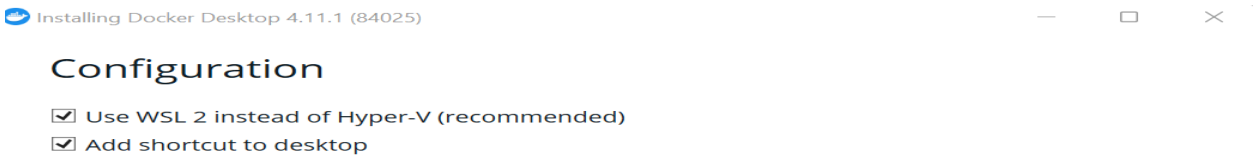
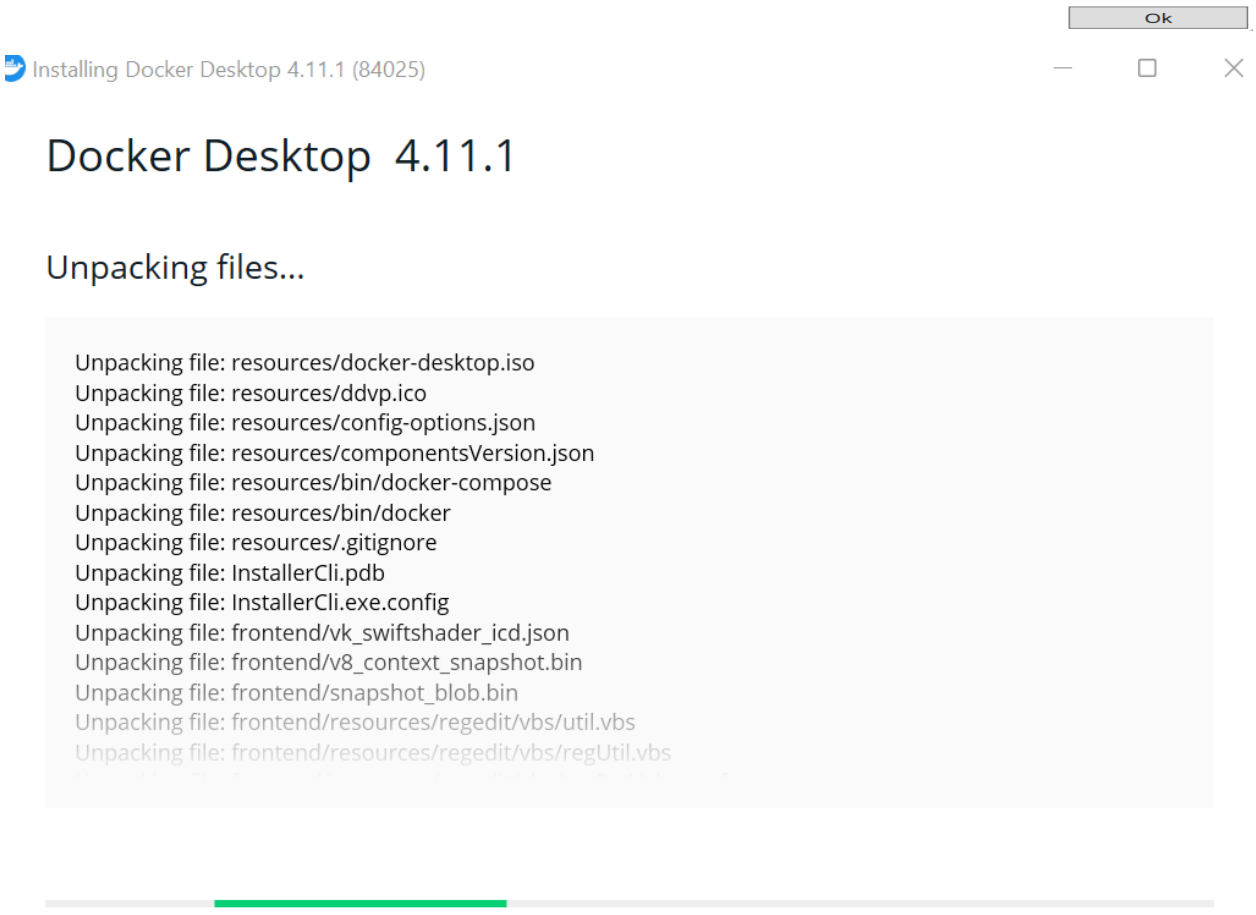
Codio Activity: Installing Docker Containers

Install Docker:

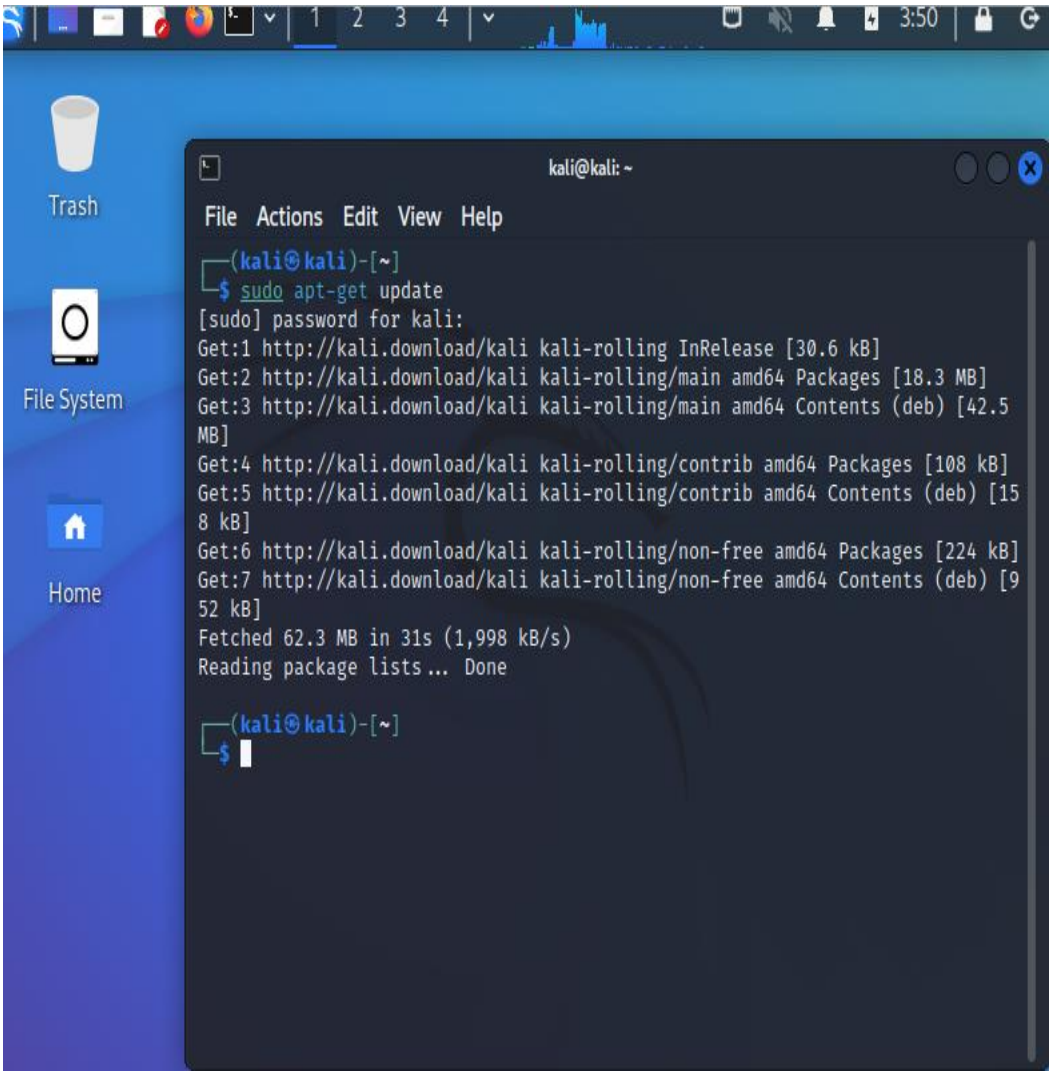




Required Virtual Box and Ubuntu

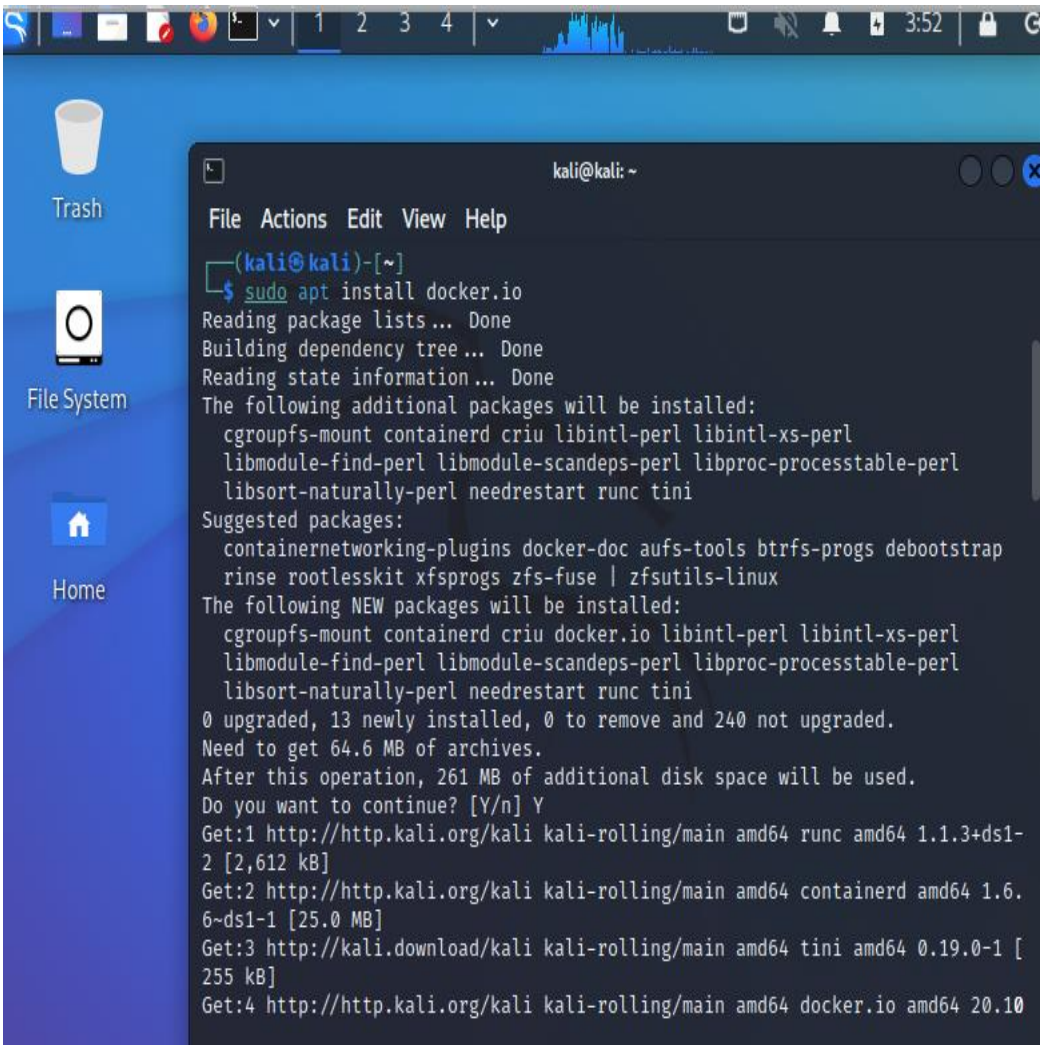
1. Perform a standard update to ensure you are using the latest libraries:

sudo apt-get update



