Aim:

Implementing Python program to parse the given timelog data files to get how many hours and minutes the author spent in each file on cloud platform using python.

Objective:

Design a cloud web page to select the time log data file from your local machine and then read the time log data file and report the result.

Introduction:

AWS (**Amazon Web Services**) is a comprehensive, evolving cloud computing platform provided by Amazon that includes a mixture of infrastructure as a service (IaaS), platform as a service (PaaS) and packaged software as a service (SaaS) offering.

Flask is an API of Python that allows us to build up web-applications. It was developed by Armin Ronacher. **Flask's** framework is more explicit than Django's framework and is also easier to learn because it has less base code to implement a simple web-Application.

Implementation:

Using Flask build a web-application for python program(tlparesr.py). The below is the code for tlparser_app.py

tlparser_app.py:

```
#!/usr/bin/env python

# coding: utf-8

import os

from flask import Flask, render_template, request, redirect, url_for

#from werkzeug.utils import secure_filename

from tlparser import *


app = Flask(__name__)

app.config['UPLOAD_PATH'] = 'uploads'

@app.route('/')

def home():
```

```
    return render_template('index.html')
@app.route('/', methods = ['POST'])
def upload_tl_file():
    uploaded_tl_file = request.files['file']
    if uploaded_tl_file.filename != '':
        tl_file = os.path.join(app.config['UPLOAD_PATH'], uploaded_tl_file.filename)
        #filename = secure_filename(f.filename)
        uploaded_tl_file.save(tl_file)
        total_work_time_hours, total_work_time_mins = tlparse(tl_file)
        return render_template("index.html", total="Total Work Time: " +
str(int(total_work_time_hours)) + " Hours and " + str(int(total_work_time_mins)) + " Minutes",
                               filen=uploaded_tl_file.filename)
    else:
        #render_template("index.html")
        redirect(url_for('home'))
#b = open(os.path.join(app.config['UPLOAD_PATH'], filename), 'r')
#output = tlparse(b)
if __name__ == '__main__':
   app.run(host='0.0.0.0', debug=True)
```

tlparser.py:

```
import argparse
import datetime
import re


def tlparse(tl_file):
    with open(tl_file, 'r') as f:
```

```
    #all_lines = f.readlines()
    all_lines = f.read().splitlines()
time_log_flag = False
total_work_time_seconds = 0
time_log_pattern = 'Time Log'
work_date_pattern = '[0-9]{1,2}/[0-9]{1,2}/[0-9]{2,4}'
work_times_pattern = '\d{1,2}:\d{1,2}[ap]m\s+-\s+\d{1,2}:\d{1,2}[ap]m'
for index, line in enumerate(all_lines):
    print("Line #%d: %s" %(index+1, line))
    if (not time_log_flag):
        time_log_found = re.match(time_log_pattern, line, re.IGNORECASE)
        if time_log_found:
            time_log_flag = True
            print(f"Time Log Found: {time_log_found}\n")
        else:
            print(f"Ignoring Line {index+1}: {line}\n")
    else:
        work_date_found = re.search(work_date_pattern, line, re.IGNORECASE)
        work_times_found = re.search(work_times_pattern, line, re.IGNORECASE)
        #print("Work Date Found: ", work_date_found)
        #print("Work Times Found: ", work_times_found)
        if work_date_found and work_times_found:
            work_start_date_str = work_date_found.group(0)
            previous_work_date_str = work_start_date_str
        elif work_times_found and (not work_date_found):
            work_start_date_str = previous_work_date_str
        print("work start date: ", work_start_date_str)
```

```
        print("work start date length: ", len(work_start_date_str))
        if work_times_found:
            work_times_str = work_times_found.group(0)
            print(work_times_str)
            work_times = work_times_str.upper().split("-")
            work_start_time_str = work_times[0].strip()
            work_end_time_str = work_times[1].strip()
            work_start_time_of_day = work_start_time_str[-2:].upper()
            work_end_time_of_day = work_end_time_str[-2:].upper()
            if (work_start_time_of_day == "PM") and (work_end_time_of_day == "AM"):
                if (len(work_start_date_str.split("/")[-1]) == 2):
                    work_end_date = datetime.datetime.strptime(work_start_date_str, '%m/%d/%y')
+ datetime.timedelta(days=1)
                    work_end_date_str = work_end_date.strftime('%m/%d/%y')
                elif (len(work_start_date_str.split("/")[-1]) == 4):
                    work_end_date = datetime.datetime.strptime(work_start_date_str,'%m/%d/%Y')
+ datetime.timedelta(days=1)
                    work_end_date_str = work_end_date.strftime('%m/%d/%Y')
            else:
                work_end_date_str = work_start_date_str
            work_start_date_time_str = work_start_date_str + " " + work_start_time_str
            work_end_date_time_str = work_end_date_str + " " + work_end_time_str
            print(f"Start Date Time: {work_start_date_time_str} End Date Time:
{work_end_date_time_str}")
            if (len(work_start_date_str.split("/")[-1]) == 2):
                work_start_datetime_obj = datetime.datetime.strptime(work_start_date_time_str,
'%m/%d/%y %I:%M%p')
                work_end_datetime_obj = datetime.datetime.strptime(work_end_date_time_str,
'%m/%d/%y %I:%M%p')
```

```
            elif (len(work_start_date_str.split("/")[-1]) == 4):
                work_start_datetime_obj = datetime.datetime.strptime(work_start_date_time_str, '%m/%d/%Y %I:%M%p')
                work_end_datetime_obj = datetime.datetime.strptime(work_end_date_time_str, '%m/%d/%Y %I:%M%p')
                work_time = work_end_datetime_obj - work_start_datetime_obj
                work_time_seconds = work_time.total_seconds()
                total_work_time_seconds += work_time_seconds
                print(f"Work Time Minutes: {work_time_seconds/60}")
                print(f"Total Work Time Minutes: {total_work_time_seconds/60} \n")
            else:
                print(f"Ignoring Line #{index+1}: {line}\n")
    print(f"Total Work Time: {int(total_work_time_seconds)} Seconds")
    total_work_time = datetime.timedelta(seconds=total_work_time_seconds)
    print("Total Work Time: ", total_work_time)
    total_work_time_mins, total_work_time_secs = divmod(total_work_time_seconds, 60)
    total_work_time_hours, total_work_time_mins = divmod(total_work_time_mins, 60)
    return (total_work_time_hours, total_work_time_mins)


if __name__ == '__main__':
    # Get command line arguments
    parser = argparse.ArgumentParser()
    parser.add_argument("time_log_file", help="Time Log File")
    args = parser.parse_args()
    time_log_file = args.time_log_file
    total_work_time_hours, total_work_time_mins = tlparse(time_log_file)
    print(f"Total Work Time: {int(total_work_time_hours)} Hours and {int(total_work_time_mins)} Minutes")
```

