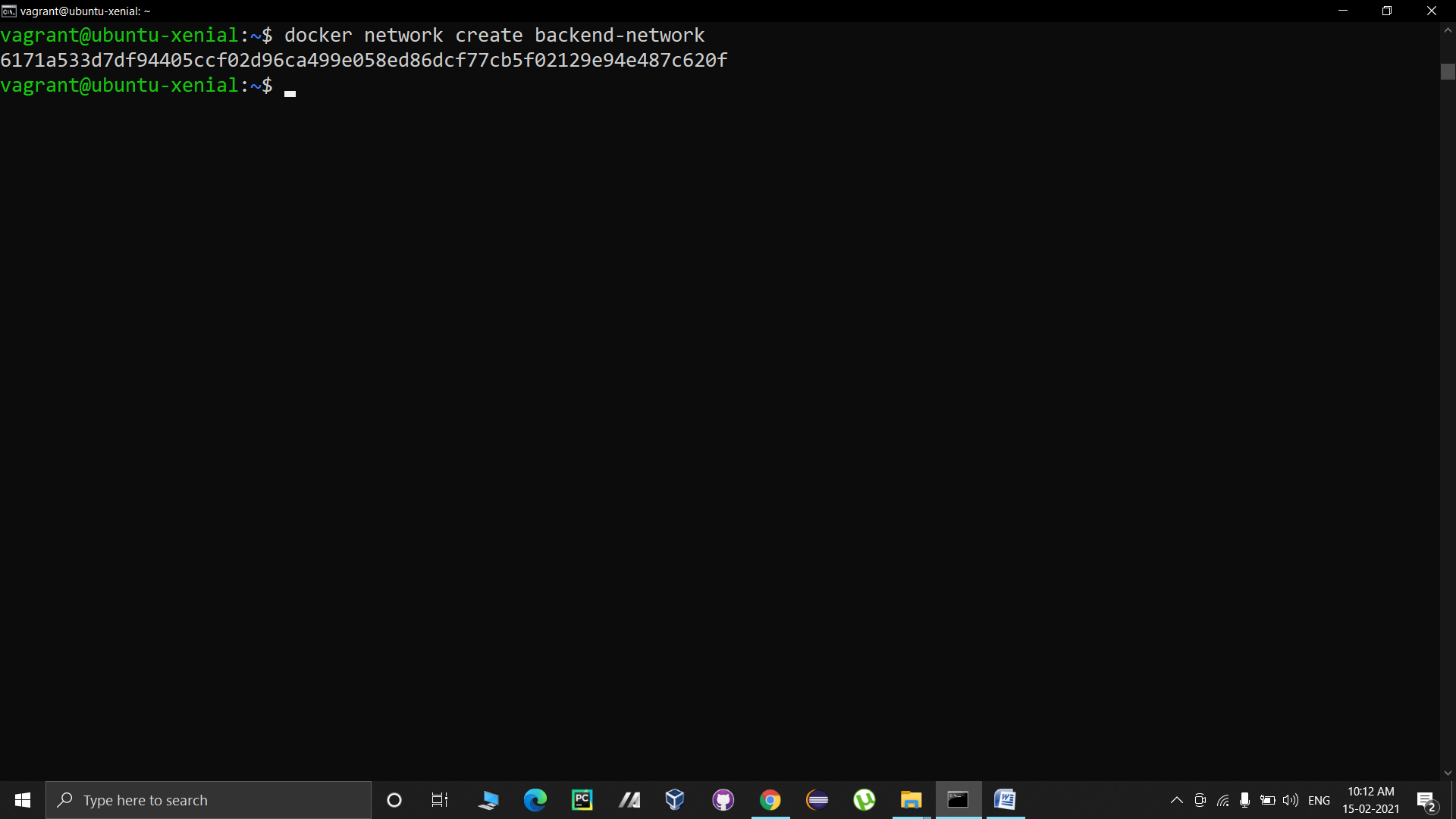
**Lab Experiment-3**

**Networks in Docker.**

The steps that needs to be followed are:

