ADNAN MENDERES UNIVERSITY

CSE423-Cloud Computing

WORD COUNTER

Ayşe AKIŞIK - 171805007

Ayşen ALPASLAN - 171805060

Betül Berna SOYLU - 171805019

Saliha APAK - 171805022

FINAL PROJECT

- WORD COUNTER WITH DJANGO –

Project Description:

Running a simple web interface on the cloud platform with use of web application framework (django etc.) on a cloud platform to perform simple word count job.

What is the Django?

Django is a high-level Python web framework that enables rapid development of secure and maintainable websites. It is free and open source, has a thriving and active community, great documentation, and many options for free and paid-for support.

Summary:

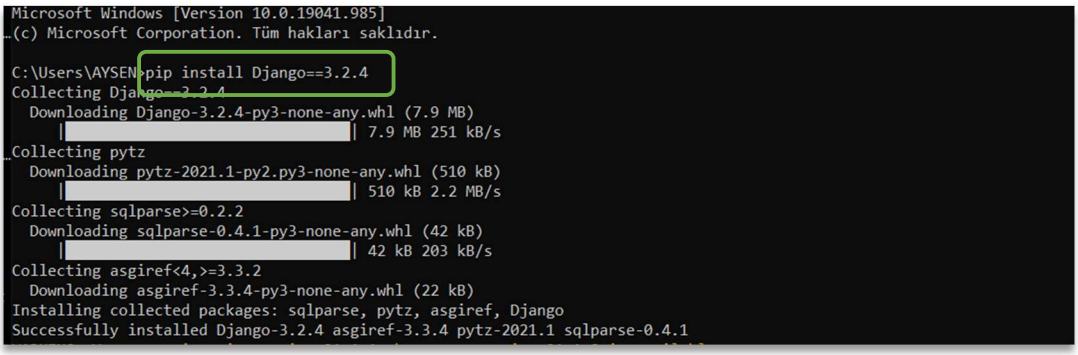
1- Django Installation

- Run the cmd

  Windows + cmd + enter

- Install the Django with pip [1]

  C:\Users\AYSEN>**pip install Django==3.2.4**



Picture1 – Django install command

2- Create a Django Project

- Open a folder called ‘workwork (changeable)’ on your desktop where you will collect your projects.

- Create your project folder named ‘work-counter-project (changeable)’ inside the workwork folder.

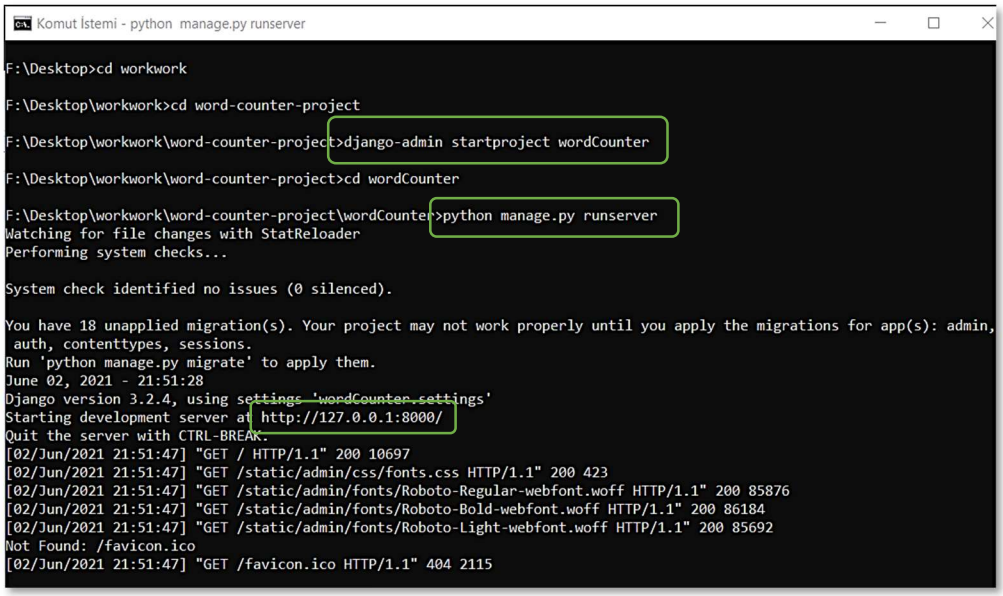
- Run the cmd
  Windows + cmd + enter

- Go inside the work-counter-project folder.
  C:\Users\AYSEN>**cd /d F:/Desktop**
  F:\Desktop>**cd workwork**
  F:\Desktop\workwork>**cd work-counter-project**

- Create a Django project named wordCounter.
  F:\Desktop\workwork\work-counter-project>**django-admin startproject wordCounter**

- Get into the project and run it.
  F:\Desktop\workwork\work-counter-project>**cd wordCounter**
  F:\Desktop\workwork\work-counter-project\wordCounter>**python manage.py runserver**

- After the project is running, you can view your project by going to the address shown in the command prompt.



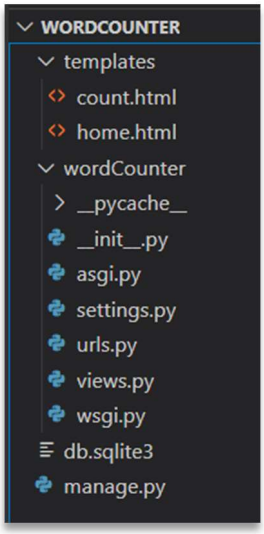
Picture2 - project build and run commands

3- Creating Project Frontend Files

Since we organize our project with Visual Studio Code, we run that program first. Then, as you can see in picture 3, we open the templates folder and create the count.html and home.html files in it.

home.html → The file containing the design of the page where we will enter the text.

count.html → The file containing the design of the page where the results are displayed after the counting process is completed.



Picture3 - view of project files

4- Url Redirects and Render Operations

- We edit the url.py file as shown in picture4. We can render home.html and count.html pages created with these arrangements.

```
urls.py
wordCounter > urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('', views.home, name='home'),
    path('count/', views.count, name='count'),
]
```

Picture4 - view of the url.py file

5- Coding of Project

- The operations that will take place while rendering the pages are written to the views.py file. Picture 5 shows the functions imported to the views.py file.

```
views.py
wordCounter > views.py
from django.http import HttpResponse
from django.shortcuts import render
import operator
import string
import re
```

Picture5 – imports on views.py

- In Picture 6, the rendering process of the home page is shown. In the rendering of the home page, only the home.html page is rendered.

```python
def home(request):
    return render(request, 'home.html')
```

