* Install VMWare.
  + Download the installer [here](https://support.broadcom.com/group/ecx/productdownloads?subfamily=VMware%20Workstation%20Pro&freeDownloads=true).
    - You will need to make an account and sign in with it.
    - For me, when I signed in it brought me to the dashboard. Just click the link again to the download page.
    - Download the installer for your OS and install it.
* Download the pre-built Kali image from [Kali's official site](https://www.kali.org/get-kali/#kali-virtual-machines)
* Open VMWare and go to File -> Open -> navigate to where you extracted the zip -> select the .vmx file.
* Go to “Edit virtual machine settings” and then “Options” to change the name of the VM. You can also optionally change the memory/CPU allocation if you want. I changed the RAM from 2 GB to 8 and left the cores at 4.
* Power on the VM. The default login credentials should be “kali” for the username and password.
* **NOTE**: When I was first messing with this, I noticed my files weren’t saving across reboots. Making a snapshot seemed to fix it, but when running through the setup again with a fresh download of the image, the files seemed to save without needing to do this. **So before you do anything, I would create a test file, reboot, and check to see if it saved**.
* Create a project folder called “GamifiedICS” on the desktop (GUI or command line).
  + Make sure there’s no space and correct capitalization.
* Paste the contents of the “**VM Setup**” folder into it.
  + If you get a “permission denied” error, run “sudo chown -R kali:kali /Desktop/GamifiedICS” in the terminal.
* Now run the following 3 commands:
  + **cd Desktop/GamifiedICS**
    - If it says it doesn’t exist, just run “cd GamifiedICS”.
  + **sudo chmod +x setup.sh**
    - You will be prompted for the password, again it should be “kali”.
  + **./setup.sh**
    - This will run the startup script and do the rest. It will take a few minutes.
* Once that’s done, the containers should all be up and running. To access the relevant web interfaces, you will need to know the IP of the VM. To find this, run **ip a**, and look for a connection that says **eth0**, or something similar. The first IP in purple is the one you will use to connect to the various containers:
  + **Wordpress:**
    - **EV Charger**: [IP of VM]:8000/wp-admin
      * username ***user***, password ***!MTixTnKZWeR0pI1xF***
    - **Monorail**: [IP of VM]:8001/wp-admin
      * username ***user***, password ***5YNNqPB3&vYiEWq&8o***
    - **Power Plant**: [IP of VM]:8002/wp-admin
      * username ***user***, password ***HuPqEpb%qC#^Z7qyai***
  + **OpenPLC:**
    - **EV Charger**: [IP of VM]:8083
      * username and password: ***openplc***
    - **Monorail**: [IP of VM]:8084
      * username and password: ***openplc***
    - **Power Plant**: [IP of VM]:8085
      * username and password: ***openplc***
  + **Guacamole:**
    - **Laptop:**
      * [IP of VM]:8080/guacamole
        + username and password: ***guacadmin***
        + You also need to make a connection to the Laptop container, which will simulate the user’s in-game laptop:

Click on the “guacadmin” in the top right -> Settings.

Go to Connections (the bar near the top) -> New Connection.

Fill in the following fields:

Name: choose a name: **Your Laptop**

Protocol: **SSH**

Parameters -> Network -> Hostname: **Laptop**

Parameters -> Network -> Port: **22** (SSH default port)

Parameters -> Authentication -> Username: **root**

Parameters -> Authentication -> Password: **root**

Go to the bottom and click Save.

Click on the “guacadmin” in the top right again -> Home

You should see the connection now. Click it to test it out.

You should see a terminal if it worked correctly.

* + - **EV Charger:**
      * [IP of VM]:8090/guacamole
        + Make a connection the same as before, but change the name to “**EV Charger**” and the authentication parameters to “**user**” and “**password**” (instead of “root” and “root”).
    - **Monorail:**
      * [IP of VM]:8091/guacamole
        + Make a connection the same way as above, changing only the name.
    - **Power plant:**
      * [IP of VM]:8092/guacamole
        + Make a connection the same way as above, changing only the name.
* And that’s it. If you want to start/stop the containers manually, you can use **sudo docker compose up -d** and **sudo docker compose stop** respectively.

