

University of Wales

Trinity Saint David

An Assignment on

Cloud Engineering Principles and Practice

Module code: ACCA7027

Submitted To:

Renuga Jayakumar

Lecturer

Department of Cybersecurity and Digital Forensics

University of Wales Trinity Saint David

Submitted By:

Md Obaydul Hoque

ID: 2435442

Programme:

M.Sc. in Cybersecurity and Digital Forensics

University of Wales Trinity Saint David

Date of Submission

18th December 2025

Table of Content

1. Abstract	
2. Introduction	03
3. Cloud Technology and Containerization	
2.1. Principles of Virtualization	04
2.2. Principles and Benefits of Containerization	04
2.3. Practical Implementation of Docker	06
2.3.1. Building a Docker Image	06
2.3.2. Publishing Images to DockerHub	10
2.3.3. Container Volumes and Deploy	12
2.3.4. Container Networking	13
2.4. Video Demonstration	14
4. Cloud Computing Concepts	
3.1. Overview of Concepts	15
3.2. Load Balancing	15
3.3. Scalability	15
3.4. Benefits of Combined Technologies	16
5. Driving Forces and Serverless Computing	
4.1. Driving Forces of Cloud Expansion	16
4.2. Serverless Functions (FaaS)	16
4.3. Comparison: Serverless vs. Containerization	18
6. Conclusions	20

Abstract

Scalable, efficient and flexible solutions are the key of current cloud engineering; it is image/containerization technologies like Docker which has been widely adopted. Unlike the classical virtual machine model, as far as Docker container is concerned it is a lightweight container which encompasses an application and all of its dependencies in order to be efficiently portable and utilize resources effectively. This paper studies the ground rules for cloud engineering (focusing on Docker in the sense of containerization). This article analyses where docker can help make software development, deployment and management easier by looking at its position in cloud service models (IaaS, PaaS, SaaS), virtualization technologies as well as orchestration and continuous integration/deployment (CI/CD). In this paper, we demonstrate the possible if Point Cloud could be extended by modules and help to understand what Docker can do related with Cloud Computing, why Docker is better than traditional VMs, how about it uses like architecture inside of Kubernetes and other cloud technology-wise. Finally, this paper also introduces real world use case of Docker with respect to cloud engineering, security related concerns and insights into future of containerization as next generation cloud platform.

Introduction

Cloud computing was technology in the past but it is now digital transforming driving force in contemporary IT and global market. Moving beyond the on-premises IT infrastructure, cloud computing offers firms a level of speed, agility and cost — advantages that have never existed in history. Big Data Curation and Application Deployment Cloud this as micro-services including according to the modern companies [1] are inter-related disciplines, data mining based on Big-Data for slicing and dicing data [2], (Kwon et al., 2016).

And when I reflect on the technological progress we have made in the past decade, oh man has this industry come a long way. IT systems were once such a model that led the way to the fat client/ server-side computing, where hardware management and resource provisioning almost posed as fun bottlenecks. All that is kind of blurry in the age of Virtualization. This was a breakthrough as it enabled more than one Virtual Machine (VM) to operate on an individual physical server. But the spread didn't end there. Nowadays the industry is headed towards Containerization and Serverless Architectures, which has revolutionized App Development by making it lighter, scalable, and faster [3].

The subject of this paper is the critical study of these new cloud engineering technologies. The report first introduces the concepts of Virtualization and Containerization, then delivers a step-by-step tutorial on Docker technology [4]. It then introduces some basic cloud design patterns, Scalability and Load Balancing: two of the most important factors that the system architect should take into account [5]. Finally, the report concludes with Serverless Computing motivations for cloud adoption and comparison between container and serverless function [6].

2.0: Cloud Technology and Containerization

We now discuss two building blocks of cloud: Virtualization and Containers. These advances have transformed the cloud, providing greater efficiency, scalability and cost savings in application deployment. Empowering organizations to respond to changing needs, leverage resources and get applications faster in the hands of end users is one of my favorite outcomes by enabling a more dynamic infrastructure. Virtualization and containerization have played a huge role in the development of contemporary cloud architecture, and they remain essential to shaping where technology is headed, including transformative digital experiences. [1]

2.1 The Principles of Virtualization

Virtualization is one of main technologies converting physical hardware into multiple virtual environments, which can improve efficiency, flexibility and scalability for computing [2]. It is capable of running more than one VM on a single physical server, allowing them to run different operating systems (OS). At the core of virtualization is breaking up the physical server into isolated, individual pieces that can be used more efficiently, resulting in better performance overall.

The Role of the Hypervisor:

Hypervisor Virtualization is based on the hypervisor, an essential component of software that takes care of and allocates resources to a virtual machine. The hypervisor enables multiple VMs to exist on a single physical host, and gives them all the appearance of independent boxes. Hypervisors are of two main types:

Type 1 Hypervisor (Bare Metal): This hypervisor runs on the host's physical hardware to server directly. An example is VMware ESXi and Microsoft Hyper-V. Type 1 hypervisors are normally considered for use in the enterprise data center as a result of their high levels of performance and direct physical hardware access that is required to support implementing large scale infrastructure.

Type 2 Hypervisor (Hosted): These hypervisors are service applications that operate on the host OS and are suitable for small environments. This includes Oracle VirtualBox and VMware Workstation. Type 2 hypervisors, on the other hand, are often limited to development, testing and departmental scale-out deployments because they are easy to use and work well on a typical desktop/laptop.

Limitations of Virtualization:

Virtualization was a game-changer in computing, but it has limitations, especially as they relate to modern application architectures such as microservices:

Resource Overhead: Each virtual machine needs its own guest operating system, which consumes substantial CPU, memory and storage resources. This overhead can be costly for lightweight applications or in environments with tight resource constraints, which will relatively reduce scalability and performance of large amount VMs.

Slow Boot: With multiple operating systems, sometimes booting up or shutting down your virtual machine can take long. This can result in reduced scalability for applications running on devices like cloudlets where fast VM launch and booting is essential to keep the system responsive.

However with all its limitations, virtualization is still a fundamental technology in cloud computing that allows for more efficient use, isolation and management of workloads. But, with the explosion of modern application architectures, new technologies such as containers are growing in speed partly because they solve some of those inefficiencies.

2.2 The Principles and Benefits of Containerization

Containerization is an effective approach for remedying the issues of conventional virtualization. It is very important to note that, whereas virtualization works at the hardware level, in containerization it is the operating system (OS) level where magic happens [3]. Containers enable applications and their dependencies to run the same in any environment, from a developer's laptop to a production system.

Concept:

Containers are lightweight, isolated environments that run on a shared OS kernel. Each container contains all of the code, runtime, static libraries and other files that an application needs to work, but does not have a full OS attached. This lean design achieves greater resource efficiency than VMs (Virtual Machines) and shorter startup times.

Key Benefits of Containerization:

Portability: Containers adhere to the "write once, run anywhere" concept. Containers package the application and its dependencies together, so they can run consistently in disparate environments. What this means is that an application in a container will operate the same way whether it's running on your developer's laptop, staging, or in production," she said. "This doesn't guarantee no bugs; rather bugs won't occur due to environment differences.

- **Efficiency:** Virtual machines need a separate OS for every application, whereas the host OS kernel can be shared with containers. As a result, overhead decreases and less resources are consumed by containers and the server density increases. Containers tend to be more lightweight and can run at a scale for greater efficiency in resource use.
- **Isolation:** Containers can provide you with much stronger isolation between applications. By running in their own container, if one fails or has issues, the others do not. This isolation leads to rock solid stability and ensures that containerized environments are very reliable.
- **Quick Boot:** Containers can boot up in just milliseconds, as opposed to the number of seconds required by VMs. This speed of containers makes it an excellent fit for dynamic and scalable environments where fast provisioning, rapid scaling, drift automation, and responsiveness are mandatory.

