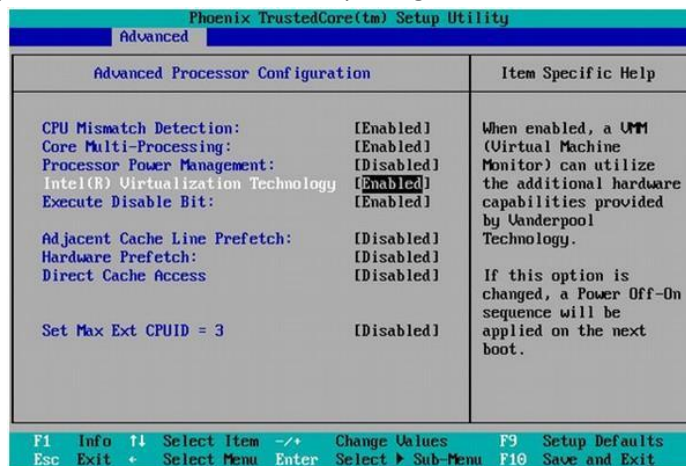


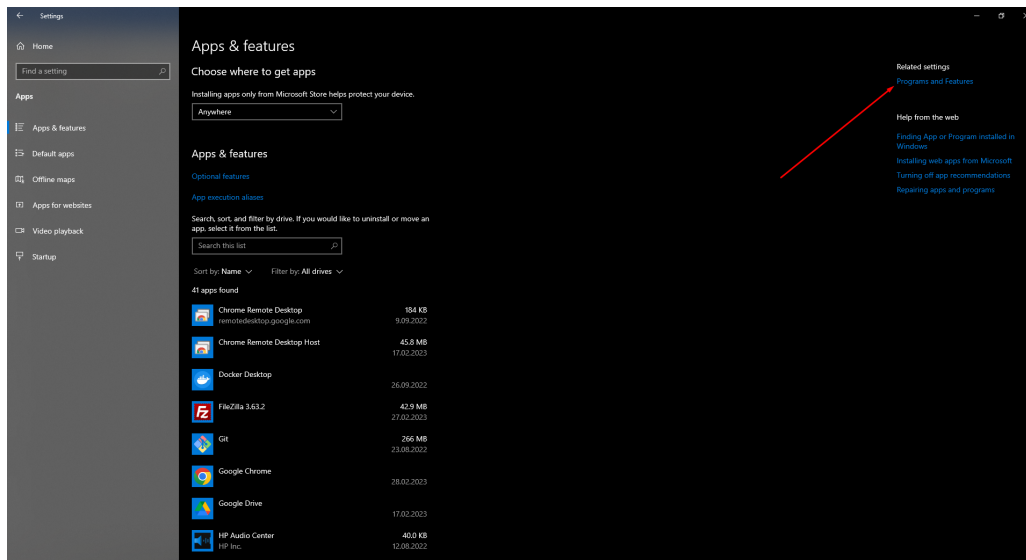
ITU Computer Engineering Department
BLG 335E Algorithm Analysis I, Fall 2023-2024
Development Environment Setup – Windows (10-11)

Preparing Windows for Docker

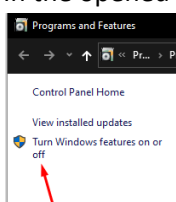
1. Start your computer and **press the key that brings BIOS settings**. Since different manufacturers have **different BIOS keys**, you can search online for yours (it can be one of the F1,F2,F4,F5 or delete buttons).
2. In the BIOS settings, find the configuration items related to the CPU. These can be in under the headings **Processor, Chipset, or Northbridge**.
3. Enable virtualization; the setting may be called **VT-x, AMD-V, SVM, or Vanderpool**. Enable **Intel VT-d** or **AMD IOMMU** if the options are available. An example is given below:



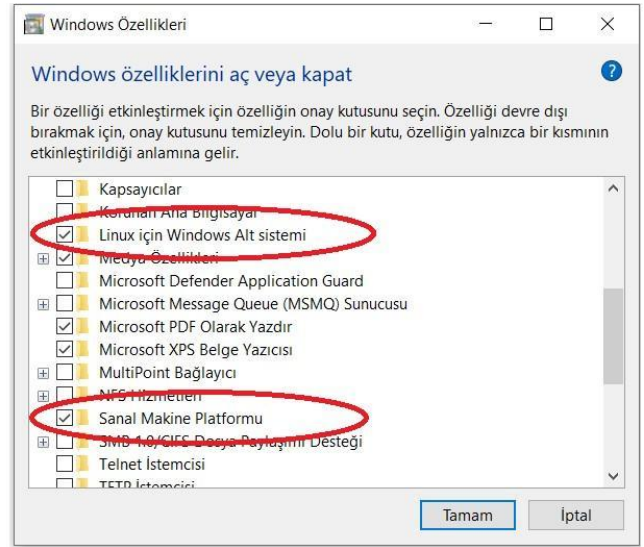
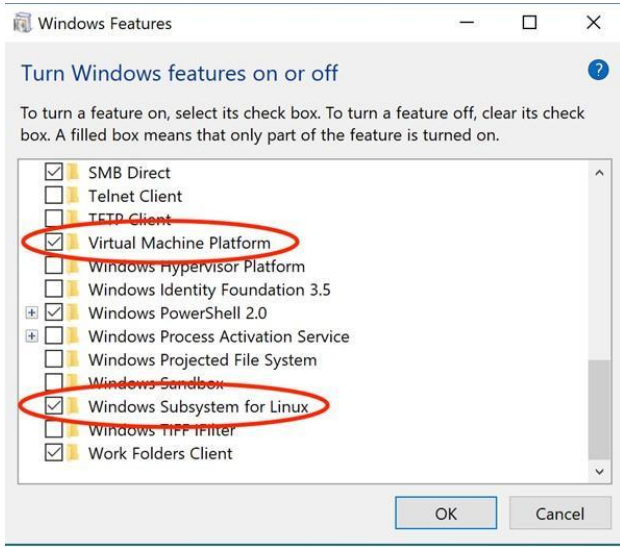
4. Navigate to “Apps & Features” in Windows after successful boot.
5. Click “Programs and Features” as in below image:



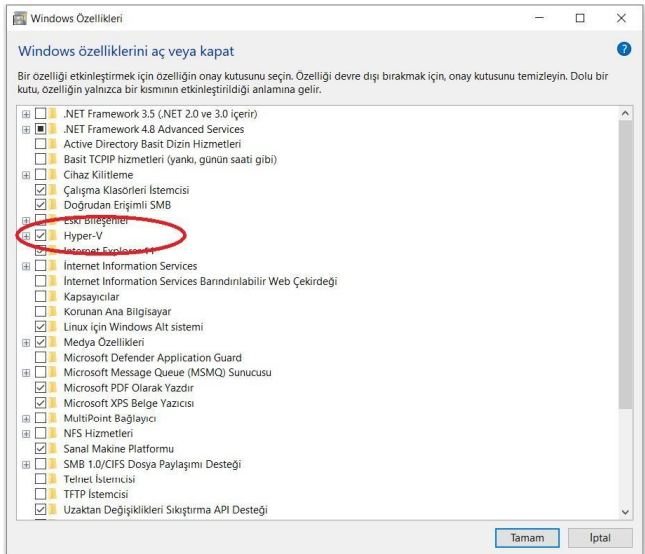
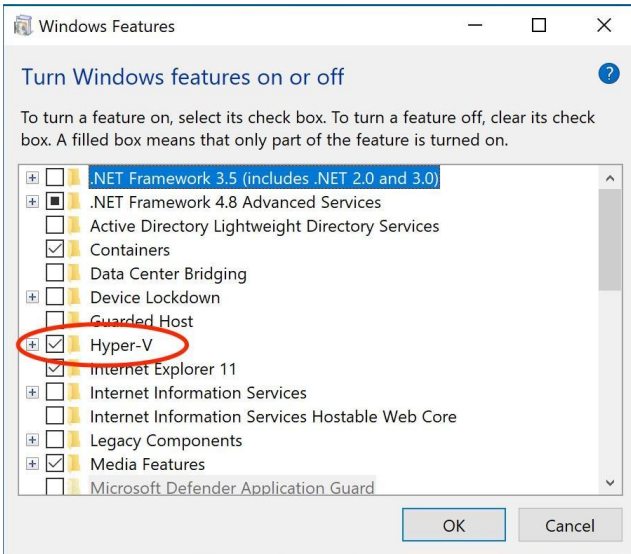
6. In the opened window, click “Turn Windows Features on or off” in left pane as in image below:



7. Enable [Virtual Machine Platform](#) and [Windows Subsystem for Linux \(WSL\)](#) Windows features:



8. Enable [Hyper-V](#) Windows feature:



9. After enabling these features, a reboot is needed.

10. Update Windows 10 ([Start->Settings->Update & Security->Windows Update](#)) so that the changes are applied (e.g., installation of WSL). Then, restart your computer.

11. Check if WSL is installed, type wsl in your favorite terminal (cmd, PowerShell etc.). If command is not recognized, then install [WSL from Microsoft Store](#) (Windows Subsystem for Linux). Again, type the command to ensure that wsl is installed.



Current expected output is: "Windows Subsystem for Linux has no installed distributions" You do not need to install a distro. Just ensure that "wsl" command is recognized by the Windows.

