SOFTWARE ENGINEERING

UNIT - 4

TOPIC – 5

INTRODUCTION TO ORCHESTRATION USING KUBERNETES

What is Orchestration?

Definition: Orchestration refers to organizing and automating tasks to ensure they happen in the correct order and work smoothly together.

Purpose: Modern applications consist of many components like servers, databases, and containers. Manually managing these components can be time-consuming, error-prone, and complex. Orchestration simplifies this by automating these processes, saving time, and ensuring efficiency.

Example of Orchestration: Think of a wedding planner. The planner ensures that the caterer, decorator, and musicians do their tasks at the right time, so the wedding goes smoothly. Similarly, orchestration ensures all parts of a software system work together properly.

What is Container Orchestration?

Definition: Container orchestration manages containers to ensure they run properly, scale as needed, and communicate with each other efficiently.

Purpose: Containers are lightweight packages containing everything an application needs to run. While managing a single container is straightforward, handling hundreds or thousands is a daunting task. Container orchestration automates this, making it manageable.

What Does Container Orchestration Do?

- Starts containers when needed.
- Stops containers when they are no longer required.

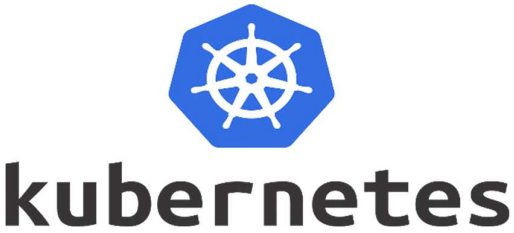
- Scales the number of containers up or down based on the workload.
- Ensures containers communicate with one another.
- Replaces failed containers.

Example of Container Orchestration: Imagine delivering lunchboxes to students. Each lunchbox contains everything needed for a meal. Managing and delivering a few lunchboxes is easy, but organizing and delivering lunchboxes to hundreds of students is challenging. Container orchestration is like a delivery system that automates this process for efficiency.

What is Kubernetes?

Definition: "Kubernetes is an open-source platform that helps automate the deployment, scaling, and management of applications inside containers."

The short form **K8s** is used for Kubernetes. The "8" refers to the eight letters between "K" and "s" in the word "Kubernetes."



Purpose: Without Kubernetes, developers must manually start, stop, and manage containers, which is time-consuming and prone to errors. Kubernetes automates these tasks, ensuring applications run reliably and scale efficiently.

