University of Wales Trinity Saint David, London

Cloud Engineering Principles and Practice

ACCA7027

Title: Assignment on Cloud computing, Docker and containerization.

Submitted by : Shafee Ul Mahmud Chowdhury

Student ID : 2436374

Programme :M.Sc in Cybersecurity and digital forensic

Submitted to : Renuga Jayakumar

Submission Date : 18th December 2025

Table of Contents

1.Introduction....3
2. Principle of Virtualization....3
2.1 Types of Virtualization....4
2.2 limitations of virtualization....4
3. Principle of Containerization....4
3.1 Benefits of containerization....5
3.2 Example: Demonstrate the development of a Docker file to convert an image....5
Figure-1: Docker file....5
Figure2: Docker file in txt format....6
Figure-03: Docker file and application....6
Figure-4: Change directory....7
Figure5: Current working directory....7
Figure6: Building images....8
Figure7: Image creating processes....8
Figure8: Image print....9
4. Images and published image....9
Figure9: Publishing image....9
5. Container Volume....10
Figure10: Volume create....10
Figure 12: Docker pull....11
Figure 13: Container creation and volume attachment....11
Figure14: Container verification....12
Figure-15: Volume verification....12
6. Container network....12
Figure16: default network settings....13
Figure17: Network creation....13
Figure-18: List of network....14
Figure19: Connect to network....14
Figure20: Network Verification....15
Figure21: Web-application....15
7. What is load balance and how it works in cloud computing....16

7.1 Impact of load balance in cloud computing 16
7.2 Scalability 16
7.2.1 Types scalibility 16
7.3 Application Gateway 17
7.4 Features of application gateway 17
8. Application of Load balance, Scalability and application gateway in real-life 17
8.1 How load balance works at Netflix. 17
Figure 22: Netflix Architechture [34] 18
8.2 Scalability mechanisms in netflix 18
8.3 Application gateway(Zuul) in Netflix architecture 19
8.4 How all three ofthese component work together 19
8.5 Resiliency and fault tolerance 19
8.6 Performance and user experience 20
8.7 Operational benefits and continuous delivery 20
8.8 Security considerations 20
9.Serverless function in cloud computing 20
9.1 Benefits of serverless function 20
9.2 Containerization benefits over serverless function 21
9.3 Comparison between server less function and containerization based on cost and benfits. 21
9.4 Key Observation 21
10. Conclusion 22
11. Video link of image demonstration 22
12.References 22

1.Introduction

The winds of cloud computing, virtualization and containers are changing modern IT infrastructure and the way software is deployed. With virtualization, It is possible to run several operating systems on one physical system by decoupling the hardware with a hypervisor. [1]The approach enables full hardware utilization, fault isolation and resource allocation. For instance, Consolidation of shared hardware supports operation of several virtualized machines (VMs) on one server simultaneously and minimizes costs as well as improving disaster recovery [1][2].

Containerization takes that idea and runs with it, offering lightweight, transportable environments for applications. Containers package the software and its dependencies into impermeable containers, which share a host operating system kernel and allow for quicker deployment time, more efficient resource utilisation, and consistency in a variety of environments. [7]Technologies like Docker empower developers to build container images with Dockerfile and distribute them to registries such as Docker Hub for easy provisioning [3][4].

It utilizes cloud computing principles around load balancing, scalability, and application gateways to provide high availability and service resiliency of scale. Load balancing disperses incoming traffic among servers, and a scalable application dynamically adds or removes resources in response to user demand. Secure routing, authentication and traffic management are offered at the application gateway. Load balancing goes hand in hand with scalability, and it does so, because scaling directs traffic where and when needed as well as resources [5].

New serverless developments for example around AWS Lambda and Azure Functions simply add the wave of cloud agility executing just-enough code (i.e. function) to be acceptable, without managing anything! Containers give persistent runtime environments to complex applications, whereas serverless computing focus on event-driven, auto-scaling execution that emphasises operation efficiency and cost-effectiveness [6]. In combination these technologies demonstrate the forces of cloud adoption - scaling, resilience and fast deployment.

2. Principle of Virtualization

Virtualization is the key technology in computing era and allows multiple virtual machines running concurrently on a on a physical server. All of those VMs will act as individual computers, complete with a virtual processor (CPU), memory, storage and network interface. This abstraction becomes feasible due to a software layer named hypervisor, that allocates the physical resources requested by each VM in isolated and safe way. This allows companies to consolidate their hardware, reduce operating costs spend and deploy multiple operating systems and applications on a single physical server.[8][9]

Increased efficiency is one of the major reasons why virtual servers are better than dedicated physical servers. In the traditional sense, a server is a self-contained system that is dedicated

to a single workload and does not fully utilize its hardware resources.[8] Virtualization has the effect of consolidating multiple workloads onto a smaller number of machines, reducing their power consumption and cooling needs and – significantly – their data centre footprint. It also improves flexibility because VMs can be easily created, cloned, m oved or restored with snapshots. This results in easier disaster recovery and a higher system availability.[10]

2.1 Types of Virtualization

Virtualization can also be classified as hardware virtualization and host virtualization.

Bare Metal Hypervisor: This hypervisor virtualizes the hardware itself, it runs directly on server hardware without any OSs. Some are VMware ESXi and Microsoft Hyper-V. This one gives you more performance, more isolation from resources and a greater ease to scale.11[]

Host (Type-2) Virtualization – Here, the hypervisor runs over a host OS like VMWare Workstation or VirtualBox. It's easier to install than this, and is good for development, testing, and desktop usage, but carries some overhead.[11]

2.2 limitations of virtualization

Virtualization does come with a few drawbacks when compared to containerization, however. VMs need a complete OS instances, so are heavier, slower to start and more resource consuming.[10] Containerization, on the other hand, shares a host OS kernel that results in faster to roll out, great scalability and density. Containers are also better integrated into DevOps cycles, and VMs have higher management over-headshaled compared to VMs[11]

3. Principle of Containerization

Containerization is a modern way to deploy applications where an application along with all the libraries, dependencies, and other configuration files are combined into a single executable package called as a container. Unlike VMs, containers need not be loaded with a full OS instead they share the same OS kernel so there is much less overhead.[7] Because of this architectural efficiency, containers can launch in seconds, they consume far less computational resources and scale almost instantly in response to demand on dynamic cloud platforms.

Containers are self-sufficient: they include everything an application needs to run without depending on the system it’s executed by so that they can be run consistently across different environments—from a developer’s laptop, or on-premise server, to cloud platforms. With tools like Docker, Podman, and orchestration platforms like Kubernetes it’s easy to build-deploy-manage-scale containerized applications in a uniform way.[13]

3.1 Benefits of containerization

The Advantages of Containerization Containers have many clear advantages over the classic VMs. [8] First, containers provide much quicker startup times that make deployment fast and CI/CD even faster. Second, due to low-resource usage density it is possible to run more instances of applications on the same hardware than with virtual machines that need dedicated OS instances.[14] Third, containers are used in micro services architecture where you have a big application and you break it down into small independent deployable services. This makes fault isolation, development speed and flexibility better. Container images also allow for immutable infrastructure, which reduces configuration drift and makes versioning easier.[15]

3.2 Example: Demonstrate the development of a Docker file to convert an image

To prepare an image for an application, there is a prerequisite that is instruction includes in a file and named it as a Docker file. Moreover, application and Docker file should be in same directory or folder. [16]

In example a web application considers as an application. Therefore, below steps will be followed for preparing webserver image.

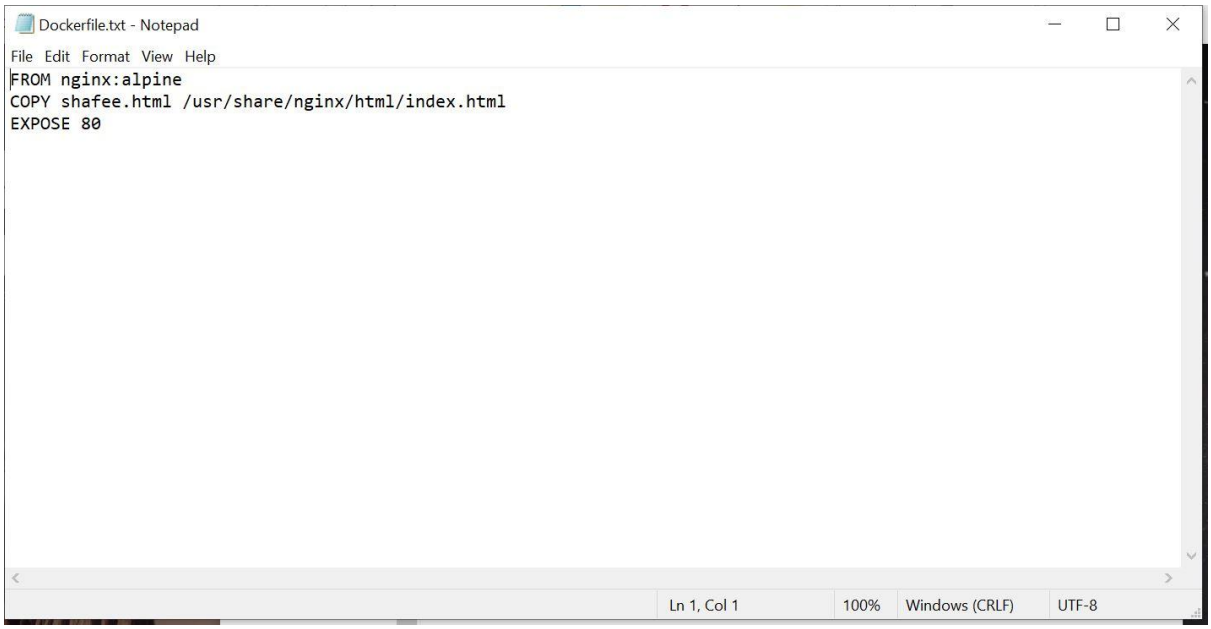

Figure-1: Docker file

Above figure shown the how to prepare an image what will be the base platform.