index.html:

```
<!DOCTYPE html>
<html>
   <head>
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
   </head>
  <body>
  <div id="div1" >
        <center><h2>Comparative Programing Languages</h2></center>
     </div>
      <div id="div2" >
        <center><h2>Kalyan Veluri</h2>
        <h2>2820437</h2></center>
     </div>
     <div id="div3" >
        <center><h1>Time Log Parser</h1></center>
     </div>
     <div id="div4">
        <center>
    <form action = "" method = "POST"
      enctype = "multipart/form-data">
      <input type = "file" name = "file" accept=".txt" class ="inp" id ="inp"/>
      </br></br>
     </br></br>
      <input type = "submit"/>
    </form>
    </center>
    </div>
    <div id='div5'><center>
       <h3>{{output}}</h3></br>{{tot}}</center>
    </div>
    <footer id='div6'></footer>
  </body>
</html>

<meta name="viewport" content="width=device-width, initial-scale=1.0">
<h1><b>Result:</b></h1>
 {{filen}}
 </br>
 </br>
```

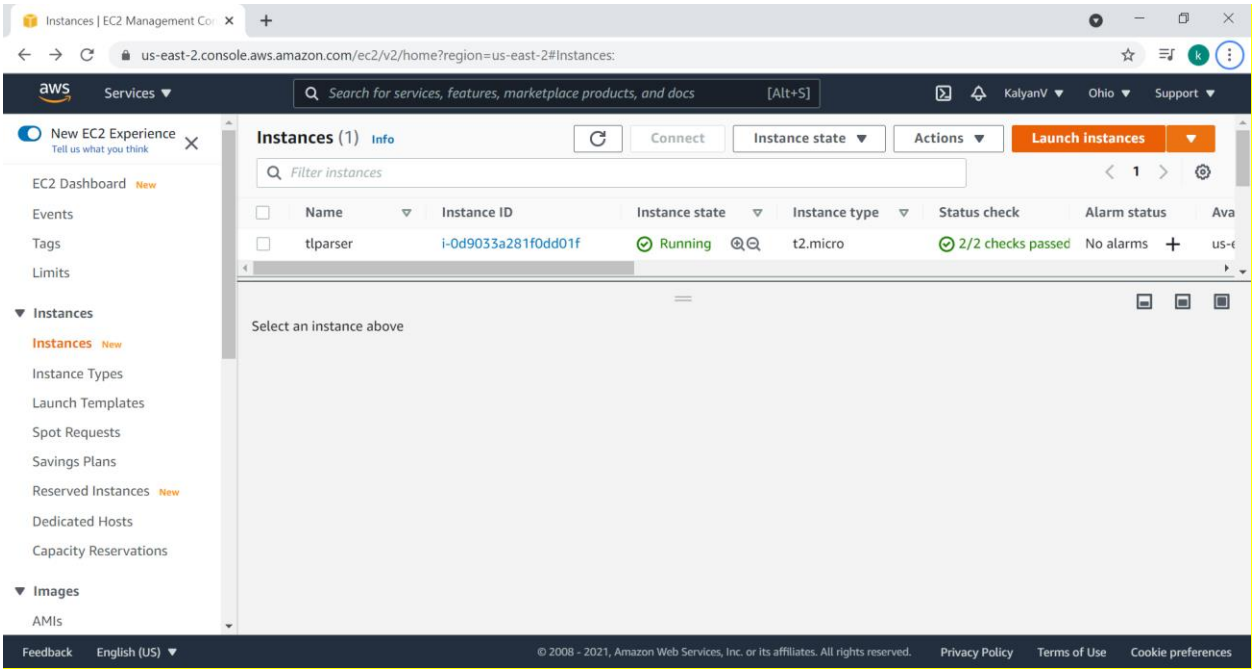
</br>
</br>
{{total}}

Implementation of Cloud Webpage using AWS:

An **EC2 instance** is a virtual server in Amazon's Elastic Compute Cloud (**EC2**) for running applications on the Amazon Web Services (**AWS**) infrastructure. ... Users can select an AMI provided by **AWS**, the user community, or through the **AWS** Marketplace. Users can also create their own AMIs and share them.

