



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
  ✓ 215ba3ecd26 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
=> [frontend internal] load build definition from Dockerfile
=> => transferring dockerfile: 735B
=> [backend internal] load build definition from Dockerfile
=> => transferring dockerfile: 938B
=> [backend] resolve image config for docker-image://docker.io/docker/dockerfile:1.4
=> [frontend auth] docker/dockerfile:pull token for registry-1.docker.io
=> [frontend] docker-image://docker.io/docker/dockerfile:1.4@sha256:9ba7531bd80fb0a858632727cf7a112fbfd19b17e94c4e84ced81e24ef1a0dbc
=> => resolve docker.io/docker/dockerfile:1.4@sha256:9ba7531bd80fb0a858632727cf7a112fbfd19b17e94c4e84ced81e24ef1a0dbc
=> => sha256:1328b32c40fca9bcf9d70d8ecb72eb873d1124d72dadce04db8badbe7b08546 9.94MB / 9.94MB
=> => extracting sha256:1328b32c40fca9bcf9d70d8ecb72eb873d1124d72dadce04db8badbe7b08546
=> [backend internal] load .dockerignore
=> => transferring context: 60B
=> [frontend internal] load .dockerignore

```

Step 3: Verify the Deployment

Check that all containers are up and running by executing:

docker ps

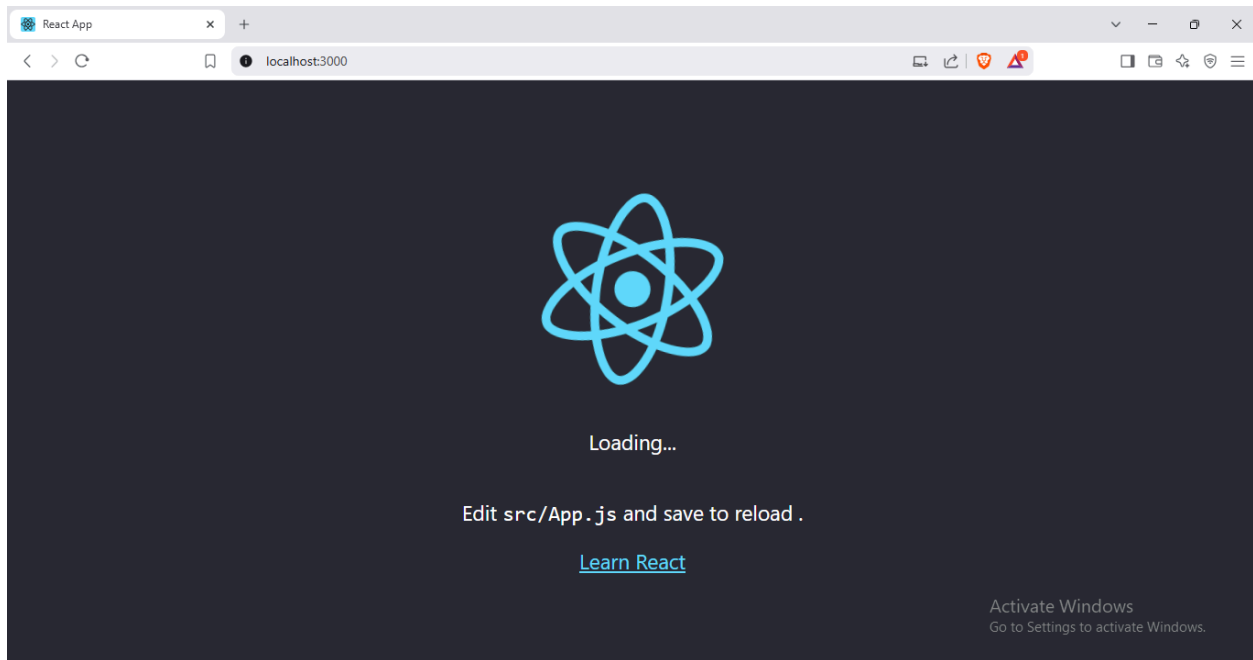
```

