DevOps Interview Guide: Docker, Ansible, and More

July 2025

Introduction

This guide provides a comprehensive set of approximately 30 interview questions and detailed answers for each of the key DevOps tools: Docker and Ansible. It is designed to help candidates prepare for technical interviews by covering fundamental concepts, practical applications, and common scenarios. The questions range from beginner to advanced levels to cater to various expertise levels.

1 Docker

1.1 What is Docker, and how does it differ from a virtual machine?

Answer: Docker is a containerization platform that packages applications and their dependencies into lightweight, portable containers. Unlike virtual machines (VMs), which include a full OS and require a hypervisor, Docker containers share the host OS kernel, making them more lightweight, faster to start, and less resource-intensive. Containers isolate processes but use fewer resources compared to VMs, which emulate hardware.

1.2 What are the main components of Docker?

Answer: Key Docker components include:

- **Docker Engine**: The runtime that builds and runs containers.
- **Docker Images**: Read-only templates for creating containers.
- **Docker Containers**: Runnable instances of images.
- **Docker Registry**: Stores and distributes Docker images (e.g., Docker Hub).
- **Dockerfile**: A script with instructions to build an image.

1.3 What is a Dockerfile, and what are its common instructions?

Answer: A Dockerfile is a script containing instructions to build a Docker image. Common instructions include:

- `FROM`: Specifies the base image.
- `RUN`: Executes commands during image building.
- `COPY` or `ADD`: Copies files into the image.
- `CMD`: Specifies the default command to run in the container.
- `EXPOSE`: Indicates ports the container listens on.

Example:

```
FROM ubuntu:20.04
RUN apt-get update && apt-get install -y python3
COPY app.py /app/
CMD ["python3", "/app/app.py"]
```

1.4 How do you create a Docker image?

Answer: To create a Docker image:

1. Write a Dockerfile with required instructions.
2. Run `docker build -t image-name:tag .` in the directory containing the Dockerfile.
3. The image is built and stored locally or can be pushed to a registry using `docker push`.

1.5 What is Docker Compose, and how is it used?

Answer: Docker Compose is a tool for defining and running multi-container applications using YAML files. It configures services, networks, and volumes. Usage involves:

- Creating a `docker-compose.yml` file.
- Running `docker-compose up` to start the application.

Example:

```
version: '3'
services:
  web:
    image: nginx:latest
    ports:
      - "80:80"
  db:
    image: mysql:5.7
    environment:
      MYSQL_ROOT_PASSWORD: example
```

1.6 What is the difference between `CMD` and `ENTRYPOINT` in a Dockerfile?

Answer: `CMD` specifies the default command to run when a container starts, but it can be overridden. `ENTRYPOINT` defines a command that is always executed, and arguments passed to `docker run` are appended to it. Using `ENTRYPOINT` in exec form (e.g., `["/bin/bash"]` allows for more consistent behavior.

1.7 How do you remove unused Docker images and containers?

Answer: To clean up:

- Remove stopped containers: `docker container prune`.
- Remove unused images: `docker image prune`.
- Remove all unused resources: `docker system prune`.

1.8 What is a Docker volume, and why is it used?

Answer: A Docker volume is a way to persist data generated by containers. It is used to store data outside the containers filesystem, ensuring data persists even if the container is removed. Example: `docker run -v myvolume:/app/data mysql`.

1.9 What is Docker networking, and what are the default network modes?

Answer: Docker networking enables communication between containers and external systems. Default modes include:

- **Bridge**: Default network for containers on the same host.
- **Host**: Container shares the hosts network stack.
- **None**: No networking.

1.10 How do you run a container in detached mode?

Answer: Use the `-d` flag: `docker run -d nginx`. This runs the container in the background.

1.11 What is the purpose of `docker inspect`?

Answer: `docker inspect` provides detailed information about a container or image in JSON format, including configuration, network settings, and volumes.

1.12 How do you scale a Docker service?

Answer: Using Docker Swarm or Docker Compose, scale a service with `docker-compose scale service=num` or `docker service scale service=num` in Swarm mode.

1.13 What is a multi-stage build in Docker?

Answer: Multi-stage builds use multiple `FROM` statements in a Dockerfile to create smaller, optimized images by separating build and runtime environments. Example:

```
FROM golang:1.16 AS builder
WORKDIR /app
COPY . .
RUN go build -o myapp
FROM alpine:latest
COPY --from=builder /app/myapp /usr/bin/
CMD ["myapp"]
```

1.14 How do you secure a Docker container?

Answer: Security measures include:

- Using minimal base images (e.g., `alpine`).
- Running containers as non-root users.
- Limiting container capabilities with `-cap-drop`.
- Scanning images for vulnerabilities using tools like Trivy.

1.15 What is the difference between `docker stop` and `docker kill`?

Answer: `docker stop` gracefully stops a container by sending a SIGTERM signal, allowing it to shut down cleanly. `docker kill` sends a SIGKILL signal, forcefully terminating the container.

1.16 How do you push a Docker image to a registry?

Answer: Tag the image with `docker tag image-name registry/repo:tag`, then push using `docker push registry/repo:tag`.

1.17 What is Docker Swarm?

Answer: Docker Swarm is Dockers native orchestration tool for managing a cluster of Docker nodes, enabling scalable, distributed application deployment.

