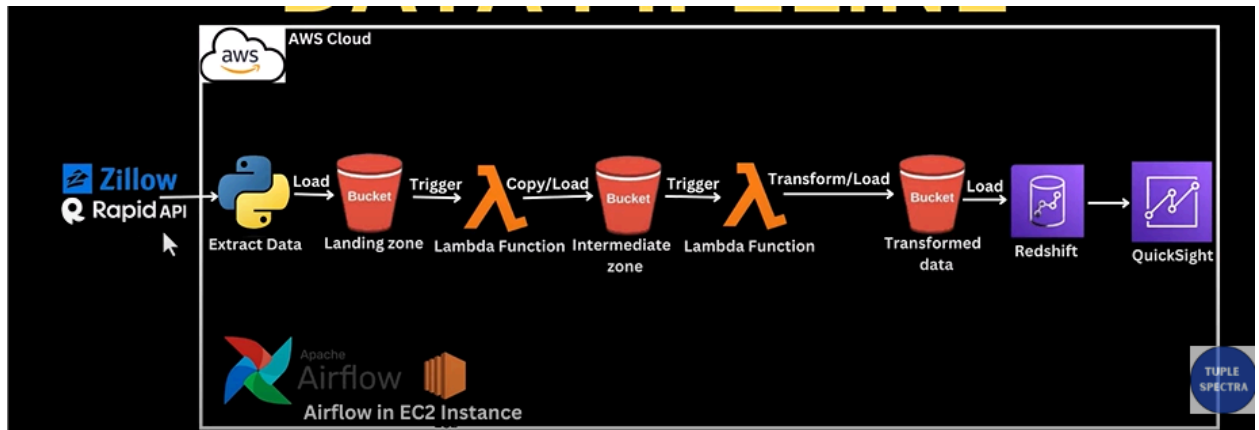


PipeLine Architecture



- 1.
2. Here we will be using python to Extract data from RapidAPI
3. Then we load this data into aws s3 bucket and the bucket is called as landing zone
4. When ever the data gets into landing zone..then lambda function will get triggered..and copies the data into another bucket(we are doing this because ..we dont want to temper the initial data)
5. Now as the data gets into intermediate zone..then our lambda func gets triggered and which make some transformations and send this data to another bucket which is transformed data
6. Next we'll load this data into aws redshift for analytical purpose
7. Also we'll be using quick sight to make some visualizations
8. This entire process we'll be made using airflow inside an EC2 instance

9. Our DAG will look like this

DAG TO BUILD

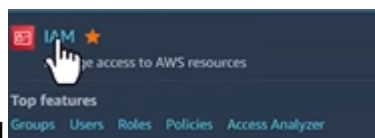


1. **Task 1: tsk_extract_zillow_data_var** - This PythonOperator task presumably extracts data relevant to Zillow. Zillow is a popular real estate marketplace, so this data could be property listings or market trends.
2. **Task 2: tsk_load_to_s3** - This task loads the extracted data from task 1 into an Amazon S3 bucket. S3 is a scalable object storage service for various data types.
3. **Task 3: tsk_is_file_in_s3_available** - This S3KeySensor operator checks if the data file from task 1 has finished loading into the S3 bucket.
4. **Task 4: tsk_transfer_s3_to_redshift** - This S3ToRedshiftOperator transfers the data from the S3 bucket in task 3 to Amazon Redshift. Redshift is a cloud-based data warehouse service used for large-scale data analytics.

10. And if our data is in redshift..we can use quick sight to make some quick visualizations

AWS - Handson(creating user groups and user)

1. Now here we'll create a group of users
2. Why do we need user group? If you want to give some privileges to s3 bucket for juniors to read and write data
3. Then we create a user group called Juniors and assign the rules to access the s3 bucket
4. Creating users_groups

