

Students Name: Md. Mashrurul Haque

Student ID: 2435257

Module: Cloud Security Principles and Practice

Programme: MSc in Cybersecurity and Digital Forensics

Date of Submission: 18/12/2025

Table of Contents

1. An Overview of Virtualisation, Containerisation, and Cloud Computing Technologies	4
2. Virtualization	6
2.1 Virtualization of Shared Hardware	6
3. Containerisation.....	8
3.1 Docker Images	8
3.2 Docker Volumes	11
3.3 Docker Networks	13
4. Cloud Migration and Architectural Concepts in the Context of Khan Academy	16
5. The Complementary Relationship Between Load Balancing and Scaling	18
6. Customer Value in Integrated Cloud Architecture	19
7. Key Drivers Behind Cloud Adoption: Cost, Flexibility and Strategic Benefits	20
8. Serverless Computing and Its Contrast with Containerisation	21
9. Conclusion	22
10. References.....	23

List of Figures

Fig. 1: Docker file Skeleton	09
Fig. 2: Building a Docker Image.....	09
Fig. 3: Running a Docker Container.....	09
Fig. 4: Docker Container Running on Localhost.....	10
Fig. 5: Pushing a Docker Image to DockerHub.....	10
Fig. 6: Docker Image Successfully Published on DockerHub.....	11
Fig. 7: Creating and inspecting the Docker volume.....	12
Fig. 8: Mounting Volume to the Container.....	12
Fig. 9: Inspecting Volume after Mounting to the Container.....	13
Fig. 10: Creating and Verifying Network Creation.....	14
Fig. 11: Assigning Network to the Container.....	14
Fig. 12: Inspecting The Network Configuration	15
Fig. 13: Verifying the container's IPv4 address allocation.....	15

1. An Overview of Virtualisation, Containerisation, and Cloud Computing Technologies

Modern computational services depend on blending three key technological ideas. These are virtualization, containerization such as Docker, and the cloud computing service model. All of them work together to form the idea of computing as an everyday utility. In this setup, resources become available right when needed. Users pay only for what they use [1].

Cloud computing captures the notion of treating computing like a utility [2]. Resources get accessed easily whenever required [1]. It covers applications provided as services via the Internet, often called Software as a Service or SaaS. It also includes the vast data center setups with hardware and system software that support those services. One key aspect of cloud computing involves elasticity. This means resources can expand or shrink fast to handle sudden spikes in demand. Such flexibility lets users skip long-term planning for resources. It avoids wasting money on extra capacity or losing business from shortages. From an economic angle, the approach uses big, affordable data centers that achieve major savings through scale. These create major savings through scale, which makes the whole utility model cost-effective [2]. When a cloud provider offers access to the public on a pay-as-you-go plan, people call it utility computing [1].

The dynamic handling and setup of resources play a big role in cloud elasticity. Virtualization provides the core support for this. A Virtual Machine Monitor, or VMM, also referred to as a hypervisor, acts as a slim software layer. It runs straight on the hardware and divides it into separate Virtual Machines (VMs). The VMM oversees every physical resource. It guarantees solid separation, so software in one VM stays unable to affect the VMM or other VMs [3]. That separation creates a strong security wall. In more complex setups, the design enables Virtual Machine Introspection (VMI). Here, an outside system checks the hardware state of a watched VM to examine its internal software. VMI proves vital since it keeps the monitoring tool visible and tough against attacks, even if the guest operating system faces total compromise [4]. Certain virtualization tools, like Xen, use paravirtualization. This requires minor changes to the guest operating system. It presents a smoother virtual setup, which boosts speed and keeps resource isolation firm [3].

Containerization, as seen in tools like Docker, serves as a growing option or addition to VMs. When workloads are configured correctly to run efficiently, containers match or exceed VM performance nearly in every measured metrics. They add almost no extra load compared to apps running straight on bare metal hardware. Containers deliver the speed of regular non-virtualized servers while bringing the separation and management perks of VMs. Still, some features in container tools can hurt speed for tasks heavy on input-output. Things like layered filesystems, such as AUFS, or Network Address Translation for networks create these issues. As a result, users face choices between simple handling and top performance [5].

These technologies, which range from virtualization providing hardware-level isolation to containerization offering lightweight, high-performance deployment, are essential for implementing the elastic, pay-as-you-go utility model of cloud computing [1][2]. With this overview established, the report now turns to a detailed exploration of each technology, covering its core concepts, working principles, and advantages. Docker is used to demonstrate the containerisation methodology.

2. Virtualization

Virtualization is the abstraction of physical computing resources so that multiple isolated virtual machines (VMs) can run concurrently on a single host. A hypervisor, or virtual machine monitor (VMM), mediates privileged operations, schedules CPU time, manages memory and multiplexes I/O across guest operating systems [3][6][9]. Two major hypervisor architectures exist: Type 1 (bare-metal) runs directly on hardware, and Type 2 (hosted), which operates as an application on top of a traditional operating system [6]. Virtualization mechanisms differ based on how they handle privileged instructions. Full virtualization provides a complete hardware abstraction for unmodified guest OSs, historically achieved using binary translation to rewrite sensitive instructions [11]. Paravirtualization improves performance through guest-aware hypercalls, while hardware-assisted virtualisation (Intel VT-x, AMD-V) integrates direct CPU support, reducing the overhead of older VMM techniques [10].

Virtualization provides several well-established benefits. By consolidating multiple workloads on fewer machines, organizations achieve significantly improved resource utilisation and cost efficiency, a trend documented across early and modern hypervisor deployments [3][10]. Strong isolation between VMs enhances fault containment and security, as failures remain confined to individual guests rather than the entire system [9]. The encapsulation of VM state also enables snapshotting, live migration, reproducibility, and simplified management, features widely recognised as core operational advantages in virtualized data centres [10][11].

2.1 Virtualization of Shared Hardware

When several VMs share a single host, the hypervisor arbitrates access to CPU cores, memory and I/O devices. CPU time is distributed via virtual CPU (vCPU) scheduling, memory is controlled using shadow or nested page tables, and I/O is multiplexed through virtual device drivers or passthrough techniques such as SR-IOV. Although effective, shared hardware introduces performance interference, where one VM's workload impacts another due to contention in caches, memory controllers or I/O queues. Studies consistently document measurable cross-VM interference even under modern scheduling and isolation techniques [7][11].

Shared microarchitectural components also create side-channel vulnerabilities. Because VMs inevitably share CPU caches and predictors, attackers can infer sensitive data by analysing timing patterns or cache line behaviour. Research shows that scheduler-based isolation and

resource partitioning can reduce, but not fully eliminate, these leakages [8][9]. Achieving strong isolation while maintaining high consolidation efficiency remains a persistent challenge.

Despite its strengths, virtualization introduces non-negligible overhead due to context switching, memory address translation, and device emulation, which collectively increase latency and reduce raw performance compared to bare-metal execution [11]. Running a full guest OS for each VM also results in a larger memory and storage footprint, making VMs less suitable for highly elastic or lightweight workloads [10]. Furthermore, persistent issues such as resource contention, complex VM lifecycle management, and unresolved side-channel vulnerabilities present structural limitations documented across multiple studies [7][8][9]. These drawbacks motivate the adoption of container-based virtualization, where applications share the host kernel rather than replicating entire operating systems. The next section examines containerization as a more lightweight and scalable alternative.

