KUBERNETES

Kubernetes (often abbreviated as **K8s**) is an open-source platform for **automating the deployment, scaling, and management of containerized applications**. It helps run applications reliably across clusters of machines.

Structure of Kubernetes consists of a **master-node (control plane)** and multiple **worker nodes**.

1. Control Plane (Master Node)

Responsible for managing the entire Kubernetes cluster.

- **API Server**: Entry point for all commands (kubectl or API calls).
- **Scheduler**: Assigns workloads (pods) to nodes based on resources and rules.
- **Controller Manager**: Ensures cluster state matches the desired configuration (e.g., restarts failed pods).
- **etcd**: A distributed key-value store to save cluster configuration and state.
- **Cloud Controller Manager** *(optional)*: Integrates with cloud provider APIs.

---

2. Worker Nodes

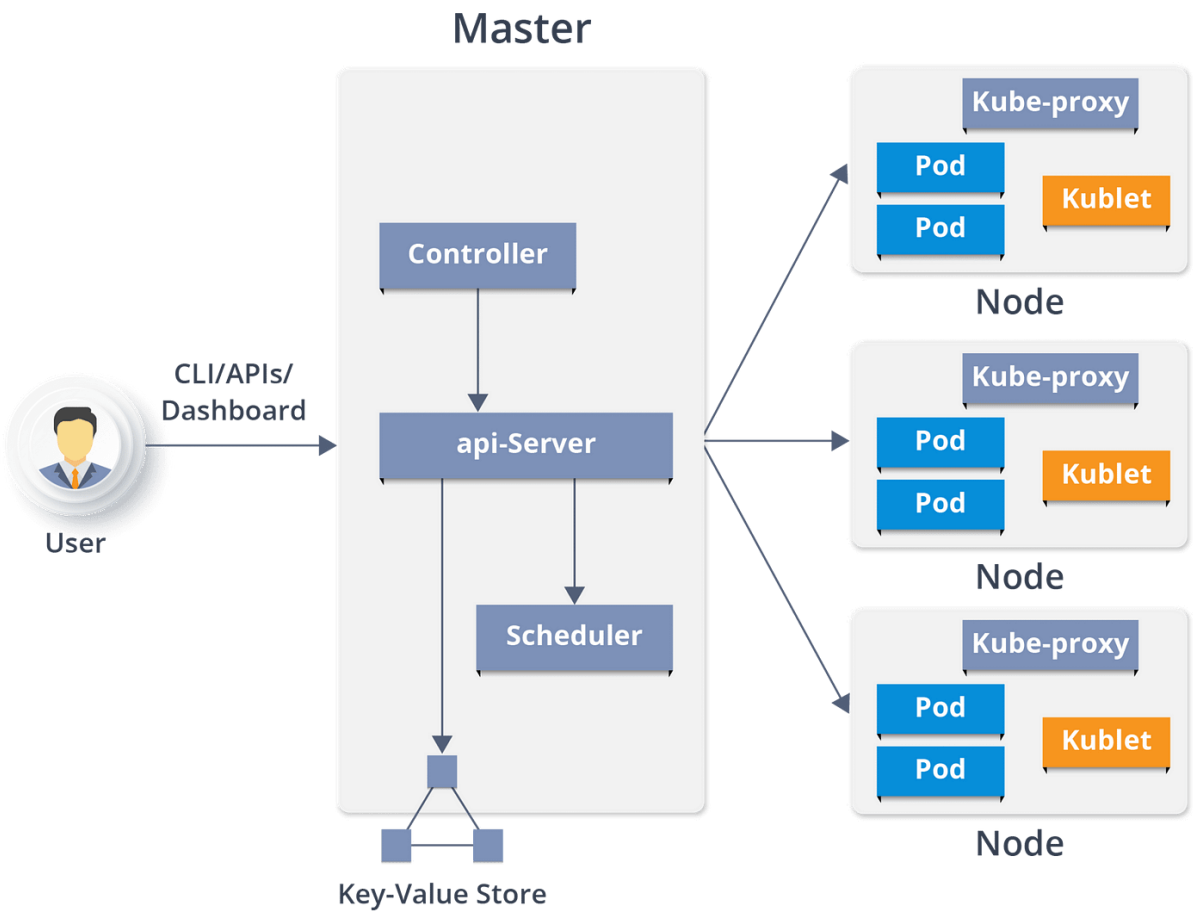
Run the actual applications (containers).

