

# Project: Spotify Data Pipeline Using Airflow

Monday, August 21, 2023 9:30 AM

## Process to Execute the Project :-

**Step 1:** First lets login to Website : <https://developer.spotify.com/>

Note : You must have a spotify account for that.

**Step 2:** Let's see the account overview -

The screenshot shows the 'Account overview' section of the Spotify Developer website. On the left, there is a sidebar with icons for 'Account overview', 'Edit profile', 'Address', 'Change password', 'Notification settings', 'Privacy settings', 'Saved payment cards', and 'Recover playlists'. The main content area is titled 'Account overview' and contains a 'Profile' section with fields for 'Username' (31gsopdjrjx6jjqm4ttjg7325j6q), 'Email' (rahulkousik123@outlook.com), 'Date of birth' (June 12, 1997), and 'Country or region' (India). A 'Edit profile' button is located below these fields. At the bottom of the page, there are links for 'Spotify for Developers', 'Documentation', 'Community', and a user profile for 'Kousik'. A 'Create app' button is also visible.

## Dashboard

[Create app](#)



You haven't created any apps yet

[Create app](#)

Then lets create an app into that :

Dashboard &gt; Create app

## Create app

App name

App description

Website

Redirect URI

A URI where users can be redirected after authentication success or failure

## Dashboard

[Create app](#)
 Condensed view
**python\_airflow\_ETL**

This is a project which includes both the work of airflow and the spotify api.

So, we have created an app name : **python\_airflow\_ETL**

In essence, creating an application that fetches data from an API allows you to harness external data sources and services to enhance the functionality, accuracy, and real-time nature of your application. It's a fundamental concept in modern software development that enables dynamic, interconnected, and feature-rich applications.

Now we need client ID and secret :

Dashboard &gt; python\_airflow\_ETL &gt; Settings &gt; Basic Information

## Basic Information

[Basic Information](#)[User Management](#)[Extension Requests](#)

Client ID

a9a65070f7dd43408a9b1185f92c8966

App Status

Development mode

[ⓘ](#)[View client secret](#)

App name

python\_airflow\_ETL

App description

This is a project which includes both the work of airflow and the spotify api.

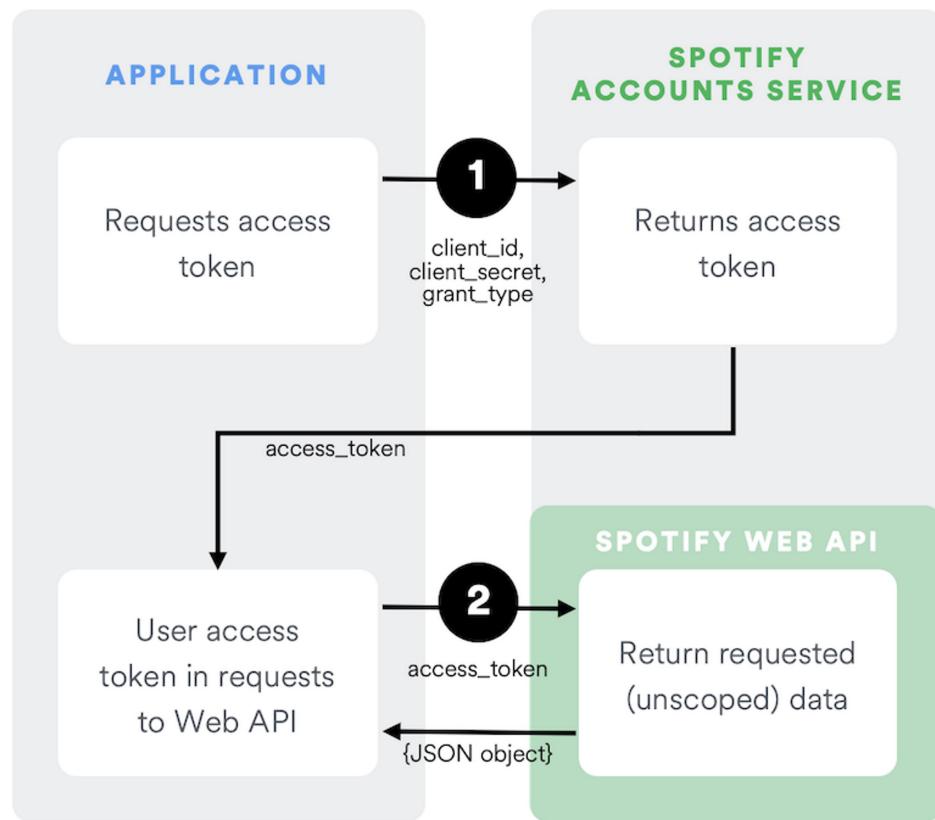
A client ID and client secret are often used as part of the OAuth 2.0 authentication framework for APIs. OAuth 2.0 is a widely used protocol for securing API access and allowing applications to interact with protected resources on behalf of users without exposing their credentials. The client ID and client secret play specific roles in this process:

