Django Complete Project Setup with AWs, Nginx and git

- *Narendra Allam, Copyright 2018.*

os requirements:

- **Ubuntu Desktop Linux 16.0.4/18.04**

Steps :

- **Open terminal**
- **Update 'sudo ' for latest system packages**

```
$ sudo apt-get update
```

make directory:

- **create a directory for storing all the project files in that directory**

```
$ mkdir dev

$ cd dev
```

installing pip:

- **install 'pip3' for installing python packages**

```
$ sudo apt-get install python3-pip
```

virtual environment:

- **it is a tool to create isolated python environments. It creates a folder which contains all the necessary packages that a python project would need.**

```
$ sudo apt-get install virtualenv
```

- **creating a folder with virtual environment**

```
$ virtualenv venv
```

- **here "venv" is the virtual environment folder name , the folder name is of as your wish**

- **activating the virtual environment**

  $ source venv/bin/activate

Mysql database setup:

$ sudo apt-get install python3-dev mysql-server libmysqlclient-dev

$ pip install mysql-python

$ pip install mysql-connector==2.1.6

login to mysql:

- **While installing mysql, copy and save the default password provided by the installer.**

  Installer prompts and show

- **you the default password. save it and alter root password as below**

  $ mysql -u root -p[root password]

- **Change the password**

  $ mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'MyNewPass';

- **Create a user**

  $ mysql> CREATE USER 'newuser'@'localhost' IDENTIFIED BY 'password';

- **Give privileges**

  $ mysql > GRANT ALL PRIVILEGES ON *.* TO 'newuser'@'localhost';

- **Create database**

  $ mysql > create database databasename;

Install django & restframework:

- **Install Django and Django REST framework into the virtualenv**

$ pip install django

$ pip install djangorestframework

Installing atom:

$ sudo apt-get install snapd

$ snap install atom --classic

Setting up a new project :

$ django-admin startproject projectname

Open the project with text editor:

$ atom projectname

- **now in atom you can see the project and app folders with files in it.**

Move to projects directory:

$ cd projectname

create an app:

$ python manage.py startapp appname

Make changes in settings.py file:

- **in atom go to settings.py file and change the database credentials and also under the installed apps give your app name and rest_framework**

```
DEBUG = True

ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'squares',
    'rest_framework',
]
```

```
# https://docs.djangoproject.com/en/1.11/ref/set

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME':  'count',
        'USER': 'ram',
        'PASSWORD':'Raghuram@9',
        'HOST':'localhost',
        'PORT':'3306',
    }
}
```

Create super user:

- **Super user is created so that the admin the authentication to login to the application page**
  $ python manage.py createsuperuser

- **While creating superuser it ask you to give username, mail, password**

Wrting models:

- **It contains the essential fields and behaviors of the data you're storing. Generally, each model maps to a single database table. Each model is a Python class that.subclasses django.db.models.Model**

```
from __future__ import unicode_literals

from django.db import models

class Square(models.Model):
    number = models.IntegerField()
    sqr = models.IntegerField()
```

- **After writing models next you have to see the changes and want to migrate them**

  $ python manage.py makemigrations
- **Now you can see the changes then migrate them**

  $ python manage.py migrate
- **After migrating the models check the database whether the tables are created or not**

Admin page:

- **This administrate Django groups and users, and all registered models in your app. This interface gives you the ability to Create, Read, Update, Delete operations on your registered models.**

```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals

from django.contrib import admin
from .models import Square
class SquareDetail(admin.ModelAdmin):
    list_display=('number','sqr')
admin.site.register(Square,SquareDetail)

# Register your models here.

```

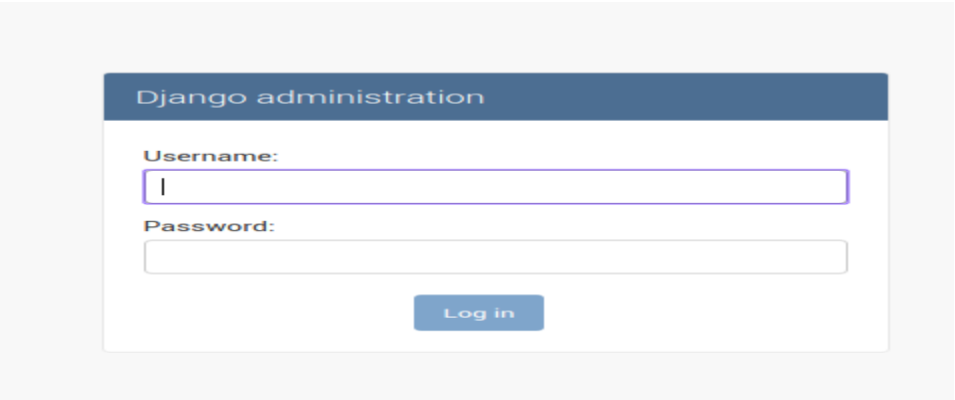
- **Now run the server.**

  $ python manage.py runserver

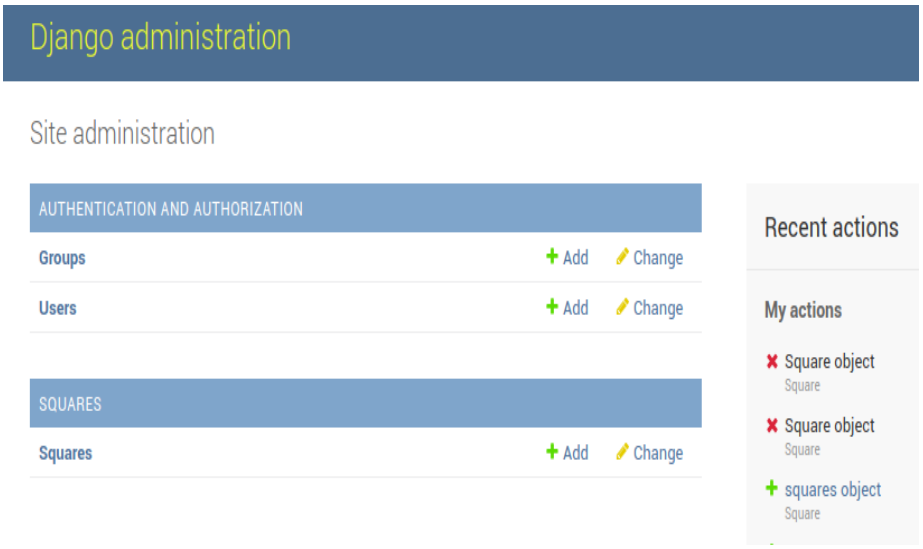
- **You will see an link in the terminal open it in browser.**

- **“it’s working” is displayed on the browser because your application is created successfully.**

- **Now add ‘/admin’ at the end of link then it will redirect to the admin page.**



- **Login to the django administration page.**



Writing views:

- **Your entire logic should be here.**

- **A view function, is simply a Python function that takes a Web request and returns a Web response. Return response can be the HTML contents of a Web page, or a 404 error, or an XML document, or an image . . . or anything.**

- **If you are using djangorestframework the response will be in json format.**

- **The below one is a simple api_view get function.**

```
from __future__ import unicode_literals
from django.contrib.messages.views import SuccessMessageMixin
from django.shortcuts import render
from rest_framework.decorators import api_view
from rest_framework.response import Response
from django.http import JsonResponse
from .models import Square
from .serializers import SquareSerializer
from rest_framework import status




@api_view(['GET'])
def getsquare(request,_number):

    cal = Square.objects.filter(number=_number)
    print (list(cal))
    if cal:
        serializer = SquareSerializer(cal,many=True)
        return Response(serializer.data)
```

