UNIT 5

SCALABLE BIG DATA ANALYTICS PLATFORMS

Syllabus:

1. Cloud-Based Big Data Analytics Platforms

- **Amazon Web Services (AWS)**
- **Microsoft Azure**
- **Google Cloud Platform (GCP)**

2. Containerization and Orchestration Tools

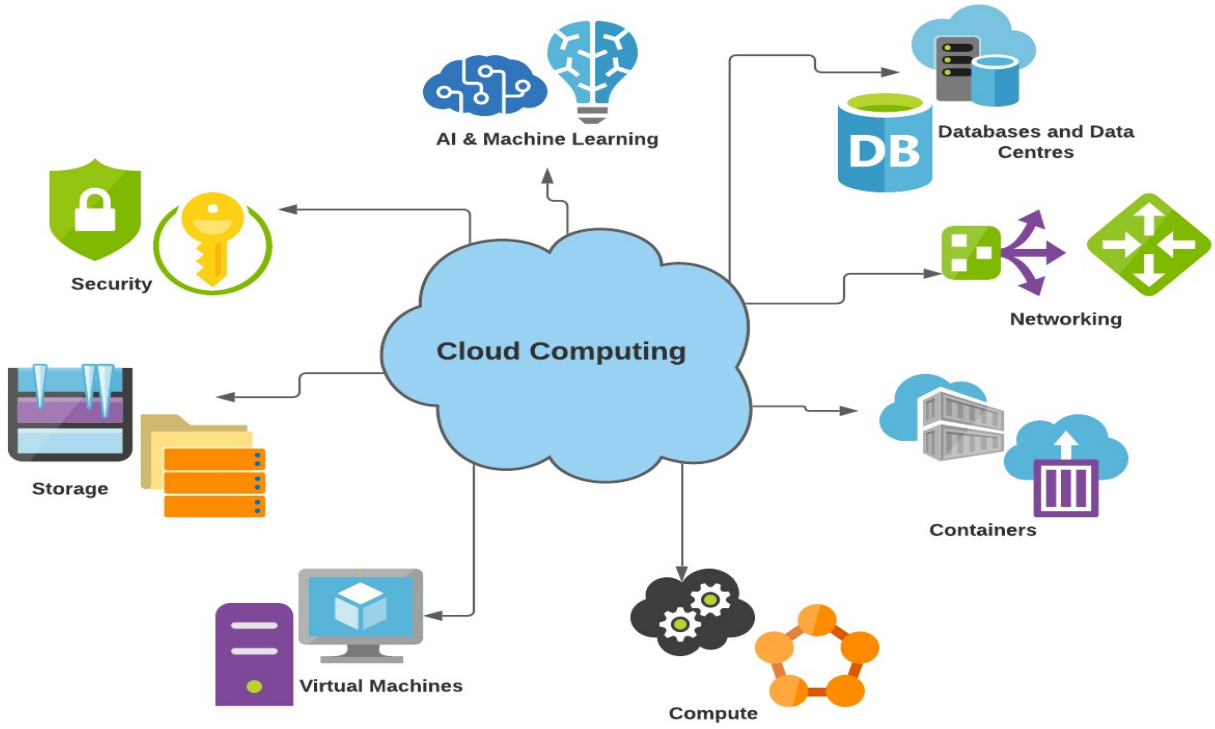
- **Containerization:** Concept and benefits
    - Docker
- **Orchestration:** Managing multiple containers
    - Kubernetes

3. Big Data Deployment and Scaling Strategies

4. Case Studies and Applications

- Real-world applications of big data analytics in various domains:

Cloud-Based Big Data Analytics Platforms



- *cloud computing is the delivery of computing services – servers, storage, databases, networking, software, analytics and more – over the Internet ("the cloud").*
- *Companies offering these computing services are called cloud providers and typically charge for cloud computing services based on usage, like how you're billed for gas or electricity at home.*
- *"Cloud Computing offers on-demand, scalable and elastic computing (and storage services).*
- *The resources used for these services can be metered and users are charged only for the resources used.*

Shared Resources and Resource Management:

- Cloud uses a shared pool of resources
- Uses Internet techn. to offer **scalable** and **elastic** services.
- The term **"elastic computing"** refers to the ability of **dynamically** and **on-demand** acquiring computing resources and supporting a variable workload.
- Resources are metered and users are charged accordingly.
- It is more cost-effective due to **resource-multiplexing.** Lower costs for the cloud service provider are past to the cloud users.

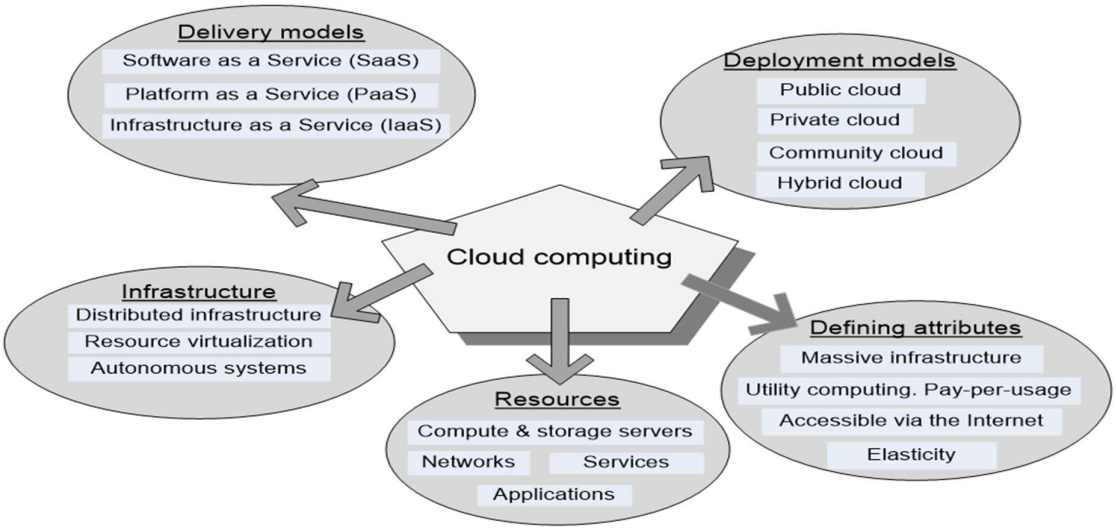
Data Storage:

- Data is stored:
  - in the "cloud", in certain cases closer to the site where it is used.
  - appears to the users as if stored in a location-independent manner.
- The data storage strategy can increase reliability, as well as security, and can lower communication costs.

Management:

- The maintenance and security are operated by service providers.
- The service providers can operate more efficiently due to specialization and centralization.

Cloud Computing Models, Resources, Attributes



Amazon Web Services (AWS)

A comprehensive cloud computing platform provided by Amazon, offering a broad range of services for

Computing
Storage
Networking
Machine learning
Analytics
databases, and more

Launched in 2006, AWS provides on-demand cloud infrastructure and resources, allowing individuals, businesses, and organizations to access and scale IT infrastructure without the need for physical hardware or data centres.