Conclusion:

Containerization has radically transformed the methods of building, deploying, and managing applications with dramatic performance efficiency gains over traditional virtualization. Its low overhead, which accelerates boot times, and solid isolation features make it a go-to solution for emerging cloud-computing architectures such as microservices. With more and more organizations implementing container-based workflows, containerization is now an even lighter-weight way to quickly develop applications in the cloud in a consistent manner.

2.3 Practical Implementation with Docker

Docker is a popular containerization platform that simplifies the creation, deployment, and management of containers [4]. In this section, we will demonstrate how to deploy a static web server using Docker, building on the theoretical concepts discussed above.

2.3.1 Containerization & Image Build:

A Docker image is a read-only template used to create containers. A Dockerfile defines the instructions to build a custom image.

Traditionally, we used virtualization. Suppose you have a physical server. Using a Hypervisor, we would create multiple Virtual Machines or VMs on it. The problem is, each VM requires a separate Operating System or Guest OS. This consumes a massive amount of RAM and CPU.

However, in Docker or Containerization, we don't use a 'Guest OS'. Here, we have the Main OS on the hardware, and sitting on top of that is the Docker Engine. This engine shares the host's resources to run your applications. This makes containers incredibly fast and lightweight.

Building a Docker Image:

- Create a folder named “docker-demo/ ohk-web-server”.
- Inside it, create a file named “index.html”.
- Now, create a file named “Dockerfile” (no extension).

Create a Dockerfile:

Dockerfile is just a script to build Docker image. The Dockerfile will then declare the base image and copy in index. html file to the container and configure web server.

Dockerfile Explanation:

```
FROM nginx:latest                # Use the latest Nginx base image

COPY ./index.html /usr/share/nginx/html/  # Copy our code to the container

EXPOSE 80                        # Expose port 80 for traffic
```

This Dockerfile declares that the Nginx web server will be used, imports an HTML file into the container and opens up port 80 for HTTP traffic.

Well, it is time for us to witness the containerization process. We want to take a basic website and 'containerize' it. First, I'm writing the Dockerfile.

- FROM: We are specifying that we want our base image to be Ubuntu.
- RUN: We are going to install the Apache web server here.
- COPY: We are copying our local index. html file into the container.
- CMD: This is to automatically start the server when this container starts.

Build the image in the terminal: To build the image, we use the following command:

```
docker build -t ohk-web-server .
```

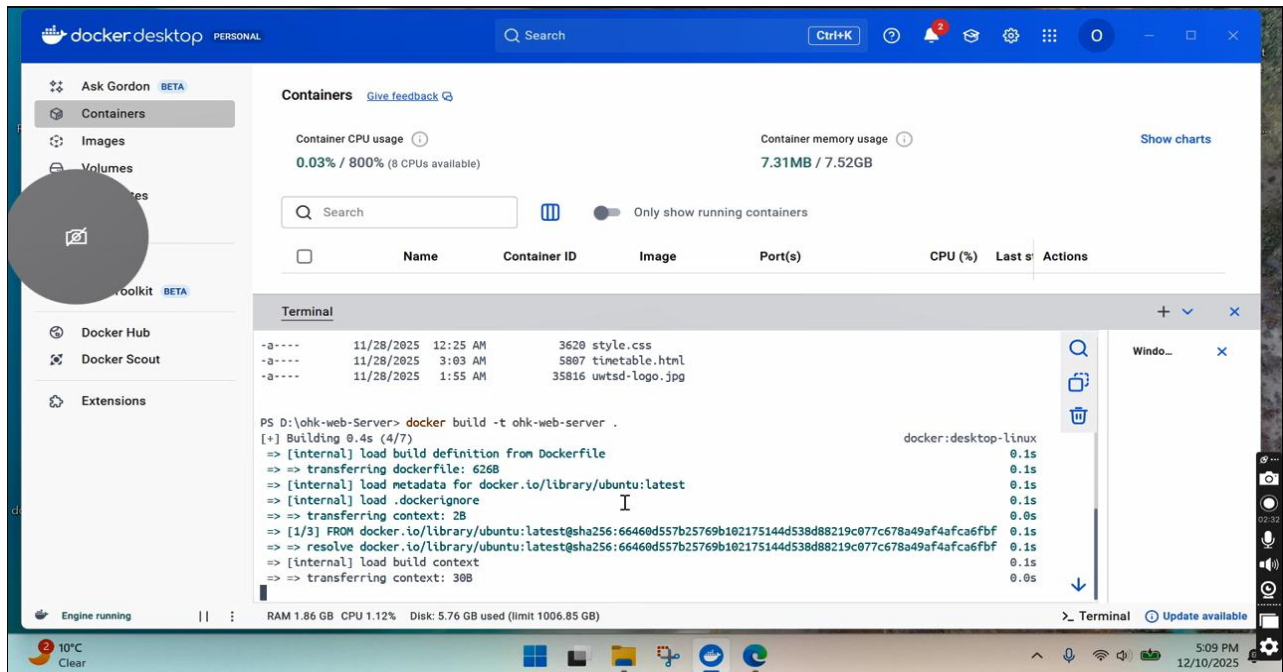


Fig: Build the image

Run the container:

```
docker run -d -p 8080:80 ohk-web-server
```

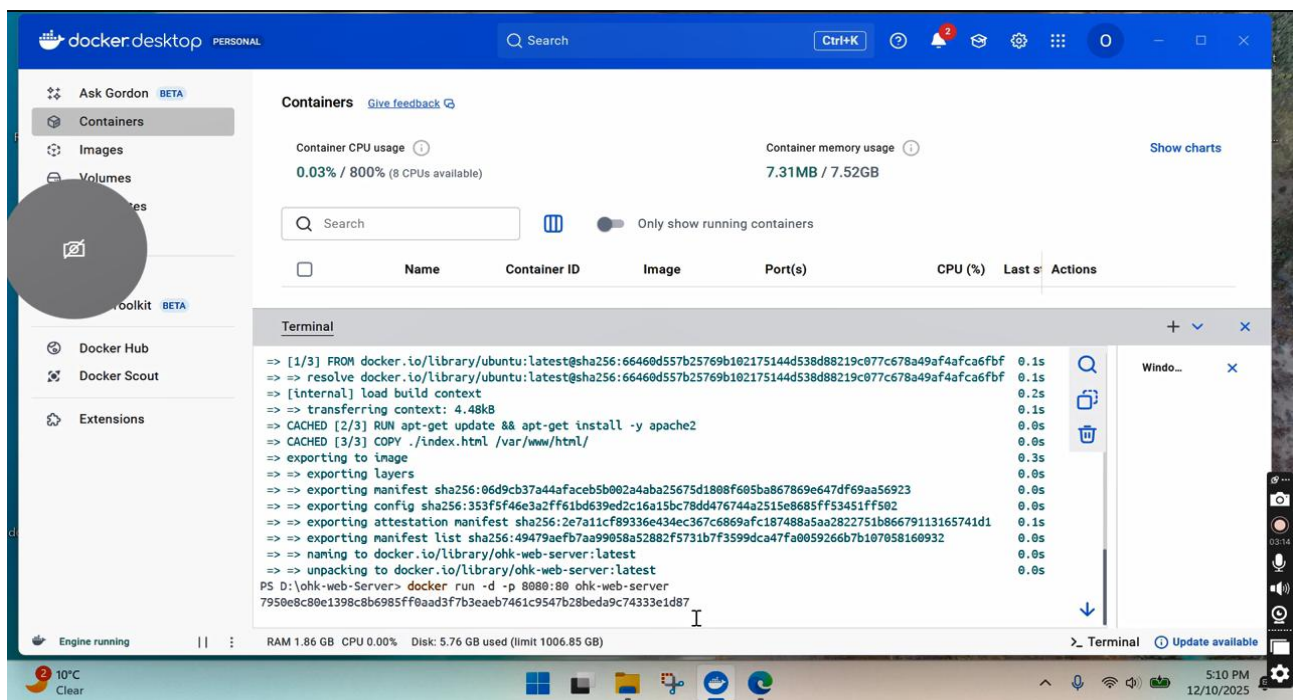


Fig: Run the container

As soon as you have this file, build an image with `docker build`. Now, I'll start it with `docker run`. Now, if I open my browser and navigate to `https://localhost:8080`, our website is live! I even don't have Apache running on my computer but it works inside the Docker container.

