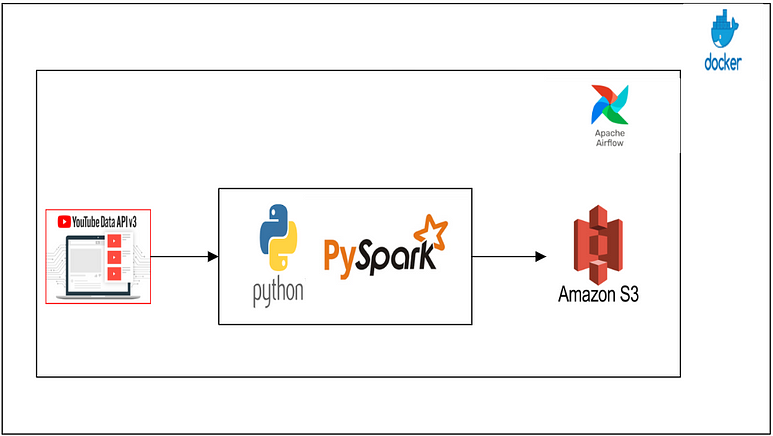
# ETL with Airflow, Spark, S3 and Docker

Now we will begin the actual project ! Get Ready for it Folks !!



Project Architecture

I’ve structured this article into 4 key steps:

1. Software installations and Setup
2. Extracting the data from Youtube Data API
3. Transforming the data using PySpark
4. Loading the data into AWS S3

***1. Software installations and Setup:***

* **VS Code —**Download and install [VS Code](https://code.visualstudio.com/).
* **Docker Desktop —**Download and install Docker Desktop.
* **(optional) Windows Subsystem for Linux (WSL) —**Many tools and libraries used in data engineering, such as Apache Airflow and PySpark, are originally developed for Unix-like systems. Running these tools in a native Linux environment (through WSL) can help avoid compatibility issues that might arise when using them on Windows.
* -> Open PowerShell as Administrator.
* -> Run the command: wsl --install.
* -> Follow the prompts to install WSL and choose a Linux distribution (e.g., Ubuntu) from the Microsoft Store.
* -> Set up your Linux distribution with a username and password.

We do not strictly need WSL to run this project. Docker Desktop can run natively on Windows, and it uses a lightweight Linux virtual machine (VM) managed by Docker itself. However, using WSL with Docker Desktop offers several benefits, because it allows us to run Linux commands and workflows directly on Windows, providing a more native development experience.

Now let’s begin the Setup:

**Part 1 — Create a Docker Image**

* Create a new folder for your project and name it as “Airflow-Project”
* Open the Command Prompt in this folder.
* In the Command Prompt, run the below command:

>> code .

* This will open the folder as a project in VS Code.
* In VS Code create a new file say “dockerfile” and paste the below code:

FROM apache/airflow:latest  
  
# Switch to root user to install system dependencies  
USER root  
  
# Install git, OpenJDK, and clean up apt cache  
RUN apt-get update && \  
 apt-get -y install git default-jdk && \  
 apt-get clean && \  
 rm -rf /var/lib/apt/lists/\*  
  
# Switch to airflow user to install Python packages  
USER airflow  
  
# Install necessary Python packages  
RUN pip install --no-cache-dir pyspark pandas google-api-python-client emoji boto3

This Docker file includes all the necessary packages to run the project.

* Right-click on the file and select the “Build Image” option in VS Code. When prompted for a name, enter “airflow-project”. This command will create a Docker image. However, the image won’t be utilized until you create a docker-compose.yml file and configure it to use the image.

(Fun Fact: Wondering why there’s no Python installation in the file? Turns out, the base image apache/airflow:latest used in the Dockerfile already has Python chilling inside because Airflow itself is written in Python and primarily uses Python for defining workflows and tasks. So, no need to fuss about installing Python separately in your Dockerfile!)