Serializers:

- **Django's serialization framework provides a mechanism for translating Django models into other formats.**

```
from rest_framework import serializers
from .models import Square


class SquareSerializer(serializers.ModelSerializer):

    class Meta:
        model = Square
        fields = '__all__'
```

<u>Urls</u>:

- **Create urls.py file in the application folder.**
- **Map the application url with project url.**

Project urls.py

```
from django.conf.urls import url,include
from django.contrib import admin
from django.http import HttpResponse
from squares import urls

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^squares/',include('squares.urls')),
]
```

Application urls.py

```
from django.conf.urls import url,include
from django.http import HttpResponse
from .views import getsquare,postnum,cal_list


urlpatterns = [
    url(r'^calc/(?P<_number>\d+)',getsquare),
    url(r'^post/',postnum),
    url(r'^list/(?P<_number>\d+)',cal_list),


]
```

- **Now run the server and check it in the browser.**

<u>Requirements file</u>:

- **This file is to save the packages that you have used for this project.**
  $ pip freeze
- **Now you can see all the packages that you have used for this project like this.**

```
(venv) ubuntu@ip-172-31-45-254:~/Square$ pip freeze
Django==2.0.6
djangorestframework==3.8.2
mysqlclient==1.3.13
Pillow==5.2.0
pkg-resources==0.0.0
pytz==2018.5
```

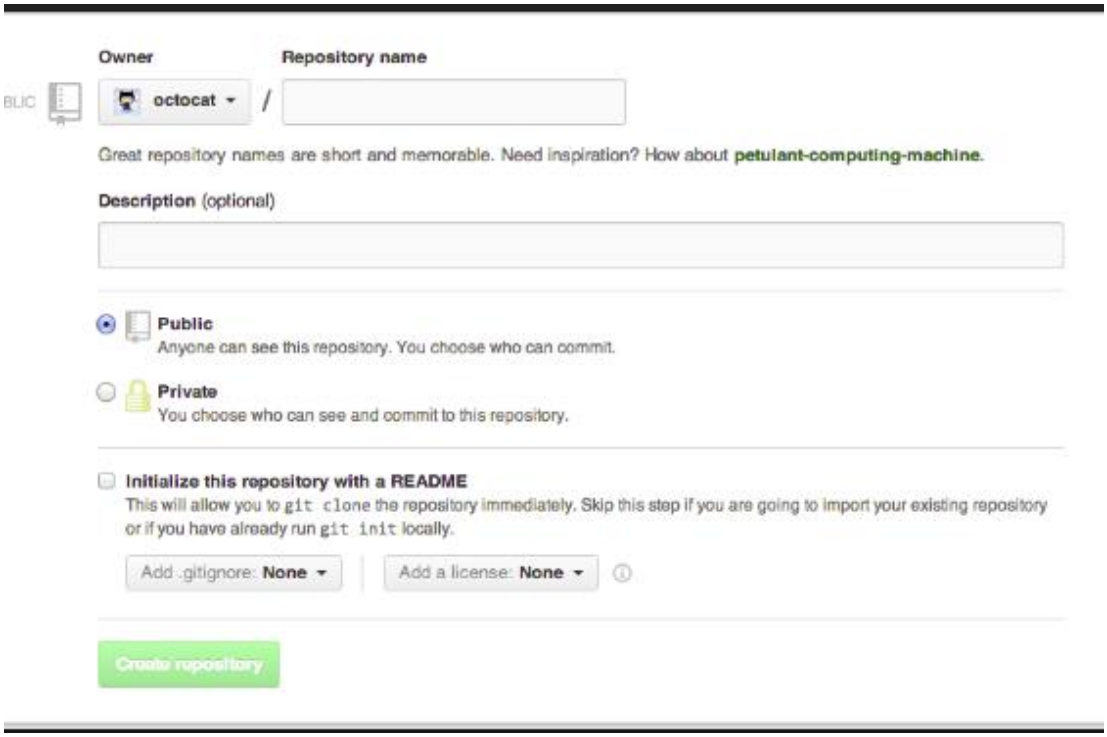
- **You can save it for further use by the following command.**

  $ pip freeze >requirements.txt

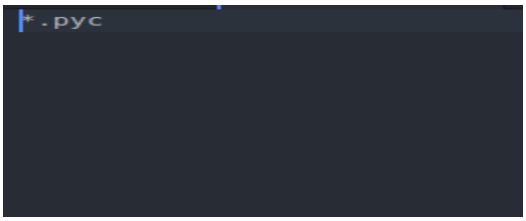
Pushing code to github:

- **if you don't have a github account , Create an account in github. Create a repository for storing and managing the folders.**

- **Create a repository to manage a project or folders at any time.**



- **Create a .gitignore file in your project folder to ignore the specific files and mention which files you want to ignore. In below screenshot we can see the example "*.pyc" it ignores all .pyc files in your project while commit to github.**



Then follow the below commands to push the projects to repository:

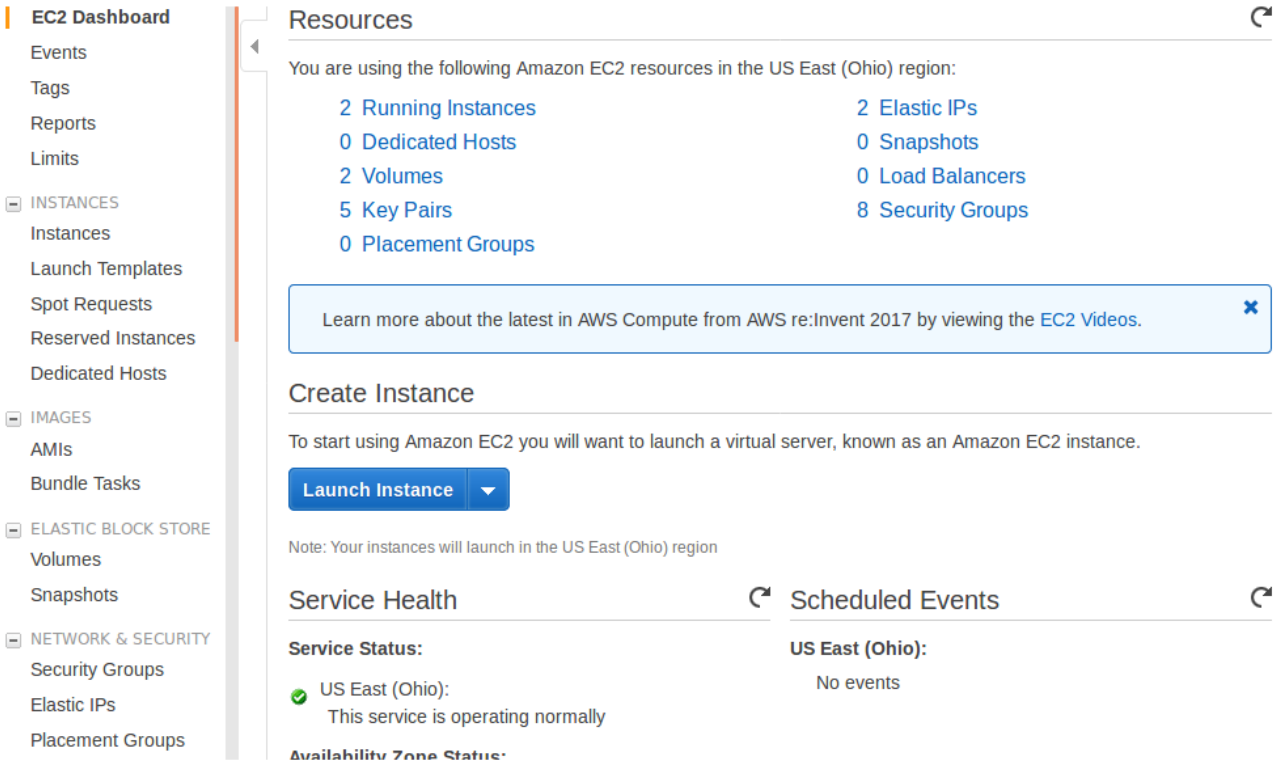
- **sudo apt-get install git**
- **git init.**
- **git add "your project name/".**
- **git status.**
- **git commit -m "message".**
- **git remote add origin "repository root".**
- **git push -u origin master.**