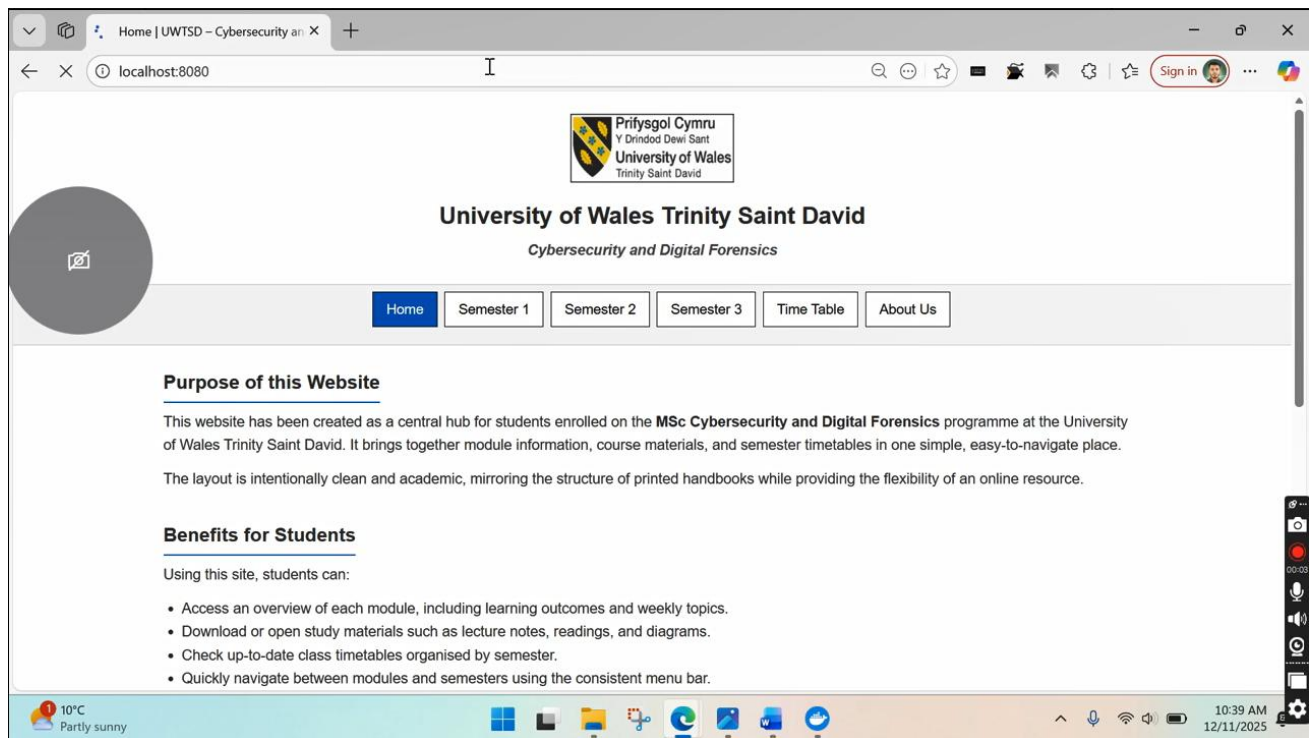


Fig: Website is live

Publishing the Image:

Tag the image:

```
docker tag my-web-server obayduldoker/my-web-server:v1
```

I built the image, but I need to deploy it on a server to share it with others. By pushing the image to Docker Hub, it becomes accessible for deployment on any cloud platform or server.

Push the image:

```
docker push obayduldoker/ohk-web-server:v1
```

First, I'm logging in via the terminal. Before pushing an image we must tag it with username. Now, simply run the `docker push` command, and it will upload to the cloud.

Deploy:

