Docker Lab: Container Basics

Goal

Learn the fundamentals of working with Docker containers. You'll practice pulling images, running containers, managing their lifecycle, and interacting with running containers using popular web server and database images.

Prerequisites

- Linux system with Docker installed
- Docker daemon running (`sudo systemctl status docker`)
- Internet connection

Tasks

Task 1: Run Your First Container

Objective: Run Docker's hello-world image to verify your installation

Instructions:

1. Run the hello-world container:

```
docker run hello-world
```

2. Read the output message explaining what just happened

Expected Output: You should see a message that says "Hello from Docker!" followed by an explanation of the steps Docker took to display this message.

What Happened: - Docker pulled the `hello-world` image from Docker Hub - Docker created a container from that image - Docker ran the container, which printed the message - The container exited after completing its task

---

Task 2: List Docker Images

Objective: View the images you have on your system

Instructions:

1. List all Docker images:

```
docker images
```

2. Observe the hello-world image you just pulled
3. Note the image ID, size, and creation date

Expected Output:

```
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
hello-world   latest    xxxxxxxxxxxx  X months ago   XX.XkB
```

Task 3: Run an NGINX Web Server

Objective: Run a web server container and access it from your browser

Instructions:

1. Run nginx container with port mapping:

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

   - `-d` runs the container in detached mode (background)
   - `-p 8080:80` maps port 8080 on your host to port 80 in the container
   - `--name my-nginx` gives the container a friendly name

2. Verify the container is running:

   ```
   docker ps
   ```

3. Test the web server:

   ```
   curl http://localhost:8080
   ```

Expected Output: - `docker ps` should show your nginx container with STATUS "Up" - `curl` should return HTML content with "Welcome to nginx!"

Browser Test: Open `http://localhost:8080` in your web browser to see the nginx welcome page.

Task 4: View Container Logs

Objective: Check the logs of a running container

Instructions:

1. View the logs from your nginx container:

   ```
   docker logs my-nginx
   ```

2. Make another request to the web server:

   ```
   curl http://localhost:8080
   ```

3. View the logs again to see the new request:

   ```
   docker logs my-nginx
   ```

Expected Output: You should see access logs showing GET requests to "/" with HTTP status code 200.

Task 5: Execute Commands Inside a Container

Objective: Run commands inside a running container

Instructions:

1. Execute a bash shell inside the nginx container:

```
docker exec -it my-nginx bash
```

- `-it` gives you an interactive terminal

2. Once inside, run some commands:

```
ls /usr/share/nginx/html/
cat /usr/share/nginx/html/index.html
hostname
exit
```

Expected Output: - You'll see the nginx HTML files - The hostname will be the container ID - `exit` returns you to your host terminal

---

Task 6: Stop and Remove Containers

Objective: Learn how to stop and clean up containers

Instructions:

1. Stop the nginx container:

```
docker stop my-nginx
```

2. List all containers (including stopped):

```
docker ps -a
```

3. Remove the stopped container:

```
docker rm my-nginx
```

4. Verify it's removed:

```
docker ps -a
```

Expected Output: - `docker stop` should return the container name - `docker ps -a` first shows the stopped container with STATUS "Exited" - After `docker rm`, the container should no longer appear in the list

---

Task 7: Run a PostgreSQL Database Container

Objective: Run a database container with environment variables

Instructions:

1. Run a PostgreSQL container (copy this entire line):

```
docker run -d --name my-postgres -e POSTGRES_PASSWORD=mysecretpassword -e POSTGRES_DB=testdb -p 5432:5432 postgres
```

(Note: `-e` sets environment variables. Use the single line above when copying to avoid paste errors.)

