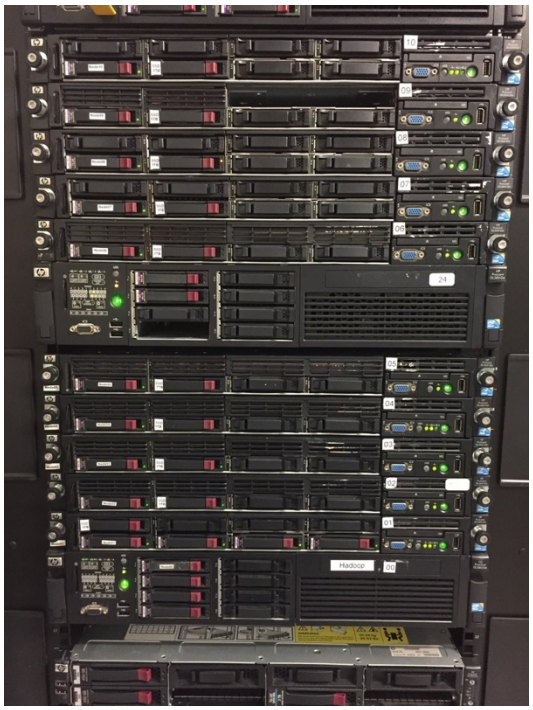
Hands-On Exercise 2-2
Building Docker images – PostgreSQL

1 CSCluster at UIS



11-node Hadoop nodes

- CPU: Intel Xeon **4C/8T** per node
- Memory: **266 GB** in total
  - Master (head) node: 74GB RAM
  - 2nd Master node: 48GB RAM
  - 9 worker nodes: 16GB/node
- Storage: **10 TB SSD** in total
  - Master node: 400GB,
  - 10 worker nodes: 1TB SSD/node
- Hadoop: **Cloudera CDH 6.3**
  - HDFS, MapReduce
  - Spark 2, HBase, Hive
  - Pig, HUE, and etc.

VMs for Docker/K8S (You will use these VMs)

- CPU: Intel Xeon **8** core/node
- Memory: **16GB** RAM/node
- Storage: **100GB**

3-node Cassandra cluster (NoSQL-Column family DB)

- CPU: Intel Xeon **4C/8T** per node
- Memory: **48GB** RAM (16GB/node)
- Storage: **3TB SSD** (1TB/node)

PostgreSQL node (Relational DB)

- CPU: Intel Xeon **4C/8T** per node
- Memory: **18 GB RAM**
- Storage: **1TB SSD**

MongoDB node (NoSQL-Document DB)

- CPU: Intel Xeon **4C/8T** per node
- Memory: **16 GB RAM**
- Storage: **1TB SSD**

2 Accessing your VM for Docker and Kubernetes

2.1 Accessing campus network

If you are in UIS campus, you are fine. If you are not in UIS campus, you should install a Cisco VPN client software. The VPN client software gives remote users a secure and encrypted VPN (Virtual Private Network) connection to the UIS campus network. Please see below website, https://www.uis.edu/informationtechnologyservices/connect/vpn/

2.2 Accessing your VM using SSH

After you install the VPN client software and make a VPN connection to UIS network, you can access your VM using a terminal (Mac), PowerShell (Windows), or Putty (Windows).

- Check the IP address for your VM in the 'Course Information' under 'Modules' in Canvas

Type below command in your preferred SSH shell client:

```
ssh your-login@10.92.128.36
```

your-login: your UIS NetID (for example, **slee675** from slee675@uis.edu)
Initial password & IPs: See the Virtual Machine IPs page in the 'Course Information' under 'Modules' in Canvas.

