# 

# **Connected Operations (COps) Platform**

# Installation Guide

# 

# 

# **Skyward Federal**

# **CSC 492: Team 32**

# Jonathan Balliet

# Caleb Boswell

# Daniel Mills

# Jeen Shaji

# Spencer Yoder

## Content

1. Installing SEPostgres
2. Installing/Compiling SELinux Policy
3. Installing Docker on CentOS 7
4. Installing COps on CentOS 7
5. Confirming Successful Course Manager Installation
6. Configuring PostgreSQL with Course Manager and Docker on CentOS 7
7. Generating PostgreSQL Tables, Labels, and Data for Course Manager
8. Creating the Docker Image for Course Manager
9. Confirming Successful Container Runtime Installation

## Installing SE Postgres

|  |  |
| --- | --- |
| Primary Author | Spencer Yoder |
| Secondary Author | Daniel Mills |
| Editor | Jeen Shaji |

### What is SEPostgres?

[SEPostgres](https://wiki.postgresql.org/wiki/SEPostgreSQL_Documentation) is a module for [Postgres](https://www.postgresql.org/about/) which enforces MAC (Mandatory Access Control) through [SELinux](https://www.redhat.com/en/topics/linux/what-is-selinux). It allows database elements to be protected by SELinux.

### Cloning the Postgres Source Code

As we discovered, the only way to get SE Postgres is to build it from the [Postgres source code](https://github.com/postgres/postgres). We cloned straight from the master branch; however, we found that Postgres maintains a branch for each version. We would recommend cloning a particular version since Postgres is already sparsely documented as it is. After you get your code, you should build Postgres.

### Building Postgres

After we cloned Postgres, we had to build it. Before it could be successfully built, we installed the following packages with yum:

* [readline-devel](https://tiswww.case.edu/php/chet/readline/rltop.html)
* [zlib-devel](https://www.zlib.net/)
* [libselinux-devel](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/6.2_technical_notes/libselinux)
* [bison](https://www.gnu.org/software/bison/)
* [gcc](https://gcc.gnu.org/)
* F[lex](https://en.wikipedia.org/wiki/Flex_(lexical_analyser_generator))

The build process will halt if you’re missing a package. We installed every package it said we were missing. However, some options can be ignored upon configuration. [Here](https://www.postgresql.org/docs/12/install-procedure.html) is a list of configuration options. Note that this list is incomplete, such is the way of Postgres documentation.  
Speaking of configuration, that’s done by running the configure binary inside the Postgres directory with ./configure. To install SE Postgres, we used

./configure --enable-debug --enable-cassert --with-selinux

As described [here](https://wiki.postgresql.org/wiki/SEPostgreSQL_Documentation#Installation). If configure runs successfully, the next step is to run

sudo make and then

sudo make install.

If both of these run successfully, Postgres should be installed in /usr/local/pgsql. If it’s not, you may be able to find the installation directory with

sudo find / -n pgsql

### Building SEPostgres

SE Postgres is still not built. We found our SEPostgres directory at /home/maintuser/postgres/contrib/sepgsql. Again, this can be located with

sudo find / -n sepgsql

After changing to this directory, you can run

sudo make

sudo make install

If both of these succeed, SE Postgres should be built

### Configuring the System to Run Postgres

Postgres creates [three databases](https://chartio.com/resources/tutorials/how-to-list-databases-and-tables-in-postgresql-using-psql/) by default, template0, template1, and postgres. A Linux user called ‘postgres’ needs to be created to use Postgres with the postgres database. After postgres is launched with this database, [other databases can be created and mapped to users](https://medium.com/coding-blocks/creating-user-database-and-adding-access-on-postgresql-8bfcd2f4a91e).

* Create the postgres user with:

sudo adduser postgres

Many Postgres commands start with sudo -i -u postgres which runs the succeeding command as the user postgres. In order to authorize this on your system, run sudo visudo to edit /etc/sudoers.tmp to allow your user or root to login as postgres. In our file, we uncommented an existing line:

%wheel ALL=(ALL) NOPASSWD: ALL

Which says, all users in the group wheel can login as any user without a password. If you’re missing such a line, you can enter i to insert text. Type a line like the following:

username ALL=(ALL) NOPASSWD: ALL.

For your default username. If that doesn’t work, you can use the username ‘root’. Enter esc :wq to exit out of vi.

* Next, run to give postgres full access to the pgsql directory and initialize a database.

sudo chown postgres /usr/local/pgsql

sudo -i -u postgres /usr/local/pgsql/bin/initdb -D /usr/local/pgsql/data --no-locale

This may result in some issues with accessing files in this directory if you’re not logged in as postgres. The code below will allow you to login as postgres:

sudo -i -u postgres

* This next part is *optional*: if you want Postgres to launch every time your system starts up, do these next steps.