Line-1: **FROM nginx:alpine**

Where from is syntax for collecting the base platform of webserver of docker image.

Nginx is the base platform for docker webserver image which is called by FROM command

And alpine is the linux tag.

Line-2: **COPY shafee.html /usr/share/nginx/html/index.html** [16]

Copy for copying the shafee.html web application to linux server /usr/share/nginx/html/index.html

shafee.html is the application name which is should be exact as stored in folder.

/usr/share/nginx/html/index.html this is the nginx default root directory and destination of where shafee.html will be kept as named as index.html

After that this txt file will be saved as **Dockerfile** without extension.[16]

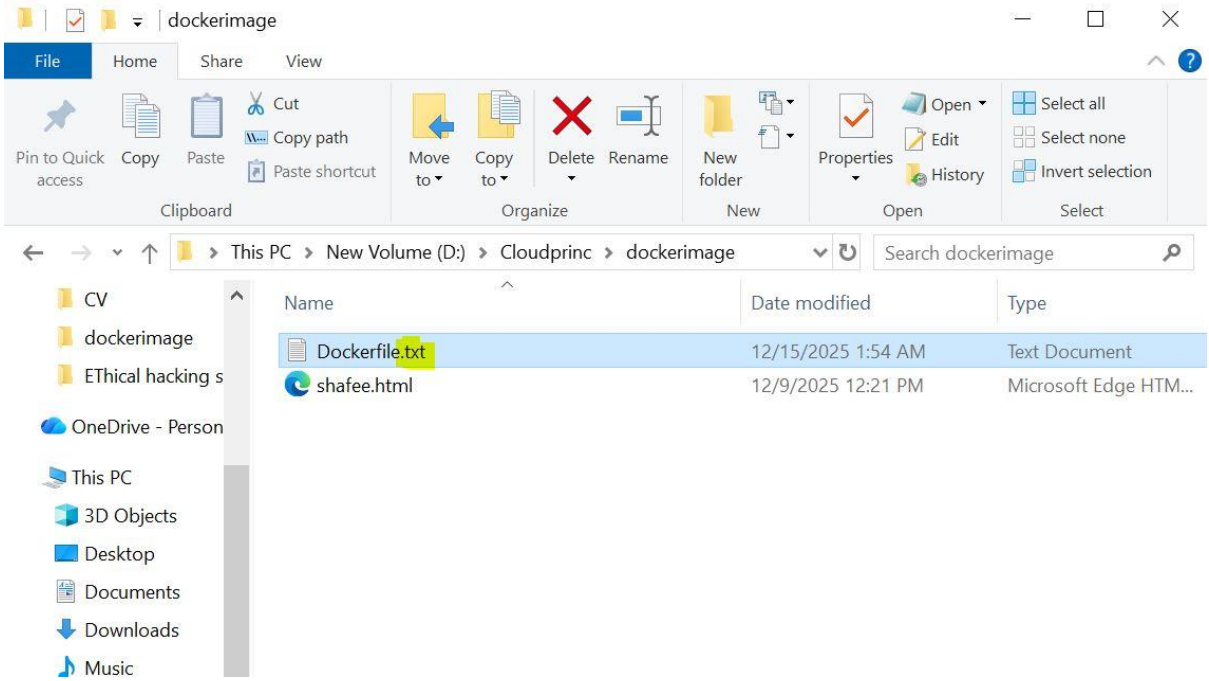

Figure2: Docker file in txt format

After putting all the information file saved as a txt file. [16]

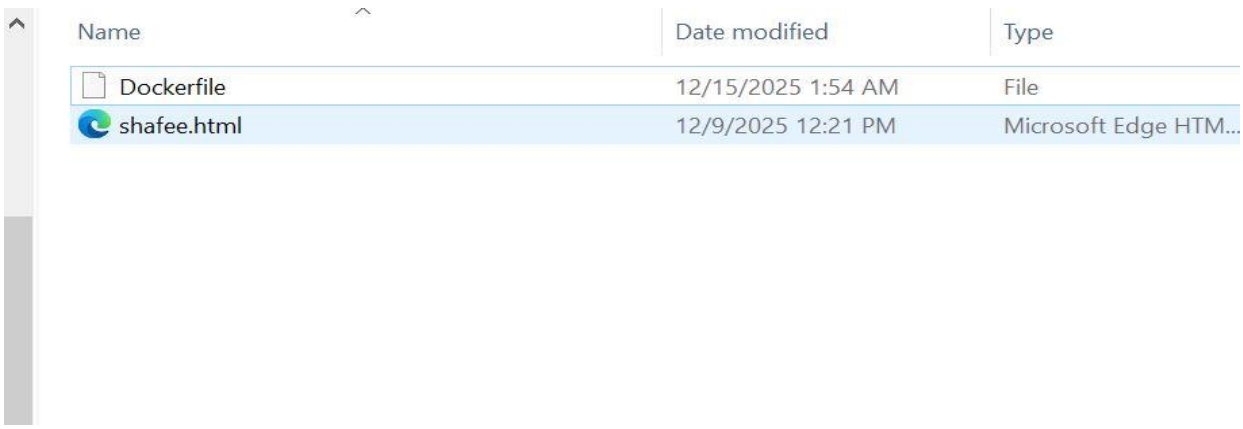

Figure-03: Docker file and application.

Figure-03 depicts after removing the .txt extension Dockerfile should be look like the above figure. However, application and Dockerfile must be in the same directory. Otherwise docker build command cannot create image.[16]

Now Docker file and application is ready to make image with build command. Before run the build command make sure Docker desktop current working directory is same as the files directory. If not, then changed the directory.

Figure-4: Change directory.

Figure-4 demonstrated changing directory. Here cd = change directory D:\cloudprinc\dockerimage. This will change directory to the path where all your Docker relevant files are. And of course, you need to be in the correct directory for Docker commands to run based on the right project.[16]

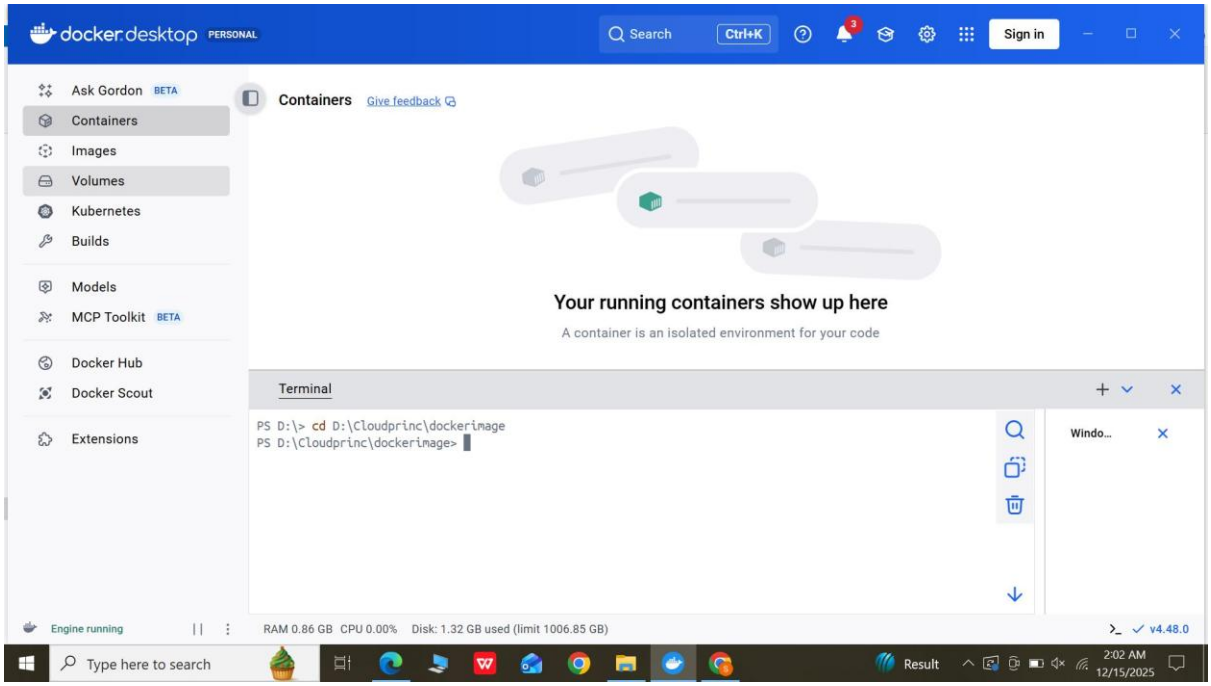

Figure5: Current working directory

Figure5 depicts the change directory successfully. Now build command will successfully execute.[16]

