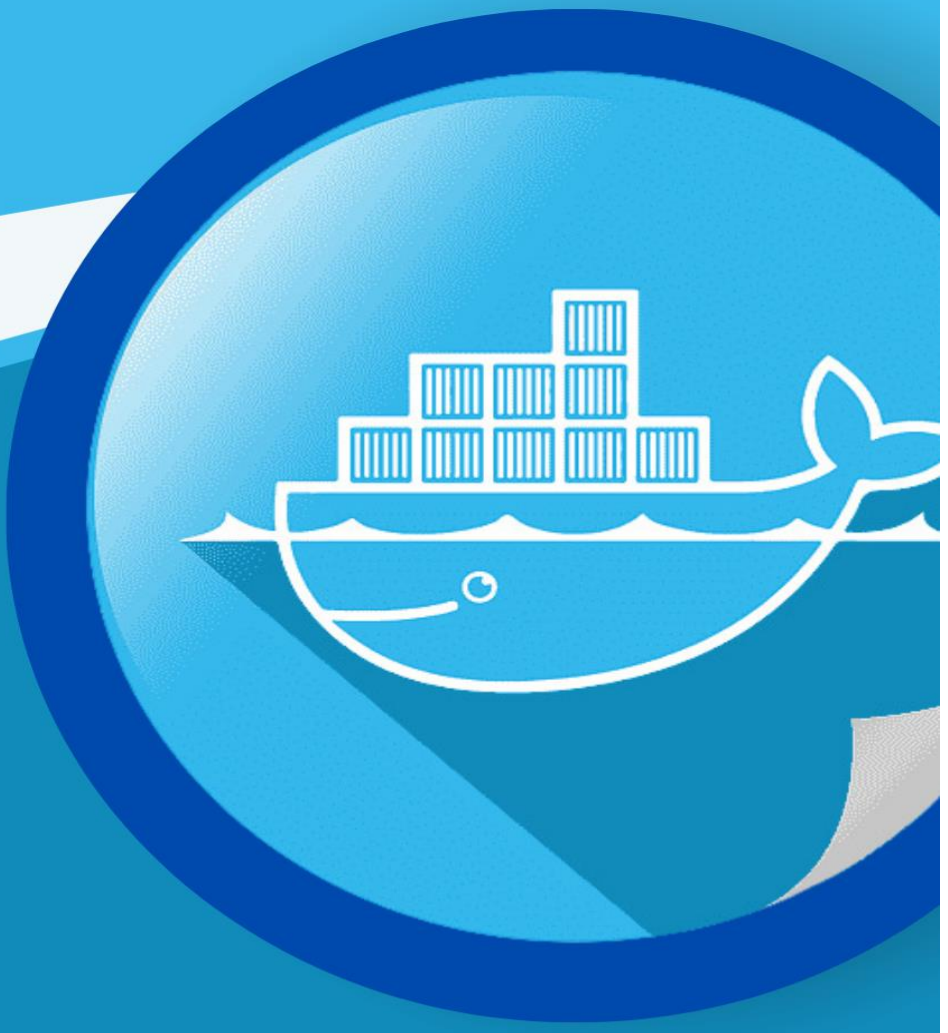
Devops Shack

EBOOK 2025

Docker Interview

Scenario-Based Questions with Solutions



Presented By:
DevOps Shack

www.devopsshack.com

[Click her for DevSecOps & Could DevOps Course]

DevOps Shack

Docker Scenario-Based Questions with Solutions: A Practical Guide

Table of Contents

18. Copying Files from a Running Container
19. Removing Unused Docker Resources
20. Running a Container with a Host File Mapping

Monitoring and Debugging

21. Viewing Logs for All Running Containers
22. Inspecting the IP Address of a Running Container
23. Creating a Custom Docker Network
24. Container Resource Monitoring
25. Handling Container Exit Codes
26. Rebuilding a Docker Image after Code Changes
27. Running a Container with Capabilities
28. Troubleshooting Network Connections in a Container
29. Running a Container with Specific Timezone
30. Building an Image with Custom Build Arguments

Advanced Docker Scenarios

31. Running a Container in Read-Only Mode
32. Building an Image with Cached Layers
33. Exporting and Importing a Docker Image
34. Running a Detached Interactive Container
35. Setting Up Log Rotation for Containers
36. Using Docker to Simulate a Cron Job
37. Running a GPU-Enabled Container
38. Troubleshooting Container Startup Issues
39. Running a Container with a Restart Policy