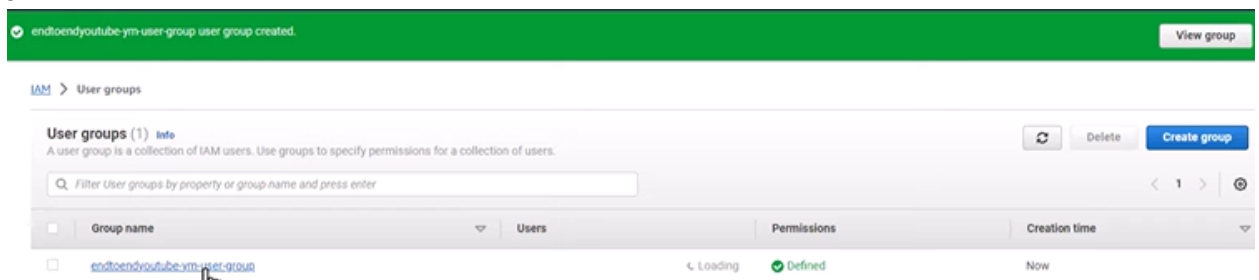


5. First we'll go to IAM

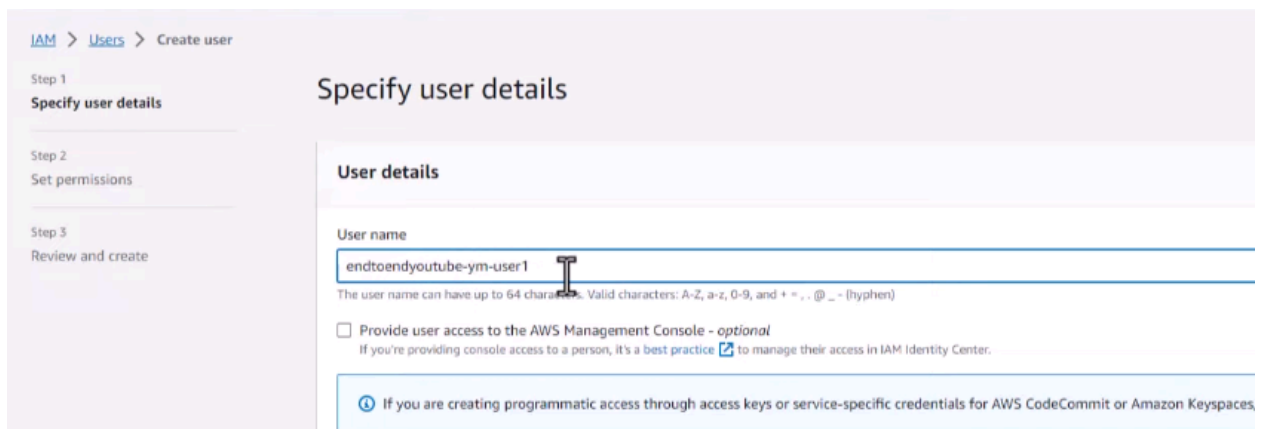
6. The click on user groups then click on create group



7. Next we'll attach administrator access to our user_group..and then we click on create group



8. Next we'll create user



User name

enotoendyoutube-ym-user1

The user name can have up to 64 characters. Valid characters: A-Z, a-z, 0-9, and + = _ . @ - (hyphen)

☒ Provide user access to the AWS Management Console - optional
If you're providing console access to a person, it's a best practice [to manage their access in IAM Identity Center](#).

Are you providing console access to a person?

User type

☐ Specify a user in Identity Center - Recommended
We recommend that you use Identity Center to provide console access to a person. With Identity Center, you can centrally manage user access to their AWS accounts and cloud applications.

☒ I want to create an IAM user
We recommend that you create IAM users only if you need to enable programmatic access through access keys, service-specific credentials for AWS CodeCommit or Amazon Keyspaces, or a backup credential for emergency account access.

Console password

☐ Autogenerated password
You can view the password after you create the user.

☒ Custom password
Enter a custom password for the user.

- Must be at least 8 characters long
- Must include at least three of the following mix of character types: uppercase letters (A-Z), lowercase letters (a-z), numbers (0-9), and symbols ! @ # \$ % ^ & * () _ + - (hyphen) = [] { } ' "

☐ Show password

☐ Users must create a new password at next sign-in - Recommended
Users automatically get the [IAMUserChangePassword](#) policy to allow them to change their own password.

