Day 3

Plugins: Operators, Views, Hooks !!

Own Operators and Own Hooks

Customize Operators and Hooks : Extend Existing

How :

Airflow Plugin. - My Customize Class. - Python Module

Create a Plugin for Elastic Search:

Elastic Search:

Continuously Handle Large Volumes of Data
Scale Automatically
Be available to continuous data updates

Step 1 : yaml file / Setup
Step 2 : Create Connection
Step 3 : Create Plugin - Hook
Step 4 : Register Plugin
Step 5 : Write DAG

Step 1: Docker Instance

Provider: Elastic Search

Search, Analyze and Viz

Commands

docker-compose up -d

If Not visible in

docker-compose ps

Increase Memory Size to 8 GB and Retry

Check Provider :

docker-compose ps

docker exec -it day3-airflow-scheduler-1 /bin/bash

To check ES :

curl -X GET 'http://elastic:9200'

Elastic Search With Docker - Done

Step 2 : Create a Connection:

Name : elastic_default
Conn Type : http
Host :elastic
Port : 9200

Step 3 : Hook

Create Folders:

Plugins / hooks / elastic

Create file : elastic_hook.py

```
from airflow.plugins_manager import AirflowPlugin
from airflow.hooks.base import BaseHook


from elasticsearch import Elasticsearch

# Elastic Hook Inherited from Basehook
class ElasticHook(BaseHook):

    def __init__(self, conn_id='elastic_default', *args, **kwargs):

        # initialize basehook class
        super().__init__(*args, **kwargs)

        #Get Connection
        conn = self.get_connection(conn_id)

        # Get Connection Details of host, port and login
        conn_config = {}
        hosts = []

        if conn.host:
            hosts = conn.host.split(',')
        if conn.port:
            conn_config['port'] = int(conn.port)
        if conn.login:
            conn_config['http_auth'] = (conn.login, conn.password)

        # Initialize Elastic Search Connection Hook
        self.es = Elasticsearch(hosts, **conn_config)

        # Data is Stored in Form of Indexes
        self.index = conn.schema



    # Get info about Elastic Search
    def info(self):
```

```
        return self.es.info()

    # Set Index to store docs
    def set_index(self, index):
        self.index = index

    # Add data into ES in specific Index
    def add_doc(self, index, doc_type, doc):
        self.set_index(index)
        res = self.es.index(index=index, doc_type=doc_type, doc=doc)
        return res


# Register Elastic Hook for Airfloe Plugin Manager
class AirflowElasticPlugin(AirflowPlugin):
    name = 'elastic'
    hooks = [ElasticHook]
```

Step 4 : Register Plugin

Check For Plugins:

```
docker-compose -f docker-compose-es.yaml ps
```

```
docker exec -it day3-airflow-scheduler-1 /bin/bash
```

```
Airflow plugins
```

TO Register:

In elastic_hook.py

```
class AirflowElasticPlugin(AirlowPlugin):
        name =’elastic’
        hooks = [ElasticHook]
```

Step 5 : Write DAG

Create a dag to show Elastic Search Info

```python
from airflow import DAG
from airflow.operators.python import PythonOperator
from hooks.elastic.elastic_hook import ElasticHook


from datetime import datetime

def _print_es_info():
    hook = ElasticHook()
    print(hook.info())

with DAG('elastic_dag', start_date=datetime(2022, 1, 1), schedule_interval='@daily',
catchup=False) as dag:

    print_es_info = PythonOperator(
        task_id='print_es_info',
        python_callable=_print_es_info
    )
```

Plugins / Operators / timedoperator

```python
from airflow.operators.python_operator import PythonOperator
import time

class TimedPythonOperator(PythonOperator):
    def __init__(self, **kwargs) -> None:
        super().__init__(**kwargs)

    def execute(self, context) -> None:
        start = time.time()
        super().execute(context)
        done = time.time()
        print(f"Time Take for PythonOperator to complete execution {done - start} seconds")
```

DAG

```
from airflow import DAG
from ope rators.timed_python_operator import TimedPythonOperator
from datetime import datetime

def say_hello():
    for i in range(10):
        print("I am running using TimedPythonOperator custom operator")

with DAG( dag_id="test-customop-dag", start_date=datetime(2022, 8, 25),
schedule_interval=None
) as dag:
    first_task = TimedPythonOperator(
        task_id="test-customop-dag",
        python_callable=say_hello,
    )
first_task
```

