



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

Module Title:
Cloud Engineering Principles and Practice

Module Code:
ACCA7027

Assignment Title:
Assignment

Student Name:
MD Mahadi Hasan

Student ID:
2505591

Programme:
MSc in Software Engineering and Artificial Intelligence

University Name:
University of Wales Trinity Saint David (Swansea Campus)

Lecturer:
Mark Huntly

Submission Date:
11th May 2026

Table of Contents

Abstract

1. Introduction

2. Principles of Virtualisation

2.1 Virtualisation of Shared Hardware

2.2 Advantages of Virtualisation

2.3 Limitations of Virtualisation

3. Principles and Benefits of Containerisation

3.1 How Containerisation Works

3.2 Benefits of Containerisation

3.3 Virtual Machines and Containers Compared

3.4 Limitations and Security Concerns

4. Dockerfile and Image Creation

4.1 Example of Dockerfile

4.2 Explanation of Dockerfile Instructions

4.3 Building the Docker Image

5. Running a Container

6. Images and Publishing Images to Docker Hub

6.1 Publishing an Image

6.2 Importance in Cloud Deployment

7. Container Volumes

7.1 Creating and Using a Volume

7.2 Benefits of Volumes

8. Container Networking

8.1 Creating a Docker Network

8.2 Networking in Microservices

8.3 Security Considerations

9. Cloud Computing Concepts: E-commerce Application

9.1 Load Balancing

9.2 Scalability

9.3 Application Gateway

10. Relationship Between Load Balancing and Scalability

11. Benefits of Combining Load Balancing, Scalability and Application Gateway

12. Driving Forces Behind Cloud Computing

12.1 Cost Efficiency

12.2 Flexibility and Agility

12.3 Scalability and Elasticity

12.4 Reliability and Disaster Recovery

12.5 Global Accessibility and Innovation

13. Serverless Computing: Functions as a Service

13.1 How Serverless Works

13.2 Benefits of Serverless Computing

13.3 Limitations of Serverless Computing

14. Comparison Between Containerisation and Serverless Computing

15. Conclusion

References

Abstract

Cloud computing has transformed the development, deployment and management of modern-day software applications for organisations. Organisations were required to purchase physical servers, network equipment and storage devices and employ IT experts to operate data centres using traditional IT equipment. This model tends to require large initial investment and inflexibility as businesses invest in infrastructure before it is needed. Cloud computing, on the contrary, provides the capacity to obtain computing services, analytics, networking, databases, storage, and servers, on demand from the Web. It enables organisations to scale resources as per the actual workload demand rather than having extra infrastructure costs and allows for ease of operations.

This report addresses the concepts and real-world implementation techniques of cloud engineering, including virtualisation, containerisation, Docker and design of cloud-native applications. One of the key technologies clouds is discussed in terms of virtualisation, which is an enabling feature that enables multiple virtual machines to be running on one physical machine. The study of containerisation as an alternative solution which is lighter and more mobile, allowing modern development methods such as microservices and DevOps. Samples of Docker usage are provided: Creating an image, running a container, publishing an image to Docker Hub, persistent storage using Docker volumes, and Docker networking.

This report also explores the three basic concepts of the cloud computing – load balancing, Application gateways and scalability. The concepts are presented in the context of an e-commerce application scenario, as e-commerce traffic is sometimes highly dynamic, particularly during sales campaigns or during seasons. The report explains what is meant by load balancing, scalability of adding/removing capacity and application gateways, which provide intelligent routing and security. Finally, the report reviews some of the main challenges in the process of cloud adoption including cost, flexibility, resilience, availability, and innovation. Also, serverless is contrasted with containerisation to demonstrate the different modern options for implementing cloud-based web services. The report concludes that, overall, the technologies of Docker, containerisation, cloud services, and serverless functions are key in the construction of modern, scalable, resilient, and efficient applications.

1. Introduction

Cloud computing is one of the greatest advancements in modern information technology as it alters the way in which businesses utilize computing resources and the method in which they pay for them. The organizations do not need to build their own infrastructure; they can lease computing power from cloud service providers that they can access via Internet. Cloud computing is defined by the National Institute of Standards and Technology (NIST) as a model of convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort [1]. The essence of this definition is that clouds include many key features such as on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service.

Prior to the widespread use of cloud computing, a lot of organisations were using conventional on-premises infrastructure. This involved the purchase of physical servers, manual software installation, setup of networking hardware and the handling of security and backup management tasks. This type of infrastructure was costly and hard to expand. If an organisation anticipated that it would have to scale to a certain level during a busy time, for instance, then it needed to be equipped with hardware that would be sufficient. If demand is less than anticipated, resources would not be used. The system might become slow and unavailable if the demand is higher than expected. Cloud computing is solving this problem by enabling the organisation to provision resources on an as-needed basis and release them when no longer needed [2].

