ECE444-PRA5:
Deploying an ML APP to the Cloud

1. Learning objective:

The objective of this assignment is for students to get familiar with deploying machine learning (ML) models to the cloud using a cloud provider. We will focus on deploying a model using AWS Elastic Beanstalk, making the model accessible as a service using a REST API call.

2. Scenario:

Fake news has recently become a topic of concern as more of the information we receive about the world is delivered through the web. You, as a developer on the Fake News detection team at Not Fake News Co. have been tasked with building a barebones REST API that takes in a snippet of text (from a news article or source) and determines if it is considered fake news or not (by returning a 1 if it is Fake News, 0 otherwise).

3. Background

AWS Elastic Beanstalk (https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/Welcome.html) is an easy-to-use service for deploying and scaling web applications and services developed with Java, .NET, PHP, Node.js, Python, Ruby, Go, and Docker on familiar servers such as Apache, Nginx, Passenger, and IIS. scaling to application health monitoring. At the same time, you retain full control over the AWS resources powering your application and can access the underlying resources at any time.

4. Activities on AWS

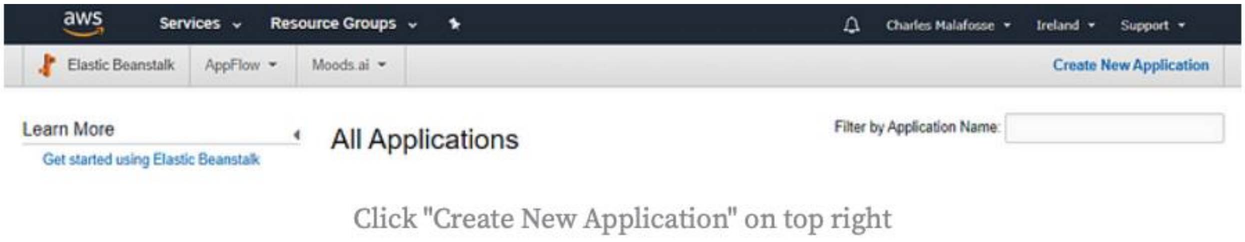
i. Sign-up for AWS free account

For this task, make sure you have registered an account with AWS, which will give you 750 Hours of compute for free per month, which should be more than enough for this lab. You can register for AWS here: https://aws.amazon.com/free/

ii. Create an App in AWS Elastic Beanstalk

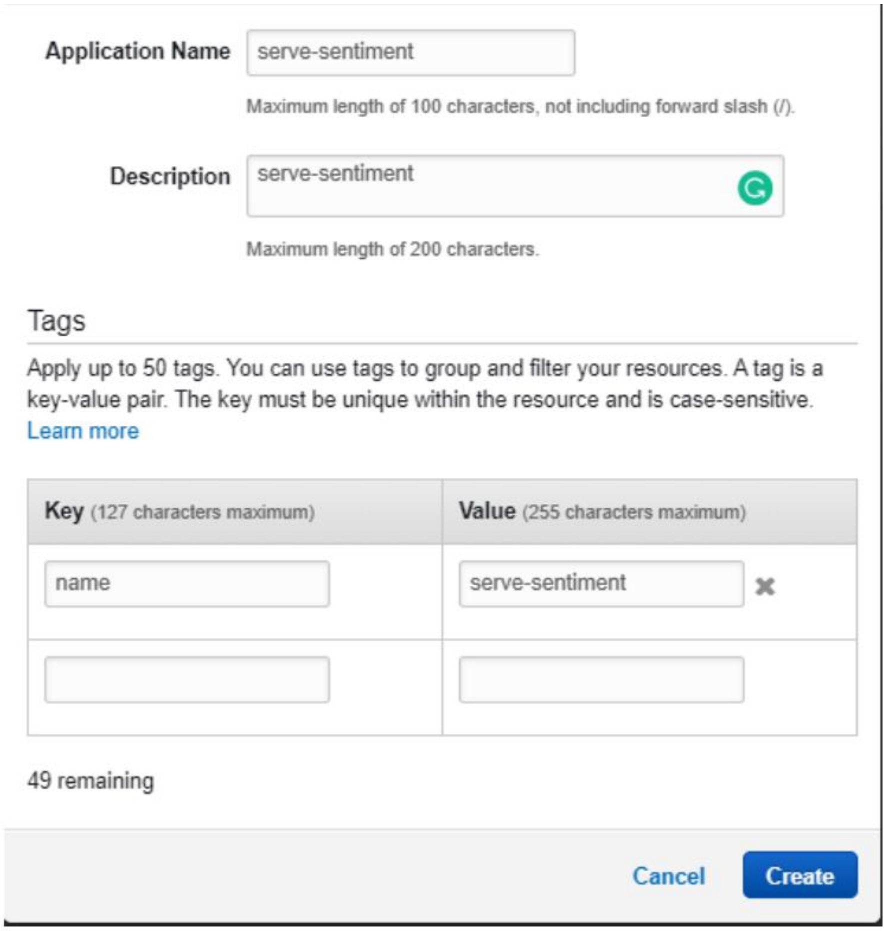
An Elastic Beanstalk App (https://aws.amazon.com/elasticbeanstalk/) is a logical collection of Elastic Beanstalk components, including environments, versions, and environment configurations. In Elastic Beanstalk an application is conceptually similar to a folder.