PS D:\Docker\Lab-3\compose-demo\react-rust-postgres-sample-project> docker ps
CONTAINER ID   IMAGE                                     COMMAND                  CREATED        STATUS              PORTS
a2a50d1f8de1   react-rust-postgres-sample-project-backend   "cargo run --offline"   5 seconds ago   Restarting (101)   Less than a second ago
45034bb58a16   react-rust-postgres-sample-project-backend-1   "docker-entrypoint.s..." 6 seconds ago   Up 4 seconds       0.0.0.0:5
433->5432/tcp   postgres:17-alpine                             "docker-entrypoint.s..." 6 seconds ago   Up 4 seconds       0.0.0.0:3
75242d42662d   react-rust-postgres-sample-project-frontent   "docker-entrypoint.s..." 6 seconds ago   Up 4 seconds       0.0.0.0:3
000->3000/tcp   react-rust-postgres-sample-project-frontent-1   "docker-entrypoint.s..." 6 seconds ago   Up 4 seconds       0.0.0.0:3
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

```

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

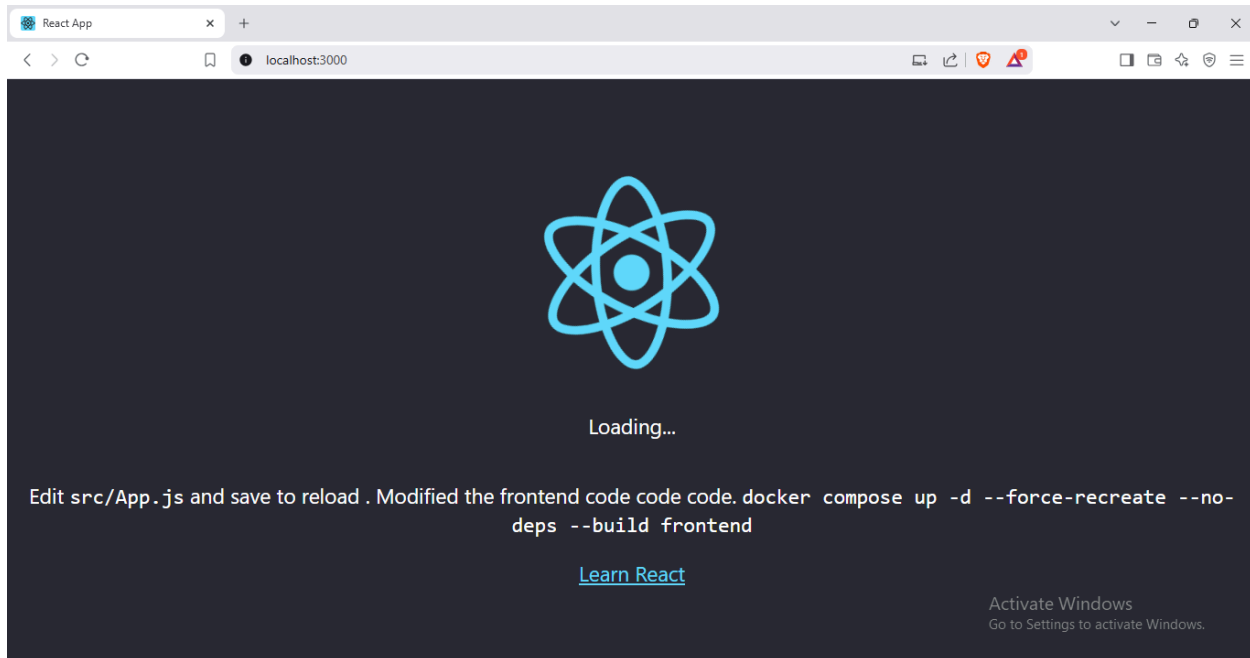
Suppose you modify the frontend code, then best practice is to rebuild and restart frontend service only, and vice versa

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

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\Docker\Lab-3\compose-demo\react-rust-postgres-sample-project> docker compose up -d --force-recreate --no-deps --build frontend
[+] Building 4.4s (16/16) FINISHED
=> [frontend internal] load build definition from Dockerfile
=> => transferring dockerfile: 735B
=> [frontend] resolve image config for docker-image://docker.io/docker/dockerfile:1.4
=> [frontend auth] docker/dockerfile:pull token for registry-1.docker.io
=> CACHED [frontend] docker-image://docker.io/docker/dockerfile:1.4@sha256:9ba7531bd80fb0a858632727cf7a112fbfd19b17e94c4e84ced81e24ef1a0dbc
=> => resolve docker.io/docker/dockerfile:1.4@sha256:9ba7531bd80fb0a858632727cf7a112fbfd19b17e94c4e84ced81e24ef1a0dbc
=> [frontend internal] load .dockerignore
=> => transferring context: 61B
=> [frontend internal] load metadata for docker.io/library/node:1ts
=> [frontend auth] library/node:pull token for registry-1.docker.io
=> [frontend development 1/6] FROM docker.io/library/node:1ts@sha256:f6b9c31ace05502dd98ef777aaa20464362435dcc5e312b0e213121dcf7d8b95
=> => resolve docker.io/library/node:1ts@sha256:f6b9c31ace05502dd98ef777aaa20464362435dcc5e312b0e213121dcf7d8b95
=> [frontend internal] load build context
=> => transferring context: 1.81kB
=> CACHED [frontend development 2/6] WORKDIR /code
=> CACHED [frontend development 3/6] COPY package.json /code/package.json
=> CACHED [frontend development 4/6] COPY package-lock.json /code/package-lock.json
=> CACHED [frontend development 5/6] RUN npm ci
=> [frontend development 6/6] COPY . /code
=> [frontend] exporting to image
=> => exporting layers
=> => exporting manifest sha256:b3449b0d7232db6aa900e042553fd8bfefae69129930e84f3dea4e2c425c2d38
=> => exporting config sha256:e3866f6d553b35a8b72a89bfbf006014b32bca6a0b36872bed5aa333cc483817
=> => exporting attestation manifest sha256:c96477b00e8daad18e57b1d133da5493175ee09d47a623a1476ae2ec5ab47184

