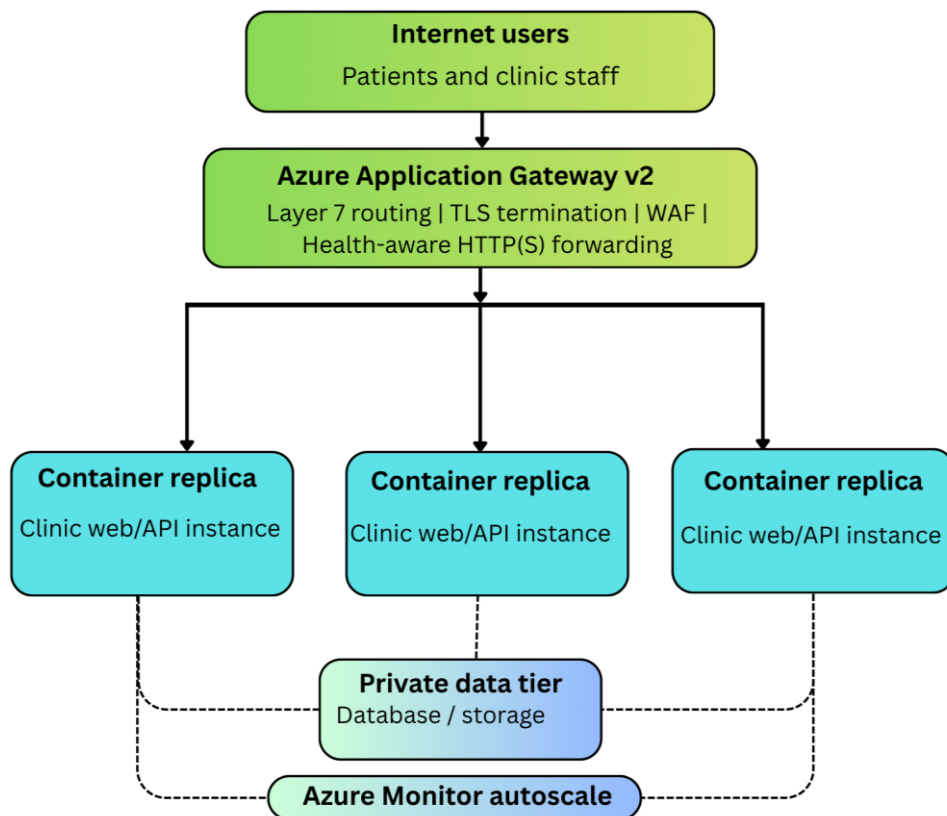


Figure 10. Proposed Azure architecture for a customer-facing clinic web service (Source: Author's illustration based on [13]–[16]).

4.2 Application Gateway

Azure Application Gateway is a web traffic service on Layer-7 that is available regionally. It can provide routing to other backends such as appointments and staff, terminate TLS and redirect HTTP to HTTPS, and apply a Web Application Firewall. Health probes are used to avoid traffic from reaching downstream backends [13], [14]. There is consistent routing and security between replicas from this central entry point. It adds, though configuration and costs. When designing a resilient solution, the following needs to be taken into account: zones, routing, and encryption between the gateway and untrusted backend networks.

4.3 Load balancing

Load balancing distributes requests amongst the healthy instances, thus avoiding concentrating all of the traffic on one instance. Azure Load Balancer operates at layer 4 of the TCP/IP protocol for both UDP and TCP services, and can be used with services that run in the public internet, as well as services that run in the Azure network. Application Gateway already handles loads at Layer 7 for HTTP traffic. Thus, it is not advisable to put both of these products on the same

simple HTTP path unless they both have a definite function. Distribution is limited by having at least one replica that is healthy. Application state must also be stored outside application state replicas for horizontal scaling. The data in the session should then be stored in shared storage or managed data services, and load balancing distributes requests across the healthy replicas.