After you logged in to your VM, you will see below prompt. The 'us2004lts' is a name of your VM and stands for 'Ubuntu Sever 20.04 LTS' that we are using as an OS.

```
your-login@us2004lts:~$
```

Example1: using terminal in Mac

```
LeeMBP15:~ sslee777$ ssh sslee777@10.92.128.36
The authenticity of host '10.92.128.36 (10.92.128.36)' can't be established.
ECDSA key fingerprint is SHA256:dXhKfHsYIXe/53hvU+HOK2V6fVrTbz/QxmUhpnPXpzA.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.92.128.36' (ECDSA) to the list of known hosts.
sslee777@10.92.128.36's password: <== Use initial password until you change it
Welcome to Ubuntu 20.04 LTS (GNU/Linux 5.4.0-33-generic x86_64)
...

sslee777@us2004lts:~$
```

3 Dockerize PostgreSQL Database – Building a PostgreSQL image

Docker allows us to create lightweight, portable containers that can run any application easily.

Related Chapters in textbook

- Chapter 8: Containerizing an app
  - Containerizing an app - The deep dive
- Chapter 7: Containers

3.1 Creating a custom Dockerfile for PostgreSQL

To create a Docker image, we need to create a text file named Dockerfile and use the available commands and syntax to declare how the image will be built. We will use a Dockerfile from the Docker official site. https://docs.docker.com/engine/examples/postgresql_service/

The Dockerfile is shown below

```
#
# example Dockerfile for https://docs.docker.com/engine/examples/postgresql_service/
#

FROM ubuntu:16.04

# Add the PostgreSQL PGP key to verify their Debian packages.
# It should be the same key as https://www.postgresql.org/media/keys/ACCC4CF8.asc
RUN apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --recv-keys
B97B0AFCAA1A47F044F244A07FCC7D46ACCC4CF8

# Add PostgreSQL's repository. It contains the most recent stable release
#     of PostgreSQL, ``9.3``.
RUN echo "deb http://apt.postgresql.org/pub/repos/apt/ precise-pgdg main" >
/etc/apt/sources.list.d/pgdg.list

# Install ``python-software-properties``, ``software-properties-common`` and PostgreSQL 9.3
#  There are some warnings (in red) that show up during the build. You can hide
#  them by prefixing each apt-get statement with DEBIAN_FRONTEND=noninteractive
RUN apt-get update && apt-get install -y python-software-properties software-properties-common
postgresql-9.3 postgresql-client-9.3 postgresql-contrib-9.3

# Note: The official Debian and Ubuntu images automatically ``apt-get clean``
# after each ``apt-get``

# Run the rest of the commands as the ``postgres`` user created by the ``postgres-9.3``
package when it was ``apt-get installed``
USER postgres

# Create a PostgreSQL role named ``docker`` with ``docker`` as the password and
# then create a database `docker` owned by the ``docker`` role.
# Note: here we use ``&&\`` to run commands one after the other - the ``\``
#       allows the RUN command to span multiple lines.
RUN    /etc/init.d/postgresql start &&\
    psql --command "CREATE USER docker WITH SUPERUSER PASSWORD 'docker';" &&\
    createdb -O docker docker

# Adjust PostgreSQL configuration so that remote connections to the
# database are possible.
```

```
RUN echo "host all  all    0.0.0.0/0  md5" >> /etc/postgresql/9.3/main/pg_hba.conf

# And add ``listen_addresses`` to ``/etc/postgresql/9.3/main/postgresql.conf``
RUN echo "listen_addresses='*'" >> /etc/postgresql/9.3/main/postgresql.conf

# Expose the PostgreSQL port
EXPOSE 5432

# Add VOLUMEs to allow backup of config, logs and databases
VOLUME  ["/etc/postgresql", "/var/log/postgresql", "/var/lib/postgresql"]

# Set the default command to run when starting the container
CMD ["/usr/lib/postgresql/9.3/bin/postgres", "-D", "/var/lib/postgresql/9.3/main", "-c",
"config_file=/etc/postgresql/9.3/main/postgresql.conf"]
```

Let's look into the file. At the beginning of the file, we need to specify the base image we are going to use. We are using Ubuntu 16.04 as base image.

```
#
# example Dockerfile for https://docs.docker.com/engine/examples/postgresql_service/
#

FROM ubuntu:16.04
```

To verify their packages, we need to add PostgreSQL PGP key

```
# Add the PostgreSQL PGP key to verify their Debian packages.
# It should be the same key as https://www.postgresql.org/media/keys/ACCC4CF8.asc
RUN apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --recv-keys
B97B0AFCAA1A47F044F244A07FCC7D46ACCC4CF8
```

We also need to add PostgreSQL package repository.

```
# Add PostgreSQL's repository. It contains the most recent stable release
#     of PostgreSQL, ``9.3``.
RUN echo "deb http://apt.postgresql.org/pub/repos/apt/ precise-pgdg main" >
/etc/apt/sources.list.d/pgdg.list
```

We are installing version 9.3 of PostgreSQL, but instructions would be very similar for any other version of the database.

