

**Tugas Besar IF4054 Pengoperasian Sistem Perangkat Lunak**  
**Customer Churn Prediction Ops Pipeline**  
**Semester 1 Tahun 2024/2025**



Disusun oleh:  
Kelompok

Ghazi Akmal Fauzan	13521058
Yanuar Sano Nur Rasyid	13521110
Ahmad Ghulam Ilham	13521118
Muhammad Habibi Husni	13521169

**PROGRAM STUDI TEKNIK INFORMATIKA**  
**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**  
**INSTITUT TEKNOLOGI BANDUNG**  
**2024**

## Problem Statement

Industri telekomunikasi dapat mendapatkan kerugian finansial akibat churn pelanggan yang menurun. Oleh karena itu, prediksi churn pelanggan secara akurat menjadi hal yang penting dilakukan sebagai bagian dari proses bisnis perusahaan. Prediksi tersebut dapat dibuat dengan menggunakan teknologi *machine learning* untuk membuat model prediksi churn pelanggan dari data mengenai pelanggan yang meliputi *service* yang didaftarkan, informasi akun, dan demographic pelanggan.

Tantangan utama adalah pembuatan end to end pipeline serta workflow untuk memprediksi churn pelanggan. Alur dari *workflow* termasuk data preprocessing, model training, evaluation, serta deployment tracking. Selain itu, diperlukan pipeline dan workflow untuk pembersihan data (*data cleanup*) dengan menangani data null, string kosong, atau nilai-nilai yang tidak valid, pengembangan alur kerja simulasi *drift* (perubahan distribusi data), serta pemantauan *drift* yang terjadi dan secara otomatis menyesuaikan model untuk pelatihan ulang (*retraining*).

## Dataset and Method

Dataset yang akan digunakan dalam proyek ini adalah Telco Customer Churn yang tersedia di Kaggle (<https://www.kaggle.com/datasets/blastchar/telco-customer-churn/code>). Dataset ini berisi informasi pelanggan, seperti demografi, layanan yang digunakan, serta rincian kontrak dan pembayaran. Variabel target (*dependent variable*) dalam dataset ini adalah Churn yang menunjukkan apakah seorang pelanggan telah berhenti menggunakan layanan perusahaan.

Dalam mendeteksi *data drift*, proyek ini menggunakan Population Stability Index (PSI). PSI berfungsi untuk mengukur perubahan distribusi data dari waktu ke waktu dengan membandingkan distribusi data aktual terhadap distribusi referensi. Jika nilai PSI melebihi ambang batas tertentu, ini mengindikasikan adanya *drift* yang dapat mempengaruhi kinerja model prediksi. Dengan metode ini, sistem dapat memantau perubahan distribusi data secara otomatis dan memutuskan kapan model perlu dilakukan *retraining* untuk menjaga akurasi prediksi.

