Introduction to Kubernetes

- Kubernetes is a key player in DevOps, frequently appearing in job descriptions.
- It's considered the future of DevOps for long-term careers.

Prerequisites for Kubernetes

- Microservices architecture is prevalent in modern applications.
- Containers (like Docker) are essential for microservices.
- Understanding Docker and container concepts (from previous videos) is the only prerequisite for Kubernetes.
- Focus on basics: container vs. VM, networking/namespace isolation, why containers are lightweight, and container security.

Docker vs. Kubernetes: Understanding the Difference

- Docker: A container platform that provides a complete container lifecycle (engine, CLI) for easy interaction with containers.
- Kubernetes: A container orchestration platform.

Problems Kubernetes Solves (Docker's Limitations)

1. Single Host Limitation

- Problem: Docker containers typically run on a single host. If one container consumes excessive resources (e.g., memory), it can negatively impact or prevent other containers on the same host from running.
- Example: One container taking too much memory can "kill" another container on the same host.

2. Lack of Auto-Healing

- Problem: If a Docker container is killed (e.g., due to resource issues, manual termination), the application inside becomes inaccessible.
- Docker requires manual intervention (e.g., a DevOps engineer running `docker ps` to identify and restart containers).
- DevOps engineers cannot continuously monitor thousands of containers manually.
- Auto-healing: The ability for a container to start by itself without manual intervention. Docker inherently lacks this.

3. No Auto-Scaling

- Problem: Docker doesn't support automatic scaling based on load changes.

- Example: During peak seasons (e.g., festivals, popular movie releases), user load can dramatically increase (e.g., 10,000 to 100,000 users).
- Manually increasing container counts is inefficient.
- Docker doesn't automatically create new containers to handle increased load.
- Requires manual configuration of load balancing to distribute traffic across multiple manually created containers.

4. Minimalistic/Lack of Enterprise-Level Support

Problem: Docker is a simple platform that lacks built-in enterprise-level application support. Essential enterprise features not supported by default:

- Load balancing
- Firewall
- Auto-scaling (as discussed above)
- Auto-healing (as discussed above)
- API Gateways
- Whitelisting/Blacklisting (e.g., to prevent DDoS attacks)

Docker is not suitable for running applications in production independently.

How Kubernetes Solves These Problems

1. Single Host Problem -> Kubernetes Cluster Architecture

- Kubernetes is a cluster by default, meaning a group of nodes.
- In production, it's installed in a master-node architecture (can be standalone for development).
- Advantage: If one container (pod) on a node is faulty or impacting others, Kubernetes can automatically reschedule or move that pod to a different, healthy node.

2. Lack of Auto-Healing -> Replication Controller/Replica Sets

- Kubernetes uses concepts like Replication Controllers (older) or Replica Sets (newer).
- It ensures a desired number of identical pods are always running.
- When a container (pod) goes down, Kubernetes (via its API server) detects this and proactively rolls out a new container before the old one completely dies.
- This creates a seamless user experience, minimizing downtime.

3. No Auto-Scaling -> Horizontal Pod Autoscaler (HPA)

- Kubernetes allows for both manual and automatic scaling.
- Manual Scaling: Change the replica count in a YAML file (e.g., `replicationcontroller.yaml`, `replicaset.yaml`, `deployment.yaml`) to increase the number of pods (e.g., from 1 to 10).

- Automatic Scaling: Use Horizontal Pod Autoscaler (HPA). HPA can be configured to spin up new containers automatically when a defined threshold is met (e.g., CPU utilization reaches 80%).

4. Minimalistic/Lack of Enterprise-Level Support -> Enterprise-Grade Platform

    - Kubernetes originated from Google's internal tool "Borg."
    - It's built as an Enterprise-grade container orchestration platform.
    - It natively supports or provides mechanisms for essential enterprise features like load balancers and firewalls.
    - Kubernetes is highly extensible through Custom Resources (CRs) and Custom Resource Definitions (CRDs). This allows integration with external tools (e.g., NGINX for advanced load balancing via Ingress Controllers), extending its capabilities beyond basic functionality.