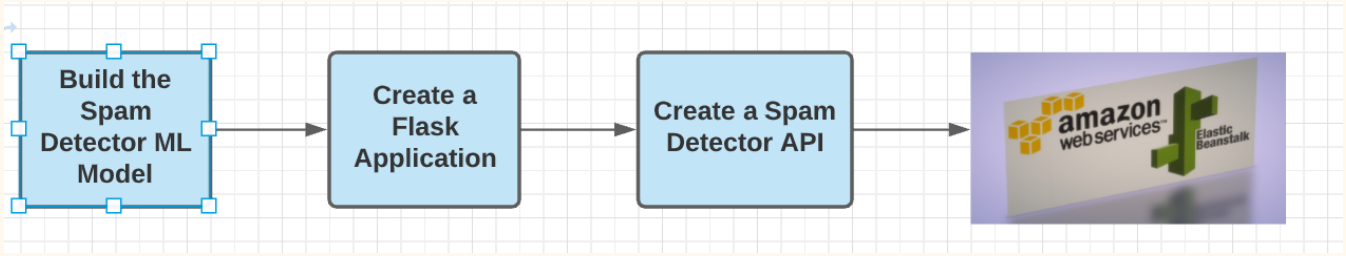
IDS-594 MACHINE LEARNING DEPLOYMENT

DEPLOYMENT OF MACHINE LEARNING MODEL ON AWS ELASTIC BEANSTALK

BY PRITHA GHOSH (650992335) & VIBHANSHU (679278119)



OUTLINE:

In this project we have built a spam detector model using machine learning and launched it as an API using AWS Elastic Beanstalk technology. We have used the Flask python framework to create the API, basic machine learning methods to build the spam detector and AWS desktop management console to deploy the spam detector into the AWS cloud servers. We have also explored additional Elastic Beanstalk services like how to switch between different versions of a web application as well as monitoring AWS servers using Elastic Beanstalk Management Console.

POSSIBLE SOLUTIONS:

VIRTUAL MACHINE VS CONTAINERS VS SERVERLESS vs PaaS: WHY WE CHOSE PaaS DEPLOYMENT

VIRTUAL MACHINES

Virtual Machines are built to mimic a physical machine, with a complete operating system, full support for (virtual) hardware and peripherals and complete isolation between virtual machines.

CONTAINERS

With containers, we move one level up in the stack, from the virtual hardware level to the virtual operating system level. With containers, applications run in an isolated space in the operating system. Containers running on the same host share the underlying operating system, thereby decreasing storage and computing resources.

SERVERLESS

With serverless functions, the abstraction layer is moved up the stack even further to a single process. It replaces long-running virtual machines with compute power that comes into existence on demand and disappears immediately after use. We can configure events, such as API requests or file uploads, that trigger the serverless function to execute. When the action is complete, the server goes idle until another action is requested, and we are not billed for the idle time.

PLATFORM AS A SERVICE(PaaS)

PaaS and serverless computing are similar in that for both, all a developer has to worry about is writing and uploading code, and the vendor handles all backend processes. However, scaling is vastly different when using the two models. Applications built using serverless computing, or FaaS, will scale automatically, while PaaS applications will not unless programmed to do so. Startup times also vary greatly; serverless applications can be up and running almost instantly, but PaaS applications are more like traditional applications and have to be running most of the time or all of the time in order to be immediately available for users.

Another difference is that serverless vendors do not provide development tools or frameworks, as PaaS vendors do.

TECHNOLOGY	ADVANTAGES	DISADVANTAGES
VIRTUAL MACHINES	• VMs are capable of running far more operations than a single	• Expanded functionality makes VMs far less portable

	container, which is why they are the traditional way monolithic workloads are usually packaged.	because of their dependence on the OS, application, and libraries.
CONTAINERS	• They are portable. • Full control over the application - resources, security, policies, deployment . • Suitable for large and complex applications as there are no memory or time limitations like there are with serverless.	• More work to set up and manage every time there is a code change. • More expensive as payment is charged even when the servers are idle. • Data is often lost when containers are rescheduled or destroyed.
SERVERLESS	• Payment is only charged when the server is active. • Application scales automatically depending on traffic. • No server management. • Best suited for event-driven programs that have specific, delineated events that can be used to trigger specific actions.	• Less control over the deployment environment. • Vendor lock-in is a concern. • Due to the event-based nature of serverless, it's not the best choice for long-running apps.
PLATFORM AS A SERVICE (PaaS)	• Similar to serverless in terms of developers only having to write the application code and not involved in server management. • Provides complete software development and deployment platforms. • Best suited for conventional application development where we would want to have a lot more control over the underlying environment.	• Application has to be configured to scale up. • Vendor lock-in is a concern.