- **Client ID:** The client ID is a unique identifier assigned to the application (also known as the "client") by the API provider. It helps the API provider recognize and identify the application making requests to the API. Each application that wants to access the API is given a distinct client ID. This allows the API provider to keep track of which applications are using their services.
- **Client Secret:** The client secret is a confidential string that's associated with the client ID. It's a form of authentication to verify the identity of the application making the request. The client secret should be kept secure and not shared publicly or exposed in the application code. It's used to authenticate the application with the API provider and establish trust.

In summary, the client ID and secret are used in OAuth 2.0-based API authentication to ensure secure and controlled access to protected resources while providing the API provider with the means to manage and monitor how applications are interacting with their API.

Follow this documentation : <https://developer.spotify.com/documentation/web-api/tutorials/client-credentials-flow>

Now the flow how it works is :



So how it works is, we have our access token and we will be requesting an access token from spotify, by providing the **client\_id**, **client\_secret** and few other information. Then this service will release a temporary access token, having this access token we can give it to spotify web api and we can get in return json object.

```

def get_token():
    auth_string = client_id + ":" + client_secret # this we need to get the access token
    auth_bytes = auth_string.encode("utf-8") # Next we need to encode the string with utf - 8
    auth_base64 = str(base64.b64encode(auth_bytes),"utf-8") # then we need to encode the auth_string with the base64 encoding

    url = "https://accounts.spotify.com/api/token" # we will be sending a request to this website
    headers = {
        "Authorization" : "Basic " + auth_base64,
        "Content-Type" : "application/x-www-form-urlencoded"
    }
    data = {"grant_type" : "client_credentials"}
    result = post(url, headers = headers, data = data)
    json_result = json.loads(result.content)
    token = json_result["access_token"]
    return token

```

This headers we will be using for further request from Web API.

```

def get_auth_header(token):
    return {"Authorization": "Bearer " + token}

```

This token we will be using for the further requests.

- Bearer tokens are a much simpler way of making API requests, since they don't require cryptographic signing of each request.

Now this is the website for the documentation , whatever we want with the help of the API we can find it here.

<https://developer.spotify.com/documentation/web-api/tutorials/client-credentials-flow>

The screenshot shows a web browser displaying the Spotify for Developers Documentation. The URL in the address bar is <https://developer.spotify.com/documentation/web-api/tutorials/client-credentials-flow>. The page has a purple header with the Spotify logo and the word 'Documentation'. On the left, there's a sidebar with 'How-Tos' and a 'REFERENCE' section containing links to various API endpoints like Albums, Artists, Audiobooks, etc. The main content area shows a code example for creating an authorization request:

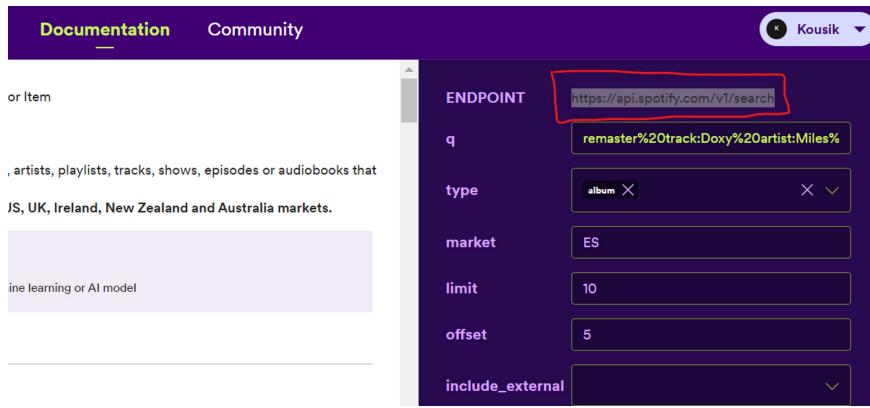
```

1 var client_id = 'CLIENT_ID';
2 var client_secret = 'CLIENT_SECRET';
3
4 var authOptions = {
5   url: 'https://accounts.spotify.com/api/token',
6   headers: {
7     'Authorization': 'Basic ' + (new Buffer.from(client_id + ':' + client_secret)).toString('base64'),
8   },
9   form: {
10     grant_type: 'client_credentials'
11   },
12   json: true
13 };
14
15 request.post(authOptions, function(error, response, body) {
16   if (!error && response.statusCode === 200) {
17     var token = body.access_token;
18   }
19 });

```

Below the code, a note says: "If everything goes well, you'll receive a response similar to this containing t".

Our endpoint to access the resources :



```

def search_for_artist(token, artist_name):
    url = "https://api.spotify.com/v1/search"
    headers = get_auth_header(token)
    query = f"?q={artist_name}&type=artist&limit=1" # whatever we search in the search bar , the first suggestion we will take
    query_url = url + query
    result = get(query_url,headers = headers)
    json_result = json.loads(result.content)[ "artists" ][ "items" ]
    if len(json_result) == 0:
        print("no artist with this name exists")
        return None
    return json_result[0]

def get_song_by_artist(token,artist_id):
    url = f"https://api.spotify.com/v1/artists/{artist_id}/top-tracks?country=IN"
    headers = get_auth_header(token)
    result = get(url, headers = headers)
    json_result = json.loads(result.content)[ 'tracks' ]
    return json_result


token = get_token()
# input
artist_name = "arijit"
result = search_for_artist(token, artist_name)
#print(result["name"])
artist_id = result['id']
songs = get_song_by_artist(token,artist_id)
ids = []
song_name = []
for idx,song in enumerate(songs):
    ids.append(idx+1)
    song_name.append(song[ 'name' ])
# Saving the data in the dataframe : top tracks of a artist

songs_data = pd.DataFrame({ 'id': ids, 'Top_Tracks' : song_name})
songs_data = songs_data.set_index('id')
songs_data.to_csv(f"result_{artist_name}.csv")

```

Put everything in a single function :

```

def spotify_artist_search_and_save():

    # Replace these with your actual client ID, client secret, and artist name
    client_id = "a9a65070f7dd43408a9b1185f92c8966"
    client_secret = "cbf316d2e45f468680e1ca3f759813ec"
    artist_name = "arjit"

    def get_token():
        auth_string = client_id + ":" + client_secret
        auth_bytes = auth_string.encode("utf-8")
        auth_base64 = str(base64.b64encode(auth_bytes), "utf-8")

        url = "https://accounts.spotify.com/api/token"
        headers = {
            "Authorization": "Basic " + auth_base64,
            "Content-Type": "application/x-www-form-urlencoded"
        }
        data = {"grant_type": "client_credentials"}
        result = post(url, headers=headers, data=data)
        json_result = json.loads(result.content)
        token = json_result["access_token"]
        return token

    def get_auth_header(token):
        return {"Authorization": "Bearer " + token}

    def search_for_artist(token, artist_name):
        url = "https://api.spotify.com/v1/search"
        headers = get_auth_header(token)
        query = f"?q={artist_name}&type=artist&limit=1"
        query_url = url + query
        result = get(query_url, headers=headers)
        json_result = json.loads(result.content)[ "artists" ][ "items" ]

        if len(json_result) == 0:
            print("No artist with this name exists")
            return None
        return json_result[0]

    def get_song_by_artist(token, artist_id):
        url = f"https://api.spotify.com/v1/artists/{artist_id}/top-tracks?country=IN"
        headers = get_auth_header(token)
        result = get(url, headers=headers)
        json_result = json.loads(result.content)[ 'tracks' ]
        return json_result

    token = get_token()
    result = search_for_artist(token, artist_name)
    artist_id = result['id']
    songs = get_song_by_artist(token, artist_id)

    ids = []
    song_name = []
    for idx, song in enumerate(songs):
        ids.append(idx + 1)
        song_name.append(song[ 'name' ])

    songs_data = pd.DataFrame({ 'id': ids, 'Top_Tracks': song_name })
    songs_data = songs_data.set_index('id')
    songs_data.to_csv(f"s3://airflow-spotify-bucket-new/result_{artist_name}.csv")

