

ECE444-PRA5: Deploying an ML APP to the Cloud

| | |
|--|---|
| 1. Learning objective: | 1 |
| 2. Scenario: | 1 |
| 3. Background | 2 |
| 4. Activities on AWS | 2 |
| i. Sign-up for AWS free account | 2 |
| ii. Create an App in AWS Elastic Beanstalk | 2 |
| iii. Create an Environment | 3 |
| iv. Configurations the application | 4 |
| v. Creating instance profile | 5 |
| 5. Activities on your Flask Application Folder | 7 |
| i. Create Elastic Beanstalk config files | 7 |
| ii. Create Flask App | 7 |
| iii. Zip the whole Flask application folder and then upload & deploy | 8 |
| 6. Testing Your API | 8 |
| 7. Submission: | 9 |
| Acknowledgement | 9 |

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.

For more info refer to <https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/GettingStarted.html>

Compute

Amazon Elastic Beanstalk

End-to-end web application management.

Amazon Elastic Beanstalk 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.

Get started

You simply upload your code and Elastic Beanstalk automatically handles the deployment, from capacity provisioning, load balancing, and automatic scaling to web application health monitoring, with ongoing fully managed patch and security updates. [Learn more](#)

Benefits and features

Get started

Easily deploy your web application in minutes.

[Create application](#)

Pricing

There's no additional charge for Elastic Beanstalk. You pay for Amazon Web Services resources that we create to store and run your web application, like Amazon S3 buckets and Amazon EC2 instances.

Getting started

[Launch a web application](#)

Figure 1: create new application

Step 1
☒ **Configure environment**
 Step 2
☐ Configure service access
 Step 3 - optional
☐ Set up networking, database, and tags
 Step 4 - optional
☐ Configure instance traffic and scaling
 Step 5 - optional
☐ Configure updates, monitoring, and logging
 Step 6
☐ Review

Configure environment info

Environment tier info

Amazon Elastic Beanstalk has two types of environment tiers to support different types of web applications.

☒ **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](#)

Application information info

Application name

serve-sentiment

Maximum length of 100 characters.

► **Application tags (optional)**

Environment information info

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

Environment name

Serve-sentiment-env

Must be from 4 to 40 characters in length. The name can contain only letters, numbers, and hyphens. It can't start or end with a hyphen. This name must be unique within a region in your account.

Domain

Leave blank for autogenerated value .eu-north-1.elasticbeanstalk.com [Check availability](#)

► **Environment description**

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:** In 'environment tier', select 'web server environment' (Fig. 2)
- **Step 3:** Choose 'Python' as the preconfigured platform, use any name for the environment (Fig. 3)

The screenshot shows the 'Create new function' wizard in the AWS Lambda console. It is divided into three main sections: Platform, Application code, and Presets. The Platform section has three dropdown menus: Platform (set to Python), Platform branch (set to Python 3.13 running on 64bit Amazon Linux 2023), and Platform version (set to 4.7.3 (Recommended)). The Application code section has two radio buttons: 'Sample application' (selected) and 'Existing version' (Application versions that you have uploaded). Below these is an 'Upload your code' option with a subtext 'Upload a source bundle from your computer or copy one from Amazon S3.' The Presets section has a heading 'Configuration presets' and five radio buttons: 'Single instance (free tier eligible)' (selected), 'Single instance (using spot instance)', 'High availability', 'High availability (using spot and on-demand instances)', and 'Custom configuration'. At the bottom right, there are 'Cancel' and 'Next' buttons.

Platform [Info](#)

Platform ▼ Python

Platform branch ▼ Python 3.13 running on 64bit Amazon Linux 2023

Platform version ▼ 4.7.3 (Recommended)

Application code [Info](#)

☒ Sample application

☐ Existing version
Application versions that you have uploaded.

☐ Upload your code
Upload a source bundle from your computer or copy one from Amazon S3.

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

[Cancel](#) [Next](#)

Figure 3: Create a web server environment

iv. Configurations the application

By default, you have many possible predefined configurations (Fig 4.)

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

The screenshot shows the 'Presets' section of the AWS Lambda console. It has a heading 'Configuration presets' and five radio buttons: 'Single instance (free tier eligible)' (selected), 'Single instance (using spot instance)', 'High availability', 'High availability (using spot and on-demand instances)', and 'Custom 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 4: Presets for configuration

v. Creating a service role

Step 1: Please create a service role using the default options (Fig 5)

Figure 5: Configuring service access

Figure 6: Creating Service role

vi. Creating EC2 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>

X

Step 1 Configure environment

Step 2 **Configure service access**

Step 3 - optional Set up networking, database, and tags

Step 4 - optional Configure instance traffic and scaling

Step 5 - optional Configure updates, monitoring, and logging

Step 6 Review

Configure service access Info

Service access
IAM roles, assumed by Elastic Beanstalk as a service role, and EC2 instance profiles allow Elastic Beanstalk to create and manage your environment. Both the IAM role and instance profile must be attached to IAM managed policies that contain the required permissions. [Learn more](#)

Service role
Choose an IAM role for Elastic Beanstalk to assume as a service role. The IAM role must have the required IAM managed policies.