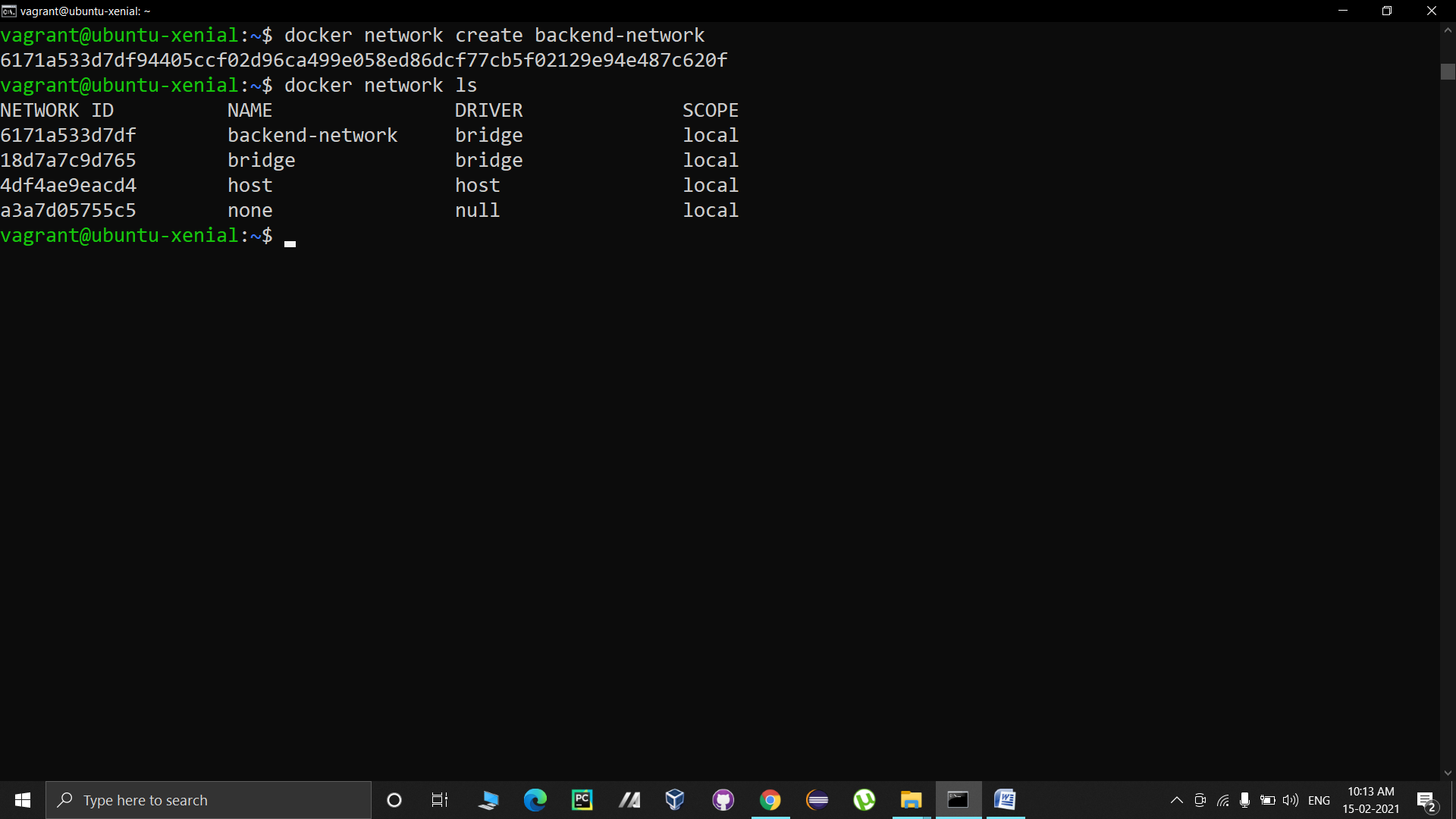
1. Create a network

Command: docker network create backend-network



2. You can verify the creation of the network by listing the available networks.

Command: docker network ls



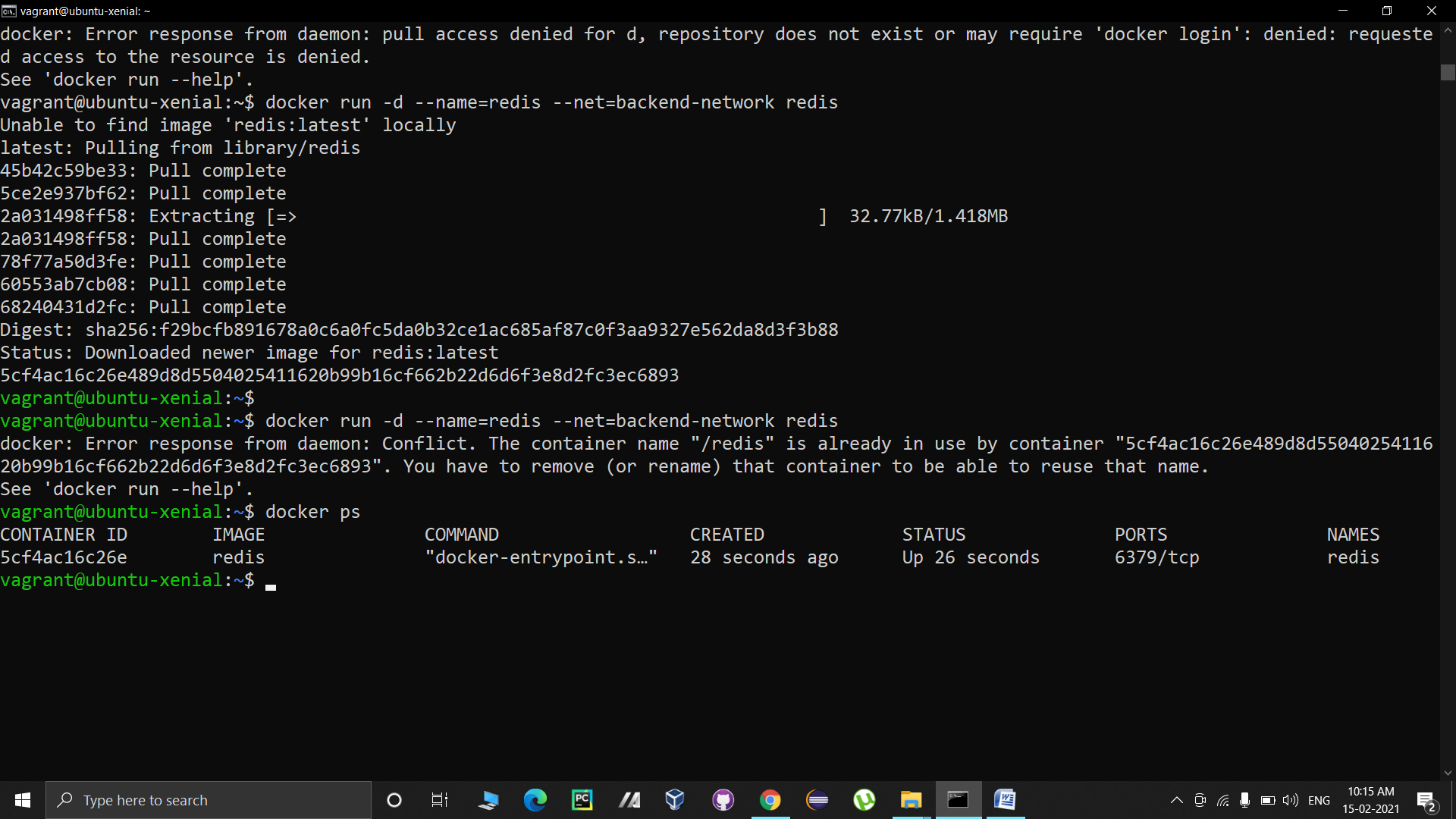
The above displayed output verifies the creation of our network.

