

**ACCA7027 – CLOUD ENGINEERING PRINCIPLES
AND PRACTICE**

**CLOUD ENGINEERING PRINCIPLES
AND DOCKER CONTAINERIZATION**

Sayem Akther Talukdar (2502919)

UNIVERSITY OF WALES TRINITY SAINT DAVID - WALES INSTITUTE OF SCIENCE AND
ARTS - SCHOOL OF APPLIED COMPUTING



Table of Contents

Introduction	1
Practical Demonstration videos Link	1
1.Principles of virtualization and containerization	1
1.1 Virtualization of shared hardware	1
1.1.1 Types of Hypervisors.....	2
1.1.2 Real-World Implementation	2
1.1.3 Types of Virtualizations.....	2
1.1.4 Challenges of Virtualization	2
1.1.5 Benefits of Virtualization	2
1.2 Containerization principles and benefits	3
1.2.1 Container Architecture	3
1.2.2 Benefits of Containerization	3
1.2.3 Docker vs Kubernetes	3
1.2.4 Security Issues in Containerization	4
1.3 Virtualization vs. containerization	4
2. Practical Docker work	5
2.1 Building and pushing Docker images	5
2.2 Container volumes for persistent storage	7
2.3 Container networking and Communication between MULTI-containers	8
2.4 Use case: Persistent to the Database	9
2.5 Putting It All Together– Sample Project.....	9
3.Core cloud-computing concepts.....	11
3.1 Load balancing	11
3.2 Scalability	11
3.3 Application gateways.....	12
3.4 Serverless Functions vs Containerization	12
3.5 Real-World Example.....	13
3.6 Why load balancing and scaling are complementary to one another.....	13
3.7 Combining load balancing, scalability and application gateways.....	13
3.8 Summary	14
4. Grounds for the hybrids of cloud adoption or the serverless functions vs containerization...	14
4.1 Economic and operational drivers of cloud computing.....	14

4.2 What are serverless functions (FaaS)?.....	14
4.3 Serverless functions vs containerization	15
5. Critical Evaluation	16
6. Conclusion	16
References	17

Picture of Contents

Figure 1: Docker images created successfully using the docker build command.	5
Figure 2: Docker Hub authentication completed successfully.	6
Figure 3: Docker image tagged with latest version.	6
Figure 4: Docker image pushed to Docker Hub repository.	6
Figure 5: Successful Docker image push verification on the web repository, confirming that the container image was uploaded correctly and is available in the remote registry.....	7
Figure 6: Docker volume created for persistent storage.	7
Figure 7: Custom Docker bridge network created successfully.	8
Figure 8: Multiple containers connected within the same Docker network.	8
Figure 9: Successful communication between containers using ping.	9
Figure 10: Persistent storage verified using Docker volumes.	9
Figure 11: Running Docker container verified using docker ps.	10
Figure 12: Web application successfully accessed through browser on localhost:8080.	10
Figure 13: Docker volume mounted into the container.	10

Cloud Engineering Principles and Docker Containerization

Introduction

Cloud computing has become one of the most important technologies in modern computing because it provides scalable, flexible and cost-effective solutions for organisations and individuals. The rapid growth of cloud platforms has changed how applications are developed, deployed and maintained by enabling services to run efficiently across distributed infrastructures. Technologies such as virtualization, containerization and serverless computing now play a major role in supporting modern cloud environments.

This report explores the principles of virtualization and containerization and explains their importance in cloud computing. It examines how virtualization allows multiple virtual machines to run on shared hardware and how containerization improves application deployment through lightweight and portable containers. The report also demonstrates practical Docker implementations, including building and publishing Docker images, creating Docker volumes, configuring container networking and running multi-container applications.

In addition, the report discusses key cloud-computing concepts such as load balancing, scalability and application gateways, highlighting their role in improving performance, availability and reliability in cloud-based systems. The report further investigates the main driving forces behind cloud adoption, including flexibility, cost efficiency and operational agility, while comparing serverless computing with container-based approaches.

Finally, the report provides a critical evaluation of modern cloud technologies by discussing their advantages, limitations and future trends within cloud computing environments.

Practical Demonstration videos Link

https://www.youtube.com/watch?v=qG0Y6MxD_AE

1.Principles of virtualization and containerization

1.1 Virtualization of shared hardware

One of the core technologies in cloud computing is virtualization, which allows a single physical machine to host multiple VMs — each containing its own OS and applications. This is made possible by a hypervisor, through which you can use hardware resources (CPU, memory and storage) between virtual environments [1] [2].

In traditional computing environments, each physical server had its own OS and app so machines sat half-tapped & under-utilized. As a result, resources were under-utilized, and infrastructure costs grew. Virtualization addresses this limitation with the ability to run many operating systems

concurrently within one host system, keeping each VM isolated from other VMs for stability and security [2].

1.1.1 Types of Hypervisors

Hypervisors are classically divided into 2 types. Bare-metal or Type 1 hypervisors: This type of hypervisor runs directly on physical hardware. It is used mainly for enterprise and cloud infrastructures due to its performance & efficiency. Virtualization examples : VMware ESXi | Microsoft Hyper-V on the other hand, Type 2 (hosted) hypervisors run above an additional operating system, mainly for the purpose of testing and development [2].

1.1.2 Real-World Implementation

For example, a practical instance of virtualization can be found in Amazon Web Services, with its Elastic Compute Cloud (EC2) service. Without the need for physical infrastructure, EC2 allows customers to get virtual servers on-demand showcasing how cloud needs scale and flexibility thanks to virtualization. Likewise, modern data centers implement virtualization to combine several workloads onto fewer physical machines -- cutting hardware costs and improving energy use intensity along the way.

1.1.3 Types of Virtualizations

There are also alternatives to your kind of virtualization. Full virtualization provides stronger isolation guarantees but tends to incur performance penalties by allowing unmodified operating systems on independent nodes. On the other hand for para-virtualization, performance is improved by allowing guests to communicate directly with the hypervisor, but this requires operating system modifications [3].

