

## NYC-Taxi-Analytics-Practice-DataEngineer

Tags ที่ควรใส่ในทุก Resource:

Key: Project | Value: NYC-Taxi-Analytics-Practice-DataEngineer

Key: Environment | Value: Dev

Key: Owner | Value: Kamonphan

S3 Bucket: ชื่อ nyctaxi-dev-datalake-0116

สร้างโฟลเดอร์ raw/ และ processed/

IAM Role: ชื่อ nyctaxi-dev-role-glue

Attach Policies: AmazonS3FullAccess, AWSGlueServiceRole

IAM Role: ชื่อ nyctaxi-dev-role-func-ingest

Attach Policies: AmazonS3FullAccess, AWSLambdaBasicExecutionRole

Lambda: ชื่อ nyctaxi-dev-func-ingest

IAM Role: ชื่อ nyctaxi-dev-role-func-ingest

ใช้โค้ด Python 3.10

Timeout 5 นาที

Code: nyctaxi-dev-func-ingest.py

```
# ingest data มาก **ต้องต่อเน็ต** ของ ทั้ง 4 ประเภท
```

```
import boto3
```

```
import urllib.request
```

```
import json

# --- Config ---

S3_BUCKET = "mini-challenge-taxitype-0116"

TARGET_PREFIX = "raw"

# --- ตัวแปร ---

YEAR_MONTH = "2024-01" ##### กำหนด เดือน และ ปี ได้ที่นี่ #####
# ตั้ง "ปี" ออกมายก YEAR_MONTH

# ใช้ 'split' เพื่อแยก "2024-01" ที่เครื่องหมาย - แล้วเอาตัวแรก
YEAR = YEAR_MONTH.split('-')[0]

# [cite_start] รายการประเภท taxi
TAXI_TYPES = ["yellow", "green", "fhv", "fhvhv"]

# URL พื้นฐาน
BASE_URL = "https://d37ci6vzurychx.cloudfront.net/trip-data"

s3_client = boto3.client('s3')

def lambda_handler(event, context):

    print(f"Starting ingestion process for {YEAR_MONTH}...")

    results = []

    # วนลูปตามประเภท taxi
    for taxi_type in TAXI_TYPES:

        # สร้างชื่อไฟล์และ URL แบบไดนามิก
        filename = f"{taxi_type}_tripdata_{YEAR_MONTH}.parquet"
        url = f"{BASE_URL}/{filename}"

        # ดำเนินการอ่านไฟล์จาก URL หรือบันทึกใน S3 ตามเงื่อนไข
        # ตัวอย่าง: s3_client.download_file(Bucket=S3_BUCKET, Key=filename, Filename=filename)

        results.append({
            "taxi_type": taxi_type,
            "url": url,
            "filename": filename
        })

    return {
        "results": results
    }
```

```

s3_key = f'{TARGET_PREFIX}/{YEAR}/{taxi_type}/{filename}'

print(f"Processing: {filename}")
print(f" -> Source URL: {url}")
print(f" -> Target S3 Key: {s3_key}")

try:
    with urllib.request.urlopen(url) as response:
        s3_client.upload_fileobj(response, S3_BUCKET, s3_key)

    message = f"SUCCESS: Uploaded {filename} to s3://{S3_BUCKET}/{s3_key}"
    print(message)
    results.append(message)

except Exception as e:
    message = f"ERROR: Failed to process {filename}. Error: {str(e)}"
    print(message)
    results.append(message)

print("Ingestion process finished.")

return {
    'statusCode': 200,
    'body': json.dumps({
        'message': f'Ingestion for {YEAR_MONTH} finished.',
        'upload_results': results
    })
}