4.4 Scalability and elasticity

Scalability relates to the ability to expand capacity without undesirable slowdowns in performance. Vertical scaling would increase the resources of an instance, and horizontal scaling would add more instances. This capacity is adapted for fluctuations in demand, thanks to elasticity. Azure Monitor autoscale can add or remove replicas based on metrics, schedules, as well as minimum and maximum limits [16]. But scaling rules need to be validated. The rules can be too sensitive or too slow, which can result in either a higher cost or a delay during peak periods. Furthermore, the databases and other downstream services need to be able to accept the added workload caused by extra replicas.

4.5 Why load balancing and scaling are complementary

Scaling increases the capacity; load balancing works within that capacity. If it's not scaled, a balancer can distribute traffic only to a predetermined pool. If no load balancing is implemented, new web replicas might not get any traffic. Monitoring, autoscaling, and health-aware distribution, then, are one control process. The ideas are still unique. While the use of a request balancer could be required for load balancing to increase availability, queue workers can scale without a request balancer. However, for the proposed clinic web tier, both are required as the users come to the endpoint and the number of replicas changes behind the endpoint.

4.6 Benefits and limitations of combining all three

Combined, the three technologies offer a consistent HTTPS gateway, web-enabled security, healthy traffic distribution, and agility in capacity. This can help to minimize interruption in the event of failure and ensure response times when it is busy. It also provides the ability to have a gradual deployment as traffic can flow both between healthy old container versions and new ones. The design isn't necessarily secure or accessible, however. Costs/access to the system can be raised or eliminated due to poor rules, probes, or gateway settings. Dependencies also include DNS, identity, databases, and the Azure region. It is still important to monitor and conduct recovery tests and adhere to cost limits [25].

Table 2 summarises the separate role, benefit and main caution for each technology.

Technology	Role	Clinic use	Benefit	Main caution
Application Gateway v2	Layer-7 HTTPS ingress, routing, TLS and WAF	Secure public entry point	Consistent routing and web protection	Regional cost and policy complexity
Azure Load Balancer	Layer-4 TCP/UDP distribution	Private non-HTTP service where needed	Health-aware traffic distribution	Can be redundant on a simple HTTP path
Azure Monitor autoscale	Changes replica capacity	Adds or removes service replicas	Supports peaks and lowers idle use	Rules and downstream limits need testing

Table 2. Evaluation of load balancing, scalability and application gateways (Source: Author's evaluation based on [13]–[16]).

5. Driving Forces Behind Cloud Expansion and Serverless Computing

5.1 Economic and operational driving forces

Cost, flexibility, and access to managed capabilities are drivers for cloud growth. With measured billing, the need for buying hardware to meet the demand at peak times can be greatly reduced. But this isn't always cheaper, as data transfer, HA, monitoring, and unused resources can all add to the cost of cloud. Suitable architecture, tagging, budgets, and regular review are essential for good cost control [1], [18]. Cloud platforms offer other advantages like provisioning in no time, quick-to-deploy test beds, elastic capacity, and global infrastructure. Managed identity, databases, messaging, and monitoring can help to streamline routine infrastructure tasks. But there is also organizational readiness, management support, compatibility, and outside pressure that must also exist for adoption to be successful [18]. These are the driving forces that are linked. The governance of rapid provisioning, scalable software for elasticity, and clear targets for backup and recovery for resilience are all required. More of the platform is managed by the provider, while architecture, identity, data, configuration and spending are managed by the customer.

5.2 Serverless functions

With Serverless Function as a Service, developers can deploy code that is executed once an event occurs, and the server is provisioned, patched, and scaled by the provider. Azure Functions can respond to queues and storage events, timers, and HTTP requests [22]. Some of these plans can scale towards zero when not in use, and this may be appropriate in case of short and irregular workloads [24], [23]. An appointment confirmation, document validation, image resize, and removal of expired files are examples of functions that could be sent in the clinic. These tasks will be event-driven and require no worker service to be continuously running. But there are limitations to serverless. Delay may be caused by cold starts, and platforms might limit the execution time, storage, and network [19], [21]. There are numerous small functions that can complicate testing/tracing and/or portability can be lessened by provider-specific

triggers. Issues regarding security, state, and observability are also highlighted from the research [20].