[If you are creating programmatic access through access keys or service-specific credentials for AWS CodeCommit or Amazon Keyspaces, you can generate them after you create this IAM user. \[Learn more\]\(#\)](#)

- 9.
10. Now we'll add this user to the group
11. Next we'll create access keys to our users

Access key

If you lose or forget your secret access key, you cannot retrieve it. Instead, create a new access key and make the

Access key	Secret access key
AKIASSWARKAA3CN42OS7	***** Show

12. Now we'll logout of root account and sign in with user account
13. This user will be responsible for this project

AWS EC2(user1)

1. Here we need EC2 instance to create and run our project

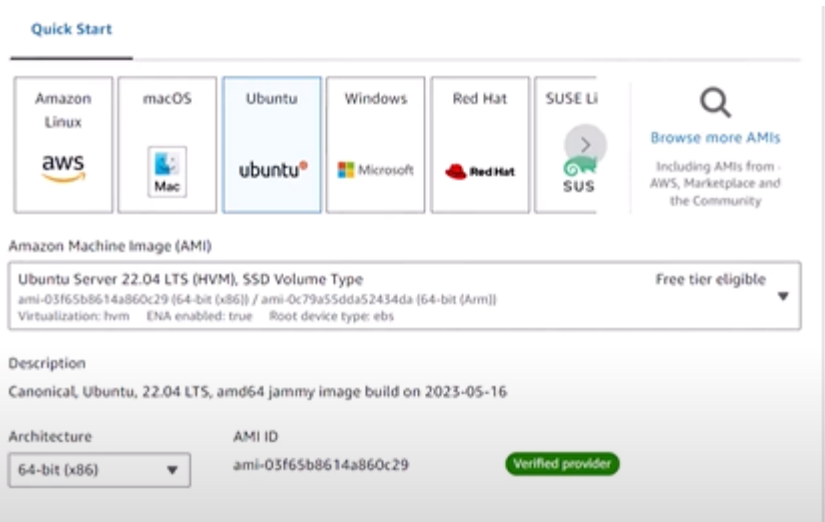
Launch instance

To get started, launch an Amazon EC2 instance, which is a virtual server in the cloud.

[Launch instance](#) [Migrate a server](#)

Note: Your instances will launch in the US West (Oregon) Region

2. We click on launch instance



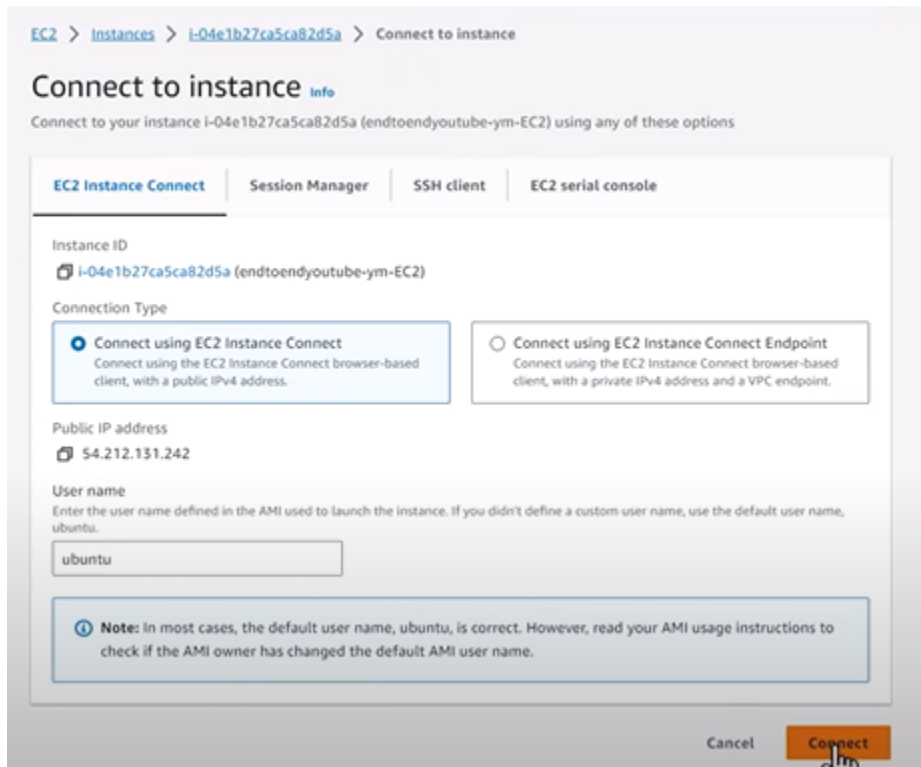
3. We select this machine
4. And in instance type we choose t2.medium

Instances (1/3) Info

Find instance by attribute or tag (case-sensitive)