Picture6 – render function for home page

- home.html page codes are as seen in picture 7. The actions that will take place when the form is submitted are the count function on the views.py page. We perform the action with the code block in picture 8.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Word Counter</title>
    <link href="https://fonts.googleapis.com/css?family=Lato" rel="stylesheet">
    <style>···
    </style>
</head>
<body>
    <!-- NAVBAR -->
    <div class="navbar">
        <div class="left"></div>
        <div class="right">
            <p class="title">WORD COUNTER</p>
        </div>
    </div>
    <!-- TEXTAREA -->
    <div class="container">
        <div class="flex-container">
            <div class="box left">
                <form action="{% url 'count' %}">
                    <textarea id="input" placeholder="Paste your text here" name="fullText"></textarea>
                    <br />
                    <input type="submit" id="count" value="Count!" />
                </form>
            </div>
        </div>
    </div>
    <!-- FOOTER -->
    <footer>
        <p>Ayşe Akışık - Ayşen Alpaslan - Betül Berna Soylu - Saliha Apak</p>
    </footer>
</body>
</html>
```

Picure7 – home.html page codes

```html
<form action="{% url 'count' %}">
    <textarea id="input" placeholder="Paste your text here" name="fullText"></textarea>
    <br />
    <input type="submit" id="count" value="Count!" />
</form>
```

Picture8 – code blocks of count action

- After pressing the Count button, the 'fullText' value received from the textarea is sent to the count operation as a request. The function is shown in Picture 9. The following operations are performed in order:

    - The sentences in the ‘fullText’ value are counted. Regex structure is used for this.
    - For fullText to be case sensitive, all letters are lowercase and saved as ‘lowerFullText’.
    - Punctuation in ‘lowerFullText’ is identified and removed. It is saved as ‘newFullText’.
    - The split function is used to determine the words in the ‘newFullText’ value. Each split word is saved in a list called 'wordlist'.
    - A dictionary called 'wordDictionary' opens to record the number of words in the list.
    - With the loop, every word in the ‘wordlist’ is checked. If the word exists in the ‘wordDictionary’, its value is increased by one, otherwise it is added to the dictionary.
    - The resulting dictionary is sorted by count values and saved as 'sort_wordDictionary'.
    - Finally, the ‘count.html’ page is rendered with the values to be used.

```
def count(request):
    fullText = request.GET['fullText']
    sentenceslist = re.split(r'[.!?]+', fullText)

    lowerFullText = fullText.lower()
    newFullText = ""

    for i in lowerFullText:
        if i not in string.punctuation:
            newFullText += i

    wordlist = newFullText.split()

    wordDictionary = {}

    for word in wordlist:
        if word in wordDictionary:
            #Increase
            wordDictionary[word] += 1
        else:
            #Add to the dictionary
            wordDictionary[word] = 1
    sort_wordDictionary = sorted(wordDictionary.items(), key=lambda x: (x[1], x[0]), reverse=True)
    return render(request, 'count.html',
        {'fullText':fullText, 'count':len(wordlist), 'wordDictionary': sort_wordDictionary, 'sentencesCount':len(sentenceslist)-1})
```

Picture9 – codes of count function

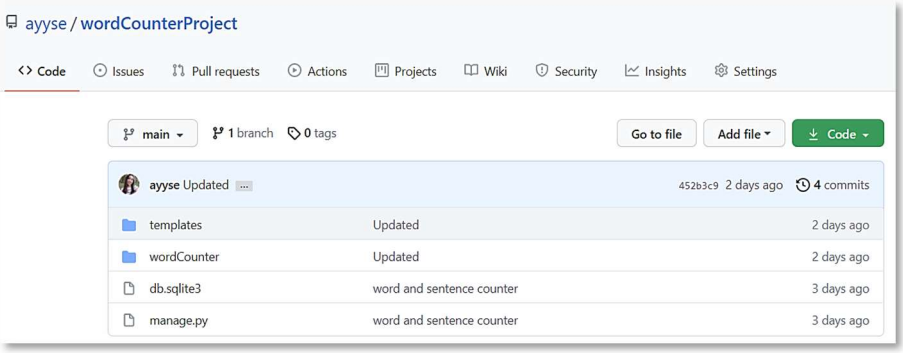
- The design of the rendered count.html page is shown in picture10.

```
<body>
    <div class="navbar">
        <div class="left"></div>
        <div class="right">
            <p class="title">WORD COUNTER</p>
        </div>
    </div>

    <div class="container">
        <div class="target-container">
            <p class="target">There are <span>{{ count }}</span> words in your text.</p>
            <p class="target">There are <span>{{ sentencesCount }}</span> sentences in your text.</p>
        </div>
        <div class="flex-container">
            <div class="box left">
                <p id="fullText">{{ fullText }} </p>
                <br />
                <a href="{% url 'home' %}">
                    <span class="background"></span>
                    <span>COUNT AGAİN!</span>
                </a>
            </div>
            <div class="box">
                {% for word, counttotal in wordDictionary %}
                <p><span>{{ counttotal }}</span>  {{ word }}</p>
                {% endfor %}
            </div>
        </div>
    </div>

