

University of Wales Trinity Saint David (UWTSD)

Program: MSc Cybersecurity and Digital Forensics

Module Name: Cloud Engineering Principles and Practice

Module Code: ACCA7027

Assessment Title: A Critical Examination of Containerization, Virtualization, and Contemporary Deployment Techniques

Lecturer: Dr. Renuga Jayakumar

Student Name: Md. Asaddoz Jaman

Student ID: 2526322

University Email: 2526322@student.uwtsd.ac.uk

Submission Date: 28 April 2026

Declaration:

I declare that this assessment is my own work and that all sources have been properly acknowledged in accordance with academic integrity and referencing guidelines. I understand the university regulations on plagiarism and confirm that this submission complies with those requirements.

Table of Contents

ABSTRACT.....	3
1. INTRODUCTION.....	3
2. FUNDAMENTALS OF VIRTUALIZATION.....	5
2.1 Architecture and Working of Hypervisor	5
2.2 Hypervisor Types.....	6
2.3 Lifecycle of a Virtual Machine and Resource Management	7

2.4 Benefits of Virtualisation	7
2.5 Disadvantages of Virtualisation	7
2.6 Importance of Virtualisation in Cloud Computing	8
3. THE FUNDAMENTALS AND ADVANTAGES OF CONTAINERIZATION	8
3.1 Container Architecture and Operating Principle	9
3.2 Containerisation vs. Virtualisation	9
3.3 Docker and Container Ecosystem	10
3.4 Advantages of Containerisation	10
3.5 Microservices Architecture and Containerisation	11
3.6 Security Implications in Containerization	11
3.7 Importance of Containerisation in Cloud Computing	11
4. DOCKER IMPLEMENTATION AND PRACTICAL DEMONSTRATION.....	12
4.1 Application Environment and Local Development	12
4.2 Building the Dockerfile: Infrastructure as Code.....	13
4.3 The Image Building Process and Layer Caching	14
4.4 Container Deployment & Lifecycle Management.....	14
4.5 Validating, Monitoring and Resources Analysis.....	16
4.6 Advanced Cloud Features: Volumes, Networking, and Distribution	17
4.7 Critical Analysis of Practical Results.....	18
5. FUNDAMENTAL IDEAS IN CLOUD COMPUTING	18
5.1 Load Balancing	19
5.2 Scalability in Cloud Computing	20
5.3 Application Gateways	21
5.4 Relation between Load Balancing, Scalability, and Application Gateway	21
5.5 Significance in Current Cloud Technologies.....	22
6. CONTAINERIZATION VS. SERVERLESS COMPUTING.....	23
7. MOTIVATORS FOR THE GROWTH OF CLOUD COMPUTING	24
8. CONCLUSION	25
REFERENCES	27
PRESENTATION DETAILS AND LINK	27
Presentation File Name.....	27
Presentation Links.....	27

ABSTRACT

Cloud computing has become one of the most revolutionary paradigms in current information technologies by providing computing capabilities to businesses and other organizations in a scalable and cost-effective way by accessing them over the Internet. The purpose of this report is to present an overview of key cloud computing concepts and their practical implementation, especially with respect to virtualization, containerization, and its execution via Docker, together with other key elements of cloud architecture, including scalability, load balancing, and application gateway.

Virtualization has been chosen as the first topic due to the critical role of virtualization for cloud computing, as it provides a possibility to run several vms on the same host using a virtualizing layer called a hypervisor. In this section, types of hypervisors will be compared, along with discussing advantages and drawbacks of using vms as an underlying cloud computing architecture with regard to resource consumption, performance impact, and isolation. On the basis of this, containerization will be reviewed as an effective alternative approach.

Another aspect covered in the report is Docker – an application containerisation solution. The report presents the process of creating Dockerfile, building images, running containers, managing volumes and networking, as well as distributing images with the help of Docker Hub. Thus, a real-life example of developing and deploying cloud-native applications using Docker is provided.

The report also provides analysis of such vital concepts of cloud computing as load balancing, which means distribution of loads among several servers, and scalability that involves handling more loads with either vertical or horizontal growth. One more vital topic that the report covers is application gateways that play an important role in distributing traffic.

Lastly, the report covers the comparison of containerisation and serverless computing by providing detailed information on each of the two approaches. Thus, it becomes possible to learn about their strengths, weaknesses, and appropriate scenarios. As a result, the report shows that cloud computing keeps on evolving quickly due to flexibility and cost effectiveness.

1. INTRODUCTION

In this regard, cloud computing is regarded as one of the most revolutionary approaches to information technology in modern times by offering an entirely new approach to computing resource provision, access, and management. Cloud computing refers to a model whereby a user can access computing resources in terms of processing capabilities, storage, database, networks, and application software via internet without the need to manage underlying physical infrastructure. The adoption of this revolutionary approach has led to numerous improvements in scalability and efficiency for various organisations across the world.

Based on National Institute of Standards and Technology (NIST) definition, cloud computing can be described as a form of computing that provides on-demand and convenient network access to a shared pool of resources that can be rapidly provided and released with minimal effort or intervention from service providers [1]. In this context, some of the key features include the following: on-demand self-

service, broad network access, resource pooling, rapid elasticity, and measured service. These features make cloud computing different from normal IT infrastructures in many aspects.

The emergence of cloud computing technology is largely linked to innovations in distributed computing, networking, and virtualization. One key component of cloud computing is the concept of virtualization that makes it possible to abstract physical computer resources into several isolated virtual machines (VM). Every such VM works as a separate computing system having its operating system and applications but shares hardware. Virtualization is controlled by special software that acts as a manager of computing resources including CPU, memory, and storage and allocates those to multiple VMs effectively [2]. For these reasons, virtualization has evolved into one of the most important enablers of data center technology and clouds in particular.

Virtualization comes, however, at a price because running a dedicated copy of the operating system on each virtual machine adds up significant computational overhead. Such issues have led to the creation of containerization, a new technology offering greater efficiency compared to the classic way of implementing virtualization. Unlike in case of virtualization, containerization involves operating system-level virtualization where multiple applications are executed inside isolated environments called containers and share the same operating system. Containerization is much more effective in terms of resource utilization, startup speed, and portability.

Docker has become one of the most popular tools for containerisation of applications, allowing easier deployment and management of containerized applications. Developers can easily create images that run their applications on all kinds of systems in a reliable way. Docker has become an essential element in the process of DevOps development, cloud development and microservices development where large applications are split up into small services that can be independently deployed and scaled.

In addition to using virtualization and containers, the design of cloud computing architectures includes a number of elements that guarantee reliability and scalability of the architecture. Load balancing helps to spread network requests evenly between servers, thus avoiding server overload and ensuring its availability. The scalability principle provides for dynamic resource allocation in the case when the load on systems grows, whether vertically, adding more resources to a machine, or horizontally, adding more machines. Application gateways work at the application level, performing additional functions such as traffic routing, security policies enforcement, and SSL termination, as well as acting as WAF [4].

Nevertheless, cloud computing poses certain challenges along with the benefits it offers. Security and privacy issues, vendor lock-in problems, network latency and reliability, as well as the reliance on consistent and stable network connectivity, are some of the challenges that come with this technology. Considering the fact that many companies are moving their vital applications to the cloud environment, solving these challenges is necessary for designing reliable systems. In addition, adequate architecture is needed to achieve an effective combination of performance, efficiency, and security.

This report aims at delivering an insightful understanding of the fundamental concepts associated with cloud computing, including the use of virtualisation and containerisation, as well as Docker. Moreover, an analysis of cloud infrastructure elements such as load balancing, scalability, and application gateways will be presented. The current report will also shed light on the new paradigms that have emerged in recent years, such as serverless computing, and discuss their pros and cons as compared to containerization.

2. FUNDAMENTALS OF VIRTUALIZATION

Virtualisation is an essential enabler technology in contemporary cloud computing, as it permits the abstraction and sharing of hardware computing capabilities within isolated execution environments. Virtualisation enables transitioning from hardware-based computing to software-defined infrastructure through the creation of an additional abstraction layer between hardware components and applications running on top of them. Using virtualisation techniques, multiple VMs can be hosted by a single hardware device, and every of those VMs behaves like an isolated computer system with its own operating system and allocated computing resources.

In essence, virtualisation provides an abstraction layer between hardware components and applications running on them, which is managed by special software called hypervisor or a Virtual Machine Monitor (VMM). Hypervisor manages computing resources allocated to multiple operating systems running on the same hardware, providing necessary isolation between virtual machines. Hypervisors ensure controlled sharing of hardware between multiple operating systems and prevent interferences between them. As Rosenblum and Garfinkel explain, hypervisors create controlled computing environments that enable the operation of multiple operating systems within one physical environment without interfering with each other [2].