**Part 2 — Create a docker compose file**

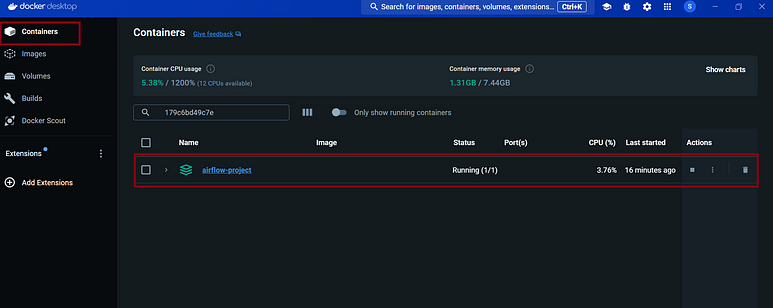
Using a docker-compose.yml file is beneficial for handling multi-container Docker applications. It enables us to define and run several Docker containers with a single command and allows us to configure environment variables, volumes, ports, and other settings for each service in a clear and organized way. With Docker Compose, you can start, stop, and manage multiple services effortlessly using a single command (docker-compose up or docker-compose down).

* Let’s create a file and name it as “docker-compose.yml” file
* Paste the below code into the file:

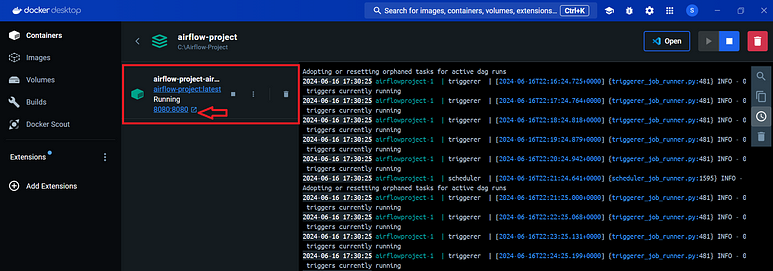
version: '3'  
services:  
  
 airflowproject:  
 image: airflow-project:latest  
 environment:  
 - AWS\_ACCESS\_KEY\_ID=your-aws-access-key  
 - AWS\_SECRET\_ACCESS\_KEY=your-aws-secret-access-key  
 - YOUTUBE\_API\_KEY=your-youtube-api-key  
 volumes:  
 - ./airflow:/opt/airflow  
 ports:  
 - "8080:8080"  
 command: airflow standalone

* Now, right-click on the file and select the ‘Compose Up’ option in VS Code. Click on it to set up the environment.
* SURPRISE SURPRISE !! After doing this, you may notice a new folder named “airflow” appearing in your project directory in VS Code.

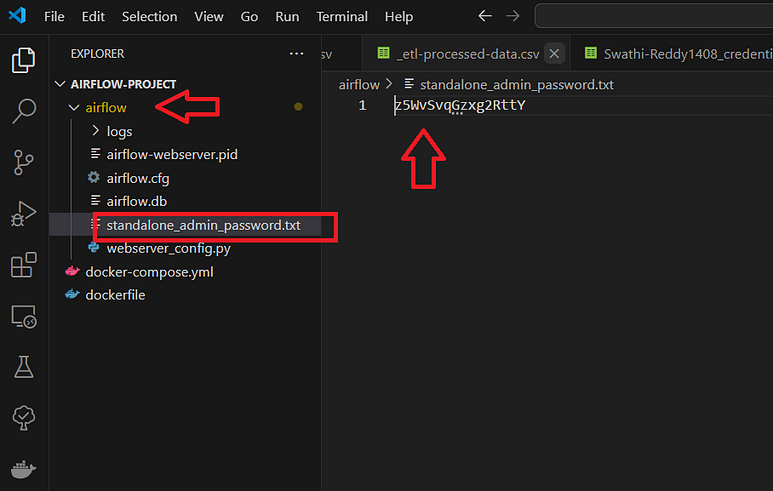
Open the docker desktop and if everything is done correctly , you will see something like this.



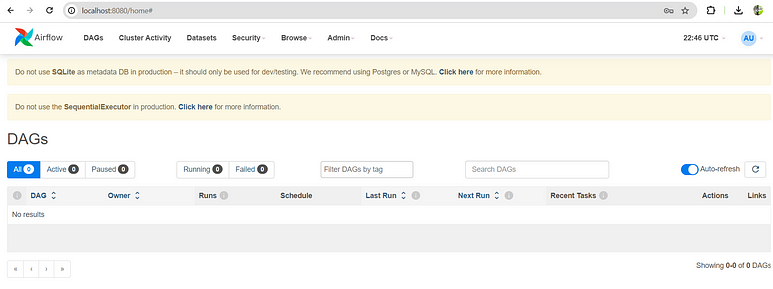
* Now, click on the Airflow project, which will open a screen displaying logs and indicating that Airflow is running on port 8080.



