Deploy a Python Django website to Apache Server running on Ubuntu

Preliminary note

in this tutorial we will deploy an ecommerce python base website on a virtual ubuntu server.

Most important prerequisites for this project

- Ubuntu
- Apache
- Django
- PostgreSQL

Creating our ubuntu virtaul machine.

- open your gitbash and move to the home directory `cd ~`
- create a new directory and navigate into it

```
$ mkdir myecommerceapp
$ cd myecommerceapp
```

- now create a new file Vagrantfile in the myecommerceapp folder
- copy and paste the code below inside and save.

```
# -*- mode: ruby -*-
# vi: set ft=ruby :
VAGRANTFILE_API_VERSION = "2"
Vagrant.configure(VAGRANTFILE_API_VERSION) do|config|
    config.ssh.insert_key = false
    config.vm.provider :virtualbox do|vb|
        vb.customize ["modifyvm", :id, "--memory", "2048"]
    end

    #Web Server
    config.vm.define "website-server" do|web|
        web.vm.hostname = "website-server"
        web.vm.box = "bento/ubuntu-18.04"
        web.vm.network "private_network", ip: "192.168.43.14"
    end
    config.vm.box_check_update = false
    config.vbguest.auto_update = false
end
```

- now run vagrant up to start the virtual server.
- `$ vagrant up`

we have our server running so lets go in to deploying our eccomerce website

Naming the host

vagrant ssh to your ubuntu server. the default password is vagrant.

```
# Local Machine
$ vagrant ssh website-server
```

After entering the correct password, you should be logged in as root to the web server. Now you are ready to update the server, the software and its dependencies and also install python-pip for the installation of python base debendencies.

```
# Web Server
# switch to the root user
vagrant@website-server:~$ sudo su
root@website-server:/home/vagrant# sudo apt-get update && apt-get
upgrade
root@website-server:/home/vagrant# sudo apt-get -y install python3-pip
root@website-server:/home/vagrant# pip3 install --upgrade pip
```

The next step would be to give the web server a host name and add the IP address and the given name in the host file located in the /etc/ path of your linux distribution.

we are going to give our host server a name of **website** and the ip of the host is **192.168.43.14**

```
# Web Server
# lets move to the home directory
root@website-server:/home/vagrant# cd ~
root@website-server:~# hostnamectl set-hostname website
root@website-server:~# vi /etc/hosts
```

on the file that you just opened, add the hostname and the api address just below the local host as shown below

```
127.0.0.1       localhost
192.168.43.14   website

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

Save the file and exit.

Installing the firewall

The next step should be to install the firewall first and then configure it.

```
# Web Server
root@website-server:~# sudo apt-get install ufw
#here we allow outgoing traffic
root@website-server:~# sudo ufw default allow outgoing
#here we block all incoming traffic
root@website-server:~# sudo ufw default deny incoming
```

Now it´s important to allow SSH in the rules before we enable them. For test purposes we open port 8000 in the rules as well. Finally we enable the rules of the firewall ufw, an acronym that stands for uncomplicated firewall. The final command status displays the rules currently in force.

```
# Web Server
# here we allow ssh connection
root@website-server:~# sudo ufw allow ssh
# here we configure the fire wall to allow port 8000
root@website-server:~# sudo ufw allow 8000
#here we are enabling the fire wall
root@website-server:~# sudo ufw enable
# here we are getting the status of the fire wall to confirm that it is
active.
root@website-server:~# sudo ufw status
Status: active

To                         Action      From
--                         ------      ----
22/tcp                     ALLOW       Anywhere
8000                       ALLOW       Anywhere
22/tcp (v6)                ALLOW       Anywhere (v6)
8000 (v6)                  ALLOW       Anywhere (v6)
```

Installing the PostgreSQL database on Ubuntu

The next thing we will do is to install PostgreSQL as our database and latter connect it to our application

```
# Web Server
root@website-server:~# sudo apt-get install postgresql postgresql-
contrib
```

next lets set the password and username of our data base then switch to the user. i will use a simple password **utrains** but you can use any one you are confortable with just make sure you take note of it.

