DOCUMENTATION FOR THE ASSIGNMENT:

Steps for setting up the environment :

\* Install Python 3.6 in Ubuntu.

\* Setting up a virtual environment for proper working .

\* Installing flask packages.

\*Installing psycopg2 for creating postgreSQL database in python script.

\* Install Jinja template engine to render template at the local host.

\*Install Gunicorn

\*Install nginx

\*Install ngrok

Steps for creating the API:

\*Making the database

* Create a CSV file of the countries and their capital
* Create a PostgreSQL database . Create a table .
* Copy the csv file in the table.

The code for creating the database has been given in database.py file in the repository.

\*Making the RESTful API

* A simple flask application “app” is created in the file myproject.py.
* In the application first a connection with the database is done using SQLAlchemy.
* Configuration of the application is done then.
* Database model is created for the application.
* A query is sent to the database in python script in the flask application.
* The aplication is connected to the local host .
* A html file is made and connected using Jinja template engine.
* First at local host a html file is rendered which takes user input of the country whose capital is to be searched .
* This user input of capital is queried in the database through the application .
* If the country entered is present in the database then its capital is given as output on the html page at the local host or else an error message is sent.

For Gunicorn Server to interact with the application , an entry point was needed to be created

and that is the WSGI entry point. It is done by creating a file that will serve as the entry point for the application. This fiile is the wsgi.py file in the repository.

\*Connection with the Gunicorn HTTP Server

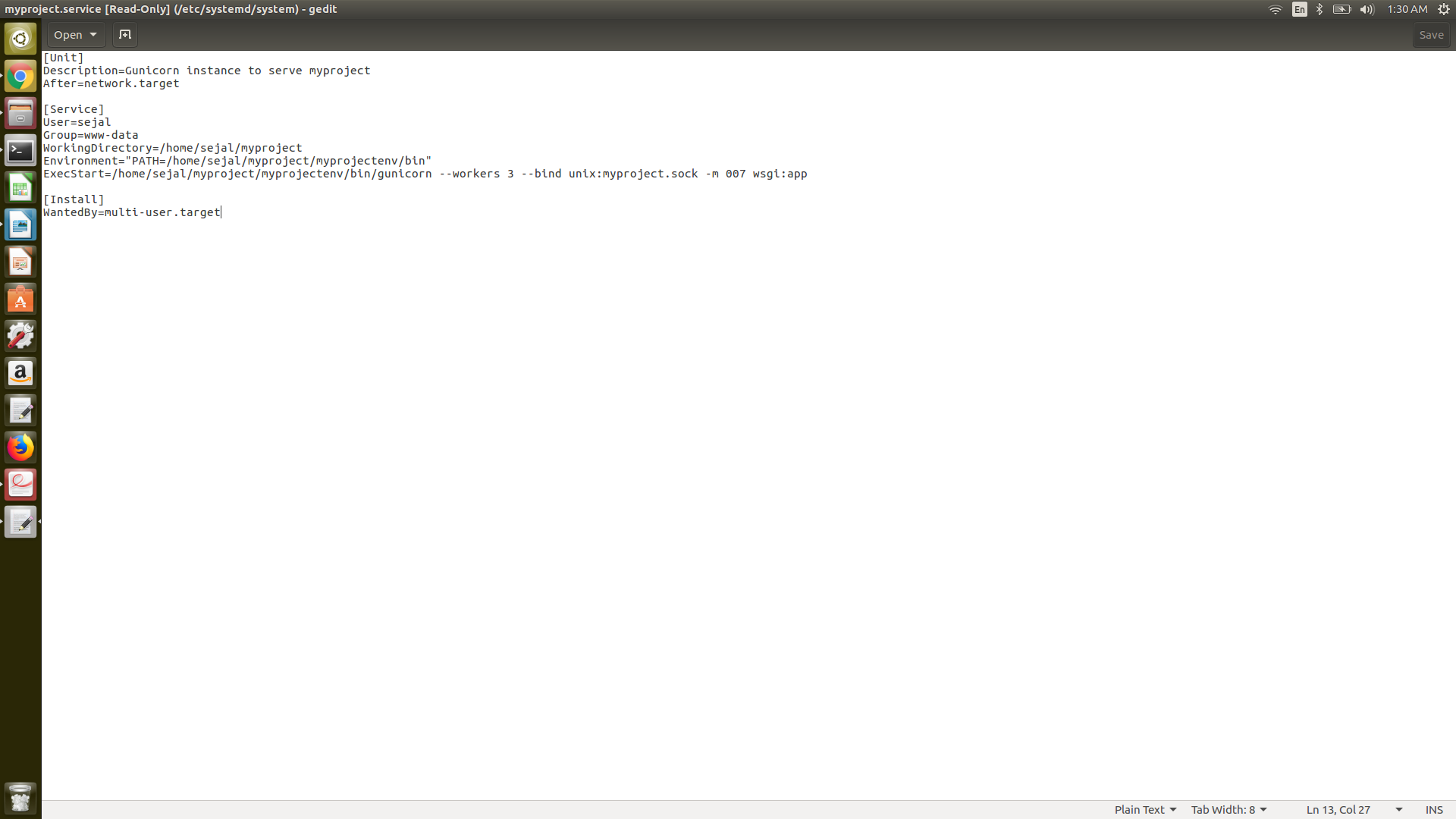
* We can specify the interface and port to bind to so that it will be started on a publicly available interface.
* Now the application can be tested again in web browser with port appended to the local host address.