```

**Step 3 :** Next Create S3 bucket to store the data fetched from Spotify :

**Amazon S3**

Buckets, Access Points, Object Lambda Access Points, Multi-Region Access Points, Batch Operations, IAM Access Analyzer for S3.

Block Public Access settings for this account.

**Storage Lens**: Dashboards, AWS Organizations settings.

**Amazon S3**

**Account snapshot**: Storage lens provides visibility into storage usage and activity trends. [Learn more](#)

**Buckets (7) Info**: Buckets are containers for data stored in S3. [Learn more](#)

Name	AWS Region	Access	Creation date
airflow-spotify-bucket-new	Asia Pacific (Mumbai) ap-south-1	Only authorized users of this account	August 21, 2023, 00:49:26 (UTC+05:30)

#### Step 4: Create an EC2 instance for running the Airflow server :

**Instances (2) Info**

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
project_airflow	i-074a729ce9250bf29	Stopped	t2.small	-	No alarms	ap-south-1

**Configuration : 1Vcpu, 2 GB Ram : Virtual machine (Ubuntu - Linux)**

**In general terms, a CPU with "1 CPU" typically refers to a single physical processing unit, also known as a "core." So, in this context, when you have "1 CPU," it means that the CPU has a single core.**

Now, we will be running the EC2 machine ( Ubuntu ) from our terminal. From same folder where our pem key is available.

**Spotify Data Pipeline using Airflow**

- Home
- OneDrive - Perso
- Desktop
- Downloads
- Documents
- Pictures
- Music
- Videos
- Excel
- Final Solution Dc
- Project Notes De

Name	Date modified	Type	Size
.ipynb_checkpoints	8/21/2023 12:55 AM	File folder	
__pycache__	8/21/2023 12:55 AM	File folder	
ETL DAG RUN - Ubuntu	8/21/2023 3:02 AM	Text Document	1 KB
fresh_spotify_function_file_for_py	8/21/2023 10:42 AM	Jupyter Source File	8 KB
Function text For EC2 - UBUNTU	8/21/2023 3:02 AM	Text Document	3 KB
project_airflow.pem	8/20/2023 11:11 PM	PEM File	2 KB
spotify_dag	8/21/2023 1:16 AM	Jupyter Source File	5 KB
spotify_ETL	8/21/2023 10:36 AM	Jupyter Source File	10 KB
spotify_functions	8/21/2023 12:54 AM	Python Source File	3 KB
Twitter_ETL	8/20/2023 8:39 PM	Jupyter Source File	37 KB

```

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Futurense\Desktop\Spotify Data Pipeline using Airflow> ssh -i "project_airflow.pem" ubuntu@ec2-13-232-166-171.ap-south-1.compute.amazonaws.com
The authenticity of host 'ec2-13-232-166-171.ap-south-1.compute.amazonaws.com (13.232.166.171)' can't be established.
ED25519 key fingerprint is SHA256:Dw5k1PfMscobtx29pfwePWr7HxutnVsBzHBQjscjyG0.
This host key is known by the following other names/addresses:
  C:\Users\Futurense/.ssh/known_hosts:17: ec2-43-205-236-170.ap-south-1.compute.amazonaws.com
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-13-232-166-171.ap-south-1.compute.amazonaws.com' (ED25519) to the list of known hosts.
Welcome to Ubuntu 22.04.2 LTS (GNU/Linux 5.19.0-1025-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Mon Aug 21 05:22:03 UTC 2023

System load: 0.10205078125 Processes: 98
Usage of /: 36.9% of 7.57GB Users logged in: 0
Memory usage: 12% IPv4 address for eth0: 172.31.45.95
Swap usage: 0%

* Ubuntu Pro delivers the most comprehensive open source security and
compliance features.