2.1 Architecture and Working of Hypervisor

The hypervisor acts as the controlling component within a virtualised environment. Hardware requests from virtual machines are intercepted by the hypervisor and converted to physical hardware actions. This guarantees that each virtual machine works as though it has access to the physical hardware, even though this is not always true.

A general architecture of virtualisation comprises of three main layers in a computer: physical hardware layer, hypervisor layer, and virtual machine layer. The physical layer represents real computing resources in the form of processor units, memory cards, storage, and other networking components. The hypervisor is placed above the physical layer and is used to allocate computing resources. On top of the hypervisor are various virtual machines, which run their own operating systems and software programs.

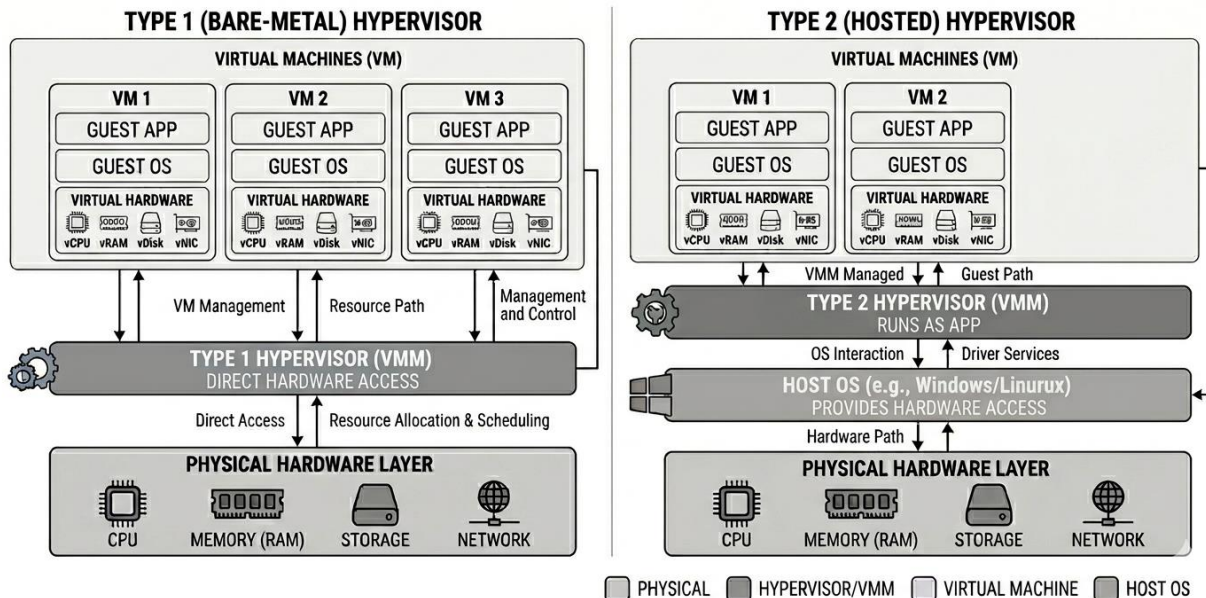


Figure 1: Virtualization Architecture Using Hypervisor Layer

This figure-1 shows how a physical server can be converted into various virtual servers through the use of hypervisor software. The virtual machines run their own operating systems yet share the same hardware resources.

Hypervisor performs resource allocation tasks such as CPU scheduling, memory management, and input/output functions. Isolation is also provided by the hypervisor, which ensures that any problem experienced by one VM will not affect other virtual machines.

2.2 Hypervisor Types

There are two kinds of hypervisors that enable virtualization: Type 1 and Type 2. Both types have specific architectural design features and applications.

Type 1 Hypervisors (Bare Metal Virtualization)

Type 1 hypervisors operate on physical machines without the need for a host operating system. As they work directly with the physical hardware, they are highly efficient and provide good isolation capabilities. Type 1 hypervisors are commonly utilized in enterprise data centres and infrastructure clouds because they offer high-performance virtualization. Some of the Type 1 hypervisors are VMware ESXi, Microsoft Hyper-V, and Xen. Type 1 hypervisors are employed in Infrastructure as a Service (IaaS) in major cloud providers like AWS and Microsoft Azure, where numerous virtual machines are deployed [2].

Type 2 Hypervisors (Hosted Virtualization)

Type 2 hypervisors require a host operating system to function, meaning they do not interact directly with physical devices. They are commonly used for learning, software testing, and development purposes. Type 2 hypervisors are slower than Type 1 hypervisors since they need the host OS, making them more complicated to set up and maintain.

2.3 Lifecycle of a Virtual Machine and Resource Management

Virtual machine is a software implementation of an actual computer which has virtual hardware components like CPU, memory, disk space, and network interfaces. Life cycle of virtual machine includes its creation, deployment, execution, suspension, live migration, and eventually termination.

Resources of the VM are allocated by hypervisor depending on needs. Fair sharing of CPU time among various VMs is achieved by appropriate scheduling algorithm. Memory management methods include ballooning and overcommitting that make efficient use of available RAM. Virtual storage is also possible through abstraction of disks, realized via virtual hard disk image.

The ability of virtual machine to be migrated live is undoubtedly the most significant aspect of virtualization. It makes the process of transferring a running VM from one physical host to another seamless, and is very important in cloud computing environments.

2.4 Benefits of Virtualisation

Virtualisation brings about several advantages, making it a key element of today's cloud computing architecture:

- **Increased Hardware Utilization:** Conventional computing systems have been characterized by inefficient hardware utilization. With virtualisation, several workloads can be executed simultaneously on a single computer system, increasing efficiency.
- **Cost Effectiveness:** The use of fewer physical computers lowers capital expenditures due to reduced cost incurred in procuring hardware, power consumption, cooling, and management expenses.
- **Separation and Isolation:** Each virtual machine works in an isolated environment. This ensures that any problem occurring in one virtual machine will not affect other running machines. Isolation is especially relevant in multi-tenant cloud computing environments.
- **Flexibility and Scalability:** Virtual machines can be quickly instantiated, duplicated, scaled up or down, and terminated according to demand, providing scalability needed in cloud computing applications.
- **Disaster Recovery and Backups:** The virtualization environment provides the capability of creating backups or snapshots of virtual machines.

2.5 Disadvantages of Virtualisation

However, alongside its benefits, virtualisation brings a number of disadvantages too:

- **Overhead from Running OSes:** The full-blown operating systems running inside VMs consume extra CPU, memory, and disk space compared to native operations.
- **Higher VM Start-Up Times:** Boot times of full-fledged OSes inside VMs are longer than that of lightweight containers.
- **Inefficiency of Resource Usage in Dynamic Environments:** VMs still use their resources even when not doing anything, which makes them inefficient in dynamic cloud infrastructures.
- **Complexity Management:** Hypervisor-based management of numerous VMs becomes extremely complex.

2.6 Importance of Virtualisation in Cloud Computing

In cloud computing, virtualisation plays an important role especially in IaaS. It allows cloud providers to utilise hypervisor-based virtualisation to carve out elastic VMs out of the underlying hardware infrastructure. This makes it possible to provide on-demand resources to users anywhere around the world.

One of the most important benefits of virtualisation in cloud infrastructure is high workload isolation, which allows different clients to share one piece of infrastructure while maintaining isolation of their workloads.

Finally, virtualisation allows implementing dynamic resource allocation and management, including scaling, load balancing, and other techniques, by combining it with cloud orchestration frameworks. Therefore, virtualisation is a critical component underpinning all cloud-based applications [2].

3. THE FUNDAMENTALS AND ADVANTAGES OF CONTAINERIZATION

In contrast to hardware or software virtualisation, containerisation can be described as operating system-level virtualisation where an application runs in an isolated user space called container. While conventional virtual machines have an entire guest operating system on board, containerisation shares the host operating system's kernel but maintains logical isolation among applications. Consequently, containers are considerably more lightweight and performant compared to virtual machine implementations. For this reason, containerisation can be seen as one of the drivers for the current trends towards cloud-native application development.

The main idea of containerisation can be summarised by the concept of packaging an application along with its dependencies and environment into a deployable unit. In other words, the application will have the same behaviour in different environments. As noted by Merkel, the advantages of containerisation include process isolation, resource management capabilities and avoidance of virtual machine overhead, thus making it very well suited for scalable distributed systems [3].

