**Installations for Container Runtime and Docker on the CentOs 7 Development Machines**

**Install Docker CE:**

*sudo yum install docker*

**Start docker service:**

*sudo systemctl start docker*

**Enable docker Service** (*OPTIONAL: This makes it start docker on boot automatically*)

*sudo systemctl enable docker*

Reference for above the instructions on getting Docker installed on CentOs7: <https://phoenixnap.com/kb/how-to-install-docker-centos-7>

**Install docker python library (eventually will be added to requirements.txt)**

pip install docker

**Give docker sudo permissions for a user:**

Create the docker group:

*sudo groupadd docker*

Add the user to the docker group:

*sudo usermod -aG docker $USER*

Restart virtual machine in order for new changes to take effect:

*sudo su*

*reboot*

Test out that this working by using the following command without sudo:

*docker ps*

Referenced from this linked tutorial:

[**https://docs.docker.com/engine/install/linux-postinstall/**](https://docs.docker.com/engine/install/linux-postinstall/)

**Setup the host’s Postgres Database Permissions**

# **Make your postgresql listen to an external ip address**

First find the location of postgresql.conf:

sudo find / -type f -name postgresql.conf

In our case, this was at the following location:

/usr/local/pgsql/data/postgresql.conf

Now, open this file (with elevated permission) with a text editor to edit it.

In our case, we will be using the nano editor:

sudo nano /usr/local/pgsql/data/postgresql.conf

Look for the line: *#listen\_addresses = 'localhost' # what IP address(es) to listen on;*

Uncomment and set the external ip address/es that'll be trying to access your DB. Since we are just having the Docker Container access this ip address and our localhost, we will change it to the following:

listen\_addresses = “172.17.0.1, localhost”

If you don't know it, or want to free all ips to access it (not that safe), set it to '\*':

listen\_addresses = '\*' # what IP address(es) to listen on;

Restart postgres

sudo systemctl restart postgres

You can check if this worked out with this command:

netstat -nlt

It will output something like this:

Proto Recv-Q Send-Q Local Address Foreign Address State

tcp 0 0 172.17.0.1:5432 0.0.0.0:\* LISTEN

# **Let your container access your postgresql database with a given user**

Find your pg\_hba.conf file

$ sudo find / -type f -name pg\_hba.conf # => /etc/postgresql/9.5/main/pg\_hba.conf

In our case, this is at the following location:

/usr/local/pgsql/data/pg\_hba.conf

Now, open this file (with elevated permission) with a text editor to edit it.

In our case, we will be using the nano editor:

$ sudo nano /usr/local/pgsql/data/pg\_hba.conf # or the path you found before

We want to add the following line to the table of access permissions for this database:

host johndb john 172.17.0.0/16 trust

This allows the “john” user to connect to the “johndb” database from Docker containers (the ip addresses for Docker Containers consist of the range of addresses within 172.17.0.0/16).

The *trust* method means that we will allow this connection without a password. In reality you would want to use the *md5* or *password* methods for security reasons.

**Change firewall rules to allow access:**

For our system the following firewall changes were necessary to allow the docker container to connect to the database running on the host:

*firewall-cmd --permanent --zone=trusted --change-interface=docker0*

*firewall-cmd --permanent --zone=trusted --add-port=4243/tcp*

*firewall-cmd --reload*

After entering these commands, restart the Docker service:

*sudo systemctl restart docker*

**Always make sure the 3 following steps are true before attempting to test docker container connection to host PostgreSQL database:**

1. **SELinux is in permissive mode (\*\*Optional\*\*)**

***The system is now working with SELinux in enforced mode***. However, if running for the first time, it’s better to run in permissive mode to make sure everything is running correctly. SELinux can cause some issues if not configured correctly. TODO: We go over configuring SELinux with Course Manager in the following guide: <insert link>

Check SELinux enforcing status:

*sudo sestatus*

Check to make sure permissive is the value of “Current mode”

*Current mode: permissive*

If not, enter the following to change to permissive mode:

*sudo setenforce 0*

1. **Both the localhost interface (127.0.0.1) and the docker0 interface (172.17.0.1) are labeled with SELinux**

You can check if these interfaces are labeled by using the following command:

*sudo netlabelctl unlbl list*

This command should have the following output:

*accept:on interface:DEFAULT,address:172.17.0.0/16,label:"unconfined\_u:unconfined\_r:unconfined\_t:s0"*

If not, run the following commands:

*sudo netlabelctl unlbl add default address:127.0.0.1 label:course\_manager\_u:course\_manager\_r:coordinator\_t:s0*

*~~sudo netlabelctl unlbl add default address:172.17.0.0/16 label:unconfined\_u:unconfined\_r:unconfined\_t:s0~~*

1. **Postgres is listening on the address 172.17.0.1 or address 0.0.0.0**

For some reason, when restarted a virtual machine, postgres does not automatically listen on this address.

