Docker Questions and Practical Exercises

This document provides a structured set of Docker-related questions and practical exercises, categorized by difficulty (Basic, Intermediate, Advanced) and topic, with detailed answers and solutions. The content is designed to match the depth and style of typical DSA question sets, covering theoretical concepts and hands-on tasks to prepare for interviews or certifications.

Basic Docker Concepts

Theory Questions

1. **What is Docker?**
    - **Answer**: Docker is an open-source containerization platform that packages applications and their dependencies into containers. Containers are lightweight, portable units that share the host OS kernel, enabling consistent execution across different environments. Unlike virtual machines, Docker containers do not require a full guest OS, making them faster and more efficient. (https://intellipaat.com/blog/interview-question/docker-interview-questions-answers/)
2. **What is a Docker container?**
    - **Answer**: A Docker container is a runnable instance of a Docker image. It encapsulates an application and its dependencies (libraries, runtime, system tools) in an isolated environment, using the host OS kernel. Containers are lightweight and portable, ensuring consistent behavior across development, testing, and production. (https://intellipaat.com/blog/interview-question/docker-interview-questions-answers/)
3. **How does Docker differ from a virtual machine?**
    - **Answer**: Docker containers share the host OS kernel and use resource isolation (namespaces, cgroups), making them lightweight and faster. Virtual machines (VMs) run a full guest OS on a hypervisor, consuming more resources. Containers provide process-level isolation, while VMs offer full OS isolation. (https://www.testgorilla.com/blog/tricky-docker-interview-questions-answers/)
4. **What is a Docker image?**
    - **Answer**: A Docker image is a read-only template used to create containers. It consists of layered instructions (e.g., from a Dockerfile) defining the application, dependencies, and configuration. Images are stored in registries like Docker Hub. (https://medium.com/devopscurry/top-25-advanced-docker-interview-questions-91de77a80d16)
5. **What is a Dockerfile?**

- **Answer**: A Dockerfile is a text file containing instructions to build a Docker image. It includes commands like `FROM` (base image), `RUN` (execute commands), `COPY` (add files), and `CMD` (default command at runtime). (https://www.bitdegree.org/tutorials/docker-interview-questions)

6. **What is the purpose of Docker Hub?**

   - **Answer**: Docker Hub is a cloud-based registry for storing and sharing Docker images. It allows users to pull public or private images and push their own for collaboration and deployment. Alternatives include Amazon ECR and GitHub Container Registry. (https://www.datacamp.com/blog/docker-interview-questions)

7. **What is containerization?**

   - **Answer**: Containerization is a virtualization technique that packages applications and their dependencies into containers. It ensures portability and consistency across environments by isolating processes while sharing the host OS kernel. (https://intellipaat.com/blog/interview-question/docker-interview-questions-answers/)

8. **What is a hypervisor, and how does it relate to Docker?**

   - **Answer**: A hypervisor is software that enables virtualization by running virtual machines with separate OS instances. Docker does not use a hypervisor; it relies on the host OS kernel for containerization, making it more lightweight than VMs. (https://www.edureka.co/blog/interview-questions/docker-interview-questions/)

9. **What are the benefits of using Docker?**

   - **Answer**: Docker simplifies configuration, ensures consistency across environments, supports rapid deployment, enables scalability, and improves resource efficiency. It allows developers to package applications with dependencies, reducing environment-specific issues. (https://intellipaat.com/blog/interview-question/docker-interview-questions-answers/)

10. **What is the difference between `CMD` and `ENTRYPOINT` in a Dockerfile?**

    - **Answer**: `CMD` specifies the default command to run when a container starts, which can be overridden. `ENTRYPOINT` defines a fixed executable, making the container behave like a specific program. `CMD` arguments can serve as parameters for `ENTRYPOINT`. (https://www.geeksforgeeks.org/devops/docker-interview-questions/)