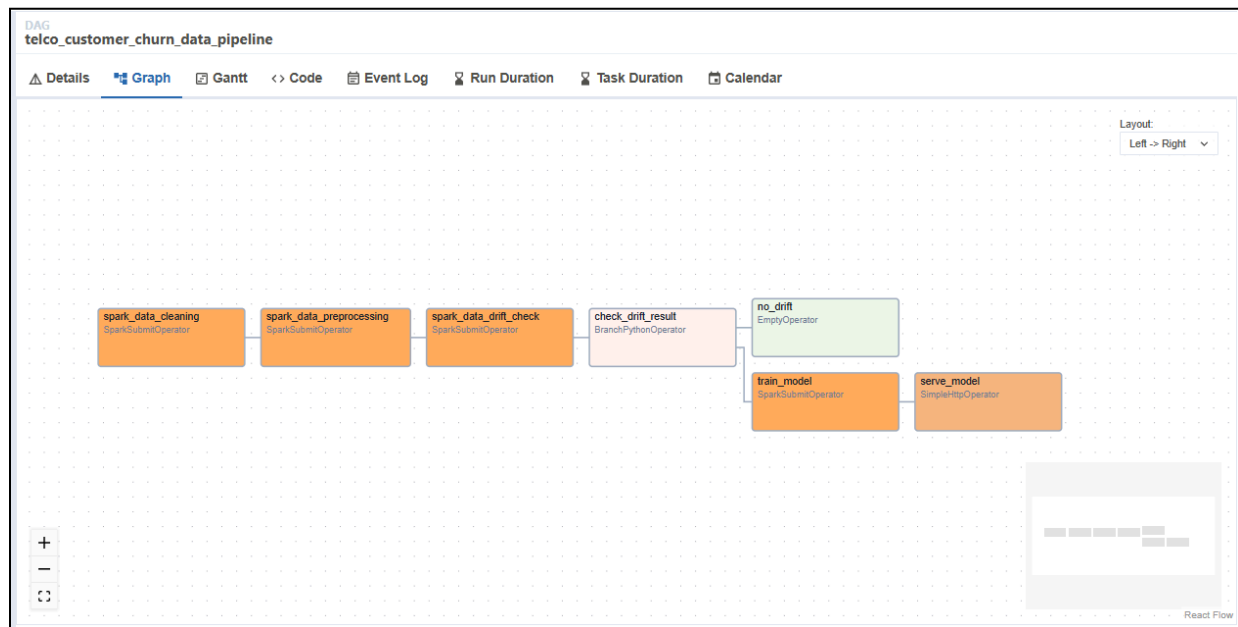
## Tools

- **Apache Spark** digunakan untuk memproses data besar (*big data processing*) dan *feature engineering*.
- **Apache Airflow** digunakan untuk mengorkestrasi dan menjadwalkan alur kerja (*pipeline workflows*).
- **MLflow** digunakan untuk melacak eksperimen, mengelola model, dan mendukung proses deployment.
- **Docker** digunakan untuk membuat *container* aplikasi yang ringan dan portable
- **GitLab** digunakan untuk sistem *version control* dan CI/CD (*Continuous Integration and Continuous Deployment*).

## Pipeline and Workflow

### Airflow

Airflow digunakan untuk membuat DAG yang menggambarkan Task yang akan dijalankan beserta alurnya dalam Airflow. Dalam proses pengolahan data Telco Customer Churn ada beberapa tahapan yang dilakukan yaitu pembersihan data, preprocessing, pemeriksaan drift data, dan penanganan drift jika terdeteksi.



### Data Cleaning

Dataset ini terdiri dari beberapa informasi attribute sebagai berikut:

1. Identifier:
  - customerID: Nomor ID pelanggan.
2. Informasi Demografi:
  - gender: Jenis kelamin pelanggan (pria atau wanita).
  - SeniorCitizen: Apakah pelanggan merupakan warga senior atau tidak.
  - Partner: Apakah pelanggan memiliki pasangan atau tidak.
  - Dependents: Apakah pelanggan memiliki tanggungan atau tidak.
3. Layanan yang Digunakan Pelanggan:
  - tenure: Jumlah bulan pelanggan telah menggunakan layanan.
  - PhoneService: Apakah pelanggan memiliki layanan telepon atau tidak.
  - MultipleLines: Apakah pelanggan memiliki beberapa jalur telepon atau tidak.
  - InternetService: Penyedia layanan internet pelanggan.
  - OnlineSecurity: Apakah pelanggan memiliki layanan keamanan online atau tidak.
  - OnlineBackup: Apakah pelanggan memiliki layanan pencadangan online atau tidak.

- DeviceProtection: Apakah pelanggan memiliki layanan perlindungan perangkat atau tidak.
  - TechSupport: Apakah pelanggan memiliki dukungan teknis atau tidak.
  - StreamingTV: Apakah pelanggan memiliki layanan streaming TV atau tidak.
  - StreamingMovies: Apakah pelanggan memiliki layanan streaming film atau tidak.
4. Informasi Akun Pelanggan:
- Contract: Jenis kontrak pelanggan.
  - PaperlessBilling: Apakah pelanggan menggunakan tagihan tanpa kertas atau tidak.
  - PaymentMethod: Metode pembayaran pelanggan.
  - MonthlyCharges: Jumlah biaya yang dibebankan ke pelanggan setiap bulan.
  - TotalCharges: Total jumlah biaya yang telah dibebankan ke pelanggan.
5. Target:
- Churn: Status churn, menunjukkan apakah pelanggan berhenti menggunakan layanan atau tidak.