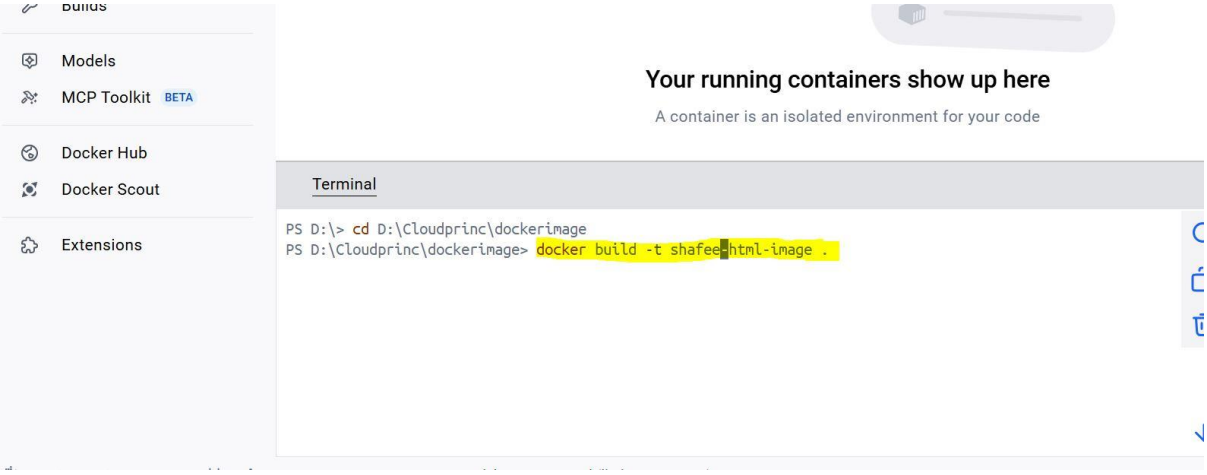

Figure6: Building images

Figure6 contain the **build** command for creating images and name will be **shafee-html-image** which is tagged by **–t** .Most importantly end of this command here is a **(.) dot** which means current working directory. [16]

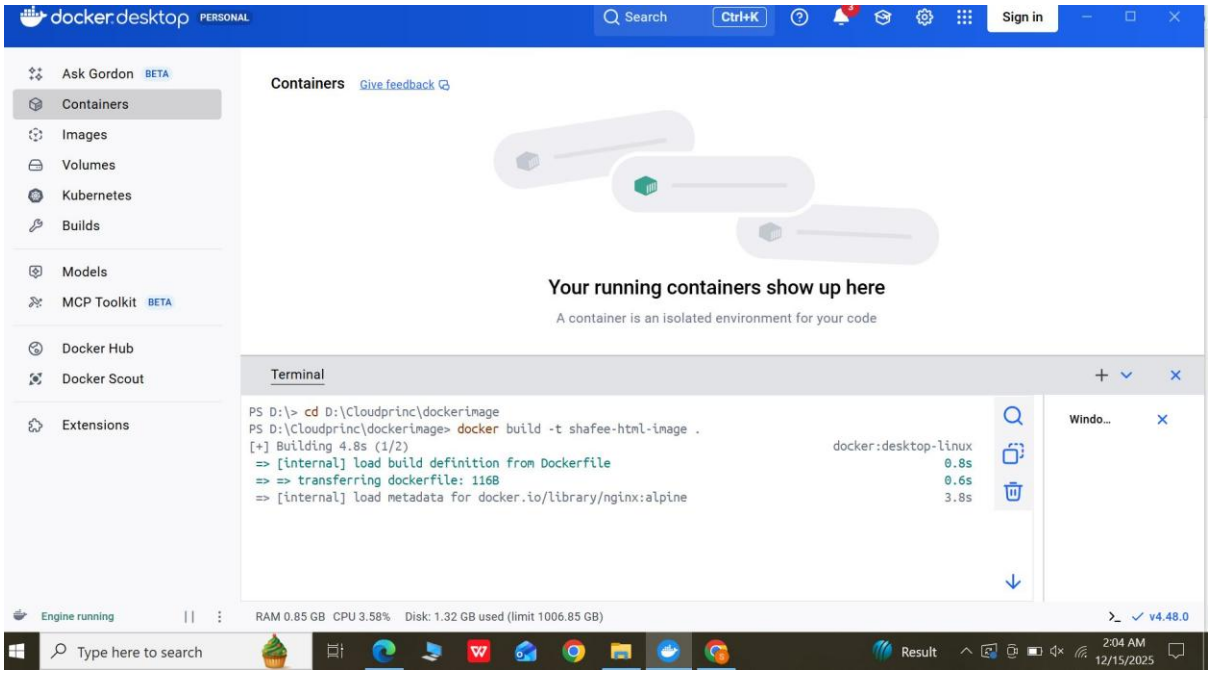

Figure7: Image creating processes.

A verbose mode of message can be seen while creating image at Figure 7. It is these messages which give a rundown of each step taken during building the image, such as pulling down the base image, copying files etc. This sort of output is useful to confirm that things have been done correctly and also can help if there are any errors in the process of building a Docker image, since compiling/compressing/forgetting about different files.

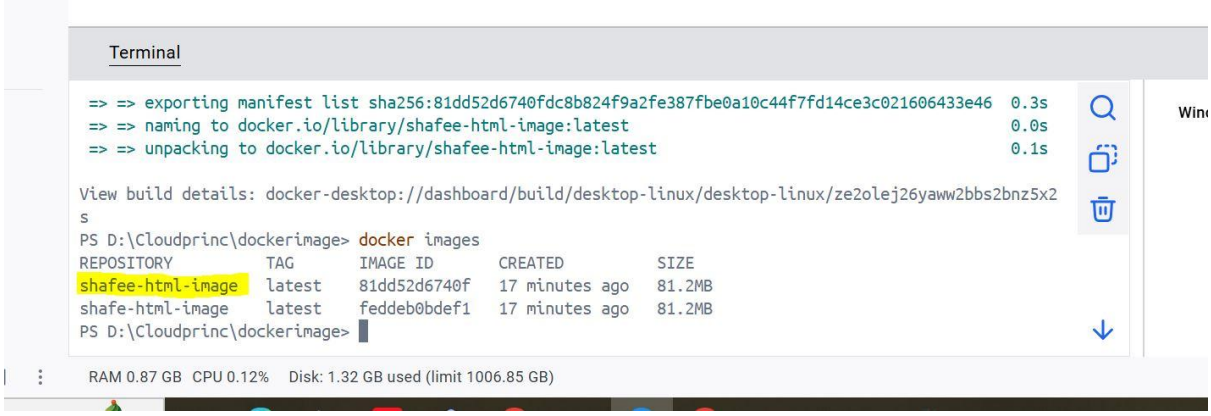

Figure8: Image print

To do so, Figure 8 illustrates the list of Docker images using the docker images command and presents the new image. This output indicates that the image build process was completed without a problem. It also gives information like the name, tag, ID and date of creation and size of the image.

4. Images and published image

In containerization, images serve as the portable and executable definition of an application and its dependencies.[1] They are constructed from a Dockerfile and can be versioned to have consistency and repeatability in the deployment of environments. Once images are built, they can then be pushed to a container registry like Docker Hub or private registries.[17] This allows for ease of sharing, deploying and re-usability of applications across a range of environments (dev, test and prod), with the consistent behavior and configuration.[17]

Figure9: Publishing image

Above figure9 shown how to push images to docker hub. Here push is command to upload the images into hub. Moreover, shafee2436374 is the docker hub user-id and after forward slash(/) is the image name which intended to upload. [16]

5. Container Volume

Docker volumes are used to ensure that data survives even if the container is stopped, deleted or rebuilt. [18]They provide a means to share data between containers, increase performance, and improve the backup/restore processes for that data when compared with writing it directly into a container's writable layer. Volumes Provide applications with durable, portable, and maintainable data Store state ful data Dynamic volumes support databases and CMS components.[19]

In this example docker volume is attached with image which already available in docker hub. Therefore, pull the images into docker desktop then run the images to build an container with attaching volume. [16]

Before that volume need to be create.