```
# Web Server
# setting password for user postgres
root@website-server:~# sudo passwd postgres
#changing to postgres user
root@website-server:~# sudo -u postgres psql
```

On the PostgreSQL terminal, we first create the database for the website and a user who performs the database queries . You have to grant this very user all privileges. This sensitive data such as database, user name or password must of course be stored in the database area of settings. py or the corresponding environment variables. With the last command \q the terminal can be left again.

```
postgres-# CREATE DATABASE website;

postgres-# CREATE USER webuser WITH PASSWORD 'utrainsdb';

postgres-# GRANT ALL PRIVILEGES ON website to webuser;

postgres-# \q
```

Finally, we restart the PostgreSQL server back inside the virtual environment. Using the second command is useful if you want to examine the system status of PostgreSQL.

```
# Web Server
root@website-server:~# sudo service postgresql restart
 * Restarting PostgreSQL 9.3 database
server                            [ OK ]
root@website-server:~# sudo service postgresql status
9.3/main (port 5432): online
```

Clone the website code from GitHub.

Now that we have completed the Linux configuration, it is time to clone our website code from our github repository

we are going to use git, so first we will install git, then use it to clone the repository.

```
root@website-server:~# apt-get -y install git
root@website-server:~# git clone https://github.com/tatahnoellimnyuy
/eccommerce.git
```

If everything worked fine, the project folder should be listed inside the user's home directory.

```
#if the clone is successfull  you will get see the eccommerce folder in
the home directory
root@website-server:~# ls
eccommerce
```

Adding our postgres configurations to the python app

for our application to connect to postgresql we need to add the user configurations in the settings.py of our app.

```
root@website-server:~# vi eccommerce/demo/settings.py
```

scroll down to DATABASES and edit the configuration as shown below.

```
#DATABASES = {
#    "default": {
#        "ENGINE": "django.db.backends.sqlite3",
#        "NAME": os.path.join(BASE_DIR, 'db.sqlite3')
#    }
#}
DATABASES = {
    'default': {
        'ENGINE': "django.db.backends.postgresql",
        'NAME': "website",
        'USER': "webuser",
        'PASSWORD':"utrainsdb" ,
        'HOST': "localhost",
        'PORT': "",
    }
}
```

Creating a virtual environment to install the python website dependencies.

now lets create a virtual environment where we will install the dependencies of our application. we will need to install virtualenv used for the creation of python virtual environments

```
# Web Server
#now lets install vitualenv we will use it to create a python virtaul
environment
root@website-server:~# pip3 install virtualenv
```

Now we can create the virtual environment. Inside our ecommerce project folder there will be generated a new folder that manages the virtual environment. The name of the folder is in our case venv.

```
# Web Server
root@website-server:~# virtualenv eccommerce/venv
```

After switching to the Django project folder, the virtual environment can now be activated.

```
# Web Server
root@website-server:~# cd eccommerce
root@website-server:~/eccommerce# source venv/bin/activate
```

Now, if the prompt brackets (venv) are displayed at the beginning, it´s a good sign, that the virtual environment was successfully activated. Now it´s time to install all dependencies.

