Configuring Multiple Instances

As part of homework -2, I implemented a database system for a university with multiple Postgres instances using Docker.

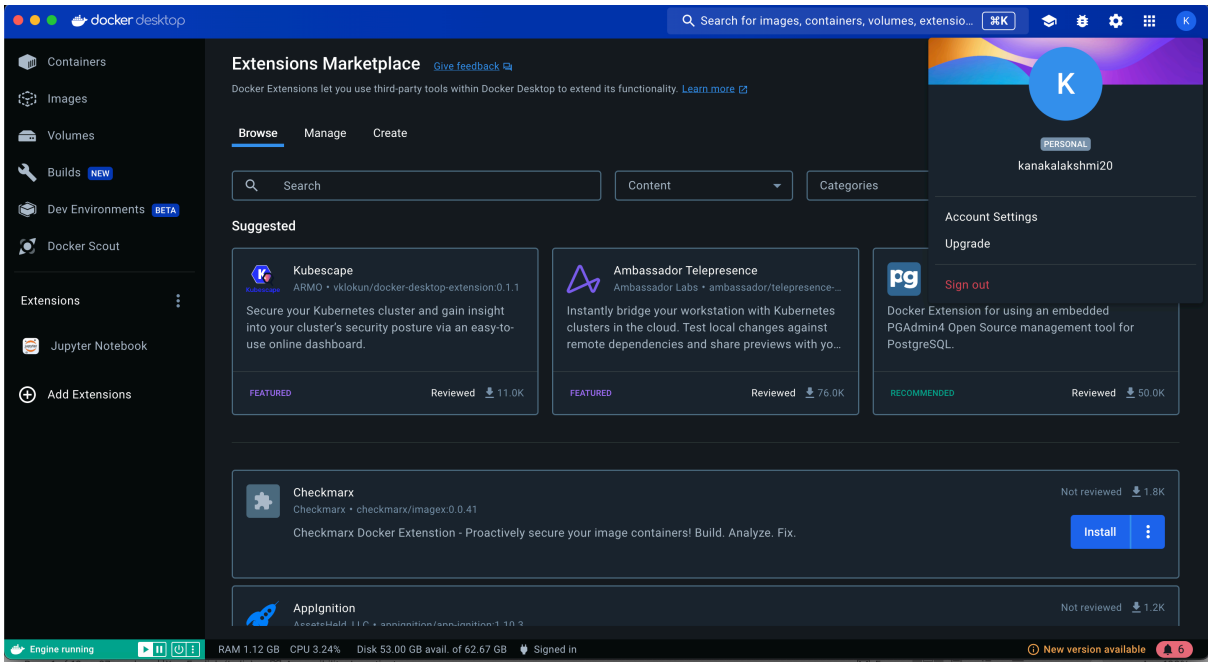

Fig-1: Docker dashboard

Explanation: This is the docker dashboard. We can download the required applications and start working on it. This is widely used when we are integrating with multiple technologies. As part our homework, it required multiple postgres instances, so I found docker is the best choice to implement it easily.

Command:

```
docker run --name primary_node -p 5432:5432 -e POSTGRES_PASSWORD=password -v
/Users/lk/Documents/lakshmi_DPDB/HW-
2/data_base_volume_mounts/primary:/var/lib/postgresql/data -d postgres
```

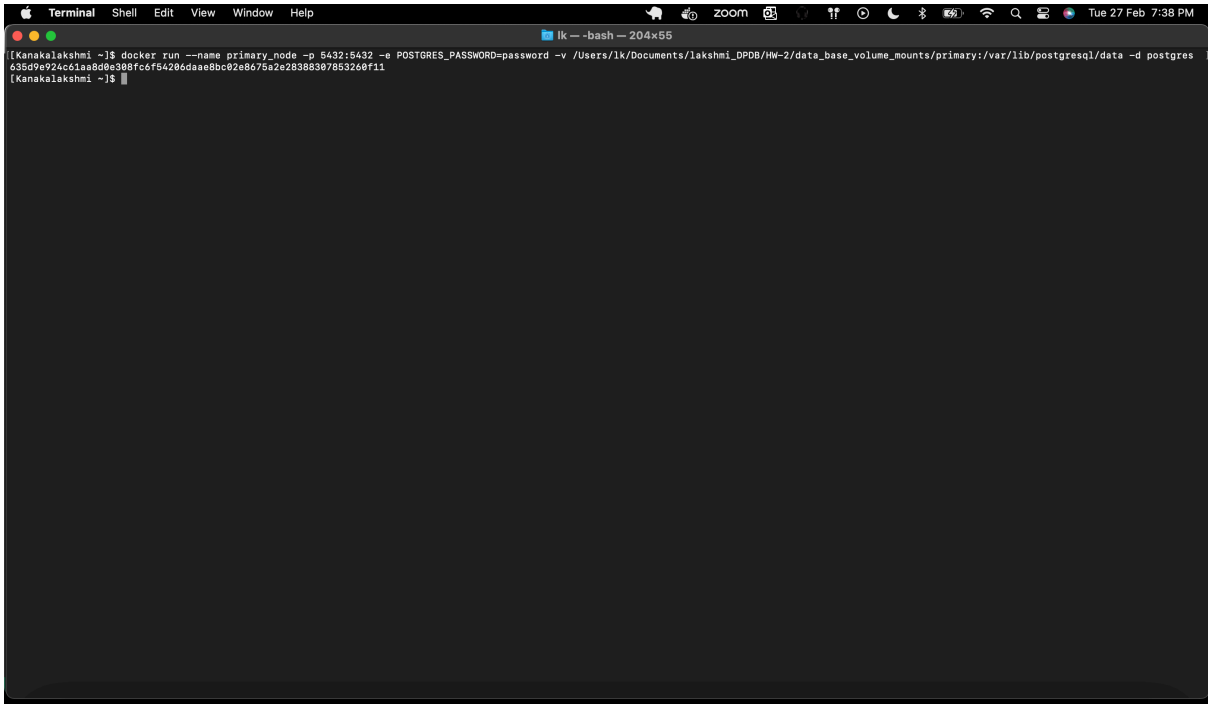

Fig-2: Pulled the docker image.

Explanation: We need to pull the postgres image as we want to work postgres. While pulling the postgres image, we need to provide the name, port_no, password. We are also creating the data_base_volume_mounts, which will be helpful to configure the replica node.

Command:

```
docker ps
docker exec –it primary_node bash
```

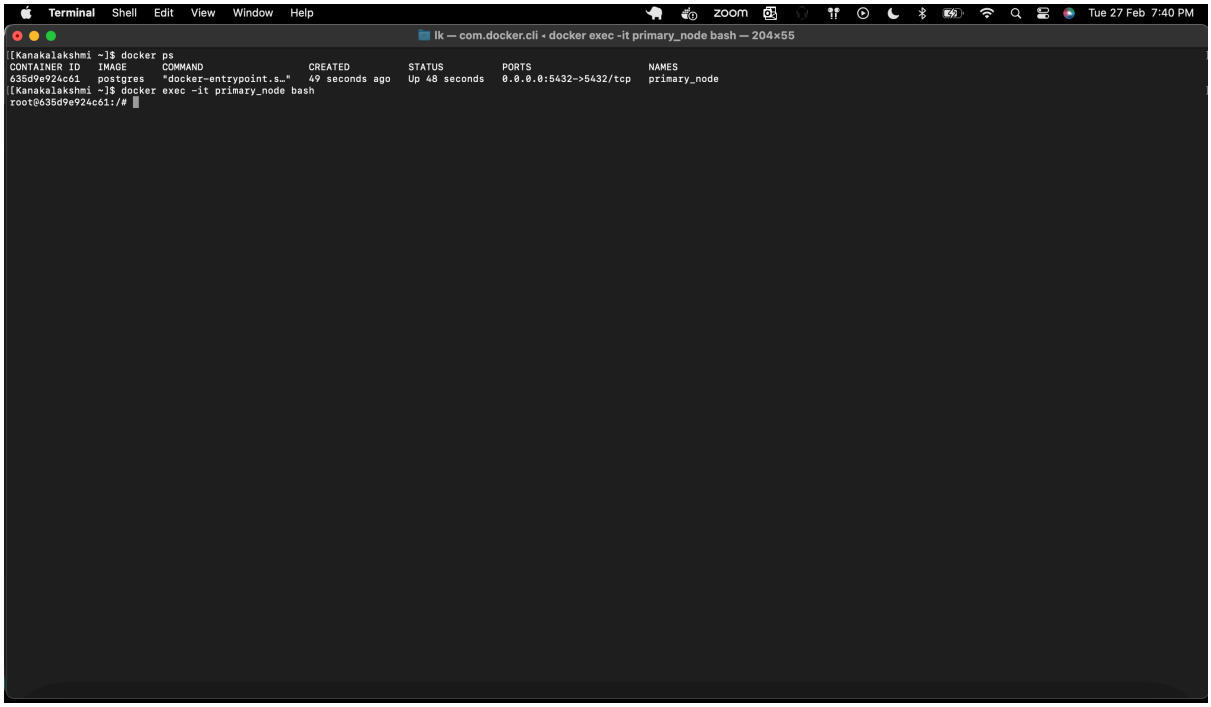

Fig-3: verify docker instances and run the docker container.

Explanation: once the postgres container is created we can see the status of it using the ‘docker ps’ command. We need to run the docker container using the container name.

