# DEVOPS TASK

## 1.NODEJS

1.docker login

2.check docker images

3.docker pull python or nodejs: downloading python images or nodejs images

4.docker images checking: check whether python or noejs file is downloaded or not

5.now reach to the location of the pythonflask or nodejs

6.then use docker build -t tag name : to give the tag name .

7.create repository by reaching to the hub.docker.com in chrome search

8.the copy the docker\_id/repository\_name

9.then use docker tag tag\_name:tag paste the step 8

10.then check docker images

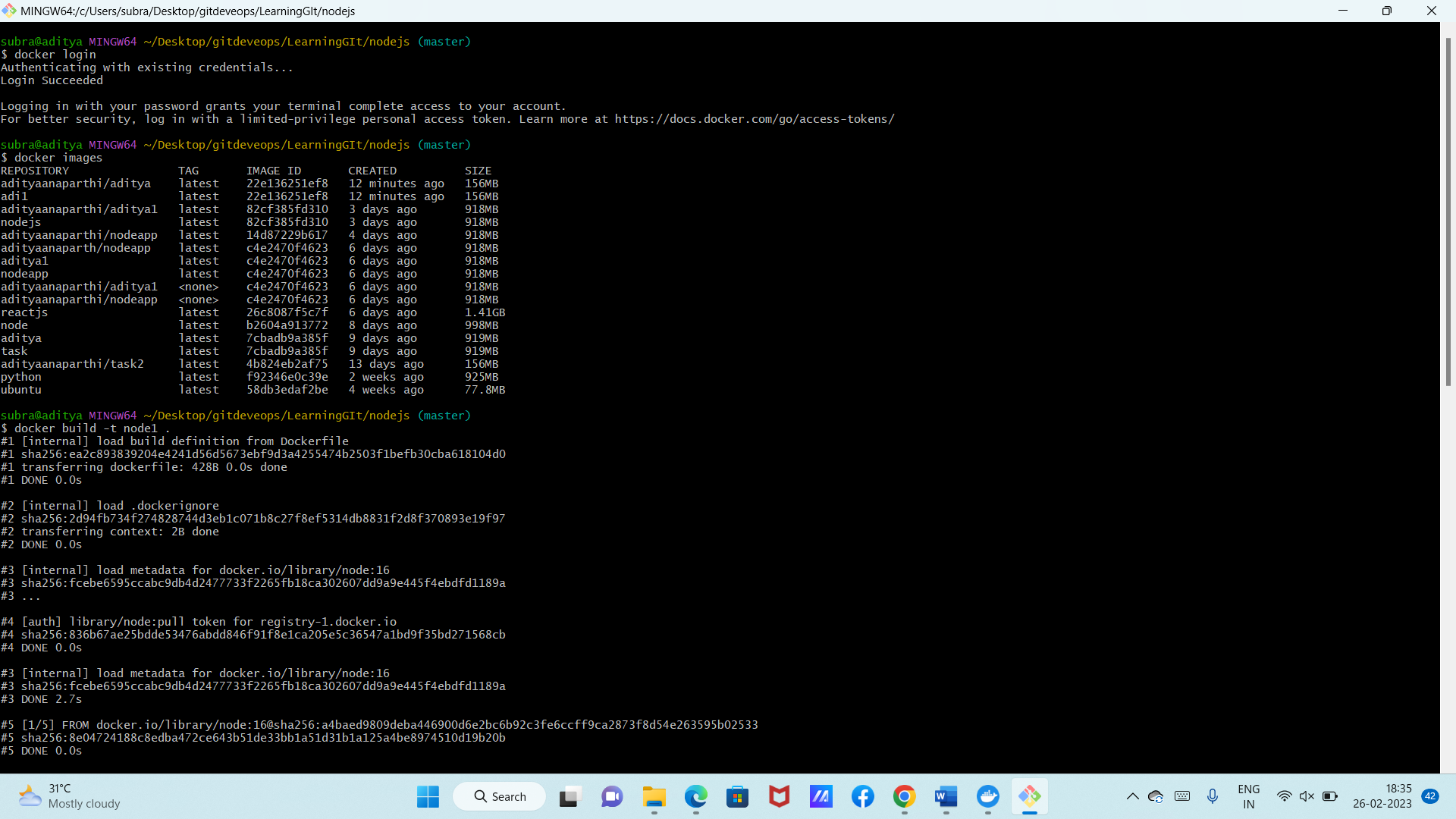
11.then use docker push paste step 8 : then push the tag\_file image to the repository

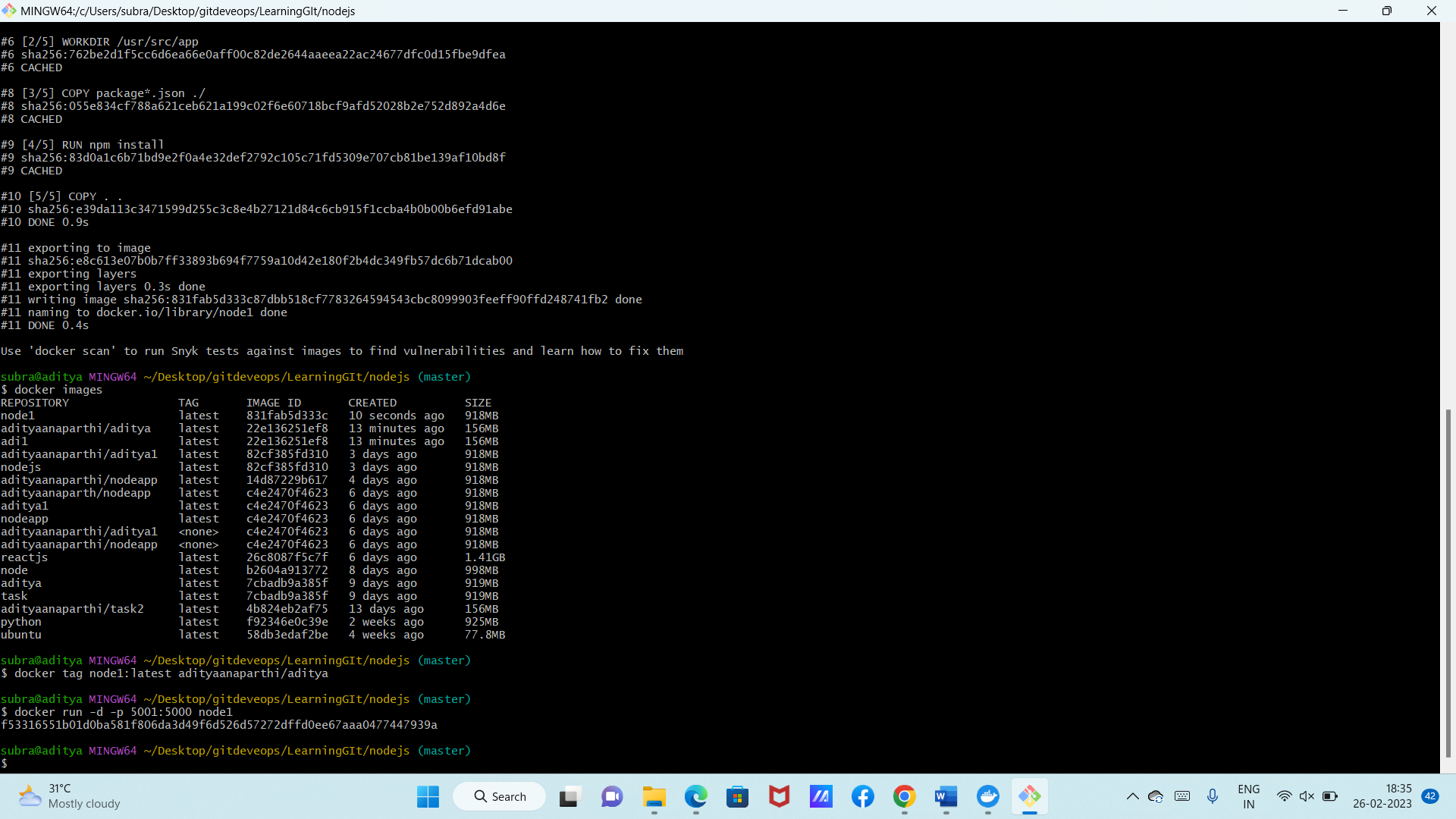
12.now open docker file in pythonflask or nodejs and check the port number