For the system to automatically start Gunicorn and serve the flask aplication whenever the server boots, a systemd unit file need to be created.

Following steps were performed;

* Create a unit file ending in .service within the /etc/systemd/system directory.
* In the file , create a [Unit]section, which is used to specify metadata and dependencies. Put a description of the service there and init system to will start this after the networking target is reached.
* Open up the Service section. Specify the user and group under which the process will run. I gave our regular user account ownership of the process since it owns all of the relevant files. Give group ownership to the www-data group so that Nginx can communicate easily with the Gunicorn processes.
* Map out the working directory and set the PATH environmental variable so that the init system knows where our the executables for the process are located (within our virtual environment). Specify the commanded to start the service. Systemd requires the full path to the Gunicorn executable, which is installed within our virtual environment.
* Start 3 worker processes . Tell it to create and
* bind to a Unix socket file within the project directory called myproject.sock. Set a umask value of 007 so that the socket file is created giving access to the owner and group, while restricting other access. Pass in the WSGI entry point file name and the Python callable within.

This file is placed at /etc/systemd/system/myproject.service in the system.

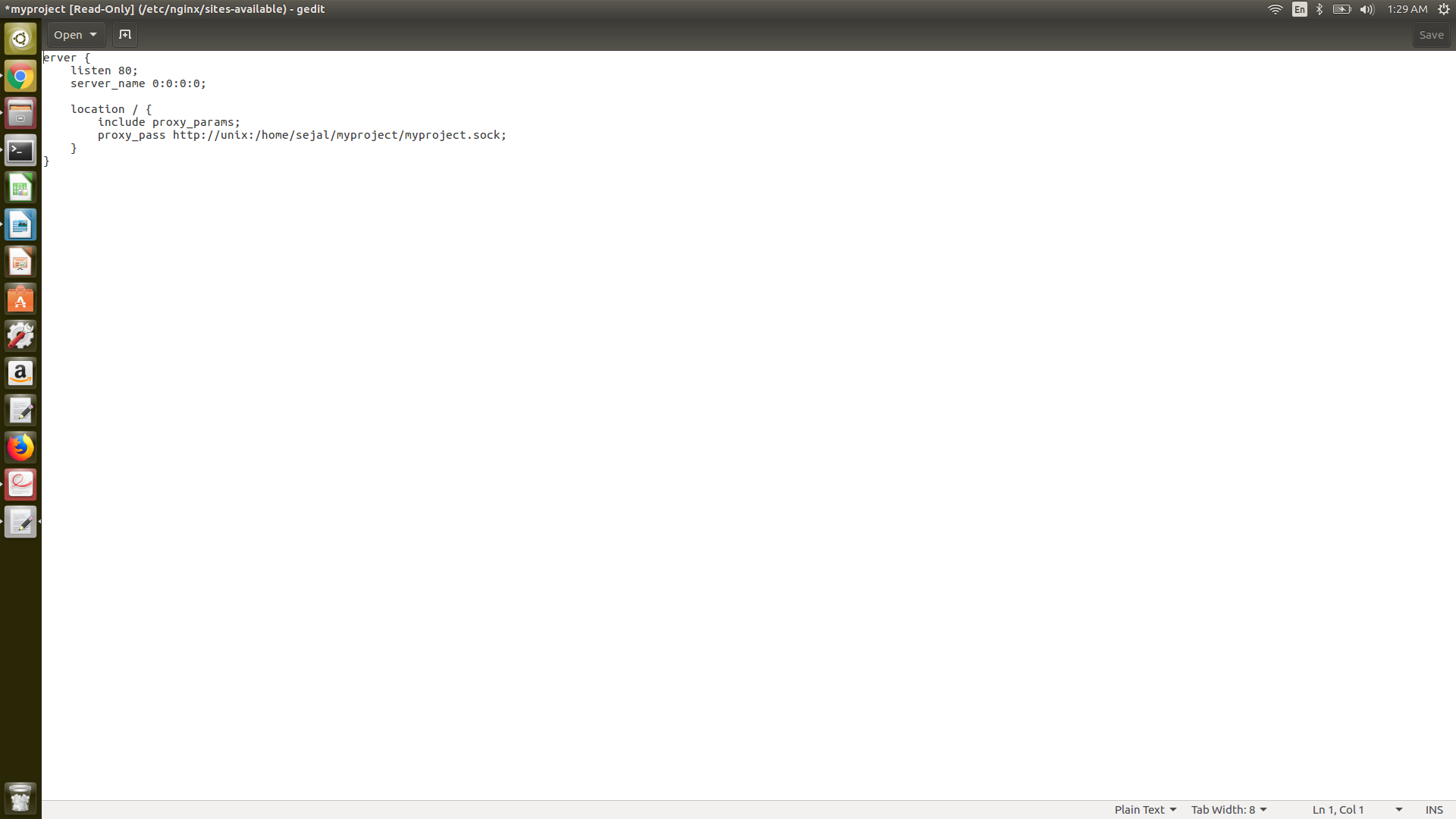


Now start the Gunicorn service and enable it so that it starts at boot.

Now Nginx needs to be configured to proxy requests.

Configure Nginx to pass web requests to that socket by making some small additions to its configuration file.