12. Update wsl by following command:

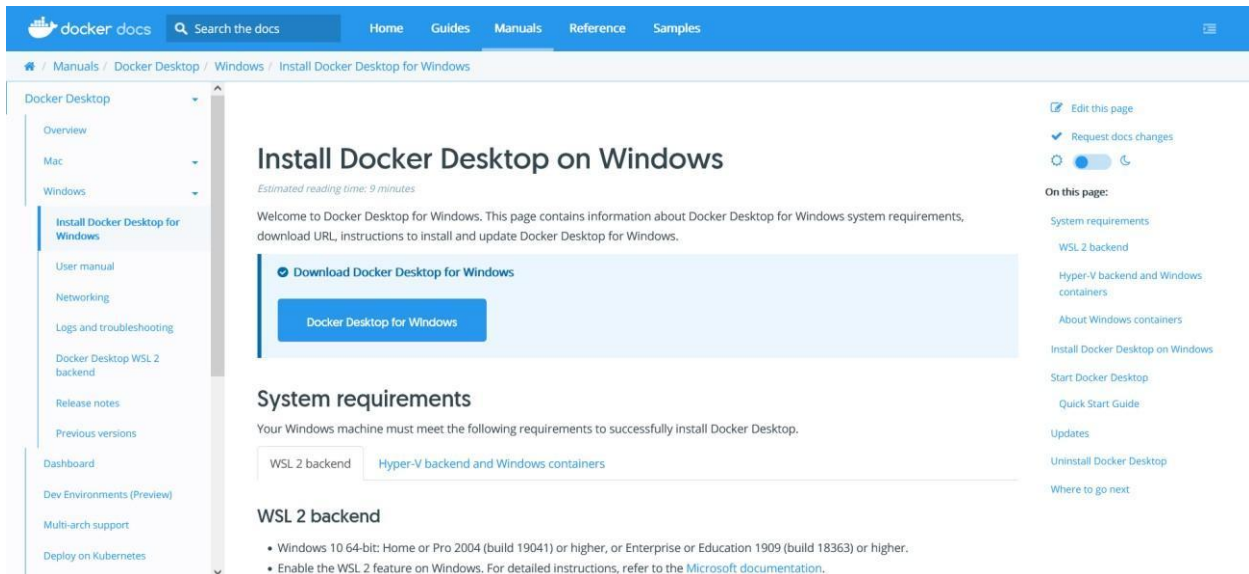
```
wsl --update
```

Installing Docker for Desktop

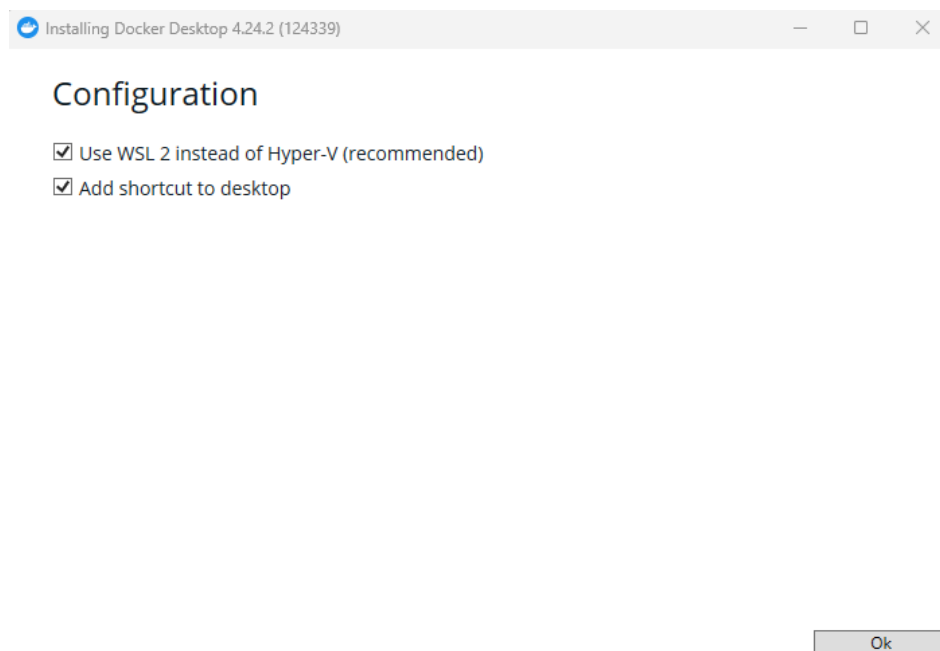


For convenience, please follow the same naming with the tutorial for folders, images and containers.

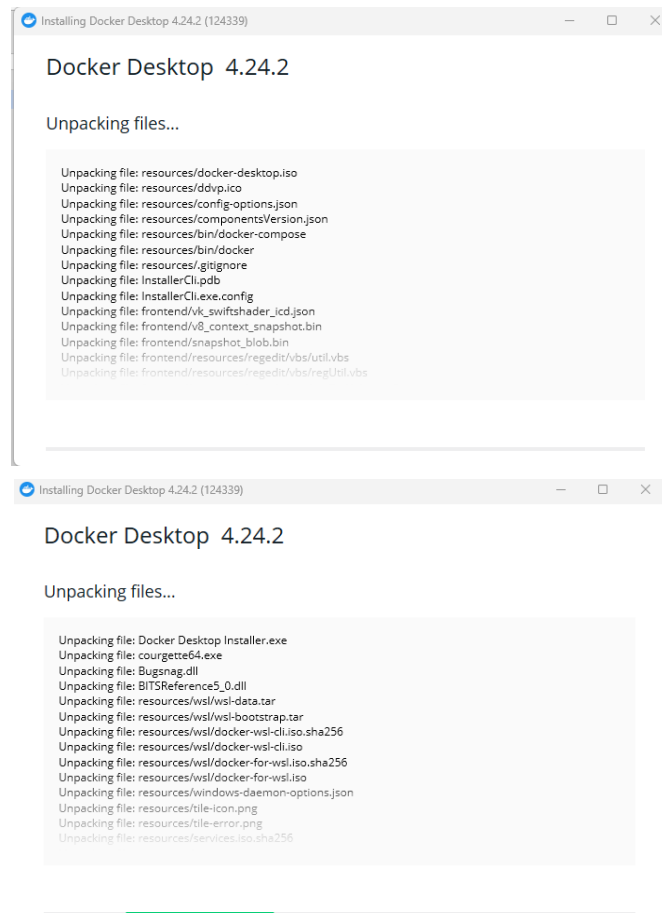
- Open your web browser (e.g, Firefox or Chrome) and go to the [Install Docker Desktop on Windows](#) web page to download [Docker Desktop Installer](#) application:



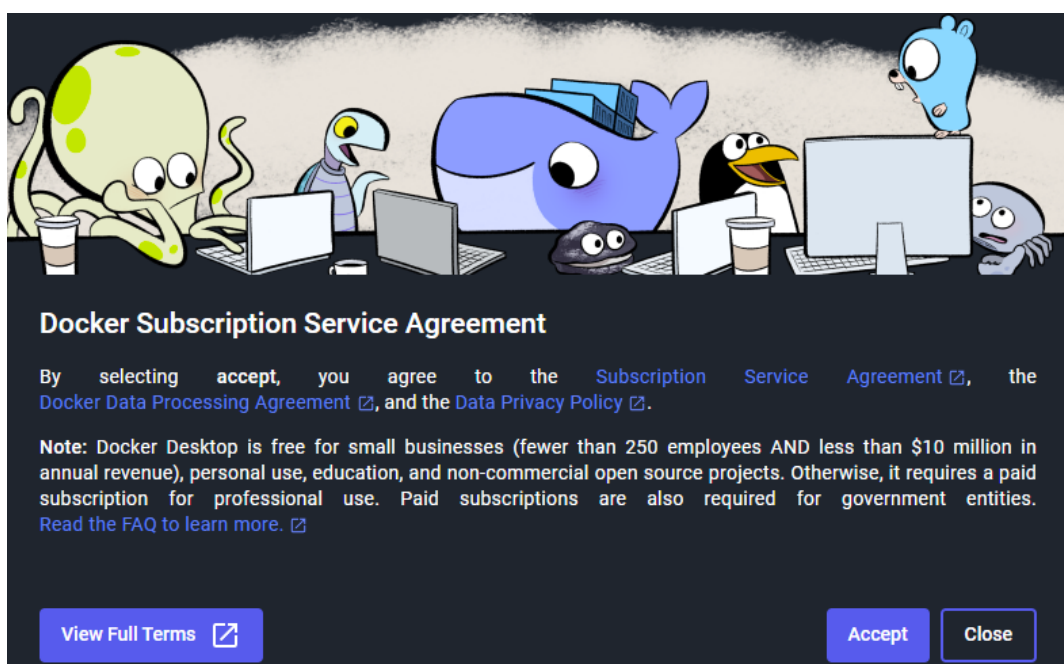
- Run the [Docker Desktop Installer](#) and accept default [Configuration](#) settings (we will use WSL 2 instead of only Hyper-V support as recommended):



