

**HUMBER COLLEGE**  
**AWS Cloud Computing**

**Project – Sentiment Analysis using AWS Comprehend**

**22 – 03 – 2024**

- By Sree Kodavanti

**Project Description:**

Show the sentiment score of the text that is posted in the text field, i.e., the "positive," "negative," and "neutral" scores together.

In order to integrate AWS Comprehend into a Flask-developed web application, communication between the frontend and backend levels must be established. In this design, HTTP requests are used to transfer user input from the frontend to the Flask backend. The text data is then processed within the Flask application and sent via the AWS SDK for Python (Boto3) to the AWS Comprehend service. The data is formatted and sent back to the frontend by the Flask backend when it receives the analysis results from AWS Comprehend. The sentiment analysis results are shown dynamically in the user interface via the frontend, which is created with HTML, CSS, and JavaScript. Here, I am going to use a simple HTML file to create a front-end website.

Steps for our project:

- Develop HTML Templates: Create HTML files for the frontend interface, including a form to capture text input.
- Create Flask App File: Write a Python script (e.g., app.py) to define the Flask application.
- Import Flask: Import Flask module in your app.py script.
- Define Route for Homepage: Define a route in your Flask app to render the homepage HTML template.
- Run Flask App: Add code to run the Flask app (if `__name__ == '__main__'`: `app.run()`) at the end of the script.
- Access Cloud9 Preview: Access the Flask app preview in your Cloud9 environment to verify functionality.
- Integrate AWS Comprehend: Install Boto3 library and use it to interact with AWS Comprehend within your Cloud9 environment.
- Implement Sentiment Analysis Route, create an instance, create a web server: Define a route in your Flask app to handle sentiment analysis requests. Here we use Nginx and Gunicorn.
- Process Text Input: Extract text input from HTTP requests and pass it to your sentiment analysis function.
- Return Analysis Results: Return the sentiment analysis results as JSON or HTML response.
- Update HTML Template: Update the HTML template to display the sentiment analysis results.

- Test Integration: Test the integration by entering text in the frontend form and submitting it.
  - Deploy: Once satisfied with the functionality, you can deploy the Flask app to a production server.

For this project, I have used two files: index1.html and app.py (codes mentioned below).

# Creating a Front – end Web Application:

For our project, we need to create a simple front-end application where we input the text, and the output is displayed on that website.

For this we use html language and create a simple front-end website.

Below is the screenshot of the code file and before loading it to AWS we need to make sure we don't have any errors in the code, so we use VSCode to run the file.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Sentiment Analysis</title>
</head>
<body>
  <h1>Sentiment Analysis</h1>
```

```
<form id="sentimentForm">
  <label for="textInput">Enter Text to Analyze:</label><br>
  <textarea id="textInput" name="textInput" rows="4" cols="50"></textarea><br>
  <button type="submit">Analyze Sentiment</button>
</form>
<div id="result"></div>

<script>
  document.getElementById('sentimentForm').addEventListener('submit',
function(event) {
  event.preventDefault();
  const text = document.getElementById('textInput').value;
  analyzeSentiment(text);
});

function analyzeSentiment(text) {
  fetch('/analyze_sentiment', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({ text: text })
  })
  .then(response => response.json())
  .then(data => {
    displayResults(data);
  })
  .catch(error => {
    console.error('Error:', error);
  });
}

function displayResults(sentimentScores) {
  const resultDiv = document.getElementById('result');
  resultDiv.innerHTML =
    <h2>Sentiment Analysis Results:</h2>
```

```

<p>Positive Score: ${sentimentScores.positive}</p>
<p>Negative Score: ${sentimentScores.negative}</p>
<p>Neutral Score: ${sentimentScores.neutral}</p>
    `;
}
</script>
</body>
</html>

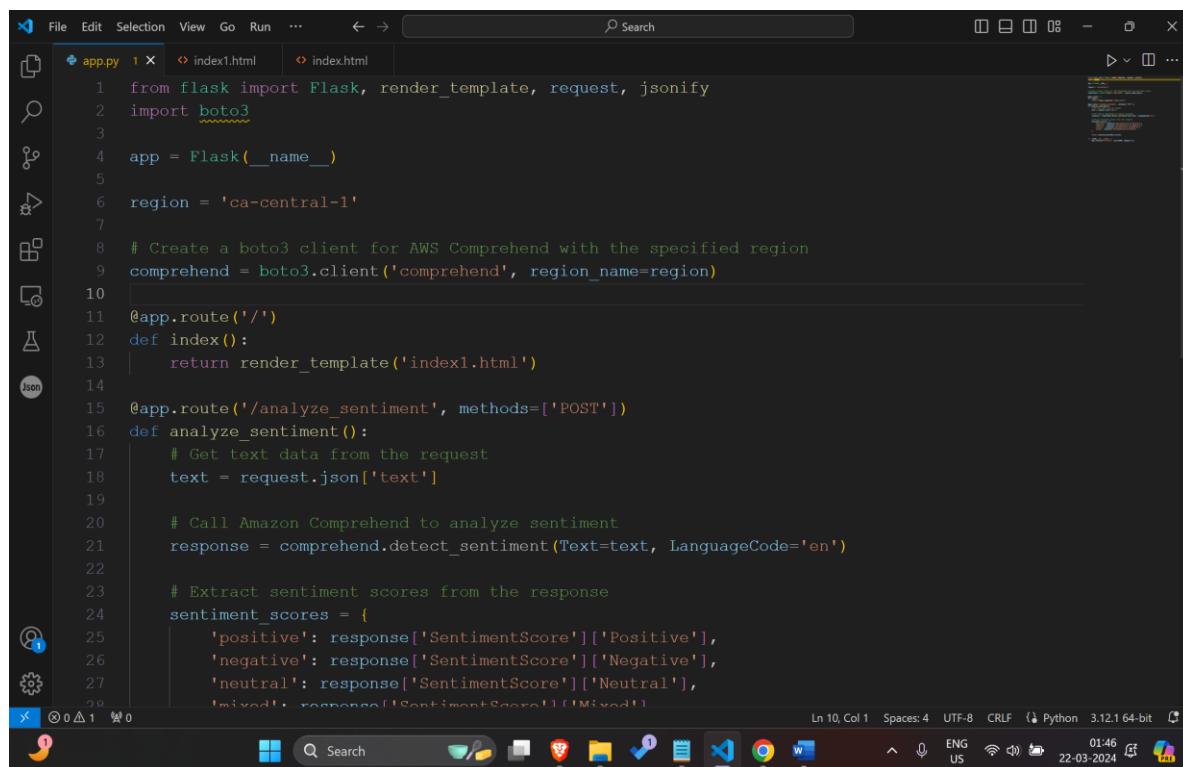
```

Similarly, we create a flask application, which calls the amazon AWS comprehend and the sentiment detect method.

This directly uses Amazon AWS comprehend methods to analyze the texts. This is a fully managed AWS NLP service that offers text detection, key phrases, entities, parts of speech.

It basically provides us the pre – trained models with APIs that are easier to use for text analyzation.

Below is the flask code:



```

File Edit Selection View Go Run ... ⏪ ⏩ Search
app.py 1 x index1.html index.html
1 from flask import Flask, render_template, request, jsonify
2 import boto3
3
4 app = Flask(__name__)
5
6 region = 'ca-central-1'
7
8 # Create a boto3 client for AWS Comprehend with the specified region
9 comprehend = boto3.client('comprehend', region_name=region)
10
11 @app.route('/')
12 def index():
13     return render_template('index1.html')
14
15 @app.route('/analyze_sentiment', methods=['POST'])
16 def analyze_sentiment():
17     # Get text data from the request
18     text = request.json['text']
19
20     # Call Amazon Comprehend to analyze sentiment
21     response = comprehend.detect_sentiment(Text=text, LanguageCode='en')
22
23     # Extract sentiment scores from the response
24     sentiment_scores = {
25         'positive': response['SentimentScore']['Positive'],
26         'negative': response['SentimentScore']['Negative'],
27         'neutral': response['SentimentScore']['Neutral'],
28         'mixed': response['SentimentScore']['Mixed']}
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
99

```

Ln 10, Col 1 Spaces: 4 UTF-8 CRLF Python 3.12.1 64-bit

01:46 22-03-2024