Practical Exercises

1. **Create a Docker container from an image**

   - **Task**: Pull the `nginx:latest` image from Docker Hub and create a container that runs a web server on port 8080.
   - **Solution**:

```
docker pull nginx:latest
docker run -d -p 8080:80 --name my-nginx nginx:latest
```

- **Explanation**: Pulls the `nginx:latest` image and runs a detached container (`-d`), mapping host port 8080 to container port 80. Verify by accessing `http://localhost:8080` in a browser. (https://medium.com/%40KANIKA_VERMA/docker-practice-questions-set-2-8c11e3225983)

2. **Write a Dockerfile for a simple web server**

- **Task**: Create a Dockerfile that uses the `alpine:latest` image, installs `curl`, and runs `curl google.com`.
- **Solution**:

```
FROM alpine:latest
RUN apk add --no-cache curl
CMD ["curl", "google.com"]
```

- **Explanation**: Uses `alpine:latest` as the base image, installs `curl`, and sets the default command to fetch Google's HTML response. Build with `docker build -t curl-test .` and run with `docker run curl-test`. (https://dev.to/docker/docker-exercises-part-1-26mc)

3. **Check Docker installation details**

- **Task**: Write a command to display detailed information about the Docker installation.
- **Solution**:

```
docker info
```

- **Explanation**: The `docker info` command provides details about Docker Client, Server versions, running containers, images, and system resources. (https://www.edureka.co/blog/interview-questions/docker-interview-questions/)

Intermediate Docker Concepts

Theory Questions

1. **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 specifies services, networks, and volumes, enabling simplified management of complex applications. It's commonly used for development, testing, and CI/CD pipelines. (https://www.toptal.com/docker/interview-questions)

2. **What is Docker Swarm?**

- **Answer**: Docker Swarm is a native orchestration tool for managing a cluster of Docker hosts. It enables scaling, load balancing, and high availability for containers using an overlay network. It's simpler than Kubernetes but suitable for smaller-scale deployments. (https://www.toptal.com/docker/interview-questions)

3. **What is a Docker volume, and why is it used?**

   - **Answer**: A Docker volume is a mechanism for persisting data outside a container's filesystem. It allows data to be shared between containers and persists even if the container is deleted. Volumes are used for databases, logs, or any stateful application data. (https://www.testgorilla.com/blog/tricky-docker-interview-questions-answers/)

4. **What is a load factor in the context of Docker?**

   - **Answer**: In Docker, load factor isn't a standard term, but in hash tables (relevant to container registries or orchestration), it refers to the ratio of used slots to total slots in a hash table. For Docker, consider it analogous to resource utilization (e.g., CPU/memory usage in containers). High load factors may require scaling or optimization. (https://www.geeksforgeeks.org/devops/docker-interview-questions/)

5. **What is the difference between Docker logging at the daemon level vs. container level?**

   - **Answer**: Daemon-level logging configures global logging for all containers (e.g., debug, info, error, fatal). Container-level logging captures logs specific to an individual container, accessible via `docker logs <container_id>`. (https://www.geeksforgeeks.org/devops/docker-interview-questions/)

6. **What are Docker restart policies?**

   - **Answer**: Docker restart policies control container behavior after stopping or failing:
     - `no`: No restart (default).
     - `on-failure`: Restart only on non-user-initiated failures.
     - `unless-stopped`: Restart unless explicitly stopped by the user.
     - `always`: Always restart. Policies are set using `docker run --restart`. (https://www.interviewbit.com/docker-interview-questions/)

7. **What are common Docker networking configurations?**

   - **Answer**: Docker supports several network types:
     - **Bridge**: Default network for container-to-container communication on the same host.
     - **Host**: Containers use the host's network stack directly.
     - **Overlay**: Enables communication across multiple Docker hosts (used in Swarm).
     - **None**: No network access. (https://www.datacamp.com/blog/docker-interview-questions)

8. **What is the difference between `docker stop` and `docker kill`?**

   - **Answer**: `docker stop` sends a SIGTERM signal, allowing the container to gracefully shut down. `docker kill` sends a SIGKILL signal, forcefully terminating the container immediately. (https://www.geeksforgeeks.org/devops/docker-interview-questions/)