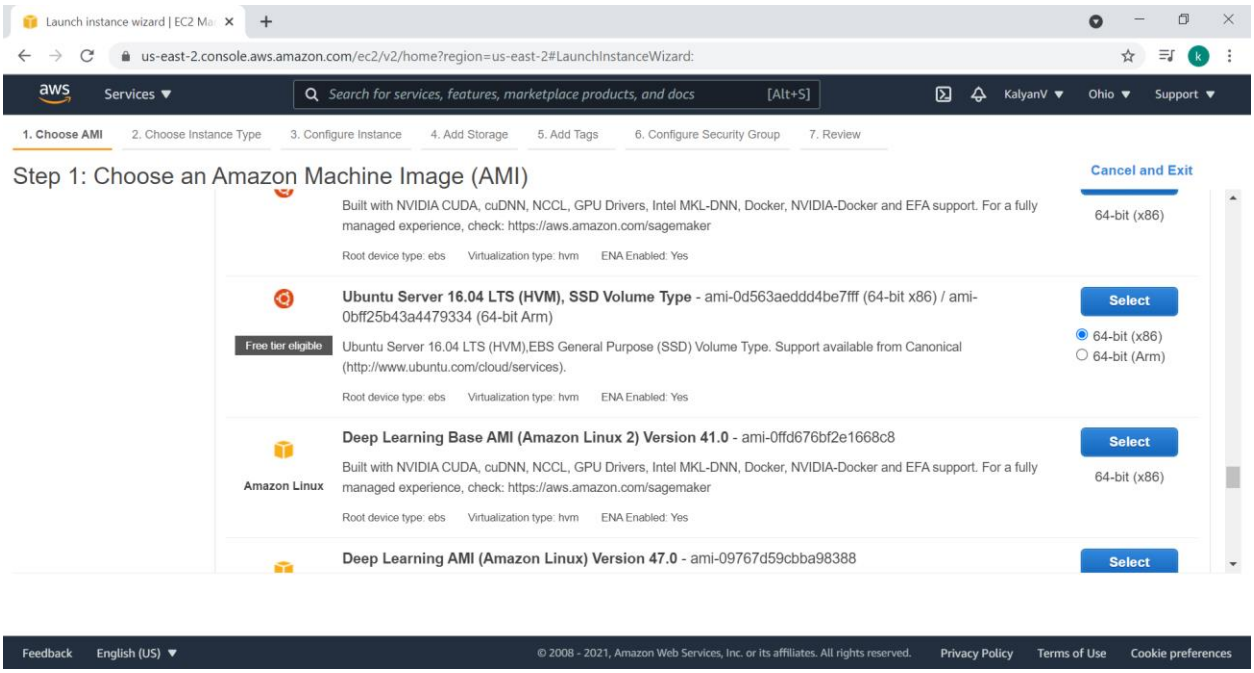
Created an Instance using EC2 instance in AWS,

Launch instances using Launch instances on right top of screen.

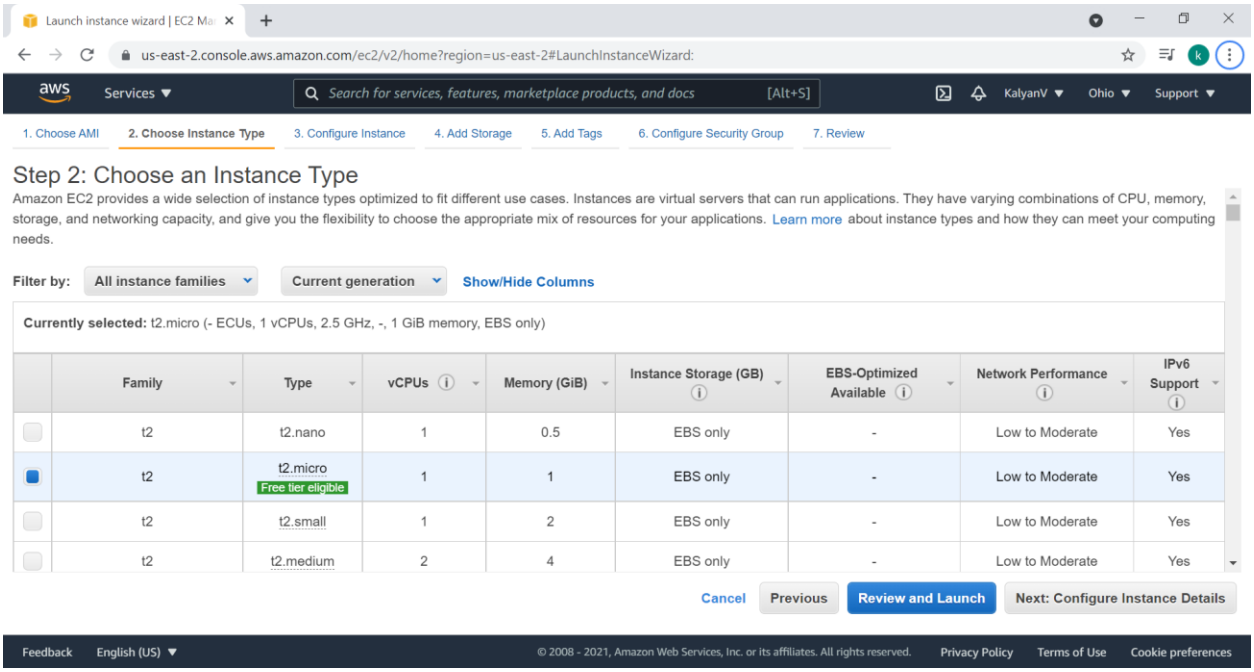


Step 1: Choose an Amazon Machine Image (AMI)

