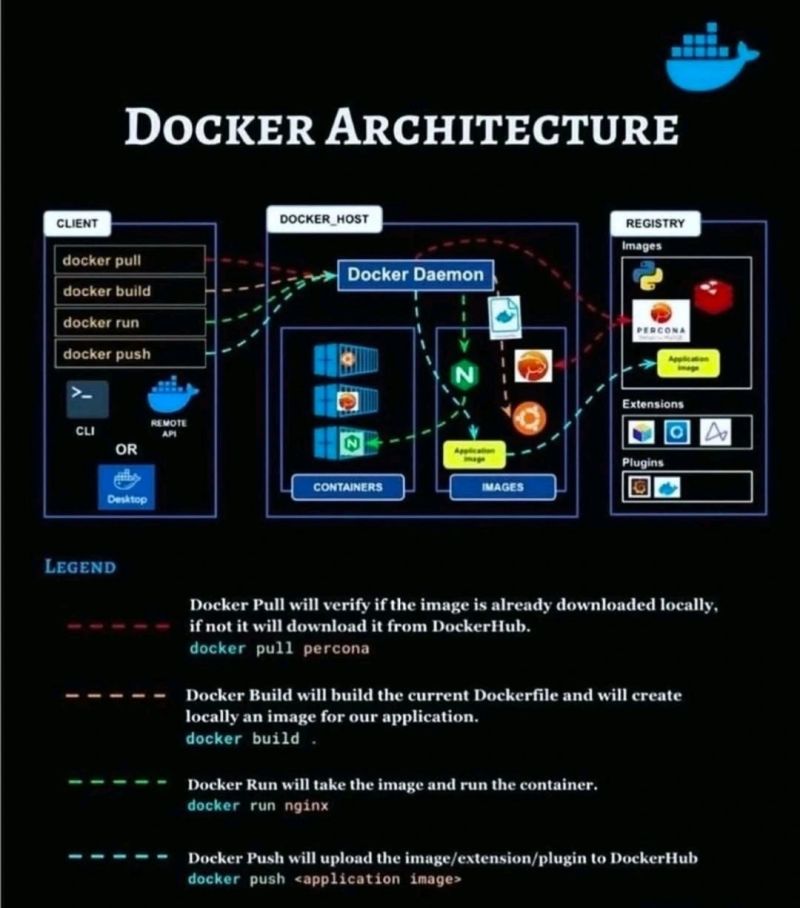
**Docker Architecture**



Docker follows a **client-server architecture** that enables containerization, packaging applications and dependencies into portable units.

**🔹 Docker Architecture Components**

**1️⃣ Docker Client**

* CLI (docker commands) or API-based communication with Docker daemon.
* Sends requests (docker build, docker run, docker push) to the Docker Engine.

**2️⃣ Docker Daemon (dockerd)**

* Runs in the background, managing images, containers, networks, and storage.
* Listens for client requests and handles container lifecycle.

**3️⃣ Docker Images**

* Blueprint/template for containers (e.g., nginx:latest).
* Built from Dockerfile → Layers are cached for efficiency.
* Stored in local cache or pushed to a container registry.

**4️⃣ Docker Container**

* Running instance of an image (lightweight, isolated process).
* Uses **UnionFS (OverlayFS)** to share storage across multiple containers.
* Containers are ephemeral but can persist data using **Volumes**.

**5️⃣ Docker Registry (Hub, ECR, ACR, GCR)**

* Stores and distributes Docker images.
* **Public:** Docker Hub, Quay.io
* **Private:** AWS ECR, Azure ACR, GCP GCR

**6️⃣ Docker Storage (Volumes & Bind Mounts)**

* **Volumes:** Managed by Docker, best for persistent storage.
* **Bind Mounts:** Directly maps host directories to containers.

**7️⃣ Docker Networking**

* **Bridge:** Default, enables container-to-container communication.
* **Host:** Shares host network (no isolation).
* **Overlay:** Multi-host communication (Docker Swarm/K8s).
* **Macvlan:** Assigns IPs to containers for direct access.

**🔹 How Docker Works? (Workflow Explanation)**

1️⃣ **Pull Image:** docker pull nginx:latest

2️⃣ **Run Container:** docker run -d -p 8080:80 nginx

3️⃣ **Container Execution:** Runs isolated with assigned resources.

4️⃣ **Networking:** Containers communicate via virtual networks.

5️⃣ **Persistent Data:** Volumes mounted (docker volume create app-data).

6️⃣ **Scaling:** Docker Compose (docker-compose.yml) or Kubernetes for orchestration.

**🔹 Docker in CI/CD & Kubernetes**

* **CI/CD Pipelines:** Jenkins builds Docker images → Pushes to AWS ECR → Deploys to K8s.
* **Kubernetes Integration:** K8s pulls images from registry → Deploys as pods.
* **Security Practices:** Image scanning, least privilege execution, secrets management.

**🔹 Interview-Proven Answer (Docker Use Case)**

*"In my last project, I used Docker to containerize microservices, ensuring consistency across environments. I created a Dockerfile to package applications, built images using Jenkins, and stored them in AWS ECR. We deployed these containers to Kubernetes (EKS), where services communicated via Kubernetes networking. For persistent storage, I used Docker Volumes, ensuring stateful applications maintained data integrity. This setup enabled seamless CI/CD, scalability, and efficient resource utilization."*

**Kubernetes Architecture - Interview Answer**

A screenshot of a chart

AI-generated content may be incorrect.