Key Characteristics of AWS

On-Demand Resources:	AWS offers resources such as servers, databases, storage, and applications on a pay-as-you-go basis, meaning users only pay for what they use.
Scalability and Elasticity:	AWS can scale resources up or down based on demand, making it suitable for varying workloads, from small-scale startups to large enterprises.
Global Reach:	AWS has data centers worldwide, allowing users to deploy applications closer to their customers, improving latency and performance.
Comprehensive Service Offerings:	AWS provides over 200 fully featured services, covering areas such as computing, storage, databases, machine learning, analytics, IoT, security, and more.
Security and Compliance:	AWS prioritizes security with features like data encryption, identity and access management (IAM), and compliance with major regulatory standards (e.g., GDPR, HIPAA).

Amazon EMR : Easily run and scale Apache Spark, Hive, Presto, and other big data workloads. Amazon EMR is a big data processing service that accelerates analytics workloads with unmatched flexibility and scale. EMR features performance-optimized runtimes for Apache Spark, Trino, Apache Flink, and Apache Hive, drastically cutting costs and processing times. The service integrates seamlessly with AWS, simplifying data lake workflows and enterprise-scale architectures. With built-in auto-scaling, intelligent monitoring, and managed infrastructure, EMR lets you focus on extracting insights—not managing clusters—delivering petabyte-scale analytics efficiently without the operational overhead of traditional solutions.

Microsoft Azure

Overview:

- **Microsoft Azure** is a **cloud computing platform** that provides a wide range of services for **data storage, analytics, AI, and big data processing**.
- It enables organizations to build, deploy, and manage applications on a global network of Microsoft-managed data centers.

Key Big Data Services:

- **Azure HDInsight:**
    - Managed cloud service for **Hadoop**, **Spark**, and **Hive** clusters.
    - Used for large-scale data processing and analytics.
- **Azure Synapse Analytics:**
    - Integrated analytics service for **data warehousing and big data**.
    - Supports querying structured and unstructured data.
- **Azure Databricks:**
    - Collaborative platform for **Apache Spark-based analytics**.
    - Enables real-time analytics and machine learning.
- **Azure Data Lake Storage:**
    - Scalable storage designed for big data workloads.
    - Supports both structured and unstructured data.
- **Azure Stream Analytics:**
    - Real-time data stream processing and event analysis.

Advantages:

- Seamless integration with Microsoft tools (Power BI, Office 365).
- High scalability and security.
- Supports hybrid and multi-cloud environments.

Google Cloud Platform (GCP)

Overview:

- **Google Cloud Platform (GCP)** is Google's suite of **cloud services** for **data analytics, machine learning, and big data storage**.
- It leverages Google's infrastructure used for services like **YouTube and Search**, ensuring high scalability and reliability.

Key Big Data Services:

- **BigQuery:**
  - Fully managed, serverless data warehouse for **fast SQL-based analytics** on large datasets.
- **Cloud Dataflow:**
  - Unified stream and batch data processing service (based on Apache Beam).
- **Cloud Dataproc:**
  - Managed **Hadoop/Spark** environment for processing big data efficiently.
- **Cloud Storage:**
  - Secure, scalable storage for raw and processed data.
- **AI and ML APIs:**
  - Pre-trained models and custom ML services for predictive analytics.

Advantages:

- High-speed query execution (BigQuery).
- Simplified data pipeline creation (Dataflow).
- Cost-effective, serverless scaling.
- Strong integration with **AI and ML tools** (TensorFlow, Vertex AI).

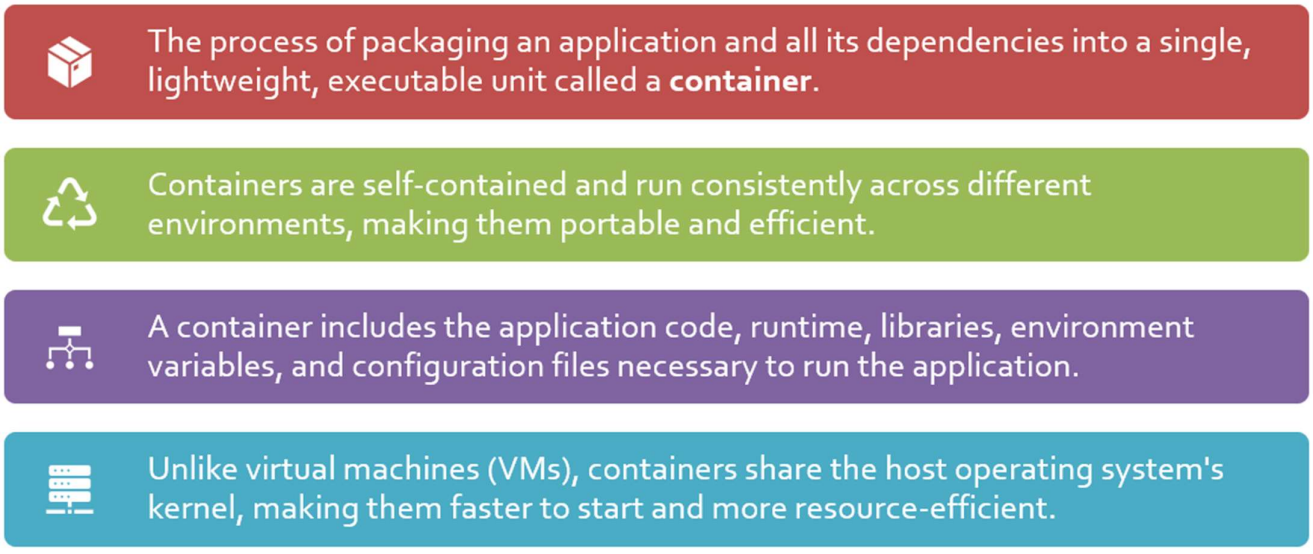
S no	Feature/Aspect	AWS	Azure	GCP
1	Provider	Amazon	Microsoft	Google
2	Launch Year	2006	2010	2008
3	Global Market Share (2023)	Leading (around 32%)	2nd largest (around 22%)	3rd largest (around 10%)
4	Compute Services	EC2 (Elastic Compute Cloud)	Virtual Machines (VMs)	Compute Engine
5	Storage	S3 (Simple Storage Service), EBS	Blob Storage, Disk Storage	Cloud Storage, Persistent Disks
6	Networking	VPC (Virtual Private Cloud)	Virtual Network	Virtual Private Cloud (VPC)
7	Database Services	RDS, DynamoDB, Aurora, Redshift	SQL Database, Cosmos DB, MySQL	Cloud SQL, Bigtable, Firestore
8	AI & Machine Learning	SageMaker, Lex, Polly, Deep Learning AMIs	Azure Machine Learning, Cognitive Services	AI Platform, TensorFlow, AutoML
9	Big Data	EMR (Elastic MapReduce), Redshift	HDInsight, Synapse Analytics	BigQuery, Dataproc
10	Serverless	Lambda	Azure Functions	Cloud Functions

