<u>Task 2</u>:

We first start by creating each of the individual dockerfiles for each of the services (*Generator*, *API*, *Loader* and *Transformer*). In each dockerfile, we have:

- **`FROM python:3.13.1`** → *Defines and creates the base image, with Python 3.13.1.*
- **`WORKDIR /app`** → *Sets the directory to the /app folder for the service's container*
- **`COPY . .`** → *Copies all files from the current service folder into its respective /app folder*
- **`RUN pip install -r requirements.txt`** → *Installs all required dependencies for that service's app (app.py), found in the requirements.txt file in the service's folder.*
- **`CMD ["python", "app.py"]`** → *Runs for the given service once the multicontainer has been created.*

We then run `docker build -t [service] .` for each of the services to create an image for each.

In the `docker-compose.yml` file, we create each of our containers for each service under "`services:`", where we have our generator, transformer, loader, and api services. The `volumes` at the bottom define the docker volumes for the raw data, processed data, and databases, which persist previous data even if the multicontainer is deleted. For each service, we first build the image of the service using "`build:`", then we mount each of the volumes to the specified service container and folder location in "`volumes:`", and to maintain the correct pipeline execution order (*Generator → Transformer → Loader → API*), we specify for the transformer, loader, api services that the previous service must be run first before running the current service using `depends_on` and specifying the required previous service (e.g. loader depends_on transformer), and the API service also has `ports` which when running the api, exposes the API service on port 5000.

We can then create our multicontainer using `docker compose up`, which also runs each of the services (in proper order) and allows us to access the API results via *http://localhost:5000*