```
# Web Server (Virtual Environment)
(venv) root@website-server:~/eccommerce# pip3 install -r requirements.
txt
```

Collecting static files of our Ecommerce app

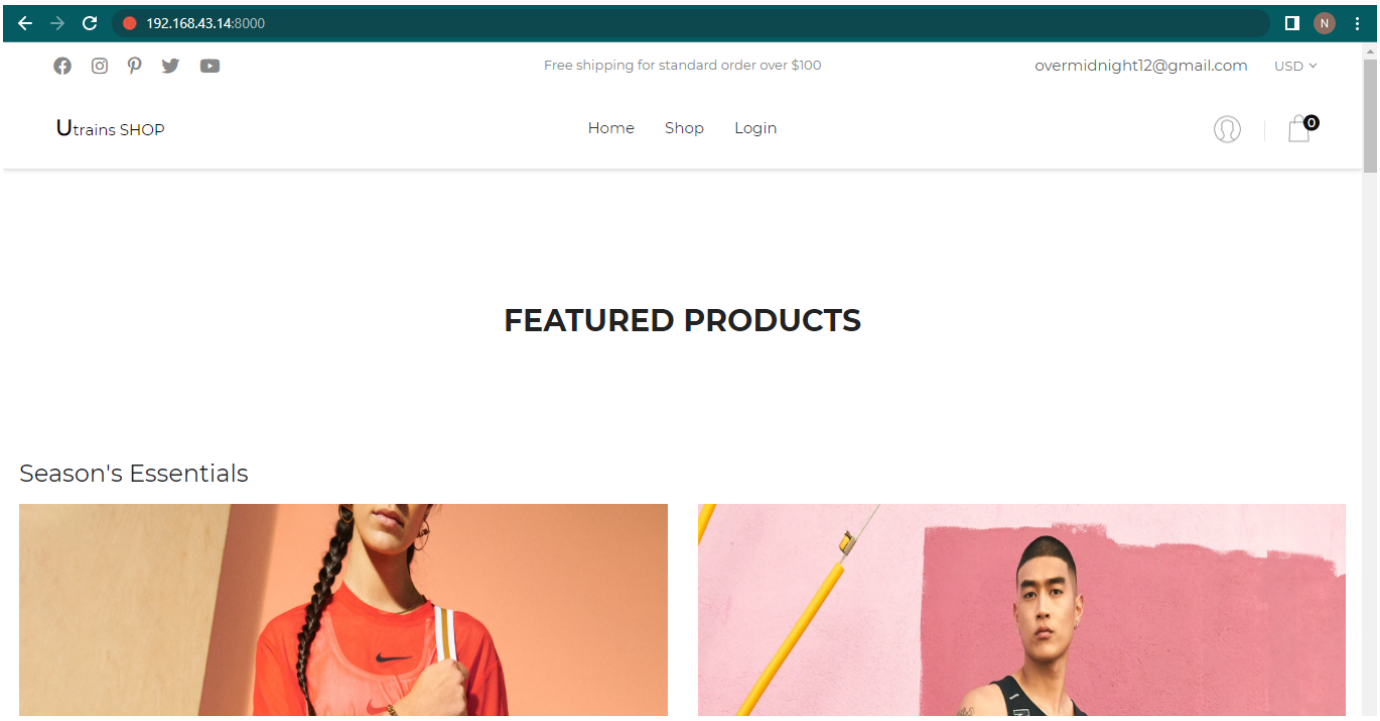
now run the command below to collect all the static folders and files and also the migrations into the database.

```
# Web Server (Virtual Environment)
(venv) root@website-server:~/eccommerce# python manage.py collectstatic
(venv) root@website-server:~/eccommerce# python manage.py makemigrations
(venv) root@website-server:~/eccommerce# python manage.py migrate
```

Now you can start the web server on port 8000 like in the development mode. As you can remember we had set up the firewall accordingly to be open port 8000.

```
# Web Server (Virtual Environment)
(venv) root@website-server:~/eccommerce# python manage.py runserver
0.0.0.0:8000
```

After entering the IP address and port in the address bar of the browser (http://192.168.43.14:8000/) the website should render properly. You can see for yourself by testing the functionality of the website.



The website was running using the dev server but for best practice we need to use a more reliable server. The dev server is vulnerable to attacks and it's built just for development purpose.

Installing Apache and WSGI

For production mode, the website needs a reliable web server. There are two main options for web development: nginx and Apache. We will use Apache and also install WSGI, which is an acronym for Web Server Gateway Interface and acts as an interface between Apache and Python Django. First let´s install both Apache and WSGI in our virtual environment.