1.1.4 Challenges of Virtualization

But along with its benefits, virtualization poses a few challenges. The biggest trade-off is performance overhead because the hypervisor layer can impose a small throughput penalty relative to native execution, but typically this limitation is minimal (around 1-2%). Another challenge is resource contention when multiple virtual machines compete for very limited physical resources such as CPU and memory. This can impact system performance, especially in high-demand environments—if not managed well.

In general, virtualization is still an important part of the cloud — helping with resource usage, scalability and infrastructure management at a lower cost. Which is still in use of cloud platforms and makes it significant part around modern IT systems.

1.1.5 Benefits of Virtualization

There are different important benefits of virtualization for improving efficiency, flexibility and system reliability. These advantages are summarized below:

- Efficient use of resources: commercial cloud providers can improve maximum availability of the hardware by sharing CPUs, memory and storage across virtual machines [1].
- Simplified management & rapid provisioning: VMs are software-defined, provisioned in a matter of minutes from templates and automation [1]

- Isolation and disaster recovery: A failure in one virtual machine will not affect the other VMs & as they are more portable, vms can be moved or replicated for disaster recovery [1]
- Cost efficiency: fewer physical servers means less hardware, power and maintenance expenditure on this piece of IT infrastructure [1].

1.2 Containerization principles and benefits

Containerization (or containerization) is a new way to deploy applications: an app and all its dependencies bundled together as a lightweight, portable unit called a container. Compared to virtual machines, containers do not need a complete operating system but share the kernel of the host system while running isolated user environments [3]. Hence, containers are way more efficient in terms of resources and start time.

1.2.1 Container Architecture

Containerization has an underlying architecture that is one of the key aspects. All containers on a host system share the same operating system kernel, which is called kernel sharing relied by containers. This is lighter than the traditional virtual machines, where every instance has to have a separate OS. Containers also use namespaces to provide isolation. Namespaces, as the name suggests, guarantees that each container has an isolated view of system resources including processes, network interfaces and file systems therefore preventing interference between containers. Control groups (groups), which allow a set of processes to manage the resources consumed by containers (CPU, memory, and other). These features will enable efficient, secure and more scalable container execution [3].

1.2.2 Benefits of Containerization

The above benefits give an overview of why containerization has emerged as the go-to approach for developing modern-day native applications.

- Fast turnaround and scalability: As containers can start almost instantaneously, they can scale very quickly at high demand [3].
- Portability — a container wraps all the application dependencies needed to run it reliably for different environments on any cloud provider [3].
- Support for microservices: containers are well-suited for microservices architectures because they enable applications to be composed of smaller, independent services where each service runs in its own container [3].
- Resource efficiency: containers require less resource consumption compared to VMs and hence provide high density, low cost [3].
- Concretely with a few compromises, containers isolate applications but not from each other 9 therefore industry uses VM's as they have security isolation [3].

1.2.3 Docker vs Kubernetes

Similarly, clarifying between container technologies such as Docker and orchestration tools like Kubernetes is also essential. You use Docker to create, package and run containers for your application and Kubernetes will manage the orchestration of many such applications across a cluster of machines. Docker is a set of tools for managing the individual container lifecycle, whereas Kubernetes offers advanced features like automated scaling, load balancing and fault

tolerance. This scalability aspect makes things like large deployments at production scale more straightforward for Kubernetes.

1.2.4 Security Issues in Containerization

However, with the benefits of containerization also come numerous security challenges. A prominent scenario is container escape, in which a rogue process is launched by the container and breaks out of it to gain control over the underlying host system and thus other containers. Another problem is image vulnerabilities which is simply because the containers are based on publicly available base images that might contain deprecated software or security flaws. If not scanned on time and patched, then it can be a potential leak point of Cybercriminal [3].

In summary, containerization is here to stay and forms a vital part of cloud computing. Nonetheless, organizations need to mitigate security risks and apply the right instruments and best practices for safe deployment in cloud environments.

1.3 Virtualization vs. containerization

Both virtualization and containerization abstract physical hardware resources, yet they exist at different layers of the system stack and have differing scopes. Some VM (Virtual Machines) provide virtualization of the complete hardware stack including the OS, enabling multiple independent operating systems on a single physical host. This solution offers well isolated environments and is particularly suitable when you want to run applications with different operating systems or need strong security boundaries. [2] Unlike containerization, which virtualizes the user-space environment but shares the host operating system kernel. That overwhelmingly cuts down the overhead since containers do not have to start up a full instance of the operating system for every instance. Consequently, containers are lightweight, start up much quicker than virtual machines and use less system resources [3]. On the performance side, containers provide better utilization and scalability. Since they can be deployed in rapid time and easily replicated, it is perfect for cloud applications, stateless services and micro-services architecture. In such environments, applications are decomposed into smaller services that can be developed, deployed and scaled separately, which fits naturally into the container model [3]. But this efficiency has a few caveats. Because containers share the kernel of the host operating system, they provide weaker isolation than virtual machines. This can create a security risk, especially if there is a container escape which allows an attacker to access the host system or other containers [3]. On the other hand, virtual machines provide better isolation as each VM runs on its own full OS making them more appropriate for sensitive workloads or applications where strict security controls are required. Speaking of use cases, while virtual machines are usually recommended for legacy systems, applications that need complete control over the operating system (such as database servers), etc., containers are great for microservices-based architecture. Containers, on the other hand, are specifically designed for modern apps that need to be deployed quickly and in a repeatable way; fast iterative scaling; transportability across environments, and high utilization of physical resources. In conclusion, while both technologies are essential for cloud computing, containerization is often preferred for modern application development due to its speed and efficiency, whereas virtualization continues to be important in use cases where stronger isolation and flexibility are needed.

