



University of Wales
Trinity Saint David
LONDON

CLOUD ENGINEERING PRINCIPLES AND PRACTICE

Docker and AWS Cloud Implementation: A Technical and Analytical Study

Module: Cloud Engineering Principles and Practice (ACCA7027)

Student Name: Sameer Rajubhai Panchal

Student ID: 2516024

Table of Contents

1. Introduction.....	3
2. Cloud Technology (Docker) and Practical Examples.....	3
2.1 Principles of Virtualisation	3
2.2 Principles and Benefits of Containerisation.....	4
2.3 Images and Publishing Images.....	6
2.4 Container Volumes	7
2.5 Container Networking.....	8
2.6 Full Demonstration (End-to-End Example).....	9
3. Cloud Computing Concepts.....	12
3.1 Load Balancing	12
3.2 Scalability	12
3.3 Application Gateways.....	13
3.4 Why Load Balancing and Scaling are Complementary	14
3.5 Combined Benefits (Load Balancing, Scaling and Application Gateway).....	15
4. Main Driving Forces Behind Cloud Expansion.....	16
4.1 Cost	16
4.2 Flexibility and Agility.....	17
4.3 Evolution to Serverless Computing	18
5. Conclusion	19
References.....	20

1. Introduction

Cloud computing has turned to be a central component of contemporary information systems, which has helped organisations to gain access to scalable, flexible, and cost-effective computing resources. With the trend towards enterprise migration to cloud platforms, solutions that enable the efficient provision and administration of applications have become very important. Of these, the most significant aspect is containerisation, mostly achieved by Docker and similar tools, to provide consistent, portable, low-resource consumption execution environments on a variety of heterogeneous infrastructure. With the concept of virtualisation and containerisation, the earlier methodology of deployment in applications has changed to increase the isolation, automation and operational stability in the cloud ecosystems. The report discusses major cloud technologies and practices that were introduced in the course of classroom work, specifically Docker and its application in cloud-based application development. The concepts of virtualisation, creating images, creating volumes of containers, networking, and releasing images are discussed, and real-life examples are provided. Also, basic concepts of cloud computing, such as load balancing, scalability and application gateways, are examined to explain how they aid in the facilitation of resilient and high-performance services. The discussion also considers the key factors that lead to the further growth of cloud computing, such as cost-effectiveness and operational scalability and compares the idea of containerisation and newly introduced serverless computing technologies. Generally, the report offers a critical discussion of the concept of cloud engineering and its use in the modern computing world.

2. Cloud Technology (Docker) and Practical Examples

2.1 Principles of Virtualisation

Hardware virtualisation is one of the underlying technologies that facilitated the development of cloud computing. Applications in the conventional computing world were directly implemented on physical workstations, which were often inefficient in the sense that they did not maximise the available resources and were also limited by system setups. Hardware virtualisation also modifies this model and creates an abstraction layer to isolate physical resources into a series of virtual environments. Every environment, a virtualised computing environment otherwise known as a virtual machine (VM), simulates one virtualised CPU, memory, storage and a network stack, yet on the same physical server (Ugwueze, 2024).

The most important mechanism that makes this abstraction possible is the hypervisor. Hypervisors handle the allocation of hardware facilities among various VMs to maintain the

seclusion and independence of all the environments. There are usually two types of hypervisors. Bare-metal hypervisors, usually referred to as type 1 hypervisors, are the kind that operate on the host hardware itself. The design is also conducive to high performance, good isolation and effective resource utilisation and Type 1 hypervisors can be used in an enterprise cloud infrastructure. Examples usually cover general data centre platforms.

In contrast, type 2 hypervisors can run on top of an already existing host operating system. This type of architecture is typically deployed in development, testing and educational systems instead of large-scale production workloads, due to the added layer of software overhead and lower performance compared to Type 1 systems.

The shared hardware virtualisation allows various operating systems and applications to be run on the same machine, but isolated and reliably performant (Sannapureddy, 2025). This not only enhances the implementation of resource utilisation, but also makes the process of providing workloads, as well as backing up and moving them, easier.