3. Containerisation

Containerisation serves as an operating system level virtualisation approach. It packages an application along with its dependencies into a lightweight and portable unit called a container. Containers differ from virtual machines in keyways. They share the host kernel while isolating user space environments using kernel features like namespaces and control groups, often called as Cgroups. This setup cuts down on resource overhead quite a bit. It also speeds up start up times. Such benefits make it ideal for microservice architectures and continuous delivery pipelines. In those setups, rapid scaling and reproducibility matter a great deal [12][13][14]. During runtime, containers rely on kernel namespaces for isolation. These handle process identifiers, network interfaces, mount points, and inter process communication, or IPC. Cgroups take care of resource limits and accounting. Layered filesystems build the container root filesystems from immutable layers. OCI compliant runtimes like runc and containerd manage these aspects. They create and oversee the lifecycles of containers. All these elements together allow for high workload density. They support quick instance creation and efficient storage use. Performance studies comparing containers to virtual machines highlight these advantages consistently [14][15]. Containers prove useful for reproducible deployments. They avoid dependency conflicts by bundling libraries. Plus, they enable fast horizontal scaling in distributed systems [12][13].

3.1 Docker Images

A Docker image acts as an immutable and read only artifact. It consists of stacked filesystem layers plus metadata that outline how a container runs. Each layer captures a change from a Dockerfile instruction. That could mean adding binaries or tweaking configuration files. The layered design aids caching and reuse. It makes distributing images efficient since only changed layers transfer. Large scale analyses of public registries confirm this point repeatedly. They stress that image composition, layer reuse, and size optimisation drive operational efficiency. They also boost security practices [16]. Developers use clear Dockerfiles to document images. These specify base images, exposed ports, environment variables, needed volumes, and execution commands. Once docker image has been built, the image can be published to the Docker Hub by authenticating with proper credentials. This publishing step promotes reuse. It simplifies integration into CI/CD workflows. Still, it brings supply chain risks. Those call for vulnerability scanning and cutting out unnecessary packages [16][17].

Below is a demonstration of how to create an image in docker:

1. Creating Docker image

```
docker build -t mdmashrurulhaque/interactive-portfolio:v1 .
```

2. Running an image

```
docker run -d -p 8085:80 --name mdmashrurul-web mdmashrurulhaque/interactive-portfolio:v1
```

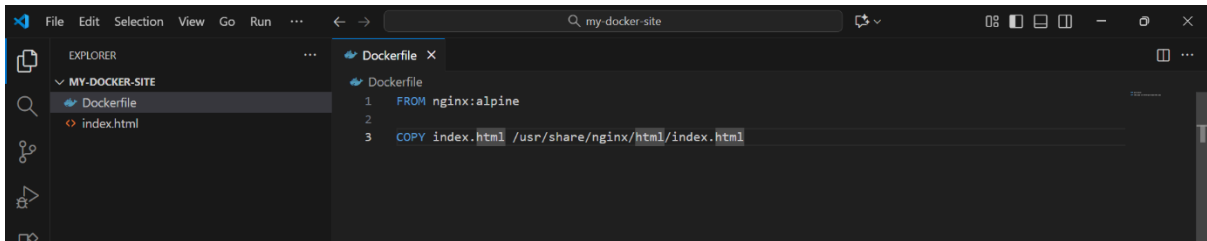


Fig. 1: Docker File Skeleton

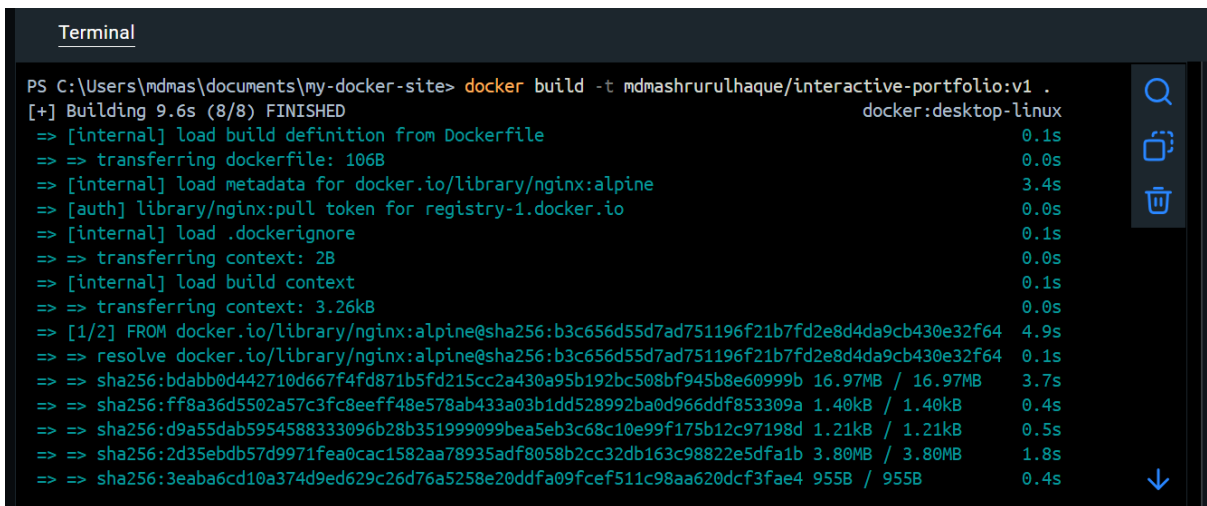


Fig. 2: Building a Docker Image

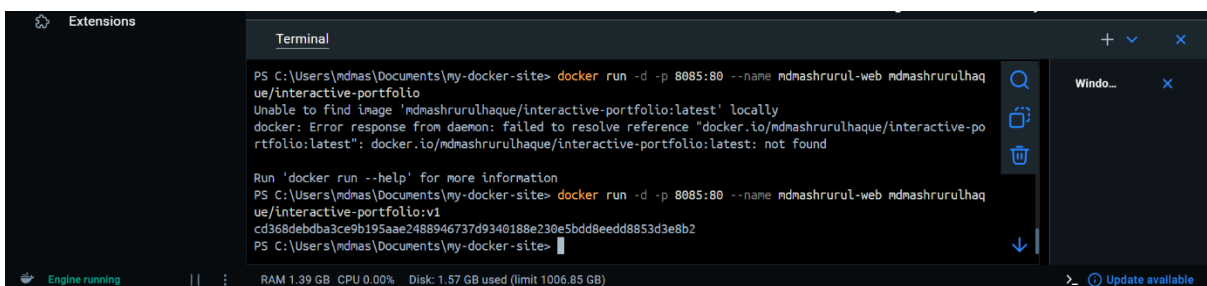


Fig. 3: Running a Docker Container

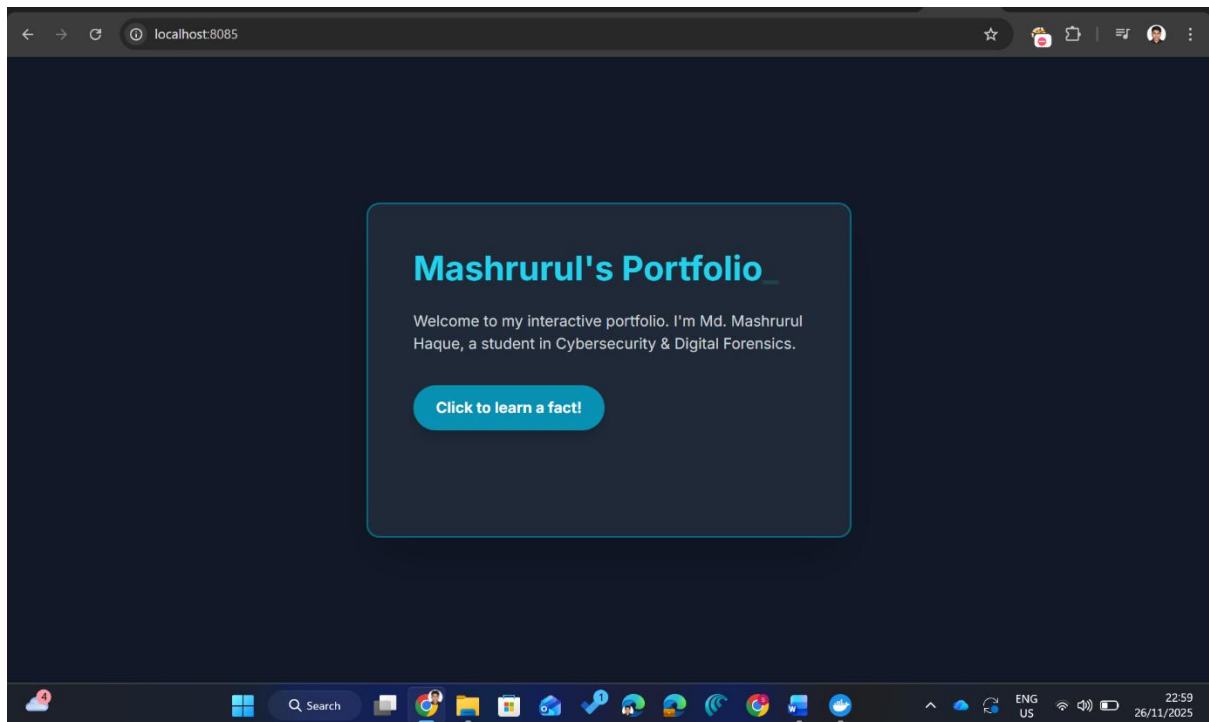


Fig. 4: Docker Container Running on Localhost

```
Terminal
=> => exporting manifest list sha256:80d2658deed5270e945ba059dd0813bdf101ceb3eb2986d2cfe247064902 0.0s
=> => naming to docker.io/mdmashrurulhaque/interactive-portfolio:v1 0.0s
=> => unpacking to docker.io/mdmashrurulhaque/interactive-portfolio:v1 0.1s
PS C:\Users\mdmas\documents\my-docker-site> docker push mdmashrurulhaque/interactive-portfolio:v1
The push refers to repository [docker.io/mdmashrurulhaque/interactive-portfolio]
2d35ebdb57d9: Pushed
194fa24e147d: Pushed
3eaba6cd10a3: Pushed
ff8a36d5502a: Pushed
bdabb0d44271: Pushed
8f6a6833e95d: Pushed
df413d6ebdc8: Pushed
d9a55dab5954: Pushed
1e1f43d03fae: Pushed
37a4cab844b8: Pushed
v1: digest: sha256:80d2658deed5270e945ba059dd0813bdf101ceb3eb2986d2cfe247064902ac2b size: 856
PS C:\Users\mdmas\documents\my-docker-site>
```

Fig. 5: Pushing Docker Image to DockerHub

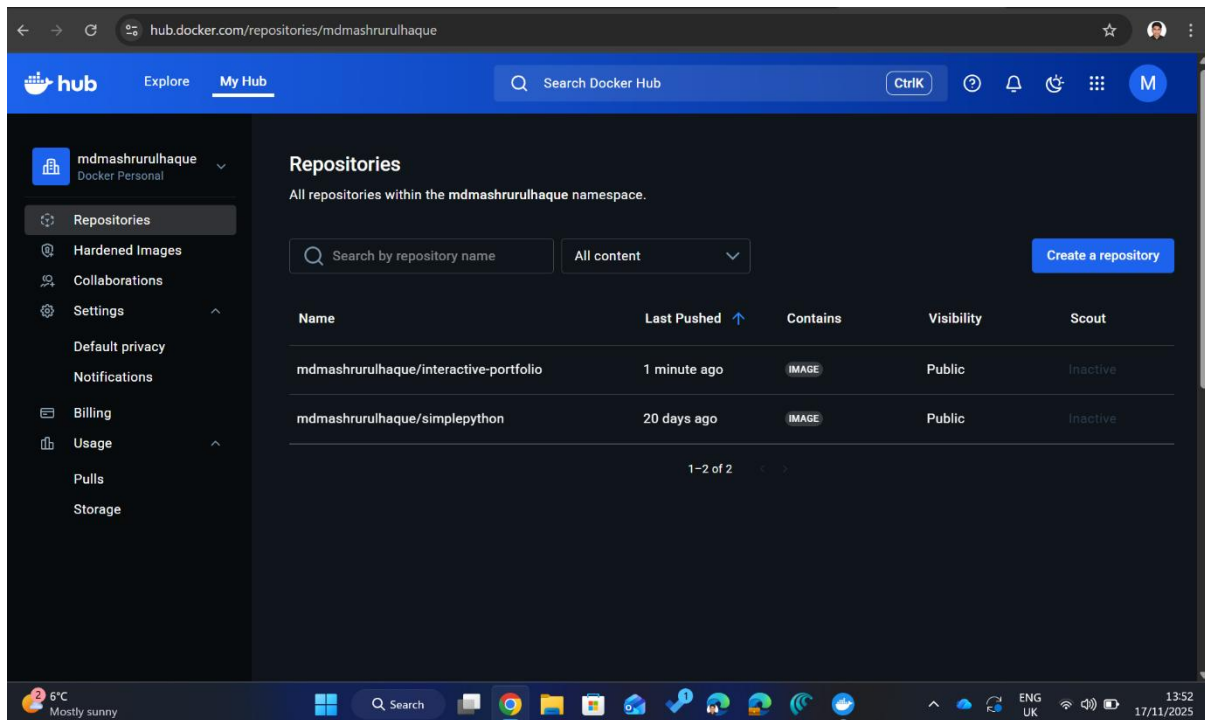


Fig. 6: Docker Image Successfully Published on Docker Hub

3.2 Docker Volumes

Docker volumes provide persistent storage that remains available even when containers stop, restart, or get recreated. Since a container's writable layer is ephemeral, volumes become essential for databases, logs, and other workloads that must retain state. They also tend to deliver better I/O performance than the container's temporary filesystem. Docker handles two main types of volumes. Named volumes get automatically managed by the Docker engine. Bind mounts map host directories straight into containers. Things get more complicated in multi-node setups. There, persistent storage needs orchestrator-level abstractions. Examples include Kubernetes Persistent Volumes and CSI drivers. These ensure consistency and durability. Empirical assessments of container persistence at scale often point out this challenge [18].

Below is a demonstration of how to create a volume in Docker:

3. Creating Docker Volume

```
docker volume create volume-for-interactive-portfolio
```

4. Mounting the volume to the container

```
docker run -d --name volume-for-interactive-portfolio -p 8085:80 -v portfolio-  
vol:/data mdmashrurulhaque/interactive-portfolio:v1
```