3.1 Container Architecture and Operating Principle

Containers operate through the use of the features offered by the host operating system kernel. In Linux, there are two core functions of the host system kernel required for container operation: namespaces and control groups.

Namespaces isolate processes and ensure that each container has a separate vision of all the resources used by its processes. The resources isolated include process ID space, mount points, inter-network interfaces, user IDs, etc. As a result, any process within one container is unaware of other containers' or even host system's processes.

Control Groups manage resource allocation. With their help, the CPU usage time, RAM, disk I/O operations, and network bandwidth available to each particular container are restricted. This makes sure that multiple containers running at the same time allocate system resources evenly.

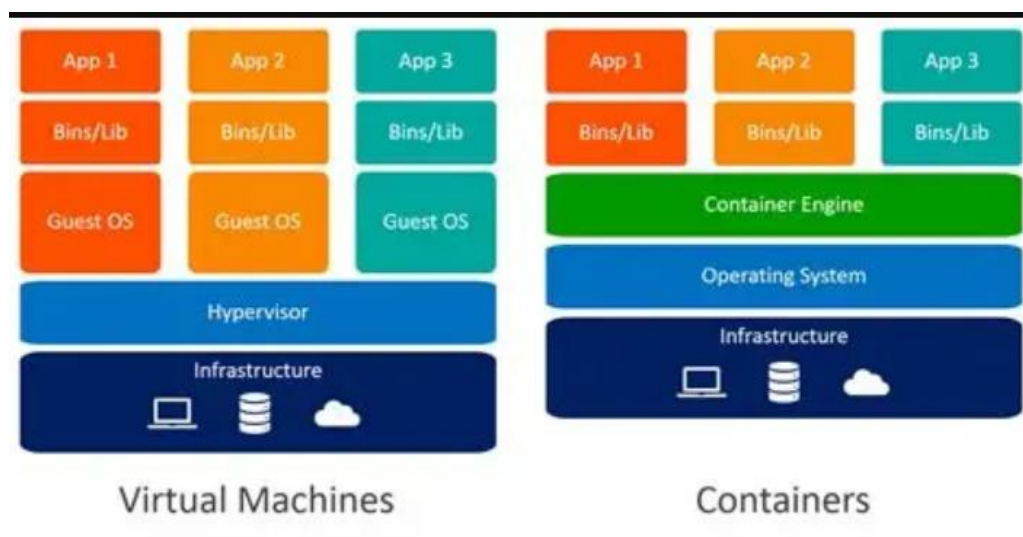


Figure 2: Container Architecture vs. Virtual Machine Architecture

In the figure-2, the presented diagram highlights the architecture differences between VMs and containers. VMs have the guest operating system placed on top of the hypervisor, while containers do not require a separate copy of the operating system but only the relevant libraries needed for applications.

3.2 Containerisation vs. Virtualisation

An important concept when examining containerisation is the comparison between the two concepts of containerisation and virtualisation. Although both concepts share similarities such as application isolation and better resource utilisation, they are different in many aspects.

While virtualisation involves using a complete copy of the operating system through a hypervisor to run an application, containers use the kernel from the host OS with the application alone. The main implications are that virtualisation consumes much more resources and takes time to initiate.

As a result, there are some clear performance benefits to using containerisation rather than virtualisation:

- The speed of starting the application is reduced by seconds since it starts almost instantly.
- Resource consumption is much lower, which means the number of applications per server can be increased.
- Portability is enhanced by consistent application packaging across different platforms.

Virtualisation provides more application isolation since no kernels are shared.

3.3 Docker and Container Ecosystem

Docker is the most popular containerization platform and has been pivotal in promoting the usage of containerization technology within the industrial as well as the academic circles. Docker offers a full-blown ecosystem that makes building, shipping, and running containers easier than ever. It offers a standardized format known as Docker images which act as read-only templates used to create containers.

Docker image refers to a read-only blueprint consisting of all the necessary files such as the application code, runtime, system tools, libraries, and dependencies needed for application execution. Once a Docker image is run, a new container instance is created.

The main components of Docker include the following:

- Docker Engine: Creates and manages containers.
- Docker Images: Used as a template to create containers.
- Docker Containers: Instances of Docker images in action.
- Docker Hub: Cloud-based storage location for Docker images.

Docker has proven to be an indispensable tool when it comes to DevOps. The tool supports CI/CD where developers are able to build an application and deploy it on various environments seamlessly.

3.4 Advantages of Containerisation

There are numerous benefits associated with containerisation which have made it very popular in the field of cloud computing.

- Light Weightness: A key feature of containerisation is the absence of the requirement for an operating system. This minimises memory and CPU requirements, resulting in greater application efficiency.
- Quick Start-up and Shuts Down: The absence of boot time for the operating system means that containers start up and shut down within just seconds.
- Portability: Containers bundle the entire application, making sure that it runs consistently on various platforms, whether they are developer's laptops or physical/on-demand servers.
- Scalability: Containerisation is known for being highly scalable due to the ability of running a separate service for each container and scaling according to the requirements.

- **Efficient Resource Utilisation:** Multiple containers run on the same operating system kernel which optimizes the utilisation of resources by increasing the density of containers per node.

3.5 Microservices Architecture and Containerisation

The implementation of microservices architecture with the help of containerisation has become one of the major use cases in contemporary software engineering. The idea of microservices involves the separation of monolithic applications into smaller components called microservices that operate independently and communicate via lightweight communication protocols, such as HTTP or messaging systems.

Thus, each microservice may be implemented within the same container, providing developers with flexibility during the development process, deployment, and maintenance. For instance, major web services, including Netflix and Amazon, implement microservices architecture using containerisation technology to process thousands of requests from millions of users daily.

3.6 Security Implications in Containerization

Though containerization offers isolation at the process level, it lacks the isolation of VMs since both use the same kernel of the host OS. This exposes the process to certain security concerns in case of kernel vulnerabilities.

Some of the security issues include container escape, vulnerabilities in images, and permission configuration issues. In order to address such risks, various security measures include:

- Regular scanning of container images for vulnerabilities
- Use of small base images to minimize the attack vector
- Runtime security monitoring in the environment
- Implementing role-based access control (RBAC)
- Container network segregation

In line with the security best practices offered by IBM Cloud, security for containerized applications requires an integrated strategy that incorporates infrastructure, image, and runtime protection [4].

3.7 Importance of Containerisation in Cloud Computing

It can be claimed that the adoption of containerisation has made an enormous contribution to cloud computing. Currently, many cloud providers, such as Amazon Web Services, Microsoft Azure, and Google Cloud Platform, widely utilise containers in their computing infrastructures. These platforms provide a wide range of cloud-related services using containers as the key building blocks.

In addition, containers work seamlessly with orchestration tools, including Kubernetes, which can be used for deploying containerised applications in cloud computing environments.

4. DOCKER IMPLEMENTATION AND PRACTICAL DEMONSTRATION

The move from theory-driven virtualization to actual containerization can best be shown through the case of Docker, an open-source software tool used to automatically create containers within which applications can run. Unlike conventional VMs which emulate the hardware abstraction layer, Docker works on an operating system abstraction level to achieve high density resource utilization with quick boot times. In order to illustrate cloud-native deployment process using the containerization model, a web application was developed using Flask framework for this assessment. The following section discusses the implementation process comprehensively with supporting empirical data.

4.1 Application Environment and Local Development

The life-cycle of any containerized application starts with the development process where the application is tested locally before being deployed remotely. In order to fulfill this purpose, a Python code named app.py was created to return personalized data regarding student IDs. This guarantees uniqueness of the application under test. In line with the philosophy of cloud engineering where environmental consistency should be assured, a requirements.txt file specifying the Flask version used was created.



```
Terminal
PS C:\Users\R_Asad\Desktop\docker-python-app> docker run -d -p 5000:5000 flask-docker-app
0ee282b03c3e78d626599d249ca442226e896f04116b09034db98b1be657b8ae
PS C:\Users\R_Asad\Desktop\docker-python-app> docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS        NAMES
PS C:\Users\R_Asad> cd C:\Users\R_Asad\Desktop\docker-python-app
PS C:\Users\R_Asad\Desktop\docker-python-app> dir

Directory: C:\Users\R_Asad\Desktop\docker-python-app

Mode                LastWriteTime         Length Name
----                -
-a----            4/28/2026   2:24 AM             192 app.py
-a----            4/28/2026   2:17 AM             128 Dockerfile
-a----            4/28/2026   1:32 AM              5 requirements.txt

PS C:\Users\R_Asad\Desktop\docker-python-app> docker build -t flask-docker-app .
[+] Building 9.2s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 167B
                                docker:desktop-linux
                                0.0s
                                0.0s
RAM 1.54 GB  CPU 1.39%  Disk: 3.27 GB used (limit 1006.85 GB)
```

Figure 1: Project Structure

The first action entails setting up the project structure. The screenshot below of the directory C:\Users\R_Asad\Desktop\docker-python-app illustrates that all three pillars have been implemented, namely the source code (app.py), dependencies (requirements.txt), and Dockerfile, shown on figure-1. This structure is critical since it provides the necessary information needed by the Docker "context" when determining what files should be uploaded to the Docker daemon during the building process.