* Click on the port, and it will take you to the Airflow sign-in page. If you are opening this link for the first time, you will need to provide credentials.
* The username is “admin,” and the password can be found in the “standalone\_admin\_password.txt” file within the Airflow folder that was created after running the compose up command.



* After entering your credentials on the sign-in page, you will find Airflow running on your local host. It will appear as follows:



This indicates our docker image of Airflow with all the dependencies is running in the docker container.

Your environment setup is complete! Whew !!

**2. Extracting the data from Youtube Data API:**

* Create a folder named “dags” under airflow folder and create a python file as youtube\_etl\_dag.py under dags folder
* Now import the following in youtube\_etl\_dag.py

import logging  
import os  
import re  
import shutil  
from datetime import datetime, timedelta  
  
import boto3  
import emoji  
import pandas as pd  
from googleapiclient.discovery import build  
from pyspark.sql import SparkSession  
from pyspark.sql.functions import col, to\_date, udf  
from pyspark.sql.types import (DateType, IntegerType, LongType, StringType,  
 StructField, StructType)  
  
from airflow import DAG  
from airflow.operators.python\_operator import PythonOperator

* All the above libraries are required in implementing this project(they will all become useful once we start building the code)
* You will see errors in vs code because all dependencies are installed on docker but not on local machine so don’t focus on them.
* If there are any syntax errors Airflow shows it on the top of screen and for logical errors/exceptions we can see them in airflow logs

# Define the DAG and its default arguments  
default\_args = {  
 'owner': 'airflow', # Owner of the DAG  
 'depends\_on\_past': False, # Whether to depend on past DAG runs  
 'email\_on\_failure': False, # Disable email notifications on failure  
 'email\_on\_retry': False, # Disable email notifications on retry  
 'retries': 1, # Number of retries  
 'retry\_delay': timedelta(minutes=5), # Delay between retries  
 'start\_date': datetime(2023, 6, 10, 0, 0, 0), # Runs everyday at midnight (00:00) UTC  
}  
  
dag = DAG(  
 'youtube\_etl\_dag', # DAG identifier  
 default\_args=default\_args, # Assign default arguments  
 description='A simple ETL DAG', # Description of the DAG  
 schedule\_interval=timedelta(days=1), # Schedule interval: daily  
 catchup=False, # Do not catch up on missed DAG runs  
)

We are defining a DAG named ‘youtube\_etl\_dag’ that runs daily at midnight (12 AM). This DAG will be managed and triggered by Airflow, so there’s no need to run anything in VS Code. Simply update the Python file, and Airflow will automatically detect and incorporate the changes.

Currently, the DAG appears in Airflow, but it doesn’t show any tasks since none have been defined yet. To make the DAG functional, let’s create a task for extracting the data.

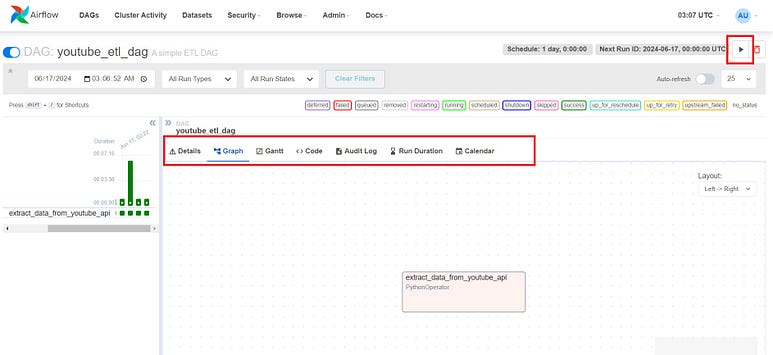
# Python callable function to extract data from YouTube API  
def extract\_data(\*\*kwargs):  
 api\_key = kwargs['api\_key']  
 region\_codes = kwargs['region\_codes']  
 category\_ids = kwargs['category\_ids']  
   
 df\_trending\_videos = fetch\_data(api\_key, region\_codes, category\_ids)  
 current\_date = datetime.now().strftime("%Y%m%d")  
 output\_path = f'/opt/airflow/Youtube\_Trending\_Data\_Raw\_{current\_date}'  
 # Save DataFrame to CSV file  
 df\_trending\_videos.to\_csv(output\_path, index=False)  
  
def fetch\_data(api\_key, region\_codes, category\_ids):  
 """  
 Fetches trending video data for multiple countries and categories from YouTube API.  
 """  
 # Initialize an empty list to hold video data  
 video\_data = []  
  