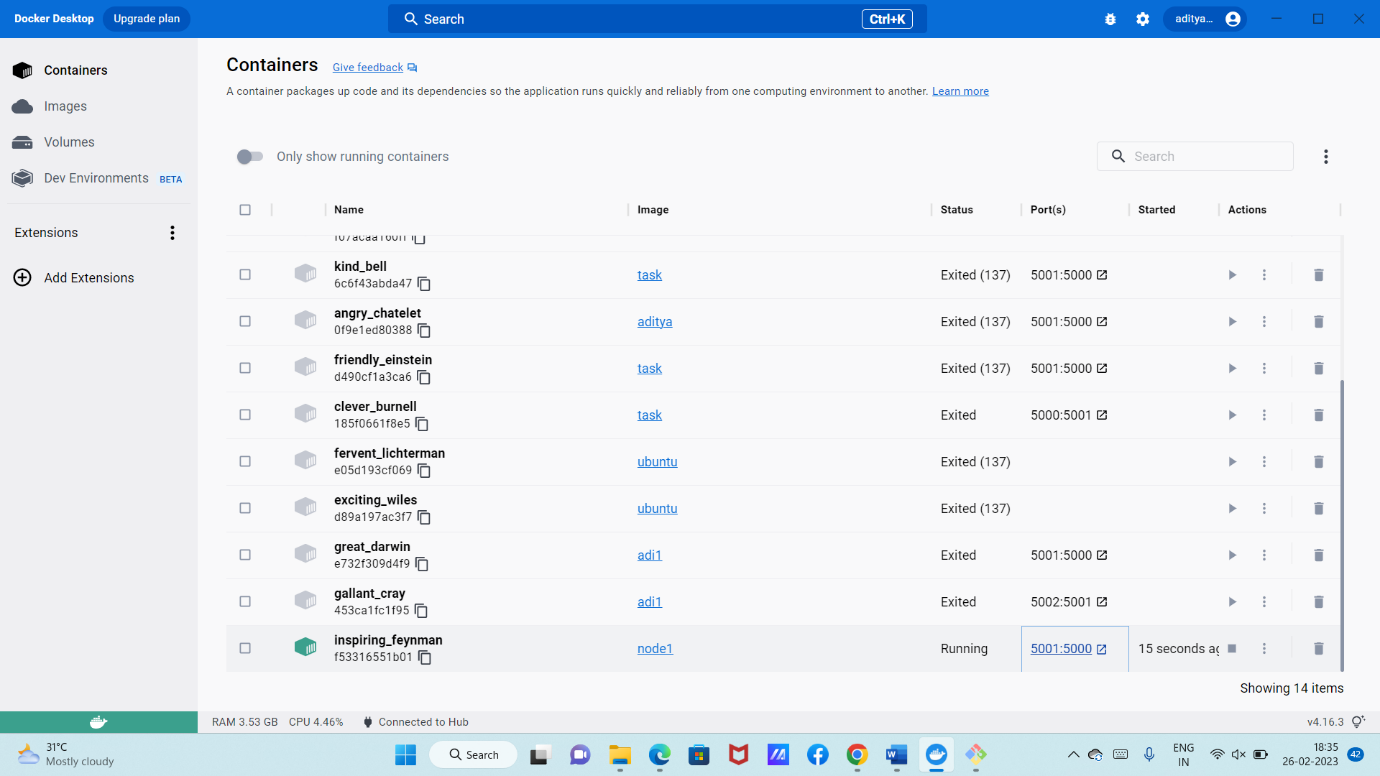
13.then use docker run -d -p port number : given\_port\_number

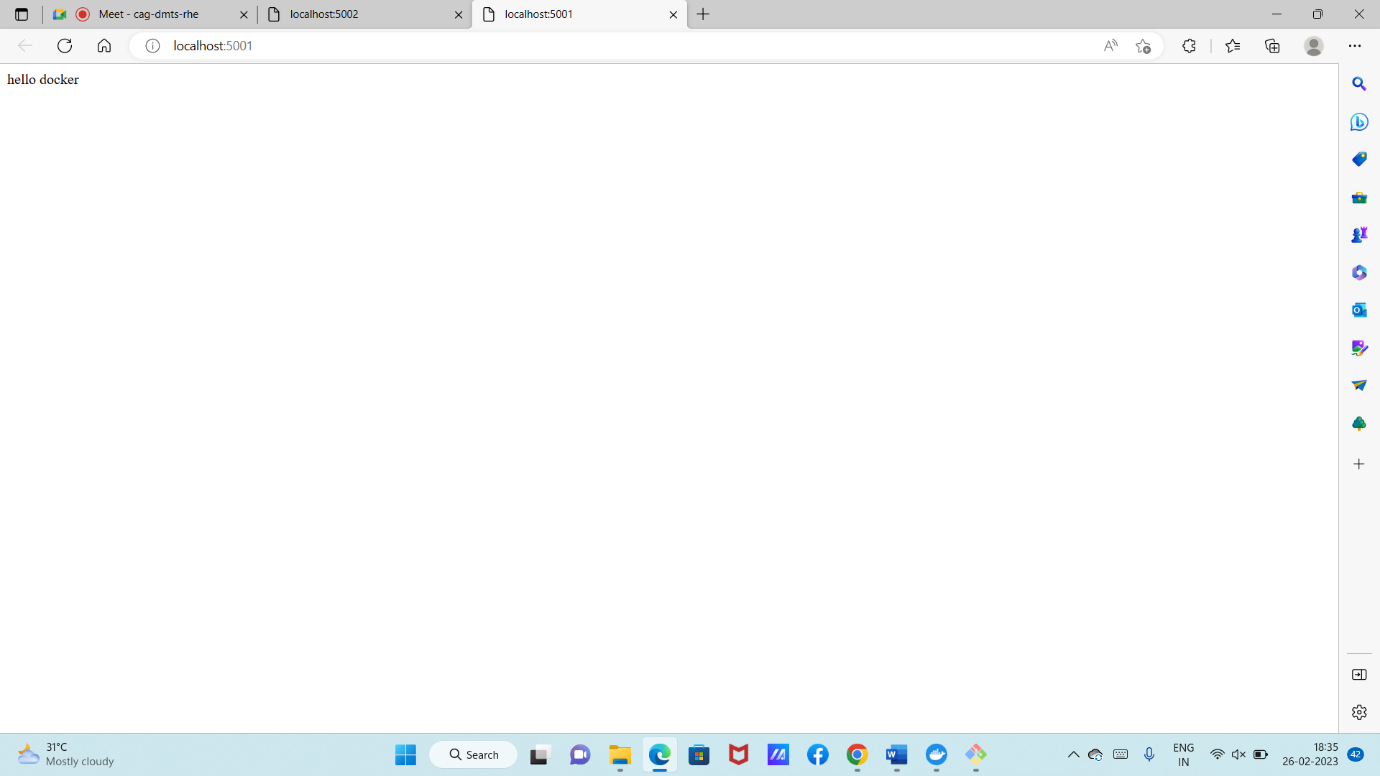
14.then check localhost : given port number