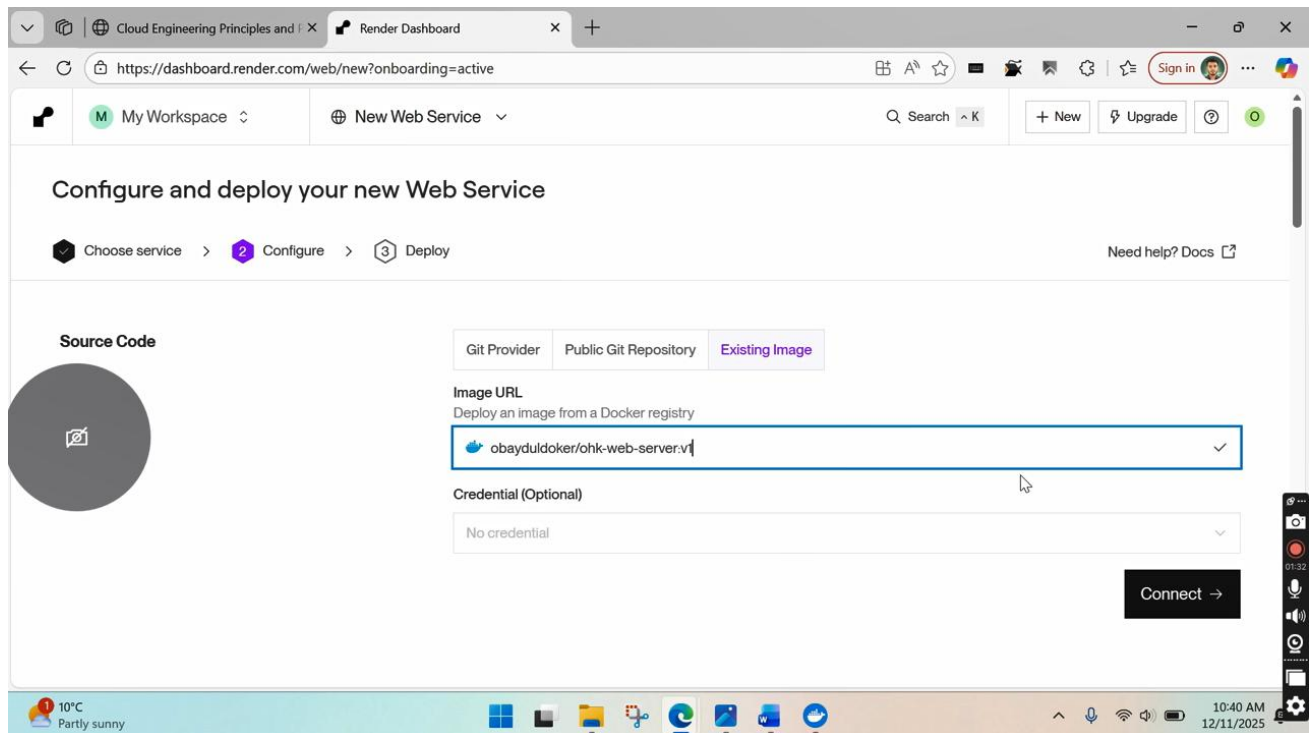


Fig: Deploy

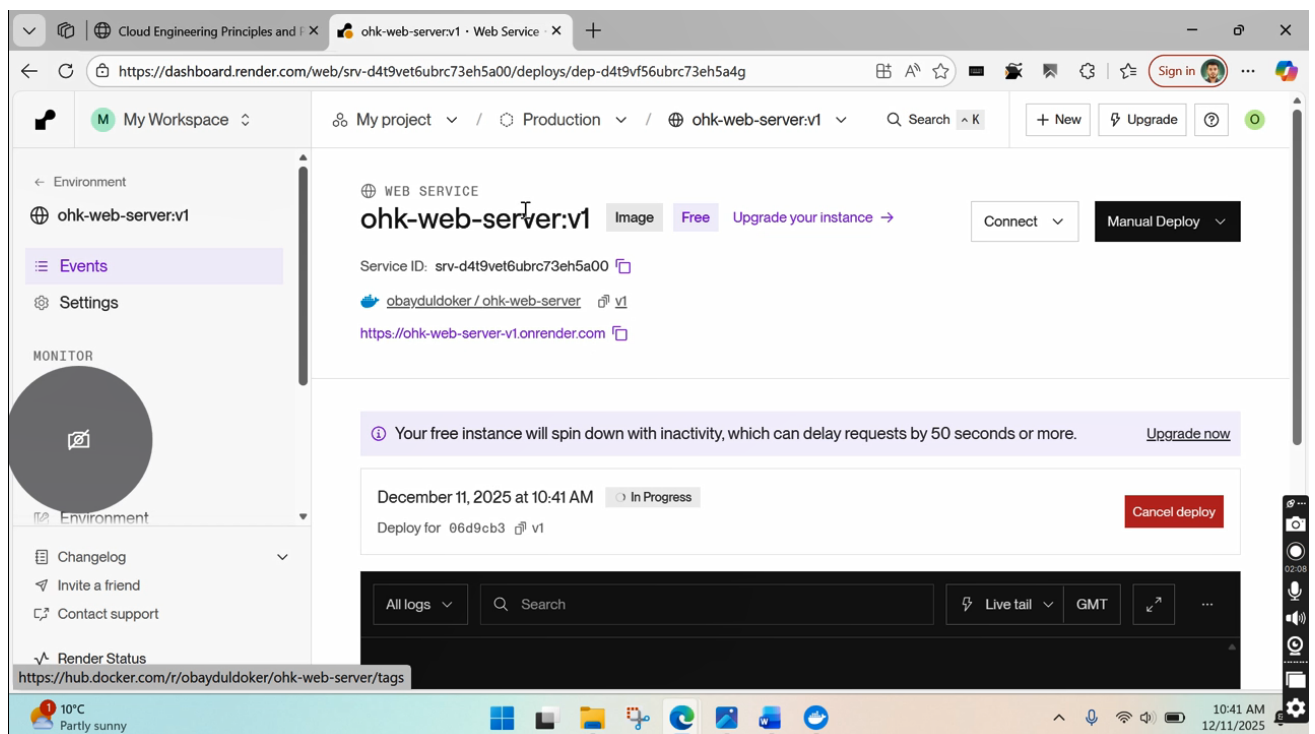
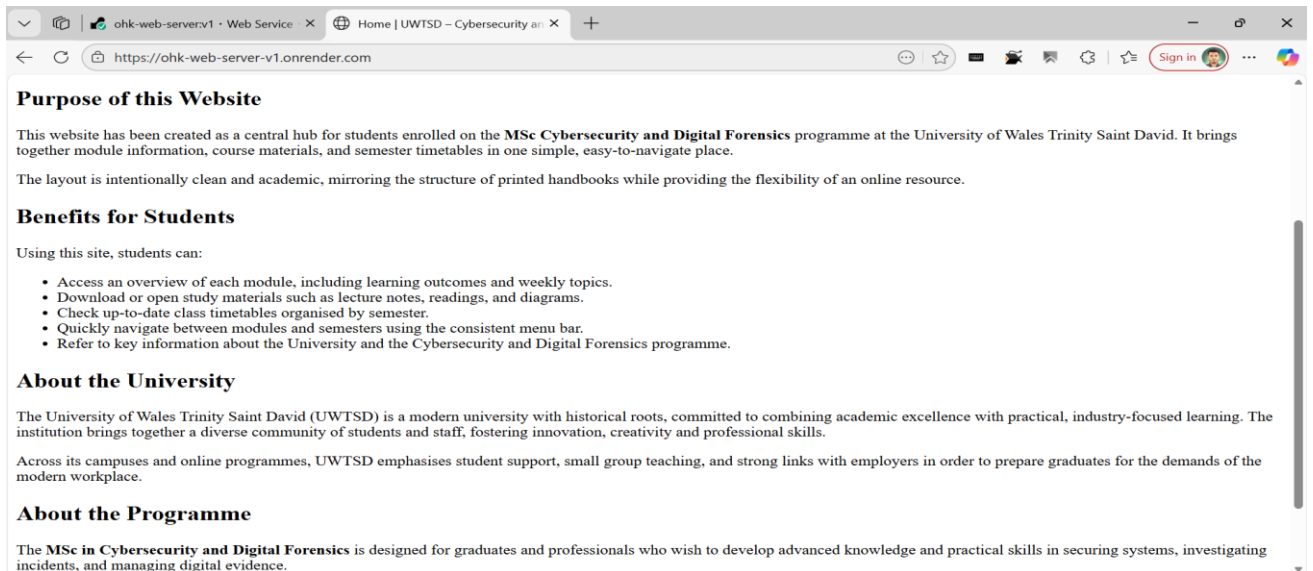


Fig: Deploy



I then used this picture to ship my site and put it ops on Render (Render. com – one of the modern and popular Cloud Hosting Platform / PaaS). Now, this image can be pulled and run on any computer in the world.

Live demo link: <https://my-simple-website-v1.onrender.com/index.html>

Anyone and everyone anywhere in the world may use this link to my website.

2.3.2 Container Volumes:

One challenge with containers is that they are ephemeral if a container is deleted, all data stored inside it is lost. To persist data across container restarts, we use Docker volumes.

Implementation:

```
docker volume create ohk_data_vol
```

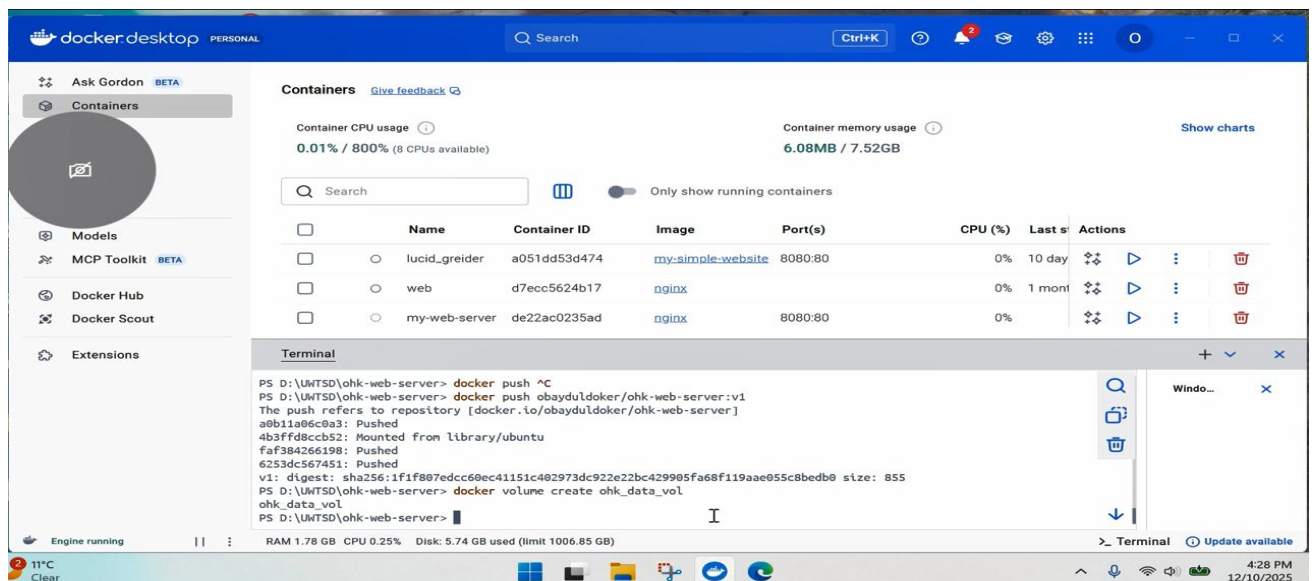


Fig: Create docker volume

Run a container with the volume (Using Alpine Linux)

```
docker run -it -v ohk_data_vol:/data --name storage-container alpine sh
```