You can check if it is listening by entering the following command:

*netstat -nlt*

It will output something like this:

*Proto Recv-Q Send-Q Local Address Foreign Address State*

*tcp 0 0 172.17.0.1:5432 0.0.0.0:\* LISTEN*

*tcp 0 0 0.0.0.0:5432 0.0.0.0:\* LISTEN*

If the Local Address it is listening on is either 0.0.0.0:5432 (listening on all network interfaces at port 5432) or 172.17.0.1 (listening on the docker0 network interface at port 5432); then this connection should work.

If it does not display this in the table of results, try restarting postgres. This would always work for me. Use the following command to restart Postgres:

*sudo systemctl restart postgres*

**Build the docker image for Course Manager:**

**\*\*NOTE:** Whenever changes are made to Course Manager make sure to rebuild the Docker image for Course Manager. This is the specific Course Manager application that will be launched in the Docker Container with the Container Runtime**\*\***

*Change directory to cops\_platform/services/course\_manager*

***sudo docker build -t course\_manager\_test:latest .***

This creates the docker image from the Dockerfile. Once a docker image is created you can then run this image using the “docker run” command.

**Test out if the docker container can connect to the Postgres Database**

***docker run --name test -d -p 8000:5000 -e POSTGRES\_URL=172.17.0.1 -e POSTGRES\_USER=john -e POSTGRES\_DB=johndb -e POSTGRES\_PW=password -e ROLE=student course\_manager\_test:latest***

The above command runs the course manager docker image as a background process (detached mode).

The -p option designates which ports this container will run on.The docker container will be running on port 5000, which is the default port for FLASK applications, but this is linked to the host’s 8000 port. Therefore, you contact this running course manager application by sending a request to the host’s 8000 port on localhost.

The --name option designates the name of the container. If no name is provided, then the container will be assigned a randomized name instead. In this example we are using “*test*” as the name of our container.

The -e option is used for providing environment variables to the running docker container. These are vital so the running Course Manager application can connect to the database.

*course\_manager\_test:latest* is the current name of this docker image that we built previously.

***docker exec -it test /bin/bash***

This command executes a command inside a running container. This allows us to connect to this container and open up a shell on this container (in other words, kind of sshing into this container)

This command requires the name of the container. In this case, we are using the container we created in the previous command that had the name of ***test***.

***docker ps***

This command lists all running docker containers

***docker system prune***

This command deletes all stopped docker containers

**docker rm -f test**

This command deletes (and forces delete if it’s still running) the container with the name ‘test’. Insert the name of the container you would like to delete instead.

**Run the Container Runtime flask application:**

Make sure the docker service is running

*Change directory to 2020SpringTeam32*

Start running the virtual environment:

*source env\_vars.sh*

*Change directory to cops\_platform/container\_runtime*

If you want to run the application with SELinux enforced:

*source flask\_app.sh*

This scripts sets the ENFORCED env to being true and the FLASK\_APP env to being the container\_runtime.py

Otherwise, manually set the flask app env to the container\_runtime application:

*export FLASK\_APP=container\_runtime.py*

*Change directory to cops\_platform/container\_runtime*

*flask run*

*The application will print out whether it is running in SELinux enforced mode when it starts. By default it will be running on port 5000.*

*If you want to run this application with SELinux enforce, set the ENFORCED env to any value. If not, unset the ENFORCED env to delete it.*

*Ctrl-C will stop the running Container Runtime if running it in the foreground.*

If you want to run the container in the background, run with

*flask run &*

stop it with

*kill -SIGINT <procid>*

Document for Curl commands:

<https://docs.google.com/document/d/1zputxB02olvAJv3_v18ZATA23nkO1hBVCUcfVxQLBIY/edit>