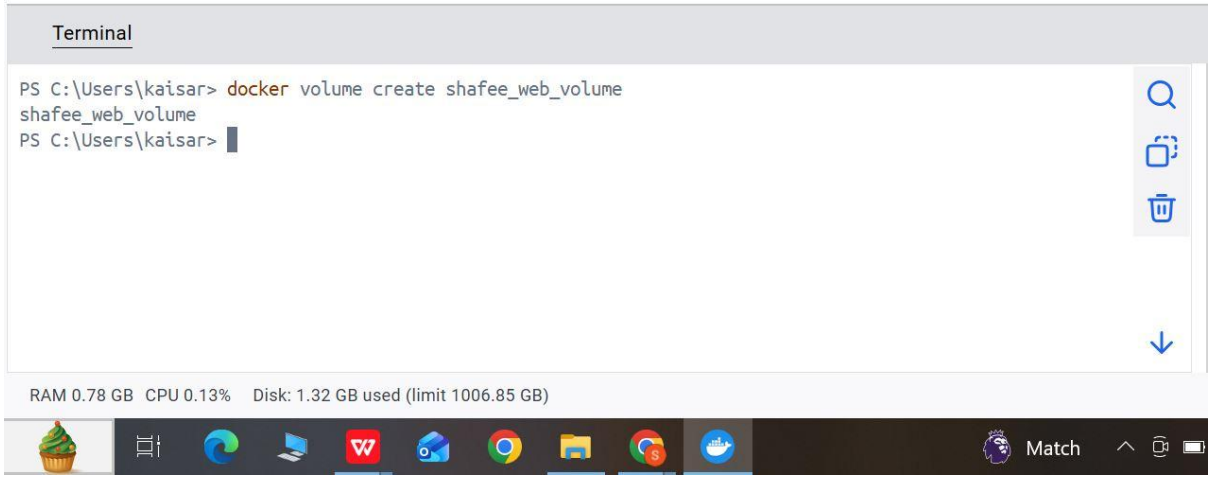

Figure10: Volume create

In figure-10 through docker volume create shafee_web_volume. This command will create one volume named as **shafee_web_volume**.

Most importantly before run an image volume should be attached.This is because it is not technically supported to attach volume in running container. [16]

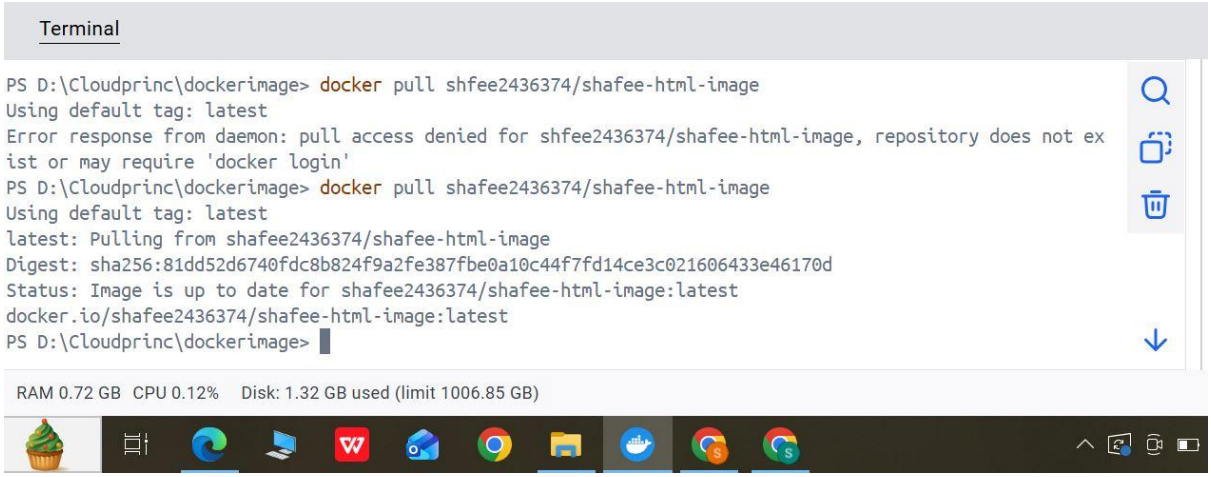

Figure 12: Docker pull

In this figure shows how to pull an image from docker hub.which is required to make an container [16]

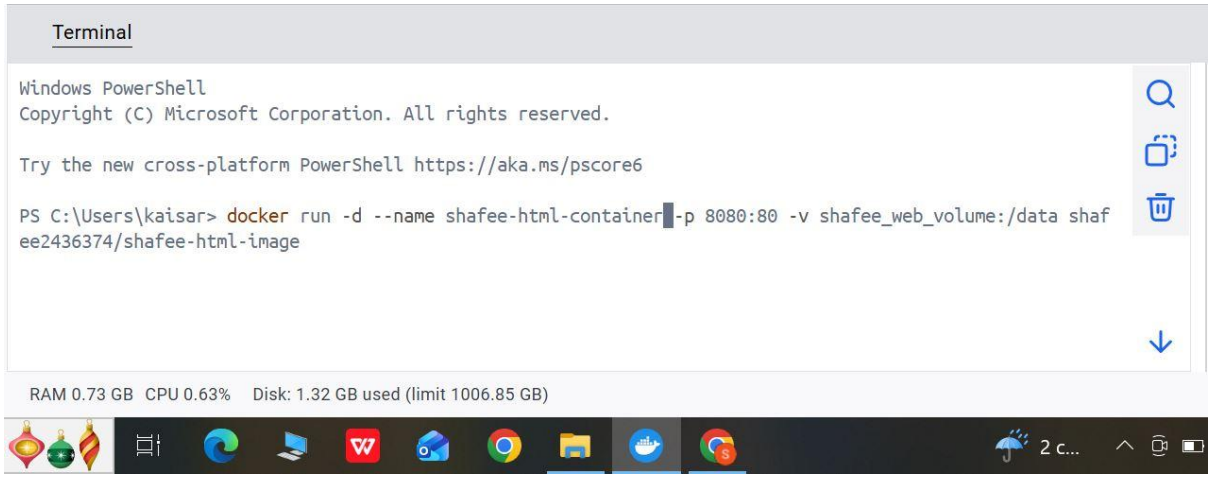

Figure 13: Container creation and volume attachment.

In above figure run the downloaded image where –name for specify the container name –v for attaching the volume with define the volume name **shafee_web_volume:/data** and end of the command mentioned the image name.[16]

Figure14: Container verification

In Figure14 confirms that container is created named as **shafee-html-container** and service listening port is 8080, which is customized web port. [16]

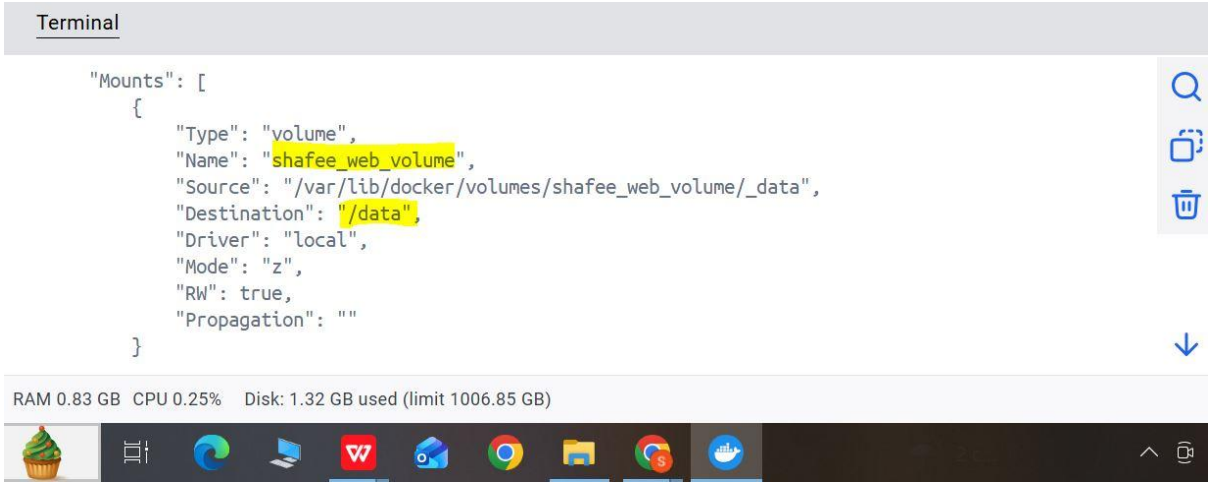

Figure-15: Volume verification

In the above figure it is observed that newly created volume is attached with the container. [16]

6. Container network

Container networking allows containers and services to speak with each other when required. Every container is an island by default which makes networking crucial when it comes to data sharing, connecting micro services and exposing the applications to users. [20]It provides scalability, load balancing and secure communication through network polices. Container networks offer consistent connectivity to enable distributed applications to operate transparently, facilitate service orchestration and en-sure interactions of containerized workloads run smoothly in the cloud or on-premises.[21]

Now the sample container connects with a customized network. Though container is getting ip by default but it is best to use in customized network. Therefore, a network need to be creates then connect with the sample container.

Before that check the default IP details of the sample container.