One of the key technologies that enabled the emergence of cloud computing was virtualisation. It enables multiple operating systems and applications to run on a single physical server. This makes hardware use more efficient and enables multiple customers to securely use the same physical infrastructure provided by the cloud service. Virtual machines are, however, relatively weighty since each one needs a full operating system. The deployment cycle of software has become faster, agile, and microservices are gaining popularity, and so is containerisation. The concept of containers is to bundle an application and its dependencies into a light-weight isolated environment that shares the host operating system kernel [5].

Docker is one of the most popular containerisation platforms. It enables developers to create application environments with Dockerfiles, create reusable images, execute containers and share images in Docker registries like Docker Hub. The common issue of “it works on my machine” is minimized by packaging the application and its dependencies together in Docker. This renders Docker very useful for cloud deployment, testing, continuous integration and continuous delivery pipelines.

The focus of this report is on Docker, cloud services and modern deployment strategies. It covers virtualisation, containerisation, provides real-life examples of Docker and

explore cloud concepts such as load balancing, scalability, and application gateways. It also explores the reasons for continued growth of Cloud Computing and the differences between Containerisation and Serverless Computing.

2. Principles of Virtualisation

Virtualisation is the technique used to create virtual versions of the physical computing resources. The resources can be servers, operating systems, storage devices, network components, and desktop environments. Server virtualisation uses software called hypervisor to run several virtual machines on one physical machine. Virtual machines are independent computers with their own operating system, applications, memory allocation, storage, and network configuration [3].

The hypervisor oversees managing access to the underlying hardware. Assigns CPU resources, memory, storage, and network resources to each virtual machine. There are two main types of hypervisors. A Type 1 hypervisor, or a “bare metal” hypervisor, is installed directly on the physical hardware. Some examples are VMware ESXi, Xen and Microsoft Hyper-V. They are often employed in enterprise and cloud data centre environments due to their robust performance and isolation capabilities. A Type 2 hypervisor is the one that is installed on top of an existing operating system. These include Oracle VirtualBox and VMware Workstation. Type 2 hypervisors are good for testing, education, and development, but typically are more overhead than Type 1 hypervisors.

2.1 Virtualisation of Shared Hardware

An example of virtualisation is a cloud data centre server. If it wasn't virtualised, a physical server could be used for one application. This can be an inefficiency if the app consumes a minimal amount of server CPU, memory or storage. Virtualisation technology lets multiple virtual machines run on a single server. The web server could be a Linux based application, the application server could be a Windows application, and the database service could be a Linux based application. The VMs run independently of one another but share the same physical hardware.

This is a shared hardware model fundamental to cloud computing as cloud providers must share resources from large data centres with many customers. Customers need not be concerned with the precise physical machine that is running their workload. Rather, they ask for virtual resources like virtual CPUs, memory and storage. The provider shares these resources among multiple customers or tenants. The benefit of this model is that the provider can better utilize the hardware in a multitude of customers and workloads [4].

Isolation is also possible with virtualisation. When one VM crashes or has an application error, other VMs on the same host will keep running. It is useful in multi-tenant environments in which various customers share infrastructure. This also

facilitates testing and development since developers can develop isolated environments without having to install separate physical machines.

2.2 Advantages of Virtualisation

There are several advantages to virtualisation. First, it increases the utilization of hardware. Organisations can run multiple workloads on the same server, instead of running one workload on one physical server. This minimizes the waste and hardware recoverability. Secondly, virtualisation can help to cut the footprint of physical infrastructure. There is less of a need for physical servers, thus less electricity consumption, cooling and data centre space.

Third, Virtualisation makes provisioning easier. It takes typically less time to create a virtual machine than to buy and install a physical machine. Virtual machines can be deployed in cloud platforms in minutes. Fourth, Virtualisation enhances Disaster Recovery. Virtual machines are more easily backed up, cloned, snapshotted and migrated than physical servers. Workloads can be migrated or restored to other hosts in the event of a physical host failure.

Virtualisation also provides increased flexibility. Different OS and software stacks can be used on the same physical server. This is helpful when applications have differing compatibilities. A legacy app might need an old operating system, for instance, or another service might need a newer version of Linux.

2.3 Limitations of Virtualisation

Virtualisation is a great technology, but it is not without its problems. The biggest drawback is overhead. Virtual machine's need a complete guest OS. This will result in multiple VMs on the same host will be consuming large amounts of memory, storage and CPU resources. Startup time is also slow as the entire operating system loads. This results in less efficiency for some modern application workloads compared to virtual machines [6].

