Microservices & Distributed Computing with Kubernetes

Microservices: The Real-World Scenario

Imagine you are building a machine learning system to recommend movies to users (like Netflix). This ML system has several components:

1. **Data Ingestion** – Collects and processes data from users (e.g., their watch history and ratings).
2. **Feature Engineering** – Transforms raw data into meaningful inputs for your ML model.
3. **Model Training** – Continuously trains and updates your recommendation algorithm.
4. **Model Serving** – Hosts the trained model and responds to user requests in real time.
5. **User Interface** – Provides the website or app where users can browse and see recommendations.

Monolithic Architecture vs. Microservices

In the **monolithic architecture**, all these components are bundled into a single application. Scaling the system means scaling the entire application, even if only the **Model Serving** component needs more resources.

With **microservices**, each component runs independently, making scaling, updates, and maintenance easier.

Microservices in Action

Instead of bundling everything together, microservices break down the application into smaller, independent components:

- **Data Ingestion Service** – Collects and processes user data.
- **Feature Engineering Service** – Transforms data into ML-ready features.
- **Training Service** – Retrains the model on new data at scheduled intervals.
- **Model Serving Service** – Hosts the model and serves API requests for recommendations.
- **UI Service** – Provides the user interface.

This approach enables independent scaling – for example, only the **Model Serving Service** can be scaled up during peak hours without affecting other services.

Real-Life Analogy: Microservices = A Food Court

- Each food stall specializes in one type of cuisine (e.g., burgers, pizza, coffee).
- They operate independently, so if one stall runs out of ingredients, others keep functioning.
- A **food court manager (Kubernetes)** ensures all stalls have electricity, water, and resources to operate smoothly.

Distributed Computing

Distributed computing refers to a system where multiple computers (**nodes**) work together to solve a large problem or process data in parallel. Tasks are divided among nodes for faster and more efficient computation.

Components of Distributed Computing

- **Cluster** – A group of interconnected computers (nodes) working together.
- **Lead Node (Master Node)** – Manages the cluster by assigning workloads and monitoring health.
- **Communication** – Nodes exchange data via network protocols for synchronization and task distribution.
- **Concurrency** – Multiple tasks run in parallel, increasing speed and ensuring fault tolerance.
- **MapReduce (Apache Spark)** – Splits tasks into "map" (processing) and "reduce" (aggregation) steps.

Benefits of Distributed Computing

□ **Scalability** – Distributes tasks among machines for handling larger workloads.
□ **Fault Tolerance** – If one node fails, the workload shifts to others.
□ **Improved Performance** – Parallel processing reduces latency.
□ **Cost Efficiency** – Uses multiple cheap machines instead of expensive hardware.
□ **Flexibility** – Mix different types of machines or cloud providers.

Challenges of Distributed Computing

□ **Resource Management** – Ensuring no machine is overloaded while others are idle.
□ **Scaling** – Adding/removing machines without disrupting the system.
□ **Communication & Networking** – Handling latencies, failures, and misconfigurations.
□ **Fault Handling** – Detecting and recovering from node failures.

□ **Load Balancing** – Evenly distributing tasks across machines.
□ **Deployment Complexity** – Configuring multiple machines manually.
□ **Monitoring & Debugging** – Tracking logs and performance across multiple machines.

How Kubernetes Solves These Challenges

Kubernetes is a **container orchestration platform** designed to simplify distributed computing.

Challenge	How Kubernetes Helps
Resource Management	Optimizes CPU, memory, and storage allocation.
Effortless Scaling	Auto-scales pods up/down based on demand.
Networking	Provides seamless pod communication.
Self-Healing	Restarts failed pods automatically.
Load Balancing	Evenly distributes traffic across pods.
Simplified Deployment	Uses declarative YAML configuration.
Monitoring & Debugging	Integrates with Prometheus, ELK Stack.

Kubernetes Internals

Control Plane (Master Node)

- **API Server** – Acts as Kubernetes' "receptionist," handling user requests.
- **etcd (Database)** – Stores cluster state and configurations.
- **Scheduler** – Decides which node runs a new pod.
- **Controller Manager** – Ensures the system maintains the desired state.

Worker Nodes

- **Kubelet** – Ensures containers (apps) are running properly.
- **Kube-Proxy** – Manages network traffic within the cluster.

- **Pods** – Smallest deployable unit, usually wrapping one or more containers.

Kubernetes Features

- **ReplicaSets** – Ensures a fixed number of pods are always running.
- **Services** – Provides stable network access to pods.
- **Namespaces** – Creates virtual clusters for better resource organization.
- **Persistent Volumes (PV)** – Provides shared storage for data.
- **YAML Manifests** – Define Kubernetes objects declaratively.

Real-Life Analogy: Distributed Computing Without vs. With Kubernetes

Without Kubernetes:

Imagine managing a **restaurant chain manually** – overseeing chefs, waiters, supply restocking, and handling peak-hour traffic. If one branch runs out of ingredients, you need manual intervention.

With Kubernetes:

Now, imagine a **central management system** that:

- Monitors every branch in real-time.
- Auto-restocks ingredients when supplies are low.
- Hires temporary staff when someone quits.
- Redirects customers to the least crowded branches.

This is exactly how **Kubernetes manages distributed computing!**

Summary

- **Microservices** – Break applications into independent services for better scalability.

- **Distributed Computing** – Uses multiple machines for parallel processing and fault tolerance.
- **Challenges** – Resource management, networking, fault handling, and monitoring.
- **Kubernetes** – Solves these challenges by automating deployment, scaling, and monitoring.
