**Kubernetes - Introduction & Architecture**

**Table of Contents**

1. **Introduction to Kubernetes**
   * Why Kubernetes?
   * Comparison with Docker Swarm
2. **Kubernetes Architecture**
   * Control Plane (Master Node) Components
     + API Server
     + etcd
     + Scheduler
     + Controller Manager
   * Data Plane (Worker Node) Components
     + kubelet
     + kube-proxy
   * Interaction Flow
3. **Key Terms & Definitions**
   * Pods, Services, Deployments, ReplicaSets
   * NodePort vs. ClusterIP vs. LoadBalancer
4. **Real-World Use Cases**
   * Uber (175+ containers)
   * Swiggy (75+ containers)
   * General Microservices Deployment
5. **Prerequisites for Kubernetes Deployment**
   * DNS Name
   * AWS Account Setup
   * S3 Bucket & IAM Roles
6. **Conclusion & Next Steps**
7. Key Features of Kubernetes (K8s) with Simple Examples

**1. Introduction to Kubernetes**

**Why Kubernetes?**

* Kubernetes is a **container orchestration tool** that automates deployment, scaling, and management of containerized applications.
* Docker Swarm (another orchestration tool) has limitations:
  + No **TLS Secrets**
  + No **Role-Based Access Control (RBAC)**
  + No **Namespaces**
  + No **StatefulSets**
  + No **Readiness/Liveness Probes** (Health Checks)
  + No **Environment Variable Secrets**
* Organizations are migrating from Docker Swarm to Kubernetes due to its **scalability, robustness, and advanced features**.

## **Brief of above terms:** **1. No TLS Secrets**

**What is a TLS Secret?**

* TLS (Transport Layer Security) ensures secure communication between services (like HTTPS).
* **Secrets** are Kubernetes objects used to store sensitive data (passwords, API keys, certificates).

**Problem in Docker Swarm:**

* Docker Swarm does not natively support storing TLS certificates securely.
* You’d have to manually manage certificates, increasing security risks.

**Example:**

* In Kubernetes, you can store an SSL certificate as a **Secret**:

sh

kubectl create secret tls my-cert --cert=server.crt --key=server.key

→ Kubernetes encrypts and securely manages this certificate.

**2. No Role-Based Access Control (RBAC)**

**What is RBAC?**

* RBAC restricts who can access and modify resources in a cluster.
* Example roles:
  + **Admin** (full access)
  + **Developer** (can only deploy apps)
  + **Viewer** (read-only access)

**Problem in Docker Swarm:**

* No built-in way to restrict user permissions.
* Everyone with access can modify anything (security risk).

**Example in Kubernetes:**

yaml

apiVersion: rbac.authorization.k8s.io/v1

kind: Role

metadata:

name: dev-role

rules:

- apiGroups: [""]

resources: ["pods"]

verbs: ["get", "list", "create"] *# Only allows viewing & creating pods*

→ Developers can’t delete pods or access secrets.

**3. No Namespaces**

**What are Namespaces?**

* Namespaces divide a cluster into virtual sections (like folders).
* Example:
  + dev, prod, testing (isolated environments).

**Problem in Docker Swarm:**

* All containers run in a single global space.
* No isolation between teams (e.g., Dev vs. Production).

**Example in Kubernetes:**

sh

kubectl create namespace dev

kubectl run nginx --image=nginx --namespace=dev

→ nginx runs only in the dev namespace, separate from prod.

**4. No StatefulSets**

**What is a StatefulSet?**

* Manages **stateful applications** (databases, storage-dependent apps).
* Ensures:
  + Stable network hostnames (db-0, db-1).
  + Persistent storage (data isn’t lost if pod restarts).

**Problem in Docker Swarm:**

* Only supports **stateless** apps (containers are disposable).
* Databases (MySQL, MongoDB) would lose data if moved.

**Example in Kubernetes:**

yaml

apiVersion: apps/v1

kind: StatefulSet

metadata:

name: mysql

spec:

replicas: 3

serviceName: mysql

template:

spec:

containers:

- name: mysql

image: mysql:5.7

volumeMounts:

- name: mysql-storage

mountPath: /var/lib/mysql

→ Each MySQL pod keeps its data even if rescheduled.

**5. No Readiness/Liveness Probes (Health Checks)**

**What are Probes?**

* **Liveness Probe:** Checks if a container is running (restarts if failed).
* **Readiness Probe:** Checks if a container is ready to serve traffic.