2. Wait a few seconds for the database to initialize, then check logs:

```
docker logs my-postgres
```

Look for “database system is ready to accept connections”

3. Connect to the database using psql:

```
docker exec -it my-postgres psql -U postgres -d testdb
```

4. Inside psql, run these SQL commands (type them one at a time and press Enter, or copy-paste the block). Each command is explained below so you know what you’re doing:

```
\l
\dt
CREATE TABLE users (id SERIAL PRIMARY KEY, name VARCHAR(100));
INSERT INTO users (name) VALUES ('John Doe');
SELECT * FROM users;
\q
```

What each command does:

- **`\l`** — Lists all databases on the server. You should see `testdb` in the list (the one we created with the container).
- **`\dt`** — Lists all tables in the current database. At first it will be empty; after you create the table, you’ll see `users`.
- **`CREATE TABLE users (...)`** — Creates a new table named `users`. The columns are: `id` (auto-incrementing number, used as the primary key) and `name` (text, up to 100 characters). The semicolon `;` at the end tells PostgreSQL to run the command.
- **`INSERT INTO users (name) VALUES ('John Doe');`** — Adds one row into the `users` table: the `name` column gets the value `'John Doe'`. The `id` is filled automatically.
- **`SELECT * FROM users;`** — Shows all rows in the `users` table. You should see one row with id 1 and name John Doe.
- **`\q`** — Quits the psql client and returns you to your normal terminal (still inside the container until you type `exit`).

Expected Output: - The logs show PostgreSQL starting successfully - You can connect to the database - `\l` shows a list including `testdb`; `\dt` is empty at first, then shows `users` - After INSERT and SELECT, you see one row: id=1, name=John Doe - `\q` exits the psql client

---

Task 8: Inspect Container Details

Objective: View detailed information about a container

Instructions:

1. Inspect the PostgreSQL container:

```
docker inspect my-postgres
```

2. View specific information using grep or formatting:

```
docker inspect my-postgres | grep IPAddress
```

3. Check container resource usage:

```
docker stats my-postgres --no-stream
```

Expected Output: - `docker inspect` returns detailed JSON about the container - You can see the container’s IP address - `docker stats` shows CPU, memory, and network usage

Task 9: Run Multiple Containers

Objective: Run multiple containers simultaneously

Instructions:

1. Run another nginx container on a different port:

   ```
   docker run -d -p 8081:80 --name nginx-2 nginx
   ```

2. List all running containers:

   ```
   docker ps
   ```

3. Test the nginx container you just started (the first nginx was stopped and removed in Task 6, so only nginx-2 is running now):

   ```
   curl http://localhost:8081
   ```

Expected Output: - `docker ps` shows postgres and nginx-2 running (two containers) - The curl command to port 8081 returns the nginx welcome page - Each container runs independently; you can have multiple containers at once

---

Task 10: Clean Up Everything

Objective: Stop and remove all containers and images

Instructions:

1. Stop all running containers:

   ```
   docker stop my-postgres nginx-2
   ```

2. Remove all containers (including the exited hello-world container from Task 1; it stays as “Exited” and blocks removing its image until you remove it):

   ```
   docker rm $(docker ps -aq)
   ```

   This removes every stopped container. Without this, `docker rmi hello-world` in step 4 would fail with “container is using the image”.

3. View all images:

   ```
   docker images
   ```

4. (Optional) Remove images to free up space:

   ```
   docker rmi nginx postgres hello-world
   ```

Expected Output: - All containers are stopped and removed - `docker ps -a` shows no containers - Images are removed (if you chose to do so)

Completion

Congratulations! You've successfully completed the Docker basics lab!

Key Concepts Learned:

- Pulling and running Docker images
- Running containers in detached mode
- Port mapping between host and container
- Viewing container logs
- Executing commands inside containers
- Managing container lifecycle (start, stop, remove)
- Working with environment variables
- Running multiple containers
- Inspecting container details

Docker Commands Mastered:

- `docker run` - Create and start containers
- `docker ps` - List containers
- `docker images` - List images
- `docker logs` - View container logs
- `docker exec` - Execute commands in containers
- `docker stop` - Stop running containers
- `docker rm` - Remove containers
- `docker rmi` - Remove images
- `docker inspect` - View detailed container info
- `docker stats` - View resource usage

What You've Achieved:

✓ Ran your first Docker container ✓ Deployed a web server (nginx) ✓ Deployed a database (PostgreSQL) ✓ Managed multiple containers simultaneously ✓ Interacted with running containers ✓ Understood container lifecycle management

Next Steps:

Now that you understand Docker basics, you can explore: - Creating custom Docker images with Dockerfiles - Docker networking and container communication - Docker volumes for persistent data - Docker Compose for multi-container applications - Building and pushing your own images to Docker Hub