S no	Feature/Aspect	AWS	Azure	GCP
11	DevOps Tools	CodeBuild, CodeDeploy, CodePipeline	Azure DevOps, GitHub Actions	Cloud Build, Cloud Source Repositories
12	Containers	ECS, EKS (Elastic Kubernetes Service)	Azure Kubernetes Service (AKS)	Google Kubernetes Engine (GKE)
13	Hybrid Cloud	Outposts, VMware Cloud on AWS	Azure Arc, Azure Stack	Anthos
14	Pricing Model	Pay-as-you-go, Reserved Instances, Spot Instances	Pay-as-you-go, Reserved Instances, Hybrid Benefit	Pay-as-you-go, Sustained use discounts
15	Compliance & Security	Extensive certifications, IAM, KMS	Extensive certifications, Azure Active Directory	Extensive certifications, IAM, KMS
16	Enterprise Adoption	Widely adopted across industries	Strong presence in enterprises (Microsoft ecosystem)	Strong in AI/ML and data-heavy industries
17	Developer Ecosystem	Large community, extensive documentation	Strong integration with Microsoft tools	Strong community with a focus on data science
18	Free Tier	12-month free tier, always free offers	12-month free tier, always free offers	12-month free tier, always free offers
19	Hybrid Cloud Focus	Limited compared to Azure	Strong focus with Azure Stack	Strong focus with Anthos
20	Customer Support	24/7 support with various plans	24/7 support with various plans	24/7 support with various plans

Containerization

Definition:

- **Containerization** is a lightweight virtualization technique that packages an application and all its dependencies (libraries, configurations, runtime) into a single **container**.
- This ensures the application runs **consistently across different environments** — from development to production.



Key Features:

- **Isolation:** Each container runs in its own isolated environment.
- **Portability:** Containers can run on any system that supports a container runtime.
- **Efficiency:** Containers share the same OS kernel, making them faster and lighter than virtual machines.
- **Scalability:** Easily deploy, replicate, and scale applications.

Benefits:

- Simplified deployment and version control.

- Faster startup times compared to VMs.
- Improved resource utilization.
- Easier DevOps integration and CI/CD pipeline management.

Benefits of Containerization

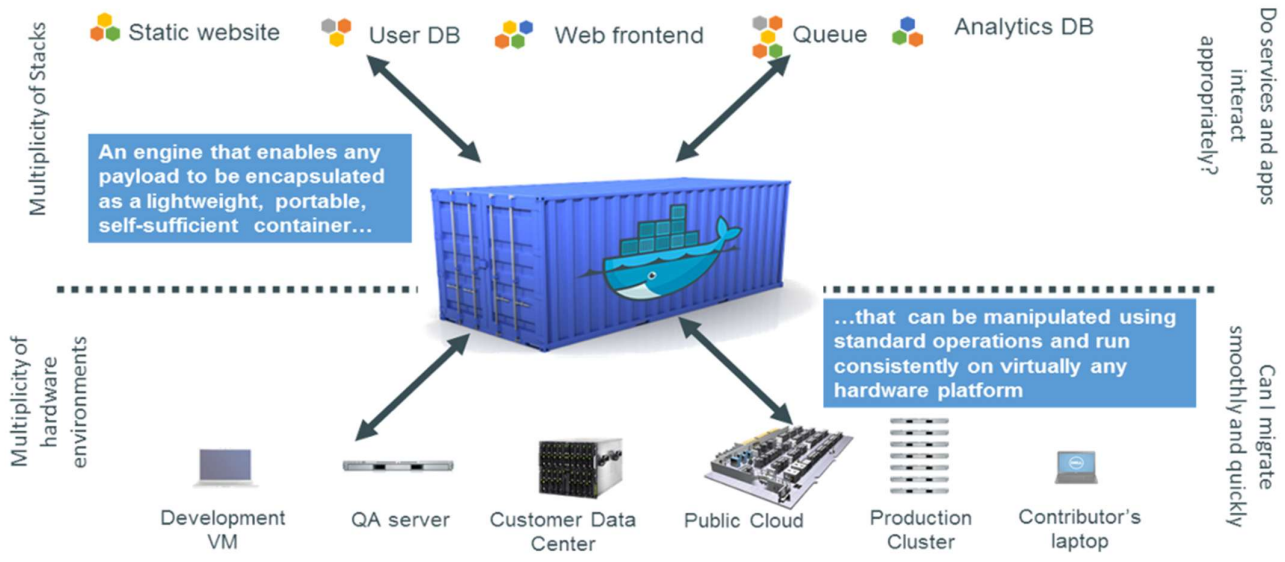
Portability:	• Containers run the same way across different environments (developer laptops, on-premises servers, or cloud).
Consistency:	• Containers ensure that applications run with the same configurations and dependencies, reducing "it works on my machine" issues.
Efficiency:	• Containers share resources and start up quickly, making them more lightweight and cost-effective than traditional VMs.
Isolation:	• Each container operates in its own isolated environment, allowing multiple applications to run on the same machine without conflicts.
Example	• Docker is a popular tool for containerization, allowing developers to build and deploy containers consistently and efficiently.

Docker

Definition:

- **Docker** is the most widely used **containerization platform** that automates the creation, deployment, and management of containers.
- It allows developers to **"build once, run anywhere."**

Docker is a Container System for Code



Docker Solves **"Matrix from hell"**

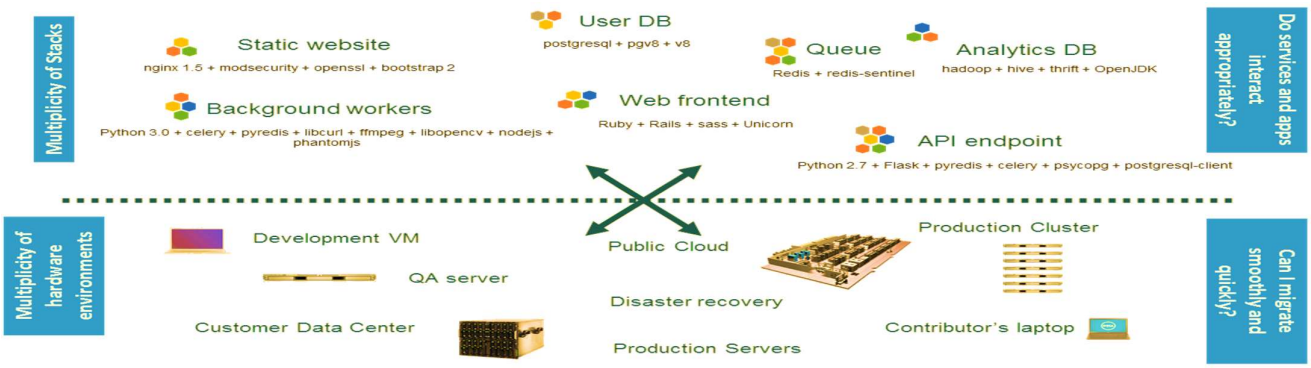
Core Components:

1. **Docker Image:**
    - A lightweight, read-only template that defines what's inside a container (OS, app code, dependencies).
2. **Docker Container:**
    - A running instance of a Docker image.
3. **Dockerfile:**
    - A script containing instructions to build a Docker image.
4. **Docker Engine:**
    - The runtime that builds and runs containers.
5. **Docker Hub:**
    - A cloud-based registry for storing and sharing container images.