```

Glue Database: ชื่อ nyctaxi\_dev\_db\_catalog

Glue Crawler 1: ชื่อ nyctaxi-dev-crawl-raw

Target:

s3://nyctaxi-dev-datalake-0116/raw/2024/fhv/fhv\_tripdata\_2024-01.parquet

s3://nyctaxi-dev-datalake-0116/raw/2024/fhvfhv/fhvfhv\_tripdata\_2024-01.parquet

s3://nyctaxi-dev-datalake-0116/raw/2024/green/green\_tripdata\_2024-01.parquet

s3://nyctaxi-dev-datalake-0116/raw/2024/yellow/yellow\_tripdata\_2024-01.parquet

IAM Role: ชื่อ nyctaxi-dev-role-glue

Prefix: raw-

Glue Job: ชื่อ nyctaxi-dev-job-transform

IAM Role: ชื่อ nyctaxi-dev-role-glue

Code: nyctaxi-dev-job-transform.py

```
import sys
import boto3
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from pyspark.sql.functions import col, lit, to_timestamp

# --- 1. Setup ---

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
sc = SparkContext()
glueContext = GlueContext(sc)
```

```

spark = glueContext.spark_session

job = Job(glueContext)

job.init(args["JOB_NAME"], args)

# --- 2. Configuration ---

# ชื่อ Database
DATABASE_NAME = "nyctaxi_dev_db_catalog"

# MAPPING ชื่อตาราง
TABLE_MAP = {
    "yellow": "raw-yellow_tripdata_2024_01_parquet",
    "green": "raw-green_tripdata_2024_01_parquet",
    "fhv": "raw-fhv_tripdata_2024_01_parquet",
    "fhvhv": "raw-fhvvhv_tripdata_2024_01_parquet"
}

# Output Config

BUCKET_NAME = "nyctaxi-dev-datalake-0116"
OUTPUT_PREFIX = "processed/all_rides/"
OUTPUT_PATH = f"s3://{BUCKET_NAME}/{OUTPUT_PREFIX}"
NEW_FILENAME = "final_all_rides_Jan2024.parquet"

# ช่วงเวลา
START_TIME = "2024-01-01 00:00:00"
END_TIME = "2024-02-01 00:00:00"

print(f"Starting ETL job...")

print(f"Reading from database: {DATABASE_NAME}")

# --- 3. Read Data ---

try:

```

```

yellow_df = glueContext.create_dynamic_frame.from_catalog(database=DATABASE_NAME, table_name=TABLE_MAP["yellow"]).toDF()

green_df = glueContext.create_dynamic_frame.from_catalog(database=DATABASE_NAME, table_name=TABLE_MAP["green"]).toDF()

fhv_df = glueContext.create_dynamic_frame.from_catalog(database=DATABASE_NAME, table_name=TABLE_MAP["fhvhv"]).toDF()

fhvhv_df = glueContext.create_dynamic_frame.from_catalog(database=DATABASE_NAME, table_name=TABLE_MAP["fhvhv"]).toDF()

except Exception as e:

    print("ERROR: Could not read tables. Check table names. Error: {}")  

    sys.exit(1)

# --- 4. Clean & Transform ---

# Yellow

yellow_clean = yellow_df.filter(
    (col("tpep_pickup_datetime").isNotNull()) &
    (col("tpep_dropoff_datetime").isNotNull()) &
    (col("tpep_pickup_datetime") >= lit(START_TIME)) &
    (col("tpep_pickup_datetime") < lit(END_TIME)) &
    (col("tpep_dropoff_datetime") >= col("tpep_pickup_datetime"))
).select(
    col("tpep_pickup_datetime").alias("pickup_datetime"),
    lit("yellow").alias("type")
)

# Green

green_clean = green_df.filter(
    (col("lpep_pickup_datetime").isNotNull()) &
    (col("lpep_dropoff_datetime").isNotNull()) &
    (col("lpep_pickup_datetime") >= lit(START_TIME)) &
    (col("lpep_pickup_datetime") < lit(END_TIME)) &
    (col("lpep_dropoff_datetime") >= col("lpep_pickup_datetime"))
).select(
    col("lpep_pickup_datetime").alias("pickup_datetime"),
    lit("green").alias("type")
)