Add this file: [postgres.service](https://drive.google.com/file/d/1vDtNvee8rRlv5WoPiGwFUgmbAod6CXh7/view) to the directory /usr/lib/systemd/system. This file is a file which was modified by us from Postgres’s default service file. After adding this file, run the commands:

sudo systemctl daemon-reload

which tells systemctl about your new service file

sudo systemctl enable postgres

which tells systemctl to start the postgres server every time your

systemctl is-enabled postgres

which will let you know whether your postgres service was enabled. This step is relatively simple, all we did was edit /usr/local/pgsql/data/postgresql.conf, which contains a line saying shared\_preload\_libraries = ''. Note that this file is very large. This line is towards the bottom. We used nano as a text editor, which allows you to search for text with ctrl+w. Modify this line to say

shared\_preload\_libraries = 'sepgsql'

* Finally, run the command below.

This should all be on one line, it was broken up to maximize readability. This command was extracted from [here](https://wiki.postgresql.org/wiki/SEPostgreSQL_Documentation#Installation). We don’t know what it does exactly.

sudo -i -u postgres /usr/local/pgsql/bin/postgres

--single -F -O -c exit\_on\_error=true

-D /usr/local/pgsql/data postgres <

/usr/local/pgsql/share/contrib/sepgsql.sql > /dev/null

A quick note about the full command paths: we originally added /usr/local/pgsql/bin to the path for the default user, but that doesn’t transfer when we run as user postgres. A way to fix that would be to run sudo -i -u postgres and then [modify the .bashrc file](https://docs.oracle.com/cd/E19062-01/sun.mgmt.ctr36/819-5418/gaznb/index.html) to add the postgres bin to the PATH environment variable. We ended up just using the full path names to the binaries to run everything.

There’s some information [here](https://wiki.postgresql.org/wiki/SEPostgreSQL_Documentation#Regression_Tests) about configuring SE Postgres for regression tests, but we did not have to do that to get our application to work.

### Launching SEPostgres

To start the SE Postgres server (if it’s not already started), you can run

sudo systemctl start postgresif you have the postgres.service file configured, or

sudo -i -u postgres /usr/local/pgsql/bin/pg\_ctl -D /usr/local/pgsql/data start if you don’t.

[pg\_ctl](https://www.postgresql.org/docs/10/app-pg-ctl.html) is a command which helps manage your server. The -D argument tells it where to find your database files.

To issue commands to the server, run below code to open an interface where you can type in SQL commands.

sudo -i -u postgres /usr/local/pgsql/bin/psql

If you create some data elements, you can assign them SELinux labels with the [SECURITY LABEL](https://www.postgresql.org/docs/9.1/sql-security-label.html) command. Security labels are stored in the [pg\_seclabel](https://www.postgresql.org/docs/10/catalog-pg-seclabel.html) catalog. We automated a lot of this process in this shell script: [sepostgres.sh](https://drive.google.com/file/d/1UDtKxdY7nGUHCW0B4zzM1qbAYTc--s-v/view)

## Installing/Compiling SELinux Policy

|  |  |
| --- | --- |
| Primary Author | Spencer Yoder |
| Editor | Jeen Shaji |

Our SELinux policy module for CourseManager is located in:

cops\_platform/policies/course\_manager

Requirements for compilation/installation:

* [selinux-policy-devel](https://pkgs.org/download/selinux-policy-devel)
* [policycoreutils-python](https://pkgs.org/download/policycoreutils-python) (or whichever package provides the ‘semodule’ command for your operating system).
* SELinux installed on your system

On our CentOS7 machines, installing selinux-policy-devel provided a Makefile in the directory:

/usr/share/selinux/devel

which is used to compile an SELinux policy module, and policycoreutils-python provided the ‘semodule’ command which allows the module to be built onto the existing SELinux policy. Inside the policy directory, there are 2 pertinent files: course\_manager.te and compile\_alias.sh

Running

$ source compile\_alias.sh

Will provide bash aliases for compiling and building our policy module:

$ secompile

which compiles the policy module into course\_manager.pp using the Makefile, and

$ sebuild

which adds the module onto the existing policy using semodule -i. course\_manager.te contains the source code for our policy module should you wish to make changes to the policy.