Aws Instance:

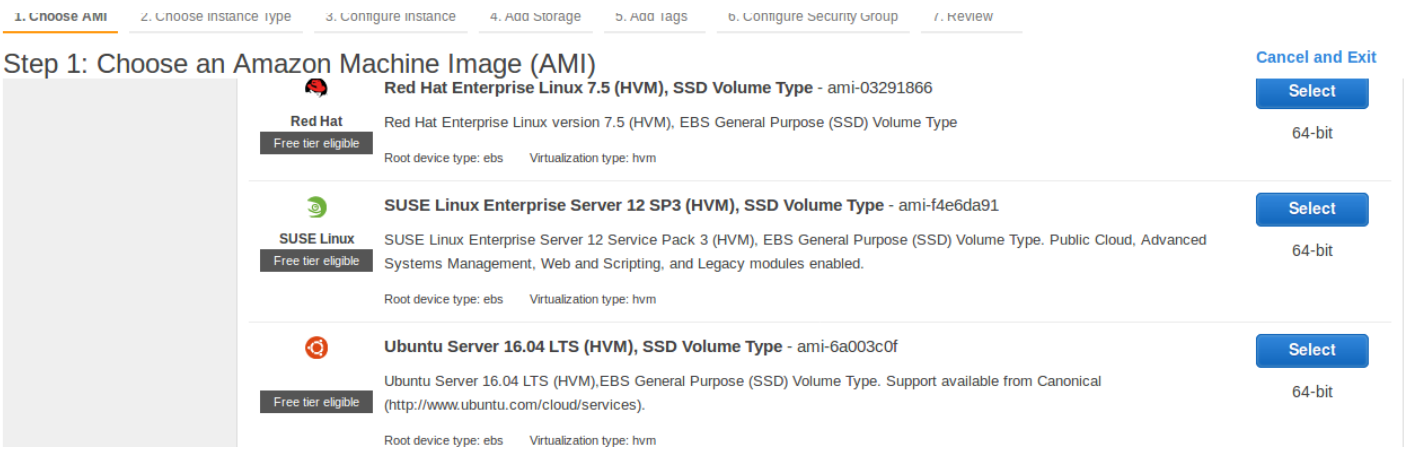
- **An EC2 instance is a virtual server in Amazon's Elastic Compute Cloud (EC2) for running applications on the AWS.**

How to launch a AWS Instance:

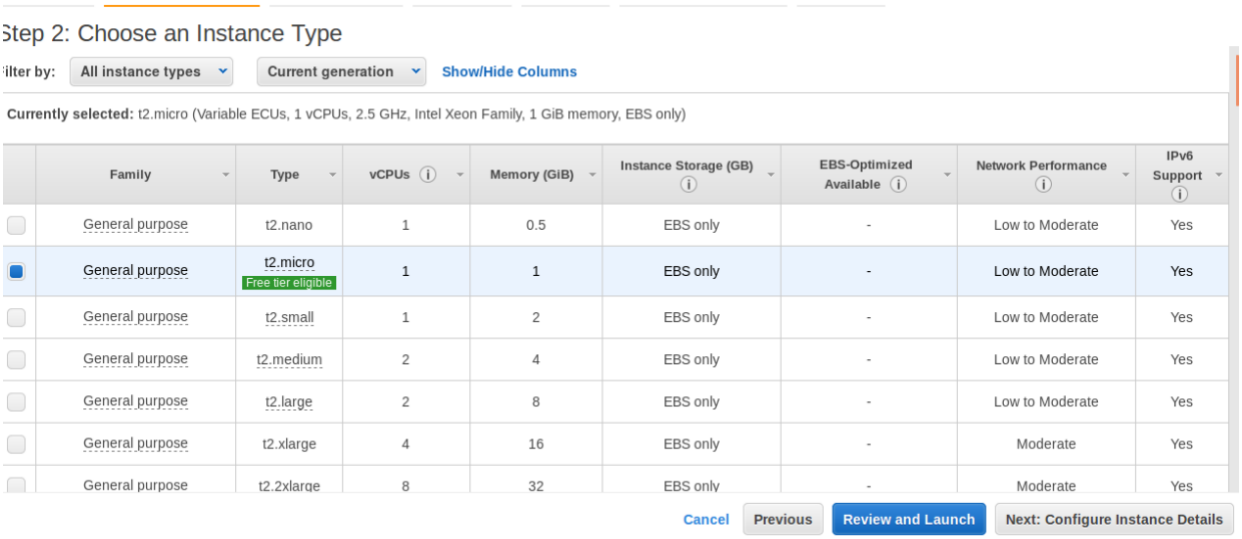
- **Create an account in aws.**
- **After going into console dashboard go to EC2**
- **Click on launch Instance to launch the instance.**



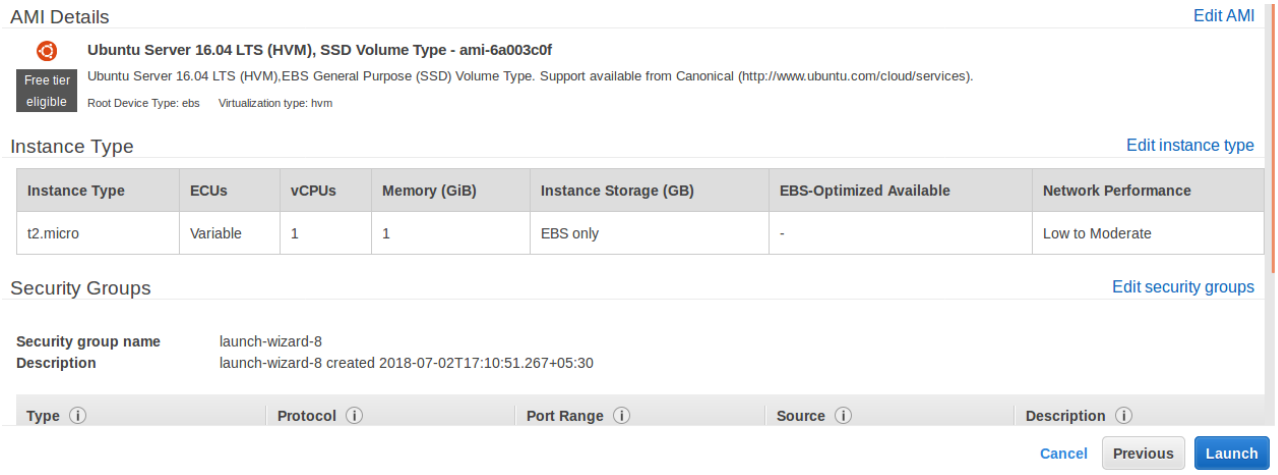
- **Select an ubuntu server 16.04 AMI in below screenshot.**