Another restriction is that of complexity in its management. For large environments, it is necessary to track and monitor hundreds or thousands of virtual machines, patch them, backup and manage them, provide access control, and allocate resources. However, virtual machines can be hard to manage and secure if not handled properly. Many unused or forgotten virtual machines (VMs) can cost and risk the business in what is known as VM sprawl.

However, from an economic point of view, virtualisation is still crucial for cloud computing because it allows for good isolation and flexibility in infrastructure abstraction. But it's not always the optimal solution for cloud-native apps that move quickly. Containerisation can be more appropriate for certain applications that require quick start-ups, regular deployments, and fast scalability.

3. Principles and Benefits of Containerisation

Containerisation is a lightweight approach to packaging and running applications in isolated environments called containers. The difference between virtual machines and containers is that the containers do not contain a complete guest operating system. Rather, they run the same host operating system kernel but each with separate application processes, file systems and dependencies [5]. This makes containers more compact and quicker than virtual machines. Typically, a container will contain application code, runtime libraries, configuration files, and any necessary dependencies. This implies that the application incorporates the same behaviour in various settings, including during development on the developer's laptop, testing on a cloud-based system and production on a cloud-based system. Docker popularised containerisation with its easy-to-use image and registry-based methods for creating, running and sharing containers [5].

3.1 How Containerisation Works

Operating system level isolation is used with containers. In Linux, technologies like namespaces and control groups enable the separation of processes and control over resources. Namespaces are used to separate containers by isolating process IDs, networking, file systems and users. Control groups can help regulate and monitor the consumption of resources like CPU and memory. This creates an "out-of-the-box" experience for applications that are sharing the same kernel of an operating system. This design is the rationale behind why containers are faster than virtual machines. Containers do not require the loading of a full operating system, so that they can be launched in seconds or less. They're also smaller since they just consist of the application and essential dependencies instead of a whole OS. This renders containers useful in such scenarios as apps that need to scale up rapidly.

3.2 Benefits of Containerisation

A great advantage of containerisation is portability. If the machine with Docker installed has the same architecture and platform as the machine from which the Docker container was created, then the first machine can run the second machine's Docker container. This helps to minimize the issue of environment inconsistency. Same container image for developers, testers and operations teams minimising deployment errors. The other advantages include efficiency. Containers are more efficient than virtual machines, and they allow for the running of more instances of applications on the same infrastructure. This can help to lower infrastructure costs and enhance performance. Containers are also known for their ability to be built, tested and started rapidly, which allows the deployment of containers faster. Microservices architecture also lends itself to containerisation. A microservices system breaks up a large application into smaller services that can be developed, deployed and scaled separately [8]. For instance, an e-commerce system can have services such as user

authentication, product search, shopping basket, payment processing and order management. These services can be packaged as containers, and the changes can be applied without redeployment of the entire system.

3.3 Virtual Machines and Containers Compared

Feature	Virtual Machine	Container
Operating system	Full guest OS per VM	Shared host OS kernel
Size	Large	Small
Startup time	Slower	Faster
Resource usage	Higher	Lower
Isolation	Strong OS-level isolation	Process-level isolation
Best use case	Full OS isolation and legacy systems	Cloud-native application deployment

Table 1: Virtual Machines and Containers Compared

Virtual machines are not completely replaced by containers. Virtual machines offer greater isolation as each virtual machine runs its own operating system. That's useful when workloads need to be separated or to use different operating system kernels. Containers are ideal for lightweight application deployment, quick scaling and microservices.

3.4 Limitations and Security Concerns

There are challenges to the containerisation also. The host kernel is shared by containers, so that can be a kernel vulnerability that impact several containers. poorly configured containers can have excessive privileges, which can imply security threat. For instance, running containers with root access and exposing unnecessary ports and services can compromise security. Secure container practice requires using trusted images, scanning for vulnerabilities, limiting privileges and regularly updating images. Managing containers at scale can get complex. It's easy to run a handful of containers by hand but for hundreds of containers, you'll need to use orchestration tools like Kubernetes. Kubernetes offers the ability to schedule, scale, network, discover, and self-heal services, but it brings in the complexity [7]. Hence, even though containerisation enhances deployment efficiency, it must be managed, monitored and secured properly.

4. Dockerfile and Image Creation

A Dockerfile is a text file that contains the instructions that are used to build the Docker image. These instructions are used to build images using Docker. According to the Docker documentation, a Dockerfile is a text file with instructions to build an image [2]. This is particularly significant since Dockerfile take the tedious work of environment preparation out of the equation. A Docker image is a template that can be used repeatedly, a Docker image includes the application, its dependency packages, its configuration and instructions on how to run a Docker container. Images are fixed, that is they are not changed after creation. A container is created from an image when it is run. This separation of image & container is significant - the image is the packaged template; the container is the running instance.

