Development Environment Setup for MAC

Before starting with the installation, keep in mind that by default, MAC's virtualization is enabled. Yet, if it is for somehow disabled, then this article may help (for intel-based MACs). But this is hard to happen and there is no straight user-interface to enable/disable that setting.

1 XCode Command Line Tools Installation

The XCode Command Line Tools package facilitates the installation of open source software within Terminal, enabling users to utilize UNIX-style commands. It is self-contained and can be downloaded separately from XCode.

1. Check that the **XCode Command Line Tools** are installed on your computer by running the **"g++ -v"** command on a terminal window. You should see version information similar to Figure 1. Otherwise, please install XCode Command Line Tools using the **"xcode-select –install"** command (Figure 2). The installation may take a while. Once completed, check again with the **"g++ -v"** command and you should see the output in Figure 1.

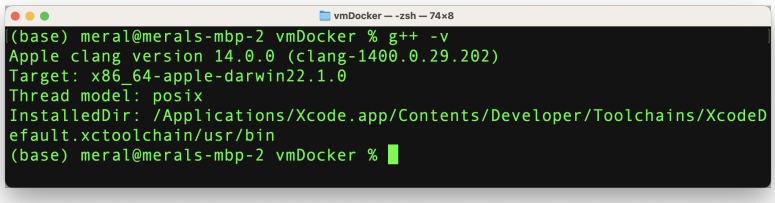

Figure 1: g++ Version Message

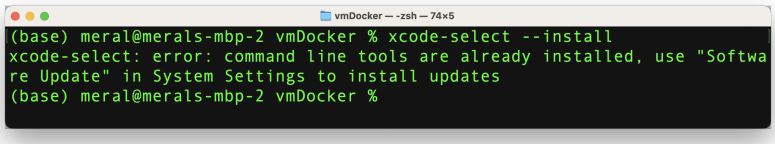

Figure 2: Install XCode Command Line Tools

2 Docker Installation

Docker is a platform for developing, shipping, and running applications which provides users with an abstraction from their local environment. Docker containers can be regarded as a fully functional and portable computing environment surrounding the application and keeping it independent from other parallel environments. Each container simulates a different software application and runs isolated processes

by bundling related configuration files, libraries and dependencies.For more information and detailed documentation, please visit Docker docs.

1. Download and install the appropriate version of Docker. You will need to choose a version depending on the processor of you computer (i.e. Intel or Apple Silicon).

2. When you install, you should receive a permission request for Docker to run background processes. After granting permission, run **"docker images ls"** (lists docker images) and **"docker ps -all"** (lists containers) to check that the daemon is running in the background. You should receive outputs similar to Figure 3. If this is your first installation, you won't find anything listed. These commands are only to make sure the Docker Daemon is working in the background.

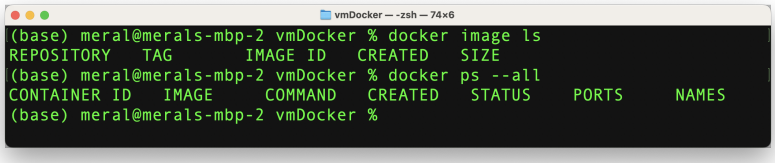

Figure 3: Docker Images and Containers

3 Docker Container Configuration

1. Create an empty folder in any directory and name it to your preference (for example: /Users/meral/Desktop/blg252e/vmDocker).

2. Download the Docker image **(dockerfile)** from Ninova and place it into this folder.

3. Open up a terminal window and navigate to the folder containing the dockerfile. Run the command: **"docker build -t dockerfile ."** Make sure to include the **"."** at the end of the command. Here "`dockerfile`" is the image name. The output should resemble that given in Figure 4. It could take a while to build the image.
   Note that it is safe to ignore the following warning messages:

   - WARNING: apt does not have a stable CLI interface. Use with caution in scripts.
   - debconf: delaying package configuration, since apt-utils is not installed.

4. You should now be able to see the newly created image under all docker images (Figure 5).

5. Create a directory **(<dev_path>)** that is visible to your container.

6. Run a command in the following format to boot a container instance from the image you have just built:

   "**docker run -p <local_port>:<container_port> -v**
   <dev_path>:<container_path> –name <container_name>
   –hostname <container_host_name> -d <image_name>"

   - **-p <local_port>:<container_port>:** Maps a port of the container to one of your PC ports. This is required to be able to connect into your container via SSH.
   - **-v <dev_path>:<container_path>** Maps the files in a local directory to a directory inside the container.
   - **–name <container_name>:** Provides name for the container.
   - **–hostname <container_host_name>:** Provides name for the host.
   - **-d <image_name>:** Provides the Docker image.