    <!-- FOOTER -->
    <footer>
        <p>Ayşe Akışık - Ayşen Alpaslan - Betül Berna Soylu - Saliha Apak</p>
    </footer>
</body>
```

Picture10 – design of count.html

6- Sharing on Github (with GitHub Desktop)

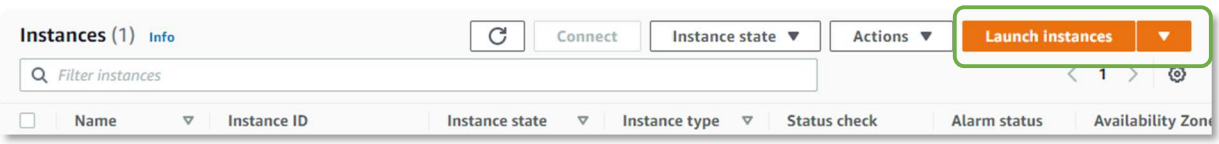
Open GitHub Desktop and click File > New Repository and enter repo informations. Then, push changes.



Picture11 - github image of the project

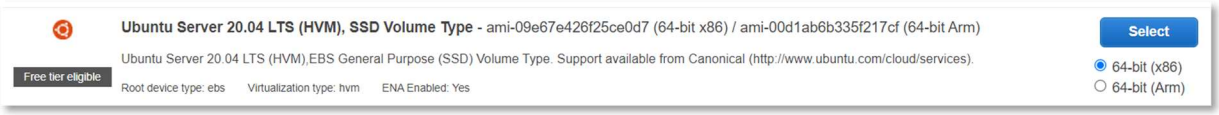
7- Launch Instance [2]

- From EC2 Dashboard, click on the launch instance button to open the wizard.



Picture12 – launch instance

- Select free tier eligible Ubuntu Server 20.04 LTS (HVM), SSD Volume Type.

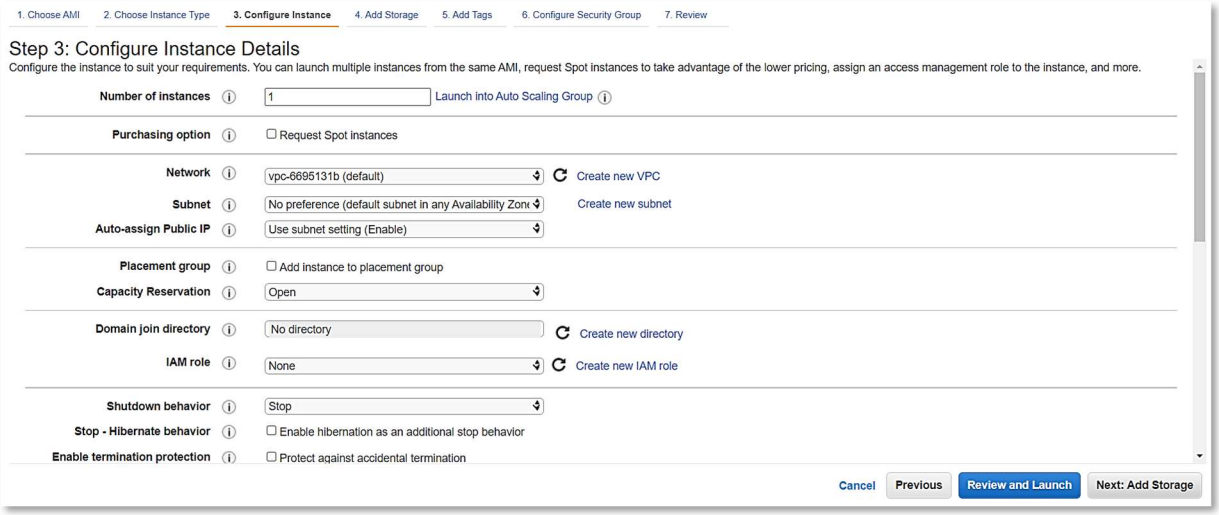


Picture13 – select ami

- Choose the instance type as shown below.

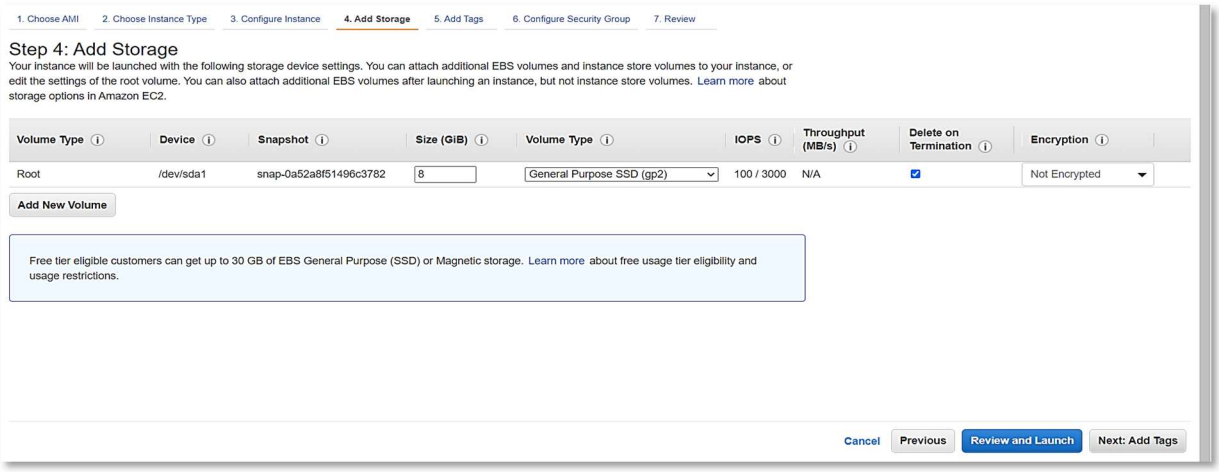
Picture14 - select instance type

- Configure the instance. Then click the "Next:Add Storage" button.



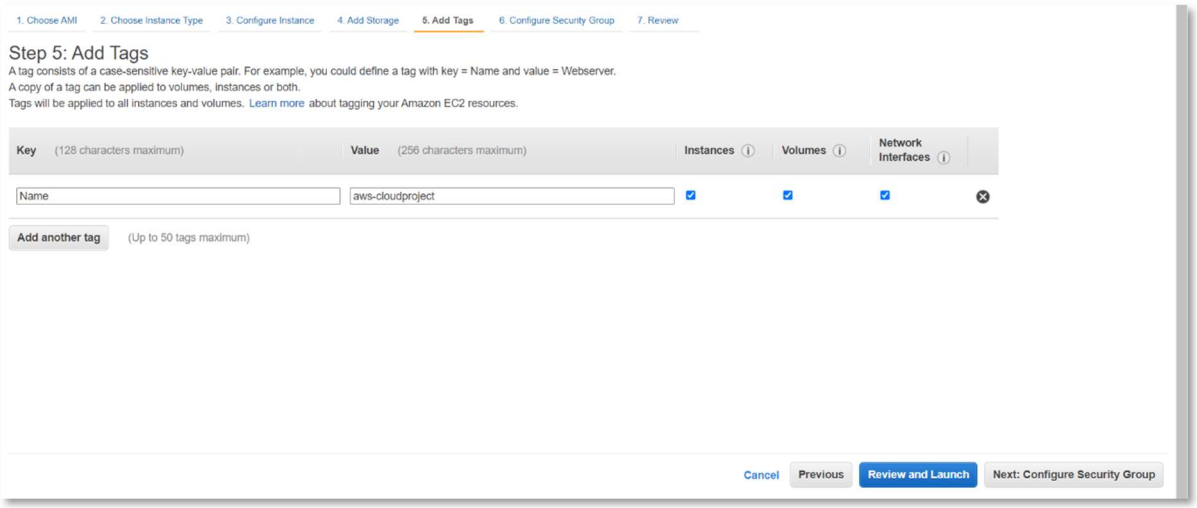
Picture15 - configure instance

- Configure the instance storage. Then click the “Next:Add Tags” button.



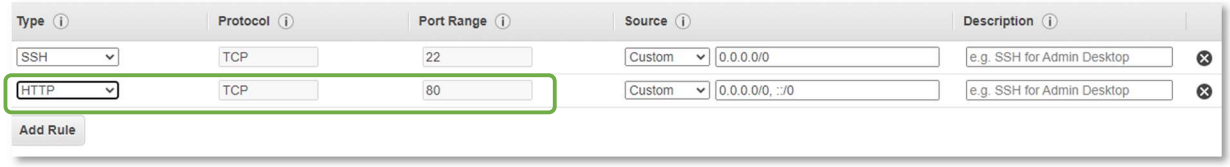
Picture16 - add storage

- Add tags to identify the instance. Then click the “Next: Configure Security Group” button.



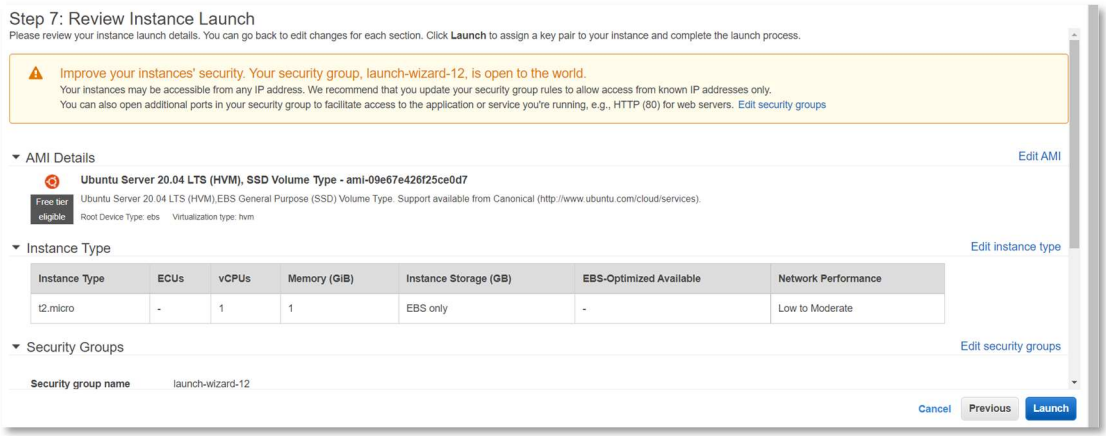
Picture17 - add tags

- Create a new security group. Open ports 22 and 80. Add these two rules . Then click the “Review and Launch” button.



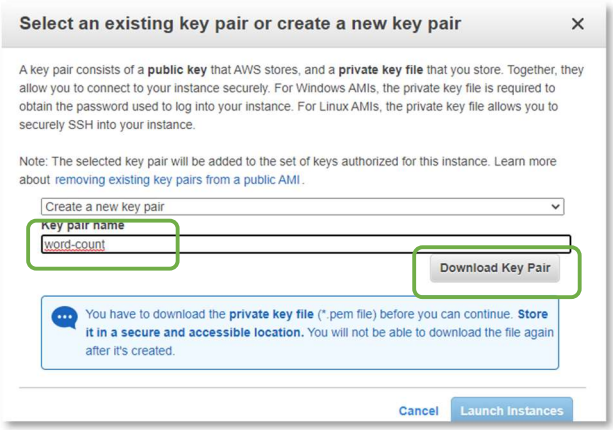