```
# Web Server (Virtual Environment)
(venv) root@website-server:~/eccommerce# sudo apt-get --yes install
apache2
[...]
(venv) root@website-server:~/eccommerce# sudo apt-get --yes install
libapache2-mod-wsgi-py3
[...]
```

Let's now turn to the Apache server configuration in the Ubuntu configuration folder. Switching to the Apache subfolder sites-available we copy a default config file for our purposes and open it with a text editor.

```
# Web Server (Virtual Environment)
(venv) root@website-server:~/eccommerce# cd /etc/apache2/sites-
available/
[...]
(venv) root@website-server:/etc/apache2/sites-available# sudo cp 000-
default.conf ecommerce_site.conf
[...]
(venv) root@website-serverr:/etc/apache2/sites-available# sudo vi
ecommerce_site.conf
```

Now let us make some changes inside the newly created configuration file and add some data at the end of the config file (before the closing VirtualHost tag).

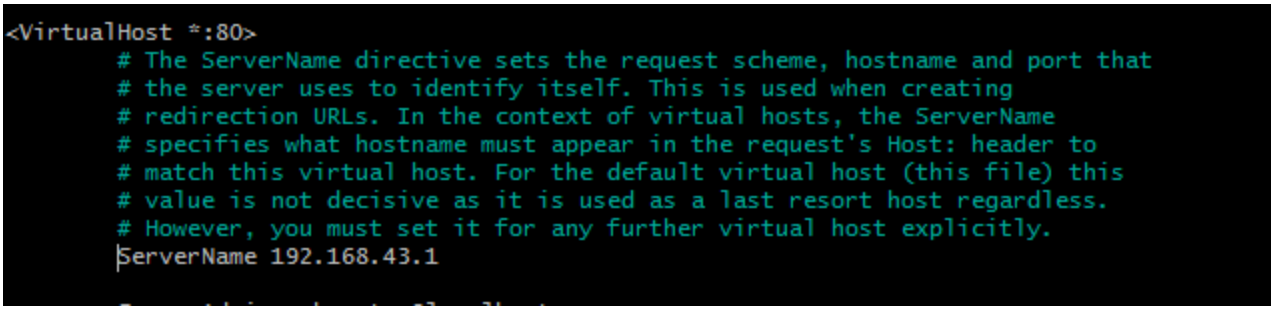
```
        Alias /static /root/eccommerce/static_root
        <Directory /root/eccommerce/static_root>
                Require all granted
        </Directory>

        Alias /media /root/eccommerce/media_root
        <Directory /root/eccommerce/media_root>
                Require all granted
        </Directory>

        <Directory /home/root/eccommerce/demo>
                <Files wsgi.py>
                        Require all granted
                </Files>
        </Directory>
        # Make sure to customize the following paths to the file
structure on your web server
        WSGIScriptAlias / /root/eccommerce/demo/wsgi.py

        #python-path: path to project / python-home: path to virtual
environment
        WSGIDaemonProcess app python-path=/root/eccommerce/ python-home=
/root/eccommerce/venv
    WSGIProcessGroup app
```

still in the config file, uncomment the server name and add your server ip address (192.168.43.1)



Back in the Virtual Environment home directory, we now enable the python site configuration file we just edited.Back in the Virtual Environment home directory, we now enable the python site configuration file we just edited and disable the default configuration file.