```

```

)

# FHV

fhv_clean = fhv_df.filter(
    (col("pickup_datetime").isNotNull() &
     (col("dropoff_datetime").isNotNull() &
      (col("pickup_datetime") >= lit(START_TIME)) &
      (col("pickup_datetime") < lit(END_TIME)) &
      (col("dropoff_datetime") >= col("pickup_datetime"))))
).select(
    col("pickup_datetime").alias("pickup_datetime"),
    lit("fhv").alias("type")
)

# FHVHV

fhvhv_clean = fhvhv_df.filter(
    (col("pickup_datetime").isNotNull() &
     (col("dropoff_datetime").isNotNull() &
      (col("pickup_datetime") >= lit(START_TIME)) &
      (col("pickup_datetime") < lit(END_TIME)) &
      (col("dropoff_datetime") >= col("pickup_datetime"))))
).select(
    col("pickup_datetime").alias("pickup_datetime"),
    lit("fhvhv").alias("type")
)

# --- 5. Union ---

all_rides_df = yellow_clean.union(green_clean).union(fhv_clean).union(fvhv_clean)

# --- 6. Write (Standard Parquet) ---

all_rides_df.coalesce(1).write.mode("overwrite").parquet(OUTPUT_PATH)

print(f"Successfully wrote initial data to {OUTPUT_PATH}")

```

```

job.commit()

# --- 7. Rename Output File ---

print("Starting file rename process...")

s3 = boto3.resource('s3')

bucket = s3.Bucket(BUCKET_NAME)

found_file = False

# วนหาไฟล์ part-xxx.parquet ในโฟลเดอร์
for obj in bucket.objects.filter(Prefix=OUTPUT_PREFIX):
    if obj.key.endswith(".parquet") and "part-" in obj.key:
        print(f"Found file: {obj.key}")

        copy_source = {'Bucket': BUCKET_NAME, 'Key': obj.key}

        new_key = OUTPUT_PREFIX + NEW_FILENAME

        # Copy ไฟล์มาใหม่
        s3.meta.client.copy(copy_source, BUCKET_NAME, new_key)
        print(f"Renamed to: {new_key}")

        # ลบไฟล์ที่อเดิม
        obj.delete()

        found_file = True

# ลบไฟล์ _SUCCESS (cleanup)
for obj in bucket.objects.filter(Prefix=OUTPUT_PREFIX + "_SUCCESS"):
    obj.delete()

if found_file:
    print(f"Rename COMPLETED. File available at: s3://{BUCKET_NAME}/{OUTPUT_PREFIX}/{NEW_FILENAME}")
else:

```

```
print("Warning: No parquet file found to rename.")
```

Glue Crawler 2: ชื่อ nyctaxi-dev-crawl-processed

Target: s3://nyctaxi-dev-datalake-0116/processed/all\_rides/

IAM Role: ชื่อ nyctaxi-dev-role-glue

Prefix: processed-

Athena

```
SELECT
    type,
    COUNT(*) AS rides
FROM
    "nyctaxi_dev_db_catalog"."processed-all_rides"
GROUP BY
    type
ORDER BY
    rides DESC;
```

## sprint 2

Tags ที่ควรใส่ในทุก Resource:

Key: Project | Value: NYC-Taxi-Analytics-Practice-DataEngineer

Key: Environment | Value: Dev

Key: Owner | Value: Kamonphan

IAM Role: ชื่อ nyctaxi-dev-role-airflow

Attach Policies: AmazonS3FullAccess, AWSLambda\_FullAccess, AWSGlueConsoleFullAccess, CloudWatchLogsReadOnlyAccess

1.

Security group name: nyctaxi-dev-airflow-sg

Description: Allow SSH access to Airflow

VPC: Default

Inbound rules:

- Type: SSH Protocol: TCP Port range: 22 Source type: MyIP
- Type: Custom TCP Protocol: TCP Port range: 8080 Source type: Anywhere-IPv4

Outbound rules:

- Default

Tags - optional:

- Key: Name

- Value - optional: nyctaxi-dev-airflow-sg

EC2

Name: nyctaxi-dev-airflow-server

OS: Ubuntu (22.04 or 24.04 LTS)

Instance type: t3.medium

Key pair: สร้างใหม่ nyctaxi-dev-airflow-server-keypair

Security group: nyctaxi-dev-airflow-sg

Configure storage: 20GB

IAM instance profile: nyctaxi-dev-role-airflow

SSH เข้าไปผ่านปุ่ม Connect สัมๆ

รันคำสั่งตามนี้ทีละบล็อก:

```
sudo apt-get update
```

```
sudo apt-get install docker.io docker-compose -y
```

```
sudo usermod -aG docker $USER
```

```
newgrp docker
```