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def home():
    return "Md. Asaddoz Jaman - 2526322"

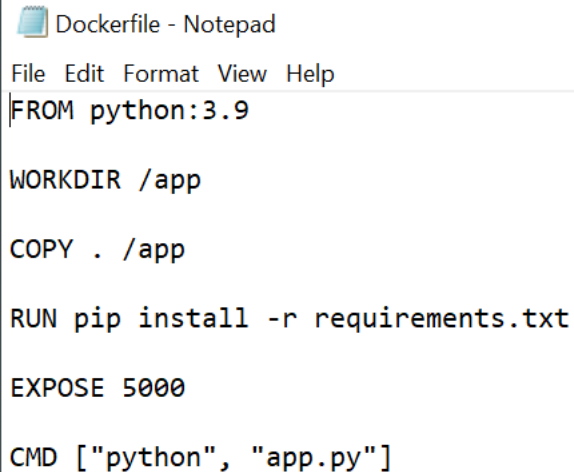
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Figure 2: Source Code Implementation

The program logic is incorporated into the app.py file, shown on figure-2. The application is set to run on port 5000 using host='0.0.0.0'. Binding to the address 0.0.0.0 is important because it enables the Flask server to listen on all interfaces inside the container, allowing the computer to forward incoming requests to it through the Docker bridge.

4.2 Building the Dockerfile: Infrastructure as Code

Dockerfile is a declarative specification for building the container image. The manual server configuration process is avoided via the use of a set of consecutive commands which Docker performs in order to build the final container image. In accordance with best practices, the Dockerfile used in this case was built according to the "infrastructure as code" paradigm and therefore uses layers.



```
FROM python:3.9

WORKDIR /app

COPY . /app

RUN pip install -r requirements.txt

EXPOSE 5000

CMD ["python", "app.py"]
```

Figure 3: Dockerfile Instructions

In the figure-3 you see this particular implementation uses a certain base image named python:3.9. Using a versioned image allows avoiding possible changes introduced due to a possible release of new Python versions in the future. Finally, the WORKDIR /app command sets up a working folder for the

application and COPY ./app places the local context into it. Finally, the pip install run command is executed inside the image.

4.3 The Image Building Process and Layer Caching

The image building converts the static Dockerfile to an executable item. In the course of running docker build -t flask-docker-app ., the complex caching process is used by Docker. The Docker file instructions form read-only layers; therefore, if the code and instructions remain unchanged from one build to another, Docker uses the same layers, thus greatly speeding up the process of continuous integration.

```
PS C:\Users\R_Asad\Desktop\docker-python-app> docker build -t flask-docker-app .
[+] Building 9.2s (10/10) FINISHED
=> => transferring dockerfile: 167B
=> [internal] load metadata for docker.io/library/python:3.9
=> [auth] library/python:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 2B
=> => resolve docker.io/library/python:3.9@sha256:da5aee29682d12a6649f51c8d6f15b87deb3e6c524b923c41d0cb3304d07c913
=> [internal] load build context
=> [3/4] COPY . /app
=> [4/4] RUN pip install -r requirements.txt
=> exporting to image
=> => exporting layers
=> => exporting manifest sha256:e41aa03b4116daf7d217b2570a6fbbf46c5e9cf73ab3bdc6a7c2bd3da5f91c7d
=> => exporting config sha256:3cf97b2a32e31f1aee6cf1eed3aa6086d4f234326a047a8a8ef8173b3dd8410c
=> => exporting attestation manifest sha256:5ee4fa3f5d26f9119815b6970e2eb44ea15487edca38011977b6fa76453fc4af
=> => exporting manifest list sha256:d51fcddbe1399002a716a3441da588010cc9ae7e180b1d049130bfe6adfd9671
=> => naming to docker.io/library/flask-docker-app:latest
=> => unpacking to docker.io/library/flask-docker-app:latest
07a9c23e2b5a5f7b80c7c65c3683cc13fcdd66790b91ca9436bc5279a26fc71c
```

Figure 4: Build Automation Process

In figure-4 the terminal output describes the entire 10-stage process of creating the Docker image. This includes loading meta-information about the image and eventually exporting the layers into a separate file. At this stage, each image gets a unique ID, which is a cryptographic checksum for the image layers.

```
PS C:\Users\R_Asad\Desktop\docker-python-app> docker run -d -p 5000:5000 flask-docker-app
0ee282b03c3e78d626599d249ca442226e896f04116b09034db98b1be657b8ae
PS C:\Users\R_Asad\Desktop\docker-python-app> docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
--------------	-------	---------	---------	--------	-------

Figure 5: Build Conclusion and Metadata

The log output proves that the image creation took place and the final size of the resulting file is nearly 1.59GB shown on figure-5. Although this is considered a fat image because it uses the Python base, it includes everything needed to launch an application on any computer, be it a local machine or even an AWS EC2 environment.

4.4 Container Deployment & Lifecycle Management

Once the image is ready, it is instantiated into a container. A container can be described as a stateful process running inside the host operating system. In order to enable communication, Port Mapping

needs to take place. Since the Flask server is bound to the host port 5000, we bind the host port 5000 with the container port 5000 via the -p flag, shown on figure-6.

```

=> => transferring context: 296B
=> CACHED [2/4] WORKDIR /app
=> [3/4] COPY . /app
=> [4/4] RUN pip install -r requirements.txt
=> exporting to image
=> => exporting layers
=> => exporting manifest sha256:e41aa03b4116daf7d217b2570a6fbbf46c5e9cf73ab3bdc6a7c2bd3da5f91c7d
=> => exporting config sha256:3cf97b2a32e31faee6cf1eed3aa6086d4f234326a047a8a8ef8173b3dd8410c
=> => exporting attestation manifest sha256:5ee4fa3f5d26f9119815b6970e2eb44ea15487edca38011977b6fa76453fc4af
=> => exporting manifest list sha256:d51fcddbe1399002a716a3441da588010cc9ae7e180b1d049130bfe6adfd9671
=> => naming to docker.io/library/flask-docker-app:latest
=> => unpacking to docker.io/library/flask-docker-app:latest
PS C:\Users\R_Asad\Desktop\docker-python-app> docker run -d -p 5000:5000 flask-docker-app
07a9c23e2b5a5f7b80c7c65c3683cc13fcd66790b91ca9436bc5279a26fc71c
docker: Error response from daemon: failed to set up container networking: driver failed programming external connectivity on endpoint magical_k
eller (9738eb72daed1e36f6d9714d5f1a7d4d85683d02f609b40d2d3bf473ea1cc367): Bind for 0.0.0.0:5000 failed: port is already allocated

Run 'docker run --help' for more information
PS C:\Users\R_Asad\Desktop\docker-python-app> docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                               NAMES
0ee282b03c3e   cf317fa43389   "python app.py"         13 minutes ago Up 13 minutes  0.0.0.0:5000->5000/tcp, [::]:5000->5000/tcp   ecstatic_hodgkin

```

Figures 6: Port Management/Resource Contention

Port management is one of the key aspects when managing containers in the cloud. During the initial deployment, there was an issue where "Bind for 0.0.0.0:5000" could not take place because there was another active container running on the host port 5000. This experience proved the importance of executing the commands docker stop and docker rm in preparation for a re-deployment of an updated application.