4.1 Example of Dockerfile

```
FROM nginx:latest  
  
COPY index.html /usr/share/nginx/html/index.html  
  
EXPOSE 80
```

A basic nginx based web server build with Dockerfile. It is appropriate for visualizing the general Docker process as it uses a base image, an HTML file and a port exposed for the browser to communicate with.

4.2 Explanation of Dockerfile Instructions

FROM nginx: latest specifies that the base image should be the official Nginx image. A base image is the foundation the image for the container. Nginx is already installed and running to deliver web pages in this situation. This is the default 'latest' tag that is present in the image repository, but in a professional environment a version tag might be more repeatable.

The second line copies the local index.html file to the default Nginx web directory /usr/share/nginx/html/index.html within the image. This means that Nginx will display the custom HTML page when the container is running.

The third line EXPOSE 80, indicates that the container is listening on port 80. It doesn't automatically publish the port to the host; port publishing is performed when the container is run with the -p option. But EXPOSE is helpful in that it tells users and tools what port the network is intending to use.

A simple index.html file could contain:

```
<!DOCTYPE html>

<html>

<head>

  <title>Cloud Engineering Principles and practice</title>

</head>

<body>

  <h1>Welcome to my Docker Web Application</h1>

  <p>This page is running inside a Docker container.</p>

</body>

</html>
```

4.3 Building the Docker Image

The Docker image can be built using:

```
docker build -t my-web-app .
```

The docker build command is used to read the Dockerfile and create an image. The -t my-web-app option provides a readable name for the image. The dot at the end indicates that the current directory is the source of the build context. The Dockerfile and index.html should be in the build context.

This is significant because it ensures re-entering the environment. The Dockerfile specifies the complete setup, which is not done manually on each server, but rather via Docker. This helps to ensure consistency of infrastructure and to minimise configuration drift.

5. Running a Container

After the image has been built, it can be executed as a running container using:

```
docker run -d -p 8080:80 my-web-app
```

docker run is used to create and launch a container based on the image. The -d option runs the container in detached mode, that is it runs in the background. Using the -p 8080:80 option will bind port 8080 on the host machine to the port 80 in the container.

This allows the web application to be accessed from a browser using:

```
http://localhost:8080
```

In this example, a full deployment process is shown. A developer writes an HTML file, writes a Dockerfile, builds the image, runs a container, maps a port and accesses the application via a browser. The web server and application files are contained in the image, which is the main reason Docker has a faster deployment time than traditional deployment.

The following command can be used to check running containers:

```
docker ps
```

To stop the container:

```
docker stop <container_id>
```

This workflow is a good example for the assignment video demonstration, as it is practical and makes Docker's purpose very clear. It illustrates packaging, running and accessing an application. It also is a chance to help the users understand the distinction between an image and a container: the image is the blueprint; the container is the active running instance.

Docker enhances consistency and speed from a critical standpoint, but for production there are other things to consider. Images need to be scanned for vulnerabilities, unnecessary packages need to be avoided, and containers should not run with abundance of privileges. Also set up logs, monitoring, restart policies and resource limits to support production reliability.

6. Images and Publishing Images to Docker Hub

Container images can be registered and shared using container registries. The registry is a system for storing and distributing container images. Docker Hub is a popular public registry where people can publish, download and share images [2]. In professional settings, private registries like Amazon Elastic Container Registry, Azure container Registry or GitHub container Registry can be used.

6.1 Publishing an Image

The following commands can be used to publish an image:

```
docker login
```

```
docker tag my-web-app username/my-web-app:v1
```

```
docker push username/my-web-app:v1
```

The docker login command logs in the user to Docker Hub. The docker tag command provides the image with a repository name and version tag. A docker push command will push the image to the registry.

The use of version tags is important. For instance, v1, v2 and v3 may be various versions of the app. This will enable teams to be sure of which version of the image is deployed. It also supports rollback. In the event of failure with the new version, the team can then return to the previous known working version.

6.2 Importance in Cloud Deployment

Images can be published to enable DevOps and continuous deployment. A standard pipeline could create an image from source code, execute automated tests on the image, upload the image to a registry, and deploy it to a cloud platform. This assures the consistency of the used tested artefact for production. It also enables increased collaboration as everyone involved in the development, testing and deployment can have access to the same image.