2. Practical Docker work

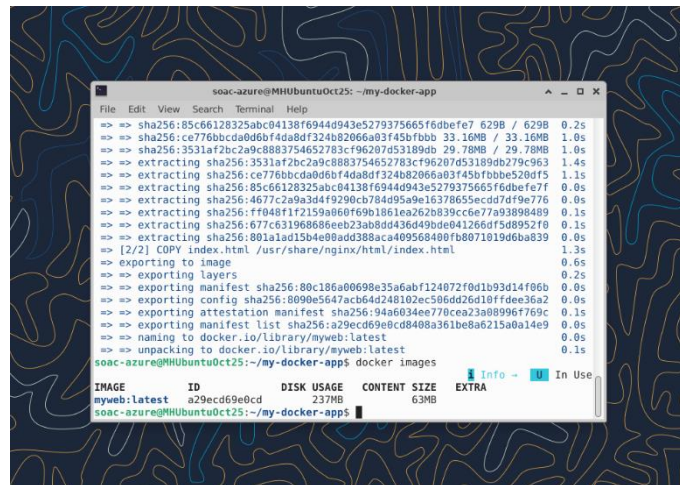
2.1 Building and pushing Docker images

A Docker image is a read-only, layered template that you use to create containers. Images are usually defined in a Docker file — a file with instructions about the base image, what files to copy and what commands to run. To build an image, run:

```
#Build an image based upon the Docker file in the current directory
```

```
$ docker build -t username/appname:1.0
```

Docker build commands will read the Docker file, download Base image and execute all the commands one by [4].



```
soac-azure@MHUbuntu0ct25: ~/my-docker-app
$ docker build -t myweb:latest .
[+] Building 0.2s
[+] sha256:85c66128325abc04138f6944d943e5279375665f6dbefe7 629B / 629B 0.2s
[+] sha256:ce776bbcdad06bf4da8df324b82066a03f45bfbbb 33.16MB / 33.16MB 1.0s
[+] sha256:3531af2bc2a9c8883754652783cf96207d53189db 29.78MB / 29.78MB 1.0s
[+] extracting sha256:3531af2bc2a9c8883754652783cf96207d53189db279c963 1.4s
[+] extracting sha256:ce776bbcdad06bf4da8df324b82066a03f45bfbbb520df5 1.1s
[+] extracting sha256:85c66128325abc04138f6944d943e5279375665f6dbefe7f 0.0s
[+] extracting sha256:4677c2a9a3d4f9290cb784d95a9e16378655ecdd7df9e776 0.0s
[+] extracting sha256:ff048f1f2159a860f69b1861ea262b839cc6e77a93898489 0.1s
[+] extracting sha256:677c63196888eeeb23a08d0436049bde041266d75d8952f0 0.1s
[+] extracting sha256:801ad15b4e08dd388aca409568409f08071019d6ba839 0.0s
[2/2] COPY index.html /usr/share/nginx/html/index.html 1.3s
[+] exporting to image 0.6s
[+] exporting layers 0.2s
[+] exporting manifest sha256:80c186a80698e35a6abf124072f0d1b93d14f06b 0.0s
[+] exporting config sha256:8090e5647ac64d248102ec506dd26d10ffdee36a2 0.0s
[+] exporting attestation manifest sha256:94a6034ee770cea23a088996f769c 0.1s
[+] exporting manifest list sha256:a29ecd69e0cd8408a361be8a6215a0a14e9 0.0s
[+] naming to docker.io/library/myweb:latest 0.0s
[+] unpacking to docker.io/library/myweb:latest 0.1s
soac-azure@MHUbuntu0ct25:~/my-docker-app$ docker images

```

IMAGE	ID	DISK USAGE	CONTENT SIZE	EXTRA	In Use
myweb:latest	a29ecd69e0cd	237MB	63MB		

Figure 1: Docker images created successfully using the docker build command.

First, login using docker login before pushing to Docker Hub Then tag and push the image:

```
soac-azure@MHUbuntu0ct25:~/my-docker-app$ docker login

USING WEB-BASED LOGIN

Info → To sign in with credentials on the command line, use 'docker login -u <username>'

Your one-time device confirmation code is: BDTc-WMMW
Press ENTER to open your browser or submit your device code here: https://login.docker.com/activate

Waiting for authentication in the browser...

WARNING! Your credentials are stored unencrypted in '/home/soac-azure/.docker/config.json'.
Configure a credential helper to remove this warning. See
https://docs.docker.com/go/credential-store/

Login Succeeded
soac-azure@MHUbuntu0ct25:~/my-docker-app$
```

Figure 2: Docker Hub authentication completed successfully.

Tag the image with my Docker Hub repository

\$ docker tag username/appname:1.0 username/appname:latest

```
soac-azure@MHUbuntu0ct25:~/my-docker-app$ docker tag myweb sayem78446/myweb
soac-azure@MHUbuntu0ct25:~/my-docker-app$ docker images
```

IMAGE	ID	DISK USAGE	CONTENT SIZE	EXTRA
myweb:latest	a29ecd69e0cd	237MB	63MB	U
sayem78446/myweb:latest	a29ecd69e0cd	237MB	63MB	U

```
soac-azure@MHUbuntu0ct25:~/my-docker-app$
```

Figure 3: Docker image tagged with latest version.

Push the image to Docker Hub

\$ docker push username/appname:1.0

```
soac-azure@MHUbuntu0ct25:~/my-docker-app$ docker push sayem78446/myweb
Using default tag: latest
The push refers to repository [docker.io/sayem78446/myweb]
36010b6066b1: Pushed
ff048f1f2159: Pushed
93d34145ed2f: Pushed
85c66128325a: Pushed
3531af2bc2a9: Pushed
ce776bbcdad: Pushed
4677c2a9a3d4: Pushed
801a1ad15b4e: Pushed
677c63196868: Pushed
latest: digest: sha256:a29ecd69e0cd8408a361be8a6215a0a14e9a72cd414e770fde7f92fb034d245a size: 856
soac-azure@MHUbuntu0ct25:~/my-docker-app$
```

Figure 4: Docker image pushed to Docker Hub repository.