**Other Info (mostly notes for myself):**

* I needed to copy the wordpress sql server data (/var/lib/mysql) and general wordpress data (/var/www/html) over from an already-working container to any newly created ones to allow the user to just run the setup script and have the wordpress sites work seamlessly. Before, I kept getting permission denied errors, requiring me to install wordpress from scratch, and I wanted to keep the setup process as easy as possible.
  + To do this for the sql server side of things (/var/lib/mysql):
    - I dumped the database from one of the pre-installed wordpress containers with **sudo docker exec sql-server\_EV-charger mysqldump --no-tablespaces -uuser -ppassword db\_EV-charger > db\_EV-charger.sql**.
      * The **--no-tablespaces** just prevented the command from attempting to create a tablespace, which it couldn’t without certain permissions I didn’t have/couldn’t figure out how to obtain
      * The **user** and **password** are those defined in the yaml file under the “sql-server\_EV-charger” container
      * **db\_EV-charger** is the database we want to dump, also defined in the yaml
      * **“> db\_EV-charger.sql”** writes the dump to the working directory of the host and names it “db\_EV-charger.sql” (could also do “> /path/name.sql”, but the working directory was fine and simpler).
    - Then I did that for the Monorail and Power plant and copied the sql files to my host PC.
    - Then I pasted them into the test VM (in the same ‘init’ folder as the guacamole DB initialization file) and mounted it in the container’s entrypoint folder (/docker-entrypoint-initdb.d). As far as I could tell, .sql files in this folder are automatically loaded, populating the database defined in the “environment” section.
      * Full statement was **- ./init/db\_EV-charger.sql:/docker-entrypoint-initdb.d/db\_EV-charger.sql:ro**, under “volumes”. The ‘:ro’ just declares it as read-only.
  + For the general wordpress data (/var/www/html), I just had to copy the contents of the pre-installed container’s /var/www/html, paste it into the test VM, and mount it to each wordpress container’s /var/www/html path.
    - To copy the files, I first made another subfolder in the project folder, named “wordpressData”, with 3 subfolders, EV-charger, Monorail, and Power-plant.
    - Then I ran **sudo docker cp wordpress\_EV-charger:/var/www/html ./wordpressData/EV-charger**, which copied the data in /var/www/html in the “wordpress\_EV-charger” container to the “./wordpressData/EV-charger” folder on the host.
    - Then I copied this “wordpressData” folder to the test VM, also placing it in the project folder, and mounted each subfolder’s contents to its respective container with **- /home/kali/Desktop/TEST/wordpressData/EV-charger:/var/www/html**, replacing the line using the named volume “wordpress\_data\_EV-charger”. It didn’t really serve a purpose and was interfering anyway (couldn’t do this and have the persistent named volume simultaneously without doing some goofy workaround).
* Commands in the “setup.sh” and explanation of what they do:
  + cd Desktop/GamifiedICS
  + sudo apt update
  + sudo apt install -y docker.io
    - Installs docker
  + sudo apt install -y curl
    - Needed to install docker compose v2 (v1 is deprecated)
  + mkdir -p /usr/local/lib/docker/cli-plugins
    - Create install path for docker compose
  + ARCH=$(uname -m)
    - Creates ARCH variable that holds the system architecture, passes it in the next step to download the correct docker compose version
  + sudo curl -SL https://github.com/docker/compose/releases/latest/download/docker-compose-linux-$ARCH -o /usr/local/lib/docker/cli-plugins/docker-compose
    - Installs docker compose v2
  + sudo chmod +x /usr/local/lib/docker/cli-plugins/docker-compose
    - Adds execute permission to it
  + mkdir init
    - Creates a project folder for the compose file and a subfolder call “init” to initialize SQL databases
  + cd init
  + sudo docker run --rm guacamole/guacamole /opt/guacamole/bin/initdb.sh --mysql | sudo tee initdb.sql > /dev/null
    - \*Must run this while inside of the [project folder]/init directory
    - Runs guacamole/guacamole image to generate the “initdb.sql” database initialization file, outputting it to the /init directory
  + cd ../OpenPLC-Setup
  + sudo docker build -t openplc-docker .
    - Don’t forget that period at the end! That tells the command to look in the current directory for a Dockerfile to build off of.
  + sudo docker compose up -d
    - Starts the containers. You must be inside the project folder for this though, otherwise the command won’t see the docker-compose.yaml.

***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
    - Doing **sudo docker compose down** instead of “stop” deletes the containers as well as stops them. Can be useful, but use with caution (don’t mix them up!)
      * **sudo docker compose down -v** also removes any volumes in use by containers. Could be useful for debugging.
  + **sudo docker compose exec -it [name of container] bash**
    - SSH into the specified container.
  + **sudo docker compose logs [name of container]**
    - Displays logs of specified container. 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 volume ls/rm**
    - View/remove current docker volumes
  + **sudo docker cp [container]:/path1/file1 /path2/file2**
    - Copy files (or entire directories) from a container to the host. Path1 and file 1 refer to the path/file to the desired data in the container, and path2/file2 refer to the location you want that data copied to on the host.
    - To do the other way around (copy from host to container), you need to mount the files to a specified path via the volumes section in the yaml.
      * For example, “/home/kali/Desktop/TEST:/var/www/html” copies the contents of /home/…/TEST to /var/www/html in whatever container you’re creating this volume for.
  + **sudo docker container inspect [name of container] --format "{{json .NetworkSettings.Ports}}"**
    - Check specified container’s open ports
  + **(In SQL servers after docker execing into them):**
    - **mysql -uuser -ppassword** (or -uroot -prootpassword if you get a permission error).
      * Sign into mysql as a user to use sql commands.
    - **use [database name]** (ex: “use db\_EV-charger”, the name you chose for the DB in the yaml).
      * Specify the database you want to view/manipulate.
    - **SHOW TABLES;**
      * Show all tables in the database, useful for troubleshooting.
    - **SELECT \* FROM [table];**
      * See the contents of a table, also useful for troubleshooting.