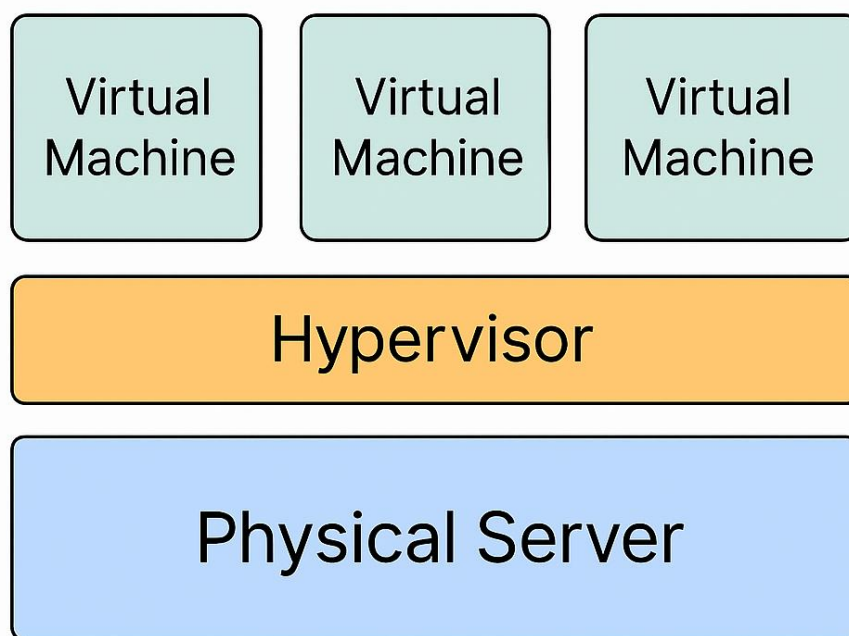


Figure 1: Virtualisation architecture

2.2 Principles and Benefits of Containerisation

Although virtualisation is still needed at the infrastructure level to share resources, containerisation is now offered as an optional solution for its application-level deployment.

The containers offer lightweight and compactly implementable execution settings that package application code, dependencies and runtime setups whilst sharing the host operating system kernel. This feature is unique to containers, and VMs require individual operating systems on virtual instances.

The architectural difference creates several operational benefits. Containers use much fewer resources than VMs since they do not need to execute multiple full operating systems. It enhances density, minimises memory overhead and decreases the start-up times, making the cloud-native applications faster to scale and more responsive. Isolation is provided by kernel-level controls to ensure that all the containers run independently, and they do not interfere with one another (Kader and Ahmed, 2025).

Another great advantage is portability. The behaviour of a container image that executes on a single machine is the same across other systems that are capable of executing the same container engine. This uniformity minimises configuration mistakes and enhances deployment consistency in the field of development, testing and operations.

Containerisation is a heavy proposition of cloud-native architectures that fosters the application design in microservices. The microservices may be built as individual containers, which can be deployed independently, scaled, updated and rolled back. This framework makes things more resilient and safe, and compliant with modern DevOps operations like continuous integration and continuous delivery pipelines.

Example: Constructing a Dockerfile

A Dockerfile is a document that has the building instructions of a container image. Not everything can be shown in the form of code; however, the process can be described conceptually. A Dockerfile usually starts with a base image, which gives the basic building blocks on which the application will be running on (Farooq, 2022). This base may be a loosely-linked Linux distribution or even a runtime system, or platform, e.g. a web server platform or a programming language run time.

The latter directives put a duplicate of the application files into the image and install dependability, and set up the execution environment. Instructions can be given to the system to install libraries, set up directories or environmental variables. Extra directions can be left open at certain ports so that they can be accessed by outside people once the container is started. Lastly, there is an execution command that identifies the behaviour to be entered when the container is launched, either starting a server or an application process.

During the processing of the Dockerfile, an image is created in small steps. Instruction creates a new layer, and Docker stores them in order to build them faster. With this stratified

architecture, the development velocity is maximised and decreases storage specifications in different images that contain common elements (Patel, 2025).

2.3 Images and Publishing Images

Building a Docker Image:

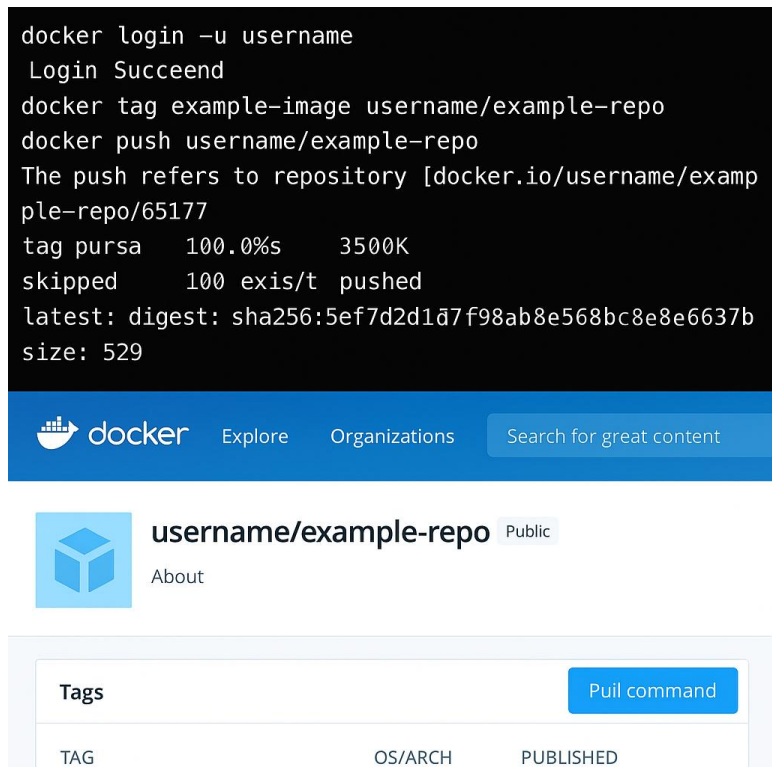
The Docker image is created by executing the process that is proven to make the corresponding command-line that reads the Dockerfile and builds the specified layers. The process examines the instructions, retrieves any base images needed and constructs an unmodifiable image of all constituents of the application.