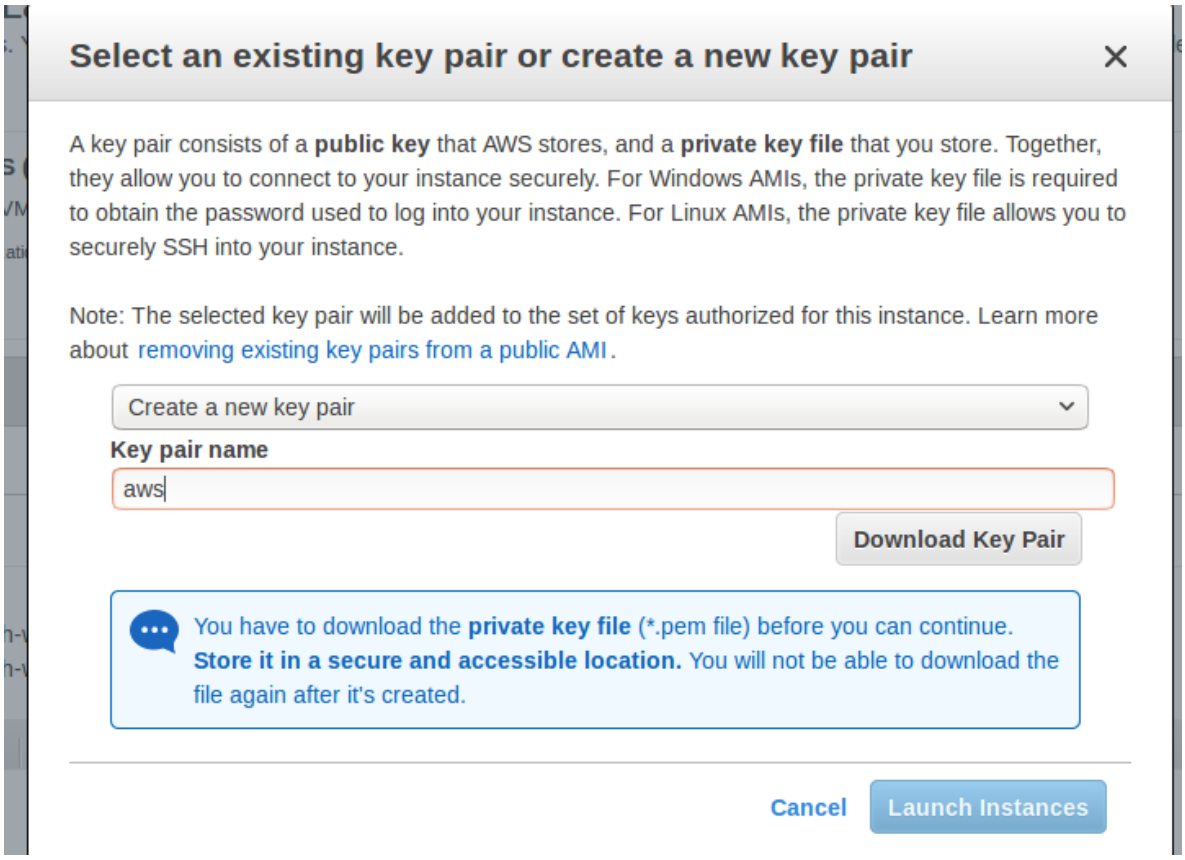
- **Select an instance type t2.micro and Click on review and launch .**



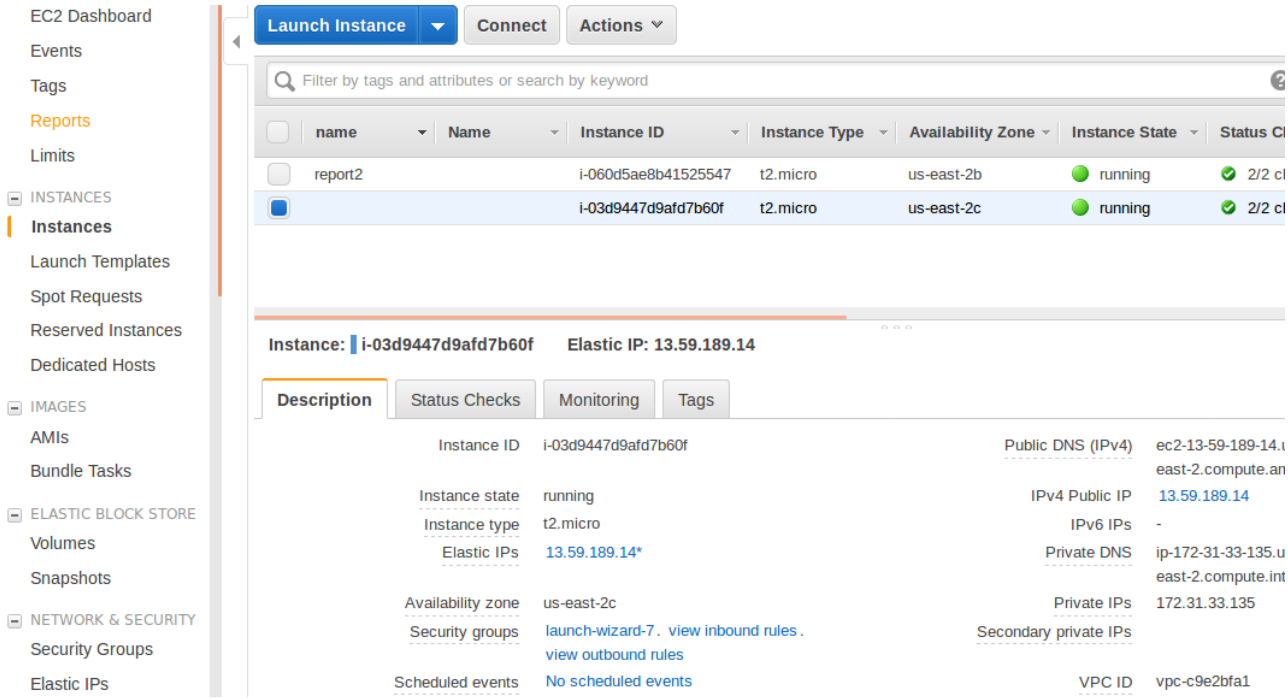
- **Click on launch.**



- **Select create a new key pair and give a name in key value pair , and click on**
  download key pair and click on launch to launch the instance.

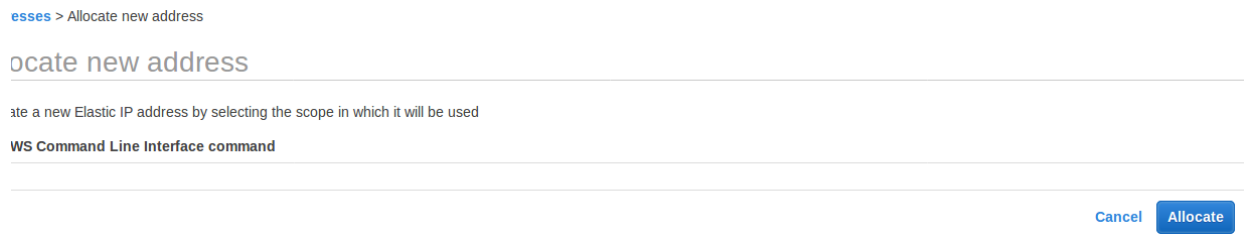


- **select a instance and click on Elastic ip in network&security to set the static ip.**