```

PS C:\Users\R_Asad\Desktop\docker-python-app> docker images
IMAGE                ID                DISK USAGE  CONTENT SIZE  EXTRA
alpine:latest        25109184c71b      13.1MB      3.95MB
flask-docker-app:latest d51fcddbe139      1.6GB       410MB
nginx:latest         341bf0f3ce6c      240MB       65.8MB
rkbanu/mini_siem:latest f0d9fd8185d5      1.58GB      405MB
PS C:\Users\R_Asad\Desktop\docker-python-app> docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                               NAMES
5971184b2f85   flask-docker-app "python app.py"         15 minutes ago Up 15 minutes  0.0.0.0:5000->5000/tcp, [::]:5000->5000/tcp   happy_almeida
PS C:\Users\R_Asad\Desktop\docker-python-app>

```

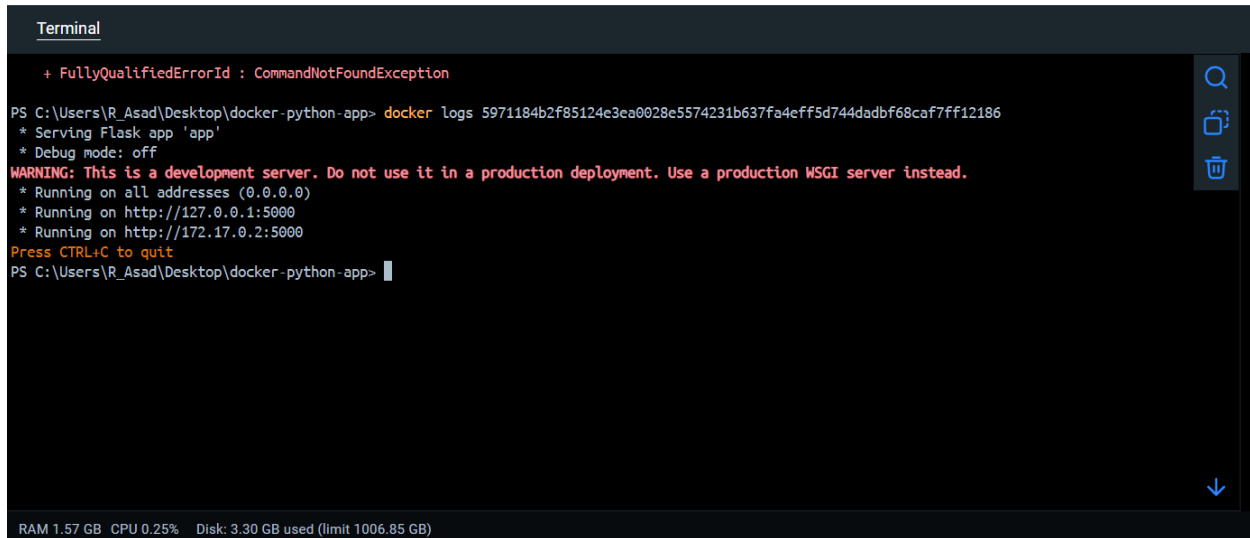
RAM 0.85 GB CPU 0.25% Disk: 3.30 GB used (limit 1006.85 GB)

Figure 7: Running Containers Verification

On the figure-7 show, once the conflict had been resolved, we use the command docker ps to confirm that the application container (aka. happy_almeida) is successfully up and running. From the "Status" row, you can get the information about the application being up for a while now.

4.5 Validating, Monitoring and Resources Analysis

Validations are done to confirm if the internal process is running as intended. Docker's logs tool enables streaming of the internal application output to the host terminal; an important requirement for troubleshooting microservices.



```
Terminal

+ FullyQualifiedErrorId : CommandNotFoundException

PS C:\Users\R_Asad\Desktop\docker-python-app> docker logs 5971184b2f85124e3ea0028e5574231b637fa4eff5d744dadbf68caf7ff12186
* Serving Flask app 'app'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://172.17.0.2:5000
Press CTRL+C to quit
PS C:\Users\R_Asad\Desktop\docker-python-app>
```

RAM: 1.57 GB CPU: 0.25% Disk: 3.30 GB used (limit 1006.85 GB)

Figure 8: Internal Application Log Messages

On the figure-8 shown, the log shows a regular start-up message from Flask, confirming that the server is running and gives the container's internal IP (usually 172.17.0.2). The mentioned IP belongs to Docker Bridge and cannot be accessed from the outer environment without port mappings.

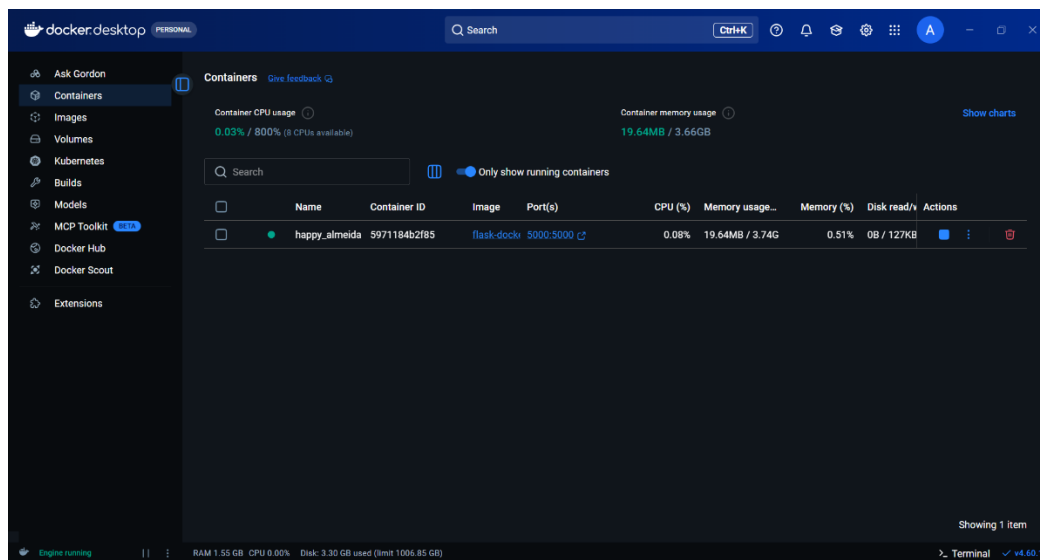


Figure 9: Telemetry & Resource Consumption

A key advantage of using containerisation is lightweight usage. On the figure-9, the Docker desktop UI displays real-time telemetry data. In this case, the Flask container consumed only 19.64 MB of RAM and nearly 0% CPU during the idle state. The mentioned resource consumption makes it possible for cloud

engineers to “bin-pack” hundreds of containers on one single physical server, something that would be impossible with VMs.

4.6 Advanced Cloud Features: Volumes, Networking, and Distribution

To progress beyond the simple isolation of one container, we have to look at issues concerning persistence, networking, and distribution through the cloud.

```
Terminal
docker: Error response from daemon: failed to set up container networking: driver failed programming external connectivity on endpoint nostalgic_
_lehmann (8b9da35a1f8ce19977a7dc5495dc8a086c2a52859a7954b77b81e0619b85c5ec): Bind for 0.0.0.0:5000 failed: port is already allocated

Run 'docker run --help' for more information
PS C:\Users\R_Asad\Desktop\docker-python-app> docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                               NAMES
5971184b2f85   flask-docker-app  "python app.py"         30 minutes ago Up 30 minutes  0.0.0.0:5000->5000/tcp, [::]:5000->5000/tcp  happy_almeida
PS C:\Users\R_Asad\Desktop\docker-python-app> docker stop 5971184b2f85
5971184b2f85
PS C:\Users\R_Asad\Desktop\docker-python-app> docker run -d -p 5000:5000 -v mydata:/app flask-docker-app
98c2130b0e1a5c5591f7fd142cb7c16f7b259af8851c4d5a854e7461c8031724
PS C:\Users\R_Asad\Desktop\docker-python-app> docker run -d -p 5000:5000 -v mydata:/app flask-docker-app
8223c701539a4ee8d0b5f97c4402c3f471f326d47bad1aab8ac562053e689945
docker: Error response from daemon: failed to set up container networking: driver failed programming external connectivity on endpoint dazzling_
euclid (b3b1c7f060ff6f902885b3be2875265b3bad22be360454d4379d4032251e11e1): Bind for 0.0.0.0:5000 failed: port is already allocated

Run 'docker run --help' for more information
PS C:\Users\R_Asad\Desktop\docker-python-app> docker volume ls
DRIVER      VOLUME NAME
local       Asad
local       mydata
PS C:\Users\R_Asad\Desktop\docker-python-app>
```

RAM 0.92 GB CPU 0.13% Disk: 3.30 GB used (limit 1006.85 GB)

Figure 10: Persistence Through Docker Volumes

On the figure-10, the container is transient, anything that gets written onto its filesystem will be lost when the container is destroyed. In order to avoid this, the use of a Docker Volume has been done by mounting it. This shows how data persistence works in a cloud setting by creating a persistent directory and linking it to the container.

