Rakshita Belwal

Docker Scenario Based Interview Questions with Expected Answers

Scenario 1: Debugging a Container Failure

Question:
You've deployed a containerized web application, but users are reporting that it's not accessible. How would you debug and resolve the issue?

Expected Answer:

- Check the container status using **`docker ps`** and **`docker logs [container_id]`.**
- Verify if the application inside the container is running by using commands like `docker exec` to access the container.
- Ensure the correct ports are exposed using `docker inspect`.
- Verify network connectivity between the host and the container.
- Check for errors in the application's configuration**.**

Scenario 2: Scaling an Application

Question:
You have a service running in a single container, but traffic has increased significantly. How would you scale this application using Docker?

Expected Answer:

- Use Docker Compose or Swarm to define multiple replicas of the service.
- Update the **`docker-compose.yml`** or use **`docker service scale`** for Docker Swarm.
- Ensure the load balancer is configured to distribute traffic across the replicas.

Scenario 3: Handling Data Persistence

Question:
You need to deploy a database container while ensuring the data remains intact even if the container is removed. How would you achieve this?

Expected Answer:

- Use Docker Volumes to persist data.
- Create a volume using **`docker volume create`**.
- Mount the volume to the database container using the **`-v`** or **`--mount`** flag.
- Ensure backups are taken using external tools or scripts.

Scenario 4: Optimizing Docker Image Size

Question:
Your Docker image size has grown significantly, affecting deployment speed. What steps would you take to reduce the image size?

Expected Answer:

- Use a lightweight base image like **`alpine`**.

- Combine **RUN** commands to reduce the number of layers.
- Remove unnecessary files and cache during the build process.
- Use multistage builds to separate build and runtime dependencies.

Scenario 5: Securing Docker Containers

Question:
Your team wants to ensure that sensitive data like API keys and passwords are securely handled in a containerized application. How would you achieve this using Docker?

Expected Answer:

- Use Docker Secrets to securely store sensitive data.
- Encrypt data at rest and in transit.
- Use tools like Ansible Vault or HashiCorp Vault to manage secrets.
- Avoid including sensitive information in Dockerfiles or environment variables.

Scenario 6: Handling Docker Networking Issues

Question:
Your containers need to communicate with each other, but you're noticing connectivity issues. How would you troubleshoot this?

Expected Answer:

- Verify the network type (bridge, overlay, host, etc.) using **`docker network inspect`.**
- Check container connectivity with **`ping`** or **`curl`** commands.
- Ensure that containers are attached to the same network.
- Confirm firewall rules and host system configurations.

Scenario 7: Migrating an Application

Question:
You need to migrate a monolithic application to Docker containers. How would you approach this migration?

Expected Answer:

- Break down the application into microservices, if possible.
- Create Dockerfiles for each service.
- Use Docker Compose for local development and multi-container setup.
- Gradually containerize and test each component to ensure stability.

Scenario 8: Container Restart Loops

Question:
You've deployed a container, but it keeps restarting. How would you investigate and resolve this?

Expected Answer:

1. Check container logs using **`docker logs [container_id]`** to identify the error.

2. Inspect the **`docker inspect`** command to ensure configurations (like environment variables) are correct.
3. Verify resource limits (e.g., memory) in the Docker Compose or **`docker run`** command.
4. Ensure the application inside the container is correctly configured.
5. Check for issues in the **`CMD`** or **`ENTRYPOINT`** in the Dockerfile.

Scenario 9: Troubleshooting High CPU/Memory Usage

Question:
A container is consuming excessive CPU and memory resources. How do you diagnose and fix the issue?

Expected Answer:

1. Use **`docker stats`** to monitor container resource usage.
2. Inspect the application logs using **`docker logs`** to identify performance bottlenecks.
3. Set resource limits in the Compose file or **`docker run`** command **(`--memory`, `--cpus`).**
4. Optimize the application running inside the container.

Scenario 10: Blue-Green Deployment

Question:
You need to implement a blue-green deployment strategy using Docker containers. What steps would you take?

Expected Answer:

1. Deploy a new version of the application (green) alongside the existing one (blue).
2. Test the green environment to ensure stability.
3. Switch traffic from blue to green by updating the load balancer.
4. Roll back to blue if issues occur.

Scenario 11: Handling a Container Crash During Deployment

Question:
Your application crashes immediately after being deployed in a container. How would you troubleshoot this?

Expected Answer:

1. Check the container's logs for error messages using **`docker logs`.**
2. Use **`docker inspect`** to ensure the configuration (e.g., ports, environment variables) is correct.
3. Debug the container by running it interactively **(`docker run -it` with `bash` or `sh`).**
4. Verify the application's compatibility with the base image or runtime environment.

Scenario 12: Handling Dependency Issues in Containers

Question:
A Python-based containerized app fails due to missing dependencies. How do you resolve this issue?

Expected Answer:

1. Update the **`requirements.txt`** file with all necessary dependencies.
2. Modify the Dockerfile to install the dependencies using **`RUN pip install -r requirements.txt`**.
3. Rebuild the Docker image using **`docker build`**.
4. Test the container locally before deployment.