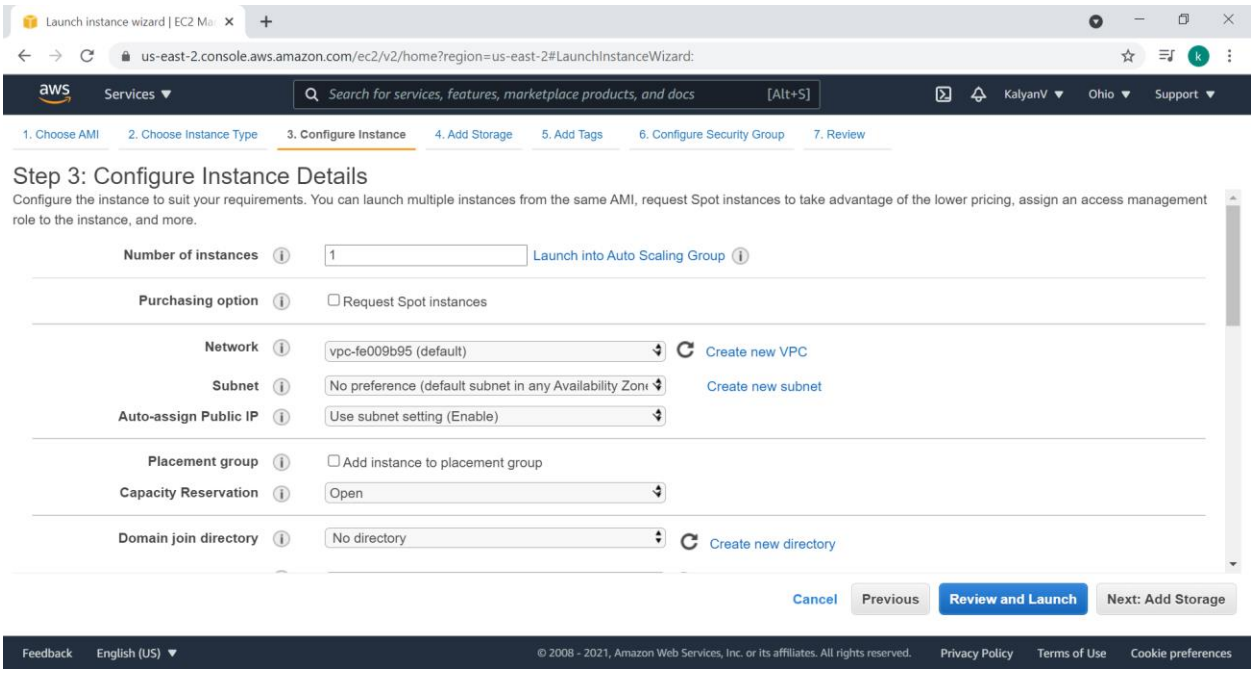
Select **Ubuntu Server 16.04 LTS (HVM), SSD Volume Type**