40. Verifying Docker Version Compatibility

Docker File and Configuration Management

41. Writing a Multi-Stage Dockerfile for Smaller Images
42. Setting Up Docker Secrets for Secure Credentials
43. Restricting Container Access to Specific Networks
44. Setting Up Docker Health Checks
45. Running a Service with Custom DNS
46. Simulating Cron Jobs Using Docker Commands
47. Using docker stats for Real-Time Resource Monitoring
48. Running Containers with Specific Build Arguments
49. Inspecting Metadata of Running Containers
50. Configuring Dockerfile for Timezone and Locale Management

Introduction

Docker has become a cornerstone of modern DevOps practices, enabling developers and organizations to build, ship, and run applications in lightweight, portable containers. While Docker simplifies many aspects of application development and deployment, real-world scenarios often require tailored solutions, advanced configurations, and precise troubleshooting. This guide presents **50 Docker scenario-based questions** to help you deepen your understanding of Docker and apply it effectively in practical situations.

Purpose of the Guide

1. **Practical Learning**: Provides real-world scenarios and challenges commonly faced by Docker users, ensuring hands-on expertise.
2. **Comprehensive Coverage**: Covers a wide range of topics, including multi-container deployments, networking, resource management, debugging, and advanced configurations.
3. **Skill Enhancement**: Prepares developers, DevOps engineers, and system administrators to solve practical problems, optimize Docker usage, and troubleshoot efficiently.

Each scenario includes detailed questions and answers, complete with commands, code examples, and configuration files. Whether you're a beginner looking to expand your Docker knowledge or an experienced professional seeking advanced use cases, this guide will help you gain valuable insights and practical skills.

1. Deploying a Multi-Container Application

You need to deploy a web application with a database using docker-compose. The application runs on port 5000, and the database (PostgreSQL) requires persistent storage.

Question: Write a docker-compose.yml file for this setup. Ensure the web app connects to the database service and includes volume mapping for persistent database storage.

Answer:

```
version: "3.8"
services:
  web:
    image: mywebapp:latest
    ports:
      - "5000:5000"
    depends_on:
      - db
    environment:
      - DATABASE_URL=postgres://user:password@db:5432/mydb

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

volumes:
  db_data:
```

2. Debugging a Failing Container

A container exits immediately after starting. You suspect the entrypoint script is failing.

Question: How would you debug the container to find the root cause?

Answer: Run the container in interactive mode to bypass the entrypoint:

```
docker run -it --entrypoint /bin/bash myimage:latest
```

Check the logs of the container:

```
docker logs <container_id>
```

3. Building an Optimized Docker Image

You are tasked to build a Docker image for a Python Flask app. The image must be small and secure. How would you write the Dockerfile?

Answer:

```
# Use a lightweight base image
FROM python:3.9-slim

# Set working directory
WORKDIR /app

# Copy only requirements for caching
COPY requirements.txt .

# Install dependencies
RUN pip install --no-cache-dir -r requirements.txt

# Copy application code
COPY . .

# Expose the application port
EXPOSE 5000
```

```
# Set default command
CMD ["python", "app.py"]
```

4. Managing Environment Variables in Containers

You want to pass environment variables securely to a container running in production.

Question: How would you achieve this using Docker?

Answer:

- Use the --env flag or .env file for secure environment variable management.
- Example using .env file:
    - **.env file:**

```
DB_USER=user
DB_PASS=secret
```

    - **Run command:**

```
docker run --env-file .env myimage:latest
```

5. Persistent Storage for Containers

You need to ensure that a MySQL container's data is not lost when the container is restarted.

Question: How would you achieve this using volumes?

Answer: Run the container with a named volume:

```
docker run -d --name mysql -e MYSQL_ROOT_PASSWORD=root \
 -v mysql_data:/var/lib/mysql mysql:latest
```

Inspect volumes:

```
docker volume inspect mysql_data
```

6. Dockerizing a Node.js Application

You are tasked to create a Docker image for a Node.js application. How would you write the Dockerfile?

Answer:

```
FROM node:14

WORKDIR /app

COPY package*.json ./

RUN npm install --production

COPY . .

EXPOSE 3000

CMD ["node", "server.js"]
```

7. Running a Container with Limited Resources

You need to run a container for a memory-intensive process, ensuring it doesn't exceed 2GB of memory and 1 CPU core.

Question: Write the command to achieve this.

Answer:

```
docker run -d --name limited_container \
  --memory="2g" --cpus="1.0" myimage:latest
```

8. Debugging Network Issues in a Container

A container cannot access external internet resources. How would you debug and resolve this issue?