15.then open docker desktop and check whether it is running or not

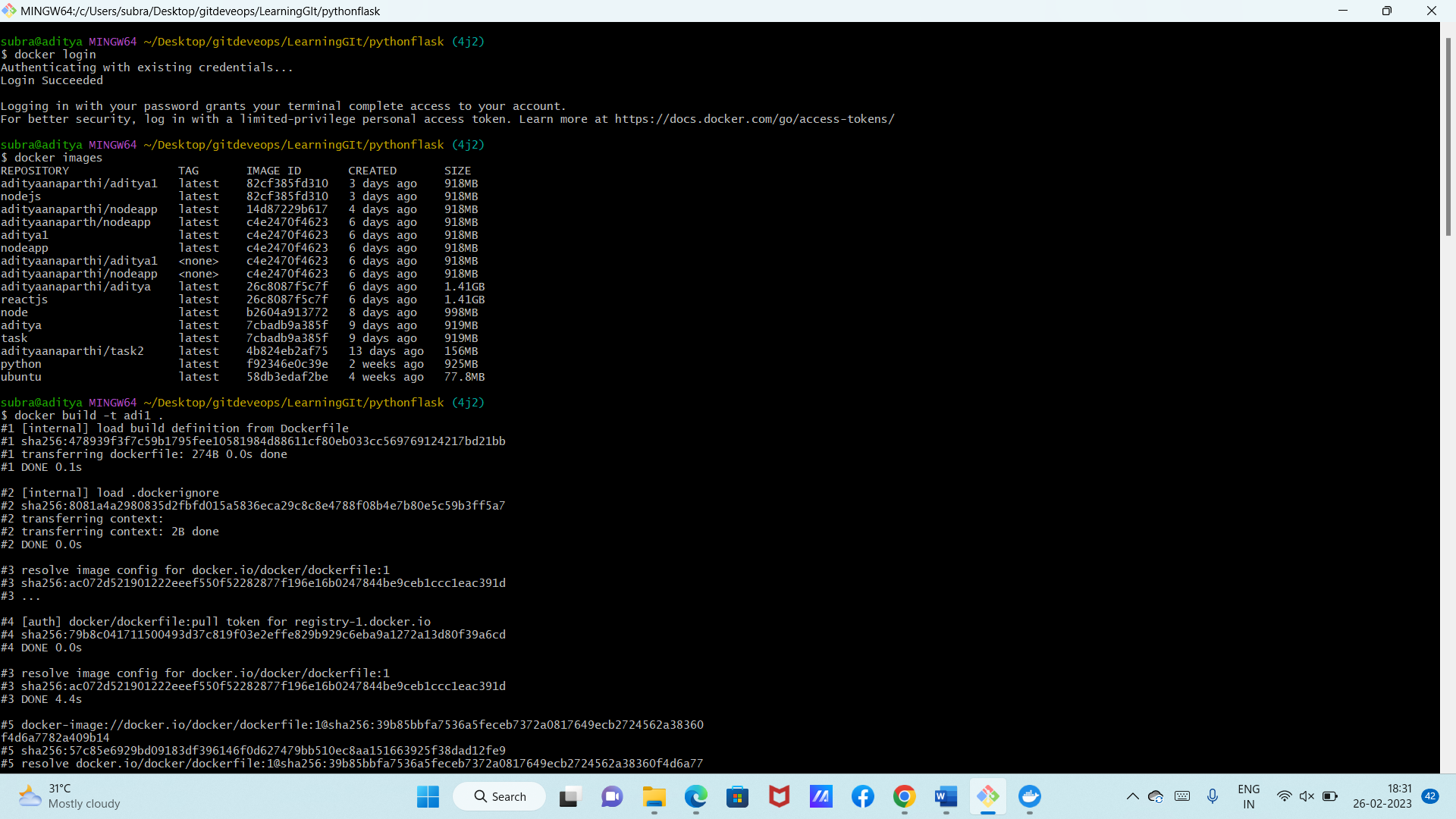


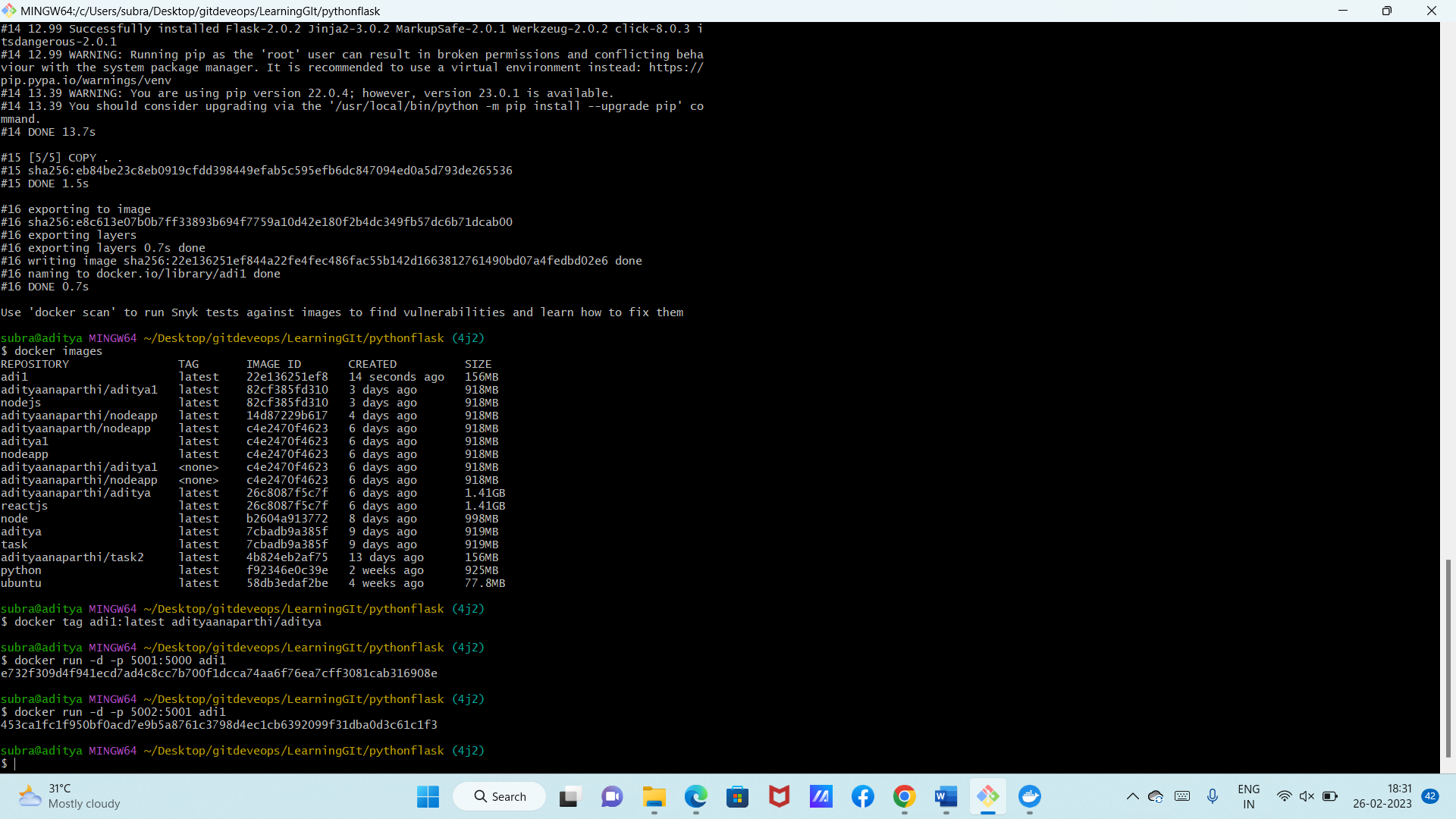


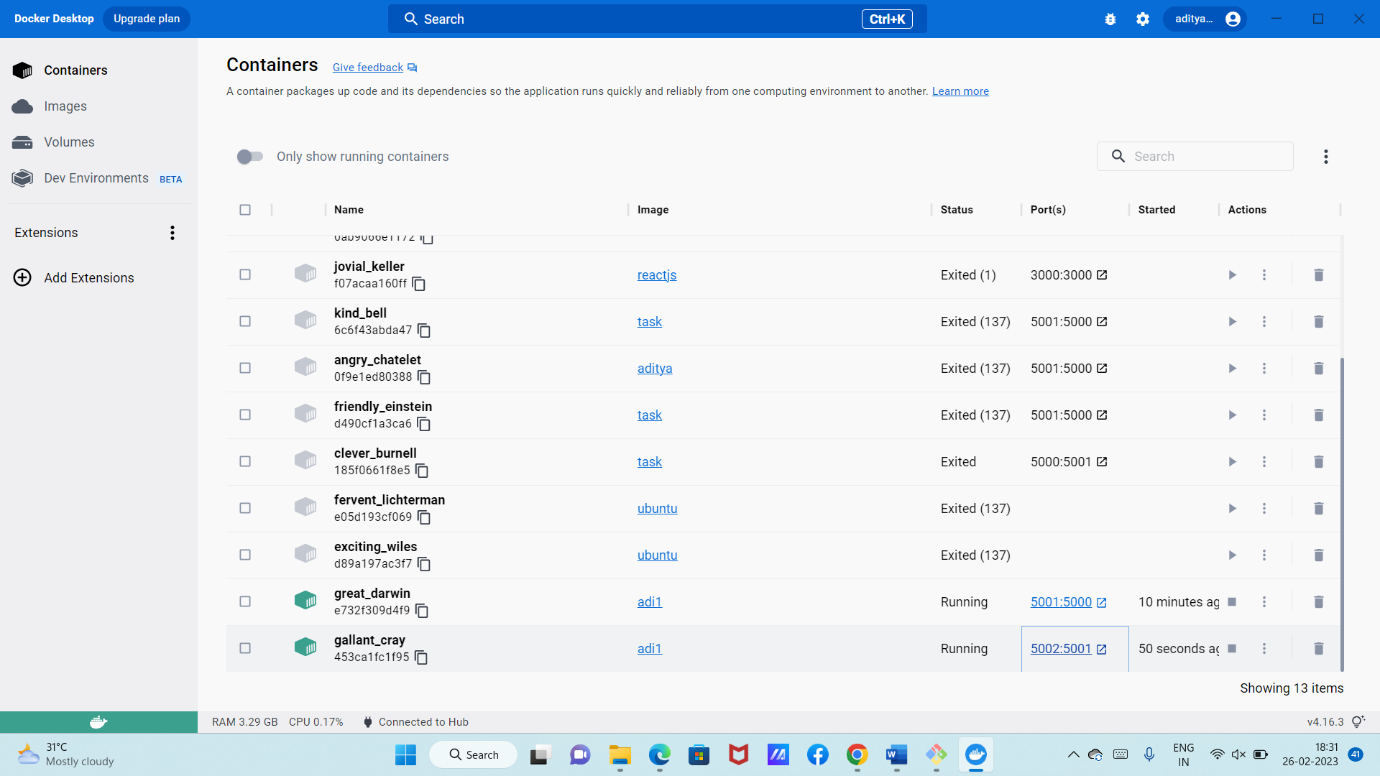


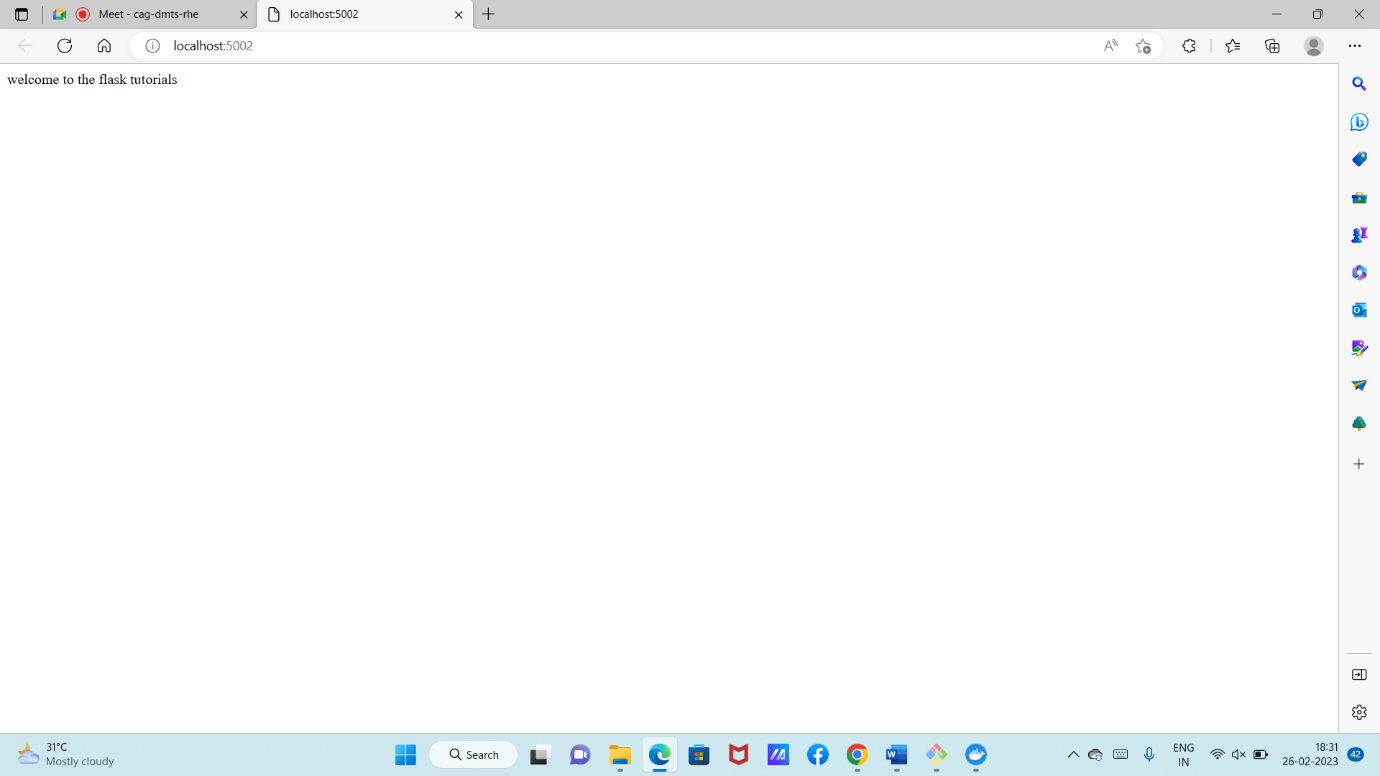


PYTHON









**FROM** defines the base image to use to start the build process.

The **WORKDIR** directive is used to set where the command defined with CMD is to be executed.

