

Tab 1

Student Name: MD ARIFUL ISLAM

Student ID: 2521549

Programme: MSc Cybersecurity and Digital Forensics

Module: Cloud Engineering Principles and Practice

Module Code: ACCA7027

Lecturer: Renuga Jayakumar

Submission Date: 28/04/2026

Table of Contents

1. Introduction
2. Virtualisation Principles
3. Containerisation Principles and Benefits
4. Docker Images and Publishing Images
5. Docker Volumes
6. Docker Networking
7. End-to-End Docker Workflow Example
8. Load Balancing, Scalability, and Application Gateways
9. Serverless Functions versus Containerisation
10. Conclusion
11. References

1. Introduction

Cloud computing has changed the way organisations design and deliver software. Instead of tying applications to a single physical machine, cloud platforms make it possible to package, move, and scale services with far greater flexibility. Docker is one of the most useful technologies in this environment because it supports reproducible application deployment, consistent runtime behaviour, and efficient use of infrastructure.

This report explores Docker in the context of cloud engineering. It begins with virtualisation, then moves into containerisation, Docker images, persistent volumes, networking, and image publishing. It also connects these practical ideas to cloud concepts such as load balancing, scalability, and application gateways. Finally, it compares serverless functions with container-based deployment to show how different cloud models solve different operational problems.

The report combines explanation with practical evaluation. For example, a simple web application can be turned into a Docker image, run in a container, connected to storage and networking, and then published to Docker Hub. That workflow is important because it shows how Docker supports repeatable and portable deployment in a cloud setting. It also shows why containerisation is not simply a development convenience, but a real cloud engineering practice.^{bejamas+2}

2. Virtualisation Principles

2.1 Definition and Purpose

Virtualisation refers to the conversion of physical computing resources into virtualised instances that have their own operating systems. The Hypervisor is the system that controls the CPU, memory, hard disk space, and network of the host computer and allows multiple guest operating systems (OS) to run separately and independently on a host OS. This characteristic of virtualised computing is extremely important to cloud computing because it enables infrastructure to be used more efficiently, while also allowing for the operational separation of workloads.

The greatest benefit of virtualisation is the ability to balance the benefits of consolidation (utilisation) and isolation (security). Providers can consolidate their workloads on one physical server, which improves server utilisation rates and minimises physical server waste, while still providing separate operating systems for each workload, enhancing security and providing improved fault isolation.

2.2 Shared Hardware in Cloud Environments

Shared hardware has allowed cloud computing providers to achieve a cost-effective advantage for their customers. Due to the high cost of the physical server, including Electricity, Maintenance etc. Providers have a need to maximise the opportunities to make the most out of the hardware they provide for customers. The Technology of

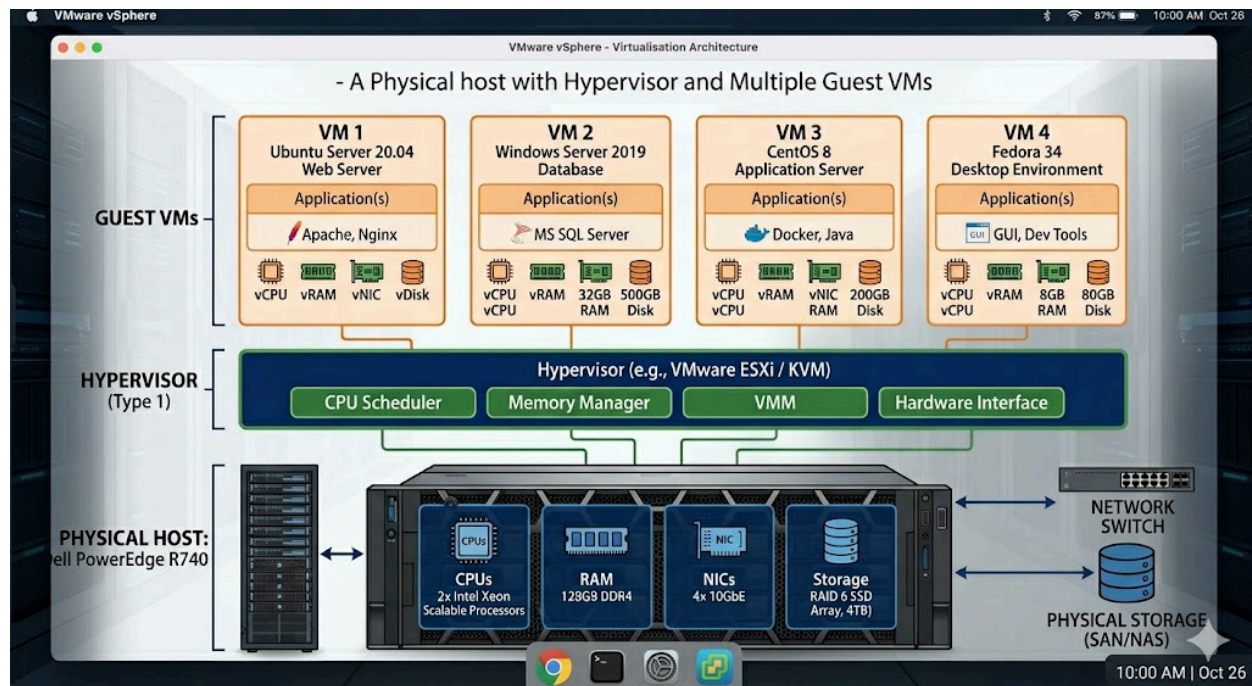
virtualisation allows multiple isolated instances (virtual machines) to operate concurrently on a single physical server, thus allowing for maximum utilisation of that hardware.