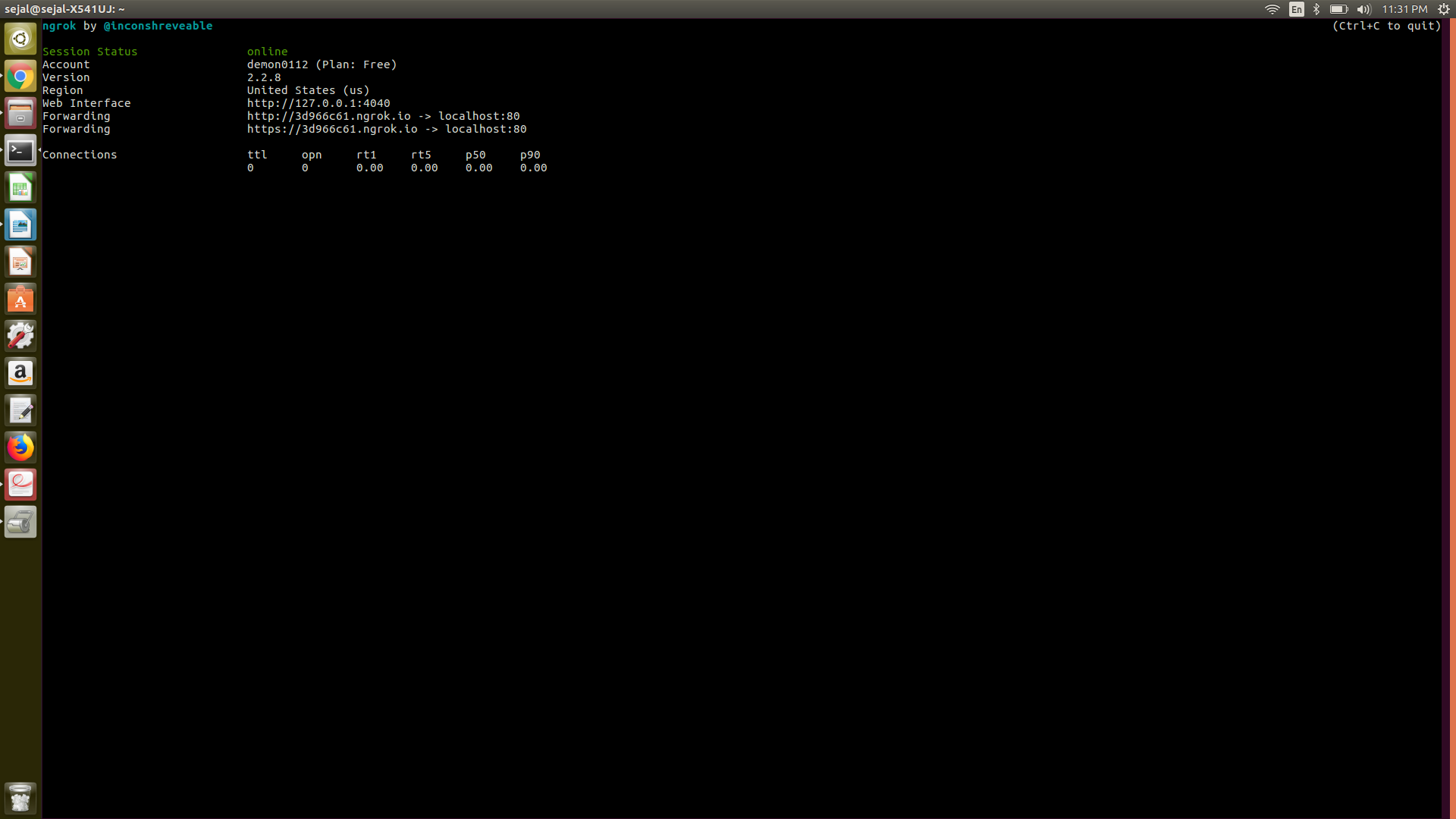
* creating a new server block configuration file in Nginx's sites-available directory.
* Open up a server block and tell Nginx to listen on the default port 80.
* to add is a location block that matches every request. Include the proxy\_params file that specifies some general proxying parameters that need to be set. pass the requests to the socket is defined using the proxy\_pass directive.
* To enable the Nginx server block configuration , link the file to the sites-enabled directory.
* test for syntax errors
* restart the Nginx process to read the new config



* Adjust the firewall again. Port 5000 can be removed and then allow access to the Nginx server:
* Visit the local host.

**Create a ngrok HTTP share**

* Download & install ngrok
* add account's authtoken to ngrok.yml
* open a tunnel pointing to the port where it's currently running

ngrok http 80

**The url link for the api using ngrok is http://3d966c61.ngrok.io+local host**

However this is not currently working due to few errors . I am working on that and will update as soon as possible .

The flask api “app” can be however tested at local host as that is working.

Overall the exercise was quite interesting .The connection part was a bit challenging ..I am still stuck on few last minute errors. The fun part was creating an api . Connecting application to database was a bit complex but was done at the end.

Primarily I went through the documentation of these packages for learning about them and how to deploy them.

First I setup the environment for the api and then started building the api step by step:

installing all the required pakages

creating the database

building the api

connecting api to the database

querying the database through api

rendering the results through html page on local host

creating an entry point for the server

connenting the entry file with gunicorn server

checking the file at that address

creating a systemd file for gunicorn to start immediately as the system starts

creating a socket file

using nginx proxy

starting the api after this at local host

using ngrok to create url for testing