COSC349 Assignment 1: Virtualisation with Docker

James Robiony-Rogers (5793901) & Corban Surtees (4658948)

September 9, 2024

Contents

1 Introduction

This report details the design, development, and deployment of a job application tracker using modern cloud-ready development practices. The Job Application Tracker is a web-based application aimed at students and job seekers, allowing them to efficiently manage their job search by recording key information about their applications, such as job roles, companies, statuses, and dates of submission. The project demonstrates the use of virtualisation technologies to achieve a portable and distributed software application, operating across three containers through Docker and Docker Compose. This setup showcases the power of containerisation, ensuring that the application is easily deployable on various host systems.

The technologies utilised in this project include:

- Frontend: React (with TypeScript) for the user interface.
- Backend: Python Flask REST API to handle business logic and interact with the database.
- Database: Supabase, a cloud-based backend platform built on PostgreSQL, used for data storage and interaction.

2 Application Design and Virtual Machines

System Architecture

The Job Application Tracker follows a three-tire architecture:

1. Frontend: A React-based web interface styled with TailwindCSS where users record and view details of job applications they have applied to.
2. Backend: A Flask-based REST API that manages business logic and handles HTTP requests from the frontend. It interacts with Supabase for data management.
3. Database: Supabase is used for persistent storage of user data, including job applications and profile information.

Container Configuration

The application leverages virtualisation through the use of Docker containers. The frontend, backend, and Supabase database are each encapsulated in their own containers.

While the frontend and backend run in separate Docker containers, supabase pulls its own Docker containers (for the database and other services) and starts them using its own Docker Compose file.

To package the application, the project uses Docker Compose to orchestrate these containers. We utilise the `include` command in Docker Compose to integrate Supabase's Docker Compose file with the main one, enabling all containers to start together.

Justifying our Approach

Using separate containers for each tier offers several advantages:

Scalability: Each tier can be scaled independently based on demand. For example the frontend container could be scaled horizontally to handle more concurrent users.

Isolation: Keeping the frontend, backend, and database separate ensures that issues or updates in one tier do not affect the others. This separation also improves security, as each service runs in its own isolated environment.

Portability: Docker ensures that the application can run consistently across different environments, from development to production, making it easier to deploy on various platforms or cloud providers.

3 Automated Build and Deployment Process

The project is designed to be easily built and deployed using Docker and Docker Compose

Docker Setup

The Dockerfile for the frontend is configured to install Node.js and all of the projects dependencies, build the React application and serve it on port 80 using Nginx.

The Dockerfile for the backend installs Python and the necessary packages, then runs the Flask application.

Supabase pulls its required containers and services (PostgreSQL, authentication, etc.) from the internet. The main Docker Compose file starts both the frontend and backend containers and integrates the Supabase containers into the application stack.

To demonstrate the functionality of the application, a separate Docker container is used to populate the database with test data. This ensures that when the application is started, it already contains sample job applications and users, allowing for immediate testing and demonstration.

Build Process

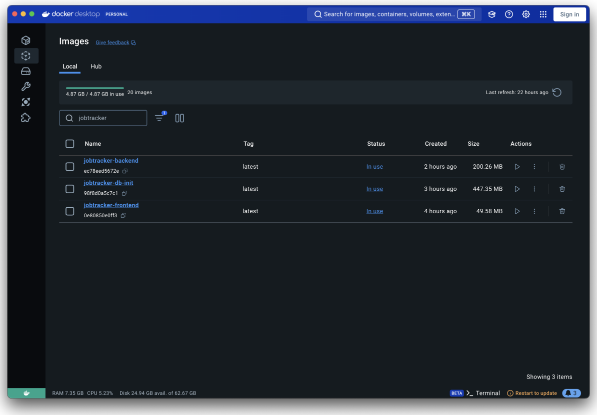
The build process is automated and only requires the docker compose build and up commands to be run. This will download the necessary containers, build, and start the application.

For a first-time build, the following components are downloaded:

Container	Dependencies	Approximate Image Size
Frontend	Node.js, Nginx	50 MB
Backend	Python, Flask	200 MB
Populate Database	PostgreSQL	450 MB
Database	All Supabase services	4.27 GB

Table 1: Approximate image sizes for each container

For subsequent builds, nothing extra is downloaded, as the images are cached locally. This makes the build process faster and more efficient.



