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 มา **ทีละเดือน** 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 ##########################################

# ดึง "ปี" ออกมาจาก 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}"
```

```
        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/fhvhv/fhvhv_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-fhvhv_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["fhv"]).toDF()

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

except Exception as e:
```

```
    print(f"ERROR: Could not read tables. Check table names. Error: {e}")
    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(fhvhv_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 ไปชื่อใหม่
        s3.meta.client.copy(copy_source, BUCKET_NAME, new_key)
        print(f"Renamed to: {new_key}")

        # ลบไฟล์ชื่อเดิม
        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;
```