- **Kubelet**: Communicates with the control plane, manages containers on the node.
- **Kube Proxy**: Manages networking and load balancing for pods.
- **Container Runtime**: Runs containers (e.g., Docker, containerd).

---

3. Key Concepts

- **Pods**: The smallest deployable unit; contains one or more containers.
- **Services**: Expose pods to the network; handles load balancing.
- **Deployments**: Manage the rollout and scaling of pods.
- **Namespaces**: Virtual clusters within a Kubernetes cluster for isolation.



PODS

A **Pod** is the **smallest and simplest deployable unit** in Kubernetes. It represents **a single instance of a running process** in your cluster.

Key Features of a Pod:

1. **One or More Containers**
   Most Pods run **a single container**, but they can run multiple containers that need to share:
   - **Storage volumes**
   - **Network (same IP address & port space)**
2. **Shared Resources**
   - **Network namespace**: All containers in a Pod share the same IP and port range.
   - **Storage volumes**: Shared persistent or ephemeral storage.
3. **Lifecycle**
   - Pods are **ephemeral**: If a Pod dies, it's replaced by a new one (with a different ID).
   - Higher-level objects like **Deployments** or **StatefulSets** manage this replacement.

Example Use Cases:

- Running a **web server** container.
- A Pod with a main app container and a **sidecar** container (e.g., for logging or proxying).

Analogy:

A Pod is like a **wrapper** around one or more containers that are tightly coupled and should always be scheduled on the same machine.

Deployment

Deployment – [Deployment + Replica Set + Pod**]**

- **Purpose**: Manages stateless applications.
- **Use Case**: Automatically handles rolling updates, rollbacks, and scaling.

- **Behavior**: Creates **ReplicaSets** underneath to ensure a specified number of pod replicas.

Example: Running multiple instances of a web server like Nginx.

ReplicaSet

ReplicaSet [No. of Pods = No of Replica Set**]**

- **Purpose**: Ensures a specified number of pod replicas are always running.
- **Use Case**: Low-level component, typically not used directly—used by **Deployments**.
- **Behavior**: Watches pods and replaces them if they fail or are deleted.

Example: A Deployment creates a ReplicaSet which keeps 3 copies of a pod alive.

StatefulSet

StatefulSet

- **Purpose**: Manages **stateful applications** that need **persistent storage** and **stable network identity**.
- **Use Case**: Databases like MySQL, MongoDB, Cassandra.
- **Behavior**:
    - Pods get **unique, stable names** (e.g., mysql-0, mysql-1)
    - Supports ordered deployment, scaling, and updates.
    - Each pod can retain its **persistent volume** even after being deleted.

DeamonSet

- No of Pods = No of Worker Nodes

A **DaemonSet** ensures that **a specific Pod runs on all (or selected) nodes** in your Kubernetes cluster.

Key Features:

1. **Runs One Pod per Node**
    - Automatically adds the Pod to **every new node** added to the cluster.
    - Removes the Pod when a node is removed.
2. **Use Cases**:
    - **Log collection agents** (e.g., Fluentd, Filebeat)
    - **Monitoring agents** (e.g., Prometheus Node Exporter)
    - **Security tools** or **network plugins**
3. **No Replica Count Needed**
   Unlike Deployments, you don't specify replicas. It runs exactly one Pod **per node** (unless restricted by selectors or taints/tolerations).