Command:

```
ls
mkdir backup
pg_basebackup  -D /backup/ -R -v -U postgres
```

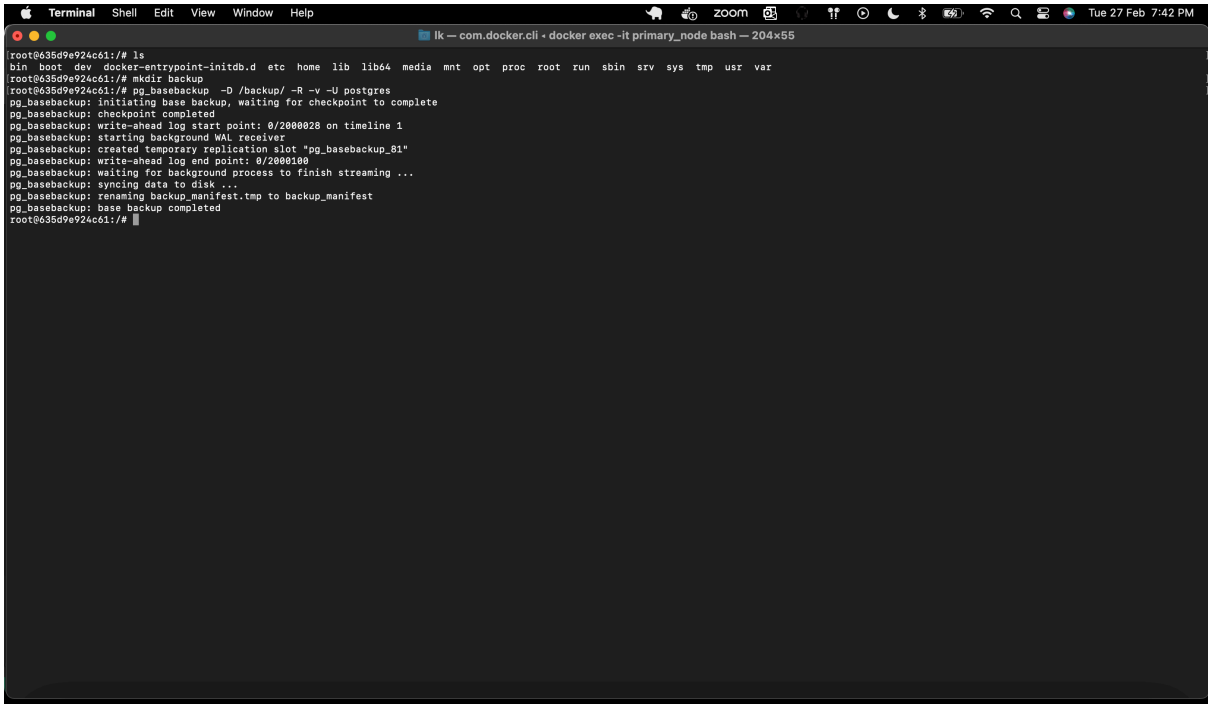

Fig-4: Display the file list and make a backup of the files for the replica node.

Explanation: In the docker container, we can see the files related to the postgres configurations. We are taking the backup of the respective files using pg_basebackup command.

Command:

```
cp –r ./backup/ /var/lib/postgresql/data
```

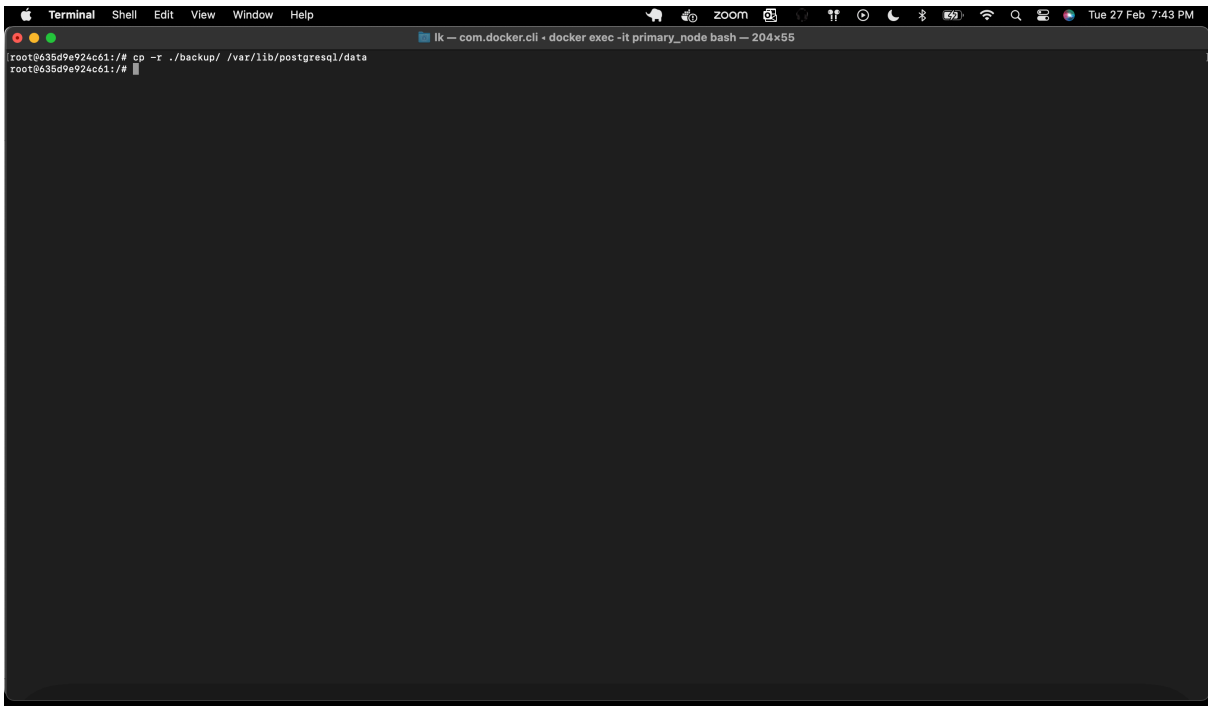

Fig-5: Copy recursively into ‘var/lib/postgresql/data’ folder.

Explanation: Copying the data recursively into the postgres/data folder.

Command:

```
psql –U postgres
\l –– To fetch list of files
```

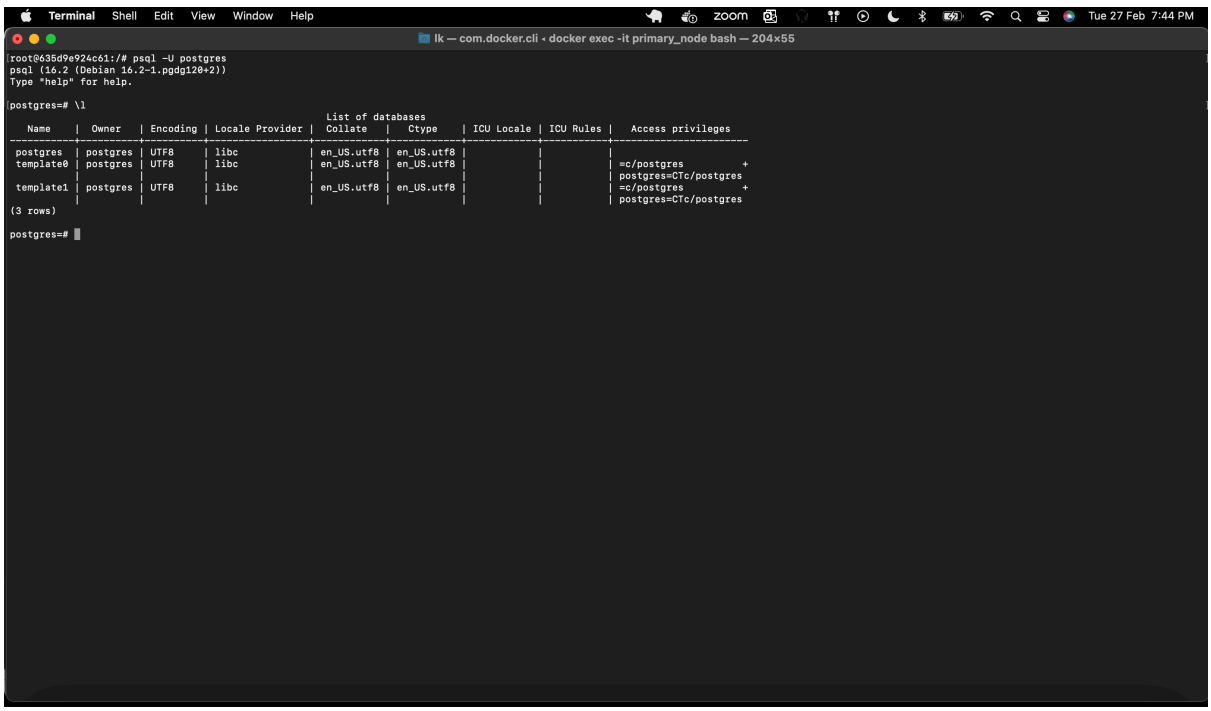

Fig-6: Connect to the postgres from docker container and display list of databases