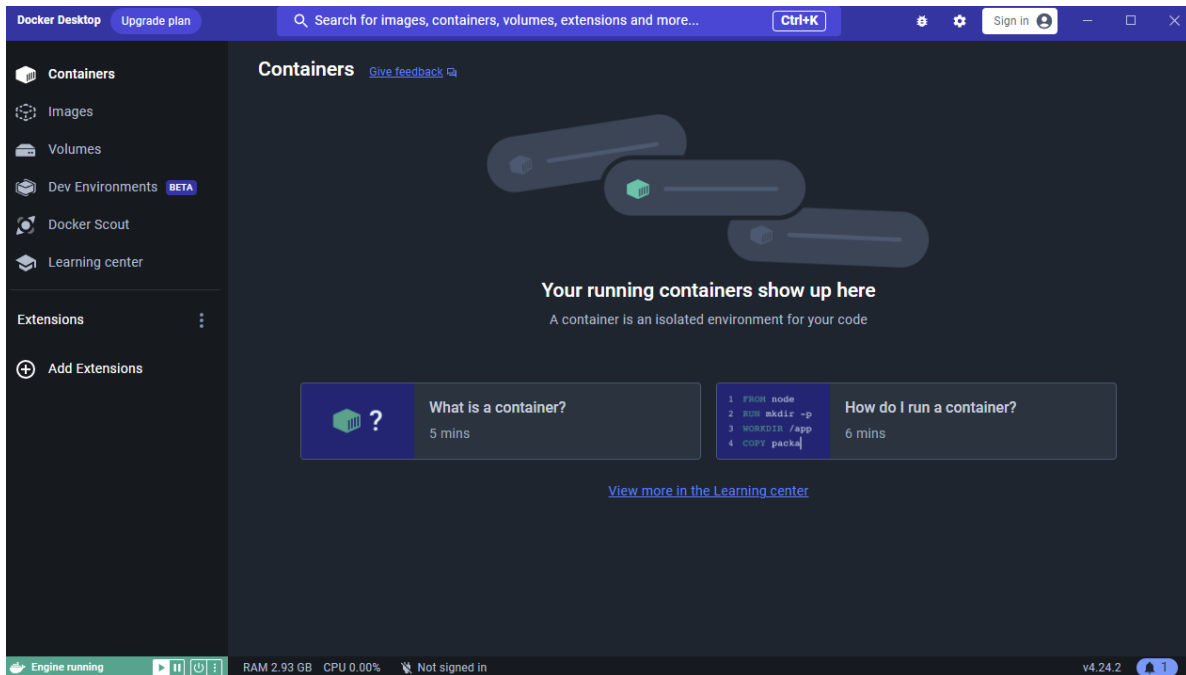
15. After [Unpacking files...](#) and [Installing..](#) the Docker, you will see [Installation Succeeded](#) message. [Close](#) the window, now on you can start using the Docker:



16. Open [Docker Desktop](#) application and accept the terms:



17. You end up with the following screen which states that there are no running containers:



18. Open a terminal window and create a directory (e.g., named as `vm-docker`) with `mkdir vm-docker` command. File location can be in user documents folder like in given example. Change your working directory as `vm-docker` with the `cd vm-docker` command.

```
Command Prompt
Microsoft Windows [Version 10.0.19044.2604]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ERHAN-LAB>mkdir vm-docker

C:\Users\ERHAN-LAB>cd vm-docker

C:\Users\ERHAN-LAB\vm-docker>_
```

19. Put the `dockerfile` (available in ninova) to the `vm-docker` directory.

20. Run the following command to build the docker image:

`docker build -t dockeralgo .`

- Pay attention to the `dot` at the end of the command.
- For consistency, it is recommended to use the name of the docker image in our example.
- Here, **dockeralgo** is the name of the docker image.
- It should take a while to build the image, a screen similar to below should appear at the end:

```
Komut İstemi
C:\Users\Erhan\vm-docker>docker build -t dockeralgo .
[+] Building 1.6s (23/23) FINISHED
=> [internal] load build definition from dockerfile
=> => transferring dockerfile: 653B
=> [internal] load .dockerignore
=> => transferring context: 28
=> [internal] load metadata for docker.io/library/ubuntu:bionic
=> [ 1/19] FROM docker.io/library/ubuntu:bionic@sha256:152dc842452c496887f07ca9127571cb9c29697f
=> CACHED [ 2/19] RUN apt update
=> CACHED [ 3/19] RUN apt install openssh-server sudo -y
=> CACHED [ 4/19] RUN useradd -rm -d /home/ubuntu -s /bin/bash -g root -G sudo -u 1000 test
=> CACHED [ 5/19] RUN usermod -aG sudo test
=> CACHED [ 6/19] RUN service ssh start
=> CACHED [ 7/19] RUN echo 'test:test' | chpasswd
=> CACHED [ 8/19] RUN apt update
=> CACHED [ 9/19] RUN apt install build-essential -y
=> CACHED [10/19] RUN apt install gdb -y
=> CACHED [11/19] RUN apt update
=> CACHED [12/19] RUN apt install valgrind
=> CACHED [13/19] RUN apt update
=> CACHED [14/19] RUN apt install git -y
=> CACHED [15/19] RUN apt update
=> CACHED [16/19] RUN apt install python3 -y
=> CACHED [17/19] RUN apt install python3-pip -y
=> CACHED [18/19] RUN sudo update-alternatives --install /usr/bin/pip pip /usr/bin/pip3 1
=> CACHED [19/19] RUN pip install calico
=> exporting to image
=> => exporting layers
=> => writing image sha256:8d112d383c336c4a775b44bba47210bcb11b540f830efbb2470f515bf48f634b
=> => naming to docker.io/library/dockeralgo
```

21. Check if the image has been built with the `docker image ls` command:

```
C:\Users\Erhan\vm-docker>docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
dockeralgo	latest	8d112d383c33	12 minutes ago	686MB

22. Create a folder (`<dev_path>`) to store the source codes and necessary files. Whenever you update the content of this folder, content of the folder within the container (`<container_path>`) will be updated too (You will write your code for the assignments within this folder).

23. Run the following command **by only changing the underlined red colored part with your local development folder path**. This command will boot a container instance from the image you have just built.

