Load balancing

Load balancing is the process of distributing incoming network traffic or computational tasks across multiple servers or resources to ensure no single server is overwhelmed, and all requests are handled efficiently.

🔗 Why Load Balancing Matters
Without load balancing:

One server might receive too many requests and crash or slow down.

Other servers may sit underutilized.

Users could experience slow responses or downtime.

With load balancing:

Requests are spread across multiple servers.

Performance, reliability, and availability are greatly improved.

Failover and redundancy are possible (if one server fails, others take over).

Kubernetes doesn't manage containers directly.

Instead, containers are wrapped inside Pods.

So, you don’t tell Kubernetes: “run this container” —

you say: “run this Pod, which contains this container.”

Kubernetes Architecture

Control Plane (Master) Components – Manage the cluster.

Worker Nodes (Minions) – Run the actual applications (containers).

Control Plane Components

kube-apiserver: Frontend of the control plane. All communication goes through this REST API.

etcd: Key-value store that holds the cluster’s state and configuration data.

kube-scheduler: Assigns pods to nodes based on resource requirements and policies.

kube-controller-manager: Runs controllers like Node, Replication, and Endpoints controllers.

Node (Worker) Components

kubelet Primary node agent. Ensures containers are running in a Pod.
kube-proxy Maintains network rules for Pod communication and service discovery.

Container Runtime Software used to run containers (e.g., Docker, containerd, CRI-O).

Kubernetes (host on the top of docker inside VM)

Kubernetes (often abbreviated as K8s) is an open-source platform for automating the deployment, scaling, and management of containerized applications. It was originally developed by Google and is now maintained by the Cloud Native Computing Foundation (CNCF).

📦 At its core, Kubernetes does the following:

1. Container Orchestration
It manages groups of containers (usually Docker) across a cluster of machines, ensuring they run reliably and efficiently.

2. Scheduling and Load Balancing
Kubernetes automatically schedules containers on nodes based on resource requirements and availability, and can load-balance traffic between them.

3. Self-Healing
If a container crashes, Kubernetes can automatically restart, replace, or reschedule it.

4. Scaling
You can scale applications up or down (manually or automatically) based on demand.

5. Service Discovery and Networking
Kubernetes handles internal DNS, IP management, and allows containers to discover and communicate with each other securely.

6. Configuration Management and Secrets
You can manage application configs and sensitive information (secrets) without rebuilding images.

🔧 Key Kubernetes Concepts

Pod - The smallest deployable unit, usually a single container or a group of tightly coupled containers. (In Memory (as running processes))

Node - A physical or virtual machine that runs your application.

Cluster - A set of nodes managed by Kubernetes.

Deployment - Defines how to deploy and manage a set of Pods (e.g. update strategy, replicas).

Service - Exposes a Pod (or group of Pods) internally or externally and provides load balancing.

Namespace A way to divide cluster resources between multiple users or projects.

🚀 Why Use Kubernetes?

Handles complex container infrastructure

Improves application availability

Simplifies CI/CD pipelines

Works across clouds and on-premises

Has a large and active ecosystem (Helm, Prometheus, Istio, etc.)

Practically

namespace (isolated environment) inside the namespace we have

container (mostly docker) - pod (pod manages single or multiple containers) - deployment (write how many pods manage replicas) - service (from which user can access pod) - user

First we have to create namespace then pod - deployment - service

- kubectl get nodes

Namespace

- kubectl create ns nginx

- kubectl get ns

- kubectl delete ns nginx

---

pod - image dockerhub se uthata hai

- kubectl run nginx --image=nginx -n nginx - To create pod inside namespace

- kubectl get pods -n nginx

- kubectl delete pod nginx -n nginx

- kubectl exec -it nginx-pod -n nginx -- bash

Deployment (To scale the pod) - deployment also able to create pod

Replica bana kar dega - replicaset / statefulset / deployment (rolling update)

replicaset is same as deployment but we cannot perform rolling update

- kubectl get deployment -n nginx

- kubectl scale deployment/nginx-deployment -n nginx --replicas=5 we can scale pods by using this

- kunectl set image deployment/nginx-deployment -n nginx nginx=nginx:1.27.3

[rolling update we can update the image] [we are doing some changes on some pods remaining pods still running called rolling updates]

YAML Files (Manifest files)

- kubectl apply -f namespace.yaml

Debugging logs

- kubectl describe pod/nginx-pod -n nginx