Layering of the image is an important optimization process. Docker stores shared layers only once two or more container images are similar in terms of their dependencies, e.g. they share a common operating system layer. This method is highly beneficial towards minimising storage overhead and increasing image distribution.

With an image tagging process, various versions or variations of the same application can be managed. Tags are distinctive identifiers to image names, such that they allow organising releases of a stable, development or experimental version (Kettunen, 2024). Good tagging is an important factor in version control and the predictability of deployment.

Example: Publishing an Image to DockerHub

DockerHub is an extensive container image distributor application. However, the authentication should be conducted before publishing an image by a command termed a login. After authentication, an image may be tagged with a namespace of the DockerHub repository. The image layers are uploaded to the registry by a push command. When the image has layers in common with those uploaded before, then only new or edited layers in the image are relayed. Once it is completed, the picture is made publicly available or privately available based on repository arrangements.



2.4 Container Volumes

Purpose of Volumes:

Containers are a disposable default; removing them causes them to lose all internal state unless permanently set up to the contrary. Container volumes offer long-term storage, which continues to exist even after the death of the individual containers (Kertész, Farkas and Szabó, 2021). This provides the opportunity to maintain application data and logs or database files consistently during updates, recreation and replacement of containers as well.

Containers allow access to shared data as well, with the help of container volumes. As an example, two containers could also be able to share a common data directory in which data is processed collaboratively through file exchange or a caching process. Using externalised persistent storage, the application logic and underlying filesystem can be decoupled by volumes, allowing more of them to be configured in looser and more resilient structures.

Example: Creating a Docker Volume

A volume is created as a volume based on the corresponding command-line instruction that gives the memory of a lasting storage place in the managed storage system of Docker. After making it, the volume can be attached to a vessel. The volume can then be mounted on a given path inside the container, to which the application may then write and read (Indukuri, 2025).

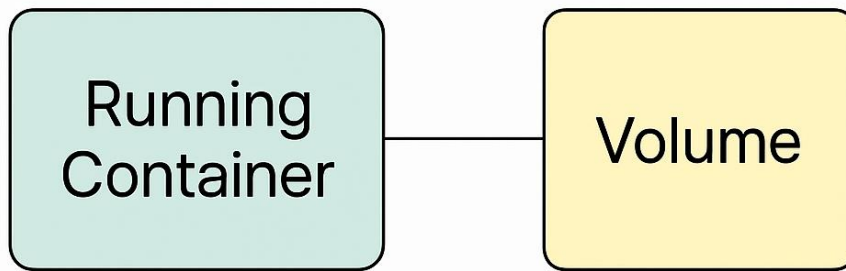


Figure 3: Volume mounted in a running container

2.5 Container Networking

Docker Networking Principles:

Networking is a major component in container orchestration as it allows communication between and among application components. Docker has some networking modes that fit the various applications.

The default configuration is the bridge network that is used in standalone containers and enables inter-container communication that makes use of internal IP addresses. Bridges network containers can securely communicate with one another, and they are not exposed to the external systems at any point.

The host network mode eliminates the network isolation layer, and a container can access the host network stack directly. It minimises overhead and could apply to performance-intensive applications, although the isolation offered by containerisation is lost (Rice, 2025).

The overlay network expands the networking possibilities in the distributed systems through linking containers running on various hosting systems. The overlay networks play a crucial role in the clustered environment and container orchestration platforms, such that they facilitate the communication between multiple nodes without needing to resolve their two complex configurations.

Example: Creating a Docker Network

A simple command can be used to create a custom network, and the containers can be attached to the custom network. Once two or more containers are connected to the same network, they may exchange names via the predictable convention of DNS naming, which the Docker

resolver provides. The feature eases the process of service discovery and enhances maintainability in a microservices-based architecture.