1.18 How do you troubleshoot a Docker container that fails to start?

Answer: Steps include:

- Check logs with `docker logs container-id`.
- Inspect configuration with `docker inspect container-id`.
- Verify port conflicts or resource limits.

1.19 What is the purpose of `docker exec`?

Answer: `docker exec` runs a command inside a running container, e.g., `docker exec -it container-id /bin/bash` for an interactive shell.

1.20 How do you limit a containers CPU and memory usage?

Answer: Use flags like `-cpus="0.5"` and `-memory="512m"` in `docker run`.

1.21 What is a Docker registry?

Answer: A Docker registry is a storage and distribution system for Docker images, such as Docker Hub or a private registry.

1.22 How do you create a custom Docker network?

Answer: Use `docker network create my-network` to create a network, then attach containers with `-network my-network`.

1.23 What is the difference between `docker run` and `docker create`?

Answer: `docker create` prepares a container without starting it, while `docker run` creates and starts the container.

1.24 How do you handle secrets in Docker?

Answer: Use Docker secrets in Swarm mode or third-party tools like HashiCorp Vault to securely manage sensitive data.

1.25 What is the purpose of `docker history`?

Answer: `docker history image-name` shows the layers and commands used to build an image.

1.26 How do you monitor Docker containers?

Answer: Use tools like `docker stats`, Prometheus, or cAdvisor to monitor container performance and resource usage.

1.27 What is the difference between a Docker image and a container?

Answer: An image is a read-only template, while a container is a runnable instance of an image with a writable layer.

1.28 How do you update a running Docker container?

Answer: Stop the container, pull the updated image, and recreate the container with `docker run`.

1.29 What is the role of the Docker daemon?

Answer: The Docker daemon (`dockerd`) manages containers, images, networks, and volumes on the host.

1.30 How do you enable communication between containers?

Answer: Place containers in the same Docker network and use container names as hostnames for communication.

2 Ansible

2.1 What is Ansible, and what are its key features?

Answer: Ansible is an open-source automation tool for configuration management, application deployment, and task automation. Key features include:

- **Agentless**: Uses SSH, no client software needed.
- **Idempotent**: Ensures consistent results.
- **YAML Playbooks**: Human-readable automation scripts.
- **Modular**: Extensive module library.

2.2 What is an Ansible Playbook?

Answer: A Playbook is a YAML file defining tasks to execute on target hosts. It includes hosts, tasks, variables, and handlers. Example:

```
---
- hosts: webservers
  tasks:
    - name: Install Nginx
      apt:
        name: nginx
        state: present
```

2.3 How does Ansible handle inventory management?

Answer: Ansible uses an inventory file (e.g., `hosts.ini`) to define target hosts and groups. It can be static or dynamic (from cloud providers). Example:

```
[webservers]
192.168.1.10
192.168.1.11
```

2.4 What is the difference between a role and a playbook?

Answer: A playbook is a standalone YAML file with tasks, while a role is a reusable, organized collection of tasks, templates, and files.

2.5 How do you run an Ansible playbook?

Answer: Use `ansible-playbook playbook.yml` to execute a playbook. Add `-inventory hosts.ini` for custom inventory.

2.6 What are Ansible modules?

Answer: Modules are reusable scripts that perform specific tasks, like `apt` for package management or `file` for file operations.

2.7 How do you handle variables in Ansible?

Answer: Variables are defined in playbooks, inventory, or files (e.g., `vars/main.yml`). Example:

```
---
- hosts: all
  vars:
    package: nginx
  tasks:
    - name: Install package
      apt:
        name: "{{ package }}"
        state: present
```

2.8 What is Ansible Galaxy?

Answer: Ansible Galaxy is a repository for sharing and downloading Ansible roles to simplify automation tasks.

2.9 How do you debug an Ansible playbook?

Answer: Use `-verbose`, `-check` for dry runs, or the `debug` module to print variables.

2.10 What is an Ansible handler?

Answer: A handler is a task triggered by a `notify` action, typically for restarting services. Example:

```
- name: Restart Nginx
  service:
    name: nginx
    state: restarted
  listen: "restart nginx"
```

2.11 How do you secure Ansible communication?

Answer: Use SSH keys, configure `ansible.cfg` with $\texttt{vault}_{p}ass, and encrypt sensitive data with Ansible Vault$.

2.12 What is Ansible Vault?

Answer: Ansible Vault encrypts sensitive data in playbooks or files using ansible-vault encrypt file.yml.

2.13 How do you manage remote hosts with Ansible?

Answer: Define hosts in the inventory file and use SSH keys for authentication. Example: ansible all -m ping.

2.14 What is the difference between ansible and ansible-playbook?

Answer: ansible runs ad-hoc commands, while ansible-playbook executes structured playbooks.

2.15 How do you use Ansible with cloud providers?

Answer: Use dynamic inventory plugins (e.g., $\text{aws}_e c2) to fetch hosts from cloud platforms like AWS.$

2.16 What is idempotency in Ansible?

Answer: Idempotency ensures tasks only make changes if needed, preventing redundant operations.

2.17 How do you handle errors in Ansible?

Answer: Use $\text{ignore}_e rrors : true or \text{failed}_w hen to manage task failures gracefully.$ 2.18 What is the becom