

Containerization As A Tool For Learning

Henning W.

May 29, 2021

1 Introduction

Choice of application: Some kind of social media platform thing. Due to large time constraints, some suboptimal solutions will have to be taken. I usually would start off the project by making CI/CD with a server. These factors have been discarded entirely. This means, that there won't be any online capabilities in this project. Instead the entire application will be containerized, in an attempt at making the local solution easier to run. Another suboptimal solution is, that the database containers and the containers which utilize them have been split into two separate container files. You'd usually want both of them to share the same docker-compose file. The problem lies with migration strategies, and what I like calling "A chicken or egg" complication. Both data processing and my backend are reliant on the database. Further more, my backend is reliant on the data-processing. Because of this, the containers will need to be run in chunks. The standard solution to this is somewhat suboptimal in itself. It involves [files which lets other processes run first](#) before executing itself.

Since the project requires a larger amount of data, the project has been created a bit backwards. I've looked at some datasets, and then slowly formulated data models from them.

I'm using a Rust backend, since it's a solo project, and I already have a setup for it. It's effective and stable. It's a role I don't trust Python all that much with.

The Python container for data processing is only run once. Its purpose is to populate the databases with some dummy data. The data is acquired from several sources, and translated via. the Python library Pandas, and then subsequently formatted and populated into Mongo. From there, the data will once again be formatted and populated into Meilisearch. The reason why the Python container isn't populating the Postgres database as well, is because it'd mean, that both the Rust backend and the Python data processing container would have to have the complete ORM set up. I'd also need the two ORMs be completely alike. The other option is to use mongo inside the Rust container, but only have to create 1 ORM with Postgres setup. I chose the latter.

2 Technologies

2.1 Databases

2.1.1 PostgreSQL

2.1.2 MongoDB

2.1.3 Redis

2.1.4 Meilisearch

2.2 Backend

2.3 Data-Population

2.4 Frontend

3 Structure

4 Goals

This section is dedicated to the following assignment definition question:

Define functional and non-functional requirements to your project

4.1 Implemented

-

4.2 Planned

- Gathering of datasets
- Create diagrams for database
- data_population: Processing of datasets to usable format
- data_population: Datasets inserted into Mongo with Cython
- data_population: connection tests
- data_population: logging
- backend: Postgres, Redis, Mongo connectors and tests
- backend: Generate Postgres entries from Mongo database
- backend: Redis caching
- frontend: Basic site setup with routes
- frontend: Meilisearch functionality implemented
- frontend: Interaction with Redis routes.

4.3 Stretch Goals

- data_population: documentation with Sphinx
- backend: Documentation rustdocs

4.4 Scrapped

- Apply solution to Digital Ocean droplet
 - Create and setup server on digital ocean
 - docker-compose deployment version
 - CI with github actions.
 - github actions secrets with docker-compose
 - SSH SCP appleboy execution of docker-compose solution
 - nginx

5 Installation