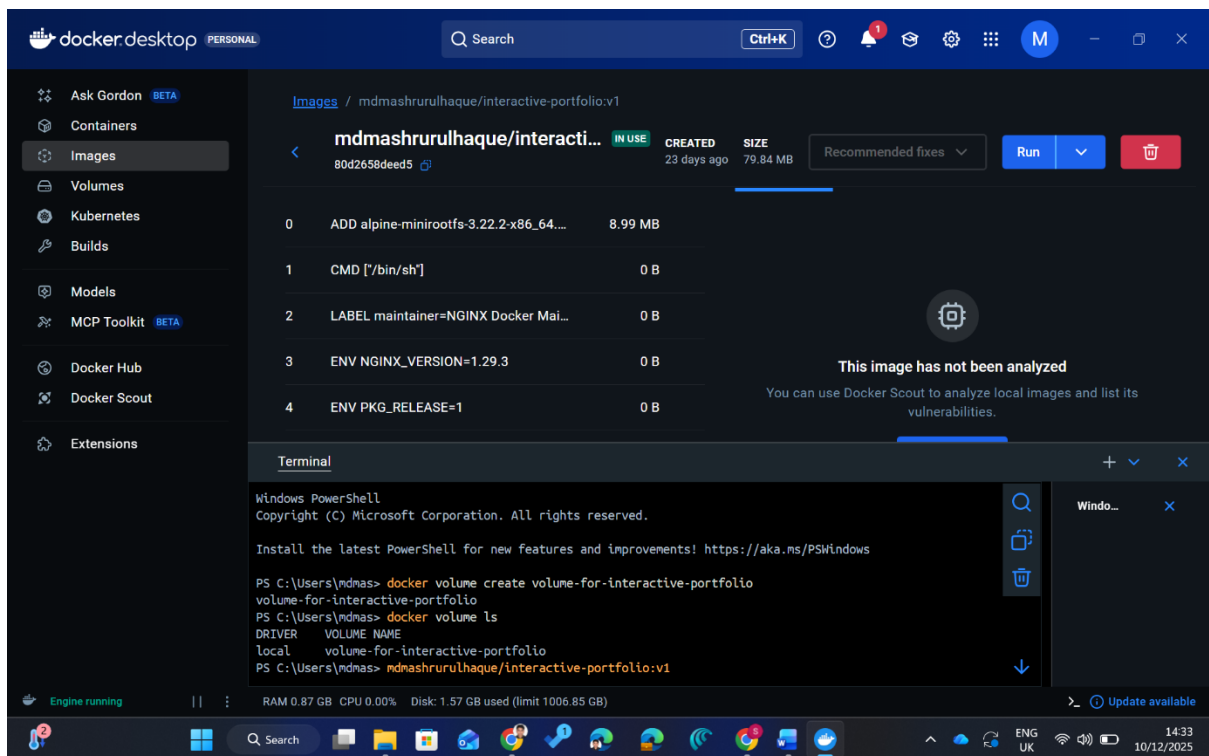


Fig. 7: Creating and inspecting the Docker volume

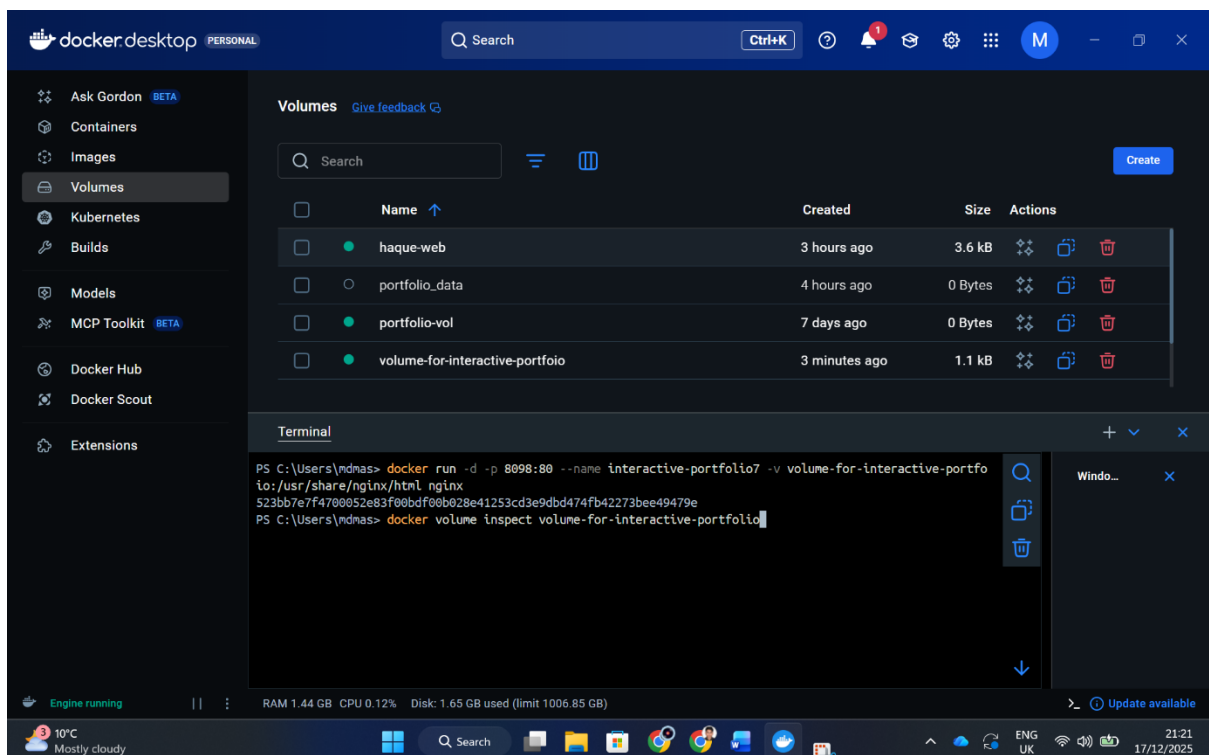


Fig. 8: Mounting Volume to the Container

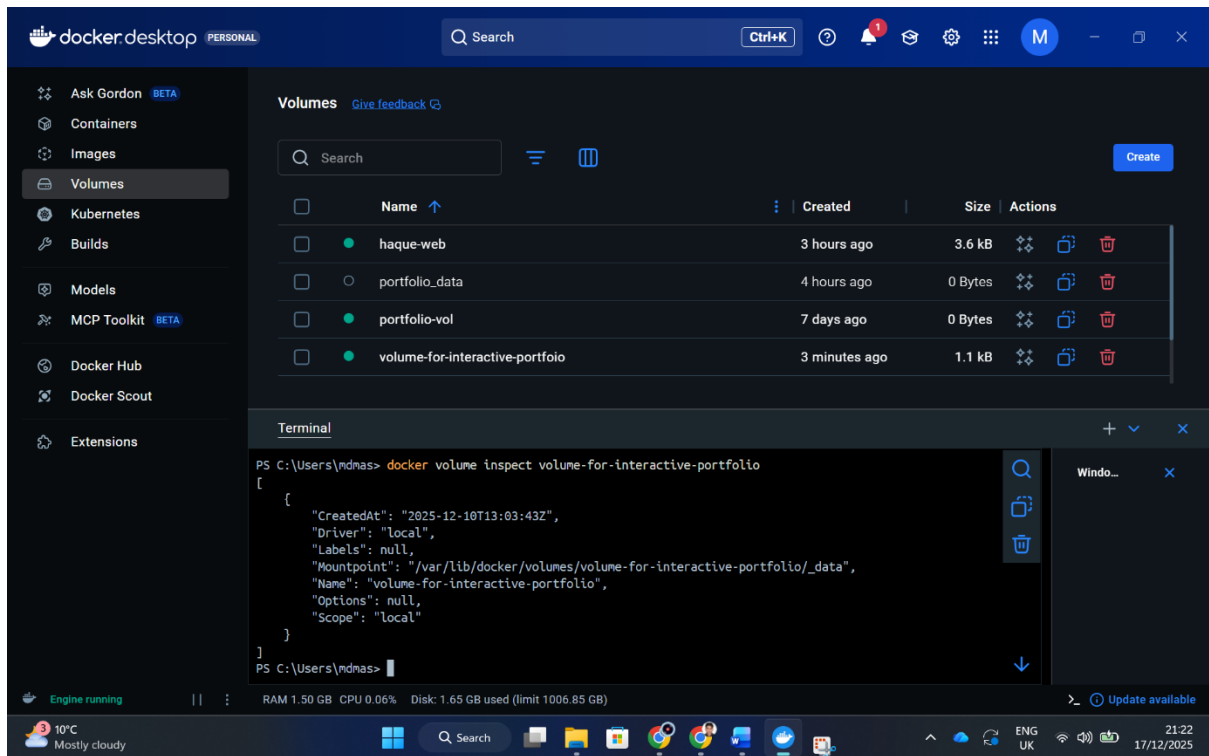


Fig. 9: Inspecting Volume after Mounting to the Container

3.3 Docker Networks

Container networking handles communication, service discovery, and isolation. It works between services on one host or across distributed clusters. Docker offers various network drivers. The default bridge driver creates isolated local networks. The host driver lets containers use the host network namespace directly. The overlay driver supports multi host communication in clusters. Macvlan assigns unique MAC addresses to containers on physical networks. These models let developers build microservices in secure and flexible ways. Orchestrators like Kubernetes build on this with CNI plugins. They add network policies and distributed service routing [15].