**COPY** is a dockerfile command that copies files from a local source location to a destination in the Docker container.

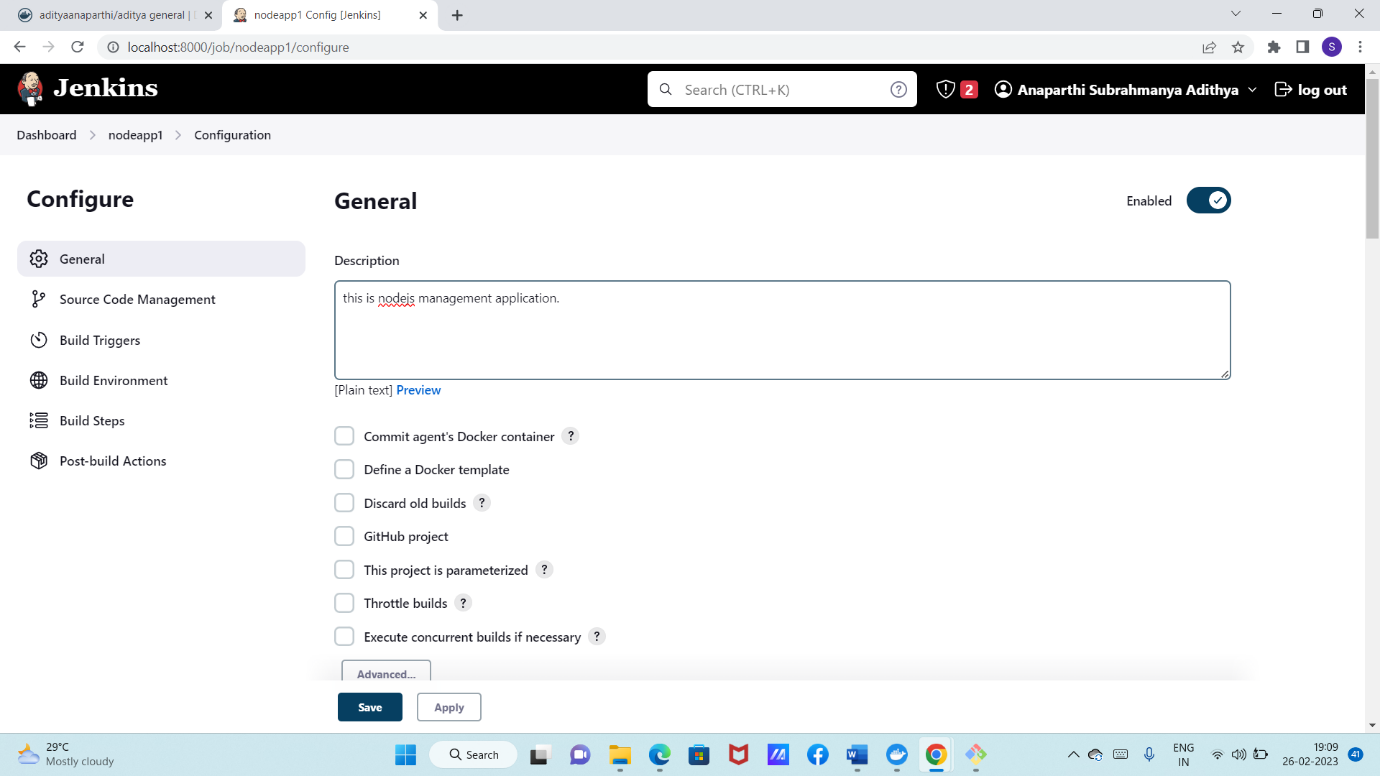
The **RUN** command is the central executing directive for Dockerfiles.

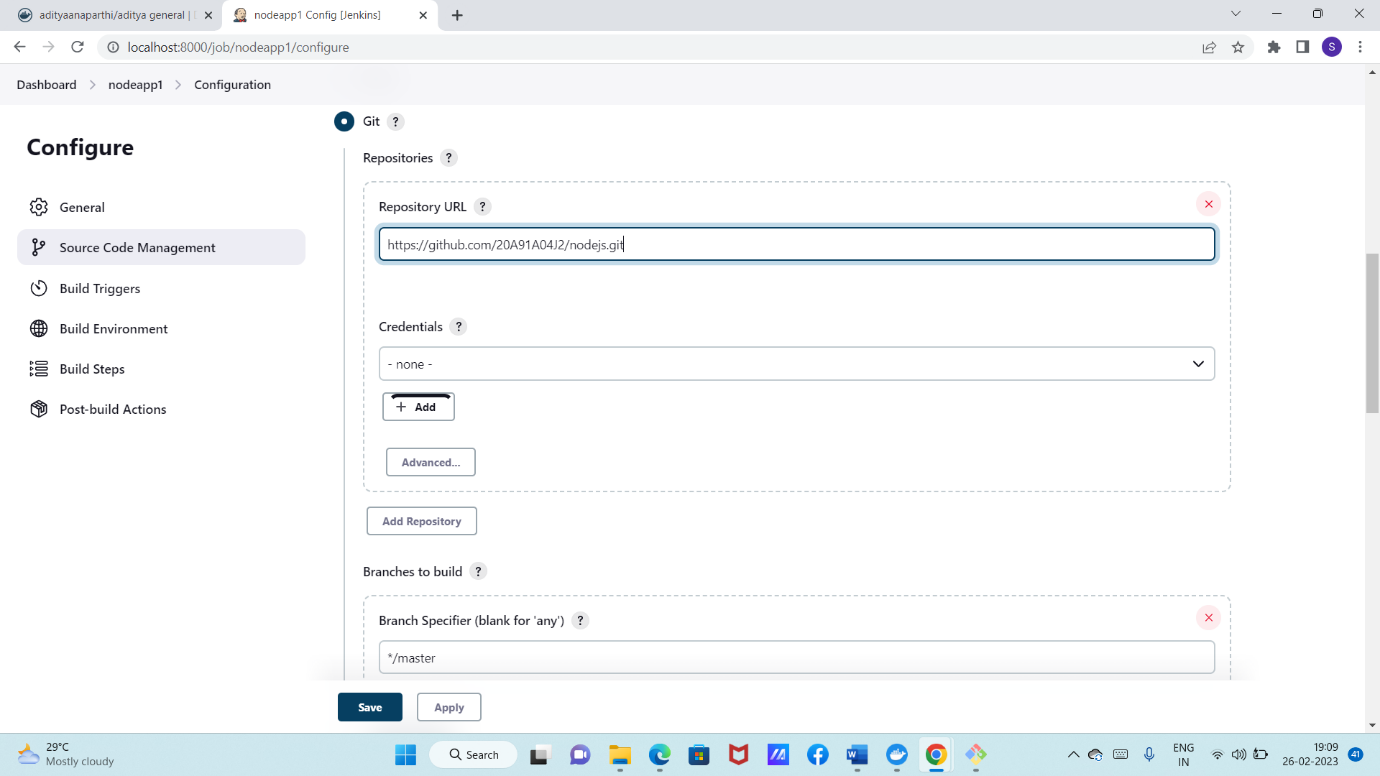
The **EXPOSE** command is used to associate a specified port to enable networking between the running process inside the container and the outside world (i.e. the host).