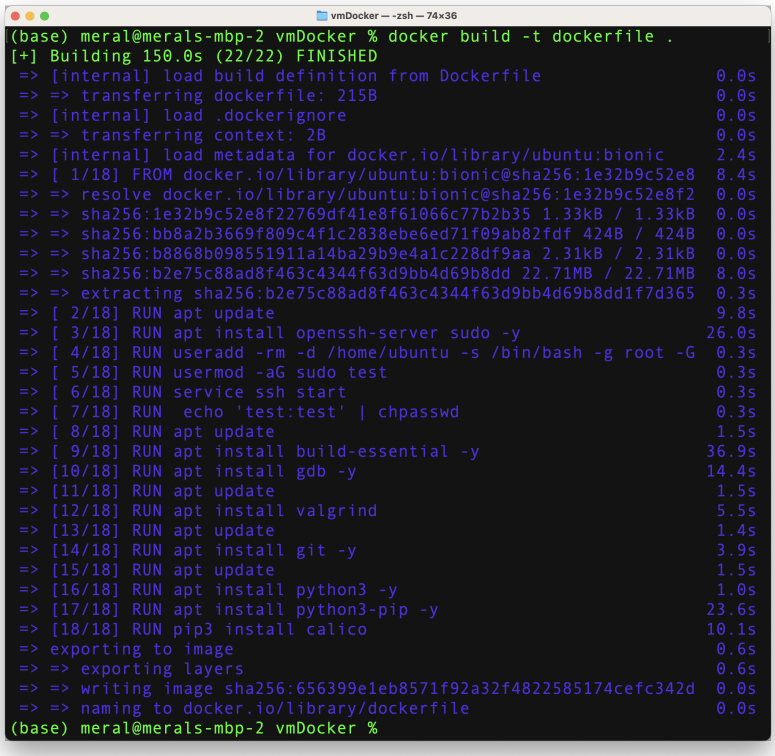

Figure 4: Docker Image Build Success

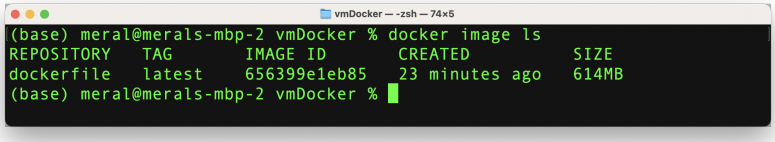

Figure 5: Docker Images

Once the container is successfully created (Figure 6), it should be visible when all containers are listed ("`docker ps -all`" Figure 7). An example command for creating a container should resemble:

```
docker run -p 2222:22 -v /Users/dgknrsln/Desktop/vm_docker/:
/home/ubuntu/hostvolume --name vm_docker
--hostname vm_docker -d dockerfile
```

7. Check if you are able to connect to your container via SSH using a command in the following format: **"ssh test@localhost -p <local_port>"**. For the example input provided above, this command would be: **"ssh test@localhost -p 2222"** (Figure 8). Both the username and password are initialized to "test". You can modify them by editing the dockerfile, rebuilding the image, and re-running the container. Don't forget to exit from the SSH session (`exit`).

> Note: In this step, if you encounter a warning stating "WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!", you need to set a different port number while booting up the docker instance after deleting it on Docker Desktop. This happens since you are trying to connect to a previous instance by using the same port, same address, and same username.

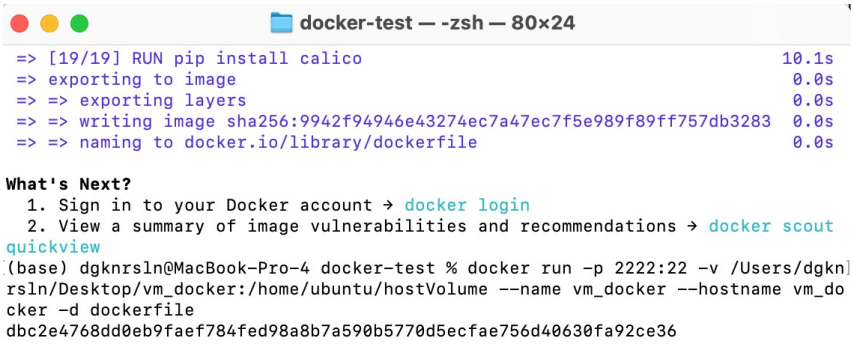

Figure 6: Docker Container is Created Successfully

Figure 7: Docker Container is Visible Under Container List

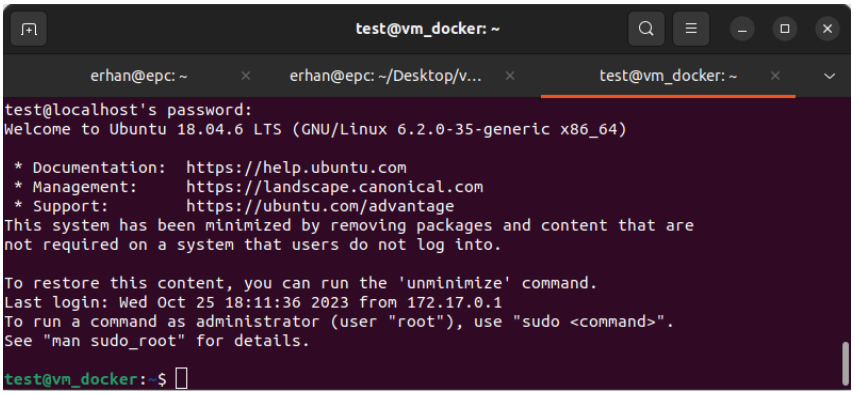

Figure 8: SSH Connection to Container

4 VSCode Installation and Configuration

4.1 Installing Extensions

1. Install `Visual Studio Code` in your system. Visit here for installation instructions and FAQ.
2. Run `Visual Studio Code` and install the extension named `"Remote SSH"` (Figure 9).

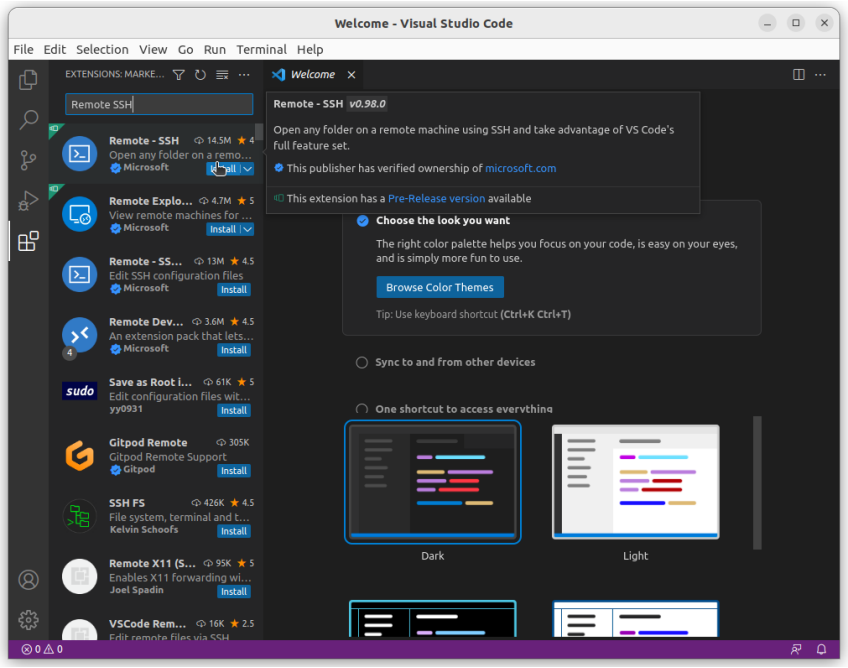

Figure 9: Installing Remote SSH extension in VS Code.