Basic Workflow:

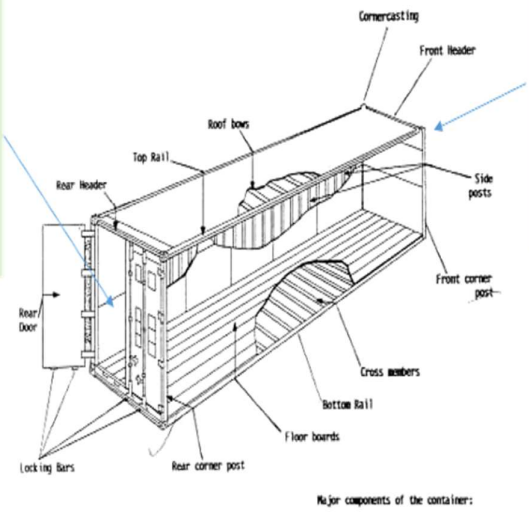
1. Developer writes a **Dockerfile** describing the environment.
2. Docker builds an **image** from it.
3. The image is used to create one or more **containers**.
4. Containers are deployed across machines using orchestration tools like **Kubernetes**.

The challenge



Why it Works: Separation of Concerns

- Dan the Developer
  - Worries about what's "inside" the container
    - His code
    - His Libraries
    - His Package Manager
    - His Apps
    - His Data
  - All Linux servers look the same



Major components of the container:

- Oscar the Ops Guy
  - Worries about what's "outside" the container
    - Logging
    - Remote access
    - Monitoring
    - Network config
  - All containers start, stop, copy, attach, migrate, etc. the same way

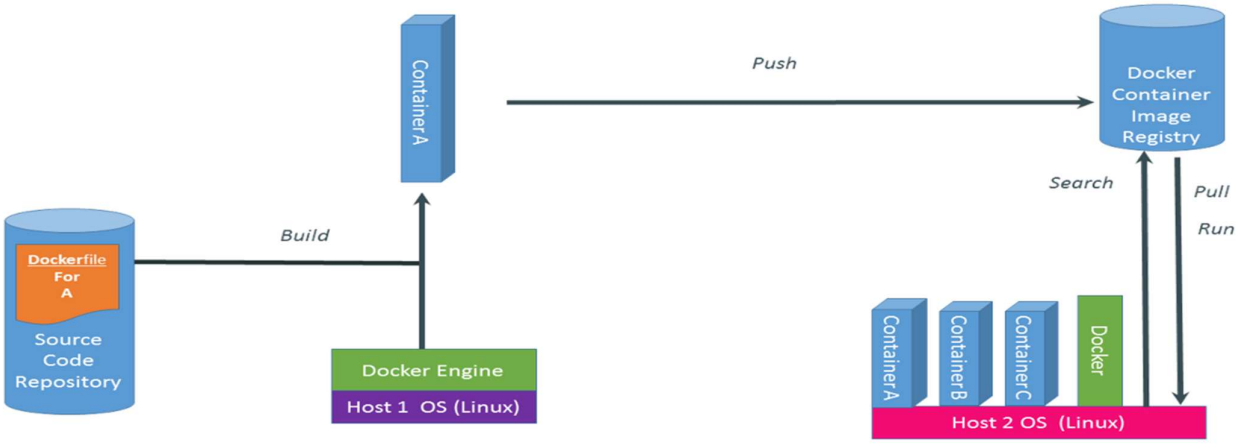
Why

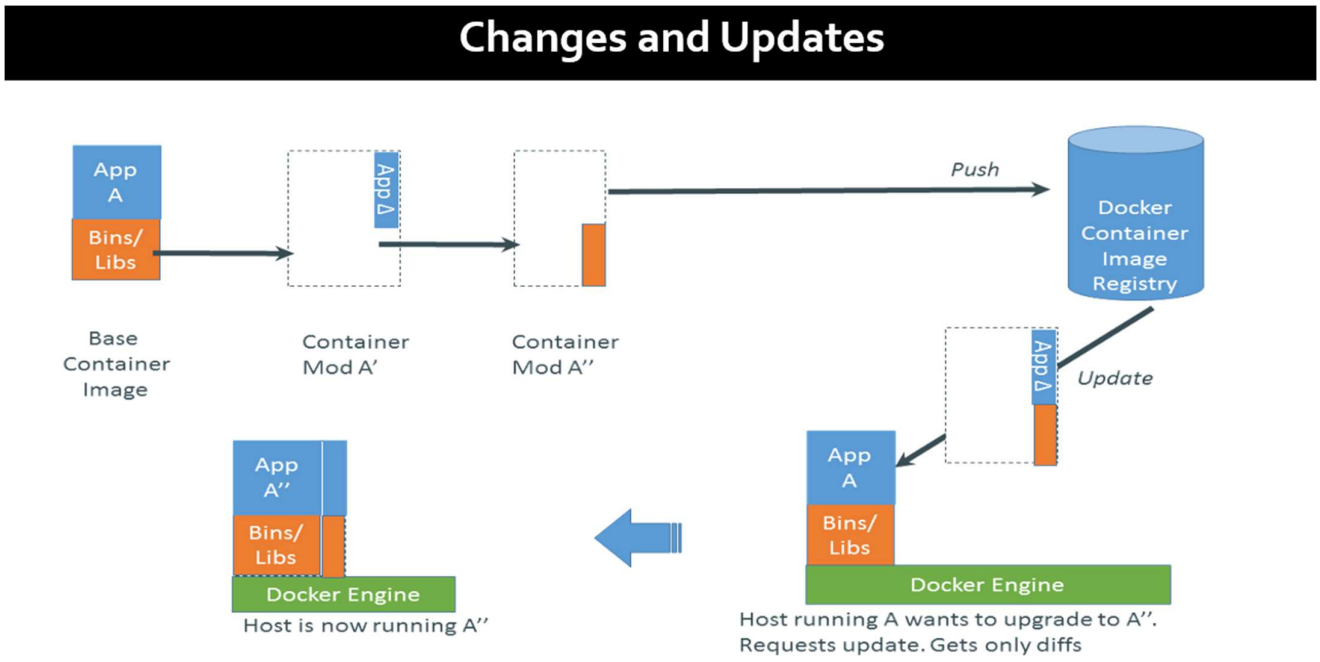
- Run everywhere
- Regardless of kernel version
- Regardless of host distro
- Physical or virtual, cloud or not
- Container and host architecture must match…
- Run anythingIf it can run on the host, it can run in the container
- If it can on a Linux kernel, it can run

