Laporan Tugas Besar

IF4054 Pengoperasian Sistem Perangkat Lunak

Customer Churn Prediction Ops Pipeline

Anggota:

Farizki Kurniawan	13521082
Frankie Huang	13521092
Farhan Nabil Suryono	13521114
I Putu Bakta Hari Sudewa	13521150

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

Tahun Ajaran 2024/2025

A. Pendahuluan

a. Deskripsi

Pada tugas besar ini, mahasiswa ditugaskan untuk membuat *pipeline machine learning* dalam memprediksi *customer churn* dari data pengguna Telco yang dilampirkan pada Telco Customer Churn | Kaggle. Kapabilitas *pipeline* yang perlu diimplementasi adalah sebagai berikut.

i. Melakukan *data cleanup*, seperti membersihkan data dari NULL, data kosong, dan sebagainya.

ii. Melakukan simulasi *drift* pada data.

iii. Memonitor terjadinya *drift* dan melakukan *retraining* model prediksi secara otomatis.

b. *Tech Stacks*

i. Apache Airflow

Apache Airflow digunakan sebagai tool *orchestration* dan *scheduling* workflow yang akan digunakan pada tugas besar ini.

ii. Apache Spark

Apache Spark digunakan sebagai tool pemrosesan *big data* dan *feature engineering* melalui paradigma pemrograman Map-Reduce.

iii. Docker

Docker digunakan sebagai media *containerization* untuk mempermudah proses deployment dan integration antar komponen di berbagai device yang berbeda.

iv. MLflow

Sebuah *container* khusus yang digunakan pada tahap *experiment tracking*, *model management*, dan *deployment*.

v. Gitlab

Gitlab digunakan sebagai tool *version control system* yang disimpan secara *remote*.

c. System Requirements

Untuk menjalankan sistem ini diperlukan tools sebagai berikut:

- Git
- Docker

B. Implementasi

a. *Data Preprocessing*

Data preprocessing dilakukan dengan cara mengubah kolom dengan tipe string menjadi integer. Selain itu, kolom customerID dibuang karena tidak digunakan pada model.

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, when, udf
from pyspark.sql.types import IntegerType, FloatType


input_path = "/shared/data/customer_churn_old.csv"
output_path = "/shared/processed/customer_churn_cleaned.parquet"

def convert_to_binary(datum: str) -> int:
    if datum == 'No':
        return 0
    return 1

def convert_to_service_code(datum: str) -> int:
    if datum == 'No':
        return 0
    if datum == 'Yes':
        return 1
    return 2

def clean(input_path, output_path):
    spark = SparkSession \
        .builder \
        .appName("Customer Churn Data Cleaning") \
        .getOrCreate()

    data = spark.read.csv(input_path, header=True, inferSchema=True)

    data = data.drop("customerID")
    data = data.dropna().dropDuplicates()
```

```
    data = data.withColumn('gender', when(col('gender') == 'Female',
0).otherwise(1))

    data = data.withColumn('MultipleLines', when(col('MultipleLines')
== 'No', 0)
                                            .when(col('MultipleLines') ==
'Yes', 1)
                                            .otherwise(2))

    data = data.withColumn('InternetService',
when(col('InternetService') == 'No', 0)
                                             .when(col('InternetService')
== 'DSL', 1)
                                             .otherwise(2))

    data = data.withColumn('PaymentMethod', when(col('PaymentMethod')
== 'Bank transfer (automatic)', 0)
                                            .when(col('PaymentMethod') ==
'Credit card (automatic)', 1)
                                            .when(col('PaymentMethod') ==
'Electronic check', 2)
                                            .otherwise(3))

    data = data.withColumn('Contract', when(col('Contract') ==
o'Month-to-month', 0)
                                       .when(col('Contract') == 'One
year', 1)
                                       .otherwise(2))

    data = data.withColumn('MonthlyCharges',
col('MonthlyCharges').cast(FloatType()))
    data = data.withColumn('TotalCharges',
col('TotalCharges').cast(FloatType()))