To create a new application, follow the screenshots below. You can name your application with any name.



Click "Create New Application" on top right

Figure 1: create new application



Application Name: serve-sentiment

Maximum length of 100 characters, not including forward slash (/).

Description: serve-sentiment

Maximum length of 200 characters.

Tags

Apply up to 50 tags. You can use tags to group and filter your resources. A tag is a key-value pair. The key must be unique within the resource and is case-sensitive. Learn more

Key (127 characters maximum)	Value (255 characters maximum)
name	serve-sentiment

49 remaining

Create a new App — Apps are equivalent to folders in Elastic Beanstalk

Figure 2: name your application

iii. Create an Environment

Background:

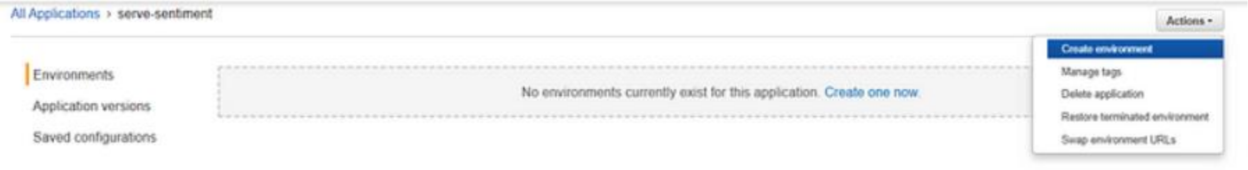
A Web Server Environment (https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/concepts-webserver.html) is a collection of AWS resources running an application version. When you create an environment, Elastic Beanstalk provisions the resources required to run your application. Each environment runs only one application version at a time.

A Web Server Environment runs web server processes (Apache, Nginx, etc.), while Worker environment (https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/concepts-worker.html) deals with long-running processes and communicates with AWS SQS.

A platform is a combination of an operating system, programming language runtime, web server, application server, and Elastic Beanstalk components. For platform select Python.

Your task:

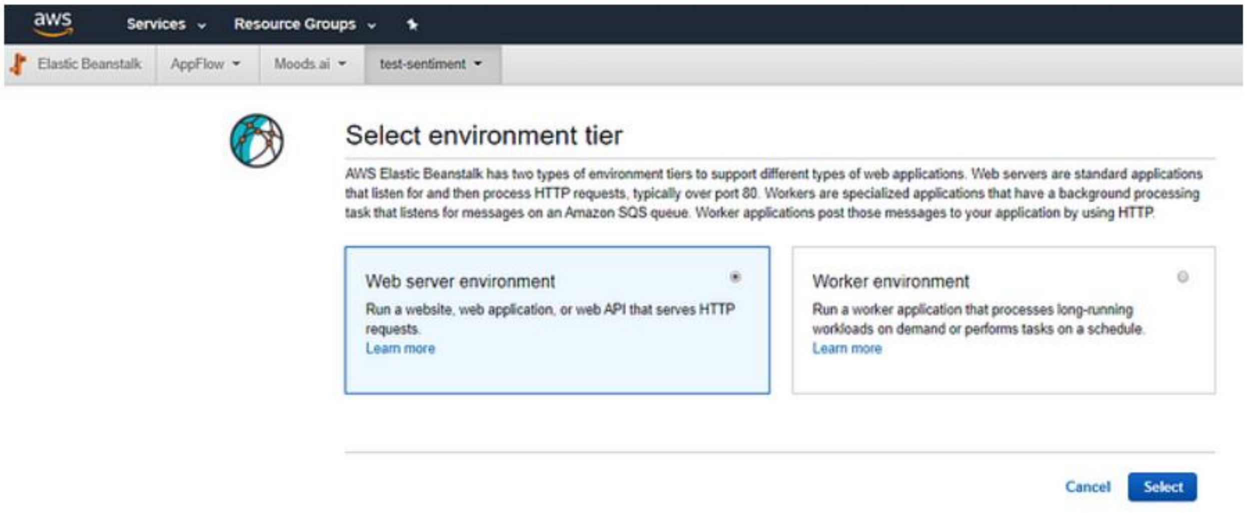
- **Step 1:** Click 'Actions', then choose 'create environment' (Fig. 3).
- **Step 2:** Then select 'web server environment' (Fig. 4)
- **Step 3:** Choose 'Python' as the preconfigured platform, use any name for the environment (Fig. 5)



No environments currently exist for this application. Create one now.

Create an environment by clicking on the top right

Figure 3: create environment



Select environment tier

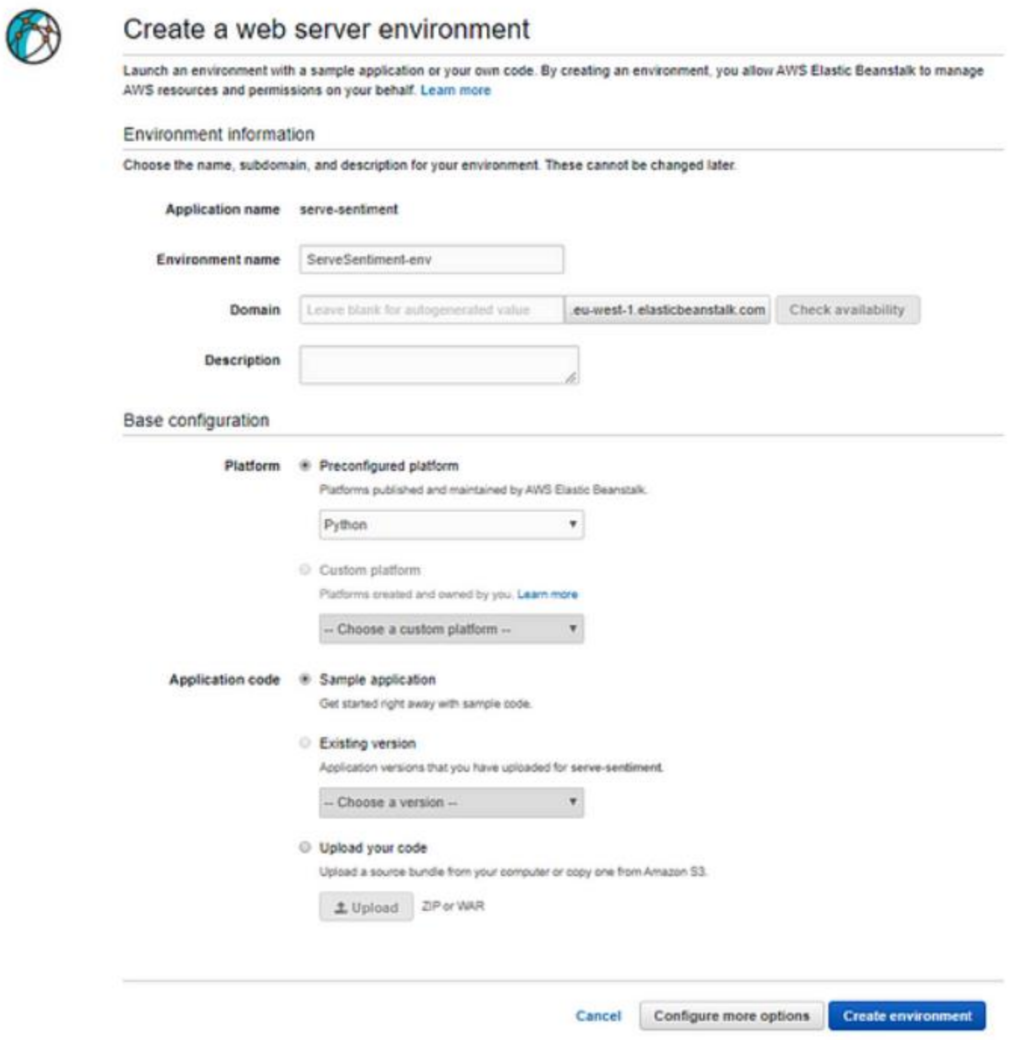
AWS Elastic Beanstalk has two types of environment tiers to support different types of web applications. Web servers are standard applications that listen for and then process HTTP requests, typically over port 80. Workers are specialized applications that have a background processing task that listens for messages on an Amazon SQS queue. Worker applications post those messages to your application by using HTTP.