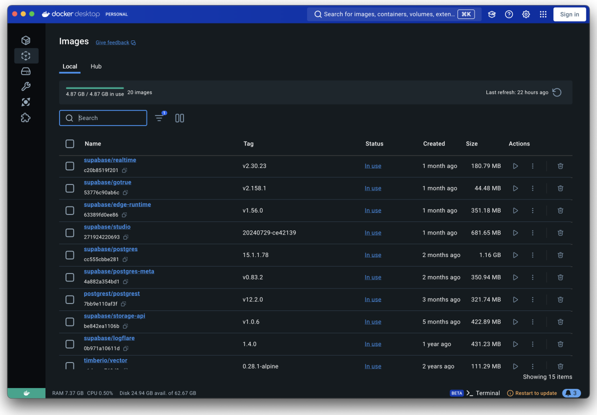
4.87 GB / 4.87 GB in use 20 images   Last refresh: 22 hours ago

Name	Tag	Status	Created	Size
jobtracker-backend ec78eed5672e	latest	In use	2 hours ago	200.26 MB
jobtracker-db-init 98f8d0a5c7c1	latest	In use	3 hours ago	447.35 MB
jobtracker-frontend 0e80850e0ff3	latest	In use	4 hours ago	49.58 MB

Showing 3 items

RAM 7.35 GB CPU 5.23% Disk 24.94 GB avail. of 62.67 GB

Docker images for frontend, backend & populate database



4.87 GB / 4.87 GB in use 20 images   Last refresh: 22 hours ago

Name	Tag	Status	Created	Size
supabase/realtime c20b8519f201	v2.30.23	In use	1 month ago	180.79 MB
supabase/gotrue 53776c90ab6c	v2.158.1	In use	1 month ago	44.48 MB
supabase/edge-runtime 63389fd0ee86	v1.56.0	In use	1 month ago	351.18 MB
supabase/studio 271924220693	20240729-ce42139	In use	1 month ago	681.65 MB
supabase/postgres cc555cbbe281	15.1.1.78	In use	2 months ago	1.16 GB
supabase/postgres-meta 4a882a354bd1	v0.83.2	In use	2 months ago	350.94 MB
postgrest/postgrest 7bb9e110af3f	v12.2.0	In use	3 months ago	321.74 MB
supabase/storage-api be842ea1106b	v1.0.6	In use	5 months ago	422.89 MB
supabase/logflare 0b971a10611d	1.4.0	In use	1 year ago	431.23 MB
timberio/vector	0.28.1-alpine	In use	2 years ago	111.29 MB

Showing 15 items

RAM 7.37 GB CPU 0.50% Disk 24.94 GB avail. of 62.67 GB

Docker images for all supabase services

4 Suggested Improvements

4.1 Secure Authentication

The current implementation uses basic authentication for user management. A key improvement would be to implement more secure authentication mechanisms such as OAuth or JWT. This would enhance the security of the application, making it suitable for production use where sensitive user data like resumes and cover letters are involved.

After implementing secure authentication and testing in development mode, the application can be rebuilt and rerun using the same Docker Compose commands outlined in the project's README file.

4.2 Supabase Object Storage

Adding Supabase Object Storage would allow users to store and manage additional documents, such as resumes, cover letters, or application receipts. This would extend the functionality of the Job Application Tracker, making it a more comprehensive solution for job seekers.

Once again rebuilding and rerunning the application simply requires running the Docker Compose commands.

5 Attribution

This project was developed by James Robiony-Rogers and Corban Surtees for COSC349 at the University of Otago. The project would not have been made possible without various open-source projects and libraries that we have used. These include:

- Frontend: TailwindCSS components `https://tailwindui.com/components`
- Frontend: Material Tailwind `https://material-tailwind.com/`
- Frontend: Preline UI `https://preline.co`
- Backend: Flask `https://flask.palletsprojects.com/en/2.0.x/`
- Database: Supabase `https://supabase.io/`

6 Project Links

Below are links to the project demo video and the GitHub repository:

- Repository: `https://github.com/JamesRobionyRogers/Assignment01-VirtualisingSoftware/`
- Project demo video: `https://youtu.be/ISDUj34Jlh0`