Lab 3

Lab 3: Manage Multiple Containers Using Docker Compose Setup

Objective

In this lab, you will learn how to configure and deploy a full-stack application consisting of a React frontend, Rust backend, and a PostgreSQL database using Docker Compose. This will demonstrate the power and convenience of managing multi-container Docker applications.

Prerequisites

- Docker and Docker Compose installed on your machine.
- Git installed in the machine (only essential for cloning the project from GitHub)
- Basic knowledge of Docker, React, Rust, and PostgreSQL.
- Ensure that port 3000 on your host is free as it will be used by the React application.

Project Structure

Here's what the directory structure of the project looks like:

```
.
├── backend
│   ├── Dockerfile
│   ├── Cargo.toml
│   └── src
├── frontend
│   ├── Dockerfile
│   ├── package.json
│   └── src
├── compose.yaml
├── README.md
├── fe.env # Environment variables for the frontend
├── be.env # Environment variables for the backend
└── db.env # Environment variables for the database
```

Service Configuration in Compose File

The compose.yaml file defines three services as follows:

- **Backend**: A Rust-based service built from a Dockerfile in the backend directory.
- **DB**: A PostgreSQL database using the postgres:12-alpine image.
- **Frontend**: A React application built from a Dockerfile in the frontend directory. It maps port 3000 of the container to port 3000 on the host.

This docker-compose.yaml file defines a multi-service application with three primary services: frontend, backend and db. It sets up the necessary configurations for building the images, linking containers, managing volumes, and setting environment variables.

Services

The services section is where each container in your application is defined.

Frontend Service

- **build**:
  - **context**: Specifies the directory containing the Dockerfile and the source code for the frontend. Docker will look here when building the image.
  - **target**: Used in multi-stage builds to specify which build stage to target. Here, it targets the development stage for building the frontend.
- **networks**:
  - **client-side**: This service is attached to the client-side network, allowing it to communicate with other services on the same network.
- **ports**:
  - **3000:3000**: Maps port 3000 of the container to port 3000 on the host, making the frontend accessible via http://localhost:3000.
- **volumes**:
  - **./frontend/src:/code/src:ro**: Mounts the src directory from the host inside the container at /code/src. The read-only option prevents the container from modifying the source code on the host.
- **env_file**:
  - **fe.env**: Specifies a file from which to load environment variables for the frontend service.
- **restart**:
  - **always**: Ensures that the frontend service always restarts unless it is manually stopped. Useful for maintaining availability.

Backend Service

- **build**:
  - **context**: Specifies the directory containing the Dockerfile and the source code for the backend.
  - **target**: Specifies the development stage in the Dockerfile for building the backend.
- **networks**:
  - **client-side**, **server-side**: Connects the backend service to both the client-side & server-side networks, facilitating communication with both the frontend and the database.
- **volumes**:
  - **./backend/src:/code/src**: Mounts the backend source code into the container for live code changes.
  - **backend-cache:/code/target**: Uses a named volume build-cache to persist build artifacts and dependencies, which can speed up build times.
- **depends_on**:
  - **db**: Specifies that the backend service should start only after the db service has started.
- **env_file**:
  - **be.env**: Specifies a file from which to load environment variables for the backend service.
- **restart**:
  - **always**: Ensures that the backend service always restarts unless manually stopped.

Database Service

- **image**:
  - **postgres:17-alpine**: Uses the PostgreSQL 17 image based on Alpine Linux. It's a lightweight version of the PostgreSQL image.
- **networks**:
  - **server-side**: Attaches the database service to the server-side network, allowing communication with the backend.
- **ports**:
  - **5433:5432**: Maps port 5432 inside the container to port 5433 on the host, it is not mapped with port 5432 on host as usually it is already occupied by our on-host postgres server.
- **volumes**:

- **db-data:/var/lib/postgresql/data**: Persists the database data in a named volume db-data to ensure data is saved across container restarts.