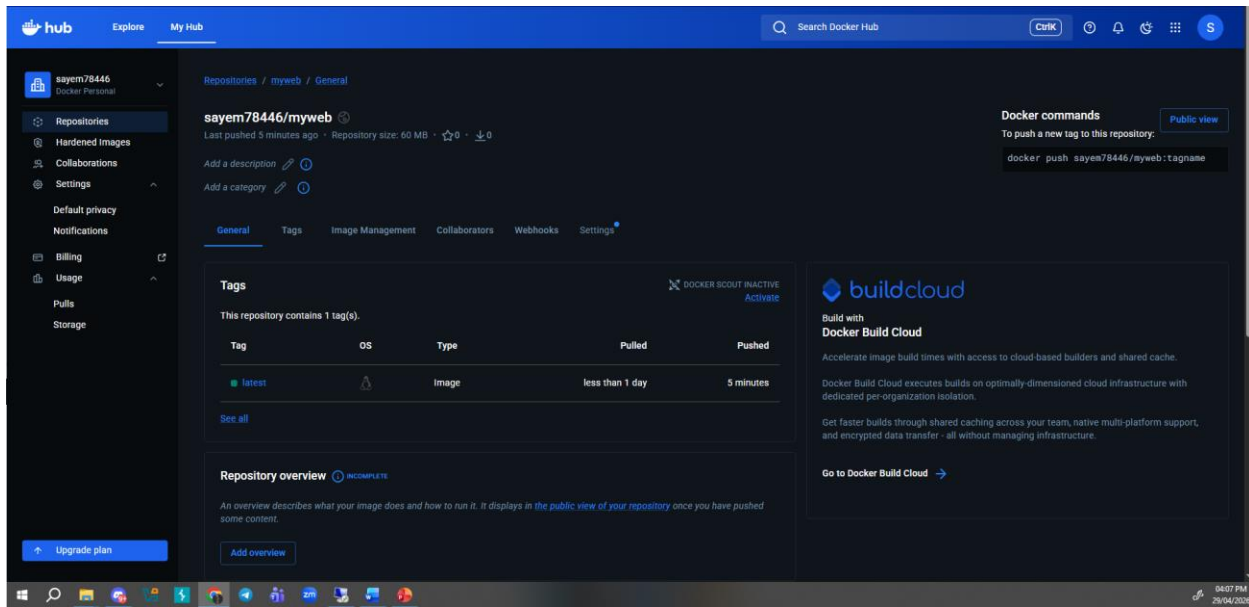


Figure 5: Successful Docker image push verification on the web repository, confirming that the container image was uploaded correctly and is available in the remote registry.

```
$ docker push username/appname:latest
```

A tagging gives human-readable name and where you want it to be [4]. Pushing pushes the image to a registry (Docker Hub AWS ECR, Azure CR) so others can pull it.

2.2 Container volumes for persistent storage

Docker containers are stateless: any changes made to files disappear when the container is deleted. Volumes offer persistent storage independent of the life cycle of containers. Creating a volume is straightforward:

```
$ docker volume create sayemvol
```

```
$ docker volume ls # list volumes
```

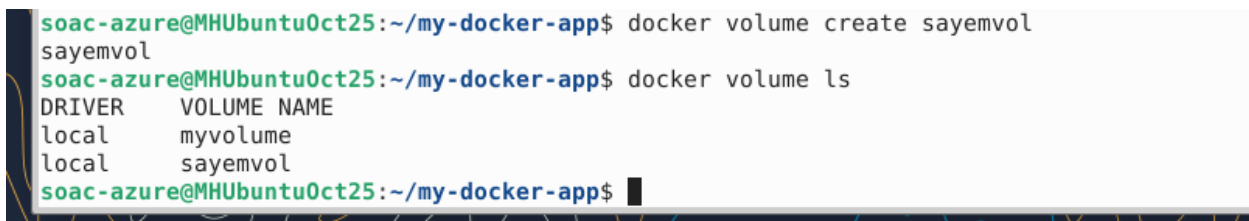


Figure 6: Docker volume created for persistent storage.

You then reference this volume when you run up a container:

This example specifies that any files written to /var/www/html in the container are persisted on the sayemvol volume. The files are persistent even if you remove and recreate the container. Volumes can also be mounted into multiple containers for cases like logging or distributing the data to a list of downstream processors [5].

Volume management commands: These are the commands that include docker volume ls (list), docker volume rm (remove) and docker volume prune (remove unused volumes) [5].

2.3 Container networking and Communication between MULTI-containers

Docker has multiple network drivers; bridge networks are the default. Create isolated network from containers, has two or more connected together:

\$ docker network create mynetworkwork

```
soac-azure@MHUbuntu0ct25:~/my-docker-app$ docker network create sayemnetwork
c9e4d3cea149e1001f2a2da71fb837ccb5a384005b5b129ea2a1f4d0a97f76b5
soac-azure@MHUbuntu0ct25:~/my-docker-app$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
54d1c97c3bee        bridge              bridge              local
a130d51700d9        host                host                local
0e79821b68ea        mynetwork           bridge              local
7adellacd4d3        none                null                local
c9e4d3cea149        sayemnetwork        bridge              local
soac-azure@MHUbuntu0ct25:~/my-docker-app$
```

Figure 7: Custom Docker bridge network created successfully.