## Installing Docker on CentOS 7

|  |  |
| --- | --- |
| Primary Author | Jonathan Balliet |
| Editor | Jeen Shaji |

### Steps for installation:

* 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

* Give docker sudo permissions for a user:
  + Create the docker group:

sudo groupadd docker

* Add the user *($USER signifies the current 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

### Resources:

* Instructions on getting Docker installed on CentOs7: <https://phoenixnap.com/kb/how-to-install-docker-centos-7>
* Giving Docker sudo permissions:

<https://docs.docker.com/engine/install/linux-postinstall/>

## Installing COps on CentOS 7

|  |  |
| --- | --- |
| Primary Author | Jonathan Balliet |

Pull latest changes from remote Github repo of the 2020SpringTeam32 directory. The following installations are needed for Python to work with Postgres:

sudo yum -y install python3-devel

sudo yum -y install postgresql-libs

sudo yum -y install postgresql-devel

Change to the root directory of our repo (2020SpringTeam32)

Create a Python virtual environment::

python -m venv venv

source venv/bin/activate

Install all needed dependencies in this virtual environment:

pip install -r requirements.txt

Add the location of /pgsql/bin to your PATH:

export PATH=/usr/local/pgsql/bin:$PATH

The following set of instructions allow us to create a test database and user in Postgres that we will be using in development for testing our application. Feel free to use any names for the corresponding database and user. However, it is important to note that for running our application the specific environment variables are coded in some of our bash scripts to correspond to a database with the name *“johndb”* and the user’s name as *“john”*. Changes you make to the database name and user will need to made within these bash scripts as well.

The following command will switch to the Postgres user and open up a *psql* terminal that is used to interact with Postgres:

sudo -i -u postgres /usr/local/pgsql/bin/psql

Now add a new user to the database with the name *“john”* and password *“password”*:

CREATE USER john with password ‘password’;

Next we will create a new database named *“johndb”*:

CREATE database johndb;

Finally, we will grant all privileges for the user *john* on the *johndb* database.

GRANT ALL privileges on database johndb to john;

In order for both our Course Manager and Container Runtime applications to communicate with the Postgres Database running on the host machine, there are a number of environmental variables that must be set with specific values. We have a bash script, titled “env\_vars.sh” that you can run to manually set all of these environment variables. It is located in the 2020SpringTeam32 root directory.

The bash script can be run with the following command to set these environment variables:

source env\_vars.sh

If you would like to set these manually instead, enter the following commands:

export FLASK\_CONFIG=development

export FLASK\_APP=course\_manager.py

export POSTGRES\_URL=127.0.0.1

export POSTGRES\_USER=john

export POSTGRES\_PW=password

export POSTGRES\_DB=johndb

export LC\_ALL=en\_US.UTF-8

export PYTHONPATH=/home/maintuser/2020SpringTeam32

***NOTE:*** The following environment variable was required on one of our development machines to successfully run the Course Manager application. This was related to a *PQconninfo error* message.

Run the following command to set this environment variable if you receive this *PQconninfo error* on your machine:

export LD\_LIBRARY\_PATH=/usr/local/pgsql/lib:/usr/local/lib64

## Confirming Successful Course Manager Installation:

|  |  |
| --- | --- |
| Primary Author | Jonathan Balliet |

If you followed all the steps in the previous section, you should now have everything installed to successfully run Course Manager on CentOS 7.

Change directory to the course\_manager directory in our repo.

Example of the command to use starting from the root directory of our repo:

cd 2020SpringTeam32/cops\_platform/services/course\_manager/

Now run the following command to start Course Manager:

flask run

You should be greeted with a series of printed messages saying course\_manager.py is running on the local host at port 5000 (This is the default URL and port when running Flask). You can then stop running the Course Manager Flask application at any time by pressing “Ctrl-C” on the terminal.

If this installation was not successful, make sure the following are true and then confirm the installation was successful again:

1. All the environment variables listed in the previous section for *Installing COps on CentOS 7* have been set. These are vital so Python knows where to launch the application (PYTHONPATH) and so Flask knows which Flask application to run (FLASK\_APP).
2. You are running this inside the Python virtual environment listed in the previous section for *Installing COps on CentOS 7*. This virtual environment contains all of the libraries for running Course Manager, that were previously installed via the requirements.txt file. You can confirm you are in the virtual environment by seeing the text *(venv)* on the terminal. If you are not in the virtual environment, run the following command from the 2020SpringTeam32 directory which this was previously installed in to activate the virtual environment:

source venv/bin/activate

The bash script, env\_vars.sh, will automatically activate this virtual environment, along with setting all of the environment variables needed to run Course Manager.