3. Install "Docker" extension on VS-Code (Figure 10).

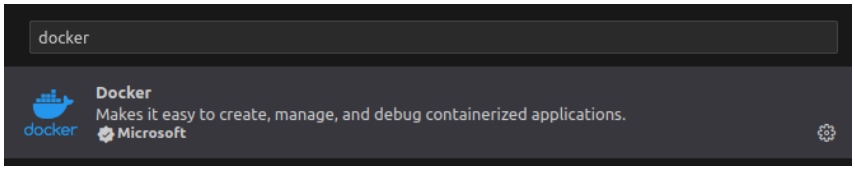

Figure 10: Installing Docker extension in VS Code.

4. To attach Visual Studio Code on Docker containers, you need to install “Dev Containers” (formerly, remote containers extension). See Figure 11:

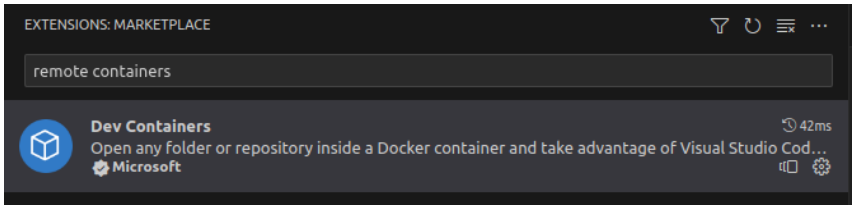

Figure 11: Installing Dev Containers extension in VS Code.

5. From now on, you can work with containers in VS-Code in 2 different ways: (1) Connecting with SSH within VS-Code, (2) Connecting with Docker extension. Choose which way to follow and navigate to the relevant instructions below. After that, follow instructions in 4.4 "In-container set-up" and so on.

4.2 Connecting with SSH within VS-Code

1. Click on the "Remote Explorer" button on the left pane and start a connection with localhost (Figure 12).

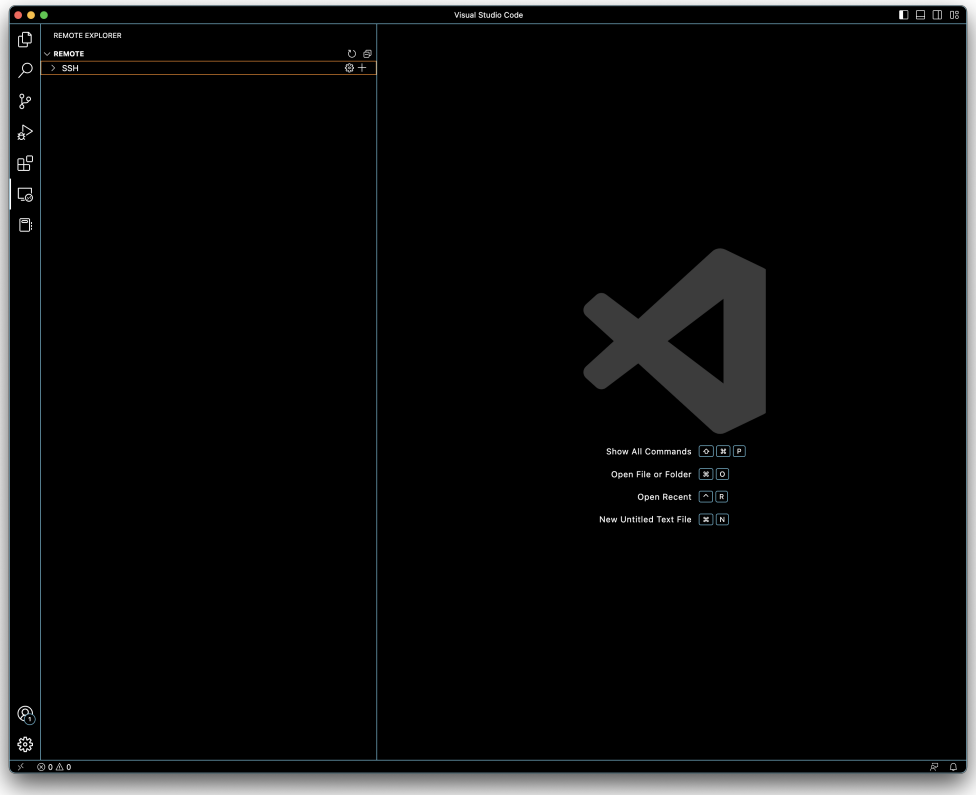

Figure 12: Navigate to Remote Explorer

2. If there is no SSH target, you can add one by clicking on the + button next to the SSH section (Figure 13).
3. Select an SSH configuration file (Figure 14).
4. Click the arrow button next to localhost to start open the remote computer in the same window (Figure 15).
5. Enter the password for the remote computer. If the initial settings aren't changed, it should be **"test"** (Figure 16).

4.3 Connecting with Docker extension

1. 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 (Figure 17).

   If containers and images are not showing up and error message is shown as "permission denied". Your user might not have the permission to access docker.

2. 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 (Figure 18).

4.4 In-container set-up

1. In the new window, install a few more extensions for C++ development. Search for **"C++"** and install "C/C++ Extension Pack" (Figure 19).