Dataset tersebut memiliki beberapa data yang perlu dibersihkan yaitu kolom TotalCharges memiliki nilai null ketika diubah tipenya dari String menjadi Double. Hal ini terjadi karena tenure yang bernilai 0 memiliki TotalCharges ' ' spasi kosong yang menyebabkan nilai menjadi null saat di-convert. Solusi untuk kasus ini adalah dengan mengisi data TotalCharges 0 untuk yang memiliki tenure 0. Setelah dilakukan hal tersebut sudah tidak ada lagi data yang null.

```
df = df.withColumn("TotalCharges", col("TotalCharges").cast("double"))

df = df.withColumn(
    "TotalCharges",
    F.when(F.col("tenure") == 0, 0).otherwise(F.col("TotalCharges"))
)
```

Selain itu, dilakukan penyederhanaan data dengan melakukan drop kolom customerID yang tidak diperlukan serta mengubah nilai 'No phone service' dan 'No internet service' menjadi 'No' karena memiliki arti yang sama serta mengubah nilai dalam SeniorCitizen menjadi 'No' dan 'Yes' agar konsisten. Terakhir, dilakukan perubahan nama kolom menjadi snake\_case agar nama konsisten serta menyimpan data yang sudah dibersihkan dalam format csv.

```
columns_to_replace = [
    "MultipleLines", "OnlineSecurity", "OnlineBackup",
    "DeviceProtection", "TechSupport", "StreamingTV", "StreamingMovies"
]

for col_name in columns_to_replace:
    df = df.withColumn(
        col_name,
        when(col(col_name).isin(["No phone service", "No internet service"]),
        "No")
        .otherwise(col(col_name))
    )
```

```
df = df.withColumn(  
    'SeniorCitizen',  
    when(col('SeniorCitizen') == 0, 'No')  
    .when(col('SeniorCitizen') == 1, 'Yes')  
    .otherwise(col('SeniorCitizen'))  
)
```

## Data Preprocessing

Data dilakukan *preprocessing* dengan melakukan beberapa langkah.

Pertama melakukan pembagian dataset menjadi data latih (training data) dan data uji (testing data) menggunakan metode `randomSplit` dengan rasio 70:30.

Selanjutnya, data kategori diindeks menggunakan *StringIndexer* dan diubah menjadi representasi *one-hot encoding* menggunakan *OneHotEncoder*. Untuk data numerik, dilakukan proses normalisasi menggunakan *MinMaxScaler* setelah diolah menjadi vektor melalui *VectorAssembler*.

Setelah itu, setiap fitur numerik yang telah diskalakan diekstrak menggunakan *VectorSlicer* untuk menghasilkan kolom individu yang sesuai dengan fitur asli. Data kemudian diproses lebih lanjut untuk mengkonversi vektor menjadi nilai individu dengan UDF (*User-Defined Function*), serta mengubah fitur kategori yang telah diencode menjadi kolom terpisah dengan eksplosif matriks vektor.