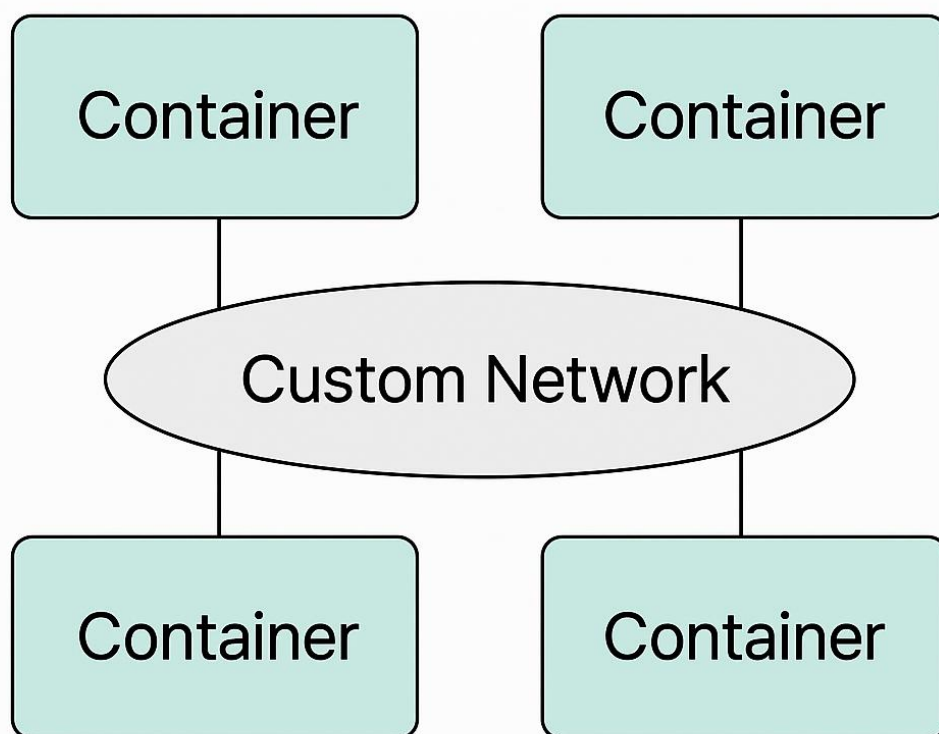


Figure 4: Containers communicating on a custom network

2.6 Full Demonstration (End-to-End Example)

As a part of the workflow entails a full demonstration of and so as to satisfy the need for a complete demonstration on how an application could advance all the way through the local

development into the subsequent deployment in a cloud-compatible format, this section illustrates the end-to-end demonstration (Silva et al., 2025).

Building and Running a Web Server Container:

The simplest web server application may be realised with the help of the simplest HTML file in a specific directory. The Docker image that is coupled with this application lists a base image with a web server runtime and includes the down command of copying the HTML file to the suitable directory in the image.

The process of building will produce an image with the web assets and server setup. Once the image is available, a container can be run with a command that creates an external port as mapped to a port on the internal web server. When the container is started, it can be accessed by a web browser on the IP address of the host machine and port mapping.

This demonstration shows the fundamental nature of containerisation, i.e. portability, isolation, fast deployment and consistency of the environment (Venugopal, 2025). One container image can be run on any host that it is compatible with, without changes, ensuring predictable behaviour.

Publishing the Image:

As soon as the web server image works properly, it can be sent to DockerHub after the process of authentication and tagging, which was mentioned above. Endeavouring the picture renders it accessible to co-workers or computerised release streams. Then, the image can be pulled on the cloud platform with DockerHub and deployed to the production environments or to the auto-scaling groups.



username / web-server

Public

Overview



web-server
Latest

11.7 MB

a day ago

Figure 5: Web server image published to DockerHub

Discussion:

This is one end-to-end demonstration that brings to the fore some benefits that containerisation embodies. To begin with, it points to the portability of container images as the application is capable of being compiled once and executed on different systems without being reconfigured. Second, the demonstration indicates the ease with which image versioning control and publish processes facilitate collaboration in the workspace, constant deployment lines and autonomous scale production (Kratzke, 2023). Third, the feature of adding networking, volume management and Dockerfile building shows the holistic character of deploying applications in a container-based on the deployment.

In general, the example demonstrates that containerisation can be used to simplify cloud-oriented development by removing the complexity, rapid deployment cycles, and consistent runtime behaviour of an application, which are crucial attributes in the current and modern cloud engineering practice.

3. Cloud Computing Concepts

3.1 Load Balancing

Load balancing is an important process in cloud computing infrastructure whereby incoming traffic is smoothly spread among a number of compute resources. Major cloud providers have accomplished load-balancing technologies of great specialisation to serve availability, performance and fault tolerance.

Amazon Web Services (AWS), as an example, provides the service of the Elastic Load Balancer (ELB), which includes various types depending on their intended usage. Application load balancer, which is defined as ALB, functions on Layer 7 of the OSI model, which facilitates advanced content-based routing, host-based and path-based routing and microservices (Raj, Vanga and Chaudhary, 2022). Automatically allowed, ALB is normally applied to HTTP and HTTPS loads when they need to stabilise applications or routing, or monitor their application-level data. Network Load Balancer (NLB), the other one, is a Layer 4 protocol activated and is designed to provide very fast throughput and low latency, thereby suitable for TCP, UDP, and TLS traffic. NLB can sustain millions of simultaneous connections using expected performance.

