Docker Fundamentals - Comprehensive Notes

1. Introduction to Docker

What is Docker?

- **Docker**: A containerization platform that packages applications and their dependencies into lightweight, portable containers
- **Container**: A standardized unit of software that packages code, runtime, system tools, libraries, and settings
- **Key Benefits**:
  - Consistency across environments (dev, test, prod)
  - Lightweight compared to virtual machines
  - Fast startup and deployment
  - Scalability and orchestration
  - Version control for applications

Docker vs Virtual Machines

Docker Containers	Virtual Machines
Share host OS kernel	Each VM has full OS
Lightweight (MBs)	Heavy (GBs)
Fast startup (seconds)	Slow startup (minutes)
Less resource usage	More resource usage
Process-level isolation	Hardware-level isolation

Docker Architecture

```
┌─────────────────────────────────────────────────┐
│           Docker Client                         │
│         (docker commands)                       │
└───────────────────────┬─────────────────────────┘
                        │ Docker API
┌───────────────────────▼─────────────────────────┐
│           Docker Daemon                         │
│  ┌────────────┐ ┌────────────┐ ┌────────────┐   │
│  │ Containers │ │   Images   │ │  Networks  │   │
│  └────────────┘ └────────────┘ └────────────┘   │
│  ┌────────────┐ ┌────────────┐ ┌────────────┐   │
│  │  Volumes   │ │  Plugins   │ │   Swarm    │   │
│  └────────────┘ └────────────┘ └────────────┘   │
└─────────────────────────────────────────────────┘
```

2. Core Docker Concepts

Images

- **Definition**: Read-only template used to create containers
- **Layered Architecture**: Built in layers using Union File System
- **Base Image**: Starting point (e.g., ubuntu, alpine, python)
- **Image Registry**: Repository for storing and sharing images (Docker Hub, ECR, etc.)

Containers

- **Definition**: Running instance of an image
- **Lifecycle**: Created → Running → Stopped → Removed
- **Isolation**: Process, network, and file system isolation
- **Stateless**: Containers should be stateless; data persisted via volumes

Dockerfile

- **Definition**: Text file containing instructions to build an image
- **Declarative**: Describes the desired state of the image
- **Caching**: Docker caches layers to speed up builds
- **Best Practices**: Minimize layers, use specific tags, clean up in same layer

Docker Registry

- **Docker Hub**: Default public registry
- **Private Registries**: ECR, GCR, Harbor, etc.
- **Image Naming**: `registry/namespace/repository:tag`

3. Docker Installation and Setup

Installation Steps (Ubuntu/Debian)

```bash
# Remove old versions
sudo apt remove docker docker-engine docker.io containerd runc

# Update package index
sudo apt update

# Install packages for HTTPS repository
sudo apt install ca-certificates curl gnupg lsb-release

# Add Docker GPG key
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive

# Add Docker repository
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://dowr

# Install Docker Engine
sudo apt update
sudo apt install docker-ce docker-ce-cli containerd.io docker-compose-plugin

# Start and enable Docker
sudo systemctl start docker
sudo systemctl enable docker

# Add user to docker group (optional)
sudo usermod -aG docker $USER
```

Verify Installation

```bash
# Check Docker version
docker --version

# Run hello-world container
docker run hello-world

# Check Docker info
docker info
```

4. Essential Docker Commands

Image Management

```bash
# Search for images
docker search ubuntu

# Pull image from registry
docker pull ubuntu:20.04
docker pull nginx:latest

# List local images
docker images
docker images -a        # Show all images including intermediates

# Remove images
docker rmi image_name:tag
docker rmi image_id
docker image prune       # Remove unused images
docker image prune -a    # Remove all unused images

# Build image from Dockerfile
docker build -t my-app:1.0 .
docker build -t my-app:1.0 -f Dockerfile.prod .

# Tag image
docker tag source_image:tag target_image:tag

# Push image to registry
docker push my-app:1.0

# Image history
docker history image_name

# Inspect image
docker inspect image_name
```

Container Management

```bash
```

```
# Run container
docker run ubuntu:20.04            # Run and exit
docker run -it ubuntu:20.04         # Interactive mode
docker run -d nginx:latest          # Detached mode (background)
docker run --name my-container nginx  # Named container
docker run -p 8080:80 nginx         # Port mapping
docker run -v /host/path:/container/path ubuntu  # Volume mount

# List containers
docker ps               # Running containers
docker ps -a            # All containers
docker ps -q            # Only container IDs

# Start/Stop containers
docker start container_name
docker stop container_name
docker restart container_name
docker pause container_name
docker unpause container_name

# Remove containers
docker rm container_name
docker rm -f container_name      # Force remove running container
docker container prune         # Remove all stopped containers

# Execute commands in running container
docker exec -it container_name /bin/bash
docker exec container_name ls -la
docker exec -u root container_name /bin/bash

# View container logs
docker logs container_name
docker logs -f container_name   # Follow logs
docker logs --tail 100 container_name

# Inspect container
docker inspect container_name

# Container statistics
docker stats
docker stats container_name

# Copy files to/from container
docker cp file.txt container_name:/path/to/destination
docker cp container_name:/path/to/file.txt ./local/path
```

