**Bank Marketing Prediction Report**

**Designed Solution**

The goal of this project is to build a system where users can input data (such as age, job, marital status, etc.) through a web interface, and the system predicts whether the client will subscribe to a term deposit.

1. **Model Training**

* Trained Support Vector Machine (SVM) and Logistic Regression (LR) models using a dataset in a python notebook.

1. **Web Application**

* A Flask-based web application collects user inputs and displays the prediction ("Yes" or "No") based on the trained model.

1. **Data Preprocessing**

* The user input is preprocessed (e.g., encoding categorical values, scaling numerical values) before being passed to the machine learning model for prediction.

1. **Model Integration**

* The preprocessed data is sent to the trained model, which predicts whether the client will subscribe to the term deposit.

1. **Deployment**

* The application is containerized using Docker and deployed on AWS ECS for scalability and reliability.

1. **CI/CD Pipeline**

* A CI/CD pipeline automates building, testing, and deploying the application, ensuring efficient and error-free updates.

**Solution Architecture**

The architecture of the Bank Marketing Prediction system is designed to efficiently handle user inputs, process them, and provide predictions on term deposit subscriptions.

Diagram -[Solution Architecture Diagram](https://drive.google.com/drive/u/0/folders/1DNd04I1KF25pBtN8kobAQcNq9LIeqP3S)

**Flow of Requests:**

**1. User Input**

* The user enters data (such as age, job, marital status, etc.) using inputs & dropdowns and submit it.

**2. Request to Application**

* The form data is sent via a POST request to the Flask web application running in the backend.
* Flask receives the input and preprocesses it.

**3. Model Prediction**

* The preprocessed data is passed to the trained machine learning model.
* The model predicts whether the client will subscribe to the term deposit.
* The prediction (e.g., "Yes" or "No") is returned.

**4. Return Prediction to User**

* The prediction result is sent back as a response to the user’s web browser.

**Data Flow:**

* **Input Data:** User-provided values (such as age, job, marital status, etc.) are sent from the web application.
* **Preprocessed Data:** After encoding and scaling, the data is passed to the trained model.
* **Prediction Result:** The model returns a prediction, which is then displayed in the web application.

**Deployment Architecture**

The solution is deployed on AWS ECS (Elastic Container Service) to manage and scale the web application efficiently. Several AWS services are integrated to ensure the application's reliability and performance.

Diagram -[Deployment Architecture Diagram](https://drive.google.com/drive/u/0/folders/1DNd04I1KF25pBtN8kobAQcNq9LIeqP3S)

**1. User Access**

* The user accesses the deployed web application through a public URL.
* The application is hosted on AWS ECS and exposed via an Elastic Load Balancer (ELB) to handle traffic distribution.

**2. ECS Container**

* The user’s request is routed to the ECS container running the web application.
* ECS automatically manages the container’s scaling and availability.

**3. Model Inference**

* The application accesses the pre-trained model, which is embedded within the container.
* The model processes the user input and makes a prediction.

**4. Response Back to User**

* The prediction result is returned through the Elastic Load Balancer (ELB) and displayed in the user’s browser.

**CI-CD Pipeline Process**

The CI/CD (Continuous Integration and Continuous Deployment) pipeline automates the process of building, testing, and deploying the web application to AWS ECS. Below are the steps in the pipeline.

Diagram - [CI-CD Pipeline Diagram](https://drive.google.com/drive/u/0/folders/1DNd04I1KF25pBtN8kobAQcNq9LIeqP3S)

**Process:**

1. **Code Commit**

* Developers push the latest code changes to a version control repository (GitHub).

1. **Continuous Integration (AWS CodeBuild)**

* The AWS CodeBuild workflow is triggered after code changes are detected.
* It pulls the latest code from GitHub.
* It runs automated tests, including unit tests and integration tests.
* After tests, it builds a Docker image of the application.

1. **Docker Image Push (AWS ECR):**

* The Docker image is tagged and pushed to AWS ECR (Elastic Container Registry) for storage.

1. **Continuous Deployment (AWS ECS):**

* AWS ECS (Elastic Container Service) fetches the latest Docker image from ECR.
* ECS updates the running container with the new version of the app.
* The application is automatically restarted, ensuring zero downtime.