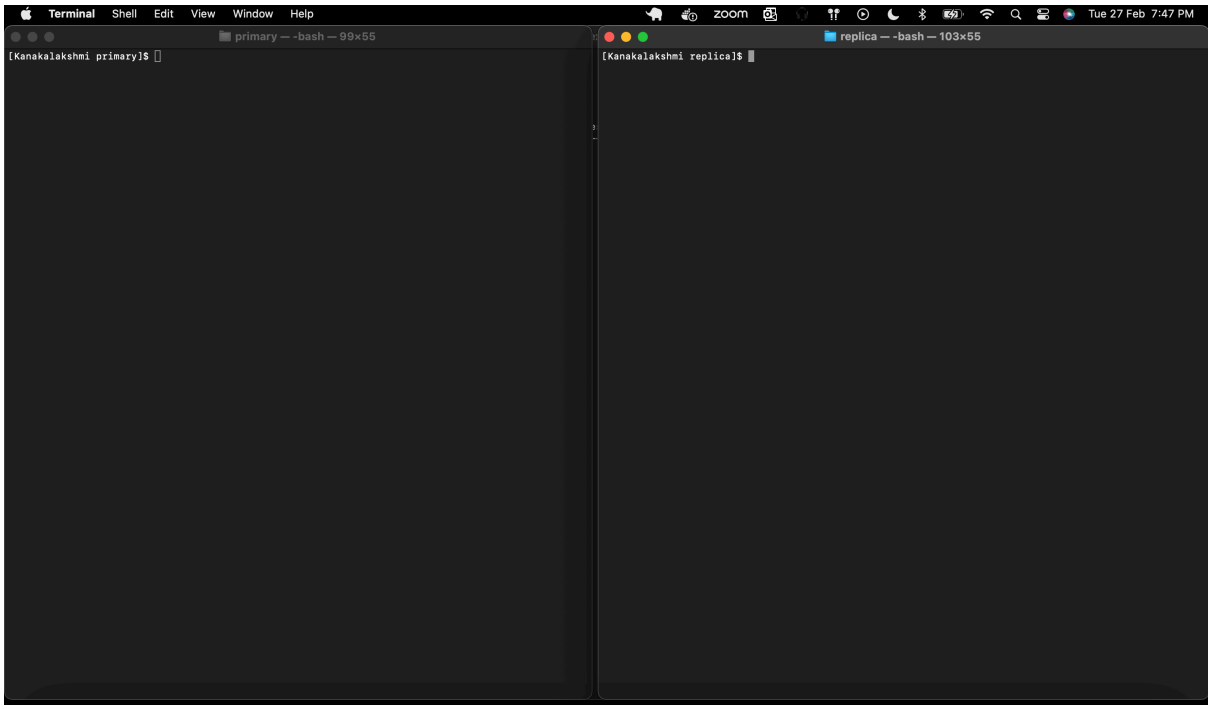

Fig-7: Open the two terminals from system, where primary and replica folders are created.

Explanation: Now open the two terminals which are created in the local system, as we initially created while pulling the postgres image. These two folders are created in data_base_volume_mounts.

Command:

```
\l -- To fetch list of files
```

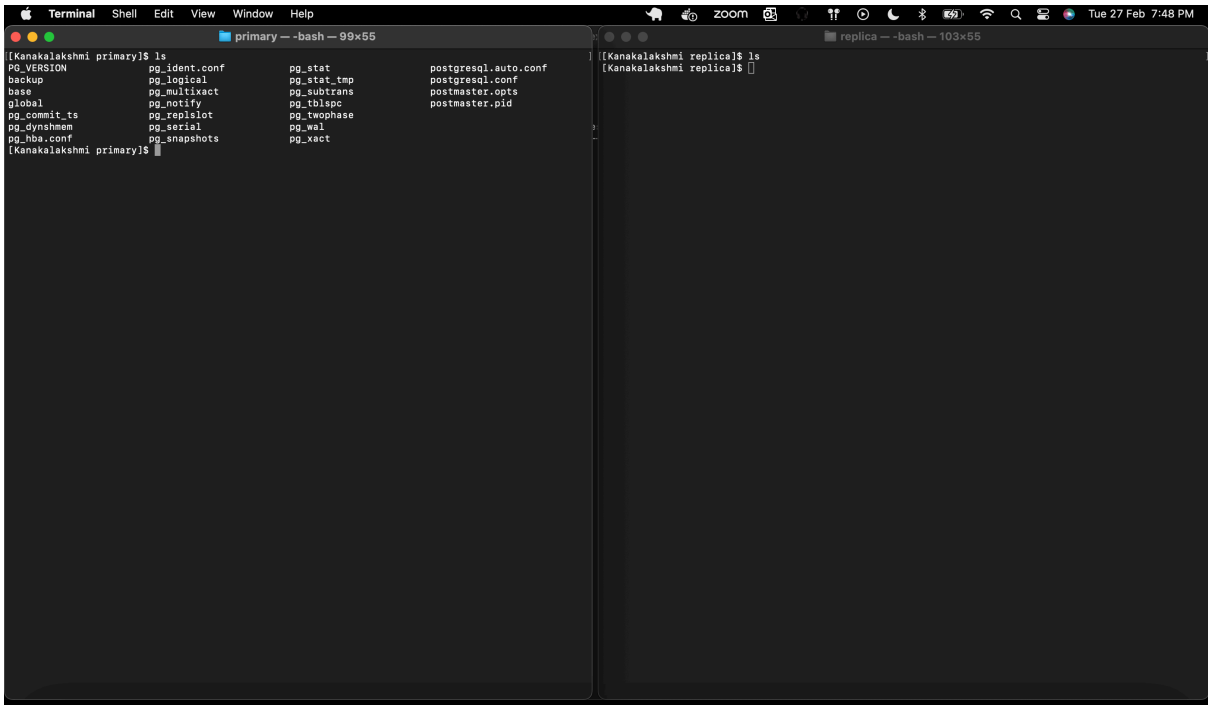

Fig-8: Display the list of files in primary folder.

Command:

```
cp –r ./backup/ ../replica/
ls
```

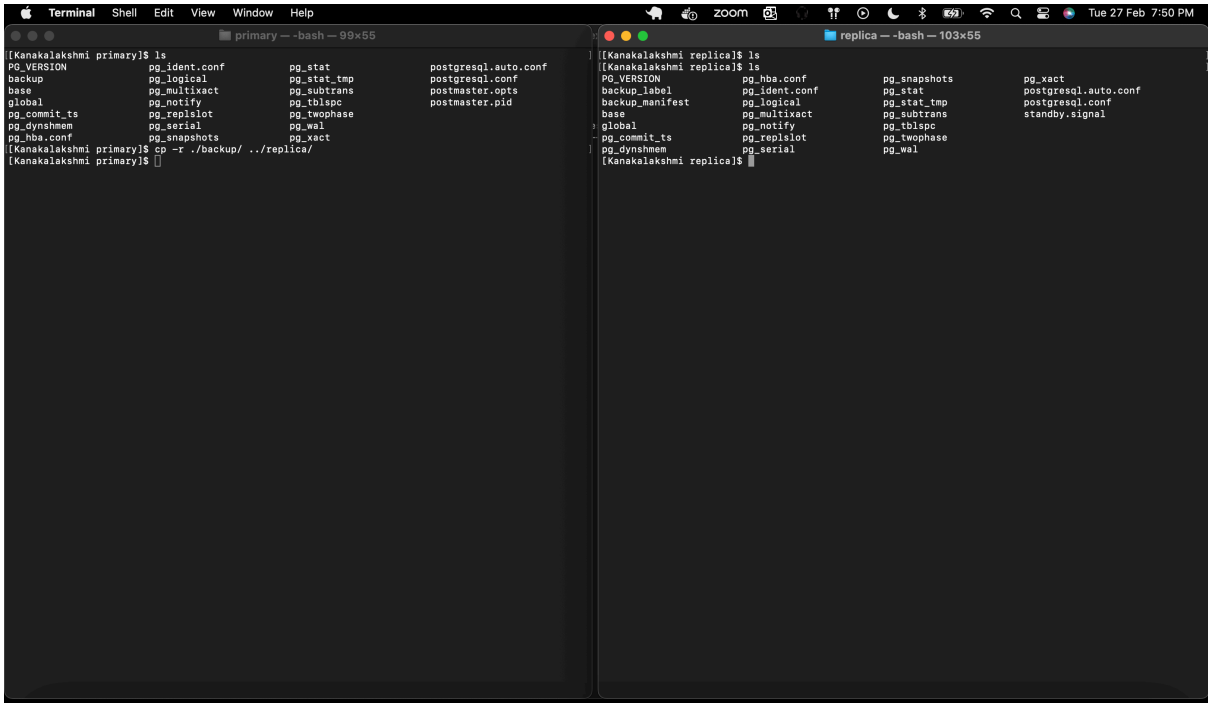

Fig-9: Copy to replica folder from primary folder and display the list of files in replica folder.

Explanation: As we have the files in the primary_node, copying recursively to the replica_node to have the same configurations. In the replica_node the standby.signal file plays the vital role in communication between the primary_node and replica_node.

Command:

```
vi postgresql.conf – wal_log_hints = on, archive_mode = on
archive_command = 'test ! –f /var/lib/postgresql/data/archiver/%f &&
/var/lib/postgresql/data/archiver/%f', wal_keep_size = 512
```

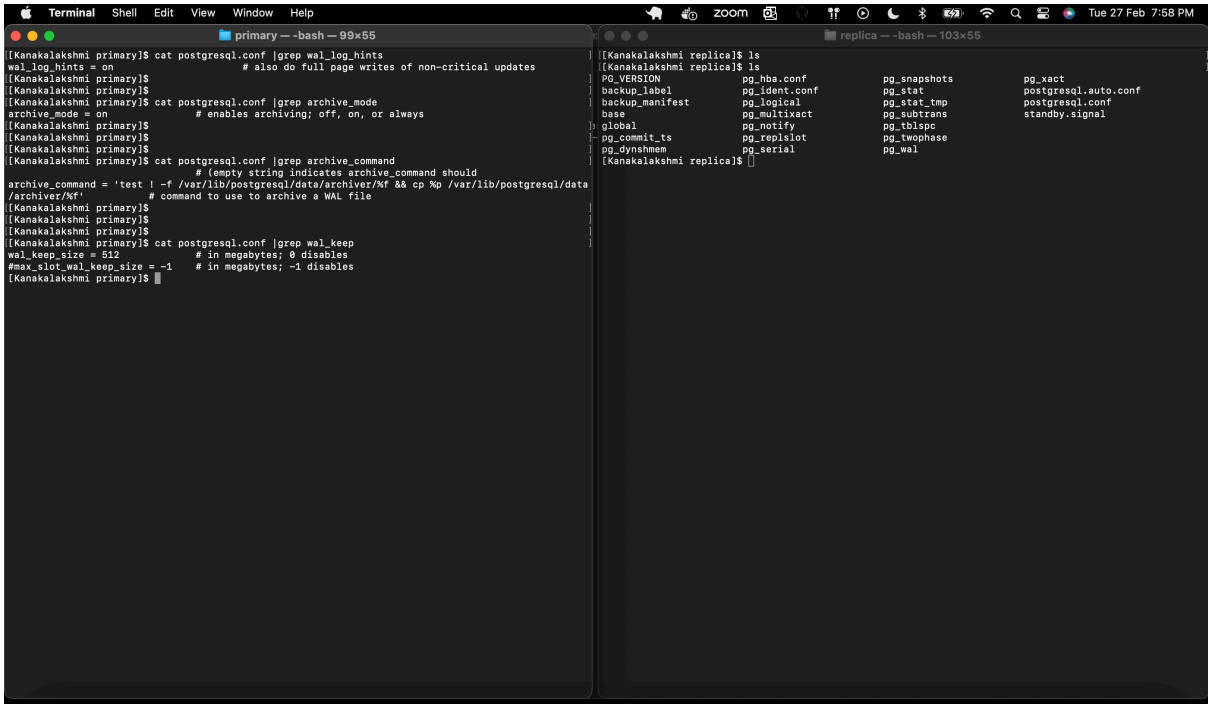

Fig-10: update the configurations in postgresql.conf file in primary folder.