Container Lifecycle Commands

```bash
# Create container without starting
docker create --name my-container nginx

# Attach to running container
docker attach container_name

# Export container as tar
docker export container_name > container.tar

# Import container from tar
docker import container.tar new_image:tag

# Commit container changes to new image
docker commit container_name new_image:tag
```

5. Dockerfile Deep Dive

Basic Dockerfile Structure

```dockerfile
```

```dockerfile
# Use official base image
FROM ubuntu:20.04

# Set maintainer info
LABEL maintainer="your-email@example.com"
LABEL version="1.0"
LABEL description="Sample application"

# Set working directory
WORKDIR /app

# Copy files from host to container
COPY ./app /app
COPY requirements.txt .

# Install dependencies
RUN apt-get update && \
    apt-get install -y python3 python3-pip && \
    pip3 install -r requirements.txt && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*

# Set environment variables
ENV APP_ENV=production
ENV PORT=8000

# Expose port
EXPOSE 8000

# Create non-root user
RUN useradd -m -s /bin/bash appuser
USER appuser

# Define volume
VOLUME ["/app/data"]

# Define entry point
ENTRYPOINT ["python3"]
CMD ["app.py"]
```

Dockerfile Instructions

```
dockerfile
```

```dockerfile
# FROM - Base image
FROM python:3.9-slim

# LABEL - Metadata
LABEL version="1.0" maintainer="dev@company.com"

# WORKDIR - Set working directory
WORKDIR /usr/src/app

# COPY - Copy files/directories
COPY . .                    # Copy everything
COPY src/ /app/src/         # Copy specific directory

# ADD - Copy and extract archives
ADD app.tar.gz /app/

# RUN - Execute commands during build
RUN pip install --no-cache-dir -r requirements.txt

# ENV - Set environment variables
ENV PYTHONPATH=/usr/src/app
ENV DEBUG=False

# EXPOSE - Document port usage
EXPOSE 8000

# USER - Set user context
USER nobody

# VOLUME - Create mount point
VOLUME ["/data", "/logs"]

# ENTRYPOINT - Configure container executable
ENTRYPOINT ["python"]

# CMD - Default command/arguments
CMD ["app.py"]

# ARG - Build-time variables
ARG BUILD_DATE
ARG VERSION=1.0
```

Multi-stage Dockerfile Example

dockerfile

```
# Build stage
FROM node:16-alpine AS builder
WORKDIR /app
COPY package*.json ./
RUN npm ci --only=production

# Production stage
FROM node:16-alpine AS production
WORKDIR /app
COPY --from=builder /app/node_modules ./node_modules
COPY . .
EXPOSE 3000
CMD ["node", "server.js"]
```

6. Docker Networking

Network Types

```bash
# List networks
docker network ls

# Default networks:
# bridge    - Default network for containers
# host      - Use host networking
# none      - No networking
```

Network Management

```bash
```

```
# Create custom network
docker network create my-network
docker network create --driver bridge my-bridge-network

# Run container with custom network
docker run -d --name web --network my-network nginx

# Connect container to network
docker network connect my-network container_name

# Disconnect from network
docker network disconnect my-network container_name

# Inspect network
docker network inspect my-network

# Remove network
docker network rm my-network
```

Port Mapping

```bash
# Map container port to host port
docker run -p 8080:80 nginx        # Host:Container
docker run -p 127.0.0.1:8080:80 nginx  # Bind to specific interface
docker run -P nginx               # Map all exposed ports randomly
```

7. Docker Volumes

Volume Types

1. **Named Volumes**: Managed by Docker
2. **Bind Mounts**: Direct host directory mapping
3. **tmpfs Mounts**: Temporary in-memory storage

Volume Management

```bash
```

```bash
# Create named volume
docker volume create my-volume

# List volumes
docker volume ls

# Inspect volume
docker volume inspect my-volume

# Remove volume
docker volume rm my-volume
docker volume prune        # Remove unused volumes

# Use named volume
docker run -v my-volume:/app/data nginx

# Use bind mount
docker run -v /host/path:/container/path nginx
docker run -v $(pwd):/app nginx    # Current directory

# Use tmpfs mount
docker run --tmpfs /tmp nginx
```

Volume Examples

bash

```bash
# Database with persistent storage
docker run -d \
  --name postgres-db \
  -v postgres-data:/var/lib/postgresql/data \
  -e POSTGRES_PASSWORD=mypassword \
  postgres:13

# Development environment with code binding
docker run -d \
  --name dev-container \
  -v $(pwd)/src:/app/src \
  -v node_modules:/app/node_modules \
  -p 3000:3000 \
  node:16-alpine
```

8. Docker Compose

What is Docker Compose?

