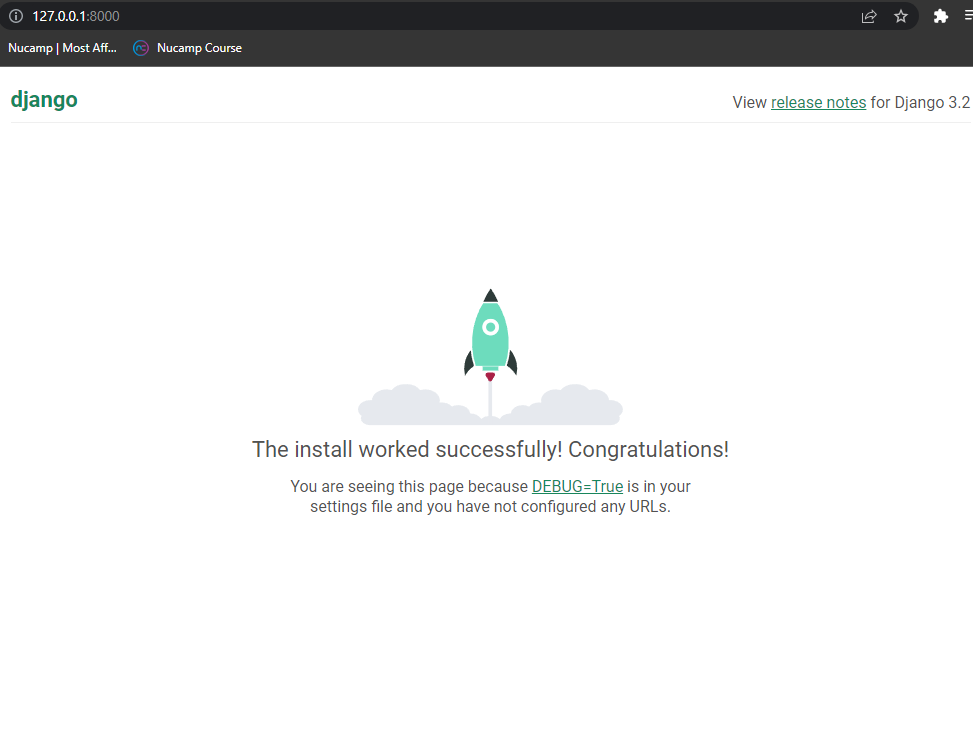
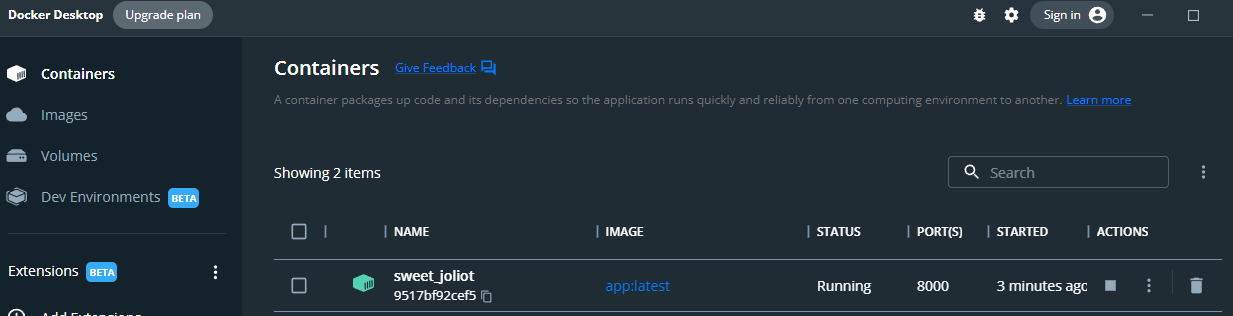
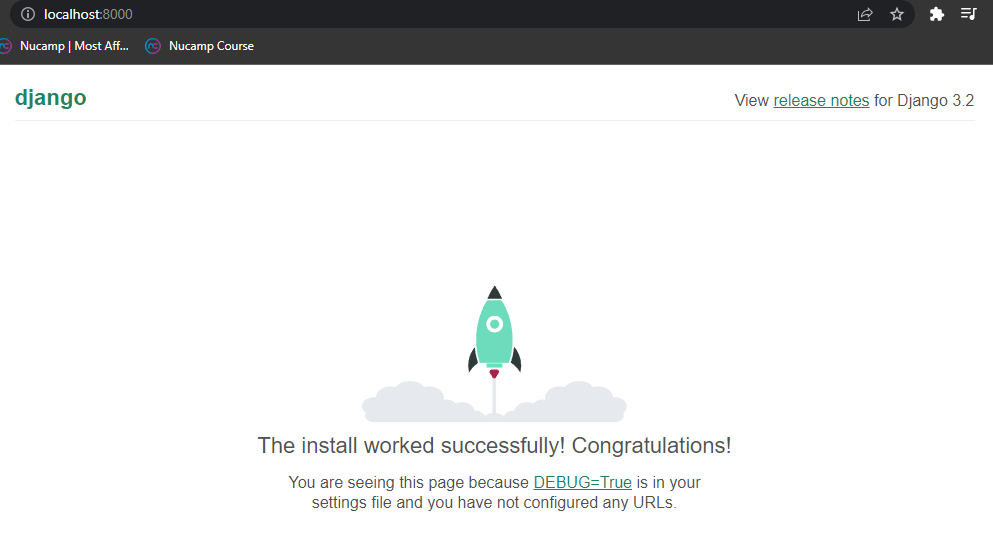
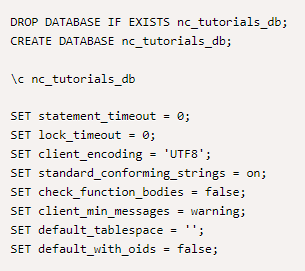
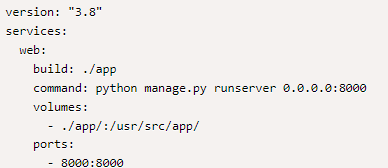
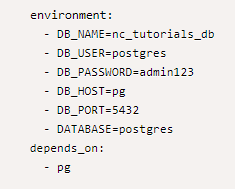
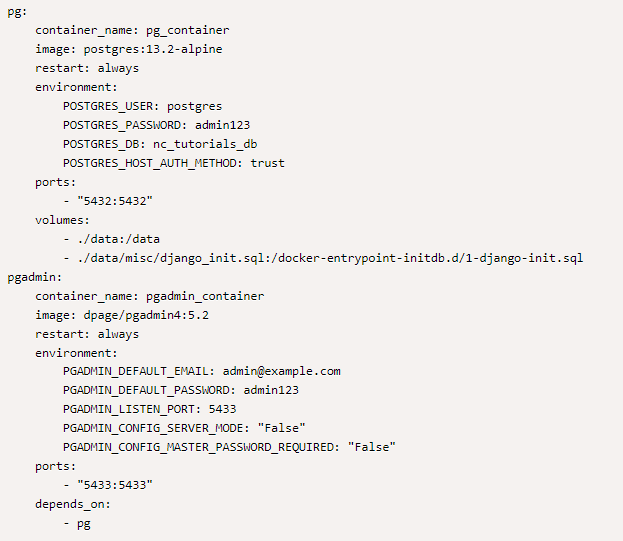
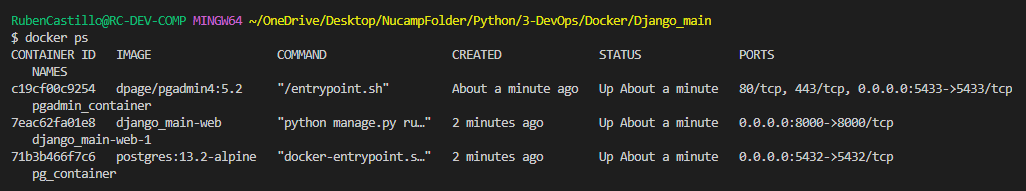
Project Scope: To create a Django application and deploy it to Docker, then to Docker compose.

1. Environment setup
   1. Created a virtual environment
      1. Cmd to create: python -m venv venv
      2. Cmd to start: . venv/scripts/activate
2. Create an app folder with a requirements.txt that has the following
   * 1. django==3.2.2
     2. gunicorn==20.0.4
     3. djangorestframework==3.12.4
     4. python-decouple==3.4
     5. psycopg2-binary==2.9.1
3. Install packages while in the app folder:
   1. CMD: python -m pip install -r requirements.txt
4. Use django\_admin to scaffold out the django project
   1. CMD: django-admin startproject nc\_tutorials .
5. Test the development server by commenting out the following:
6. To start the Django Development server run the following command:
   1. CMD: python manage.py runserver 8000
7. Open web browser to <http://127.0.0.1:8000/>
8. 
9. Create a docker file while in the App folder
   1. Command palette: Add Docker Files to Workspace
   2. Select application platform: Python:Django
   3. Choose the apps entry point: manage.py
   4. Port to listen to: 8000
   5. Include option docker compose file? No
   6. Two files will be generated .dockignore and Dockerfile to build docker image
10. In Dockerfile find the very last instruction that binds the Gunicorn server to wsgi file.