● Web server environment
Run a website, web application, or web API that serves HTTP requests.
Learn more

○ Worker environment
Run a worker application that processes long-running workloads on demand or performs tasks on a schedule.
Learn more

Select "Web server environment" — Web Servers Env run web server processes (apache, nginx, etc)

Figure 4: select web server environment



Create a web server environment

Launch an environment with a sample application or your own code. By creating an environment, you allow AWS Elastic Beanstalk to manage AWS resources and permissions on your behalf. Learn more

Environment information

Choose the name, subdomain, and description for your environment. These cannot be changed later.

Application name	serve-sentiment
Environment name	ServeSentiment-env
Domain	Leave blank for autogenerated value .eu-west-1.elasticbeanstalk.com Check availability
Description	

Base configuration

Platform
● Preconfigured platform
Platforms published and maintained by AWS Elastic Beanstalk.
Python ▾
○ Custom platform
Platforms created and owned by you. Learn more
-- Choose a custom platform -- ▾

Application code
● Sample application
Get started right away with sample code.
○ Existing version
Application versions that you have uploaded for serve-sentiment.
-- Choose a version -- ▾
○ Upload your code
Upload a source bundle from your computer or copy one from Amazon S3.
Upload ZIP or WAR

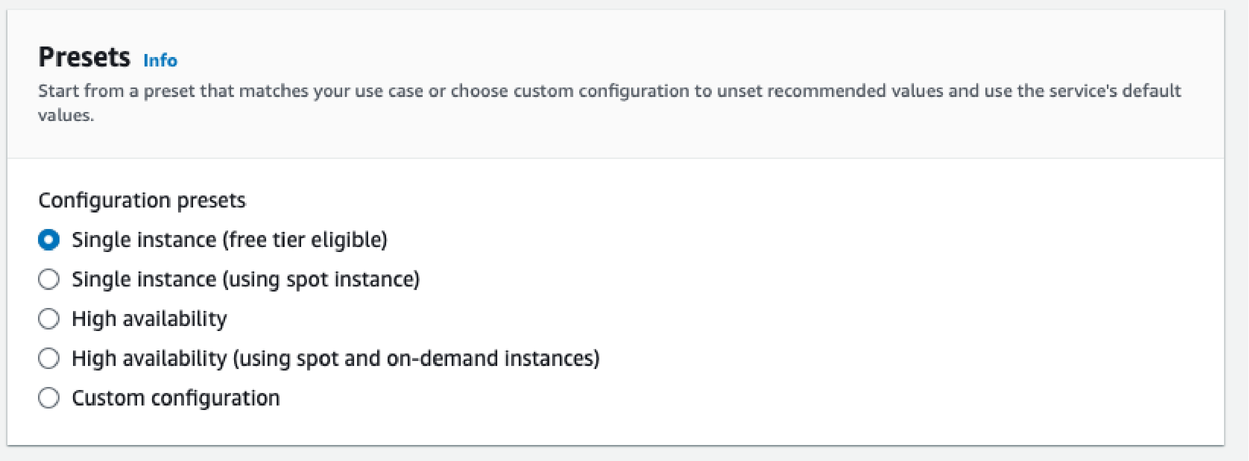
Create an environment with Python as a preconfigured platform

