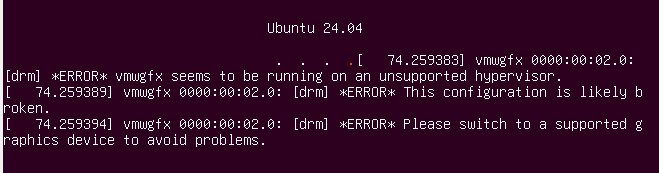
First download Ubuntu Desktop (<https://ubuntu.com/download/desktop>) and make a VM with it:

* (VirtualBox) Click “New”
* Chose a name (no spaces)
* Select the ISO that you downloaded
  + Should auto-populate a few fields
* Then go to the “Unattended Install” tab and change the username and password (I just did user:password)
* Then go to “Hardware” and select an appropriate amount of memory/CPU threads to allocate to the VM. I have a 16 thread CPU and 32 GB of RAM, so I allocated 6 threads and 8GB of RAM.
* \*Optional: Go to “Hard Disk” and note the path the VDI is stored/change it if you want.
* Start up the VM and install Ubuntu (“Try or Install Ubuntu”)
  + **Note**: I was getting the following error, which may be because of my GPU it seems (“Please switch to a supported graphics device”), and I don’t know how to fix it other than simply trying again. I kept trying to install until it magically just worked. The “safe graphics mode” installation didn’t help either. You can *kind of* continue even if you get this error, but then I ran into all sorts of issues like the VM freezing on the login screen, getting stuck on the Ubuntu installation for over an hour, or failing to allocate more than like 1GB of space to the VM when I clearly allocated 25 for the VDI in the VirtualBox settings. When I got lucky and didn’t have the error show up, all of these issues went away. So I have no idea.



* Once it’s installed, it will automatically restart. You will need to close the VM and reopen it, but before you reopen, we will adjust some settings.
  + Go to General->Advanced->Shared Clipboard->change from Disabled to Bidirectional (no drag ‘n’ drop because it just doesn’t work, at least for me).
  + Network->Adapter 1->Attached to->change from NAT to Bridged Adapter
  + Network->Adapter 1->Promiscuous Mode->change from Deny to Allow VMs
* (Set up bidirectional clipboard here with the below instructions, if needed for project?)
* Now reopen the VM and run the following commands in the terminal once you’re in:
* 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)
* sudo 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
* docker compose version
  + Check version to see if it installed correctly
* sudo mkdir -p [project folder name]/init
  + Make a project folder for the compose file and a subfolder call “init” to initialize SQL databases
* sudo docker run --rm guacamole/guacamole /opt/guacamole/bin/initdb.sh --mysql | sudo tee init/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 we just created
* sudo nano docker-compose.yaml
  + Creates the yaml file and opens it in the nano text editor
  + \*\*\*Populate the yaml file with the following:

[get the docker compose file from the GitHub]

* Then run “sudo docker compose up -d” while inside the project folder to spin up the containers.
* In guacamole, when creating the connection, fill out the following fields:
  + Name: choose a name
  + Protocol: SSH
  + Parameters -> Network -> Hostname: name of the container that you want to SSH into
  + Parameters -> Network -> Port: 22 (SSH default port)
  + Parameters -> Authentication -> Username: root
  + Parameters -> Authentication -> Password: root
* To stop the containers, run “sudo docker compose stop”
* Getting bidirectional clipboard to work:
  + sudo apt update
  + sudo apt install -y build-essential dkms linux-headers-$(uname -r)
  + Go into the VM’s VirtualBox window, go to Devices at the top, then Insert Guest Additions CD image.
  + cd /media/$USER
  + ls (check name of the directory in here)
    - * If there is no directory then you will need to manually mount the guest additions ISO apparently?
  + sudo ./[directory you found from above step]/VBoxLinuxAdditions.run
  + sudo ./VBoxLinuxAdditions.run
  + sudo reboot
    - Obviously reboots the VM so save any work that needs to be saved.

***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. 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