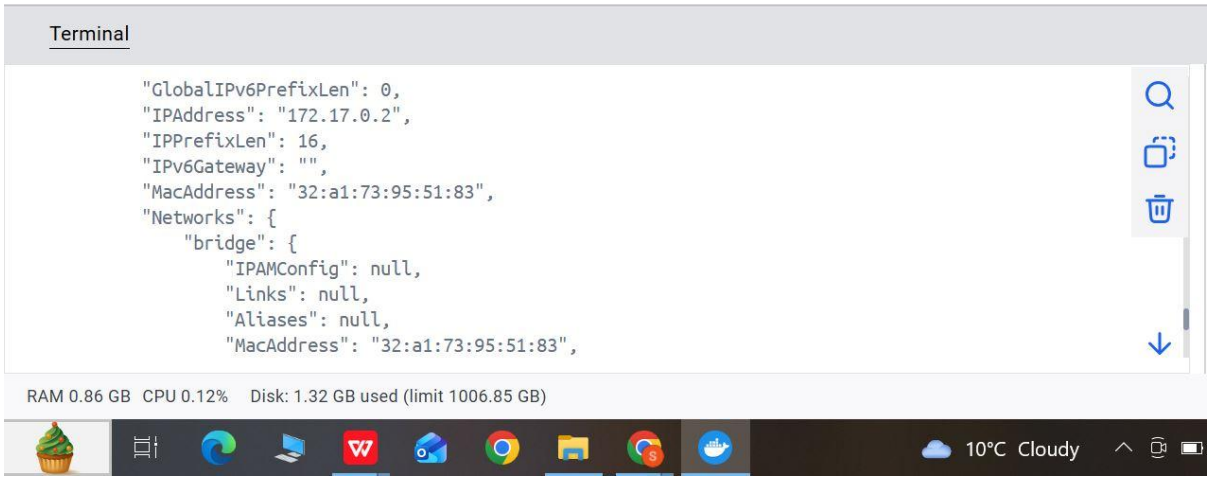

Figure16: default network settings

In this figure default IP of this container is **172.17.0.2 [16]**

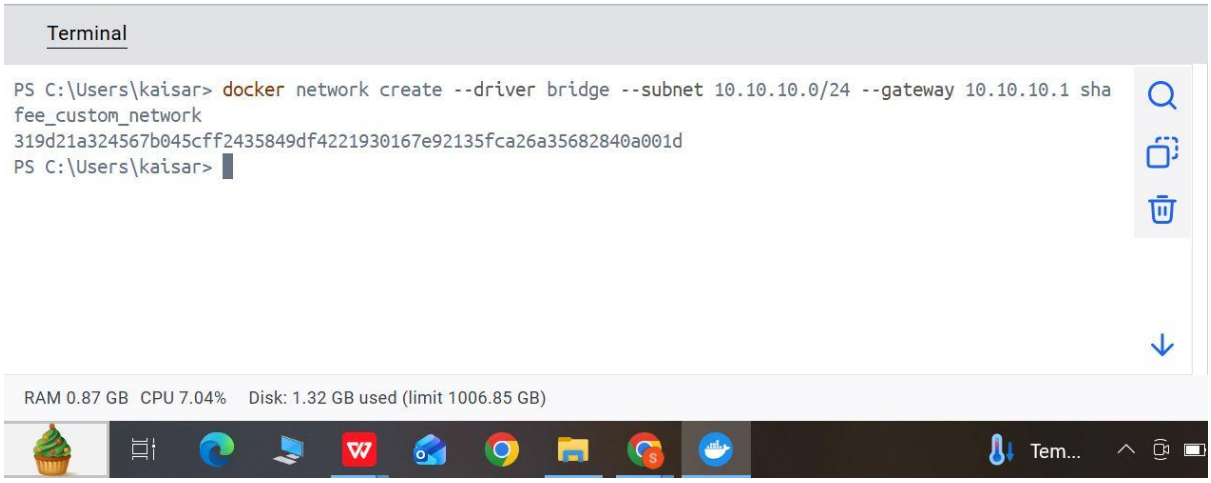

Figure17: Network creation

In above figure a customized network is created with a 10.10.10.0/24 subnet where gateway is 10.10.10.1 and provide a name is shafee_custom_network.[16]

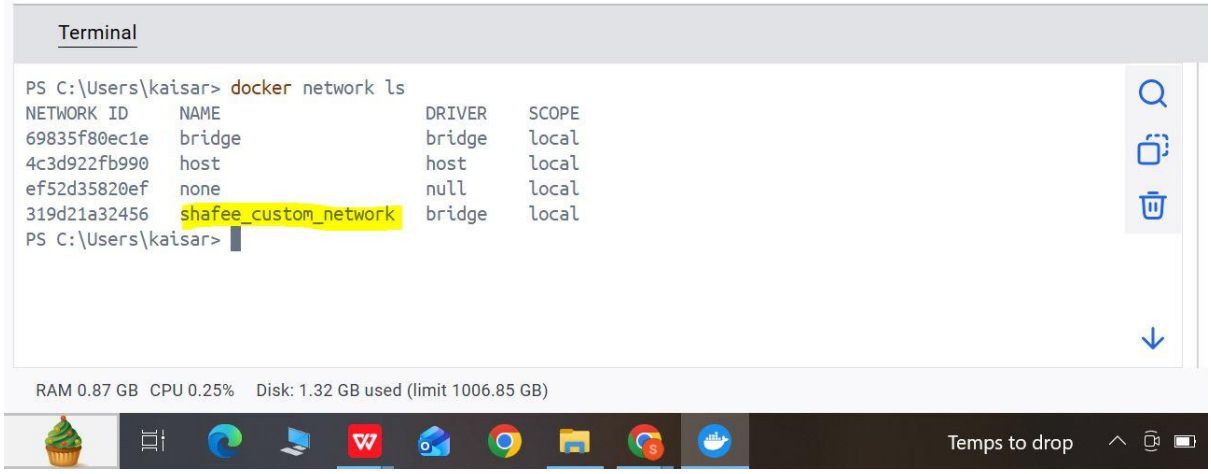

Figure-18: List of network

In this figure **docker network ls** command output shown the list of network and newly created network also present here.[16]

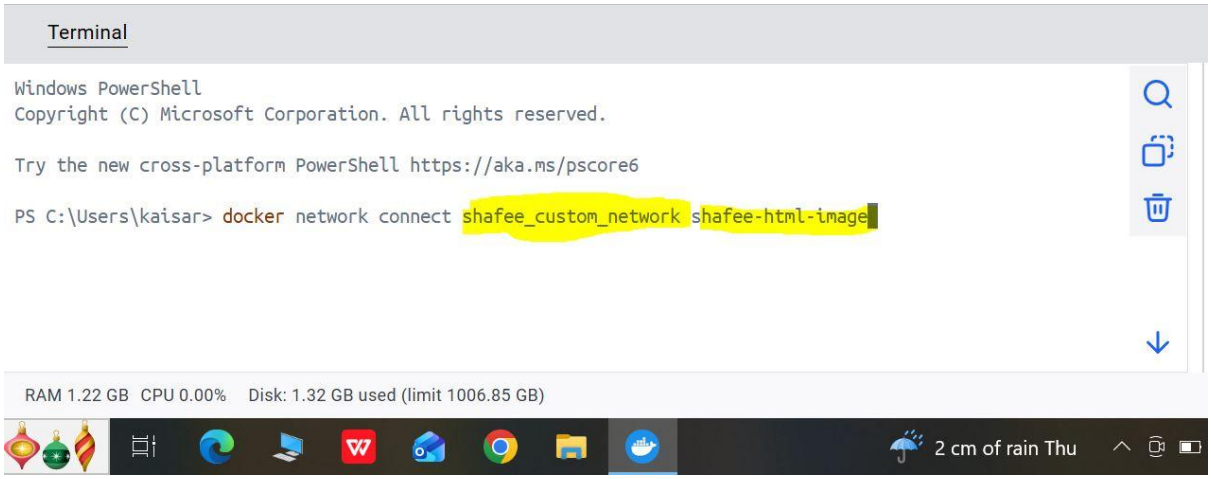

Figure19: Connect to network.

In figure-19 container connect with the newly created network through network connect command. P[16]

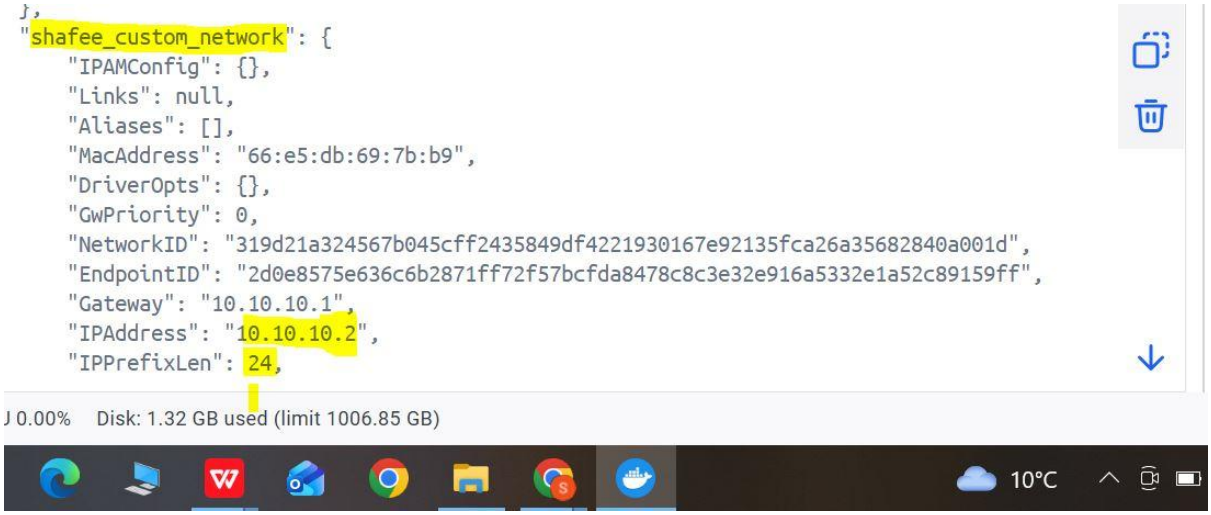

Figure20: Network Verification

In figure-20 it has been seen that container changed the default ip settings to customized network and current IP is 10.10.10.2 which is created earlier.[16]