```
from flask import Flask, render_template, request, jsonify
import boto3

app = Flask(__name__)

region = 'ca-central-1'

# Create a boto3 client for AWS Comprehend with the specified region
comprehend = boto3.client('comprehend', region_name=region)

@app.route('/')
def index():
    return render_template('index1.html')

@app.route('/analyze_sentiment', methods=['POST'])
def analyze_sentiment():
    # Get text data from the request
    text = request.json['text']

    # Call Amazon Comprehend to analyze sentiment
    response = comprehend.detect_sentiment(Text=text, LanguageCode='en')

    # Extract sentiment scores from the response
    sentiment_scores = {
        'positive': response['SentimentScore']['Positive'],
        'negative': response['SentimentScore']['Negative'],
        'neutral': response['SentimentScore']['Neutral'],
        'mixed': response['SentimentScore']['Mixed']
    }

    return jsonify(sentiment_scores)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

This code includes a route that is basically used for analyzing sentiment using amazon AWS comprehend.

from flask import Flask, render\_template, request, jsonify: This line imports necessary modules from the Flask library. Here, we import Flask (the main Flask class), render\_template (for rendering HTML templates), request (for accessing request data), and jsonify (for creating JSON responses).

import boto3: We use boto3 to interact with AWS services.

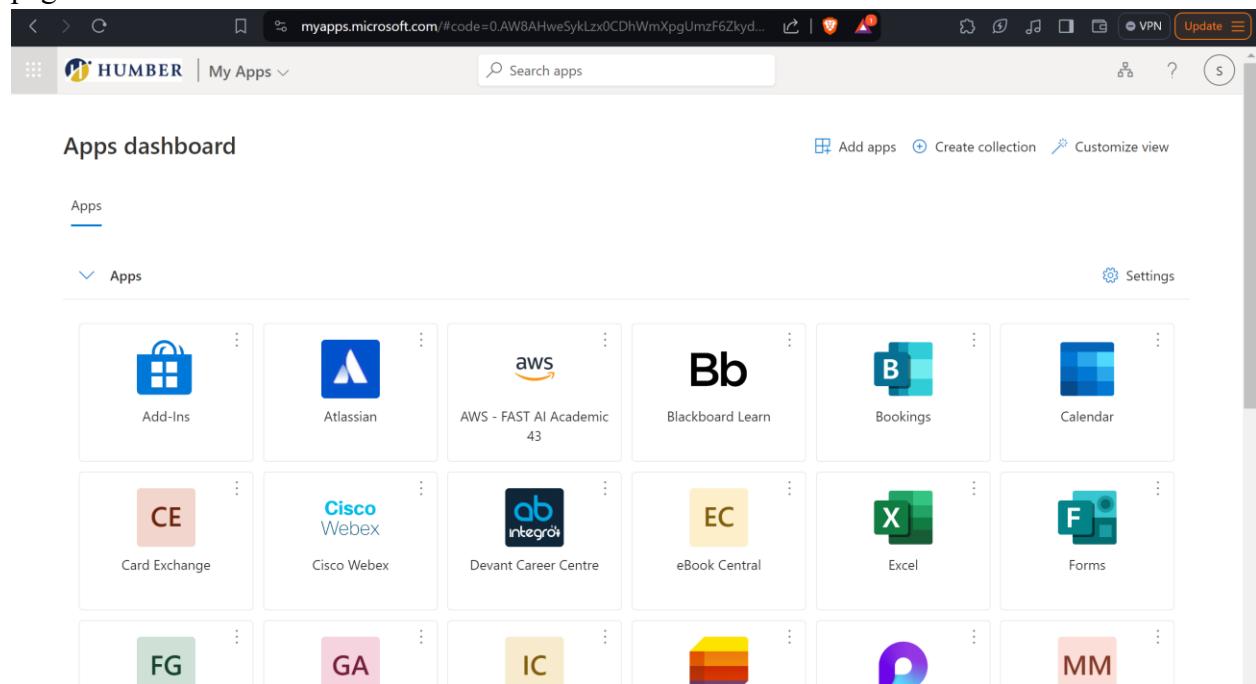
app = Flask(\_\_name\_\_): This line creates a Flask application instance named app. In this case, it represents the name of the script (`__main__` when executed directly).

region = 'ca-central-1': This line of code defines the AWS region (ca-central-1) where the AWS Comprehend service is located.

comprehend = boto3.client('comprehend', region\_name=region): This line creates a boto3 client for the AWS Comprehend service. This client object will allow us to call methods on the AWS Comprehend service.

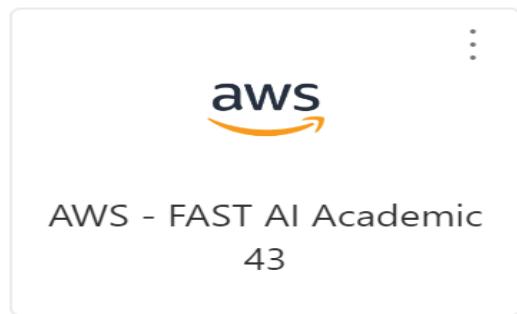
### Setting up Amazon instance

Firstly we will login to our Microsoft.myapps.com and login with our student id. You will see a page like this:



The screenshot shows the Microsoft myapps dashboard. At the top, there is a header bar with a back arrow, forward arrow, refresh button, a search bar containing 'myapps.microsoft.com/#code=0.AW8AHweSykLzx0CDhWmXpgUmzF6Zkyd...', and various system icons. Below the header, the title 'HUMBER | My Apps' is displayed next to a search bar labeled 'Search apps'. On the right side of the header, there are buttons for 'Add apps', 'Create collection', and 'Customize view'. The main area is titled 'Apps dashboard' and has a subtitle 'Apps'. It features a grid of app tiles. The first row contains tiles for 'Add-Ins', 'Atlassian', 'AWS - FAST AI Academic 43', 'Blackboard Learn', 'Bookings', and 'Calendar'. The second row contains tiles for 'Card Exchange', 'Cisco Webex', 'Devant Career Centre', 'eBook Central', 'Excel', and 'Forms'. The third row contains tiles for 'FG', 'GA', 'IC', a yellow square icon, 'P', and 'MM'. Each tile has a three-dot menu icon in the top right corner.

Click on AWS- FAST AI Academic and you will see a page similar to AWS learner's lab.



A screenshot of the AWS Console Home page. The top navigation bar shows the URL "ca-central-1.console.aws.amazon.com/console/home?region=ca-central-1" and the region "Central". The sidebar on the left lists "Recently visited" services: EC2, Amazon SageMaker, S3, CloudWatch, Lambda, Amazon Transcribe, and Amazon Translate. The main content area shows the "Applications" section, which is currently empty. It includes a "Create application" button and a search bar for "Find applications". The bottom of the screen shows the Windows taskbar with various pinned icons and the system tray.

Now once we set ourselves up, Make sure we are in the ‘Central’ Region and click on EC2 instance to create our instance.

Below is the page you will see to create an instance.