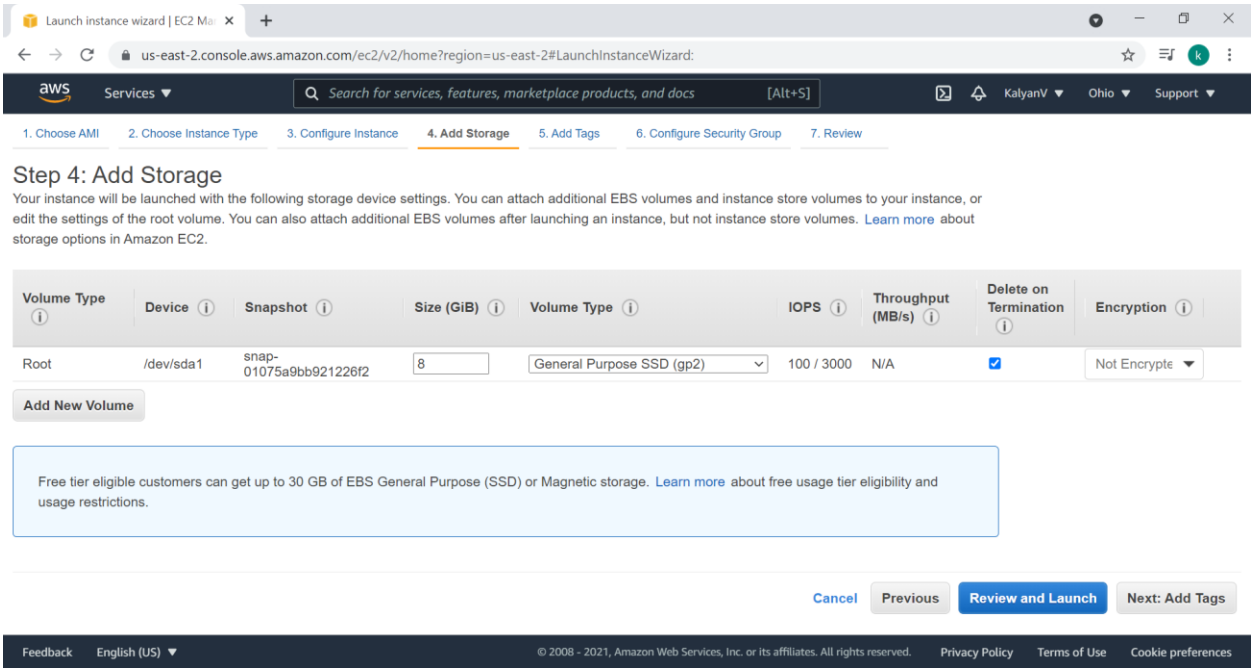
Step 2: Choose an Instance Type



Step 3: Configure Instance Details



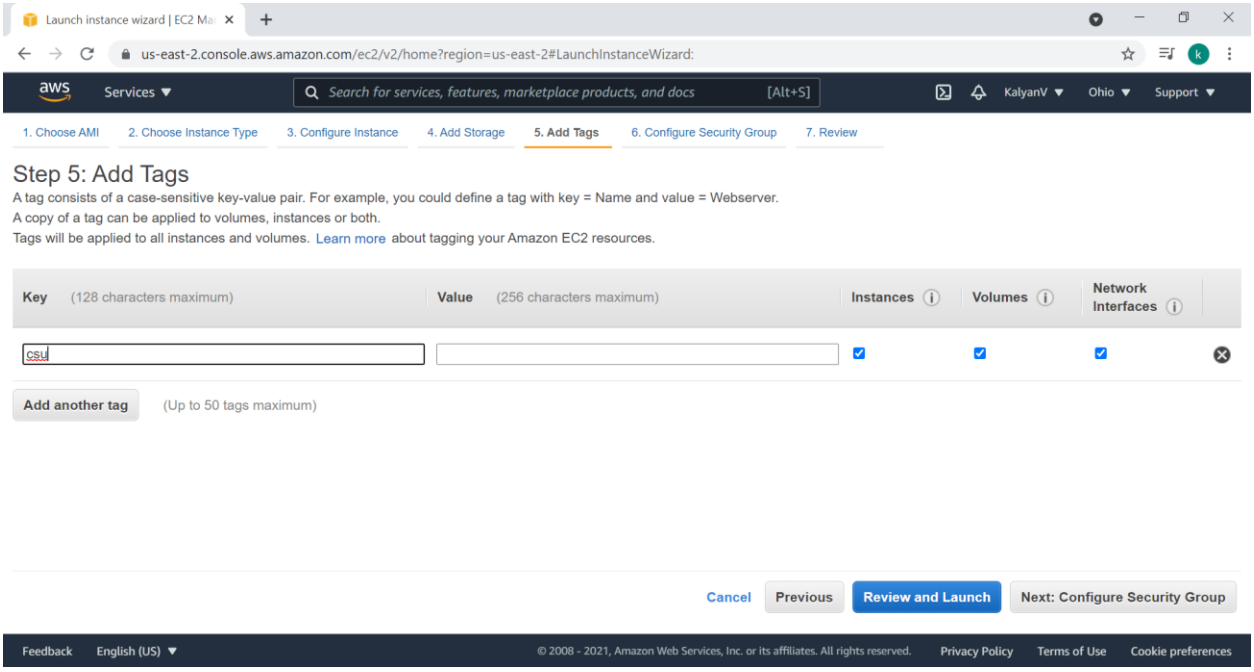
Step 4: Add Storage



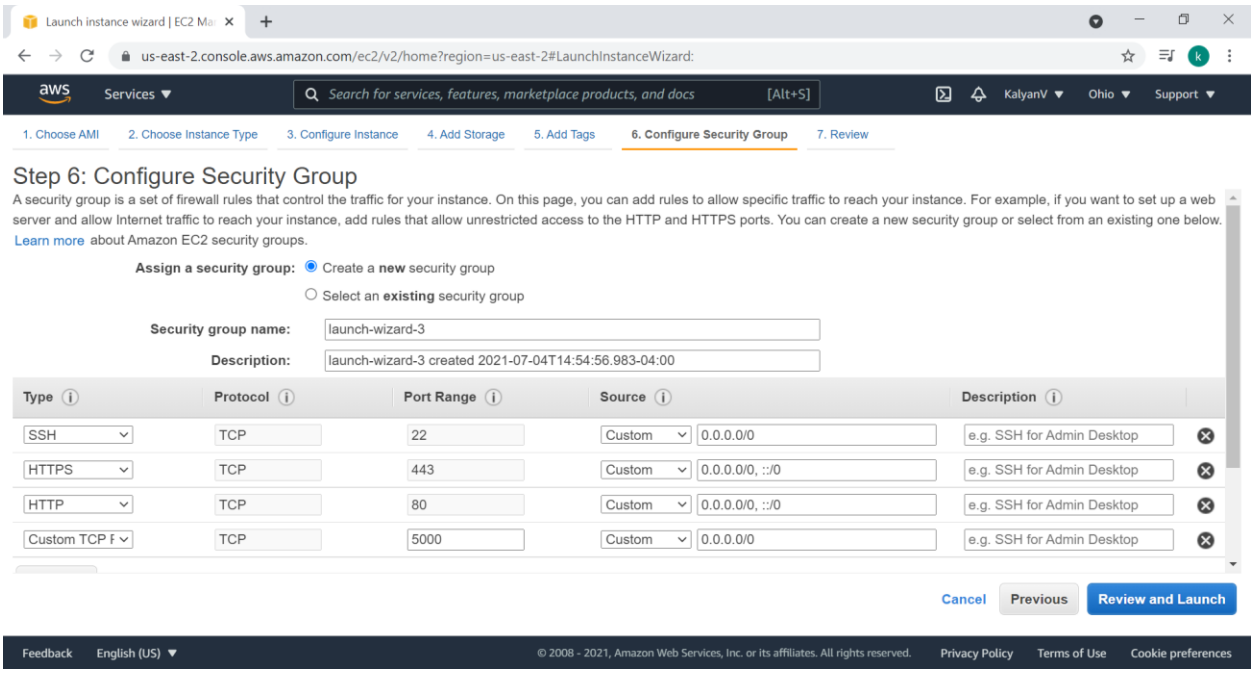
Step 5: Add Tags

Create a key and download it.