Make two containers to run in same network

\$ docker run -d --name web --network mynetwork nginx

\$ docker run -it --name client --network mynetwork alpine sh

```
Run 'docker run --help' for more information
soac-azure@MHUbuntu0ct25:~/my-docker-app$ docker run -dit --name containerX --network mynetwork nginx
873cef93ab38a609d32ab6325bb4dd00266616e39c7b96065ad9d2b3358e2424
soac-azure@MHUbuntu0ct25:~/my-docker-app$ docker run -dit --name containerY --network mynetwork nginx
282b7ca95797ea21598e15f0b8ed6f0ed11a25eae5d0ab3509f14ec410eda9e5
soac-azure@MHUbuntu0ct25:~/my-docker-app$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS                               NAMES
282b7ca95797   nginx    "/docker-entrypoint..." 21 seconds ago Up 20 seconds 80/tcp                             containerY
873cef93ab38   nginx    "/docker-entrypoint..." 34 seconds ago Up 33 seconds 80/tcp                             containerX
7e6dcf201ed9   nginx    "/docker-entrypoint..." About a minute ago Up About a minute 80/tcp                             container1
28881c8bacf8   nginx    "/docker-entrypoint..." 2 minutes ago   Up 2 minutes   80/tcp                             container2
bae38b459e69   nginx    "/docker-entrypoint..." 6 minutes ago   Up 6 minutes   0.0.0.0:8081->80/tcp, [::]:8081->80/tcp unruffled_haibt
491603b841ac   myweb    "/docker-entrypoint..." 52 minutes ago   Up 52 minutes   0.0.0.0:8080->80/tcp, [::]:8080->80/tcp vibrant_wozniak
soac-azure@MHUbuntu0ct25:~/my-docker-app$
```

Figure 8: Multiple containers connected within the same Docker network.

Container tip Use ping web or ping to test connectivity from inside of the alpine container. For example, Docker gives each container its own IP address on the bridge network and runs a simple DNS service that maps container names to IPs [6]. By using names rather than hard-coded addresses to communicate, you enable services to run in containers. Do you have to remove networks (docker network rm mynetwork) or docker network prune Danish unused networks.

```
soac-azure@MHUbuntu0ct25:~/my-docker-app$ docker run -dit --name test1 --network sayemnetwork busybox sh
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
481282afbc43: Pull complete
aecba0016ef2: Download complete
Digest: sha256:1487d0af5f52b4ba31c7e465126ee2123fe3f2305d638e7827681e7cf6c83d5e
Status: Downloaded newer image for busybox:latest
e7a57d40377560fb20f4ff190c89c948e89f70563ef8e0b7fbff83192f7fbbf9
soac-azure@MHUbuntu0ct25:~/my-docker-app$ docker run -dit --name test2 --network sayemnetwork busybox sh
df6fd1d124fefe63dd04036a0afe07b2d40eb7762b671baf1ca8f74da58448dd
soac-azure@MHUbuntu0ct25:~/my-docker-app$ docker exec -it test1 sh
/ # ping test2
PING test2 (172.19.0.5): 56 data bytes
64 bytes from 172.19.0.5: seq=0 ttl=64 time=0.135 ms
64 bytes from 172.19.0.5: seq=1 ttl=64 time=0.116 ms
64 bytes from 172.19.0.5: seq=2 ttl=64 time=0.201 ms
64 bytes from 172.19.0.5: seq=3 ttl=64 time=0.087 ms
64 bytes from 172.19.0.5: seq=4 ttl=64 time=0.126 ms
64 bytes from 172.19.0.5: seq=5 ttl=64 time=0.088 ms
^C
--- test2 ping statistics ---
6 packets transmitted, 6 packets received, 0% packet loss
round-trip min/avg/max = 0.087/0.125/0.201 ms
/ #
```

Figure 9: Successful communication between containers using ping.

2.4 Use case: Persistent to the Database

One of the most common use cases with Docker volumes in real world is persistence volumes for databases. Take for instance running a database container like MySQL or PostgreSQL, one should avoid data loss between starting and stopping the running container or removing it entirely. Because we mounted a Docker volume to the database container, all data is saved outside of the container environment. So, the db. would be there even if you restarted the container or destroyed and recreated it. This strategy is a common practice in actual production environment, which is concerned with data quality and reliability. Docker volumes provide an important component of today's extensive (and cloud-backed) systems, supporting backup, migration and scaling applications.



Figure 10: Persistent storage verified using Docker volumes.

2.5 Putting It All Together– Sample Project

For an end-to-end life cycle—building, running, persisting data and publishing—let us take a web server that serves custom HTML page. The following steps are a reflection of the instructions provided in docker. Create a video that shows how to perform the following “docx” exercises and you should include:

1. This Docker file has the following requirements: write it with a smaller base image (preferably nginx:alpine) Copy your index. html into /usr/share/nginx/html.

2. Create the docker image and give it alias: `docker build -t username/webpage:1.0` [4].
3. Run the container with port 80 exposed `docker run -d -p 8080:80 --name webpage username/webpage:1.0`

```
soac-azure@MHUBuntu0ct25:~/my-docker-app$ docker images
INFO In Use
IMAGE ID DISK USAGE CONTENT SIZE EXTRA
myweb:latest a29ecd69e0cd 237MB 63MB
soac-azure@MHUBuntu0ct25:~/my-docker-app$ docker run -d -p 8080:80 myweb
491603b841ac8199cbf39b6a9dac157b5b7c760e9f21f9cea3b6d897fc290f35
soac-azure@MHUBuntu0ct25:~/my-docker-app$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
491603b841ac myweb "/docker-entrypoint..." About a minute ago Up About a minute 0.0.0.0:8080->80/tcp, [::]:8080->80/tcp vibrant_wozniak
soac-azure@MHUBuntu0ct25:~/my-docker-app$
```

Figure 11: Running Docker container verified using `docker ps`.

NOTE: Check that your my is loading in a browser (`http://localhost:8080`).



Figure 12: Web application successfully accessed through browser on `localhost:8080`.