The screenshot shows the AWS EC2 Instances page. The left sidebar is collapsed. The main area displays a table with the following columns: Name, Instance ID, Instance state, Instance type, Status check, and Alarm status. A search bar at the top allows filtering by attribute or tag. A large button labeled "Launch instances" is prominently displayed. Below the table, a message states "No instances" and "You do not have any instances in this region".

The screenshot shows the "Launch instance" wizard. The first step, "Select an instance", has a "Name" field set to "myproject". The second step, "Application and OS Images (Amazon Machine Image)", shows a search bar and a "Quick Start" section with icons for Amazon Linux, Ubuntu, Windows, Red Hat, and SUSE Linux. The third step, "Summary", shows a "Launch instance" button. The bottom navigation bar includes CloudShell, Feedback, and a list of open tabs.

Instance type:

▼ Instance type [Info](#) | [Get advice](#)

Instance type

**t2.micro** Free tier eligible

Family: t2 1 vCPU 1 GiB Memory Current generation: true

On-Demand RHEL base pricing: 0.0728 USD per Hour

On-Demand Windows base pricing: 0.0174 USD per Hour

On-Demand SUSE base pricing: 0.0128 USD per Hour

On-Demand Linux base pricing: 0.0128 USD per Hour

All generations

[Compare instance types](#)

**Additional costs apply for AMIs with pre-installed software**

In the network settings change the subnet type to public. By default it will be private subnet.

Click on edit network settings and you will see a dropdown where you can find different subnets.

Make sure we have to set it to public subnet: sbn-fast-ai-academic-1-public-ca-central-1a

▼ Network settings [Info](#)

VPC - required [Info](#)

vpc-02dbd36fb74992622 (aws-controltower-VPC)  
172.31.0.0/16

Subnet [Info](#)

subnet-0f604b9157a24f883  
sbn-fast-ai-academic-43-public-ca-central-1a  
VPC: vpc-02dbd36fb74992622 Owner: 085812980140  
Availability Zone: ca-central-1a IP addresses available: 4091  
CIDR: 172.31.144.0/20

Auto-assign public IP [Info](#)

Disable

Firewall (security groups) [Info](#)

Assign EC2 role: fast-ai-academic-1-Student-EC2

As we scroll down, we will find IAM instance profile, make sure we have it in student

ca-central-1.console.aws.amazon.com/ec2/home?region=ca-central-1#LaunchInstances:

Services: EC2, CloudWatch, Lambda

Advanced details [Info](#)

Domain join directory [Info](#)

Select [Create new directory](#)

IAM instance profile [Info](#)

Select [Create new IAM profile](#)

humber-fast-ai-academic-43-foundation-iam-ec2-role-AzureFacultyInstanceProfile-NeUGcrxWEdl2  
arn:aws:iam::085812980140:instance-profile/humber-fast-ai-academic-43-foundation-iam-ec2-role-AzureFacultyInstanceProfile-NeUGcrxWEdl2

humber-fast-ai-academic-43-foundation-iam-ec2-role-AzureStudentInstanceProfile-jjNMHQNgr0LL  
arn:aws:iam::085812980140:instance-profile/humber-fast-ai-academic-43-foundation-iam-ec2-role-AzureStudentInstanceProfile-jjNMHQNgr0LL

Summary

Cancel [Launch instance](#) Review commands

CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Cloud Shell 1°C Cloudy 14:01 19-03-2024 ENG US

Once, we set these settings, we scroll to the instance network settings -> inbound security group rules

The screenshot shows the AWS Control Tower Network settings configuration page. At the top, there's a navigation bar with the AWS logo, Services (dropdown), Search (input field), and [Alt+S] keybinding. Below the navigation is a header with EC2, CloudWatch, and Lambda icons.

The main content area is titled "Network settings" with an "Info" link. It includes the following sections:

- VPC - required** (Info): A dropdown menu showing "vpc-02dbd36fb74992622 (aws-controltower-VPC)" and "172.31.0.0/16". To the right is a "Create new VPC" button.
- Subnet** (Info): A dropdown menu showing "subnet-0f604b9157a24f883" and details: "sbn-fast-ai-academic-43-public-ca-central-1a", "VPC: vpc-02dbd36fb74992622", "Owner: 085812980140", "Availability Zone: ca-central-1a", "IP addresses available: 4091", and "CIDR: 172.31.144.0/20". To the right is a "Create new subnet" button.
- Auto-assign public IP** (Info): A dropdown menu set to "Disable".
- Firewall (security groups)** (Info): A section explaining security groups and their purpose. It includes two radio buttons: "Create security group" (selected) and "Select existing security group".
- Security group name - required**: An input field containing "launch-wizard-1".

Below this is a section titled "Inbound Security Group Rules" with a "Remove" button. It lists one rule:

Type	Protocol	Port range
ssh	TCP	22

Details for the rule:

Source type	Source	Description - optional
Anywhere	0.0.0.0/0	e.g. SSH for admin desktop

A warning message at the bottom states: "⚠️ Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only." with a close button.

Once you land in the screen shown in the screenshot above, click below on the source type and click on HTTP. Once you click on HTTP the source will be created that is port 80.

We can also do this later after creating the instance, but to make things easier, we will create our port here itself to connect it with our web application.

The screenshot shows the configuration of a security group rule. At the top, there is a header for "Security group rule 2 (TCP, 80)". On the right, there is a "Remove" button. Below the header, there are several input fields and buttons:

- Type**: A dropdown menu set to "HTTP".
- Protocol**: A dropdown menu set to "TCP".
- Port range**: A text input field containing "80".
- Source type**: A dropdown menu set to "Custom".
- Source**: A text input field with a placeholder "Add CIDR, prefix list or security group".
- Description - optional**: A text input field with placeholder text "e.g. SSH for admin desktop".
- Add security group rule**: A button at the bottom left.

Here we need to have both the ports, one which is SSH that is connected to our terminal and another one which will be connected to our web application.

The screenshot shows the "Inbound Security Group Rules" section. It lists two rules:

- ▶ Security group rule 1 (TCP, 22)
- ▶ Security group rule 2 (TCP, 80)

On the right, there are two "Remove" buttons, one for each rule. At the bottom, there is a "Add security group rule" button.

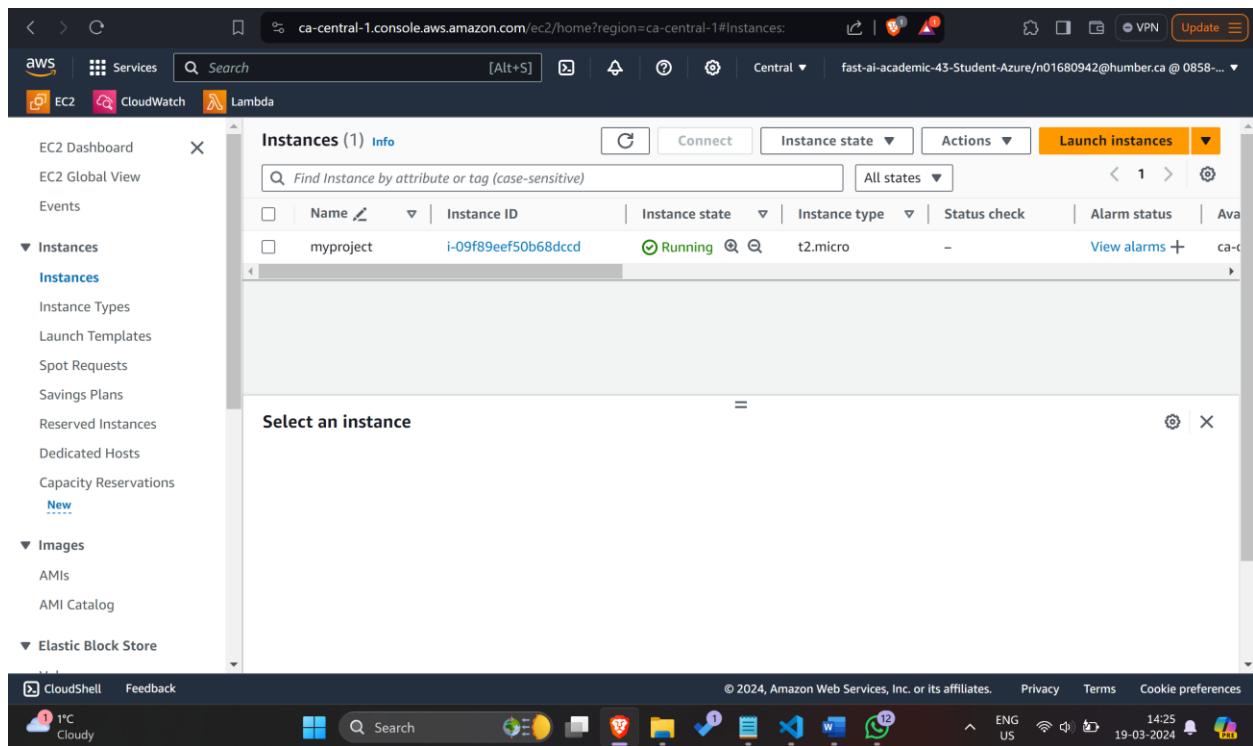
Once all the set-up is done, click on launch the instance.