- **env_file**:
  - **db.env**: Loads environment variables specific to the database, such as passwords, user names, and database names.
- **restart**:
  - **always**: Configures the database service to always restart unless it is manually stopped.

Networks

- **client-side** and **server-side**: Defines two custom networks. Services on the same network can communicate with each other. The client-side network is likely used for communications between the frontend and backend, while the server-side network connects the backend with the database.

Volumes

- **backend-cache** and **db-data**: These named volumes are used for storing data outside of containers, maintaining data persistence across container rebuilds and restarts.

This setup is structured to facilitate development and interaction between services by defining clear network boundaries and persistent storage options. Each component is modular, allowing for isolated development and scaling.

Step 1: Clone the project

Create a folder: "compose-demo" & then go inside the folder.

Open any terminal in this folder and run:

git clone https://github.com/saquibmansuri/react-rust-postgres-sample-project (Links to an external site.)

```
PS D:\Docker\Lab-3> cd compose-demo
PS D:\Docker\Lab-3\compose-demo> git clone https://github.com/saquibmansuri/react-rust-postgres-sample-project
Cloning into 'react-rust-postgres-sample-project'...
remote: Enumerating objects: 51, done.
remote: Counting objects: 100% (51/51), done.
remote: Compressing objects: 100% (41/41), done.
remote: Total 51 (delta 4), reused 51 (delta 4), pack-reused 0 (from 0)
Receiving objects: 100% (51/51), 290.79 KiB | 4.69 MiB/s, done.
Resolving deltas: 100% (4/4), done.
PS D:\Docker\Lab-3\compose-demo> ls


    Directory: D:\Docker\Lab-3\compose-demo


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----          3/5/2025   4:31 PM                react-rust-postgres-sample-project
```

Step 2: Deploy with Docker Compose

Navigate to the root of the project directory where the compose.yaml file is located and run the following command to start the services:

docker compose up -d

```
PS D:\Docker\Lab-3\compose-demo\react-rust-postgres-sample-project> docker compose up -d
 [+] Running 11/11
  ✓ db Pulled                                                                                                    10.6s
    ✓ 8f4971a5dfe7 Download complete                                                                              0.4s
    ✓ f18232174bc9 Download complete                                                                              1.2s
    ✓ 9e4acb9ca7d3 Download complete                                                                              0.8s
    ✓ 215ba3ecdc26 Download complete                                                                              3.9s
    ✓ d8d8fb695a5a Download complete                                                                              0.6s
    ✓ c24c1ba610df Download complete                                                                              2.0s
    ✓ 4f18957d9158 Download complete                                                                              1.1s
    ✓ 15ca4c67ed92 Download complete                                                                              1.1s
    ✓ 83efd74bc97e Download complete                                                                              1.1s
    ✓ 404c53e09e31 Download complete                                                                              1.0s
 [+] Building 358.4s (29/29) FINISHED                                                                docker:desktop-linux
  => [frontend internal] load build definition from Dockerfile                                                    0.3s
  => => transferring dockerfile: 735B                                                                             0.0s
  => [backend internal] load build definition from Dockerfile                                                     0.3s
  => => transferring dockerfile: 938B                                                                             0.0s
  => [backend] resolve image config for docker-image://docker.io/docker/dockerfile:1.4                            3.0s
  => [frontend auth] docker/dockerfile:pull token for registry-1.docker.io                                        0.0s
  => [frontend] docker-image://docker.io/docker/dockerfile:1.4@sha256:9ba7531bd80fb0a858632727cf7a112fbfd19b17e94c4e84ced81e24ef1a0dbc  1.3s
  => => resolve docker.io/docker/dockerfile:1.4@sha256:9ba7531bd80fb0a858632727cf7a112fbfd19b17e94c4e84ced81e24ef1a0dbc  0.0s
  => => sha256:1328b32c40fca9bcf9d70d8eccb72eb873d1124d72dadce04db8badbe7b08546 9.94MB / 9.94MB                    0.9s
  => => extracting sha256:1328b32c40fca9bcf9d70d8eccb72eb873d1124d72dadce04db8badbe7b08546                         0.2s
  => [backend internal] load .dockerignore                                                                        0.1s
  => => transferring context: 60B                                                                                 0.0s
  => [frontend internal] load .dockerignore                                                                       0.1s
```