    binary_columns = ['Partner', 'Dependents', 'PhoneService',
'PaperlessBilling', 'Churn']

    for column in binary_columns:
        data = data.withColumn(column, binary_udf(col(column)))

    trinary_columns = ['OnlineSecurity', 'OnlineBackup',
'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies']

    for column in trinary_columns:
```

```
        data = data.withColumn(column, service_code_udf(col(column)))

    data = data.dropna().dropDuplicates()
    data.write.parquet(output_path, mode="overwrite")


if __name__ == "__main__":
    binary_udf = udf(convert_to_binary, IntegerType())
    service_code_udf = udf(convert_to_service_code, IntegerType())

    clean(input_path, output_path)
```

b. *Model Training*

Model training dilakukan menggunakan pustaka sklearn pada Python dengan model yang digunakan adalah Random Forest Classifier. Berikut adalah kode yang digunakan untuk melatih model. Pada kode ini, model juga di-*'deploy'* pada direktori yang ada.

```
import mlflow
import mlflow.sklearn
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

data_path =
"/shared/processed/customer_churn_cleaned.parquet"

def train_model(data_path):
    data = pd.read_parquet(data_path)

    X = data.drop("Churn", axis=1)
    y = data["Churn"]

    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

```
    model = RandomForestClassifier(n_estimators=100,
random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)

    mlflow.log_metric("accuracy", accuracy)
    mlflow.sklearn.log_model(model, "costumer_churn_model")
if __name__ == "__main__":
    mlflow.set_experiment("Customer Churn Model Training")
    with mlflow.start_run():
        train_model(data_path)
```

c. *Drift Detection*

Drift Detection dilakukan untuk mendeteksi apabila terdapat *drift* dari data yang lama. Mekanisme perhitungan drift dilakukan dengan menggunakan PSI.

```
import sys
import mlflow
import mlflow.sklearn
import pandas as pd
import numpy as np
from airflow.models import TaskInstance
from airflow import settings
from airflow.models import DagBag
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

old_data_path = f"/shared/data/customer_churn_old.csv"
new_data_path = f"/shared/data/customer_churn_new.csv"

def calculate_psi(expected, actual, buckets=10):
    breakpoints = np.linspace(np.min(expected), np.max(expected),
buckets + 1)

    expected_dist = np.histogram(expected, bins=breakpoints,
density=True)[0]
    actual_dist = np.histogram(actual, bins=breakpoints,
density=True)[0]
```

```
    expected_dist += 1e-6
    actual_dist += 1e-6

    expected_dist /= expected_dist.sum()
    actual_dist /= actual_dist.sum()

    psi = np.sum((expected_dist - actual_dist) * np.log(expected_dist
/ actual_dist))

    return psi

def run_drift_detection():
    run_id = sys.argv[1]

    dag_bag = DagBag()

    data = pd.read_csv(old_data_path)

    X = data.drop("Churn", axis=1)
    y = data["Churn"]

    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

    new_data = pd.read_csv(new_data_path)
    new_data = new_data[X_train.columns]

    dag = dag_bag.get_dag("drift_detection_dag")
    currentTask = dag.get_task("drift_detection_task")
    session = settings.Session()

    # Iterate through numerical columns
    for col in X_train.select_dtypes(include=np.number).columns:
        psi = calculate_psi(X_train[col].values,
new_data[col].values)
        if psi > 0.1:
            # send true
            ti = TaskInstance(task=currentTask, run_id=run_id)
            ti.set_state('success', session)
            ti.xcom_push(key="psi_result", value="true")
            break
    else:
        # send false
        ti = TaskInstance(task=currentTask, run_id=run_id)
```

```
        ti.set_state('success', session)
        ti.xcom_push(key="psi_result", value="false")


if __name__ == "__main__":
    run_drift_detection()
```