3. Now we will create a new container and we will assign backend-network to this container.

Command syntax: docker run --net=<network-name><container-image>

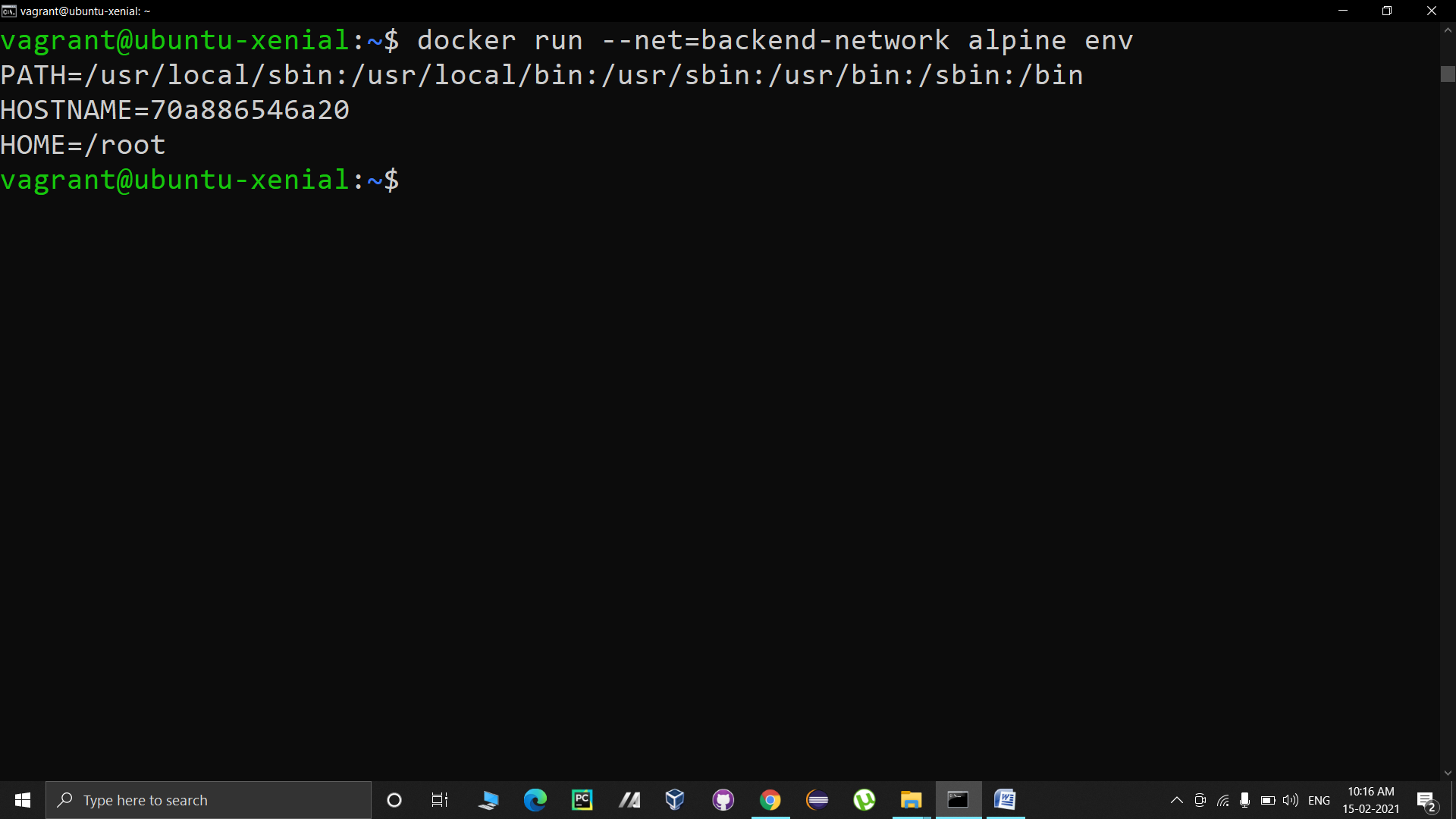
Command: docker run –d --name=redis --net=backend-network redis

We can see our newly created container is running using the command docker ps.



4. The networks in Docker do not use environment variables to discover other containers. We can verify this by creating a new container and assigning it the same network.