---

```
sudo fallocate -l 4G /swapfile
```

```
sudo chmod 600 /swapfile
```

```
sudo mkswap /swapfile
```

```
sudo swapon /swapfile
```

```
echo '/swapfile none swap sw 0 0' | sudo tee -a /etc/fstab
```

---

```
free -h
```

---

```
# 1. สร้างโฟลเดอร์
```

```
mkdir airflow-project
```

```
cd airflow-project
```

```
# 2. โหลดไฟล์ Config
```

```
curl -Lfo 'https://airflow.apache.org/docs/apache-airflow/2.10.2/docker-compose.yaml'
```

```
mkdir -p ./dags ./logs ./plugins ./config
```

```
echo -e "AIRFLOW_UID=$(id -u)" > .env
```

```
# 3. ปิดการโหลดตัวอย่าง 70 อัน (เพื่อความเบาเครื่อง)
```

```
vim docker-compose.yaml
```

หา

```
AIRFLOW__CORE__LOAD_EXAMPLES: 'true'
```

แก้เป็น

```
AIRFLOW__CORE__LOAD_EXAMPLES: 'false'
```

```
# 4. เริ่มระบบ (Init)
```

```
docker-compose up airflow-init
```

---

```
# 2. ติดตั้ง Library AWS ให้มันรู้จัก (รอสัก 1 นาทีให้ container ขึ้นก่อนค่อยรัน)
```

```
vim .env
```

เพิ่มบรรทัด

```
_PIP_ADDITIONAL_REQUIREMENTS=apache-airflow-providers-amazon
```

```
# 1. 启动 Airflow
```

```
docker-compose up -d
```

```
docker-compose ps
```

---

เข้าเว็บ <http://<Public-IP>:8080>

User: airflow

Pass: airflow

ไปที่ไฟล์ dags: cd ~/airflow-project/dags

สร้างไฟล์ nyctaxi\_pipeline.py (วางโค้ดได้เลย)

```
from airflow import DAG
from airflow.providers.amazon.aws.operators.lambda_function import LambdaInvokeFunctionOperator
from airflow.providers.amazon.aws.operators.glue import GlueJobOperator
from airflow.providers.amazon.aws.operators.glue_crawler import GlueCrawlerOperator
from airflow.operators.bash import BashOperator
from airflow.utils.dates import days_ago
import json

# -- CONFIG --
LAMBDA_FUNCTION_NAME = 'nyctaxi-dev-func-ingest'
GLUE_RAW_CRAWLER = 'nyctaxi-dev-crawl-raw'
GLUE_JOB_NAME = 'nyctaxi-dev-job-transform'
GLUE_PROCESSED_CRAWLER = 'nyctaxi-dev-crawl-processed'
AWS_REGION = 'ap-southeast-1'

default_args = {
    'owner': 'airflow',
    'start_date': days_ago(1),
    'depends_on_past': False,
}

with DAG(
    dag_id='nyctaxi_pipeline_master',
    default_args=default_args,
```

```

schedule_interval=None,
catchup=False,
tags=['production', 'nyc_tax'],
) as dag:

    wait_for_ingest = BashOperator(
        task_id='wait_for_data',
        bash_command='sleep 200',
    )

    # 1. Trigger Lambda

    ingest_task = LambdaInvokeFunctionOperator(
        task_id='1_ingest_lambda',
        function_name=LAMBDA_FUNCTION_NAME,
        invocation_type='Event',
        payload=json.dumps({'trigger': 'airflow'}),
        region_name=AWS_REGION,
    )

    # 2. Crawler Raw

    crawl_raw_task = GlueCrawlerOperator(
        task_id='2_crawl_raw',
        config=[{'Name': GLUE_RAW_CRAWLER},
        wait_for_completion=True,
        region_name=AWS_REGION,
    )

    # 3. Glue ETL Job

    etl_task = GlueJobOperator(
        task_id='3_run_glue_job',
        job_name=GLUE_JOB_NAME,
        wait_for_completion=True,
        verbose=True,
        region_name=AWS_REGION,
    )

    # 4. Crawler Processed

    crawl_proc_task = GlueCrawlerOperator(
        task_id='4_crawl_processed',
        config=[{'Name': GLUE_PROCESSED_CRAWLER},
        wait_for_completion=True,
    )