d. *Drift Simulation*

Drift Simulation dilakukan untuk mensimulasikan masuknya data baru yang biased agar dapat mensimulasikan model retraining secara ulang.

```
import random
import csv
import os
import string

column_values_pair = {
    'gender': ['Male', 'Female'],
    'SeniorCitizen': [0, 1],
    'Partner': ['Yes', 'No'],
    'Dependents': ['Yes', 'No'],
    'tenure': [0, 72],
    'PhoneService': ['Yes', 'No'],
    'MultipleLines': ['Yes', 'No', 'No phone service'],
    'InternetService': ['DSL', 'Fiber optic', 'No'],
    'OnlineSecurity': ['Yes', 'No', 'No internet service'],
    'OnlineBackup': ['Yes', 'No', 'No internet service'],
    'DeviceProtection': ['Yes', 'No', 'No internet service'],
    'TechSupport': ['Yes', 'No', 'No internet service'],
    'StreamingTV': ['Yes', 'No', 'No internet service'],
    'StreamingMovies': ['Yes', 'No', 'No internet service'],
    'Contract': ['Month-to-month', 'One year', 'Two year'],
    'PaperlessBilling': ['Yes', 'No'],
    'PaymentMethod': ['Bank transfer (automatic)', 'Credit card
(automatic)', 'Electronic check', 'Mailed check'],
    'MonthlyCharges': [15, 125],
    'TotalCharges': [15, 1000],
    'Churn': ['Yes', 'No'],
}

NEW_FILE_PATH = '/shared/data/customer_churn_new.csv'

OLD_FILE_PATH = '/shared/data/customer_churn_old.csv'
```

```
def generate_biased_data(min_val, max_val, mid_val, is_int):
    std_dev = (max_val - min_val) / 6

    value = random.gauss(mid_val, std_dev)
    value = max(min_val, min(value, max_val))

    if is_int:
        return round(value)
    return round(value, 2)

def generate_normal_data():
    customer_id = f"{random.randint(1000,
9999)}{''.join(random.choices(string.ascii_uppercase, k=5))}"

    gender = random.choice(column_values_pair['gender'])

    senior_citizen =
random.choice(column_values_pair['SeniorCitizen'])

    partner = random.choice(column_values_pair['Partner'])

    dependents = random.choice(column_values_pair['Dependents'])

    tenure = random.randrange(column_values_pair['tenure'][0],
column_values_pair['tenure'][1]+1)

    phone_service = random.choice(column_values_pair['PhoneService'])
    if phone_service == 'Yes':
        multiple_lines =
random.choice(column_values_pair['MultipleLines'])
    else:
        multiple_lines = column_values_pair['MultipleLines'][-1]

    internet_service =
random.choice(column_values_pair['InternetService'])
    if internet_service == 'No':
        online_security =  column_values_pair['OnlineSecurity'][-1]
        online_backup = column_values_pair['OnlineBackup'][-1]
        device_protection = column_values_pair['DeviceProtection'][-1]
        tech_support = column_values_pair['TechSupport'][-1]
        streaming_tv = column_values_pair['StreamingTV'][-1]
        streaming_movies = column_values_pair['StreamingMovies'][-1]
    else:
        online_security =
```

```
random.choice(column_values_pair['OnlineSecurity'])
        online_backup =
random.choice(column_values_pair['OnlineBackup'])
        device_protection =
random.choice(column_values_pair['DeviceProtection'])
        tech_support =
random.choice(column_values_pair['TechSupport'])
        streaming_tv =
random.choice(column_values_pair['StreamingTV'])
        streaming_movies =
random.choice(column_values_pair['StreamingMovies'])

    contract = random.choice(column_values_pair['Contract'])

    paperless_billing =
random.choice(column_values_pair['PaperlessBilling'])

    payment_method =
random.choice(column_values_pair['PaymentMethod'])

    monthly_charges =
random.randrange(column_values_pair['MonthlyCharges'][0],
column_values_pair['MonthlyCharges'][1] + 1)

    total_charges =
random.randrange(column_values_pair['TotalCharges'][0],
column_values_pair['TotalCharges'][1] + 1)

    churn = random.choice(column_values_pair['Churn'])

    return [
        customer_id, gender, senior_citizen, partner, dependents,
tenure, phone_service, multiple_lines, internet_service,
        online_security, online_backup, device_protection,
tech_support, streaming_tv, streaming_movies, contract,
        paperless_billing, payment_method, monthly_charges,
total_charges, churn
    ]

def drift_simulator(data_count):
    random_column =
random.choice(list(column_values_pair.keys())[:-1])

    data = []
```

```
    while len(data) < data_count:
        normal = generate_normal_data()

        if random_column == 'tenure':
            min_val = column_values_pair['tenure'][0]
            max_val = column_values_pair['tenure'][1]
            mid_val = random.randrange(min_val, max_val + 1)

            normal[5] = generate_biased_data(min_val, max_val,
mid_val, True)

        elif random_column == 'MonthlyCharges':
            min_val = column_values_pair['MonthlyCharges'][0]
            max_val = column_values_pair['MonthlyCharges'][1]
            mid_val = random.randrange(min_val, max_val + 1)

            normal[18] = generate_biased_data(min_val, max_val,
mid_val, False)
        elif random_column == 'TotalCharges':
            min_val = column_values_pair['TotalCharges'][0]
            max_val = column_values_pair['TotalCharges'][1]
            mid_val = random.randrange(min_val, max_val + 1)

            normal[19] = generate_biased_data(min_val, max_val,
mid_val, False)

        data.append(normal)

    if os.path.exists(NEW_FILE_PATH):
        with open(NEW_FILE_PATH, 'r') as f:
            f.readline()

            lines = f.readlines()

        with open(OLD_FILE_PATH, 'a') as f:
            f.write('\n')
            f.write(lines)

    with open(NEW_FILE_PATH, 'w', newline='') as f:
        writer = csv.writer(f)

        writer.writerow(['customerID'] +
list(column_values_pair.keys()))

        writer.writerows(data)
```

```
if __name__ == "__main__":
    drift_simulator(7000)
```