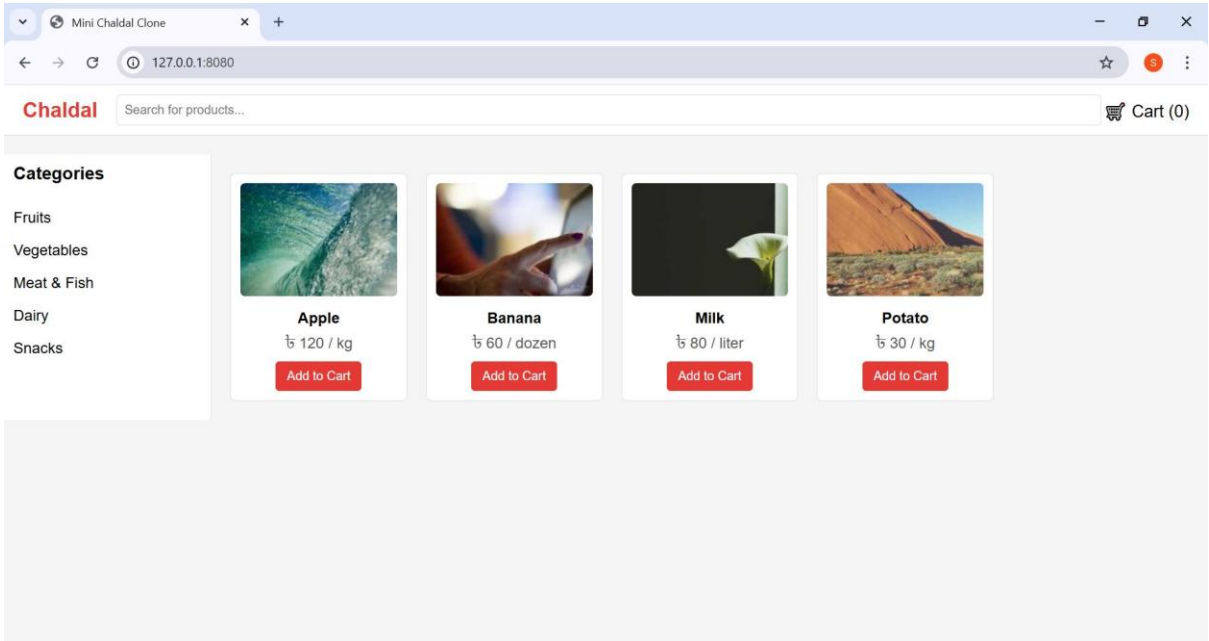

Figure21: Web-application

Finally in figure21 gives an output where sample container is shown the website with loopback IP and webport 8080 which was set while created the container. [16]

7. What is load balance and how it works in cloud computing

Cloud infrastructure load balancing is an important operation that facilitates the equal and uniform distribution of workloads, or computational processing demand for different internet applications or data requests, across servers on a server farm or between networked computer hosts. Load balancing also helps improve the overall system performance and reduces response and processing times, as well as resource utilization while upholding onQoS. [23]A cloud environment has virtualization of resources where the resources can be provisioned or de-provisioned dynamically and hence, load balancing assumes a more significance in order to cope with varying demands. It provides high availability, scalability and load balancing to applications under fluctuated traffic patterns, offering an essential module for cloud infrastructure managing.[22]

7.1 Impact of load balance in cloud computing

Good load balancing can bypass abnormal symptoms of system caused by overload, avoid service lagging and reduce task completing time. Meanwhile, other high-performance load balancing mechanisms can also be used to map tasks or virtual machines onto hosts in cloud data centers that consider the trade-off between system performance and resource utilization of system resources, due to the heterogeneity of cloud infrastructure.[23]

7.2 Scalability

In cloud computing, scalability is defined as the capability of a system to automatically expand or contract its resource pool based on actual workload needs without degrading service quality. It's a fundamental feature of cloud computing that makes it possible to support the rapid growth in users, data volume and processing needs of an application, naturally. Scalable cloud system are capable of allocating and reclaiming resources, such as virtual machine, storage, network bandwidth [24].

7.2.1 Types scalibility

There are two different kinds of scalability: vertical scalability, which refers to the ability to increase the capacity of existing resources (such as CPU, memory, etc.), and horizontal scalability or scale out that is adding or removing more computing instances such as disk [24]. Typically, cloud service providers apply auto-scaling techniques that monitor the performance of the system and automatically redistribute resources in order to provide acceptable response times and quality of service.[25]

Scalability ensures that organisations can pay on a per-use basis for resources whilst maintaining application uptime during periods of high usage. Additionally, it helps ensure business continuity and systems integrity by responding to unexpected spikes in traffic. Thus, the scalability has been widely regarded as a supposed paramount benefit of cloud computing and an essential aspect when designing applications for modern distributed environments [26]

7.3 Application Gateway

An application gateway in cloud computing is a Layer-7 (application-level) traffic management component that acts as a single entry point for client requests to cloud-hosted applications. Unlike traditional network gateways, an application gateway understands application-specific protocols such as HTTP and HTTPS, enabling it to perform intelligent request routing, security enforcement, and traffic inspection [1]. Incoming requests are analysed and then forwarded to appropriate backend services or microservices based on defined routing rules. [27]

7.4 Features of application gateway

Application gateways commonly provide features such as SSL/TLS termination, request filtering, authentication support, and protection against web-based attacks. By centralising these functions, application gateways simplify application architecture and improve security and reliability. In distributed cloud environments, they also support high availability and fault tolerance by integrating with load balancers and auto-scaling mechanisms to route traffic only to healthy instances [28]

In modern cloud-native systems, application gateways are essential for managing traffic in microservices architectures, ensuring consistent performance and secure access across diverse client platforms. As a result, application gateways are widely regarded as a fundamental component in scalable and secure cloud application design [29]

8. Application of Load balance, Scalability and application gateway in real-life

Now describing a real-world application NETFLIX which is using cloud service and implement above mentioned three component of cloud computing.

Netflix runs a massive distributed streaming platform that delivers over 1/3 of all peak-time Internet traffic to over 100M members across the globe [30]. For delivering high-quality content with minimum latency, the company aggregates three architectural features: load balancing, scalability and an application gateway [2]. Together, these technologies collaborate to help of our platform stay performant, secure and reliable at massive scale.[31]

8.1 How load balance works at Netflix.

Load balance technique helps in routing request to various service instances to avoid bottleneck and for make the system highly available [32] In the past Netflix pioneered client-side load balancing by combining the Ribbon library and Eureka for service discovery: clients could "pick" a healthy service instance directly from Eureka [5]. This model does not answer on centralized balancers, allows clients to react on failures (with the use of retries, timeouts and blacklisting 4).

Netflix also load balances at the network edge to regulate ingress global traffic [1]. Using a CDNs approach paired with regional routing and internal load balancing spreads the requests efficiently among several available zones in addition to regions for increased resiliency [35]