```

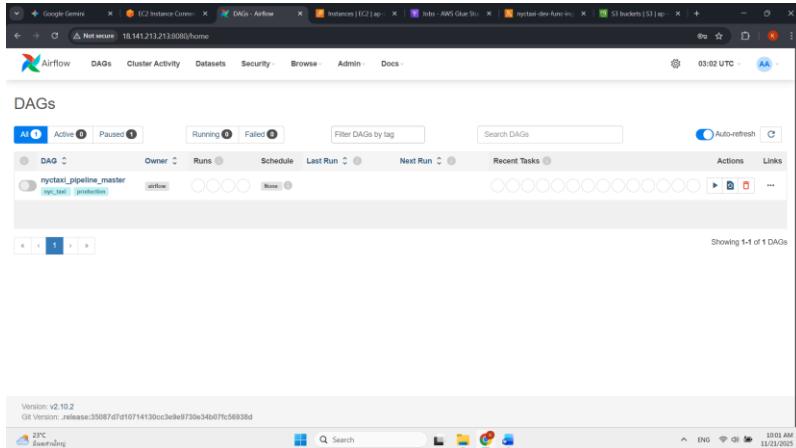
```

region_name=AWS_REGION,
)

ingest_task >> wait_for_ingest >> crawl_raw_task >> etl_task >> crawl_proc_task

```

กลับมาหน้าเว็บ Refresh



กด on ช่องสุด + กด trigger

ดูผล

จะเห็นว่า success ทั้ง 4 กล่อง -> จะได้ข้อมูลที่พร้อม query แล้ว

sprint 3

add web page for dashboard

แก้

IAM Role: ชื่อ nytaxi-dev-role-airflow

เพิ่ม policy

AmazonAthenaFullAccess

Security group name: nyctaxi-dev-airflow-sg

เพิ่ม Inbound rules:

- Type: Custom TCP Protocol: TCP Port range: 8501 Source type: Anywhere-IPv4

connect เข้า ec2

```
sudo apt install python3-pip -y
```

```
sudo apt install python3-venv -y
```

```
python3 -m venv venv
```

```
source venv/bin/activate
```

```
# ติดตั้ง Library Python
```

```
pip install streamlit altair pandas boto3
```

```
cd ~/airflow-project
```

```
vim dashboard.py
```

```
import streamlit as st
import boto3
import pandas as pd
import time

# --- CONFIGURATION ---
# ตั้งค่าไฟร์ฟรอนต์ AWS Account ของคุณ
DATABASE_NAME = 'nyctaxi_dev_db_catalog'
TABLE_NAME = 'processed-all_rides'
```

```

S3_OUTPUT_BUCKET = 's3://nyctaxi-dev-datalake-0116/athena/'

AWS_REGION = 'ap-southeast-1'

# --- PAGE SETUP ---

st.set_page_config(page_title="NYC Taxi Analytics", layout="wide")

st.title("📊 NYC Taxi Data Platform")

st.markdown("Dashboard สรุปข้อมูลการเดินทาง Taxi ใน New York (Data Pipeline by Airflow)")

# --- SIDEBAR (เมนูริบบิ้ง) ---

with st.sidebar:

    st.header("🗓️ ตั้งค่าการค้นหา")

    selected_year = st.selectbox("ปี", ["2024"])

    selected_month = st.selectbox("เดือน", ["January", "February", "March"])

    st.markdown("---")

    st.caption("สถานะระบบ: 🟢 Online")

# --- FUNCTIONS ---

def query_athena(query):

    """ส่งคำสั่ง SQL ไปที่ Athena"""

    client = boto3.client('athena', region_name=AWS_REGION)

    # 1. ส่งคำสั่ง Query

    response = client.start_query_execution(
        QueryString=query,
        QueryExecutionContext={'Database': DATABASE_NAME},
        ResultConfiguration={'OutputLocation': S3_OUTPUT_BUCKET}
    )

    query_execution_id = response['QueryExecutionId']

    # 2. รอผลลัพธ์ (Polling)

    with st.spinner("กำลังรับผลลัพธ์จาก Data Lake..."):

        while True:

            stats = client.get_query_execution(QueryExecutionId=query_execution_id)

            status = stats['QueryExecution']['Status']['State']

            if status in ['SUCCEEDED', 'FAILED', 'CANCELLED']:

                break

            time.sleep(1)