### Potential Installation Issues:

1. If the Course Manager application complains of not being able to import a specific module/library, there could be an issue where the *requirements.txt* file, which contains all of the needed dependencies, has not been updated with all the latest libraries used. To fix this, Google how to *“pip install”* this specific missing library. Make sure you are running inside the Python virtual environment and then use the following command (with the specific library you are installing) to install a missing dependency:

pip install <library-name>

1. If the Course Manager application itself throws an exception and refuses to run, it’s possible there are some other installations needed for the specific environment you are running in. Your best bet for this situation is copy the error and paste it in Google to troubleshoot and find a solution. This was a common tactic when troubleshooting installation issues on CentOS 7, which is admittedly something we had to do frequently throughout the course of this project. Google and StackOverflow never failed to give us a working solution.

## 

## Configuring PostgreSQL for Course Manager and Docker on CentOS 7

|  |  |
| --- | --- |
| Primary Author | Jonathan Balliet |

### 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. 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 accessing your DB. Since we are just having the Docker Container interface (at address 172.17.0.1) 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 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

In the example above, the PostgreSQL database is listening on the 172.17.0.1 address at the port 5432. Port 5432 is the default port used for PostgreSQL and indicates this is PostgreSQL is listening at this address.

### 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. We will be using the nano editor:

$ sudo nano /usr/local/pgsql/data/pg\_hba.conf #

or the path you found before

* 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.
* As noted previously, we are using *“johndb”* for the database and *“john”* for the user throughout these installation instructions for consistency. Make sure to replace with the specific names for the database and user you are using.
* An additional line needs to be added to the table of access permissions. This is so the Container Runtime can access PostgreSQL through this specific database and user (in our case *johndb* and *john,* respectively). This is also necessary for Course Manager to run normally (not inside a Docker Container) if using a PostgreSQL connection and so it can generate and label the tables of this database via a script.
* Add the following line to the table of access permissions for this database:

host johndb john 127.0.0.1 /32 trust

Instructions on how to set Postgres config files to accept connections with Docker:

<https://gist.github.com/MauricioMoraes/87d76577babd4e084cba70f63c04b07d>

### Change firewall rules to allow access

* On our CentOS 7 development machines, 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

## Generating PostgreSQL Tables, Labels, and Data for Course Manager

|  |  |
| --- | --- |
| Primary Author | Jonathan Balliet |
| Editor | Jeen Shaji |

Successfully connecting the PostgreSQL database with the Course Manager application, requires a few additional configuration steps. In order for SEPostgreSQL to accept connections from a specific client, these connections must be labeled with SELinux on the host machine. Running the Course Manager application normally (in other words, not through Container Runtime), requires the localhost connection (at address 127.0.0.1) to be labeled with SELinux.

First install net-label tools on your machine:

sudo yum -y install netlabel\_tools

Now use the following command to label the 127.0.0.1 address:

sudo netlabelctl unlbl add default address:127.0.0.1 label:unconfined\_u:unconfined\_r:unconfined\_t:s0

You can confirm this was successful but using the following command:

netlabelctl unlbl list

The above added label for the address 127.0.0.1 should be listed after *“accept on:”*.

Next we are going to create the tables, along with some mock data, for the PostgreSQL database that Course Manager is going to use. To do this we will run a script that resets the state of the database. This script also labels all of the columns and tables of this database according to our SELinux policy. As a precondition, all the previous installation sections of this guide must be completed.

***NOTE:*** Whether you want to label the database columns and tables is based on whether the environment variable *ENFORCING* has been set.

Use the following command to set this environment variable:

export ENFORCING=true

This environment variable can be set to any arbitrary value, it simply must exist.

If you do not wish to label the database, you can use the following to unset this environment variable if it does contain a value:

unset ENFORCING

Change to following directory:

2020SpringTeam32/cops\_platform/services/course\_manager/tests/db/

Now use the following command:

python generate\_db.py

If no error was thrown, then the tables were successfully generated for the database. It’s important to note that everytime this script is run, the database is destroyed (the tables are dropped) and then recreated. These tables are created based on the model classes defined inside Course Manager. Some mock data for the database is also generated via this script file.

### Potential Issues

This script will not run if SELinux is in *enforcing* mode, but it will run if SELinux is in *permissive* mode. This is because we did not end up creating a SELinux Policy that allows all of the SQL commands running within this script to run on the labeled local host connection.

To check SELinux’s status, enter the following command:

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

If you are still encountering issues after doing the above, make sure all the *preconditions* have met (all previous installation steps have completed) and refer to the potential issues section for ***Confirming Successful Course Manager Installation***. It’s very likely it is either a missing dependency or environment error, which would need to be fixed in the same manner as listed in this section.

## Creating the Docker Image for Course Manager

|  |  |
| --- | --- |
| Primary Author | Jonathan Balliet |

### *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.

Make sure all of the previous installation steps have been completed.