    1. CMD ["gunicorn", "--bind", "0.0.0.0:8000", "nc\_tutorials.wsgi"]
    2. Gunicorn is typically used as a webserver that routes HTTP requests and responses for Django app. Gunicorn is also known as WSGI server.
11. Build the docker image then run it on the container
    1. From the folder that has the Dockerfile run the following
    2. Command palette: Docker images: Build image
    3. Select image group: hellodjango
    4. Select image: latest
    5. 
    6. Confirmed the Django web app can be viewed at [http://localhost:8000](http://localhost:8000/).
    7. 
    8. Configure Postgres database and implement Docker Compose to use for Postgress and pgAdmin containers
       1. Create a folder in root directory called data then create subfolder named misc.
       2. Then in subfolder create django\_init.sql file and copy the following:
          1. 
       3. Next, create a docker-compose.yml in the root directory. Add the following:
          1. 
          2. Add the following to the web service of the .yml file
          3. 
          4. Add the following to the .yml file such that the lines pg and pg admin are at the same leve of the web service
          5. 
          6. Next, update the docker file such that the RUN adduser, USER appuser, and CMD are commented out. No longer needed.
          7. Use docker compoe to bring up the containers defined in the docker-compose.yml together and run them in detach mode.
             1. CMD: docker compose up -d
             2. Confirm that the containers are running by command line and postgres
             3. 
             4. 