Machine Learning Pipeline :

```
#python_functions.py

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

import pandas as pd
import numpy as np

def download_dataset_fn():

  iris = load_iris()
  iris = pd.DataFrame(
  data = np.c_[iris['data'], iris['target']],
  columns = iris['feature_names'] + ['target'])

  pd.DataFrame(iris).to_csv("iris_dataset.csv")


def data_processing_fn():

  final = pd.read_csv("iris_dataset.csv",index_col=0)
  cols = ["sepal length (cm)","sepal width (cm)","petal length (cm)","petal width (cm)"]
  final[cols] = final[cols].fillna(final[cols].mean())
  final.to_csv("clean_iris_dataset.csv")

def ml_training_RandomForest_fn(**kwargs):

  final = pd.read_csv("clean_iris_dataset.csv",index_col=0)
  X_train, X_test, y_train, y_test = train_test_split(final.iloc[:,0:4],final.iloc[:,-1], test_size=0.3)
  clf = RandomForestClassifier(n_estimators = 100)
  clf.fit(X_train, y_train)

  # performing predictions on the test dataset
  y_pred = clf.predict(X_test)

  # using metrics module for accuracy calculation
  print("ACCURACY OF THE MODEL: ", accuracy_score(y_test, y_pred))
  acc = accuracy_score(y_test, y_pred)

  kwargs['ti'].xcom_push(key='model_accuracy', value=acc )
```

```
def ml_training_Logisitic_fn(**kwargs):

  final = pd.read_csv("clean_iris_dataset.csv",index_col=0)
  X_train, X_test, y_train, y_test = train_test_split(final.iloc[:,0:4],final.iloc[:,-1], test_size=0.3)
  logistic_regression = LogisticRegression(multi_class="ovr")
  lr = logistic_regression.fit(X_train, y_train)

  y_pred = lr.predict(X_test)

  print("ACCURACY OF THE MODEL: ", accuracy_score(y_test, y_pred))
  acc = accuracy_score(y_test, y_pred)

  kwargs['ti'].xcom_push(key='model_accuracy', value=acc )


def identify_best_model_fn(**kwargs):
  ti = kwargs['ti']
  fetched_accuracies = ti.xcom_pull(key='model_accuracy',
task_ids=['ml_training_RandomForest', 'ml_training_Logisitic'])
  print(f'choose best model: {fetched_accuracies}'
```

Dag File :

```
from datetime import timedelta
# The DAG object; we'll need this to instantiate a DAG
from airflow import DAG

from airflow.operators.python import PythonOperator
from airflow.operators.empty import EmptyOperator
from airflow.operators.bash import BashOperator
from airflow.utils.dates import days_ago


from python_functions import download_dataset_fn,
from python_functions import data_processing_fn
from python_functions import ml_training_RandomForest_fn
from python_functions import ml_training_Logisitic_fn
from python_functions import identify_best_model_fn

args={
  'owner' : 'airflow',
  'retries': 1,
```

```
  'start_date':days_ago(1)  #1 means yesterday
}


with DAG(
  dag_id='airflow_ml_pipeline', ## Name of DAG run
  default_args=args,
  description='ML pipeline',
  schedule = None,
) as dag:

# Task 1 - Just a simple print statement
dummy_task = EmptyOperator(task_id='Starting_the_process', retries=2)


# Task 2 - Download the dataset
task_extract_data = PythonOperator(
task_id='download_dataset',
python_callable=download_dataset_fn
)


# Task 3 - Transform the data
task_process_data = PythonOperator(
task_id='data_processing',
python_callable=data_processing_fn
)

# Task 4_A - Train a ML Model using Random Forest
task_train_RF_model = PythonOperator(
task_id='ml_training_RandomForest',
python_callable=ml_training_RandomForest_fn
)

# Task 4_B -Train a ML Model using Logistic Regression
task_train_LR_model = PythonOperator(
task_id='ml_training_Logisitic',
python_callable=ml_training_Logisitic_fn
)

# Task 5 -Identify the best model of the two
task_identify_best_model = PythonOperator(
task_id='identify_best_model',
python_callable=identify_best_model_fn
)
```

```
# Define the workflow process
dummy_task >> task_extract_data >> task_process_data >>
[task_train_RF_model,task_train_LR_model] >> task_identify_best_model
```

Twitter:

```
import tweepy
import pandas as pd
import json
from datetime import datetime


def run_twitter_etl():

    access_key = ""
    access_secret = ""
    consumer_key = ""
    consumer_secret = ""


    # Twitter authentication
    auth = tweepy.OAuthHandler(access_key, access_secret)
    auth.set_access_token(consumer_key, consumer_secret)

    # # # Creating an API object
    api = tweepy.API(auth)
    tweets = api.user_timeline(screen_name='@elonmusk',
                            # 200 is the maximum allowed count
                            count=200,
                            include_rts = False,
                             # Necessary to keep full_text
                            # otherwise only the first 140 words are extracted
                            tweet_mode = 'extended'
                            )

    list = []
    for tweet in tweets:
        text = tweet._json["full_text"]

        refined_tweet = {"user": tweet.user.screen_name,
                        'text' : text,
                        'favorite_count' : tweet.favorite_count,
                        'retweet_count' : tweet.retweet_count,
                        'created_at' : tweet.created_at}

        list.append(refined_tweet)

    df = pd.DataFrame(list)
    df.to_csv('refined_tweets.csv')
```

Twitter Dag:

```
from datetime import timedelta
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
from airflow.utils.dates import days_ago
from datetime import datetime
from twitter_etl import run_twitter_etl

default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2020, 11, 8),
    'email': ['airflow@example.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=1)
}

dag = DAG(
    'twitter_dag',
    default_args=default_args,
    description='Our first DAG with ETL process!',
    schedule_interval=timedelta(days=1),
)

run_etl = PythonOperator(
    task_id='complete_twitter_etl',
    python_callable=run_twitter_etl,
    dag=dag,
)

run_etl
```

Create a data processing pipeline that takes data from a CSV file, transforms it, and stores the results in a PostgreSQL database.

The pipeline should run every day at midnight, and we want to be notified by email if there are any errors or if the pipeline fails.

Possible Steps:

1. A file_sensor task that waits for a CSV file to be uploaded to a specified directory.
2. A data_processing task that reads the CSV file, applies some transformations, and stores the results in a PostgreSQL database.
3. An email_notification task that sends an email notification if there are any errors or if the pipeline fails.

docker cp scheduler:/opt/airflow/airflow.cfg .