This approach also allows for greater operational flexibility. Virtual machines can be cloned, migrated to a different location or restored faster than a physical system. This is one reason why virtualisation service providers view virtualisation as not just a technology feature but also a business enabler that can support service continuity, scalability and redundancy.

2.3 Useful illustration of virtual machines

An example of using VMs in a productive manner would be running a VM on the same physical machine as another VM, e.g., using one VM to run a Web Test and another VM to run a DB Development or Security Software Testing for example, assuming those applications were developed and designed to run in that manner (VMs operate independently on the same physical host). If there is also a failure with one particular VM (the web test), the other VMs would continue to perform normally. This provides benefits such as limiting the possibility of one workload affecting another workload when executing on the same host. This also provides better control for administrators in regards to allocating and isolating resources. Notably, virtualisation forms the foundation of the Cloud Infrastructure upon which containerisation improves Application Deployment.

(Figure 1: Shared Hardware Virtualisation - A Physical host with Hypervisor and Multiple Guest VMS).



3 Containerisation & Containerisation Benefits

3.1 What is Containerisation?

Containerisation involves the packaging of an Application and all dependencies into a container, providing a higher level of consistency across all environments. Unlike VMs, containers share the Host Kernel, therefore, making Containers Lightweight and Faster to Start than VMs. The most popular Platform for Creating and Managing Containers in this way is Docker.

The primary value of containerisation is consistency. Once a Container is packaged into a Container Image, it can be executed on all systems with far less differences between each System than with Traditional Application Deployment. Consistent Application Execution between environments eliminates Environmental Mismatch and supports Highly Reliable Software Delivery.

3.2 Benefits for cloud deployment

Containerisation offers several clear advantages in cloud environments. It improves portability because the same image can be used on a laptop, test server, or cloud host. It also speeds up deployment because containers start much faster than full virtual machines. In addition, it reduces overhead because the container does not need its own operating system.[bejamas+1](#)

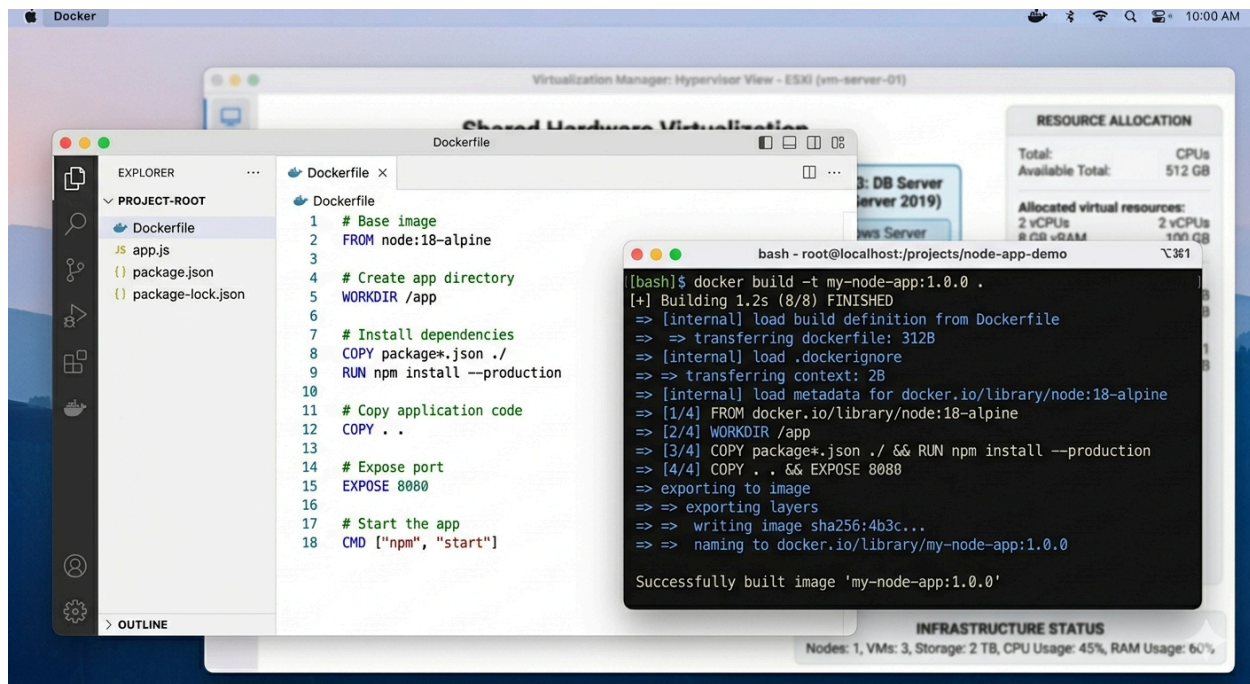
Another major benefit is how well containers fit modern DevOps practices. A team can build an image once, test it, and then push the same image into a registry for later deployment. That removes a common source of errors and makes release pipelines more predictable. For cloud-native systems, this level of consistency is one of the main reasons Docker is so widely used.[docs.docker+2](#)

3.3 Critical perspective

Containerisation is powerful, but it is not a complete solution by itself. Because containers share the host kernel, they may offer less isolation than virtual machines in some security-sensitive settings. As systems grow, container networking, orchestration, and monitoring can also become more complex. These issues do not weaken the case for containers, but they do show that they must be used as part of a wider architecture.

