



ACCA7027

Cloud Engineering Principles and Practice

Lecturer: Renuga Jayakumar

Submitted by-

Md. Mehedi Hasan Mixon

ID- 2436061

Submission deadline: Wednesday ,18th December 2025

Table of Contents

1. Introduction	3
2. Principles of Virtualization and Containerization	
2.1 Principles of Virtualization	5
2.2 Principles and Benefits of Containerization	11
3. Docker Technology in Depth	
3.1 Building Docker Images with Dockerfile	20
3.2 Publishing Images to Docker Hub	23
3.3 Container Volumes	23
3.4 Container Networking	24
3.5 Comprehensive Example: Web Server Deployment	25
4. Core Cloud Computing Concepts: Load Balancing, Scalability, and Application Gateways	
4.1 AWS Approach to Load Balancing	28
4.2 AWS Approach to Scalability	31
4.3 AWS Approach to Application Gateways	34
4.4 Why Load Balancing and Scaling are Complementary	35
4.5 Benefits of Combining All Three	35
5. Driving Forces for Cloud Computing Expansion and Serverless Functions	
5.1 Main Driving Forces: Cost and Flexibility	35
5.2 Serverless Functions (FaaS)	35
5.3 Comparison with Containerization	36
6. Conclusion	36
7. References	37

1. Introduction

Cloud computing has fundamentally transformed modern IT infrastructure, enabling organizations to access scalable, on-demand resources without significant upfront investments in hardware. As of December 16, 2025, the global cloud computing market continues its robust expansion, with public cloud services projected to reach approximately \$723 billion in spending this year, reflecting a year-over-year growth of around 21.5%. This unprecedented growth is primarily driven by the rapid integration of artificial intelligence (AI) and machine learning workloads, which require immense computational power, vast data storage, and real-time processing capabilities that legacy on-premises systems struggle to deliver efficiently.

The acceleration of generative AI adoption across industries has emerged as a dominant force, contributing substantially to market expansion. Organizations are increasingly leveraging AI for automation, customer personalization, predictive analytics, and content generation, all of which rely heavily on cloud infrastructure. In addition to AI, key trends shaping the 2025 landscape include:

- **Hybrid and multi-cloud strategies** to mitigate vendor lock-in, enhance resilience, and optimize costs.
- **Edge computing** integration for low-latency applications, such as IoT deployments and real-time analytics.
- **Sustainability initiatives** with providers focusing on carbon-neutral data centers and energy-efficient operations.
- **Quantum computing as a service** becoming more accessible through cloud platforms.
- **Enhanced security and compliance** features in response to rising cyber threats and stringent regulations.

The major hyperscalers—Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform—collectively command over 60% of the global market share. Meanwhile, specialized providers and regional players are gaining traction by offering tailored solutions for specific industries such as healthcare, finance, and manufacturing.

Within this rapidly evolving ecosystem, containerization technologies like **Docker** remain indispensable for enabling efficient, portable, and scalable application deployment. Docker containers provide lightweight, operating system-level virtualization by packaging applications along with their dependencies, libraries, and configuration files into standardized, executable units. This approach ensures consistent behavior across development, testing, staging, and production environments—eliminating the infamous "it works on my machine" problem.

Docker's significance in cloud engineering stems from several core advantages:

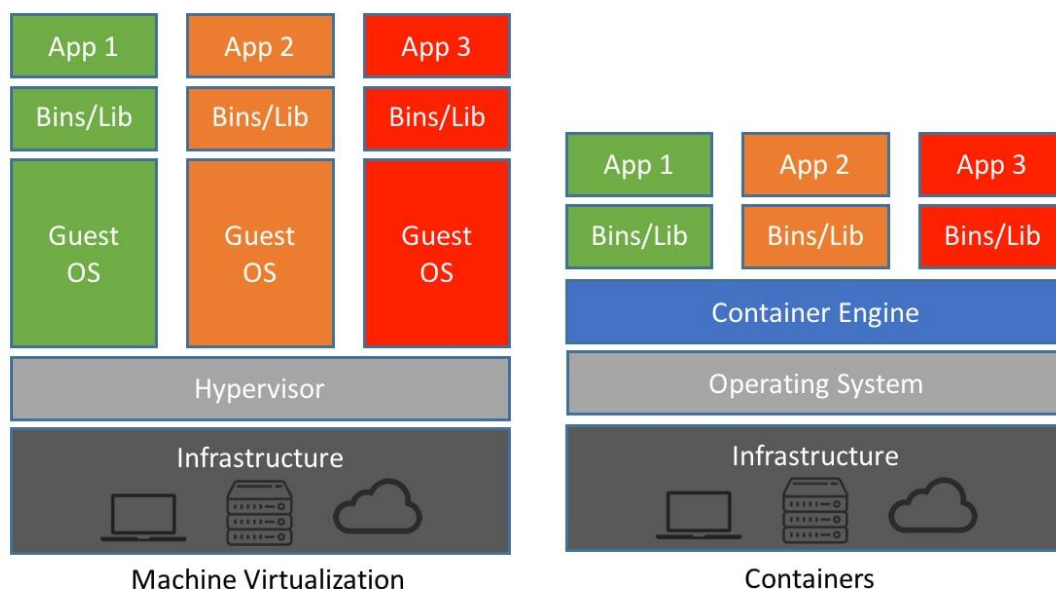
- **Portability** across on-premises, public, private, and hybrid clouds.

- **Resource efficiency** with significantly lower overhead than traditional virtual machines.
- **Support for microservices architectures** and DevOps practices, including continuous integration and continuous deployment (CI/CD).
- **Seamless integration** with orchestration platforms such as Kubernetes, enabling automated scaling, self-healing, and rolling updates.
- **Cost optimization** through higher container density and faster startup times.

In 2025, as organizations increasingly adopt cloud-native and AI-driven applications, Docker facilitates multi-cloud portability, rapid scaling, and reduced operational complexity. Studies indicate that containerized workloads can achieve up to 70% reduction in cloud costs compared to VM-based deployments due to better resource utilization and shorter provisioning times.

This report critically evaluates the role of Docker containerization within cloud computing, drawing on practical class exercises to illustrate key concepts including virtualization principles, containerization benefits, Docker image management, volumes, networking, and end-to-end deployment workflows. It further explores essential cloud computing concepts—load balancing, scalability, and application gateways—using AWS as a reference platform and an e-commerce application as a practical case study. Finally, the report investigates the primary driving forces behind cloud adoption, such as cost savings and operational flexibility, and provides a detailed comparison between traditional containerization and serverless functions (e.g., AWS Lambda) to highlight their complementary strengths in contemporary cloud architectures.

Through this analysis, the report aims to demonstrate how containerization and advanced cloud services collectively enable the development of resilient, efficient, and future-ready systems in today's fast-evolving digital landscape.



2. Principles of Virtualization and Containerization

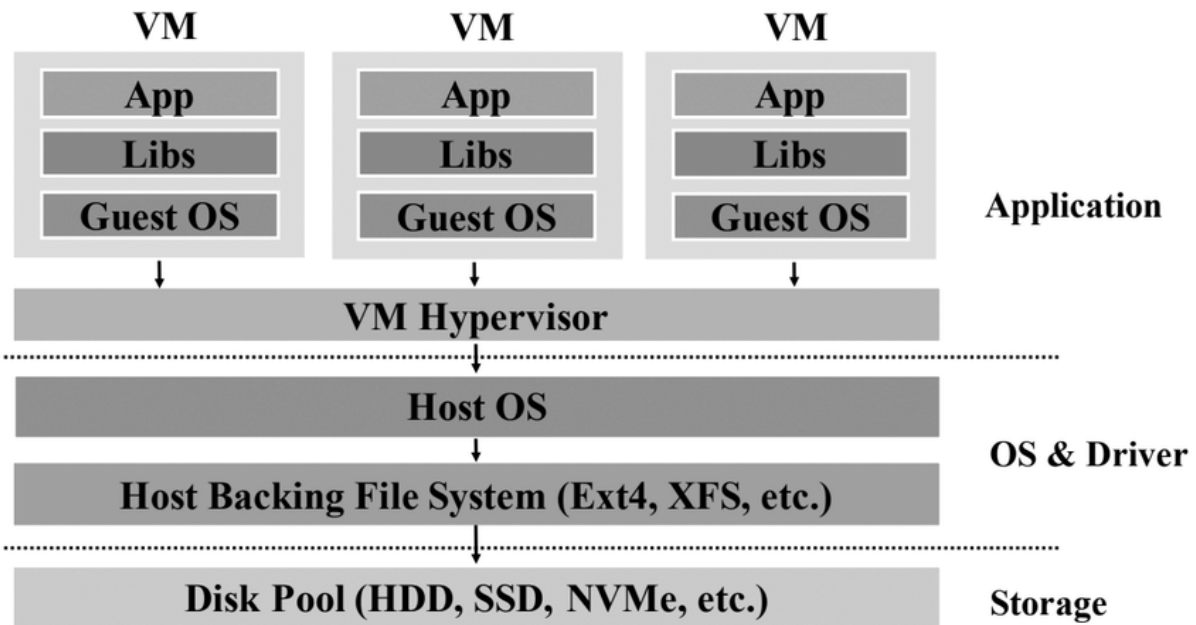
2.1 Principles of Virtualization

Virtualization creates virtual versions of hardware resources using a hypervisor (e.g., VMware, Hyper-V). It allows multiple virtual machines (VMs) to run on one physical server, each with its own guest OS.

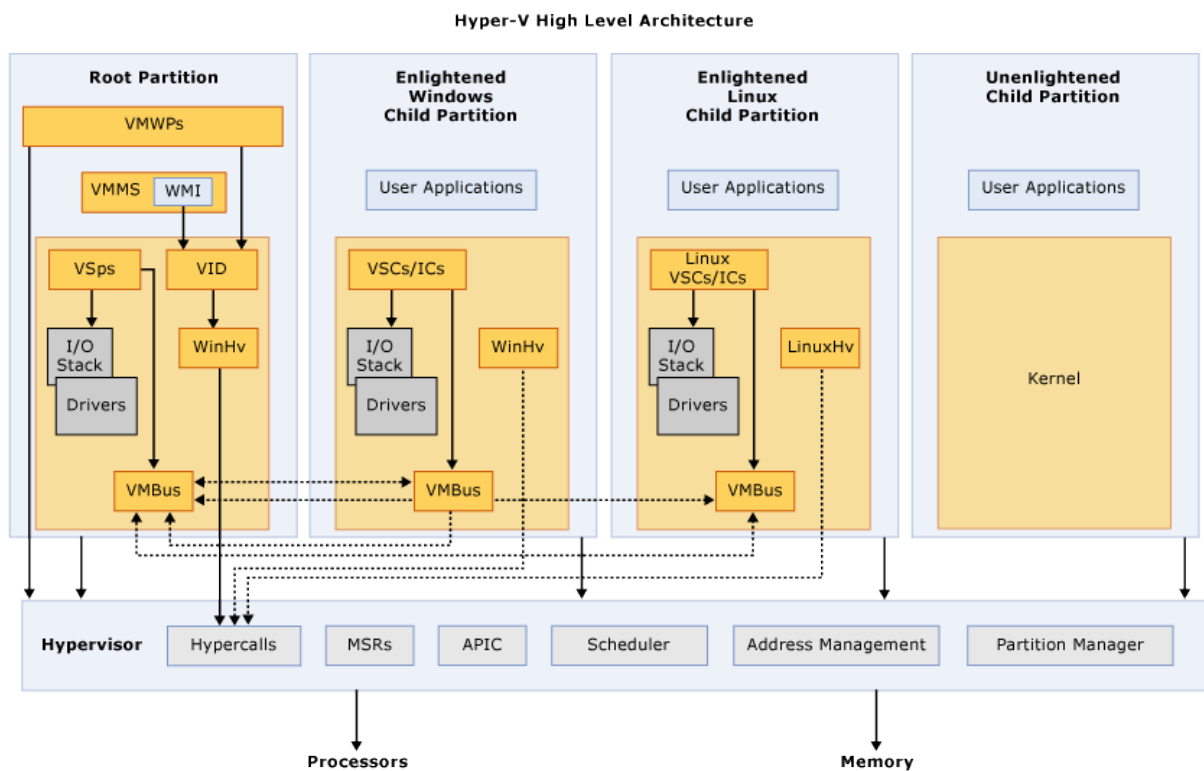
Virtualization is a foundational technology in cloud computing that enables the creation of multiple simulated computing environments from a single physical hardware system. By abstracting physical resources such as CPU, memory, storage, and networking, virtualization allows organizations to run multiple operating systems and applications simultaneously on the same server, maximizing hardware utilization and enabling efficient resource allocation. This abstraction is managed by a **hypervisor** (also known as a virtual machine monitor or VMM), which acts as an intermediary layer between the physical hardware and the virtual environments.

Key Principles of Virtualization:

- **Resource Pooling** — Physical hardware resources are combined into a shared pool that can be dynamically allocated to virtual machines (VMs) based on demand.
- **Isolation** — Each VM operates independently, with its own operating system, applications, and configurations. This ensures that processes in one VM do not interfere with others, providing security and fault isolation.
- **Abstraction** — Virtualization hides the underlying hardware details from guest operating systems, presenting a standardized virtual hardware interface.
- **Elasticity** — Resources can be scaled up or down dynamically, supporting on-demand provisioning and efficient handling of variable workloads.
- **Portability and Migration** — VMs can be easily moved (live migration) between physical hosts without downtime, enhancing disaster recovery and load balancing.