- **Definition**: Tool for defining and running multi-container Docker applications
- **YAML Configuration**: Use `docker-compose.yml` file
- **Service Management**: Manage multiple services as a single application
- **Networking**: Automatic network creation for services

Basic docker-compose.yml

```
yaml
```

```
version: '3.8'

services:
  web:
    build: .
    ports:
      - "8000:8000"
    depends_on:
      - db
      - redis
    environment:
      - DATABASE_URL=postgresql://user:pass@db:5432/mydb
    volumes:
      - ./app:/app
      - static_volume:/app/static

  db:
    image: postgres:13
    environment:
      POSTGRES_DB: mydb
      POSTGRES_USER: user
      POSTGRES_PASSWORD: pass
    volumes:
      - postgres_data:/var/lib/postgresql/data

  redis:
    image: redis:6-alpine
    ports:
      - "6379:6379"

volumes:
  postgres_data:
  static_volume:

networks:
  default:
    driver: bridge
```

Docker Compose Commands

```
bash
```

```
# Start services
docker-compose up
docker-compose up -d          # Detached mode
docker-compose up --build       # Rebuild images

# Stop services
docker-compose down
docker-compose down -v          # Remove volumes
docker-compose down --rmi all     # Remove images

# View services
docker-compose ps
docker-compose logs
docker-compose logs -f web       # Follow logs for specific service

# Execute commands
docker-compose exec web bash
docker-compose exec db psql -U user -d mydb

# Scale services
docker-compose up -d --scale web=3

# Build images
docker-compose build
docker-compose build --no-cache

# Pull images
docker-compose pull
```

9. Docker Best Practices

Dockerfile Best Practices

```
dockerfile
```

```
# Use specific tags, not 'latest'
FROM python:3.9-slim

# Use multi-stage builds
FROM node:16-alpine AS builder
# ... build steps ...
FROM node:16-alpine AS production
COPY --from=builder /app/dist ./dist

# Minimize layers
RUN apt-get update && \
    apt-get install -y package1 package2 && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*

# Don't run as root
RUN useradd -m -s /bin/bash appuser
USER appuser

# Use .dockerignore
# Create .dockerignore file to exclude unnecessary files
```

Security Best Practices

```dockerfile
# Use official base images
FROM python:3.9-slim

# Don't store secrets in images
# Use environment variables or secrets management

# Use non-root user
USER 1000:1000

# Use specific versions
FROM python:3.9.10-slim

# Scan images for vulnerabilities
# docker scan image_name
```

Performance Best Practices

```dockerfile
```

```
# Order layers by change frequency
FROM python:3.9-slim
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .

# Use .dockerignore
# Exclude node_modules, .git, etc.

# Use multi-stage builds for smaller images
FROM node:16-alpine AS builder
# ... build steps ...
FROM nginx:alpine
COPY --from=builder /app/dist /usr/share/nginx/html
```

10. Container Orchestration Basics

Docker Swarm

```bash
# Initialize swarm
docker swarm init

# Join swarm as worker
docker swarm join --token <token> <manager-ip>:2377

# Deploy stack
docker stack deploy -c docker-compose.yml my-stack

# List services
docker service ls

# Scale service
docker service scale my-stack_web=3

# Remove stack
docker stack rm my-stack
```

Kubernetes Concepts

- **Pods**: Smallest deployable units
- **Services**: Network access to pods
- **Deployments**: Manage pod replicas
- **ConfigMaps/Secrets**: Configuration management

- **Ingress**: External access to services

11. Docker Registry and Image Management

Docker Hub

```bash
# Login to Docker Hub
docker login

# Tag and push image
docker tag my-app:latest username/my-app:latest
docker push username/my-app:latest

# Pull specific version
docker pull username/my-app:1.0
```

Private Registry

```bash
# Run local registry
docker run -d -p 5000:5000 --name registry registry:2

# Tag for private registry
docker tag my-app:latest localhost:5000/my-app:latest

# Push to private registry
docker push localhost:5000/my-app:latest
```