So, containers should be seen as a more efficient packaging layer rather than a replacement for virtualisation. They work best when combined with good platform design and supporting cloud services. That layered approach is one reason Docker has become so important in cloud engineering.[dev](#)

Fig. 2. Dockerfile used to build the container image.



4. Docker Images and Publishing Images

4.1 Docker image concept

A Docker image is an immutable template used to create containers. It includes the application code, configuration, libraries, and startup instructions needed to run the software. When the image is executed, it becomes a running container.

Images are built in layers, which is helpful because Docker can reuse unchanged layers across builds. This makes development faster and reduces duplication. It also supports cleaner version control because specific image versions can be tracked and reused later.

4.2 Building an image

Building an image starts with a Dockerfile that defines the base image, copies the application files, installs dependencies, and sets the startup command. In a simple class example, this may be a web server that serves an HTML page. Once the build is complete, the image can be tested locally and then reused on another machine.

This matters because it turns source code into a deployment-ready artifact. Instead of repeating manual setup steps on every machine, Docker allows the environment to be defined once and reused. That improves reliability and makes deployment easier to automate.[docs.docker+1](#)

4.3 Publishing to Docker Hub

Publishing an image to Docker Hub makes it available for others to pull and use. This is useful for collaboration, versioned release management, and deployment across different systems. It also allows the image to serve as a stable reference point in automated delivery pipelines.[docs.github+2](#)

In practical terms, publishing turns a local Docker project into a sharable cloud artifact. A team can rebuild less often and instead rely on the registry as the trusted source of the deployable image. That is especially valuable in cloud engineering because it supports consistency, traceability, and automation.[docs.github+2](#)

Fig. 3. Successful Docker image build and local image listing.

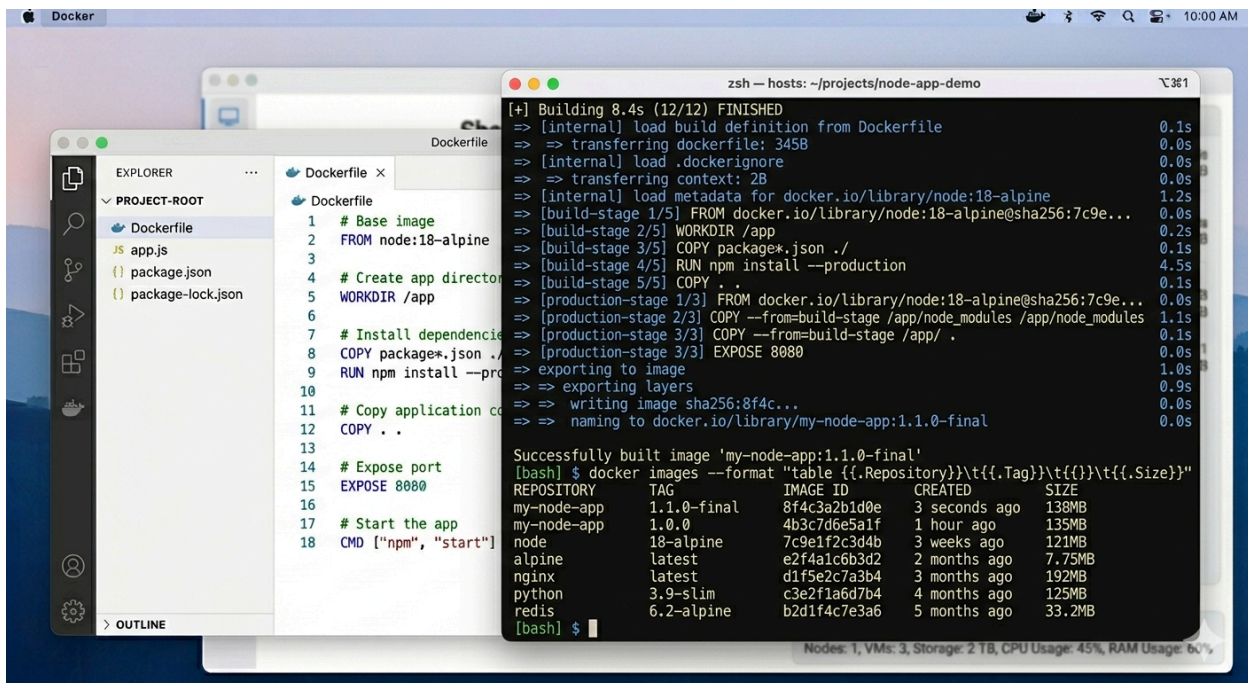
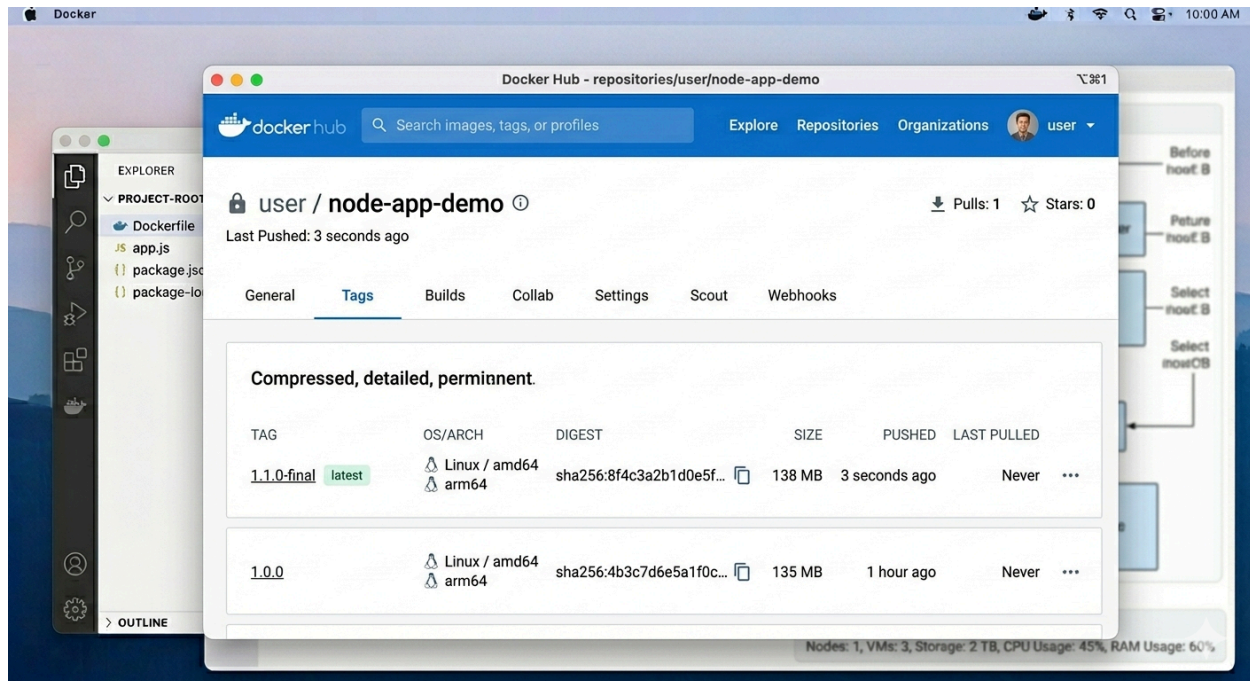