We need to update the packages available in Ubuntu and install PostgreSQL:

```
# Install ``python-software-properties``, ``software-properties-common`` and
PostgreSQL 9.3
#  There are some warnings (in red) that show up during the build. You can hide
#  them by prefixing each apt-get statement with DEBIAN_FRONTEND=noninteractive
RUN apt-get update && apt-get install -y python-software-properties software-
properties-common postgresql-9.3 postgresql-client-9.3 postgresql-contrib-9.3
```

Note: it's important to have apt-get update and apt-get install commands in the same RUN line, else they would be considered two different layers by Docker and in case an updated package is available it won't be installed when the image is rebuilt.

```
# Note: The official Debian and Ubuntu images automatically ``apt-get clean``
# after each ``apt-get``
```

At this point we switch to postgres user—default admin account for PostgreSQL— to execute the next commands:

```
# Run the rest of the commands as the ``postgres`` user created by the ``postgres-
9.3`` package when it was ``apt-get installed``
USER postgres

# Create a PostgreSQL role named ``docker`` with ``docker`` as the password and
# then create a database `docker` owned by the ``docker`` role.
# Note: here we use ``&&\`` to run commands one after the other - the ``\``
#       allows the RUN command to span multiple lines.
RUN    /etc/init.d/postgresql start &&\
    psql --command "CREATE USER docker WITH SUPERUSER PASSWORD 'docker';" &&\
    createdb -O docker docker
```

'/etc/init.d/postgresql start' is for starting postgresql daemon(service). Then run psql shell to create docker user with password 'docker'. Finally, we will create a databse, named docker and owner of the DB will be docker, see https://www.postgresql.org/docs/9.3/manage-ag-createdb.html

We complete a configuration for remote connections. We will add some settings in the configuration files of PostgreSQL, e.g. pg-hba.conf and postgresql.conf

```
# Adjust PostgreSQL configuration so that remote connections to the
# database are possible.
RUN echo "host all  all    0.0.0.0/0  md5" >> /etc/postgresql/9.3/main/pg_hba.conf

# And add ``listen_addresses`` to ``/etc/postgresql/9.3/main/postgresql.conf``
RUN echo "listen_addresses='*'" >> /etc/postgresql/9.3/main/postgresql.conf
```

In the context of servers, 0.0.0.0 means all IPv4 addresses on the local machine. See Reference [14]

We expose the port where PostgreSQL will listen to. Usually PostgreSQL use port number 5432.

```
# Expose the PostgreSQL port
EXPOSE 5432
```

We setup volumes that we will use.

```
# Add VOLUMEs to allow backup of config, logs and databases
VOLUME  ["/etc/postgresql", "/var/log/postgresql", "/var/lib/postgresql"]
```

We define the entry command for this image:

```
# Set the default command to run when starting the container
CMD ["/usr/lib/postgresql/9.3/bin/postgres", "-D", "/var/lib/postgresql/9.3/main", "-c", "config_file=/etc/postgresql/9.3/main/postgresql.conf"]
```

3.2 Building a custom PostgreSQL Docker Image

Once the Dockerfile is ready, we need to build an image from the Dockerfile and assign it a name. Note: Don't ignore the period, '.', in the command.

```
sslee777@us2004lts:~/myPostgreSQL$ docker build -t mypostgresql .
Sending build context to Docker daemon  4.096kB
Step 1/11 : FROM ubuntu:16.04
16.04: Pulling from library/ubuntu
...
```

You will see long outputs from Step 1 to Step 11. It may take time, e.g. 10 mins, especially Step 4 updating OS.

