RICE UNIVERSITY

COMP543 Lab 2: Connecting Postgres and Jupyter

COMP543 Instructors

Last edited: May 13, 2019

The first two assignments will be done in SQL using Postgres. Today we'll be running through some examples of accessing SQL databases using Jupyter Notebooks. If you were successful in setting up Docker in the previous lab, then today should be very straightforward, otherwise you might have a bit of installation to do.

1 Background & Setup

1.1 Docker Compose

This section applies to you if you plan on using Docker for running Postgres and Jupyter during this class (most of you).

In the last lab, we introduced you to Docker and had you run some basic commands to get a Jupyter Notebook server running. Today, rather than executing several commands at the command line to get your environment set up, we're going to use a popular tool distributed with Docker called Docker Compose. The basic idea is that you define your environment in a text file rather than on the command line. This allows us to share environments as easy as we can share a text file! See below for an example that sets up a Postgres container and a Jupyter notebook container:

```
version: "3"
services:
  jupyter:
    image: gvacaliuc/db-notebook
    ports:
      - "8888:8888"
    volumes:
      - "./notebooks:/home/jovyan/notebooks"
      - "./data:/home/jovyan/data"
```

```
postgres:
  image: postgres:alpine
  restart: always
  environment:
    POSTGRES_USER: dbuser
    POSTGRES_PASSWORD: comp543isgreat
    POSTGRES_DB: comp543
  volumes:
    - "./data:/data"
```

This file format is known as YAML and is sometimes described as a "Pythonic-JSON". We have provided this file for you.

The section under `jupyter` creates a container running the `gvacaliuc/db-notebook` container, binds port `8888` on the host to port `8888` on the container, and shares two volumes with the container.

The section under `postgres` creates a container running the `postgres:alpine`[1] image, with a default hostname of `postgres`, and sets some environment variables that determine how the container should build the database / how users will access it. It also shares a data volume with the container.

To actually start these containers, one would navigate to the directory containing this file and simply run `docker-compose up`. No more long command line arguments to docker!

1.1.1 Absolute vs. Relative Paths

Some of you may notice that we used relative paths in the compose file (`./notebooks:/home/jovyan/notebooks`) to declare the volumes to share, which is not allowed when specifying the host mount path from the regular docker CLI (`docker run -v /home/user/notebooks:/home/jovyan/notebooks image`). This restriction is relaxed in compose, allowing us to more conveniently create host mounts.

1.2 Manual Installation

This section applies to you if were unable to get Docker set up in the last lab.

To follow along with this lab, you'll need to install Postgres on your computer as well as Jupyter and some related Python packages. If you do not have a Python installation on your computer yet, it is recommended you install the Anaconda distribution.

1.2.1 Postgres

Install using the instructions listed here: http://postgresguide.com/setup/install.html.

- Windows: https://www.postgresql.org/download/windows/
- Mac: https://postgresapp.com/
- Linux: See guide linked above.

[1] Note that the image syntax is generally `user/repository[:tag]`, but for useful and common programs such as Postgres, Docker, Inc. maintains a set of images that can be accessed by their name.

1.2.2 Jupyter / Python

It's recommended to use the Anaconda Distribution of Python 3.6: https://www.anaconda.com/download.

1.2.3 Python Packages

You'll need to install the following packages:

```
jupyter
pandas
numpy
scipy
ipython-sql
sqlalchemy
psycopg2
```

2 Exercises

The data and notebooks for this lab are available as a zip file on canvas linked to the lab assignment. Extract it and navigate to that directory.

2.1 Start Postgres and Jupyter

Make sure you do not already have Jupyter running on your machine. If you do, either shut it down or change the port mapping specified in the .yml file.

If you're not using Docker, please start Postgres and Jupyter as you normally would and then open the Jupyter notebook provided in the lab zip file.

Otherwise, navigate to the base of the zip file (in the terminal environment you run Docker from) and execute

```
docker-compose up
```

You should see some pretty output about `postgres` starting and some about `jupyter` starting. Open the URL that's printed from Jupyter and open the notebook interface.

2.2 Connect to Postgres via `psql`

Before we begin writing SQL queries, we need to load some data into our database. We'll do this by connecting to our database engine via the command line utility `psql`, although this can also be done through Jupyter, or via a GUI interface.

2.2.1 Connecting with Docker

Your `username/password` pair is `dbuser/comp543isgreat`. From the base of the lab folder (with the `docker-compose.yml` file), execute:

```
docker-compose exec postgres psql -d comp543 -U dbuser
```

You may be prompted for a password. On a successful connection, you should see output this:

```
psql (10.1)
Type "help" for help.

comp543=#
```

2.2.2 Connecting Manually

You'll want to follow the documentation from Postgres to use psql. On Mac / Linux you might be able to use `man psql` for help connecting. If you're having trouble getting connected, please grab a TA for help.

2.3 Adding data to Postgres

Since we're working in a fresh database, we shouldn't have any tables yet. We can make sure of this using the `\d` meta-command:

```
psql (10.1)
Type "help" for help.

comp543=# \d
Did not find any relations.
```

Let's create two new tables:

- `cypher(letter CHAR, value INTEGER)`
- `message(seq INTEGER, value INTEGER)`

```
comp543=# CREATE TABLE cypher(
  letter CHAR,
  value INTEGER
);
comp543=# CREATE TABLE message(
  seq INTEGER,
  value INTEGER
);
```

If you made a mistake and need to reload or recreate these tables, you can easily remove them from the database with the commands

```
comp543=# DROP TABLE cypher;
comp543=# DROP TABLE message;
```

Then load up the provided cypher data. Note that if you're using docker, you're actually executing the psql command on the `postgres` container. Where are the data files locally, and where do they end up in the container? You'll have to look at the `docker-compose.yml` file and the provided data files to figure out the correct command below. If you're not using docker you'll still have to figure out the correct path and delimiter.

Use the psql COPY command to load data from the cypher.txt and secretmessage.txt files into their corresponding tables. The syntax for this command is:

```
comp543=# \COPY cypher FROM '<path to cypher data>' DELIMITER '<delimiter>' CSV;
comp543=# \COPY message FROM '<path to message data>' DELIMITER '<delimiter>' CSV;
```

Where `<path to cypher data>` is the location where the docker container can access the file which contains the file cypher.txt, `<path to message data>` is the location where the docker container the secretmessage.txt file, `<delimiter>` is the delimiter used in the provided data files. Note that the path to the files you need to specify is relative to the folder where you kicked off docker-compose. So, if you were in /Users/rbm2/Documents/543/lab2-postgres-jupyter, your path should be relative to that location.

You can quit `psql` using the meta-command `\quit`.

2.4 Connect to SQL and start Querying!

From here on out, follow the instructions in the provided notebook. Please submit a `.ipynb` file of your completed notebook on Canvas.

2.5 Shutting down your containers and cleaning up

To shutdown and remove your containers, type `docker-compose down` in the lab directory. Alternatively, one can type `docker-compose stop` followed `docker-compose rm`. Note that there is a notion of stopping containers, as well as removing containers. Stopping simply stops execution and preserves the filesystem, while removing actually removes all data related to the containers.