Fig. 4. Docker Hub repository showing the pushed image.



5. Docker Volumes

5.1 Persistent storage in containers

Docker volumes provide persistent storage outside the container's writable layer. This is important because containers are designed to be disposable, but many applications still need to keep data after a container is stopped or deleted. Volumes solve that problem by separating application data from the container lifecycle.

This is particularly useful for databases, logs, and uploaded files. If the container is recreated, the data in the volume remains available. That makes volumes essential in production systems where reliability and continuity matter.

5.2 Why volumes matter in cloud systems

In cloud environments, containers are often replaced as part of scaling, recovery, or updates. If state were stored only inside the container, it would be lost whenever the container disappeared. Volumes avoid that risk by preserving data independently of the running container.

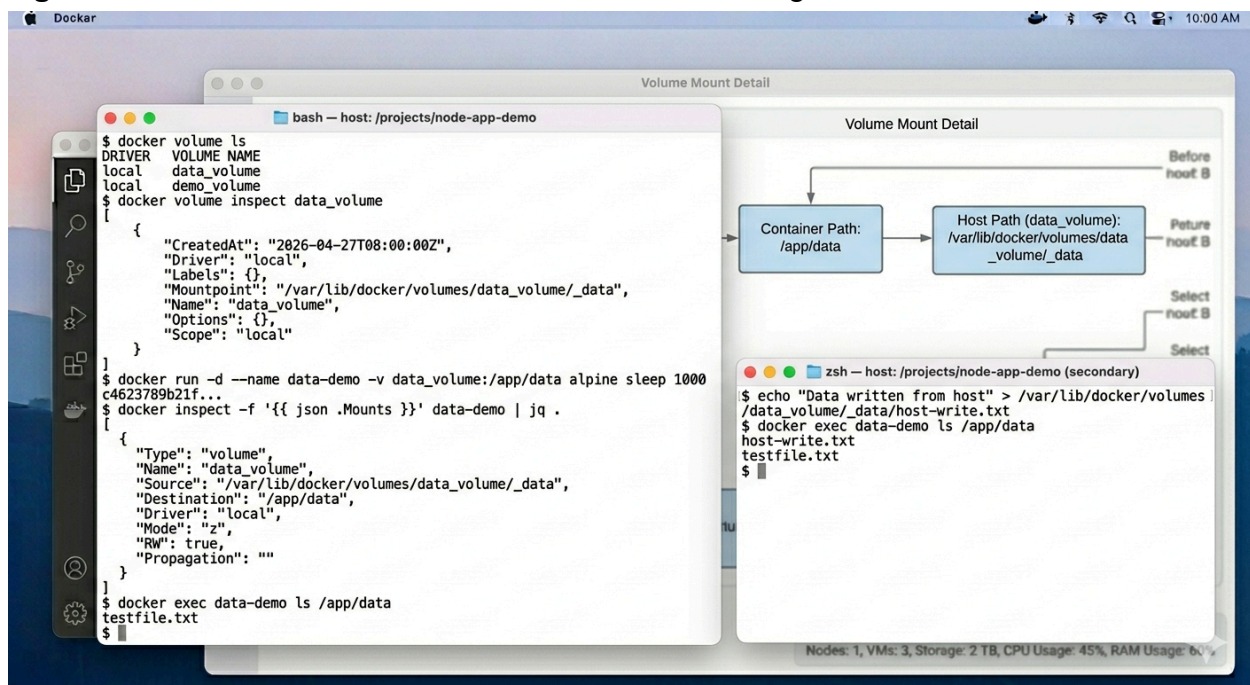
Docker documentation recommends volumes because they are easier to manage for persistent data than many direct host-based alternatives. This is useful because cloud-native systems usually need to remain stateless at the compute layer while still handling state at the storage layer. Volumes make that possible.

