Hands-on Exercise 3: Docker Container Fundamentals

Overview

These exercises will help you master container lifecycle management and resource allocation - essential skills for running containers in production environments.

Prerequisites

- Docker installed and running on your system
- Basic familiarity with Docker commands
- Terminal or command prompt access

Exercise 3.1: Container Lifecycle Management

Objective

Understand container states, transitions between states, and how to implement proper restart policies.

Tasks

1. Create containers with different configurations

```
# Create a container that runs and exits immediately
docker run --name single-run alpine echo "This container runs once and exits"

# Create a container that runs in the background
docker run -d --name always-running nginx

# Create a container with a custom command
docker run -d --name custom-command ubuntu sleep 300

# Create a container with environment variables
docker run -d --name env-container -e VARIABLE1=value1 -e VARIABLE2=value2
nginx
```

Check the status of all containers:

```
docker ps -a
```

2. Explore lifecycle states and transitions

```
# Start an interactive container
docker run -it --name interactive-container ubuntu bash

# Inside the container, run some commands, then exit
ls -la
echo "Testing container interaction"
exit

# Restart the exited container
docker start interactive-container

# Attach to the running container
docker attach interactive-container

# From another terminal, pause the container
docker pause interactive-container

# Check the container status
docker ps -a

# Unpause the container
docker unpause interactive-container

# Stop the container
docker stop interactive-container
```

3. Implement restart policies

```
# Create a container with "always" restart policy
docker run -d --name restart-always --restart always nginx

# Create a container with "on-failure" restart policy (max 5 attempts)
docker run -d --name restart-on-failure --restart on-failure:5 ubuntu sleep 10

# Create a container with "unless-stopped" restart policy
docker run -d --name restart-unless-stopped --restart unless-stopped nginx

# Test restart policies
# Stop the Docker service (will vary based on your system)
sudo systemctl stop docker

# Start the Docker service
sudo systemctl start docker

# Check which containers restarted
docker ps -a
```

4. Test volume data persistence

```
# Create a named volume
docker volume create data-volume

# Run a container with the volume and create a file
docker run -it --name volume-test -v data-volume:/data ubuntu bash
# Inside the container:
echo "This data will persist" > /data/test-file.txt
ls -la /data
exit

# Remove the container
docker rm volume-test

# Create a new container using the same volume
docker run -it --name volume-test-2 -v data-volume:/app-data ubuntu bash
# Inside the container:
cat /app-data/test-file.txt
exit

# Clean up
docker rm volume-test-2
```

Expected Results

- You should understand how containers transition between different states
- You should understand how different restart policies affect container behavior
- You should confirm that data in volumes persists even when containers are removed

Exercise 3.2: Container Resource Management

Objective

Learn how to allocate, monitor, and manage container resources including memory, CPU, and I/O.

Tasks

1. Set memory and CPU constraints

```
# Run container with memory limits
docker run -d --name memory-test --memory=512m --memory-reservation=256m nginx

# Run container with CPU limits (0.5 CPU)
docker run -d --name cpu-test --cpus=0.5 nginx

# Run container with CPU shares (relative weight)
docker run -d --name cpu-shares --cpu-shares=512 nginx

# Verify the settings
docker inspect memory-test | grep -i memory
docker inspect cpu-test | grep -i cpu
docker inspect cpu-shares | grep -i cpu
```

2. Monitor resource usage under load

```
# Install stress tool in a container
docker run -d --name stress-test ubuntu:latest sleep 600
docker exec -it stress-test bash

# Inside the container:
apt-get update && apt-get install -y stress
exit

# Run stress test with specific resource constraints
docker run -d --name stress-memory --memory=512m ubuntu sleep 600
docker exec -it stress-memory bash

# Inside the container:
apt-get update && apt-get install -y stress
# Run memory stress test (allocate 256MB)
stress --vm 1 --vm-bytes 256M --timeout 60s
exit

# Monitor container stats while tests are running
docker stats
```

3. Implement and test I/O throttling

```
# Create container with I/O limits
docker run -d --name io-limited --device-write-bps /dev/sda:10mb ubuntu sleep
600

# Compare to a container without I/O limits
docker run -d --name io-unlimited ubuntu sleep 600

# Test I/O performance
docker exec -it io-limited bash

# Inside the container:
apt-get update && apt-get install -y dd
time dd if=/dev/zero of=test.img bs=1M count=100 oflag=direct
exit

# Test the unlimited container
docker exec -it io-unlimited bash

# Inside the container:
apt-get update && apt-get install -y dd
time dd if=/dev/zero of=test.img bs=1M count=100 oflag=direct
exit
```

4. Observe and resolve resource constraint issues

```
# Create a memory-constrained container
docker run -d --name memory-limit-test --memory=50m nginx

# Check container is running
docker ps -a

# Generate memory pressure
docker exec -it memory-limit-test bash

# Inside the container:
apt-get update && apt-get install -y python3
python3 -c "a = ' ' * 100 * 1024 * 1024"
exit

# Check if container was killed (OOMKilled)
docker ps -a
docker inspect memory-limit-test | grep -i oomkilled

# Resolve by increasing memory limit
docker rm -f memory-limit-test
docker run -d --name memory-limit-test --memory=150m nginx

# Verify the new container stays running
docker ps
```

Expected Results

- You should observe the effect of resource constraints on container performance
- You should be able to monitor container resource usage
- You should understand the impact of OOMKilled and how to resolve it by proper resource allocation

Cleanup

```
# Remove all containers created in this exercise
docker rm -f $(docker ps -aq)

# Remove the volume
docker volume rm data-volume
```

Challenge Tasks

1. Create a container that auto-heals: When it crashes, it should automatically restart, and you should be able to see the restart count.

2. Create a setup where one container writes data to a volume, and another container reads that data automatically.

3. Create a container with specific CPU and memory limits, then monitor its resource usage while running a real-world application (like a web server under load).