```
# Web Server (Virtual Environment)
(venv) root@website-server:~/eccommerce# cd ~
(venv) root@website-server:~# sudo a2ensite ecommerce_site.conf
[...]
(venv) root@website-server:~# sudo a2dissite 000-default.conf
[...]
```

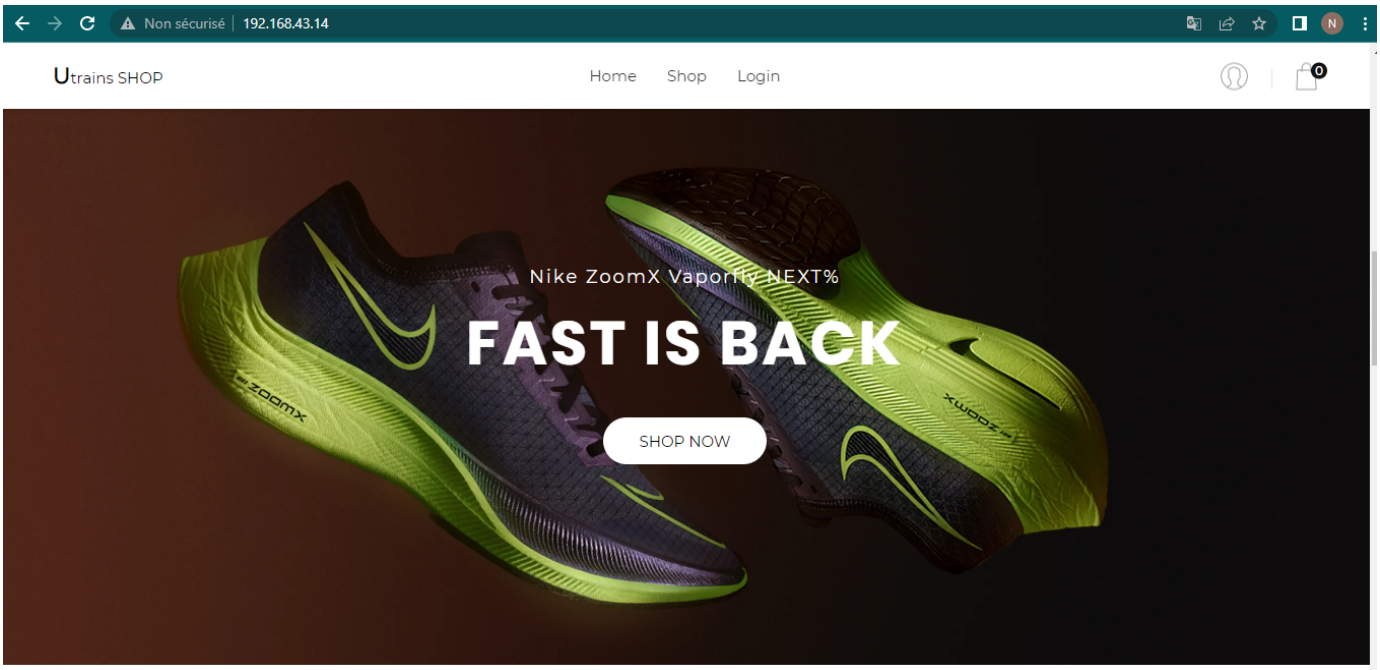
At the end of the Apache configuration we have to adjust some permissions for file folders and files.

```
# Web Server (Virtual Environment)
(venv) root@website-server:~# sudo chown :www-data eccommerce/
(venv) root@website-server:~# sudo chown :www-data eccommerce
/static_root
(venv) root@website-server:~# sudo chmod -R 775 eccommerce/static_root
(venv) root@website-server:~# sudo chown :www-data eccommerce/media_root
(venv) root@website-server:~# sudo chmod -R 775 eccommerce/media_root
```

Before we start the Apache server again, we close port 8000 in the firewall, which we opened earlier for testing purposes. We also allow http traffic.