The screenshot shows the "Launch log" for an instance. It displays a green success message:

Success  
Successfully initiated launch of instance (i-09f89eef50b68dcd)

Below the message, there is a link "▶ Launch log".

You wont be able to see the instance immediately as it takes time. But once you refresh the page, you should see the screen like below:



## Setting Up Cloud9:

Open the clou9 environment and you create an environment with new ec2 instance. Make sure you are again in ca-central and in the network settings you have to choose SSH connection. We can also do this through putty or SSH.

### Network settings Info

#### Connection

How your environment is accessed.

##### AWS Systems Manager (SSM)

Accesses environment via SSM without opening inbound ports (no ingress).

##### Secure Shell (SSH)

Accesses environment directly via SSH, opens inbound ports.

### ► VPC settings Info

Make sure the ARN is in student.

The screenshot shows the AWS Cloud9 Environments page. At the top, there are buttons for 'Delete', 'View details', 'Open in Cloud9', and 'Create environment'. Below this is a search bar with the placeholder 'My environments'. A table lists the environment details:

Name	Cloud9 IDE	Environment type	Connection	Permission	Owner ARN
<a href="#">myappproject</a>	<a href="#">Open</a>	EC2 instance	Secure Shell (SSH)	Owner	arn:aws:sts::085812980140:assumed-role/fast-ai-academic-43-Student-Azure/n01680942@humber.ca

Once you set the environment ready, open the ide. This is the OS of the server.

Libraries Installed:

Python3

Flask

Boto3

Web servers:Nginx and Gunicorn for handling requests.

Firstly, You will be landed in a page shown below:

The screenshot shows a terminal window with an SSH session to an EC2 instance. The session is titled 'ec2-user@ip-172-31-152-x'. The user is prompted to add a new host key fingerprint. They accept the addition and enter their password. The session then runs a command to change file permissions. Finally, the user creates a directory named 'project' and attempts to upgrade pip, which fails because Python is not found.

```
fast-ai-academic-43-Student-Azure:~/environment $ ssh -i pro.pem ec2-user@99.79.147.154
The authenticity of host '99.79.147.154 (99.79.147.154)' can't be established.
ED25519 key fingerprint is SHA256:McUNownSZHnNzCJaqbSFsU+fZZZ/bM9KXtGHyvwBU.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '99.79.147.154' (ED25519) to the list of known hosts.
@       WARNING: UNPROTECTED PRIVATE KEY FILE!
@-----[REDACTED]-----@       WARNING: UNPROTECTED PRIVATE KEY FILE!
@-----[REDACTED]-----@       WARNING: UNPROTECTED PRIVATE KEY FILE!
@-----[REDACTED]-----@       WARNING: UNPROTECTED PRIVATE KEY FILE!
Permissions 0644 for 'pro.pem' are too open.
It is required that your private key files are NOT accessible by others.
This private key will be ignored.
Load key "pro.pem": bad permissions
ec2-user@99.79.147.154: Permission denied (publickey,gssapi-keyex,gssapi-with-mic).
fast-ai-academic-43-Student-Azure:~/environment $ chmod 0400 pro.pem
fast-ai-academic-43-Student-Azure:~/environment $ ssh -i pro.pem ec2-user@99.79.147.154
,
  _###_          Amazon Linux 2023
  ~ \###\
  ~  \##\
  ~   \'#_
  ~    \#_
  ~     \'#_
  ~      https://aws.amazon.com/linux/amazon-linux-2023
  ~     V-' .->
  ~    /`_
  ~   /`_
  ~  /`_
  ~ /`_
[ec2-user@ip-172-31-152-188 ~]$ mkdir project
[ec2-user@ip-172-31-152-188 ~]$ ls
project
[ec2-user@ip-172-31-152-188 ~]$ python -m pip install --upgrade pip
-bash: python: command not found
```

You will be seeing amazon Linux. This is where our backend, proxy and the rest half of the project Is done.

Click on the folder icon in the top left corner and upload the .pem file. We need to remember that for SSH .pem file works and for putty we need to download .ppk file.

Once you are in this page, type in a ssh command to connect to your ec2 instance.

Command: ssh -I filename.pem username@IP address

You will be prompted with a permission denied and warning message.

Just type in ‘chmod 0400 filename.pem. This is a way to give used to set permissions on a file in Unix-like operating systems, such as Amazon Linux.

In the case of chmod 0400, it sets the following permissions:

0 for the owner: No permissions.

4 for the group: Read permission.

0 for others: No permissions.

Again run the SSH command mentioned above and it will open up Amazon linux connected to our instance.

```
ec2-user@99.79.147.154: Permission denied (publickey,gssapi-keyex,gssapi-with-mic).
fast-ai-academic-43-Student-Azure:~/environment $ chmod 0400 pro.pem
fast-ai-academic-43-Student-Azure:~/environment $ ssh -i pro.pem ec2-user@99.79.147.154
,
      #
~\_ ####_          Amazon Linux 2023
~~ \#####\
~~  \###|
~~   \#/ __ https://aws.amazon.com/linux/amazon-linux-2023
~~    V~' '-'>
~~     /
~~.. _/
~~/_/ _/
~/m/'
```

Now we create a directory with mkdir command:

```
_/"/"
[ec2-user@ip-172-31-152-188 ~]$ mkdir project
[ec2-user@ip-172-31-152-188 ~]$ ls
project
```

This directory is to store and run our python application.

Now before uploading our python file, we need to make sure that we install all the libraries we are using: boto3, flask and python3.