2. Install Docker from the standard repository:

sudo apt install docker.io



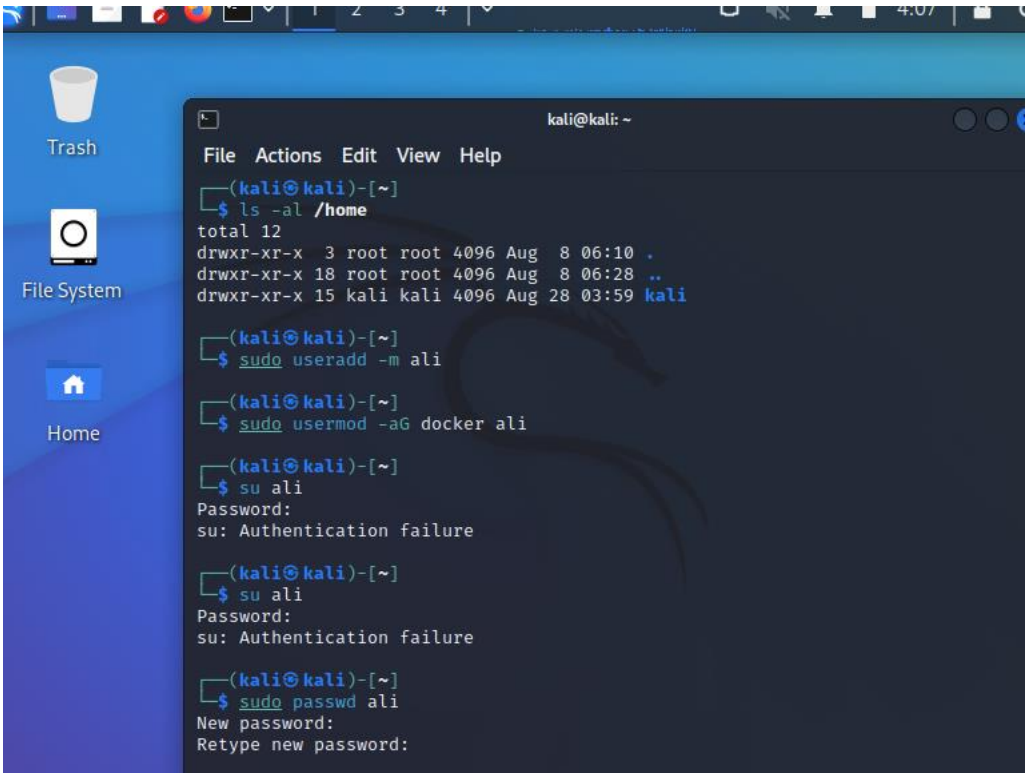
3. As part of the installation process, Docker will create a new group (called docker). You need to add your username to that group:

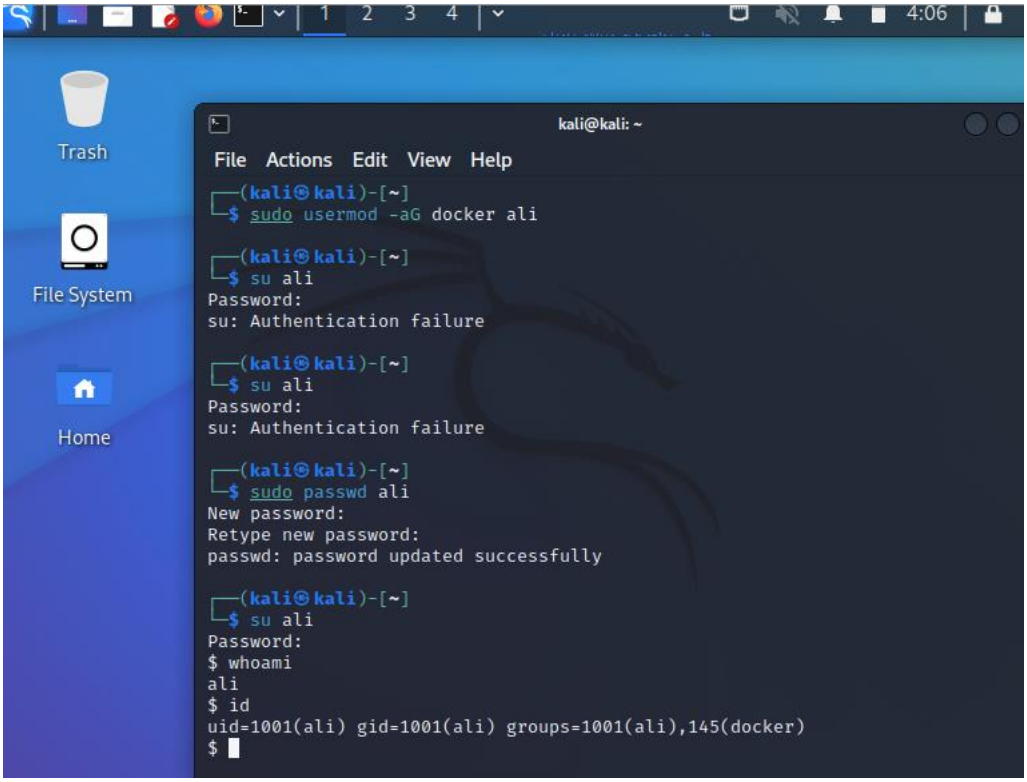
sudo usermod -aG docker [ali]

4. Log out and then back in to update your user status.

5. Check that you are a member of the docker group:

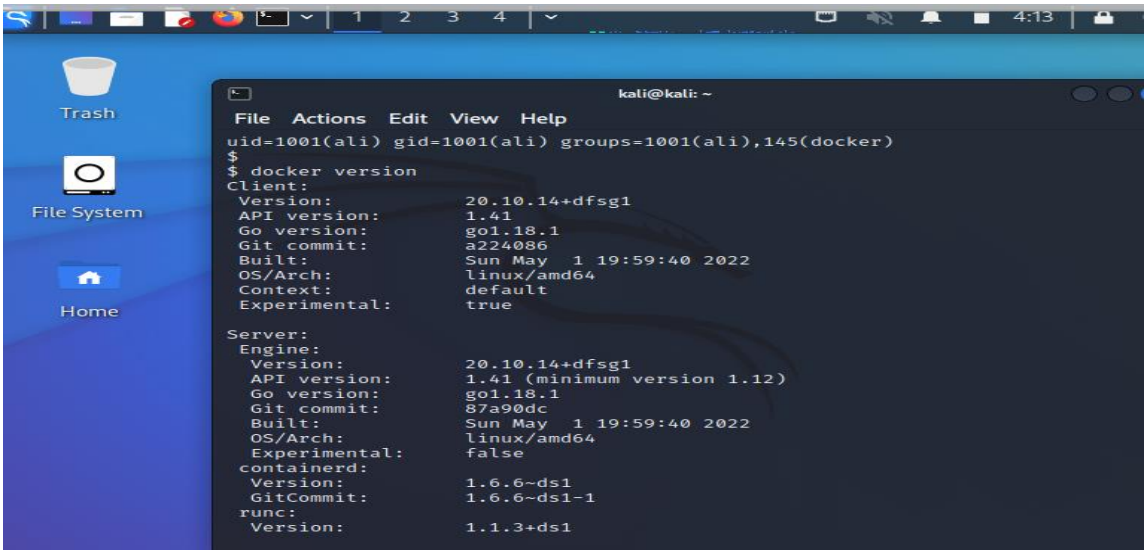
id (produces the following output:


Trash
File System
Home
kali@kali: ~
File Actions Edit View Help
┌──(kali㉿kali)-[~]
└─$ ls -al /home
total 12
drwxr-xr-x 3 root root 4096 Aug 8 06:10 .
drwxr-xr-x 18 root root 4096 Aug 8 06:28 ..
drwxr-xr-x 15 kali kali 4096 Aug 28 03:59 kali
┌──(kali㉿kali)-[~]
└─$ sudo useradd -m ali
┌──(kali㉿kali)-[~]
└─$ sudo usermod -aG docker ali
┌──(kali㉿kali)-[~]
└─$ su ali
Password:
su: Authentication failure
┌──(kali㉿kali)-[~]
└─$ su ali
Password:
su: Authentication failure
┌──(kali㉿kali)-[~]
└─$ sudo passwd ali
New password:
Retype new password:

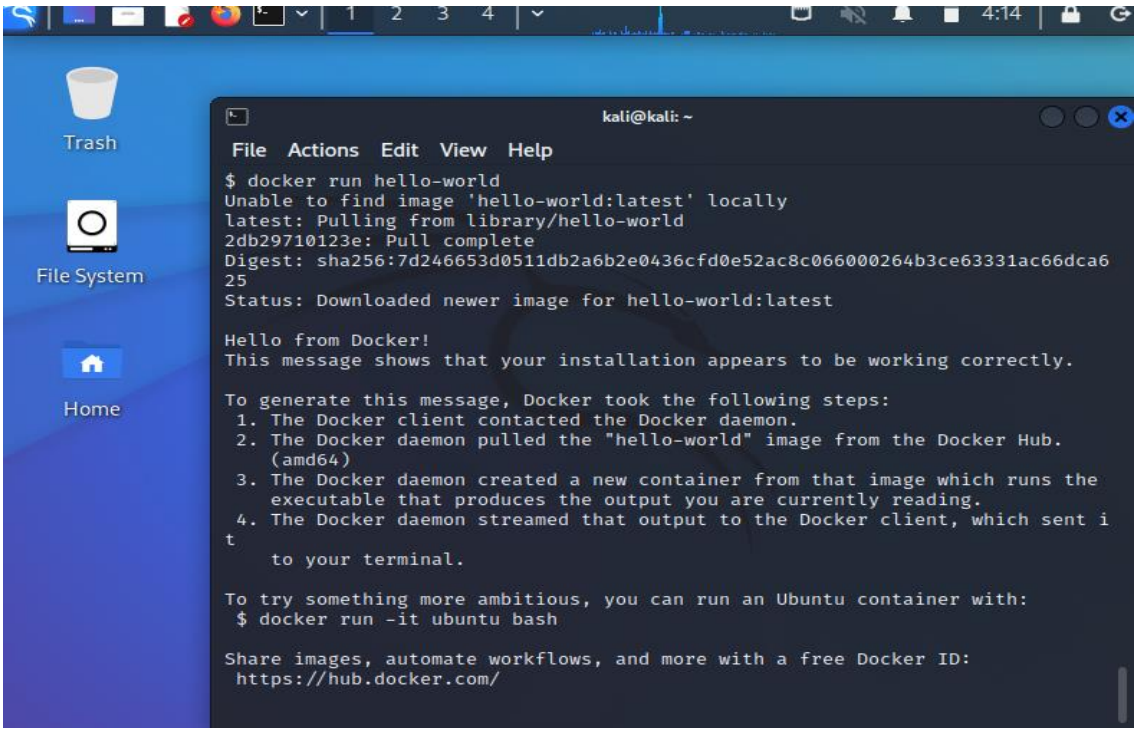

Trash
File System
Home
kali@kali: ~
File Actions Edit View Help
┌──(kali㉿kali)-[~]
└─$ sudo usermod -aG docker ali
┌──(kali㉿kali)-[~]
└─$ su ali
Password:
su: Authentication failure
┌──(kali㉿kali)-[~]
└─$ su ali
Password:
su: Authentication failure
┌──(kali㉿kali)-[~]
└─$ sudo passwd ali
New password:
Retype new password:
passwd: password updated successfully
┌──(kali㉿kali)-[~]
└─$ su ali
Password:
$ whoami
ali
$ id
uid=1001(ali) gid=1001(ali) groups=1001(ali),145(docker)
$