Command:

```
vi pg_hba.conf – host replication replication_user   trust
```

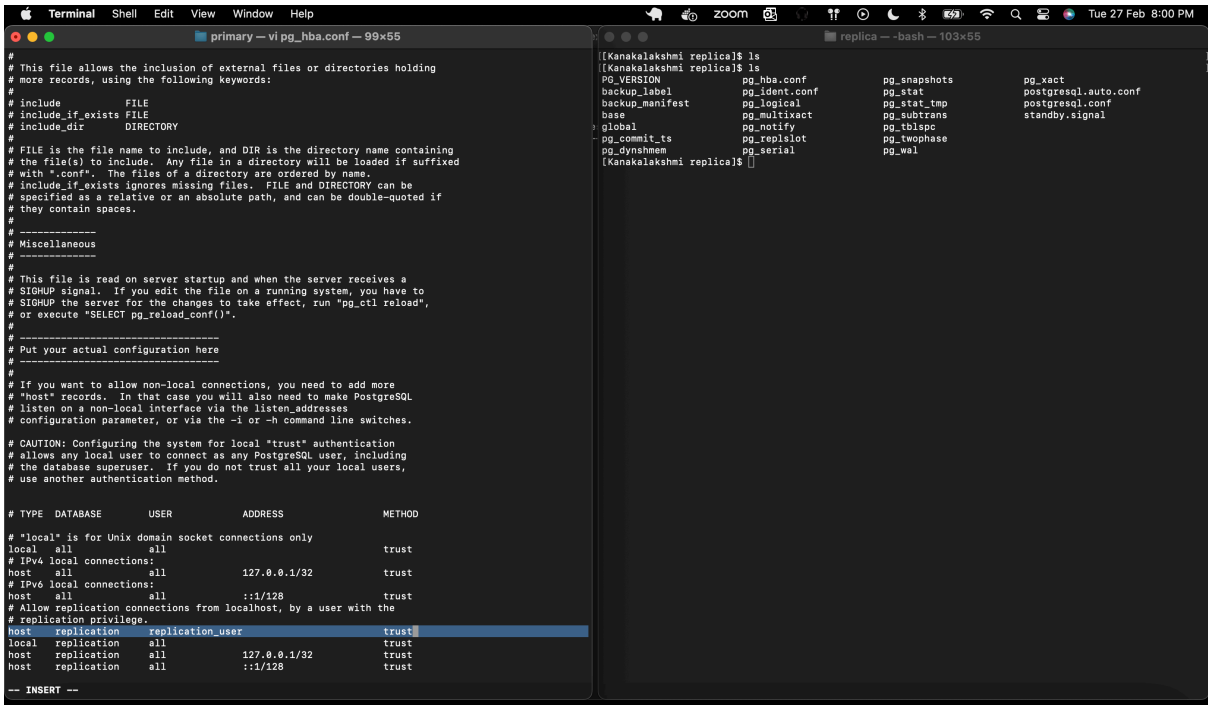

Fig-11: update the configuration in pg_hba.conf file in primary folder.

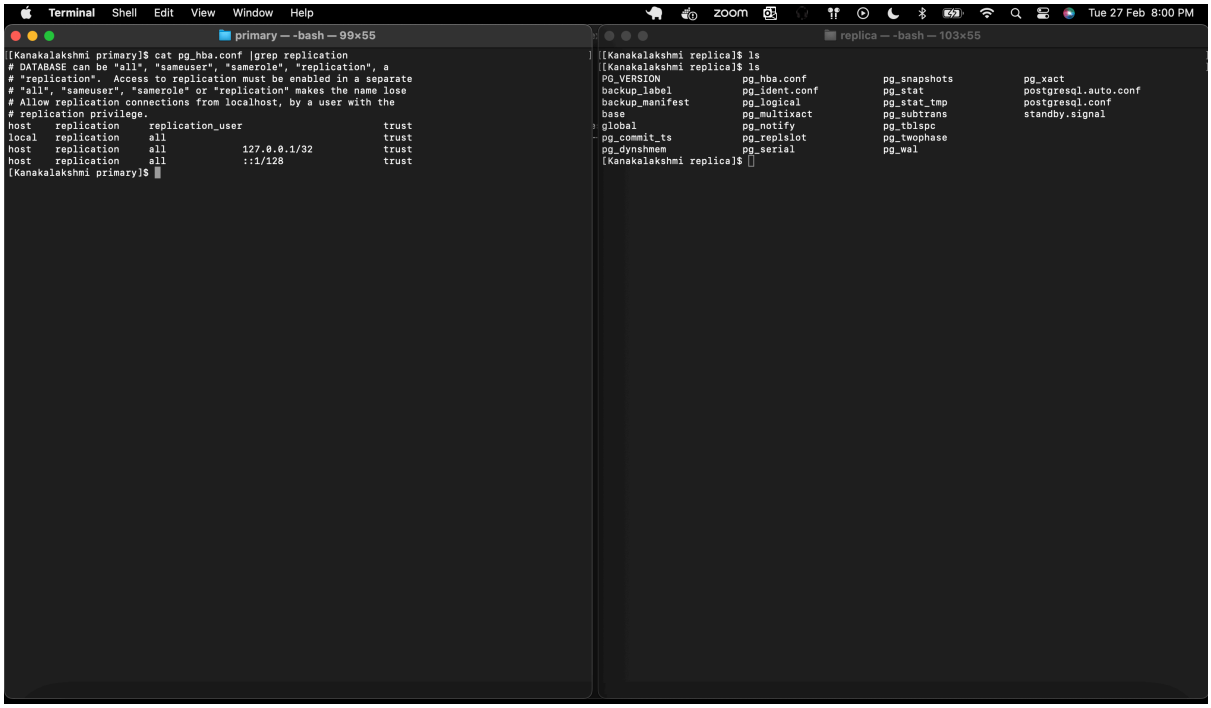

Fig-12: verify the updated configurations in primary folder.

Command:

```
docker inspect primary -- get the IP address of postgres docker container from terminal
```

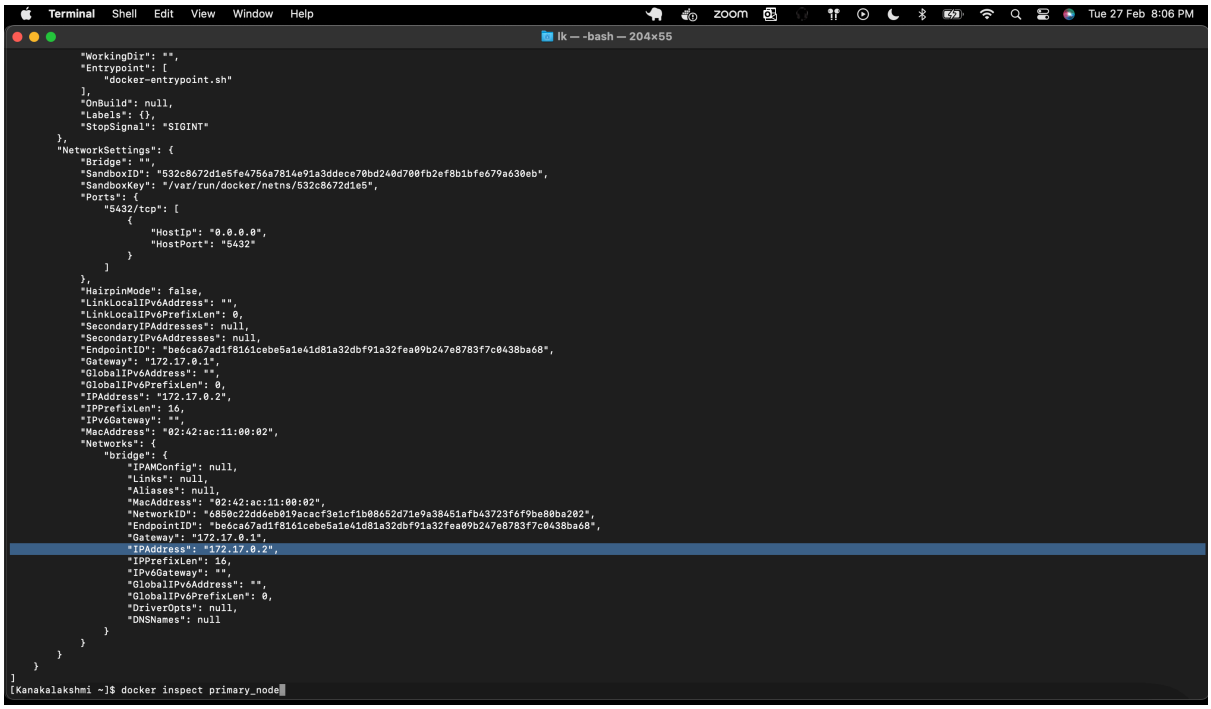

Fig-13: Fetch the IP of primary_node docker container from terminal.