e. Orchestration

Orchestration dari beberapa modul yang ada dilakukan dengan menggunakan Airflow yang diprogram menggunakan bahasa Python.

```
from airflow import DAG
from airflow.operators.bash import BashOperator
from airflow.operators.python import BranchPythonOperator
from airflow.operators.dagrun_operator import TriggerDagRunOperator
from airflow.operators.python import PythonOperator
from airflow.providers.apache.spark.operators.spark_submit import
SparkSubmitOperator
from airflow.utils.dates import days_ago
from airflow.utils.trigger_rule import TriggerRule

with DAG(dag_id='churn_pipeline', start_date=days_ago(1),
schedule_interval='@daily') as dag:
    etl_task = SparkSubmitOperator(
        task_id='run_etl',
        application='/shared/scripts/clean.py',
        conn_id='spark-conn'
    )

    train_model = SparkSubmitOperator(
        task_id='train_model',
        application='/shared/scripts/model.py',
        conn_id='spark-conn'
    )

    etl_task >> train_model

def check_drift(ti):
    psi_result = ti.xcom_pull(task_ids='drift_detection_task',
key='psi_result')
    if psi_result == "true":
        return 'trigger_model_training'
    else:
        return 'no_drift_detected'

with DAG(dag_id='drift_detection_dag', start_date=days_ago(0),
```

```
schedule_interval='@daily') as dag:
    drift_detection_task = BashOperator(
        task_id='drift_detection_task',
        bash_command='python /shared/scripts/drift-detection.py {{
run_id }}',
        do_xcom_push=True
    )

    check_drift_task = BranchPythonOperator(
        task_id='check_drift',
        python_callable=check_drift,
    )

    trigger_model_training = TriggerDagRunOperator(
        task_id='trigger_model_training',
        trigger_dag_id='churn_pipeline'
    )

    no_drift_detected = BashOperator(
        task_id='no_drift_detected',
        bash_command='echo "No drift detected"',
        trigger_rule=TriggerRule.NONE_FAILED_MIN_ONE_SUCCESS
    )

    drift_detection_task >> check_drift_task >>
[trigger_model_training, no_drift_detected]

with DAG(dag_id='drift_simulator_dag', start_date=days_ago(0) ,
schedule_interval='@daily') as dag:
    simulate_drift = BashOperator(
        task_id='simulate_drift',
        bash_command='python /shared/scripts/drift-simulator.py'
    )
```