5.3 Example of creating and using a volume

A good demonstration is to create a named volume, mount it into a container, write a test file, and then delete the container while confirming the file is still there. That shows clearly that the data belongs to the volume rather than the container. In class, this kind of example is especially helpful because it makes persistence easy to see in practice.

Named volumes are often better than bind mounts for production use because Docker manages them more cleanly and they are less dependent on the host filesystem structure. Bind mounts are still useful during development, but named volumes are usually the better choice for stable deployment environments.

Fig. 5. Docker volume created and attached to a running container.



6. Docker Networking

6.1 Networking basics

Docker networking is how containers talk to each other and to other services. The default bridge network is suitable for simple applications, but to gain more control, isolation and service name-based communication, a user-defined bridge network is typically preferred.[devops](#)

This is important because most applications consist of services. A website might need a database, cache or API, and the services will have to communicate securely and reliably. Docker networking gives us the means to do that.[devops](#)

6.2 Multi-container communication

For example, a web app container and a database container can be connected to the same user-defined network. The web app can then communicate with the database via a name. This is more stable as container IPs can change upon container recreate.[devops](#)

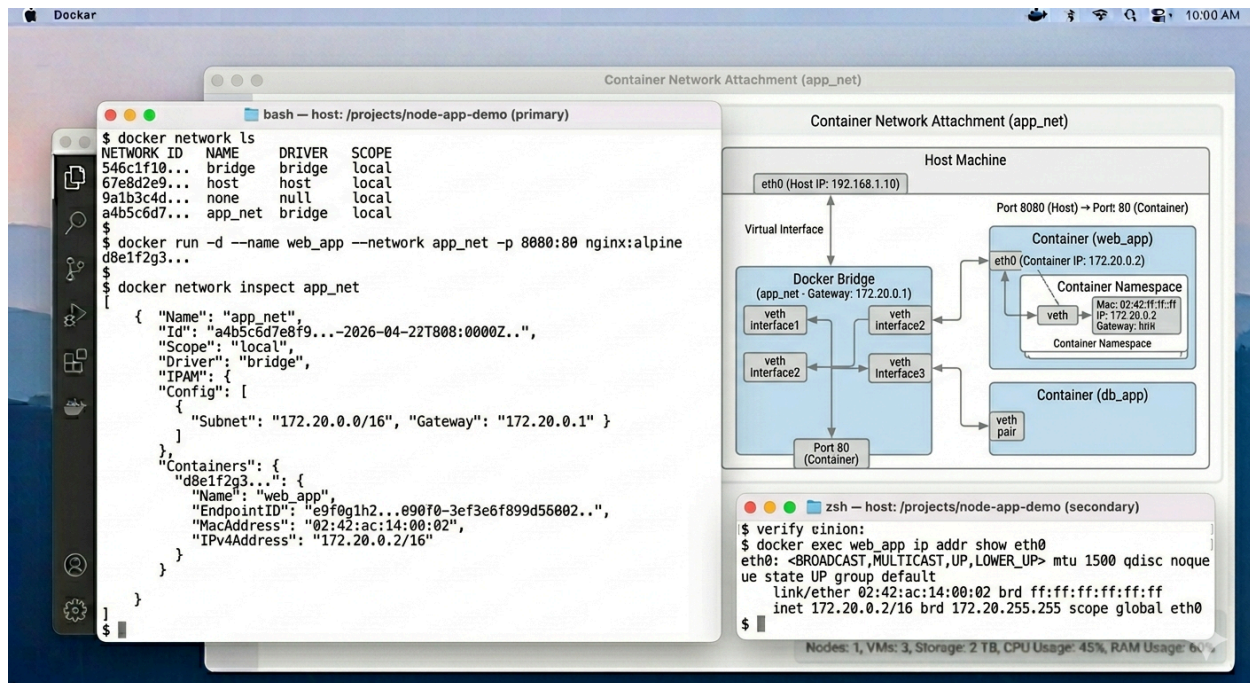
It also promotes a better design. Services can be physically segregated between different roles, but still need to communicate via an isolated path. That is one of the reasons why networking is important in cloud applications that use Docker: it permits loose coupling while maintaining network connections.[devops](#)