Figure 5: Create a web server environment

iv. Configurations the application

By default, you have many possible predefined configurations (Fig 6.)

Choose the “single instance (free tier eligible)” configuration.



Presets Info

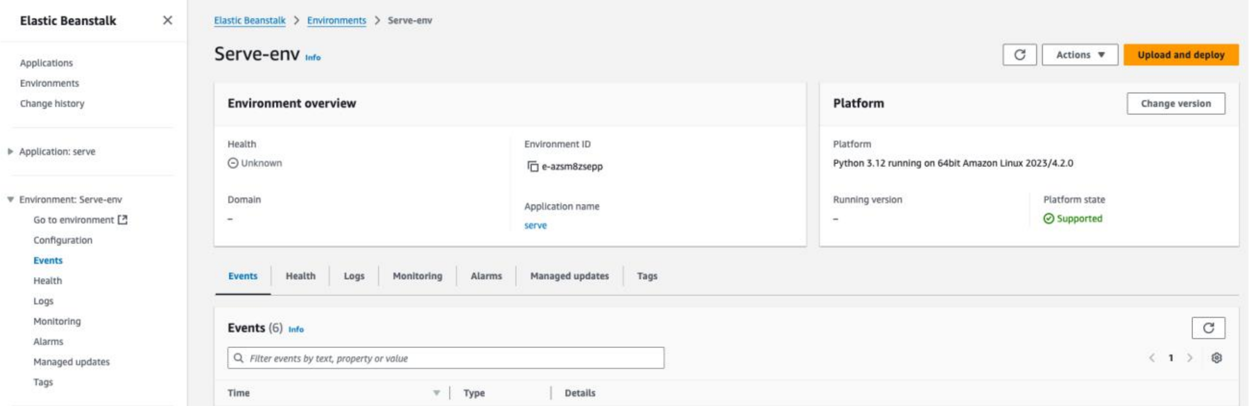
Start from a preset that matches your use case or choose custom configuration to unset recommended values and use the service's default values.

Configuration presets

- ● Single instance (free tier eligible)
- ○ Single instance (using spot instance)
- ○ High availability
- ○ High availability (using spot and on-demand instances)
- ○ Custom configuration

Figure 6: Presets for configuration

Leave everything else as default and launch your application (this will take few minutes)