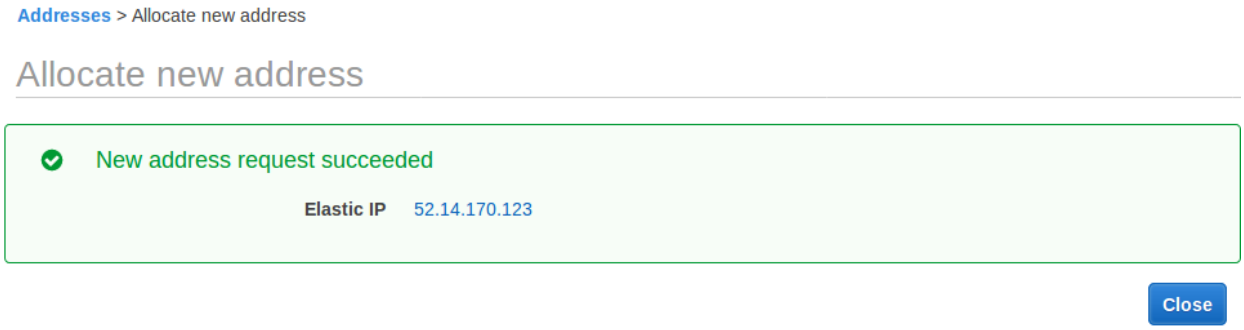


- **click on allocate new address.**

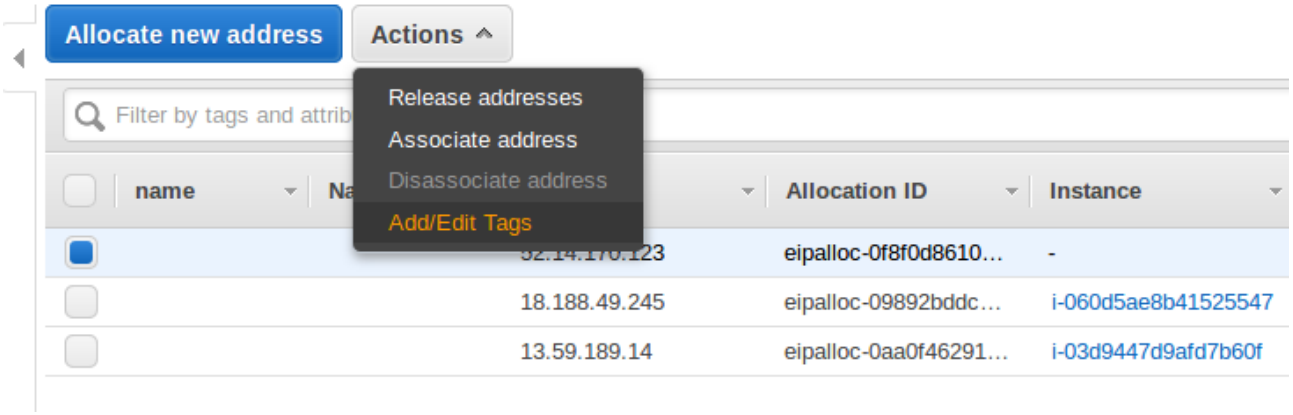
- **Click on allocate.**



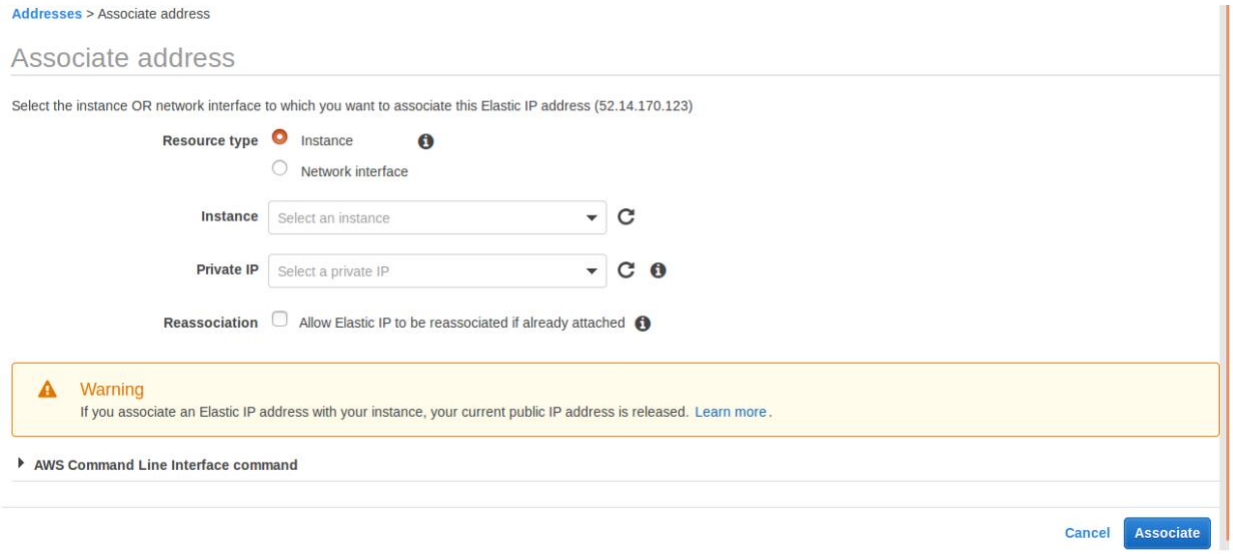
- **Now you can have the elastic ip address.**
- **Then you have to sync it to the instance.**



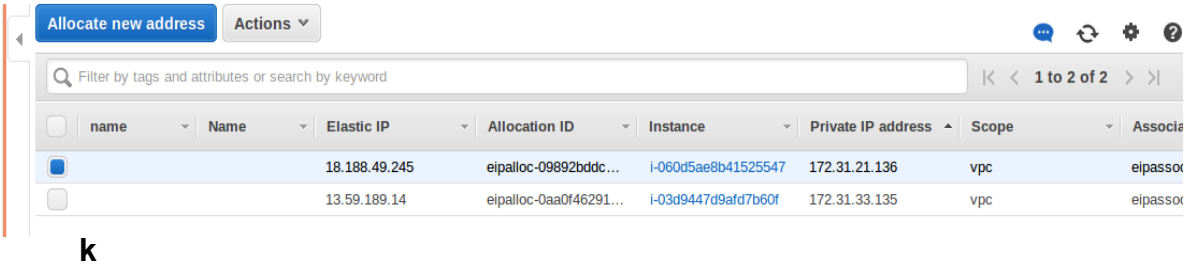
- **Select the ip address , click actions and select associate address.**



