CLOUD COMPUTING

Docker & Kubernetes

Modern Container Orchestration

UNDERSTANDING CLOUD COMPUTING

Cloud computing delivers on-demand computing resources and services over the internet with pay-as-you-go pricing.

Key Benefits:

- Access anywhere, anytime
- No upfront hardware investment
- Automatic scaling and updates
- Enterprise-grade security

TYPES OF CLOUD SERVICES

Infrastructure as a Service

- Virtualized computing resources
- Complete infrastructure control

Examples:
Amazon EC2, Azure VMs, Google Compute

Platform as a Service

- Development and deployment platform
- Built-in development tools

Examples:
Heroku, Firebase, App Engine

Software as a Service

- Ready-to-use applications
- Accessible via web browser

Examples:
Office 365, Salesforce, Dropbox

DEPLOYMENT STRATEGIES

Public Cloud

Shared infrastructure managed by third-party providers. Most cost-effective option for standard workloads.

Private Cloud

Dedicated infrastructure for single organization. Maximum control and security for sensitive data.

Hybrid Cloud

Combines public and private clouds. Balance between cost, security, and flexibility for diverse needs.

Multi-Cloud

Uses multiple cloud providers. Avoid vendor lock-in and optimize for best features from each platform.

CONTAINERIZATION REVOLUTION

Containers package applications with all dependencies into lightweight, portable units that run consistently across environments.

Lightweight & Fast - Containers share the host OS kernel, making them faster to start and use less resources than traditional VMs.

Consistent Environments - 'Works on my machine' problem solved. Same container runs identically in development, testing, and production.

Microservices Architecture - Break applications into smaller, independent services that can be developed, deployed, and scaled separately.

Rapid Deployment - Deploy applications in seconds instead of minutes. Perfect for CI/CD pipelines and agile development workflows.

WHAT IS DOCKER?

Docker is the leading containerization platform that enables developers to package, distribute, and run applications in isolated containers.

Docker Image

Template containing app code, libraries, and dependencies. Like a snapshot or blueprint for containers.