Analogy:

Think of a DaemonSet like a **background service** that runs on every machine in your cluster.

Step-by-Step Process:

1. **You define the desired state**
   You create a YAML or JSON file describing what you want (e.g., "run 3 replicas of a web server") and submit it using kubectl or an API call.
2. **API Server receives the request**
   The **API Server** accepts and validates your request, then records the desired state in **etcd** (the cluster's database).
3. **Scheduler assigns Pods to Nodes**
   The **Scheduler** looks for a suitable node with enough resources and assigns the Pod to it.
4. **Kubelet creates the Pod on the Node**
   The **Kubelet** on that node reads the desired state, pulls the container

image, and starts the container using the **container runtime** (like Docker or containerd).

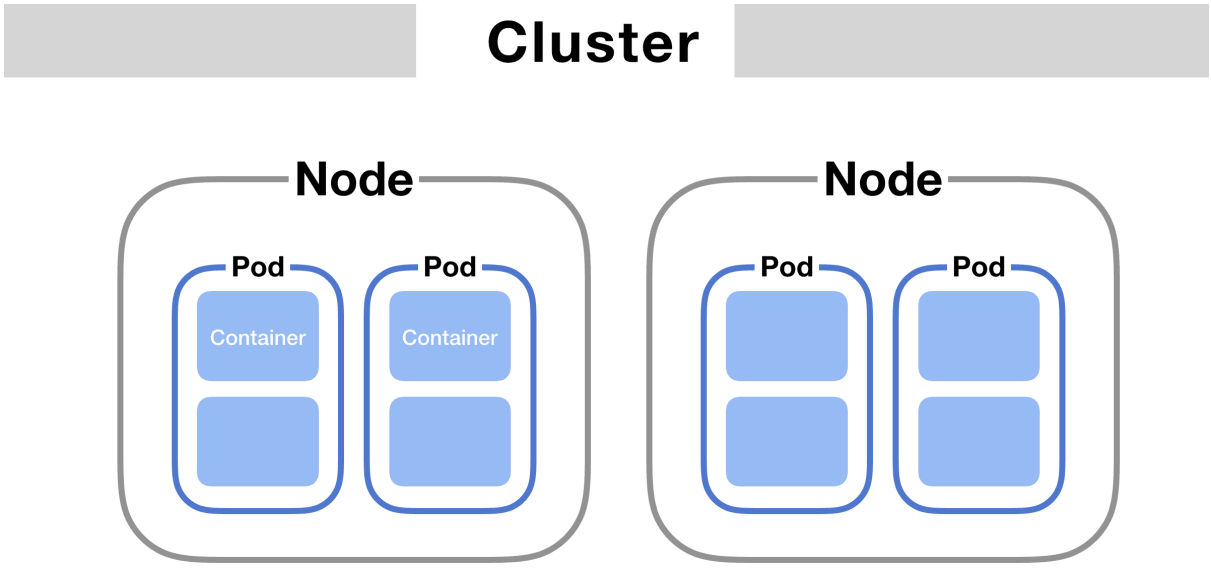
5. **Kube Proxy sets up networking**
   The **Kube Proxy** ensures the Pod can communicate with other Pods and Services inside or outside the cluster.
6. **Controller Manager ensures desired state**
   If a Pod crashes or a node fails, the **Controller Manager** detects it and recreates the Pod to match the desired state.

Kubernetes = Continuous Reconciliation

Kubernetes constantly monitors the current state and **automatically makes changes** to match the desired state you defined.