Microsoft Azure has the same functionality with its intrinsic load-balancing service. Azure Load Balancer provides Layer 4 distribution between virtual machines to ensure that large-scale deployments remain in operation when resource requirements are higher than usual or when a node has gone offline. Applying traffic management at the application level, Azure Application Gateway offers routing at Layer 7, termination of an SSL/TLS and combination with the Azure Web Application Firewall (WAF). This integration allows analysis of traffic in more depth and with high security, as well as supporting current web-based architectures (Chippagiri and Ravula, 2021).

The basic goal of load balancing is the same in all providers: to achieve continuous availability of services without creating a bottleneck in one of the computer instances. The strategy used by cloud platforms includes a combination of health checks, automatic failure and adaptive traffic control to ensure that while the workload is operating between very high and very low levels, the deployed system stays at the best level of performance.

3.2 Scalability

Scalability refers to the ability of a cloud system to scale its computing resources as a result of varying levels of demand. There are two principal strategies: vertical scaling and horizontal scaling.

Vertical scaling or scaling up adds to the performance of a given machine by assigning more CPU, memory or storage facilities. This is a very simplistic method, and can be frequently helpful in monolithic applications or environments where the architectural constraints restrict distribution across many nodes. Vertical scaling is, however, limited by physical and economic factors; to some degree or other, it becomes impossible to expand or costly to do so (Ukene, 2024).

Horizontal scaling or scaling out involves introducing new instances that run in parallel, effectively distributing the workload among more than one machine. Such an approach is intrinsically compatible with the principles of cloud-native and microservices-oriented systems, in which the parts of the application are created in such a manner that they can be run concurrently across replicated nodes.

Horizontal scaling has to be automated by cloud providers using systems like AWS Auto Scaling Groups and Azure Virtual Machine Scale Sets. A policy in an Auto Scaling Group specifies the event under which instances should be created or removed. Efforts like CPU usage, the number of requests or application latency are the metrics that determine the need for the addition of additional nodes. In the case of a reduced demand, the redundant cases are cancelled gracefully, and this operation is cost-effective.

Azure Virtual Machine Scale Sets have principles close to Azure Virtual Machine, but they have close integration to Azure networking, identity and monitoring (Olabanji, 2022). Scale Sets enable load balancers, rolling upgrades and orchestration strategy to minimise the interference on a scaling event. These features guarantee capacity growth to be transparent and foreseeable, and automatically gain momentum.

Scalability thus aids elasticity, which is a main feature of the cloud, so that the applications are able to handle the unpredictable demand patterns without compromising efficiency and without having to manage costs.

3.3 Application Gateways

Application gateways are higher-level Layer 7 reverse proxies that can inspect and smartly route application-level traffic. Application gateways alter the routing decisions based on high-level protocols (HTTP headers, cookies, URI paths, and similar attributes) instead of the traditional load balancer that customarily bases its routing decisions on the transport-level metadata.

One of the main roles of an application gateway is to terminate a connection between the SSL/TLS. The transfer of the process of encryption and decryption to the backend servers

decreases the need to process the tasks of third parties on the servers, and standardises the management of certificates (Deshmukh, Malviya and Mohammed, 2025). This eases the compliance and increases the throughput. Application gateways are also used to support advanced routing policies, including routing requests to address other backend pools depending on domain names, API version or type of device.

An important aspect of current application gateways is that they use a Web Application Firewall (WAF). The WAF guards against some such common vulnerabilities as SQL injection, cross-site scripting and protocol irregularities. Cloud platforms offer managed rule sets, which make sure that security patches are automatically deployed, and administrators do not have to work so much.

An application gateway configuration decision can be one of path-based routing versus host-based routing. To illustrate, the path-based routing is able to route requests containing the path /images to a dedicated media service and routing those containing the path /api to a microservices backend. Host-based routing is able to differentiate among various domains that are provided by the same infrastructure (Marie-Magdelaine, 2021). Performance, scalability and operational complexity depend on application architecture in such decisions.

Application gateways improve control, security, as well as performance in sophisticated distributed systems through all these capabilities.

3.4 Why Load Balancing and Scaling are Complementary

Scaling and load balancing work best as a combined mechanism since each serves a distinct issue regarding resource management. Scaling consists of adding or increasing compute resources so that the system can be sufficient in terms of processing power to satisfy demand. No distribution strategy would avoid overload unless there is adequate capacity.

Load balancing is the process whose purpose is to allocate intelligent traffic over these resources. With enough other nodes present, inappropriate distribution may result in hotspots, idle servers or low performance. Load balancers are used to ensure that every instance is appropriately allocated the workload concerning the health, the availability and routing policies (Singh, 2025).

Scaling and load balancing will ensure that the system is not overwhelmed with resources through capacity changes and demand-balanced distribution. This combination stabilises the performance, cuts down the latency and limits the chances of outages in the peak periods. Besides, health checking by use of health checks ensures that the cases of unhealthy or non-responsive cases are automatically eliminated in a traffic rotation, ensuring service continuity.

Scaling offers the amount of resources, whereas load balancing offers the quality of distribution. The combination of the two approaches is the foundation of resilient cloud infrastructure.