- **Select your running instance id and select private ip ,Click on associate.**

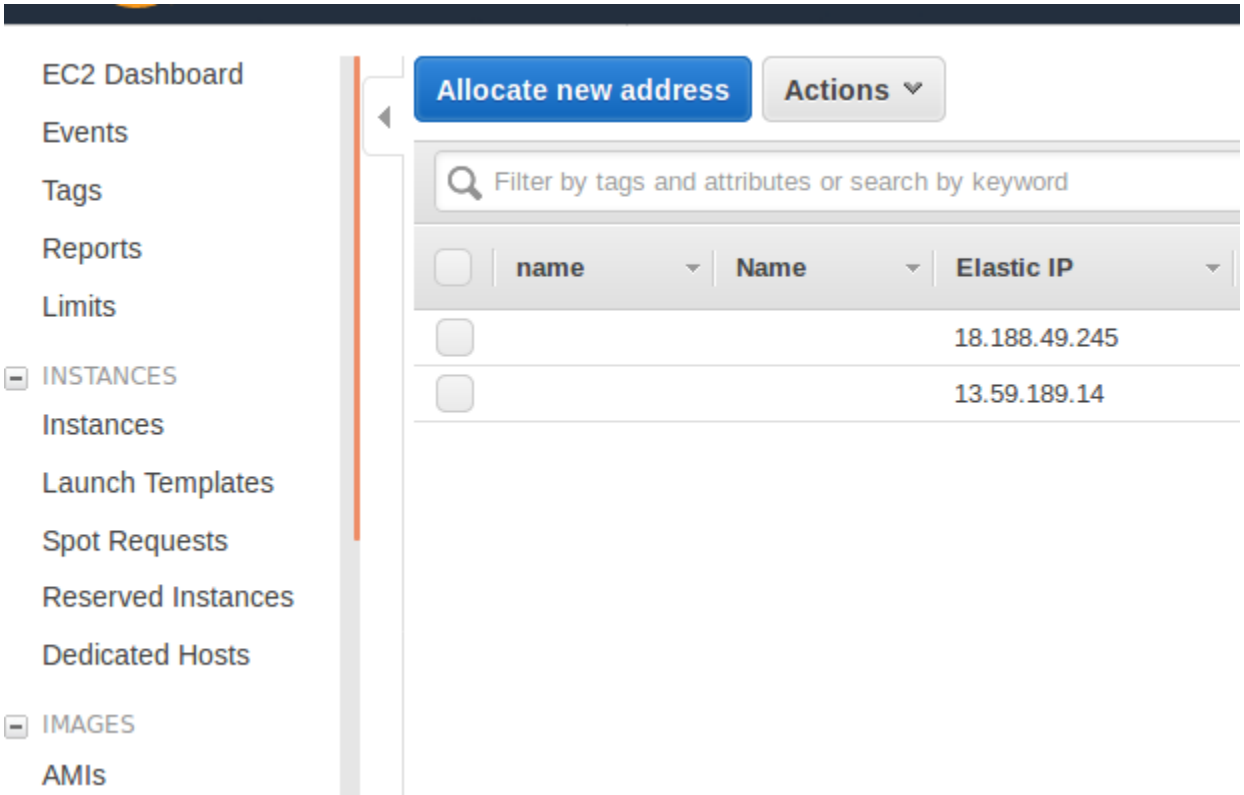


- **Check whether the Elastic ip address is associated with instance or not.**

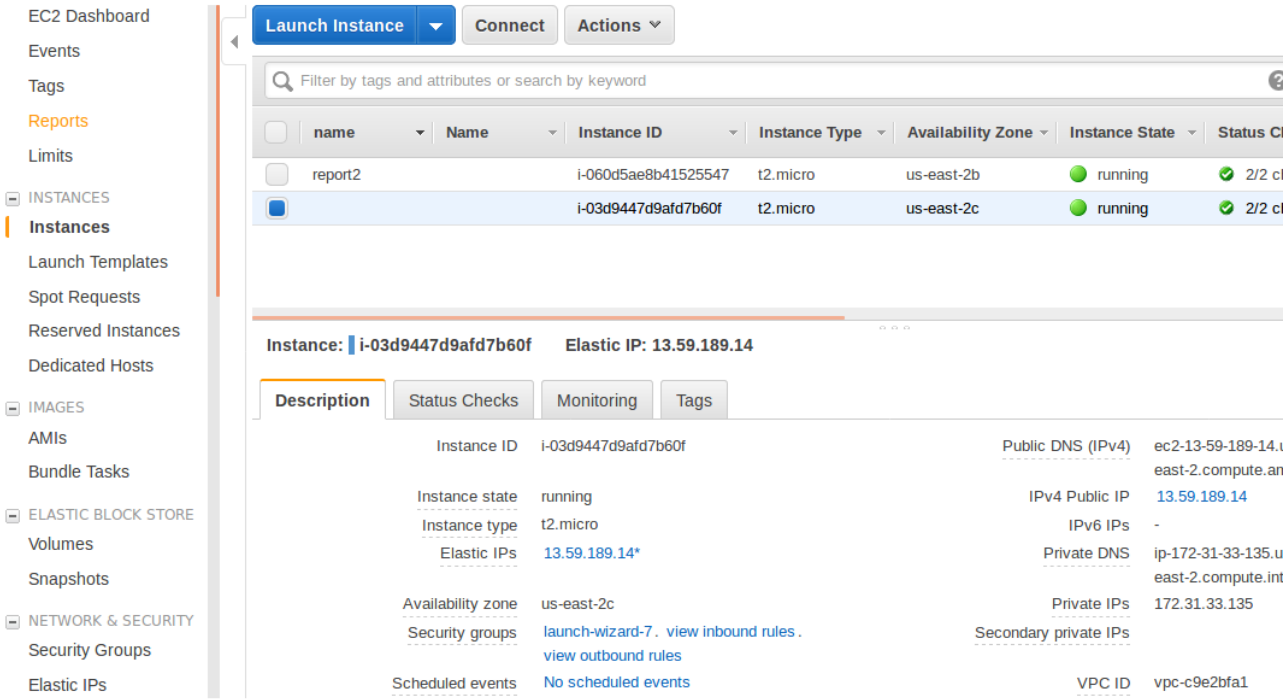


k

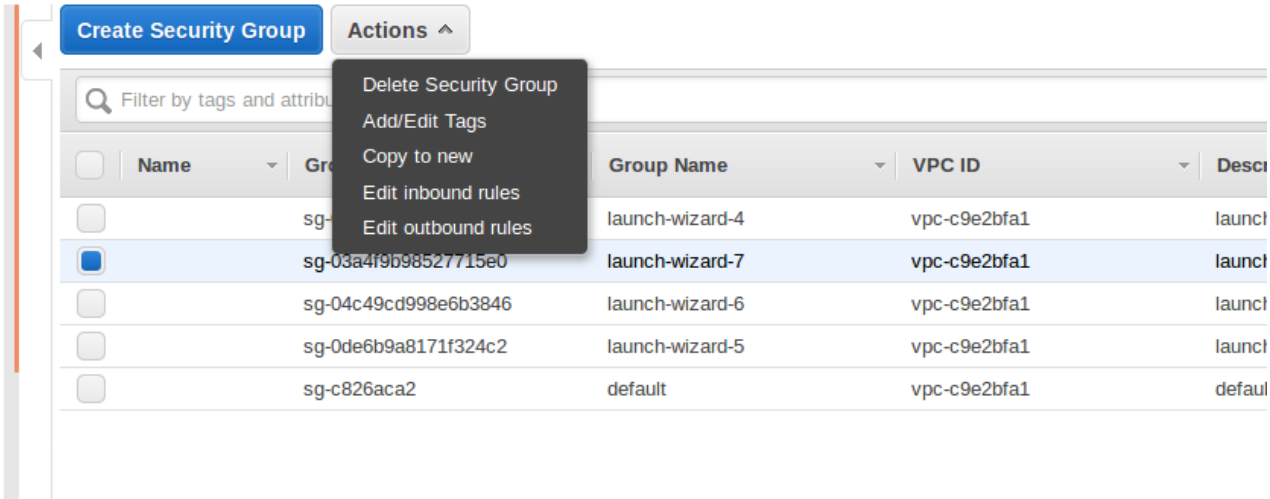
- **Click on instances.**



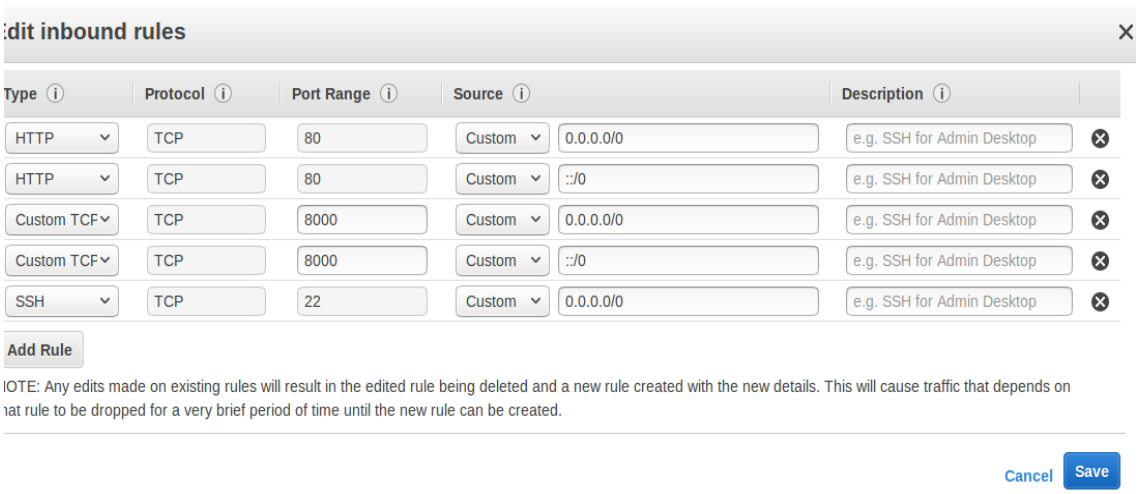
- **Select the instance and click on security groups.**



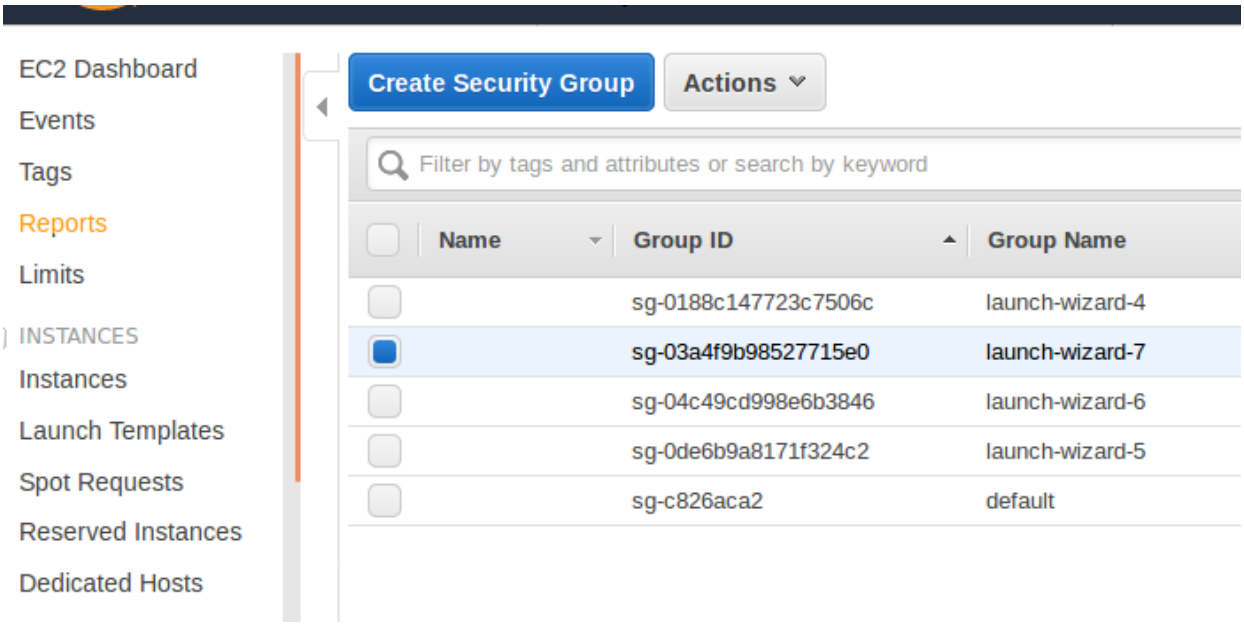
➢ **Select your instance group, Click on actions to edit inbound rules.**