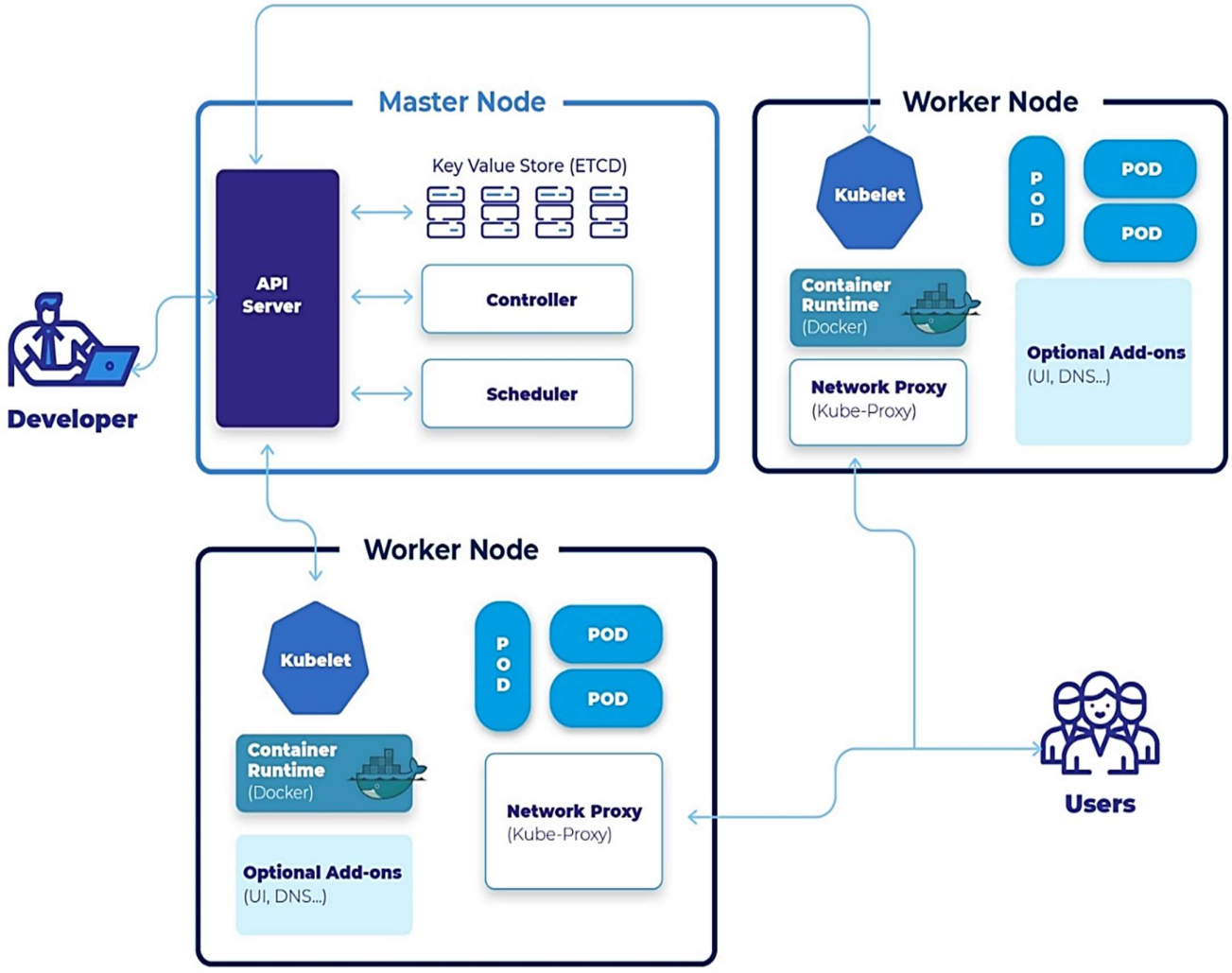
Features of Kubernetes:

- **Automatic Deployment:** Starts and stops containers as needed.
- **Scaling:** Automatically increases or decreases the number of containers based on demand.
- **Self-Healing:** Restarts containers that fail.
- **Load Balancing:** Distributes workloads evenly across containers.
- **Rolling Updates:** Updates applications without downtime.

Example of Kubernetes: Imagine running a pizza delivery service. Kubernetes acts like a manager who:

- Assigns tasks to delivery drivers (containers).
- Sends more drivers during busy hours (scaling).
- Replaces drivers who are unavailable (self-healing).

Kubernetes Architecture: Kubernetes operates like a well-organized company, where each component has a specific role:



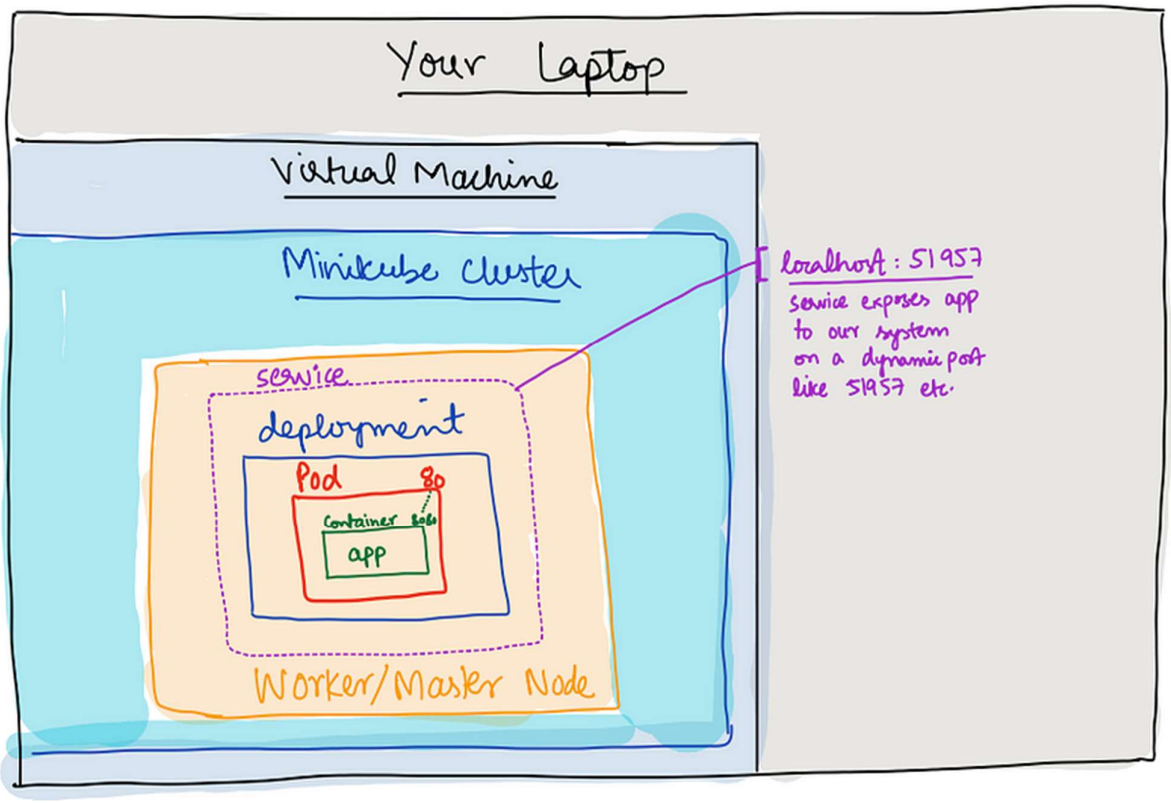
1. **Master Node:** The Master Node acts like the boss. It decides where and how containers should run, tracks the system's state, and fixes issues like restarting failed containers.
    - **API Server:** Acts as the receptionist, receiving and handling requests.

- **Scheduler:** Assigns tasks to worker nodes.
- **Controller Manager:** Ensures the system stays in the desired state, such as maintaining the correct number of containers.
- **etcd:** A database that stores information about the entire Kubernetes cluster.

2. **Worker Nodes:** Worker Nodes perform the actual work of running applications. Each worker node includes:
   - **Pods:** Groups of containers working together.
   - **Kubelet:** A helper that ensures containers run correctly.
   - **Kube Proxy:** Manages network connections between containers and external systems.
3. **Pods:** A Pod is the smallest unit in Kubernetes. It:
   - Holds one or more containers sharing resources like storage and networking.
   - Acts as a wrapper around containers to help them function together.
4. **Controller:** The Controller acts like a supervisor. It:
   - Ensures the correct number of pods are running.
   - Creates new pods if existing ones fail.
5. **Service:** A Service acts like a directory, helping users or other containers find the right pod, even if the pod moves to a different worker node.

Example of Kubernetes Architecture:

- The Master Node is like the central office managing tasks.
- Worker Nodes are like delivery vans carrying packages (pods).
- Pods are like the packages themselves.
- The Controller ensures enough vans are on the road.
- The Service connects customers to the right delivery van.