[researchgate.net](https://www.researchgate.net)



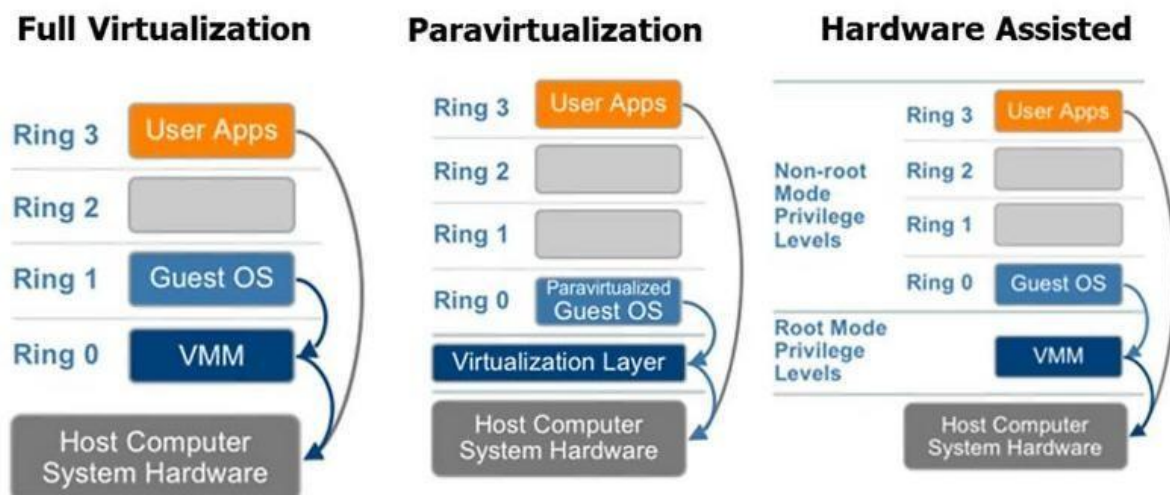
learn.microsoft.com

Types of Virtualization Techniques: Virtualization can be implemented in several ways, each balancing performance, compatibility, and resource efficiency.

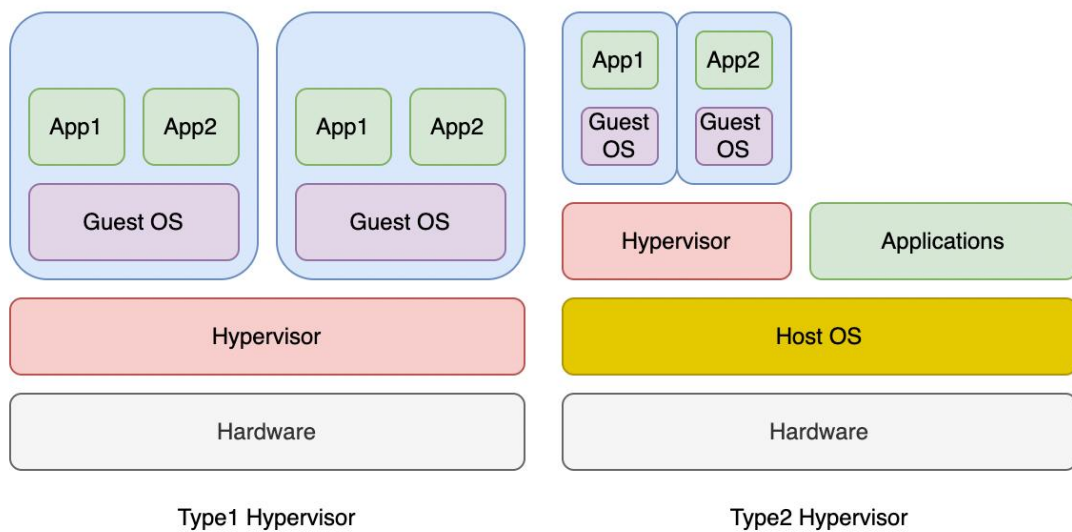
- **Full Virtualization** — The hypervisor fully emulates the underlying hardware, allowing unmodified guest operating systems to run as if on physical hardware. This is achieved through techniques like binary translation or hardware-assisted methods. It offers excellent compatibility and isolation but may introduce performance overhead due to emulation.
- **Paravirtualization** — The guest operating system is aware of the virtualization layer and communicates directly with the hypervisor using hypercalls. This reduces overhead and improves performance but requires modifications to the guest OS kernel, limiting portability.
- **Hardware-Assisted Virtualization** — Modern processors (e.g., Intel VT-x or AMD-V) provide native instructions for virtualization, enabling efficient full virtualization without binary translation. This approach combines the compatibility of full virtualization with near-native performance.

Comparison of Hypervisor Types – Full Virtualization, Paravirtualization, and Hardware-Assisted.

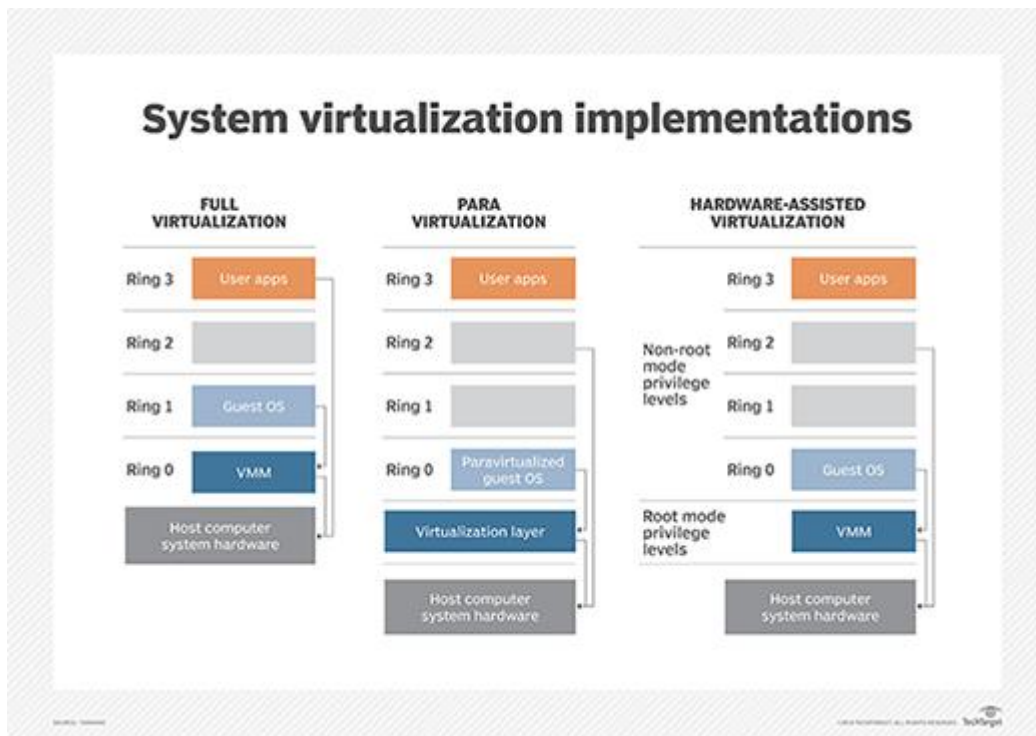
Architectural Comparison



[linkedin.com](https://www.linkedin.com)



mohitdtumce.medium.com



techtarger.com

Hypervisor Types:

- **Type 1 (Bare-Metal)** — Runs directly on hardware (e.g., VMware ESXi, Microsoft Hyper-V, KVM, Xen). These offer high performance and are common in cloud environments.
- **Type 2 (Hosted)** — Runs on top of a host OS (e.g., VMware Workstation, VirtualBox). These are suitable for development and testing.

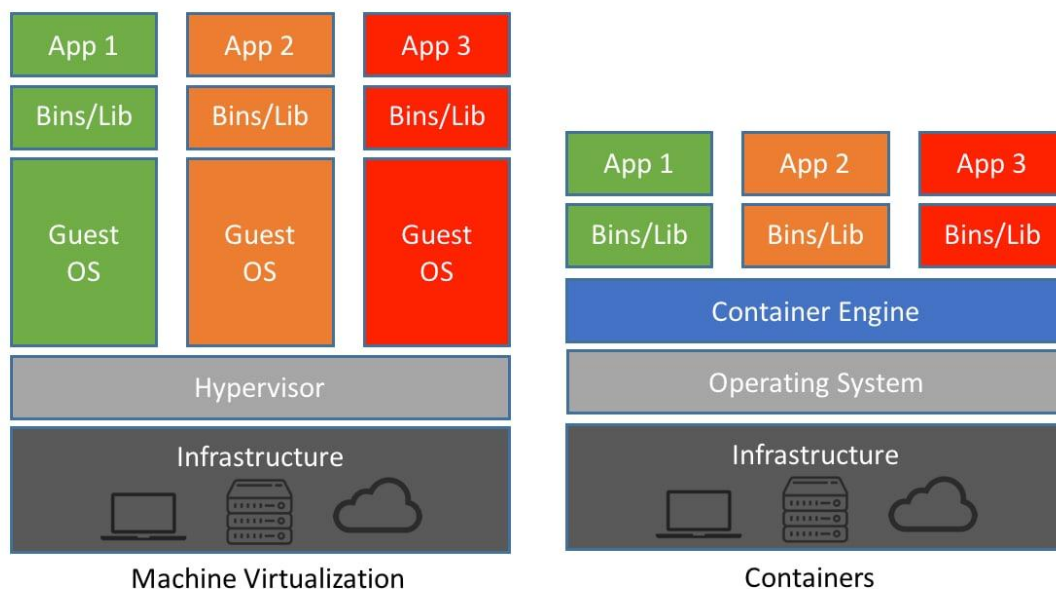
Common Hypervisor Examples:

- **VMware ESXi** — Enterprise-grade Type 1 hypervisor with advanced management features.
- **Microsoft Hyper-V** — Integrated with Windows Server, ideal for Microsoft ecosystems.
- **KVM (Kernel-based Virtual Machine)** — Open-source, integrated into Linux kernel, widely used in cloud platforms.
- **Xen** — Supports both paravirtualization and hardware-assisted modes, powering services like AWS EC2.

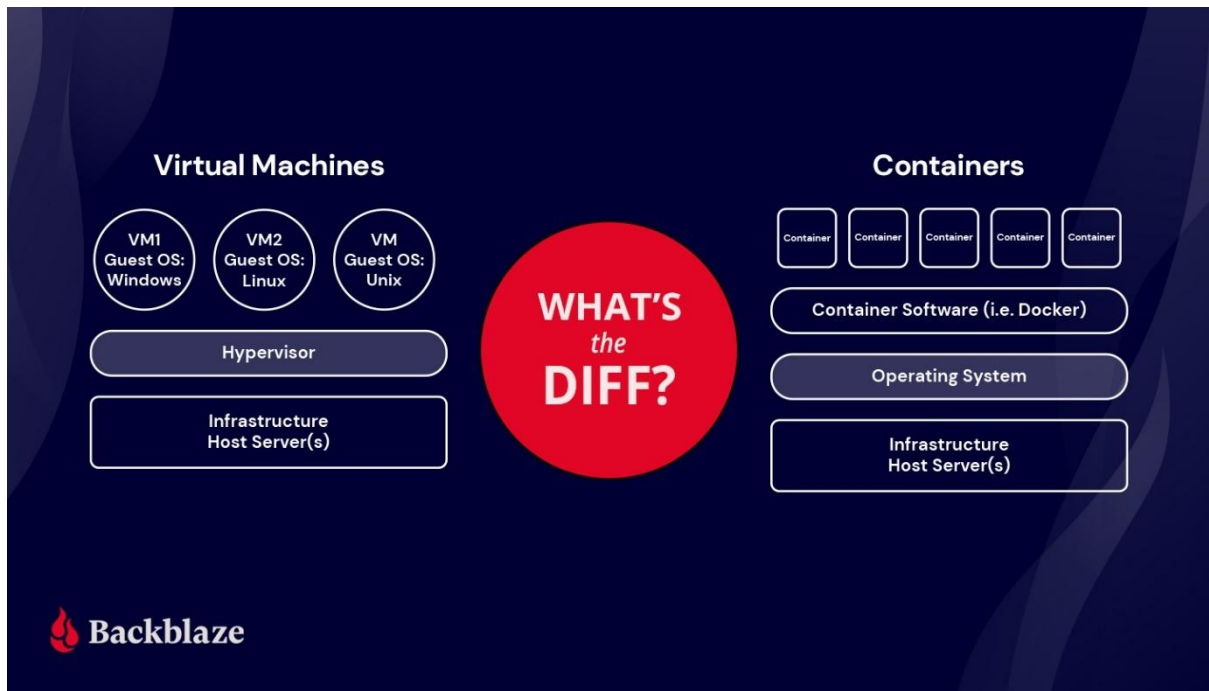
Benefits and Challenges: Virtualization provides significant advantages, including cost savings through hardware consolidation, improved scalability, faster provisioning, and enhanced disaster recovery. However, challenges include potential performance overhead (in non-hardware-assisted setups), security risks from shared resources, and management complexity (e.g., VM sprawl).

