**Containerization with Docker**

**1. Docker Fundamentals**

**What is Docker?**

Docker is an **open-source containerization platform** that allows developers to package applications with their dependencies into portable, lightweight containers.

* **Container** = Application + Dependencies + Configuration → Runs consistently across environments.
* Unlike VMs, containers **share the host OS kernel**, making them lightweight and faster.

**Why Docker?**

* Eliminates “works on my machine” issues.
* Faster startup compared to virtual machines.
* Portable across development, testing, and production.
* Efficient resource utilization.

**Key Docker Concepts**

* **Image** → Read-only template to create containers.
* **Container** → Running instance of an image.
* **Dockerfile** → Script defining how to build an image.
* **Docker Engine** → Core runtime that manages containers.
* **Volumes** → Persistent storage for containers.
* **Networks** → Communication between containers.

**2. Docker Architecture**

Docker follows a **client-server architecture**.

**Components**

1. **Docker Client**
   * CLI (docker run, docker build, etc.) or Docker Desktop GUI.
   * Sends requests to the Docker daemon.
2. **Docker Daemon (dockerd)**
   * Runs on the host machine.
   * Builds, runs, and manages containers/images.
3. **Docker Images**
   * Stored in local cache or remote registries.
   * Built using Dockerfile.
4. **Docker Containers**
   * Isolated runtime environments created from images.
5. **Docker Registries**
   * Store and distribute images (e.g., Docker Hub, AWS ECR, GCP Artifact Registry).

**3. Creating and Managing Containers**

**Basic Workflow**

1. **Pull an image**

docker pull nginx

**Run a container**

docker run -d -p 8080:80 nginx

* + -d → detached mode
  + -p → maps container port 80 to host port 8080

1. **List running containers**

docker ps

1. **Stop and remove container**

docker stop <container\_id>

docker rm <container\_id>

1. **Build custom image (using Dockerfile)**  
   Example Dockerfile:

FROM python:3.9-slim

COPY app.py /app/

WORKDIR /app

RUN pip install flask

CMD ["python", "app.py"]

Build and run:

docker build -t flask-app .

docker run -p 5000:5000 flask-app

**Container Management Commands**

* docker ps -a → List all containers
* docker exec -it <container\_id> bash → Enter container shell
* docker logs <container\_id> → View logs
* docker network ls → List networks
* docker volume ls → List volumes

**4. Container Registries**

Container registries store and distribute images.

**Types of Registries**

1. **Public Registries**
   * Docker Hub (default registry).
   * Accessible to everyone.
2. **Private Registries**
   * Hosted by organizations for internal use.
   * Examples: AWS ECR, Azure Container Registry (ACR), Google Artifact Registry, Harbor.

**Registry Workflow**

1. **Tag an image**

docker tag flask-app myrepo/flask-app:v1

1. **Login to registry**

docker login myrepo

1. **Push to registry**

docker push myrepo/flask-app:v1

1. **Pull from registry**

docker pull myrepo/flask-app:v1

**Table: Popular Container Registries**

| **Registry** | **Type** | **Highlights** |
| --- | --- | --- |
| Docker Hub | Public | Largest open-source image library |
| AWS ECR | Private | Integrated with AWS IAM & ECS/EKS |
| Azure ACR | Private | Azure-native registry, RBAC support |
| GCP Artifact Reg | Private | GCP-native, multi-format support |
| Harbor | Private | Open-source, enterprise-grade features |

**5. Example Use Case: Microservices Application**

* **Scenario:** A company wants to deploy a microservices-based e-commerce platform.
* **Solution with Docker:**
  + Each service (cart, payment, user, catalog) is containerized with its own Dockerfile.
  + Services communicate via Docker networks.
  + Images stored in AWS ECR for security.
  + CI/CD pipeline pulls latest images and deploys them to Kubernetes.
* **Outcome:** Faster deployment, easy scaling, and consistent environments across dev, test, and prod.

**6. Summary**

* Docker enables **lightweight, portable, and consistent** application environments.
* Its architecture follows a **client-daemon-registry** model.
* Containers are easy to **build, run, and manage** with simple commands.
* Registries (public or private) allow secure image distribution.
* Essential for **DevOps, microservices, and cloud-native development**.