4. Make it a volume and mount this to `/usr/share/nginx/html` so the updates is persistent: `docker volume create html-data`, Then: `docker run -d -p 8080:80 -v html-data:/usr/share/nginx/html username/webpage:1.0`[5].

```
soac-azure@MHUBuntu0ct25:~/my-docker-app$ docker run -d -p 8081:80 -v sayemvol:/usr/share/nginx/html nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
Digest: sha256:6e23479198b998e5e25921dff8455837c7636a67111a04a635cf1bb363d199dc
Status: Downloaded newer image for nginx:latest
bae38b459e69514d6492bb5c39792472ddb795851b73186ffdb88c1290152274
soac-azure@MHUBuntu0ct25:~/my-docker-app$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
bae38b459e69 nginx "/docker-entrypoint..." 9 seconds ago Up 9 seconds 0.0.0.0:8081->80/tcp, [::]:8081->80/tcp unruffled_haibt
491603b841ac myweb "/docker-entrypoint..." 45 minutes ago Up 45 minutes 0.0.0.0:8080->80/tcp, [::]:8080->80/tcp vibrant_wozniak
soac-azure@MHUBuntu0ct25:~/my-docker-app$
```

Figure 13: Docker volume mounted into the container.

5. Tag and push the image to Docker Hub `docker tag username/webpage:1.0 username/webpage:latest && docker push username/webpage:1.0` [4]
6. Build a network (`docker network create demo-net`) and launch other containers on the same net (container like DB). Ping the other from one container to demonstrate connectivity [6].

This example shows how to build your image, run it locally, persist data, publish to a registry and orchestrate multi-container communication.

3.Core cloud-computing concepts

3.1 Load balancing

Load balancer: is the practice of distributing multiple client requests among various application instances to maximize performance and reliability. Software-based load balancer run on the same infrastructure as applications, adjusting automatically to demand changes in cloud environments [7]. Over conventional hardware appliances, cloud load balancers provide a variety of advantages:

- Availability and performance of applications: they route traffic to healthy instances and support zero-downtime maintenance [7].
- Elasticity: They combine with auto-scaling mechanisms to automatically add or remove instances depending on workload [7].
- Cost and Scalability: Since software-based solutions do not require specialized hardware, they offer a lower cost solution delivered as an on-demand service on virtually any network [7].

Load balancers provided by a cloud provider. Example: Azure Load Balancer works on Layer 4 (TCP/UDP), Application Gateway is used for Layer 7 (HTTP/HTTPS) routing. AWS Elastic Load Balancer (Classic, Application and Network load balancer) for AWS like regional load balancing per application and network level or Google Cloud Platform provides dual load balancing which are regional and global. Besides these implementations, we would see different load balancing algorithms articulate to distribute the traffic efficiently. The most common technique is round robin which distributes requests in sequence over the available servers. So, this is a simple method that works when there is some kind of uniformity in the capacity of all your servers. Another option is least connection, which makes requests to go to the server with the fewest number of active connections. In real-world environments, workloads are rarely uniform and this method serves to smooth out traffic per node on the fly.

3.2 Scalability

Scalability: system ability to handle increasing workloads by ready resources accordingly. Using cloud computing allows organizations to incubate up and down the assets, as it saves costs by not investing in physical infrastructure [8]. Scaling can be broken into two significantly different types: Vertical scaling and Horizontal scaling. Vertical scaling — Increasing the capacity of a single server i.e. adding more CPU, memory or storage to the current server. It is true that this method is basic, but it will always be limited to the maximum little of your devices. However, horizontal scaling is adding more servers or instances to scale out the load. This method is more versatile and can be used in cloud settings, helping with fault tolerance and high availability [9]. Load balancing and scalability both work hand in hand to keep the system reliable. Scaling adds more resources, but load balancing is a way to distribute the incoming traffic across resources. This minimizes system congestion and enhances reliability while improving user experience [9].

3.3 Application gateways

An application gateway is a Layer 7 load balancer that makes routing decisions in the context of application data like URLs, headers and cookies. This enables traffic to be routed to various backend services according to the type of request. Application gateways offer many advanced features, including SSL / TLS termination to cut the processing load on backend servers-and integration with web application firewalls to bolster security. They even can route based on path, which means given a different part of your application (like /api or/images) should connect to back-end services. These functionalities are highly valuable in modern cloud architecture where applications are composed of multiple microservices and require intelligent management of workflow and traffic [10].

3.4 Serverless Functions vs Containerization

Two popular modern approaches for deploying and managing applications in cloud environments are serverless computing and containerization. Although both are aimed at improving scale and efficiency of blockchains, they differ significantly in execution model, control, cost and use cases. Serverless Computing (FaaS or Function-as-a-Service) is an event-driven model of execution. The applications are divided into several simple, event-driven functions that execute tasks like responding to an HTTP request, performing some action for every update on a database or every file uploaded. These functions execute only when requested, and the cloud provider automatically manages all aspects of the underlying infrastructure (scale, resource allocation etc.) [12]. On the other hand, applications packaged in containers are executed as long-running processes with developers managing deployment, scaling and orchestration. Serverless computing costs are based on a pay-per-execution pricing method that allows users to run their code only when necessary. This makes it extremely affordable for unpredictable or bursty workloads. Stanley Peters, Computer Science Department, Stanford University 21 Containers need assigned resources, though; as a result, were allocated CPU and memory (the "active processes") even when not busy these can cost more for low-use applications [13]. Scalability is another important distinction. We can call Serverless platforms that automatically scales up or down according to demand. Gives you awesome elasticity without hands on support. Unlike container-based systems where scaling needs to be done explicitly using orchestration tools or ecosystems (exchange fast deployment and horizontal scaling) [14]. Control and flexibility are also drastically different. In serverless environments, developers have less control over the runtime environment since they are limited by the configurations that the cloud provider supports. This means containers give total control of the application environment, including OS, libraries and dependencies while virtual machines are more suited to complex or custom applications [13]. On the performance side of things again, serverless computing might face issues with cold start latency where there is a delay when the function is invoked after being idle for some time. This can impact real-time applications. Compared to traditional VMs, for example, containers usually have a more stable performance as they are pre-initialized [12] and run constantly.