 # Build YouTube API service  
 youtube = build('youtube', 'v3', developerKey=api\_key)  
  
 for region\_code in region\_codes:  
 for category\_id in category\_ids:  
 # Initialize the next\_page\_token to None for each region and category  
 next\_page\_token = None  
 while True:  
 # Make a request to the YouTube API to fetch trending videos  
 request = youtube.videos().list(  
 part='snippet,contentDetails,statistics',  
 chart='mostPopular',  
 regionCode=region\_code,  
 videoCategoryId=category\_id,  
 maxResults=50,  
 pageToken=next\_page\_token  
 )  
 response = request.execute()  
 videos = response['items']  
  
 # Process each video and collect data  
 for video in videos:  
 video\_info = {  
 'region\_code': region\_code,  
 'category\_id': category\_id,  
 'video\_id': video['id'],  
 'title': video['snippet']['title'],  
 'published\_at': video['snippet']['publishedAt'],  
 'view\_count': int(video['statistics'].get('viewCount', 0)),  
 'like\_count': int(video['statistics'].get('likeCount', 0)),  
 'comment\_count': int(video['statistics'].get('commentCount', 0)),  
 'channel\_title': video['snippet']['channelTitle']  
 }  
 video\_data.append(video\_info)  
  
 # Get the next page token, if there are more pages of results  
 next\_page\_token = response.get('nextPageToken')  
 if not next\_page\_token:  
 break  
  
 return pd.DataFrame(video\_data)  
  
# Define extract task for the DAG  
extract\_task = PythonOperator(  
 task\_id='extract\_data\_from\_youtube\_api',  
 python\_callable=extract\_data,  
 op\_kwargs={  
 'api\_key': os.getenv('YOUTUBE\_API\_KEY'),  
 'region\_codes': ['US', 'GB', 'IN', 'AU', 'NZ'],  
 'category\_ids': ['1', '2', '10', '15', '20', '22', '23']  
 },  
 dag=dag,  
)  
  
extract\_task #makes dag to execute this task

In this code, two main actions are taking place:

1. We are creating a task named extract\_task for the DAG.
2. We are defining a callable function, extract\_data, which is invoked by extract\_task. This function fetches data from the YouTube Data API and stores it in a CSV file starting with "Youtube\_Trending\_Data\_Raw" using a pandas DataFrame.

You can refer to the YouTube Data API documentation for a detailed understanding of the data available in different parts of the API. Since we are interested in trending video data, we focus on that specific part of the API. The next\_page\_token ensures that we retrieve data from all available pages.

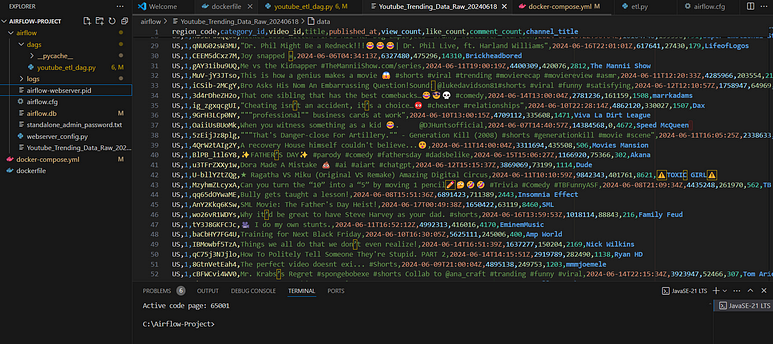
After modifying the code, your Airflow page should reflect the changes. You can manually trigger the DAG by clicking the run button in the top left corner. The status of the task (queued, running, success, etc.) is indicated by different colors in the graph. You can also view the logs once the DAG is running.



Once you click the run button, it will take some time to fetch the data and store it in the file. You will see the graph change colors at each stage of the task. How cool is that? :)

When the task status turns green, indicating success, you can check in VS Code for a new file named “Youtube-Trending-Data-Raw”

This is how our raw-data looks:



With this, our extraction task is complete, let’s move on to the next tasks!

**3. Transforming the data using PySpark:**

If you examine the raw data file, you’ll notice a lot of hashtags and emojis in the data, which are unnecessary for our project. Let’s preprocess and clean the data to make it useful for further analysis.