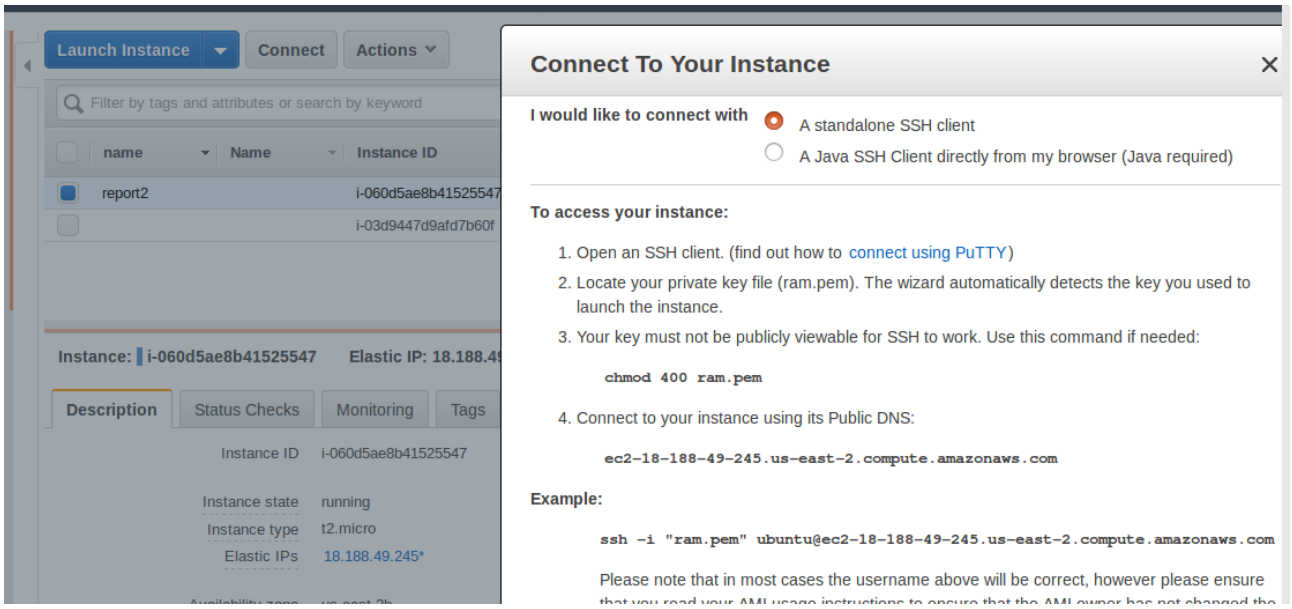
- **By default it has 22 port and remaining port ranges and types you have to set like below mentioned in the screen shot and click on save.**



- **Click on instances.**



- **Select the instance and click on connect.**



- **You can get the dialogue box like this.**
- **You can connect in your instance in your systems terminal.**

<u>Steps to connect in terminal</u>:

- **Open terminal and change the directory to where you downloaded the .pem file .**

  $cd downloads.