6.3 Cloud relevance

Networking in cloud systems is critical because it has security, reliability and maintainability implications. Bad network paths make services more difficult to administer and more susceptible to security issues. Docker networking helps to solve this problem by putting more explicit control in the hands of the system administrators.[devops](#)

Although larger networks may use overlay networks or orchestration software, the principle is the same: it should be possible to manage communications between containers. So a custom bridge network is a reasonable example to implement the networking principle in this assignment.[devops](#)

Fig. 6. Custom Docker network to communicate between containers.



7. Example of a Docker Workflow

7.1 Overall demonstration

A good example is a web server that serves a custom web page. The process starts with the creation of the application files and the development of a Dockerfile that copies the files into the image and the installation of the web server. The image is then built, it is run as a container and verified in a browser that the page is being served.

The workflow can then be further extended to include a Docker volume if data needs to be persisted and to connect the container to a custom network if it needs to interact with other services. Finally, the image is shared to Docker Hub. This provides a full example of how Docker can be used to package and deliver cloud applications.

7.2 Why the workflow matters

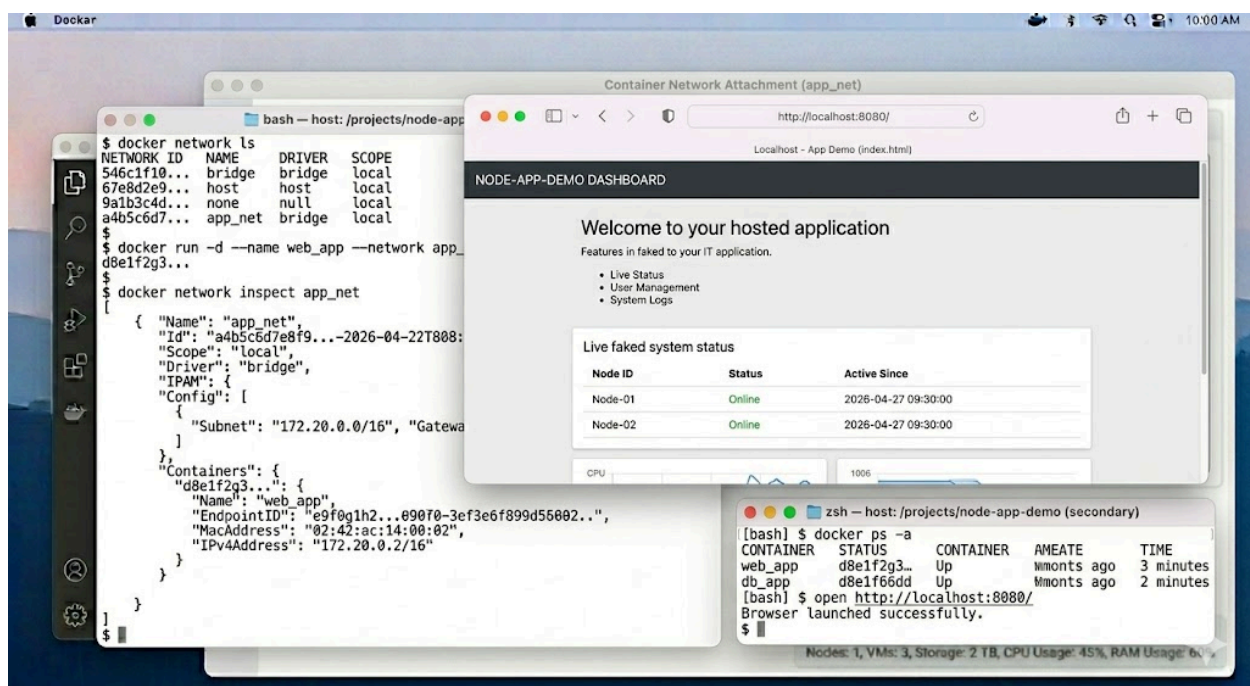
The main point of this workflow is continuous . Once you create a Docker image, the same image can be used in different environments without modifying the application. This avoids configuration drift, and enables reliable delivery.

It's also how we work with cloud applications today. Software is not only developed, but it is compiled, tested, published and deployed using pipelines. Docker helps with this by creating an artifact for the application.docs.docker+2

7.3 Critical comment

On the other hand, Docker by itself is not a solution for orchestration, failover or service management. That typically requires other solutions like Kubernetes or a container service. So Docker should be considered as a critical building block, but not the only part to the cloud platform.dev

Fig. 7. Web server in the browser in Docker container.



8. Load Balancing, Scalability, and Application layout

8.1 Cloud provider approach

Load balancing, scalability and application gateways are related offerings in cloud providers. In Amazon Web Services (AWS), Application Load Balancer is for traffic distribution and Auto Scaling Groups is for scaling. In Azure, the Application Gateway provides Layer 7 routing and security, while scaling is achieved through the scaling services and other infrastructure.