Using curl command to install pip

```
[ec2-user@ip-172-31-152-188 ~]$ curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
[ec2-user@ip-172-31-152-188 ~]$ curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
% Total    % Received % Xferd  Average Speed   Time     Time      Current
          Dload  Upload Total Spent   Left Speed
100 2574k  100 2574k    0     0  6363k      0 ---:--- ---:--- ---:--- 6355k
[ec2-user@ip-172-31-152-188 ~]$ python3 get-pip.py
Defaulting to user installation because normal site-packages is not writeable
Collecting pip
  Downloading pip-24.0-py3-none-any.whl.metadata (3.6 kB)
Collecting wheel
  Downloading wheel-0.43.0-py3-none-any.whl.metadata (2.2 kB)
  Downloading pip-24.0-py3-none-any.whl (2.1 MB)
    2.1/2.1 MB 10.5 MB/s eta 0:00:00
  Downloading wheel-0.43.0-py3-none-any.whl (65 kB)
    65.8/65.8 kB 9.8 MB/s eta 0:00:00
Installing collected packages: wheel, pip
```

Once installation is done, install python. For SSH the commands are slightly different as it's a red hat Enterprise linux.

```
[ec2-user@ip-172-31-152-188 ~]$ python3 -m pip --version
pip 24.0 from /home/ec2-user/.local/lib/python3.9/site-packages/pip (python 3.9)
[ec2-user@ip-172-31-152-188 ~]$ sudo yum install python3
Last metadata expiration check: 0:24:37 ago on Thu Mar 21 22:35:45 2024.
Package python3-3.9.16-1.amzn2023.0.6.x86_64 is already installed.
Dependencies resolved.
```

You will see the downloading dependencies.

```
[ec2-user@ip-172-31-152-188 ~]$ sudo yum install python3-pip
Last metadata expiration check: 0:24:48 ago on Thu Mar 21 22:35:45 2024.
Dependencies resolved.
=====
Package           Architecture Version       Repository      Size
=====
Installing:
python3-pip      noarch      21.3.1-2.amzn2023.0.7  amazonlinux   1.8 M
Installing weak dependencies:
libcrypt-compat  x86_64      4.4.33-7.amzn2023  amazonlinux   92 k

Transaction Summary
=====
Install 2 Packages

Total download size: 1.9 M
Installed size: 11 M
Is this ok [y/N]: pip list
Is this ok [y/N]: y
Downloading Packages:
(1/2): libcrypt-compat-4.4.33-7.amzn2023.x86_64.rpm          921 kB/s |  92 kB   00:00
(2/2): python3-pip-21.3.1-2.amzn2023.0.7.noarch.rpm          15 MB/s | 1.8 MB   00:00
Total
Running transaction check
=====

```

This is optional but for my reference, I am checking list of pip or installations I have in my system, to check it you type in ‘pip list’.

```
Complete!
[ec2-user@ip-172-31-152-188 ~]$ pip list
[ec2-user@ip-172-31-152-188 ~]$ pip list | grep boto3
[ec2-user@ip-172-31-152-188 ~]$ pip list | grep python
dbus-python          1.2.18
python-daemon        2.3.0
python-dateutil      2.8.1
systemd-python        235
[ec2-user@ip-172-31-152-188 ~]$ pip list | grep python3
[ec2-user@ip-172-31-152-188 ~]$ python3 --version
Python 3.9.16
```

Now once you install python, we need to install boto3. I have installed it by going to the folder I previously created. But you can also download it anywhere in the IDE since it will be installed globally.

```
[ec2-user@ip-172-31-152-188 ~]$ cd project
[ec2-user@ip-172-31-152-188 project]$ cd ..
[ec2-user@ip-172-31-152-188 ~]$ pip install boto3
Defaulting to user installation because normal site-packages is not writeable
Collecting boto3
  Downloading boto3-1.34.68-py3-none-any.whl.metadata (6.6 kB)
Collecting botocore<1.35.0,>=1.34.68 (from boto3)
  Downloading botocore-1.34.68-py3-none-any.whl.metadata (5.7 kB)
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in /usr/lib/python3.9/site-packages (from boto3)
Collecting s3transfer<0.11.0,>=0.10.0 (from boto3)
  Downloading s3transfer-0.10.1-py3-none-any.whl.metadata (1.7 kB)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /usr/lib/python3.9/site-packages
Requirement already satisfied: urllib3<1.27,>=1.25.4 in /usr/lib/python3.9/site-packages (from boto3)
Requirement already satisfied: six>=1.5 in /usr/lib/python3.9/site-packages (from python-dateutil)
  Downloading boto3-1.34.68-py3-none-any.whl (139 kB)
139.3/139.3 kB 1.6 MB/s eta 0:00:00
```

Now once this has been done, install flask for our application.

```
[ec2-user@ip-172-31-152-188 ~]$ pip install flask
Defaulting to user installation because normal site-packages is not writeable
Collecting flask
  Downloading flask-3.0.2-py3-none-any.whl.metadata (3.6 kB)
Collecting Werkzeug>=3.0.0 (from flask)
  Downloading werkzeug-3.0.1-py3-none-any.whl.metadata (4.1 kB)
Collecting Jinja2>=3.1.2 (from flask)
  Downloading Jinja2-3.1.3-py3-none-any.whl.metadata (3.3 kB)
Collecting itsdangerous>=2.1.2 (from flask)
  Downloading itsdangerous-2.1.2-py3-none-any.whl.metadata (2.9 kB)
Collecting click>=8.1.3 (from flask)
  Downloading click-8.1.7-py3-none-any.whl.metadata (3.0 kB)
Collecting blinker>=1.6.2 (from flask)
  Downloading blinker-1.7.0-py3-none-any.whl.metadata (1.9 kB)
Collecting importlib-metadata>=3.6.0 (from flask)
  Downloading importlib_metadata-7.1.0-py3-none-any.whl.metadata (4.7 kB)
Collecting zipp>=0.5 (from importlib-metadata>=3.6.0->flask)
  Downloading zipp-3.18.1-py3-none-any.whl.metadata (3.5 kB)
Collecting MarkupSafe>=2.0 (from Jinja2>=3.1.2->flask)
  Downloading MarkupSafe-2.1.5-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (101 kB)
  Downloading flask-3.0.2-py3-none-any.whl (101 kB)
    101.3/101.3 kB 1.9 MB/s eta 0:00:00
  Downloading blinker-1.7.0-py3-none-any.whl (13 kB)
  Downloading click-8.1.7-py3-none-any.whl (97 kB)
    97.9/97.9 kB 3.7 MB/s eta 0:00:00
  Downloading importlib_metadata-7.1.0-py3-none-any.whl (24 kB)
  Downloading itsdangerous-2.1.2-py3-none-any.whl (15 kB)
  Downloading Jinja2-3.1.3-py3-none-any.whl (133 kB)
    133.2/133.2 kB 5.0 MB/s eta 0:00:00
```

Now once we install all the libraries, create a file, in my case its app.py and copy the flask code into this folder and save it. You can use the commands as mentioned in the below picture:

```
[ec2-user@ip-172-31-152-188 ~]$ cd project
[ec2-user@ip-172-31-152-188 project]$ sudo nano app.py
[ec2-user@ip-172-31-152-188 project]$ cat app.py
```

The cat command shows you what's actually in the file.

```
[ec2-user@ip-172-31-152-188 project]$ cat app.py
from flask import Flask, render_template, request, jsonify
import boto3

app = Flask(__name__)

region = 'ca-central-1'

# Create a boto3 client for AWS Comprehend with the specified region
comprehend = boto3.client('comprehend', region_name=region)

@app.route('/')
def index():
    return render_template('index1.html')

@app.route('/analyze_sentiment', methods=['POST'])
def analyze_sentiment():
    # Get text data from the request
    text = request.json['text']

    # Call Amazon Comprehend to analyze sentiment
    response = comprehend.detect_sentiment(Text=text, LanguageCode='en')

    # Extract sentiment scores from the response
    sentiment_scores = {
        'positive': response['SentimentScore']['Positive'],
        'negative': response['SentimentScore']['Negative'],
        'neutral': response['SentimentScore']['Neutral'],
    }

    return jsonify(sentiment_scores)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)

[ec2-user@ip-172-31-152-188 project]$ sudo nano app.py
[ec2-user@ip-172-31-152-188 project]$ python3 app.py
* Serving Flask app 'app'
```

Once your code is shown or printed on the screen, save the nano file and exit by pressing ctrl + x.