**Refer:** <https://yuminlee2.medium.com/kubernetes-understanding-kubernetes-architecture-through-a-restaurant-chefs-analogy-b89f38d8b95a>

Kubernetes follows a **Master-Worker Node Architecture**, managing containerized applications at scale.

**🔹 1️⃣️⃣ Kubernetes Control Plane (Master Node)**

Manages the cluster and schedules workloads. Key components:

* **API Server** (kube-apiserver): Central communication hub (REST API).
* **Scheduler** (kube-scheduler): Assigns Pods to Nodes.
* **Controller Manager** (kube-controller-manager): Ensures desired state (scaling, rolling updates).
* **Etcd**: Stores cluster state (high availability DB).

**🔹 2️⃣️⃣ Kubernetes Worker Nodes**

Each node runs containers and is managed by the Control Plane.

Key components:

* **Kubelet**: Ensures container health, reports to API Server.
* **Container Runtime**: Docker/Containerd to run containers.
* **Kube Proxy**: Handles networking between pods & services.

**🔹 3️⃣️⃣ Kubernetes Objects & Resources**

* **Pods**: Smallest deployable unit, encapsulates containers.
* **Deployments**: Manages scaling & rolling updates.
* **Services**: Exposes applications internally (ClusterIP) or externally (LoadBalancer).
* **Ingress Controller**: Manages external access via domain names.

🔹 **How Kubernetes Manages Deployment?**

1️⃣ Jenkins triggers Kubernetes deployment.

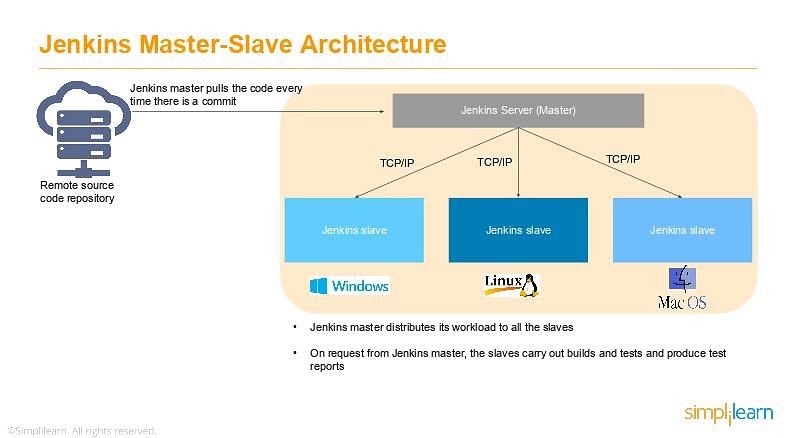
2️⃣ K8s Scheduler assigns Pods to Worker Nodes.

3️⃣ **Kubelet starts Containers** using Docker/Containerd.

4️⃣ Service & Ingress expose apps externally.

5️⃣ Auto-scaling & self-healing maintain uptime.

**Jenkins Architecture - Interview Answer**



Jenkins follows a **master-agent architecture** for scalable, distributed CI/CD execution.

**🔹 Components of Jenkins Architecture**

1️⃣ **Jenkins Master (Controller)**

* The **brain** of Jenkins, responsible for UI, job scheduling, and plugin management.
* Stores configurations and triggers job executions.
* Can distribute jobs to multiple agent nodes.

2️⃣ **Jenkins Agent (Worker Node)**

* Executes build tasks assigned by the master.
* Runs on different environments (Linux, Windows, Docker, Kubernetes).
* Connects via SSH, JNLP, or Kubernetes cloud plugins.

3️⃣ **Jenkins Pipeline**

* A scripted or declarative **CI/CD pipeline** defined in Jenkinsfile.
* Automates build, test, and deployment stages.

4️⃣ **Plugins Ecosystem**

* Supports **Git, Docker, Kubernetes, SonarQube, AWS, Terraform**.
* Extends Jenkins functionality based on project needs.

**🔹 Workflow of a Jenkins Pipeline**

1️⃣ **Developer commits code** → Triggers Jenkins job via Webhook.

2️⃣ **Jenkins pulls code from Git** → Initiates Build Stage.

3️⃣ **Unit & Integration Tests** → Test Stage.

4️⃣ **Build & Push Docker Image** → Deployment Stage.

5️⃣ **Deploy to Kubernetes / AWS** (if configured).

🔹 **Jenkins in Kubernetes CI/CD Pipeline**

* Uses **Jenkins Kubernetes Plugin** to spin up agents dynamically.
* Builds and deploys microservices into **AWS EKS / GKE**.
* Uses **Helm charts** for Kubernetes deployments.
* Implements **Blue-Green & Canary Deployments**.

**Interview-Proven Answer (Jenkins + Kubernetes)**

*"In my last project, I used Jenkins as part of a Kubernetes-based CI/CD pipeline. Jenkins was running inside Kubernetes as a pod, with dynamically provisioned agent nodes. The pipeline used a Jenkinsfile to build Docker images, push them to AWS ECR, and deploy them to Kubernetes using Helm charts. Kubernetes managed deployments via rolling updates, and the Ingress controller handled external traffic. This setup enabled a scalable, resilient, and automated CI/CD workflow with minimal downtime."*