**MLOps Foundations**

1. **Abstract**

This report outlines the implementation of a simple CI/CD pipeline for a machine learning project to understand the basics of MLOps. The project integrates version control using Git and automates key processes such as linting, testing, and deployment using GitHub Actions (or GitLab CI). The CI/CD pipeline ensures code quality through linting, verifies functionality with unit tests, and deploys a trained machine learning model as an artifact.

A feature-branching workflow was employed to manage version control, demonstrating effective collaboration via branching, merging, and pull requests. The report includes details of the pipeline configuration, logs of successful runs, and challenges faced during the setup process. This work highlights the value of CI/CD in improving development efficiency and ensuring reliability in machine learning workflows. The repository for the project is provided, including commit history, branch structures, and the pipeline YAML configuration.

1. **Git Version Control**

Version control systems (VCS) are used to track changes in files and collaborate with multiple developers. Git is a distributed version control system that allows for tracking changes, collaborating, and maintaining different versions of a project. It helps manage code, track history, and resolve conflicts during collaboration.

* 1. **Git Repository**

To begin version control for a project, initialize a Git repository in your local project directory. The command “git init” initializes a git directory in the project folder, marking it as a Git repository.

* 1. **Branching**

Branching allows you to work on different versions of a project independently. The default branch is main (or master), but other branches can be created to work on features, bug fixes, or experiments without affecting the main codebase.

* 1. **Pull Request**

A Pull Request is a way of submitting contributions to a repository. It allows collaborators to propose changes to the codebase and review them before merging.

1. **CI/CD Pipeline**

The CI/CD pipeline implementation successfully automated the linting, testing, and deployment stages of the machine learning project. This automation improves code quality, ensures reliability, and accelerates development. The experience highlights the importance of CI/CD in modern MLOps workflows, laying a foundation for more complex pipelines in future projects.

**Tools and Technologies Used**

* Version Control System: Git
* CI/CD Orchestrator: GitHub Actions
* Programming Language: Python
* Containerization: Docker
* Deployment Environment: AWS EC2
  1. **Pipeline Stages**

The pipeline has four stages:

* Linting
* Testing
* Build
* Deployment
  + 1. **Linting Stage**

The linting stage ensures that the code adheres to Python best practices and maintains a clean, error-free codebase. The process includes:

* Code Checkout: The pipeline begins by checking out the latest code from the repository to ensure it has the most up-to-date version.
* Environment Setup: Python 3.9 is set up as the runtime environment using GitHub Actions' setup tools.
* Dependency Installation: The pip package manager is upgraded, and the Flake8 tool is installed.
* Code Analysis with Flake8: The Flake8 tool performs two types of analysis:
  + Critical Errors: Identifies Python syntax errors and undefined names.
  + Code Complexity and Style: Checks for line lengths exceeding 127 characters and ensures the code complexity remains manageable (maximum complexity set to 10). This stage stops the pipeline if critical errors are found. The process ensures that all submitted code maintains readability and adheres to coding standards.

Below in the config done for doing the linting using flake8.

**Config**

**Output**

* + 1. **Testing Stage**

The testing stage validates the functionality and correctness of the application using automated tests. It involves the following steps:

* Code Checkout: Retrieves the latest code for consistency.
* Environment Setup: Installs Python 3.9 and all dependencies listed in requirements.txt.
* Environment Variables: Sets paths for essential files, such as DATA\_FILE\_PATH for the dataset and PKL\_FILE\_PATH for the trained model.
* MLflow Server Initialization: Starts an MLflow server to facilitate model tracking during tests.

Executing Tests: Runs all test cases in the tests’s directory using Pytest. These tests validate the machine learning pipeline and its components, ensuring that data preprocessing, model inference, and other functions work as expected.

**Config**

**Output**

As part the testing pipeline, the corresponding environment variables for reading the test data is setup up and the tests are all successfully run. These tests include the following

* Data Testing – to ensure preprocessing is working correctly.
* Model testing – to ensure the model built has a minimum accuracy, f1 score, recall and precision.
* Prediction testing – to predict a few positive/negative/neutral texts to ensure the prediction is behaving correctly.
  + 1. **Build and Push Stage**

The build-and-push stage focuses on containerizing the application and preparing it for deployment. The steps are as follows:

* Code Checkout: Ensures the latest version of the code is used.
* Docker Authentication: Logs into Docker Hub securely using stored credentials.
* Image Creation: Builds a Docker image for the application and tags it as latest.
* Image Push: Uploads the Docker image to Docker Hub, making it available for deployment. This stage ensures the application is encapsulated in a container for portability and consistency.

**Config**

**Output**

* + 1. **Deployment stage**

The deployment stage automates the rollout of the application to a AKS. It includes:

* **Azure CLI** installed and authenticated.
* AKS cluster created and kubectl configured to connect to it.
* Docker image of the ML model is built and pushed to a container
* YAML deployment and service files for Kubernetes.

1. **Validation of Deployment**

Now we can verify the deployment and validate the prediction using the latest code or model changes.

* 1. **Access the service**
* Once deployed, the service will create a Load Balancer
* Get the external IP
  1. **Test the Deployment**
* Use the external IP to send requests to the ML mode

1. **Guthub branch details**