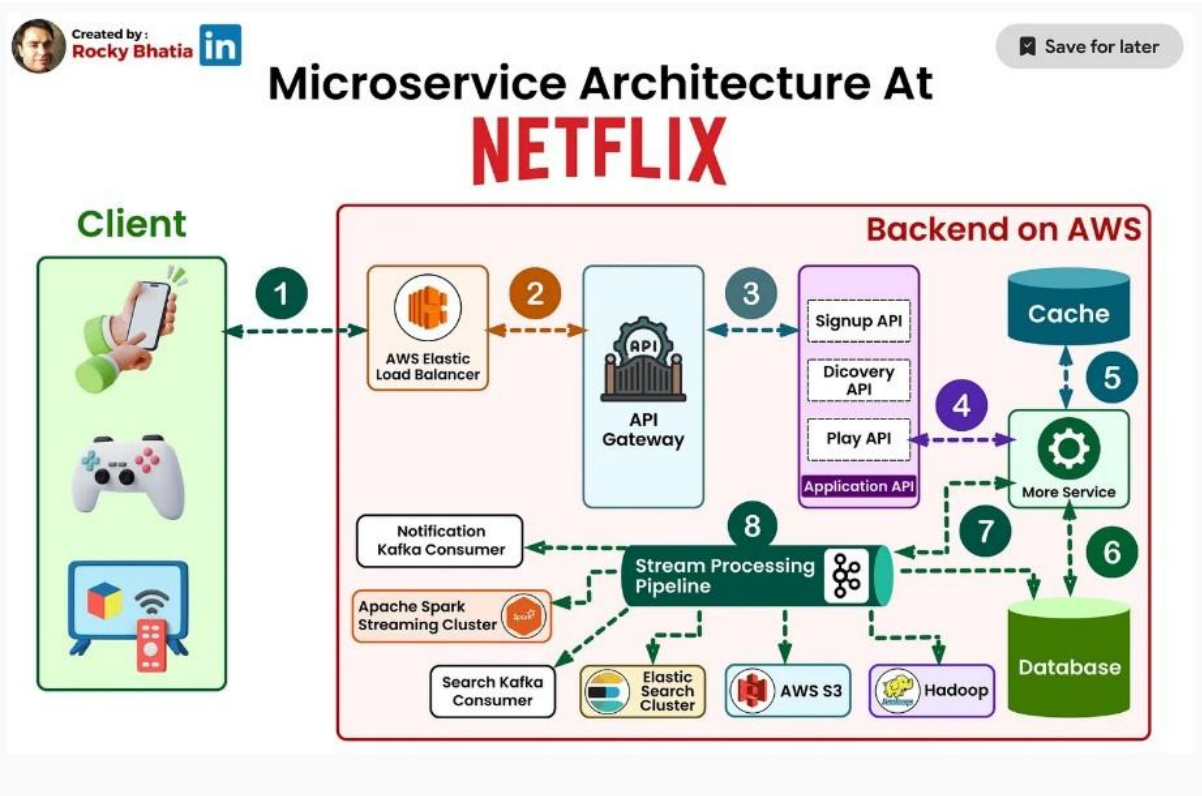

Figure 22: Netflix Architechture [34]

8.2 Scalability mechanisms in netflix

In order to work around this limitation and provide a highly scalable system, microservices are run on Netflix's own container orchestration platform called Titus on top of microVMs running in AWS. With this approach you can do easy scaling by adding more stateless service instances on the fly as demand increases to have elasticity and high availability[35]Service deployment systems like Spinnaker similarly provide strong instrumentation points for supporting continuous delivery, providing you to push application updates many times a day without disruption in service, and to dynamically scale as based on current traffic patterns [36] Performance holds up well, in particular under every-day production workload with the aid of distributed caching and messaging mechanisms, as well as strict separation between stateful/stateless services. These architectural patterns mitigate contention, enhance fault isolation and lower latency for traffic spikes. In combination, cloud-native orchestration, automated delivery pipelines and resilient service design enables Netflix to maintain consistent system performance and reliability at a scale that would be hard to achieve with on-premises solutions while responding efficiently to changing user demand.[35]

8.3 Application gateway (Zuul) in Netflix architecture

Zuul is the front door for all requests from devices and web sites to the backend of the Netflix streaming application[37]It provides dynamic routing, monitoring, resiliency, security and more. This is service handles authentication, rate limiting and request validation for securing the same as well as logging even through requests itself for observability/troubleshooting etc. One, unified Api entry is provided by Zuul and it abstract away the complexity and aligns traffi c control between all of those microservices.[27] By consolidating these cross-cutting concerns into one place, Zuul minimizes the burden of implementing them in each individual microservice as well, making it easier to manage and more consistent across the board. Additionally, Zuul promotes robustness with its fault isolation capabilities and support for graceful degradation in the presence of failure to back-end services. It also has monitoring and logging, so that operators can monitor flows in real time and identify issues with the service. In summary, Zuul contributes not only to scaling and making reliable but also to securing the communication in Netflix's microservicesplatform.[38]

8.4 How all three of these component work together

When a user initiates playback, traffic is generated from the client to the Content Delivery Network (CDN) and then to the gateway (GW), where authorization takes place with respect to the request, which is subsequently sent towards a suitable service point [35]. The gateway serves as a single-entry point, enforcing rules for security, authentication and rate limiting to multiple backend services. At this step, client-side load balancing uses service discovery information from Eureka to select a healthy and low-latency instance of a service, which increases responsiveness and decreases request latency [37]. And this reactive instance selection mechanism provides fault tolerance as any unhealthy services are excluded from request forwarding automatically. Together gateway level control, scalable microservices and smart load balancing rules form a layered traffic management design. This not only brings in more overall system reliability, but also ensures high availability under the condition of traffic surge, and keeps user experience smoothly, by cost-effectively distributing workload on distributed service instances.[39]

8.5 Resiliency and fault tolerance

Through centralized routing and security checks at the gateway, coupled with the clients making real-time localization and decision-making, we have achieved high fault isolation. From common config churns to catastrophic cloud infrastructural failures we have been able to protect clients from it.[40]Reliability patterns like Circuit Breaker, which was traditionally applied using the Hystrix library, protect against cascading failures in ways that prevent unhealthy service instances from being called and maintain overall system stability while avoiding the outage. In addition to these protective mechanisms, some powerful auto-scaling functions automatically manage the number of service instances according to traffic load, and thus the service can be continuable under heavy workloads [37], which may be partly unavailable for failure. Combined, these techniques provide an adaptive structure that ensures latency-tolerance, tolerates variable performance and reduces the system's downtime. With centralized control, fault isolation and dynamic auto-scaled resources in the cloud, Netflix is able to deliver a reliable and responsive service to millions of simultaneous users at a fraction of the cost of comparable internal datacenter-based solutions.[35]

8.6 Performance and user experience

This provides the end users with less latency and as much fewer disruptions during playback, since load balancing enables them to be served by the most responsive servers . [41]The gateway minimizes network overhead by optimizing and aggregating client requests which results in an enhanced performance over bandwidth-constrained devices Routing is made based on real-time monitoring, allowing to deliver uninterrupted gaming even at peak traffic.[39]

8.7 Operational benefits and continuous delivery

In microservices, independent beta testing of service can be done without having any dependency on the whole platform and also it will facilitate feature velocity . Zuul also offers canary testing, which directs a small percentage of traffic to new versions prior to full deployment, ensuring the functionality is working as expected [39]These capabilities together also decrease operational risk and minimize downtime while undergoing changes [38]

8.8 Security considerations

The gateway, by consolidating identity verification, request filtering and TLS termination, greatly limits the possible attack surfaces that can target internal services. [38]Load balancing serves as a method to mitigate targeting from compromised or failing server instances, and telemetry enables real-time threat monitoring and incident response Defense-in-depth architecture[41]is still preserving by using a layered security.

Load sharing avoids overload and increases availability, whilst scalability allows capacity to increase as required [40]The application gateway 10 forms a centralized point of security, routing and client-interaction efficacy [39]Together, these technologies allow Netflix to provide a secure, fast and reliable streaming experience- at scale- for our 100 million members that span across the globe.[31]

9.Serverless function in cloud computing

Serverless systems isolate application logic from server management, letting developers deploy code as individual functions which are called in response to events and billed based on execution durations and resource usage.[42] This simplifies the operational burden as provisioning, patching and scaling is managed by providers, allowing developers to concentrate on business logic instead of orchestration [42], [43]. Economically, pay-per-invocation billing can transform fixed infrastructure commitments into variable operational expenses, which may significantly decrease the hosting cost of spiky or low-duty-cycle workloads; for instance, real-world case studies have documented significant hosting-cost savings following migration to function-as-a-service (FaaS) architecture [43]

9.1 Benefits of serverless function

Flexibility and scalability-wise, serverless platforms provide fine-grained automatic elasticity: functions transparently scale with increased incoming event rates and shrink when not in use

Facts that are specially interesting considering the need for event-driven workloads support such as webhooks, file processing tasks and light APIs [42], [45]. This elasticity also simplifies capacity planning and accelerates feature iteration because provisioning decisions are mostly decoupled from the cloud provider. However, although fundamental limitations such as short maximum function lifetimes, cold-start latency and inability to control the execution environment can impact applicability for long-running or-latency sensitive workloads [42], [5]. More importantly, vendor-built integrations and dark-black runtime systems amplify threat of lock-in and break the predictability performance for multi-cloud settings [42], [43].

9.2 Containerization benefits over serverless function

Containerisation, by contrast, packages applications and their dependencies into isolated, portable runtime units that provide predictable execution environments and fine-grained control over networking, filesystem, and OS-level configuration. Containers, often orchestrated by platforms such as Kubernetes, excel for complex, stateful, or long-running services where process-level control, custom networking, or persistent volumes are required [44]. They enable consistent behaviour across environments and support microservices architectures with well-defined operational boundaries. However, container-based approaches shift the operational burden back to developers and operators, as cluster provisioning, orchestration, auto scaling policies, and upgrades must be designed and managed, increasing operational complexity relative to FaaS [45].

9.3 Comparison between server less function and containerization based on cost and benfits.

Cost behaviour varies among the two paradigms. For workloads that are highly dynamic and event-driven, with low baseline utilization, serverless can achieve lower TCO than servers due to per-invocation billing coupled with the lack of idle waste.[43],[45]. For workloads with more predictable (or consistently high) throughput or that are CPU bound, the rented resources approach is a better prediction of pricing and resource utilization than the pay-per-invocation costs[5], since paying for reserved or guaranteed-used resources has economies of scale compared to aggregate over invocation price point llewellyn-19,garcia11. Performance mutability is another key difference: server-less adds platform-induced variability such as cold starts which can be problematic for latency-sensitive applications; while containers have steadier performance, auto scaling needs to be carefully engineered for sudden load bursts [5], [45].