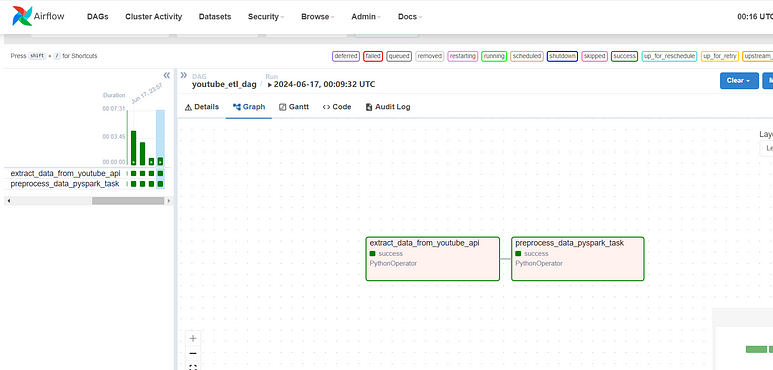
We’ll use PySpark for this task, a powerful framework designed for handling large datasets and performing transformations. Although we could use Pandas since our dataset isn’t particularly large, I prefer using PySpark. I’ve been learning PySpark recently, and I find that practical implementation is much more engaging than just studying the theory.

# Python callable function to extract data from YouTube API  
def extract\_data(\*\*kwargs):  
 api\_key = kwargs['api\_key']  
 region\_codes = kwargs['region\_codes']  
 category\_ids = kwargs['category\_ids']  
   
 df\_trending\_videos = fetch\_data(api\_key, region\_codes, category\_ids)  
 current\_date = datetime.now().strftime("%Y%m%d")  
 output\_path = f'/opt/airflow/Youtube\_Trending\_Data\_Raw\_{current\_date}'  
 # Save DataFrame to CSV file  
 df\_trending\_videos.to\_csv(output\_path, index=False)  
  
def fetch\_data(api\_key, region\_codes, category\_ids):  
 """  
 Fetches trending video data for multiple countries and categories from YouTube API.  
 Returns a pandas data frame containing video data.  
 """  
 video\_data = []  
  
 # Build YouTube API service  
 youtube = build('youtube', 'v3', developerKey=api\_key)  
  
 for region\_code in region\_codes:  
 for category\_id in category\_ids:  
 # Initialize the next\_page\_token to None for each region and category  
 next\_page\_token = None  
 while True:  
 # Make a request to the YouTube API to fetch trending videos  
 request = youtube.videos().list(  
 part='snippet,contentDetails,statistics',  
 chart='mostPopular',  
 regionCode=region\_code,  
 videoCategoryId=category\_id,  
 maxResults=50,  
 pageToken=next\_page\_token  
 )  
 response = request.execute()  
 videos = response['items']  
  
 # Process each video and collect data  
 for video in videos:  
 video\_info = {  
 'region\_code': region\_code,  
 'category\_id': category\_id,  
 'video\_id': video['id'],  
 'title': video['snippet']['title'],  
 'published\_at': video['snippet']['publishedAt'],  
 'view\_count': video['statistics'].get('viewCount', 0),  
 'like\_count': video['statistics'].get('likeCount', 0),  
 'comment\_count': video['statistics'].get('commentCount', 0),  
 'channel\_title': video['snippet']['channelTitle']  
 }  
 video\_data.append(video\_info)  
  
 # Get the next page token, if there are more pages of results  
 next\_page\_token = response.get('nextPageToken')  
 if not next\_page\_token:  
 break  
  
 return pd.DataFrame(video\_data)  
  
def preprocess\_data\_pyspark\_job():  
 spark = SparkSession.builder.appName('YouTubeTransform').getOrCreate()  
 current\_date = datetime.now().strftime("%Y%m%d")  
 output\_path = f'/opt/airflow/Youtube\_Trending\_Data\_Raw\_{current\_date}'  
 df = spark.read.csv(output\_path, header=True)  
   
 # Define UDF to remove hashtag data, emojis  
 def clean\_text(text):  
 if text is not None:  
 # Remove emojis  
 text = emoji.demojize(text, delimiters=('', ''))  
   
 # Remove hashtag data  
 if text.startswith('#'):  
 text = text.replace('#', '').strip()  
 else:  
 split\_text = text.split('#')  
 text = split\_text[0].strip()  
   
 # Remove extra double quotes and backslashes  
 text = text.replace('\\"', '') # Remove escaped quotes  
 text = re.sub(r'\"+', '', text) # Remove remaining double quotes  
 text = text.replace('\\', '') # Remove backslashes  
   
 return text.strip() # Strip any leading or trailing whitespace  
  
 return text  
 # Register UDF  
 clean\_text\_udf = udf(clean\_text, StringType())  
  