Let's look at the outputs step by step. Step 1 is outputs of the first command 'FROM ubuntu:16.04' in the Dockerfile. Docker is pulling the ubuntu 16.04 images from Docker hub.

```
Step 1/11 : FROM ubuntu:16.04
16.04: Pulling from library/ubuntu
e92ed755c008: Pull complete
b9fd7cb1ff8f: Pull complete
ee690f2d57a1: Pull complete
53e3366ec435: Pull complete
Digest: sha256:db6697a61d5679b7ca69dbde3dad6be0d17064d5b6b0e9f7be8d456ebb337209
Status: Downloaded newer image for ubuntu:16.04
 ---> 005d2078bdfa
```

Step 2 is about adding the PostgreSQL PGP key in the Dockerfile.

```
Step 2/11 : RUN apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --recv-keys B97B0AFCAA1A47F044F244A07FCC7D46ACCC4CF8
 ---> Running in 203d5701be44
Executing: /tmp/tmp.lR8YI33mCW/gpg.1.sh --keyserver
hkp://p80.pool.sks-keyservers.net:80
--recv-keys
B97B0AFCAA1A47F044F244A07FCC7D46ACCC4CF8
gpg: requesting key ACCC4CF8 from hkp server p80.pool.sks-keyservers.net
gpg: key ACCC4CF8: public key "PostgreSQL Debian Repository" imported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
Removing intermediate container 203d5701be44
 ---> 58c8b5ff1209
```

In Step 3, Docker is adding PostgreSQL's repository, so Docker can install PostgreSQL packages in Step 4.

```
Step 3/11 : RUN echo "deb http://apt.postgresql.org/pub/repos/apt/ precise-pgdg main"
> /etc/apt/sources.list.d/pgdg.list
 ---> Running in d8ca33dce49f
Removing intermediate container d8ca33dce49f
 ---> 13e7d89dd04f
```

Updating OS and installing PostgreSQL packages are done in Step 4. They are updating lots of OS packages, so it takes time.

```
Step 4/11 : RUN apt-get update && apt-get install -y python-software-properties
software-properties-common postgresql-9.3 postgresql-client-9.3 postgresql-contrib-
9.3
 ---> Running in 9a4cf143b524
Get:1 http://security.ubuntu.com/ubuntu xenial-security InRelease [109 kB]
Get:2 http://archive.ubuntu.com/ubuntu xenial InRelease [247 kB]
...
(long logs)
...
Creating config file /etc/apt/apt.conf.d/50unattended-upgrades with new version
Processing triggers for systemd (229-4ubuntu21.27) ...
Processing triggers for libc-bin (2.23-0ubuntu11) ...
Processing triggers for ca-certificates (20190110~16.04.1) ...
Updating certificates in /etc/ssl/certs...
127 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d...
done.
Processing triggers for sgml-base (1.26+nmu4ubuntu1) ...
Processing triggers for dbus (1.10.6-1ubuntu3.5) ...
Removing intermediate container 9a4cf143b524
 ---> 0f3e03ed7a58
```

Changed to the user, postgres, in Step 5.

```
Step 5/11 : USER postgres
 ---> Running in ca7c1e486809
Removing intermediate container ca7c1e486809
 ---> 7d1d1ab5adec
```

In Step 6, DB User and database has been created

```
Step 6/11 : RUN    /etc/init.d/postgresql start &&    psql --command "CREATE USER
docker WITH SUPERUSER PASSWORD 'docker';" &&    createdb -O docker docker
 ---> Running in c9ae742561ae
 * Starting PostgreSQL 9.3 database server
   ...done.
CREATE ROLE
Removing intermediate container c9ae742561ae
 ---> 2deb2977eb2a
```

Step 7 shows adding a setting in pg_hba.conf file

```
Step 7/11 : RUN echo "host all  all    0.0.0.0/0  md5" >>
/etc/postgresql/9.3/main/pg_hba.conf
 ---> Running in 90bbca0cb430
Removing intermediate container 90bbca0cb430
 ---> 0e17df217b75
```