Below is a demonstration of how to create a network in docker:

1. Creating Docker Network
`docker volume create interactive-portfolio-network`
2. Mounting the volume to the container
`docker run -d --name interactive-portfolio --network interactive-portfolio-network -p 8080:80 -v volume-for-interactive-portfolio:/data mdmashrurulhaque/interactive-portfolio:v1`

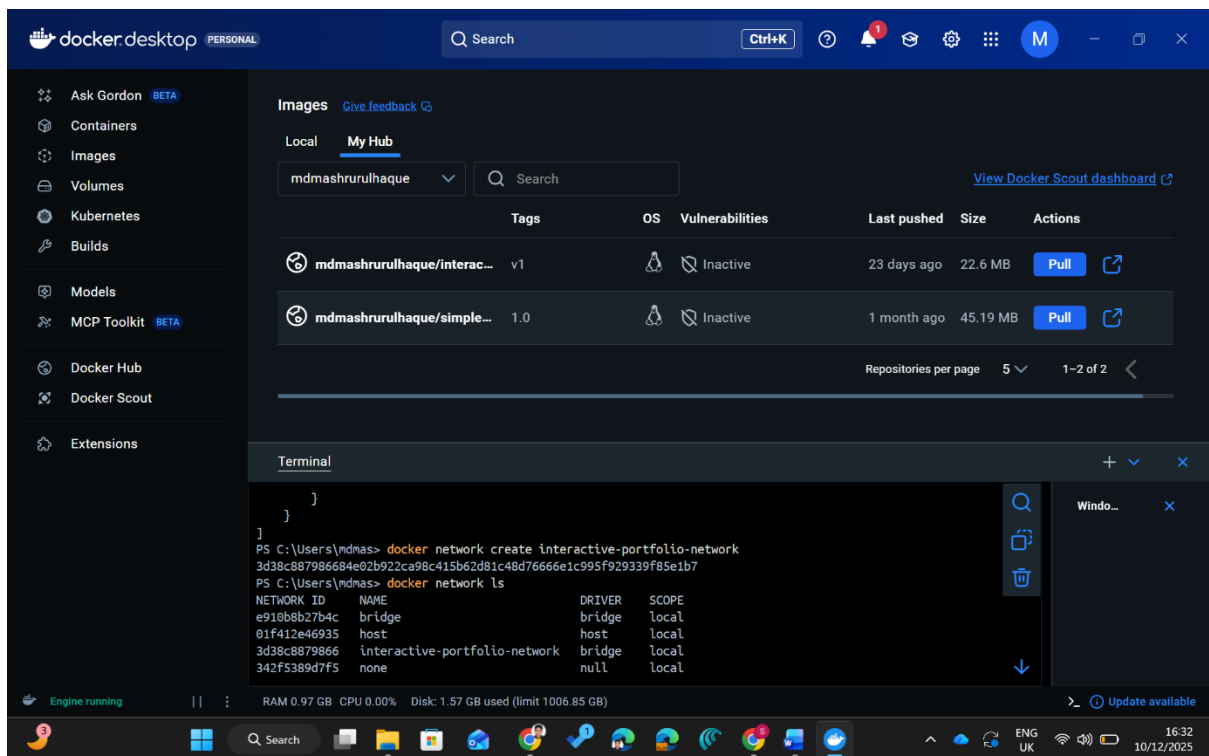


Fig. 10: Creating and Verifying Network Creation

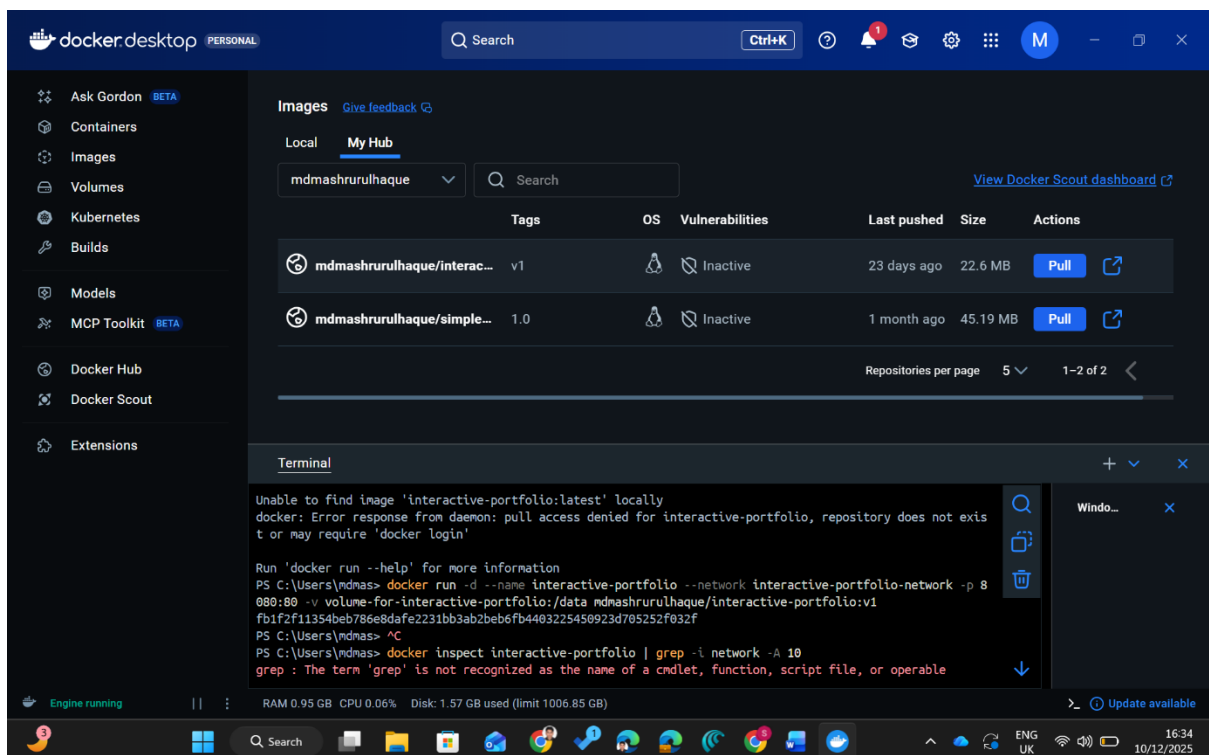


Fig. 11: Assigning Network to the Container

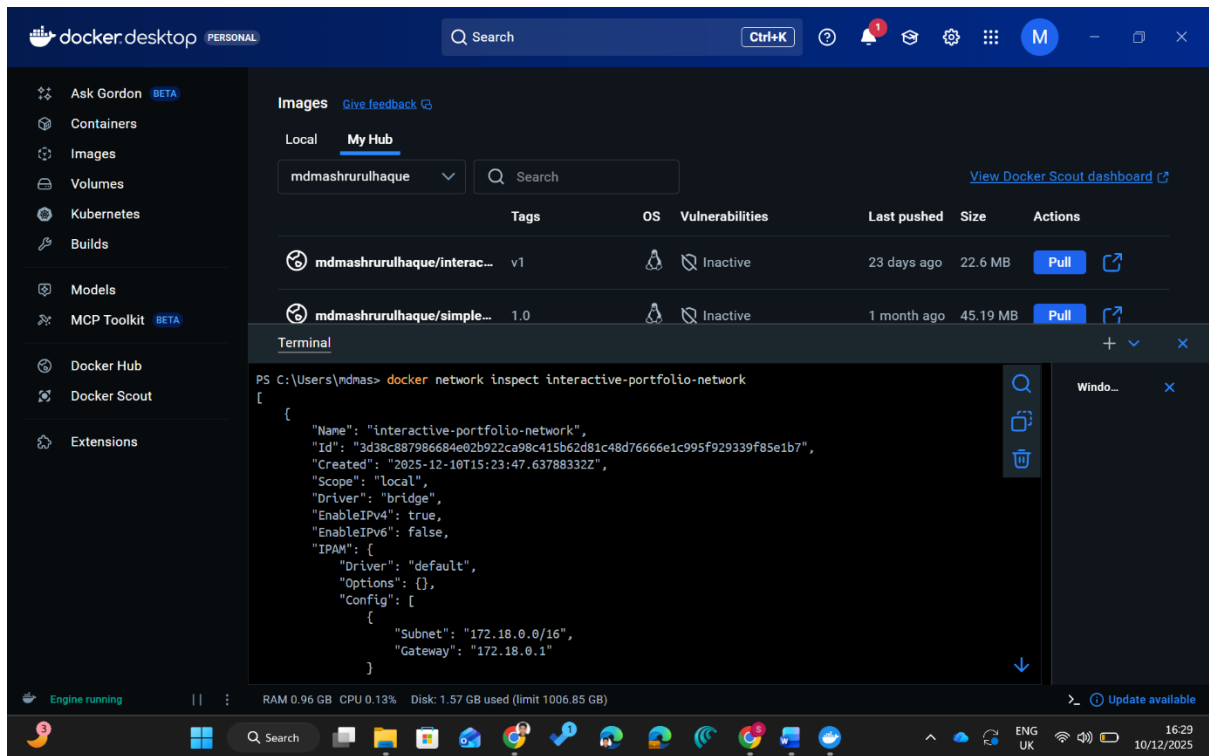


Fig. 12: Inspecting The Network Configuration

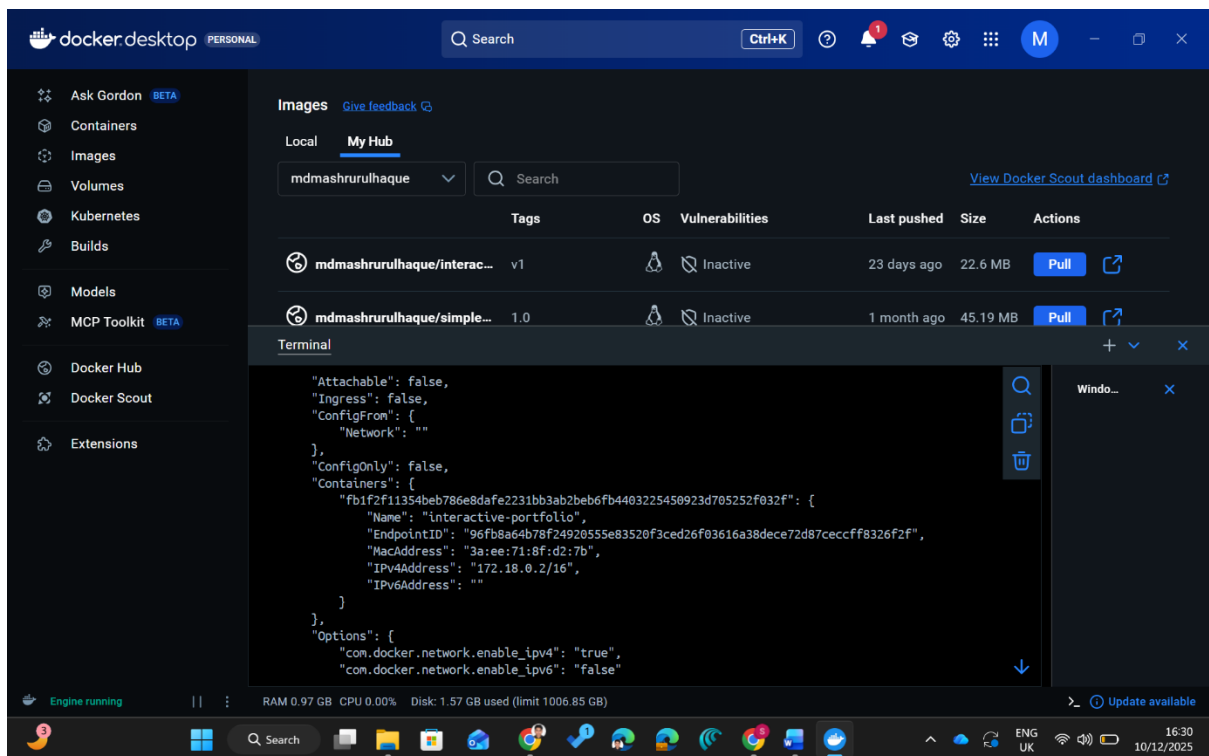


Fig. 13: Verifying the container's IPv4 address allocation

Containers come with advantages, but they also bring limitations. Since all containers share the host kernel, vulnerabilities or misconfigurations at that level can affect many at once.

Security research covers risks like container escape attacks. It also addresses malicious or vulnerable images and unsafe privilege setups. Container security stays a key operational issue [17][19]. Stateful workloads pose challenges too. Cross node persistence, backup needs, and limits of volume abstractions in distributed settings add to the trouble [18]. Production use demands orchestration tools, registry management, overlay networking, and monitoring. All that increases architectural complexity [13][15]. Modern cloud ecosystems tackle this by integrating managed container services. These include Kubernetes, scalable storage, network policy engines, and automated image scanning. They lighten the operational load. This setup leads naturally to cloud computing discussions. There, managed platforms handle constraints via elastic provisioning, distributed storage, and cloud native orchestration [13][15].

There is a practical end-to-end demonstration of the Docker container lifecycle using a simple web server project. The video walks through each stage, from building the Docker image, container, volume to network and publishing it to Docker Hub.

YouTube Video Link: <https://youtu.be/PV4jK6JvEKA>