 # Clean the data  
 df\_cleaned = df.withColumn('title', clean\_text\_udf(col('title'))) \  
 .withColumn('channel\_title', clean\_text\_udf(col('channel\_title'))) \  
 .withColumn('published\_at', to\_date(col('published\_at'))) \  
 .withColumn('view\_count', col('view\_count').cast(LongType())) \  
 .withColumn('like\_count', col('like\_count').cast(LongType())) \  
 .withColumn('comment\_count', col('comment\_count').cast(LongType())) \  
 .dropna(subset=['video\_id'])  
   
 # Generate the filename based on the current date  
 current\_date = datetime.now().strftime("%Y%m%d")  
 output\_path = f'/opt/airflow/Transformed\_Youtube\_Data\_{current\_date}'  
   
 # Write cleaned DataFrame to the specified path  
 df\_cleaned.write.csv(output\_path, header=True, mode='overwrite')   
  
  
# Define extract task for the DAG  
extract\_task = PythonOperator(  
 task\_id='extract\_data\_from\_youtube\_api',  
 python\_callable=extract\_data,  
 op\_kwargs={  
 'api\_key': os.getenv('YOUTUBE\_API\_KEY'),  
 'region\_codes': ['US', 'GB', 'IN', 'AU', 'NZ'],  
 'category\_ids': ['1', '2', '10', '15', '20', '22', '23']  
 },  
 dag=dag,  
)  
  
# Define preprocessing task for the DAG  
preprocess\_data\_pyspark\_task= PythonOperator(  
 task\_id='preprocess\_data\_pyspark\_task',  
 python\_callable=preprocess\_data\_pyspark\_job,  
 dag=dag  
)  
  
extract\_task >> preprocess\_data\_pyspark\_task

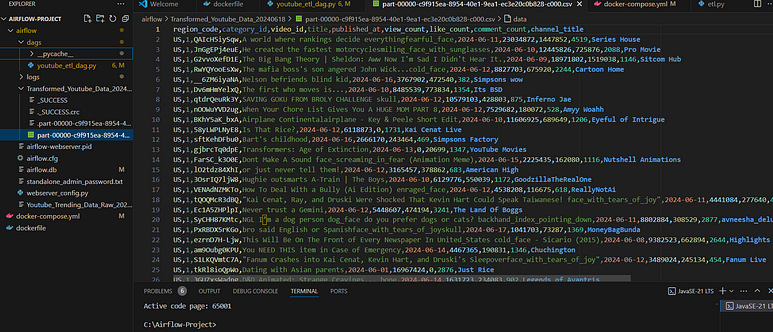
Here’s what this code does:

* It creates a task named preprocess\_data\_pyspark\_task.
* This task calls the preprocess\_data\_pyspark\_job function.
* The preprocess\_data\_pyspark\_job function cleans the data.
* The cleaned data is then stored in a folder named Transformed\_Youtube\_Data\_currentDate.
* Within this folder, a new CSV file named with the prefix “part-” will be created which has the cleaned data.

If you see Airflow there will be another task added to the first task like below:



This is how our transformed data looks:



This task is done . Now we will move on to final task.

**4. Loading the data into S3:**

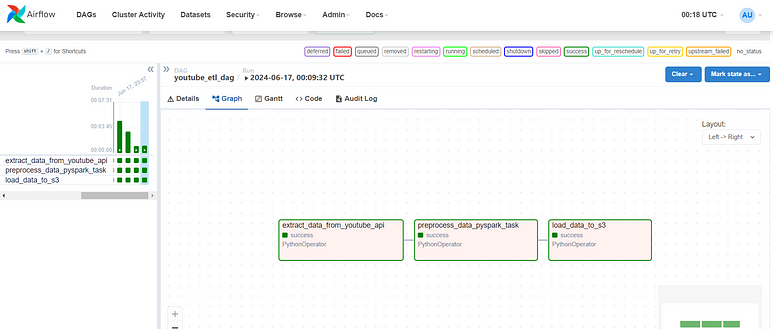
Before starting this task, create an S3 bucket using the IAM user you set up initially and note down the bucket name.