Step 8 also shows adding a network setting—adding listen_addresses.

```
Step 8/11 : RUN echo "listen_addresses='*'" >>
/etc/postgresql/9.3/main/postgresql.conf
 ---> Running in f10bad2fbbdd
Removing intermediate container f10bad2fbbdd
 ---> 5b66d6baa25b
```

Exposing port number 5432 (PostgreSQL default) is shown in Step 9.

```
Step 9/11 : EXPOSE 5432
 ---> Running in fb314ab6a57a
Removing intermediate container fb314ab6a57a
 ---> a7da341b495a
```

Volumes were added in Step 10.

```
Step 10/11 : VOLUME  ["/etc/postgresql", "/var/log/postgresql",
"/var/lib/postgresql"]
 ---> Running in 8bb5d268b40a
Removing intermediate container 8bb5d268b40a
 ---> d7d5f52159b4
```

Finally, Docker ran last command and successfully built your custom image, not pulled from Docker hub.

```
Step 11/11 : CMD ["/usr/lib/postgresql/9.3/bin/postgres", "-D",
"/var/lib/postgresql/9.3/main", "-c",
"config_file=/etc/postgresql/9.3/main/postgresql.conf"]
 ---> Running in efacdec075fe
Removing intermediate container efacdec075fe
 ---> 6a1f499741b3

Successfully built 6a1f499741b3
Successfully tagged mypostgresql:latest
sslee777@us2004lts:~/myPostgreSQL$
```

Let's verify if your custom image has been created by running below

```
sslee777@us2004lts:~/myPostgreSQL$ docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
mypostgresql        latest              6a1f499741b3        11 minutes ago      394MB
test                latest              a1ea900d3495        2 weeks ago
82.6MB
```

3.3 Container creation and verification

Once the image is built, to test the container, you just need to use this command.

```
sslee777@us2004lts:~/myPostgreSQL$ docker run -it -d --name myPoC mypostgresql
14080ec727886b15a836434a7a97fdafc4a06b6105b2c8b50d59547e2fdbd112

// Check the container
// Note: If you have unnecessary exited containers, delete it by running 'docker
container rm Container-ID'
sslee777@us2004lts:~/myPostgreSQL$ docker container ls -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS                      PORTS               NAMES
304fd524aec5        mypostgresql        "/usr/lib/postgresql…"   21 seconds ago
Up 19 seconds               5432/tcp            myPoC
e64143c099c7        postgres:latest     "docker-entrypoint.s…"   10 days ago
Exited (137) 19 hours ago                       mypostgresqlname

// Connect the container using bash shell
sslee777@us2004lts:~/myPostgreSQL$ docker exec -it myPoC bash
postgres@304fd524aec5:/$
```

It's running fine!

You may need to run the postgresql daemon if it's not running.

```
postgres@304fd524aec5:/$ service postgresql status
9.3/main (port 5432): down

postgres@304fd524aec5:/$ service postgresql start
 * Starting PostgreSQL 9.3 database server
[ OK ]
```

Let's connect the DB using psql with port number 5432, user docker, and database docker.

```
postgres@304fd524aec5:/$ psql -h localhost -p 5432 -U docker -W docker
Password for user docker:
psql (9.3.17)
SSL connection (cipher: DHE-RSA-AES256-GCM-SHA384, bits: 256)
Type "help" for help.

docker=#

// To quit
docker-# \q
postgres@304fd524aec5:/$
```

It's running well.

If you see an error message like below regarding network connection, you may need to check the pg_hba.conf file.

```
psql: could not connect to server: Connection refused
      Is the server running on host "localhost" (127.0.0.1) and accepting
      TCP/IP connections on port 5432?
```

```
postgres@304fd524aec5:/$ cat /etc/postgresql/9.3/main/pg_hba.conf
# PostgreSQL Client Authentication Configuration File
# ===================================================
#
#

...

# Database administrative login by Unix domain socket
local   all             postgres                                peer

# TYPE  DATABASE        USER            ADDRESS                 METHOD

# "local" is for Unix domain socket connections only
local   all             all                                     peer
# IPv4 local connections:
host    all             all             127.0.0.1/32            md5
# IPv6 local connections:
host    all             all             ::1/128                 md5
# Allow replication connections from localhost, by a user with the
# replication privilege.
#local   replication     postgres                                peer
#host    replication     postgres        127.0.0.1/32            md5
#host    replication     postgres        ::1/128                 md5
host    all             all             0.0.0.0/0               md5
```