9. **How does Docker handle stateful applications?**

    - **Answer**: Stateful applications store data on the local filesystem, which is challenging in Docker due to container ephemerality. Use Docker volumes or bind mounts to persist data outside the container. Avoid running stateful apps directly in containers without proper persistence mechanisms. (https://medium.com/devopscurry/top-25-advanced-docker-interview-questions-91de77a80d16)

10. **What is the time complexity of searching for a container by ID in Docker?**

    - **Answer**: Docker uses efficient indexing (likely hash-based) for container lookups. Searching for a container by ID is typically O(1) on average, assuming the Docker daemon uses a hash table for container metadata. (https://medium.com/bb-tutorials-and-thoughts/250-practice-questions-for-the-dca-exam-84f3b9e8f5ce)

Practical Exercises

1. **Implement a multi-container application with Docker Compose**

    - **Task**: Create a Docker Compose file for a web application with an Nginx web server and a Python Flask app, communicating over a bridge network.
    - **Solution**:

```
version: '3'
services:
  web:
    image: nginx:latest
    ports:
      - "8080:80"
    networks:
      - app-network
  app:
    image: python:3.9
    volumes:
      - ./app:/app
    command: ["python", "/app/app.py"]
    networks:
      - app-network
networks:
  app-network:
    driver: bridge
```

        - **Explanation**: Defines two services (`web` and `app`) connected via a bridge network. The Flask app (in `app.py`) runs on the Python image, and Nginx serves as the frontend. Create a simple `app.py` with a Flask endpoint and place it in the `app` directory. Run with `docker-compose up`. (https://www.datacamp.com/blog/docker-interview-questions)

2. **Create a circular queue-like behavior using Docker containers**

- **Task**: Simulate a circular queue by running a fixed number of containers (e.g., 5) in a loop, where each container processes a task and exits, then a new one starts.
- **Solution**:

```
# Script to simulate circular queue
MAX_CONTAINERS=5
for ((i=1; i<=MAX_CONTAINERS; i++)); do
  docker run --rm -d --name task-$i alpine echo "Processing task $i"
  sleep 2
  docker stop task-$i
done
```

  - **Explanation**: Runs 5 containers sequentially, each printing a message and exiting. The `--rm` flag ensures containers are deleted after stopping, mimicking a circular queue's bounded nature. (https://dev.to/docker/docker-exercises-part-1-26mc)

3. **Merge two sorted arrays using a Dockerized Python script**

- **Task**: Write a Python script to merge two sorted arrays and run it in a Docker container.
- **Solution**:

```
# merge_arrays.py
def merge_sorted_arrays(arr1, arr2):
    result = []
    i, j = 0, 0
    while i < len(arr1) and j < len(arr2):
        if arr1[i] <= arr2[j]:
            result.append(arr1[i])
            i += 1
        else:
            result.append(arr2[j])
            j += 1
    result.extend(arr1[i:])
    result.extend(arr2[j:])
    return result

if __name__ == "__main__":
    arr1 = [1, 3, 5]
    arr2 = [2, 4, 6]
    print(merge_sorted_arrays(arr1, arr2))
```

```
FROM python:3.9
COPY merge_arrays.py /app/merge_arrays.py
WORKDIR /app
CMD ["python", "merge_arrays.py"]
```

- **Explanation**: The Python script merges two sorted arrays in O(n) time. The Dockerfile copies the script and runs it. Build with `docker build -t merge-arrays .` and run with `docker run merge-arrays`. Output: `[1, 2, 3, 4, 5, 6]`. (https://www.geeksforgeeks.org/devops/docker-interview-questions/)