```

Then complete the following installation :

```

Sudo apt-get update
Sudo apt install python3-pip
Sudo pip install apache airflow
Sudo pip install pandas
Sudo pip install s3fs

```

Then write **airflow** : check if its running or not.

Then type **airflow standalone** : check the server is running or not

```

Last login: Sun Aug 20 19:53:34 2023 from 122.171.16.72
ubuntu@ip-172-31-45-95:~$ ls
airflow snap
ubuntu@ip-172-31-45-95:~$ sudo snap install aws-cli

```

We have to install AWS CLI as well, because without that we will not be able to configure or send the data from API through DAG to the S3 file system.

We have to configure AWS as Well :

```

ubuntu@ip-172-31-45-95:~$ aws configure
AWS Access Key ID [*****HQWA]: ^C
ubuntu@ip-172-31-45-95:~$ 

```

For that we have to create a User and then we have to assign role **{Administrator access}** --> then we can generate an access key and get the access key for getting the s3 file system access.

So, now the thing is : we can get the admin and password

```

n_after=2021-06-10T00:00:00+00:00
scheduler | [2023-08-21T05:37:40.570+0000] {dag.py:3677} INFO - Setting next_dagrun for spotify_dag to 2021-06-10T00:00:00+00:00, run_after_start囿
n_after=2021-06-11T00:00:00+00:00
scheduler | [2023-08-21T05:37:40.607+0000] {scheduler_job_runner.py:414} INFO - 1 tasks up for execution:
scheduler | <TaskInstance: spotify_dag.complete_spotify_elt scheduled_2021-06-09T00:00:00+00:00> [scheduled]>
scheduler | [2023-08-21T05:37:40.607+0000] {scheduler_job_runner.py:479} INFO - DAG spotify_dag has 0/16 running and queued tasks
scheduler | [2023-08-21T05:37:40.607+0000] {scheduler_job_runner.py:595} INFO - Setting the following tasks to queued state:
scheduler | <TaskInstance: spotify_dag.complete_spotify_elt scheduled_2021-06-09T00:00:00+00:00> [scheduled]>
scheduler | [2023-08-21T05:37:40.609+0000] {taskinstance.py:1441} WARNING - cannot record scheduled_duration for task complete_spotify_elt because previous state change time has not been saved
scheduler | [2023-08-21T05:37:40.609+0000] {scheduler_job_runner.py:638} INFO - Sending TaskInstanceKey(dag_id='spotify_dag', task_id='complete_spotify_elt', run_id='scheduled_2021-06-09T00:00:00+00:00', try_number=1, map_index=-1) to executor with priority 1 and queue default
scheduler | [2023-08-21T05:37:40.609+0000] {base_executor.py:144} INFO - Adding to queue: ['airflow', 'tasks', 'run', 'spotify_dag', 'complete_spotify_elt', 'scheduled_2021-06-09T00:00:00+00:00', '--local', '--subdir', 'DAGS_FOLDER/spotify_dag.py']
scheduler | [2023-08-21T05:37:40.615+0000] {sequential_executor.py:74} INFO - Executing command: ['airflow', 'tasks', 'run', 'spotify_dag', 'complete_spotify_elt', 'scheduled_2021-06-09T00:00:00+00:00', '--local', '--subdir', 'DAGS_FOLDER/spotify_dag.py']
standalone |
standalone | Airflow is ready
standalone | Login with username: admin password: fHySf4y8uAmUMyHQ
standalone | Airflow Standalone is for development purposes only. Do not use this in production!
standalone |
scheduler | [2023-08-21T05:37:41.563+0000] {dagbag.py:539} INFO - Filling up the DagBag from /home/ubuntu/airflow/spotify_dag/spotify_dag.py
scheduler | [2023-08-21T05:37:42.424+0000] {example_python_operator.py:89} WARNING - The virtualenv_python example task requires virtualenv, please install it.
scheduler | [2023-08-21T05:37:42.451+0000] {tutorial_taskflow_api_virtualenv.py:29} WARNING - The tutorial_taskflow_api_virtualenv example DAG requires virtualenv, please install it.
scheduler | [2023-08-21T05:37:42.502+0000] {example_kubernetes_executor.py:38} WARNING - The example_kubernetes_executor example DAG requires the kubernetes provider. Please install it with: pip install apache-airflow[cncf.kubernetes]
scheduler | [2023-08-21T05:37:42.585+0000] {example_local_kubernetes_executor.py:39} WARNING - Could not import DAGs in example_local_kubernetes_executor.py