Picture18 - configure security group

- Click launch to assign a key pair to your instance and complete the launch process.



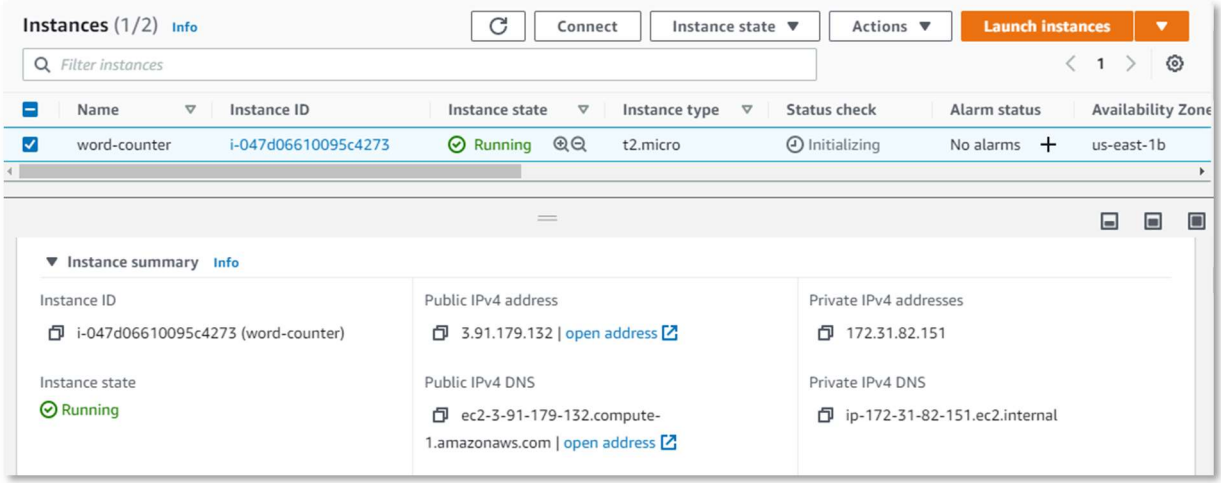
Picture19 - review

- On the screen that opens, choose to create a new key pair and enter your key pair name. Then, download key pair. Finally click the “Launch Instances” button.



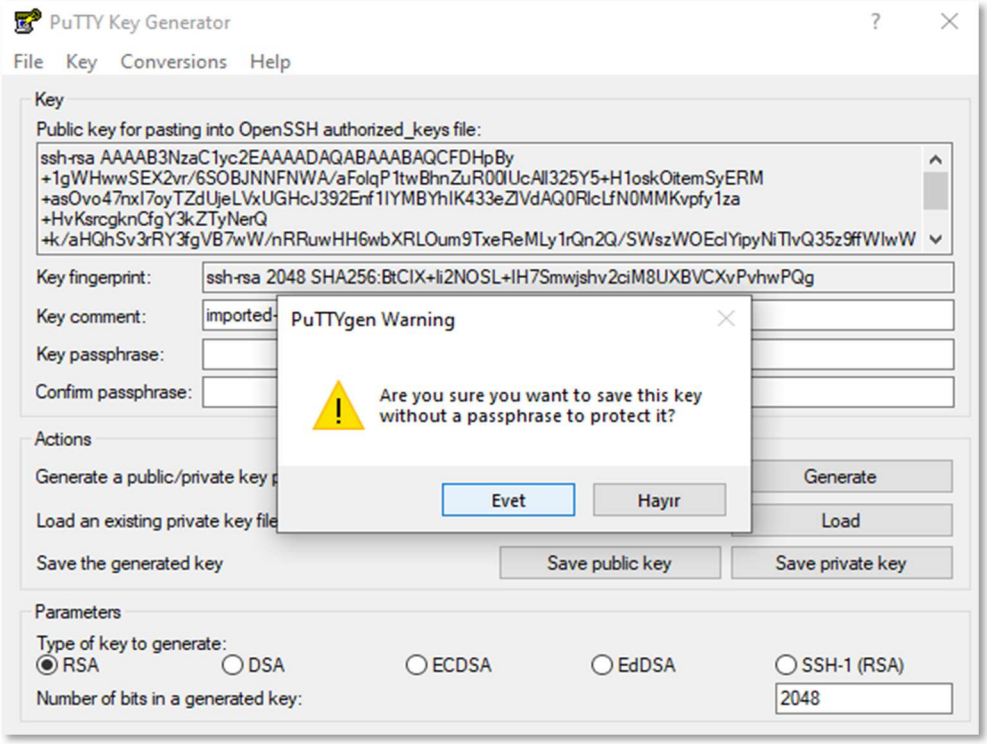
Picture20 - select key pair

- The instance you created is running.



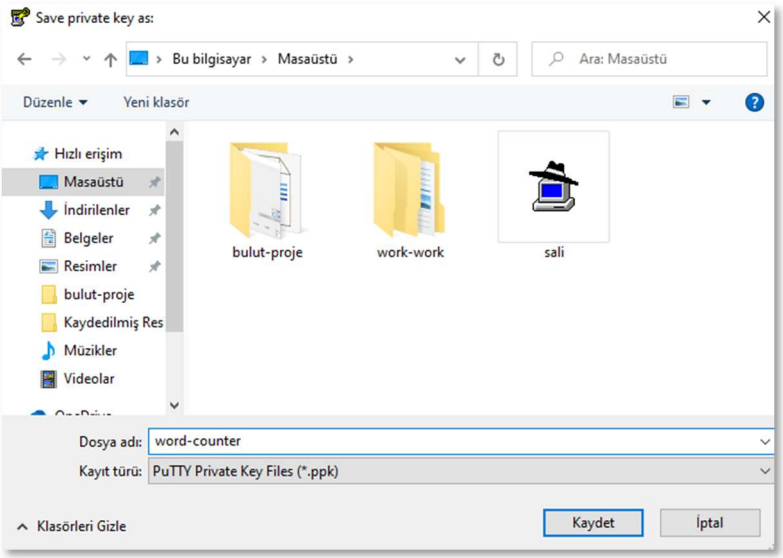
Picture21 - running instance

- Open PuTTYgen, click the "Load" button and select "All Files (*.*)" from the "Downloads" menu to install the key (.pem) you downloaded.



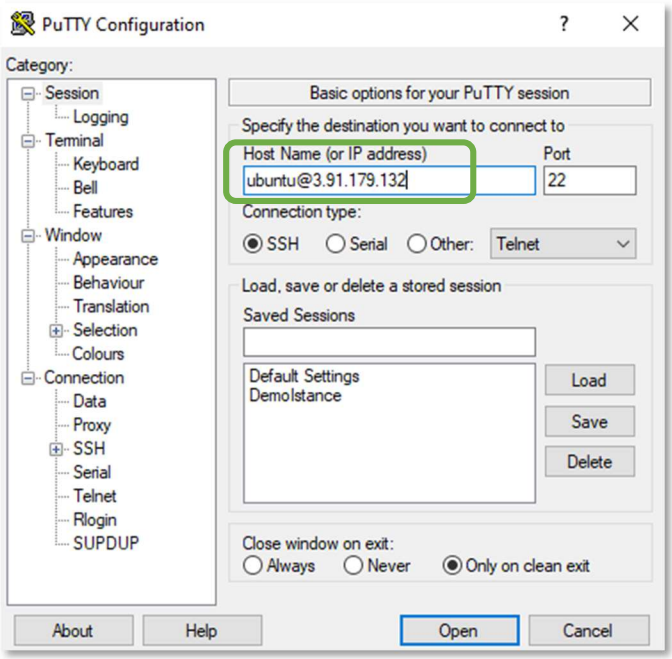
Picture22 - upload pem file to puttygen

- Click save private key with .ppk extension and close the PuTTYgen.



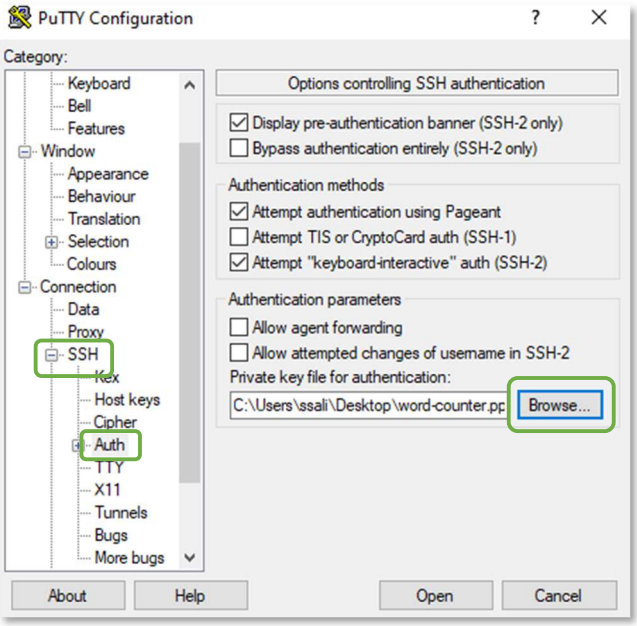
Picture23 - save pem file as ppk with puttygen

- Open PuTTY and enter your host name (**user_name@public_dns_name**). For an Ubuntu AMI as in our example, the user_name is **ubuntu**. For a public_dns_name, you can find it from your **EC2 Dashboard >> Running Instance tab.**

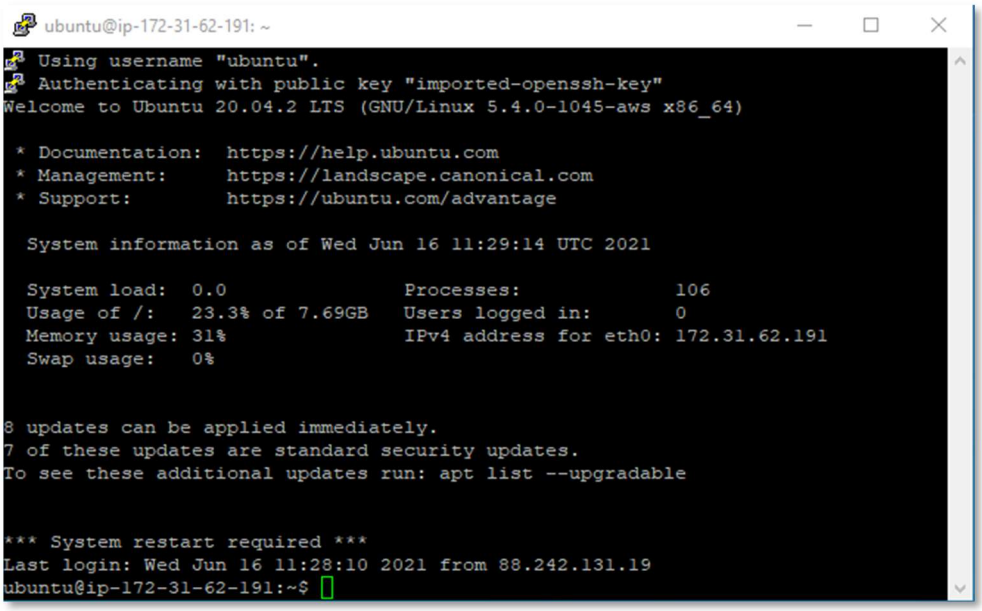


Picture24 - putty configuration 1

- Click the SSH >> Auth page and browse your private key with .ppk extension and click open.(Picture25) At the end of these processes, our terminal page will open. The remaining operations are performed on this terminal page.(Picture26)



Picture25 - putty configuration 2



Picture26 - putty command window

8- Installing and Updating Required Packages [3] [4]

- To update and upgrade the required package lists, the following commands are entered on the command screen, respectively.

  sudo apt-get update
  sudo apt-get upgrade

- Then we use the commands below to install the "pip" package first and then to install the "Django" package, which is the framework we used in our project.

  sudo apt-get install python3-pip
  sudo pip3 install Django

- Then we install our uwsgi package with the following command.

  sudo pip3 install uwsgi

  Why do we need uWSGI? The NGINX we will be using to deploy our project cannot run a Python process to host our application, for this we will need an application server to host a Python process running our Django project. NGINX and uWSGI will "talk" to each other using the uwsgi protocol and our django app will be able to run.

9- Clone the Django Application Repository [5]

- We enter the following command to clone our project that we have uploaded to github and that we want to deploy.(Our project name is "wordCounterProject")

  git clone https://github.com/ayyse/wordCounterProject.git

- Then we enter the "settings.py" file of the project we created and make the necessary adjustments.

  vi wordCounterProject/wordCounter/settings.py

- We write the public ip address of our machine into the "ALLOWED_HOSTS" box.