Create role

EC2 instance profile
Choose an IAM instance profile with managed policies that allow your EC2 instances to perform required operations.

Create role

EC2 key pair - optional
Select an EC2 key pair to securely log in to your EC2 instances. [Learn more](#)

[Cancel](#) [Skip to review](#) [Previous](#) [Next](#)

Figure 5: Create an EC2 instance profile

Step 2: Review default managed policies using the following instructions:

<https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/iam-instanceprofile.html#iam-instanceprofile-update>

Select trusted entity Info

Trusted entity type

☒ **AWS service**
Allow AWS services like EC2, Lambda, or others to perform actions in this account.

☐ **AWS account**
Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.

☐ **Web identity**
Allows users federated by the specified external web identity provider to assume this role to perform actions in this account.

☐ **SAML 2.0 federation**
Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account.

☐ **Custom trust policy**
Create a custom trust policy to enable others to perform actions in this account.

Use case
Allow an AWS service like EC2, Lambda, or others to perform actions in this account.

Service or use case

Choose a use case for the specified service.

Use case

☒ **Elastic Beanstalk - Compute**
Allows your environment's EC2 instances to perform operations required for your application.

☐ **Elastic Beanstalk - Environment**
Allows access to other AWS service resources that are required to create and manage environments.

[Cancel](#) [Next](#)

Figure 6: Creating an instance profile.

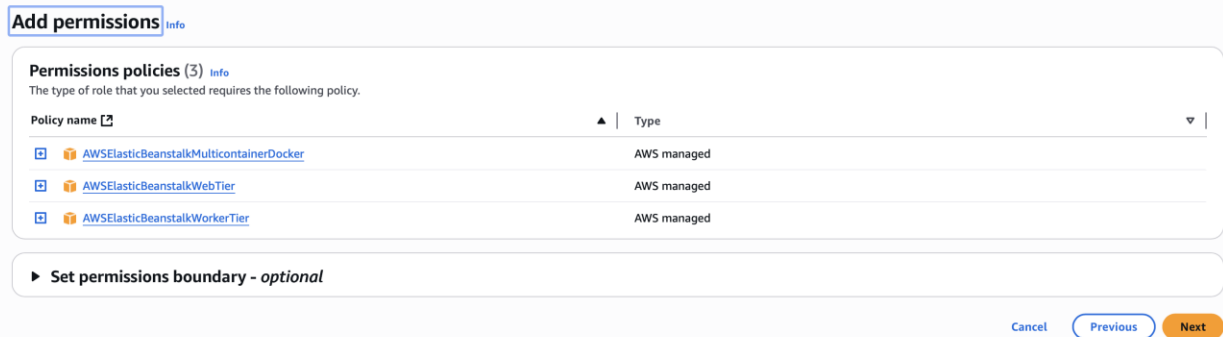


Figure 7: Default Permission policies for the EC2 instance profile

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

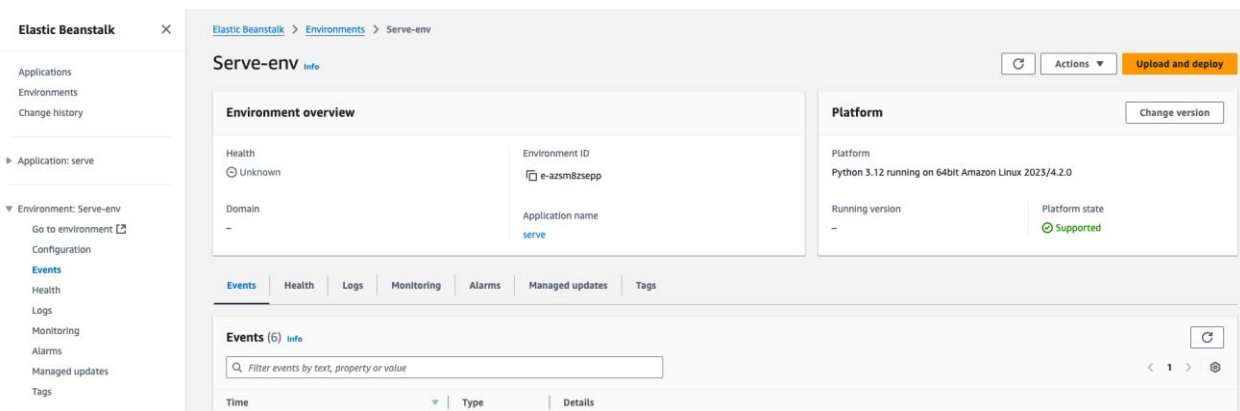


Figure 8: Deployed application

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/PRA5/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. 9) 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/PRA5/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 9: Code for loading ML model.

The app starts with the following command:

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

Figure 10: 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, 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>