Serve-env Info

Environment overview

Health: Unknown

Environment ID: e-azsm8zsepp

Domain: –

Application name: serve

Platform

Platform: Python 3.12 running on 64bit Amazon Linux 2023/4.2.0

Running version: –

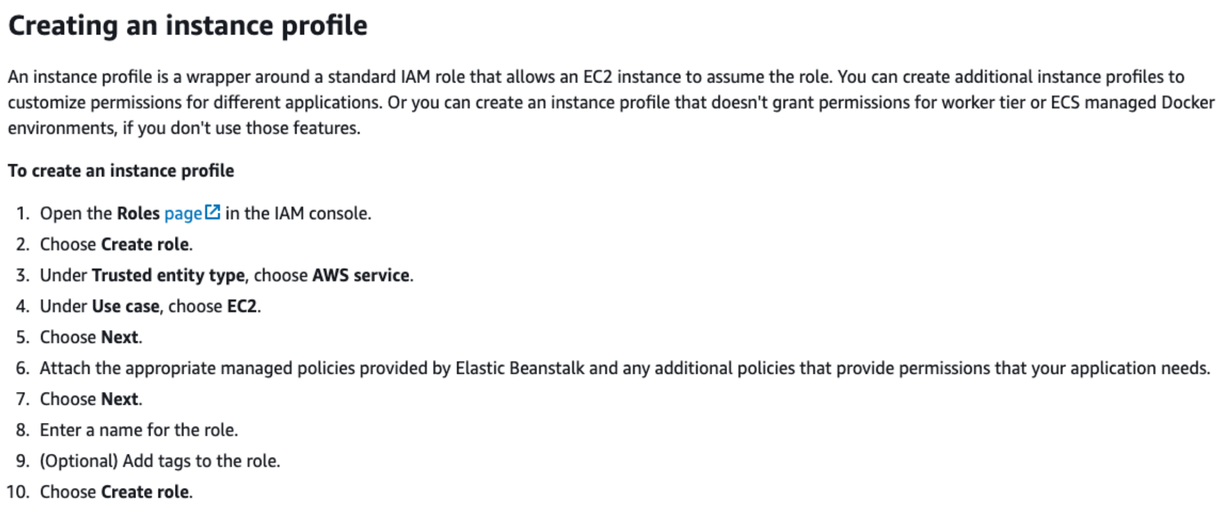
Platform state: Supported

Events (6) Info

Figure 7: Deployed application

v. Creating instance profile

Step 1: Since, AWS no longer automatically creates an instance for every beanstalk application. We need to create an instance manually. Please use the instructions here to create an instance: https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/iam-instanceprofile.html#iam-instanceprofile-create



Creating an instance profile

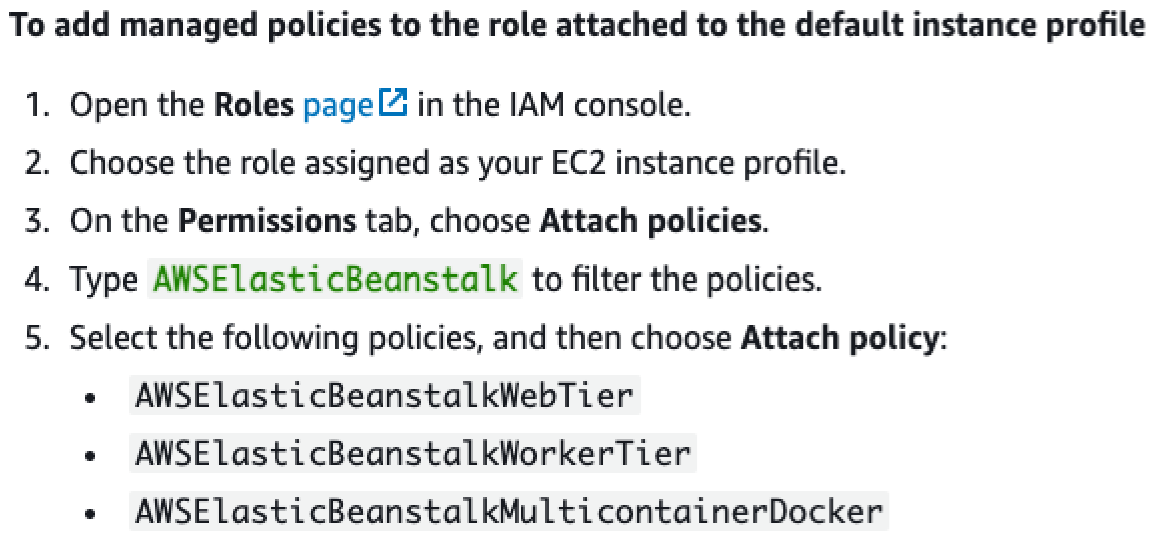
An instance profile is a wrapper around a standard IAM role that allows an EC2 instance to assume the role. You can create additional instance profiles to customize permissions for different applications. Or you can create an instance profile that doesn't grant permissions for worker tier or ECS managed Docker environments, if you don't use those features.