AWS vs AZURE

Pricing is the main reason we opted for AWS, we do not have to pay any upfront money which is ideal for small term projects. Microsoft definitely has an edge in case users have used Microsoft softwares in the model but that is not the case for us.

AWS ELASTIC BEANSTALK VS EC2 VS LAMBDA VS SAGEMAKER

Comparison with AWS EC2:

Elastic Beanstalk is one layer of abstraction away from the EC2 layer. Elastic Beanstalk will set up an environment that can contain a number of EC2 instances, an optional database, as well as a few other AWS components such as a Elastic Load Balancer, Auto-Scaling Group, Security Group. Elastic Beanstalk doesn't add any cost on top of these resources that it creates. In fact, we aren't charged for Beanstalk itself. We are charged for the AWS resources you're using, such as S3, SNS, and EC2.

Comparison with AWS LAMBDA:

AWS Elastic Beanstalk is best for businesses that need a robust computing platform. AWS Lambda allows users to easily run code in a serverless environment, but it doesn't include as many options for customizing the environment. For businesses looking for a flexible platform that they can manage however they want, AWS Elastic Beanstalk may be preferred. Additionally, while Lambda is a good choice for pieces of code that applications will call, AWS Elastic Beanstalk is more usable for fully featured application deployment.

Comparison with AWS SAGEMAKER:

AWS Sagemarker tends to sacrifice flexibility in favor of speed in performing basic operations. For example, while training a standard ML algorithm can be really easy, doing custom training can be a complete pain, because the SageMaker API's for it are underbaked and poorly documented.

WORKFLOW:

1. **Building the Spam Detector ML Model :** We use a dataset on Kaggle to build our Spam Detector Model: SMS Spam Collection Dataset | Kaggle. The SMS Spam Collection is a set of SMS tagged messages that have been collected for SMS Spam research. It contains one set of SMS messages in English of 5,574 messages, tagged according to ham (legitimate) or spam. The model takes an input from the user and classifies it as spam or ham.

```
In [8]:
1 #Predict Spam
2 model.predict(vectorizer.transform(["you won $900 in the new lottery draw. Call +123456789."]))

Out[8]: array(['spam'], dtype=object)

In [9]:
1 #Predict Ham
2 model.predict(vectorizer.transform(["Hello there. How are you doing?"]))

Out[9]: array(['ham'], dtype=object)
```

2. Create a Flask Web Application

Flask is a micro web framework written in Python. Some important features of Flask are as follows:

- Flask has its own development server and debugger.
- It supports Jinja2, a templating engine used in Python to combine it with data sources and obtain dynamic web pages.
- It is based on a Werkzeug WSGI tool that provides a web server gateway interface to communicate between a web server and web application.
- Flask classes allow extensions like database library, upload handling, form validation, and open authentication technologies.
- Easy boilerplate code for app creation and running makes Flask a better web framework choice than Django.

Comparison: Flask Vs Django

- Django is a full stack Python web framework, which makes it easier to perform user authentication, URL routing and other database migration tasks. Flask is a micro web framework, which helps in keeping a web application simple and extensible.
- Django features provide the facility to create large, complex projects while Flask is better suited for simple web applications with only the required functionality.

PREREQUISITES:

- Python 2.3 and above
- Virtual environment: To avoid compatibility issues between the different versions of libraries that we import.

```
* Serving Flask app 'application' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

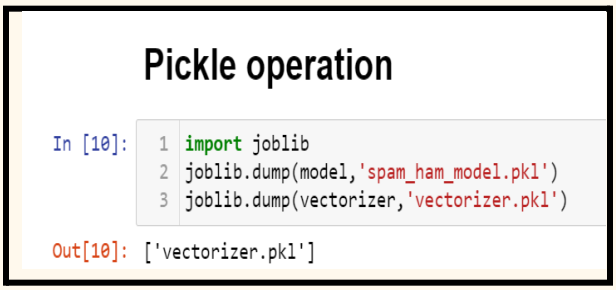
3. Build a Spam Detector API

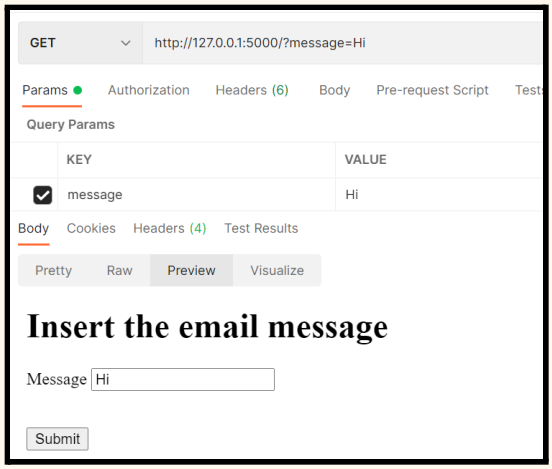
An application programming interface (API) is a computing interface that defines interactions between multiple software intermediaries. It defines the kinds of calls or requests that can be made, how to make them, the data formats that should be used, the conventions to follow, etc.

We convert our ML model into a Spam Detector API so that we can deploy it onto AWS Servers. In order to integrate our model into an API, we use a technology called Pickle.

The pickle module implements binary protocols for serializing and de-serializing a Python object structure. Pickling is a process in which a Python object hierarchy is converted into a byte stream and unpickling is the inverse operation, where a byte steam is converted back into an object hierarchy.If we want to use the Spam detector ML model in the API, we have to pickle the model as well as the vectorizer function.

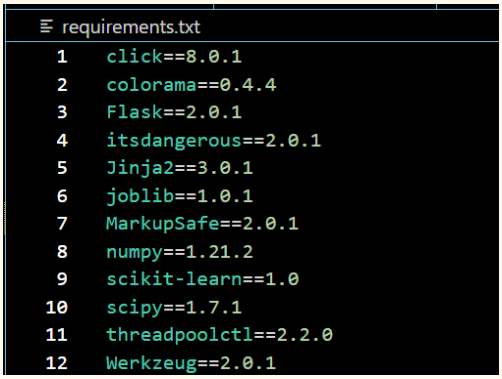
We also test our API using the Postman tool, which is popular for API testing.





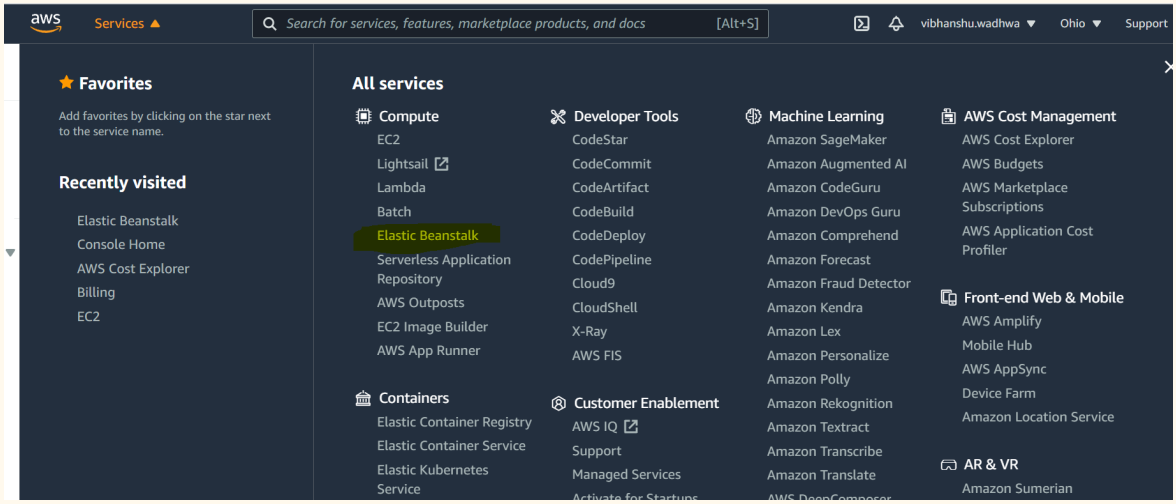
4. Deploying the ML API into AWS Servers

Before we get ready to deploy our API into AWS, we need to generate the requirements.txt file so that AWS can handle the project dependencies into its cloud servers.

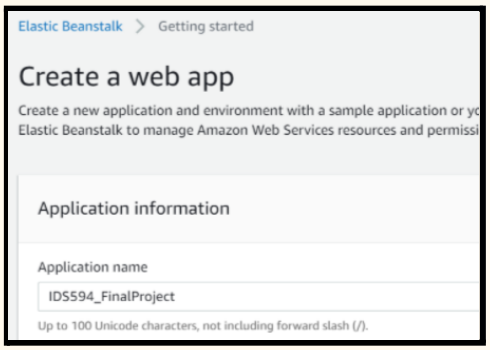


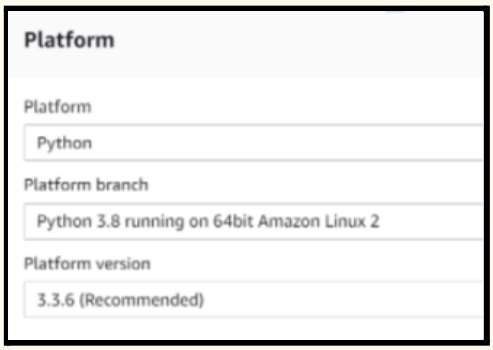
AWS Elastic Beanstalk is an easy-to-use service for deploying and scaling web applications into AWS cloud servers. You can simply upload your code and Elastic Beanstalk automatically handles the deployment, from capacity provisioning, load

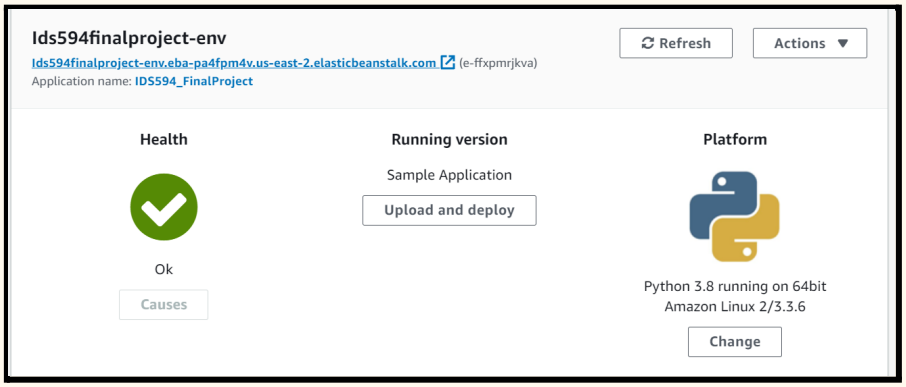
balancing, auto-scaling to application health monitoring. There is no additional charge for Elastic Beanstalk - you only pay for the AWS resources needed to store and run your applications. Once your application is running, Elastic Beanstalk automates management tasks–such as monitoring, application version deployment, a basic health check–and facilitates log file access. By using Elastic Beanstalk, developers can focus on developing their application and are freed from deployment-oriented tasks, such as provisioning servers, setting up load balancing, or managing scaling.