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS
ec2-jupyter	i-08c7e7c9daa262570	Stopped	m5.xlarge	-	No alarms	us-west-2b	-
ec2_portfolio_gpu_ym1	i-0c5adecae120d65c5	Stopped	g5.2xlarge	-	No alarms	us-west-2c	-
endtoendyoutube-ym-EC2	i-04e1b27ca5ca82d5a	Running	t2.medium	-	No alarms	us-west-2c	ec2-54-212-131-24

5. After creating an instance..we'll start installing dependencies
6. Now we'll connect to our instance using ec2 instance connect



7. When we click on connect..it gives us the terminal

8. We'll run this command in our instance

```
ubuntu@ip-172-31-7-187:~$ sudo apt update
```

9. After that we'll be installing

```
ubuntu@ip-172-31-7-187:~$ sudo apt install python3-pip
```

10. Next we'll install virtual env

```
ubuntu@ip-172-31-7-187:~$ sudo apt install python3.10-venv
```

11. Now we'll activate virtual environment

```
No VM guests are running outdated hypervisor (qemu) binaries on this host.
ubuntu@ip-172-31-7-187:~$ python3 -m venv endtoendyoutube venv
ubuntu@ip-172-31-7-187:~$ source endtoendyoutube venv/bin/activate
```

12. Now we'll install awscli in our env

```
(endtoendyoutube_venv) ubuntu@ip-172-31-7-187:~$ pip install --upgrade awscli
```

13. Next we'll install apache airflow

```
(endtoendyoutube_venv) ubuntu@ip-172-31-7-187:~$ sudo pip install apache-airflow
```

14. Next we'll initiate the airflow using airflow standalone

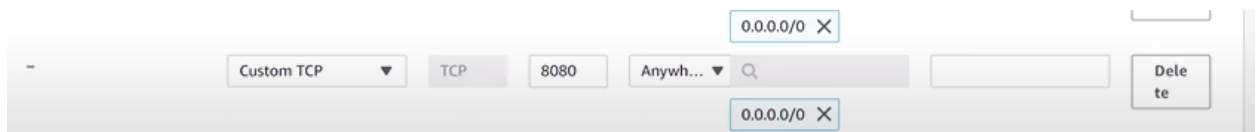
```
(endtoendyoutube_venv) ubuntu@ip-172-31-7-187:~$ airflow standalone
```

15. We'll get this output

```
Airflow is ready
Login with username: admin password: Ty5bEwN5PHhwXC4
Airflow Standalone is for development purposes only. Do not use this in production!
```

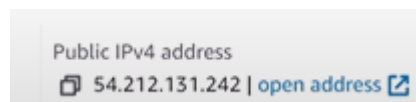
16. Next we need to go to airflow UI

17. To go there first we have change our inbound rules in security group attached to ec2 instance



and click on savw

18. To go to airflow UI copy



or copy this



19. Now enter ipaddress + 8080 in the url

Not secure | 54.212.131.242:8080/login/

20. And we get airflowUI now

Sign In

Enter your login and password below:

Username:

Password:

Please fill out this field.

Sign in

21. We login to airflow..using the credentials given to us while installing airflow

```
standalone | Airflow is ready
standalone | Login with username: admin password: Ty5bEw[[5PHhwwXC4
standalone | Airflow Standalone is for development purposes only. Do not
```

22. After logging in using our credentials..we can see our airflow UI

The screenshot shows the Databricks DAGs page. At the top, there's a navigation bar with links like Airflow, DAGs, Cluster Activity, Datasets, Security, Browse, Admin, and Docs. Below this, there are two warning banners: one about not using SQLite as a metadata DB in production, and another about not using the SequentialExecutor in production. The main section is titled 'DAGs' and contains a table of DAGs. The table has columns: DAG, Owner, Runs, Schedule, Last Run, Next Run, Recent Tasks, and Actions. The DAGs listed are 'dataset_consumes_1', 'dataset_consumes_1_and_2', 'dataset_consumes_1_never_scheduled', and 'dataset_consumes_unknown_never_scheduled'. Each DAG has a 'Runs' column with a circular progress indicator and a 'Next Run' column with a date and time. The 'Actions' column contains a play button, a stop button, and a link icon.