12. Monitoring and Logging

Container Monitoring

```bash
```

```
# View container resource usage
docker stats

# View container processes
docker top container_name

# Monitor container events
docker events

# Export container metrics
docker stats --format "table {{.Name}}\t{{.CPUPerc}}\t{{.MemUsage}}"
```

Logging

```bash
# View logs
docker logs container_name

# Follow logs
docker logs -f container_name

# Logs with timestamps
docker logs -t container_name

# Limit log output
docker logs --tail 50 container_name
docker logs --since 2h container_name
```

13. Troubleshooting Common Issues

Container Won't Start

```bash
```

```
# Check container logs
docker logs container_name

# Check container configuration
docker inspect container_name

# Run container interactively
docker run -it image_name /bin/bash

# Check if port is already in use
netstat -tulpn | grep :8080
```

Image Build Issues

```bash
# Build with no cache
docker build --no-cache -t my-app .

# Build with verbose output
docker build --progress=plain -t my-app .

# Check build context
docker build -t my-app --file Dockerfile.debug .
```

Storage Issues

```bash
# Check disk usage
docker system df

# Clean up unused resources
docker system prune
docker system prune -a    # Remove all unused images

# Remove specific resources
docker container prune
docker image prune
docker volume prune
docker network prune
```

14. Common Docker Commands Cheat Sheet

Quick Reference

```bash
# Images
docker images              # List images
docker pull image:tag         # Pull image
docker rmi image:tag          # Remove image
docker build -t name:tag .     # Build image

# Containers
docker ps                  # List running containers
docker ps -a                 # List all containers
docker run -d image:tag        # Run container detached
docker stop container_name     # Stop container
docker rm container_name       # Remove container
docker exec -it container bash  # Execute command in container

# System
docker info                 # Docker system info
docker version               # Docker version
docker system prune          # Clean up unused resources

# Logs and monitoring
docker logs container_name     # View logs
docker stats                # View resource usage
docker top container_name      # View processes
```

15. Interview Questions and Answers

Basic Questions

1. **What is Docker and why use it?**
   - Containerization platform for packaging applications
   - Benefits: Consistency, portability, scalability, isolation
2. **Difference between container and image?**
   - Image: Read-only template
   - Container: Running instance of an image
3. **What is Dockerfile?**
   - Text file with instructions to build Docker image
   - Contains commands like FROM, RUN, COPY, CMD

Intermediate Questions

1. **Explain Docker architecture**
   - Client-server architecture

   - Docker daemon manages containers, images, networks
   - Docker client communicates via REST API

2. **What are Docker volumes?**
   - Persistent storage mechanism
   - Types: Named volumes, bind mounts, tmpfs mounts

3. **How does Docker networking work?**
   - Bridge network by default
   - Custom networks for isolation
   - Port mapping for external access

Advanced Questions

1. **Multi-stage Docker builds**
   - Multiple FROM instructions in Dockerfile
   - Reduces final image size
   - Separates build and runtime environments

2. **Docker security best practices**
   - Use official base images
   - Don't run as root
   - Scan images for vulnerabilities
   - Use secrets management

3. **Container orchestration**
   - Docker Swarm for clustering
   - Kubernetes for advanced orchestration
   - Service discovery and load balancing

16. Practical Examples

Python Web Application

```dockerfile
```

```
FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

RUN useradd -m -s /bin/bash appuser
USER appuser

EXPOSE 8000

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

Node.js Application with Docker Compose

```yaml
version: '3.8'

services:
  web:
    build: .
    ports:
      - "3000:3000"
    environment:
      - NODE_ENV=production
      - DATABASE_URL=mongodb://mongo:27017/myapp
    depends_on:
      - mongo
      - redis

  mongo:
    image: mongo:4.4
    volumes:
      - mongodb_data:/data/db

  redis:
    image: redis:6-alpine

volumes:
  mongodb_data:
```

NGINX Reverse Proxy

```dockerfile
FROM nginx:alpine

COPY nginx.conf /etc/nginx/nginx.conf
COPY ./dist /usr/share/nginx/html

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

Summary

Docker is a powerful containerization platform that revolutionizes application deployment and management. Key concepts to master include:

- **Containerization**: Understanding the difference between containers and VMs
- **Images and Containers**: Building, running, and managing Docker images and containers
- **Dockerfile**: Writing efficient and secure Dockerfiles
- **Networking**: Container networking and port mapping
- **Storage**: Volumes and data persistence
- **Orchestration**: Docker Compose for multi-container applications
- **Best Practices**: Security, performance, and operational considerations

Practice these concepts hands-on to build confidence and prepare for technical interviews. Focus on understanding the underlying principles rather than just memorizing commands.