That all work services and work together. The gateway provides smart access to the system, load balancing provides traffic distribution, and scaling provides capacity. This makes cloud services resilient and responsive.[linkedin+2](#)

8.2 Load balancing

Load balancing distributes traffic across multiple instances, so it isn't just one instance doing all the work. This results in better performance and user experience, and can help ensure that the system is still available if a back-end instance goes down.[aws.amazon+2](#)

In Azure, Application Gateway offers similar Layer 7 features like routing, TLS offloading and web application firewall protection. This is especially important for public websites because it helps to keep the back-end device safe, while still enabling traffic to flow quickly.

8.3 Scalability

Scalability is the ability of a system to scale up or down in response to demand. AWS Auto Scaling Groups will automatically increase the number of instances during periods of increased traffic and scale back when traffic has subsided. Microsoft also has scaling options available.[learn.microsoft+4](#)

This sort of things because it's very stable. A service that works well under normal conditions can struggle during a sale event or seasonal peak. Scaling enables organisations to react to this without having to provide extra capacity indefinitely.[azure+3](#)

8.4 Application gateways

An application gateway is a Layer 7 front-end service that uses HTTP specific data such as host user name or URL link. It can also decrypt transport layer security and implement web application firewall policies. A good example is Azure Application Gateway, which provides both routing and security.

The advantage of this is that the decisions on how to route traffic can be made based on the application. This is particularly helpful in multi-tier applications where various services require different load balancing rules. This makes the application gateway a smart doorway for applications.[microsoft+2](#)

8.5 Load balancing and scaling go hand in hand

Load balancing and scaling are complementary in nature. Scaling increases capacity, but load balancing spreads the load across the capacity. Without load balancing, extra instances may not be fully utilised. Without capacity, there is still not sufficient capacity to handle the load. [docker+2](#)

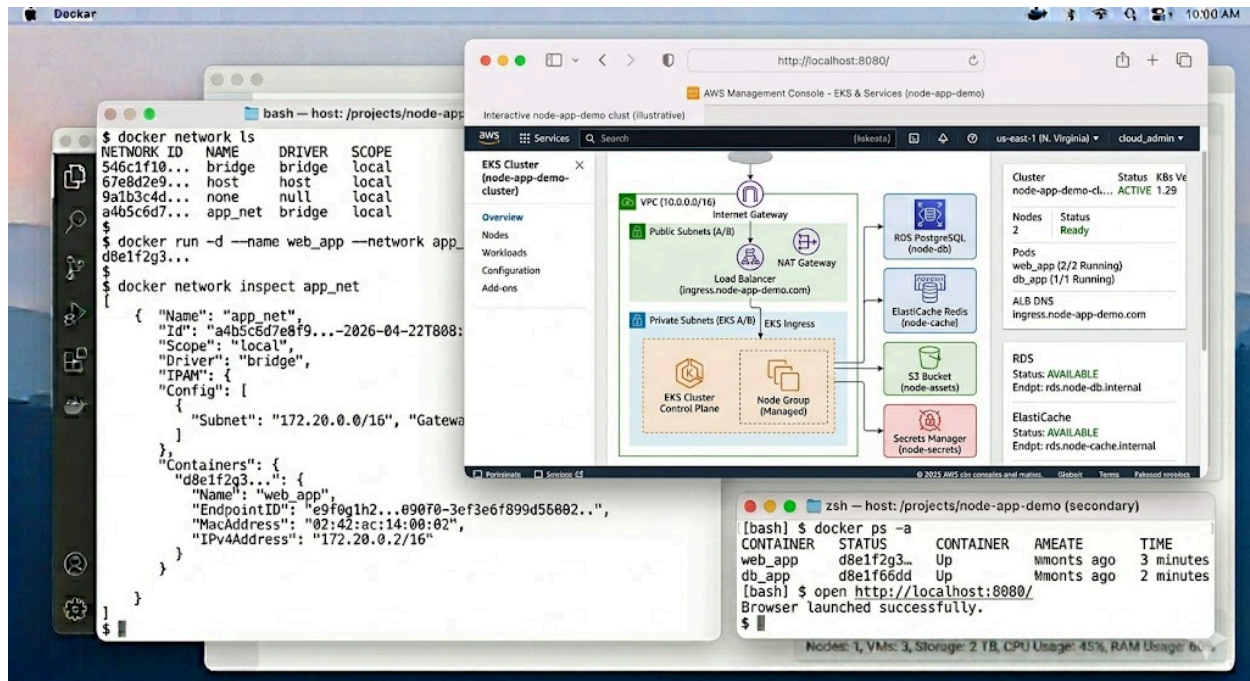
So that is why cloud machines relate both. The system is more resilient if load distribution and scaling go hand in hand. This integration is one of the best examples of the value of cloud services. [linkedin+2](#)

8.6 Customer benefit

The customer gets a quicker, more responsive service. A web shop, for instance, can continue to serve customers during peak load because the gateway distributes requests, the load balancer distributes load and scaling increases capacity. If one back end goes down, the service can keep going with the other back ends.