But the public registries should be employed with care. Do not include sensitive data like passwords, API keys, private certificates and database credentials in images. Potential vulnerabilities should be scans for images, as well. Business registry organizations often opt to use private registries to offer better access control for business applications.

7. Container Volumes

The term ephemeral or stateless is often used to describe containers, since data within containers can be lost if the container is removed. This is OK for some workloads, but not for applications requiring persistent data. Docker volumes overcome this problem by enabling the addition of persistent storage managed by Docker [10]. According to the Docker documentation, volumes are a persistent data store for containers.

7.1 Creating and Using a Volume

A Docker volume can be created using:

```
docker volume create webdata
```

The volume can then be mounted into a container:

```
docker run -d -v webdata:/usr/share/nginx/html nginx
```

In such an instance, webdata will be the Docker volume and /usr/share/nginx/html will be the directory within the container. The mounted directory keeps the data it contains after the container stops running.

7.2 Benefits of Volumes

There are several advantages to using volumes. The first thing they do is to decouple data from the lifecycle of the container. This implies that a container can be erased and created once more without losing valuable information. Secondly, volumes provide

backup and migration as they store data outside of containers. Thirdly, you can share volumes from one container to another as necessary.

For example, a database container. If a MySQL or PostgreSQL database is using the container only to store its data, then the container deletion may cause the database to be deleted. A volume mounted will have database files persist. For instance, a website that takes uploaded files from users. Any uploaded files must be kept in a persistent location and not inside a temporary container layer.

Persistent storage is particularly useful in cloud environments as containers can be rescheduled and replaced often. If the volumes are not provided, along with cloud storage services, important data can be lost. But planning of volume management is also necessary. Backup, encryption, access control and storage performance are all things organisations will need to consider.

8. Container Networking

Docker networking allows containers to communicate with each other, the host system and external services. Networking is critical since most applications today are not single process applications. Most applications consist of several parts, including the front-end services, the back-end APIs, databases, caches and message queues [11].

8.1 Creating a Docker Network

A user-defined Docker network can be created using:

```
docker network create app-network
```

A container can then be attached to that network:

```
docker run -d --name web --network app-network nginx
```

The web container can communicate with other containers attached to the same container network by using its container name. This is helpful since it does not hard code the IP address and is useful if IP addresses change upon container restart.

8.2 Networking in Microservices

In a microservices-based e-commerce application, different containers may provide different functions:

Container	Function
Front-end container	Displays the user interface
Product service container	Manages product information
Basket service container	Handles shopping basket data
Payment service container	Processes transactions
Database container	Stores application data

Table 2: *Microservices-Based E-commerce Container Functions*

These components can communicate with each other using Docker networking. For instance, the front-end container can invoke the product service, and the product service can make a query to the database. A user-defined network provides better isolation and simplifies communication between services.

8.3 Security Considerations

Networking is also significant with regards to security. Not all services should be available on the public internet. In general, the database should not be accessible from outside applications but should be accessed only by backend services. Attack surfaces are diminished with proper network design. Ports should only be published when necessary and internal services should remain on private networks.

As it gets bigger, the complexity of container networking increases. Platforms like Kubernetes offer service discovery, ingress controllers, network policies and load balancing. These tools can be very powerful, but they need to be well configured and monitored.

9. Cloud Computing Concepts: E-commerce Application

The idea of cloud computing concepts is best illustrated in a scenario such as an e-commerce platform that must face the changing demands, protect the data of the customers and at the same time be available. Product pages, search, shopping baskets and payment services must react fast when a customer clicks on them. Slow or unresponsive response time and downtime from the site can result in loss of revenue and customer confidence.

9.1 Load Balancing

Load balancing distributes incoming traffic over several application instances or servers. AWS Elastic Load Balancing automatically distributes incoming traffic across targets like instances, containers and IP addresses, monitors registered targets and routes incoming traffic to healthy targets [12].

Load balancing is crucial for an e-commerce site especially on Black Friday when there is a surge in traffic or when there are promotions. When thousands of users are using the site concurrently, a single server could get overwhelmed.

A load balancer distributes the workload of requests to multiple servers to prevent any one server from being overwhelmed. Load balancers also can do health checks. When one server becomes unhealthy, it is no longer able to receive traffic from the load balancer, and it redirects traffic to healthy servers. This will make it more fault-tolerant and less prone to downtime. Multi-availability-zone deployments can also be supported by load balancing to enhance resilience in the event of a problem in one zone.

9.2 Scalability

Scalability is the ability of a system to add or remove resources as needed. There are 2 types of scaling, vertical scaling and horizontal scaling. Vertical scaling involves scaling up an existing server, for example by adding more CPU or RAM. Horizontal scaling involves adding more servers or instances of the application.