```
Terminal
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                               NAMES
98c2130b0e1a   flask-docker-app  "python app.py"         9 minutes ago Up 9 minutes  0.0.0.0:5000->5000/tcp, [::]:5000->5000/tcp  eager_agnes
PS C:\Users\R_Asad\Desktop\docker-python-app> docker stop $(docker ps -q)
98c2130b0e1a
PS C:\Users\R_Asad\Desktop\docker-python-app> docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                               NAMES
7ea815e67992faa7e9470ca8288e5f763bf4f8e1196cc1ebb73f4c038a9fcf04
PS C:\Users\R_Asad\Desktop\docker-python-app> docker run -d --network=mynetwork -p 5000:5000 flask-docker-app
7ea815e67992faa7e9470ca8288e5f763bf4f8e1196cc1ebb73f4c038a9fcf04
PS C:\Users\R_Asad\Desktop\docker-python-app> docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
e5d1e5393b79   bridge   bridge    local
b59fd5c2869b   host     host      local
e49cfc5b418f   my-net    bridge    local
388c7bc3d78f   my-net2   bridge    local
7393f123657b   mynetwork bridge    local
e1fe5e2d9c3a   none     null      local
PS C:\Users\R_Asad\Desktop\docker-python-app> ^C
PS C:\Users\R_Asad\Desktop\docker-python-app> docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                               NAMES
7ea815e67992   flask-docker-app  "python app.py"         About a minute ago Up About a minute  0.0.0.0:5000->5000/tcp, [::]:5000->5000/tcp  conde
scending_cannon
PS C:\Users\R_Asad\Desktop\docker-python-app>
```

RAM 0.94 GB CPU 0.25% Disk: 3.30 GB used (limit 1006.85 GB)

Figure 11: Network Isolation and Bridging

On the figure-11, I use of the docker network create command creates a separate network, hence a private network. The creation of this separate network helps to automatically resolve the DNS between containers and allows for communication using names as opposed to IP addresses.

