Notes Taken By

- Muhammad Raaid Khan
- Data Science and AI (Batch - 05)
- NED - CCEE

Docker

Docker is an open source platform that enables developers to build, deploy, run, update and manage containers — i.e. standardized, executable components that combine application source code with the operating system (OS) libraries and dependencies required to run that code in any environment.

- **Docker Image:** A reusable, shareable file used to create containers. A blueprint of your Container.
- **Docker Container:** A runtime instance; a self-contained software. Created from an Image.

Docker Commands

- `docker version`
- `docker run hello-world`
    - Will download a Dummy Image to test proper Docker Installation
- Pull Anaconda Image
    - docker pull continuumio/anaconda3
- `docker images`
    - All images created in Docker
- `docker run -it continuumio/anaconda3:latest /bin/bash`
    - `-it` Run docker in *Interactive Mode*
    - Name of Image and its version (latest)
    - Run terminal in root directory (/bin/bash)
- `CTRL+PQ`
    - Exit container without terminating it.
- `docker container ls`
    - Details of all running containers
- `docker exec -it <container_name/container_id> bash`
    - Go into a Docker container that is already running. [Will not make another container]
- `docker ps`

- List All Running Containers

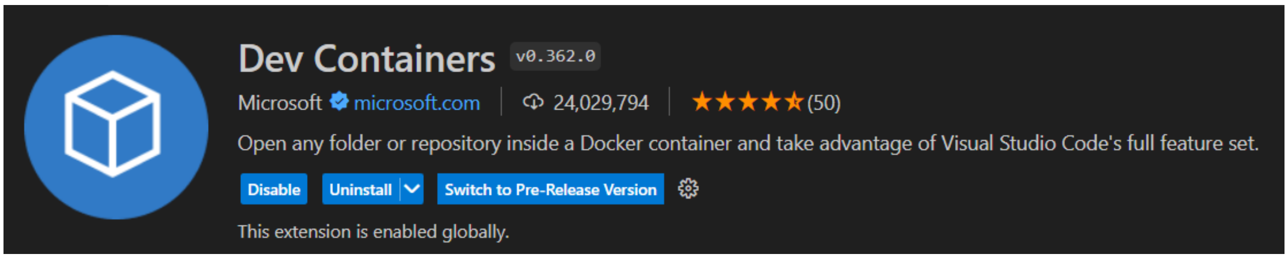
- `docker ps a`
  - List the Containers, even those that are in stopped state
- `docker stop <container_name/container_id>`
  - Stop container
- `docker rm container_name`
  - Deleting a Docker Container. [Container must be stopped first]

Linux Flags

- Single Dash (-) > Each letter will be considered as separate command
  - `- it` (i and t are separate commands)
- Double Dash (--) > Complete Word will be considered as a command
  - `--reload`

Dev Container Extension

- Install following extension in VsCode


Dev Containers v0.362.0
Microsoft microsoft.com | 24,029,794 | ★★★★★ (50)
Open any folder or repository inside a Docker container and take advantage of Visual Studio Code's full feature set.

This extension is enabled globally.

- This will show all containers, you can explore file system of Container as well
- This will allow you to code in VsCode that will execute in Docker Container.

Dockerfile

```
# Use an official Python runtime as a parent image
FROM python:3.12

LABEL maintainer="ameen-alam"
# Set the working directory in the container [Folder is created]
WORKDIR /code
# Install system dependencies required for potential Python packages
RUN apt-get update && apt-get install -y \
    build-essential \
    libpq-dev \
    && rm -rf /var/lib/apt/lists/*

# Install Poetry
```

```
RUN pip install poetry

# Copy the current directory contents into the container at /code
COPY . /code/

# Configuration to avoid creating virtual environments inside the Docker container
RUN poetry config virtualenvs.create false

# Install dependencies including development ones
RUN poetry install

# Make port 8000 available to the world outside this container
EXPOSE 8000

# Run the app. CMD can be overridden when starting the container.
# Command is passed inside and array. [Separated by Spaces]
CMD ["poetry", "run", "uvicorn", "app.main:app", "--host", "0.0.0.0", "--reload"]
```

Building a Docker Image

- Building from a Simple Dockerfile

```
docker build -t my-image .
```

- Building from .dev or .prod Dockerfile

```
docker build -f Dockerfile.dev -t my-image .
```

Running a Docker Container from Image

```
docker run -d --name container-1 -p 8000:8000 my-image
```

- `-d` = detach [Container will run in background but you will remain in terminal of base operating system]
- `8000:8000` = Expose port 8000 of Container to port 8000 of Host Computer.

Databases

Data Sanitization

Making sure that input Query does not consist of any Harmful material that can damage the Database and its Data

- SQL Alchemy

Data Validation

Making sure that Data in Query consist of only valid data and is following Schema of Database

- Pydantic

SQL Model

*A Python Package that performs both **Data Sanitization** and **Data Validation***

*SQLModel is an **ORM (Object Relational Mapper)** that converts Objects of OOP to that of Objects of Databases*

Types of Database

- Structured Database (SQL)
    - PostgreSQL
    - MySQL
- Non-Structured Database (NoSQL)
    - Key Value
    - Column Based
    - Document Based
    - Graph Databases