Horizontal Scaling is used in cloud environments due to flexible growth. In an e-commerce application, for instance, the number of application containers running during regular traffic can be two but can rise to ten during a big sale. If there is less traffic, then unneeded instances can be taken out, and it will be less costly.

Scalability isn't just about scaling up resources; it's about proper architecture. Applications should be scalable so that they can be run multiple times. Stateless application design can help as each instance can process any given user request. State should not be embedded in individual application instances instead should be stored in shared databases, caches or storage services.

9.3 Application Gateway

An application gateway controls the application layer traffic, typically the OSI layer 7. An application gateway can route requests based on details of HTTP requests, including URL path, hostname and/or headers, which a simple network-level load balancer cannot do. Azure Application Gateway enables web traffic load balancing and can be used in conjunction with Web Application Firewall features [13].

An application gateway can intelligently route requests for an e-commerce platform. For instance, requests to /products can be sent to the product service and requests to

checkout can be sent to the payment service. This enables microservices and enhanced traffic management.

Security options like SSL termination and Web Application Firewall integration are also supported by application gateways. Azure WAF offers a centralized way to protect against common web exploits and vulnerabilities, such as attacks like SQL injection and cross-site scripting [14]. It is crucial with e-commerce systems given that they supply prone client and settlement data.

10. Relationship Between Load Balancing and Scalability

Load balancing and scalability are complementary – they address related but different problems. Scalability expands available computing capacity, and load balancing provides efficient utilization of computing capacity by distributing user traffic among it.

Users could also be directed to one over-loaded server while other servers may only be utilized at a fraction of their capacity if an application is scaled out by adding more servers with no load balancing. If this is the case, the scaling won't yield the desired speed boost. Likewise, if a load balancer has fewer back-end servers than needed, it will not be able to handle high demand due to lack of capacity.

Load balancing and scalability work together to ensure a reliable and efficient system. Auto scaling can scale up or down based on demand and the load balancer can distribute traffic among healthy instances. In the e-commerce context, during the same period, having more traffic means more app instances are started to support that traffic. The load balancer then distributes the requests to all healthy instances. Occasionally instances can be eliminated when traffic is low to lower expenses.

This is a relationship that also helps improve availability. In the event of failure of one application instance, the load balancer sends traffic to the remaining instances. If demand increases, scaling ensures more instances become available. Thus, the combination is not only a performance booster, but a resilience booster as well.

11. Benefits of Combining Load Balancing, Scalability and Application Gateway

Load balancing, scalability and application gateway services are excellent building blocks for customer facing cloud applications. Different technologies provide different benefits.

Load balancing increases your availability and performance by sending requests to multiple resources. It makes sure that the system can adjust when there is a need for change. Application gateways offer intelligent routing, SSL termination and security controls. These technologies combine to form an architecture that is more reliable, secure and efficient.

This integration enhances the customer experience in an e-commerce platform. The traffic is distributed efficiently which leads to faster loading rate of pages. Even when a server fails, the system continues to function since the failed servers are excluded from traffic rotation. Cost is managed as resources can be scaled back when demand is not high. The application gateway and WAF can inspect and filter malicious traffic, allowing for enhanced security.

The advantages, however, are complicated. Organisations need to set up routing rules, scaling policies, health checks, certificates, firewall rules and monitoring. If not configured correctly, this will cause downtime or wasted expense. Thus, cloud architecture needs to be designed, tested and governed carefully. These tools from a critical point of view are very effective but need to be utilised in an architectural manner.

12. Driving Forces Behind Cloud Computing

There are multiple factors that have driven the swift growth of cloud computing. These encompass cost-effectiveness, flexibility, scalability, reliability, global innovations and reachability.

12.1 Cost Efficiency

Cost is one of the biggest reasons for the adoption of the cloud. This is because capital expenditure is needed for traditional infrastructure to execute applications, as organisations need to buy networking devices and servers. With cloud computing, that is no longer the case; it becomes an operational expenditure model, where organisations pay for the resources, they consume. It is helpful for small and medium-sized businesses, as they will be able to get enterprise level infrastructure without a huge initial investment. It saves on maintainability costs as well because the maintenance of much of the underlying physical infrastructure is the responsibility of the cloud provider.

12.2 Flexibility and Agility

Organisations can put resources into use rapidly using cloud computing. Teams can provision servers, databases, and storage services in mere minutes, as opposed to weeks. This enhances speed of business agility and helps to experiment faster. Cloud platforms also facilitate DevOps practices to software development teams. Developers can utilize automatic pipelines, containers and cloud infrastructure to build, test and release their applications. This minimises the delay in releasing and will enhance the responsiveness in meeting user needs.