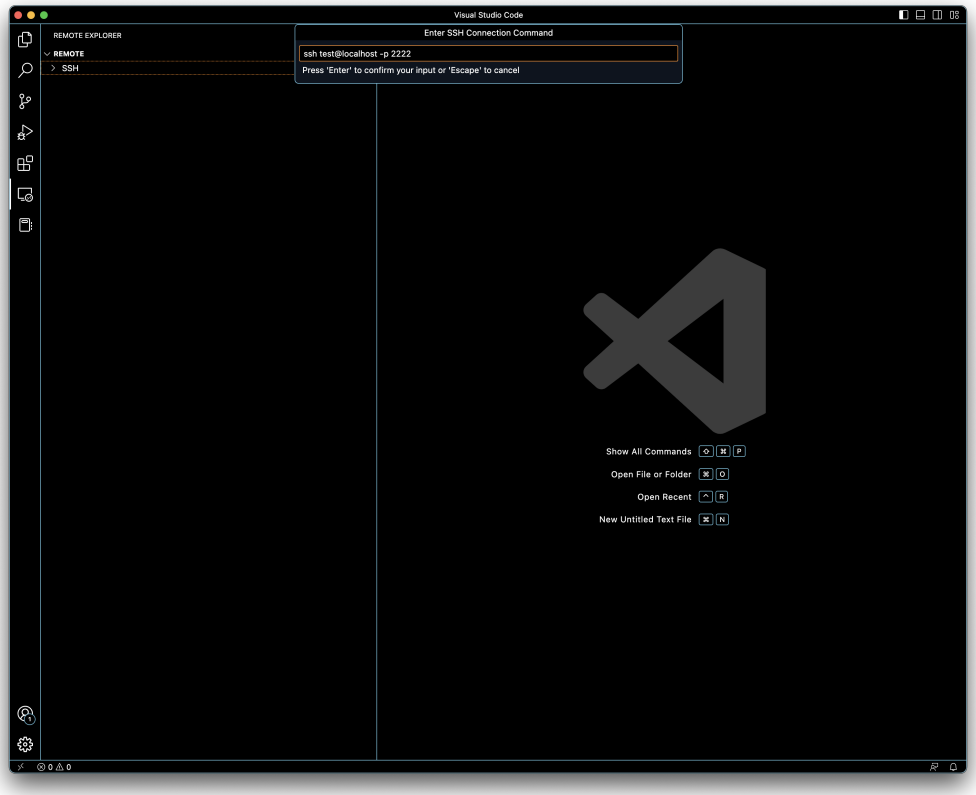

Figure 13: Add Connection

2. Click on the File Explorer at the top of the left pane and click **"Open Folder"** to view files in the **(<dev_path>)** (Figure 20).

3. Choose **"/home/ubuntu/hostvolume"** as the current working directory (Figure 21).

4. After selecting the working directory, you can test by creating a dummy file on your local computer. For the example provided above, a test file was created under **"/Users/dgknrsln/Desktop/vm_docker/"**.

5. This file should be visible on the remote computer (Figure 22).

5 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 in Figure 23

2. Press "Start debugging" from "Run" menu (Figure 24).

3. Choose g++ configuration in the pop-up window (Figure 25).

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 (Figure 26).

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 (Figure 27).