```
PS C:\Users\R_Asad\Desktop\docker-python-app> docker tag flask-docker-app asad322/flask-docker-app
PS C:\Users\R_Asad\Desktop\docker-python-app> docker push asad322/flask-docker-app
Using default tag: latest
The push refers to repository [docker.io/asad322/flask-docker-app]
aedb52583415: Pushed
79d5bd8a8d26: Mounted from rkbanu/mini_siem
795dbedde24d: Mounted from rkbanu/mini_siem
081ccf923272: Mounted from rkbanu/mini_siem
837c9d4e984c: Pushed
26dfe2fac1c4: Mounted from rkbanu/mini_siem
c9723aa529b0: Mounted from rkbanu/mini_siem
ee40e81ed03b: Pushed
91c91c91f1d2: Mounted from rkbanu/mini_siem
21f5ace1eefd: Pushed
89d573bf42b3: Mounted from rkbanu/mini_siem
latest: digest: sha256:d51fcddbe1399002a716a3441da588010cc9ae7e180b1d049130bfe6adfd9671 size: 856
PS C:\Users\R_Asad\Desktop\docker-python-app>
```

Figure 12: Image Distribution on a Global Scale

On the figure-12, I last test case to consider is the ability to run the containerized application anywhere through Docker Hub. After labeling the image locally using "docker tag" and then pushing it out using "docker push," one can access the software on a global level since all that the production machine has to do is to pull the latest image from the cloud.

4.7 Critical Analysis of Practical Results

As noted above, the implementation has succeeded in showing the full cloud-native lifecycle. Moving from a simple local script to a Docker image ready to run on the cloud revealed a number of important lessons. First, there was isolation since no changes to the Flask application have been made, and nothing has impacted the way the application operates. Second, there was immutability since the built image will not change its content anymore and therefore, the exact same version can be used both locally and on the server. Finally, scalability has been confirmed since the quick startup time of the container allows running numerous instances of it behind a load balancer.

To conclude, one may say that Docker offers the needed level of abstraction when it comes to developing and deploying applications. Moreover, the use of such concepts as volumes and networks adds even more benefits since it is now possible to achieve the needed stability and connectivity in order to implement cloud-based architectures.

5. FUNDAMENTAL IDEAS IN CLOUD COMPUTING

There are various architectural principles used in cloud computing to guarantee performance, scalability, reliability, and security. Some of these key principles include load balancing, scalability, and application gateways. These principles interact to provide functionality of traffic management, scaling, and secure

communication with the backend from the client side. It is critical to understand these principles in order to develop highly resilient cloud native applications.

5.1 Load Balancing

Load balancing is an essential feature of cloud computing whereby network traffic is distributed among various servers or computational resources. The main objective of load balancing is to ensure that one server is not overwhelmed by requests from other servers that are still underused. Through load balancing, performance improves, latency is minimized, and high availability is maintained.

In a cloud computing setting, load balancing can be achieved through the deployment of hardware or software-based load balancers. Various cloud computing platforms, including Amazon Web Services (AWS), Microsoft Azure, and Google Cloud, have automated load balancing capabilities that use algorithms such as round-robin, least connections, or IP hashing techniques. According to the AWS documentation, Elastic Load Balancing (ELB) redistributes application traffic among multiple targets like virtual machines, containers, or IPs to keep them available [6].

Another important function of load balancers is their contribution to fault tolerance. Should one server malfunction, load balancing will redistribute its tasks to other functioning servers, maintaining service availability.

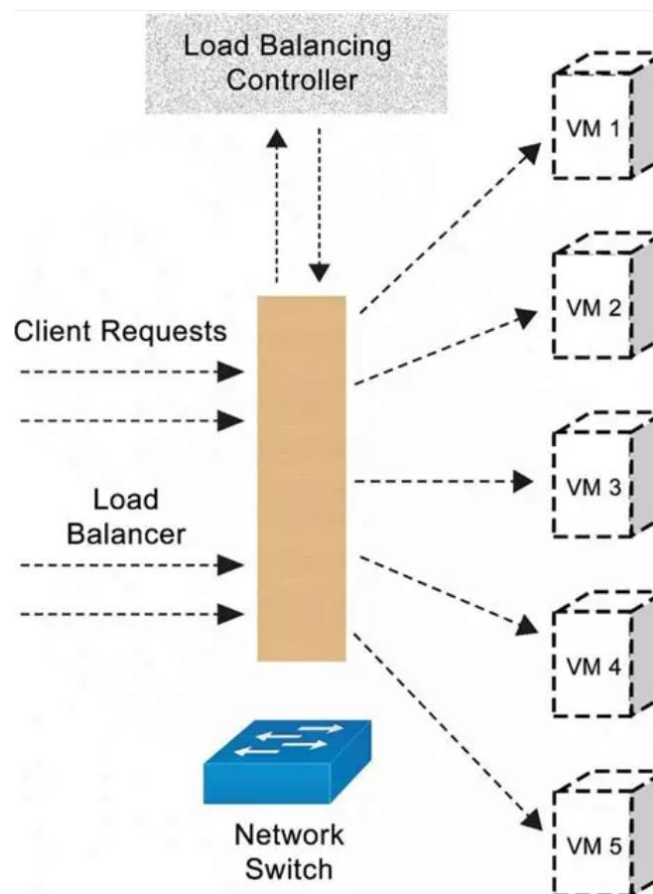


Figure 1: Cloud Architecture Load Balancing

This Figure-1 represents how the load balancer distributes client requests among several servers. Moreover, this diagram depicts the failover feature for the event where any of the servers goes down.

5.2 Scalability in Cloud Computing

Scalability is defined as the capability of the system to cope with varying amounts of loads without affecting its performance negatively. Scalability is one of the key features of cloud computing and the reason why cloud computing has become popular. Scalability is achieved in cloud computing through two main strategies: vertical scalability and horizontal scalability.

Vertical Scalability (Scaling Up)

The idea behind vertical scalability is to increase the hardware resources of the computer (such as CPU and RAM). However, the vertical scalability method has limitations since the resources of the computer cannot be expanded infinitely.

Horizontal Scalability (Scaling Out)

The horizontal scalability method is achieved by deploying additional computers or instances. This method is much more flexible, and it is commonly applied in cloud computing due to its flexibility. The horizontal scaling technique is directly linked to the load balancing technique because the latter distributes the traffic across the new instances. Elasticity, which is the capacity of the cloud system to change its resource allocation quickly, is an essential characteristic of cloud computing according to the principles of cloud computing defined by NIST [1].

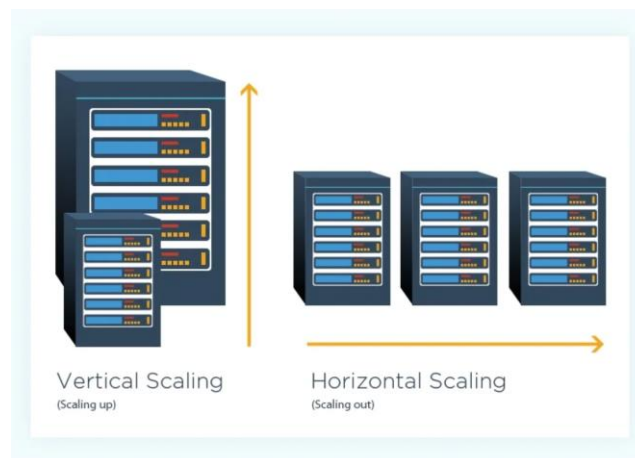


Figure 2: Horizontal & Vertical Scaling Architecture

Figure 2 shows the difference between vertical scaling (upgrading resources of one machine) and horizontal scaling (having more than one machine) to manage loads. Figure 2 highlights why horizontal scaling is preferred in cloud architecture because of its benefits.

Scalability helps maintain system responsiveness while handling different levels of activity, such as traffic surges when there is an online sale or content goes viral. It can also help save money through scaling down when traffic is low.

5.3 Application Gateways

The application gateways function in the application layer (Layer 7 of the OSI model) by implementing intelligent traffic routing and security measures, as well as handling request management. In contrast to load balancing mechanisms operating in other layers of the network, application gateways understand HTTP and HTTPS protocols, which makes them capable of making intelligent decisions about routing traffic.

The key functions performed by application gateways include:

- SSL/TLS termination: Decryption of the traffic arriving in encrypted form prior to being forwarded to the backend servers.
- Web Application Firewall (WAF): Security measures against threats such as SQL injection, XSS attack, and DDoS attack.
- Path-based routing: Routing the requests to different backend servers depending on the path of the request.
- Session affinity: Ensuring consistent routing of traffic within a user session.

Based on information provided by Microsoft Azure, application gateways can increase the level of security and performance of web applications due to their ability to manage traffic efficiently [4].

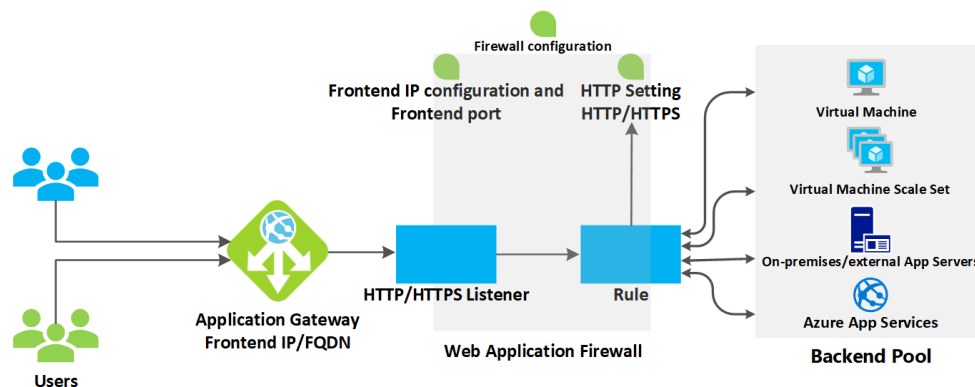


Figure 3: Application Gateway Structure

This figure-3 demonstrates the process of passing client requests through an application gateway and to the backend server.

5.4 Relation between Load Balancing, Scalability, and Application Gateway

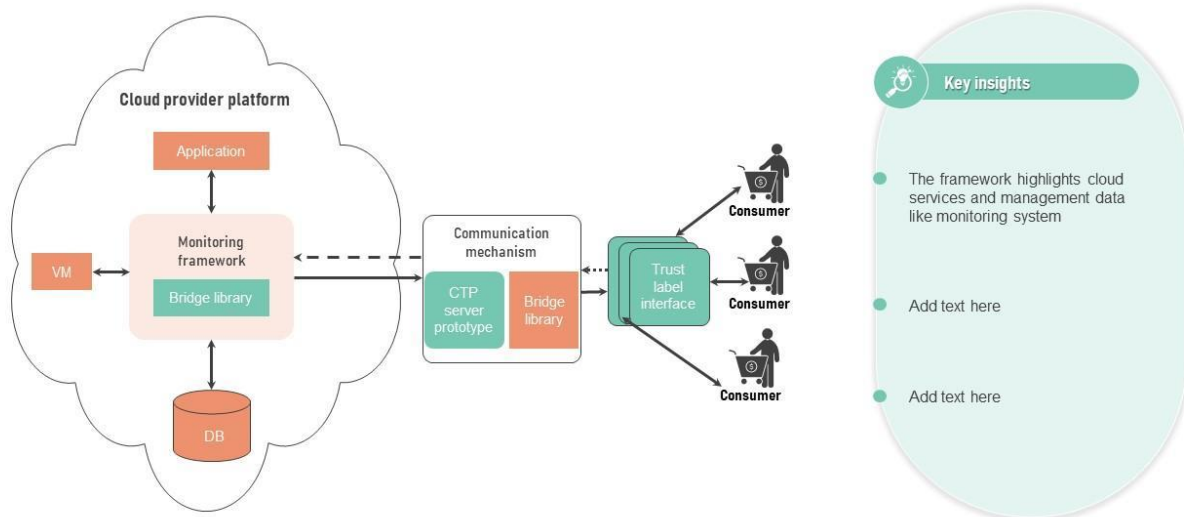
Load balancing, scalability, and application gateways are three interrelated elements of cloud architecture that make up a multi-layered system providing high reliability, performance, and protection.

While load balancing takes care of distributing the traffic, scalability helps to scale the number of instances according to the current needs. An application gateway enriches the overall framework with smart traffic routing and other security mechanisms at the application level.

Let us take the case of an e-commerce portal that uses the cloud to host its business process. When this portal becomes popular after launching a special offer for its customers, the cloud-based system scales horizontally by creating extra instances. Then, the load balancer manages to spread the incoming traffic among all those instances, and the application gateway makes sure that the requests are securely processed.

Cloud system integration architecture design

This slide highlights an architecture framework of cloud system integration to deliver accurate information to consumers. It includes key components such as cloud provider platform, application, monitoring framework, database, communication mechanism, bridge library, etc.



This slide is 100% editable. Adapt it to your needs and capture your audience's attention.

Figure 4: Integrated Cloud Architecture Framework

The following diagram on figure-4 represents the interaction between load balancers, scalable backend services, and application gateways in a cloud environment.

5.5 Significance in Current Cloud Technologies

The mentioned above cloud technologies are crucial for designing modern distributed cloud systems. These technologies include load balancing to ensure balanced network traffic, scalability to provide performance during peak load times, and application gateway capabilities.

As for their use together, it enables providing services on clouds without significant downtime, high speed, and high levels of security. The usage of the described solutions is relevant for enterprise-level apps, microservices architecture designs, and global clouds like streaming, online banks, and shopping websites.

In accordance with the research of cloud computing, all of the described aspects create a basis for cloud architectures that are resilient and suitable for digital transformation [6][8].

6. CONTAINERIZATION VS. SERVERLESS COMPUTING

Serverless computing and containers are examples of paradigms adopted in contemporary cloud application deployments. The former is characterized by the absence of any infrastructure management, while the latter implies a more structured approach to deployment of applications but still provides benefits in terms of flexibility and scalability compared to the traditional models. Despite differences in specifics of deployment, both methods share common goals, including simplification of processes involved in setting up an environment for running applications and better use of resources.