Step 3: Verify the Deployment

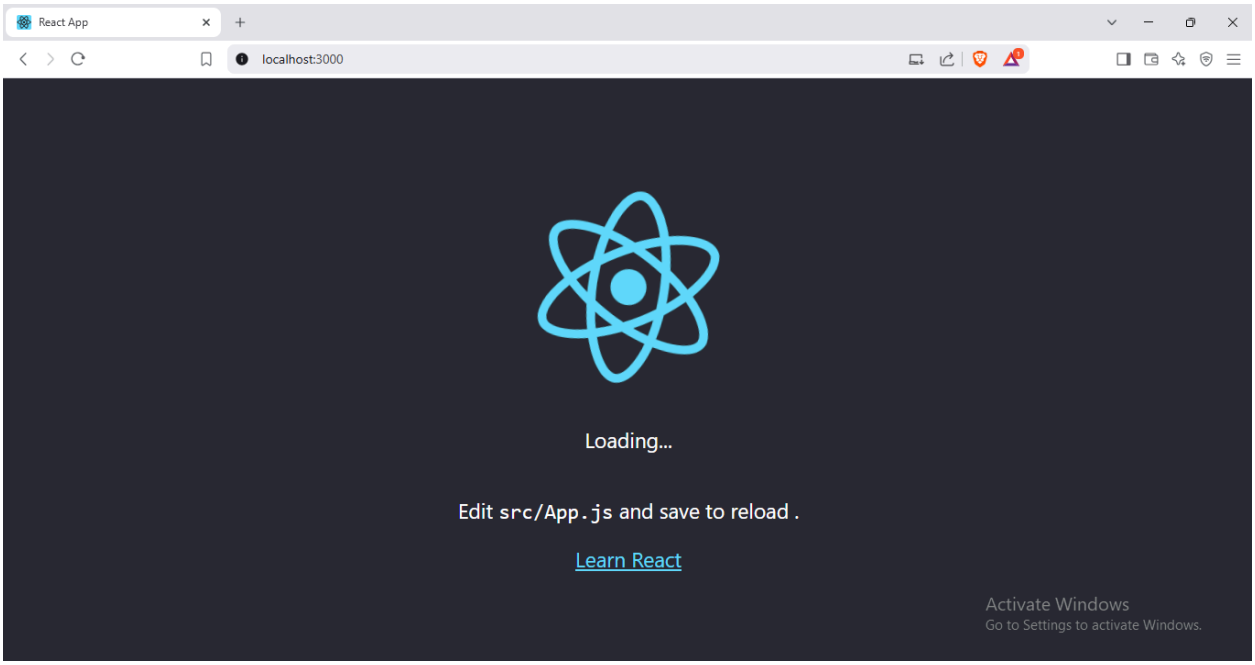
Check that all containers are up and running by executing:

docker ps

```
PS D:\Docker\Lab-3\compose-demo\react-rust-postgres-sample-project> docker ps
CONTAINER ID   IMAGE                                      COMMAND                  CREATED          STATUS                                  PORTS
               NAMES
a2a50d1f8de1   react-rust-postgres-sample-project-backend    "cargo run --offline"    5 seconds ago    Restarting (101) Less than a second ago
               react-rust-postgres-sample-project-backend-1
45034bb58a16   postgres:17-alpine                         "docker-entrypoint.s…"   6 seconds ago    Up 4 seconds                            0.0.0.0:5
433->5432/tcp   react-rust-postgres-sample-project-db-1
75242d42662d   react-rust-postgres-sample-project-frontend   "docker-entrypoint.s…"   6 seconds ago    Up 4 seconds                            0.0.0.0:3
000->3000/tcp   react-rust-postgres-sample-project-frontend-1
7b850acaec70   flask-docker-demo-app                      "python demo.py"         40 minutes ago   Up 40 minutes                           0.0.0.0:5
001->5001/tcp   flask-docker-demo-app
PS D:\Docker\Lab-3\compose-demo\react-rust-postgres-sample-project>
```