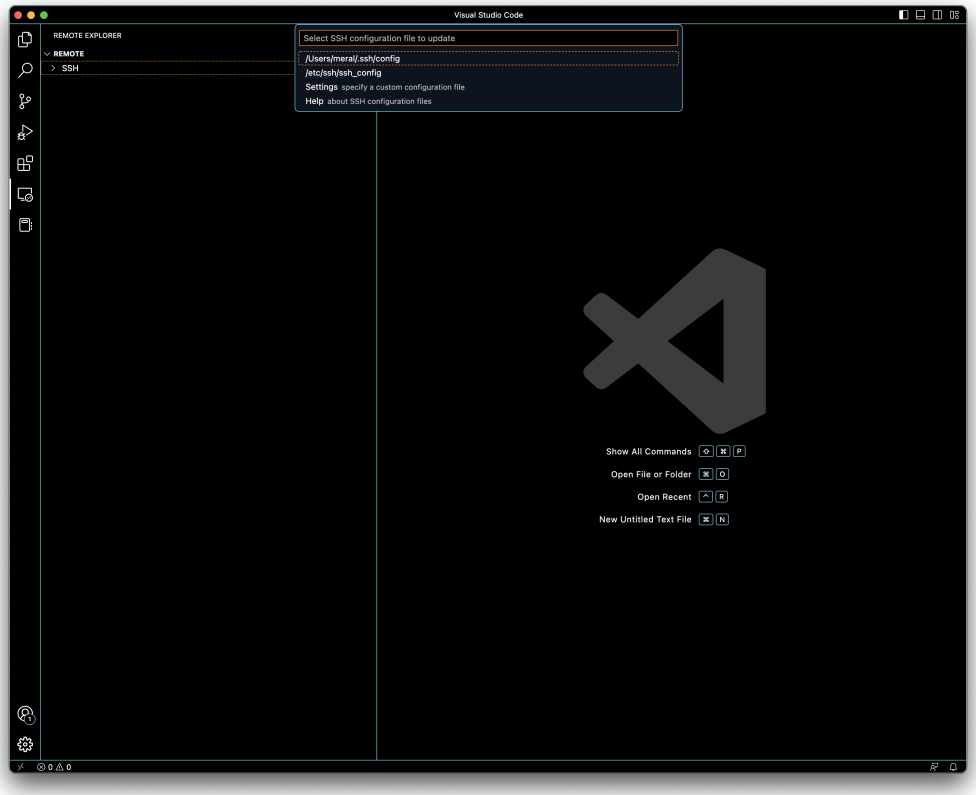

Figure 14: Specify SSH Configuration File

6 What is inside the Dockerfile?

See Figure 28. *You may ignore this part if you are not interested.*

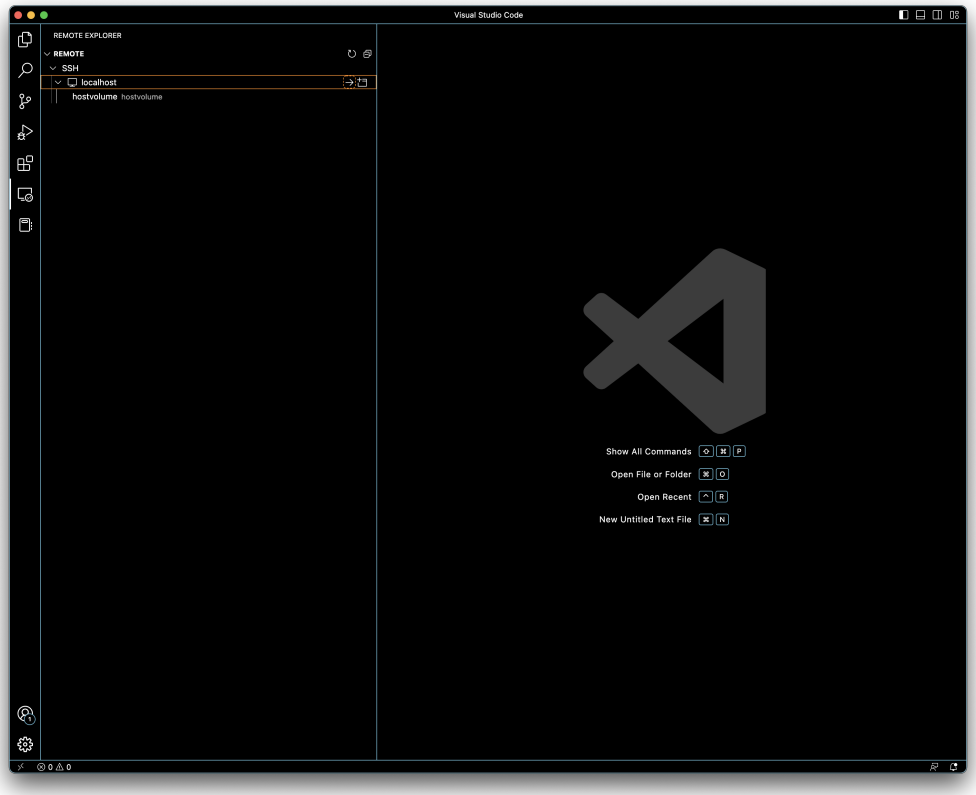

Figure 15: Open Remote Computer

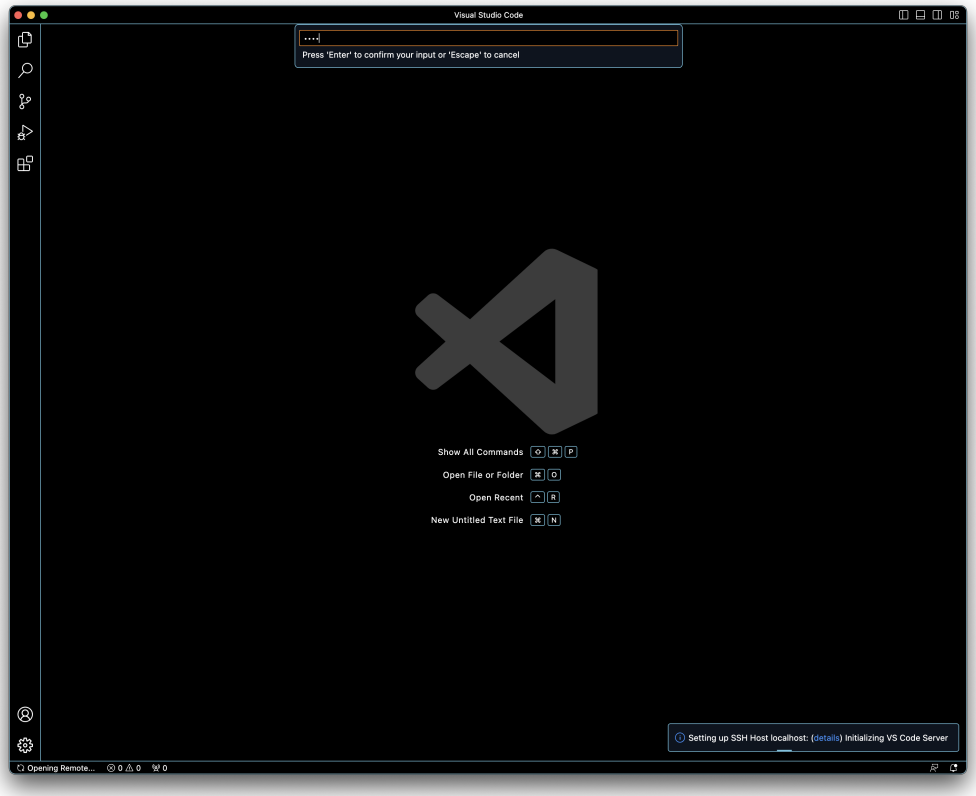

Figure 16: Enter Password for Remote Computer

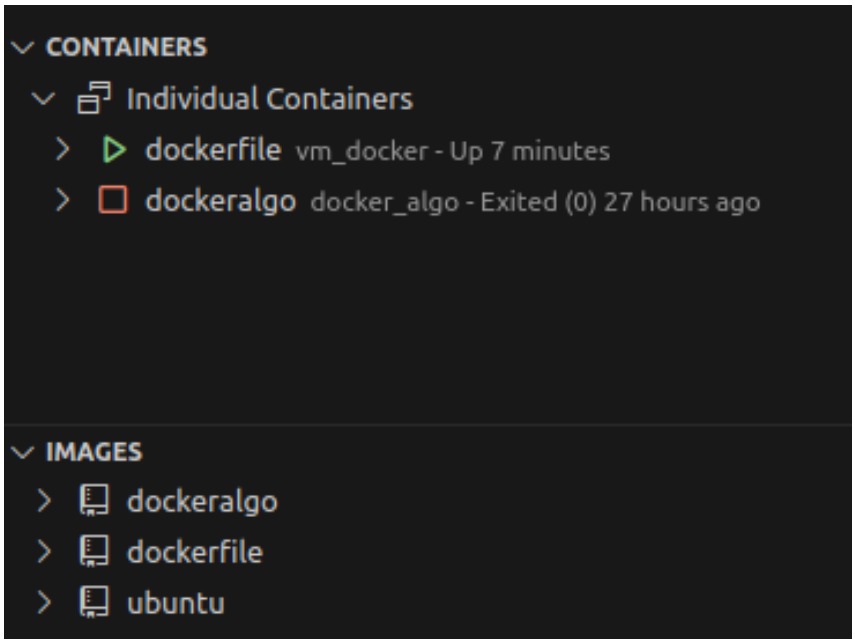

Figure 17: Container can be seen through docker extension.

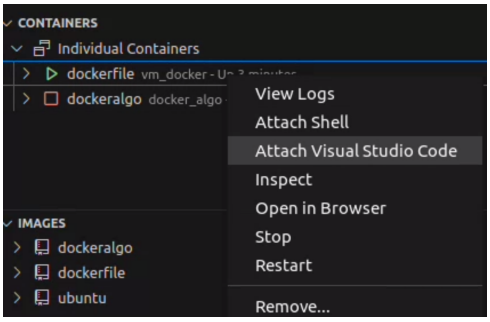

Figure 18: A new window will be opened within the container.

