**Introduction to Containerization and Orchestration**

Modern applications are expected to run **consistently across diverse environments**—from developer laptops to cloud servers. Traditional **Virtual Machines (VMs)** are bulky, slow to start, and resource-intensive. In contrast, **containers** are lightweight, fast, and portable.

* **Containers** encapsulate an application and its dependencies, ensuring it runs the same regardless of the host.
* **Docker** is the most popular tool for creating and managing containers.
* **Kubernetes** is the leading platform for orchestrating and scaling containers across clusters of machines.

Together, Docker and Kubernetes form the backbone of **cloud-native development**.

**Docker Basics**

Docker is a **containerization platform** that simplifies the process of packaging, distributing, and running applications.

**Key Docker Components:**

* **Docker Engine**: The runtime that builds and runs containers.
* **Docker Images**: Read-only templates used to create containers.
* **Docker Containers**: Live, running instances of Docker images.

Each container is isolated but shares the host OS kernel, making it efficient and lightweight.

**What Is a Docker Image?**

A Docker image is a **standalone, executable package** that includes:

* Application code
* Runtime (e.g., Python, Node.js)
* Libraries and dependencies
* OS-level configurations

**Example:**

A Python image might include:

* Python interpreter
* pip (package manager)
* Linux base (e.g., Alpine or Ubuntu)
* Your Python script

Images are **immutable** and can be reused to spin up multiple containers with identical environments.

**Image Creation and Sharing**

Images are built using a **Dockerfile** or by modifying existing images.

**Workflow:**

1. **Create Dockerfile**
2. **Build image**: docker build -t myapp:v1 .
3. **Store locally** or **push to registry** (e.g., Docker Hub)
4. **Pull image** on any machine: docker pull myapp:v1

This enables **consistent deployment** across teams and environments.

**Dockerfile Explained**

A Dockerfile is a **script of instructions** to build a Docker image.

**Example:**

FROM openjdk:17

COPY app.jar /app.jar

CMD ["java", "-jar", "/app.jar"]

**Key Commands:**

* FROM: Specifies base image
* COPY / ADD: Adds files to image
* RUN: Executes commands (e.g., install packages)
* CMD: Defines default command when container starts

Dockerfile ensures **repeatable builds** and version control for environments.

**Docker Hub**

Docker Hub is the **central registry** for Docker images—like GitHub for code.

**Common Commands:**

* docker pull nginx: Downloads the official Nginx image
* docker push myimage:v1: Uploads your custom image

You can host **public** images for open-source sharing or **private** ones for internal use.

**What Is Kubernetes?**

Kubernetes (K8s) is an **open-source container orchestration platform** that automates:

* **Deployment** of containers
* **Scaling** up/down based on demand
* **Self-healing** (restarting failed containers)
* **Load balancing** across services
* **Service discovery** within clusters

It’s designed to manage **complex, distributed systems** at scale.

**Kubernetes and Application Orchestration**

**Orchestration** means automating the lifecycle of containers—deploying, scaling, updating, and monitoring.

**Key Kubernetes Concepts:**

* **Pod**: Smallest deployable unit; can contain one or more containers
* **Deployment**: Ensures the desired number of pods are running and updated
* **Service**: Provides stable networking to access pods
* **Ingress**: Manages external HTTP/S traffic to services
* **ConfigMap & Secrets**: Store configuration and sensitive data securely

Kubernetes uses **YAML manifests** to define desired states, which it continuously enforces.

**How Docker and Kubernetes Work Together**

1. **Build with Docker**: Developers create container images using Docker.
2. **Push to Registry**: Images are stored in Docker Hub or private registries.
3. **Deploy with Kubernetes**: YAML files define how and where containers run.
4. **Manage at Scale**: Kubernetes handles scaling, updates, and recovery.

They complement each other: Docker handles **packaging**, Kubernetes handles **orchestration**.

**Introduction**

Modern applications need to run consistently across multiple environments.

Traditional VMs are heavy, while containers are lightweight and portable.

Docker helps create and manage these containers.

Kubernetes helps manage containers at scale across clusters.

**2. Docker Basics**

Docker is a containerization platform.

A container includes everything needed to run an app: code, libraries, dependencies, and runtime.

Key Docker components:

Docker Engine → Core runtime.

Docker Images → Templates.

Docker Containers → Running instances of images.

**3. What is an Image**

An Image is a lightweight, stand-alone package.

Contains the OS libraries, environment, and app code.

Example: A Python image contains Python runtime + OS dependencies.

Used to create multiple consistent containers.

**4. Image Creation and Sharing**

Images are created using a Dockerfile or by modifying existing images.

Example: Create a custom Java app image.

Once created, images can be:

Stored locally.

Pushed to a registry (e.g., Docker Hub) for sharing.

Pulled from registry by others.

**5. Docker File**

A Dockerfile is a text file with step-by-step instructions to build an image.

Example:

FROM openjdk:17

COPY app.jar /app.jar

CMD ["java", "-jar", "/app.jar"]

Commands:

FROM → Base image.

COPY/ADD → Add files.

RUN → Run commands (install dependencies).

CMD → Default command when container starts.

**6. Docker Hub**

A central registry for Docker images (like GitHub for code).

Public and private repositories available.

Commands:

docker pull nginx → Download image.

docker push myimage:v1 → Upload custom image.

**7. What is Kubernetes**

Kubernetes (K8s) = Open-source container orchestration platform.

Helps manage thousands of containers automatically.

Handles:

Scaling (add/remove containers as needed).

Self-healing (restart failed containers).

Load balancing (distribute traffic).

Service discovery (find containers in cluster).

**8. Introduction to Kubernetes and Orchestrating Apps**

Orchestration = Automating deployment, scaling, and operation of containers.

Kubernetes Key Concepts:

Pod → Smallest deployable unit (one or more containers).

Deployment → Ensures required number of pods run.

Service → Exposes pods within or outside cluster.

Ingress → Manages external traffic.

ConfigMap & Secrets → Store config and sensitive data.