6. Issue the version command to check that docker is working properly:

docker version (produces the output shown below:

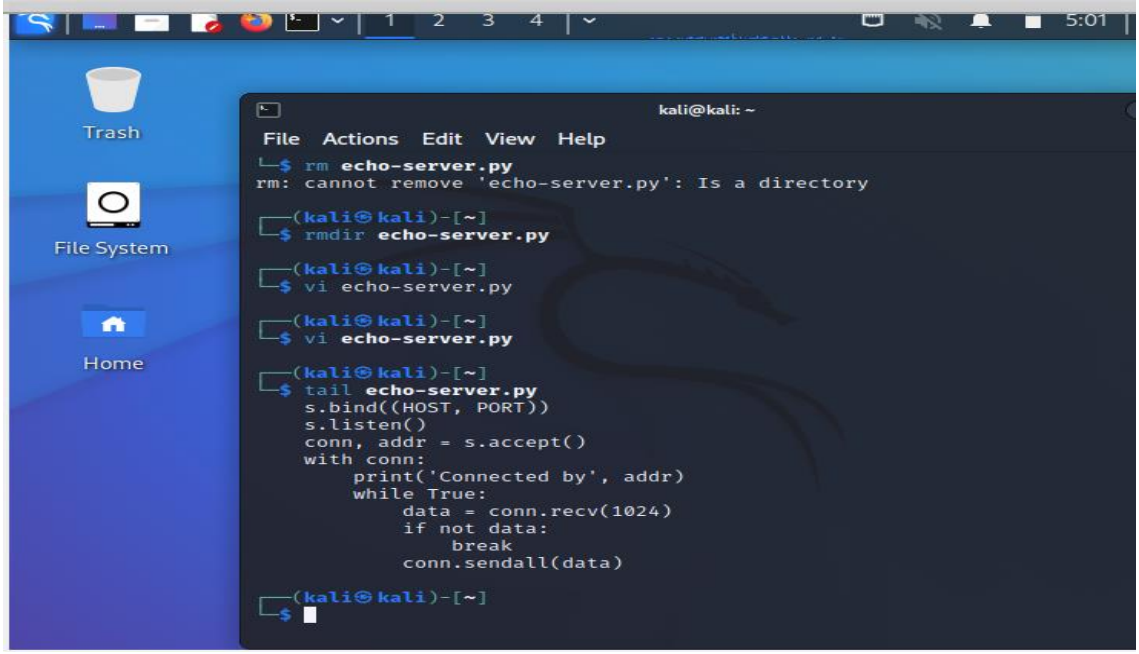


7. You can also run the “docker run hello-world" command to test a (very simple) container.

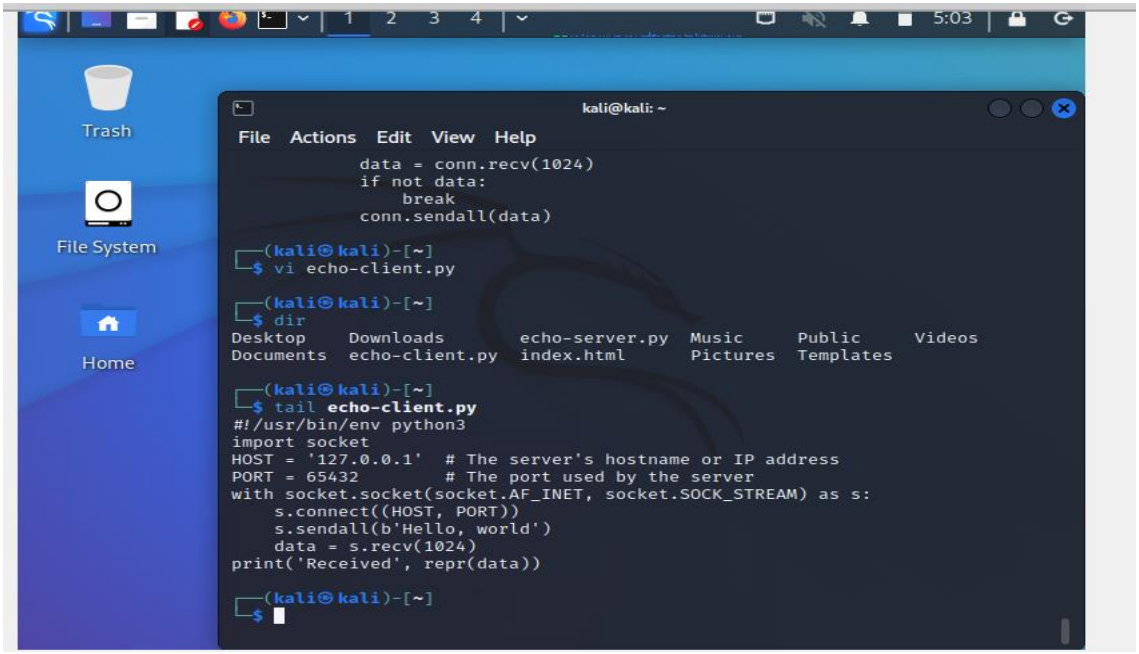


Codio Activity: Socket Programming

Copy the following code into a file named echo-server.py



Copy the following code into a file named echo-client.py



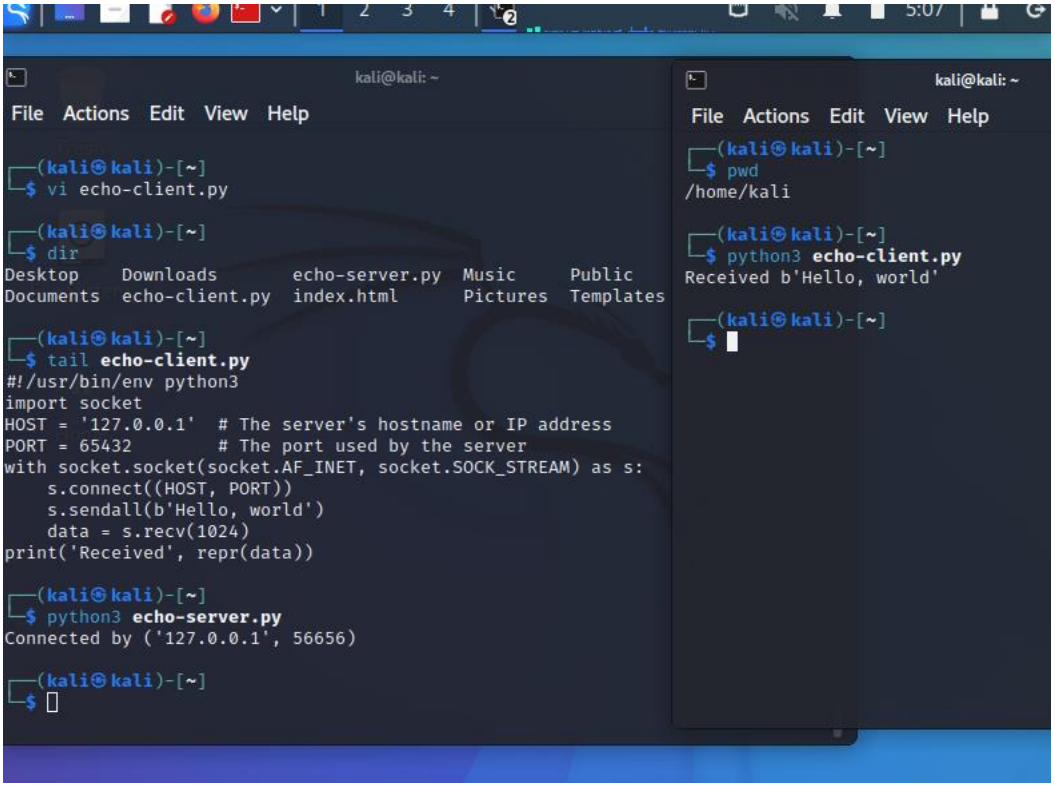
Open a terminal. Start the server by running the following command in a terminal:

```
python3 echo-server.py
```

Open a second terminal. Start the client by running the following command in the terminal:

```
python3 echo-client.py
```

The client and server will now talk with one another.



Question 1

In relation to echo-server.py, what is achieved using the command:

```
s.bind((HOST, PORT))?
```

making a Reverse Shell. and in the server.py file i got this error. i has trying in de socket_bind() s.bind((host, port))

Question 2

In relation to echo-server.py, what is achieved using the command:

s.connect((HOST, PORT))?

Used to connect to the remote socket with connect.

Codio Activity: Producer-Consumer Mechanism

Producer/Consumer Problem (also known as the 'bounded buffer' problem):

- A 'producer' is producing items at a particular (unknown and sometimes unpredictable) rate.
- A 'consumer' is consuming the items – again, at some rate.

For example, a producer-consumer scenario models an application producing a listing that must be consumed by a printer process, as well as a keyboard handler producing a line of data that will be consumed by an application program. This is shown in the picture below (Shene, 2014).