Explanation: Fetch the IP address of the primary_node using above to update in the replica_node.

Command:

```
vi postgresql.conf – primary_conninfo = 'host=172.17.0.2 port=5432
user=replication_user'
```

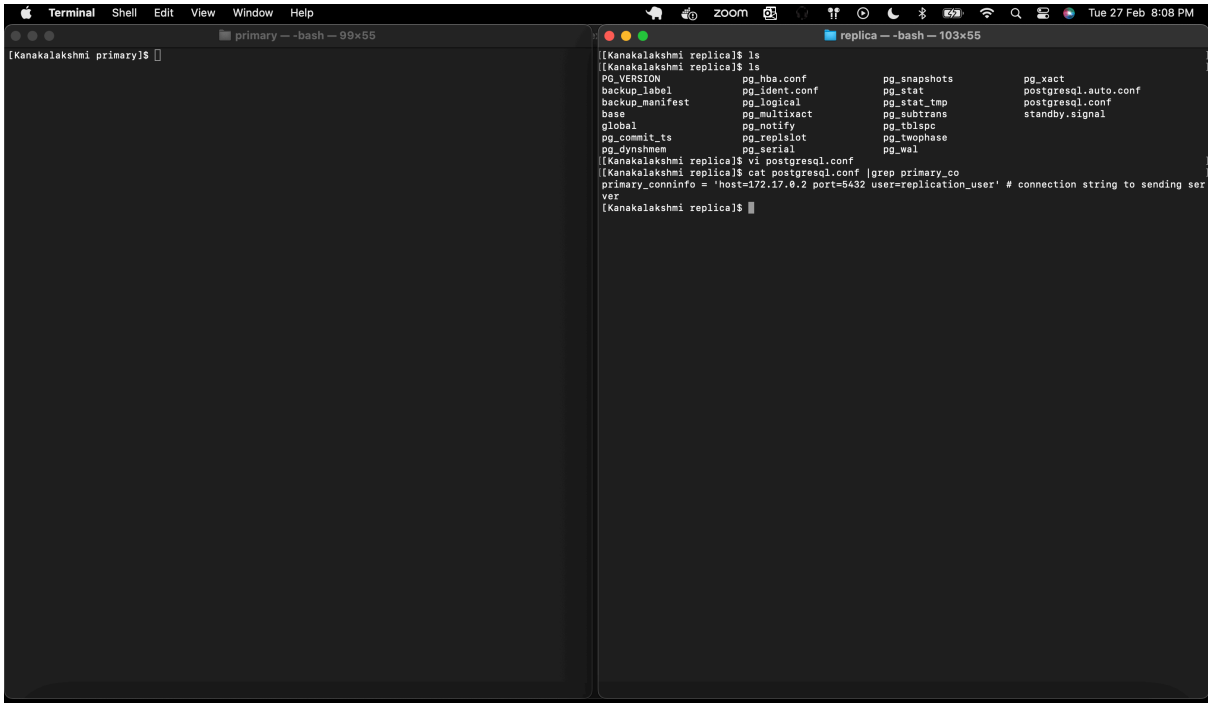

Fig-14: update the primary_conninfo in postgresql.conf file at replica folder.

Command:

```
vi postgresql.auto.conf
primary_conninfo = ‘user=replication_user port=5432 host=172.17.0.2’
```

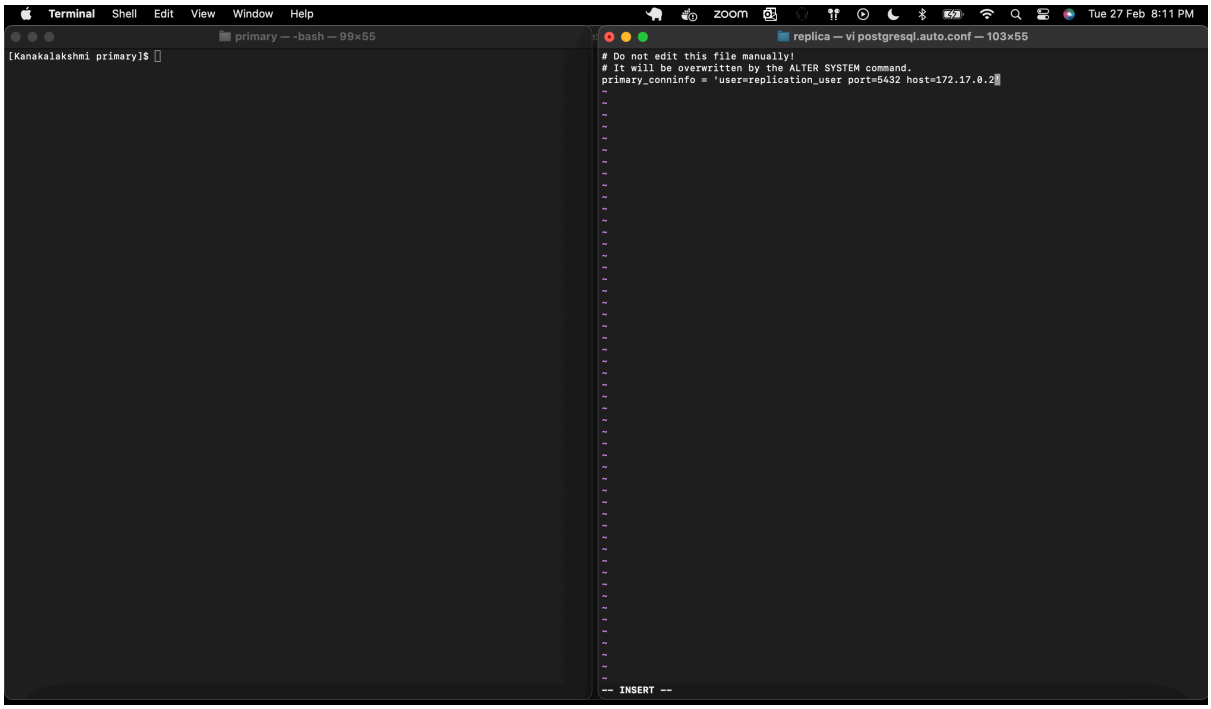

Fig-15: Update configurations in postgresql.auto.conf file at replica folder.

Command:

```
Cat postgresql.auto.conf
```

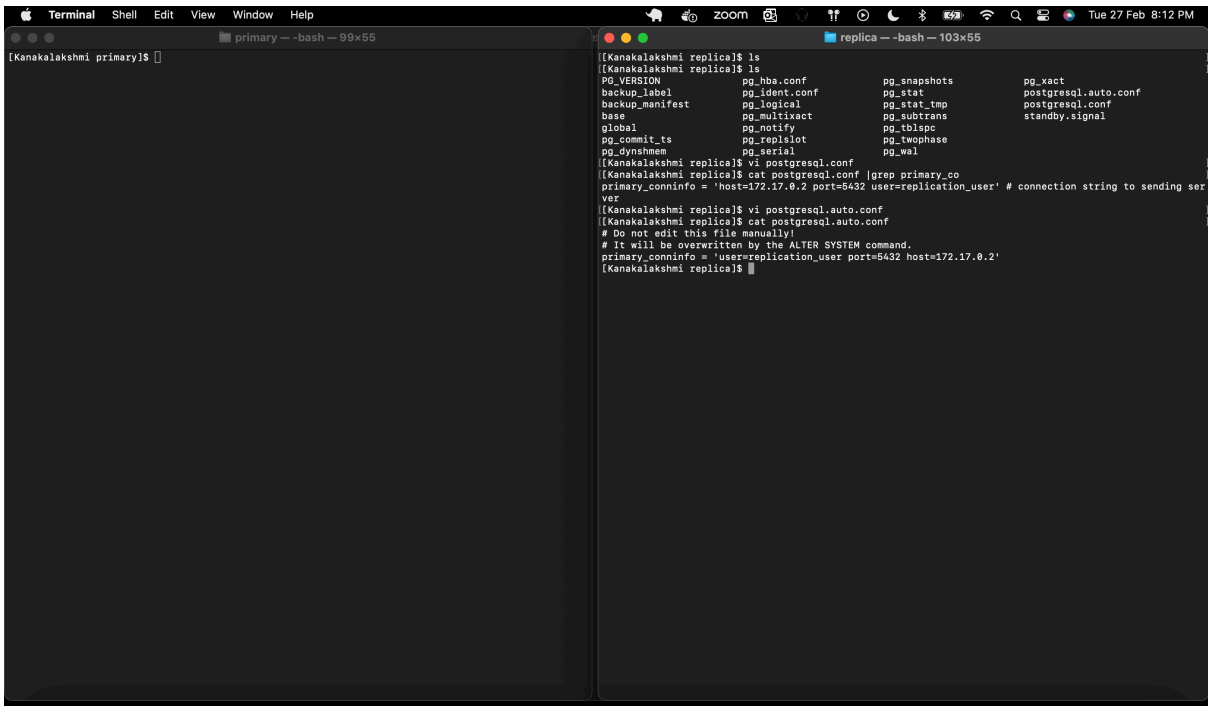

Fig-16: verify postgresql.auto.conf configurations.