Step 4: Access the Application

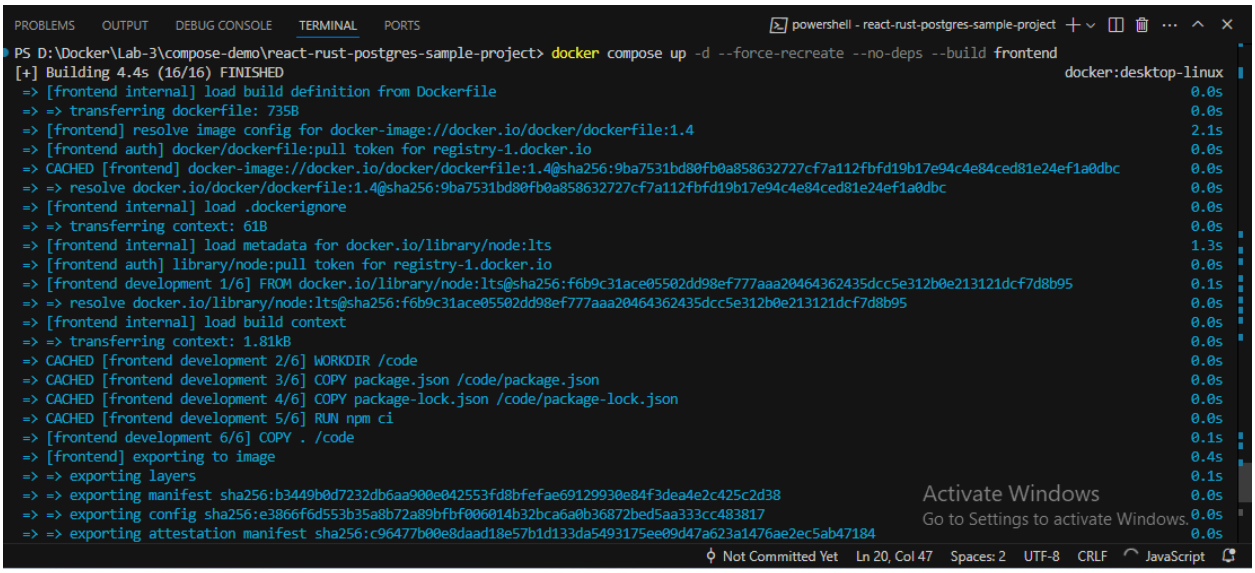
Open a web browser and navigate to http://localhost:3000. You should see the React application's frontend, which communicates with the Rust backend and retrieves data stored in the PostgreSQL database.