### Build the Docker Image for Course Manager

* Change directory to *2020SpringTeam32/cops\_platform/services/course\_manager*

Enter the following command to build the Docker image for Course Manager:

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

This creates the docker image from the Dockerfile located in the *course\_manager* directory. The name of this docker image is *“course\_manager\_test:latest”*. Once a Docker image is created you can then run this image at any time using the “*docker run*” command.

* Test out if the docker container can connect to the Postgres Database. This command runs the Course Manager docker image as a background process (in detached mode)

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 -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. All of the POSTGRES\_\*\* related environment variables are required in order for the application to contact the host’s database. The ROLE environment variable is optional and designates what the role of the user is that will be accessing this application. If none is given, it will default to “student” in the Course manager application.

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

* The following 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)

docker exec -it test /bin/bash

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 the *test*.

* This command lists all running docker containers

docker ps

Use the -a option with the above command list all containers (even ones that are stopped).

* This command deletes all stopped docker containers and anything in the docker cache

docker system prune

* 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.

docker rm -f test

## Confirming Successful Container Runtime Installation

|  |  |
| --- | --- |
| Primary Author | Jonathan Balliet |

***NOTE:*** Always make sure the following steps are true before attempting to run Container Runtime or test a Docker Container connection to the host PostgreSQL database depending on whether running in Permissive or Enforcing mode with SELinux.

Check SELinux enforcing status:

sudo sestatus

The current mode will be listed by the value of “Current mode”. In this example, it is running in permissive mode:

Current mode: permissive

The following command runs SELinux in permissive mode:

sudo setenforce 0

The following command runs SELinux in enforced mode:

sudo setenforce 1

### Running in Permissive Mode:

1. Make sure both the localhost interface (127.0.0.1) and the range of Docker Container IP addresses (172.17.0.1/16) 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" interface:DEFAULT,address:127.0.0.1,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:unconfined\_u:unconfined\_r:unconfined\_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 both address 172.17.0.1 and 127.0.0.1 *OR* on the address 0.0.0.0

For some reason, when restarting our virtual machines, PostgreSQL does not automatically listen on these addresses. 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*** 172.17.0.1:5432 (listening on the docker0 network interface at port 5432) and on 127.0.0.1:5432 (listening on localhost); then this connection should work. If it does not display this in the table of results, try restarting postgres. This would always work for us. Use the following command to restart Postgres:

sudo systemctl restart postgres

If this is still not working make sure you have added these to connect correctly in the Postgres configuration files as detailed in the previous ***Configuring PostgreSQL for Course Manager and Docker on CentOS 7*** section of this Installation guide.

1. The environment variable FLASK\_APP is set to container\_runtime.py and the environment variable ENFORCING is unset.

Set FLASK\_APP:

export FLASK\_APP=container\_runtime.py

Unset ENFORCING:

Unset ENFORCING

### Running in Enforcing Mode:

1. Make sure both the localhost interface (127.0.0.1) is labeled with SELinux. You can check if this interfaces is labeled by using the following command:

sudo netlabelctl unlbl list

This command should have the following output:

accept:on interface:DEFAULT,address:127.0.0.1,label:"unconfined\_u:unconfined\_r:unconfined\_t:s0"

If not, run the following command:

sudo netlabelctl unlbl add default address:127.0.0.1 label:unconfined\_u:unconfined\_r:unconfined\_t:s0

1. Postgres is listening on both address 172.17.0.1 and 127.0.0.1 *OR* on the address 0.0.0.0

For some reason, when restarting our virtual machines, PostgreSQL does not automatically listen on these addresses. 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*** 172.17.0.1:5432 (listening on the docker0 network interface at port 5432) and on 127.0.0.1:5432 (listening on localhost); then this connection should work. If it does not display this in the table of results, try restarting postgres. This would always work for us. Use the following command to restart Postgres:

sudo systemctl restart postgres

If this is still not working make sure you have added these to connect correctly in the Postgres configuration files as detailed in the previous ***Configuring PostgreSQL for Course Manager and Docker on CentOS 7*** section of this Installation guide.

1. The *flask\_app.sh* bash script has been run. This automatically sets the ENFORCING environment variable and the FLASK\_APP environment variable to container\_runtime.py. This script is located in the *2020SpringTeam32/cops\_platform* directory and can be run with the following command:

source flask\_app.sh

### 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 script 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>

Instructions for using the Curl command for testing REST APIs are located in the *Testing APIs with Curl Commands* section of our Developer’s Guide.

These same instructions are located on our Google Drive at the following URL:

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