Step 1 : Create an application

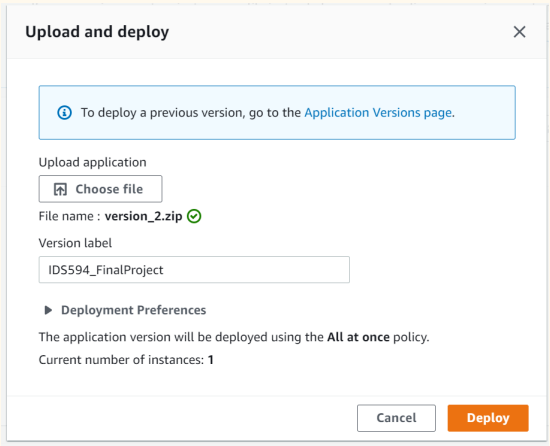




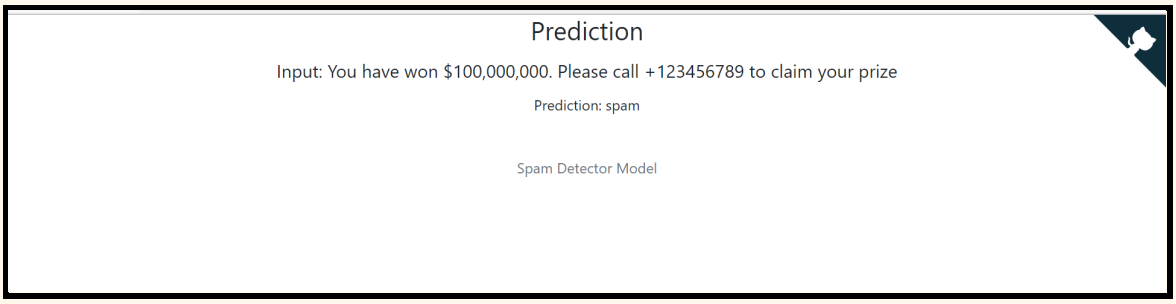


Step 2: Deploy our code into AWS virtual servers.

We upload our project zip file which contains - the python application file, the requirements.txt file, the pickle files for the model and the vectorizer function as well as the html templates.



Step 3: API Deployment on AWS



We can see that the model has been successfully deployed on the AWS server. There are additional functionalities that Elastic Beanstalk provides, which can be worth exploring in case of larger applications:

- **Monitoring:** It is important for maintaining the reliability, availability, and performance of AWS Elastic Beanstalk and your AWS solutions. We should collect monitoring data from all of the parts of theAWS solution so that we can more easily debug a multipoint failure if one occurs.
- **Enhanced Health Reporting:** Elastic Beanstalk analyzes the information to provide a better picture of overall environment health and help identify issues that can cause the application to become unavailable.
- **Logs:** The Amazon EC2 instances in the Elastic Beanstalk environment generate logs that can be viewed to troubleshoot issues with the application or configuration files.

LESSONS LEARNED

This turned out to be a really interesting project for us. We studied several methods to deploy our machine learning model such as virtual machines, containers, serverless models and PaaS. Having said that, we do realise that these deployment methods can be intertwined with each other to create a more robust environment.

For our project, we felt that AWS Elastic Beanstalk was a good fit as it not only handles all the backend processes, it also provides us flexibility to deploy our own code as it is.

We also studied the most popular cloud vendors - AWS and Azure: AWS is more “start-up” friendly as it offers a unique Pay-As-You-Go-Model which requires no upfront cost for users whereas Microsoft mostly focuses on Big Accounts such as Fortune-500 and other big companies. AWS also provided features like load-balancing, auto-scaling, capacity provisioning and application health monitoring.

We learned several different stages of cloud deployment during this project such as creating a flask application, converting our ML model into an API and deployment using AWS Elasticbeanstalk. We also learned about managing cloud resources and minimizing the cost of deployment. We figured out how to utilize AWS free resources as much as possible and selected the cost efficient option in case free options are not available. Total cost of deploying our project for a month came out to be $3.5.