Now run the python file(app.py)

This will connect to our flask application and we will see the below screen.

```
* Serving Flask app 'app'  
* Debug mode: on  
WARNING: This is a development server. Do not use it i  
* Running on all addresses (0.0.0.0)  
* Running on http://127.0.0.1:5000  
* Running on http://172.31.152.188:5000  
Press CTRL+C to quit  
* Restarting with stat  
* Debugger is active!  
* Debugger PIN: 632-901-806
```

Here we can see that its running on the addresses its running.

We choose a web browser of our own choice to check if the application is running or not.

For that, copy the public IP address of your ec2 instance with the port number(in our case its 5000 since it's the standard port number for flask).

‘IP address:5000’. This is how we need to check if our application has been connected or not.

Now we might not be seeing the application as we still did not connect our front end. For this, within the project folder create another folder named templates and then save the html file in it.

```
[ec2-user@ip-172-31-152-188 project]$ cd ..  
[ec2-user@ip-172-31-152-188 ~]$ cd project  
[ec2-user@ip-172-31-152-188 project]$ mkdir templates  
[ec2-user@ip-172-31-152-188 project]$ ls  
app.py  templates
```

Since flask looks for files only in templates we are explicitly creating this folder and putting our html file to run through it.

Run the command sudo nano... to create an empty file in the templates folder and copy paste the html code we have in our system.

```
[ec2-user@ip-172-31-152-188 project]$ cd templates  
[ec2-user@ip-172-31-152-188 templates]$ sudo nano index1.html  
[ec2-user@ip-172-31-152-188 templates]$ cat index1.html  
<!DOCTYPE html>
```

Once you perform cat command, we should be able to see the html code in the ide.

```
[ec2-user@ip-172-31-152-188 templates]$ cat index1.html  
<!DOCTYPE html>  
<html lang="en">  
  <head>  
    <meta charset="UTF-8">  
    <meta name="viewport" content="width=device-width, initial-scale=1.0">  
    <title>Sentiment Analysis</title>  
  </head>  
  <body>  
    <h1>Sentiment Analysis</h1>  
    <form id="sentimentForm">  
      <label for="textInput">Enter Text to Analyze:</label><br>  
      <textarea id="textInput" name="textInput" rows="4" cols="50"></textarea><br>  
      <button type="submit">Analyze Sentiment</button>  
    </form>  
    <div id="result"></div>  
  
    <script>  
      document.getElementById('sentimentForm').addEventListener('submit', function(event) {  
        event.preventDefault();  
        const text = document.getElementById('textInput').value;  
        analyzeSentiment(text);  
      });
```

```

        function analyzeSentiment(text) {
            fetch('/analyze_sentiment', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify({ text: text })
            })
            .then(response => response.json())
            .then(data => {
                displayResults(data);
            })
            .catch(error => {
                console.error('Error:', error);
            });
        }

        function displayResults(sentimentScores) {
            const resultDiv = document.getElementById('result');
            resultDiv.innerHTML =
                `

## Sentiment Analysis Results:


                <p>Positive Score: ${sentimentScores.positive}</p>
                <p>Negative Score: ${sentimentScores.negative}</p>
                <p>Neutral Score: ${sentimentScores.neutral}</p>
            `;
        }
    </script>
</body>
</html>
```

Now you Run the python file and reload your web browser page, you should be able to see the html format in it.

```

app.py  templates
[ec2-user@ip-172-31-152-188 project]$ python3 app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment!
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.31.152.188:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 632-901-806
```



## Sentiment Analysis

Enter Text to Analyze:



Analyze Sentiment

### Sentiment Analysis Results:

Positive Score: 0.9972243309020996

Negative Score: 0.0001463730732211843

Neutral Score: 0.002212009159848094



The above is my simple html format where I am showing the sentiment results along with its scores.



## Sentiment Analysis

Enter Text to Analyze:

1

Analyze Sentiment

### Sentiment Analysis Results:

Positive Score: 0.9988184571266174

Negative Score: 0.00008445294224657118

Neutral Score: 0.001074616564437747



As you can see we are able to see the sentiment score of how much percentage is sentence is.

Since it is a positive statement, the positive score is 0.99 which is 99% accurate.

### **Setting Up web server and running the application:**

Now once we are done with checking the backend, we now set up a web server and proxy server for our web applications to run.

Nginx: It is a web server that is used as a reverse proxy. It is used to handle the HTTP requests.

Gunicorn is a WSGI server which is used to serve python, flask web applications. It is an application server to handle the execution of the flask application files and handle multiple worker request processes. To be precise it is used to handle dynamic application content.

```
[ec2-user@ip-172-31-152-188 project]$ pip install gunicorn
Defaulting to user installation because normal site-packages is not writeable
Collecting gunicorn
  Downloading gunicorn-21.2.0-py3-none-any.whl.metadata (4.1 kB)
Collecting packaging (from gunicorn)
  Downloading packaging-24.0-py3-none-any.whl.metadata (3.2 kB)
  Downloading gunicorn-21.2.0-py3-none-any.whl (80 kB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 80.2/80.2 kB 1.7 MB/s eta 0:00:00
  Downloading packaging-24.0-py3-none-any.whl (53 kB)
    ━━━━━━━━━━━━━━━━ 53.5/53.5 kB 5.5 MB/s eta 0:00:00
Installing collected packages: packaging, gunicorn
Successfully installed gunicorn-21.2.0 packaging-24.0
[ec2-user@ip-172-31-152-188 project]$ cd ..
[ec2-user@ip-172-31-152-188 ~]$ ls
get-pip.py  project
```

Once you install Gunicorn install Nginx as well.

```
[ec2-user@ip-172-31-152-188 ~]$ pip install nginx
Defaulting to user installation because normal site-packages is not writeable
ERROR: Could not find a version that satisfies the requirement nginx (from versions: n
ERROR: No matching distribution found for nginx
[ec2-user@ip-172-31-152-188 ~]$ sudo yum install nginx
Last metadata expiration check: 1:42:23 ago on Thu Mar 21 22:35:45 2024.
Dependencies resolved.
=====
Package           Arch    Version            Repository      Size
=====
Installing:
nginx            x86_64  1:1.24.0-1.amzn2023.0.2  amazonlinux   32 k
Installing dependencies:
generic-logos-httd noarch  18.0.0-12.amzn2023.0.3  amazonlinux   19 k
gperftools-libs   x86_64  2.9.1-1.amzn2023.0.3  amazonlinux  308 k
libunwind          x86_64  1.4.0-5.amzn2023.0.2  amazonlinux   66 k
nginx-core         x86_64  1:1.24.0-1.amzn2023.0.2  amazonlinux  586 k
nginx-filesystem  noarch  1:1.24.0-1.amzn2023.0.2  amazonlinux   9.1 k
nginx-mimetypes   noarch  2.1.49-3.amzn2023.0.3  amazonlinux  21 k
Transaction Summary
=====
Install 7 Packages

Total download size: 1.0 M
Installed size: 3.4 M
Is this ok [y/N]: y
Downloading Packages:
(1/7): nginx-1.24.0-1.amzn2023.0.2.x86_64.rpm 532 kB/s | 32 kB  00:00
(2/7): gperftools-libs-2.9.1-1.amzn2023.0.3.x86_4.6 MB/s | 308 kB  00:00
(3/7): nginx-core-1.24.0-1.amzn2023.0.2.x86_64.r 7.8 MB/s | 586 kB  00:00
```

Once you installed the web server, we now have to start the Gunicorn application to see if the flask application is running on the web browser.