Answer:

- Verify network configuration:

```
docker network inspect bridge
```

- Check DNS settings inside the container:

```
docker exec -it <container_id> cat /etc/resolv.conf
```

- Restart Docker to reset the network:

```
sudo systemctl restart docker
```

9. Scaling a Service with Docker Compose

You need to scale a web application service to 5 replicas using Docker Compose.

Question: How would you modify the docker-compose.yml file to achieve this?

Answer:

```
version: "3.8"
services:
  web:
    image: mywebapp:latest
    ports:
      - "5000:5000"
    deploy:
      replicas: 5
      resources:
        limits:
          cpus: "0.5"
```

```
        memory: "512M"
```

10. Running Multiple Containers on Custom Networks

You need to create a custom bridge network and run two containers (web and db) on it.

Question: Write the commands to create the network and run the containers.

Answer: Create the network:

```
docker network create my_custom_network
```

Run the containers:

```
docker run -d --name db --network my_custom_network postgres:13
```

```
docker run -d --name web --network my_custom_network -e DATABASE_URL=postgres://user:password@db:5432/mydb mywebapp:latest
```

11. Multi-Stage Docker Build

You need to create a Docker image for a Go application using multi-stage builds to reduce the image size.

Question: How would you write the Dockerfile?

Answer:

```
# Stage 1: Build
FROM golang:1.18 AS builder
WORKDIR /app
COPY . .
RUN go build -o main .

# Stage 2: Production
FROM alpine:latest
```

```
WORKDIR /root/
COPY --from=builder /app/main .
CMD ["./main"]
```

12. Setting Up Docker Secrets

You want to securely pass sensitive credentials (e.g., database passwords) to a container using Docker secrets.

Question: How would you set up and use Docker secrets?

Answer:

- Create a secret:

```
echo "mysecretpassword" | docker secret create db_password -
```

- Reference the secret in docker-compose.yml:

```
version: "3.8"
services:
  app:
    image: myapp:latest
    secrets:
      - db_password
secrets:
  db_password:
    external: true
```

13. Restricting Container Access to a Specific Network

You want to ensure a container only communicates with other containers on a specific custom network.

Question: How would you set this up?

Answer:

- Create a custom network:

```
docker network create internal_network
```

- Run the container on the custom network:

```
docker run -d --name app --network internal_network myimage:latest
```

14. Viewing Logs in Real-Time

You need to monitor a container's logs in real-time to debug a running application.

Question: How would you view the logs?

Answer: Use the docker logs command with the -f option:

```
docker logs -f <container_id_or_name>
```

15. Restarting Containers Automatically

You want to ensure that a container restarts automatically if it crashes or the Docker daemon restarts.

Question: How would you configure this?

Answer: Run the container with a restart policy:

```
docker run -d --restart always --name myapp myimage:latest
```

In docker-compose.yml:

```
version: "3.8"
services:
  app:
    image: myapp:latest
    restart: always
```

16. Debugging Resource Utilization in a Running Container

You need to check the CPU and memory usage of a running container.

Question: How would you do this?

Answer: Use the docker stats command:

```
docker stats <container_id_or_name>
```

17. Running a Container on a Specific Host Port

You want to bind a container's service to a specific host port.

Question: Write the command to run a container on port 8080 of the host.

Answer:

```
docker run -d -p 8080:80 myimage:latest
```

18. Setting Up Custom DNS in a Container

You need to configure a container to use a specific DNS server.

Question: How would you achieve this?

Answer: Run the container with the --dns flag:

```
docker run -d --dns 8.8.8.8 myimage:latest
```

Or configure DNS in daemon.json:

```
{
  "dns": ["8.8.8.8", "8.8.4.4"]
}
```

Restart Docker after updating:

```
sudo systemctl restart docker
```

19. Inspecting Container Metadata

You need to retrieve detailed metadata for a specific container, such as its IP address and mount points.

Question: Which command would you use?

Answer: Use the docker inspect command:

```
docker inspect <container_id_or_name>
```

20. Running a Detached Interactive Container

You want to run a container interactively but still keep it running in detached mode.

Question: How would you accomplish this?

Answer: Run the container with both -d (detached) and -it (interactive terminal):

```
docker run -dit --name mycontainer ubuntu:latest
```

21. Running a Service with Multiple Ports

You need to run a containerized application that listens on two ports, 8080 and 8443.

Question: Write the docker run command to expose both ports on the host.

Answer:

```
docker run -d -p 8080:8080 -p 8443:8443 myimage:latest
```

22. Limiting Disk Space Usage for a Container

You want to limit a container's writable layer to 10GB to prevent it from consuming too much disk space.