Example: In class exercises, we explored virtualization using tools like VirtualBox (Type 2 hypervisor), where a host machine runs multiple guest VMs with isolated resources. This demonstrates efficient hardware sharing but highlights the overhead compared to containerization.

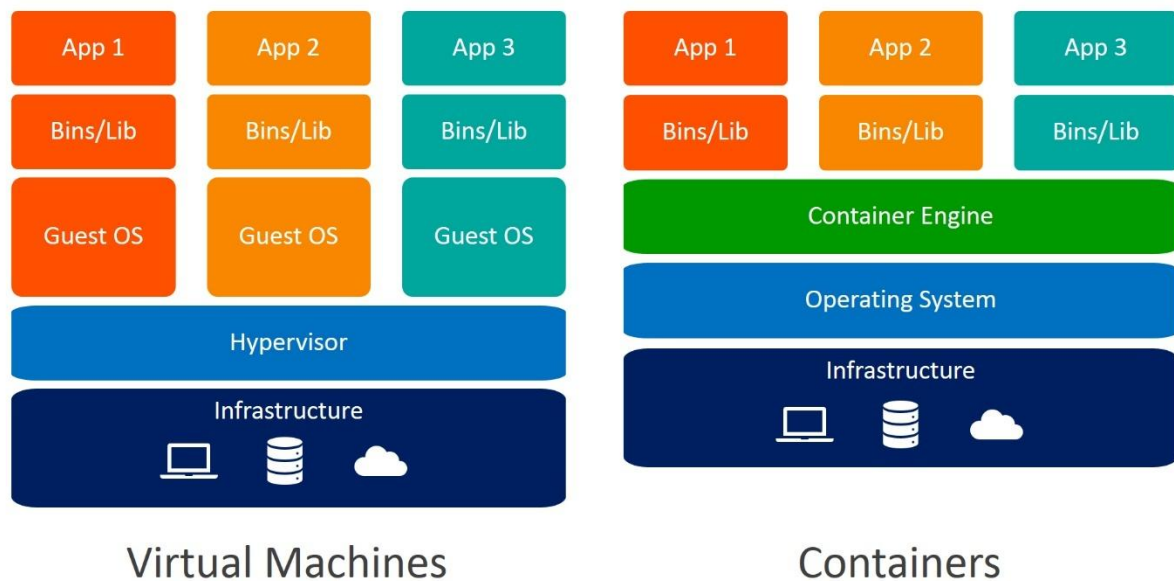
Comparison of Virtual Machines (VMs) vs. Containers.



netapp.com



backblaze.com



bmc.com

Virtualization remains a cornerstone of cloud computing in 2025, supporting multi-cloud strategies, AI workloads, and hybrid environments while evolving with hardware advancements and containerization integration.

2.2 Principles and Benefits of Containerization

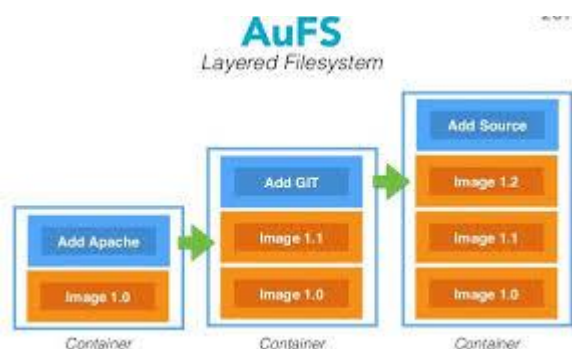
Containerization packages applications with dependencies into lightweight units that share the host OS kernel. Docker popularized this approach.

Containerization is a lightweight virtualization technology that packages an application and all its dependencies—code, runtime, system tools, libraries, and configurations—into a single, standardized unit called a **container**. Unlike traditional virtual machines, containers run directly on the host operating system's kernel, sharing it among multiple isolated containers. This approach provides near-native performance while maintaining strong isolation through Linux kernel features such as **namespaces** (for process, network, and filesystem isolation) and **cgroups** (for resource limiting and accounting).

Core Principles of Containerization:

- **Isolation** — Each container runs in its own isolated user space with private view of the filesystem, network stack, and process tree, preventing interference between applications.
- **Portability** — Containers are self-contained and include everything needed to run the application, ensuring consistent behavior across different environments (development, testing, production, or different cloud providers).
- **Immutability** — Container images are read-only templates. Once built, they remain unchanged, promoting reproducibility and predictability.
- **Layered Architecture** — Docker images are built from stacked read-only layers using union file systems (e.g., OverlayFS). Each instruction in a Dockerfile creates a new layer, enabling efficient caching, storage, and distribution.

Docker Container Layers Illustration.



stackoverflow.com



DOCKER IMAGE LAYERS

www.learnitguide.net

1. Started Image Layer 3 as a container and accessible by users

1. Started Image Layer v1 as a container.
2. Installed and Configured https web server.
3. Committed new layer v2

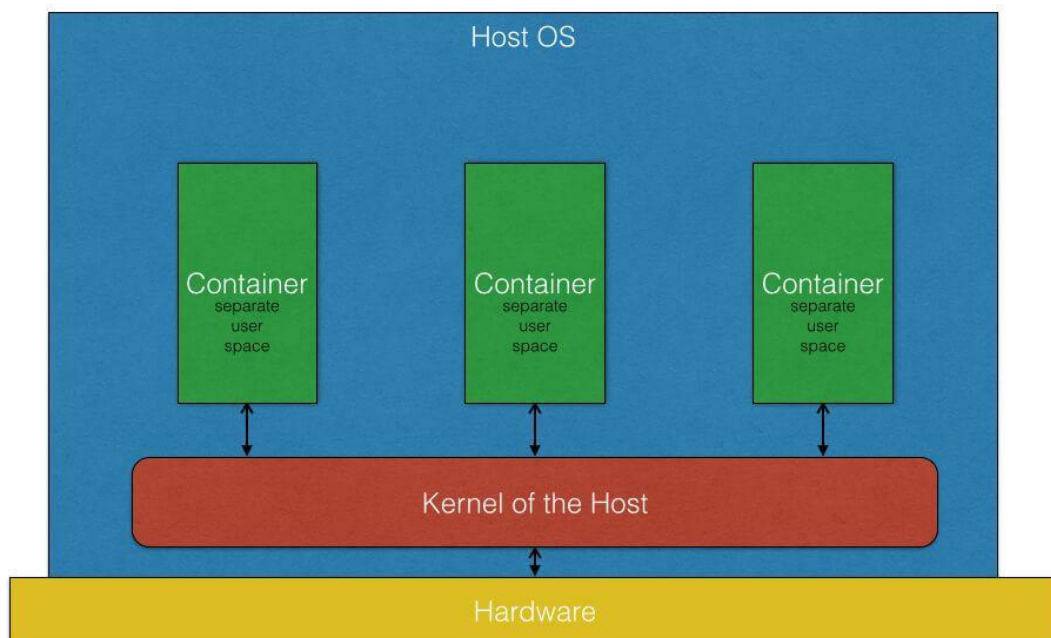
1. Started Base Image (**docker.io/centos**) as a container.
2. Package Updated on Base Image using "yum update".
3. Committed new layer v1

Pulled CentOS image from Docker Hub using docker pull command. **Repo: docker.io/centos**

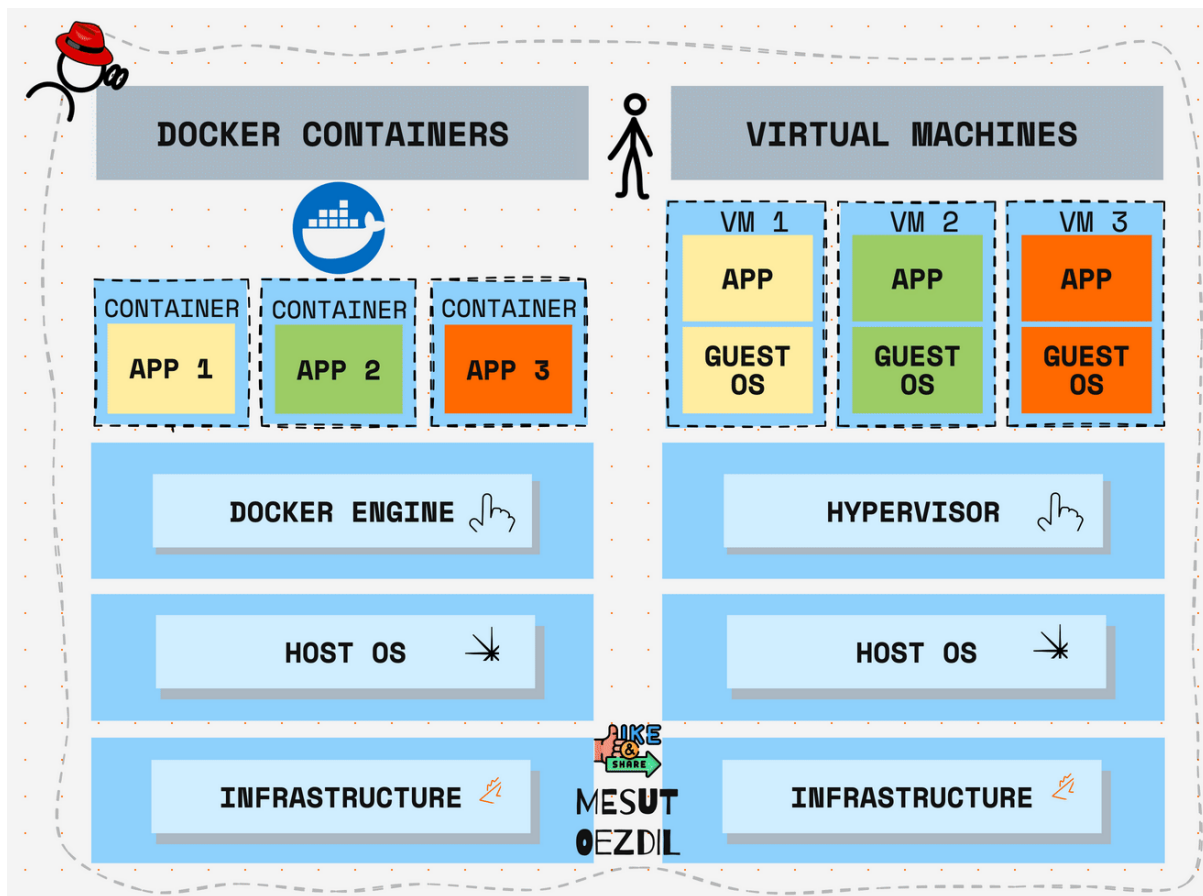


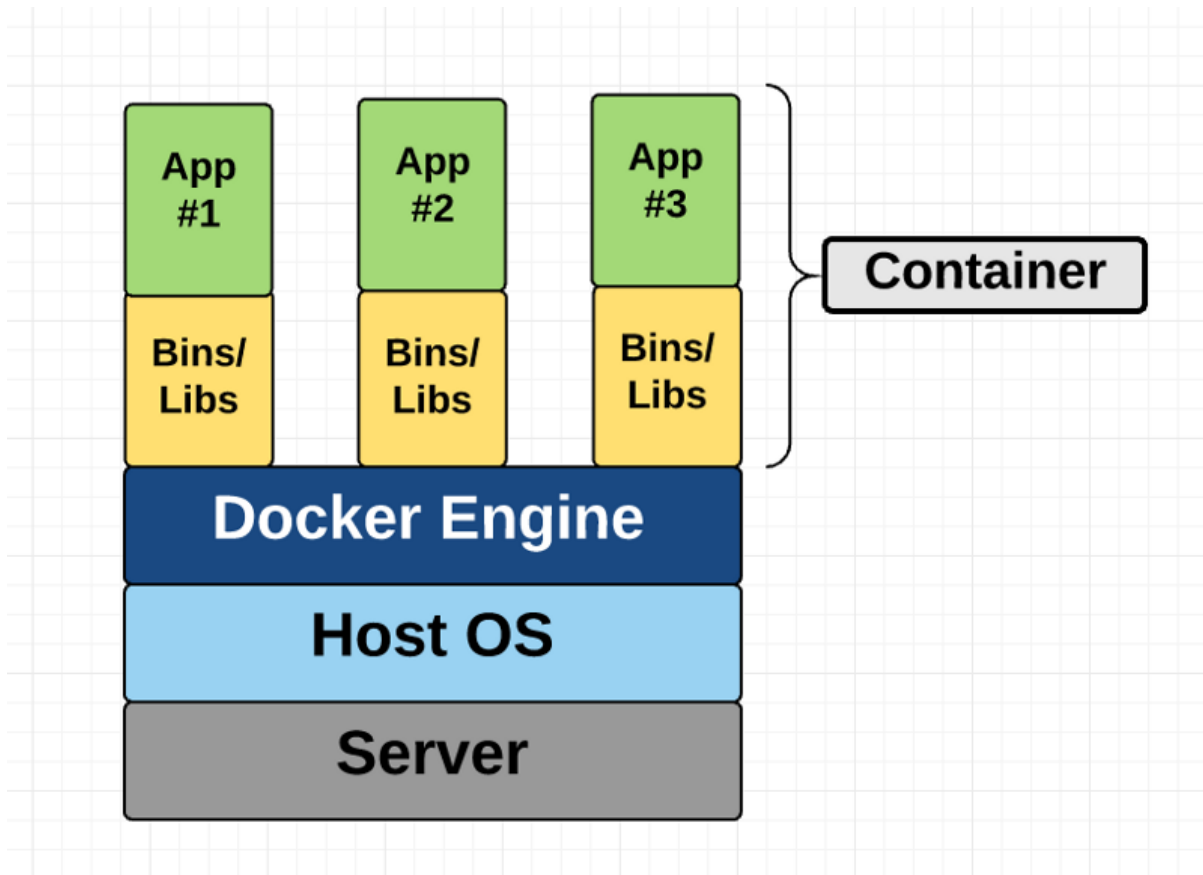
medium.com

- **Efficiency** — Containers share the host kernel and avoid the overhead of running multiple guest operating systems, resulting in faster startup times (milliseconds vs. seconds for VMs) and higher density on the same hardware.