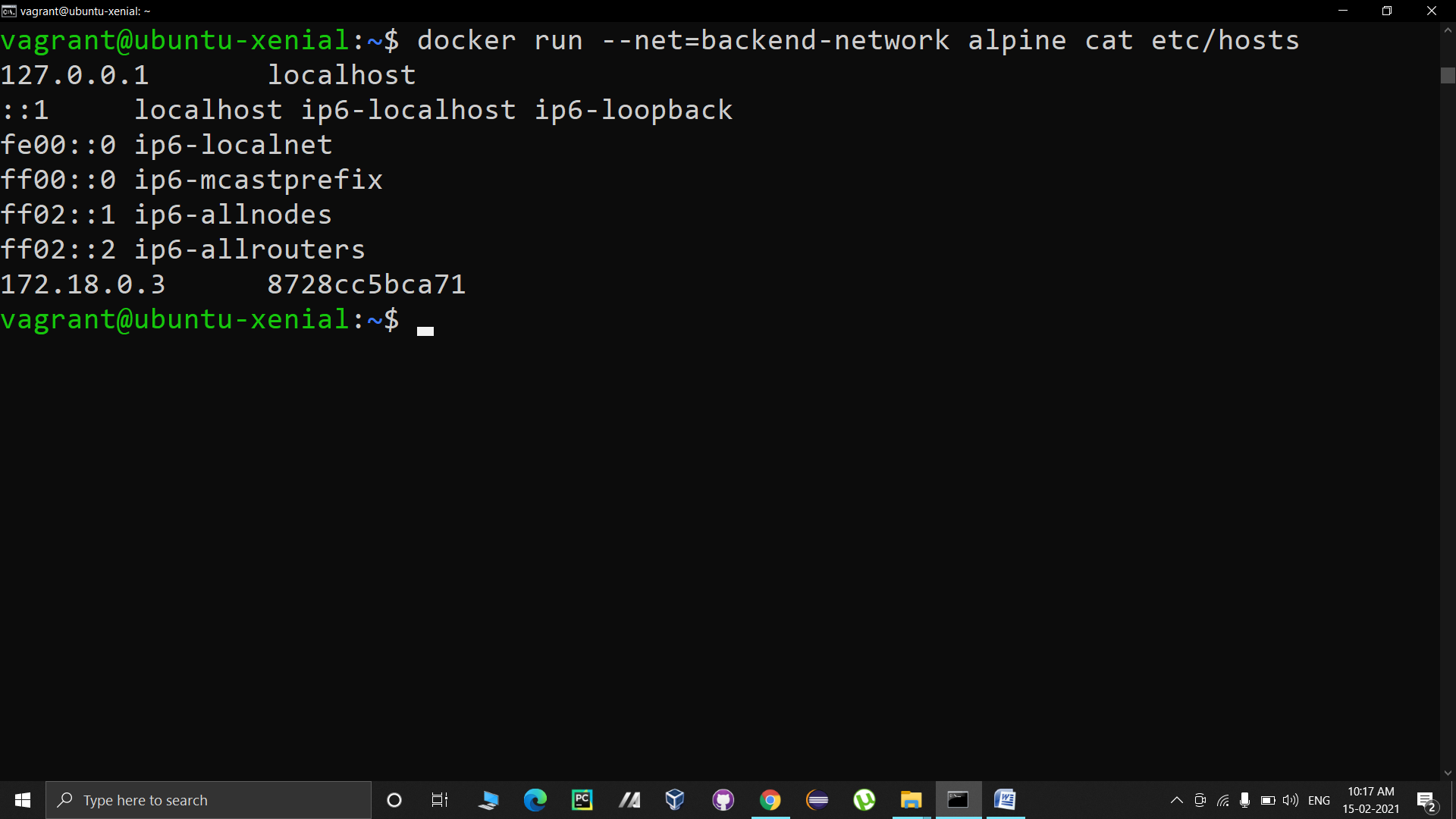
Command: docker run --net=backend-network alpine env



The above output verifies that there is no entry of the redis container which is also connected to the same network in the environment variables of this alpine container.

Also, Docker does not add any entries in /etc/host for the other containers. This can be verified as follows.

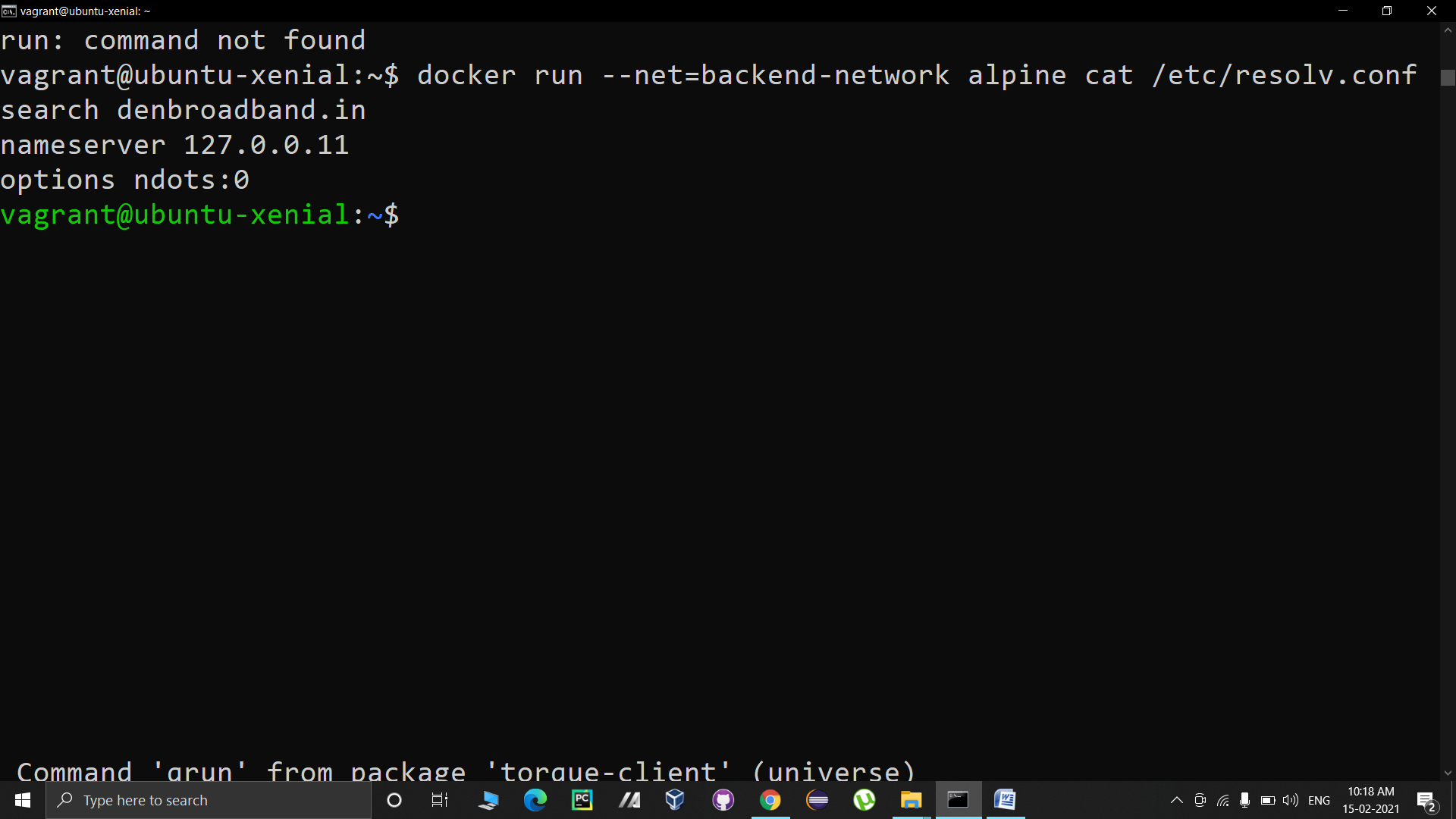
Command: docker run --net=backend-network alpine cat etc/hosts