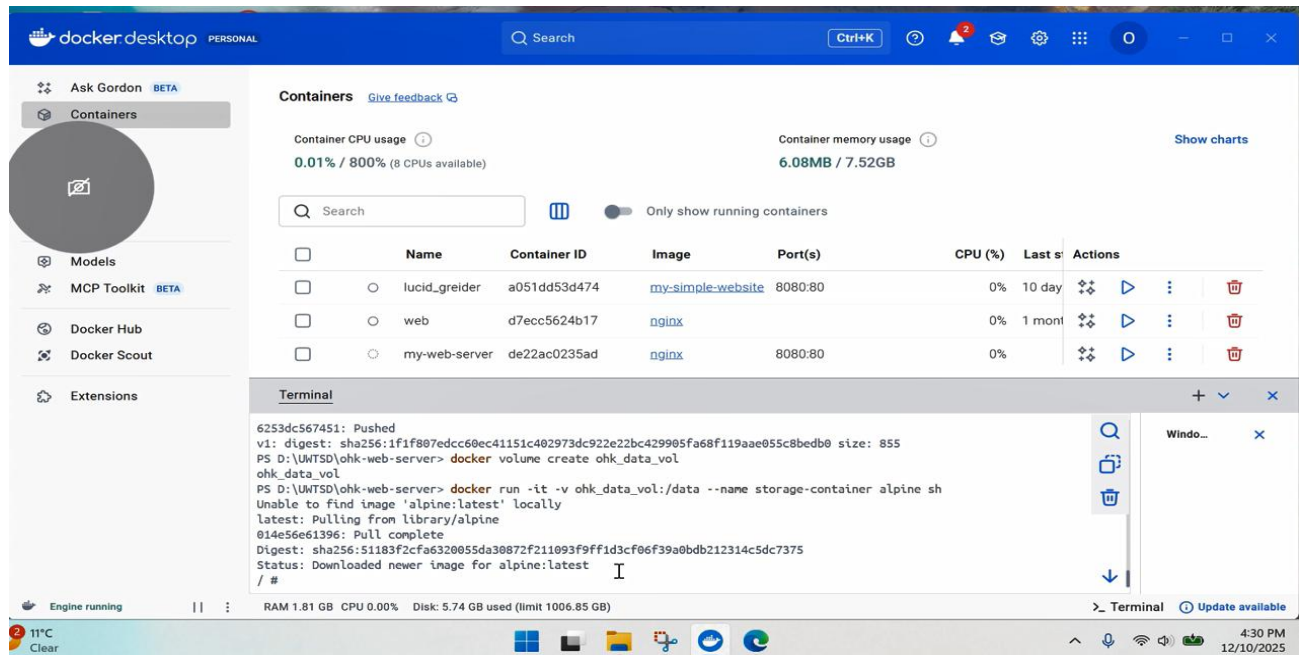


Fig: Run a container with the volume

Create a file inside the container:

```
touch /data/testfile.txt and add text echo "Hello Data" > /data/testfile.txt.
```