Operating System/Container Virtualization

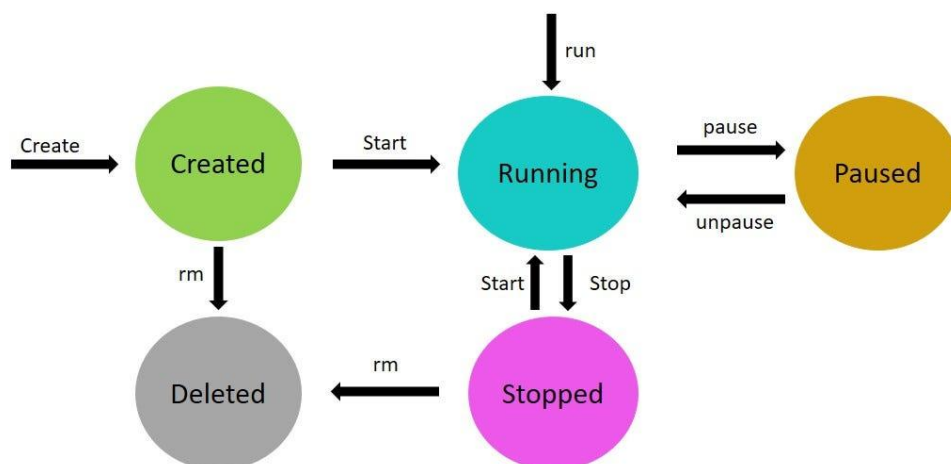




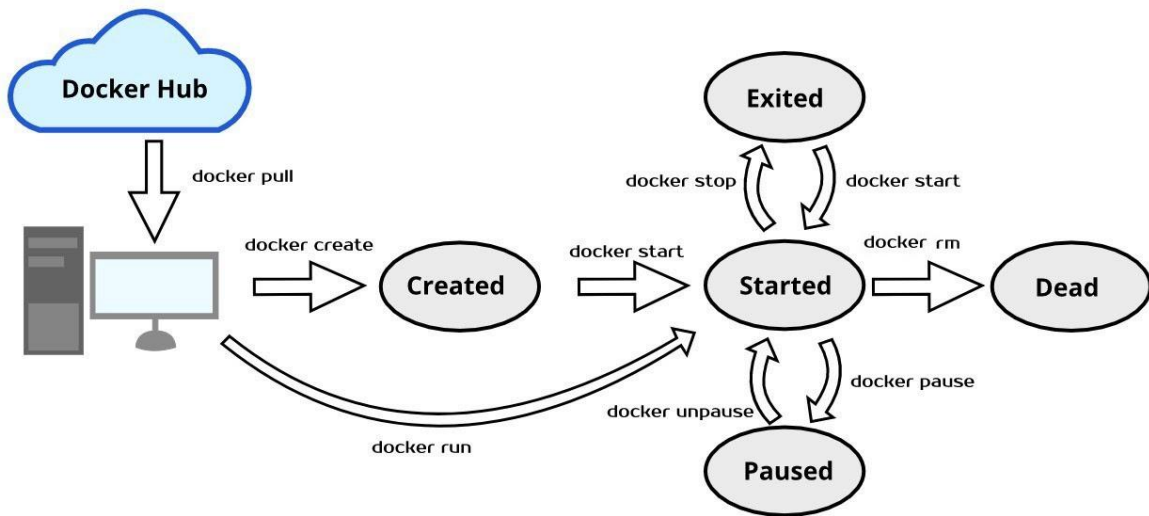
[freecodecamp.org](https://www.freecodecamp.org)

Container Lifecycle: Containers follow a clear lifecycle: created from an image, started, running, paused, stopped, and removed. This deterministic process supports automation and orchestration.

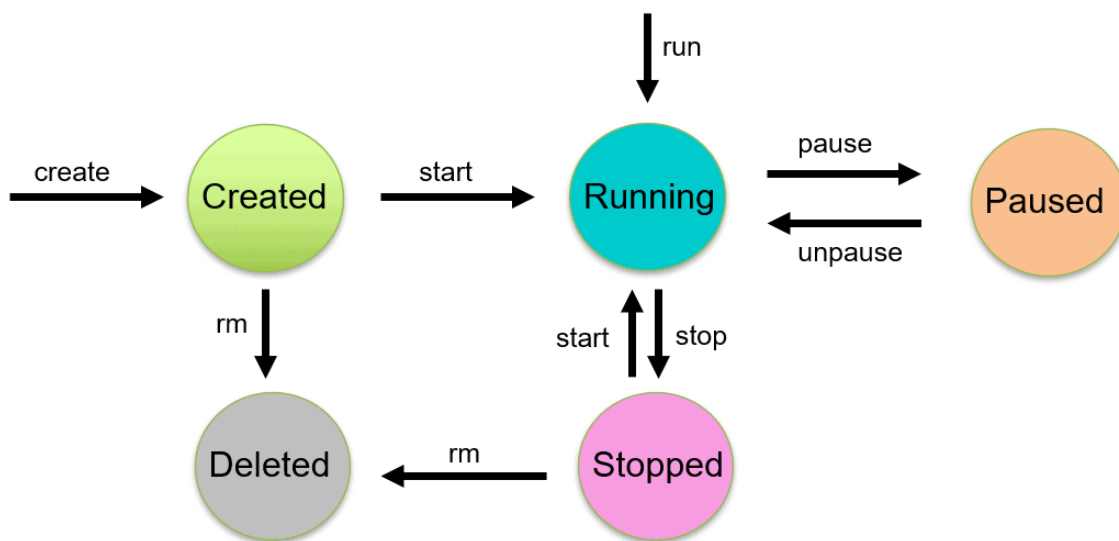
Docker Container Lifecycle Management



medium.com

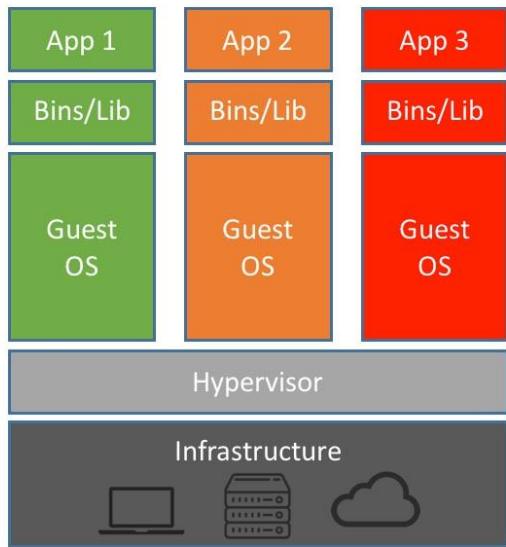


academy.jahia.com

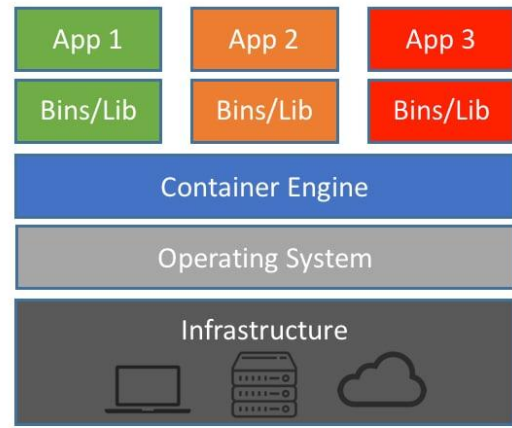


k21academy.com

Comparison with Virtual Machines: Containers and VMs both provide isolation, but they differ significantly in architecture and resource usage.