5. Neither environment variables, nor the hosts file is updated. This is because the containers communicate via DNS-based entries.

The DNS server is assigned to all containers and it is set in resolv.conf file. This can be verified as follows:

Command: docker run --net=backend-network alpine cat /etc/resolv.conf

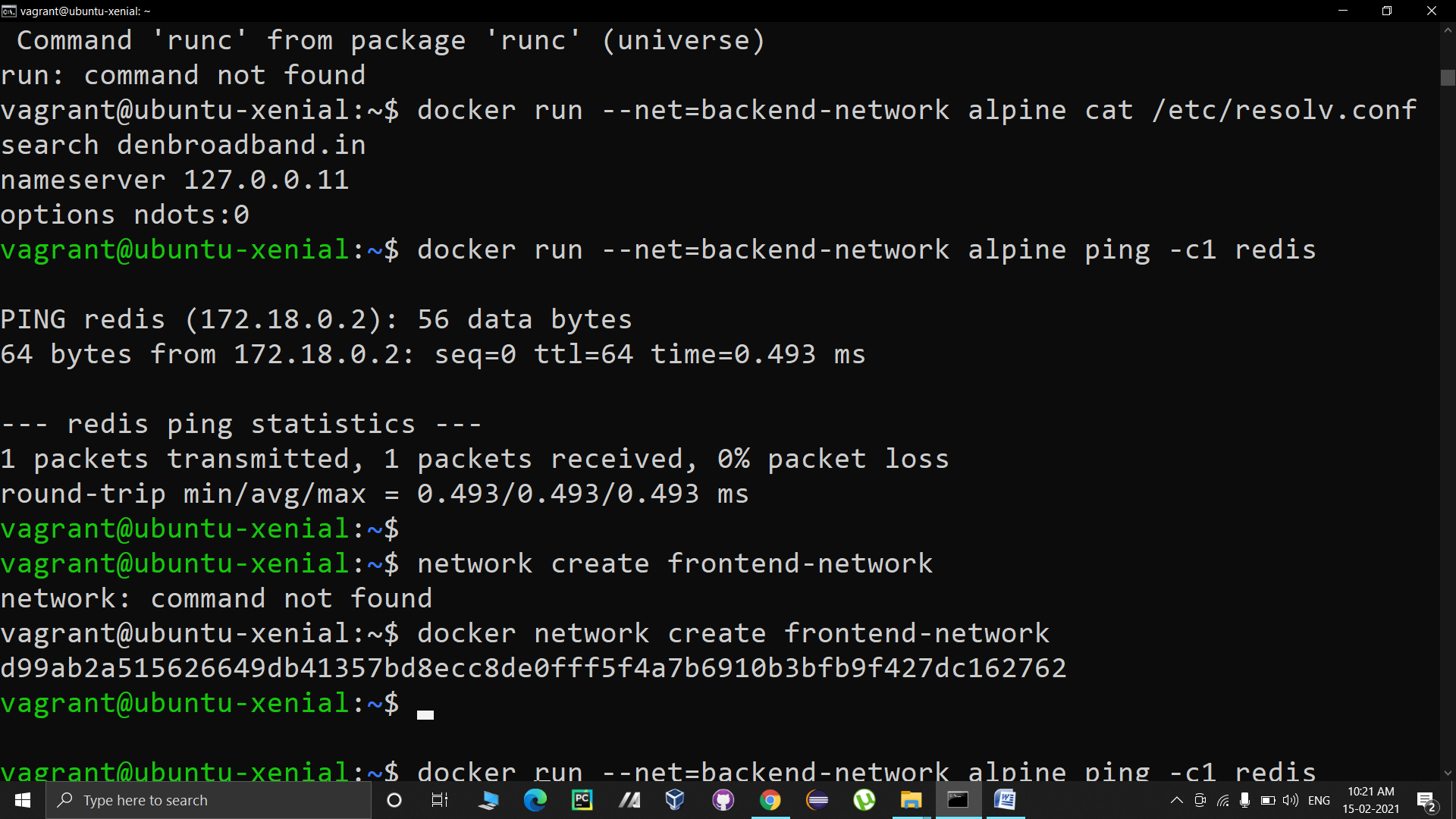


6. Now let us see how the containers connected to the same network can communicate with each other. When a container wants to access another container by its name, the DNS return the IP address of that container, and thus making the communication work. Let us try to ping the redis container which is attached to the same network.