What

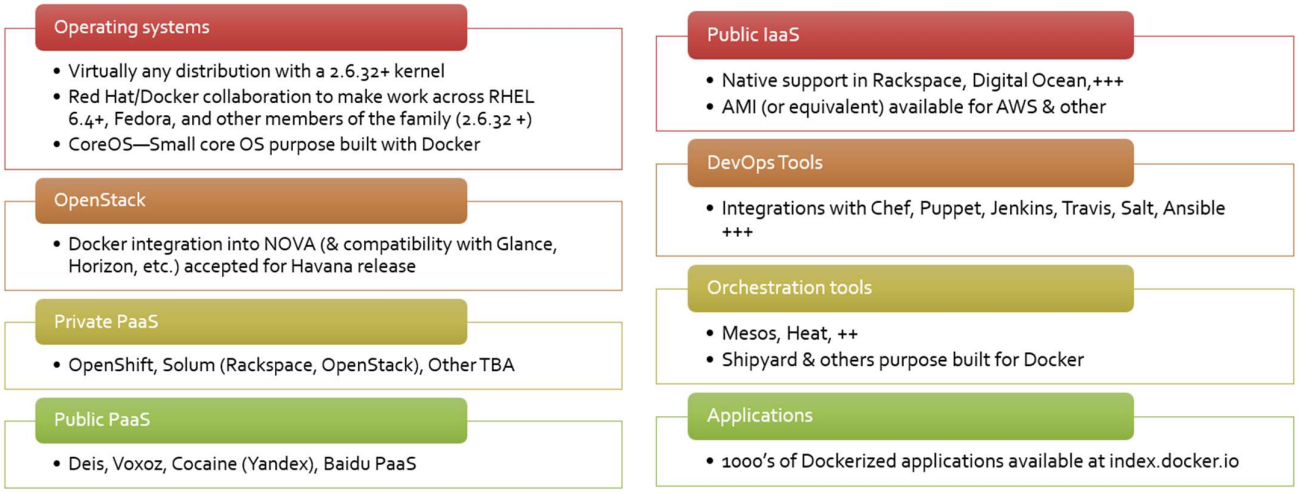
- High level: a lightweight VMOwn process space
- Own network interface
- Can run stuff as root
- Can have its own /sbin/init (different from host)
- <<machine container>>
- Low level: chroot on steroidsCan also *not* have its own /sbin/init
- Container = isolated processes
- Share kernel with host
- <="" li=""><<application container>>

What are the Basics of a Docker System?





Ecosystem support



Virtual Machines (VMs) vs Containers

1. Introduction

Both **Virtual Machines (VMs)** and **Containers** are technologies used to run multiple applications on a single physical machine.
However, they differ in **architecture, resource usage, and performance**.

2. Virtual Machines (VMs)

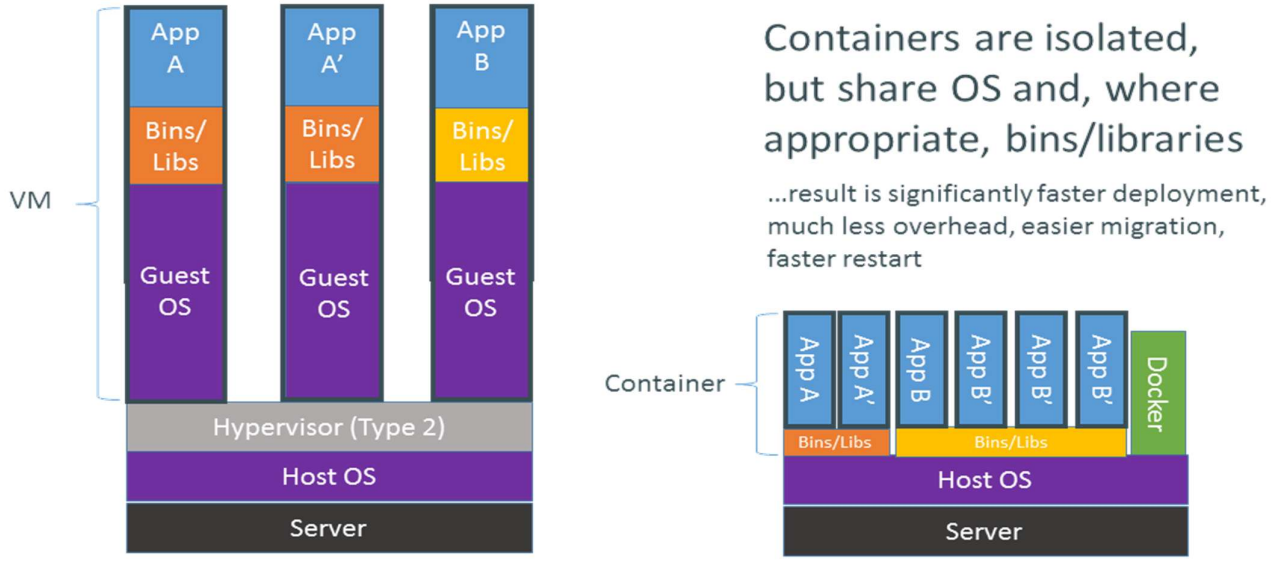
Definition:

- A **Virtual Machine** is an emulation of a physical computer that runs an **entire operating system (OS)** on top of a **hypervisor**.
- Each VM includes:
    - Its own **guest OS**

- **Application**
- **Libraries and dependencies**

Key Points:

- Heavyweight — consumes more memory and storage.
- Slower startup (due to full OS boot).
- Provides **strong isolation** between applications.
- Managed using hypervisors like **VMware**, **KVM**, or **VirtualBox**.



3. Containers

Definition:

- **Containers** virtualize at the **operating system level** instead of the hardware level.
- Multiple containers share the same **host OS kernel**, but run isolated applications with their own dependencies.

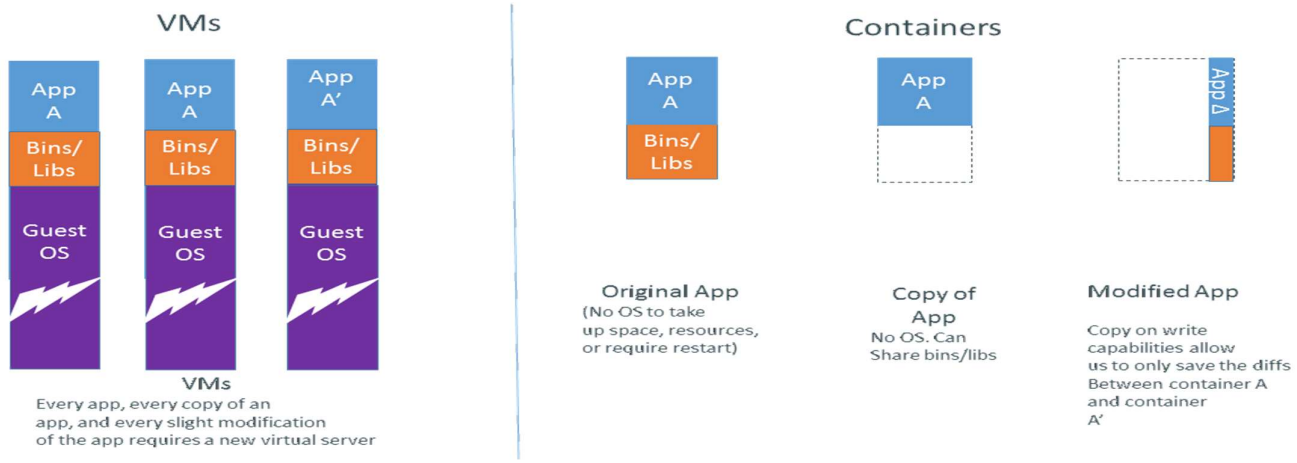
Key Points:

- Lightweight — no separate OS per container.
- Faster startup and lower resource usage.
- Easier to deploy and scale across environments.
- Managed using tools like **Docker** and **Kubernetes**.