To create an instance profile

1. Open the **Roles** page in the IAM console.
2. Choose **Create role**.
3. Under **Trusted entity type**, choose **AWS service**.
4. Under **Use case**, choose **EC2**.
5. Choose **Next**.
6. Attach the appropriate managed policies provided by Elastic Beanstalk and any additional policies that provide permissions that your application needs.
7. Choose **Next**.
8. Enter a name for the role.
9. (Optional) Add tags to the role.
10. Choose **Create role**.

Figure 8: instructions on creating an instance profile.

Step 2: Add managed policies using the following instructions: https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/iam-instanceprofile.html#iam-instanceprofile-update



To add managed policies to the role attached to the default instance profile

1. Open the **Roles** page in the IAM console.
2. Choose the role assigned as your EC2 instance profile.
3. On the **Permissions** tab, choose **Attach policies**.
4. Type `AWSElasticBeanstalk` to filter the policies.
5. Select the following policies, and then choose **Attach policy**:
   - `AWSElasticBeanstalkWebTier`
   - `AWSElasticBeanstalkWorkerTier`
   - `AWSElasticBeanstalkMulticontainerDocker`

Figure 9: instructions on adding managed policies

You should now be able to successfully launch the elastic beanstalk application environment.

Note: If your attempt to create a new environment is not successful, it could be due to the Amazon EC2 Auto Scaling service restricting your ability to create launch configurations. If the error indicates that the CreateLaunchConfiguration API throws an *UnsupportedOperationException*, then you must set a configuration option in your environment to direct Elastic Beanstalk to use launch templates instead of launch configurations. For more information, see Option settings for launch templates.

5. Activities on your Flask Application Folder

i. Create Elastic Beanstalk config files

Step 1: Create a folder *.ebextensions* for Elastic Beanstalk config files. All configuration files should be present in this folder and are necessary to configure your environment and customize the AWS resources.

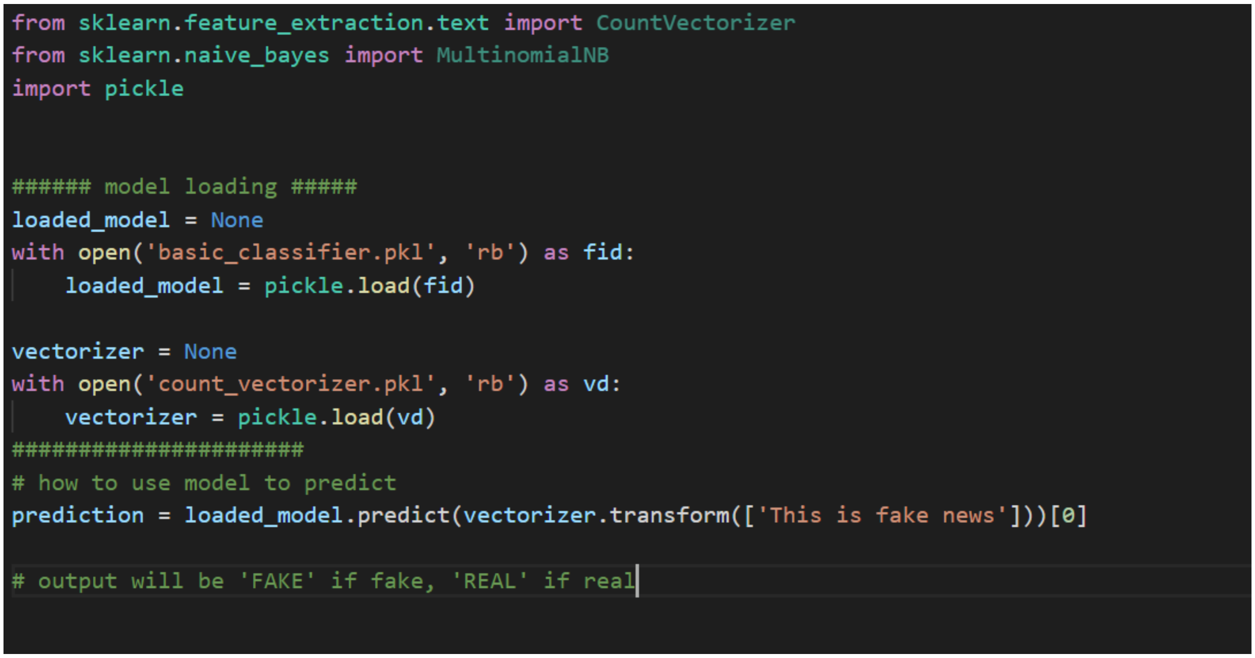
Step 2: Create the following 3 files inside the .ebextensions folder: (1) 00_application.config; (2) 01_pip-install.config; (3) 02_wsgi.config. You can find the files on Quercus here: '*Files/Lab/instructions/PRA5/*'