5.3 Containerization versus serverless functions

There are different workloads for different containers and serverless functions. The runtime, libraries, network, and process life cycle can be controlled with containers. They are for plugging in continuous web services and for custom dependencies. Serverless functions decrease infrastructure administration and may bill on an extremely granular level for short functions. So, it will all be about the workload, execution time, portability, and control requirements.

Table 3 compares the two approaches and shows why neither is always cheaper or more portable.

Criterion	Containerised service	Serverless function	Clinic evaluation
Billing	Provisioned capacity and running time	Executions and execution time	Functions can reduce idle cost for occasional tasks; containers suit continuous services.
Scaling	Orchestrator changes replicas	Provider scales event execution	Functions simplify bursts; containers give clearer capacity control.
Runtime control	High control over libraries, process, and network	Provider limits the runtime	Containers suit custom or long-running API needs.
Start-up	Image pull and start delay; warm replicas can reduce it	Cold starts may affect idle workloads	Keep patient-facing paths warm and place background work behind queues.
State	External state is needed for replaceable replicas	Execution is temporary and should externalise state	Both models need durable storage and safe retries.
Portability	OCI image can move between compatible platforms	Triggers and bindings may create provider coupling	Containers offer a stronger package boundary, but migration still requires work.

Table 3. Comparison of containerization and serverless functions (Source: Author's synthesis based on [19]–[24]).

5.4 Recommended hybrid design

Clinics with a hybrid workload are served by a hybrid model [17]. Should run in the public web tier and API tier with at least two warm replicas, health probes, and autoscaling. A queue can be used to delay slower background jobs. Azure Functions can then work out confirmations, reminders, document checks, and scheduled exports without impacting the user-facing request. The process of designing must still be very sensitive to identity, data, and failure. Managed identities should be used for functions, safe retries should be supported for messages, and

handling dead messages. There should be a linked log and trace generated by containers and functions, and cost alerts should be generated for both.

5.5 Practical limitations and production improvements

The assignment concepts are followed by a practical which demonstrates the concepts but is not an actual deployed clinical system. They would need to be able to pin and scan an image, have limited ports, health checks, central logs, TLS and WAF protection, automated deployment, authentication, encrypted data, audit records, privacy controls, backup and tested recovery [9, 12, 26].

6. Conclusion

The practical will show the necessary lifecycle of Docker. The Dockerfile was used to create a reusable image of Apache, which served the page, and the result was verified by the browsers and by the log. It was published to Docker Hub. The volume test is persistence after the containers are replaced, and the custom network test demonstrated that the container could communicate with the other container based on its name. Bigger picture analysis reveals that containers don't offer all of the cloud features. Application Gateway provides a regulated web entrance point, load balancing provides traffic distribution to healthy replicas, and autoscaling adjusts the capacity as needed. They are valuable because of clear roles and because they are able to be run across multiple replicas of an application. Cost isn't the only reason for adopting cloud. Other key features include flexibility, speed of provisioning, elasticity and managed services. While serverless functions are ideal for short event-driven tasks they can't be used in place of containers in all scenarios due to cold starts, limits imposed by the platform, and lower levels of control. Thus, it is suggested to implement the web and API tier in containers, and use functions to implement the asynchronous task.

References

- [1] P. Mell and T. Grance, "The NIST Definition of Cloud Computing". *NIST Special Publication 800-145*. Gaithersburg, MD, USA, 2011. DOI: [10.6028/NIST.SP.800-145](https://doi.org/10.6028/NIST.SP.800-145). Available: <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf> (accessed 19 July 2026).
- [2] K. Scarfone, M. Souppaya and P. Hoffman, "Guide to Security for Full Virtualization Technologies". *NIST Special Publication 800-125*. Gaithersburg, MD, USA, 2011. DOI: [10.6028/NIST.SP.800-125](https://doi.org/10.6028/NIST.SP.800-125). Available: <https://csrc.nist.gov/pubs/sp/800/125/final> (accessed 19 July 2026).

