

Docker & Kubernetes Overview

What is Docker?

Docker is a free and open-source containerization software that allows you to package applications along with their dependencies into a single unit called a Docker Image. This image can run on any system that has Docker installed, making deployment easy and consistent across different environments.

Why use Docker?

- ✓ Portability – Run the same application on any machine, regardless of OS configuration.
- ✓ Dependency Management – Ensures that all required software (e.g., libraries, databases, and runtimes) is included within the image.
- ✓ Fast Deployment – No need to manually install dependencies every time you set up a new environment.
- ✓ Resource Efficiency – Uses fewer resources compared to traditional virtual machines.

How Docker Works?

Create a Docker Image – Package the app code + dependencies into a lightweight container image.

Run the Docker Container – Deploy this image as a container using the docker run command.

Execute Anywhere – The same image runs on any machine with Docker installed.

Once the image is built, it can run on any machine without requiring additional software setup.

Kubernetes (Container Orchestration Software)

What is Kubernetes?

Kubernetes (K8s) is a free and open-source orchestration tool developed by Google to manage containerized applications.

💡 Orchestration = Managing multiple containers efficiently

Kubernetes automates key tasks like:

1. Creating, starting, and stopping containers.
2. Scaling up/down based on demand.
3. Handling failures automatically.

#####

Why use Kubernetes?

#####

- ✓ Orchestration – Efficiently manages multiple containers across a cluster of machines.
- ✓ Self-Healing – If a container crashes, Kubernetes automatically replaces it.
- ✓ Load Balancing – Distributes traffic across multiple containers to avoid overloading.
- ✓ Auto Scaling – Increases or decreases the number of running containers based on traffic load.
- ✓ Automated Deployments – Supports CI/CD for rolling updates and version control.

Kubernetes Advantages (Detailed Explanation)

1) Orchestration – Managing Containers

Kubernetes helps manage multiple Docker containers across different machines (nodes) efficiently.

- ◆ Instead of running `docker run` manually for each container, Kubernetes automates deployment.
- ◆ It ensures that all containers are running smoothly and adjusts their status as needed.

Note: Kubernetes ensures all these containers are running, healthy, and communicating with each other properly.

2) Self-Healing – Automatic Recovery

If a container crashes due to an error or system failure, Kubernetes automatically restarts a new instance.

📌 Example:


A web server container (Apache, Nginx) stops unexpectedly.

Kubernetes detects the failure and starts a new container to replace it.

Users never notice downtime.

3) Load Balancing – Distributes Traffic Efficiently

Kubernetes distributes incoming user requests across multiple containers to avoid overloading any single instance.

 Example:


A shopping website experiences high traffic during a sale.

Kubernetes ensures that requests are evenly distributed across available backend servers.

Prevents server crashes and ensures smooth performance.

4) Auto Scaling – Adjusting Resources Dynamically

Kubernetes can increase or decrease the number of containers automatically based on traffic load.

 Example:


If website traffic increases, Kubernetes adds more containers to handle the load.

If traffic reduces, Kubernetes removes extra containers to save resources.

Works similarly to cloud-based Auto Scaling Groups (ASG).

Conclusion

 Docker simplifies packaging applications into portable containers.

 Kubernetes ensures these containers are orchestrated, scalable, and reliable.

Together, Docker & Kubernetes enable modern cloud-native application deployment—making applications highly available, efficient, and automated.

#####

Kubernetes (K8s) Architecture - Explained in Detail

#####

1) Control Plane (Master Node/Control Node)

-> The control plane is responsible for managing the Kubernetes cluster. It includes the following components:

1. API Server: Receives requests from kubectl and manages cluster operations.

2. Scheduler: Identifies pending tasks in ETCD and assigns them to worker nodes.

3. Controller Manager: Ensures the cluster's desired state matches the actual state.

4. ETCD: A distributed key-value store that acts as Kubernetes' internal database.

2) Worker Nodes (Slave Nodes)

-> Worker nodes run application workloads. They include the following components:

1. Kubelet: A node agent that communicates with the control plane and manages containers.

2. Kube Proxy: Manages networking and ensures communication within the cluster.

3. Docker Engine: Runs and manages containerized applications.

4. Pod: The smallest deployable unit in Kubernetes, housing one or more containers.

5. Container: Runs inside a Pod and contains the application code.

#####

Explanation k8S working

#####

Step 1: To deploy an application, we interact with the control plane using the kubectl CLI.

Step 2: The API Server receives the request and stores it in ETCD with a pending status.

Step 3: The Scheduler identifies an available worker node to execute the task, using Kubelet for node management.

Step 4: The Kubelet ensures the worker node is running the assigned workload.

Step 5: The Kube Proxy manages networking for seamless cluster communication.

Step 6: The Controller Manager continuously monitors the cluster to ensure tasks run correctly.

#####

Note:

#####

-> A cluster in Kubernetes (K8s) refers to a group of servers (nodes) that work together to run containerized applications. It consists of:

a. Control Plane (Master Node) – Manages and controls the cluster.

b. Worker Nodes (Slave Nodes) – Run application workloads inside containers. LINE ==160