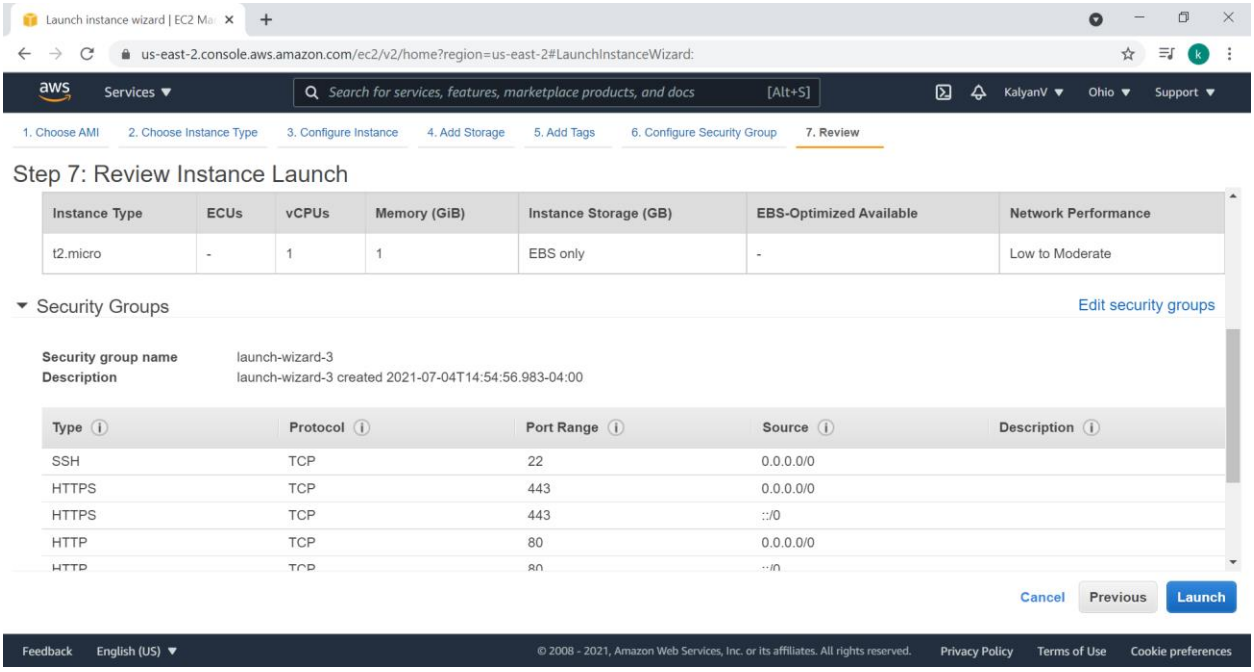
Covert the downloaded .pem file to .ppk using Puttygen