```

And then we can start the airflow :

Instance summary for i-074a729ce9250bf29 (project_airflow)		
Updated less than a minute ago	<a href="#">Info</a>	<a href="#">C</a> <a href="#">Connect</a> <a href="#">Actions</a>
Instance ID i-074a729ce9250bf29 (project_airflow)	Public IPv4 address 13.232.166.171   <a href="#">open address</a>	Private IPv4 addresses 172.31.45.95
IPv6 address -	Instance state <span style="color: green;">Running</span>	Public IPv4 DNS ec2-13-232-166-171.ap-south-1.compute.amazonaws.com   <a href="#">open address</a>
Hostname type IP name: ip-172-31-45-95.ap-south-1.compute.internal	Private IP DNS name (IPv4 only) ip-172-31-45-95.ap-south-1.compute.internal	Elastic IP addresses -
Answer private resource DNS name IPv4 (A)	Instance type t2.small	AWS Compute Optimizer finding <a href="#">Opt-in to AWS Compute Optimizer for recommend</a>
Auto-assigned IP address 13.232.166.171 [Public IP]	VPC ID vpc-002e7da7f5e4db0f8	

We can start the airflow website with that : it runs on the 8080 port

The Airflow website interface shows the DAG: spotify\_dag page. The URL in the browser is [http://ec2-13-232-166-171.ap-south-1.compute.amazonaws.com:8080/dags/spotify\\_dag/grid](http://ec2-13-232-166-171.ap-south-1.compute.amazonaws.com:8080/dags/spotify_dag/grid). The page includes navigation tabs for Grid, Graph, Calendar, Task Duration, Task Tries, Landing Times, Gantt, Details, and Code. It also features an Audit Log section and a filter bar at the bottom.

Remember if this website does not run then we can : do the **edit inbound rules**.

With Edit inbound rules we can get the website.

**Step 5:** In next step we can do is , we have to convert the function file to the .py module. Because from this source code we will be importing the data to other jupyter notebook.

spotify\_dag Last Checkpoint: 11 hours ago (autosaved)

The screenshot shows a Jupyter Notebook interface with a toolbar at the top and a code cell containing Python code. The code defines a DAG named 'spotify\_dag' with default arguments and a single task operator 'run\_etl'.

```
from datetime import timedelta
from airflow import DAG
from airflow.operators.python import PythonOperator
from airflow.utils.dates import days_ago
from datetime import datetime
from spotify_functions import spotify_artist_search_and_save

from spotify_functions import spotify_artist_search_and_save
```

Let's run the spotify DAG :

```
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('spotify_dag',
          default_args=default_args,
          description='My first etl code')

run_etl = PythonOperator(
    task_id='complete_spotify_etl',
    python_callable=spotify_artist_search_and_save,
    dag = dag)

run_etl
```

```

Last login: Mon Aug 21 05:22:03 2023 from 49.204.70.102
ubuntu@ip-172-31-45-95:~$ sudo nano spotify_etl.py
ubuntu@ip-172-31-45-95:~$ cd airflow/
ubuntu@ip-172-31-45-95:~/airflow$ ls
airflow-webserver.pid airflow.cfg airflow.db logs spotify_dag standalone_admin_password.txt webserver_config.py
ubuntu@ip-172-31-45-95:~/airflow$ sudo nano airflow.cfg
ubuntu@ip-172-31-45-95:~/airflow$ cd spotify_dag/
ubuntu@ip-172-31-45-95:~/airflow/spotify_dag$ ls
__pycache__ spotify_dag.py spotify_etl.py
ubuntu@ip-172-31-45-95:~/airflow/spotify_dag$
```