```



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
```

```
PS D:\Docker\Lab-3\compose-demo\react-rust-postgres-sample-project> docker system prune -af --volumes
Deleted Images:
untagged: react-rust-postgres-sample-project-backend:latest
deleted: sha256:9f8544b821c78b49e67f0f3e65120a5b8e6c32735e6534f187ff917551e87a48
deleted: sha256:0948d975f946ed288e04d4c63d8e5f79c0f462b69559a183c2081587d2ff5c92
deleted: sha256:b36a80cedb822dc88a10463b716e8359206343d31751921d81ce8f4e3018ad33
deleted: sha256:36a2bc7fa9cf8438c75a989bc52243b03e7becbb3a1cdd2dca616136b9f749b1
deleted: sha256:2903ff1e242d216ca5cd62b3f9fec200f33da8352c9539271e8908091839351e
deleted: sha256:5a59ea7b327f52bd8408c511e199ea555accae943a4e9b7a9e407180537432d3
untagged: react-rust-postgres-sample-project-frontend:latest
deleted: sha256:e9ccb97effbc683887ed4cd510245ddd42177e64acf860245ead5da8bd62fbcf
deleted: sha256:b3449b0d7232db6aa900e042553fd8bfe9ae69129930e84f3dea4e2c425c2d38
deleted: sha256:e3866f6d553b35a8b72a89bfbf006014b32bca6a0b36872bed5aa333cc483817
deleted: sha256:c96477b00e8daad18e57b1d133da5493175ee09d47a623a1476ae2ec5ab47184
deleted: sha256:1a5816996a1ccc3331afbe38cfe2d0a7d973df9719b28275503579d3d27
deleted: sha256:7490f2d4a6c346a2a0816b715ee9f71779c184c887524773feb6c18df72d6c
untagged: postgres:17-alpine
deleted: sha256:7062a2109c4b51f3c792c7ea01e83ed12ef9a980886e3b3d380a7d2e5f6ce3f5
deleted: sha256:0ae605e3d11c7cc82cbcd8f3e506233f18cdd40e3fc7622893f6a4d0772a5a09
deleted: sha256:0150e3200277b907866758e22a3b9bba7857325b994848fe9f44b3610c3ae17e
deleted: sha256:f18232174bc91741fdf3da96d85011092101a032a93a388b79e99e69c2d5c870
deleted: sha256:d8d8fb695a5a8cfff4a027215fd2265eb1f1cf21734370d46c5451d4966915383
deleted: sha256:c24c1ba610df1db9a18866f4461cc9dfd95d09cfff58be8a1b5446048dec5b8b7
deleted: sha256:83efd74bc97e708b3051f0dfe088ebdcfe92a57a16a4f30eba1ab7e6398f8a6

Activate Windows
Go to Settings to activate Windows.
```

You've finished the lab!!