12.3 Scalability and Elasticity

Cloud platforms are engineered to scale resources as needed. This is crucial when it comes to modern applications and the unexpected traffic that people may have. One marketing campaign can all a sudden bring in more traffic to your website and regular traffic can be quite low. Cloud elasticity enables companies to manage the highs and lows without having to invest in unused resources.

12.4 Reliability and Disaster Recovery

Cloud providers have geographically distributed availability zones and data centres. Cloud providers have large data centres in availability zones that are spread across regions. It helps to maintain high availability and disaster recovery. Applications and data can be replicated across sites, minimising the risk of total service outage. But reliability isn't guaranteed, however. Applications need to be designed in a way that leverages these cloud features.

12.5 Global Accessibility and Innovation

Cloud Service is available 24 hours a day, 7 days a week, 365 days a year, and can be accessed from anywhere if you have an internet connection. This facilitates remote working, collaboration across borders and services across the globe. Some of the advanced services offered by cloud platforms include AI, machine learning, analytics and IoT platforms. These services enable organisations to break into new markets and to innovate without having to create complex infrastructure from the ground up.

13. Serverless Computing: Functions as a Service

Serverless Computing is a cloud computing paradigm in which the developer writes and deploys code without the need to have to deal with the underlying servers. Situated between front and back ends, the serverless paradigm doesn't imply that servers are never utilized. Rather, the cloud provider takes care of the servers, runtime environment, scaling and availability [15].

Examples include AWS Lambda and Azure Functions. AWS Lambda executes on events and Azure Functions has event-driven and scheduled executions. Many people refer to serverless computing as Functions as a Service, due to the nature of developers deploying small functions that respond to triggers.

13.1 How Serverless Works

Serverless mode is when a function is called when an event occurs. Events can be HTTP requests, file uploads, database updates, message queue events or scheduled timers.

Event	Example
HTTP request	Customer submits an order
File upload	Product image uploaded
Database update	Customer record changed
Queue message	Order confirmation needed
Timer	Scheduled report generation

Table 3: Serverless Event Types and Examples

In an e-commerce app, for instance, one serverless function might send an order confirmation e-mail following the customer has actually paid. Another function might be able to resize product pictures after they are uploaded. Another might be able to run daily and provide sales reports.

13.2 Benefits of Serverless Computing

Serverless computing decreases the operational load as developers don't have to deal with servers. It also provides auto-scaling. Increased events triggers increased function instances to be run by the cloud provider. If there are no events, then there is no need for any function instances to run continuously. This can save cost for non-continuous workloads since users only pay for execution and not always on infrastructure [16].

In addition to fast development, serverless is compatible with rapid development. Developers can work on the business logic, while the infrastructure management is handled by the infrastructure team. It can be used for small jobs, background processes, APIs, automation and event-driven processes.

13.3 Limitations of Serverless Computing

Serverless has limitations. Cold start latency can happen if a function has not been leveraged for a while, and the platform takes time to start up the function. Serverless may not be a good choice for long-running tasks because of execution time limits. Another factor to consider is vendor lock-in since serverless functions are likely to rely on provider-specific services, APIs, and event types.

For that reason, serverless is a great option for event-driven workloads, but it won't necessarily be for workloads with extended runtimes or custom runtime control or strong provider portability.

14. Comparison Between Containerisation and Serverless Computing

There are two significant models of cloud deployment, containerisation and serverless computing, and they're appropriate for different scenarios.

Feature	Containerisation	Serverless
Control	High control over runtime	Lower control
Scaling	Manual or automated	Fully managed automatic scaling
Cost model	Pay for running resources	Pay per execution/duration
Startup	Fast	May experience cold starts
Best use case	Full applications and microservices	Event-driven functions
Portability	Strong across platforms	More provider-dependent
Management	Requires image/container management	Provider manages infrastructure

Table 4: Comparison Between Containerisation and Serverless Computing

Containers provide greater control. The runtime, dependencies, networking and deployment can be determined by developers. This makes containers ideal for APIs, Web applications, microservices, and long-running services. But there are image, orchestration, monitoring, networking, and security concerns to consider for containers.

Serverless also eliminates the complexity of infrastructure management, as the cloud service provider takes care of it. This is perfect for applications that process a great deal of data in an event-driven manner. But there's less control with serverless and cold starts and provider lock-in.

Many organisations will use both in practice. A collection of e-commerce services can have a main web application deployed in containers, and back-end functions deployed as serverless functions to send email, resize images, process queue messages or create reports. This combination provides organisations the best of both worlds.