```
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = ['3.91.179.132']
```

Picture27 - allowed hosts

- We show the path of "static" and "media" files.

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.2/howto/static-files/

STATIC_URL = '/static/'
STATIC_ROOT = '/home/ubuntu/static'
MEDIA_ROOT = '/home/ubuntu/media'
```

Picture28 - files path

10- Configure uWSGI to Host the Django Project [6]

- To create our uwsgi config, we create a file named "django.ini"

  vi django.ini

- Enter the following commands in the django.ini file and save it.

  [uwsgi]
  chdir = /home/ubuntu/wordCounterProject
  env = DJANGO_SETTINGS_MODULE=wordCounter.settings
  module=wordCounter.wsgi:application
  master=True
  pidfile=/tmp/project-master.pid
  vacuum=True
  max-requests=5000
  daemonize=/home/ubuntu/word-counter-uwsgi.log
  socket = 127.0.0.1:3000

- We start “uwsgi” by entering the following command

  uwsgi --ini django.ini

11- Configure NGINX to Serve the Application [4]

- We install the nginx package with the following command

  sudo apt-get install nginx

- Then we edit the "default" file for the nginx configuration with the following command.

  sudo vi /etc/nginx/sites-available/default

```
upstream django {
server 127.0.0.1:3000;
}
# configuration of the server
server {
   # the port your site will be served on
   listen      80;
   # the domain name it will serve for
   server_name _; # substitute your machine's IP address or FQDN
   charset     utf-8;
   # max upload size
   client_max_body_size 75M;   # adjust to taste
   # Django media
   location /media  {
      alias /home/ubuntu/media; # your Django project's media files - amend as
required
   }
   location /static {
      alias /home/ubuntu/static; # your Django project's static files - amend as
required
   }
   # Finally, send all non-media requests to the Django server.
   location / {
      uwsgi_pass  django;
      include     /etc/nginx/uwsgi_params; # the uwsgi_params file you installed
   }
}
```

- Final view of our "default" file

```
ubuntu@ip-172-31-62-191: ~
upstream django {
server 127.0.0.1:3000;
}

# configuration of the server
server {
    # the port your site will be served on
    listen      80;
    # the domain name it will serve for
    server_name _; # substitute your machine's IP address or FQDN
    charset     utf-8;

    # max upload size
    client_max_body_size 75M;   # adjust to taste

    # Django media
    location /media  {
        alias /home/ubuntu/media; # your Django project's media files - amend as required
    }

    location /static {
        alias /home/ubuntu/static; # your Django project's static files - amend as required
    }

    # Finally, send all non-media requests to the Django server.
    location / {
        uwsgi_pass  django;
        include     /etc/nginx/uwsgi_params; # the uwsgi_params file you installed
    }
}
```

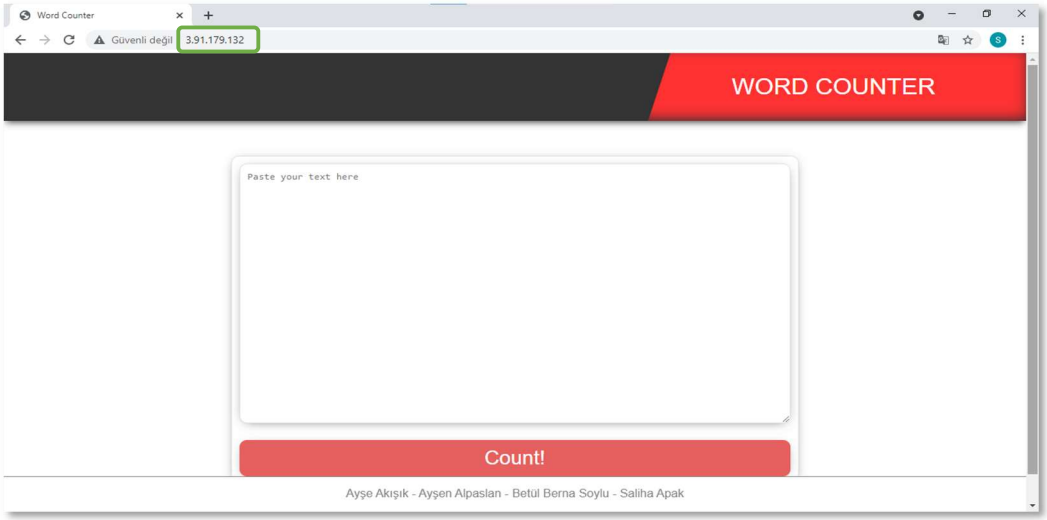
Picture29 - content of “default” file

- Finally nginx is restarted.