Diagrams and Sample Yaml Files

PODS



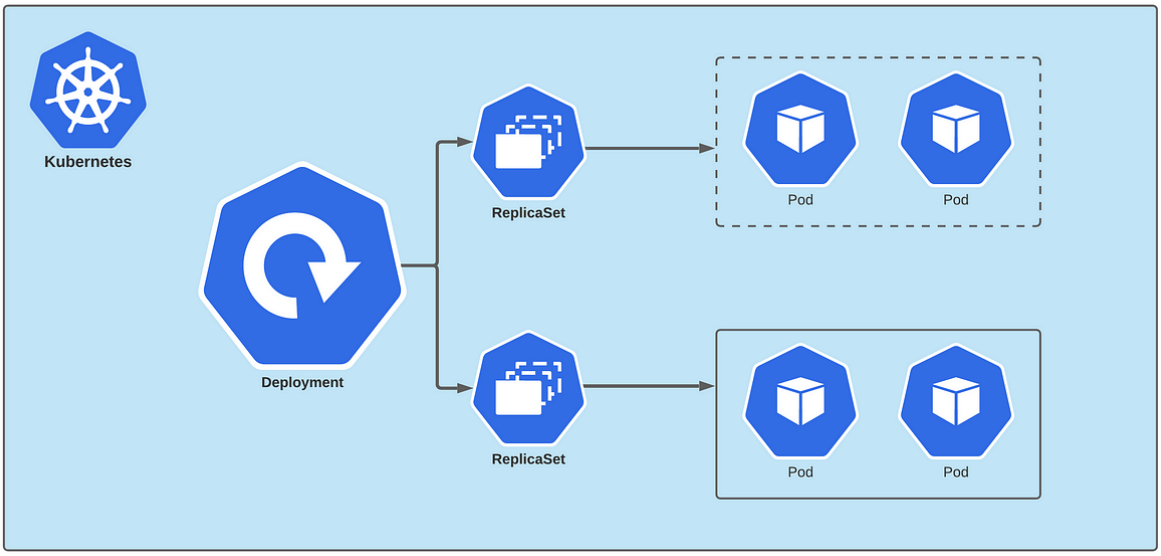
YAML FILE:

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

Explanation:

- **apiVersion**: Kubernetes API version to use (v1 for core resources like Pods).
- **kind**: Type of object (Pod).
- **metadata**: Name and labels for the Pod.
- **spec**: Specification of the containers inside the Pod.
- **containers**:
  - **name**: Name of the container.
  - **image**: Docker image to run (nginx:latest).
  - **ports**: Container port to expose (80 for web traffic).