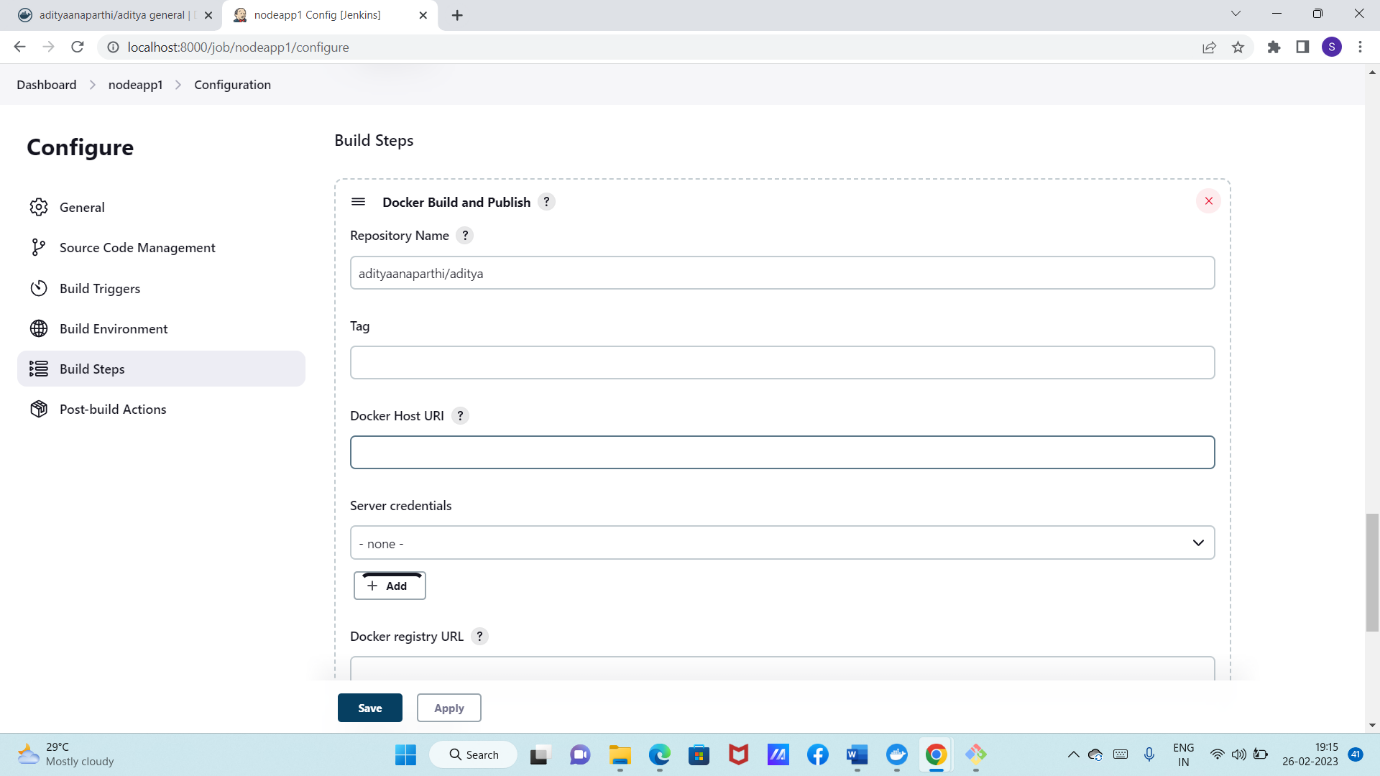
**ENTRYPOINT** argument sets the concrete default application that is used every time a container is created using the image.

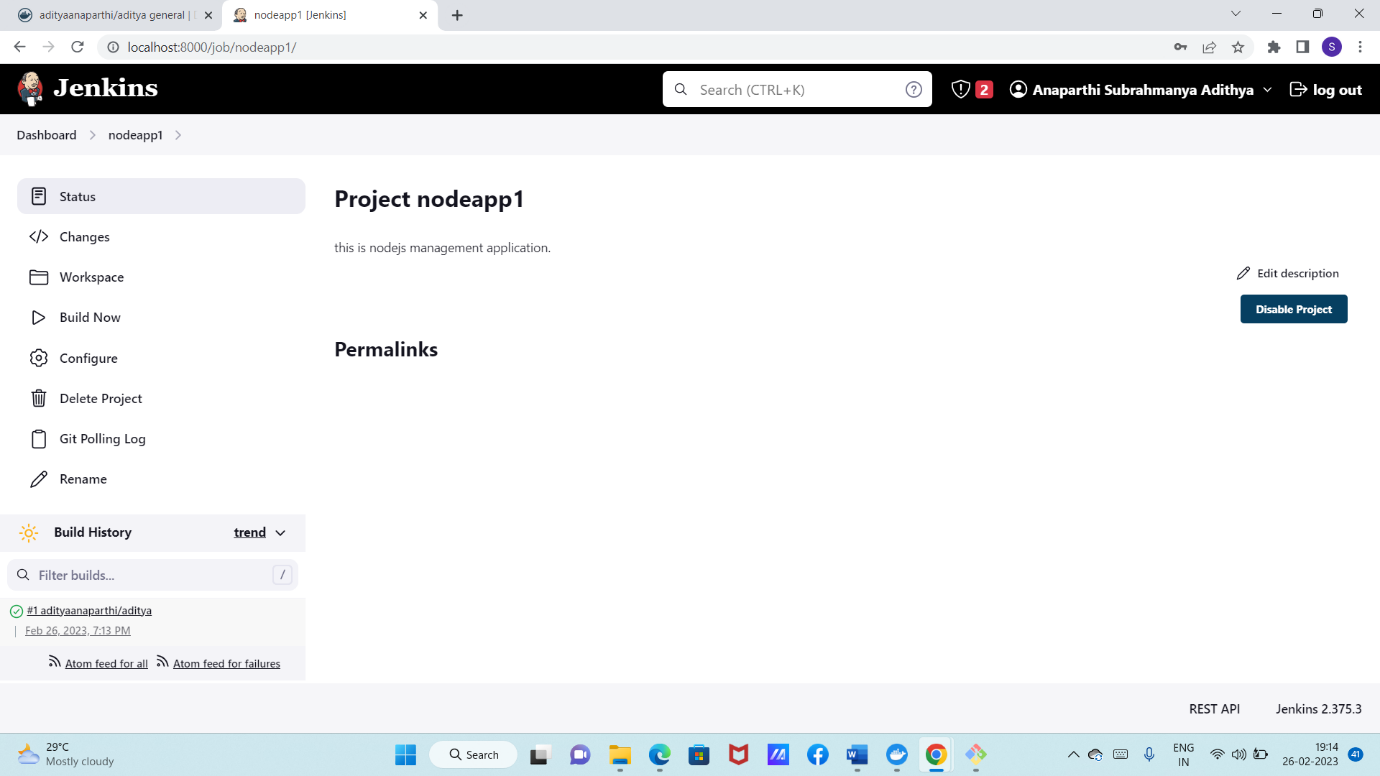
The command **CMD**, similarly to RUN, can be used for executing a specific command.