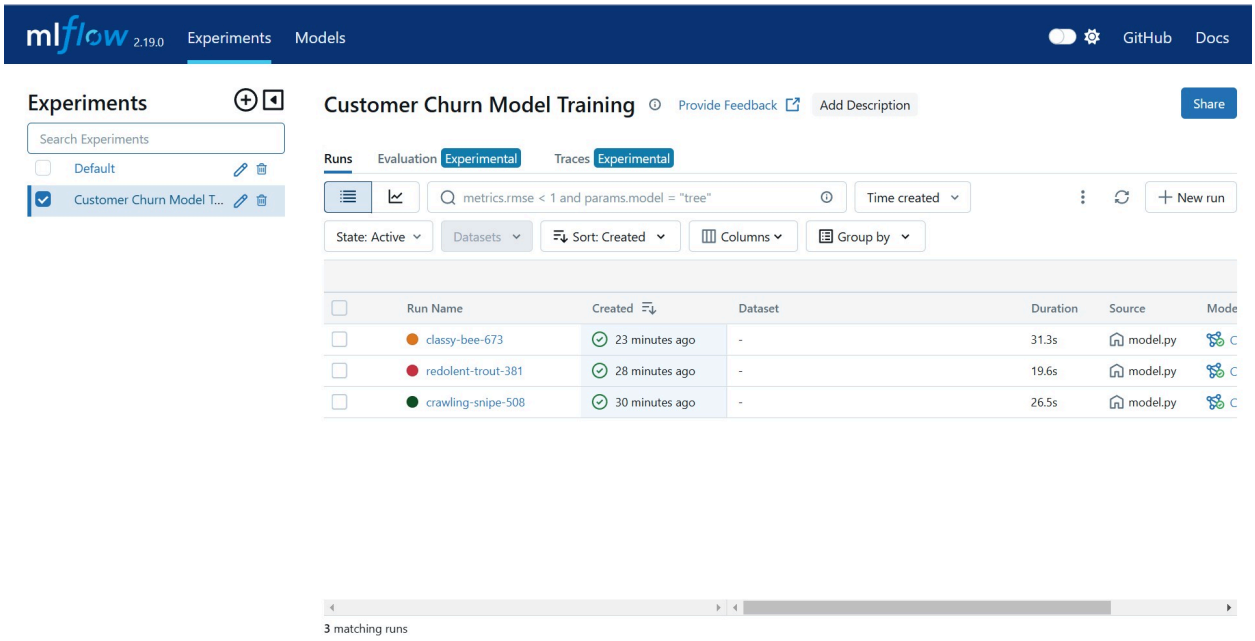
C. Hasil

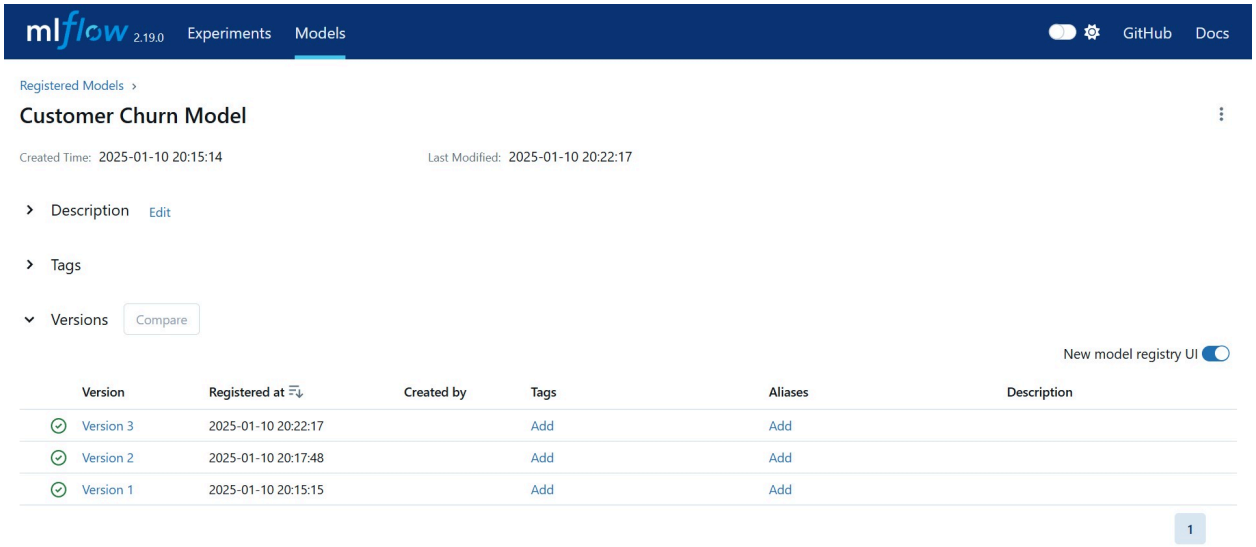
a. *Port* Docker

Port yang dijalankan oleh Docker adalah sebagai berikut.

