**1. Project Overview**

* **Objective**: Implement a customer churn prediction model using containerized machine learning environments. The focus is on leveraging Docker and Kubernetes to create scalable and portable solutions, enabling rapid deployment across cloud infrastructure.
* **Key Technologies**:
  + **Docker** for containerization.
  + **Kubernetes** for container orchestration.
  + **Python** for machine learning model development.
  + **Flask** or **FastAPI** for exposing the model as an API.

**2. Architecture**

The architecture will include the following components:

**a. Machine Learning Pipeline**

* **Data Ingestion**: Data is collected from various customer interaction sources, which are stored in cloud databases or local files (CSV).
* **Data Preprocessing**: Handle missing values, normalization, and feature engineering.
* **Model Training**: Build a machine learning model (e.g., logistic regression, decision trees, or deep learning models) for customer churn prediction using scikit-learn, TensorFlow, or PyTorch.
* **Model Evaluation**: Evaluate model performance using accuracy, precision, recall, F1 score, etc.

**b. Containerization**

* **Docker Images**: Create a custom Docker image with Python, all necessary libraries, and the ML model bundled.
  + **Dockerfile** Example:

dockerfile

Copy code

FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt ./

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

CMD ["python", "app.py"]

* **Churn Model API**: Deploy the model behind a RESTful API using Flask or FastAPI. This API will handle requests to predict customer churn.

**c. Container Orchestration with Kubernetes**

* **Kubernetes Deployment**:
  + Create a Kubernetes deployment to manage container replicas, ensure availability, and scale when needed.
  + Use a **LoadBalancer** service type to expose the churn model API to external requests.
  + Define YAML configuration for deployment, service, and scaling.
  + **Example Kubernetes Deployment YAML**:

yaml

Copy code

apiVersion: apps/v1

kind: Deployment

metadata:

name: churn-model-deployment

spec:

replicas: 3

selector:

matchLabels:

app: churn-model

template:

metadata:

labels:

app: churn-model

spec:

containers:

- name: churn-model-container

image: your-dockerhub-repo/churn-model

ports:

- containerPort: 8501

---

apiVersion: v1

kind: Service

metadata:

name: churn-model-service

spec:

selector:

app: churn-model

ports:

- protocol: TCP

port: 80

targetPort: 8501

type: LoadBalancer

**3. Workflow**

1. **Development Phase**:
   * Build the ML model.
   * Test the API locally.
   * Containerize the application using Docker.
   * Push the container to DockerHub.
2. **Deployment Phase**:
   * Deploy the container on Kubernetes.
   * Expose the service using NodePort or LoadBalancer.
   * Monitor the application using Kubernetes logs and dashboards.

**4. Implementation Steps**

**a. Model Development and API**

* Develop a churn prediction model.
* Expose it via a Flask/ FastAPI app.

**b. Dockerization**

* Write a Dockerfile to containerize the application.
* Build the Docker image:

bash

Copy code

docker build -t churn-model .

* Test the container locally:

bash

Copy code

docker run -p 8501:8501 churn-model

**c. Push to DockerHub**

* Login to DockerHub:

bash

Copy code

docker login

* Tag and push the image:

bash

Copy code

docker tag churn-model your-dockerhub-repo/churn-model

docker push your-dockerhub-repo/churn-model

**d. Deploy on Kubernetes**

* Apply the deployment and service YAML files:

bash

Copy code

kubectl apply -f churn-model-deployment.yaml

**e. Monitoring and Scaling**

* Use the Kubernetes dashboard to monitor pods, services, and performance.
* Implement auto-scaling if necessary using Horizontal Pod Autoscaler (HPA).

**5. Results and Comparison**

* Test the deployed model with different customer data to predict churn.
* Evaluate model performance and scalability under different loads.
* Measure time for deployment, latency, and container scaling behavior.