Deployment



YAML FILE

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3  # Number of pod replicas
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
```

Explanation:

- **replicas: 3** — Runs 3 identical Pods for high availability.
- **selector.matchLabels** — Matches Pods with label app: nginx to manage them.
- **template** — Defines the Pod template (just like a regular Pod YAML).
- **image: nginx:latest** — Pulls the latest official Nginx image.

ReplicaSet

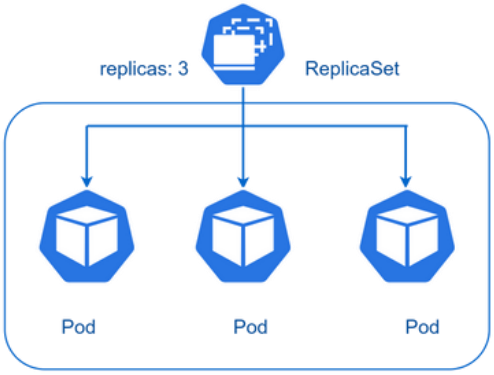


Fig: ReplicaSet

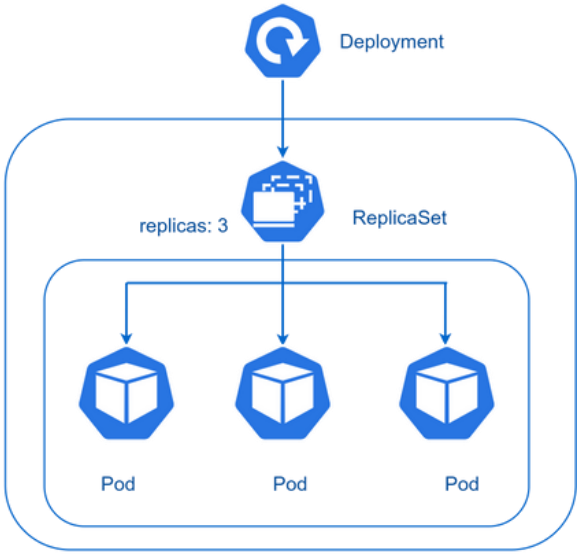
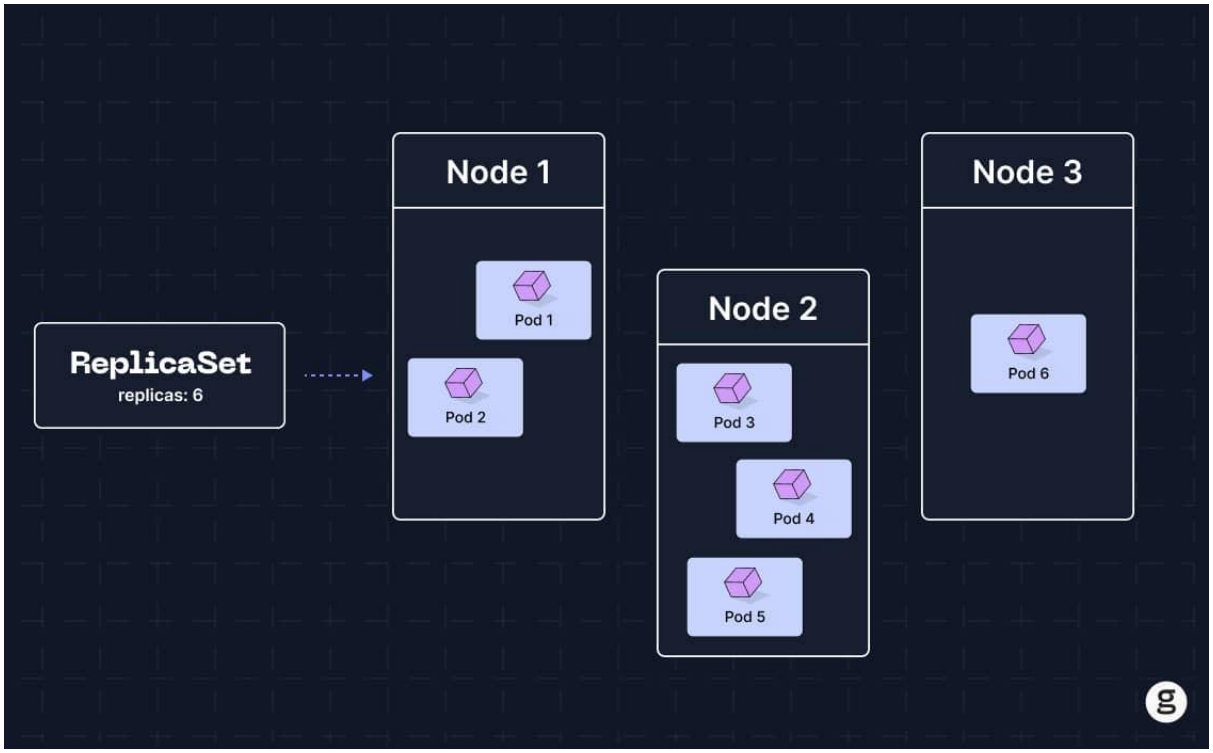


Fig: Deployment #goglides



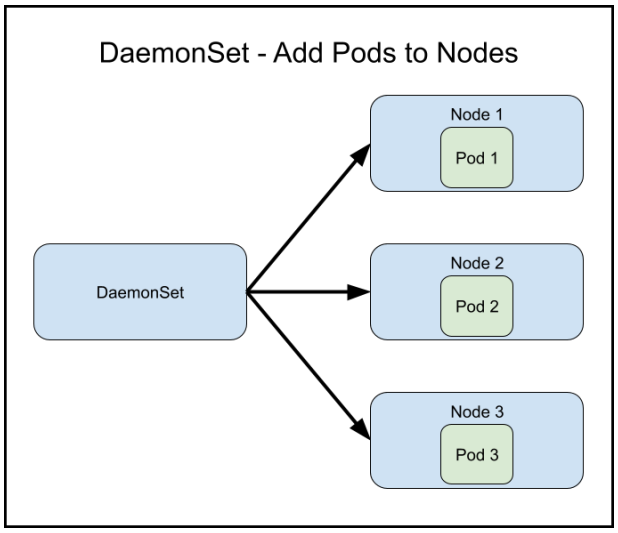
YAML FILE

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx-replicaset
  labels:
    app: nginx
spec:
  replicas: 3  # Desired number of Pods
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
```

