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.