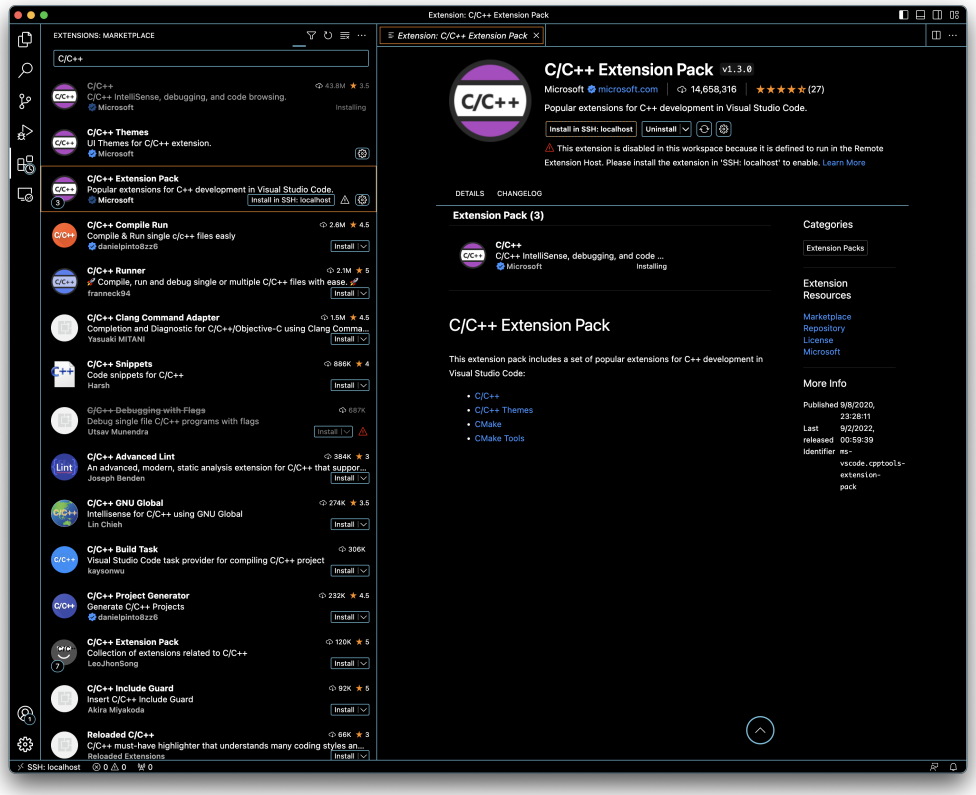

Figure 19: Install C/C++ Extension Pack

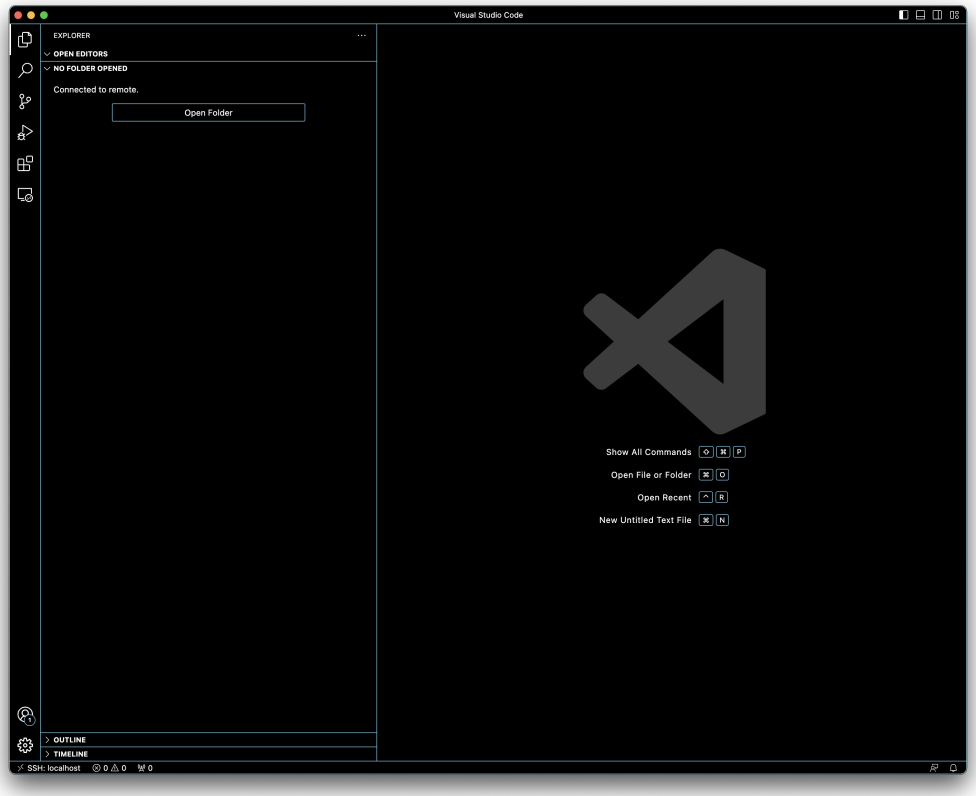

Figure 20: Open Folder

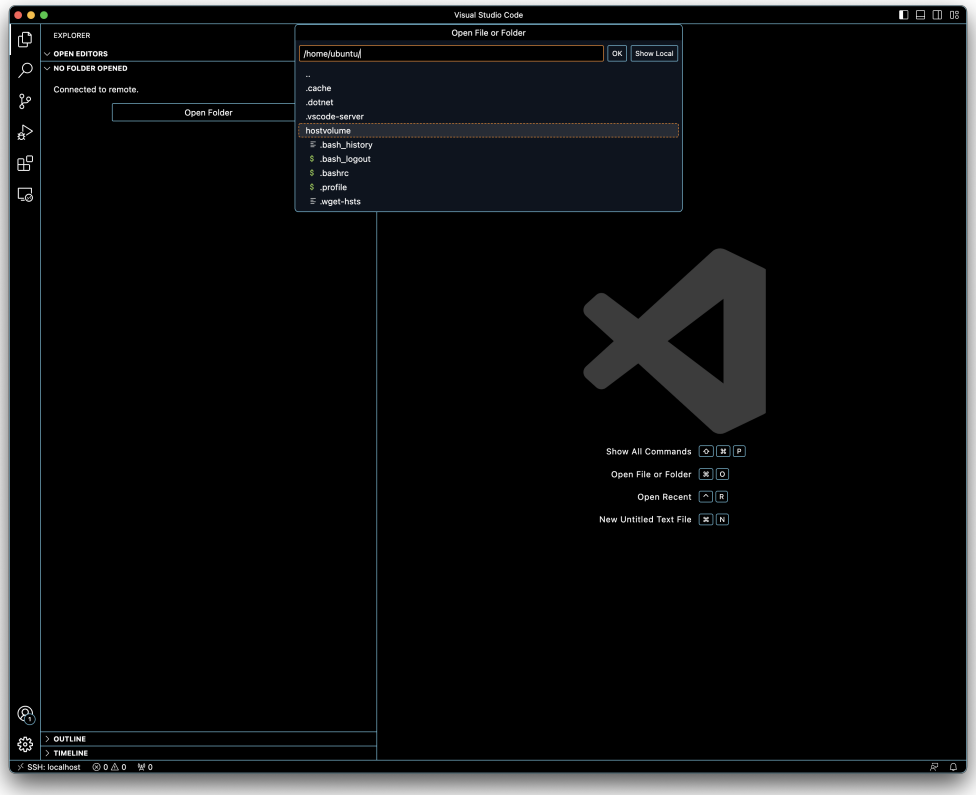

Figure 21: Choose /home/ubuntu/hostvolume

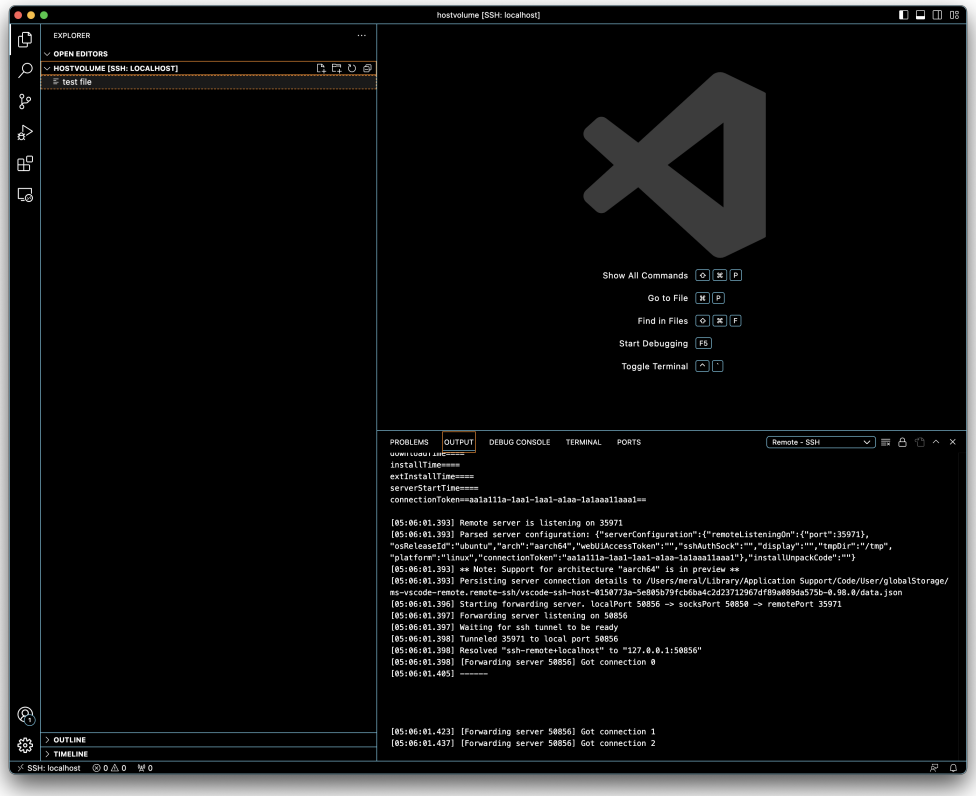

Figure 22: View Local Files on Remote Computer

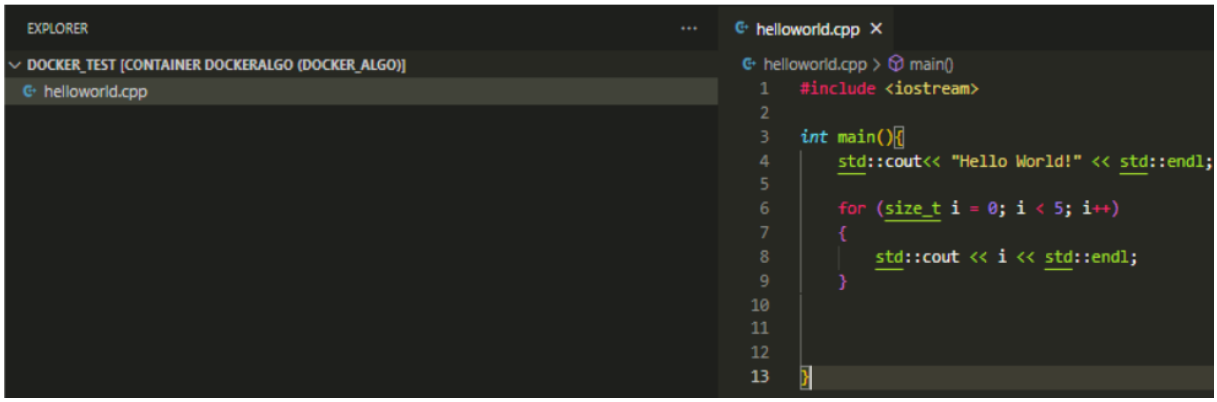

Figure 23: helloworld.cpp

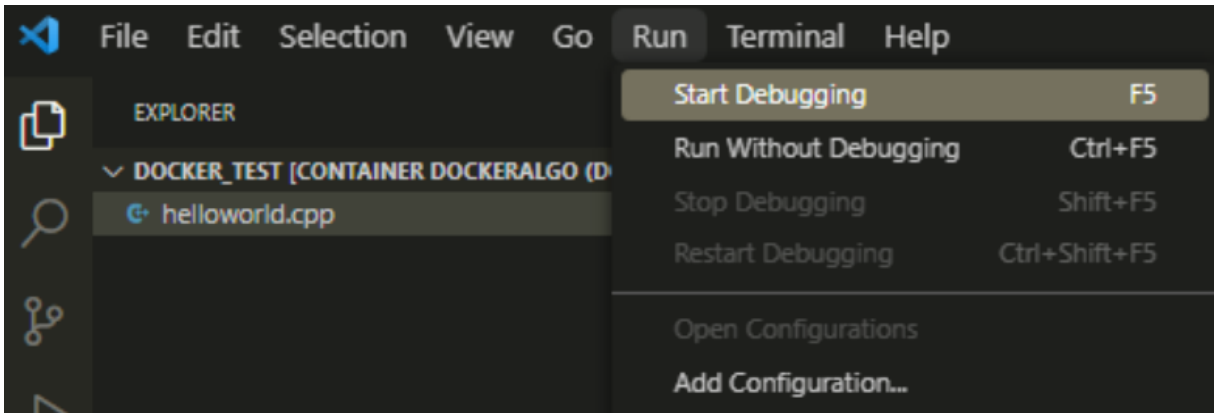

Figure 24: Run with debugging option

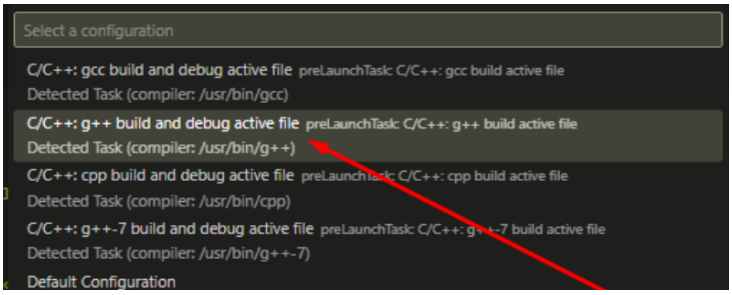