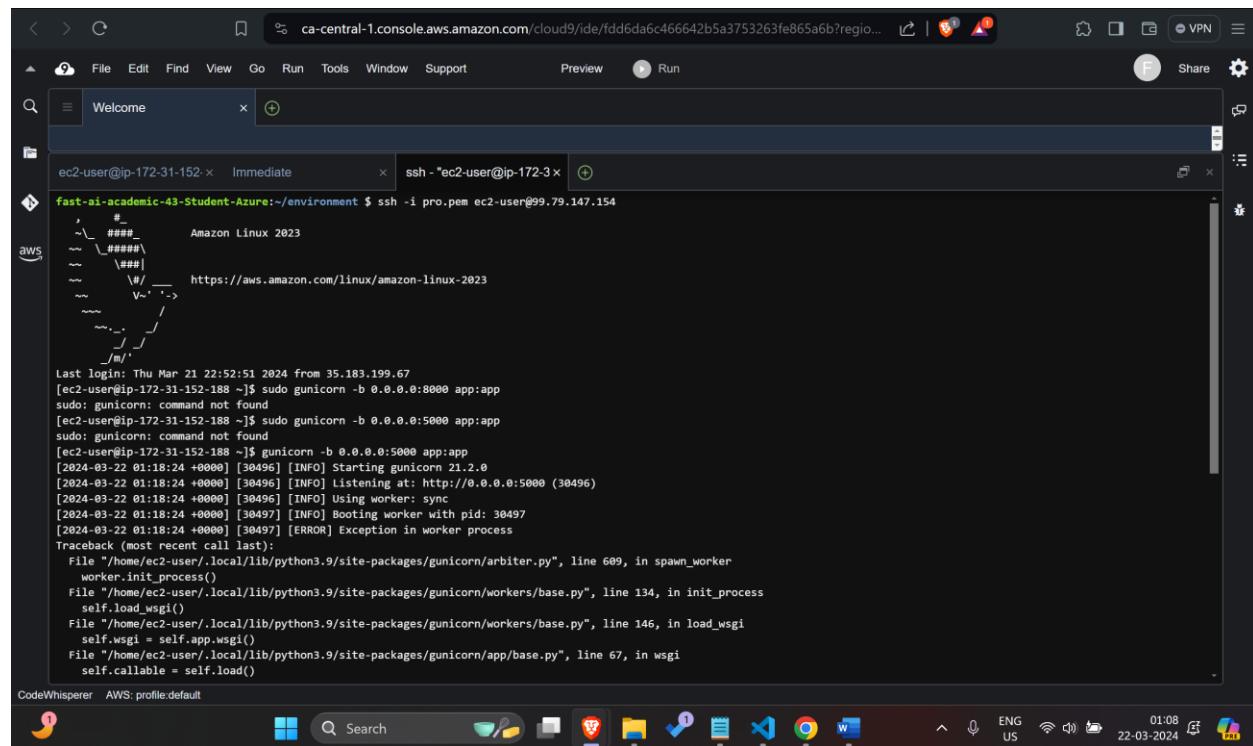
```
[ec2-user@ip-172-31-152-188 ~]$ cd project
[ec2-user@ip-172-31-152-188 project]$ ls
app.py  templates
[ec2-user@ip-172-31-152-188 project]$ gunicorn -b 0.0.0.0:5000 app:app
[2024-03-22 00:24:03 +0000] [28703] [INFO] Starting gunicorn 21.2.0
[2024-03-22 00:24:03 +0000] [28703] [INFO] Listening at: http://0.0.0.0:5000 (28703)
[2024-03-22 00:24:03 +0000] [28703] [INFO] Using worker: sync
[2024-03-22 00:24:03 +0000] [28704] [INFO] Booting worker with pid: 28704
^C[2024-03-22 00:24:43 +0000] [28703] [INFO] Handling signal: int
[2024-03-22 00:24:43 +0000] [28704] [INFO] Worker exiting (pid: 28704)
[2024-03-22 00:24:43 +0000] [28703] [INFO] Shutting down: Master
[ec2-user@ip-172-31-152-188 project]$ gunicorn -b 0.0.0.0:5000 app:app
[2024-03-22 00:24:56 +0000] [28753] [INFO] Starting gunicorn 21.2.0
[2024-03-22 00:24:56 +0000] [28753] [INFO] Listening at: http://0.0.0.0:5000 (28753)
[2024-03-22 00:24:56 +0000] [28753] [INFO] Using worker: sync
[2024-03-22 00:24:56 +0000] [28754] [INFO] Booting worker with pid: 28754
```

Run the sudo command and you will see the workers with PID. PID is nothing but a process ID who are listening(means waiting for the request) to perform the task.Now to make this work, we need someone to actually handle our requests. This is where Nginx comes into work.

Before starting it, we create a systemd: It is a tool that is called as a system service manager that allows for strict process management. Below we are creating a socket for incoming Gunicorn requests. Systemd will listen on this socket and start gunicorn automatically.

```
[2024-05-22 00:28:21 +0000] [28755] [INFO] Shutting down. Master
[ec2-user@ip-172-31-152-188 project]$ sudo nano /etc/systemd/system/project.service
[ec2-user@ip-172-31-152-188 project]$ pwd
/home/ec2-user/project
[ec2-user@ip-172-31-152-188 project]$ ls
__pycache__  app.py  templates
```

To make the Gunicorn running for the rest of the process, we open another terminal, connect our ec2 instance and run the gunicorn start command. Since this is used for our incoming requests, we keep this running in one terminal and run other parts in different terminal.



```
[2024-03-22 01:18:24 +0000] [30497] [INFO] Worker exiting (pid: 30497)
[2024-03-22 01:18:24 +0000] [30496] [ERROR] Worker (pid:30497) exited with code 3
[2024-03-22 01:18:24 +0000] [30496] [ERROR] Shutting down: Master
[2024-03-22 01:18:24 +0000] [30496] [ERROR] Reason: Worker failed to boot.
[ec2-user@ip-172-31-152-188 ~]$ cd project
[ec2-user@ip-172-31-152-188 project]$ gunicorn -b 0.0.0.0:5000 app:app
[2024-03-22 01:19:24 +0000] [30546] [INFO] Starting gunicorn 21.2.0
[2024-03-22 01:19:24 +0000] [30546] [INFO] Listening at: http://0.0.0.0:5000 (30546)
[2024-03-22 01:19:24 +0000] [30546] [INFO] Using worker: sync
[2024-03-22 01:19:24 +0000] [30547] [INFO] Booting worker with pid: 30547
```

Now we get back to our terminal 1 and type in the below commands where we are creating a unit file named project.service. Here, we define in the file that how the systemd should manage a process, in our case its flask or Nginx server.

```
[ec2-user@ip-172-31-152-188 project]$ sudo nano /etc/systemd/system/project.service  
[ec2-user@ip-172-31-152-188 project]$ sudo systemctl daemon-reload  
[ec2-user@ip-172-31-152-188 project]$ sudo systemctl daemon-reload  
[ec2-user@ip-172-31-152-188 project]$ sudo systemctl start project  
[ec2-user@ip-172-31-152-188 project]$ sudo systemctl enable project
```

In the file where we use the nano command, there are few things we need to learn about the code.

Unit Description: A brief description of the service.

Service Section: Defines the service-specific configuration, including the executable path (ExecStart), working directory (WorkingDirectory), user and group under which the service should run (User and Group), and other settings.

Dependencies: Specifies any dependencies that must be satisfied before the service can start (After, Requires, etc.).

Installation Section: Defines how the service should be installed and started automatically at boot (WantedBy, RequiredBy, etc.).

The .service extension indicates that the file is a systemd service unit file.