PostgreSQL

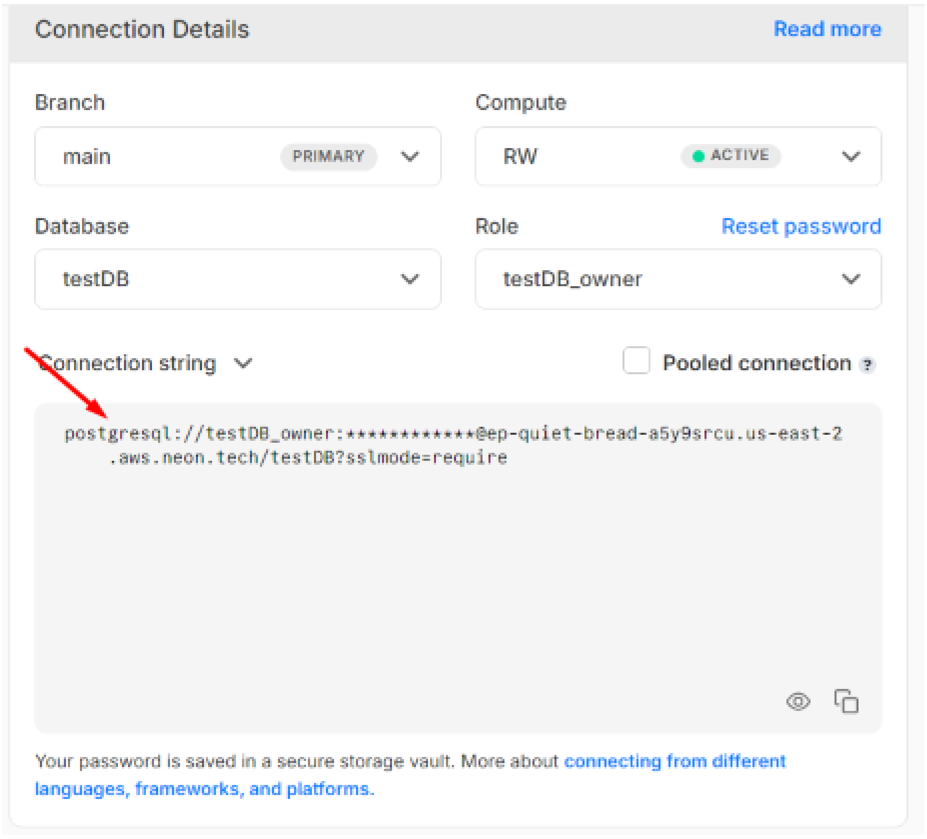
- Free
- Open Source
- Distributed

Working with SQL Model

- Go to `neon.tech` and create account.

- Create a `Test` Database

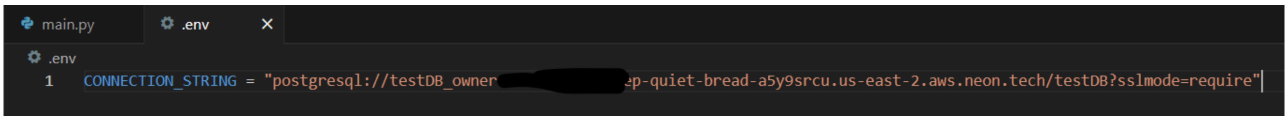
- Go to Connectivity of Database in Dashboard and copy Connection String



```
postgresql://testDB_owner:************@ep-quiet-bread-a5y9srcu.us-east-2
    .aws.neon.tech/testDB?sslmode=require
```

Your password is saved in a secure storage vault. More about connecting from different languages, frameworks, and platforms.

- Create a Poetry Project.
- Create a `.env` file in root of project.
    - Add this .env file to `.gitignore`
- Save your credentials in this `.env` file.



```
CONNECTION_STRING = "postgresql://testDB_owner[redacted]ep-quiet-bread-a5y9srcu.us-east-2.aws.neon.tech/testDB?sslmode=require"
```

- You can retrieve this secret into your program vai Environment Variables using following code:

```
from dotenv import load_dotenv, find_dotenv
import os

#read .env file and load into environment
_ : bool = load_dotenv(find_dotenv())

# Get Secret value from you OS Environment
conn_string = os.environ.get("CONNECTION_STRING")

print(conn_string)
```

- Import SQLModel

```python
from sqlmodel import Field, SQLModel, create_engine
```

- Create an Object of SQLModel

```python
# A Table with Name Hero will be created in the Database
class Hero(SQLModel, table=True):
    id: int | None = Field(default=None, primary_key=True)
    name: str
    secret_name: str
    age: int | None = None
```

  - `table=True` means that object will of Pydantic + SQLAlchemy
  - `table=False` means that object will be of Pydantic only.

- Create the Database Engine to establish DB Connectivity.

```python
engine = create_engine(conn_string, echo=True)
```

  - `echo = True` will show underlying SQL Queries

- Create a Table of your Object with Following code

```python
SQLModel.metadata.create_all(engine)
```

- Create objects of you Hero Class

```python
hero_1 = Hero()
hero_1.name = "Deadpond"
hero_1.secret_name = "Dive Wilson"
hero_1.age = 48

hero_2 = Hero(name="Spider-Boy", secret_name="Pedro Parqueador")
hero_3 = Hero(name="Rusty-Man", secret_name="Tommy Sharp", age=48)
```

- Create a Session, add Objects to DB and Commit Changes to DB.

```python
session = Session(engine)

session.add(hero_1)
session.add(hero_2)
session.add(hero_3)
session.commit()
```

- Retrieve Data from Database

```
with Session(engine) as session:
statement = select(Hero)
results = session.exec(statement)
for hero in results:
    print(hero)
```

  - This will retrieve all Data from the Hero table in our Database