- [3] R. Morabito, J. Kjällman and M. Komu, “Hypervisors vs. Lightweight Virtualization: A Performance Comparison”. *2015 IEEE International Conference on Cloud Engineering*. pp. 386-393, 2015. DOI: [10.1109/IC2E.2015.74](https://doi.org/10.1109/IC2E.2015.74).
- [4] M. Bentaleb, A. S. Z. Belloum, A. Sebaa and A. El-Maouhab, “Containerization Technologies: Taxonomies, Applications and Challenges”. *The Journal of Supercomputing*. vol. 78, no. 1, pp. 1144-1181, 2022. DOI: [10.1007/s11227-021-03914-1](https://doi.org/10.1007/s11227-021-03914-1).
- [5] Docker, “Dockerfile overview”. *Docker Docs*. 2026. Available: <https://docs.docker.com/build/concepts/dockerfile/> (accessed 19 July 2026).
- [6] Docker, “httpd Official Image”. *Docker Hub*. 2026. Available: https://hub.docker.com/_/httpd (accessed 19 July 2026).
- [7] Docker, “Build, tag, and publish an image”. *Docker Docs*. 2026. Available: <https://docs.docker.com/get-started/docker-concepts/building-images/build-tag-and-publish-an-image/> (accessed 19 July 2026).
- [8] Docker, “docker container run”. *Docker CLI Reference*. 2026. Available: <https://docs.docker.com/reference/cli/docker/container/run/> (accessed 19 July 2026).
- [9] Docker, “Port publishing and mapping”. *Docker Docs*. 2026. Available: <https://docs.docker.com/engine/network/port-publishing/> (accessed 19 July 2026).
- [10] Docker, “Volumes”. *Docker Docs*. 2026. Available: <https://docs.docker.com/engine/storage/volumes/> (accessed 19 July 2026).
- [11] Docker, “Bridge network driver”. *Docker Docs*. 2026. Available: <https://docs.docker.com/engine/network/drivers/bridge/> (accessed 19 July 2026).
- [12] R. Chandramouli, “Security Strategies for Microservices-based Application Systems”. *NIST Special Publication 800-204*. Gaithersburg, MD, USA, 2019. DOI: [10.6028/NIST.SP.800-204](https://doi.org/10.6028/NIST.SP.800-204). Available: <https://csrc.nist.gov/pubs/sp/800/204/final> (accessed 19 July 2026).
- [13] Microsoft, “What is Azure Application Gateway?”. *Microsoft Learn*. 2026. Available: <https://learn.microsoft.com/en-us/azure/application-gateway/overview> (accessed 19 July 2026).
- [14] Microsoft, “Azure Application Gateway features”. *Microsoft Learn*. 2026. Available: <https://learn.microsoft.com/en-us/azure/application-gateway/features> (accessed 19 July 2026).
- [15] Microsoft, “What is Azure Load Balancer?”. *Microsoft Learn*. 2026. Available: <https://learn.microsoft.com/en-us/azure/load-balancer/load-balancer-overview> (accessed 19 July 2026).
- [16] Microsoft, “Overview of autoscale in Microsoft Azure”. *Microsoft Learn*. 2026. Available: <https://learn.microsoft.com/en-us/azure/azure-monitor/autoscale/autoscale-overview> (accessed 19 July 2026).
- [17] Microsoft, “Choose an Azure compute service”. *Azure Architecture Center*. 2026. Available: <https://learn.microsoft.com/en-us/azure/architecture/guide/technology-choices/compute-decision-tree> (accessed 19 July 2026).
- [18] A. Al Hadwer, M. Tavana, D. Gillis and D. Rezaia, “A Systematic Review of Organizational Factors Impacting Cloud-based Technology Adoption Using Technology-Organization-Environment Framework”. *Internet of Things*. vol. 15, art. 100407, 2021. DOI: [10.1016/j.iot.2021.100407](https://doi.org/10.1016/j.iot.2021.100407).
- [19] H. Shafiei, A. Khonsari and P. Mousavi, “Serverless Computing: A Survey of Opportunities, Challenges, and Applications”. *ACM Computing Surveys*. vol. 54, no. 11s, art. 239, pp. 1-32, 2022. DOI: [10.1145/3510611](https://doi.org/10.1145/3510611).