4. **Check if a string is balanced using a Dockerized Python script**

- **Task**: Write a Python script to check for balanced parentheses and run it in a Docker container.
- **Solution**:

```
# balanced_parentheses.py
def is_balanced(s):
    stack = []
    brackets = {')': '(', '}': '{', ']': '['}
    for char in s:
        if char in brackets.values():
            stack.append(char)
        elif char in brackets:
            if not stack or stack.pop() != brackets[char]:
                return False
    return len(stack) == 0

if __name__ == "__main__":
    test = "{[()]}"
    print(is_balanced(test))  # True
```

```
FROM python:3.9
COPY balanced_parentheses.py /app/balanced_parentheses.py
WORKDIR /app
CMD ["python", "balanced_parentheses.py"]
```

- **Explanation**: The script uses a stack to check if parentheses are balanced. The Dockerfile runs the script in a container. Build and run with `docker build -t balanced-check .` and `docker run balanced-check`. (https://www.geeksforgeeks.org/devops/docker-interview-questions/)

5. **Find the first non-repeating character in a string**

- **Task**: Write a Python script to find the first non-repeating character in a string (e.g., "swiss") using a hash table and run it in a Docker container.
- **Solution**:

```
# non_repeating_char.py
def first_non_repeating(s):
    char_count = {}
    for char in s:
        char_count[char] = char_count.get(char, 0) + 1
    for char in s:
        if char_count[char] == 1:
            return char
    return None

if __name__ == "__main__":
    s = "swiss"
    print(first_non_repeating(s))  # w
```

```
FROM python:3.9
COPY non_repeating_char.py /app/non_repeating_char.py
WORKDIR /app
CMD ["python", "non_repeating_char.py"]
```

  - **Explanation**: Uses a hash table (dictionary) to count character frequencies, then finds the first character with a count of 1. Build and run with `docker build -t non-repeating .` and `docker run non-repeating`. (https://www.vskills.in/practice/docker-practice-questions)

Advanced Docker Concepts

Theory Questions

1. **How do you avoid hash collisions in Docker registries?**

   - **Answer**: In Docker registries, hash collisions occur when two images have the same digest (e.g., SHA256). To avoid collisions:
     - Use unique image tags and version numbers.
     - Ensure content-addressable storage (using SHA256 digests) to verify image integrity.
     - Use trusted registries (e.g., Docker Hub) with proper image signing.
     - Implement custom registries with collision-resistant hashing algorithms like SHA256. Collisions are rare due to the large hash space. (https://www.testgorilla.com/blog/tricky-docker-interview-questions-answers/)

2. **What is rehashing in the context of Docker?**

- **Answer**: Rehashing in Docker isn't a standard term but can be interpreted as rebuilding images when dependencies or configurations change, ensuring new image digests. In hash tables (e.g., for container metadata), rehashing occurs when the load factor exceeds a threshold, resizing the table to reduce collisions. In Docker, this concept applies to optimizing registry storage or metadata management. (https://medium.com/devopscurry/top-25-advanced-docker-interview-questions-91de77a80d16)

3. **What are the security best practices for Docker containers?**

   - **Answer**:
     - Use minimal base images (e.g., Alpine) to reduce attack surface.
     - Avoid running containers as root; use `USER` in Dockerfile.
     - Limit container privileges with Linux capabilities or seccomp profiles.
     - Use Docker secrets for sensitive data.
     - Regularly scan images for vulnerabilities using tools like `docker scout`.
     - Keep Docker and images updated.
     - Secure network communication with TLS. (https://www.testgorilla.com/blog/tricky-docker-interview-questions-answers/)

4. **What is the time complexity of Docker container operations?**

   - **Answer**:
     - **Start/Stop**: O(1) for sending signals to the Docker daemon.
     - **Image Pull/Push**: O(n) where n is the image size, due to network transfer.
     - **Container Lookup (by ID)**: O(1) assuming hash-based indexing.
     - **Image Build**: O(n) where n is the number of Dockerfile instructions, due to sequential execution. (https://medium.com/bb-tutorials-and-thoughts/250-practice-questions-for-the-dca-exam-84f3b9e8f5ce)