One drawback of Elastic Beanstalk that we faced was that it was sometimes unreliable. One time our model failed but there was no indication of the root cause and further attempts at deploying at that instant were unsuccessful. We had to troubleshoot and fix it ourselves by terminating the instance that had the deployment issue. There was no indication on the AWS site as to what could possibly have gone wrong.

In conclusion, Elastic Beanstalk helped us to easily deploy our ML model while leveraging Amazon’s powerful infrastructure. To achieve even more versatility, we may explore enhancing the deployment process with containers, like Docker. While it is not a perfect tool, if one is just looking to focus on development while reducing system operations, Elastic Beanstalk is a good option.

REFERENCES

1. https://nordicapis.com/how-to-create-an-api-using-the-flask-framework/
2. SMS Spam Collection Dataset | Kaggle
3. https://www.trustradius.com/compare-products/aws-elastic-beanstalk-vs-aws-lambda
4. https://www.cloudysave.com/aws/services/elastic-beanstalk/elastic-beanstalk-vs-ec2/
5. https://stackoverflow.com/questions/25956193/difference-between-amazon-ec2-and-aws-elastic-beanstalk
6. https://towardsdatascience.com/why-do-we-need-aws-sagemaker-79bce465f19f
7. https://www.cabotsolutions.com/an-in-depth-comparison-between-azure-app-service-and-aws-elastic-beanstalk
8. https://www.thorntech.com/containers-vs-serverless/
9. https://us-east-2.console.aws.amazon.com/elasticbeanstalk/home?region=us-east-2#/welcome
10. https://aws.amazon.com/elasticbeanstalk/faqs/
11. https://www.cloudflare.com/learning/serverless/glossary/serverless-vs-paas/
12. https://www.bmc.com/blogs/serverless-paas/#
13. https://www.zdnet.com/article/serverless-computing-vs-platform-as-a-service-which-is-right-for-your-business/

OBJECTIVE AND PROJECT TIMELINE:

TASK	TIMELINE
Research work: Choosing the best deployment method	AUG-27 to SEPT-2
Research work: Choosing the best tools	SEPT-2 to SEPT-8
Build a spam detector Machine Learning Model & Convert the ML model into API	SEPT-9 to SEPT-16
Create a Flask Web Application	SEPT-17 to SEPT-23
Deploy your ML model (API) into AWS virtual servers using Elastic Beanstalk	SEPT-24 to SEPT-30
Explore additional AWS Elastic Beanstalk features	OCT-1 to OCT-7
Report	OCT-8 to OCT-14