4. Key Differences Between VMs and Containers

Aspect	**Virtual Machines (VMs)**	**Containers**
Virtualization Level	Hardware-level	OS-level
OS Requirement	Each VM runs its own full OS	Share the host OS kernel
Resource Usage	Heavy (large size and memory)	Lightweight (small footprint)

Aspect	Virtual Machines (VMs)	Containers
Startup Time	Slow (minutes)	Fast (seconds)
Isolation	Strong (separate OS)	Moderate (shared kernel)
Portability	Less portable due to OS dependency	Highly portable across platforms
Deployment Speed	Slower	Much faster
Examples	VMware, VirtualBox	Docker, Podman



Kubernetes

Introduction

- **Kubernetes (K8s)** is an **open-source container orchestration platform** developed by **Google** and now maintained by the **Cloud Native Computing Foundation (CNCF)**.
- It automates the **deployment, scaling, and management** of containerized applications across clusters of machines.
- It is widely used with **Docker** containers.

Need for Kubernetes

- When applications run in multiple containers across different servers, manually managing them becomes complex.
- Kubernetes solves this by:
    - **Automating container deployment**
    - **Balancing loads**
    - **Handling failures (self-healing)**
    - **Scaling applications automatically**

Key Features

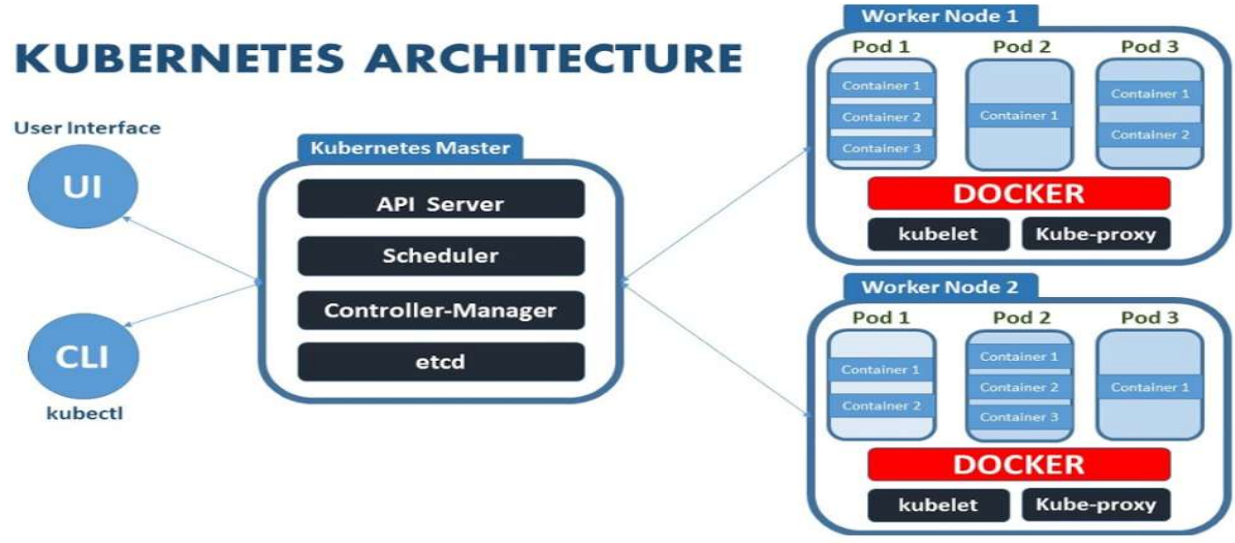
- **Automatic Scheduling:**
  Efficiently places containers based on available resources and constraints.

- **Load Balancing and Service Discovery:**
  Distributes network traffic evenly among containers.
- **Self-Healing:**
  Restarts failed containers, replaces them, and reschedules on healthy nodes.
- **Horizontal Scaling:**
  Automatically scales applications up or down based on demand.
- **Rolling Updates and Rollbacks:**
  Updates applications gradually without downtime.
- **Storage Orchestration:**
  Automatically mounts and manages storage systems (local or cloud-based).

Why Kubernetes Became Dominant

- **Vendor-Neutral and Open Source**:
  - Supported by major tech companies like Microsoft, IBM, Red Hat, and Google.
  - Not tied to any single vendor, fostering broad adoption.
- **Scalability and Performance**:
  - Handles large-scale deployments with thousands of containers.
- **Ecosystem Support**:
  - Seamlessly integrates with other tools and platforms (e.g., Helm, Prometheus).
  - Backed by cloud providers like AWS, Azure, and Google Cloud.
- **Flexibility**:
  - Supports hybrid and multi-cloud environments.
  - Can orchestrate various container runtimes beyond Docker, such as **containerd** and **CRI-O**.
- **Active Community**:
  - Fast-paced development with frequent updates and robust support.
- **Feature-Rich**:
  - Automates rolling updates, rollbacks, self-healing, and scaling.

Kubernetes Architecture



Kubernetes follows a **master–worker (control plane–node)** architecture.

a. Control Plane (Master Components):

Control Plane Components (Master Node)

The control plane is the "brain" of Kubernetes, responsible for orchestrating the cluster.

Component	Function
API Server (kube-apiserver)	Exposes the Kubernetes API as an entry point for commands and queries, enabling communication between users, control plane components, and the cluster.
etcd	A distributed key-value store that manages cluster state, configuration data, and metadata, ensuring consistency across the cluster.
Scheduler	Assigns newly created pods to nodes by evaluating resource availability, hardware/software requirements, and constraints.
Controller Manager	Manages control loops to maintain desired states for resources like pods, service endpoints, and replica sets by interacting with the API server.
Cloud Controller Manager	Interfaces with the cloud provider to manage resources like load balancers, nodes, and storage in cloud environments.

- **API Server:** Entry point for all administrative tasks; exposes Kubernetes API.
- **Scheduler:** Assigns workloads (pods) to available nodes.
- **Controller Manager:** Monitors cluster state and ensures desired configuration.
- **etcd:** A distributed key–value store for cluster configuration data.

