* Download the files from <https://github.com/jpaffrath/docker-openplc>
* Create a directory in your project folder and name it whatever you want. I just chose “OpenPLC”.
* Copy the downloaded files into this folder.
* Edit Dockerfile to be the following (I will also upload it as a .txt to the GitHub):

FROM debian:buster

USER root

WORKDIR /root/

RUN apt-get update && \

apt-get -y install git python3 python3-pip python3-venv autoconf bison build-essential pkg-config flex automake libtool make sudo sqlite3 cmake

RUN useradd --create-home --shell /bin/bash openplc

RUN adduser openplc sudo && \

echo '%sudo ALL=(ALL) NOPASSWD:ALL' >> /etc/sudoers

USER openplc

WORKDIR /home/openplc

RUN git clone https://github.com/thiagoralves/OpenPLC\_v3.git

USER root

RUN python3 -m venv /home/openplc/OpenPLC\_v3/.venv

RUN /home/openplc/OpenPLC\_v3/.venv/bin/pip install --upgrade pip

RUN /home/openplc/OpenPLC\_v3/.venv/bin/pip install flask flask-login pyserial pymodbus

USER openplc

WORKDIR /home/openplc/OpenPLC\_v3

RUN sudo ./install.sh custom

RUN sudo apt-get clean

RUN mkdir /home/openplc/scripts

COPY run.sh /home/openplc

RUN sudo chmod +x /home/openplc/run.sh

EXPOSE 502 8080 8084 20000 43628

CMD ["/home/openplc/run.sh"]

* + I had to make these changes to manually create the .venv directory in /home/openplc/OpenPLC\_v3. I was fucking around for over an hour to figure out that the directory simply was not there. The “start\_openplc.sh” file was looking for it (start\_openplc.sh is what actually runs OpenPLC, and is invoked by the last line in “run.sh”. In Zane and Thomas’s case, the container was created successfully, but OpenPLC was not started successfully), so it was silently erroring out. It was also using python2.7 instead of 3 so I updated that as well.
* Build the docker image with the following:
  + **sudo docker build -t openplc-docker .**
    - Make sure your working directory is the one you made earlier (“OpenPLC” for me, the directory containing the Dockerfile). The above command creates a docker image named “openplc-docker” based on the Dockerfile in the current working directory, as per the ‘.’ at the end.
* Once the image is finished building, go back to the main project folder (should just be 1 directory backwards). Now update the docker compose file if you haven’t already, to include the OpenPLC containers.
* Now you should be able to re-run the yaml and start up the new container. Access it with:
  + http://[IP of VM]:[non-502 port you defined in the yaml]
  + So for me it would be “http:192.168.0.138:8083”
* If it all worked correctly, you should see a login screen. Login with “openplc” as both the username and password.
* Now lets add a separate OpenPLC server for each system, similar to the Guacamole setup. Copy the same code you just wrote into the yaml under the other two systems’ sections, changing whatever parameters as necessary (ex: change any instance of “EV-charger” to “Monorail” or “Power-plant”, change the IPs’ subnets to that of the appropriate network, so 100 to 101/102 for me, and finally change the ports to not overlap, so 8084/8085:8080 and 503/504:502).
* Now we need to update what ModBus port (the 500 port) these 2 new containers are listening on (update from 502 to 503/504). We need to edit the “openplc.db” database for this, which stores some config information such as this. Run the following commands:
  + sudo docker compose exec --user root OpenPLC\_Monorail bash
  + cd webserver
  + sqlite3 openplc.db
  + \*Optional: do “SELECT \* FROM Settings;” to check what the server is currently listening on for ModBus requests. It should be 502 by default.
  + UPDATE Settings SET Value = '[new port number]' WHERE Key = 'Modbus\_port';
    - I just chose 503 and 504 for the other two systems
  + \*Optional: do “SELECT \* FROM Settings;” again to check if your change went through.
  + \*NOTE: If you accidentally put the containers down (‘sudo docker compose down’, which removes them) instead of stopping them (‘sudo docker compose stop’), you will need to make these DB changes again (I think).

NEXT STEPS:

* Define the st file with actual logic for each of the systems in the city. Currently it just uses a placeholder st file so it doesn’t error out, but until we define the st file with real logic, the OpenPLC server will be useless to us.
* Some port fuckery might happen when we try to write actual ModBus logic for the OpenPLC containers. The Dockerfile only explicitly exposes port 502, but for the Monorail and Power plant ones, I had to manually set them to listen on 503 and 504 by editing the Settings table in the database (described above). I think that should work fine, but if there are issues with the Monorail and Power plant servers in particular, that is probably why (would need to expose 503 and 504 in the Dockerfile and rebuild the image).

***Useful commands:***

* + **sudo nano docker-compose.yaml**
    - Edit the docker compose file, or create it if it doesn’t exist
  + **sudo docker compose up -d**
    - Start all of the containers defined in the yaml file
  + **sudo docker compose stop**
    - Stops all running containers defined in the yaml file
  + **sudo docker compose exec --user root [name of container] bash**
    - Provides a shell to whichever container you specify, as root. Useful for debugging.
    - **sudo docker exec -u 0 -it [name of container] bash**
      * For non-docker compose containers (standalone docker containers created manually, without a yaml file).
  + **sudo docker compose logs [name of container]**
    - Displays logs of specified container. Used on the “guacamole” ones (ones that you connect to in order to get the guacamole web interface), useful for debugging.
  + **sudo docker ps -a**
    - Check all containers and various information about each one (debugging)
    - **sudo docker rm [container name/id]**
      * Remove an old, outdated container
  + **sudo docker image ls**
    - View all currently saved Docker images. Sometimes when trying to rebuild an image, it tries to write to some of the same locations that the old one uses, causing an error. Use this to see and delete (below) conflicting images.
    - **sudo docker image rm [image name/id]**
      * Remove an image
  + **sudo docker container inspect [name of container] --format "{{json .NetworkSettings.Ports}}"**
    - Check specified container’s open ports