This is our final code!

import logging  
import os  
import re  
import shutil  
from datetime import datetime, timedelta  
  
import boto3  
import emoji  
import pandas as pd  
from googleapiclient.discovery import build  
from pyspark.sql import SparkSession  
from pyspark.sql.functions import col, to\_date, udf  
from pyspark.sql.types import (DateType, IntegerType, LongType, StringType,  
 StructField, StructType)  
  
from airflow import DAG  
from airflow.operators.python\_operator import PythonOperator  
  
# Define the DAG and its default arguments  
default\_args = {  
 'owner': 'airflow', # Owner of the DAG  
 'depends\_on\_past': False, # Whether to depend on past DAG runs  
 'email\_on\_failure': False, # Disable email notifications on failure  
 'email\_on\_retry': False, # Disable email notifications on retry  
 'retries': 1, # Number of retries  
 'retry\_delay': timedelta(minutes=5), # Delay between retries  
 'start\_date': datetime(2023, 6, 10, 0, 0, 0), # Runs everyday at midnight (00:00) UTC  
}  
  
dag = DAG(  
 'youtube\_etl\_dag', # DAG identifier  
 default\_args=default\_args, # Assign default arguments  
 description='A simple ETL DAG', # Description of the DAG  
 schedule\_interval=timedelta(days=1), # Schedule interval: daily  
 catchup=False, # Do not catch up on missed DAG runs  
)  
# Python callable function to extract data from YouTube API  
def extract\_data(\*\*kwargs):  
 api\_key = kwargs['api\_key']  
 region\_codes = kwargs['region\_codes']  
 category\_ids = kwargs['category\_ids']  
   
 df\_trending\_videos = fetch\_data(api\_key, region\_codes, category\_ids)  
 current\_date = datetime.now().strftime("%Y%m%d")  
 output\_path = f'/opt/airflow/Youtube\_Trending\_Data\_Raw\_{current\_date}'  
 # Save DataFrame to CSV file  
 df\_trending\_videos.to\_csv(output\_path, index=False)  
  
def fetch\_data(api\_key, region\_codes, category\_ids):  
 """  
 Fetches trending video data for multiple countries and categories from YouTube API.  
 Returns a pandas data frame containing video data.  
 """  
 # Initialize an empty list to hold video data  
 video\_data = []  
  
 # Build YouTube API service  
 youtube = build('youtube', 'v3', developerKey=api\_key)  
  
 for region\_code in region\_codes:  
 for category\_id in category\_ids:  
 # Initialize the next\_page\_token to None for each region and category  
 next\_page\_token = None  
 while True:  
 # Make a request to the YouTube API to fetch trending videos  
 request = youtube.videos().list(  
 part='snippet,contentDetails,statistics',  
 chart='mostPopular',  
 regionCode=region\_code,  
 videoCategoryId=category\_id,  
 maxResults=50,  
 pageToken=next\_page\_token  
 )  
 response = request.execute()  
 videos = response['items']  
  
 # Process each video and collect data  
 for video in videos:  
 video\_info = {  
 'region\_code': region\_code,  
 'category\_id': category\_id,  
 'video\_id': video['id'],  
 'title': video['snippet']['title'],  
 'published\_at': video['snippet']['publishedAt'],  
 'view\_count': video['statistics'].get('viewCount', 0),  
 'like\_count': video['statistics'].get('likeCount', 0),  
 'comment\_count': video['statistics'].get('commentCount', 0),  
 'channel\_title': video['snippet']['channelTitle']  
 }  
 video\_data.append(video\_info)  
  
 # Get the next page token, if there are more pages of results  
 next\_page\_token = response.get('nextPageToken')  
 if not next\_page\_token:  
 break  
  
 return pd.DataFrame(video\_data)  
  
def preprocess\_data\_pyspark\_job():  
 spark = SparkSession.builder.appName('YouTubeTransform').getOrCreate()  
 current\_date = datetime.now().strftime("%Y%m%d")  
 output\_path = f'/opt/airflow/Youtube\_Trending\_Data\_Raw\_{current\_date}'  
 df = spark.read.csv(output\_path, header=True)  
   
 # Define UDF to remove hashtag data, emojis  
 def clean\_text(text):  
 if text is not None:  
 # Remove emojis  
 text = emoji.demojize(text, delimiters=('', ''))  
   
 # Remove hashtags and everything after them  
 if text.startswith('#'):  
 text = text.replace('#', '').strip()  
 else:  
 split\_text = text.split('#')  
 text = split\_text[0].strip()  
   