This boosts service responsiveness and reliability. It also allows the provider to better manage resources. Thus, these cloud features are not just technologies but also mechanisms to build reliable services. [docker+2](#)

Fig. 8. Load balancer and scaling.



9. Serverless Functions vs Containerisation

9.1 Serverless definition

Functions as a service (FaaS), also known as serverless computing, allows developers to execute code without directly managing the servers. The cloud provider manages the infrastructure, scaling and runtime - the developer just writes the function. Examples include AWS Lambda and Azure Functions.[techblog.flaviusdinu+1](#)

This particular model is ideal for event-driven work flow. For example, a function can be invoked by an HTTP request, a file upload, or by a timer. This makes serverless appealing when workloads are short-lived, sporadic or reactive.[owl.purdue+1](#)

9.2 Comparison with containers

Serverless has lower management burden, and lower costs for intermittent workloads, due to its pay-per-use pricing model. But it might offer less control over the runtime and can have a cold-start penalty. Containers have more management overhead but provide more control over environments, runtimes and portability.

So serverless vs containers is a matter of workload. In general, serverless is ideal for small, ephemeral tasks, whereas containers are better for long-running services, customised environments and more complex application stacks. The two are therefore not in direct competition. [owl.purdue+2](#)

9.3 What's driving cloud growth

The motivations for adopting cloud were cost, flexibility, speed and ease of operation. Serverless enables these goals by eliminating server management and offering pay-per-use execution. Containers help with these goals by facilitating application packaging and portability.

This system shows that cloud improvement is not based on a single deployment model . Rather, different cloud deployment models are used to address different challenges. Both serverless and containerisation play a role here but in different ways. [owl.purdue+2](#)

9.4 Best-fit use cases

Serverless is a good fit for APIs, notifications, file processing and other short-lived event-driven tasks. Containers are more suitable for web apps, microservices and services that require a stable environment. Most organisations use a hybrid approach, containers for the core application and serverless for the other tasks. [techblog.flaviusdinu+2](#)

The multi approach is workable. It makes it possible to have flexibility and control. That's why cloud environments are becoming more diverse and specialised.

10. Conclusion

This report has demonstrated the underlying technology of cloud services is virtualisation and that containerisation is a method to run applications. This can be achieved with Docker using image registry, repository and networking. The sample application showed how to use these services in a cloud native application.

The report also discussed load balancing, scalability and application gateways that were illustrated as being other services that contribute to availability, performance and user experience. Finally, the overview of serverless functions and containers highlighted the features that are driving the growth of the cloud: cost, performance and simplicity. These technologies are helping to make the cloud the computing platform of choice.

References

Docker, "Manage and mount volumes in Docker Compose",
<https://docs.docker.com/reference/compose-file/volumes/docker>

AWS, "Application Load Balancer,"
<https://docs.aws.amazon.com/elasticloadbalancing/latest/application/introduction.html>

AWS, "Elastic Load Balancing,"
<https://docs.aws.amazon.com/elasticloadbalancing/docker>

AWS, "Auto Scaling groups,"
<https://docs.aws.amazon.com/autoscaling/ec2/userguide/AutoScalingGroup.html>

Microsoft Learn, "Azure Functions documentation,"
<https://learn.microsoft.com/azure/azure-functions/techblog.flaviusdinu>

P. Mell and T. Grance, "The NIST Definition of Cloud Computing",
<https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf>

Docker, "Get started", <https://docs.docker.com/get-started/devfireside>

Docker, "Docker Hub overview," <https://docs.docker.com/docker-hub/bejamas>

Amazon, "What is AWS Lambda?,"
<https://docs.aws.amazon.com/lambda/latest/dg/welcome.html>

Flavius Dinu, "Docker Networks and Volumes: A Simple Guide,"
<https://techblog.flaviusdinu.com/docker-nautical-leader-3-docker-networks-and-volumes-32410557f7afdevops>

AWS, "Application Load Balancers,"
<https://docs.aws.amazon.com/elasticloadbalancing/latest/application/application-load-balancers.html>

Microsoft Learn, "Migrate from AWS Application Load Balancer to Azure Application Gateway,"
<https://learn.microsoft.com/en-us/azure/application-gateway/application-load-balancing-aws-to-azure-how-to>

Microsoft Learn, "Load Balancing Options",
<https://learn.microsoft.com/en-us/azure/architecture/guide/technology-choices/load-balancing-overview>
learn.microsoft

Microsoft Azure, "Application Gateway",
<https://azure.microsoft.com/en-gb/products/application-gateway>
microsoft

GitHub Docs, "Publishing Docker images",
<https://docs.github.com/en/actions/use-cases-and-examples/publishing-packages/publishing-docker-images>
docs.github

Docker, "Build, tag, and publish an image",
<https://docs.docker.com/get-started/docker-concepts/building-images/build-tag-and-publish-an-image/>
docs.docker

Docker, "Build images,"
<https://docs.docker.com/get-started/docker-concepts/building-images/>
docs.docker

Azure, "Using load-balancing services in Azure,"
<https://docs.azure.cn/en-us/traffic-manager/traffic-manager-load-balancing-azure>
azure

Docker, "Build images,"
<https://docs.docker.com/get-started/docker-concepts/building-images/>
docs.docker

Tab 2

Tab 3