4. Cloud Migration and Architectural Concepts in the Context of Khan Academy

Cloud computing stands as a key element in how modern digital services work. It lets organizations run applications on a global level. They can keep things agile, resilient, and cost effective at the same time. The main features include on demand resource provisioning. There is broad network access. Resources get pooled together. Elasticity happens quickly. Service gets measured carefully. These traits show up a lot in the literature. They change things for industries that need high availability on distributed platforms [27]. Platforms like Khan Academy rely on cloud setups. They handle changing global demand. User behaviours vary widely. The learning experience stays responsive across different areas. Educational use often spikes during school terms or big events. So, adapting compute power on the demand keeps performance steady [31]. This report looks at how Microsoft Azure handles three main cloud ideas. Those are load balancing, scalability, and application gateways. It then talks about how they work together. There is value for users in combining them. Key reasons push cloud use forward. Serverless computing comes up next, compared to containerization. All along, the focus stays on what this means for Khan Academy. The goal is reliable access to learning materials on a large scale.

Load balancing, scalability, and application gateways build the base for handling workloads well in the cloud. They do it securely too. Load balancing helps in distributed systems. It stops servers from getting too busy. Response times get better. Availability improves as traffic spreads over many nodes [24]. Old systems had limits from hardware or fixed setups. But cloud places like Microsoft Azure use smart automated ways. They check backend health all the time. Traffic goes only to working instances. This cuts down on single failure points. It fits high

availability designs. Buyya and others talk about this in cloud principles. They stress resilience and handling faults as essential qualities of cloud ecosystems [25].

