For your Docker-focused DevOps interview, you should concentrate on key aspects like containerization, orchestration, Dockerfile syntax, Docker Compose, and CI/CD pipeline automation. Here's a breakdown of the topics and commands, categorized by scenario:

**1. Container Lifecycle Management**

These are basic operations you will perform daily:

* **Pull an image from Docker Hub**  
  docker pull <image\_name>:<tag>  
  Example:  
  docker pull nginx:latest
* **List Docker images**  
  docker images
* **Run a container**  
  docker run -d --name <container\_name> -p <host\_port>:<container\_port> <image\_name>  
  Example:  
  docker run -d --name webapp -p 8080:80 nginx

Note: This will create a container named as webapp with the port 80 and maps it with the host with port 8080 so when I need to access this container from the internet I need to give <https://host-ip:8080>, caz the container listens traffic from the 8080 port

Eg if docker run -d --name webapp -p 8081:90 nginx creates another container 2 then we have to give give <https://host-ip:8081> to access the container2

* **Stop, start, and restart a container**  
  docker stop <container\_name>  
  docker start <container\_name>  
  docker restart <container\_name>
* **Remove a container**  
  docker rm <container\_name>
* **Remove an image**  
  docker rmi <image\_name>:<tag>

**2. Building and Managing Docker Images**

* **Build an image from a Dockerfile**  
  docker build -t <image\_name>:<tag> .  
  Example:  
  docker build -t myapp:v1 .
* **Tag an image**  
  docker tag <source\_image>:<tag> <repository>/<image\_name>:<tag>  
  Example:  
  docker tag myapp:v1 myrepo/myapp:v1
* **Push an image to a repository**  
  docker push <repository>/<image\_name>:<tag>  
  Example:  
  docker push myrepo/myapp:v1
* **Check image layers**  
  docker history <image\_name>

**3. Networking in Docker**

* **List network**  
  docker network ls
* **Create a new bridge network**  
  docker network create <network\_name>
* **Run container in specific network**  
  docker run --network <network\_name> --name <container\_name> <image\_name>
* **Inspect container IP**  
  docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' <container\_name>

**4. Volumes & Data Management**

* **Create a volume**  
  docker volume create <volume\_name>
* **Attach volume to container**  
  docker run -d -v <volume\_name>:/path/in/container <image\_name>  
  Example:  
  docker run -d -v data\_volume:/var/lib/mysql mysql:latest
* **List volumes**  
  docker volume ls
* **Remove a volume**  
  docker volume rm <volume\_name>

**5. Logging & Debugging**

* **View container logs**  
  docker logs <container\_name>  
  Example:  
  docker logs webapp
* **Attach to running container**  
  docker attach <container\_name>
* **Inspect container details**  
  docker inspect <container\_name>

**6. Docker Compose**

For orchestrating multi-container environments, focus on the following commands:

* **Start services using Docker Compose**  
  docker-compose up -d
* **Stop services**  
  docker-compose down
* **View logs for a specific service**  
  docker-compose logs <service\_name>
* **Scale a service**  
  docker-compose up -d --scale <service\_name>=<count>
* **List running services**  
  docker-compose ps

**Dockerfile for Multi-Stage Build**

A multi-stage build helps reduce the final image size by using intermediate stages to compile or build the application.

**Example: Multi-Stage Dockerfile for a Node.js Application**

# Stage 1: Build the application

FROM node:14 AS builder

WORKDIR /app

COPY package.json ./

RUN npm install

COPY . .

RUN npm run build

# Stage 2: Run the application

FROM node:14-alpine

WORKDIR /app

COPY --from=builder /app/dist ./dist

COPY --from=builder /app/node\_modules ./node\_modules

CMD ["node", "dist/index.js"]

# Expose the application port

EXPOSE 3000

**Docker Compose for a Three-Tier Application**

A three-tier application typically includes a web server, an application server, and a database. Here’s an example using Nginx as the web server, Node.js as the application server, and MySQL as the database.

version: '3.8'

services:

web:

image: nginx:latest

ports:

- "80:80"

volumes:

- ./nginx.conf:/etc/nginx/nginx.conf

depends\_on:

- app

app:

build: ./app

ports:

- "3000:3000"

environment:

- DATABASE\_HOST=db

- DATABASE\_USER=root

- DATABASE\_PASSWORD=rootpassword

- DATABASE\_NAME=mydatabase

depends\_on:

- db

db:

image: mysql:5.7

environment:

MYSQL\_ROOT\_PASSWORD: rootpassword

MYSQL\_DATABASE: mydatabase

MYSQL\_USER: user

MYSQL\_PASSWORD: password

ports:

- "3306:3306"

volumes:

- db\_data:/var/lib/mysql

volumes:

db\_data:

**Explanation**

**Multi-Stage Dockerfile**

1. **Stage 1: Build the Application**
   * FROM node:14 AS builder: Uses Node.js 14 as the base image for the build stage.
   * WORKDIR /app: Sets the working directory.
   * COPY package.json ./: Copies the package.json file.
   * RUN npm install: Installs dependencies.
   * COPY . .: Copies the rest of the application code.
   * RUN npm run build: Builds the application.
2. **Stage 2: Run the Application**
   * FROM node:14-alpine: Uses a smaller Node.js image for running the application.
   * WORKDIR /app: Sets the working directory.
   * COPY --from=builder /app/dist ./dist: Copies the built application from the previous stage.
   * COPY --from=builder /app/node\_modules ./node\_modules: Copies the installed node modules.
   * CMD ["node", "dist/index.js"]: Sets the command to run the application.
   * EXPOSE 3000: Exposes port 3000 for the application.

**Docker Compose for a Three-Tier Application**

* **web**:
  + Uses the latest Nginx image.
  + Maps port 80 on the host to port 80 in the container.
  + Mounts a custom Nginx configuration file.
  + Depends on the app service.
* **app**:
  + Builds the application using the Dockerfile in the ./app directory.
  + Maps port 3000 on the host to port 3000 in the container.
  + Sets environment variables for database connection.
  + Depends on the db service.
* **db**:
  + Uses the MySQL 5.7 image.
  + Sets environment variables for MySQL root password, database name, user, and password.
  + Maps port 3306 on the host to port 3306 in the container.
  + Mounts a volume for persistent storage.
* **volumes**:
  + Defines a named volume db\_data for MySQL data.