Inside airflow cfg :

```

[core]
# The folder where your airflow pipelines live, most likely a
# subfolder in a code repository. This path must be absolute.
#
# Variable: AIRFLOW__CORE__DAGS_FOLDER
#
dags_folder = /home/ubuntu/airflow/spotify_dag

# Hostname by providing a path to a callable, which will resolve the hostname.
# The format is "package.function".
#
# For example, default value "airflow.utils.net.getfqdn" means that result from patched
# version of socket.getfqdn() - see https://github.com/python/cpython/issues/49254.
#
# No argument should be required in the function specified.
# If using IP address as hostname is preferred, use value ``airflow.utils.net.get_host_ip_address``.
#
# Variable: AIRFLOW__CORE__HOSTNAME_CALLABLE
#
hostname_callable = airflow.utils.net.getfqdn

# A callable to check if a python file has airflow dags defined or not
# with argument as: `(file_path: str, zip_file: zipfile.ZipFile | None = None)`
# return True if it has dags otherwise False
# If this is not provided, Airflow uses its own heuristic rules.
#
```

So, now as we can see : there is a file name **airflow.cfg** : there the name of the folder should be same as outside folder : which is **spotify\_dag** as we can see,

We have created two nano files inside the spotify dag as well :

```

ubuntu@ip-172-31-45-95:~/airflow/spotify_dag$ ls
__pycache__ spotify_dag.py spotify_etl.py
```

There we have copied both the files which we have created above : both the jupyter files.

Then we can run the DAG and get the data.

```

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

dag = DAG('spotify_dag',
           default_args=default_args,
           description='My first etl code',
           schedule_interval=timedelta(minutes=5)) # Run the DAG every 5 minutes

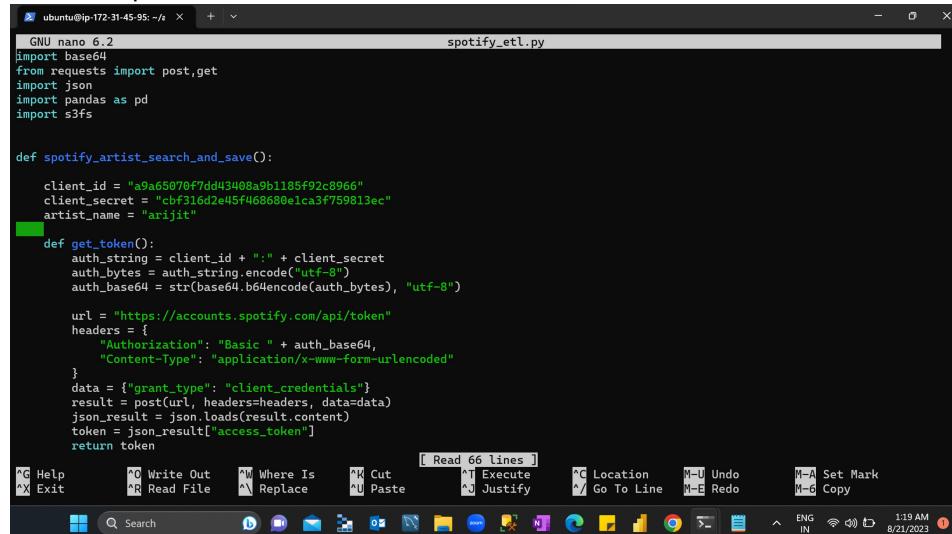
def spotify_artist_search_and_save():
    # Your ETL code here

run_etl = PythonOperator(
    task_id='complete_spotify_etl',
    python_callable=spotify_artist_search_and_save,
    dag=dag)

```

I have scheduled for every 5 minute , starting from `datetime.now()`

Like that in nano files I have pasted the data : -- >



```

GNU nano 6.2                                         spotify_etl.py
import base64
from requests import post,get
import json
import pandas as pd
import s3fs

def spotify_artist_search_and_save():

    client_id = "a9a65070f7dd43408a9b1185f92c8966"
    client_secret = "cbf316d2e45f468680e1ca3f759813ec"
    artist_name = "arjit"

    def get_token():
        auth_string = client_id + ":" + client_secret
        auth_bytes = auth_string.encode("utf-8")
        auth_base64 = str(base64.b64encode(auth_bytes), "utf-8")

        url = "https://accounts.spotify.com/api/token"
        headers = {
            "Authorization": "Basic " + auth_base64,
            "Content-Type": "application/x-www-form-urlencoded"
        }
        data = {"grant_type": "client_credentials"}
        result = post(url, headers=headers, data=data)
        json_result = json.loads(result.content)
        token = json_result["access_token"]
        return token