```
docker run -p 2222:22 -v C:\Users\Erhan\Volume:/home/ubuntu/hostVolume
--name docker_algo --hostname vm_docker -d dockeralgo
```

Explanation of the parts of the command:

```
docker run -p <local_port>:<container_port>
-v <dev_path>:<container_path>
--name <container's local name>
hostname <container's host name>
-d <image's name>
```

- `-p <local_port>:<container_port>`: Maps `container_port` of the container to one of your pc `local_ports`. This is required to be able to make ssh connection with your container.
- `-v <dev_path>:<container_path>`: Maps your local files to a directory inside the container.

- It should almost instantly boot the container and return the ID of the container. A screen similar to below should appear:

```
C:\Users\Erhan>docker run -p 2222:22 -v C:\Users\Erhan\Volume:/home/ubuntu/hostVolume --name docker_algo --hostname vm_docker -d dockeralgo
d8f58e7a87ba9aecf33c2fc0b7e0d443072b2de7132d0f89f4072fa80abf9f08
```

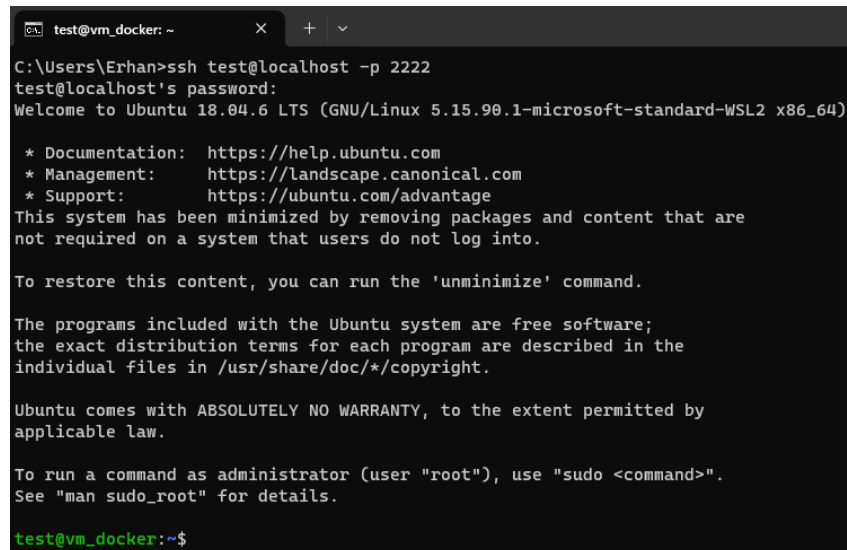
- You may perform an additional check by running the `docker ps --all` command. This will list the available containers as in image below:


```
C:\Users\Erhan>docker ps --all
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS                   NAMES
d8f58e7a87ba   dockeralgo "/usr/sbin/sshd -D"      3 minutes ago Up 3 minutes   0.0.0.0:2222->22/tcp    docker_algo
```

- Check if you can ssh into your container with the `ssh username@host_name -p <local_port>` command as follows:

```
ssh test@localhost -p 2222
```

- Your container's username and password both are set as `test`.
- Don't forget to `exit` from the ssh session.



```
test@vm_docker: ~
C:\Users\Erhan>ssh test@localhost -p 2222
test@localhost's password:
Welcome to Ubuntu 18.04.6 LTS (GNU/Linux 5.15.90.1-microsoft-standard-WSL2 x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage
This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

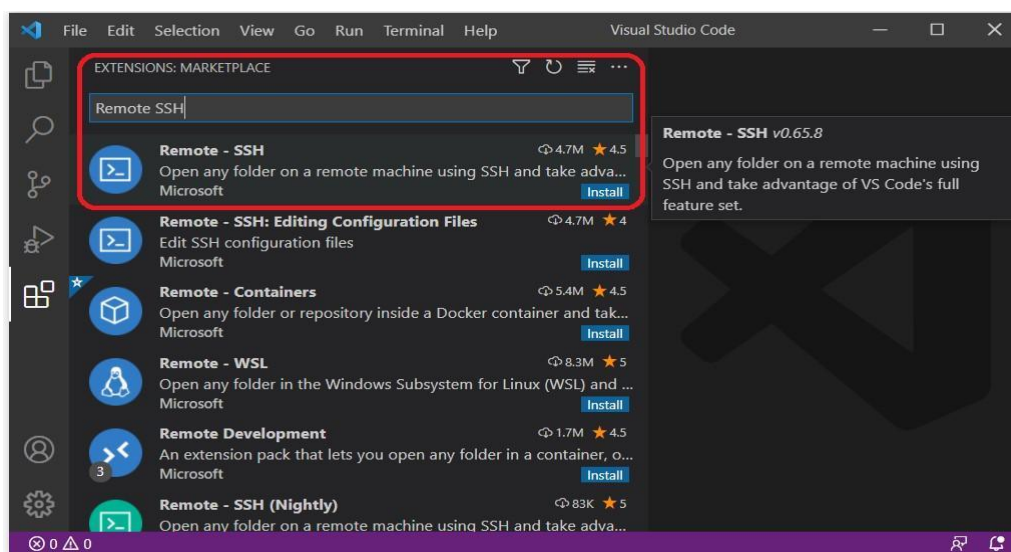
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