5. **How does Docker handle rapid deployment with tools like Jenkins?**

   - **Answer**: Docker integrates with Jenkins for CI/CD by:
     - Building images in Jenkins pipelines using `docker build`.
     - Running tests in containers for consistent environments.
     - Pushing images to registries (e.g., Docker Hub).
     - Deploying containers to production with `docker run` or orchestration tools like Docker Swarm/Kubernetes. This ensures rapid, repeatable deployments with minimal configuration changes. (https://intellipaat.com/blog/interview-question/docker-interview-questions-answers/)

6. **What is the difference between open addressing and separate chaining in hash tables, and how does it relate to Docker?**

   - **Answer**: Open addressing resolves hash collisions by probing (e.g., linear or quadratic) to find an empty slot in the same table. Separate chaining uses linked lists to store multiple items at the same hash index. In Docker, separate chaining is analogous to handling image layers or container metadata in registries, where collisions are managed by unique digests or tags. Open addressing could apply to optimizing storage in constrained environments. (https://www.geeksforgeeks.org/devops/docker-interview-questions/)

7. **What is the role of the Docker daemon?**

    - **Answer**: The Docker daemon (`dockerd`) is a background process that manages Docker objects (containers, images, networks, volumes). It handles API requests from the Docker CLI, orchestrates container lifecycles, and interacts with the host OS for resource allocation. (https://medium.com/devopscurry/top-25-advanced-docker-interview-questions-91de77a80d16)

8. **What are monotonic stacks and queues in the context of Docker?**

    - **Answer**: Monotonic stacks/queues maintain elements in sorted order (increasing or decreasing). In Docker, they aren't directly used but could be applied in container orchestration for scheduling tasks (e.g., maintaining a monotonic queue of container start times to prioritize resource allocation). (https://dev.to/docker/docker-exercises-part-1-26mc)

9. **Why is Docker preferred for microservices?**

    - **Answer**: Docker is ideal for microservices because:
        - Containers isolate services, ensuring independent scaling and deployment.
        - Lightweight containers reduce overhead compared to VMs.
        - Docker Compose/Swarm simplifies multi-container management.
        - Consistent environments reduce deployment issues. (https://intellipaat.com/blog/interview-question/docker-interview-questions-answers/)

10. **What are the disadvantages of Docker?**

    - **Answer**:
        - Limited support for stateful applications without proper volume management.
        - Security risks if containers are misconfigured (e.g., running as root).
        - Overhead in managing large-scale clusters (Kubernetes is often preferred).
        - Learning curve for advanced features like networking and orchestration. (https://www.guru99.com/docker-interview-questions.html)

Practical Exercises

1. **Implement a hash table for character frequency in a Dockerized app**

    - **Task**: Create a Python script to count character frequencies in "Mississippi" using a hash table, and run it in a Docker container.
    - **Solution**:

```
# char_frequency.py
def count_frequency(s):
    freq = {}
    for char in s:
        freq[char] = freq.get(char, 0) + 1
    return freq

if __name__ == "__main__":
    s = "Mississippi"
    print(count_frequency(s))
```

```
FROM python:3.9
COPY char_frequency.py /app/char_frequency.py
WORKDIR /app
CMD ["python", "char_frequency.py"]
```

- **Explanation**: The script uses a dictionary (hash table) to count frequencies. Build with `docker build -t char-freq .` and run with `docker run char-freq`. Output: `{'M': 1, 'i': 4, 's': 4, 'p': 2}`. (https://www.vskills.in/practice/docker-practice-questions)

2. **Reverse a string using a stack in a Dockerized app**

- **Task**: Write a Python script to reverse a string using a stack and run it in a Docker container.
- **Solution**:

```
# reverse_string.py
def reverse_string(s):
    stack = []
    for char in s:
        stack.append(char)
    return ''.join(stack.pop() for _ in range(len(stack)))

if __name__ == "__main__":
    s = "Hello, Docker!"
    print(reverse_string(s))
```

```
FROM python:3.9
COPY reverse_string.py /app/reverse_string.py
WORKDIR /app
CMD ["python", "reverse_string.py"]
```

