Task 3: Fraud detection in a government agency (spotlight: MLOps)

Course: DLBDSMTP01 – Project: From Model to Production

Student: Dmitrii Shevchuk, Matriculation: 9213737

Tutor: Frank Passing

Code: https://github.com/shevchukum/fraud/

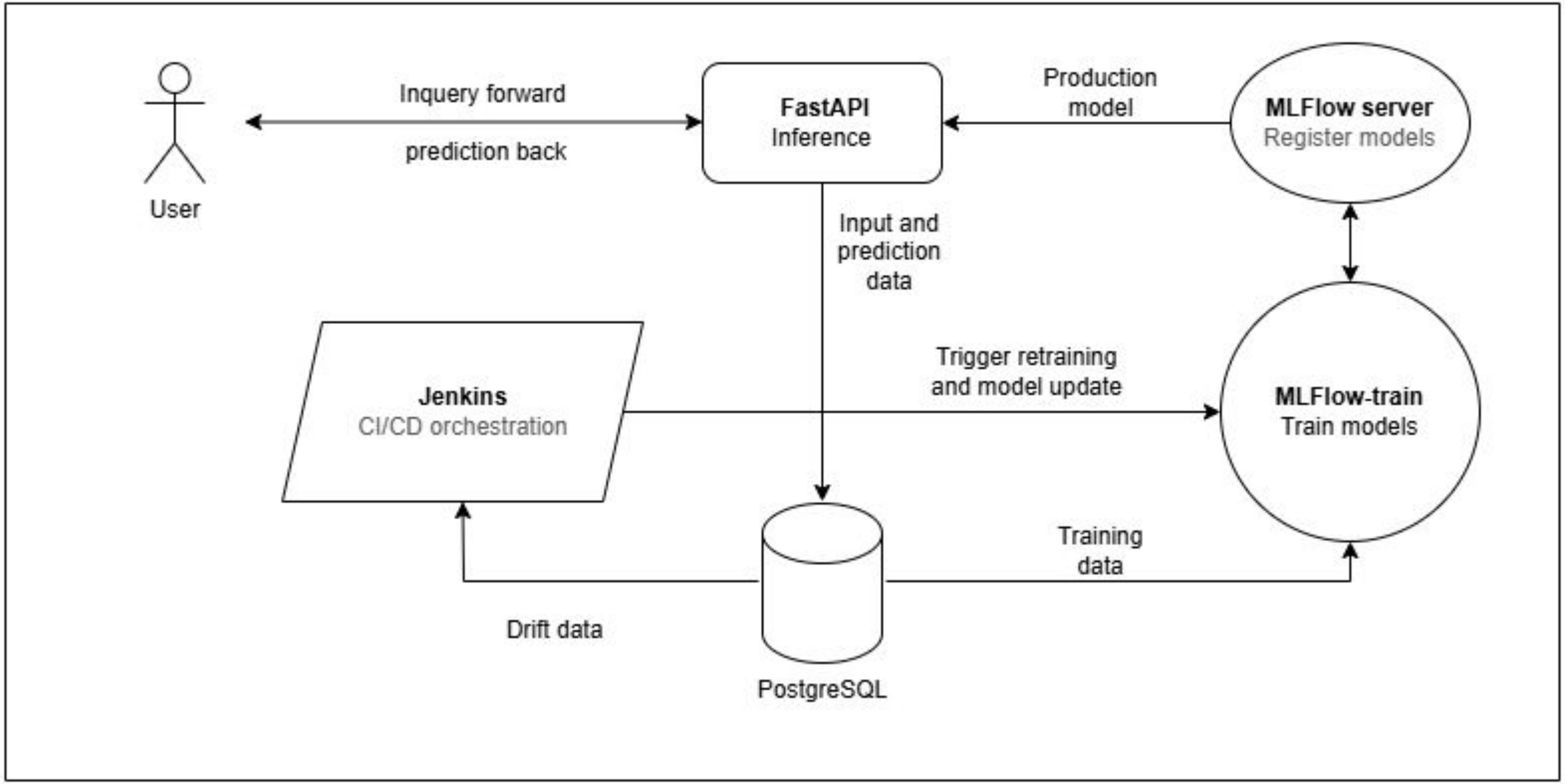
Presentation outline

1. Introduction
2. Conceptual architecture
3. Data collection
4. Simple fraud detection model
5. FastAPI
6. MLFlow
7. Jenkins
8. Tests
9. Conclusion