Untuk mengatasi ketidakseimbangan data pada variabel target (*churn*), dilakukan oversampling menggunakan teknik SMOTE (*Synthetic Minority Oversampling Technique*). Data latih yang telah diproses diubah menjadi format Pandas untuk menerapkan SMOTE kemudian hasilnya dikonversi kembali menjadi *DataFrame* Spark agar dapat digunakan dalam proses pelatihan model selanjutnya.

```
data_train, data_test = df.randomSplit([0.7, 0.3], seed=42)  
  
indexers = [StringIndexer(inputCol=col, outputCol=f"{col}_index") for col in  
column_categorical + ["churn"]]  
index_features = [f"{col}_index" for col in column_categorical]  
  
encoders = [  
    OneHotEncoder(inputCol=f"{col}_index", outputCol=f"{col}_encoded",  
dropLast=True) for col in column_categorical  
]  
  
numerical_assembler = VectorAssembler(inputCols=column_numerical,  
outputCol="numerical_features")  
  
scaler = MinMaxScaler(inputCol="numerical_features",  
outputCol="scaled_numerical_features")  
  
slicers = [  
    VectorSlicer(inputCol="scaled_numerical_features",  
outputCol=f"scaled_{col}", indices=[i])  
    for i, col in enumerate(column_numerical)  
]
```

```

pipeline = Pipeline(stages=indexers + encoders + [numerical_assembler, scaler]
+ slicers)

pipeline_fit = pipeline.fit(data_train)

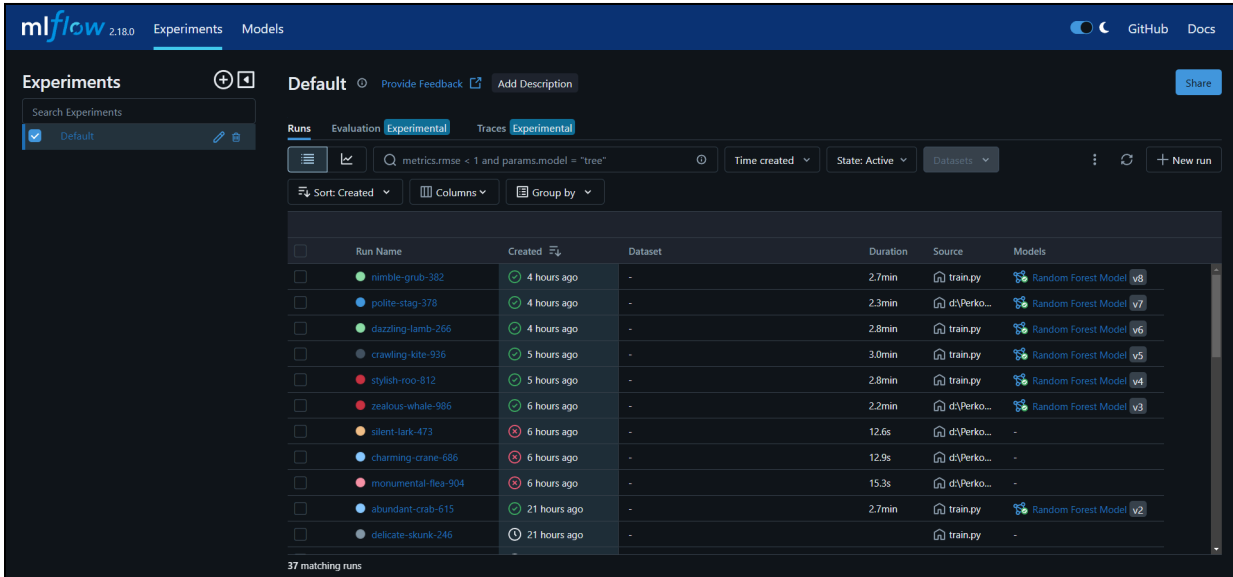
```

## Training Model

Setelah dilakukan preprocessing terhadap data, selanjutnya dilakukan model training. Proses dimulai dengan membaca data pelatihan dan pengujian menggunakan Spark dilanjutkan dengan pembuatan VectorAssembler untuk menggabungkan kolom fitur menjadi kolom tunggal bernama features. Selanjutnya, menggunakan library MLlib PySpark, berbagai model dilatih menggunakan data pelatihan dan diorganisasi dalam sebuah *pipeline* yang terdiri dari tahap *assembler* dan model. Setelah pelatihan selesai, model diuji pada data pengujian dan akurasi dievaluasi menggunakan MulticlassClassificationEvaluator. Berdasarkan hasil eksperimen, model yang terpilih adalah Random Forest Classifier karena memberikan kinerja terbaik dibandingkan model lainnya.

## Model Logging

Pada eksperimen, diperlukan sebuah sistem yang dapat melakukan pelacakan dan versioning terhadap model yang sudah dibangun. MLflow digunakan dalam kode ini untuk melacak, menyimpan, dan meregistrasi model pembelajaran mesin yang dikembangkan dengan Spark MLlib. Hasil evaluasi akurasi dicatat menggunakan logger. Model yang telah dilatih dicatat dalam server MLflow menggunakan `mlflow.spark.log_model` lengkap dengan signature yang dihasilkan dari struktur data input dan prediksi serta parameter model (`numTrees`) dan metrik akurasi. Model ini juga diregistrasi dalam MLflow Model Registry dengan nama "Random Forest Model". Dengan demikian, MLflow memastikan bahwa seluruh proses pelatihan, evaluasi, dan manajemen model terdokumentasi dan dapat diakses untuk pelacakan eksperimen maupun pengelolaan model produksi. Berikut adalah tampilan dashboard MLflow dari eksperimen.