  sudo systemctl restart nginx

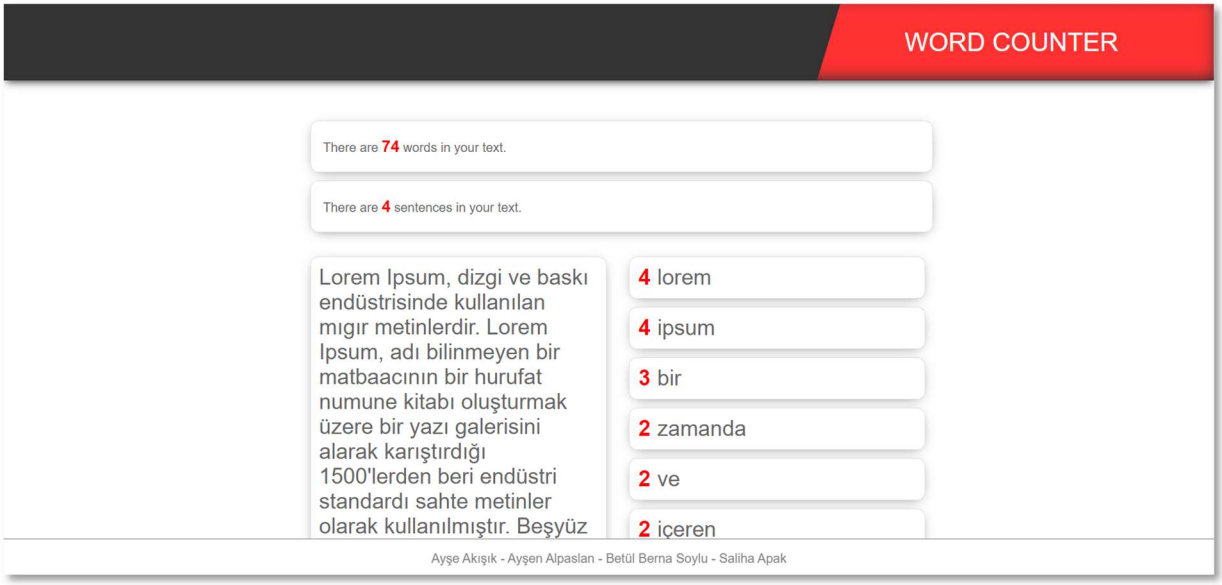
12- Test Run the Application

- We copy our public ip (http://3.91.179.132/) to the browser and check if our application is running.



Picture30 - it works!

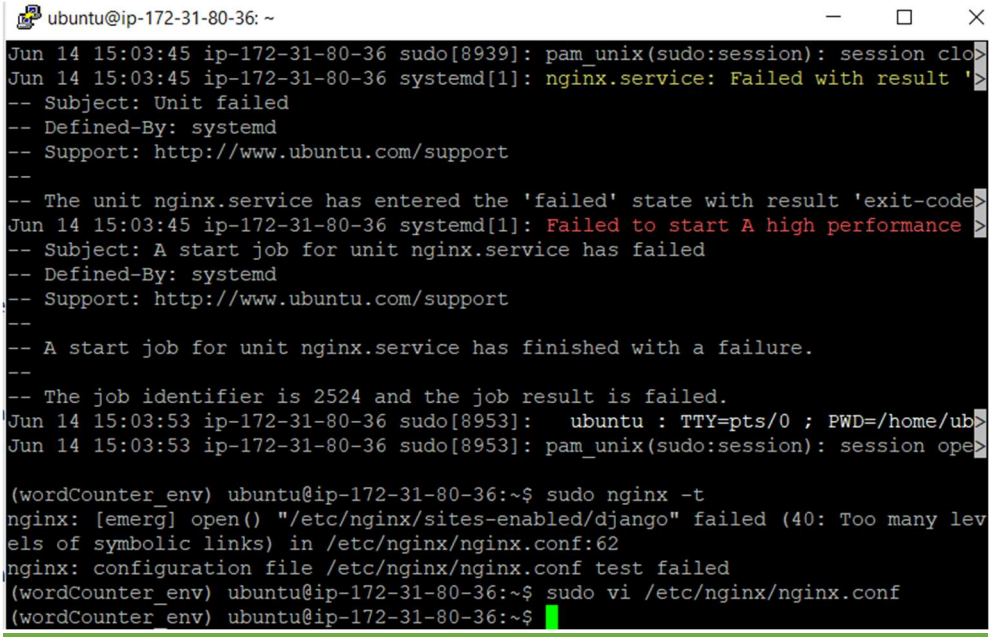
- We write the text we want to count in the input field and press the "Count" button. On the page that opens, there are how many words, how many sentences and how many times each word is repeated in the text entered. The words are ordered from the most to the least.



Picture31 - word count operation example result

Errors

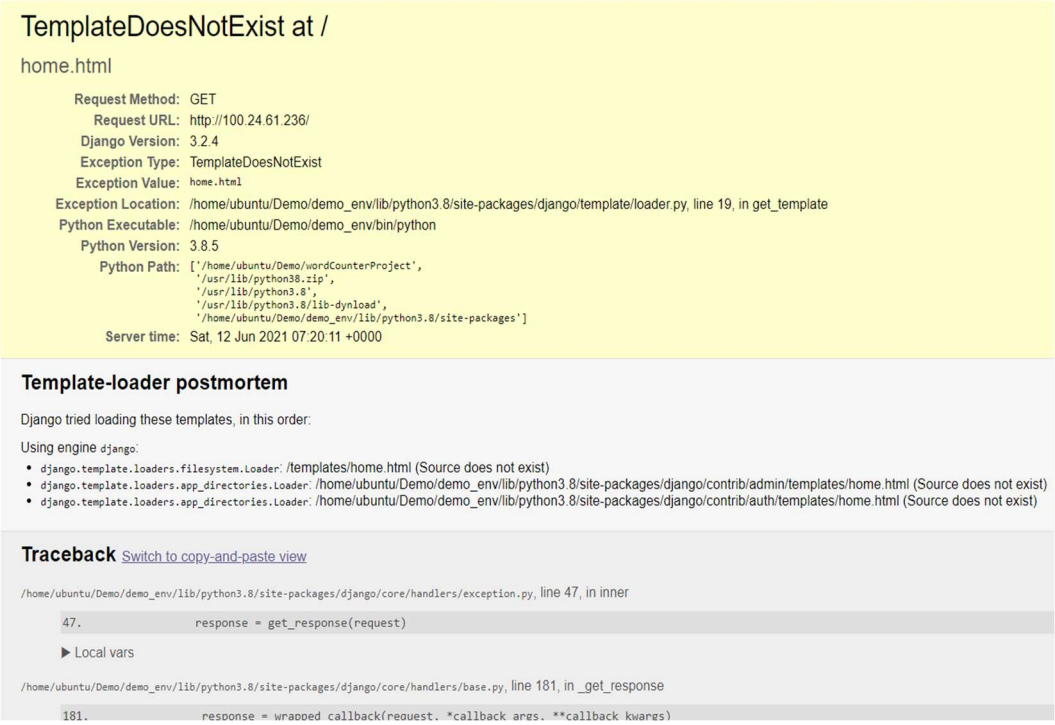
- We got an error because we wrote it in 'enable' before we should have written it in 'available' . We fixed the error when we put it in 'available'.



```
ubuntu@ip-172-31-80-36: ~
Jun 14 15:03:45 ip-172-31-80-36 sudo[8939]: pam_unix(sudo:session): session clo>
Jun 14 15:03:45 ip-172-31-80-36 systemd[1]: nginx.service: Failed with result '>
-- Subject: Unit failed
-- Defined-By: systemd
-- Support: http://www.ubuntu.com/support
--
-- The unit nginx.service has entered the 'failed' state with result 'exit-code>
Jun 14 15:03:45 ip-172-31-80-36 systemd[1]: Failed to start A high performance >
-- Subject: A start job for unit nginx.service has failed
-- Defined-By: systemd
-- Support: http://www.ubuntu.com/support
--
-- A start job for unit nginx.service has finished with a failure.
--
-- The job identifier is 2524 and the job result is failed.
Jun 14 15:03:53 ip-172-31-80-36 sudo[8953]:   ubuntu : TTY=pts/0 ; PWD=/home/ub>
Jun 14 15:03:53 ip-172-31-80-36 sudo[8953]: pam_unix(sudo:session): session ope>

(wordCounter_env) ubuntu@ip-172-31-80-36:~$ sudo nginx -t
nginx: [emerg] open() "/etc/nginx/sites-enabled/django" failed (40: Too many lev
els of symbolic links) in /etc/nginx/nginx.conf:62
nginx: configuration file /etc/nginx/nginx.conf test failed
(wordCounter_env) ubuntu@ip-172-31-80-36:~$ sudo vi /etc/nginx/nginx.conf
(wordCounter_env) ubuntu@ip-172-31-80-36:~$
```