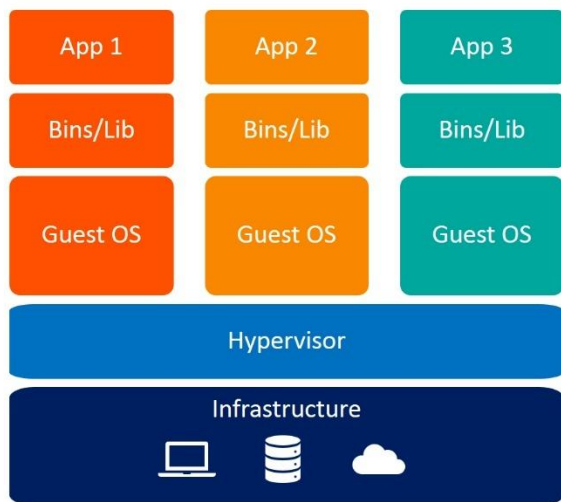


Machine Virtualization

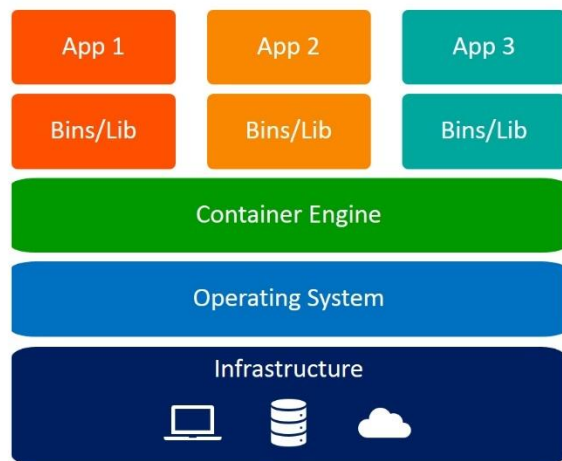


Containers

netapp.com



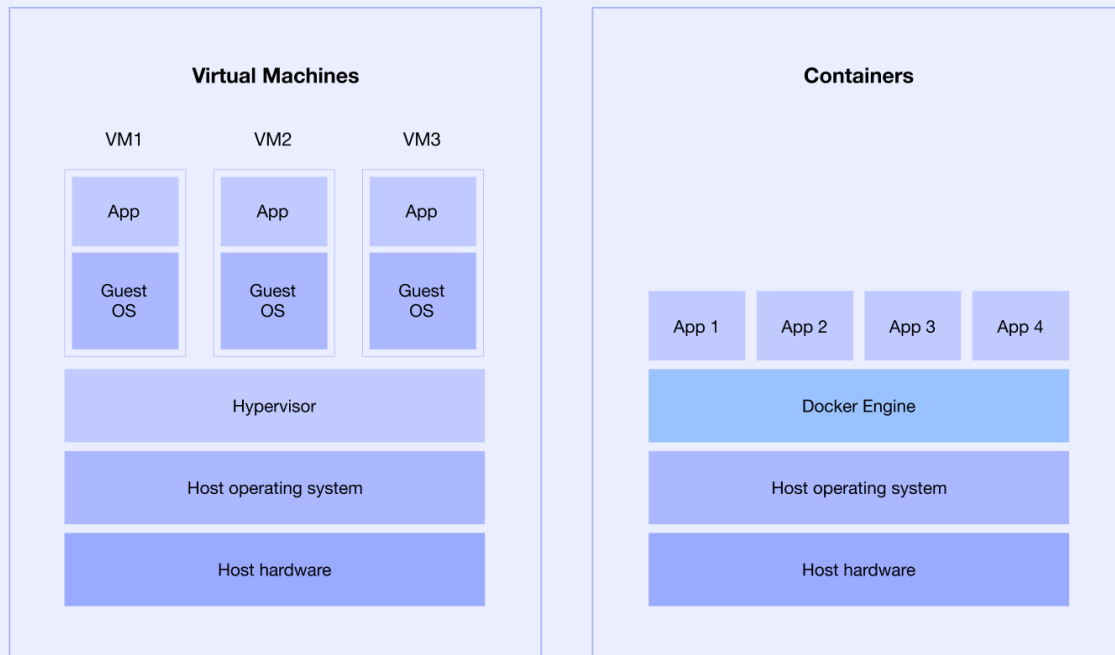
Virtual Machines



Containers

bmc.com

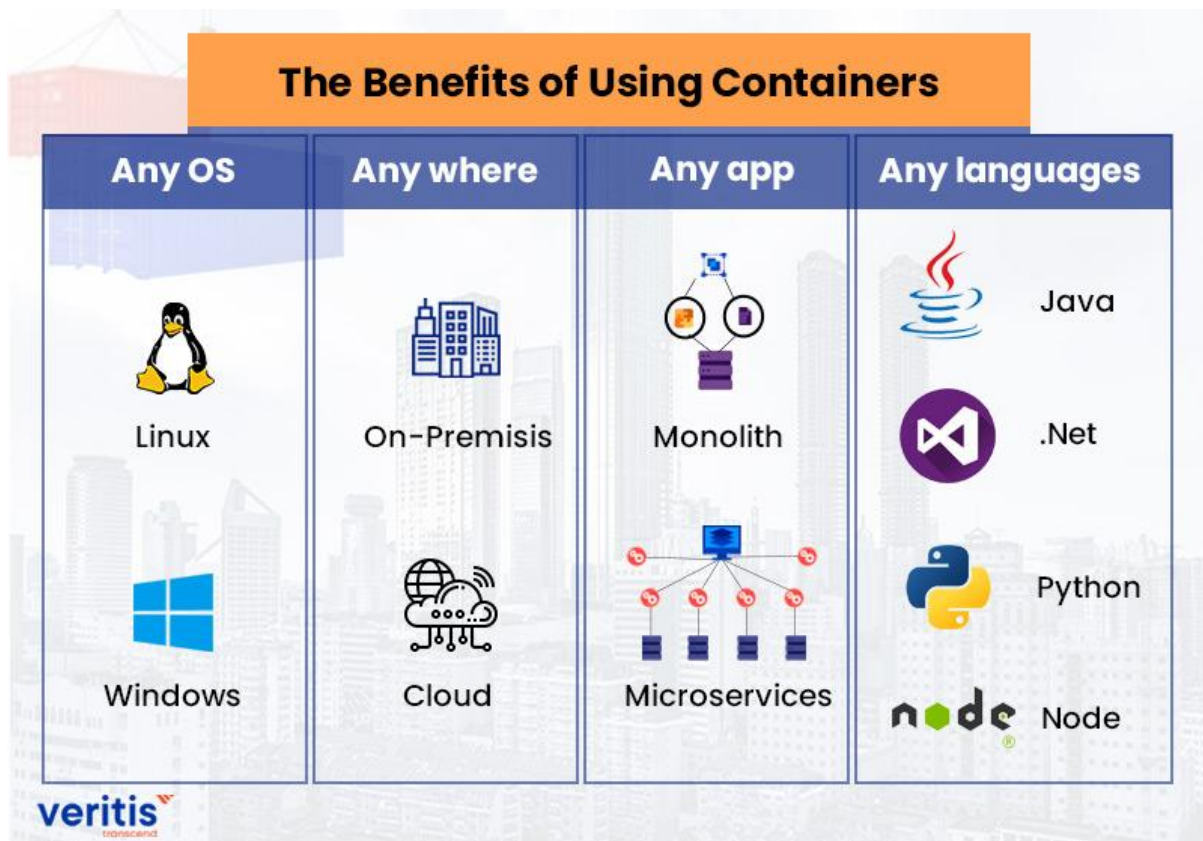
Virtual machines versus containers



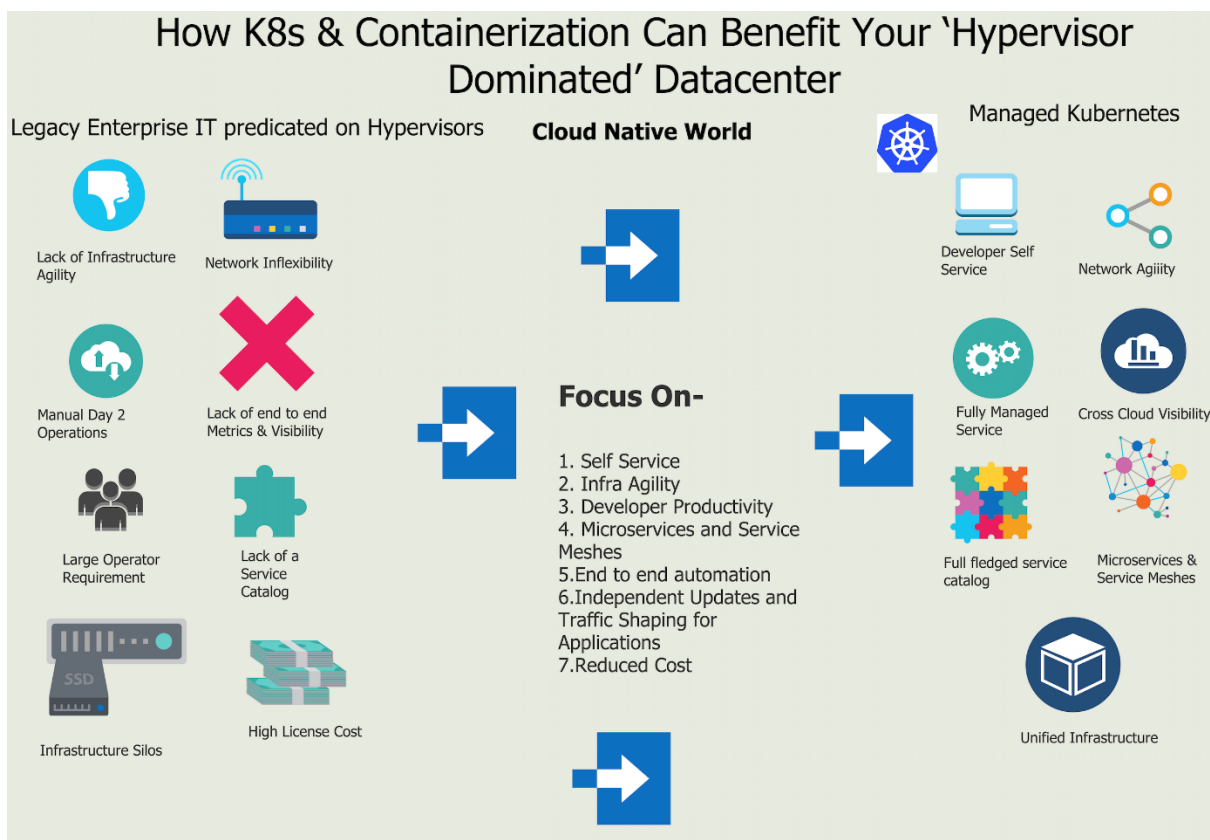
simform.com

Key Benefits of Containerization:

- **Consistency and Reliability** — Eliminates "works on my machine" issues by ensuring identical environments across the entire development-to-production pipeline.
- **Resource Efficiency** — Higher density (many more containers than VMs on the same hardware), lower overhead, and faster startup times.
- **Rapid Deployment and Scaling** — Containers start in seconds and scale horizontally easily, making them ideal for microservices and cloud-native applications.
- **Improved DevOps Practices** — Supports CI/CD pipelines, version control of application images, and seamless integration with orchestration tools like Kubernetes.
- **Cost Optimization** — Reduces infrastructure costs through better resource utilization and enables pay-per-use cloud models.
- **Enhanced Security** — Process-level isolation, immutability, and minimal attack surface (no full OS per container) improve security compared to VMs in many scenarios.
- **Ecosystem and Community** — Docker's vast ecosystem, including Docker Hub for sharing images, accelerates development and adoption.



veritis.com



vamsitalkstech.com

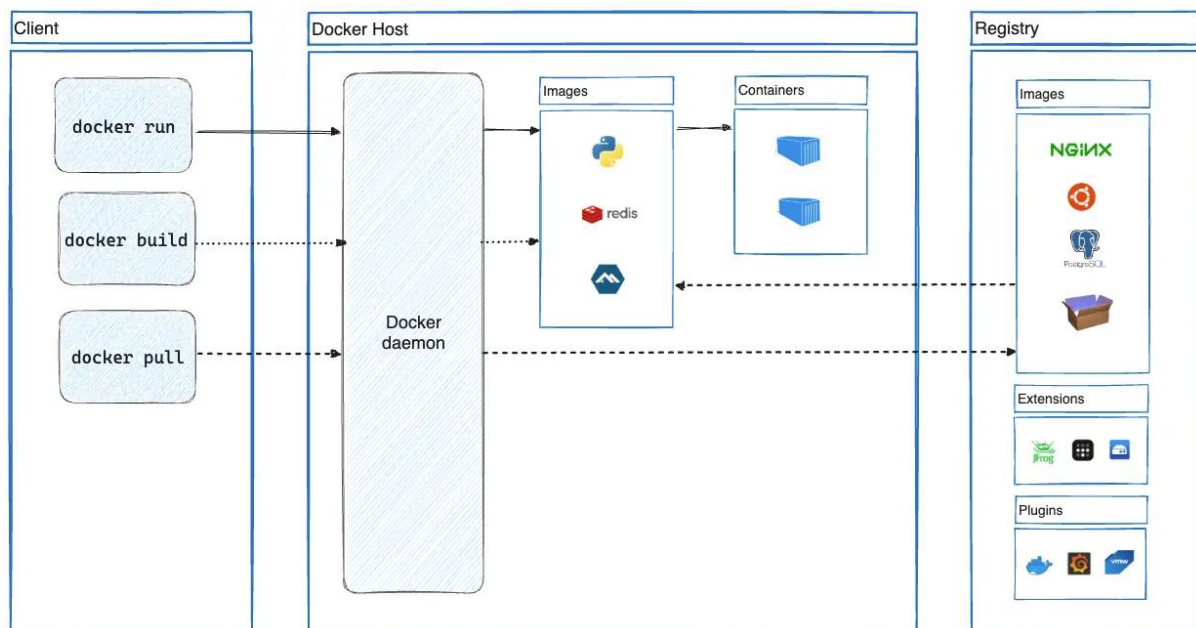


veritis.com

In 2025, containerization is a cornerstone of cloud-native computing, powering microservices architectures, serverless platforms, and AI/ML workloads. Docker's design enables organizations to build, ship, and run applications efficiently, securely, and at scale across hybrid and multi-cloud environments.

3. Docker Technology in Depth

Docker is a leading platform for containerization that enables developers to build, ship, and run applications in isolated, portable containers. It simplifies deployment in cloud environments by addressing challenges like dependency management, environment consistency, and scalability. Docker's architecture consists of a client-server model: the Docker client interacts with the Docker daemon (dockerd), which manages containers, images, networks, and volumes.



docs.docker.com

What is Docker? | Docker Docs

3.1 Building Docker Images with Dockerfile

A Dockerfile is a text file containing instructions to automate the creation of a Docker image. Each instruction creates a cached layer, allowing Docker to reuse previous layers for faster rebuilds and efficient storage.

Key Dockerfile instructions include:

- **FROM** — Specifies the base image
- **WORKDIR** — Sets the working directory
- **COPY** or **ADD** — Copies files into the image
- **RUN** — Executes commands during build
- **EXPOSE** — Declares ports
- **CMD** or **ENTRYPOINT** — Defines the default command

Example Dockerfile for a Node.js Web Server (from class exercise):

Dockerfile

```
# Use a lightweight base image
FROM node:18-alpine

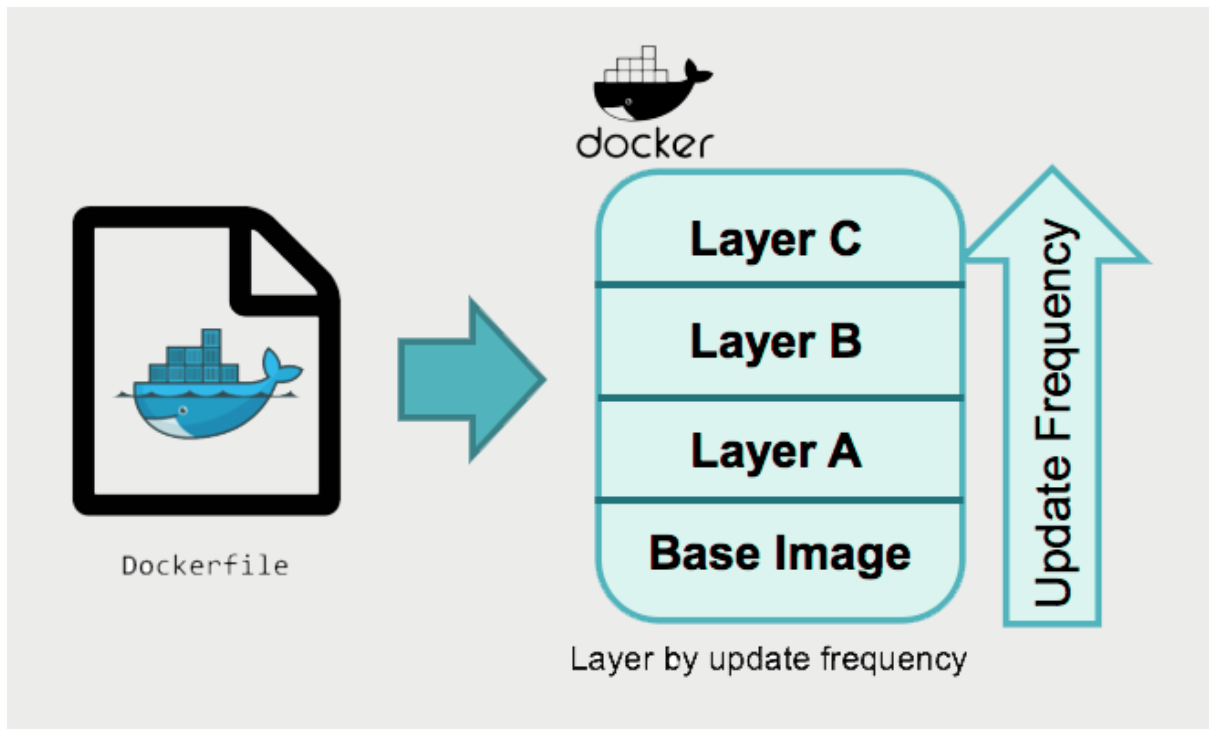
# Set working directory
WORKDIR /app

# Copy package files and install dependencies first for layer caching
COPY package*.json ./
RUN npm install

# Copy application code
COPY . .

# Expose port
EXPOSE 3000

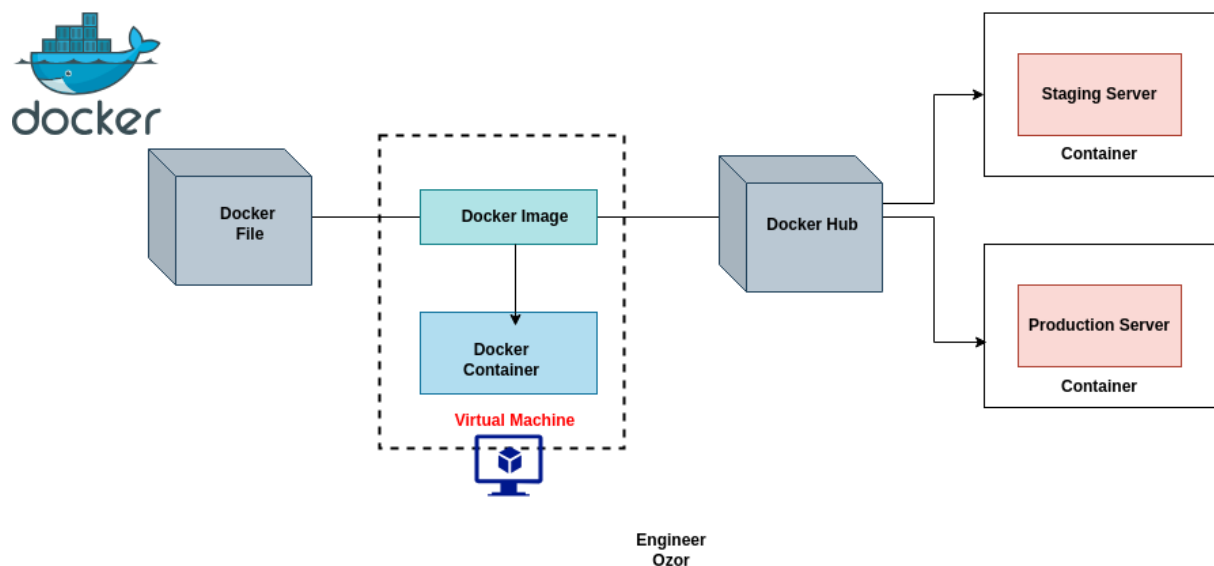
# Run the application
CMD ["node", "server.js"]
```