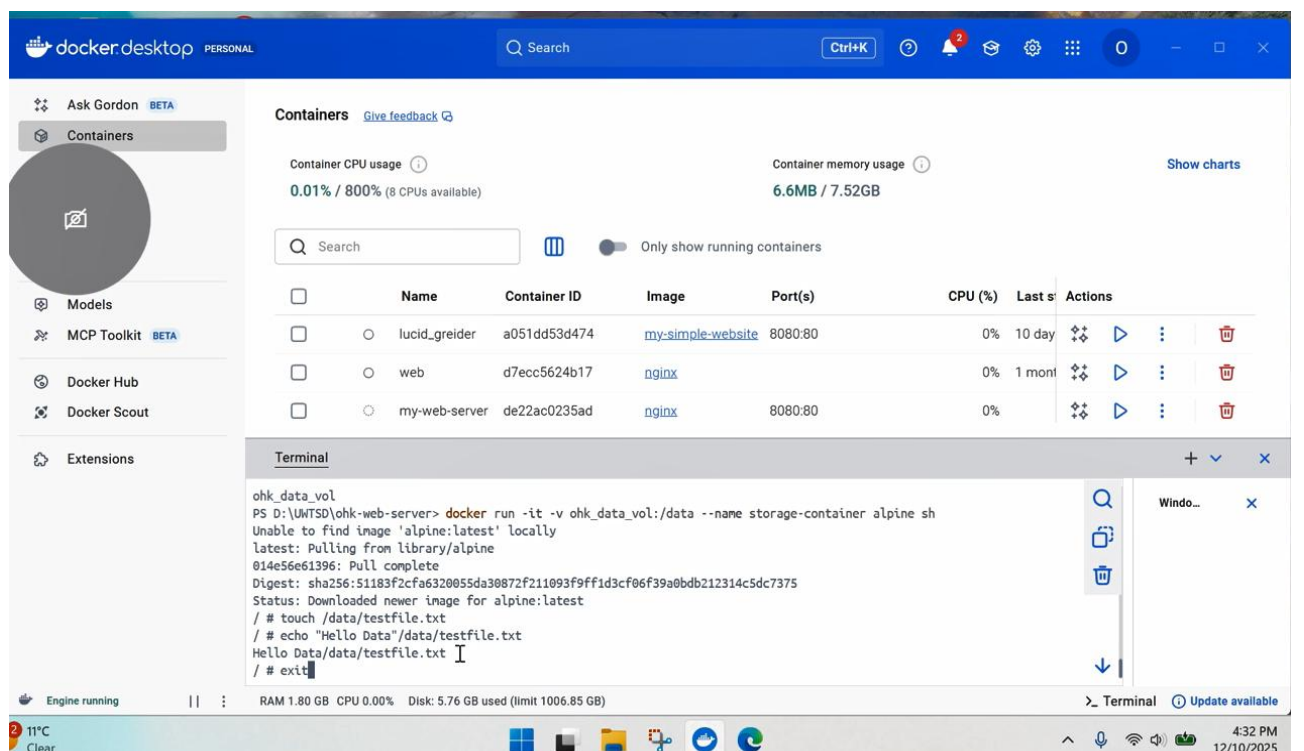


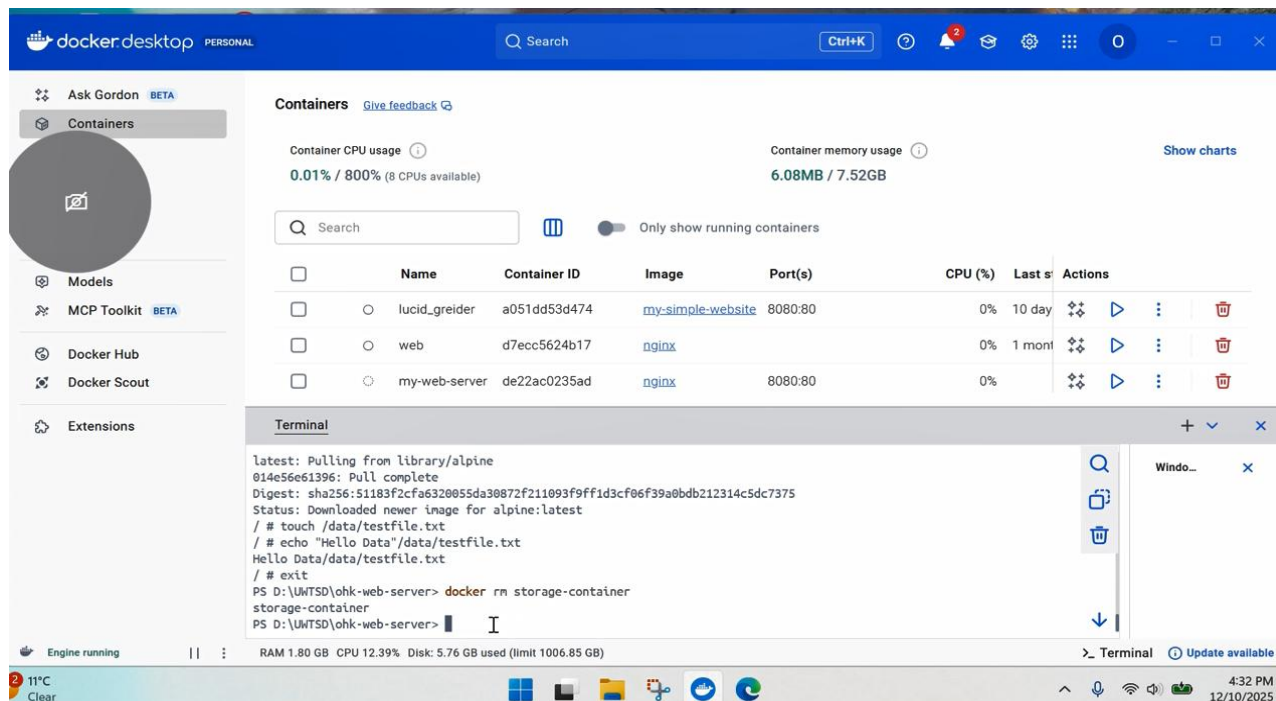
Fig: Create a file inside the container

Exit the container (exit) and delete it:

```
docker rm storage-container.
```

Run a **new** container mounting the **same** volume and check the file.

```
docker run -it -v my_data_vol:/data --name storage-container alpine sh
```



One big problem with Docker is if you delete a container the data in it is also removed. This is called being 'Ephemeral'. But we certainly don't want to lose our database data.

The solution is Volumes. I've only created a volume and making it linked to container. I am saving a file within this container. Now pay attention I'm deleting this container outright.

And when I create a new container and mount the same volume, my file still remains there. So the Volumes keep our data alive when our container dies.

2.3.3 Container Networking:

Networking is one of the key features of Docker Engine which allows you to connect your application with other containers, external users/hosts using network. Docker comes with a few default networking options available, but also allows for custom networks to be created which is great if you need more versatility and control over how your containers communicate.

Create a Custom Network:

With Docker, you can create your own networks to keep the containers on a new network that is separate from the main one and in turn control how they communicate. If you want to create a custom bridge

network, execute

```
docker network create ohk_custom_net
```

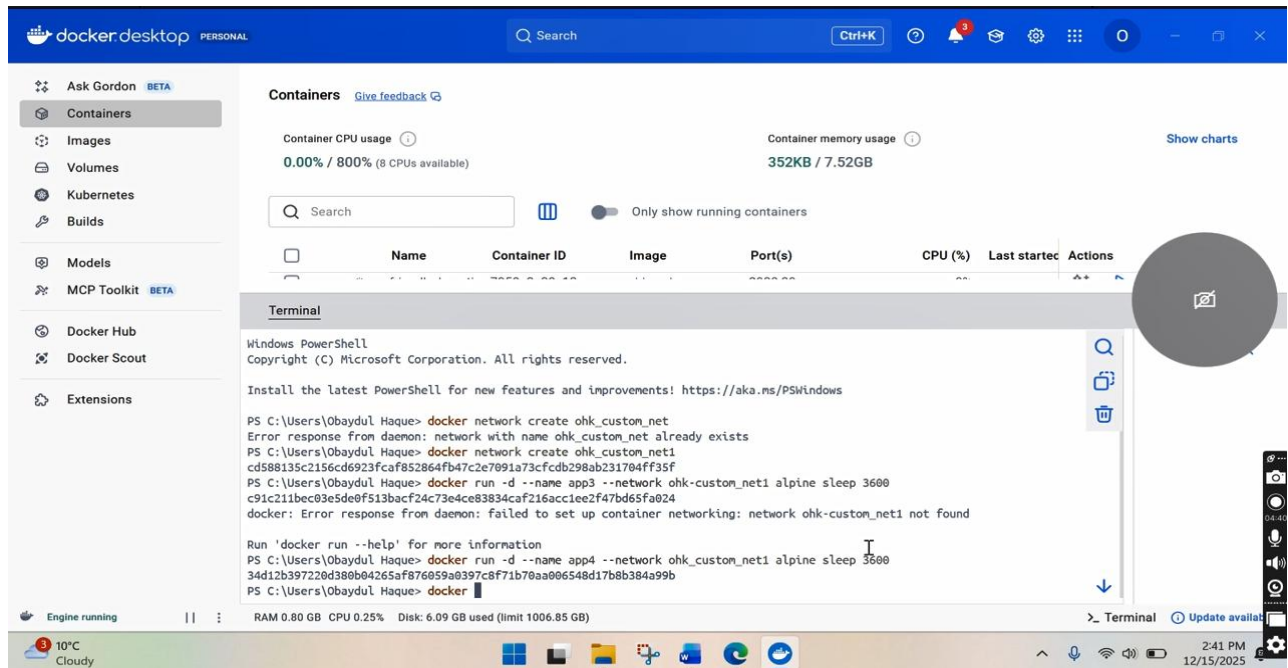


Fig: Create a Custom Network

Create and Run Multiple Containers:

Now, we need to have two containers that can talk to each other. We will use two simple Nginx containers as an example for container networking.

```
docker run -d --name app3 --network ohk_custom_net alpine sleep 3600
```

```
docker run -d --name app4 --network ohk_custom_net alpine sleep 3600
```

You can verify this at the container level by pinging / communicating between your containers, using the container names as hostnames. For instance you may run a command within the app3 container to verify that it can reach the app4 container.

First, run a bash shell in the app3 container:

These are the IP addresses at which you can reach Nginx running inside each container. It is most flexible and recommended for internal use to use the container names (app3 or app4) within the same network but it can be much easier.

```
docker exec -it app1 sh
```

```
ping app4
```