Scalability, especially elasticity, is another defining feature of cloud computing. Mell and Grance call rapid elasticity a main trait [27]. It lets organisations dynamically scale resources as workloads change. Azure autoscaling uses things like CPU use, memory, or queue lengths for HTTP. This shows the automation based on measures. It appears in both business and academic talks. For platforms like Khan Academy, this matters a lot. Demand shifts variable and spreads worldwide. Peaks hit without warning. Manual setup just does not work. Reese [26] points out elasticity turns infrastructure into ongoing costs where operational expense aligned closely with actual usage. This boosts reasons for cloud, both money wise and tech wise [26].

Application gateways tackle issues at the app level. They handle traffic securely and smartly. They work at Layer 7 in the OSI model. They check HTTP/S requests. Performing advanced routing and security like Web Application Firewall (WAF) rules applies. Azure docs say gateways do SSL end, URL routing, and filtering intrusions [22]. These help web apps spread over microservices. These mechanisms support the shift towards cloud ecosystems described by Ali Babar and Chauhan [20], where complex application interactions must be managed through cloud-native intermediaries in a scalable and secure manner.

Together, these three concepts form the core architectural strategies enabling Khan Academy to operate at global scale. Load balancing spreads learner requests over servers. Scalability adjusts services to demand automatically. Application gateways handle and protect traffic at the app level.

5. The Complementary Relationship Between Load Balancing and Scaling

Load balancing and scaling work together well for real workloads. Load balancing stops single servers from overload. But it cannot compensate for insufficient overall capacity. A group of servers, even with good distribution, fills up if demand escalates too much. Brewer says distributed systems focus on availability and handling partition under high load [23]. It gets hard to keep maintaining balance without growing resources. So, scaling steps in.

Scaling by itself does not spread work best. New servers might sit empty without load balancing. That wastes resources. Zhang and team say optimization needs capacity plus equal task split [24]. It responds to failures too. In cloud native setups, they integrate best. Scaling adds power. Load balancing uses it efficiently.

Khan Academy experienced rapid user growth in COVID 19 pandemic [31]. The relationship between these concepts shown clear there. Load balancing gives each compute instance a fair share of requests. Autoscaling makes sure enough instances exist to handle peak global demand. This combination keeps user experience steady across regions. Even if traffic can surge, but performance remain steady.

In the end, load balancing and scaling act as paired supports for elasticity and reliability as they build a full operations setup. No single mechanism could not do it alone.

6. Customer Value in Integrated Cloud Architecture

Putting load balancing, scalability, and application gateways together gives real benefits to users. It covers performance, reliability, and security and these three pillars consistently emphasised often in cloud literature. Performance improves because traffic is intelligently routed and backend workloads are distributed evenly. Zhang and others say load balancing reduces response times. It increases throughput by preventing resource saturation [24]. Autoscaling adds additional compute instances during peak hours. It stops performance slowdowns that once characterised pre-cloud architectures [26].

From a reliability perspective, customers benefit from system resilience. Application gateways filter malicious requests. They end SSL sessions and set security rules from industry level practices [22]. This builds trust and protect user data. It aligns with security priorities described in Buyya's work [25]. Load balancers keep flow going by dropping unhealthy nodes which make sure the chances of less disruptions. Such high availability is essential for educational platforms, because learners expect no interruption regardless of their location.

The integration of these technologies also makes things transparent and seamless for users. Mell and Grance note one of the defining characteristics of cloud computing is the abstraction of complexity from end users [27]. Even during surges or partial fails users experience stable and responsive. This boosts engagement. It helps those using online learning as their primary modality. Studies on microservices show distributed setups improve speed and resilience if done properly [30].

For Khan Academy, these cloud parts hold up the teaching goals. Learners can access materials consistently, safely, and quickly. The setup from integrated tech makes the platform strong for worldwide users because it does not lose reliability or performance.

7. Key Drivers Behind Cloud Adoption: Cost, Flexibility and Strategic Benefits

Cloud expansion can be attributed to several driving forces identified across academic and industrial research. They include cost, flexibility, global spread, automation, and security. Cost is often the foremost influence. Cloud moves spending from buying physical servers to using based pay. Reese [26] observes that this transition reduces financial risk and makes advanced computing environments accessible to organisations that may lack extensive infrastructure budgets. For nonprofit organisation like Khan Academy, this keeps scalability and sustainability.

Flexibility is also another critical driver. Cloud platforms support rapid deployment of new features, modular microservices, and scalable storage which allow organisations to innovate at a pace not possible with traditional infrastructure [25]. Mell and Grance stress elasticity and self-service [27]. Resources match the needs. Waste reduces. Agile development also works. It enables organisations to test, change setups, meet user shifts without hardware procurement delay.

Global distribution further pushes cloud adoption. Cloud Providers have data centers in different regions which reduces latency for dispersed users. Buyya et al. highlights distributed resource pooling offers unprecedented opportunities for high-scale service delivery [25]. Khan Academy gets steady performance for learners everywhere.

Security and automation also play key role. Clouds services add identity management, monitoring tools, auto compliance which would be hard to build independently and costly. Application Gateways and WAFs adapt to threats by security layers [22]. This aligns with Ali Babar and Chauhan's [20] observation that contemporary cloud ecosystems provide increasingly sophisticated capabilities for governance and security, supporting complex application environments with minimal operational overhead.

These factors show cloud's spread across industries. The setup is cost-effective, flexible, global, automate, and secures that fits modern organisational needs and innovative digital services.

8. Serverless Computing and Its Contrast with Containerisation

Serverless computing has rapidly emerged as a key cloud model offering event-driven execution, automatic scaling, and fully managed infrastructure. According to AWS Lambda and Azure Functions documentation [28][29], serverless functions execute only when triggered, incur no cost while idle, and scale instantly at the level of individual events. This aligns strongly with cloud computing's core characteristic of measured service, enabling granular billing and operational efficiency [27]. Ali Babar and Chauhan [20] further observe that cloud ecosystems increasingly adopt composable service models in which workloads are decomposed into modular components. It's a pattern strongly aligned with serverless design.

For Khan Academy, serverless computing offers advantages in analytics processing, background tasks, and intermittent workloads that do not require continuous server availability. These include generating learning insights, sending notifications, and automating periodic content updates. With no need to manage servers, the development team can focus on educational functionality rather than system administration that efficiency aligned closely with the cost and flexibility drivers described earlier.

Containerisation represents another major architecture, enabling applications to run in isolated, portable environments. Merkel [12] describes containers as lightweight units that encapsulate dependencies and runtime configurations, allowing consistent deployment across environments. Pahl [13] notes that containers are central to Platform-as-a-Service (PaaS) models, supporting microservice architectures through enhanced portability and control. For continuous workloads or services requiring custom configurations, containerisation provides a degree of environmental consistency and operational control that serverless models may lack.

When compared through the driving forces of cloud adoption, both serverless computing and containerisation address cost and flexibility but in different ways. Serverless excels in cost efficiency for intermittent workloads, following the pay-per-use model emphasised by Reese [26]. On the other hand, containers may provide better long-term cost efficiency for sustained workloads. In terms of flexibility, containers offer greater control over runtime environments, supporting complex or specialised workloads, whereas serverless functions rely on provider-specific execution frameworks [29]. Scalability is automated in both models. However, serverless scaling occurs at a finer granularity, often triggered at the event level [28].

Villamizar et al. [30] demonstrate that microservice deployments using containers can outperform monolithic architectures, particularly in dynamic cloud environments. However, serverless functions may be more suitable for situations requiring extreme elasticity, low operational overhead, or fine-grained billing. Ultimately, both approaches align with the broader cloud ecosystem's shift toward modular, distributed architectures. Organisations such as Khan Academy benefit most from hybrid deployment models that combine serverless and containerisation according to workload characteristics.