Practical Use Cases: This is where you see the real distinctions between the two scenarios. Serverless computing is primarily well-suited for lightweight and event-driven applications ranging from chatbots to REST APIs, processing pipelines and notification systems. An example is a chatbot application that will call serverless functions to process your user input in a dynamic way without keeping persistent servers up and running. But in case you are focusing on long-running applications like web servers, enterprise systems and microservices-based architectures, containerization is a better fit. One example could be if a web application written using containers

can run in long-running task, maintain transient connections and scale back-end services. Both approaches have advantages and disadvantages. With serverless, your application is tightly coupled with a specific cloud provider service which may create vendor lock-in. Also, the performance in latency-sensitive applications is affected by cold start issues. Containers, on the other hand, give more control to how your application runs but come with extra management burdens in terms of configuration, orchestration and security upkeep. Conclusion Overall – serverless computing and containerization are both critical technologies in cloud computing but have different workload fit. Serverless is great for event-driven and very flexible applications with variable demand, while containers will work well for complex and long-running applications that require better control and consistency.

3.5 Real-World Example

Netflix is a real-world example using these cloud concepts, whereby they use load balancing + horizontal scaling to simultaneously serve millions of users streaming content worldwide. For instance, Amazon trains their large cloud infrastructure and load balancing technologies to handle high traffic volumes in peak periods like promotional sales events. It is also used to route requests that the user makes across different services. This illustrates how the combination of load balancing, scalability and application gateways creates high-availability, enhanced performance and an ideal user experience in large-scale cloud scenarios.

3.6 Why load balancing and scaling are complementary to one another

Load balancing controls the routing of client requests, while scaling determines the timing and number of available resources. Together, they provide high availability, resilience and cost effectiveness [9]. When you have a large, distributed application system with data level, where a lot of incoming requests may result in one instance getting too high traffic and becoming the entire application's bottleneck -- having a load balancer will distribute that traffic into several instances so none gets overloaded. Having an autoscaler as well ensures there are enough available instances during spikes. Such pairing increases usability (reducing latency, reducing errors) and optimises costs by running the least number of resources [9].

3.7 Combining load balancing, scalability and application gateways

Let us assume you have a web-based retail application that is hosted in the cloud. You have a Layer-7 application gateway in front, performing SSL termination and routing (like /catalog, /checkout, /static) to the respective pool of microservices. A load balancer then distributes requests among multiple instances of each microservice (running in containers or VMs) behind the gateway. Autoscaling checks for CPU utilization and request latency and appropriately scales the number of instances up during a flash sale, and back down when it is over. By these technologies combined, you get:

- High availability: If one instance is unhealthy, traffic will automatically be directed to a healthy instance via the load balancer and autoscaling helps maintain redundancy.
- Throughput and user experience: request routing can minimize response time by routing each request to the best matched backend [7].

- Cost efficiency: singletons utilize pricing reduction through removal of idle instances; software-based load balancer and application gateways eliminate expensive hardware [7].
- Security and stewardship ELB: Centralized SSL management; can also work with web application firewalls [10].

3.8 Summary

In summary, load balancing, scalability and application gateways are the foundation of cloud computing design. Together, they form a powerful and efficient infrastructure that can run dynamic workloads, maintain system reliability and provide high-quality services to the users.

4. Grounds for the hybrids of cloud adoption or the serverless functions vs containerization

4.1 Economic and operational drivers of cloud computing

The main reason that organizations are turning to the cloud in the first place is flexibility and savings. Cloud platforms make it possible for the business to scale resources dynamically with demand, avoid over-provisioning while making remote access available to teams across the globe [11] [8]. This is because companies avoid large up-front capital expenditure on hardware—and are only billed by resource consumption on a pay-as-you make it a job basis [8]. Additional drivers include:

- Faster time to market- because backing resources can be provisioned in seconds [8]
- Enhanced collaboration and convenience: you can access data and tools from any device, enabling remote working [11].
- Cloud providers spend stupid amounts of money on security features, backup and disaster recovery [11].

4.2 What are serverless functions (FaaS)?

Also known as functions as a service, serverless computing allows developers to run code without having to manage servers. You build applications with functions that respond to triggers (an HTTP request, a database update, or an upload of files etc.). The underlying infrastructure is automatically provisioned and scaled by the cloud provider which only charges for execution time [12]. Advantages are flexibility, cost-effective and more production as developer must do only code instead of managing server [12]. Some of the popular serverless platforms are AWS Lambda, Azure Functions and Google cloud Functions [13].

4.3 Serverless functions vs containerization

Aspect	Serverless functions	Containers
Execution model	Event-driven functions run only when triggered and are stateless. The provider automatically scales the number of function instances [13].	Long-running processes: containers encapsulate applications and dependencies and run continuously; scaling requires orchestration tools like Kubernetes [13].
Billing model	Pay-per-execution; you pay for the time your code runs and the resources used [13].	Pay for allocated resources (CPU/memory) regardless of utilization [13].
Scalability	Automatic and granular; functions scale to zero when idle and up instantly for bursts [14].	Must be explicitly scaled; containers can be created or destroyed quickly but need orchestration[3].
Control & configuration	Limited control over the runtime; environment and language choice may be restricted. Useful for stateless workloads [14].	Full control over the OS environment, libraries and configuration. Suitable for long-running services or custom requirements [13].
Startup latency	Possible “cold starts” (delay when a function runs for the first time), which can impact latency [12].	Containers start faster than VMs and have predictable startup times [3].
Use cases	Event-driven tasks, infrequent or unpredictable workloads, real-time data processing, serverless APIs [13].	Microservices, long-running applications, workloads requiring custom libraries or persistent connections (e.g., web servers, databases) [12].