3.5 Combined Benefits (Load Balancing, Scaling and Application Gateway)

Scaling mechanisms and application gateway work together when load balancing, cloud services are improved to a significant extent in terms of reliability, efficiency and security.

High Availability:

Load balancers spread traffic among healthy instances, whereas scaling policies maintain sufficient instances to be active (Theodoropoulos et al., 2023). Application gateways introduce smart routing and termination of TLS, which ensures that the traffic at the application layer is handled in a consistent manner. The collective action ensures that service uptime is guaranteed even when single parts fail.

Fault Tolerance:

The system resilience is greatly enhanced when there are several nodes and the traffic is intelligently distributed. Should a single instance prove to be unhealthy, then the load balancer will remove it from its rotation and will automatically be replaced by the scaling engine. Application gateways have the added advantage of checking out malicious traffic and avoiding malfunctioning applications.

Security Enhancements:

WAF gateways have application gateways that stop the threat before the backend services are accessed. The termination of encrypted traffic enables central administration of the traffic and lowers the attack area. Load balancers conceal the identity and IP addresses of servers located behind them, thus limiting exposure. These features, when combined with scaling, assist in absorbing traffic bursts that come with malicious attacks like DDoS attacks (Nascimento et al., 2024).

Improved User Experience:

Some efficiency in distributing traffic minimises latency and guarantees a constant response time. Scaling mechanisms ensure that the performance does not reduce during the peak period, and the application gateways send requests to the best-suited backend services based on content-driven logic. All these factors will translate into seamless, predictable interaction by end users, irrespective of the changes in demand.

Optimised Operational Efficiency:

Scaling and routing are automated, which minimises administrative load and the use of manual intervention. Smart Traffic Management controls resource consumption and ensures that the number of compute resources is only used as per the need. This does not just ensure performance but also enhances cost efficiency, which is the main goal in cloud computing strategies (Girhotra and Byrisetty, 2025).

4. Main Driving Forces Behind Cloud Expansion

4.1 Cost

One of the strongest driving forces behind the use of the cloud is cost efficiency. On-premise traditional infrastructure entails a high capital cost on servers, computer networking devices, data centre and storage systems. These initial investments usually pose a huge financial challenge, especially to small and medium-scale enterprises. Cloud computing substitutes these capital-intensive forms with a model of consumption-based pricing, which is usually known as pay-as-you-go. The organisations pay in this model depending on the real usage of compute, storage, bandwidth and other services (Deng et al., 2024).

The pay-as-you-go model does not require the business to embark on long-term hardware management procurement cycles and the probability of wasting money on idle capacity. Rather than making predictions about the months or years when the highest levels of demand will occur, organisations will be able to buy only those resources that they will need at the specific time. This establishes a firsthand line in the relationship between the operational cost and actual business operations and enhances performance in cost transparency and financial management. One such advantage is the minimisation of the capital expenditure (CapEx). Cloud solutions will turn infrastructure into operational expenses (OpEX), which will be more predictable and controllable. Cloud adoption liberates financial capacities to invest in strategic activities of software creation, information-driven innovation or customer-centric services by eliminating the necessity to invest in physical servers and data centre settings.

The cost elasticity also takes centre stage. Elasticity can be defined as the capacity of cloud systems to increase or decrease resources automatically with the change of demand (Banerjee and Chatterjee, 2025). The capacity is immediately available when workloads peak; it is freed when workloads drop, hence unnecessary spending is avoided. This dynamic allocation system is such that cost is maintained relative to actual utilisation of resources, minimising wastage and maximising efficiency of operations.

Together, pay-as-you-go pricing, less capital and elastic consumption models offer unquestionable economic benefits that are ever gaining momentum to push the world to cloud-based infrastructures.

4.2 Flexibility and Agility

Cloud computing has great flexibility and agility in its operations, aside from the issue of cost. These attributes have been necessary in a place where digital transformation, speedy innovation and international service provision are the main business priorities.

Rapid Deployment:

Cloud applications aid quick deployment of virtual machinery, containers, storage prospects, networks and managed services. Environments once taken days or weeks to configure in physical data centres might now be automated in an orchestration within minutes or even self-service to end users. It allows development and testing cycles to be drastically reduced, which allows teams to create, test, and more effectively release applications (Surianarayanan and Chelliah, 2023). Quick deployment improves speed of innovation, time-to-market and competitive advantage.

Cloud environments are also flexible in the sense that organisations can be able to test new technologies without being bound to long-term investments. There is an ability to provision temporary resources to do prototyping, simulation or trials and move them onto another provisioning when they are no longer required. This mitigates the risks of the adoption of emerging technologies and facilitates the overall practices of continuous improvement.

Global Distribution:

The other significant benefit is that it enables the distribution of applications and data across geographically different regions. Cloud providers have large networks of data centres worldwide, which enables organisations to use resources near end-users across the world. This geographic flexibility enhances the performance of the application because it reduces the latency and ensures adherence to data residency or sovereignty requirements.