```
# Web Server (Virtual Environment)
(venv)  root@website-server:~# sudo ufw delete allow 8000
[...]
(venv) root@website-server:~# sudo ufw allow http/tcp
[...]
(venv) root@website-server:~# sudo service apache2 restart
```

We did it! The website should now appear correctly after entering the IP (192.168.43.14) address in the address bar of the browser (now without specifying the port).

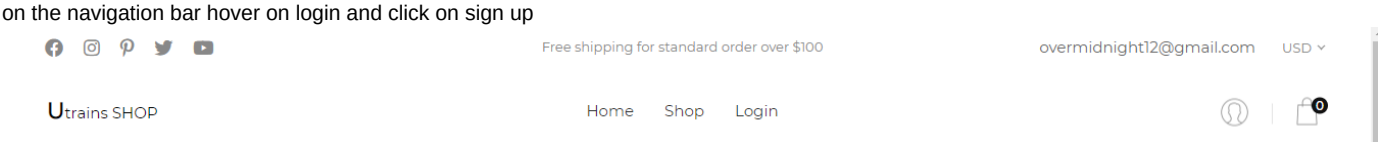


congratulations you just hosted a python base website on your dev server.

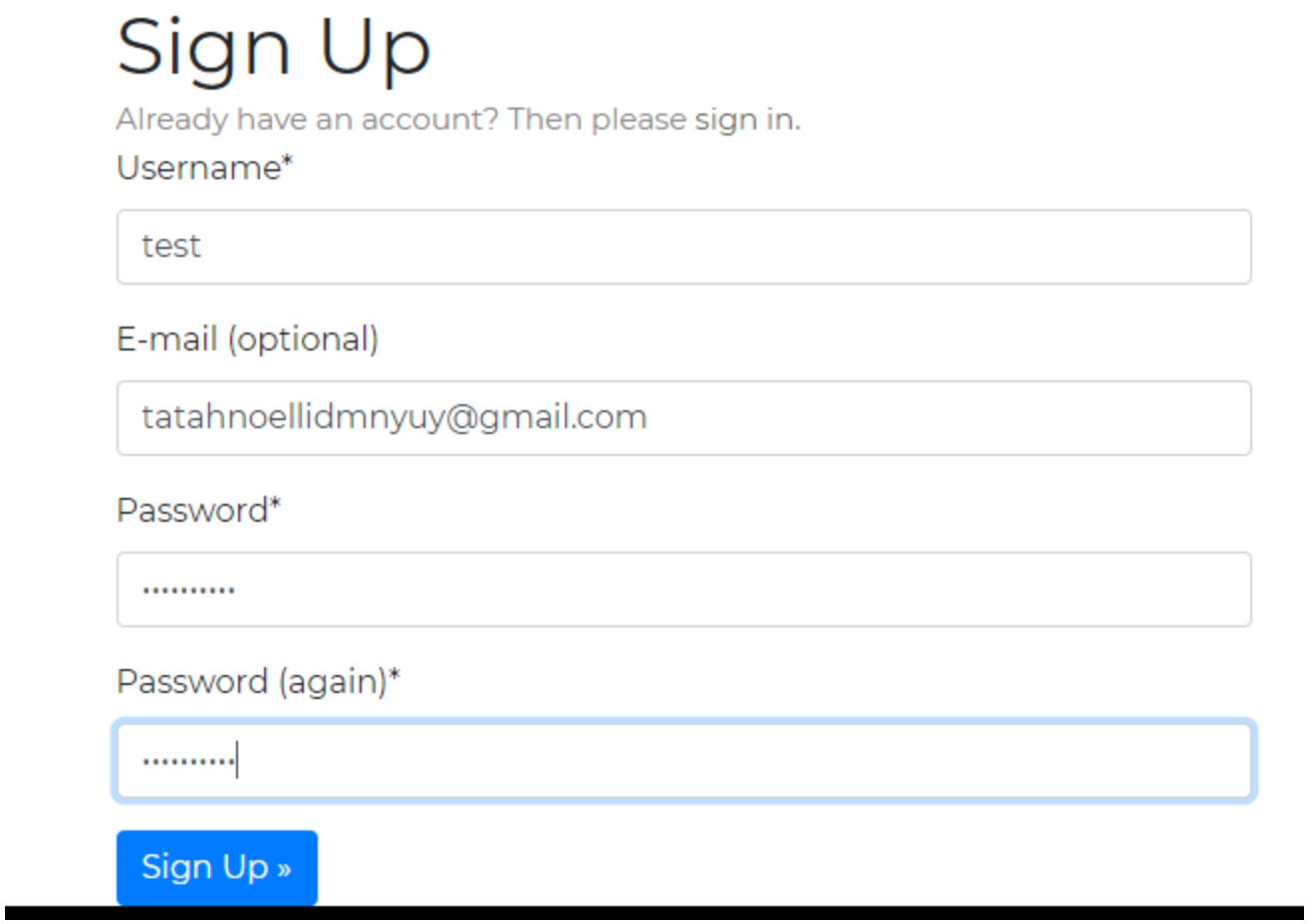
Creating a user and logging in

Now lets test our application by signing up a user and then logging in as the user.

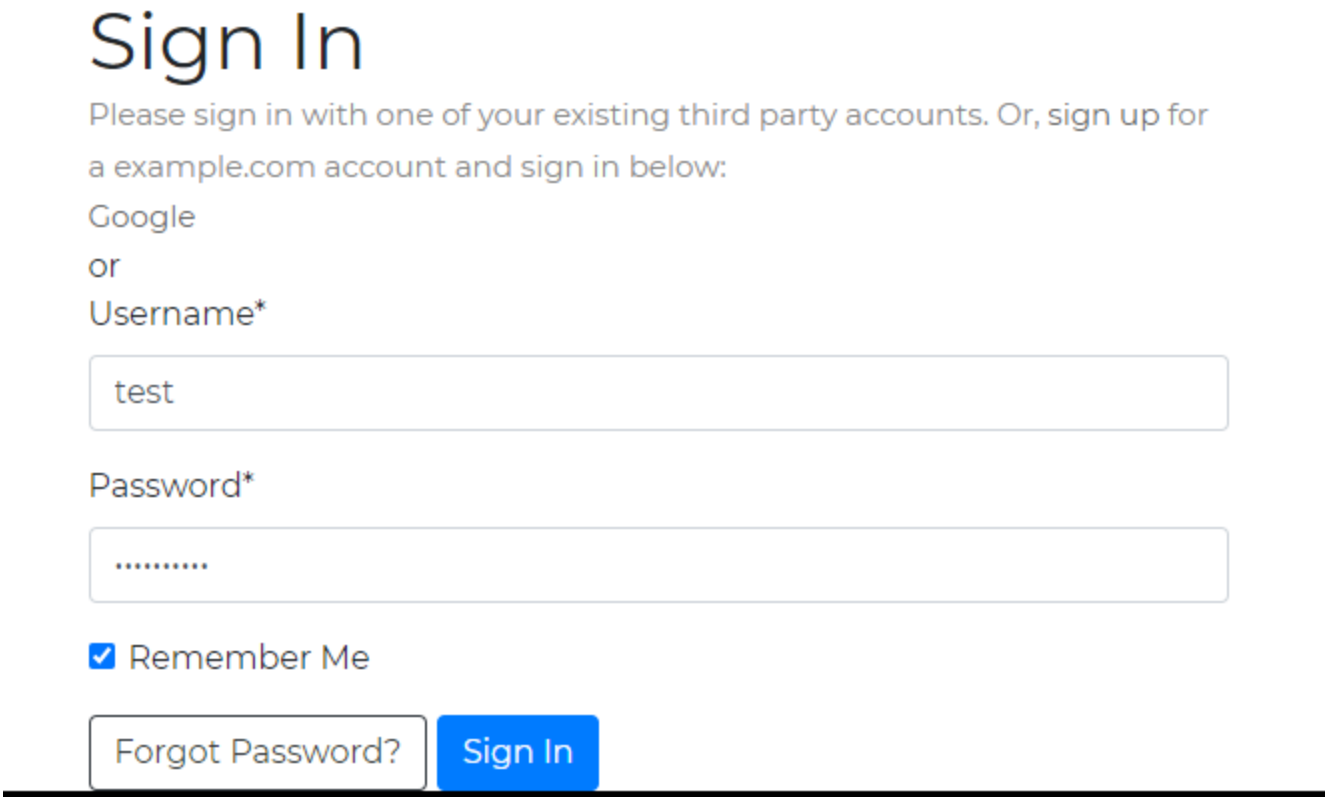
on the navigation bar hover on login and click on sign up



enter your information and click on sign up



go back to your home page and sign up



once more congratulations you just deployed your first application.