- [20] Z. Li, L. Guo, J. Cheng, Q. Chen, B. He and M. Guo, “The Serverless Computing Survey: A Technical Primer for Design Architecture”. *ACM Computing Surveys*. vol. 54, no. 10s, art. 220, pp. 1-34, 2022. DOI: [10.1145/3508360](https://doi.org/10.1145/3508360).
- [21] J. Manner, M. Endreß, T. Heckel and G. Wirtz, “Cold Start Influencing Factors in Function as a Service”. *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion*. pp. 181-188, 2018. DOI: [10.1109/UCC-Companion.2018.00054](https://doi.org/10.1109/UCC-Companion.2018.00054).
- [22] Microsoft, “Azure Functions overview”. *Microsoft Learn*. 2026. Available: <https://learn.microsoft.com/en-us/azure/azure-functions/functions-overview> (accessed 19 July 2026).
- [23] Microsoft, “Event-driven scaling in Azure Functions”. *Microsoft Learn*. 2026. Available: <https://learn.microsoft.com/en-us/azure/azure-functions/event-driven-scaling> (accessed 19 July 2026).
- [24] Microsoft, “Azure Container Apps overview”. *Microsoft Learn*. 2026. Available: <https://learn.microsoft.com/en-us/azure/container-apps/overview> (accessed 19 July 2026).
- [25] Microsoft, “Reliability design principles”. *Azure Well-Architected Framework*. 2026. Available: <https://learn.microsoft.com/en-us/azure/well-architected/reliability/principles> (accessed 19 July 2026).
- [26] M. Souppaya, J. Morello and K. Scarfone, “Application Container Security Guide”. *NIST Special Publication 800-190*. Gaithersburg, MD, USA, 2017. DOI: [10.6028/NIST.SP.800-190](https://doi.org/10.6028/NIST.SP.800-190). Available: <https://csrc.nist.gov/pubs/sp/800/190/final> (accessed 19 July 2026).

Appendix A. Command Sequence Used in the Practical

dir

docker --version

docker ps

docker build -t eyamin-cloud-ops-apache:v1 .

docker images

docker run -d --name eyamin-apache-web -p 8091:80 eyamin-cloud-ops-apache:v1

```
docker ps
```

```
docker logs eyamin-apache-web
```

```
docker login -u mdeyaminmolla
```

```
docker tag eyamin-cloud-ops-apache:v1 mdeyaminmolla/eyamin-cloud-ops-apache:v1
```

```
docker push mdeyaminmolla/eyamin-cloud-ops-apache:v1
```

```
docker volume create eyamin-apache-html-volume
```

```
docker volume ls
```

```
docker rm -f eyamin-apache-web
```

```
docker run -d --name eyamin-volume-web -p 8092:80 -v eyamin-apache-html-
```

```
volume:/usr/local/apache2/htdocs httpd:2.4-alpine
```

```
docker cp index.html eyamin-volume-web:/usr/local/apache2/htdocs/index.html
```

```
docker rm -f eyamin-volume-web
```

```
docker run -d --name eyamin-volume-web2 -p 8093:80 -v eyamin-apache-html-
```

```
volume:/usr/local/apache2/htdocs httpd:2.4-alpine
```

```
docker network create eyamin-cloud-network
```

```
docker network ls
```

```
docker run -d --name eyamin-network-web --network eyamin-cloud-network -p 8094:80 eyamin-cloud-ops-apache:v1
```

```
docker run -it --rm --network eyamin-cloud-network alpine sh
```

```
wget -qO- http://eyamin-network-web
```

```
exit
```

Appendix B. Video demonstration Link

Video Title: Cloud Engineering Video By Md Eyamin Molla

Video Link: <https://youtu.be/mf6T5yUld1s>

Duration: 8 minutes 12sec

Platform: Youtube