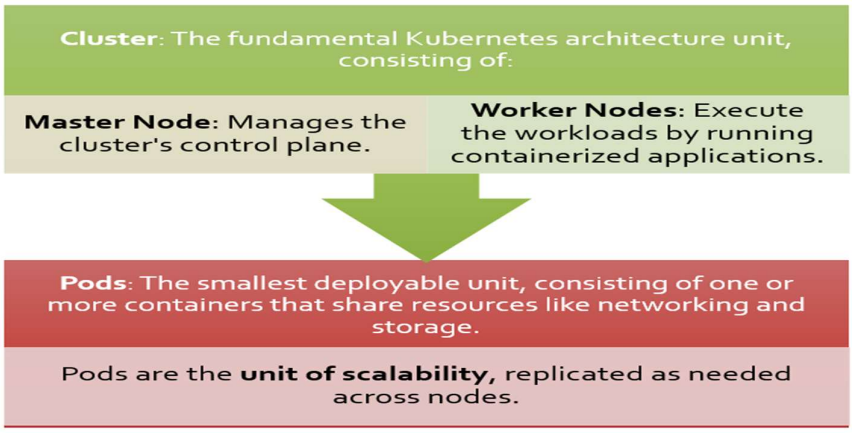
b. Worker Node Components:

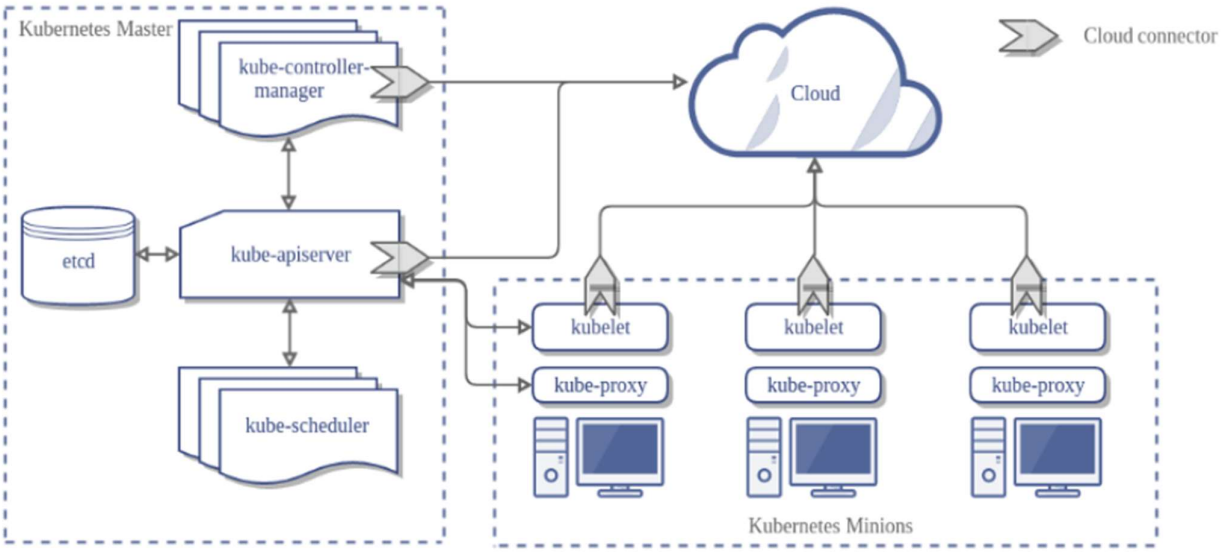
Node Components (Worker Nodes)

*Worker nodes are responsible for **running workloads** and managing pods.*

Component	Function
Kubelet	A node-level agent that ensures containers in pods are running as specified by the control plane.
Kube-proxy	Manages networking by maintaining network rules and load balancing between services and pods.

- **Kubelet:** Agent running on each node; ensures containers are running as specified.
- **Kube-Proxy:** Manages networking and load balancing for containers.
- **Container Runtime:** The engine (e.g., Docker, container) that runs containers.





Core Concepts

- **Pod:** The smallest deployable unit in Kubernetes; contains one or more containers.
- **Service:** Defines how to access a set of pods (load balancing, networking).
- **Deployment:** Manages replicas of pods, ensures desired number are running.
- **ReplicaSet:** Ensures a specified number of identical pods are maintained.
- **Namespace:** Logical separation of cluster resources.

Concept	Description
ReplicaSet	Ensures a specified number of pod replicas are running at any given time.
Deployment	Manages the lifecycle of applications, ensuring desired states and handling scaling and updates.
Kubectl	Command-line tool for managing clusters via the Kubernetes API.
DaemonSets	Ensures a specific pod is present on every node in the cluster.
Add-ons	Extend Kubernetes functionality, such as DNS for service discovery, web UI dashboards, and monitoring.
Service	Provides a stable network endpoint for accessing pods, enabling load balancing and service discovery.

Advantages of Kubernetes

- Portable across **cloud providers** (AWS, Azure, GCP).
- Improves **resource utilization** and **fault tolerance**.
- Simplifies **DevOps** and **microservices deployment**.
- Enables **scalable and resilient** big data analytics pipelines

Difference between Docker and Kubernetes

Aspect	Docker	Kubernetes
Purpose	A platform for building, packaging, and distributing containerized applications.	A container orchestration tool for managing, deploying, and scaling containerized applications.
Functionality	Provides tools to create, run, and manage individual containers.	Manages clusters of containers, ensuring high availability, load balancing, and scaling.
Focus	Focuses on containerization: packaging applications with all their dependencies.	Focuses on orchestration: scheduling, scaling, and managing containers in a distributed environment.
Core Component	Docker Engine (runtime) enables running containers.	Uses Docker (or other runtimes) to deploy and manage containers.
Deployment	Primarily for single-container applications or lightweight solutions.	Designed for managing multi-container applications and complex systems (e.g., microservices).
Scaling	Manual scaling of containers.	Automated scaling based on demand using predefined policies.
Networking	Limited networking for linking containers.	Advanced networking with service discovery, load balancing, and inter-container communication.
Self-Healing	Does not provide automated container recovery.	Automatically restarts, replaces, or reschedules containers in case of failures.
Updates	Supports building and running updated containers manually.	Provides rolling updates and rollbacks without downtime.
Portability	Ensures applications run consistently across environments (local, cloud, etc.).	Extends portability to orchestrated clusters across hybrid and multi-cloud environments.
Ease of Use	Simple and developer-friendly.	Complex, requires expertise to configure and manage effectively.
Use Case	Suitable for small-scale containerized applications and local development.	Ideal for large-scale deployments, distributed systems, and production environments.
Relationship	Docker containers are used to package applications.	Kubernetes uses Docker (or other runtimes) to orchestrate and manage containers.
Can Be Used Independently	Yes, Docker can run containers without Kubernetes.	Yes, Kubernetes can work with other container runtimes (e.g., containerd, CRI-O).

What is Container Orchestration?