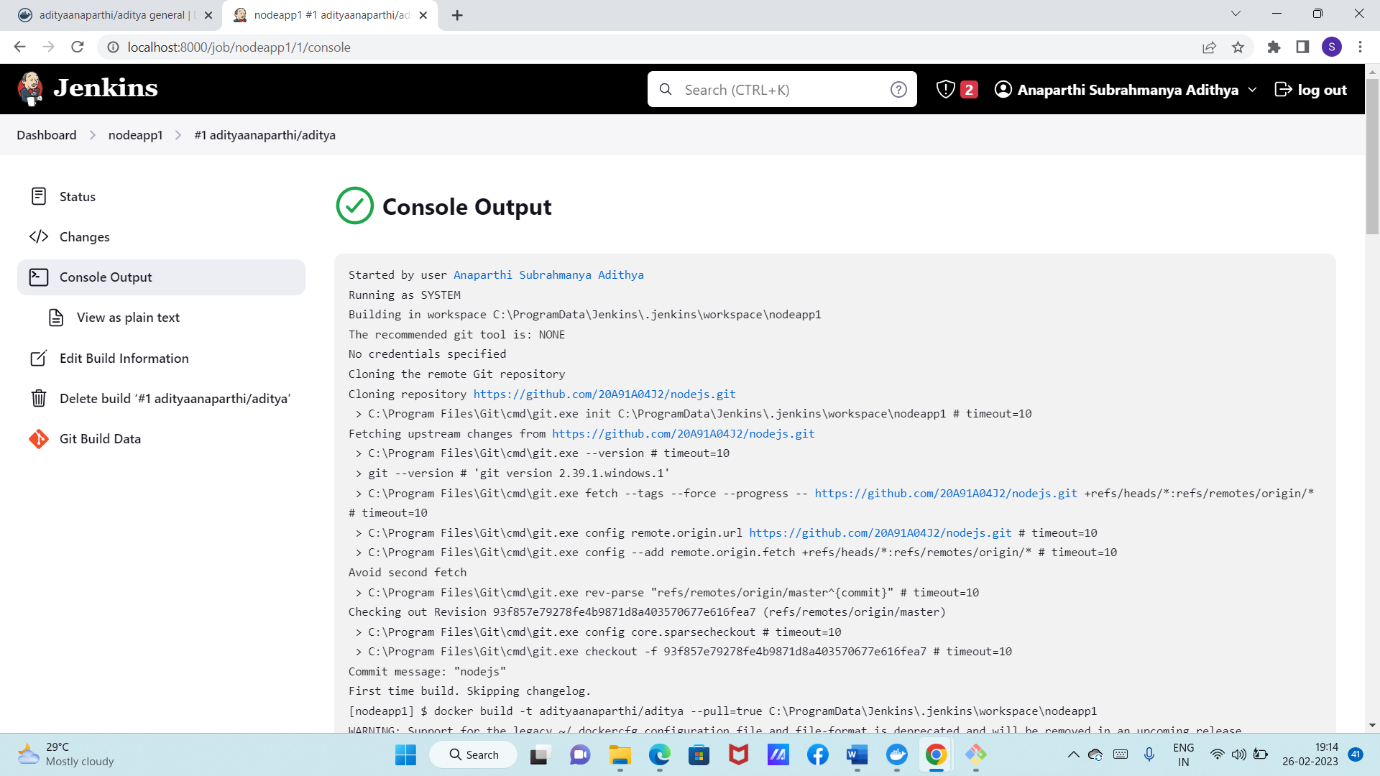
## 2.CI-CD PIPELINE AND JENKINS FOR NODEJS APPLICATION

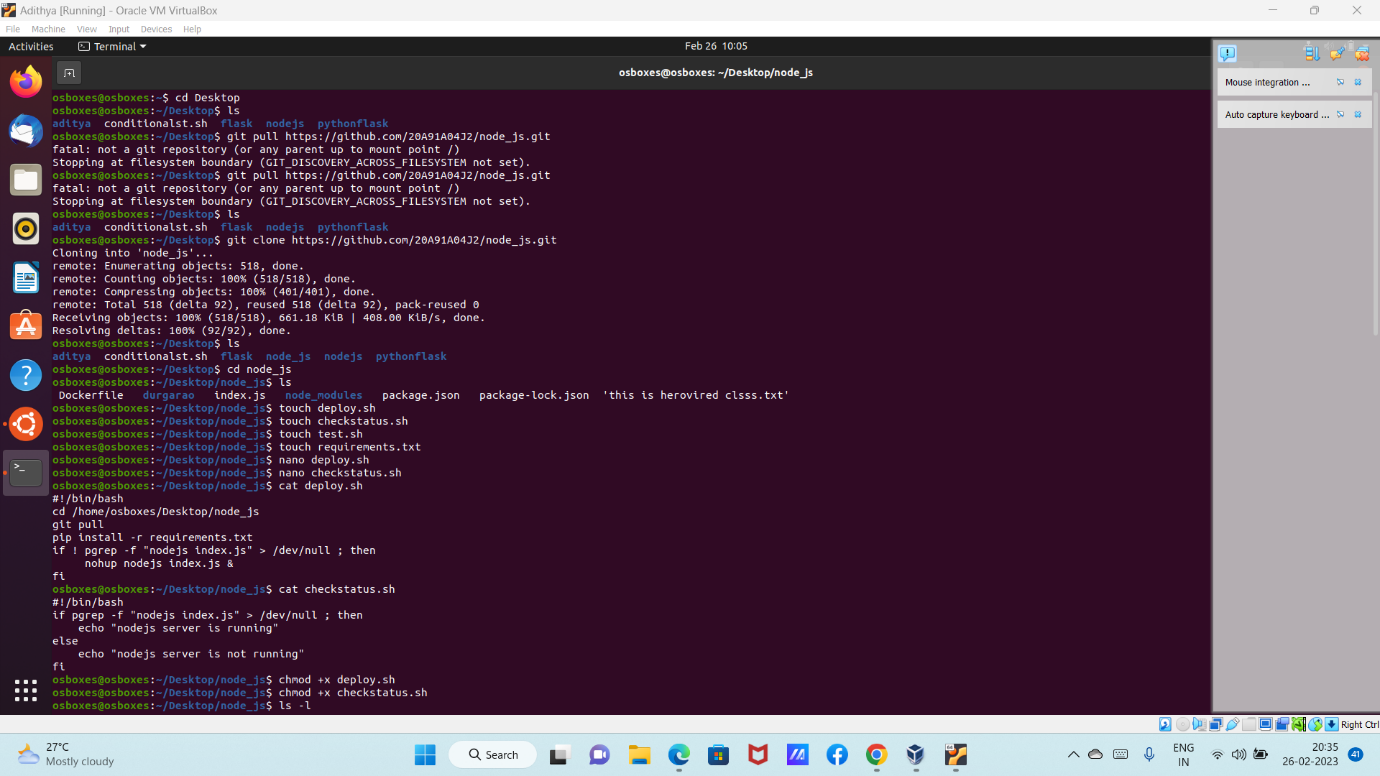


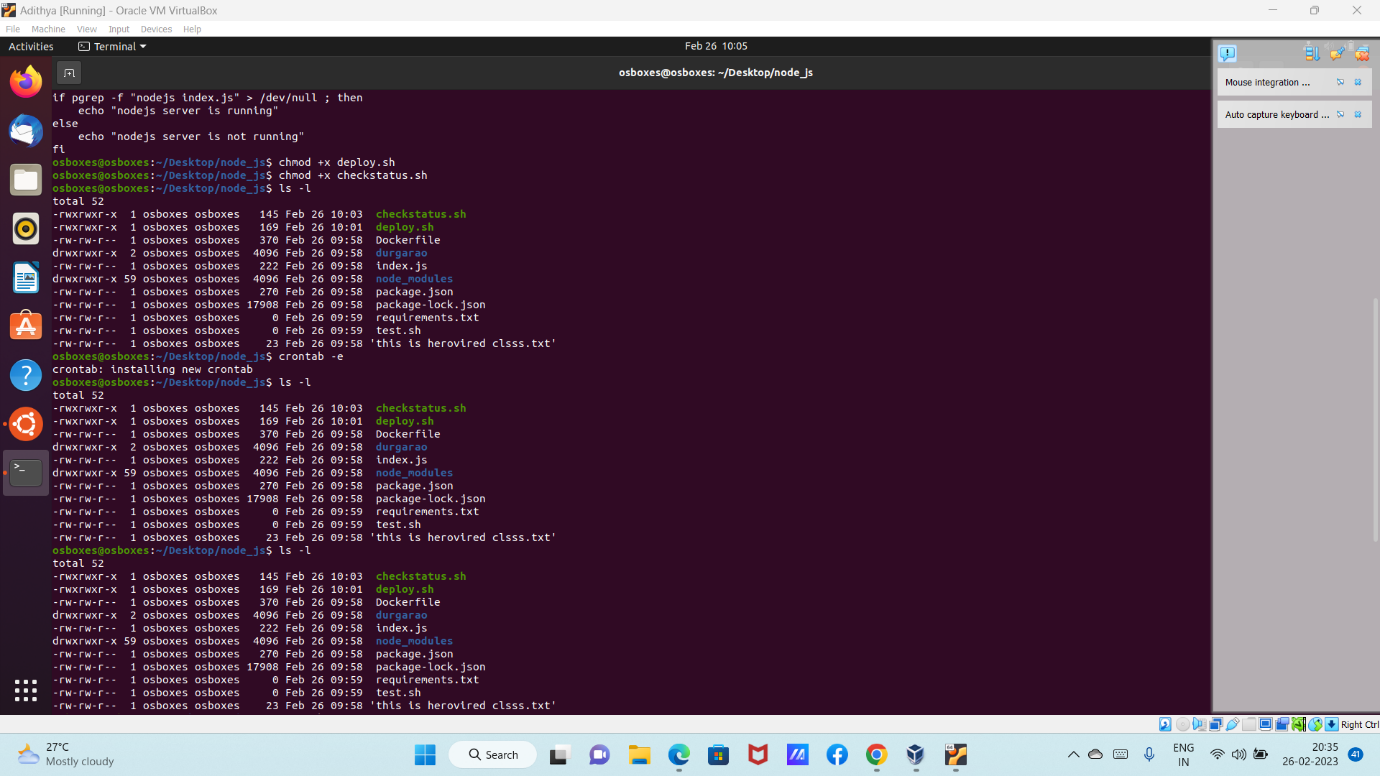


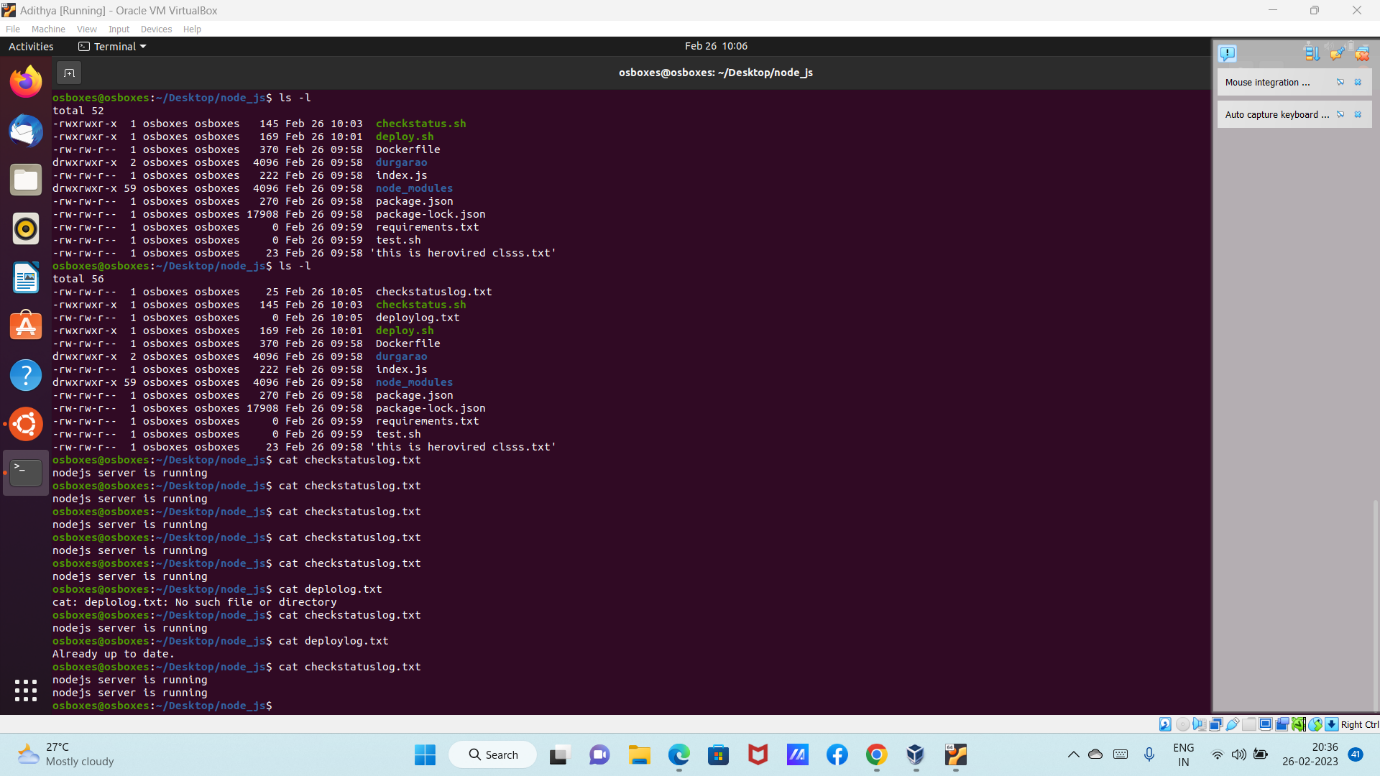


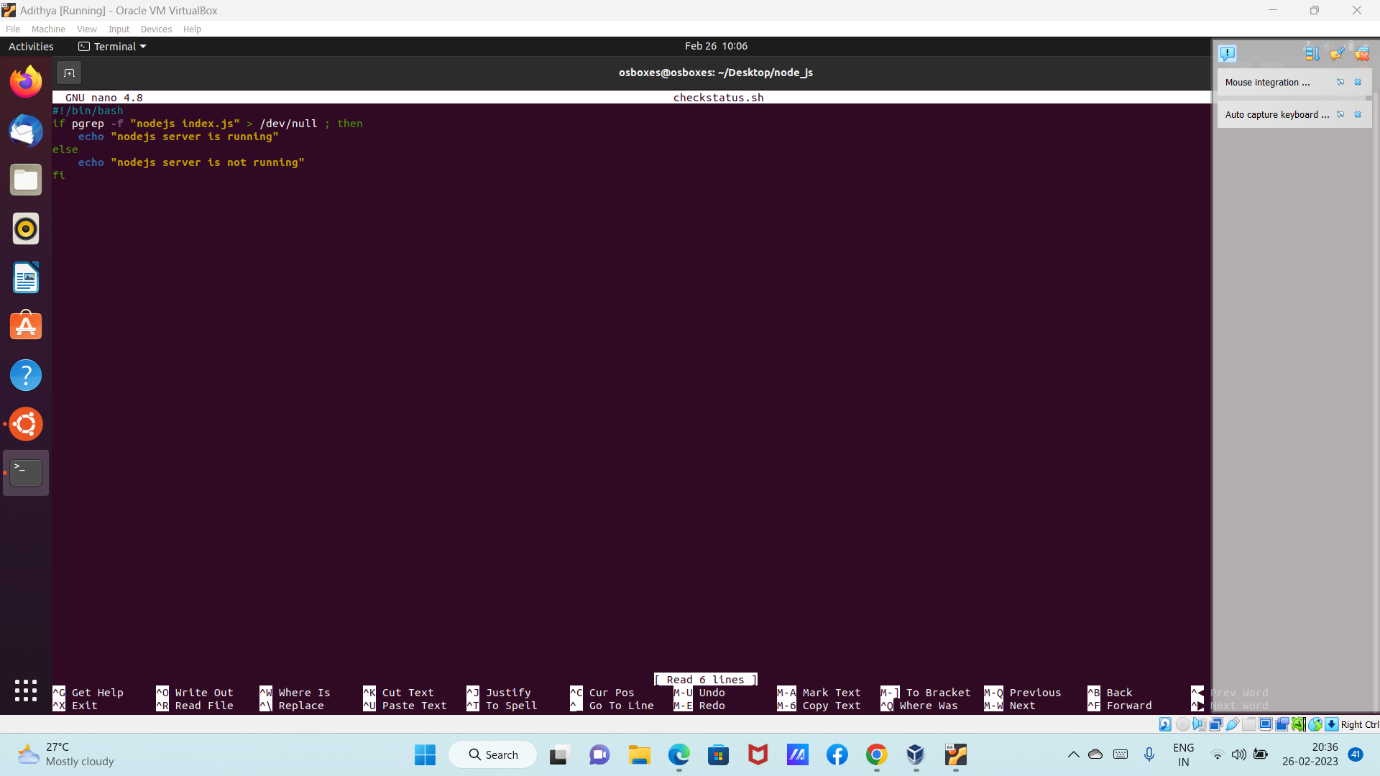


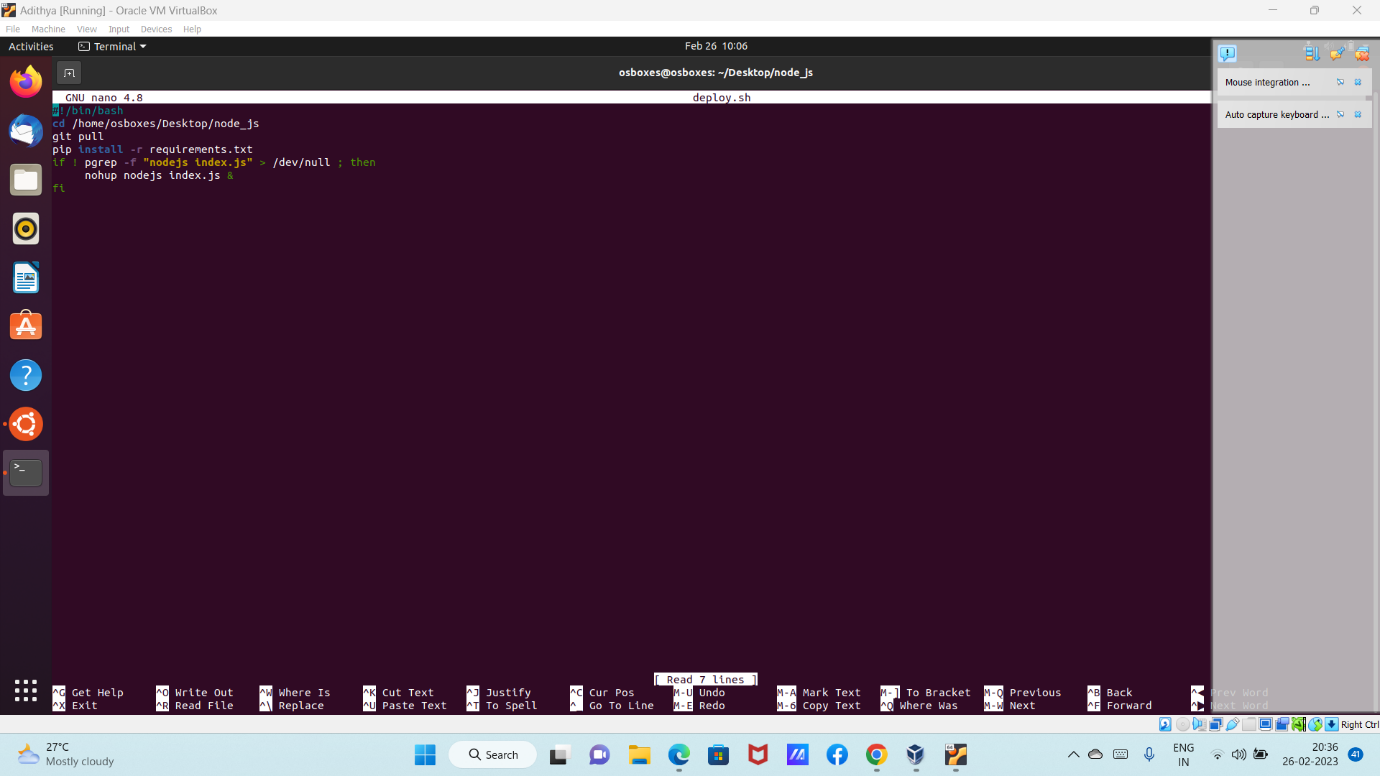






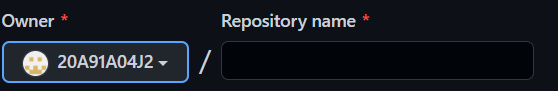






## 3.CI CD PIPELINE FOR PYTHON APPLICATION

Create a new repository in github



Upload the required files for python application

Clone the remote repository to local and push the files

Create a docker image for python app

Create a container and run it

$docker login

$docker build -t image .

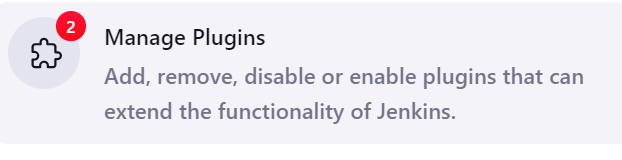
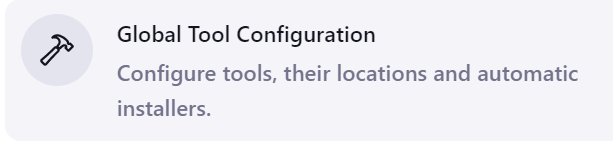