Run Name	Created	Dataset	Duration	Source	Models
nimble-grub-382	4 hours ago	-	2.7min	train.py	Random Forest Model v8
polite-stag-378	4 hours ago	-	2.3min	d:\Perko...	Random Forest Model v7
dazzling-lamb-266	4 hours ago	-	2.8min	train.py	Random Forest Model v6
crawling-kite-936	5 hours ago	-	3.0min	train.py	Random Forest Model v5
stylish-roo-812	5 hours ago	-	2.8min	train.py	Random Forest Model v4
zealous-whale-986	6 hours ago	-	2.2min	d:\Perko...	Random Forest Model v3
silent-lark-473	6 hours ago	-	12.6s	-	-
charming-crane-686	6 hours ago	-	12.9s	d:\Perko...	-
monumental-flea-904	6 hours ago	-	15.3s	d:\Perko...	-
abundant-crab-615	21 hours ago	-	2.7min	train.py	Random Forest Model v2
delicate-skunk-246	21 hours ago	-	-	train.py	-

37 matching runs

nimble-grub-382

Model registered

OverviewModel metricsSystem metricsArtifacts

Created at

2025-01-10 19:09:41

Created by

default

Experiment ID

0

Status

Finished

Run ID

c9b29b9a79004abdbfba91adb8811b10

Duration

2.7min

Datasets used

—

Tags

Add

Source

train.py

Logged models

spark

Registered models

Random Forest Model

Parameters (1)

Search parameters

Parameter	Value
numTrees	10

Metrics (1)

Search metrics

Metric	Value
accuracy	0.7742899850523169

Model Deployment

Setelah model sudah dilatih, dilakukan deployment model. Pada tugas ini memanfaatkan fitur model serving yang disediakan oleh MLflow. Dengan menyiapkan sebuah server REST API sederhana, airflow dapat melakukan trigger model deployment kepada server MLFlow melalui SimpleHttpOperator. Berikut adalah tampilan contoh request pada model yang sudah di-deploy.

POSThttp://localhost:5001/invocationsSend

Status: 200 OKSize: 9.82 KBTime: 1.30 s

QueryHeadersAuthBodyTestsPre Run

ResponseHeadersCookiesResultsDocs

JSONXMLTextFormForm-encodeGraphQLBinary

JSON ContentFormat

1{

2  "inputs": [

3    {

4      "scaled\_tenure": 0.013888889,

5      "scaled\_monthly\_charges": 0.06826109,

6      "scaled\_total\_charges": 0.0029073784,

7      "gender\_category\_0": 0.0,

8      "senior\_citizen\_category\_0": 1.0,

9      "partner\_category\_0": 1.0,

10      "dependents\_category\_0": 1.0,

11      "phone\_service\_category\_0": 0.0,

12      "multiple\_lines\_category\_0": 1.0,

13      "internet\_service\_category\_0": 0.0,

14      "internet\_service\_category\_1": 1.0,

15      "online\_security\_category\_0": 1.0,

16      "online\_backup\_category\_0": 1.0,

17      "device\_protection\_category\_0": 1.0,

18      "tech\_support\_category\_0": 1.0,

19      "streaming\_tv\_category\_0": 1.0,

20      "streaming\_movies\_category\_0": 1.0,

21      "contract\_category\_0": 1.0,

22      "contract\_category\_1": 0.0,

23      "paperless\_billing\_category\_0": 0.0,

24      "payment\_method\_category\_0": 1.0,

25      "payment\_method\_category\_1": 0.0,

26      "payment\_method\_category\_2": 0.0

27    },

28    {

29      "scaled\_tenure": 0.013888889,

30      "scaled\_monthly\_charges": 0.066766314,

31      "scaled\_total\_charges": 0.0028901068,

32      "gender\_category\_0": 0.0,

33      "senior\_citizen\_category\_0": 1.0,

34      "partner\_category\_0": 1.0,

35      "dependents\_category\_0": 1.0,

36      "phone\_service\_category\_0": 0.0,

37      "multiple\_lines\_category\_0": 1.0,

38      "internet\_service\_category\_0": 0.0,

39      "internet\_service\_category\_1": 0.0,

40      "internet\_service\_category\_2": 0.0

41    }

42  ]