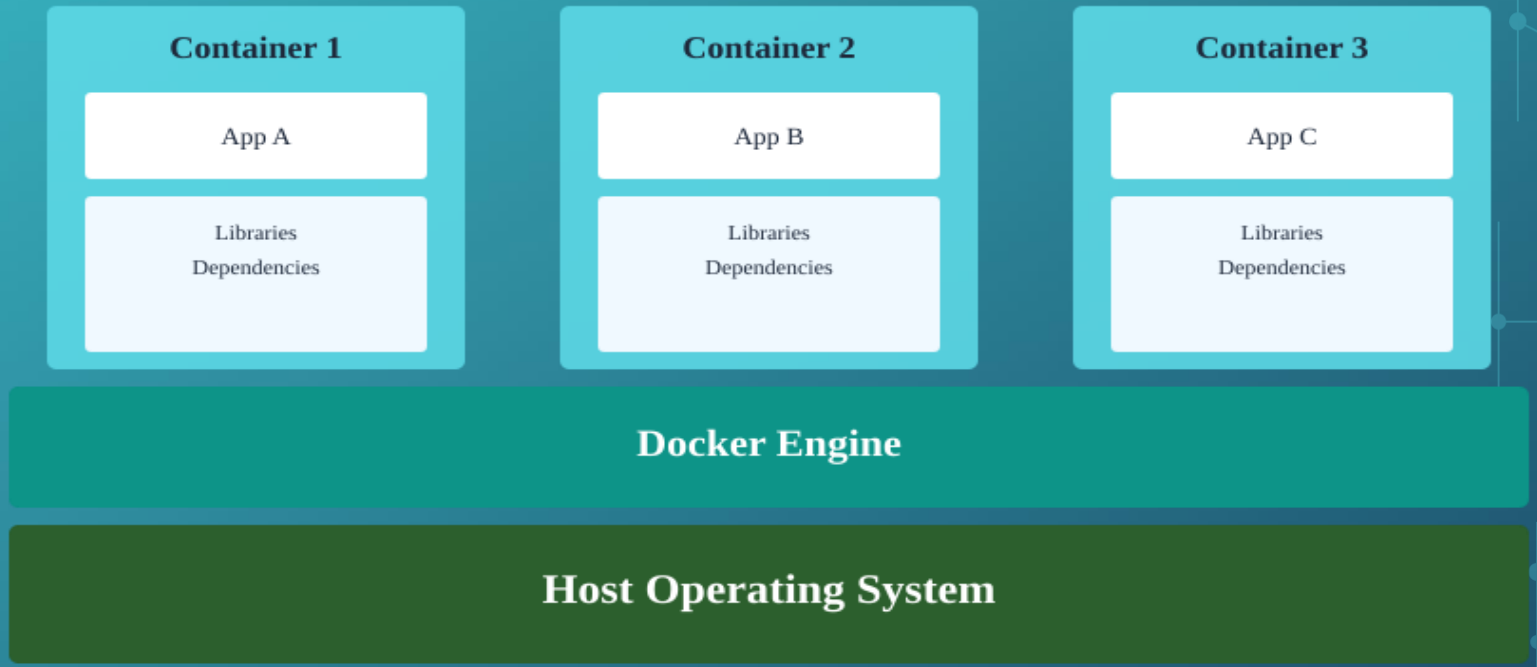
Docker Container

Running instance of an image. Isolated process with its own filesystem, networking, and resources.

Docker Engine

Runtime that creates and manages containers. The core technology that makes containerization possible.

DOCKER ARCHITECTURE



WHY USE DOCKER?

Portability Across Platforms

Docker containers run on any system - Windows, Mac, Linux, cloud platforms. Build once, run anywhere.

Resource Efficiency

Containers share the host OS kernel, using far less memory and CPU than virtual machines. Run 10x more containers than VMs on same hardware.

Version Control for Infrastructure

Docker images are versioned and stored in registries like Docker Hub. Roll back to previous versions easily.

Simplified CI/CD

Integrate seamlessly with Jenkins, GitLab CI, GitHub Actions. Automated testing and deployment in consistent environments.

WHAT IS KUBERNETES?

Kubernetes (K8s) is an open-source container orchestration platform that automates deployment, scaling, and management of containerized applications across clusters.

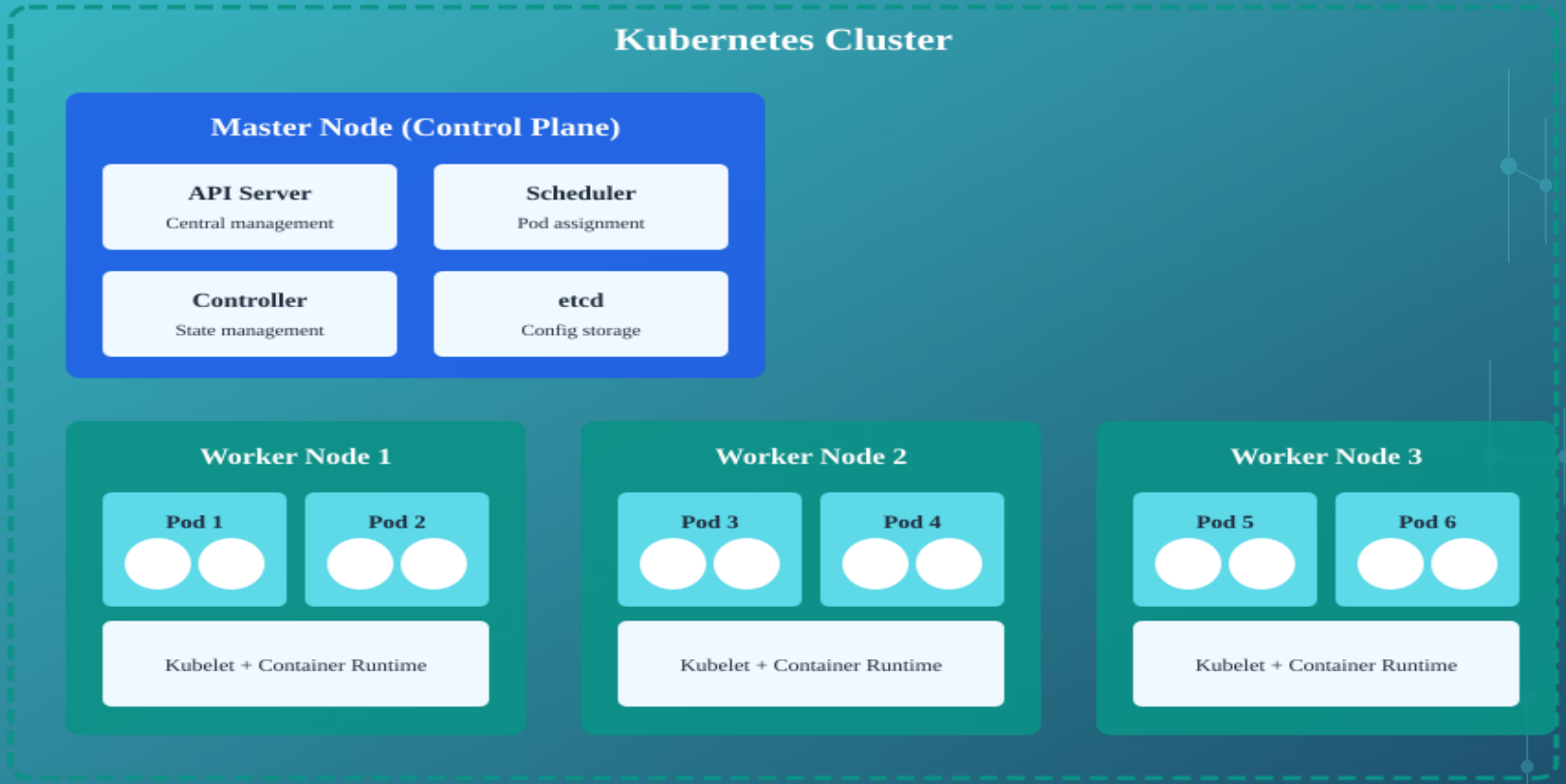
Originally developed by Google, now maintained by the Cloud Native Computing Foundation (CNCF).