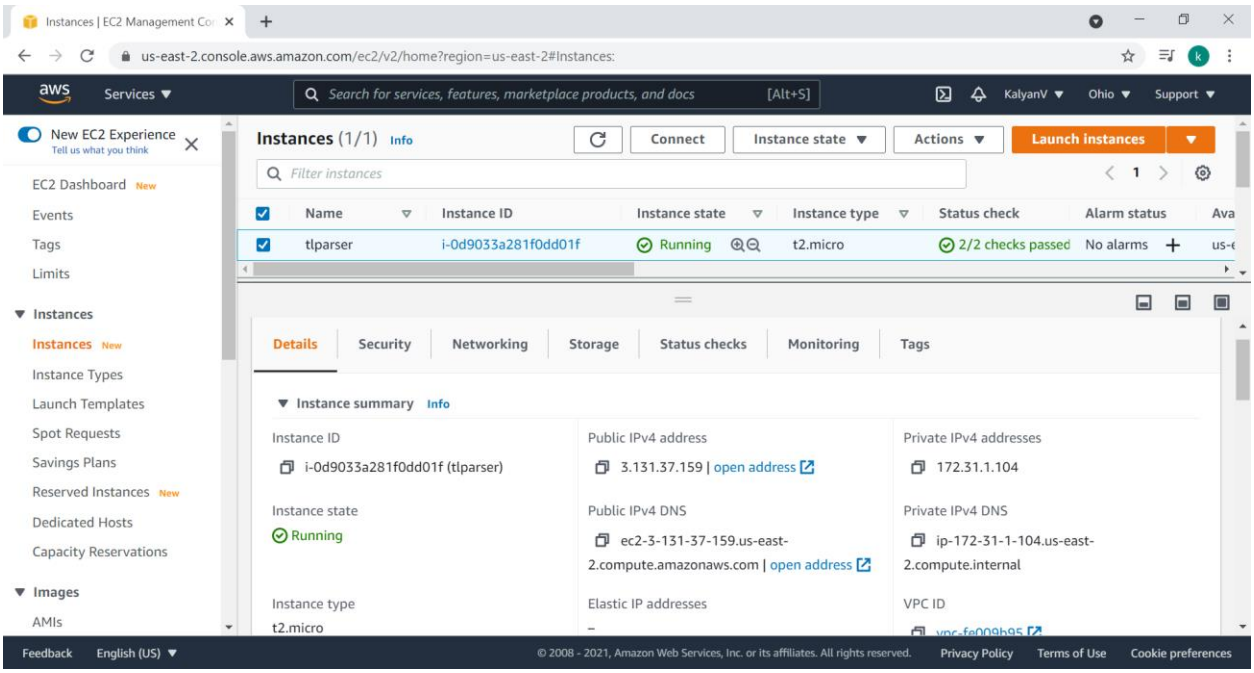


Step 6: Configure Security Group

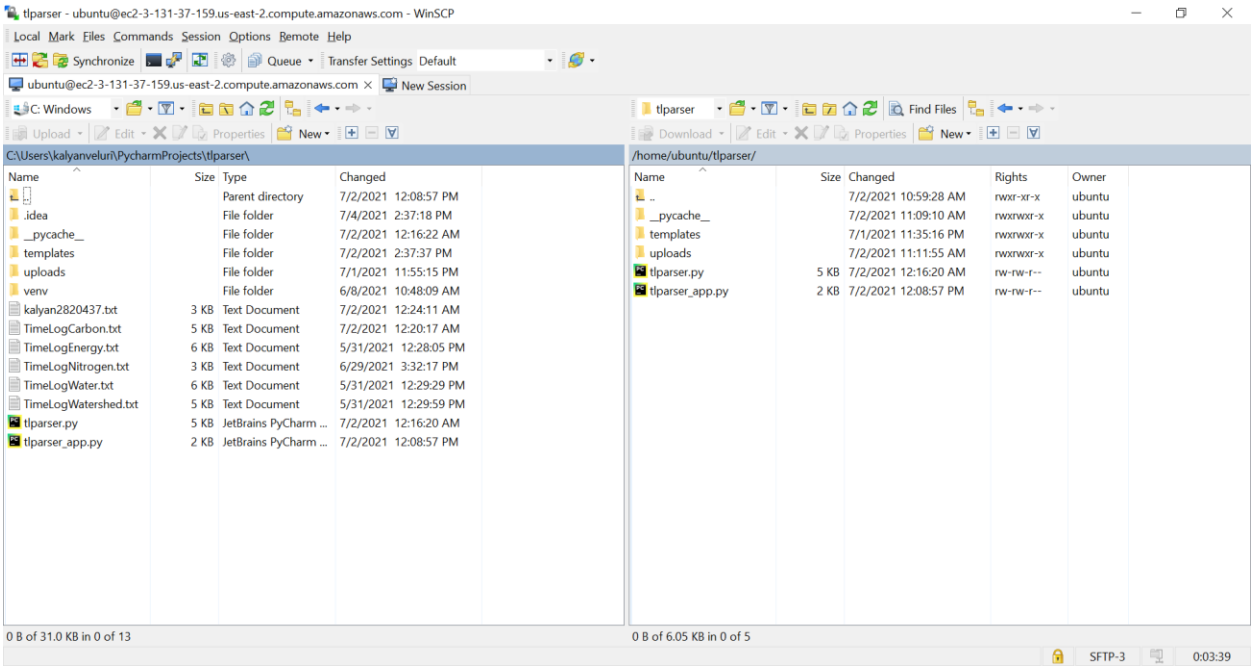


Step 7: Review Instance Launch



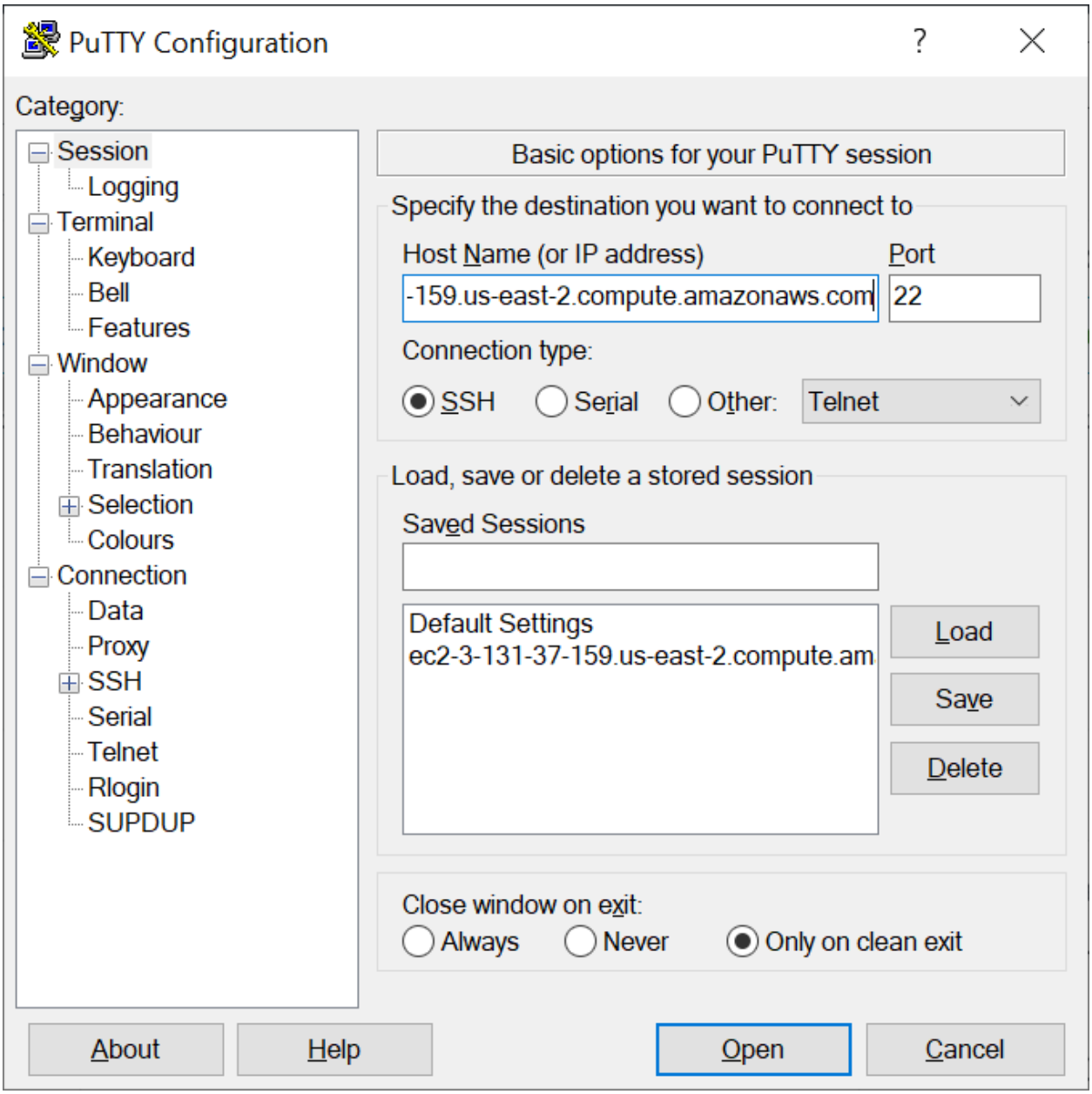


Using WinSCP connect to instance ID and copy the files to the directory.



Putty:

Use putty and the add the Public IPv4 DNS which show for your instance created in AWS in the Host Name (or IP address).



Open the Session and create a directory. In my case I created **“kven”**

Use the following commands and install Python3, Virtualenv, flask etc;

sudo apt update

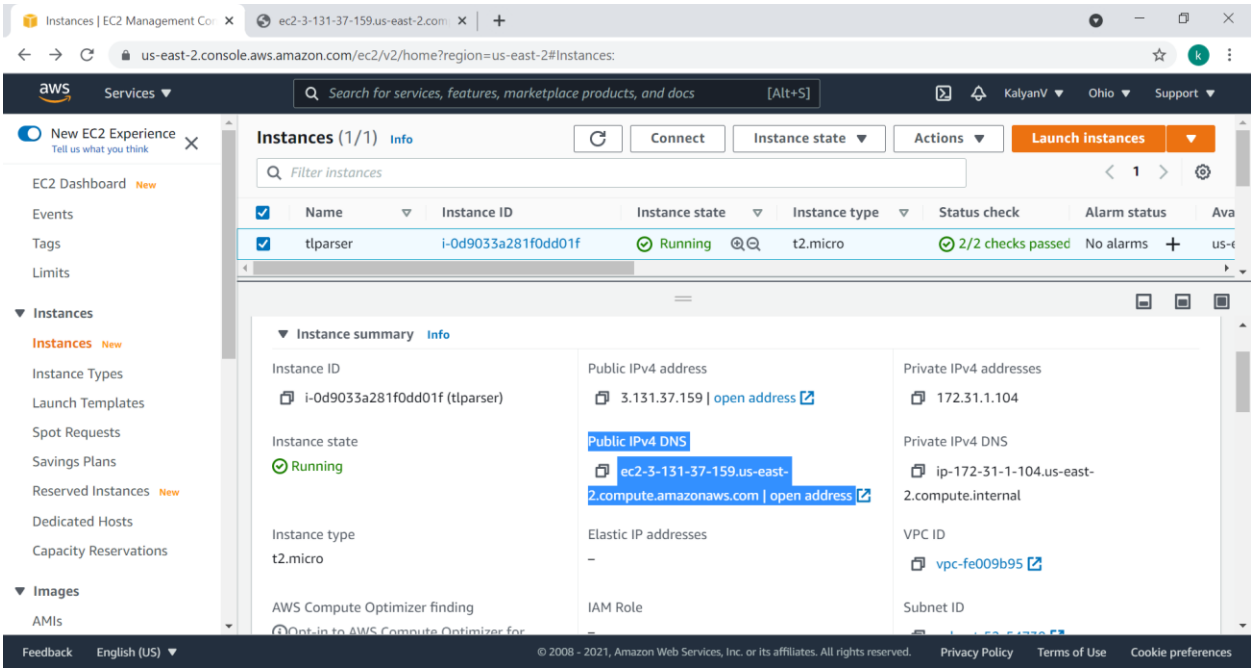
sudo apt install python3-pip

sudo apt install virtualenv

pip install flask

Open the directory where you stored the python files and Run the python code using python filename.py (I used python tlparser_app.py)

Open the instance which created in AWS and open the public IPv4 DNS.



Here is my instance address (http://ec2-3-131-37-159.us-east-2.compute.amazonaws.com:5000/)

Result:

Result:

Result:

kalyan2820437.txt

Total Work Time: 110 Hours and 50 Minutes

Reference:

- https://www.geeksforgeeks.org/python-program-convert-time-12-hour-24-hour-format/
- https://www.tutorialspoint.com/adding-time-in-python

- https://flask.palletsprojects.com/en/2.0.x/patterns/fileuploads/
- https://towardsdatascience.com/creating-a-website-to-host-your-python-web-application-f06f694a87e8

My Webpage Link:

http://ec2-3-131-37-159.us-east-2.compute.amazonaws.com:5000/