Driving forces are reflected in the comparison:

- **Cost:** serverless functions are cost-efficient for spiky or intermittent workloads because they eliminate idle-time charges[13]. Containers may be more cost-effective for long-running services that would incur frequent function invocations[13].
- **Flexibility and agility:** serverless abstracts away infrastructure management, enabling rapid development and experimentation[12]. Containers provide flexibility by encapsulating dependencies and running uniformly across environments[3].
- **Scalability:** both models support scaling, but serverless scales automatically to zero, while containers need orchestration. For highly unpredictable workloads, serverless may offer better elasticity[14].
- **Control and security:** containers offer more control and isolation at the cost of additional management overhead; serverless is simpler but may raise security and vendor-lock-in concerns[12].

5. Critical Evaluation

Containerization and serverless computing are both commonly used approaches when developing for the cloud today, however neither is always a perfect fit nor have well known limitations depending on the task to be performed. Containerization provides flexibility, portability and optimal resource utilization; but it does not always have the right solution in every scenario. In addition, the container needs extra management overhead (including instrumentation, orchestrating and security configuration). manages containers across a cluster but adds complexity and requires expertise. Additionally, containers have less isolation than virtual machines because they all share the host operating system kernel, which can lead to security issues in more sensitive environments. Like so, serverless computing offers automatic scaling, less operational overhead and cost efficiency for event-driven applications. Nevertheless, it comes with numerous shortcomings. A huge problem is vendor lock-in, where your apps are tightly coupled with a cloud provider's services and it becomes difficult to migrate. Also, there is cold start latency: functions will take longer to run if not invoked for a time (this may turn your function into a lemon on Realtime applications). On the other hand, serverless platforms typically place limits on execution duration and resources which may discourage long running or computing intensive jobs. Over the next few years, artificial intelligence (AI) and machine learning are expected to work closely with cloud computing. Cloud providers are delivering built-in AI services providing automated scaling, intelligent resource allocation and predictive management of systems. The idea behind this integration is to improve efficiency and lessen the human-to-cloud contact. In addition, hybrid architectures that use containers, serverless functions and AI-driven automation are emerging as organizations try to take advantage of the best of breed from each approach. In conclusion, finding the right mix is key to picking cloud technologies. Organizations need to dissect use cases + workload requirements and performance and security considerations instead of falling for the lure of a single architecture that promises it all.

6. Conclusion

In this ACCA7027 coursework, you test your research on the cloud-engineering principles and demonstrate your Docker practical skills. This report provided an overview of the fundamental concepts of virtualization and containerization; it also illustrated how containers work to give you an understanding of how containerization fits into cloud computing. This has demonstrated steps to build, tag and push Docker images, mount data using volumes and connect multiple containers over networks- all available in docker. docx exercise list. The report also reviewed the cloud principles of load balancing, scalability & application gateways and how we combine these principles to provide reliable, performant and cost-efficient services. Lastly, it has explored the economics of cloud adoption— cost savings, flexibility & agility—and has compared serverless functions with containers to clear up when to prefer each alternative.

References

- [1] IBM, “Virtualization,” *Ibm.com*, Apr. 09, 2025.
<https://www.ibm.com/think/topics/virtualization>
- [2] GeeksforGeeks, “Virtualization in Cloud Computing and Types,” *GeeksforGeeks*, Jan. 21, 2018. <https://www.geeksforgeeks.org/cloud-computing/virtualization-cloud-computing-types/>
- [3] amit sheps, “Containerization vs. Virtualization: Key Differences and Use Cases,” *Aqua*, Nov. 01, 2023. <https://www.aquasec.com/cloud-native-academy/docker-container/containerization-vs-virtualization/>
- [4] “Build, tag, and publish an image,” *Docker Documentation*, 2024.
<https://docs.docker.com/get-started/docker-concepts/building-images/build-tag-and-publish-an-image/>
- [5] “Persisting container data,” *Docker Documentation*, 2024. <https://docs.docker.com/get-started/docker-concepts/running-containers/persisting-container-data/> (accessed Apr. 19, 2025).
- [6] Luca Cambi, “Baeldung,” *Baeldung on Ops*, Jan. 28, 2023.
<https://www.baeldung.com/ops/docker-communicating-with-containers-on-same-machine> (accessed May 08, 2026).
- [7] “Benefits of Cloud Load Balancing,” *F5, Inc.*, 2025. <https://www.f5.com/glossary/cloud-load-balancing> (accessed Mar. 28, 2025).
- [8] Google, “Advantages Of Cloud Computing,” *Google Cloud*, 2025.
<https://cloud.google.com/learn/advantages-of-cloud-computing>
- [9] P. L. Team, “Auto Scaling vs Load Balancing: Key Differences,” *PubNub*, May 12, 2025.
<https://www.pubnub.com/blog/auto-scaling-vs-load-balancing/> (accessed Nov. 16, 2025).
- [10] greg-lindsay, “What Is Azure Application Gateway,” *learn.microsoft.com*, Sep. 27, 2023.
<https://learn.microsoft.com/en-us/azure/application-gateway/overview>
- [11] Lumenalta, “10 benefits of cloud computing | Cloud computing benefits for flexibility, security, and cost savings | Lumenalta,” *Lumenalta*, 2024, doi:
<https://doi.org/1019928/1920x728/e87520461b>.
- [12] Datadog, “Serverless vs. Containers: Which is Right for You? | Datadog,” *Datadog*, 2024.
<https://www.datadoghq.com/knowledge-center/serverless-architecture/serverless-vs-containers/>
- [13] V. Krishnakumar, “Serverless Computing vs Containerization: A Comprehensive Comparison for Modern Cloud Applications,” *CloudOptimo*, Feb. 03, 2025.
<https://www.cloudoptimo.com/blog/serverless-computing-vs-containerization-a-comprehensive-comparison-for-modern-cloud-applications/>

[14] Cloudflare, “Serverless computing vs. containers | How to choose | Cloudflare,” *Cloudflare*. Available: <https://www.cloudflare.com/learning/serverless/serverless-vs-containers/>