```

Airflow server running :

```

ubuntu@ip-172-31-45-95:~/airflow$ airflow standalone
standalone | Starting Airflow Standalone
standalone | Checking database is initialized
INFO [alembic.runtime.migration] Context impl SQLiteImpl.
INFO [alembic.runtime.migration] Will assume non-transactional DDL.
WARNI [unusual_prefix_de310399792f7c66d0607289cf7f70223c00f7ca_example_python_operator] The virtualenv_python example task requires virtualenv, please install it.
WARNI [unusual_prefix_ele1e746b8b87627ff53519aca03f086299788fc2Tutorial_taskflow_api_virtualenv] The tutorial_taskflow_api_virtualenv example DAG requires virtualenv, please install it.
WARNI [unusual_prefix_13903d1433e73a90598409829be4c61c9cdea5f_example_kubernetes_executor] The example_kubernetes_executor example DAG requires the kubernetes provider. Please install it with: pip install apache-airflow[cncf.kubernetes]
WARNI [unusual_prefix_4c1c0063a36d033a56d4343efc723c3a45c1bd59_example_local_kubernetes_executor] Could not import DAGs in example_local_kubernetes_executor.py
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/airflow/example_dags/example_local_kubernetes_executor.py", line 37, in <module>
    from kubernetes.client import models as k8s
ModuleNotFoundError: No module named 'kubernetes'
WARNI [unusual_prefix_4c1c0063a36d033a56d4343efc723c3a45c1bd59_example_local_kubernetes_executor] Install Kubernetes dependencies with: pip install apache-airflow[cncf.kubernetes]
WARNI [airflow.models.crypto] empty cryptography key - values will not be stored encrypted.
standalone | Database ready
/usr/local/lib/python3.10/dist-packages/flask_limiter/extension.py:293 UserWarning: Using the in-memory storage for trac

```

### Note :

If I want to save my data for some other singer, then I just need to change the name of the particular person it will fetch the data from the web spotify, and you need to run the AIRFLOW, and you will be able to get the data directly to the S3 Bucket.

### DAG status :



### Let's check the bucket :

Amazon S3

Buckets

- Access Points
- Object Lambda Access Points
- Multi-Region Access Points
- Batch Operations
- IAM Access Analyzer for S3

Block Public Access settings for this account

Storage Lens

- Dashboards
- AWS Organizations settings

CloudShell   Feedback   Language

Objects (4)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

[Copy S3 URI](#)  [Copy URL](#)  [Download](#)  [Open](#) [Delete](#) [Actions ▾](#)

[Create folder](#) [Upload](#)

Find objects by prefix

	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	result_karthik.csv	csv	12:10:36 (UTC+05:30)	278.0 B	Standard
<input type="checkbox"/>	result_kumar.csv	csv	August 21, 2023, 12:18:21 (UTC+05:30)	383.0 B	Standard
<input type="checkbox"/>	result_s.p..csv	csv	August 21, 2023, 12:15:20 (UTC+05:30)	166.0 B	Standard

© 2023 Amazon Web Services India Private Limited or its affiliates. Privacy Terms Cookie preferences

The screenshot shows the Amazon S3 console interface. On the left, a sidebar lists various services like Buckets, IAM Access Analyzer, and Storage Lens. The main area is titled 'Objects (4)' and displays three CSV files: 'result\_karthik.csv', 'result\_kumar.csv', and 'result\_s.p..csv'. Each file is 278.0 B in size and has a standard storage class. The files were last modified on August 21, 2023, at different times (12:10:36, 12:15:20, and 12:18:21 UTC+05:30). There are buttons for creating a folder, uploading files, and performing actions on selected objects.

We have successfully received data from Spotify web api and stored it in the S3 bucket.

**Reference :**

- [Twitter Data Pipeline using Airflow for Beginners | Data Engineering Project](#)
- [How to Use Spotify's API with Python | Write a Program to Display Artist, Tracks, and More](#)
- <https://developer.spotify.com/documentation/web-api/reference/search>