**Problem in Docker Swarm:**

* No automatic health checks → If an app crashes, Swarm won’t know.

**Example in Kubernetes:**

yaml

containers:

- name: nginx

livenessProbe:

httpGet:

path: /health

port: 80

initialDelaySeconds: 5

periodSeconds: 10

→ If /health fails, Kubernetes restarts the container.

**6. No Environment Variable Secrets**

**What are Env Secrets?**

* Securely injects passwords/API keys into containers.

**Problem in Docker Swarm:**

* Secrets are passed as plain text or files (less secure).

**Example in Kubernetes:**

sh

kubectl create secret generic db-creds --from-literal=password=12345

→ Pods can safely access password without exposing it.

**Summary Table**

| **Feature** | **Docker Swarm** | **Kubernetes** | **Why It Matters** |
| --- | --- | --- | --- |
| **TLS Secrets** | ❌ No | ✅ Yes | Secure HTTPS communication |
| **RBAC** | ❌ No | ✅ Yes | Prevents unauthorized access |
| **Namespaces** | ❌ No | ✅ Yes | Isolate teams (Dev/Prod) |
| **StatefulSets** | ❌ No | ✅ Yes | Run databases reliably |
| **Health Checks** | ❌ No | ✅ Yes | Auto-recover crashed apps |
| **Env Secrets** | ❌ No | ✅ Yes | Secure password management |

### **Comparison: Docker Swarm vs. Kubernetes**

| **Feature** | **Docker Swarm** | **Kubernetes** |
| --- | --- | --- |
| **Secrets Management** | Limited | Yes (TLS, Env Secrets) |
| **RBAC** | No | Yes |
| **Namespaces** | No | Yes |
| **Stateful Apps** | No (Stateless only) | Yes (StatefulSets) |
| **Health Checks** | No | Yes (Readiness/Liveness Probes) |
| **Auto-Scaling** | Manual | Horizontal Pod Autoscaler (HPA) |

## **2. Kubernetes Architecture**

### **Control Plane (Master Node)**

Manages the cluster and makes global decisions.

#### **Components:**

1. **API Server**
   * Entry point for all commands (kubectl).
   * Validates requests, handles authentication, and updates etcd.
   * Example:

sh

kubectl create deployment nginx --image=nginx --replicas=6

→ Request goes to API Server first.

1. **etcd**
   * Distributed key-value store (Kubernetes’ database).
   * Stores cluster state (deployments, pods, nodes).
2. **Scheduler**
   * Assigns pods to worker nodes based on resource availability.
   * Example: If a pod needs 2GB RAM, scheduler places it on a node with free RAM.
3. **Controller Manager**
   * Ensures the cluster’s desired state matches the actual state.
   * Includes:
     + **Node Controller** (Handles node failures)
     + **Replication Controller** (Ensures correct pod count)

### **Data Plane (Worker Nodes)**

Runs the actual workloads (containers).

#### **Components:**

1. **kubelet**
   * Runs on each worker node.
   * Ensures containers are running (restarts if crashed).
   * Only component running as a **daemon service** (not as a pod).
2. **kube-proxy**
   * Manages network rules (IP assignment, load balancing).
   * Enables communication between pods.

## **3. Key Terms & Definitions**

| **Term** | **Definition** | **Example** |
| --- | --- | --- |
| **Pod** | Smallest deployable unit (1+ containers) | nginx pod running a web server |
| **Service** | Exposes pods (NodePort, ClusterIP, LoadBalancer) | Access nginx via http://<NodeIP>:30000 |
| **Deployment** | Manages pod replicas | kubectl create deploy nginx --replicas=3 |
| **ReplicaSet** | Ensures pod count matches spec | If 1 pod crashes, ReplicaSet creates a new one |
| **NodePort** | Exposes a pod on a static port | 30000-32767 |
| **ClusterIP** | Internal IP for pod communication | Only accessible within the cluster |

## **4. Real-World Use Cases**

* **Uber:** 175+ containers managed by Kubernetes.
* **Swiggy:** 75+ containers for food delivery.
* **Microservices:**
  + Frontend, Backend, and DB run in separate pods.
  + Auto-scaling during peak traffic.

## **5. Prerequisites for Kubernetes Deployment**

1. **DNS Name** (Purchase from providers like .xyz).
2. **AWS Account** (For cloud-based clusters).
3. **S3 Bucket** (Storing cluster state).
4. **IAM Roles** (Permissions for Kubernetes nodes).