The disaster recoveries are also reinforced through global distribution. It is possible to replicate the resources and create multiple regions in such a way that the applications will still be available during outages, natural disasters or some other disturbances affecting a certain area (Henning and Hasselbring, 2022). Multi-region deployment offers continuity, resilience and increased reliability, allowing organisations to perform consistently irrespective of the situation in each of the regions.

Mostly, flex-agile is a fundamental feature of cloud ecology, which enables organisations to keep up with market demands, rapidly innovate and sustain operational power in international settings.

4.3 Evolution to Serverless Computing

What Are Serverless Functions?

The lightweight units of computation, which are auto-scaling and run in response to events, are called serverless functions. They allow developers to execute code without scaling or deploying servers, operating systems or runtime environments. All of the infrastructure, such as scaling, patching, load balancing and monitoring, is done by cloud providers. The outcome is a greatly achieved streamlined and hassle-free deployment model, which is very relatable to microservices and event-driven structures.

AWS Lambda and Azure Functions are considered to be two of the most popular serverless platforms. AWS Lambda is configured to execute code when startled by an event like an HTTP request, uploading files, updating databases, the occurrence of events in a message queue or a saved task (Natarajan and Jacob, 2024). Azure Functions is also able to accomplish the same, and it also adheres to the same environment, like event hubs, storage accounts, and service buses.

Serverless means that the servers can be scaled automatically with demand since they run on a per-use basis. The platform automatically provides the required resources to carry out the purpose when events take place. The resources are deallocated when the activity declines, and the cost will be zero when the activity is not being operated. This is the opposite of the past, whereby computing models have had resources allocated irrespective of workload.

Lack of infrastructure management also increases productivity and cost effectiveness. Cloud providers will be accountable in terms of maintenance of the operating system, capacity planning, runtime patching and fault resilience. This gets rid of most of the heavy load in operations and lets the organisations concentrate on application logic, business rules and innovation (Ponugoti, 2025). As a result, serverless computing has emerged as a strong driver of faster development processes and less complexity in the operation.

Serverless in the Broader Cloud Context:

The advent of serverless computing is consistent with the general forces of cloud growth. Competitiveness in costs is raised by granular billing models, which are only charged on the basis of execution time. Increasing the effectiveness of operations. Flexibility in operational terms is enhanced by automated, scaled operations as well as simpler deployment processes.

Distribution around the world is made easily, serverless functions are able to be deployed in different locations without infrastructure being configured manually.

More so, serverless functions have a great deal of integration with containerised and microservice systems. They help to support modular application design, allowing discrete units of functionality to facade, and which communicate on event streams or APIs (Arif, Jo and Park, 2025). This level of architectural synergy enables development methodologies of modern clouds and enables intricate automation, orchestration and data processing processes.

5. Conclusion

The continuous evolution of virtualisation and containerisation, and the emergence of serverless, continue to transform the design, deployment and operation of modern digital services, which is also influenced by cloud computing. Container technologies based on Docker have a key role to play in supporting stability, portability, and efficiency of resources in application environments, and real-life examples of image creation, networking setup and volume management indicate the usefulness of these tools in cloud environments. Load balancing, scalability mechanisms and application gateways further accomplish or build cloud architectures by providing better availability, performance and security. All these features are applicable in making sure that an application can be responsive and resilient to varying load levels and various operational environments. The economic and operational aspects, such as decrease in capital spending, pay-as-you-go pricing, elastic resource utilisation and world deployment alternatives, remain to hasten the uptake of clouds in industries. The development of serverless computing grows these benefits by removing the infrastructure management work and allowing event-based execution models to be executed with high efficiency. Comparisons of serverless functions and containerisation can show that both technologies deal with various architectural requirements and add to the overall flexibility and agility of cloud platforms. Altogether, the technical innovation together with economic efficiency and architectural flexibility makes cloud computing a necessary source of scalable, secure and sustainable digital transformation.