[linkedin.com](https://www.linkedin.com)

Understanding Docker Layers for Efficient Image Building

Docker Workflow



[medium.com](https://www.medium.com)

Docker Workflow. Docker is a containerization platform

3.2 Publishing Images to Docker Hub

Docker Hub is the default public registry for storing and sharing Docker images. After building an image, tag it with your Docker Hub username and push it.

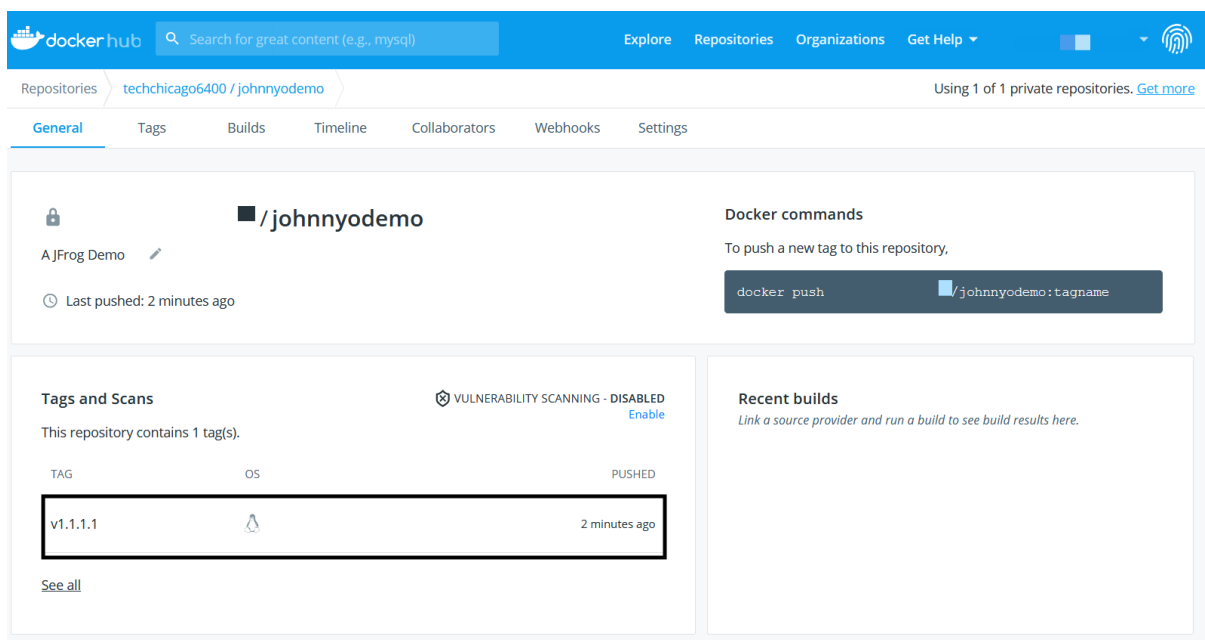
Commands:

Bash

```
docker tag my-web-app yourusername/my-web-app:latest
docker push yourusername/my-web-app:latest
```

This enables collaboration, version control, and easy deployment across environments.

Docker Hub Repository Example.



jfrog.com

Docker Hub and Docker Registries: A Beginner's Guide | JFrog

3.3 Container Volumes

Containers are ephemeral by default—data is lost when a container stops or is removed. Volumes provide persistent storage outside the container's lifecycle.

Types of volumes:

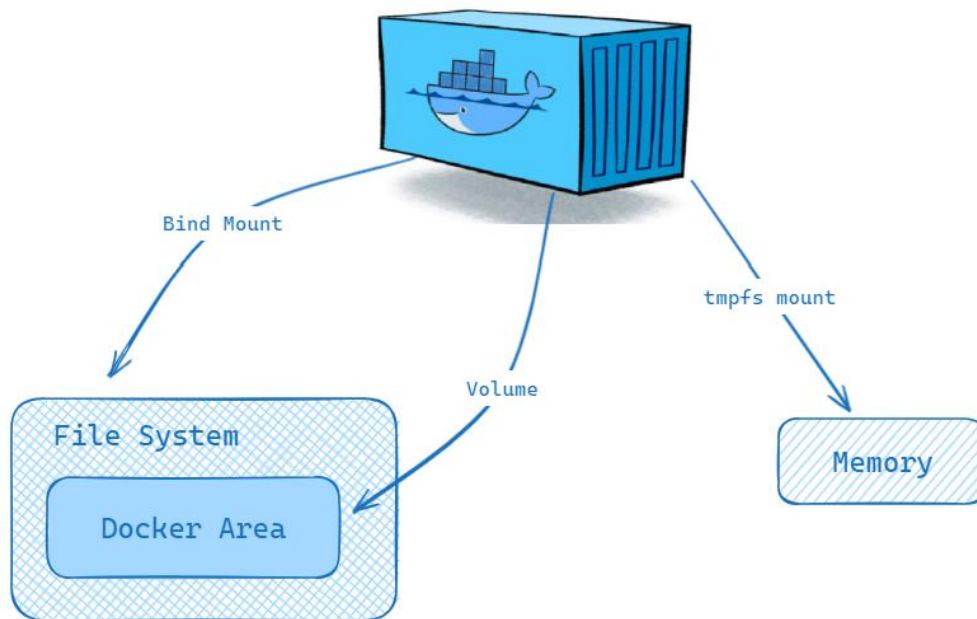
- **Named volumes** — Managed by Docker, stored in `/var/lib/docker/volumes`
- **Bind mounts** — Map host directories to container paths
- **tmpfs mounts** — In-memory storage

Example:

Bash

```
docker volume create mydata
docker run -d -v mydata:/app/data my-image
```

Docker Volumes Architecture.



therahulsarkar.medium.com

Understanding Docker Volumes: A Comprehensive Guide | by Rahul ...

3.4 Container Networking

Docker provides several network drivers:

- **bridge** (default) — Private internal network for containers
- **host** — Uses host's network stack
- **overlay** — Enables container communication across hosts (used in Swarm)
- **none** — No networking

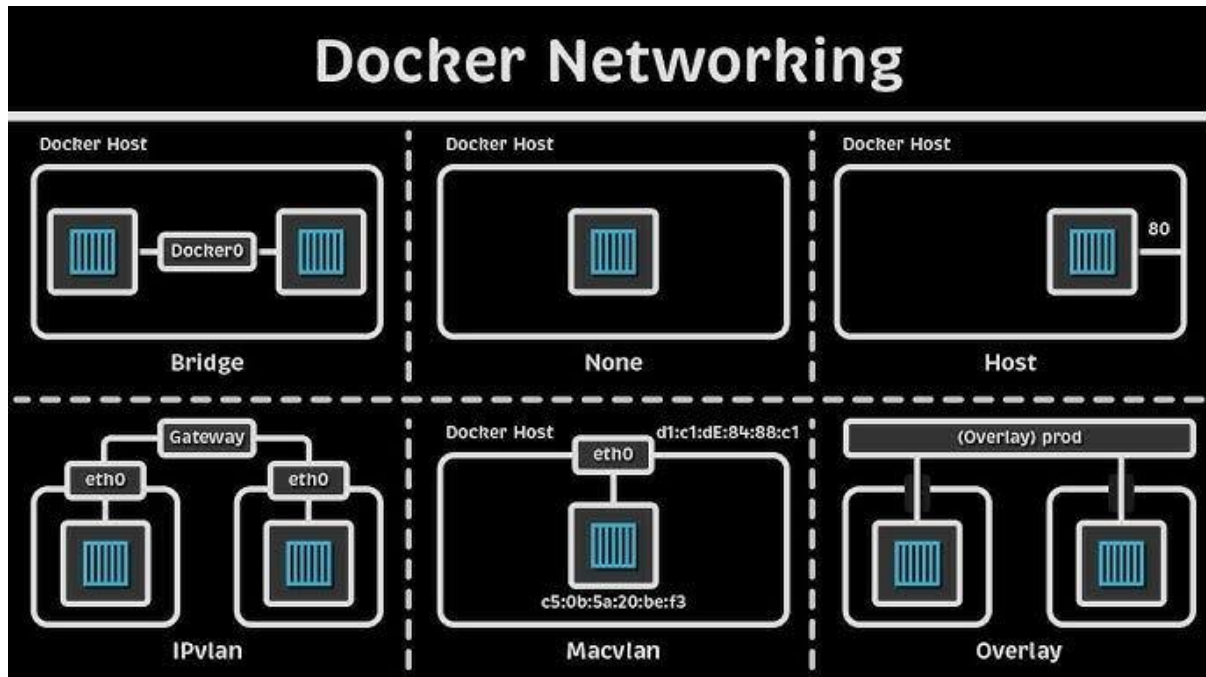
Example:

Bash

```
docker network create mynet
docker run --network mynet --name web -d my-web-app
docker run --network mynet --name db -d postgres
```

Containers on the same network can communicate using container names as hostnames.

Docker Networking Modes (Bridge, Host, Overlay).



zainnsar.medium.com

A Beginner's Guide to Docker Networking: Bridge, Host, and Overlay ...

3.5 Comprehensive Example: Web Server Deployment

This end-to-end example demonstrates building a custom Nginx web server, persisting content with a volume, connecting via a custom network, and publishing the image.

1. Create index.html:

HTML

```
<!DOCTYPE html>
```

```
<html><body><h1>Hello from Docker Container!</h1></body></html>
```

2. Dockerfile:

dockerfile

```
FROM nginx:alpine
```

```
COPY index.html /usr/share/nginx/html/index.html
```

```
EXPOSE 80
```

3. Build image: *docker build -t my-nginx-web .*

4. Create volume and network:

Bash

docker volume create html-data

docker network create web-net

5. Run container:

Bash

docker run -d -p 8080:80 --name web-server --network web-net -v html-data:/usr/share/nginx/html my-nginx-web

6. Verify: Access <http://localhost:8080> to see the page.

7. Publish to Docker Hub:

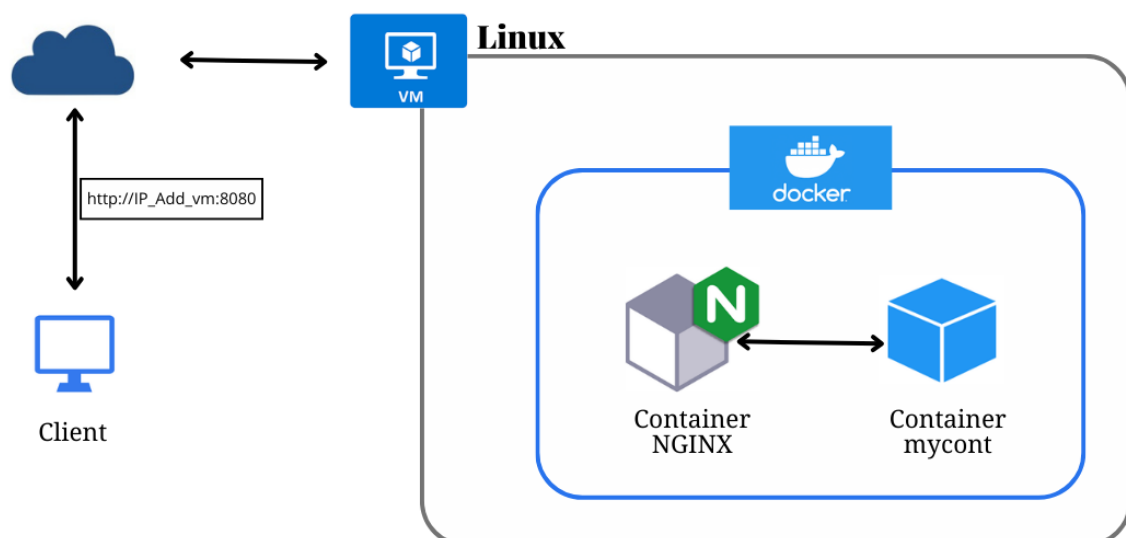
Bash

docker tag my-nginx-web yourusername/my-nginx-web:latest

docker push yourusername/my-nginx-web:latest

This workflow showcases Docker's full capabilities: building immutable images, persisting data, enabling communication, and sharing artifacts.