Explanation:

- **kind: ReplicaSet** — Tells Kubernetes to manage Pods at the replica level.
- **replicas: 3** — Runs 3 instances of the Pod.
- **selector.matchLabels** — Must match the labels in the Pod template exactly.
- **template** — Pod specification just like a standalone Pod YAML

DeamonSet

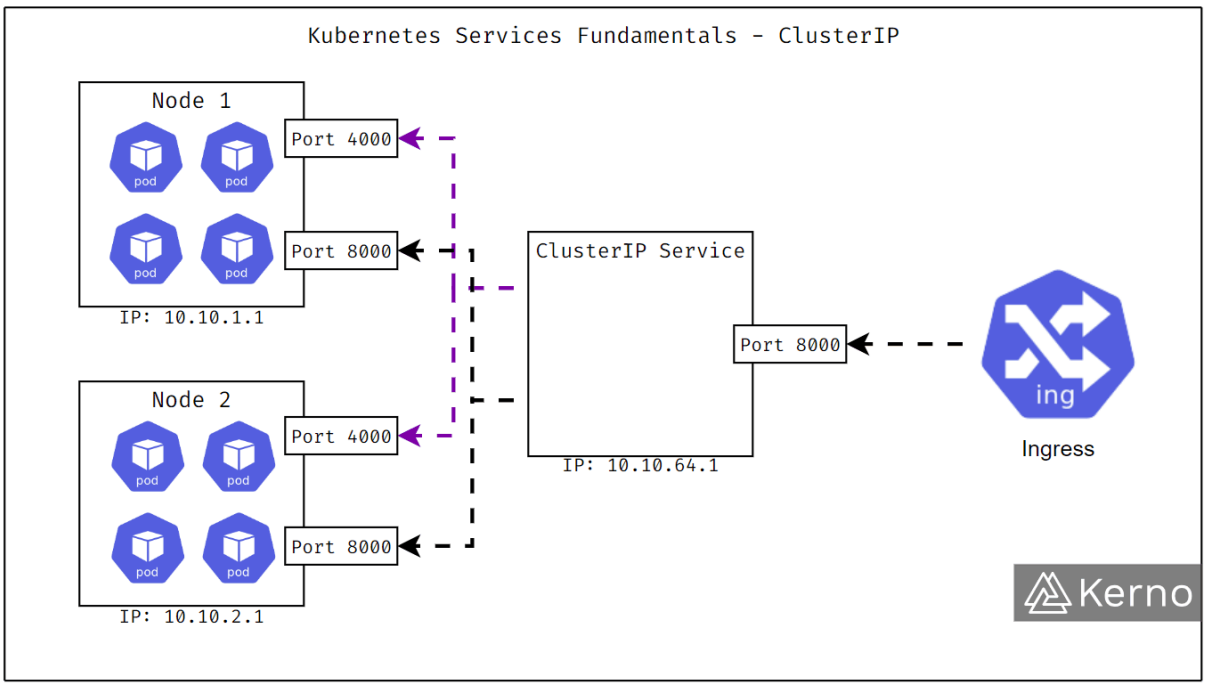


YAML FILE

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
 name: nginx-daemonset
 labels:
  app: nginx
spec:
 selector:
  matchLabels:
   name: nginx
 template:
  metadata:
   labels:
    name: nginx
  spec:
   containers:
   - name: nginx
    image: nginx:latest
    ports:
    - containerPort: 80
```

Explanation:

- **kind: DaemonSet** — Ensures a Pod is scheduled on every node.
- **selector.matchLabels** — Matches pods labeled with name: nginx.
- **template** — Pod template to run on each node.
- **image: nginx:latest** — Runs the official Nginx container.



ClusterIP

1. Exposes the Service **internally** within the Kubernetes cluster.
2. Assigns a **virtual IP (VIP)** that can only be accessed from inside the cluster.
3. Internal communication between microservices (e.g., frontend → backend).
4. A backend API accessed only by other services inside the cluster.

NodePort

1. Exposes the Service on a **static port (30000–32767)** on **each Node's IP**.
2. Maps a port on every node to your Service; you access it via NodeIP:NodePort.
3. Development, simple external access without a load balancer.
4. Access your app at http://<node-ip>:30080.

LoadBalancer

1. Creates an **external load balancer** using a cloud provider (AWS, Azure, GCP).
2. Assigns a **public IP** that routes traffic to your Service.
3. Production workloads needing public access.
4. Serving your app to end users via the internet.

ExternalName

1. Maps a Service name to an **external DNS name**.
2. Returns a CNAME record instead of routing traffic through the cluster.
3. Accessing external services (e.g., SaaS APIs) from within Kubernetes.