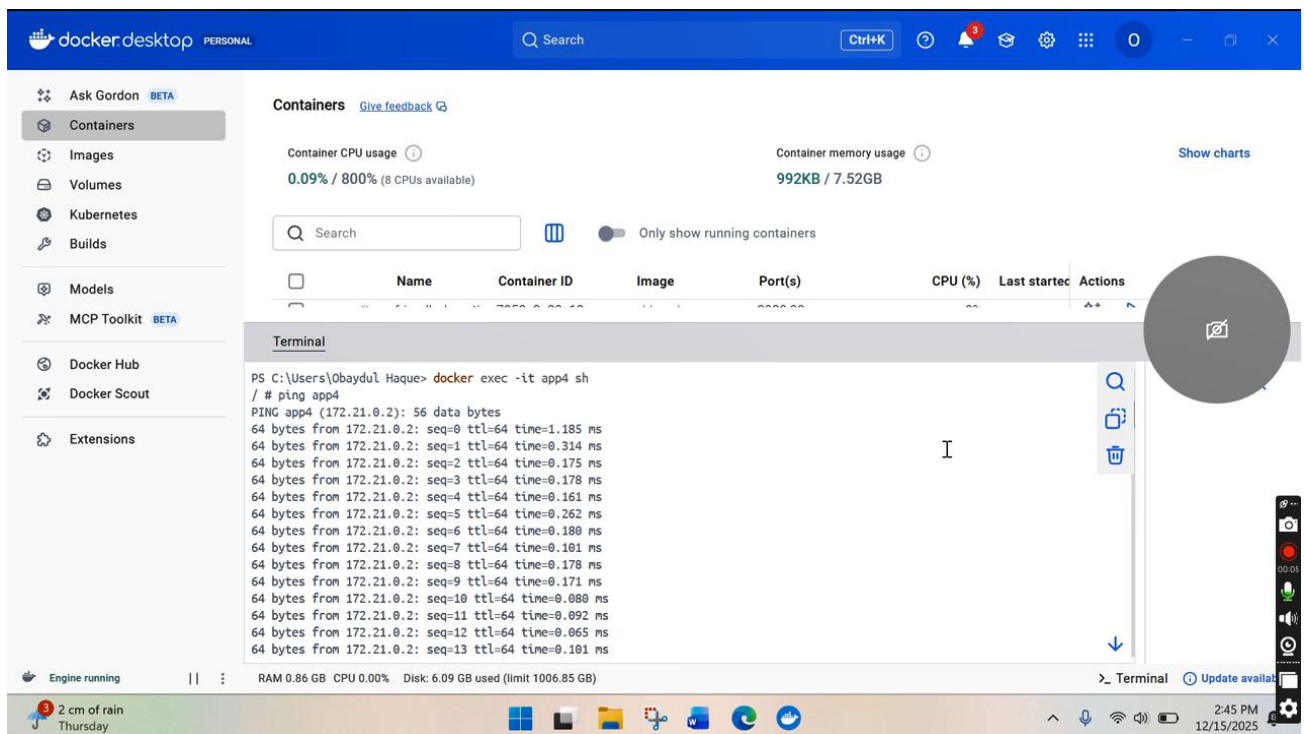


Fig: Ping

In real world, your app and database may live in different containers. They have to communicate with each other to share information.

By default, containers are isolated. Here I'm creating a bridge network using `docker network create`. Next, I have two containers running, app3 and app4, that are on this network.

The magical thing here is: iptables already knows that I am app3, checking this for me (from inside of my container) ping app4. Docker takes care of IP assignments for us and we refer to it as Service Discovery.

2.4. Video Demonstration

I have recorded a comprehensive practical demonstration covering all the required Docker concepts. In this video, I have successfully implemented Virtualization principles, Containerization, Image Building, Publishing to DockerHub, Volume Management, and Networking.

 **Watch the Full Video Here:** [\[Click Here to Watch\]](#)

Key Highlights of the Video:

-  **Virtualization vs. Containerization:** Explained the core differences.
-  **Dockerfile & Image Build:** Created a custom image with Apache Web Server.
-  **DockerHub:** Demonstrated pushing the image to the registry and Deploy.
-  **Data Persistence:** Implemented Docker Volumes to save data permanently.
-  **Networking:** Connected two isolated containers using a custom bridge network.

3.0: Cloud Computing Concepts

In this section, we present key cloud computing concepts, emphasizing the perspective of a global e-commerce platform with peaky traffic patterns as in the case of demand peak events like Black Friday's shopping. The platform needs to be scalable and highly available to handle demanded burst of traffic that may occur anytime.

3.1 Load Balancing

If we have millions of users who want to reach an e-commerce system at the same time, we would need more than one server. Load balancing is a mechanism used to guarantee that the platform can handle large numbers of requests by spreading incoming traffic over several servers [5]. Thereby no single server can become a bottle-neck for the performance, keeping the platform responsive even under heavy load.

Mechanism:

The load balancer is a reverse proxy that evenly distributes traffic among a pool of healthy servers. It's always watching the overall health of each server in the pool, instantly shifting traffic to functioning servers if one fails. This guarantees the platform is available when some of servers have failures.

Cloud Provider Approach (AWS/Azure):

Cloud providers such as AWS or Azure provide managed load balancing services to abstract much of this work [4]. For instance ELB from AWS uses health checks to ensure the up status of backend servers. If a server does not pass its health check, then ELB sends requests only to healthy instances and continues to serve traffic without disruption.

3.2 Scalability

Scalability is the way that allows a system to serve more users without losing its performance. Scalability in cloud In cloud computing, scalability is accomplished using two main methods - vertical scaling and horizontal scaling.

Vertical Scaling (Scale Up):

Vertical scaling adds more resources (CPU, RAM) to an existing server. It's a simple enough approach on its own, but unfortunately it has some caveats (especially when the hardware resources in your server reach their max and hold tight there). Moreover, vertical scaling will likely involve a period of system unavailability to install new or better hardware.

Horizontal Scaling (Scale Out):

Horizontal scaling is accomplished by adding more instances (servers) to the system. This is more agile and scalable approach comparing to outgrowing an existing machine and taking system down for upgrade. Cloud platforms offer a great feature called automatic horizontal scaling, in which new server instances are dynamically launched according to traffic requirements. This allows the platform to scale dynamically and seamlessly as traffic increases, maintaining performance and availability at all times.

3.3 Application Gateways

Application gateways are Layer 7 (Application Layer) devices that offer more sophisticated traffic distribution capabilities than their traditional load-balancing peers, which operate at Layer 4 (Transport Layer). Application gateways provide better capabilities, especially in the complicated and dynamic environments such as cloud-based applications.

Functions:

Routing on Path: App gateways can route traffic according to the URL path. For instance, the /images route might point at an image storage server and the /checkout route to secure payment processing servers. This way, different traffic are handled in the most efficient manner and using the proper backend resources.

Security: Application gateways generally have a WAF (Web Application Firewall) to protect your application from common web vulnerabilities like SQL injection and cross-site scripting (XSS). Also, application gateway can do SSL termination which unburdens the backend servers from executing encryption & decryption. This for the bid server means lower processing and better scalability/performance/security.

4: Driving Forces and Serverless Functions

In this section, we explore the main reasons behind the accelerated growth of cloud hosting and present serverless computing as a new era of cloud architecture. Server less computing is a good jump in granularity towards more and more event-driven models, until finally infra-structure management is entirely taken care of by others, allowing developers to write only the code.

4.1 Causes for Cloud Growth

Cloud computing has developed rapidly because of several driving factors, such as cost-saving and efficiency. With these factors firmly in place, companies have been compelled to move closer to cloud-based solutions and undergo an operational makeover.

Cost (CapEx to OpEx):

A key reason for the adoption of clouds is the conversion of capital expenditure (CapEx) into operational expenses (OpEx). Historically businesses have spent enormous amounts of money upfront on physical infrastructure (data centers, servers etc). It was an investment of capital and a risk. Cloud computing is changing that with a pay-as-you-go model: companies can use resources as they need, and turn big capital investments into manageable operational costs. This change means businesses can take on less financial risk and scale more easily, without having to invest significant capital upfront.

Agility and Flexibility:

With cloud computing, you can provision massive amount of resources in a MATTER OF MINUTES instead of several days or weeks thereby reducing the time to make those resources available to everyone. Real-time availability of resources helps businesses scale up or down to meet the constantly changing needs of the market and customer preferences. This agility is a key competitive differentiator that enables organizations to innovate more quickly and respond more effectively to new challenges.

4.2 FaaS (Functions as a Service) Model Serverless Functions

Serverless computing is a drastic deviation from convention cloud computing as it enables developers to write code without having to think about the infrastructure. With a serverless approach, the underlying infrastructure is abstracted out completely, allowing developers to concentrate on business logic and not dealing with provisioning and scaling infrastructure.

Concept:

Serverless computing, you write microservices that do one thing well (process an image, check your database). These functions are called when they need to be, when something happens in the app as a result of some action being taken by a user (or it can also happen in response to database updates or file uploads.) Everything is managed by the cloud provider. Somehow everything regarding infrastructure management, provisioning (initialisation), scaling (dynamism) and patching is in charge of the same entity, which has not more than 5sec of CPU time allocated for these tasks. Developers can really concentrate on coding cool things. Migrations are smooth. For a development team willing to refactor/change its technical stack or underlying used technology set it is possible with no changes on machines/stacks because already managed only full customization from all dependencies added right in programming code usually can be sufficient.