Nginx Docker Web Server Deployment Example.

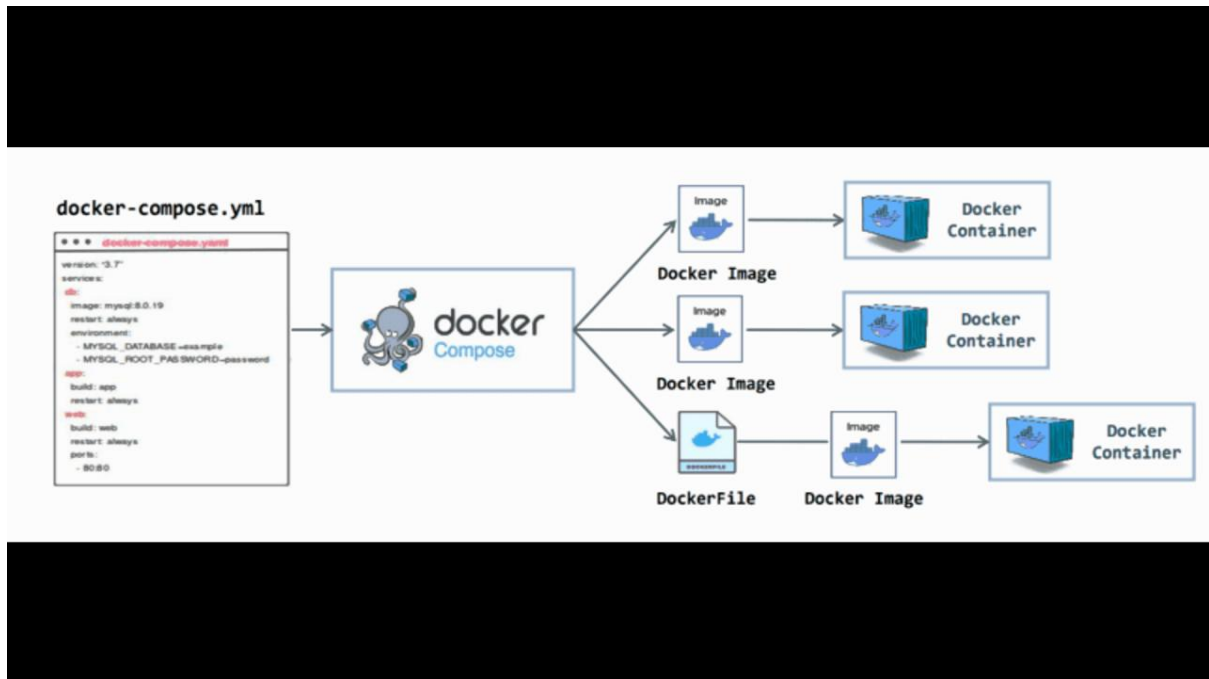


medium.com

Host A Web Page On NGINX Server Using Docker | by Gwladysgodem ...

Advanced Note: For multi-container applications, Docker Compose can define services, networks, and volumes in a docker-compose.yml file, simplifying orchestration.

Docker Compose Multi-Container Application Diagram.



[linkedin.com](https://www.linkedin.com)

Mastering Docker Compose: Simplifying Multi-Container Applications

Video Demonstration:

A 15-minute video demonstrating these steps, including Dockerfile creation, build, volume/network setup, run, and push, is available at:

This section illustrates how Docker enables efficient, reliable cloud deployments through containerization principles.

4. Core Cloud Computing Concepts

4.1 AWS Approach to Load Balancing

AWS uses Elastic Load Balancing (ELB), including Application Load Balancer (ALB) for HTTP/HTTPS

Load balancing is a critical component of cloud architecture that distributes incoming application traffic across multiple targets, such as Amazon EC2 instances, containers, IP addresses, Lambda functions, or on-premises servers. This distribution prevents any single resource from becoming overwhelmed, improves application availability, enhances fault tolerance, and ensures consistent performance even during traffic spikes.

AWS provides a fully managed **Elastic Load Balancing (ELB)** service with three primary types of load balancers, each designed for different use cases and layers of the OSI model:

1. Application Load Balancer (ALB)

- **Layer:** Layer 7 (Application layer – HTTP/HTTPS, WebSocket, gRPC)
- **Best for:** Web applications, microservices, containerized applications (ECS/EKS), and API-driven workloads
- **Key Features:**
 - Content-based routing (path-based, host-based, HTTP header-based)
 - Support for sticky sessions (session affinity)
 - WebSocket and HTTP/2 support
 - Integration with AWS WAF for web application firewall protection
 - TLS termination and offloading at the load balancer
 - Advanced request routing rules (redirects, fixed responses, etc.)
 - Health checks at the application level (HTTP/HTTPS status codes)
 - Cross-zone load balancing (distributes traffic evenly across Availability Zones)

In the e-commerce example: The ALB receives all customer requests and uses path-based routing:

- */api/products* → *Product catalog service (ECS containers)*
- */checkout* → *Payment processing service (Lambda functions)*
- */static* → *Static content served directly from S3 via CloudFront*

This routing ensures that different parts of the application can scale independently and use the most appropriate compute model.

2. Network Load Balancer (NLB)

- **Layer:** Layer 4 (Transport layer – TCP, UDP, TLS)
- **Best for:** High-performance, low-latency applications such as gaming, streaming, real-time data processing, and IoT
- **Key Features:**
 - Handles millions of requests per second with ultra-low latency
 - Static IP addresses per Availability Zone (useful for whitelisting)
 - Supports TLS termination and pass-through
 - Health checks at TCP/HTTP levels
 - Cross-zone load balancing

In the e-commerce example: NLB could be used for high-throughput background services like real-time inventory updates or WebSocket-based chat features.

3. Gateway Load Balancer (GWLB)

- **Layer:** Layer 3 (Network layer – IP)
- **Best for:** Deploying, scaling, and managing third-party virtual network appliances (firewalls, intrusion detection systems, deep packet inspection)
- **Key Features:**
 - Transparent network gateway and load balancer
 - Uses GENEVE protocol for encapsulation
 - Integrates with AWS Transit Gateway and VPC

In the e-commerce example: GWLB could route traffic through a third-party firewall appliance for additional security inspection.

Additional AWS Load Balancing Capabilities

- **Classic Load Balancer (CLB)** — Legacy (Layer 4/7), still available but not recommended for new applications
- **AWS Global Accelerator** — Enhances performance and availability by routing traffic through AWS's global network
- **CloudFront + ALB** — Combines CDN caching with dynamic content delivery
- **Elastic Load Balancing with Auto Scaling** — Automatically registers new instances with the load balancer

Target Types Supported by AWS Load Balancers

Target Type	ALB	NLB	GWLB	Description
EC2 Instances	✓	✓		Traditional virtual servers
ECS/Fargate Tasks	✓	✓		Container workloads
IP Addresses	✓	✓	✓	Any IP-reachable target
Lambda Functions	✓			Serverless workloads
Third-Party Appliances			✓	Firewalls, IDS/IPS

Load Balancing in Practice: E-commerce Scenario

During a Black Friday sale:

1. Traffic surges to 10x normal levels.
2. ALB distributes requests across an Auto Scaling group of EC2 instances.
3. Health checks remove unhealthy instances from the pool.
4. TLS termination at the ALB offloads encryption from backend servers.
5. WAF rules block malicious traffic before it reaches the application.

This combination ensures the e-commerce platform remains responsive, secure, and available even under extreme load.

Benefits of AWS Load Balancing:

- High availability (99.99% SLA)
- Automatic scaling of load balancer capacity
- Integration with other AWS services (Auto Scaling, CloudWatch, WAF, Route 53)
- Zero-downtime deployments (blue/green or canary via target groups)

In summary, AWS Elastic Load Balancing provides flexible, scalable, and highly available traffic distribution solutions tailored to different application requirements, making it an essential component for building production-grade cloud applications in 2025.

4.2 AWS Approach to Scalability

Scalability is the ability of a cloud application to handle growth in workload or demand by adding resources efficiently and cost-effectively. In cloud computing, scalability is essential for maintaining performance, availability, and user experience during traffic fluctuations—such as seasonal peaks, flash sales, or viral events.

AWS provides robust, automated scalability solutions that support both **vertical scaling** (scaling up/down by changing instance size) and **horizontal scaling** (scaling out/in by adding/removing instances). Horizontal scaling is the preferred approach for most modern cloud-native applications because it offers better resilience, higher availability, and easier automation.

Primary AWS Scalability Service: Amazon EC2 Auto Scaling

Amazon EC2 Auto Scaling automatically adjusts the number of Amazon EC2 instances in an **Auto Scaling group** based on real-time demand. It ensures that the application has the right number of instances running at all times—scaling out during high demand and scaling in during low demand.

Key Components of EC2 Auto Scaling:

- **Auto Scaling Group** — A logical collection of EC2 instances treated as a single unit. Defines minimum, maximum, and desired capacity.
- **Scaling Policies** — Define when and how to scale. Three main types:
 - **Target Tracking Scaling** — Maintains a target metric (e.g., CPU utilization at 50%).
 - **Step Scaling** — Scales in steps based on alarm thresholds.
 - **Simple Scaling** — Adds/removes a fixed number of instances based on alarms.
- **Scaling Activities** — Automatically triggered actions (launch or terminate instances).
- **Cooldown Periods** — Prevent rapid scaling oscillations.
- **Instance Refresh** — Safely replace instances with new configurations (e.g., during deployments).

Scaling Policies in Detail:

Policy Type	Description	Best For
Target Tracking	Maintains a target value for a metric (CPU, request count, etc.)	Most common; simple and effective
Step Scaling	Scales in predefined steps based on alarm breach severity	Fine-grained control
Simple Scaling	Adds/removes fixed number of instances after alarm breach	Basic scenarios
Predictive Scaling	Uses machine learning to forecast demand and scale proactively	Predictable workloads (e.g., e-commerce)

Integration Points:

- **Elastic Load Balancing (ALB/NLB)** — Automatically registers new instances with the load balancer.
- **CloudWatch** — Provides metrics and alarms for scaling decisions.
- **Elastic Load Balancing Health Checks** — Removes unhealthy instances from the group.
- **Launch Templates** — Standardize instance configuration, including AMI, instance type, and user data.
- **Mixed Instances Policy** — Use multiple instance types and purchase options (On-Demand, Spot, Reserved) for cost optimization.

Predictive Scaling with AWS Auto Scaling

Introduced in 2019 and enhanced in 2025, **Predictive Scaling** uses machine learning to analyze historical traffic patterns and forecast future demand. It proactively scales capacity before demand spikes occur.

Benefits:

- Reduces latency compared to reactive scaling.
- Optimizes costs by avoiding over-provisioning.
- Ideal for predictable workloads like e-commerce sales events, daily traffic patterns, or weekly reports.

In the e-commerce example: Predictive Scaling analyzes past Black Friday traffic patterns and pre-launches additional instances hours before the event starts, ensuring instant responsiveness.

Additional AWS Scalability Features

1. Amazon ECS / EKS Auto Scaling

- **Cluster Auto Scaling** — Scales the underlying EC2 instances.
- **Service Auto Scaling** — Scales the number of tasks/pods based on CloudWatch metrics (CPU, memory, custom metrics).

2. AWS Fargate

- Serverless compute for containers — Automatically scales container tasks without managing servers.

3. AWS Lambda

- Automatically scales from zero to thousands of concurrent executions.

4. Amazon DynamoDB

- On-demand capacity mode automatically scales throughput.

5. Amazon Aurora Serverless

- Database that scales compute capacity automatically based on application demand.

Scalability in Practice: E-commerce Scenario

During a major sales event:

1. Normal traffic: 10 instances running.
2. Predictive Scaling forecasts spike and launches 40 additional instances proactively.
3. Target Tracking maintains 50% CPU utilization by adding instances as needed.
4. ALB automatically registers new instances.
5. After the event, Auto Scaling scales down to 10 instances to minimize costs.