Furthermore, The command daemon - refers to a type of background process that runs continuously, providing specific services or functionality to the operating system or other programs.

Daemons typically do not have a user interface and operate without direct user interaction.

Now that we have set, run the port using curl command: curl 0.0.0.0:5000

```
[ec2-user@ip-172-31-152-188 project]$ curl 0.0.0.0:5000
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Sentiment Analysis</title>
</head>
<body>
    <h1>Sentiment Analysis</h1>
    <form id="sentimentForm">
        <label for="textInput">Enter Text to Analyze:</label><br>
        <textarea id="textInput" name="textInput" rows="4" cols="50"></textarea><br>
        <button type="submit">Analyze Sentiment</button>
    </form>
    <div id="result"></div>

    <script>
        document.getElementById('sentimentForm').addEventListener('submit', function(event) {
            event.preventDefault();
            const text = document.getElementById('textInput').value;
            analyzeSentiment(text);
        });
    </script>

```

By entering the curl command, it will open the html file ensuring that our files are connected to the server successfully.

Now we create a configuration file using sudo nano command. This is a configuration file for the Nginx server.

This file contains how the Nginx should run, its behaviour, security etc.

```
[ec2-user@ip-172-31-152-188 project]$ sudo nano /etc/nginx/conf.d/app.conf
[ec2-user@ip-172-31-152-188 project]$ sudo nano /etc/nginx/conf.d/app.conf
[ec2-user@ip-172-31-152-188 project]$ sudo nginx -t
```

The code:

```
server {
    listen 80;
    server_name 99.79.147.154;

    location / {
        proxy_pass http://0.0.0.0:5000;
        proxy_set_header Host $host;
```

```

proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_set_header X-Forwarded-Proto $scheme;
}

}

```

The listen and server name are used to configure a local machine. Once we save the file, we have to make sure that we enable and start the Nginx server.

By this, our server is connected to the website and can be opened on any web browser.

```

< > C   ca-central-1.console.aws.amazon.com/cloud9/ide/fdd6da6c466642b5a3753263fe865a6b?region=us-east-1
File Edit Find Go Run Tools Window Support Preview Run
F Share Settings
Welcome + 
ec2-user@ip-172-31-152-x Immediate ssh - *ec2-user@ip-172-31-152-x + 
```
    .catch(error => {
      console.error('Error:', error);
    });
}

function displayResults(sentimentScores) {
  const resultDiv = document.getElementById('result');
  resultDiv.innerHTML =
    `

## Sentiment Analysis Results:

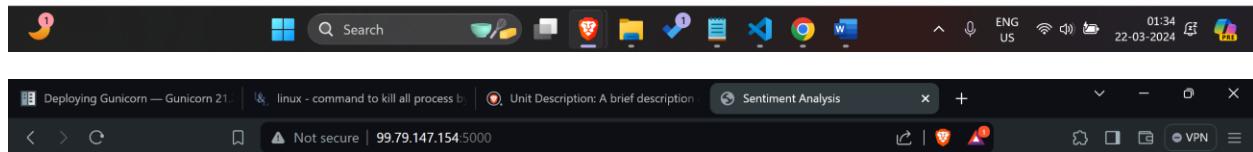
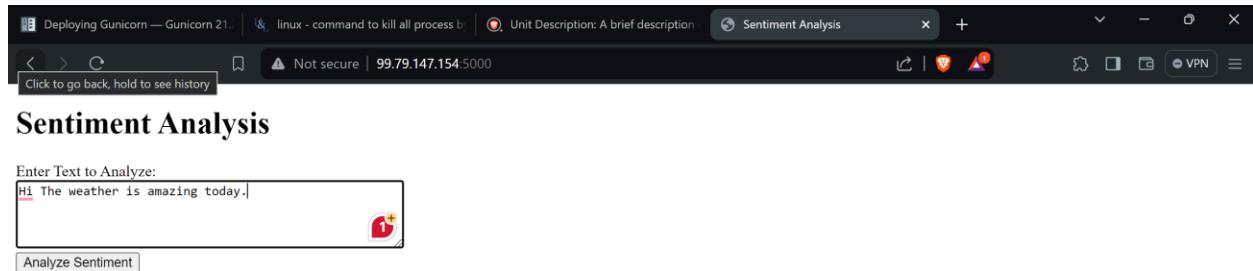

    <p>Positive Score: ${sentimentScores.positive}</p>
    <p>Negative Score: ${sentimentScores.negative}</p>
    <p>Neutral Score: ${sentimentScores.neutral}</p>
  `;
}
</script>
</body>
</html>
[ec2-user@ip-172-31-152-188 project]$ sudo nano /etc/nginx/conf.d/app.conf
[ec2-user@ip-172-31-152-188 project]$ sudo nano /etc/nginx/conf.d/app.conf
[ec2-user@ip-172-31-152-188 project]$ sudo nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
[ec2-user@ip-172-31-152-188 project]$ systemctl enable nginx.service
Failed to enable unit: Access denied
[ec2-user@ip-172-31-152-188 project]$ sudo systemctl enable nginx.service
Created symlink /etc/systemd/system/multi-user.target.wants/nginx.service → /usr/lib/systemd/system/nginx.service.
[ec2-user@ip-172-31-152-188 project]$ systemctl start nginx
Failed to start nginx.service: Access denied
See system logs and 'systemctl status nginx.service' for details.
[ec2-user@ip-172-31-152-188 project]$ sudo systemctl start nginx
[ec2-user@ip-172-31-152-188 project]$ sudo nano /etc/nginx/conf.d/app.conf
[ec2-user@ip-172-31-152-188 project]$ 
```
CodeWhisperer AWS profile=default
```
Search
ENG US 01:29 22-03-2024

```

Now to see our results, copy the IP address and the port number(IP address:5000) and run it on the browser.

Below are the results of each type of sentiment analysis of a statement and the scores showing what percentage of scores.

So if our statement is positive, the highest sentiment score will be positive else negative or else neutral.



## Sentiment Analysis

Enter Text to Analyze:

Hi The weather is amazing today.



Analyze Sentiment

### Sentiment Analysis Results:

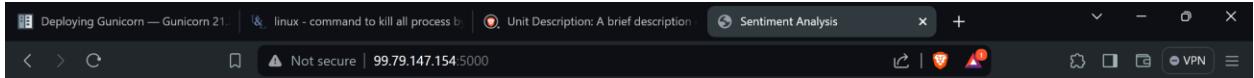
Positive Score: 0.9915598034858704

Negative Score: 0.0005385869881138206

Neutral Score: 0.007662855554372072



If the given statement is negative:



## Sentiment Analysis

Enter Text to Analyze:

The weather seems to be pretty bad today.



Analyze Sentiment

### Sentiment Analysis Results:

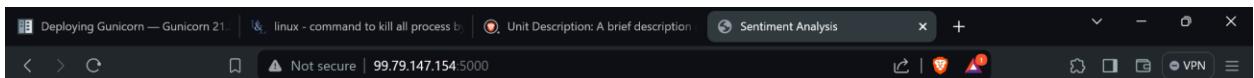
Positive Score: 0.002054785378277302

Negative Score: 0.9354740381240845

Neutral Score: 0.055993176996707916



Now lets check for the neutral statement:



## Sentiment Analysis

Enter Text to Analyze:

are you okay?



Analyze Sentiment

### Sentiment Analysis Results:

Positive Score: 0.03029191493988037

Negative Score: 0.017145387828350067

Neutral Score: 0.9524094462394714



References:

<https://docs.gunicorn.org/en/stable/deploy.html>

[https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/7/html/system\\_administrators\\_guide/chap-managing\\_services\\_with\\_systemd](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/system_administrators_guide/chap-managing_services_with_systemd)

Thank You.