Automated Orchestration - Automatically manages container placement, load balancing, and health monitoring across multiple servers.

Self-Healing - Automatically restarts failed containers, replaces containers, and kills containers that don't respond to health checks.

Horizontal Scaling - Scale applications up or down with a simple command or automatically based on CPU usage and custom metrics.

KUBERNETES ARCHITECTURE



KUBERNETES KEY COMPONENTS

Pods
Smallest deployable unit in K8s. One or more containers that share storage and network resources.

Services
Expose applications running on pods. Provide stable IP addresses and DNS names for pod groups.

Deployments
Declarative updates for pods. Define desired state and K8s manages the transition automatically.

ConfigMaps & Secrets
Store configuration data and sensitive information separately from application code.

Namespaces
Virtual clusters within a physical cluster. Organize and isolate resources for different teams or projects.

Volumes
Persistent storage for containers. Data survives container restarts and can be shared between pods.

Ingress
Manages external access to services. Provides load balancing, SSL termination, and name-based virtual hosting.

StatefulSets
Manages stateful applications. Provides stable network identities and persistent storage for databases.

DOCKER vs KUBERNETES

They're complementary, not competitors!

Docker

- Containerization platform
- Packages apps with dependencies
- Runs on single host
- Great for development & testing
- Simple to learn and use

Kubernetes

- Container orchestration platform
- Manages containerized apps
- Runs across multiple hosts/clusters
- Production-scale deployments
- Steeper learning curve

REAL-WORLD USE CASES

Microservices Architecture

Companies like Netflix, Uber, and Airbnb use K8s to manage thousands of microservices. Each service runs in containers, scaled independently based on demand.

CI/CD Pipelines

Automated testing and deployment workflows. Docker builds images, K8s deploys them. Companies achieve multiple deployments per day.

Multi-Cloud Deployments

Run the same application across AWS, Azure, and Google Cloud. K8s provides consistent management regardless of cloud provider.

Edge Computing & IoT

Deploy containerized applications to edge locations and IoT devices. Lightweight K8s distributions like K3s enable this use case.

BEST PRACTICES

Docker Best Practices:

- Use official base images
- Keep images small and focused
- Use .dockerignore files
- Don't run as root user
- Scan images for vulnerabilities
- Use multi-stage builds
- Tag images properly

Kubernetes Best Practices:

- Use namespaces for organization
- Set resource limits and requests
- Implement health checks (liveness/readiness)
- Use ConfigMaps for configuration
- Enable RBAC for security
- Monitor and log everything
- Use Helm for package management