Command:

```
create user replication_user replication;
\du
```

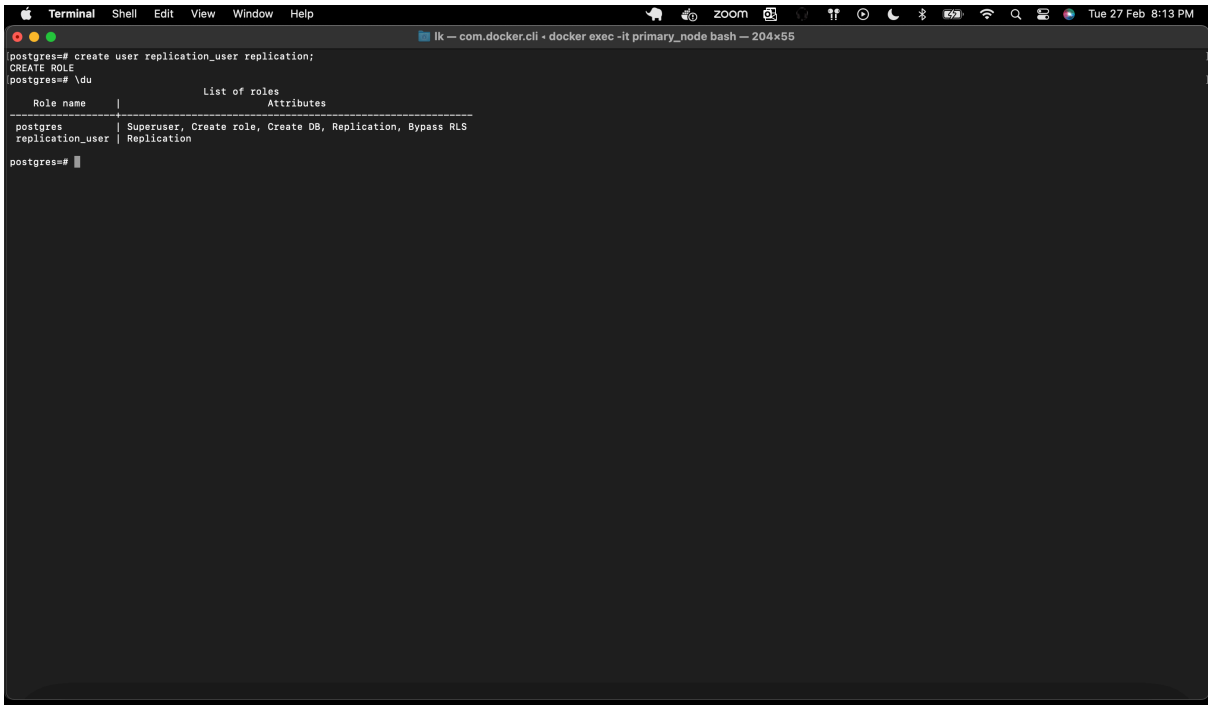

Fig-17: create replication_user in primary_node

Command:

```
docker logs --tail 200 -f primary_node
```

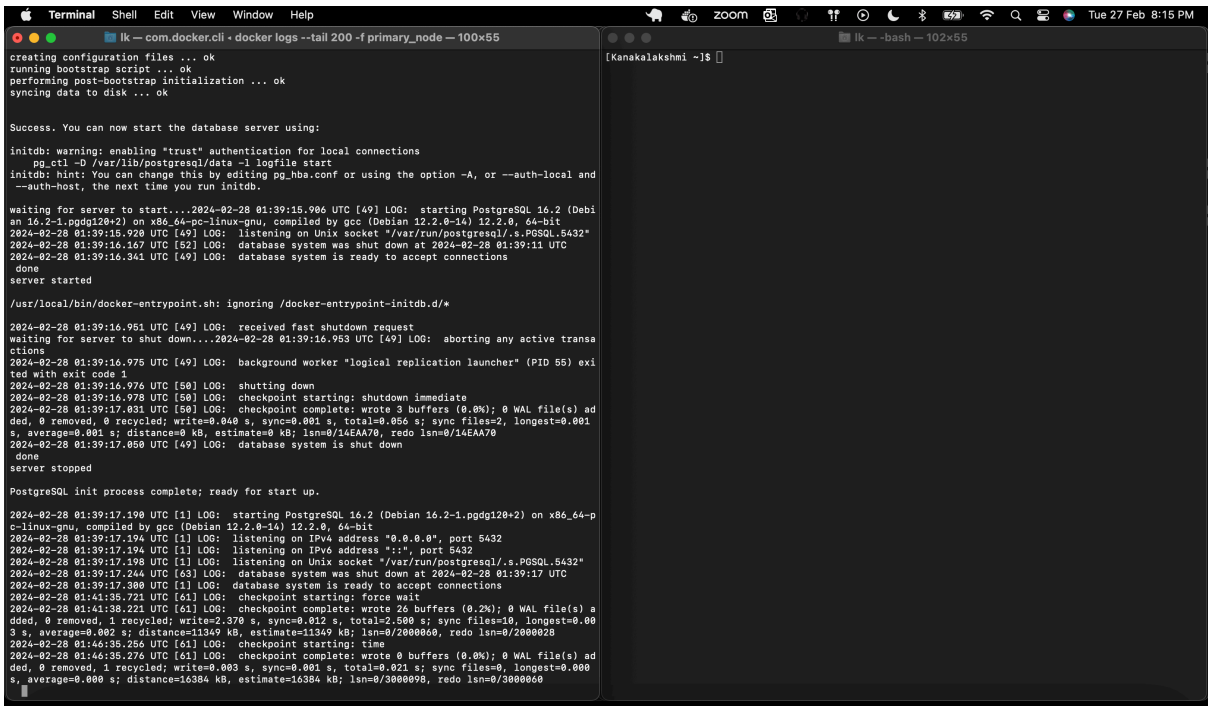

Fig-18: verify logs in primary_node.

Command:

```
mkdir archiver
ls
```

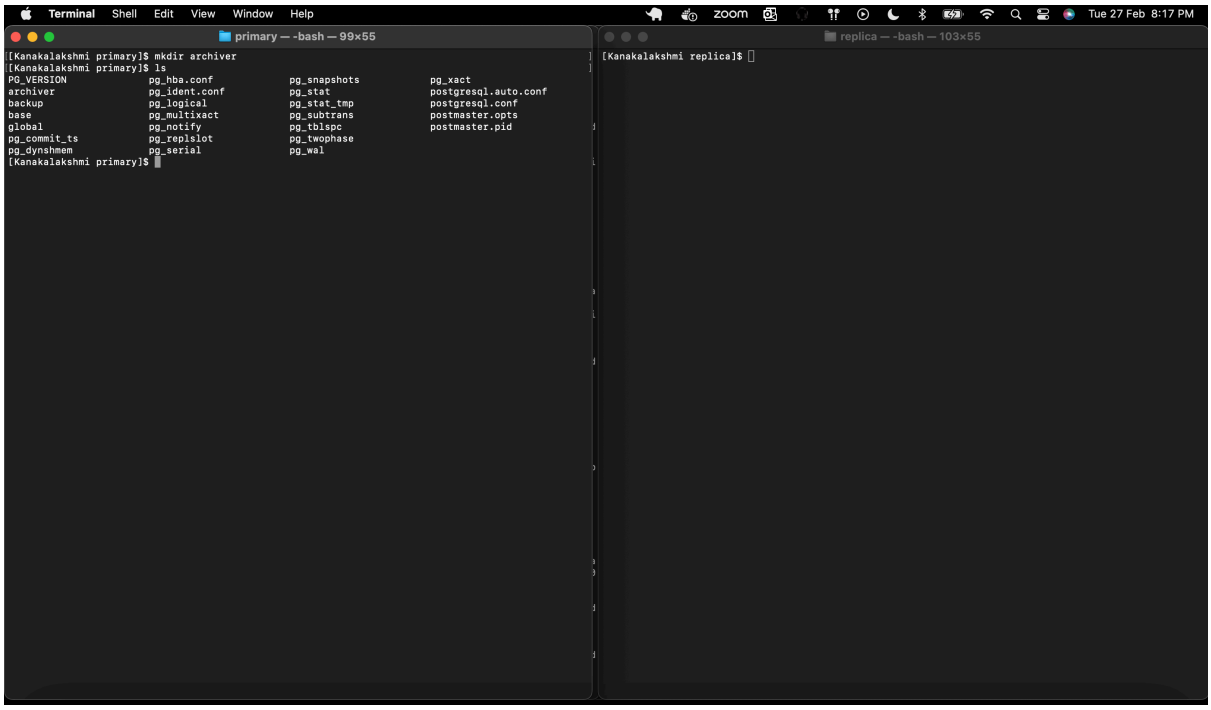

Fig-19: create archiver folder in system primary folder.

Command:

```
docker run --name replica_node -p 5433:5432 -e POSTGRES_PASSWORD=password -v
/Users/lk/Documents/lakshmi_DPDB/HW-
2/data_base_volume_mounts/replica:/var/lib/postgresql/data -d postgres
```