References

1. Arif, T., Jo, B. and Park, J.H., 2025. A Comprehensive Survey of Privacy-Enhancing and Trust-Centric Cloud-Native Security Techniques Against Cyber Threats. *Sensors*, 25(8), p.2350.
2. Banerjee, R. and Chatterjee, T., 2025. Comparative Evaluation of Cloud-Native and VM-Based CI/CD Pipelines for Automated DevOps Deployments. *Asian Journal of Research in Computer Science*, 18(11), pp.153-171.
3. Chippagiri, S. and Ravula, P., 2021. Cloud-Native Development: Review of Best Practices and Frameworks for Scalable and Resilient Web Applications. *Int. J. New Media Studie*, 8, pp.13-21.
4. Deng, S., Zhao, H., Huang, B., Zhang, C., Chen, F., Deng, Y., Yin, J., Dustdar, S. and Zomaya, A.Y., 2024. Cloud-native computing: A survey from the perspective of services. *Proceedings of the IEEE*, 112(1), pp.12-46.
5. Deshmukh, H., Malviya, R.K. and Mohammed, N., 2025. *Cloud-Native Applications on Microsoft Azure: Microservices, containers, and Kubernetes for modern application development on Azure (English Edition)*. BPB Publications.
6. Farooq, A., 2022. CLOUD-NATIVE DEVELOPMENT: BEST PRACTICES FOR BUILDING SCALABLE APPLICATIONS. *Computer Science Bulletin*, 5(02), pp.266-277.
7. Girhotra, J. and Byrisetty, A., 2025. *Securing Cloud-Native Applications (CNAs): A Case Study of Practices in a large IT Company*.
8. Henning, S. and Hasselbring, W., 2022. A configurable method for benchmarking scalability of cloud-native applications. *Empirical Software Engineering*, 27(6), p.143.
9. Indukuri, A.V., 2025. *Cloud-native transformation: Architectural principles and organizational strategies for infrastructure modernization*.
10. Kader, M.A. and Ahmed, R., 2025. Modern Cloud-Native Architectures: A Review of Full-Stack Design Patterns and AWS Best Practices. *Scientia. Technology, Science and Society*, 2(11), pp.283-290.
11. Kertész, D.R., Farkas, K. and Szabó, G., 2021. *Best Practices of Cloud Native Application Development*. Bachelor of profession's thesis, Budapest University of Technology and Economics, Budapest.
12. Kettunen, J.P., 2024. *Maintainability in cloud-native architecture*.

13. Kratzke, N., 2023. Cloud-native Computing: Software Engineering von Diensten und Applikationen für die Cloud. Carl Hanser Verlag GmbH Co KG.
14. Marie-Magdelaine, N., 2021. Observability and resources managements in cloud-native environnements (Doctoral dissertation, Université de Bordeaux).
15. Nascimento, B., Santos, R., Henriques, J., Bernardo, M.V. and Caldeira, F., 2024. Availability, scalability, and security in the migration from container-based to cloud-native applications. *Computers*, 13(8), p.192.
16. Natarajan, S. and Jacob, J., 2024. Multi-Cloud Handbook for Developers: Learn how to design and manage cloud-native applications in AWS, Azure, GCP, and more. Packt Publishing Ltd.
17. Olabanji, D.O., 2022. Towards the development of a decision framework for portability in cloud-native architecture deployment (Doctoral dissertation, University of Portsmouth).
18. Patel, J.S., 2025. Cloud-Native Performance Testing: Strategies for scalability and reliability in modern applications. *European Journal of Computer Science and Information Technology*, 13(8), pp.32-49.
19. Ponugoti, M., 2025. Cloud-Native Platform Engineering: Scalable Design Patterns for Global Enterprise Resilience. *Journal of Computer Science and Technology Studies*, 7(8), pp.48-59.
20. Raj, P., Vanga, S. and Chaudhary, A., 2022. Cloud-Native Computing: How to design, develop, and secure microservices and event-driven applications. John Wiley & Sons.
21. Rice, L., 2025. Container Security: Fundamental Technology Concepts That Protect Cloud Native Applications. " O'Reilly Media, Inc.".
22. Sannapureddy, R., 2025. Cloud-Native Enterprise Integration: Architectures, Challenges, and Best Practices. *Journal of Computer Science and Technology Studies*, 7(5), pp.167-173.
23. Silva, F.A., Trinta, F.A., Bonfim, M.S., de Macedo, J.A.F., Rego, P.A. and Lagrota, V., 2025. Performance Evaluation of Cloud Native Applications: A Systematic Mapping Study. *Journal of Network and Systems Management*, 33(4), pp.1-35.
24. Singh, H., 2025. STRATEGIES TO BALANCE SCALABILITY AND SECURITY IN CLOUD-NATIVE APPLICATION DEVELOPMENT. Available at SSRN 5267890.
25. Surianarayanan, C. and Chelliah, P.R., 2023. Demystifying the Cloud-Native Computing Paradigm. In *Essentials of Cloud Computing: A Holistic, Cloud-Native Perspective* (pp. 321-345). Cham: Springer International Publishing.

26. Theodoropoulos, T., Rosa, L., Benzaid, C., Gray, P., Marin, E., Makris, A., Cordeiro, L., Diego, F., Sorokin, P., Girolamo, M.D. and Barone, P., 2023. Security in cloud-native services: A survey. *Journal of Cybersecurity and Privacy*, 3(4), pp.758-793.
27. Ugwueze, V.U., 2024. Cloud native application development: Best practices and challenges. *International Journal of Research Publication and Reviews*, 5(12), pp.2399-2412.
28. Ukene, D.E., 2024. Assessing Performance Optimization Strategies In Cloud-Native Environments Through Containerization And Orchestration Analysis.
29. Venugopal, P., 2025. Deconstructing Cloud-Native Ecosystems: Foundations for Enterprise Scalability and Agility. *Journal of Computer Science and Technology Studies*, 7(8), pp.1137-1144.