b. *Screenshot* MLFlow

Berikut adalah cuplikan dari MLFlow yang dijalankan



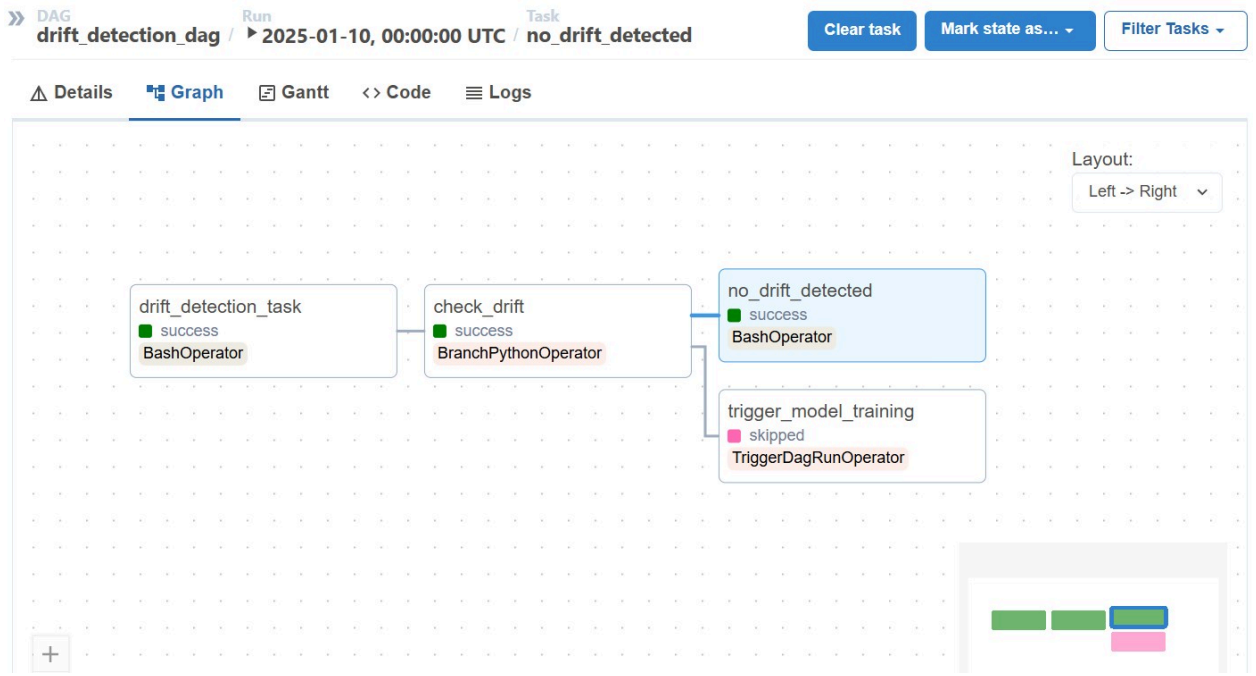


c. *Screenshot Pipeline* Airflow

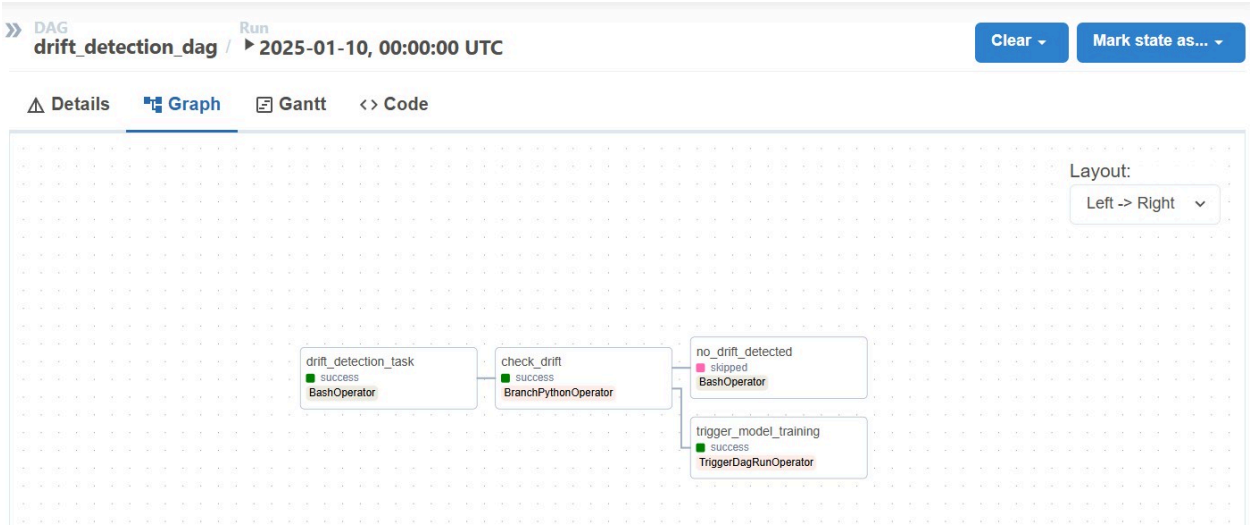
Berikut adalah cuplikan dari *pipeline* Airflow yang dijalankan.

Drift Simulator.

Drift Detector saat tidak terdeteksi ada *drift.*



Drift Detector saat terdeteksi ada *drift.*



Retraining saat terdeteksi ada drift.

State	Dag Id	Logical Date	Run Id	Run Type	Queued At	Start Date
running	churn_pipeline	2025-01-10, 13:19:42	manual__2025-01-10T13:19:42.756006+00:00	manual	2025-01-10, 13:19:42	2025-01-10, 13:19:43

Retraining yang terjadwal.

success	churn_pipeline	2025-01-09, 00:00:00	scheduled__2025-01-09T00:00:00+00:00	scheduled	2025-01-10, 13:12:44	2025-01-10, 13:12:44

D. Tautan Repositori

Situs	Tautan
GitLab	https://gitlab.informatika.org/xops-iga-gkub/customer-churn-prediction-ops-pipeline
GitHub	https://github.com/Altair1618/Customer-Churn-Prediction-Ops-Pipeline

E. Pembagian Tugas

Pembagian tugas pada tugas besar ini adalah sebagai berikut.

Nama	NIM	Pembagian Tugas
Farizki Kurniawan	13521082	Orchestration
Frankie Huang	13521092	Data Preprocessing, Drift Simulation
Farhan Nabil Suryono	13521114	**Setup Project**, Setup Tech Stacks, Model Training
I Putu Bakta Hari Sudewa	13521150	Drift Detection