9.4 Key Observation

In practice, architectural choices should be workload-based: serverless is well-suited to short-lived, event-driven functions where operational ease and cost proportionality are a concern; containerisation is more advantageous when control, runtime consistency or support for long-running or strongly networked processes comes into play. The hybrid use case, where containers are used for the core services and serverless functions for side-line or bursty tasks

usually provides the best tradeoff in terms of cost efficiency, flexibility and operational risk [42], [45], [6].

10. Conclusion

In summary, the confluence of clooud computing, virtualization, containerization and serverless paradigms has radically altered today's IT infrastructure. Virtualization An effective hardware utilization is achieved by sharing of resources in a predictable, and fault isolating manner along with low-cost deployment of multiple VMs on a single host machine [9]. Containerization takes these principles further by giving us a lightweight, portable deployment mechanism that encapsulates an application along with its dependencies for consistent operation across varied execution environments and faster provisioning time[8]. Serverless computing, such as AWS Lambda and Azure Functions [3] [46], provides the functionality of event-triggered and auto-scaling processing which is an important feature for increasing efficiency in operational activities as well as reducing cost. Together, these technologies build on cloud-native principles including load balancing, dynamic scaling and secure application gateways to bring you availability, resilience and quick time-to-deploy. With the combination of virtualization, containers and Serverless architectures - organizations will be able to deploy scalable systems [43], robust in the face of faults and also have more agile system designs which are really what's driving Cloud adoption today

11. Video link of image demonstration

Link - https://youtu.be/6EqvesCn4Rs

12.References

[1] M. J. Scheepers**,** "Virtualization and Containerization of Application Infrastructure: A Comparison," *Containerization (computing)*, 2014

[2] S. Eismann "The state of serverless applications: Collection, characterization, and community consensus," *IEEE Trans. Software Eng.*, vol. 48, no. 10, pp. 4152–4166, 2021.

[3] "Survey on serverless computing**,"** *Journal of Cloud Computing*, vol. 10, art. no. 39, 2021

[4] J. Scheuner and P. Leitner, "Function-as-a-Service Performance Evaluation: A Multivocal Literature Review," arXiv, 2020.

[5] H. B. Hassan, S. A. Barakat, and Q. I. Sarhan, “Survey on serverless computing,” *Journal of Cloud Computing: Advances, Systems and Applications*, vol. 10, art. no. 39, Jul. 2021,

[6] V. N. K. Kundavaram, “Serverless Computing: A Comprehensive Analysis of Infrastructure Abstraction in Modern Cloud Computing,” *International Journal for Multidisciplinary Research (IJFMR)*, vol. 6, no. 6, pp. 1–Nov.–Dec. 2024,

[7] S. P. Kane and K. Matthias, *Docker: Up & Running: Shipping Reliable Containers in Production*, 3rd ed., Sebastopol, CA, 2019.

[8] H. Aqasizade, E. Ataie, and M. Bastam, “Experimental Assessment of Containers Running on Top of Virtual Machines,” *arXiv*, Jan. 2024.

[9] H. Sturley *et al*., “Virtualization vs. Containerization, a Comparative Approach for Application Deployment in the Computing Continuum Focused on the Edge,” *Future Internet*, vol. 16, no. 11, 2024.

[10] F. Douglis and O. Krieger, "Virtualization," in *IEEE Internet Computing*, vol. 17, no. 2, pp. 6-9, March-April 2013,

.[11] N. Kumari and U. N. Singh, "A Study of Virtualization Techniques Applied in Cloud Computing Environment," *Research Scholar Magadh University, Bodh-Gaya; Associate Prof. & Head, Dept. of Physics, K.S.M. College, Aurangabad (Bihar)*.

[12]Z. Li *et al*., “Performance Overhead Comparison between Hypervisor and Container based Virtualization,” in *Proc. 2017 IEEE AINA*, 2017.

[13] A. M. Potdar, D. G. Narayan, S. Kengond, and M. M. Mulla, “Performance Evaluation of Docker Container and Virtual Machine,” *Procedia Computer Science*, vol. 171, pp. 1419–1428, 2020,

[14] D. Silva, J. Rafael, and A. Fonte, “Toward Optimal Virtualization: An Updated Comparative Analysis of Docker and LXD Container Technologies,” *Computers*, vol. 13, no. 4, p. 94, Apr. 2024

[15] M. Narasimhulu, D. V. Mounika, P. Varshini, A. K and T. R. K. Rao, "Investigating the Impact of Containerization on the Deployment Process in DevOps," *2023 2nd International Conference on Edge Computing and Applications (ICECAA)*, Namakkal, India, 2023

[16] https://docs.docker.com/

[17] M. H. Ibrahim, M. Sayagh, and A. E. Hassan, “Too many images on DockerHub! How different are images for the same system?,” *Empirical Software Engineering*, vol. 25,

[18] Q. Xu, M. Awasthi, K. T. Malladi, J. Bhimani, J. Yang and M. Annavaram, "Docker characterization on high performance SSDs," *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Santa Rosa, CA, USA, 2017

[19] N. Naik, "Docker container-based big data processing system in multiple clouds for everyone," *2017 IEEE International Systems Engineering Symposium (ISSE)*, Vienna, Austria, 2017

[20] H. Zeng, B. Wang, W. Deng and W. Zhang, "Measurement and Evaluation for Docker Container Networking," *2017 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*, Nanjing, China, 2017,

[21] U. Abbasi, E. H. Bourhim, M. Dieye and H. Elbiaze, "A Performance Comparison of Container Networking Alternatives," in *IEEE Network*, vol. 33, no. 4, pp. 178-185, July/August 2019

[22] M. Mesbahi and A. M. Rahmani, "Load Balancing in Cloud Computing: A State of the Art Survey," *International Journal of Modern Education and Computer Science (IJMECS)*, vol. 8, no. 3, pp. 64–78, Mar. 2016,

[23] R. Kaur and P. Luthra, "Load balancing in cloud computing," in *Proc. International Conference on Recent Trends in Information, Telecommunication and Computing (ITC)*, 2014

[24] S. Lehrig, H. Eikerling, and S. Becker, "Scalability, elasticity, and efficiency in cloud computing: A systematic literature review of definitions and metrics," in *Proc. 11th Int. ACM SIGSOFT Conf. Quality of Software Architectures (QoSA '15)*, New York, NY, USA, 2015

[25] C. Qu, R. N. Calheiros, and R. Buyya, "Auto-Scaling Web Applications in Clouds: A Taxonomy and Survey," *ACM Computing Surveys*, vol. 51, no. 4, Art. 73, Jul. 2018,

[26] S. Lehrig, H. Eikerling, and S. Becker, "Scalability, elasticity, and efficiency in cloud computing: A systematic literature review of definitions and metrics,"

[27] L. Demetrio, A. Valenza, G. Costa, and G. Lagorio, "WAF-A-MoLE: Evading Web Application Firewalls through Adversarial Machine Learning," in *Proc. 35th ACM/SIGAPP Symposium on Applied Computing (SAC '20)*, Brno, Czech Republic, Mar. 30-Apr. 3, 2020

[28] P. Genfer, S. Serbout, G. Simhandl, and U. Zdun, "Understanding security tactics in microservice APIs using annotated software architecture decomposition models – a controlled experiment," *Empirical Software Engineering*, vol. 30, article 66, 2025

[29] S. Lehrig, H. Eikerling, and S. Becker, "Scalability, elasticity, and efficiency in cloud computing: A systematic literature review of definitions and metrics," in *Proc. 11th International ACM SIGSOFT Conference on Quality of Software Architectures (QoSA '15)*, 2015,

[30] https://about.netflix.com/en

[31] https://www.fastpix.io/blog/system-design-and-architecture-for-a-netflix-like-app

[32] https://netflixtechblog.com/netflix-edge-load-balancing-695308b5548c

[34] https://rockybhatia.substack.com/p/inside-netflixs-architecture-how

[35] https://blog.bytebytego.com/p/a-brief-history-of-scaling-netflix

[37] https://netflixtechblog.com/announcing-zuul-edge-service-in-the-cloud-ab3af5be08ee

[38] https://www.fastpix.io/blog/system-design-and-architecture-for-a-netflix-like-app

[39] https://xcelore.com/blog/an-essential-guide-to-api-gateway-a-netflix-case-study/

[40] https://netflixtechblog.com/netflix-edge-load-balancing-695308b5548c

[41] https://www.clustox.com/blog/netflix-case-study/#6-databases-caching-how-does-netflix-handle-data-at-scale

[42] E. Jonas et al., "Cloud Programming Simplified: A Berkeley View on Serverless Computing," arXiv:1902.03383, 2019.

[43] G. Adzic and R. Chatley, "Serverless Computing: Economic and Architectural Impact," Proc. ESEC/FSE, 2017..

[44] I. Čilić et al., "Performance Evaluation of Container Orchestration Tools," Sensors, 2023.

[45] C. Fan et al., "Microservices vs Serverless: A Performance Comparison," CLOSER, 2020.

[46] https://aws.amazon.com/about-aws/