Command: docker run --net=backend-network alpine ping -c1 redis

7. We can attach a container with multiple networks. For this, let us create a new network named frontend-network.

Command: docker network create frontend-network



We can verify the created network by using the following command.

Command: docker network ls

8. We can connect the already created containers to a network using the following command.

Command syntax: docker network connect <network-name><container-name>

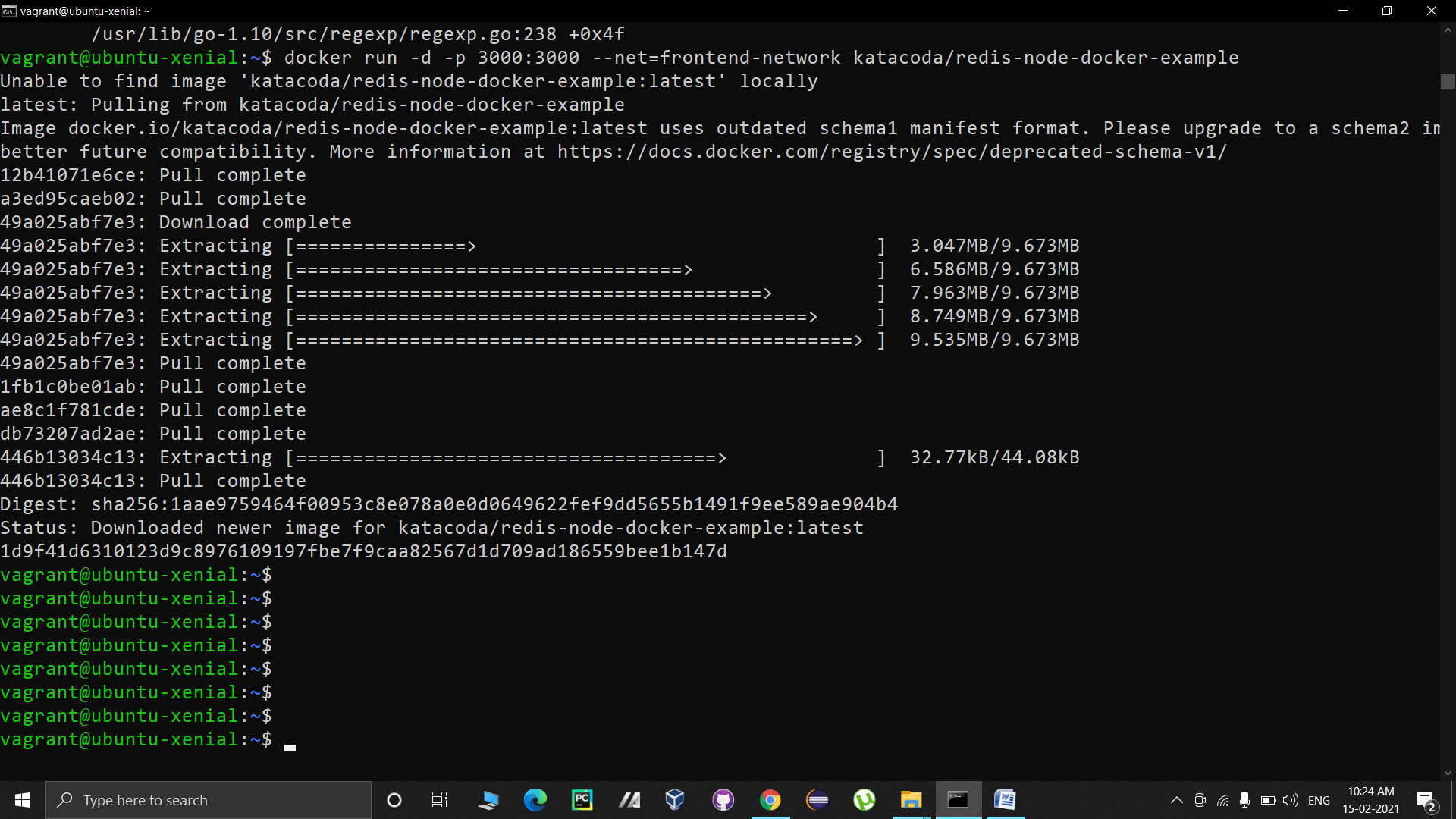
Now we will connect the previously made redis container with this frontend-network. This redis container is already connected with backend-network.