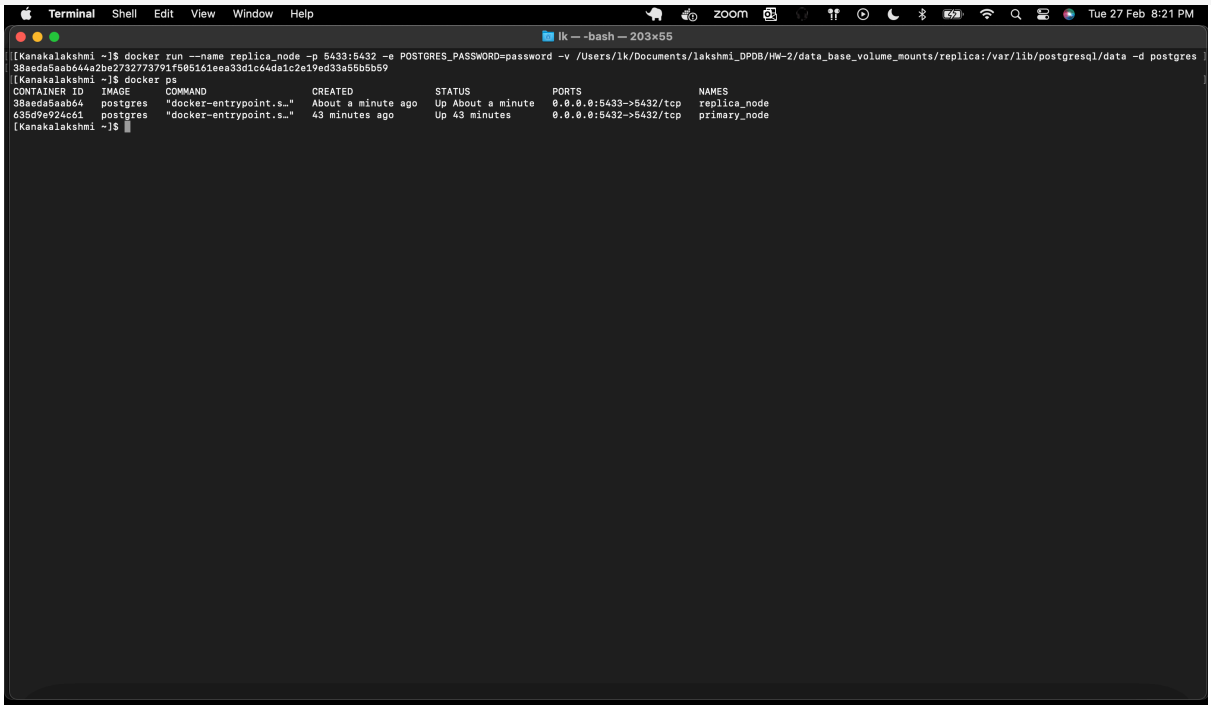

Fig-20: Pull the postgres docker image, used as postgres replica_node.

Explanation: Pull the postgres docker image using the configurations in the replica folder provide the details to create docker container such as name, port_no and password. Once image is pulled, we can see the status of the replica_node.

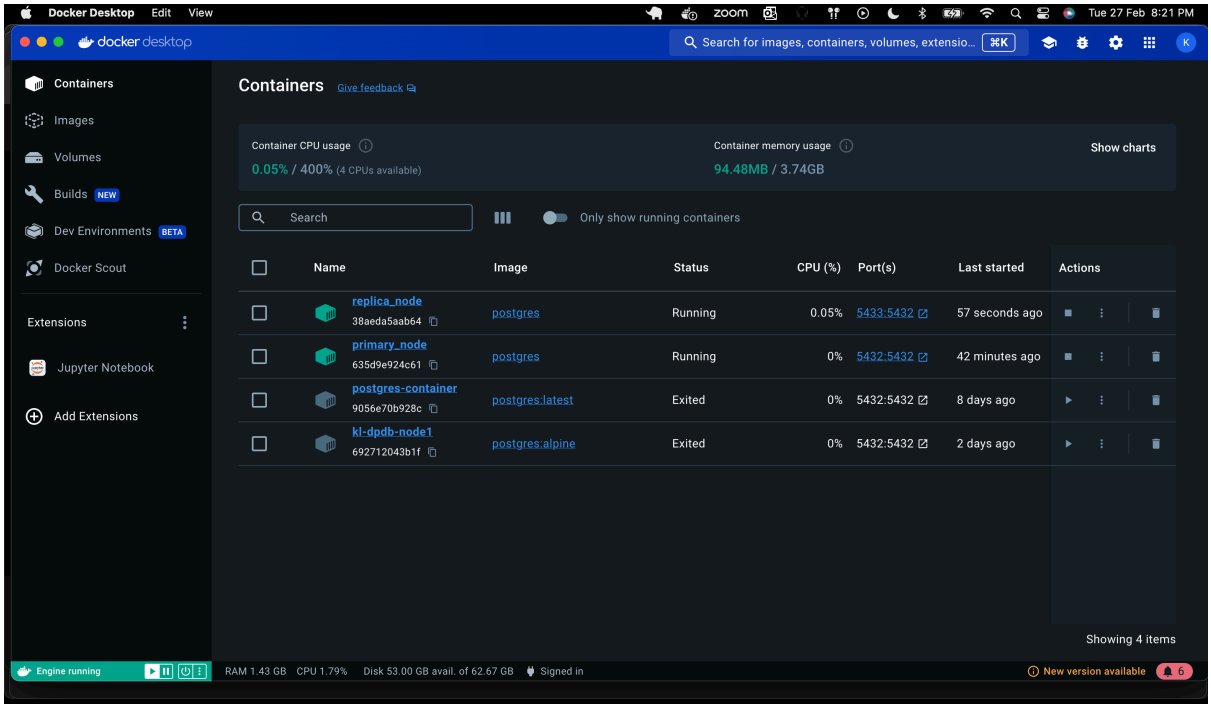

Fig-21: verify two postgres instances in docker dashboard.

Command:

```
docker logs --tail 200 -f replica_node
```

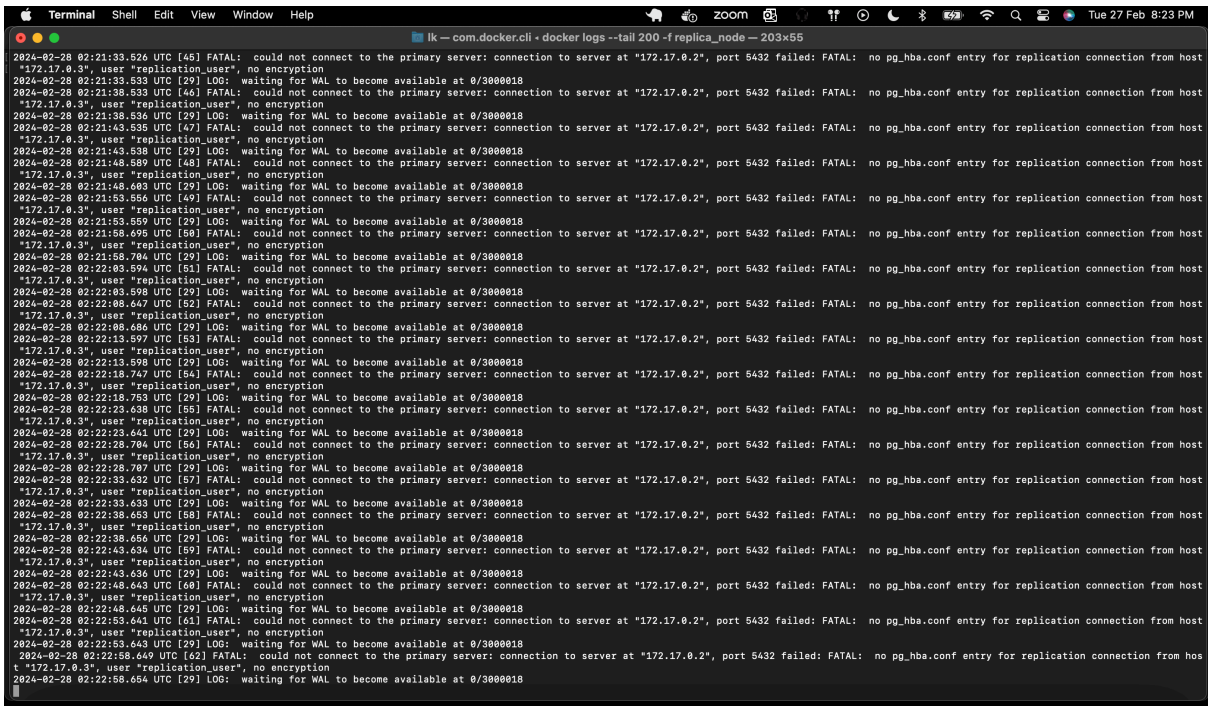

Fig-22: check the logs for replica node.

Command:

```
docker inspect replica_node
```

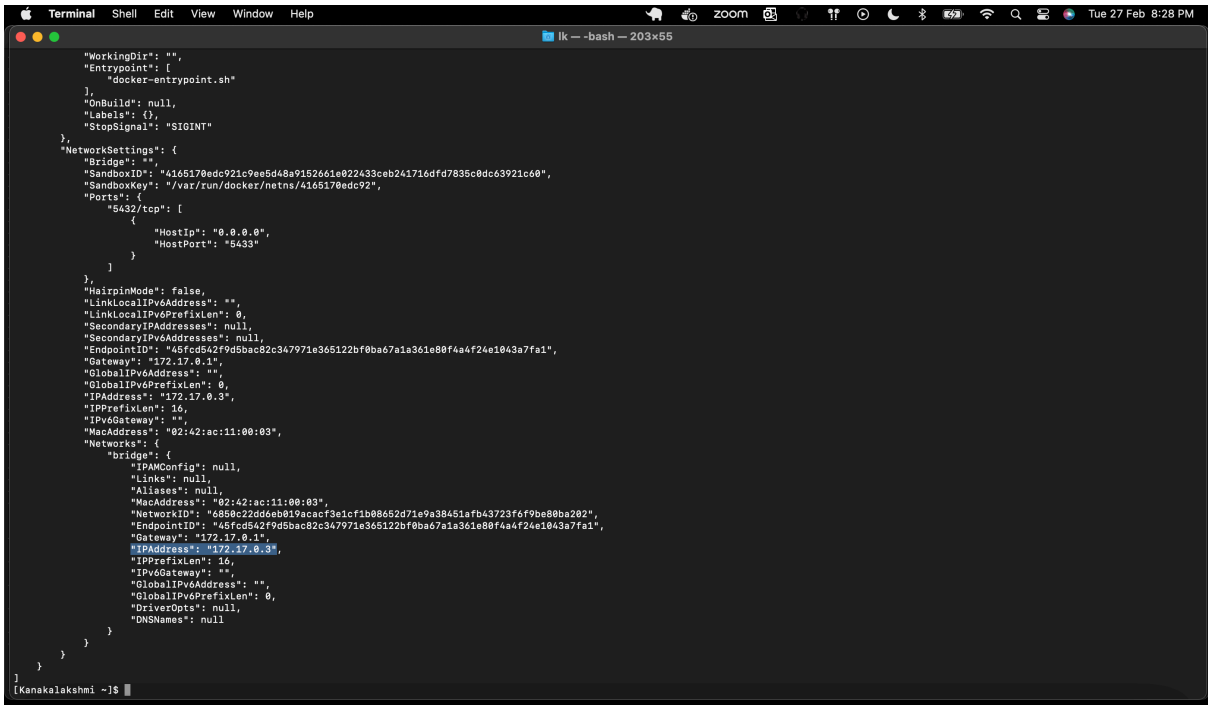

Fig-23: Get the IP of the replica node.