Figure 25: g++ for building the .cpp files

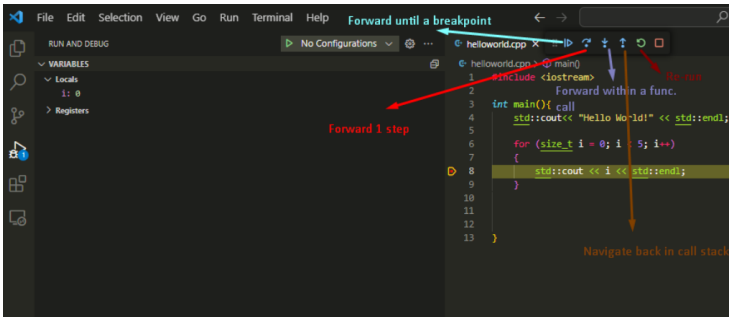

Figure 26: Debugging instructions

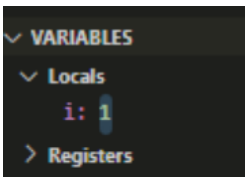

Figure 27: change in value of variable can be seen through debugging pane

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 RUN useradd -rm -d /home/ubuntu -s /bin/bash -g root -G sudo -u 1000 test RUN usermod -aG sudo test	Necessary to make ssh connection in VSCode.
RUN service ssh start	Enables ssh for connections.
RUN echo 'test:test' | chpasswd	Creates a user and password for ssh connection
RUN apt update RUN apt install build-essential -y RUN apt install gdb -y	Downloads GNU/g++ compilers, GNU debuggers and libraries for compiling.
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 RUN apt install python3-pip -y	Python is needed for calico installation.
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 CMD ["/usr/sbin/sshd","-D"]	Enable a port to connect.

Figure 28: Dockerfile