15. Conclusion

This report has critically discussed the principles of cloud engineering with emphasis on Docker, cloud services and on modern deployment methodologies. Virtualisation was mentioned as a key technology for the cloud as it allows multiple virtual machines to be run on a single physical machine. It optimises resource consumption, facilitates multi-tenancy and allows for flexible provisioning. But virtual machines need to run full operating systems, which consumes more resources and makes it take longer to start up.

Lightweight containerisation was studied as a property to enable portability, deployment speed and scalability. Docker was introduced as a practical and hands-on containerisation platform to define Dockerfiles, build images, run containers, publish images, manage volumes and create networks. The practical examples illustrate how Docker enables consistent deployment and is aligned with the modern DevOps workflows.

An e-commerce scenario also was used to assess load balancing, scalability, and application gateways. Using load balancing distributes traffic amongst healthy resources, enhancing performance and availability. Applications should be scalable and able to deal with demand changes, and gateways offer intelligent routing and security controls. These technologies provide reliable, secure and high-performing cloud applications.

Finally, the report examined the factors that have led to the development of cloud computing, such as cost, flexibility, scalability, resilience, global availability, and innovation. The capabilities and drawbacks of both serverless and container-based architectures were highlighted and contrasted. Containers offer greater control and portability, while serverless platforms offer simplicity and automatic scaling. Serverless platforms offer simplicity and auto-scaling, and containers offer control and portability. In sum, the best contemporary cloud architectures are frequently based on both serverless computing and containerisation, with the goal of offering scalability, resilience, efficiency, and flexibility.

References

- [1] P. Mell and T. Grance, “The NIST Definition of Cloud Computing,” National Institute of Standards and Technology, Special Publication 800-145, 2011.
- [2] Docker Inc., “Dockerfile reference,” Docker Docs. [Online]. Available: <https://docs.docker.com/reference/dockerfile/>.
- [3] R. Buyya, C. Vecchiola, and S. T. Selvi, *Mastering Cloud Computing: Foundations and Applications Programming*. Waltham, MA, USA: Morgan Kaufmann, 2013.
- [4] T. Erl, R. Puttini, and Z. Mahmood, *Cloud Computing: Concepts, Technology & Architecture*. Upper Saddle River, NJ, USA: Pearson, 2013.
- [5] D. Merkel, “Docker: Lightweight Linux containers for consistent development and deployment,” *Linux Journal*, vol. 2014, no. 239, 2014.
- [6] R. Morabito, J. Kjällman, and M. Komu, “Hypervisors vs. lightweight virtualization: A performance comparison,” *IEEE Cloud Computing*, vol. 2, no. 3, pp. 38–47, 2015.

- [7] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, “Borg, Omega, and Kubernetes,” *Communications of the ACM*, vol. 59, no. 5, pp. 50–57, 2016.
- [8] N. Dragoni et al., “Microservices: Yesterday, today, and tomorrow,” in *Present and Ulterior Software Engineering*. Cham, Switzerland: Springer, 2017.
- [9] G. Pahl, “Containerization and the PaaS cloud,” *IEEE Cloud Computing*, vol. 2, no. 3, pp. 24–31, 2015.
- [10] Docker Inc., “Volumes,” Docker Docs, 2026. [Online]. Available: <https://docs.docker.com/engine/storage/volumes/>
- [11] Docker Inc., “Networking overview,” Docker Docs, 2026. [Online]. Available: <https://docs.docker.com/network/>
- [12] Amazon Web Services, “Elastic Load Balancing User Guide,” AWS Documentation, 2026. [Online]. Available: <https://docs.aws.amazon.com/elasticloadbalancing/>
- [13] Microsoft, “Azure Application Gateway documentation,” Microsoft Learn, 2026. [Online]. Available: <https://learn.microsoft.com/en-us/azure/application-gateway/>
- [14] Microsoft, “Web Application Firewall documentation,” Microsoft Learn, 2026. [Online]. Available: <https://learn.microsoft.com/en-us/azure/web-application-firewall/>
- [15] Amazon Web Services, “AWS Lambda documentation,” AWS Documentation, 2026. [Online]. Available: <https://docs.aws.amazon.com/lambda/>
- [16] Microsoft, “Azure Functions documentation,” Microsoft Learn, 2026. [Online]. Available: <https://learn.microsoft.com/en-us/azure/azure-functions/>
- [17] J. Spillner, S. Mateos, and D. A. Monge, “FaaSter, better, cheaper: The prospect of serverless scientific computing and HPC,” in *Proc. Latin American High Performance Computing Conference*, 2017.
- [18] E. Jonas et al., “Cloud programming simplified: A Berkeley view on serverless computing,” *arXiv preprint arXiv:1902.03383*, 2019.