Explanation of the Diagram: This diagram represents how Kubernetes works on a local setup using Minikube, running on your laptop. Let's break it down step by step:

1. **Your laptop**: The physical machine (your laptop) is where everything is being set up.
2. **Virtual Machine**: Inside your laptop, Minikube creates a virtual machine. This is like a small computer inside your computer, and it runs the Kubernetes cluster.
3. **Minikube Cluster**: The Kubernetes cluster created by Minikube runs inside this virtual machine. The cluster includes the worker and master nodes needed for Kubernetes to function.
4. **Worker/Master Node**: Inside the Kubernetes cluster, there is at least one node (which can act as both a master and worker node in Minikube). This node handles the orchestration tasks and runs the actual containers.
5. **Pod**:

   A pod is the smallest unit in Kubernetes. It contains the actual application (or multiple applications) running in containers. In this diagram:

   - The pod includes a **container** that runs the actual application (`app`).
   - The container uses port **80** to communicate internally.

6. **Deployment**:

   The deployment is a Kubernetes object that ensures the desired number of pods are running at all times. It automatically replaces failed pods and manages updates.

7. **Service**:

   The service exposes the application (running in the pod) to the outside world. It assigns a dynamic port (like `51957`) on the local machine to make the application accessible. For example, if the application is a website, you can open a browser and go to `http://localhost:51957` to see it.

Differences Between Kubernetes, Docker, and Docker Swarm:

Kubernetes vs. Docker:

- Kubernetes and Docker work together but have different roles.
  - Docker is for creating and running containers.
  - Kubernetes organizes and manages these containers.
- **Key Differences:**

Feature	Docker	Kubernetes
Purpose	Builds and runs containers.	Manages multiple containers.
Scaling	Manual.	Automatic based on demand.
Fault Handling	Limited support.	Restarts/replaces failed containers.
Load Balancing	Basic traffic handling.	Advanced traffic management.

Example: Docker is like packing items into boxes, while Kubernetes is like a warehouse system organizing and tracking the boxes.

Kubernetes vs. Docker Swarm:

- Kubernetes is better for large, complex setups, while Docker Swarm is simpler and ideal for small projects.

- **Key Differences:**

Feature	Kubernetes	Docker Swarm
Complexity	Handles large systems.	Simpler for small setups.
Scaling	Ideal for heavy workloads.	Best for lightweight workloads.
Features	Advanced (rolling updates).	Basic features.

Example: Kubernetes is like managing a big orchestra, while Docker Swarm is like managing a small band.

What is Minikube?

Definition: Minikube is a tool that creates a small, local Kubernetes cluster on a personal computer.

Purpose: Minikube allows developers to practice Kubernetes concepts and test applications without needing a large, complex setup.

Example of Minikube: Minikube is like a small practice field where developers can try Kubernetes ideas before working on larger projects.

Deploying Applications Using Kubernetes and Minikube:

Step 1: Start Minikube Start Minikube with the command:

```
minikube start
```

This sets up a small Kubernetes cluster on the computer.

Step 2: Write a Small YAML File Create a file named `simple-app.yaml` with the following content:

```
apiVersion: v1
kind: Pod
metadata:
  name: my-app
spec:
  containers:
  - name: my-container
    image: nginx:latest
    ports:
    - containerPort: 80
```

What the YAML file does:

- **apiVersion:** Specifies the Kubernetes API version.
- **kind:** Defines the resource type (a Pod here).
- **metadata:** Names the pod (`my-app`).
- **spec:** Describes the container's details.
  - **name:** Names the container (`my-container`).
  - **image:** Uses the Nginx web server image.
  - **ports:** Exposes the container on port 80.

Step 3: Apply the YAML File Run the command to create the Pod:

```
kubectl apply -f simple-app.yaml
```

This instructs Kubernetes to create the pod as defined in the YAML file.

Step 4: Verify the Pod Check the running pod with:

```
kubectl get pods
```

Step 5: Clean Up Delete the pod after testing:

```
kubectl delete pod my-app
```