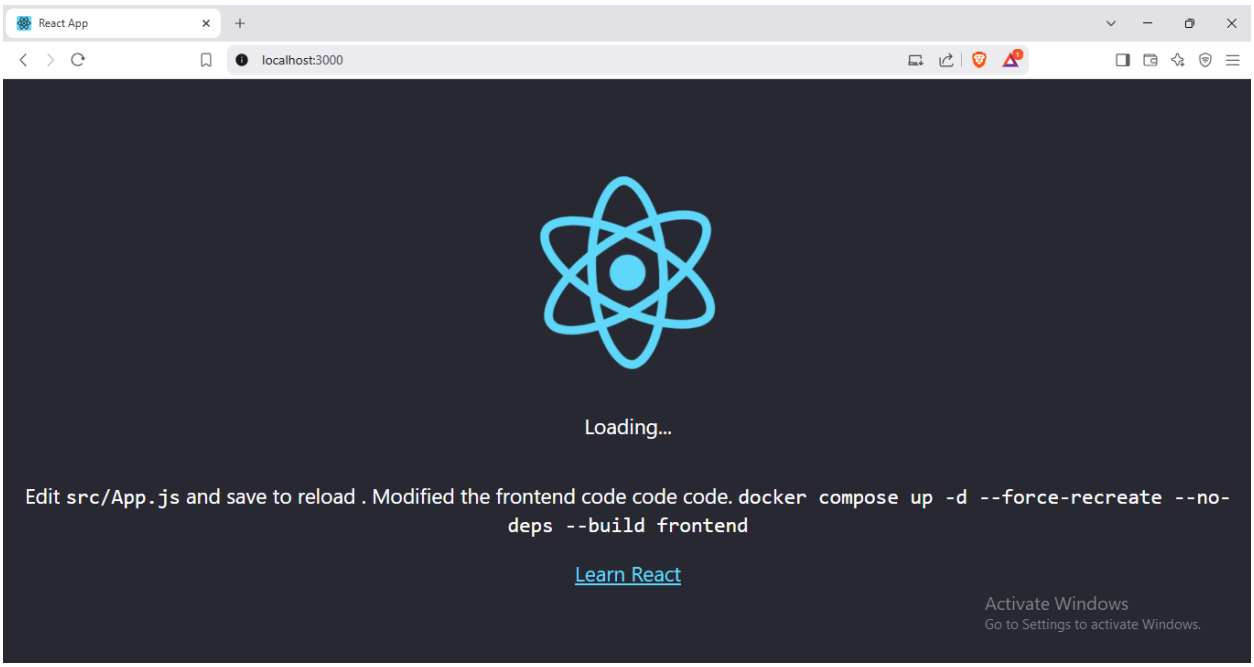


Step 5: Modify App and rebuild particular service

Suppose you modify the frontend code, then best practice is to rebuild and restart frontend service only, and vice versa

docker compose up -d --force-recreate --no-deps --build frontend



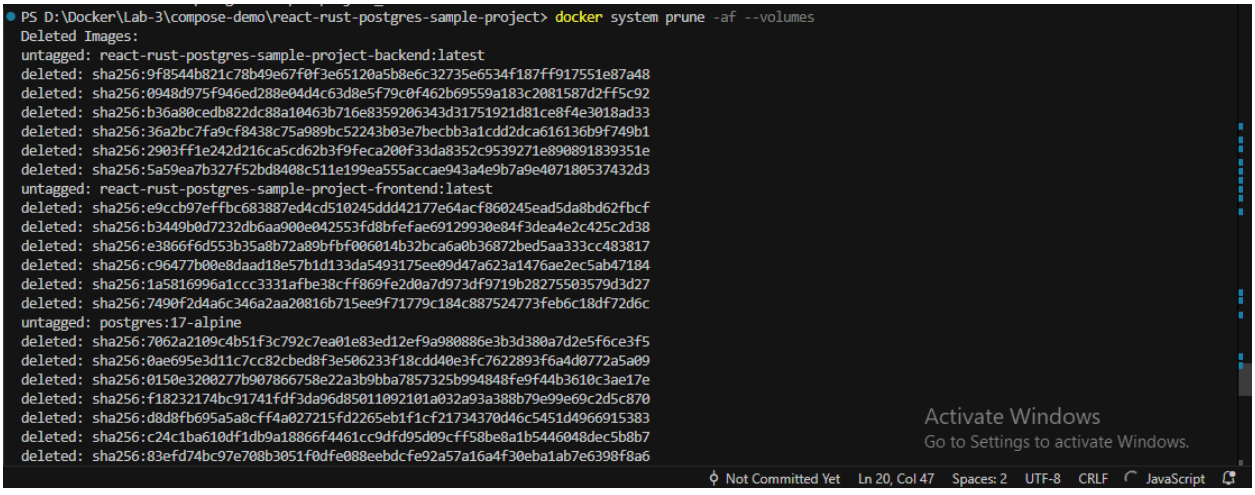


Step 6: Stop and Remove the Containers

To stop and remove all containers along with the network, volumes, images & cache, use the following command:

docker compose down

docker system prune -af --volumes



You've finished the lab!!