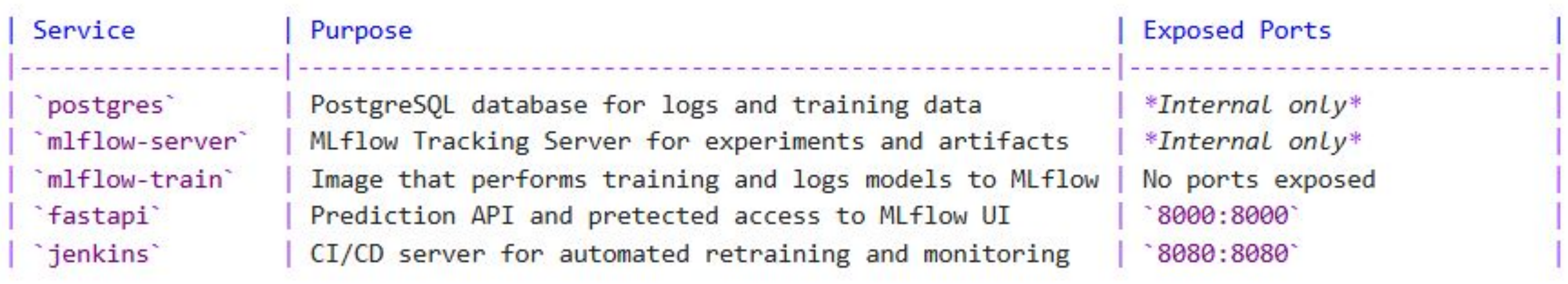
Introduction

1. **Background:** Our agency is focused on improving fertility rates. Research shows that married couples tend to have 50% more children than cohabiting ones. We designed a program offering low % loan up to 25K EUR for the first time marrying couples on starting-up their family.
2. **Problem:** increasing number of fraud applications when people divorce shortly after receiving the loan making it harder for real couples to get support due to limited amount of loans available each year.
3. **Solution:** we need to develop a fraud marriage scoring system, make it available through RESTful API and implement MLOps practices.

Conceptual architecture



Conceptual architecture



Service	Purpose	Exposed Ports
`postgres`	PostgreSQL database for logs and training data	*Internal only*
`mlflow-server`	MLflow Tracking Server for experiments and artifacts	*Internal only*
`mlflow-train`	Image that performs training and logs models to MLflow	No ports exposed
`fastapi`	Prediction API and pretected access to MLflow UI	`8000:8000`
`jenkins`	CI/CD server for automated retraining and monitoring	`8080:8080`

Data collection

Data: Auto Loan Default Prediction
https://www.kaggle.com/code/saurabhbagchi/auto-loan-default-prediction-wip/data

1. ‘Default’ -> ‘Divorce’, Filter ‘Credit_Amount’ < 25000, 'Client_Education' -> 'Education_Days', Scale down ‘Age_Days’: make people over 30 younger on 10 years
2. 10 features, ‘Divorce’, ‘Date’ -> 107K samples 2023-2024
3. Another 107K random samples with 20K new wave: age < 27y and 50% divorce rate in 2025-2026.
4. Dump to ‘Divorce_database_20250513.sql’ -> ‘initdb’ folder

Simple fraud detection model

1. XGBClassifier
2. Minimal grid_search
3. Target metric: Area Under the Precision-Recall Curve (AUC PR) - good for fraud detection (rare positive class)
4. Integrated to mlflow-train service

FastAPI

1. FastAPI is faster than Flask and uses Pydantic for input validation
2. Endpoints

localhost:8000/mlflow	Checks password and redirects to MLFlow server
localhost:8000/predict	Returns prediction on a single input
localhost:8000/batch_predict	Returns predictions on the input batch
localhost:8000/update-model	Updates the model from MLFlow server
localhost:8000/model-info	Returns data on the current model

MLFlow

1. MLFlow server registers models and logs artifacts in /mlflow/mlruns.
2. Accessible through localhost:8000/mlflow with password.
3. MLFlow-train - separate container: downloads training data from postgres database, trains the model, compares metric to the current production model and promotes it to production if the metric is better + logs artifacts through MLFlow server.

Jenkins

1. Accessible at localhost:8080 with password.
2. 3 pipelines:
   - Check the drift every 24 hours
   - Update the model every month
   - Run update 12 months ahead (test)
3. Contains drift_checker.py -> custom_jenkins container

Tests

1. Single request is checked by

```
curl -X POST \
    http://localhost:8000/predict \
   -H "Content-Type: application/json" \
   -d "$(cat request_1.json)"
```

```
probability
-----------
0.39910775423049927
```

Tests

2. Drift check

```
 curl -X POST \
     http://localhost:8000/batch_predict \
    -H "Content-Type: application/json" \
    -d "$(cat request_drift.json)"
```

And run drift-monitor-pipeline in Jenkins. It will find the drift in Age_Days, retrains the model and promotes it to production. Check localhost:8000/mlflow and localhost:8000/model-info to see that model is updated.

Tests

3. Start test-retraining pipeline in Jenkins -> 12 consecutive model training and update processes. Check localhost:8000/mlflow and localhost:8000/model-info to see that model is updated.

Conclusion

Integrating a predictive model into an application or service challenges

- Model version control
- Dependency management
- Input validation and preprocessing
- Performance and latency (memory/CPU/GPU)
- Error handling and logging
- Security and access control
- Model drift
- Data privacy and compliance (encryption, anonymization, and auditability)