ii. Create Flask App

Create the following files and folders to upload a Flask app to your AWS instance.

Step 1: Create a Python file *application.py* for sentiment analysis and write a *load_model* function. For this project, you will not have to train your own model. You can load a trained Fake News detector model by adding the following code snippet (Fig. 10) in your *load_model* function after adding the 2 pickle files in the same directory as the application.py file. You can find the 2 pickle files on Quercus here: '*Files/Lab/instructions/PRA5/PRA5_models.zip*'

Note: you will need to install scikit-learn for this code snippet to work.



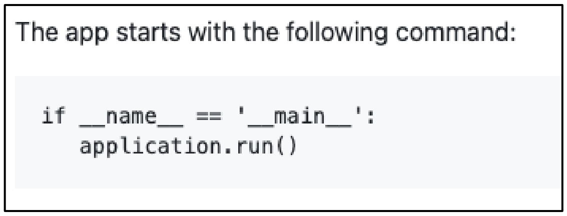
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
import pickle


###### model loading #####
loaded_model = None
with open('basic_classifier.pkl', 'rb') as fid:
    loaded_model = pickle.load(fid)

vectorizer = None
with open('count_vectorizer.pkl', 'rb') as vd:
    vectorizer = pickle.load(vd)
######################
# how to use model to predict
prediction = loaded_model.predict(vectorizer.transform(['This is fake news']))[0]

# output will be 'FAKE' if fake, 'REAL' if real
```

Figure 10: Code for loading ML model.



The app starts with the following command:

```
if __name__ == '__main__':
    application.run()
```

Figure 9: Code to launch Flask application.

iii. Zip the whole Flask application folder and then upload & deploy

Step 1: Zip the whole Flask application, containing the application.py and the .ebextensions folder.

Step 2: On your app environment (Fig. 7), click on [Upload and Deploy], and upload the zip.

6. Testing Your API

Once you have implemented the Flask Application (a.k.a REST API) with the ML model, you will need to test it for correctness. There are multiple ways of testing this app, we ask that you implement the following:

1. Functional/Unit Tests: Test your app by creating 4 test inputs (two fake news and two real news) and run these test cases with your app to ensure that it works.

2. Latency/performance Testing: Since this is a service that will be run live, it is important to know the latency of your API. Take the previous test cases and do 100 API calls to the REST server (a.k.a AWS Elastic Beanstalk server) with your examples.

For this assignment, you are required to complete the following tasks:

1. Record the timestamps for each function call in a CSV file, ensuring that there are exactly 100 rows per test case.
2. Generate a boxplot to visualize the performance results for each test case and calculate the average performance.

Be sure to include the boxplots in the README file of your repository. Additionally, during the grading session, you may be asked to present the CSV files, so have them ready for review.

7. Submission:

On Quercus, submit a link to your GitHub repository containing the code for your Flask application.

Acknowledgement:

This PRA is heavily inspired from the following sources:

1. https://medium.com/swlh/deploy-a-machine-learning-model-with-aws-elasticbeanstalk-dfcc47b6043e
2. https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/GettingStarted.CreateApp.html