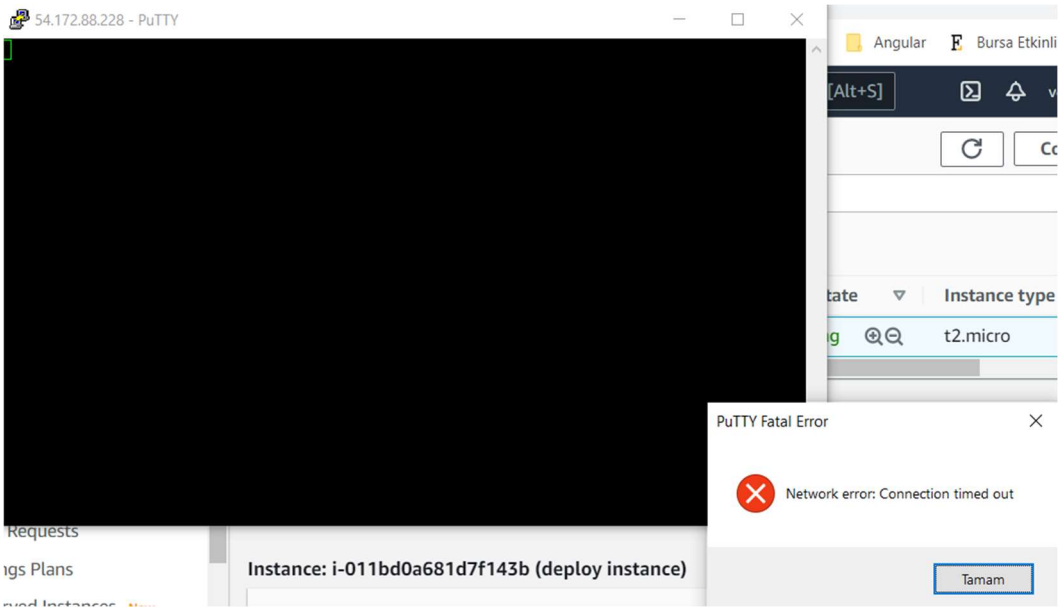
Picture32 – failed to start A high perfromance (error - 1)

- The html file may be incorrect or incomplete at the specified path. Therefore, the html file could not be found. The inside of the html file has been checked and corrected.



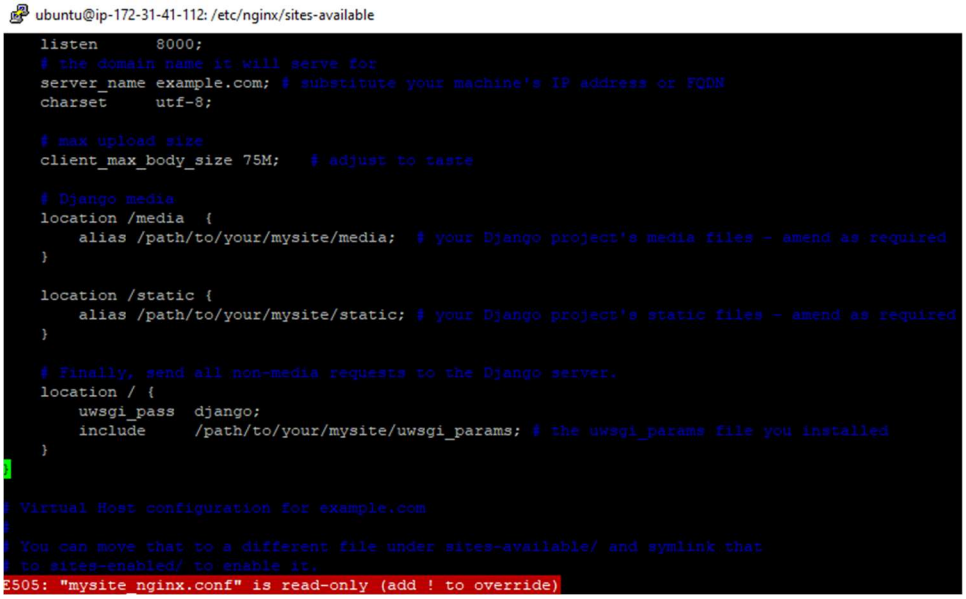
Picture33 – html file not found (error - 2)

- Internet connection is weak or disconnected and timed out. The error has been fixed by making a stronger internet connection.



Picture34 – internet Connection (error - 3)

- Since "sudo" command was not entered, it did not allow us to write in the file. We added the "sudo" command.



Picture34 – command error (error - 4)

Work Sharing

- Ayşe Akışık → Instance launch, session initialization in PuTTY, deploy operation
- Ayşen Alpaslan → Creating the django project, interface design and coding
- Betül Berna Soylu → Team Leader
- Saliha Apak → Instance launch, session initialization in PuTTY, deploy operation

References

[1] https://docs.djangoproject.com/en/3.2/intro/tutorial01/

[2] Lab1 Sheet

[3] https://docs.djangoproject.com/en/3.2/howto/deployment/wsgi/uwsgi/

[4] https://uwsgi-docs.readthedocs.io/en/latest/tutorials/Django_and_nginx.html#before-you-start-setting-up-uwsgi

[5] https://alicecampkin.medium.com/setting-up-a-forked-django-project-53d5939b7e9e

[6] https://www.guguweb.com/2019/11/13/django-nginx-deploy-your-django-project-on-a-production-server/

Github Link

- https://github.com/ayyse/wordCounterProject