Explanation: Get the IP address of the replica_node to update respective details in the primary_node.

Command:

```
vi pg_hba.conf
host replication replication_user  172.17.0.3/32   trust
cat pg_hba.conf
```

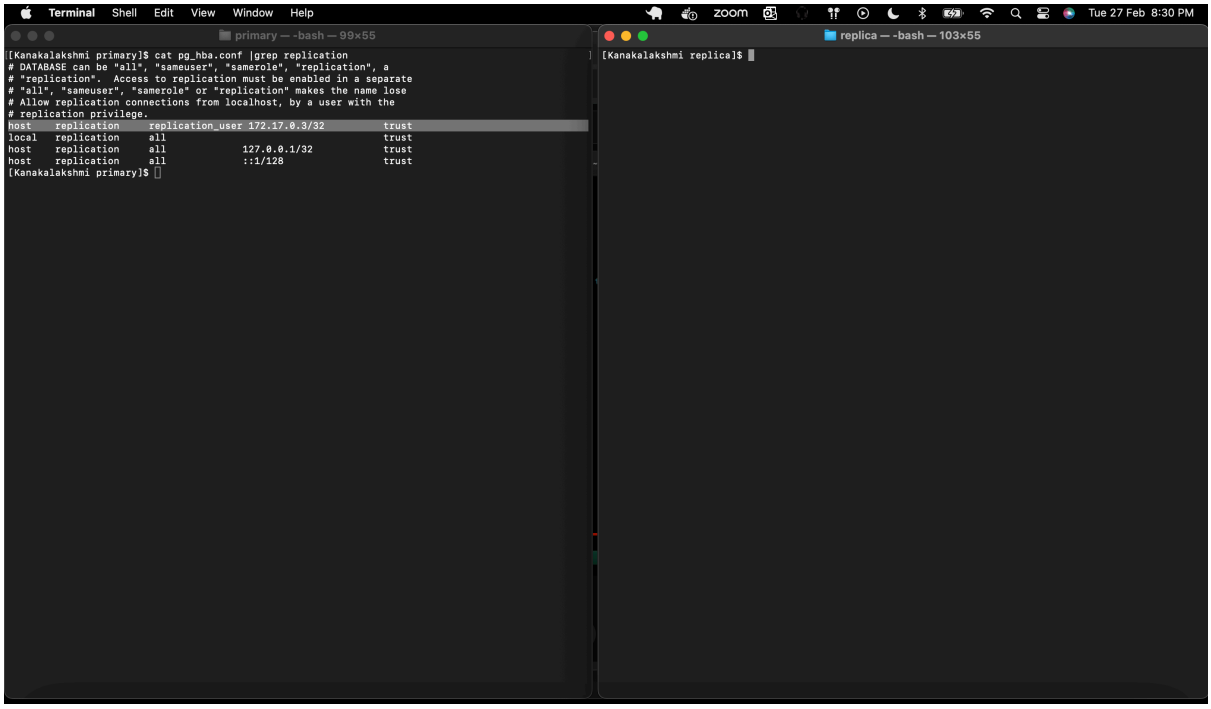

Fig-24: update replica_node IP in primary_node pg_hba.conf file.

Command:

```
docker restart primary_node
docker logs --tail 200 -f primary_node
```

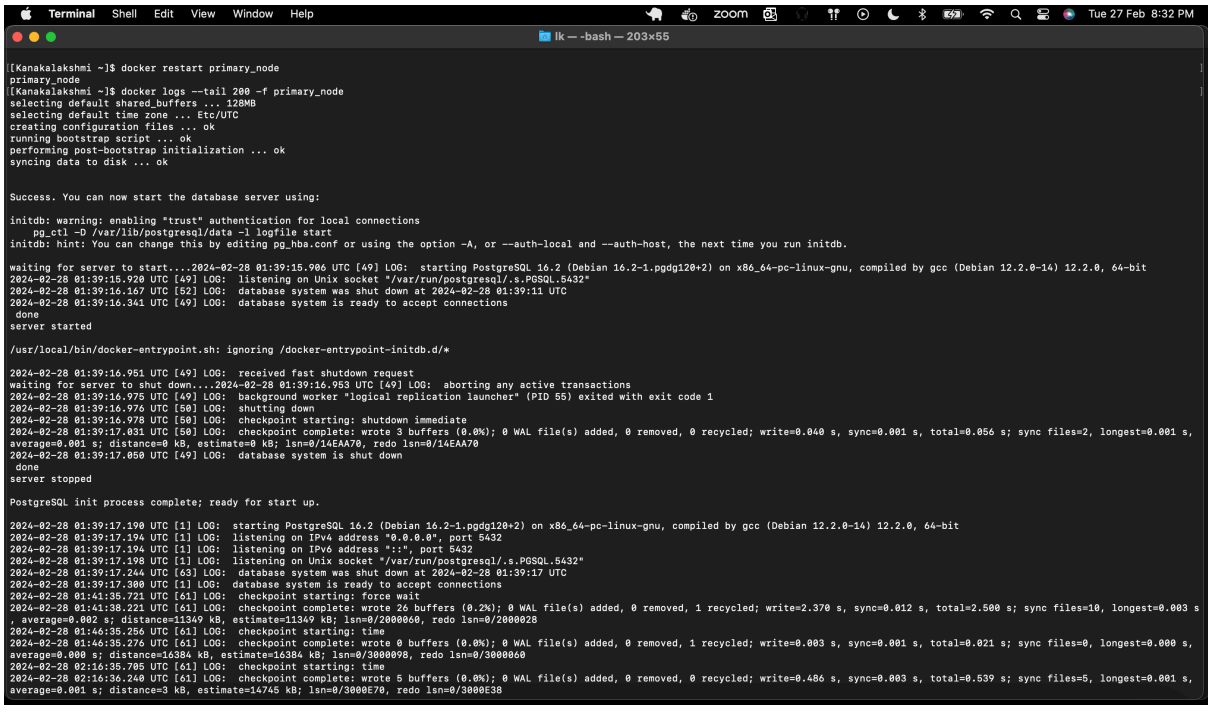

Fig-25: Restart the primary_node and verify the logs.

Now two postgres docker containers are ready with all configurations, we can start using them.

Command:

```
docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}'
primary_node
docker exec -it primary_node bash

docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}'
replica_node
docker exec -it replica_node bash
```

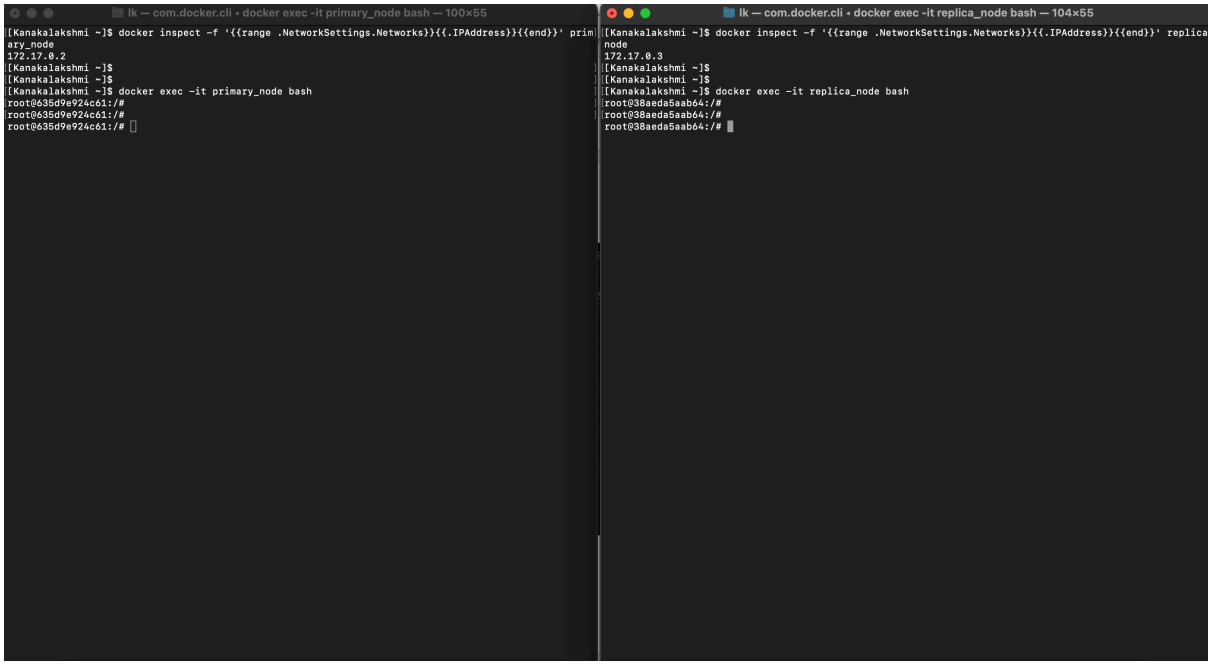

Fig-26: check the IP and execute primary and replica nodes.

Explanation: These are the two postgres docker containers. One is the **primary_node with 172.17.0.2** and the other is **replica_node with 172.17.0.3**. Exciting the run command on both the nodes.

Now we are connected to two different postgres instances.