Items are placed in a buffer when produced, so:

- Consumer should wait if there isn't an item to consume
- Producer shouldn't 'overwrite' an item in the buffer

Synchronisation is necessary because:

- If the consumer has not taken out the current value in the buffer, then the producer should not replace it with another.
- Similarly, the consumer should not consume the same value twice.

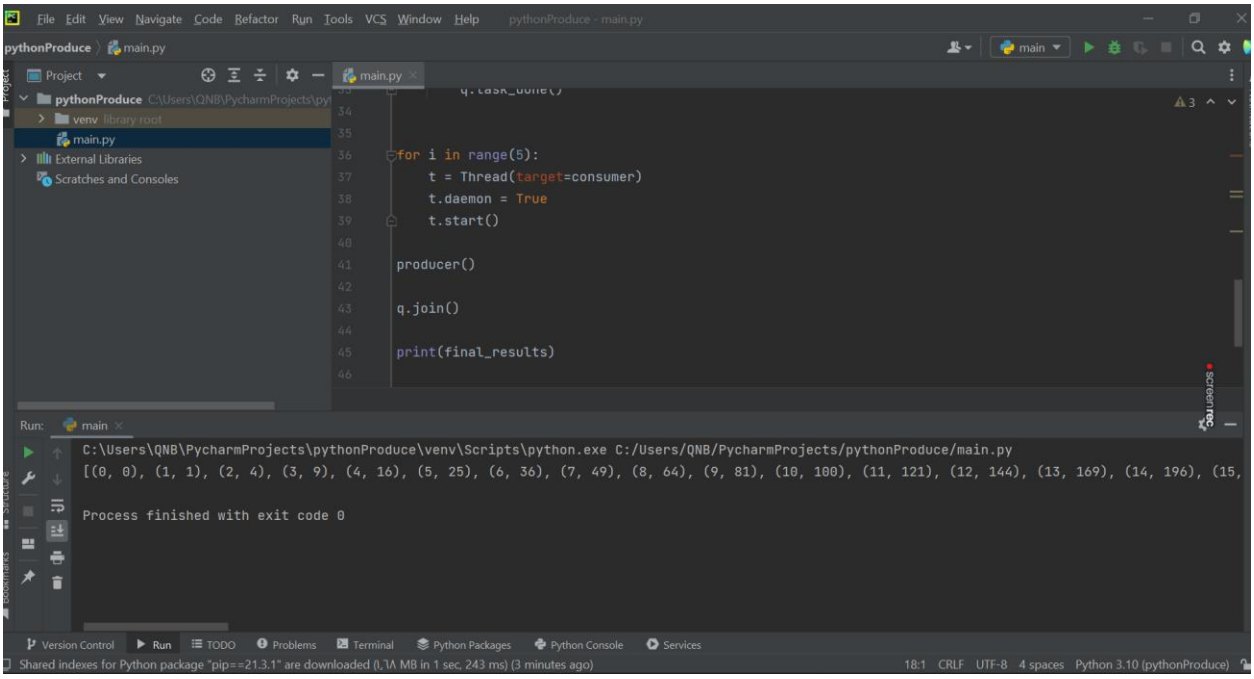
Task

Run producer-consumer.py in the provided Codio workspace (**Producer-Consumer Mechanism**), where the queue data structure is used.

A copy of the code is available here for you.

```
# code source: https://techmonger.github.io/55/producer-consumer-python/

from threading import Thread
```

```
from queue import Queue

q = Queue()
final_results = []

def producer():
    for i in range(100):
        q.put(i)


def consumer():
    while True:
        number = q.get()
        result = (number, number**2)
        final_results.append(result)
        q.task_done()


for i in range(5):
    t = Thread(target=consumer)
    t.daemon = True
    t.start()

producer()

q.join()

print (final_results)
```



Answer the following questions:

1. How is the queue data structure used to achieve the purpose of the code?

It's used to fill the produce data queue – in this case the numbers from 0 to 99. The program can then generate multiple consumer (threads) to process them.

2. What is the purpose of q.put(I)?

This is to fill the queue with the generated produce (this is from 0 to 99).

3. What is achieved by q.get()?

To get the contents of the queue, this is first in first out, thus the multiple instantiated consumer threads will process them from 0 to 99.

4. What functionality is provided by q.join()?

q.join prevents the python program to exit until all instantiated threads are finished (e.g. 0 to 99 produce has been consumed/calculated - ^2).

5. Extend this producer-consumer code to make the producer-consumer scenario available in a secure way. What technique(s) would be appropriate to apply? I don't get this question, by secure other threads must not know what others are working on? In this case we can randomize the producer generated number and/or implement encryptions (public/private keys).