Container orchestration refers to the automated management of containerized workloads and services.

It addresses the challenges that arise when organizations deploy and manage hundreds or even thousands of containers.

It includes tasks like:

Scheduling container deployment.	Managing networking between containers.	Ensuring scalability, fault tolerance, and high availability.	Monitoring and maintaining the health of containerized applications.	Kubernetes has emerged as the industry standard for container orchestration.

Kubernetes Deployment Process

- **Developer creates a deployment** using **kubectl** or another tool.
- Deployment specifies the desired number of pods and configurations.
- **API Server** processes the deployment and schedules pods via the **Scheduler**.
- **etcd** stores the cluster's state.
- **Controller Manager** ensures pods meet the desired state.
- **Kubelet** runs the pods on the nodes, monitored by **Kube-proxy** for networking.

Feature	Kubernetes	Docker Swarm	Apache Mesos
Scalability	Highly scalable for large clusters.	Suitable for smaller clusters.	Designed for massive scale but complex to manage.
Community Support	Strong global community and ecosystem.	Smaller community.	Limited adoption in modern container ecosystems.
Ease of Use	Complex setup but feature-rich.	Simpler to set up and use.	High learning curve.
Adoption	Widely adopted across industries.	Niche use cases.	Primarily used for legacy systems.

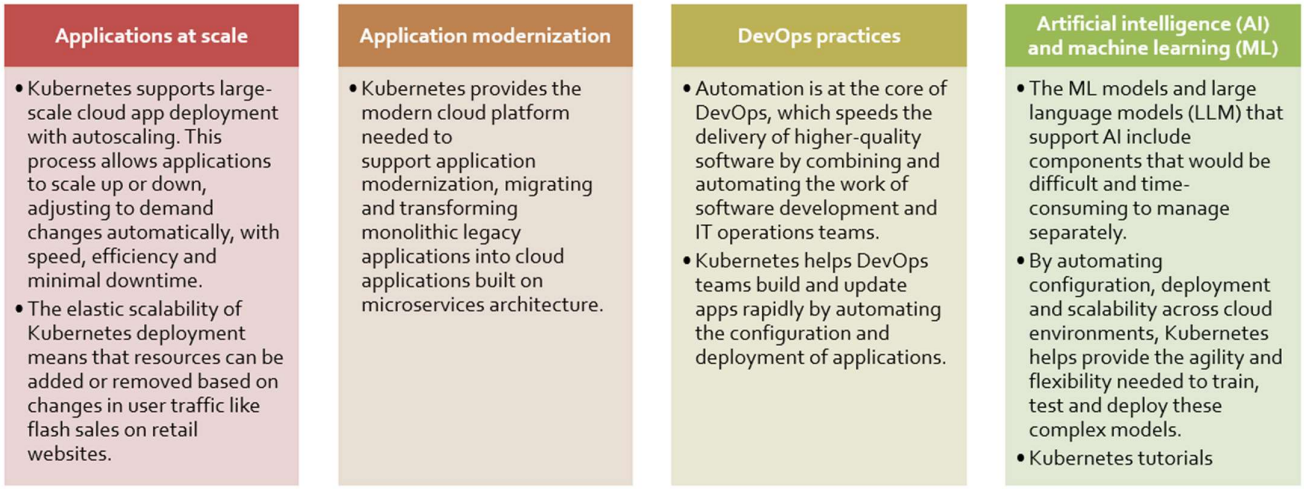
Kubernetes use cases

Microservices architecture or cloud-native development

- Cloud native is a software development approach for building, deploying and managing cloud-based applications.
- The major benefit of cloud-native is that it allows DevOps and other teams to code once and deploy on any cloud infrastructure from any cloud service provider.
- This modern development process relies on microservices, an approach where a single application is composed of many loosely coupled and independently deployable smaller components or services, which are deployed in containers managed by Kubernetes.
- Kubernetes helps ensure that each microservice has the resources it needs to run effectively while also minimizing the operational overhead associated with manually managing multiple containers.

Hybrid multicloud environments

- Hybrid cloud combines and unifies public cloud, private cloud and on-premises data center infrastructure to create a single, flexible, cost-optimal IT infrastructure.
- Today, hybrid cloud has merged with multicloud, public cloud services from more than one cloud vendor, to create a hybrid multicloud environment.
- A hybrid multicloud approach creates greater flexibility and reduces an organization's dependency on one vendor, preventing vendor lock-in. Since Kubernetes creates the foundation for cloud-native development, it's key to hybrid multicloud adoption.



Big Data Deployment and Scaling Strategies

Introduction

- **Big Data deployment** refers to the process of setting up, configuring, and managing big data systems (like Hadoop, Spark, or cloud clusters) to handle massive volumes of data.
- **Scaling strategies** ensure that as data volume and workload increase, the system continues to perform efficiently and reliably.

Types of Scaling

a. Vertical Scaling (Scale Up)

- Increasing the capacity of a single machine by adding more **CPU, RAM, or storage**.
- Example: Upgrading a node in a Hadoop cluster from 8 GB RAM to 32 GB RAM.

Advantages:

- Simple implementation
- No code or architecture changes

Disadvantages:

- Hardware limits scalability
- High cost and single point of failure

b. Horizontal Scaling (Scale Out)

- Adding **more machines (nodes)** to distribute the workload.
- Commonly used in **distributed big data systems** like Hadoop, Spark, and NoSQL databases.

Advantages:

- Highly scalable and cost-efficient
- Fault-tolerant and flexible

Disadvantages:

- Requires load balancing and distributed data management

Deployment Strategies

a. On-Premises Deployment

- Big data infrastructure is hosted within the organization's own servers.
- Provides full control over data, hardware, and security.

Best For: Organizations with strict compliance and data privacy needs.

Challenges: High setup and maintenance cost, limited scalability.

b. Cloud-Based Deployment

- Big data systems are hosted on **cloud platforms** like **AWS**, **Azure**, or **GCP**.
- Offers **on-demand resources**, **pay-as-you-go models**, and **auto-scaling**.

Advantages:

- Quick deployment and scalability
- Reduced infrastructure management
- Integration with analytics and AI services

c. Hybrid Deployment

- Combines **on-premises** and **cloud** environments.
- Sensitive data remains on-premises, while processing and analytics happen on the cloud.

Advantages:

- Balances control, cost, and scalability.
- Common in large enterprises.

Scaling Strategies in Big Data Systems

1. **Data Partitioning (Sharding):**
    - Divides large datasets across multiple nodes for parallel processing.
    - Example: Each Spark node processes a partition of data.
2. **Replication:**
    - Creates copies of data across nodes for **fault tolerance** and **high availability**.
3. **Load Balancing:**
    - Distributes workloads evenly among nodes to prevent bottlenecks.
4. **Auto-Scaling:**
    - Dynamically adds or removes compute resources based on real-time workload.
5. **Resource Optimization:**
    - Uses **cluster managers** (like YARN or Kubernetes) to allocate CPU, memory, and storage efficiently.