```

```

if status == 'SUCCEEDED':

    results = client.get_query_results(QueryExecutionId=query_execution_id)

    # แปลงผลลัพธ์เป็น Pandas DataFrame (แบบเบื้องต้น)

    # หากพบว่า ชื่อคอลัมน์ไม่ถูกต้องในรูปแบบที่ s3 select หรือ awswrangler
    # ให้เปลี่ยนชื่อคอลัมน์ตามที่ต้องการ

    columns = [col['Label'] for col in results['ResultSet'][ResultMetadata][ColumnInfo]]

    rows = []

    for row in results['ResultSet']['Rows'][1:]:

        rows.append({field.get('VarCharValue', None) for field in row['Data']})

    df = pd.DataFrame(rows, columns=columns)

    return df

else:

    # ตรวจสอบความถูกต้องของอนุญาติ

    error_reason = stats['QueryExecution']['Status'].get('StateChangeReason', 'Unknown Error')

    st.error(f"Query Failed: {status}")

    st.error(f"Reason: {error_reason}") # <-- บรรทัดนี้จะบอกให้เรา

    return None

# --- MAIN CONTENT ---

# สร้าง Tab เพื่อแยกส่วนดูข้อมูล กับ ส่วนสร้างงาน

tab1, tab2 = st.tabs(["Analytics Dashboard", "Control Plane"])

with tab1:

    st.subheader("กรุณาเลือกประจำเดือน: {selected_month} {selected_year}")

    if st.button("⟳ �新ข้อมูล", type="primary"):

        # SQL Query (ใช้ข้อมูลสรุป)

        sql = f"""

SELECT type, count(*) as rides

FROM [{DATABASE_NAME}].[{TABLE_NAME}]

GROUP BY type

ORDER BY rides DESC

"""

        df = query_athena(sql)

        if df is not None and not df.empty():

            # แปลงข้อมูลเป็น Pandas DataFrame
```

```

df['rides'] = df['rides'].astype(int)

# แมตช์ Metric
total_rides = df['rides'].sum()

col1, col2, col3 = st.columns(3)

col1.metric("Total Rides", f'{total_rides:,}')

col2.metric("Top Type", df.loc[0]['type'])

col3.metric("Data Source", "Amazon S3 (Parquet)")

# แมตต์กราฟเมล็ดชาติ
c1, c2 = st.columns([2, 1])

with c1:
    st.caption("จำนวนผู้ใช้บริการต่อเดือน")

    st.bar_chart(df.set_index('type'), color="#FF4B4B")

with c2:
    st.caption("ตารางผู้ใช้บริการ")

    st.dataframe(df, hide_index=True)

else:
    st.warning("ไม่พบข้อมูล หรือยังไม่ได้รัน Pipeline")

```

```

with tab2:
    st.subheader("สื่อสาร Data Pipeline (Airflow)")

    st.info("ปัจจุบันยังไม่มี DAG ที่สามารถเรียก Airflow API เพื่อเรียก Trigger DAG สำหรับเดือนใหม่ (Coming Soon)")

    if st.button("▶ Start Pipeline for selected month"):
        st.toast("ดำเนินการเรียก Airflow เรียบร้อย (Simulation)")

streamlit run dashboard.py

```

The terminal output shows the command being run and the Streamlit application running on port 8501. The browser screenshot shows the Streamlit dashboard with tabs for 'Overview', 'Data Pipeline', and 'Machine Learning'. The 'Data Pipeline' tab is active, displaying a bar chart of ride types and a table of users.

ເຊົ້າເວັບ <http://<Public-IP>:8501>

ລອງເລີນດູຈະເຫັນ