- **Copy and paste the below " chmod 400 ram.pem "in terminal.**

- **Next copy the line just below the example its like ssh**

```
hp@hp-HP-ProBook-4445s:~$ cd Downloads
hp@hp-HP-ProBook-4445s:~/Downloads$ chmod 400 ram.pem
hp@hp-HP-ProBook-4445s:~/Downloads$ ssh -i "ram.pem" ubuntu@ec2-18-188-49-245.us
-east-2.compute.amazonaws.com

Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.4.0-1060-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

14 packages can be updated.
0 updates are security updates.


*** System restart required ***
Last login: Fri Jun 29 11:01:53 2018 from 14.142.116.211
```

- **Now you are inside the local Ubuntu server**
- **Here once again you have to update sudo**
- **Here again you have install pip3,mysql,virtualenv,python3-dev**
- **Activate your virtualenv**
- **You have to clone the project folder from github now**

  $ git clone [your repository root]

- **If you have multiple folders in your repository, you have clone only the folder that you wants**

Cloning a particular folder from git:

$ git clone --depth 1 [repository root] [name of the folder]

$ cd [folder name]

$ git filter-branch --prune-empty --subdirectory-filter [folder name] HEAD

- **NOTE: No need of square brackets.**

- **Here again you have to do Mysql database setup and login to mysql shell.**

- **Create a database and come out of mysql shell.**

- **Install requirements.txt here.**

  $ pip install –r requirements.txt

Edit files:

- **Here you cannot install atom so you can edit the files in terminal.**
  $ vi [filename]
- **After opening the file with vi editor press i to make changes after making changes press 'esc' and ":wq" to save the file.**

- **Here you have to make changes in settings.py file. In allowed hosts you have to give your ip address and 0.0.0.0 or '*' in single or double quotes. '*' means any one.**

- **if your database credentials are not same then change them in allowed hosts.**

- **Now you have to migrate.**

  $ python manage.py makemigrations.

  $ python manage.py migrate.

- **Now your database tables are created.**

- **Create superuser again and insert the data.**

- **Now you can run the server by command.**

  $ python manage.py runserver 0.0.0.0:8000

- **Now you can check the output in the browser with ip address. And also you can see it in your moblies also.**

Nginx setup:

What is Nginx?

- **Nginx is one of the most popular webserver for its load balancing, reverse proxy and security. It hosts some of the largest traffic sites on the internet.**

Installation Steps:

- **we have to update local package index , so that we can install recent packages.**

  $ sudo apt-get update

- **install nginx.**

  $ sudo apt-get install nginx

- **Adjusting the firewall: To configure our firewall to allow acess to the service.**

  $ sudo ufw app list

- **You should get a listing of the application profiles:**

  Output:
  Available applications:
  Nginx Full
  Nginx HTTP
  Nginx HTTPS
  OpenSSH

- **As you can see, there are three profiles available for Nginx:**

- **Nginx Full: This profile opens both port 80 (normal, unencrypted web traffic) and port 443 (TLS/SSL encrypted traffic)**
- **Nginx HTTP: This profile opens only port 80 (normal, unencrypted web traffic)**
- **Nginx HTTPS: This profile opens only port 443 (TLS/SSL encrypted traffic)**

- **You can enable this by typing:**

  $ sudo ufw allow 'Nginx HTTP'

- **you can verify the change by seeing the status.**

  $ sudo ufw status

- **you can see the HTTP traffic allowed.**

  Output:

```
Status: active

To                         Action      From
--                         ------      ----
OpenSSH                    ALLOW       Anywhere
Nginx HTTP                 ALLOW       Anywhere
OpenSSH (v6)               ALLOW       Anywhere (v6)
Nginx HTTP (v6)            ALLOW       Anywhere (v6)
```

- **In case status is inactive follow these steps:**

  $ sudo ufw enable

  $ sudo ufw status

- **Check the Nginx service is active or not.**

  $ systemctl status nginx

```
ubuntu@ip-172-31-33-135:~$ systemctl status nginx
● nginx.service - A high performance web server and a reverse proxy server
   Loaded: loaded (/lib/systemd/system/nginx.service; enabled; vendor preset: en
   Active: active (running) since Fri 2018-06-29 11:57:54 UTC; 3 days ago
 Main PID: 13564 (nginx)
    Tasks: 2
   Memory: 2.5M
      CPU: 182ms
   CGroup: /system.slice/nginx.service
           ├─13564 nginx: master process /usr/sbin/nginx -g daemon on; master_pr
           └─13565 nginx: worker process
```

- **When you have your server's IP address or domain, enter it into your browser's address bar:**

  http://server_domain_or_IP

- **You should see the default Nginx landing page, which should look something like this:**

:

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org. Commercial support is available at nginx.com.

Thank you for using nginx.

- **This page is simply included with Nginx to show you that the server is running correctly.**

Uwsgi:

- **UWSGI: it is an application server which can communicate with applications through interface called wsgi.**

UWSGI Installion Steps:

- **install uwsgi by using pip**

  $ sudo -H pip install uwsgi

- **For instance, we can tell it to serve our first project by typing:**

$ uwsgi --http :8080 --home /home/ubuntu/venv --chdir /home/ubuntu/projectname -w projectname.wsgi

- **create a directory to store configure files.**

$ sudo mkdir -p /etc/uwsgi/sites

- **create a file for your project and open in a text editor.**

$ sudo nano /etc/uwsgi/sites/firstsite.ini

```
[uwsgi]
project = countsquares
uid = ubuntu
base = /home/%(uid)

chdir = %(base)/%(project)
home = %(base)/ubuntu/%(project)
module = %(project).wsgi:application

master = true
processes = 5

socket = /run/uwsgi/%(project).sock
chown-socket = %(uid):www-data
chmod-socket = 660
vacuum = true
```

- **in above screenshot you type mention project = “your projectname” , uid = “your system id”**

- **Create a systemd file to automate the uwsgi.**

$ sudo nano /etc/systemd/system/uwsgi.service

```
[Unit]
Description=uWSGI Emperor service

[Service]
ExecStartPre=/bin/bash -c 'mkdir -p /run/uwsgi; chown ubuntu:www-data /run/uwsgi'
ExecStart=/usr/local/bin/uwsgi --emperor /etc/uwsgi/sites
Restart=always
KillSignal=SIGQUIT
Type=notify
NotifyAccess=all

[Install]
WantedBy=multi-user.target
```

- **Create a server block configuartion file where our project can access.**

```
server {
    listen 80;
    server_name 13.59.189.14;

    location = /favicon.ico { access_log off; log_not_found off; }
    location /static/ {
        root /home/ubuntu/countsquares;
    }

    location / {
        include         uwsgi_params;
        uwsgi_pass      unix:/run/uwsgi/countsquares.sock;
    }
}
```

- **link your configuration file to Nginx sites-enabled directory to enable them.**

  $ sudo ln -s /etc/nginx/sites-available/projectname /etc/nginx/sites-enabled

- **Check the configuration by tying.**

  $ sudo nginx -t

```
ubuntu@ip-172-31-33-135:~$ sudo nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
ubuntu@ip-172-31-33-135:~$
```

- **if no errors are detected, you restart your nginx service to load the new configuration.**

  $ sudo systemctl restart nginx

- **start uwsgi**

  $ sudo systemctl start uwsgi

- **Let's delete the UFW rule to port 8000 and instead allow access to our Nginx server.**

  $ sudo ufw delete 'allow 8000'

  $ sudo ufw allow 'Nginx Full'

- **you can enable both of the services to start automatically at boot by typing.**

  $ sudo systemctl enable nginx

  $ sudo systemctl enable uwsgi

- **Now you can see the output in browser as nginx is running your server.**

Refer the below links if you have any doubts in nginx setup:

1) **https://www.digitalocean.com/community/tutorials/how-to-serve-django-applications-with-uwsgi-and-nginx-on-ubuntu-16-04**

2) **https://www.digitalocean.com/community/tutorials/how-to-install-nginx-on-ubuntu-16-04**