4. Internally using my-sql-service.default.svc.cluster.local to reach db.example.com.

YAML FILES

ClusterIP

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-clusterip
  labels:
    app: nginx
spec:
  selector:
    app: nginx  # This must match the pod/deployment label
  ports:
  - protocol: TCP
    port: 80          # Port to expose the service
    targetPort: 80    # Port on the Pod container
  type: ClusterIP     # Default type (can omit this line)
```

- **SELECTOR.APP: NGINX** MATCHES THE PODS FROM YOUR DEPLOYMENT (MUST MATCH LABELS).
- **PORT** IS HOW OTHER SERVICES INSIDE THE CLUSTER WILL CONNECT TO IT.
- **TARGETPORT** IS THE PORT YOUR NGINX CONTAINER IS LISTENING ON.

NodePort

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-nodeport
  labels:
    app: nginx
spec:
  type: NodePort          # Expose service via a port on each node
  selector:
    app: nginx            # Must match labels of the target pods
  ports:
  - protocol: TCP
    port: 80              # Internal service port
    targetPort: 80        # Container port in the pod
    nodePort: 30080       # Optional: static port (range 30000–32767)
```

- **CCESS THE APP AT**: HTTP://<NODE-IP>:30080
- **PORT**: CLIENTS INSIDE THE CLUSTER USE THIS.
- **NODEPORT**: EXTERNAL CLIENTS USE THIS TO REACH THE SERVICE.
- **TARGETPORT**: THE ACTUAL CONTAINER PORT INSIDE THE POD.

IF YOU OMIT NODEPORT, KUBERNETES WILL AUTO-ASSIGN ONE FROM THE 30000–32767 RANGE.

LoadBalancer

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-loadbalancer
  labels:
    app: nginx
spec:
  type: LoadBalancer          # Requests an external load balancer
  selector:
    app: nginx                # Must match the Deployment/Pod labels
  ports:
  - protocol: TCP
    port: 80                  # Exposed service port
    targetPort: 80            # Pod/container port
```

- CREATES A **PUBLIC IP** AND **LOAD BALANCER** (ON AWS, GCP, AZURE, ETC.).
- TRAFFIC TO THE EXTERNAL IP IS **LOAD-BALANCED** TO THE BACKEND PODS.
- THE SERVICE TARGETS PODS WITH APP: NGINX LABELS.