43  }

1{

2  "predictions": [

3    1.0,

4    1.0,

5    1.0,

6    1.0,

7    1.0,

8    1.0,

9    1.0,

10   1.0,

11   1.0,

12   1.0,

13   1.0,

14   1.0,

15   1.0,

16   1.0,

17   1.0,

18   1.0,

19   1.0,

20   1.0,

21   1.0,

22   1.0,

23   1.0,

24   1.0,

25   1.0,

26   0.0,

27   1.0,

28   1.0,

29   1.0,

30   1.0,

31   1.0,

32   1.0,

33   1.0,

34   1.0,

35   1.0,

36   1.0,

37   1.0,

38   1.0,

39   1.0,

40   1.0,

41   1.0,

42   1.0,

43  ]

## Drift Detection

Algoritma ini menggunakan PSI untuk mendeteksi perubahan distribusi data antara dua versi dataset, yaitu versi terbaru (*latest version*) dan versi sebelumnya (*previous version*). Berikut langkah-langkahnya:

1. **Identifikasi Versi Dataset:** Algoritma membaca file dataset di direktori tertentu, menentukan dua versi terakhir berdasarkan pola nama file, seperti `clean1.csv` dan `clean2.csv`. Jika kurang dari dua versi ditemukan, proses dihentikan.
2. **Memuat Dataset:** Dataset terbaru dan sebelumnya dimuat menggunakan Pandas, dengan fokus pada kolom numerik untuk analisis.
3. **Pembuatan Bins dan Distribusi:** Nilai dalam setiap kolom dibagi menjadi beberapa *bins* (misalnya 10 *bins*) untuk menghitung distribusi relatif pada kedua dataset.
4. **Perhitungan PSI:** PSI dihitung menggunakan rumus:

$$PSI = \sum_{i=1}^n (p_i - q_i) \cdot \ln \left( \frac{p_i q_i}{right} \right)$$

di mana  $p_i$  dan  $q_i$  adalah distribusi relatif pada *bin* ke- $i$  dari dataset sebelumnya dan terbaru.

5. **Deteksi Drift:** Jika PSI pada suatu kolom melebihi ambang batas (misalnya 0,1), kolom tersebut dianggap mengalami *drift*. Jika ada satu kolom saja yang mengalami *drift*, maka dataset dinyatakan mengalami perubahan signifikan.

Hasil deteksi *drift* dicatat dalam file teks, dengan nilai **1** menunjukkan adanya *drift* dan **0** menunjukkan tidak ada *drift*. Algoritma ini memberikan analisis kuantitatif yang jelas untuk mendeteksi perubahan distribusi data, meskipun hanya berlaku untuk kolom numerik dan bergantung pada jumlah *bins* yang dipilih.

```
# Function to calculate PSI (Population Stability Index)
def calculate_psi(base_distribution, new_distribution, bins=10):
    psi = 0
    for base_bin, new_bin in zip(base_distribution, new_distribution):
        if base_bin > 0 and new_bin > 0: # Avoid division by zero and log(0)
            psi += (new_bin - base_bin) * np.log(new_bin / base_bin)
    return psi

def calculate_column_psi(latest_col, previous_col, bins=10):
```



```

    # Create bins for both distributions
    bin_edges = pd.cut(pd.concat([latest_col, previous_col]), bins=bins,
retbins=True)[1]

    # Calculate distributions for each dataset
    latest_distribution = pd.cut(latest_col,
bins=bin_edges).value_counts(normalize=True, sort=False)
    previous_distribution = pd.cut(previous_col,
bins=bin_edges).value_counts(normalize=True, sort=False)

    # Calculate PSI
    return calculate_psi(previous_distribution, latest_distribution)

base_path = "/opt/data/versioning/"
base_name = "clean"
drift_result_file = "/opt/data/drift_result.txt"

latest_version, previous_version = get_latest_versions(base_path, base_name)

# Check if the latest version is available
if latest_version is None or previous_version is None:
    # Save drift result
    with open(drift_result_file, "w") as f:
        f.write("1")
    print("Drift check completed. Drift detected: True")
else:
    # Paths for the two files to compare
    latest_file = f"{base_path}{base_name}{latest_version}.csv"
    previous_file = f"{base_path}{base_name}{previous_version}.csv"

    # Load the CSV files into Pandas DataFrames
    latest_df = pd.read_csv(latest_file)
    previous_df = pd.read_csv(previous_file)

    drift_detected = False
    psi_threshold = 0.1 # Example threshold for PSI (adjust as necessary)

    # Iterate over numeric columns to calculate PSI
    for column in latest_df.select_dtypes(include=['number']).columns:
        latest_col = latest_df[column].dropna()
        previous_col = previous_df[column].dropna()

        # Ensure both columns are non-empty
        if not latest_col.empty and not previous_col.empty:
            column_psi = calculate_column_psi(latest_col, previous_col)
            print(f"PSI for column '{column}': {column_psi}")

            # Check if PSI exceeds the threshold
            if column_psi > psi_threshold:
                drift_detected = True
                break