4 Submit

Submit a word document (docx, doc, or PDF) to Canvas.
Assignments

- Run your code/scripts, take screenshots, and explain it.
    a. Section 3.1 Creating a custom Dockerfile for PostgreSQL
    b. Section 3.2 Building a custom PostgreSQL Docker Image
        i. Take screenshots and explain each step
        (note: take screenshots only that are shown in this exercise)
    c. Section 3.3 Container creation and verification
- Connect to **the DB using psql** from your host system
    a. Run your codes/scripts, take screenshots, and explain it
    b. This exercise showed that connecting to the DB using **psql** in the container, see Section 3.3
    c. However, you need to connect to the DB from your local host system (your host VM)
        i. Note: It's not connecting to the container using bash shell.
    d. You may need to create a new container, named myPoCHost, which is accessible from Host VM
    e. You may need to install psql on your host VM.
        i. See Reference [15]
        ii. In short,
            1. sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt $(lsb_release -cs)-pgdg main" > /etc/apt/sources.list.d/pgdg.list'
            2. sudo apt install -y postgresql-client
- Connect to **the DB using psql** from another container
    a. Run your codes/scripts, take screenshots, and explain it
    b. Within same Docker network (you may need to create a Docker network)
    c. You need to connect to the DB from another container.
    d. You may need to create a new container, named myPoCAnother, which is accessible from another host.
    e. When you create another temporary container to test accessibility, it is recommended to add an option '--rm' to delete it automatically after you exit from the container

(Hint: Read articles in Reference)

If you have any problems or questions regarding this exercise, post messages in the 'Discussions' in Canvas. Posting exact codes or links to an article that have exact codes are not allowed.

5 Reference

1. PostgreSQL: The World's Most Advanced Open Source Relational Database, https://www.postgresql.org/
2. PostgreSQL 12.3 Documentation, https://www.postgresql.org/docs/12/index.html
3. Docker Tip #79: Saving a Postgres Database in a Docker Image, https://nickjanetakis.com/blog/docker-tip-79-saving-a-postgres-database-in-a-docker-image
4. Deploying PostgreSQL on a Docker Container, https://severalnines.com/database-blog/deploying-postgresql-docker-container
5. Dockerize PostgreSQL, https://docs.docker.com/engine/examples/postgresql_service/
6. Expose PostgreSQL service to the host, https://severalnines.com/database-blog/deploying-postgresql-docker-container
7. How to create a Docker image for PostgreSQL and persist data, https://www.andreagrandi.it/2015/02/21/how-to-create-a-docker-image-for-postgresql-and-persist-data/
8. Database in a Docker container — how to start and what's it about, https://medium.com/@wkrzywiec/database-in-a-docker-container-how-to-start-and-whats-it-about-5e3ceea77e50
9. Setting up Docker & PostgreSQL — Connecting Locally — Using Advanced Functions, https://medium.com/@rrfd/setting-up-docker-postgresql-connecting-locally-using-advanced-functions-d8fe3bd58de6
10. Exploring PostgreSQL with Docker, https://markheath.net/post/exploring-postgresql-with-docker
11. Rapid API development tutorial with Docker + PostgreSQL + PostGraphile, https://medium.com/coderbunker/rapid-api-development-tutorial-with-docker-postgresql-postgraphile-e387c1c73dd8
12. Postgres Docker Official Images - start a postgres instance https://hub.docker.com/_/postgres/
13. Getting Started with Compose PostgreSQL and Jupyter Notebooks, https://www.compose.com/articles/getting-started-with-compose-postgresql-and-jupyter-notebooks/
14. What is the Difference Between 127.0.0.1 and 0.0.0.0?, https://www.howtogeek.com/225487/what-is-the-difference-between-127.0.0.1-and-0.0.0.0/
15. PostgreSQL Apt Repository, https://www.postgresql.org/download/linux/ubuntu/