Question: How would you configure this?

Answer: Run the container with the --storage-opt flag

```
docker run -d --storage-opt size=10G myimage:latest
```

23. Creating a Private Docker Registry

You need to set up a private Docker registry on your server.

Question: Write the command to create and run a Docker registry container.

Answer:

```
docker run -d -p 5000:5000 --name registry registry:2
```

Push an image to the private registry:

```
docker tag myimage:latest localhost:5000/myimage:latest
```

```
docker push localhost:5000/myimage:latest
```

24. Restarting a Stopped Container

A container named mycontainer has been stopped. You need to restart it without creating a new container.

Question: Write the command to restart the container.

Answer:

```
docker start mycontainer
```

25. Running Containers with Different User Permissions

You need to run a container as a non-root user for security reasons.

Question: How would you achieve this?

Answer: Run the container with the --user flag:

```
docker run -d --user 1001:1001 myimage:latest
```

Alternatively, specify the user in the Dockerfile:

```
USER 1001
```

26. Binding a Container to a Specific Network Interface

You want to bind a container to a specific network interface (e.g., 192.168.1.100) on the host.

Question: Write the docker run command to bind the container to this interface.

Answer:

```
docker run -d -p 192.168.1.100:8080:80 myimage:latest
```

27. Running a Health Check for a Container

You need to ensure a container is healthy by periodically checking if a specific service is running.

Question: How would you define a health check in the Dockerfile?

Answer:

```
HEALTHCHECK --interval=30s --timeout=5s --start-period=10s --retries=3 \
  CMD curl -f http://localhost:8080 || exit 1
```

28. Copying Files from a Running Container

You need to copy a configuration file named config.yaml from a running container named mycontainer to the host.

Question: Write the command to copy the file.

Answer:

```
docker cp mycontainer:/path/to/config.yaml /host/path/config.yaml
```

29. Removing Unused Docker Resources

Your system is running out of disk space. You want to remove unused Docker containers, images, and volumes.

Question: Write the command to clean up unused Docker resources.

Answer:

```
docker system prune -a --volumes
```

30. Running a Container with a Host File Mapping

You want to map a custom hosts file to a container to override DNS resolution.

Question: Write the docker run command to achieve this.

Answer:

```
docker run -d --add-host mycustomhost:192.168.1.100 myimage:latest
```

31. Viewing Logs for All Running Containers

You need to monitor logs for all running containers simultaneously.

Question: How would you view logs for all running containers?

Answer: Use the following command:

```
docker logs -f $(docker ps -q)
```

Or use:

```
docker-compose logs -f
```

32. Inspecting the IP Address of a Running Container

You need to retrieve the IP address of a container named web.

Question: Write the command to get the IP address.

Answer:

```
docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' web
```

33. Creating a Custom Docker Network

You want to create a custom bridge network with a specific subnet and run two containers on it.

Question: Write the commands to achieve this.

Answer: Create the network:

```
docker network create \
 --driver bridge \
 --subnet=192.168.1.0/24 \
  my_custom_network
```

Run containers on the custom network:

```
docker run -d --name db --network my_custom_network postgres:latest
```

```
docker run -d --name web --network my_custom_network mywebapp:latest
```

34. Container Resource Monitoring

You need to monitor real-time CPU and memory usage for all running containers.

Question: Which command would you use?

Answer:

```
docker stats
```

35. Handling Container Exit Codes

You need to determine why a container exited and its exit code.

Question: How would you retrieve this information?

Answer: Check the container's exit code:

```
docker inspect -f '{{.State.ExitCode}}' <container_id_or_name>
```

View detailed exit reason:

```
docker logs <container_id>
```

36. Rebuilding a Docker Image after Code Changes

You've updated the source code for your application and need to rebuild the Docker image.

Question: Write the commands to rebuild the image and run a new container.

Answer: Rebuild the image:

```
docker build -t myimage:latest .
```

Run a new container:

```
docker run -d --name mycontainer myimage:latest
```

37. Running a Container with Capabilities

You need to run a container with additional Linux capabilities, such as NET_ADMIN.

Question: How would you achieve this?

Answer:

```
docker run -d --cap-add=NET_ADMIN myimage:latest
```

38. Troubleshooting Network Connections in a Container

You suspect a container cannot connect to an external service. You need to test the network connection from inside the container.

Question: How would you do this?

Answer: Enter the container and use a tool like curl or ping:

```
docker exec -it <container_id> /bin/bash
```

```
curl http://example.com
```

```
ping google.com
```

39. Running a Container with Specific Timezone

You want to run a container with the Asia/Kolkata timezone configured.