```

## CI/CD




































*Continuous Integration (CI)* dan *Continuous Deployment (CD)* dilakukan dengan mengimplementasikan sebuah *pipeline CI/CD* yang mencakup langkah-langkah berikut.

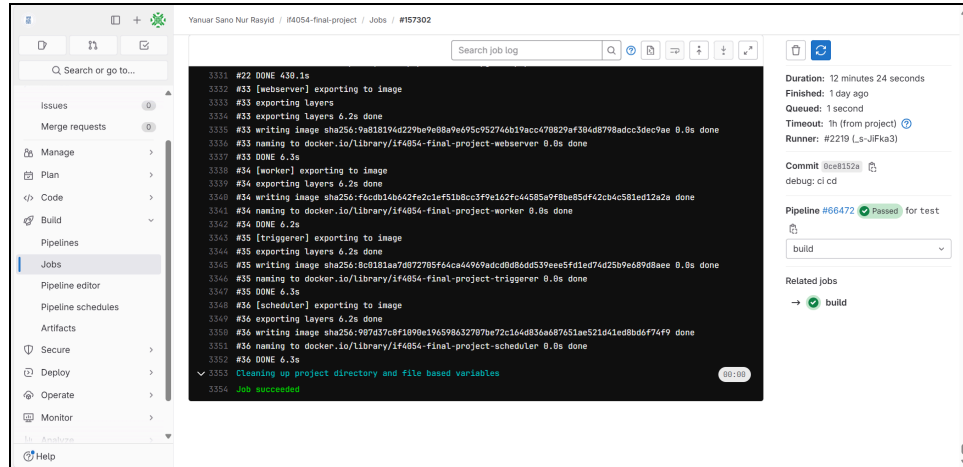
- Menyusun *pipeline* dengan tahap *build* dan *deploy*.
- Membuat koneksi *Apache Spark* di *Airflow* secara otomatis.
- Membuat koneksi *MLflow* di *Airflow* secara otomatis.
- Memastikan *DAG (Directed Acyclic Graph)* *Airflow* dijalankan hingga selesai sebelum *pipeline GitLab* selesai.

*CI/CD* dilakukan setiap kali terdapat perubahan pada repositori yang di-*push* pada *branch main* untuk memastikan integrasi dan pengujian dilakukan saat perubahan diterapkan. Berikut adalah langkah-langkah yang diimplementasikan dalam *GitLab CI/CD* berdasarkan file *.gitlab-ci.yml*.

### 1. Tahapan *Build*

- *Docker Compose*:
  - Menjalankan perintah ***docker-compose down -v*** untuk membersihkan kontainer lama.
  - Membangun ulang kontainer dengan perintah ***docker-compose build --no-cache***.
- *Environment Variables*:
  - Memastikan variabel lingkungan yang relevan seperti *MYSQL\_DATABASE*, *AWS\_ACCESS\_KEY\_ID*, dll., dikonfigurasi dengan benar pada bagian *GitLab Settings > CI/CD > Variables*.

CI/CD Variables </> 7				Reveal values	Add variable
Key ↑	Value	Environments	Actions		
AIRFLOW_UID  <a href="#">Expanded</a>	***** 	All (default) 	 		
AWS_ACCESS_KEY_ID  <a href="#">Expanded</a>	***** 	All (default) 	 		
AWS_SECRET_ACCESS_KEY  <a href="#">Expanded</a>	***** 	All (default) 	 		
MYSQL_DATABASE  <a href="#">Expanded</a>	***** 	All (default) 	 		
MYSQL_PASSWORD  <a href="#">Expanded</a>	***** 	All (default) 	 		
MYSQL_ROOT_PASSWORD  <a href="#">Expanded</a>	***** 	All (default) 	 		
MYSQL_USER  <a href="#">Expanded</a>	***** 	All (default) 	 		