- **Explanation**: The script uses a list as a stack to reverse the string. Build and run with `docker build -t reverse-string .` and `docker run reverse-string`. Output: `!rekcoD ,olleH`. (https://dev.to/docker/docker-exercises-part-1-26mc)

3. **Sort an array alphabetically using Quick Sort in a Dockerized app**

   - **Task**: Write a Python script to sort an array of strings alphabetically using Quick Sort and run it in a Docker container.
   - **Solution**:

```
# quick_sort.py
def quick_sort(arr, low, high):
    if low < high:
        pi = partition(arr, low, high)
        quick_sort(arr, low, pi - 1)
        quick_sort(arr, pi + 1, high)

def partition(arr, low, high):
    pivot = arr[high]
    i = low - 1
    for j in range(low, high):
        if arr[j] <= pivot:
            i += 1
            arr[i], arr[j] = arr[j], arr[i]
    arr[i + 1], arr[high] = arr[high], arr[i + 1]
    return i + 1

if __name__ == "__main__":
    arr = ["banana", "apple", "cherry", "date"]
    quick_sort(arr, 0, len(arr) - 1)
    print(arr)
```

```
FROM python:3.9
COPY quick_sort.py /app/quick_sort.py
WORKDIR /app
CMD ["python", "quick_sort.py"]
```

   - **Explanation**: Implements Quick Sort to sort strings alphabetically in O(n log n) average time. Build and run with `docker build -t quick-sort .` and `docker run quick-sort`. Output: `['apple', 'banana', 'cherry', 'date']`. (https://www.geeksforgeeks.org/devops/docker-interview-questions/)

4. **Implement a secure Docker container**

- **Task**: Create a Dockerfile for a Node.js app that runs as a non-root user and uses a minimal base image.
- **Solution**:

```
FROM node:18-alpine
RUN addgroup -S appgroup && adduser -S appuser -G appgroup
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
USER appuser
CMD ["node", "index.js"]
```

  - **Explanation**: Uses `node:18-alpine` for a minimal base image, creates a non-root user (`appuser`), and runs the app as that user to enhance security. Requires an `index.js` file in the project directory. Build with `docker build -t secure-node .` and run with `docker run secure-node`. (https://www.testgorilla.com/blog/tricky-docker-interview-questions-answers/)

5. **Scan a Docker image for vulnerabilities**

- **Task**: Build a Docker image and scan it for vulnerabilities using `docker scout`.
- **Solution**:

```
FROM python:3.9
COPY app.py /app/app.py
WORKDIR /app
CMD ["python", "app.py"]
```

```
docker build -t my-app .
docker scout cves my-app
```

  - **Explanation**: Builds a simple Python image and uses `docker scout cves` to check for vulnerabilities. If vulnerabilities are found, update the base image or dependencies to fix them (e.g., use `python:3.9-slim` for a smaller attack surface). (https://dev.to/docker/docker-exercises-part-1-26mc)

Guidelines

- **Practice Platforms**: Use Docker Desktop, Docker Playground, or platforms like LeetCode for container-related challenges.
- **Resources**: Refer to Docker's official documentation, DataCamp's Docker courses, and YouTube tutorials for practical solutions. (https://www.datacamp.com/blog/docker-interview-questions)

- **Best Practices**: Focus on understanding Docker concepts rather than memorizing commands. Practice debugging, optimize Dockerfiles, and ensure security in configurations.
- **Certifications**: Prepare for the Docker Certified Associate (DCA) exam with practice tests from Whizlabs or Udemy., (https://www.whizlabs.com/blog/docker-certified-associate-exam-questions/) (https://www.udemy.com/course/docker-practice-questions-practice-docker-hands-on-50-task/)
- **Optimization**: Experiment with image size reduction, multi-stage builds, and orchestration tools like Docker Swarm or Kubernetes.
- **Security**: Always follow best practices (e.g., minimal images, non-root users, vulnerability scanning) to ensure secure deployments.