## **6. Conclusion & Next Steps**

* Practice commands:

sh

kubectl get nodes

kubectl get pods -A

# **Key Features of Kubernetes (K8s) with Simple Examples**

Kubernetes (K8s) is like a **smart robot manager** for your apps. Here are its **key features** with **real-world examples** to understand how they work.

## **1. Automatic Scaling**

**What it does:**

* Automatically **increases or decreases** the number of running apps based on demand.

**Example:**

* Your website gets **1000 visitors/day** → Runs **2 Pods (containers)**.
* On Black Friday, traffic spikes to **1M visitors** → Kubernetes scales up to **50 Pods**.
* When traffic drops, it scales back down.

**Command Example:**

sh

kubectl autoscale deployment my-web-app --min=2 --max=10 --cpu-percent=80

→ If CPU usage goes above 80%, Kubernetes adds more Pods (up to 10).

## **2. Self-Healing (Auto-Recovery)**

**What it does:**

* If an app crashes, Kubernetes **restarts it automatically**.
* If a server dies, it moves apps to **healthy servers**.

**Example:**

* Your nginx Pod crashes → Kubernetes **restarts it**.
* The entire Worker Node fails → Kubernetes **moves all Pods** to another node.

**Command Example:**

sh

kubectl get pods # Shows running Pods (if one is "CrashLoopBackOff", K8s is trying to fix it)

## **3. Load Balancing**

**What it does:**

* Distributes traffic **evenly** across multiple Pods to avoid overloading one.

**Example:**

* You have **3 Pods** running nginx.
* When users visit your site, traffic is split **33% to each Pod**.

**YAML Example:**

yaml

apiVersion: v1

kind: Service

metadata:

name: my-load-balancer

spec:

type: LoadBalancer

ports:

- port: 80

selector:

app: nginx

→ Traffic to my-load-balancer is spread across all nginx Pods.

## **4. Rolling Updates & Rollbacks**

**What it does:**

* Updates apps **without downtime** (replaces Pods one by one).
* If something goes wrong, it **rolls back to the old version**.

**Example:**

* You update my-app:v1 → my-app:v2.
* Kubernetes replaces **one Pod at a time** (so your app stays online).
* If v2 has a bug, Kubernetes **switches back to**v1.

**Command Example:**

sh

kubectl set image deployment/my-app my-app=my-app:v2 # Updates to v2

kubectl rollout undo deployment/my-app # Rolls back to v1

## **5. Storage Management**

**What it does:**

* Attaches **permanent storage** (like databases) even if Pods restart.

**Example:**

* A MySQL Pod dies → A new one starts **with the same data**.

**YAML Example:**

yaml

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: mysql-storage

spec:

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 10Gi

→ Even if the MySQL Pod restarts, data is **not lost**.

## **6. Secrets & Config Management**

**What it does:**

* Securely stores **passwords, API keys, and configs**.

**Example:**

* Your app needs a **database password** → Kubernetes injects it securely.

**Command Example:**

sh

kubectl create secret generic db-password --from-literal=pass=12345

→ Pods can access pass=12345 **without exposing it in code**.

## **7. Multi-Cloud & Hybrid Deployments**

**What it does:**

* Runs apps **on any cloud (AWS, Azure, GCP) or even your laptop**.

**Example:**

* Devs test on **local Kubernetes (Minikube)**.
* Production runs on **AWS (EKS)**.  
  → **Same setup** everywhere!

## **Summary Table**

| **Feature** | **What It Does** | **Example** |
| --- | --- | --- |
| **Auto-Scaling** | Adds/removes Pods based on traffic | kubectl autoscale |
| **Self-Healing** | Restarts failed apps | Crashed Pod → K8s fixes it |
| **Load Balancing** | Splits traffic across Pods | 3 Pods → Traffic divided evenly |
| **Rolling Updates** | Updates apps with no downtime | v1 → v2 (no crashes) |
| **Storage Mgmt** | Keeps data safe after Pod restarts | MySQL data persists |
| **Secrets Mgmt** | Securely stores passwords | kubectl create secret |
| **Multi-Cloud** | Runs anywhere (AWS, Azure, laptop) | Same config on all clouds |

### **Try It Yourself!**

1. Install **Minikube** (local Kubernetes):

sh

minikube start

1. Run your first app:

sh

kubectl create deployment hello-world --image=nginx

1. Check if it’s running:

sh

kubectl get pods

Kubernetes is like a **super-smart assistant**—it **scales, heals, and manages** your apps automatically! 🚀