## 2. Tahapan *Deploy*

- *Docker Compose*:
  - Menjalankan kontainer dengan perintah ***docker-compose up -d***.
- *Airflow Configuration*:
  - Menambahkan koneksi *Spark* di *Airflow* menggunakan perintah berikut.

```
docker-compose exec webserver airflow connections add
'spark_default' \
--conn-type 'spark' \
--conn-host 'spark://spark-master' \
--conn-port '7077'
```

- Menambahkan koneksi *MLflow* di *Airflow* menggunakan perintah berikut.

```
docker-compose exec webserver airflow connections add
'mlflow_server' \
--conn-type 'http' \
--conn-host 'mlflow_server' \
--conn-port '8080'
```

- Mengaktifkan dan menjalankan DAG *telco\_customer\_churn\_data\_pipeline* menggunakan perintah berikut.

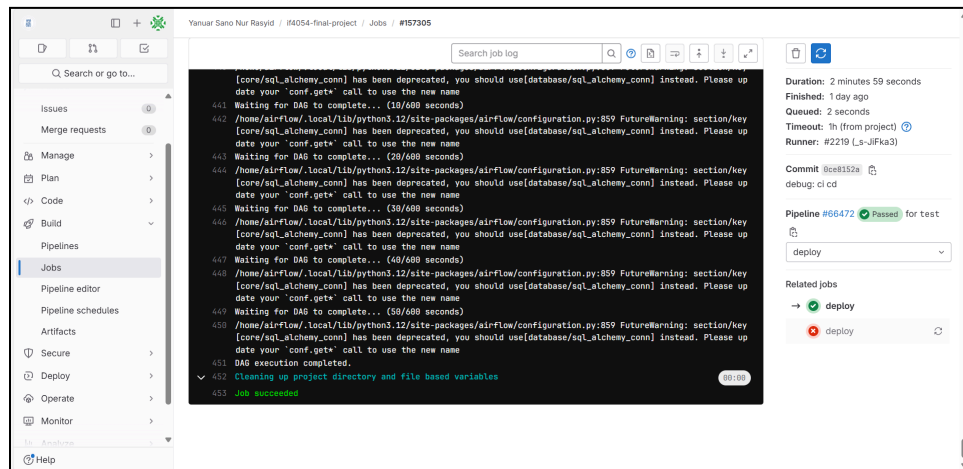
```
docker-compose exec webserver airflow dags unpause
telco_customer_churn_data_pipeline
docker-compose exec webserver airflow dags trigger
telco_customer_churn_data_pipeline
```

- *Monitoring DAG*:
  - Menggunakan polling untuk memantau status DAG hingga selesai sebagai berikut.

```

timeout=600
interval=10
elapsed=0
while true; do
    state=$(docker-compose exec webserver airflow dags list-runs -d
telco_customer_churn_data_pipeline | grep 'running')
    if [ -z "$state" ]; then
        echo "DAG execution completed."
        break
    fi
    if [ "$elapsed" -ge "$timeout" ]; then
        echo "Timeout reached. DAG execution still running."
        exit 1
    fi
    echo "Waiting for DAG to complete... ($elapsed/$timeout
seconds)"
    sleep "$interval"
    elapsed=$((elapsed + interval))
done

```



## Team Roles

NIM	Nama	Tugas
13521058	Ghazi Akmal Fauzan	DataOps, Drift Detection, Retraining
13521110	Yanuar Sano Nur Rasyid	DataOps, Cleaning, Preprocessing
13521118	Ahmad Ghulam Ilham	DevOps, Containerization, CI/CD
13521169	Muhammad Habibi Husni	MLOps, Training Model, Model Logging, Model Deployment

## Reference

- [Running Airflow in Docker — Airflow Documentation](#)
- [Get started with GitLab CI/CD | GitLab](#)
- [Build and push container images to the container registry | GitLab](#)
- [Command Line Interface and Environment Variables Reference — Airflow Documentation](#)

## Repository

[Yanuar Sano Nur Rasyid / if4054-final-project · GitLab](#)

[https://drive.google.com/file/d/1Xpo1\\_d6Q9GweOX6aFAwCcSJtMenr7ki9/view?usp=sharing](https://drive.google.com/file/d/1Xpo1_d6Q9GweOX6aFAwCcSJtMenr7ki9/view?usp=sharing)