Question: Write the docker run command to set the timezone.

Answer:

```
docker run -d -e TZ=Asia/Kolkata myimage:latest
```

40. Building an Image with Custom Build Arguments

You want to pass custom build arguments (e.g., VERSION=1.0) while building a Docker image.

Question: How would you write the Dockerfile and build command?

Answer: Dockerfile:

```
FROM alpine:latest
ARG VERSION
RUN echo "Version: $VERSION" > /version.txt
CMD cat /version.txt
```

Build Command:

```
docker build --build-arg VERSION=1.0 -t myimage:latest .
```

41. Running a Container in Read-Only Mode

You need to run a container with a read-only file system to enhance security.

Question: Write the docker run command to achieve this.

Answer:

```
docker run -d --read-only --name readonly_container myimage:latest
```

42. Building an Image with Cached Layers

You want to build a Docker image while leveraging cached layers for unchanged steps.

Question: How would you write the build command to utilize the cache?

Answer:

```
docker build -t myimage:latest .
```

Ensure the order of COPY and RUN steps in the Dockerfile minimizes cache invalidation.

43. Exporting and Importing a Docker Image

You need to export an image from one machine and import it to another.

Question: Write the commands to export and import the image.

Answer: Export the image:

```
docker save -o myimage.tar myimage:latest
```

Transfer the .tar file and import it on the new machine:

```
docker load -i myimage.tar
```

44. Running a Detached Interactive Container

You want to keep an interactive shell open in a container but run it in detached mode.

Question: How would you achieve this?

Answer:

```
docker run -dit --name interactive_container ubuntu:latest
```

```
docker attach interactive_container
```

45. Setting Up Log Rotation for Containers

You need to configure log rotation to prevent log files from consuming excessive disk space.

Question: How would you set this up?

Answer: Update the Docker daemon configuration in `/etc/docker/daemon.json`:

```
{
 "log-driver": "json-file",
 "log-opts": {
  "max-size": "10m",
  "max-file": "3"
 }
}
```

Restart Docker:

```
sudo systemctl restart docker
```

46. Using Docker to Simulate a Cron Job

You need to run a script inside a container every minute, similar to a cron job.

Question: Write the docker run command to achieve this.

Answer:

```
docker run -d --name cron_container ubuntu:latest /bin/sh -c "while true; do <your_command>; sleep 60; done"
```

47. Running a GPU-Enabled Container

You need to run a container that utilizes a GPU for a machine learning application.

Question: Write the command to run a GPU-enabled container.

Answer: Ensure NVIDIA drivers are installed and use the NVIDIA runtime:

```
docker run --gpus all -d myimage:latest
```

48. Troubleshooting Container Startup Issues

A container fails to start due to a misconfigured environment variable. You need to identify and fix the issue.

Question: How would you debug this?

Answer:

- Inspect the container logs:

```
docker logs <container_id>
```

- Run the container interactively to test configurations:

```
docker run -it --env-file .env myimage:latest /bin/bash
```

49. Running a Container with a Restart Policy

You want a container to restart up to 3 times if it fails.

Question: Write the docker run command to configure this.

Answer:

```
docker run -d --restart on-failure:3 myimage:latest
```

50. Verifying Docker Version Compatibility

You need to ensure that the Docker client and daemon versions are compatible.

Question: How would you check this?

Answer: Check the client and server versions:

```
docker version
```

Compare the output for the client (Client: Docker Engine) and daemon (Server: Docker Engine) to ensure compatibility.

Conclusion

Docker's versatility and power make it an essential tool for modern application development and deployment. However, to truly harness its capabilities, one must be prepared to tackle real-world scenarios that involve complex setups, resource optimization, and troubleshooting. This guide of **50 scenario-based Docker questions** serves as a comprehensive resource to bridge the gap between theoretical knowledge and practical application.

Key Takeaways

1. **Enhanced Problem-Solving**: Understanding real-world scenarios equips you to efficiently resolve common Docker challenges.
2. **Practical Insights**: Hands-on scenarios covering topics like multi-container deployments, networking, resource management, and advanced configurations prepare you for real DevOps workflows.
3. **Continuous Learning**: Docker is constantly evolving, and applying solutions to practical problems is a great way to stay updated and sharpen your skills.

By mastering these scenarios, you can confidently manage Docker environments, streamline workflows, and contribute to building scalable and resilient applications. Remember, every challenge is an opportunity to learn and improve your expertise in containerization.