Command: docker network connect frontend-network redis

9. Now we will use a customized redis image created by katacoda to communicateover frontend-network.

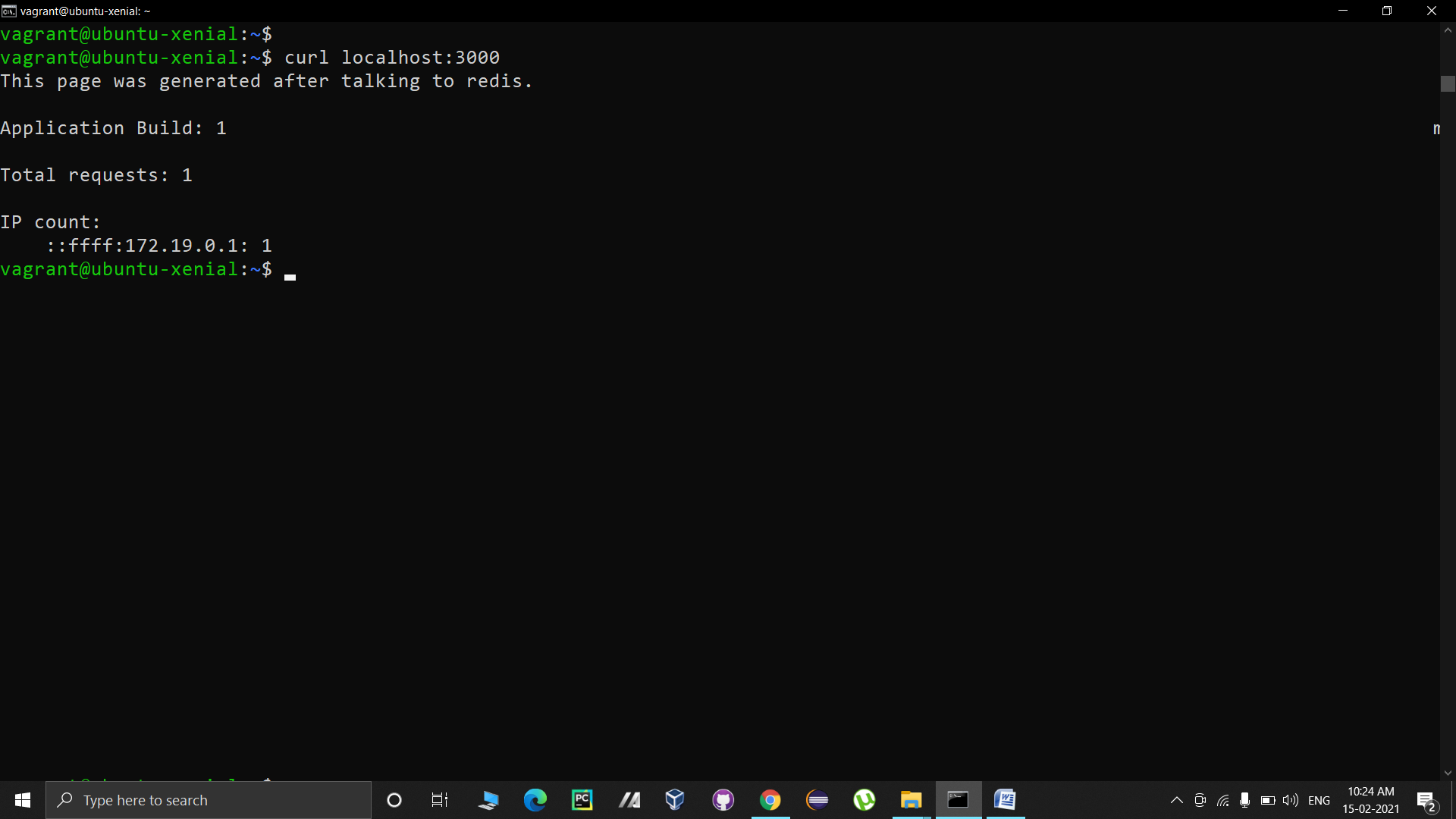
We will bind this customized image to port 3000.

Command: docker run –d –p 3000:3000 --net=frontend-network katacoda/redis-node-docker-example



We can verify that it is able to communicate as follows:

Command: curl localhost:3000



10. We can also create alias to a container. This will create a new entry for the container in the DNS. Let us see how this works.

Let us first create a new network frontend-network2

Command: docker network create frontend-network2

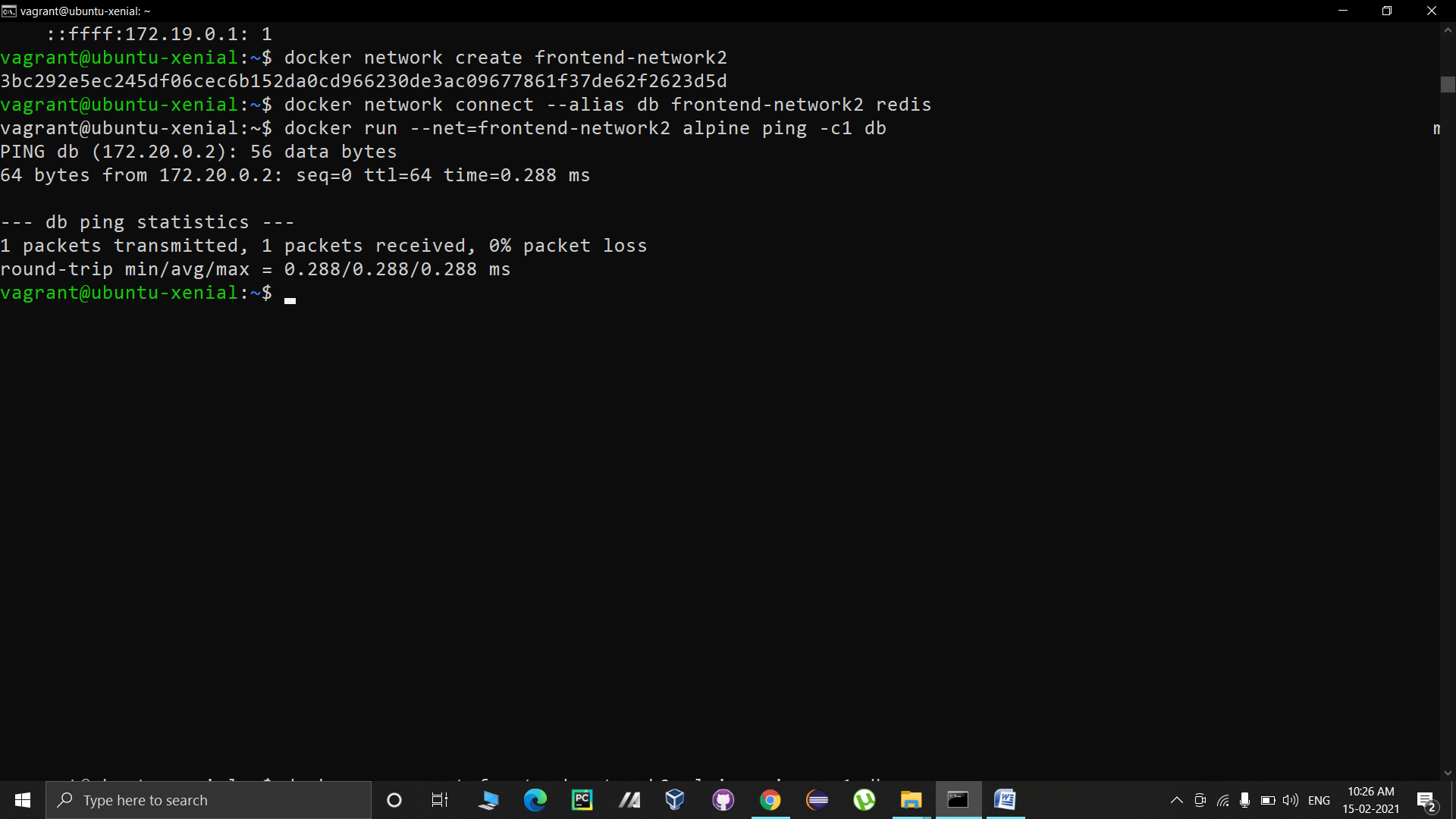
Now we will connect our redis container with this network and we will provide an alias “db”.