Benefits of AWS Auto Scaling:

- **High Availability** — Distributes instances across multiple Availability Zones.
- **Cost Efficiency** — Scale down during low demand; use Spot Instances for non-critical workloads.
- **Resilience** — Automatically replaces unhealthy instances.
- **Zero-Downtime Scaling** — New instances are added before old ones are removed.

- **99.99% Availability SLA** — When properly configured across multiple AZs.

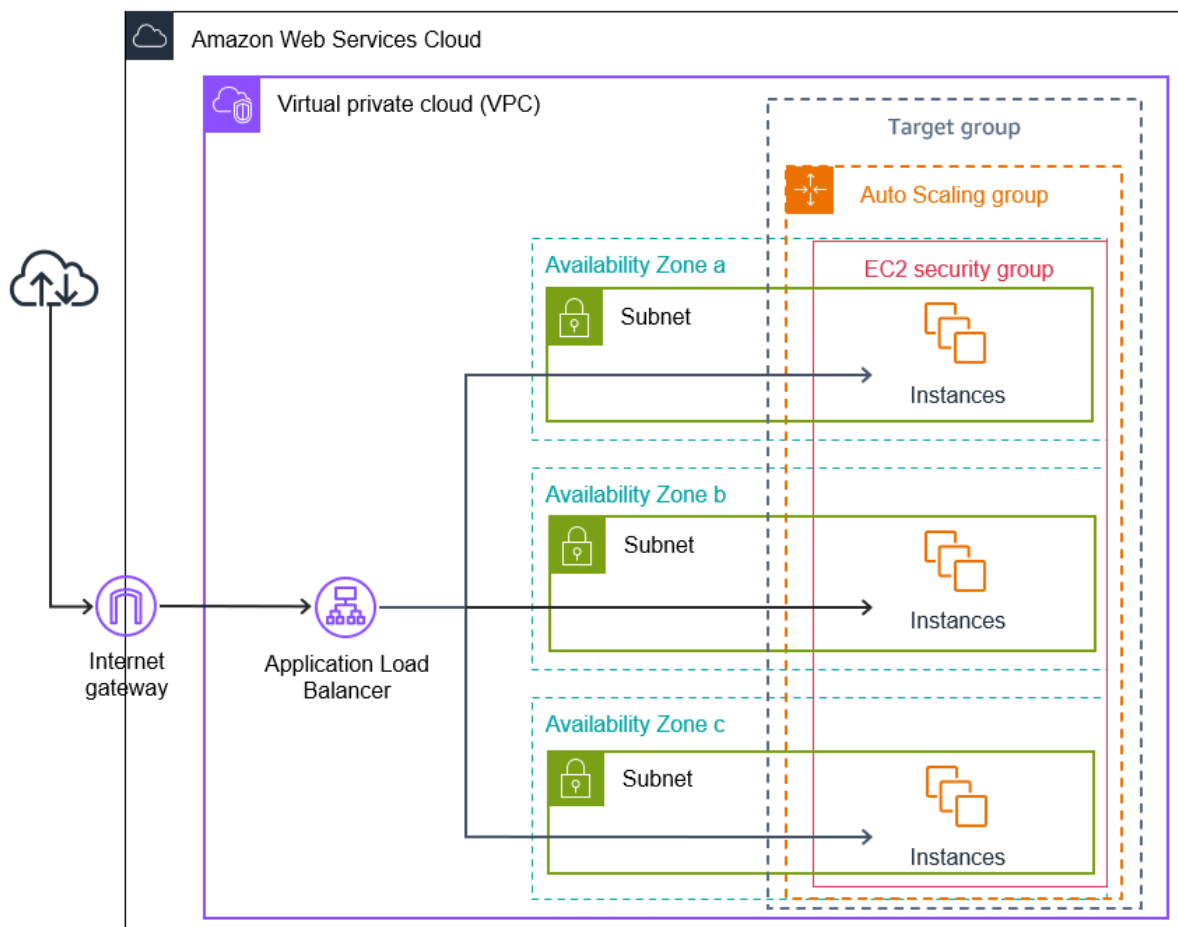
Best Practices for Scalability in 2025

- Use predictive scaling for predictable workloads.
- Combine target tracking with step scaling for granular control.
- Monitor with CloudWatch dashboards and set up alarms.
- Use instance refresh for safe deployments.
- Leverage mixed instances policies to reduce costs.
- Implement horizontal scaling over vertical where possible.

AWS Auto Scaling empowers organizations to build elastic, cost-effective, and highly available applications that automatically adapt to changing demand—making it a cornerstone of cloud-native architecture in 2025.

4.3 AWS Approach to Application Gateways

Amazon API Gateway serves as a managed front door for APIs, handling routing, authentication, and integration with backends like Lambda or EC2.



Tutorial: Set up a scaled and load-balanced application - Amazon ...

4.4 Why Load Balancing and Scaling are Complementary

Load balancing distributes traffic evenly, preventing overloads. Scaling adds/removes resources dynamically. Together, they ensure high availability and efficient resource use during traffic spikes.

4.5 Benefits of Combining All Three

An e-commerce app behind API Gateway uses load balancing for even distribution and auto-scaling for peak handling. This provides reliable, secure, scalable service with minimal downtime and optimized costs.

5. Driving Forces for Cloud Computing Expansion and Serverless Functions

5.1 Main Driving Forces: Cost and Flexibility

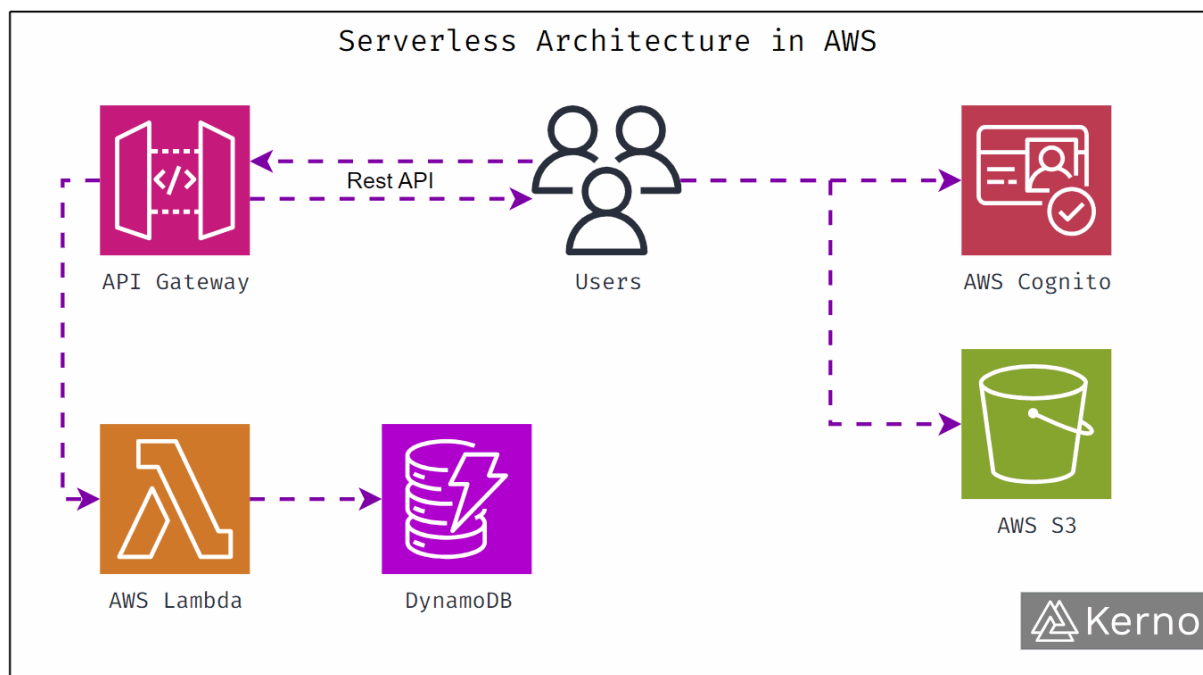
Key drivers include:

- **Cost savings** → Pay-as-you-go model reduces CapEx; no upfront hardware costs.
- **Flexibility/Scalability** → Rapid provisioning and global reach enable quick adaptation.

5.2 Serverless Functions (FaaS)

Serverless (e.g., AWS Lambda) runs code in response to events without managing servers. Billing occurs only for execution time.

Serverless Architecture with Lambda.



5.3 Comparison with Containerization

Aspect	Containerization (Docker)	Serverless (Lambda)
Management	Manage containers/orchestration (e.g., Kubernetes)	Fully managed; no infrastructure
Scaling	Manual/auto via orchestrator	Automatic per-request
Cost	Pay for running instances	Pay per execution (cheaper for sporadic)
Startup Time	Seconds	Milliseconds (cold starts possible)
State	Stateful possible	Stateless
Use Case	Long-running apps, full control	Event-driven, micro-tasks

Serverless maximizes cost/flexibility for bursty workloads, while containers suit persistent, complex apps.

6. Conclusion

In conclusion, this report has provided a comprehensive exploration of key cloud engineering principles and practices, with a particular focus on the transformative role of containerization using Docker within modern cloud environments. As demonstrated through practical examples and class exercises, Docker's lightweight, portable, and efficient approach to application packaging addresses critical challenges in virtualization, resource utilization, and deployment consistency. By leveraging Dockerfiles for image creation, volumes for persistent storage, custom networks for communication, and Docker Hub for image distribution, organizations can achieve seamless, reproducible deployments across hybrid and multi-cloud landscapes.

The integration of core cloud computing concepts—load balancing, scalability, and application gateways—further amplifies the value of containerization. AWS implementations, such as Elastic Load Balancing (ELB), EC2 Auto Scaling, and Amazon API Gateway, enable high availability, automatic resource optimization, and secure, performant application delivery. These technologies work synergistically to handle dynamic workloads, ensure fault tolerance, and deliver optimal user experiences, as illustrated in the e-commerce application case study.

The ongoing expansion of cloud computing in 2025 is propelled by compelling drivers including significant cost savings through pay-as-you-go models, unparalleled operational flexibility, and the explosive demand for AI and generative AI workloads. Serverless functions,

exemplified by AWS Lambda, offer a complementary paradigm to containerization by eliminating infrastructure management for event-driven and bursty workloads, while containers continue to excel for long-running, stateful, and highly customizable applications.

Ultimately, the future of cloud engineering lies in hybrid and multi-paradigm approaches that intelligently combine containerization, serverless computing, and traditional virtualized infrastructure. By mastering these technologies, organizations can build resilient, scalable, secure, and cost-effective systems capable of meeting the demands of an increasingly digital and AI-driven world. This report underscores the critical importance of Docker and cloud-native principles in enabling innovation, agility, and efficiency in contemporary computing landscapes.

References

1. Docker Documentation. (2025). *Get Started with Docker*. Available at: <https://docs.docker.com/get-started/overview/>
2. Docker Documentation. (2025). *Dockerfile Reference*. Available at: <https://docs.docker.com/engine/reference/builder/>
3. Docker Documentation. (2025). *Docker Volumes*. Available at: <https://docs.docker.com/storage/volumes/>
4. Docker Documentation. (2025). *Networking*. Available at: <https://docs.docker.com/network/>
5. Docker Documentation. (2025). *Docker Hub*. Available at: <https://docs.docker.com/docker-hub/>
6. Amazon Web Services. (2025). *Elastic Load Balancing User Guide*. Available at: <https://docs.aws.amazon.com/elasticloadbalancing/latest/userguide/what-is-load-balancing.html>
7. Amazon Web Services. (2025). *Amazon EC2 Auto Scaling User Guide*. Available at: <https://docs.aws.amazon.com/autoscaling/ec2/userguide/what-is-amazon-ec2-auto-scaling.html>
8. Amazon Web Services. (2025). *Amazon API Gateway Developer Guide*. Available at: <https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html>
9. Amazon Web Services. (2025). *AWS Lambda Developer Guide*. Available at: <https://docs.aws.amazon.com/lambda/latest/dg/welcome.html>
10. Amazon Web Services. (2025). *Amazon Elastic Container Service (ECS) User Guide*. Available at: <https://docs.aws.amazon.com/AmazonECS/latest/developerguide/what-is-ecs.html>
11. Kubernetes Documentation. (2025). *Kubernetes Concepts – Containers*. Available at: <https://kubernetes.io/docs/concepts/containers/>
12. Canalys. (2025). *Worldwide Cloud Infrastructure Market Q3 2025*. Available at: <https://www.canalys.com/newsroom/worldwide-cloud-infrastructure-market-q3-2025>

13. McKinsey & Company. (2025). *The Top Trends in Tech for 2025*. Available at: <https://www.mckinsey.com/capabilities/tech-and-ai/our-insights/the-top-trends-in-tech>
14. IBM. (2025). *What is Containerization?*. Available at: <https://www.ibm.com/topics/containerization>
15. Red Hat. (2025). *Containerization vs. Virtualization*. Available at: <https://www.redhat.com/en/topics/containers/containers-vs-vms>
16. CNCF. (2025). *Cloud Native Computing Foundation Annual Survey 2025*. Available at: <https://www.cncf.io/reports/>
17. AWS Blog. (2025). *Building Scalable Web Applications with ELB and Auto Scaling*. Available at: <https://aws.amazon.com/blogs/architecture/>
18. Microsoft Azure Documentation. (2025). *Azure Functions Overview*. Available at: <https://learn.microsoft.com/en-us/azure/azure-functions/functions-overview>