test@vm_docker:~$
```

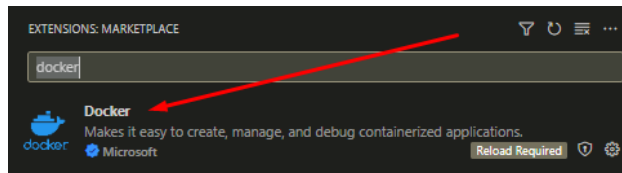
VS-Code Set-up

Installing extensions

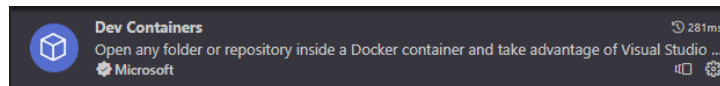
- Install Visual Studio Code in your system (Be careful: Visual Studio and Visual Studio Code are different.). Run Visual Studio Code and open **EXTENSIONS MARKETPLACE** by pressing **CTRL+SHIFT+X**. Then, search for the extension named **Remote SSH** and install it:



- Search for "Docker" extension. Installed the official extension as following screenshot:



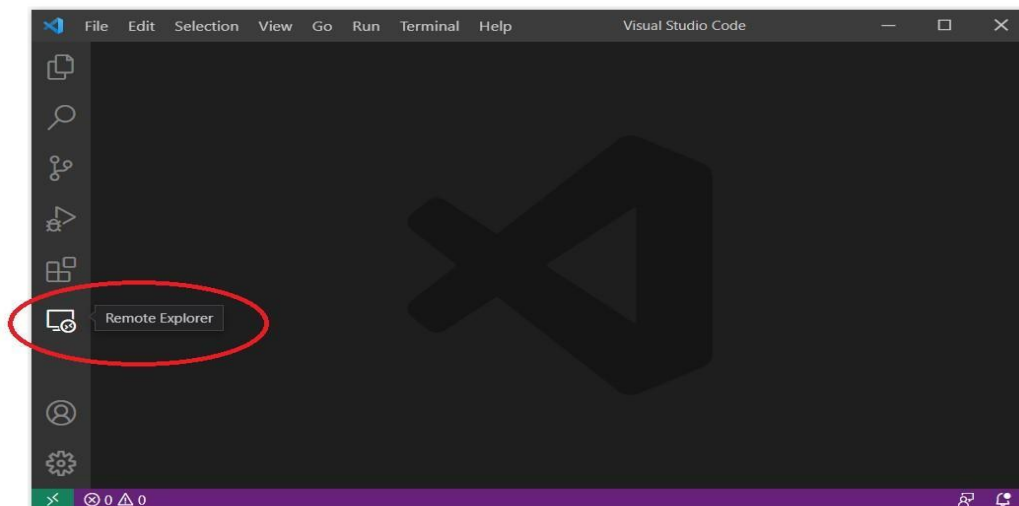
3. To attach Visual Studio Code on Docker containers, you need to install “Dev Containers” (formerly, remote containers extension):



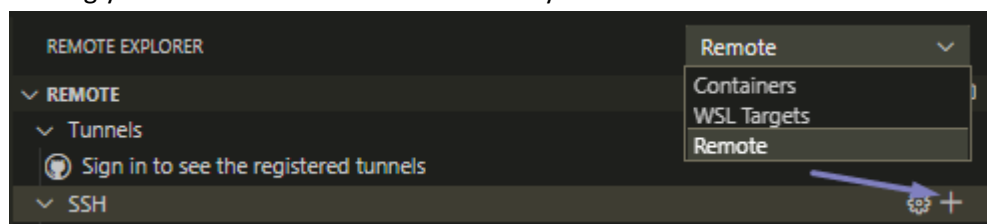
4. After installing the extensions, you can **either follow instructions in “Connecting with SSH within VS-Code” or “Connecting with Docker extension” (faster). Then, follow instructions in “In-container set-up” and so on.**

Connecting with SSH within VS-Code

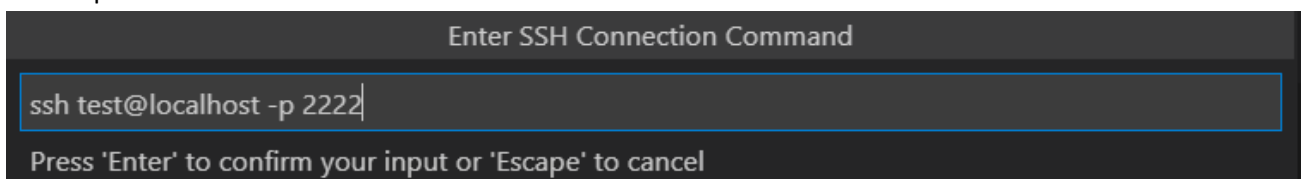
5. After installing the extension, you will see a newly added [Remote Explorer](#) button.



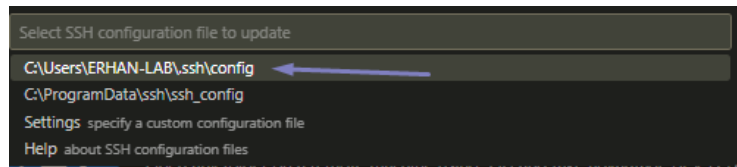
6. Click on the newly added “Remote Explorer” button and select **Remote** in the dropdown menu. Click the “+” sign to make a connection with localhost. If there is no ssh targets available, you have to add new. After adding you do not need to create each time you will use SSH.



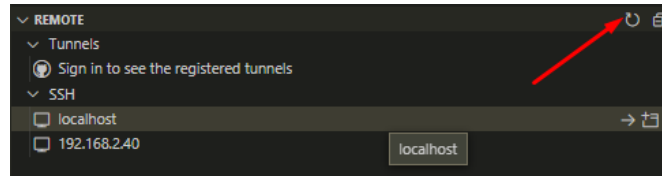
7. If there is no ssh targets available, add new target by “+” sign and run the below command as in 16th step:



8. Select the file that follows the path "**Users/{yourusername}/.ssh/config**".



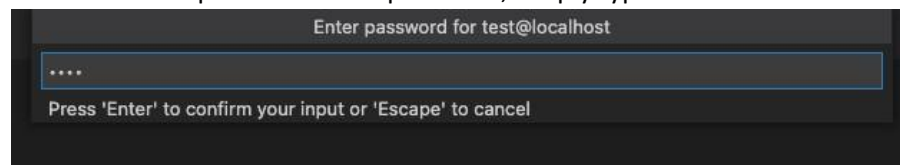
9. Refresh the connections:



10. Establish a connection by pressing “Connect in New Window” button as in image below:

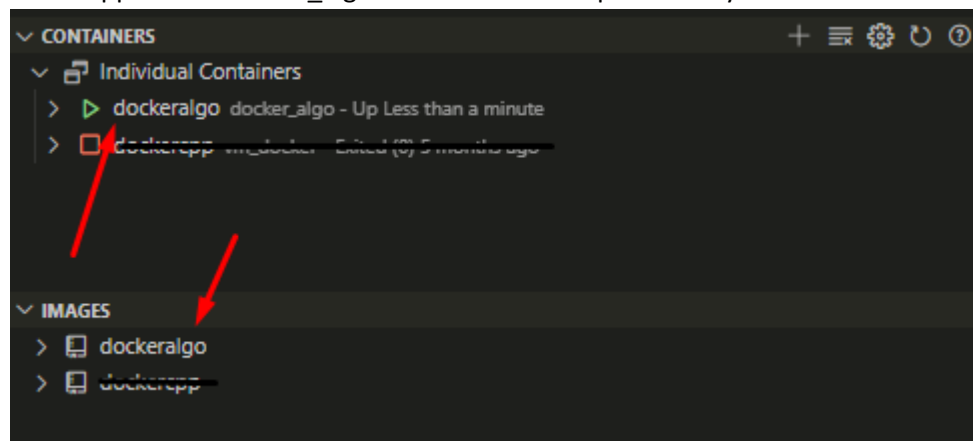


11. If VSCode asks for the operating system afterwards, choose **Linux**.