 # Remove extra double quotes and backslashes  
 text = text.replace('\\"', '') # Remove escaped quotes  
 text = re.sub(r'\"+', '', text) # Remove remaining double quotes  
 text = text.replace('\\', '') # Remove backslashes  
   
 return text.strip() # Strip any leading or trailing whitespace  
  
 return text  
 # Register UDF  
 clean\_text\_udf = udf(clean\_text, StringType())  
  
 # Clean the data  
 df\_cleaned = df.withColumn('title', clean\_text\_udf(col('title'))) \  
 .withColumn('channel\_title', clean\_text\_udf(col('channel\_title'))) \  
 .withColumn('published\_at', to\_date(col('published\_at'))) \  
 .withColumn('view\_count', col('view\_count').cast(LongType())) \  
 .withColumn('like\_count', col('like\_count').cast(LongType())) \  
 .withColumn('comment\_count', col('comment\_count').cast(LongType())) \  
 .dropna(subset=['video\_id'])  
   
 # Generate the filename based on the current date  
 current\_date = datetime.now().strftime("%Y%m%d")  
 output\_path = f'/opt/airflow/Transformed\_Youtube\_Data\_{current\_date}'  
   
 # Write cleaned DataFrame to the specified path  
 df\_cleaned.write.csv(output\_path, header=True, mode='overwrite')   
  
  
def load\_data\_to\_s3(\*\*kwargs):  
 bucket\_name = kwargs['bucket\_name']  
 today = datetime.now().strftime('%Y/%m/%d')  
 prefix = f"processed-data/{today}"  
 current\_date = datetime.now().strftime("%Y%m%d")  
 local\_dir\_path = f'/opt/airflow/Transformed\_Youtube\_Data\_{current\_date}'  
 upload\_to\_s3(bucket\_name, prefix, local\_dir\_path)  
  
  
def upload\_to\_s3(bucket\_name, prefix, local\_dir\_path):  
 aws\_access\_key\_id = os.getenv('AWS\_ACCESS\_KEY\_ID')  
 aws\_secret\_access\_key = os.getenv('AWS\_SECRET\_ACCESS\_KEY')  
  
 s3\_client = boto3.client(  
 's3',  
 aws\_access\_key\_id=aws\_access\_key\_id,  
 aws\_secret\_access\_key=aws\_secret\_access\_key  
 )  
  
 for root, dirs, files in os.walk(local\_dir\_path):  
 for file in files:  
 if file.endswith('.csv'):  
 file\_path = os.path.join(root, file)  
 s3\_key = f"{prefix}/{file}"  
 logging.info(f"Uploading {file\_path} to s3://{bucket\_name}/{s3\_key}")  
 s3\_client.upload\_file(file\_path, bucket\_name, s3\_key)  
  
# Define extract task for the DAG  
extract\_task = PythonOperator(  
 task\_id='extract\_data\_from\_youtube\_api',  
 python\_callable=extract\_data,  
 op\_kwargs={  
 'api\_key': os.getenv('YOUTUBE\_API\_KEY'),  
 'region\_codes': ['US', 'GB', 'IN', 'AU', 'NZ'],  
 'category\_ids': ['1', '2', '10', '15', '20', '22', '23']  
 },  
 dag=dag,  
)  
  
# Define preprocessing task for the DAG  
preprocess\_data\_pyspark\_task= PythonOperator(  
 task\_id='preprocess\_data\_pyspark\_task',  
 python\_callable=preprocess\_data\_pyspark\_job,  
 dag=dag  
)  
  
# Define Load Task for DAG  
load\_data\_to\_s3\_task = PythonOperator(  
 task\_id='load\_data\_to\_s3',  
 python\_callable=load\_data\_to\_s3,  
 op\_kwargs={  
 'bucket\_name': 'Please paste your Bucket name here'  
 },  
 dag=dag  
)  
  
extract\_task >> preprocess\_data\_pyspark\_task >> load\_data\_to\_s3\_task

We created our final task, named load\_data\_to\_s3\_task, which invokes the load\_data\_to\_s3 function to upload our file to the S3 bucket. You can verify the upload by checking the contents of the S3 bucket.

Finally our Airflow looks like this!



Now, take this data and connect it to Tableau or any BI tool to create exciting dashboard and visualize insights!

I hope you’ve journeyed through this pipeline with me and picked up some new skills along the way! 🚀 Congrats if you came this far successfully! 🎉 May this newfound knowledge serve you well on your future adventures in data engineering!