9. Conclusion

This report looked at the key technologies and design ideas behind modern cloud computing, focusing on virtualization, containerization, and cloud-native services.

Virtualization was shown to be great for strong isolation, security, and making better use of physical servers. By using hypervisors, multiple virtual machines can run safely on the same hardware. The downside is that virtual machines can be heavier to run and manage. They often introduce extra overhead, can suffer from resource competition, and may be less suitable for workloads that need to scale up and down very quickly.

Because of these limits, containerization has become increasingly popular. Containers share the host machine's operating system kernel, which makes them much lighter than virtual machines. This leads to performance that's close to running directly on the hardware, faster startup times, and the ability to run more workloads on the same server. Tools like Docker have therefore become central to scalable cloud platforms, continuous deployment, and microservices-based application design.

The report also explored cloud migration and architecture using Khan Academy as a case study, showing how cloud features work together to deliver reliable global services. Load balancing helps spread traffic evenly across servers, which improves performance and prevents failures if one server goes down. Autoscaling adjusts compute capacity automatically based on demand, so the system can handle sudden spikes in users. Application gateways add another layer of protection and control by managing web traffic at a higher level (Layer 7), inspecting requests, routing them intelligently, and enforcing security policies. Together, these components help keep services stable during major traffic surges—like the increases seen during the COVID-19 pandemic [31] while keeping the complexity hidden from end users.

A key point from the analysis is that load balancing and scaling are most effective when used together. Load balancing alone can't help if there simply aren't enough resources available, and scaling alone can't help if traffic isn't being distributed properly. Real elasticity requires both: smart traffic distribution and enough capacity to meet demand. For users, especially learners spread across different regions, this results in practical benefits like lower delay, better uptime, and stronger security.

Finally, the study compared container-based systems with serverless computing. Serverless platforms can scale very precisely and can be cost-effective for event-driven or occasional workloads, since you only pay when code runs. Containers, on the other hand, offer more control and predictability, which can be important for complex applications or long-running services. Rather than being competing approaches, the report shows they work best as complementary options within hybrid cloud systems.

Overall, the findings suggest that the most effective cloud systems combine virtualization, containers, managed cloud services, and serverless computing in a way that matches the needs of each workload and the goals of the organization. This careful mix is what enables modern platforms to be efficient, scalable, and valuable to users.

10. References

- [1] R. Buyya, C. S. Yeo, and S. Venugopal, “Market-oriented cloud computing: Vision, hype, and reality for delivering IT services as computing utilities,” in Proc. 10th IEEE Int. Conf. High Performance Computing and Communications (HPCC), Dalian, China, 2008.
- [2] M. Armbrust et al., “A view of cloud computing,” *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.
- [3] P. Barham et al., “Xen and the art of virtualization,” in Proc. 19th ACM Symp. Operating Systems Principles (SOSP), Bolton Landing, NY, USA, 2003.
- [4] T. Garfinkel and M. Rosenblum, “A virtual machine introspection-based architecture for intrusion detection,” in Proc. Network and Distributed System Security Symp. (NDSS), San Diego, CA, USA, 2003.
- [5] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, “An updated performance comparison of virtual machines and Linux containers,” in Proc. IEEE Int. Symp. Performance Analysis of Systems and Software (ISPASS), Philadelphia, PA, USA, 2015.
- [6] R. P. Goldberg, “Survey of virtual machine research,” *IEEE Computer*, vol. 7, no. 6, pp. 34–45, Jun. 1974.
- [7] F. Beck, R. Farahbod, and G. Bell, “Performance interference effects in virtual environments,” in Proc. Embedded Real-Time Software and Systems (ERTS), Toulouse, France, 2005.
- [8] J. Rist, “Scheduler-based defences against cross-VM side-channels,” Tech. Rep., Univ. of Wisconsin–Madison, Madison, WI, USA, Aug. 2014.
- [9] G. J. Popek and R. P. Goldberg, “Formal requirements for virtualizable third-generation architectures,” *Commun. ACM*, vol. 17, no. 7, pp. 412–421, Jul. 1974.
- [10] A. Kivity et al., “KVM: The Linux virtual machine monitor,” in Proc. Ottawa Linux Symp., Ottawa, ON, Canada, 2007.
- [11] J. Sugerman, G. Venkitachalam, and B.-H. Lim, “Virtualizing I/O devices on VMware Workstation’s hosted virtual machine monitor,” in Proc. USENIX Annu. Tech. Conf. (ATC), Boston, MA, USA, 2001.
- [12] D. Merkel, “Docker: Lightweight Linux containers for consistent development and deployment,” *Linux J.*, no. 239, Mar. 2014.
- [13] C. Pahl, “Containerisation and the PaaS cloud,” *IEEE Cloud Comput.*, vol. 2, no. 3, pp. 24–31, May–Jun. 2015.
- [14] R. Morabito, “A performance comparison of Linux containers and virtual machines,” *J. Netw. Comput. Appl.*, vol. 56, pp. 1–12, Nov. 2015.
- [15] D. Bernstein, “Containers and cloud: From LXC to Docker to Kubernetes,” *IEEE Cloud Comput.*, vol. 1, no. 3, pp. 81–84, Sep. 2014.

- [16] V. Tarasov et al., “Large-scale analysis of Docker images and their reuse,” in Proc. ACM/IFIP Middleware Conf., 2020.
- [17] S. Sultan, I. Ahmad, and T. Dimitriou, “Container security: Issues, challenges, and the road ahead,” IEEE Access, vol. 7, pp. 52976–52996, 2019.
- [18] Z. Bakhshi, G. Rodriguez-Navas, and H. Hansson, “Analyzing the performance of persistent storage for fault-tolerant container-based architectures,” J. Netw. Comput. Appl., vol. 216, 2023.
- [19] A. Y. Wong et al., “Threat modeling and security analysis of containers: A survey,” arXiv preprint arXiv:2111.11475, 2021.
- [20] M. A. Babar and M. Chauhan, “A tale of migration to cloud computing for sharing experiences and observations,” in Proc. 10th Int. Conf. Software Engineering and Applications, 2011.
- [21] Amazon Web Services, “Application Load Balancer,” Apr. 25, 2025.
- [22] Microsoft Azure, “Azure application gateway documentation,” Apr. 25, 2025.
- [23] E. Brewer, “CAP twelve years later: How the ‘rules’ have changed,” Computer, vol. 45, no. 2, pp. 23–29, Feb. 2012.
- [24] R. Morabito, “A performance evaluation of Docker containers and virtual machines on lightweight edge devices,” J. Netw. Comput. Appl., vol. 136, pp. 62–71, 2019.
- [25] R. Buyya, J. Broberg, and A. Goscinski, Cloud Computing: Principles and Paradigms. Hoboken, NJ, USA: Wiley, 2010.
- [26] G. Reese, Cloud Application Architectures: Building Applications and Infrastructure in the Cloud. Sebastopol, CA, USA: O’Reilly Media, 2009.
- [27] P. Mell and T. Grance, “The NIST definition of cloud computing,” NIST Special Publication 800-145, Gaithersburg, MD, USA, 2011.
- [28] Amazon Web Services, “AWS Lambda developer guide,” Apr. 25, 2025.
- [29] Microsoft Azure, “Azure Functions documentation,” Apr. 25, 2025.
- [30] M. Villamizar et al., “Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud,” in Proc. 10th Computing Colombian Conf. (CCC), 2015.
- [31] M. Kosarchyn, “How Khan Academy successfully handled 2.5x traffic in a week,” Khan Academy Engineering Blog, Apr. 2020.