12. A new window should open and ask for password; simply type “test”.

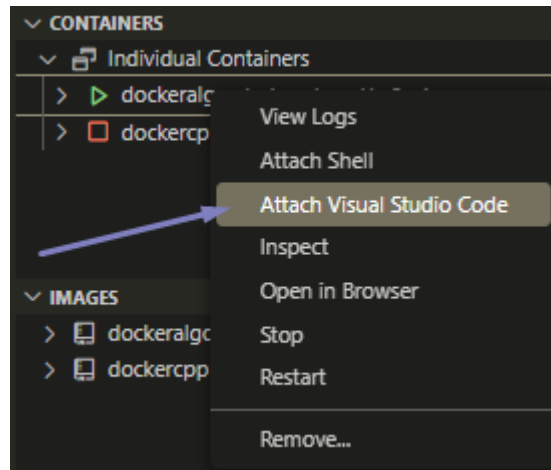


Connecting with Docker extension

5. Click on the docker icon on the left pane. Since the container is built from the image we prepared, we should see the container. Green arrow means that container is now active, red rectangle means container has stopped. Our docker_algo container is now up and ready to be used.

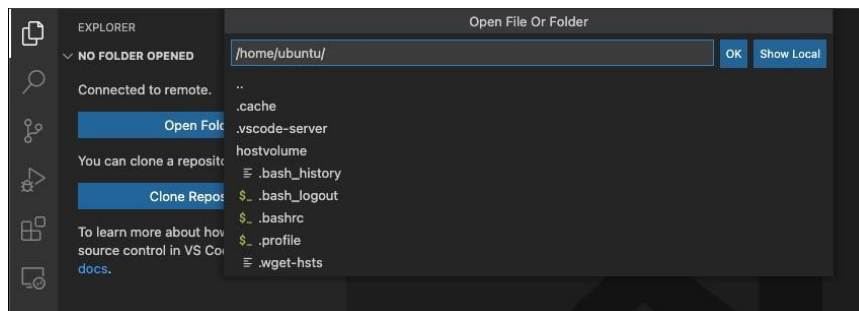


6. To work within the container, left-click on the docker_algo container and press “Attach Visual Studio Code”. This will open a new VS-Code window for the container.

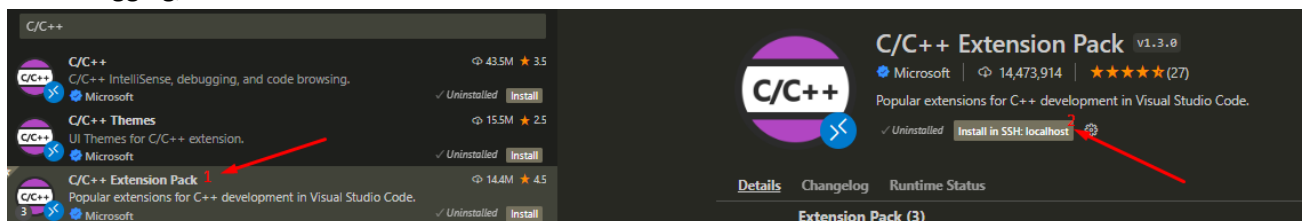


In-container set-up

When ready, check if you can open up your local folder from the container as follows (**Shortcut: CTRL+K+O**).



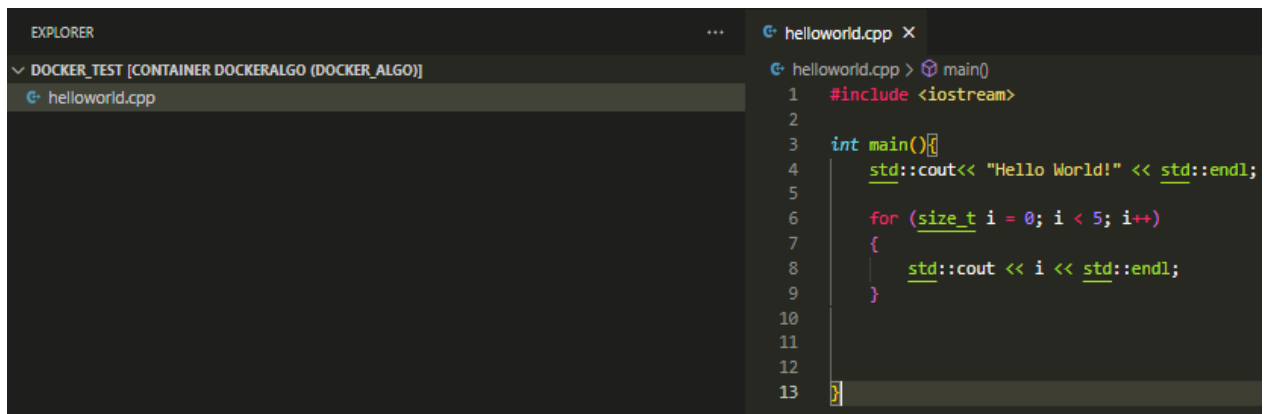
1. In the new window, we should install a few more extensions for C++ development. Search for C++ among the extensions and install **C/C++ Extension Pack**. This will install C/C++ Intellisense, debugging, themes.



2. Afterwards, you can copy your development files and paste into your `<dev_path>` and continue developing. Files within your local `<dev_path>` would be also available in the docker container's folder at the same time.

Developing in VS-Code

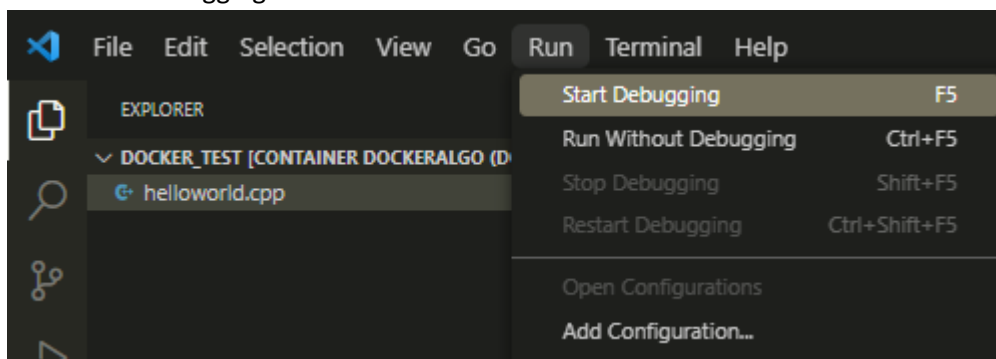
1. Press CTRL+K+O and navigate to `home/Ubuntu/hostVolume`. Create a folder named "docker_test" and open that folder with CTRL+K+O again. Create a "helloworld.cpp" within that folder like following:



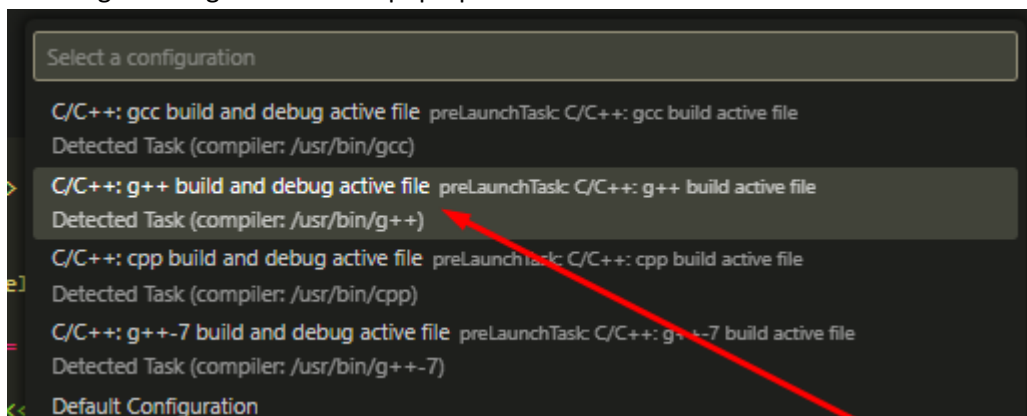
The screenshot shows the Visual Studio Code interface. On the left, the Explorer pane shows a folder named 'DOCKER_TEST [CONTAINER DOCKERALGO (DOCKER_ALGO)]' containing a file 'helloworld.cpp'. On the right, the Editor pane shows the contents of 'helloworld.cpp' with the following code:

```
1 #include <iostream>
2
3 int main()
4 {
5     std::cout << "Hello World!" << std::endl;
6
7     for (size_t i = 0; i < 5; i++)
8     {
9         std::cout << i << std::endl;
10     }
11 }
12
13
```

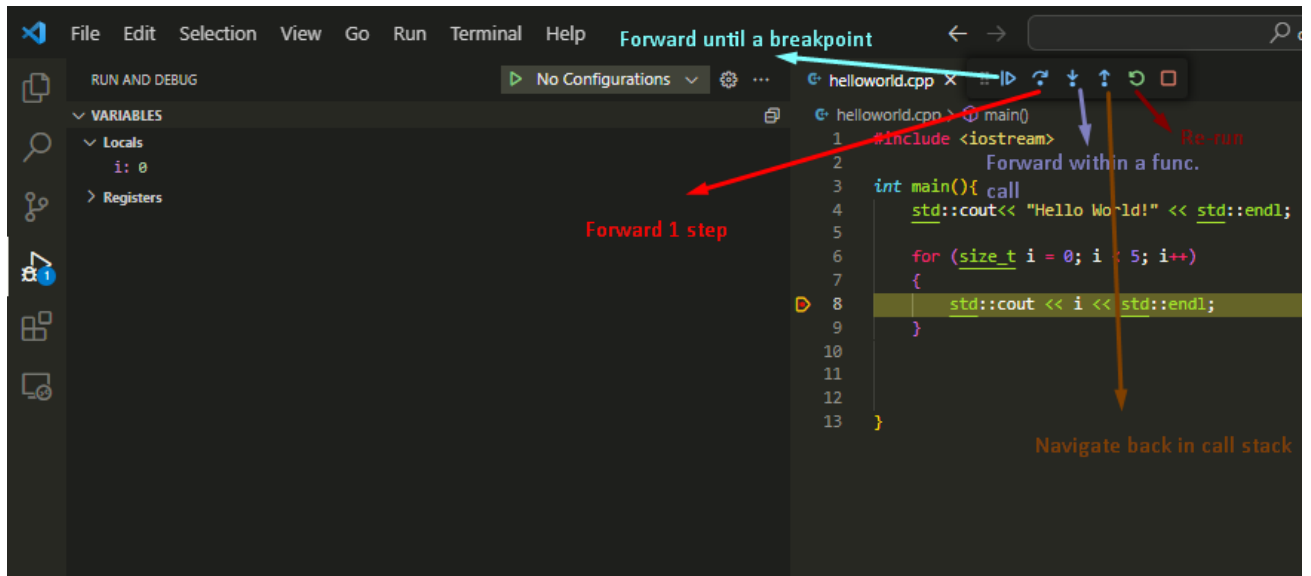
2. Press "Start debugging" from "Run" menu.



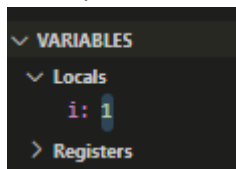
3. Choose g++ configuration in the pop-up window:



4. Place a breakpoint by clicking on the 8th line. Then start debugging again. When the breakpoint is reached, program stops at that state and shows the variables on the left pane. As can be seen, variable is 0 right now and it will be 1 in the next breakpoint encountering.



5. You can use several options when debugging and it has been shown with arrows on above image. When you forward the program until a new breakpoint is encountered, the variable would be 1.



What is inside the DockerFile?

You may ignore this part if you are not interested.

FROM ubuntu:bionic	Our Docker image is based on Ubuntu Bionic OS.
RUN apt update	Updates the package sources list with the latest version of the packages in the repositories.
RUN apt install openssh-server sudo -y	Necessary to make ssh connection in VSCode.
RUN useradd -rm -d /home/ubuntu -s /bin/bash -g root -G sudo -u 1000 test	
RUN usermod -aG sudo test	Enables ssh for connections.
RUN service ssh start	Creates a user and password for ssh connection
RUN echo 'test:test' chpasswd	
RUN apt update	
RUN apt install build-essential -y	Downloads GNU/g++ compilers, GNU debuggers and libraries for compiling.
RUN apt install gdb -y	
RUN apt update	
RUN apt install valgrind	Used for checking memory leak.
RUN apt update	
RUN apt install git -y	Installs git version control.
RUN apt update	
RUN apt install python3 -y	Python is needed for calico installation.
RUN apt install python3-pip -y	
RUN sudo update-alternatives --install /usr/bin/pip pip /usr/bin/pip3 1	Make pip command call pip3.
RUN pip3 install calico	Calico is needed for evaluating the code with cases.
EXPOSE 22	Enable a port to connect.
CMD ["/usr/sbin/sshd", "-D"]	