Serverless computing is widely used via platforms like AWS Lambda, Azure Functions, or Google Cloud Functions. This method implies that no infrastructure management takes place as servers, their provisioning, scaling, and even runtime are handled by the cloud provider automatically. The idea is to deploy functions to the cloud and wait until certain events trigger their execution. Resources will be allocated to run these functions only at the moment when needed, which allows avoiding costs for unused resources.

Nevertheless, there are some drawbacks of serverless computing. For instance, it involves "cold start latency," which implies that there could be delays during the first execution of the function due to the time taken by the environment to initialize itself. Moreover, serverless computing imposes certain limitations on execution duration, available memory, and runtime environment configuration, making it inappropriate for handling long tasks and processes, stateful applications, and those requiring low-level control [8]. Although serverless architecture is quite popular in various scenarios such as API creation, real-time file processing, and event-triggered automation.

On the other hand, containerization technology is more appropriate when deploying applications to production. Containerization enables the packaging of the software alongside all the dependencies it requires into a portable package. The process does not involve complete abstraction from the infrastructure layer, thus permitting developers to customize the runtime environment using orchestrators such as Docker and Kubernetes [5]. Therefore, the technology is highly recommendable in microservice-based architectures, whereby applications can be divided into independent components.

The second significant benefit of containerization lies in the proper balance that it strikes between isolation and performance efficiency. As demonstrated in Figure 2 above (Comparison of Virtual Machines and Containers), containers leverage the same operating system kernel as the host machine. This approach leads to reduced overhead costs compared to virtual machines, thereby promoting faster boot times, efficient use of resources, and scalability.

When discussing scalability, it is important to note that serverless computing and containerization support horizontal scaling through different means. While the serverless computing paradigm automatically scales functions based on the workload without requiring any actions from users, the containerization approach requires orchestration tools like Kubernetes to implement the required scaling policies, load balancing, and service discovery. The figure below (Figure 12) provides a conceptual representation of serverless computing architecture.

There is a difference in terms of security and portability as well. Serverless computing minimizes the exposure due to lack of direct server access. On the other hand, serverless computing makes the system dependent on the vendor ecosystem, which can cause problems related to vendor lock-in. Portability

remains an advantage in containerization compared to serverless computing, while security measures need to be taken seriously [4].

To conclude, serverless computing is a better choice for lightweight workloads, whereas containerization will be more appropriate in complex scenarios that require control and persistence. In any case, both techniques complement each other rather than being mutually exclusive. It is possible to combine them to obtain maximum benefits from cloud-based infrastructure.

7. MOTIVATORS FOR THE GROWTH OF CLOUD COMPUTING

The fast adoption of cloud computing has been caused by several technology, business, and organizational factors which have completely revolutionized how today's IT infrastructure looks like. These factors have prompted organizations to shift towards more flexible and scalable on-demand cloud systems that can increase efficiency, flexibility, and worldwide access. Firstly, traditional IT infrastructure requires high initial expenses on servers, storage systems, network hardware, and dedicated data center facilities. At the same time, cloud computing works on the "pay-as-you-go" principle when users pay solely for what they use. It means that CAPEX is reduced dramatically, and OPEX becomes the basis of spending. As demonstrated by the cloud economics model proposed by Microsoft Azure, organizations may decrease their IT expenses due to elastic cloud resources [7].

Furthermore, one of the main benefits of cloud computing lies in scalability and elasticity. Cloud environments support dynamic resource allocation based on the load. It refers especially to the applications that are subject to the fluctuation in traffic. By increasing the number of virtual machines or containers horizontally, organizations make sure that their system works efficiently during the peak times.

Availability and global access are equally significant driving factors in adopting cloud computing. Cloud computing solutions are accessible through any part of the globe with an internet connection, which allows for remote work, global collaboration, and deployment of systems. In today's digital workplaces, availability and accessibility have become particularly significant due to geographical distribution.

Innovative speed and development have always been critical drivers. The use of cloud computing solutions results in a fast and effective development process because there are available pre-made services, automated processes, and other DevOps solutions. Continuous integration and deployment are actively used in cloud computing environments and allow for shorter release cycles.

Finally, disaster recovery and reliability are important motivation aspects in cloud computing. By using redundant infrastructure in different geographic locations, cloud service providers ensure high availability and reliability of their services. Backup options are also available in the cloud and help prevent loss of data and disruptions of the system operations. Based on AWS cloud resilience guidelines, multiregion redundancy increases the reliability of applications [8].

THE CLOUD COMPUTING ADOPTION MODEL

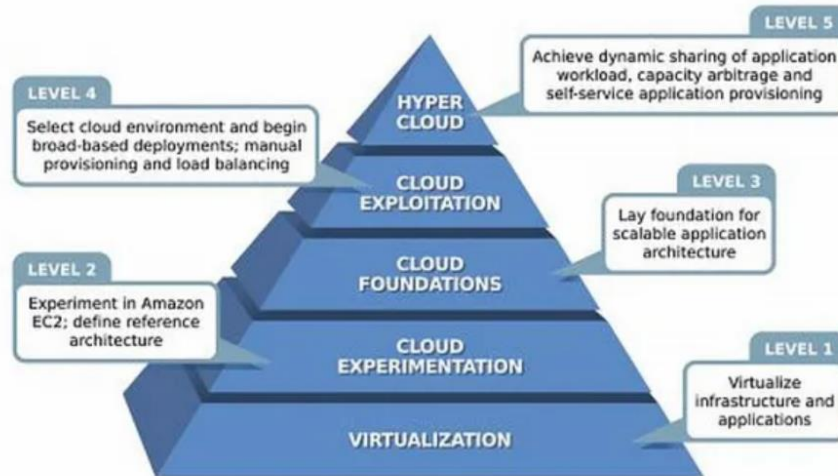


Figure 1: Motivation Factors for Adopting Cloud Computing Solutions

The above figure-1 highlights the major drivers behind the growth of cloud computing, including cost effectiveness, scalability, worldwide availability, fast innovations, and disaster recovery. All these factors combined are responsible for the popularity of cloud computing in various spheres of activity.

8. CONCLUSION

The advent of cloud computing technology has revolutionized modern-day computing technology by providing scalable, flexible, and cost-effective computing paradigms. This paper has analyzed the underlying technology behind cloud computing, specifically virtualization and containerization technologies, and highlighted their contribution towards the evolution of cloud native applications. The virtualization technology described in Section 2 is still the primary building block of cloud infrastructure because it allows the creation of multiple virtual machines that can run independently of each other on a single physical machine using hypervisor technology [2]. Nevertheless, due to its high overhead costs, newer and lighter alternatives such as containerization technologies have evolved.

As mentioned earlier, containerization technology can be utilized effectively via Docker platforms, which share the same host machine kernel but provide process-level isolation for applications. As shown schematically in Figure 2 (Comparison between VMs and Containers), containerization technologies offer significantly lower overheads compared to virtualization technologies while offering better portability and faster deployment capabilities. Such technologies have played a crucial role in the development of microservice architectures implemented by Netflix and Amazon among others.

The real-life implementation of Docker, explained in Section 4, shows how container images, volumes, and networks all contribute towards modern deployment practices. Figure 3 (Build Process for Docker Images) and Figure 7 (Overall Docker Deployment Process) illustrate the entire cycle of containerised applications, starting with the building of container images and ending with efficient deployment [5].

This allows for fast development cycles and consistency across platforms, which is a key component of the DevOps methodology.

Moreover, other cloud-native practices, such as load balancing, horizontal scaling, and application gateways, are integral to achieving high levels of system efficiency. Figure 8 (Distributed Load Balancing) and Figure 9 (Horizontally Scaled System Architecture) demonstrate how distributed traffic management and elastic resource allocation can be used to manage fluctuating loads on the system [6]. Application gateways, depicted in Figure 10 (Application Gateway Architecture), add an extra layer of security and route control by working at the application level, offering functionalities like SSL termination and firewall capabilities.

Comparing serverless computing against containerisation also further illustrates the dynamic aspect of cloud architecture. Although serverless computing platforms like AWS Lambda help minimize operations by using self-scaling capabilities and paying on demand, they also have limitations such as cold start and limited execution control [8]. On the other hand, containers are highly flexible and provide execution control, which makes them more appropriate for running sophisticated applications.

In conclusion, cloud computing is continually developing because of the efficiency it brings in reducing costs, increasing accessibility, and improving innovation [7]. Together, virtualization, containerization, and cloud infrastructure build a strong foundation upon which distributed systems are constructed.

REFERENCES

- [1] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," NIST, 2020.
- [2] M. Rosenblum, "Virtual Machine Monitors," IEEE, 2019.
- [3] D. Merkel, "Docker: Lightweight Linux Containers," Linux Journal, 2014.
- [4] IBM, "Container Security Guide," 2023.
- [5] Docker Inc., "Docker Documentation," 2025.
- [6] AWS, "Elastic Load Balancing," 2024.
- [7] Microsoft Azure, "Cloud Cost Management," 2024.
- [8] AWS, "Lambda Developer Guide," 2025

PRESENTATION DETAILS AND LINK

Presentation File Name

Cloud Engineering Principles and Practice, Presentain Record. (Md. Asaddoz Jaman_2526322)

Presentation Links

YouTube Link: https://youtu.be/OYxHSt_c68I

Drive Link: <https://drive.google.com/file/d/11yJKFszWl6u48toeZv-5lkvCsaRs2usm/view?usp=sharing>