Event-Driven:

Serverless functions are usually event-driven; it means that they're fired off in response to a certain type of action (HTTP request, database update, file upload). When that event fires, the function is triggered automatically and you don't have to worry about deploying servers or infrastructure. It is an extremely efficient utilization of resources, the system only draws compute resources when required and therefore it can prove to be a more cost-effective solution for sporadic workloads.

One of the key benefits of serverless is the "pay-per-execution" model. Under this model, companies pay for compute time only when they use it measured down to milliseconds used. This makes serverless computing an excellent option for bursty or uncertain workloads, where transient demand is inconsistent. For instance, a function that runs when an image is uploaded might only execute for a few seconds and the user would be charged only for the time spent computing during that brief execution.

Benefits of Serverless Functions:

Cost-effective: Enterprises and businesses are only billed for the specific compute time they consume, so serverless computing can be far less expensive than full-time workloads that experience unpredictable or very low usage.

- **No Infrastructure:** Developers are no longer required to manage servers, scale infrastructure or take care of patches. This simplifies operations and lets teams focus on development.
- **Automatic Scalability:** Serverless infrastructures scale automatically in the face of demand, meaning the system is prepared to cope with sudden peaks in your traffic or usage without human intervention.
- **Quicker development:** Without infrastructure to manage, developers can focus on crafting business logic and pushing code out more quickly speeding up the overall development cycle.

4.3 Comparison: Containerization vs. Serverless

The following critical analysis compares these two dominant technologies:

Feature	Containerization	Serverless
Control	High. Developers are software In OS environment, library and run-time version.	Low. The Black Box is vendor-managed infrastructure.
Cost Model	Running Cost. You merely pay for the underlying instances on an hourly basis no matter your traffic.	Pay-per-execution. When the code runs, you pay. Zero traffic equals zero cost.
Use Cases	Background Tasks, Microservice with special environments, Legacy migration.	Image processing (download images sends to S3 bucket), Sporadic traffic, Real-time APIs.

Driving Forces Analysis:

When reviewing cloud computing offerings one should look at the type of workloads are going to run, so that they can maximize cost savings and performance [7]. Each of containerization and serverless computing have certain merits, depending on characteristics of the workload. Here's how each approach lines up with different business needs:

Containerization for Consistent, Heavy Workloads:

If your workloads are fairly static, predictable and very busy over a significant duration, they may be excellent candidates for containerization. In such cases, the power of containers becomes evident for the following purposes:

Steady Demand cost effective: Containers are most cost effective with demand that is steady and of high volume, as they enable businesses to reserve resources that can be predicted in advance without the overhead being incurred from provisioning and de-provisioning them.

Resource Efficiency Containers run on the host OS kernel, which offers a light-weight solution compared to traditional VMs with less overhead and better resource utilization. Businesses can therefore launch multiple containers on less physical machines, which ultimately will save costs.

More predictable performance: Containers are simpler to scale as they can be replicated across instances. The concept of horizontal scaling, companies have ability to increase their capacity (and performance) in a linear fashion - proportional to demand.

Serverless Computing (SLC) for Sporadic, Unpredictable Workloads:

For workloads that are intermittent, event-driven or prone to bursts of unpredictable activity, serverless computing offers a more cost-effective model based on "pay-per-execution." Here's why:

Pay Per Compute Time: In a serverless world, such as AWS lambda or Azure functions, you only pay for the compute consumption that occurs when your function is running. This is highly useful for low- to fluctuating-traffic, because there exists no fixed costs for the idle resources. For workloads that do not need constant computing power, serverless computing can result in cost savings because it charges for actual use rather than idle time.

Seamless Scalability: Serverless platforms will scale up or down automatically based on events like a file upload, an HTTP request, or a change in the database. Volume elasticity enables your business to accumulate volumes with varying pavement demands without manually managing resources for better cost and performance.

Less Infrastructure Management: No need to keep servers etc. maintained, businesses also don't worry about additional costs like provisioning and scaling; the server scales up with your business as you grow without increasing burden on admin resources for maintenance.

Hybrid Approach by William Holley: Merging Containerization and Serverless for Complicated Infrastructures:

And it's not like the real world, where the best choice is often a mix of container delivery and serverless. Such a hybrid model enables companies to optimize the season, cost and resource management based on the workload nature.

Containerization for Core Systems: Enterprises can benefit from containerization of the core components which need predictable and consistent performance. For instance, one may deploy backend services, databases or stateful applications requiring a stable orchestration of long-scale processes.

Serverless for event-driven/bursting workloads: It can also be used for bursty, occasional tasks (e.g., serving user requests) or longer running workflows that have specific triggers. This enables businesses to easily provision capacity as needed to manage unpredictable traffic surges, e.g., during flash sales, holiday shopping seasons or special campaigns.

Cost Savings: To save on costs, businesses can leverage both technologies and deploy the most suitable solution for their workload. Serverless functions can also scale out as high-traffic times without having to provision more resources, and the containerized services will be able to run indefinitely for mission-critical applications that you need up 24/7.

Conclusion

This trajectory of cloud computing, from hardware-intensive virtualization to resource-effective containerization and on to serverless computing, represents a profound transition in how IT infrastructure is utilized. These innovations have fundamentally transformed how applications are built, scaled and managed in the cloud. As companies from startups at a garage to multinational corporations move more and more of their workloads, processes, and products into cloud services, the flexibility, scalability and cost-effectiveness of both technologies help transform cloud computing one of the main concepts in modern IT infrastructure.

In the following section, we investigated containerization as a solution for providing high portability, efficient resource utilization and rapid deployment required by modern cloud-based apps. Docker, being the de facto industry standard for containers, proved how they make development and deployment pipelines faster by abstracting away cumbersome operations that were formerly directly managed by the infrastructure.

We have also investigated serverless computing (FaaS) that provides unprecedented agility and cost efficiency, especially for event driven applications with burst workload [6]. The pay-per-execution paradigm of serverless platform such as AWS Lambda or Azure Functions, allows the organization to dynamically scale up without managing hardware, which is perfect for bursty workloads.

The key value drivers for cloud (cost saving, agility and flexibility) have driven the pace of adoption. Enterprises are now tasked to select the appropriate architecture as per their workload characteristics. Containerization is best for workloads needing high control, predictable performance, and resource utilization while serverless computing is best for event driven processing scenarios that have a wide variation in demand.

In summary, the cloud horizon is not a one size fits all. The halfway house, with containers for core systems and serverless for dynamic, bursty workloads is the best strategy in most enterprise environments. And by cloud engineers, I mean getting hands dirty with these technologies & knowing when to use them according to business requirements & scale needs is very significant for anyone in today's evolving cloud world. The evolution from virtualization and containers to serverless computing underscores the cloud's ability to support business innovation—putting it firmly at the center of digital transformation today.

References

- [1] Hwang, K., & Dongarra, J. (2017). *Cloud computing: Principles, systems and applications*. Springer.
- [2] Rosenblum, M., & Garfinkel, T. (2005). *Virtual machine monitors: Current technology and future trends*. IEEE Computer, 38(5), 39-47.
- [3] Boettiger, C. (2015). *An introduction to Docker for reproducible research*. ACM SIGOPS Operating Systems Review, 49(1), 71-79.
- [4] Docker Inc. (2021). *Docker Documentation*. [Online]. Available: <https://docs.docker.com/>.
- [5] Cloud Academy. (2020). *What is Load Balancing in Cloud Computing?* [Online]. Available: <https://cloudacademy.com>.
- [6] Williams, S., & Yegulalp, S. (2019). *Serverless Computing: A New Paradigm for Cloud Architecture*. IEEE Cloud Computing, 6(2), 25-33.
- [7] McKendrick, J. (2018). *Why Businesses Are Moving to the Cloud*. Forbes. [Online]. Available: <https://www.forbes.com>.
- [8] Holley, W. (2020). *Hybrid Architectures: Combining Serverless and Containers*. International Journal of Cloud Computing and Services Science, 8(4), 122-130.

