Kubernetes Concepts: Pods, Nodes, Master Node, Worker Node & Components

Pod

- The **smallest unit** in Kubernetes.
- A **Pod runs one or more containers** (usually one).
- All containers in a Pod share the same **network** and **storage**.

Think of it like a wrapper around your application container.

🖥 Node

- A **physical or virtual machine** where Kubernetes runs.
- Two types:
    - **Master Node** (Control Plane)
    - **Worker Node** (where your apps actually run)

👑 Master Node (Control Plane)

Manages the whole Kubernetes cluster.

Key Components:

- `kube-apiserver`: Frontend for the control plane (talks to users and other parts).
- `etcd`: Stores all cluster data (like a database).
- `kube-scheduler`: Assigns Pods to Nodes.
- `kube-controller-manager`: Watches cluster state and responds (e.g., starts/stops Pods).
- `cloud-controller-manager`: Integrates cloud-specific functions (if using AWS, GCP, etc.)

⚙ Worker Node

Runs the actual application workloads.

Key Components:

- `kubelet`: Talks to the master, runs containers in Pods.
- `kube-proxy`: Handles networking and load balancing.
- **Container Runtime**: Like Docker, containerd, or CRI-O to run containers.

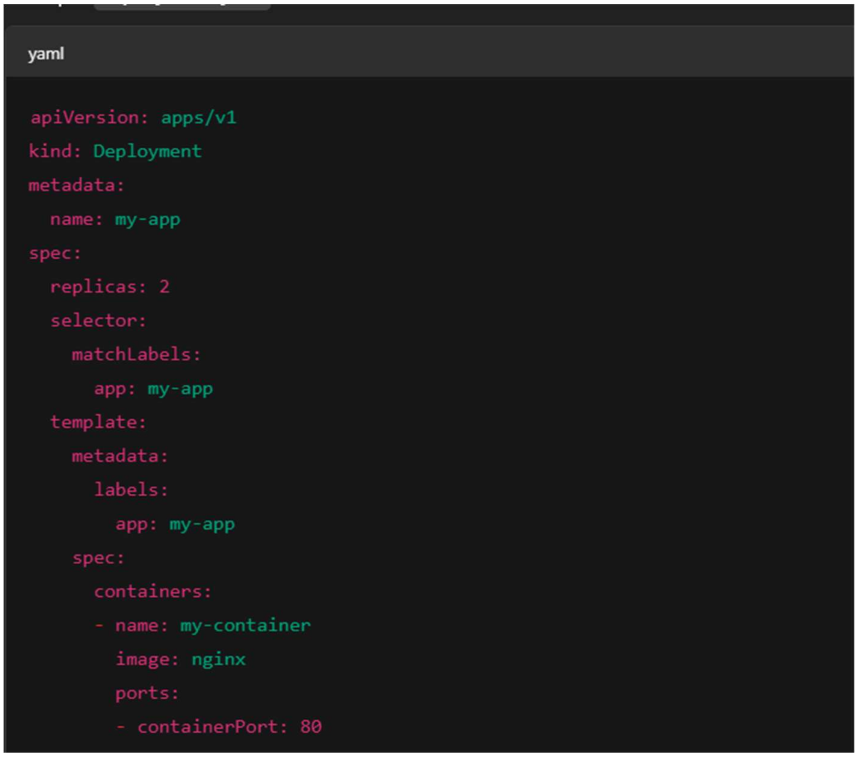
🚀 2. Application Deployment: Requirement Files & Where to Deploy

📝 Requirement File: `deployment.yaml`

This YAML file defines:

- **What to deploy** (app/container image)
- **How many replicas**
- **Which port to expose**
- **Resource limits**
- **Environment variables**
- **Volumes** if needed

Example `deployment.yaml`:



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: my-container
        image: nginx
        ports:
        - containerPort: 80
```

To deploy this:

```
bash
CopyEdit
kubectl apply -f deployment.yaml
```

You usually **deploy on the Kubernetes cluster** (either local like Minikube, or cloud like GKE/EKS/AKS).

🛠 3. Basic Docker & Kubernetes Commands Used in Real Projects

🐳 Docker Commands

Task	Command
Build Image	`docker build -t my-image .`
Run Container	`docker run -d -p 8080:80 my-image`
List Containers	`docker ps`
Stop Container	`docker stop <container-id>`
Remove Container	`docker rm <container-id>`
Remove Image	`docker rmi <image-id>`
Push to Docker Hub	`docker push my-image`

☸ Kubernetes Commands

Task	Command
Check Nodes	`kubectl get nodes`
Check Pods	`kubectl get pods`
Apply Deployment	`kubectl apply -f deployment.yaml`
Check Deployments	`kubectl get deployments`
Describe Pod	`kubectl describe pod <pod-name>`
Delete Deployment	`kubectl delete -f deployment.yaml`
Logs from Pod	`kubectl logs <pod-name>`
Exec into Pod	`kubectl exec -it <pod-name> -- /bin/bash`
Port Forward Pod	`kubectl port-forward <pod-name> 8080:80`

✅ Summary

- **Docker**: Packages your app into containers.
- **Kubernetes**: Deploys and manages those containers at scale.
- **YAML files** define deployments.
- **Master Node** controls the cluster; **Worker Nodes** run the actual containers.
- Use **basic `kubectl` commands** daily to check, deploy, and debug.

Let me know if you want:

- A simple practice project
- Local setup steps with Minikube or Docker Desktop
- A cheatsheet PDF or Markdown version

Happy DevOps learning! 💻⚙

4o

You said:

same for Docker

ChatGPT said:

Absolutely! Here's a beginner-friendly **Docker Theory Guide** tailored for someone starting out in a **DevOps role**, just like we did for Kubernetes.

🐳 Docker Explained

✅ 1. What is Docker?

Docker is an open-source platform that allows you to:

- Package applications with their dependencies into **containers**.
- Run those containers consistently across different environments—dev, test, production.

 Think of Docker as a lightweight, portable "box" for your app.

📦 2. Core Docker Concepts

🔹 Image

- A **blueprint** for a container.
- Built using a `Dockerfile`.
- Example: `nginx`, `ubuntu`, `python:3.10`

🔹 Container

- A **running instance** of an image.
- Lightweight and isolated from the host system.

◆ Dockerfile

- A text file that defines **how to build an image**.
- Contains instructions like `FROM`, `COPY`, `RUN`, `CMD`.

Example Dockerfile:

```
Dockerfile
CopyEdit
FROM node:18
WORKDIR /app
COPY . .
RUN npm install
CMD ["node", "app.js"]
```

⚙ 3. Docker Architecture Overview

□ Components:

- **Docker Client**: CLI tool to run Docker commands.
- **Docker Daemon**: Background service that builds, runs, and manages containers.
- **Docker Images**: Stored in **Docker Hub** (public) or private registries.
- **Docker Containers**: Running instances of Docker images.

📡 *The client talks to the daemon, which pulls images, builds containers, and runs them.*

🚀 4. Docker Application Deployment

You usually:

1. Write a `Dockerfile` to define your app environment.
2. Build it into an image:

   ```
   bash
   CopyEdit
   docker build -t myapp:latest .
   ```

3. Run it as a container:

   ```
   bash
   CopyEdit
   docker run -d -p 3000:3000 myapp
   ```

4. Optionally push it to a registry:

   ```
   bash
   CopyEdit
   docker push username/myapp:latest
   ```

📂 You can also use **`docker-compose.yml`** to run multi-container apps.

🛠 5. Common Docker Commands (Used Daily in Real Projects)

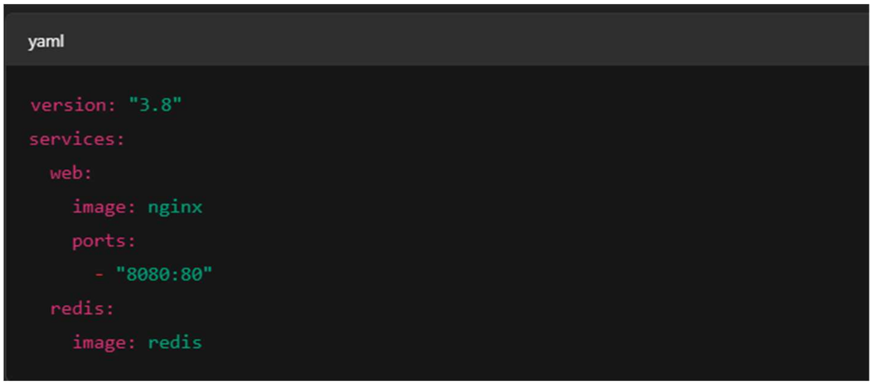
Task	Command
Check Docker version	`docker --version`
Build an image	`docker build -t myapp .`
Run a container	`docker run -d -p 8080:80 myapp`
List containers	`docker ps`
List all (even stopped) containers	`docker ps -a`
Stop a container	`docker stop <container-id>`
Remove a container	`docker rm <container-id>`
Remove an image	`docker rmi <image-id>`
See image list	`docker images`
Container logs	`docker logs <container-id>`
Shell access	`docker exec -it <container-id> /bin/bash`
Tag an image	`docker tag myapp username/myapp:v1`
Push image to Docker Hub	`docker push username/myapp:v1`
Pull an image	`docker pull nginx`

□ 6. Why Use Docker in DevOps?

- Consistency across environments
- Simplifies CI/CD pipelines
- Faster deployment and rollback
- Easier scaling with Kubernetes
- Efficient use of system resources

💼 Optional: Docker Compose (Multi-Container Setup)

`docker-compose.yml` example:



```yaml
version: "3.8"
services:
  web:
    image: nginx
    ports:
      - "8080:80"
  redis:
    image: redis
```

Run all services:

```
bash
CopyEdit
docker-compose up -d
```

✅ Summary

Concept	Docker	Kubernetes
Packaging	Uses Images & Containers	Uses Pods (which hold containers)
Deployment	Manual or via Compose	Declarative via YAML
Scaling	Manual	Auto-scaling supported
Best Use	Dev/testing, small apps	Production, microservices at scale

Docker Interview Questions & Answers

---

1. What is Docker, and why do we use it?

Answer: Docker is a platform for developing, shipping, and running applications in containers. It's used to ensure **environment consistency**, **faster deployment**, and **isolated app execution** across different stages of development.

---

2. What is the difference between a Docker image and a Docker container?

Answer:

- **Image**: Blueprint or template for creating containers. It's read-only.
- **Container**: A running instance of an image with its own filesystem, process tree, and network.

---

3. How do you create a Docker image?

Answer:
By writing a `Dockerfile`, then using the build command:

```
bash
CopyEdit
docker build -t myimage:latest .
```

---

4. How do you share a Docker image with others or in CI/CD?

Answer:
Push it to a registry like Docker Hub or a private registry:

```
bash
CopyEdit
docker tag myimage username/myimage:v1
docker push username/myimage:v1
```

---

5. What is the use of Docker Compose?

Answer:
Docker Compose is used to run **multi-container applications** using a single `docker-compose.yml` file. It simplifies deployment of services like databases + apps together.

---

6. What is the difference between Docker and a Virtual Machine?

Answer:

- Docker uses **OS-level virtualization**; containers share the host OS kernel.
- VMs run **full OS** instances and are heavier.
- Docker is faster and more lightweight.

7. How do you troubleshoot a non-starting container?

Answer:

- Check logs: `docker logs <container-id>`
- Inspect container: `docker inspect <container-id>`
- Try entering: `docker exec -it <container-id> /bin/bash`

☸ Kubernetes Interview Questions & Answers

1. What is Kubernetes and why do we use it?

Answer:
Kubernetes is a **container orchestration platform** used to manage containerized applications at scale. It handles deployment, scaling, networking, and self-healing of containers automatically.

2. What is a Pod in Kubernetes?

Answer:
A **Pod** is the smallest unit in Kubernetes. It wraps one or more containers that share the same network and storage. Usually, one container per pod is used.

3. What is the difference between a Pod and a Deployment?

Answer:

- **Pod**: Direct container execution unit.
- **Deployment**: A higher-level abstraction that manages **replicas of pods**, rollout strategies, and self-healing.

4. What is the role of the kubelet and kube-proxy?

Answer:

- **kubelet**: Runs on each node, ensures containers in Pods are running.
- **kube-proxy**: Handles networking and load balancing for Pods across nodes.

---

5. What are ConfigMaps and Secrets?

Answer:

- **ConfigMap**: Stores configuration data in key-value pairs.
- **Secret**: Stores sensitive data (e.g., passwords, API keys), base64-encoded.

---

6. How do you update a running application in Kubernetes?

Answer:
Update the deployment YAML or use:

```
bash
CopyEdit
kubectl set image deployment/myapp mycontainer=myimage:v2
```

This triggers a **rolling update**.

---

7. What is the difference between StatefulSet and Deployment?

Answer:

- **Deployment**: Best for stateless apps, no persistent identity.
- **StatefulSet**: For **stateful applications**, where each Pod has a unique identity and persistent storage (like databases).

---

8. How do you expose your application in Kubernetes?

Answer:
Using:

- **ClusterIP** (internal access)
- **NodePort** (external on node IP)
- **LoadBalancer** (cloud environments)
- Or **Ingress** (advanced HTTP routing)

---

9. What is the use of `kubectl describe` and `kubectl logs`?

Answer:

- `kubectl describe pod <pod-name>`: Shows detailed info about pod status, events.
- `kubectl logs <pod-name>`: Prints container logs for debugging.

---

10. What is a namespace in Kubernetes?

Answer:
Namespaces allow you to **divide cluster resources** between multiple users/projects. Useful in **multi-team environments** for resource isolation.