Command: docker network connect --alias db frontend-network2 redis

When other containers try to communicate with db, they will be provided with the ip of redis.

We can verify that other container can communicate to our container by using alias.

Command: docker run --net=frontend-network2 alpine ping –c1 db

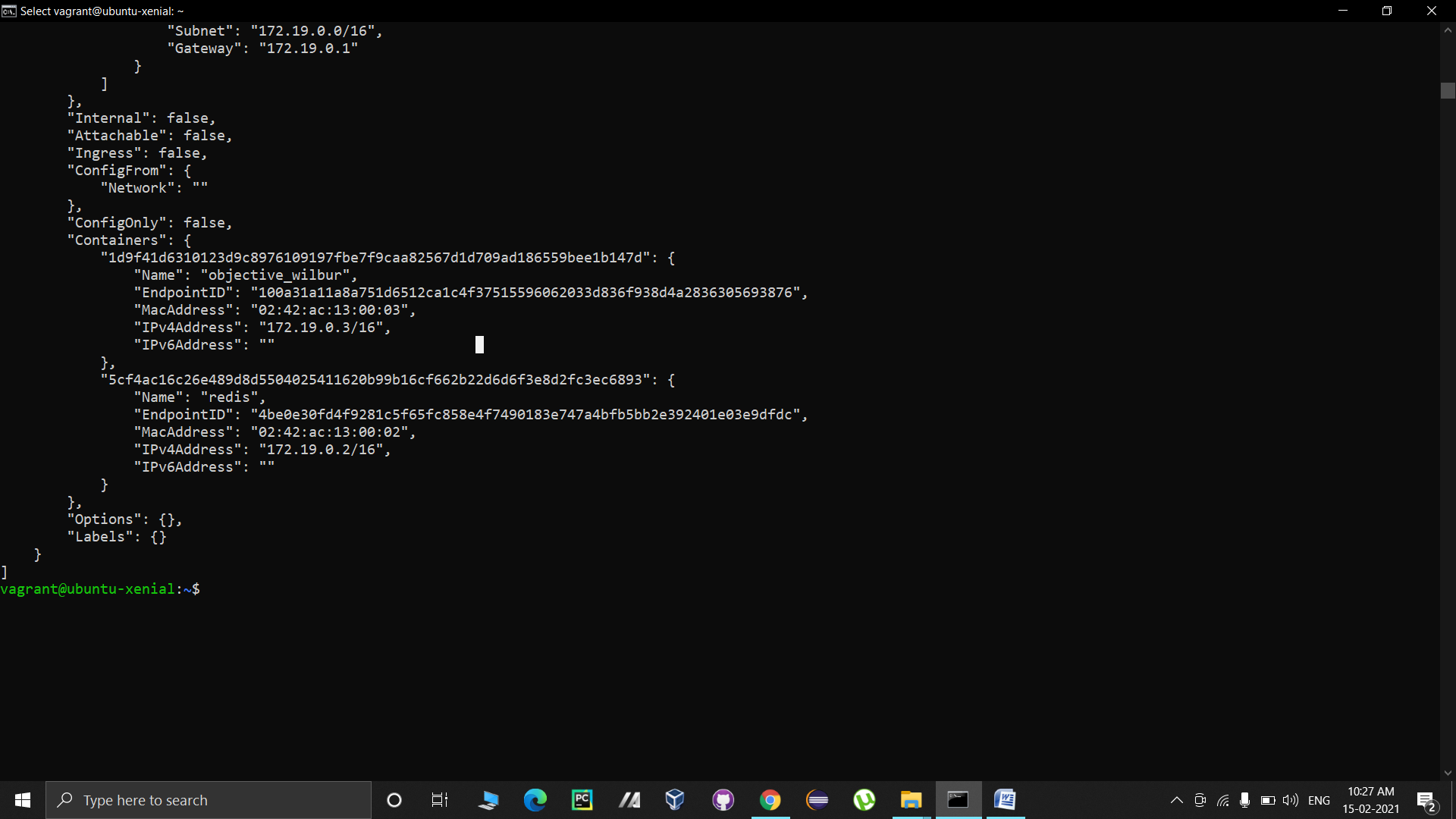


11. We can get the details of any network by using the following command.

Command syntax: docker network inspect <network-name>

Let us inspect frontend-network to see the connected containers.

Command: docker network inspect frontend-network



The above output is truncated form of the complete output which just focuses on the connected containers.

12. We can disconnect a container from a network by disconnect command.

Command syntax: docker network disconnect <network-name><container-name>

Let us use this to disconnect redis container from frontend-network.

Command: docker netwok disconnect frontend-network redis