$docker tag image:latest dockerhub\_username/docker\_repository\_name

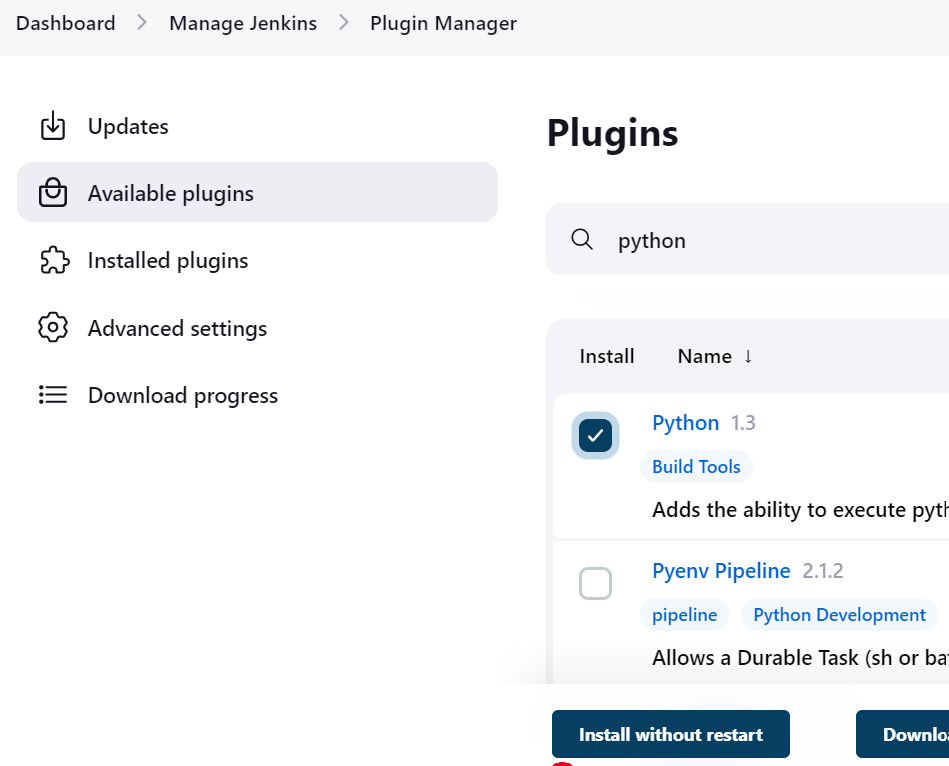
$docker push tag dockerhub\_username/docker\_repository\_name:latest

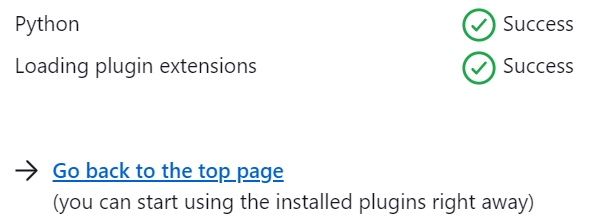
$docker run -d -p port:expose\_port image

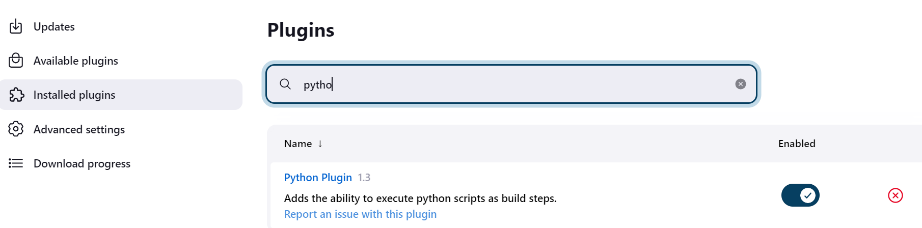
If its working then login to Jenkins

Select the suitable plugins inorder to build the app and install them before configurations.

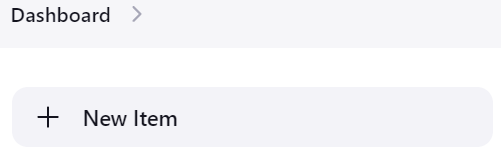
 

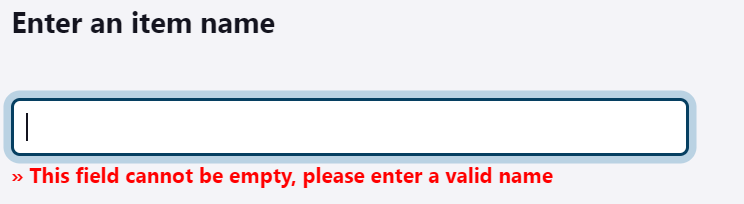


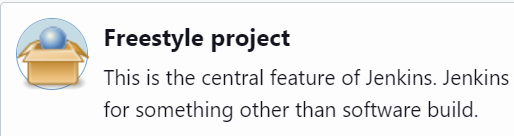




Create a new item or project .



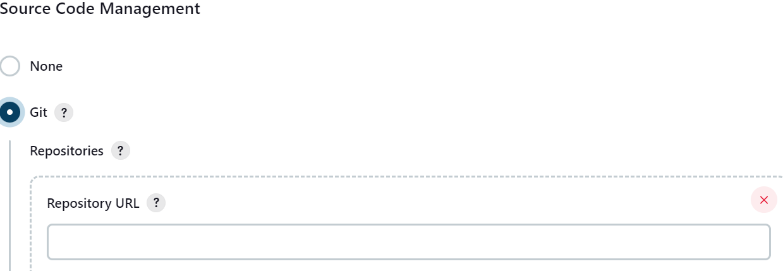
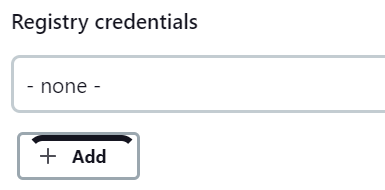




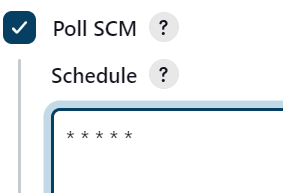
Then give description to your project or application or item.

Then select git and give your python repository llink and select the branch where

Python files are stored.

select poll SCM and \* \* \* \* \* (that means the item will run for every minute)



After go to build steps and select Docker build and publish



In this docker and publish you will give your repository id and give username and

Password of docker hub. So that Jenkins can access into your repository.

After all the steps are done click on capply option and save .

Then click on build now and so now the project or item or application will build

If not check console output and redo the same process.

If the application build then we will get output as tick mark.



Then check the console output.

This is how the whole process of building the python application will be done.