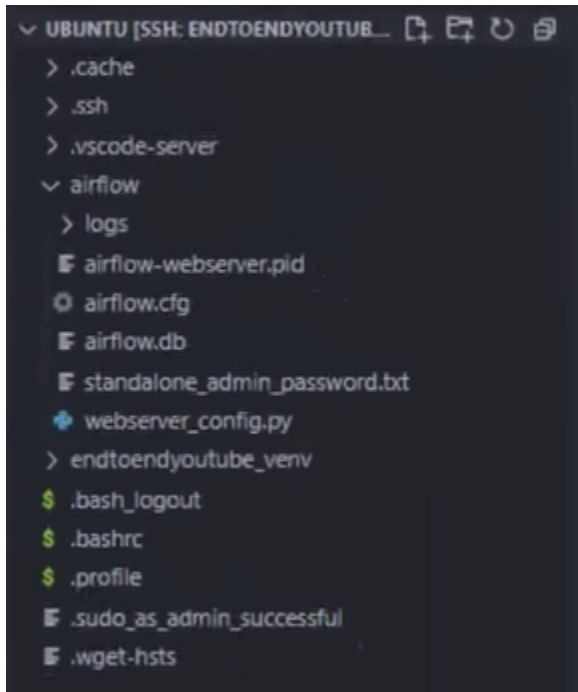
DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks	Actions
dataset_consumes_1 consumes dataset scheduled	airflow	○○○○	Dataset		On schedule/output_1.txt	○○○○○○○○○○○○○○○○○○○○	▶ ⏏ 🔗
dataset_consumes_1_and_2 consumes dataset scheduled	airflow	○○○○	Dataset		0 of 2 datasets updated	○○○○○○○○○○○○○○○○○○○○	▶ ⏏ 🔗
dataset_consumes_1_never_scheduled consumes dataset scheduled	airflow	○○○○	Dataset		0 of 2 datasets updated	○○○○○○○○○○○○○○○○○○○○	▶ ⏏ 🔗
dataset_consumes_unknown_never_scheduled dataset scheduled	airflow	○○○○	Dataset		0 of 2 datasets updated	○○○○○○○○○○○○○○○○○○○○	▶ ⏏ 🔗

23.

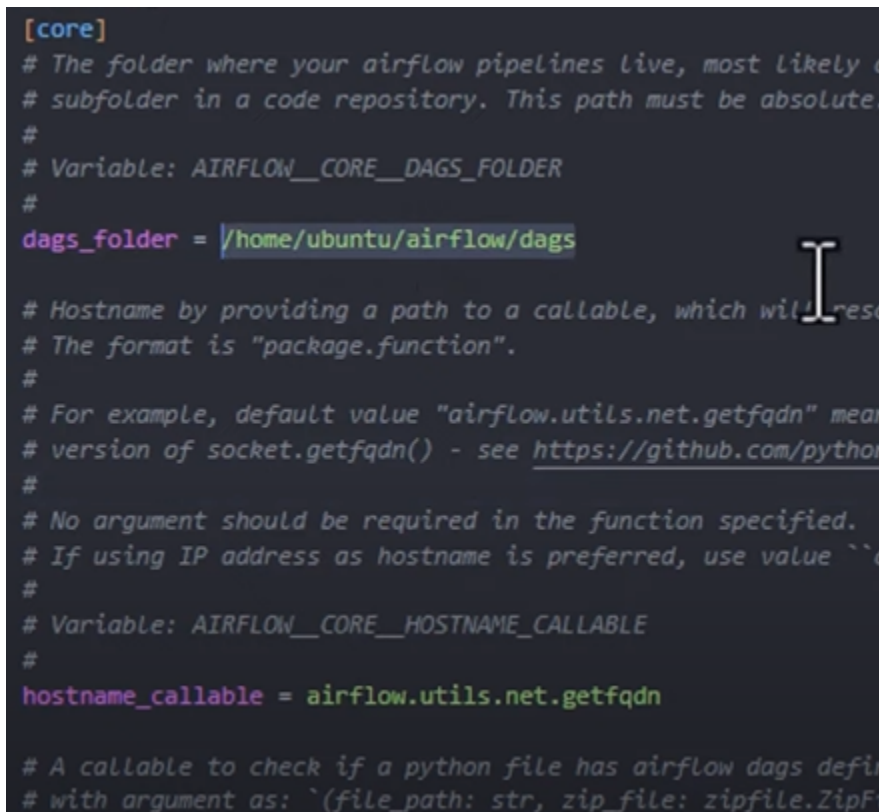
if we dont want to see this samples dags ..then we can set `load_examples = False` and restart our airflow

24. Now we'll connect our ec2 to vs code...(tutorial is in the same channel)

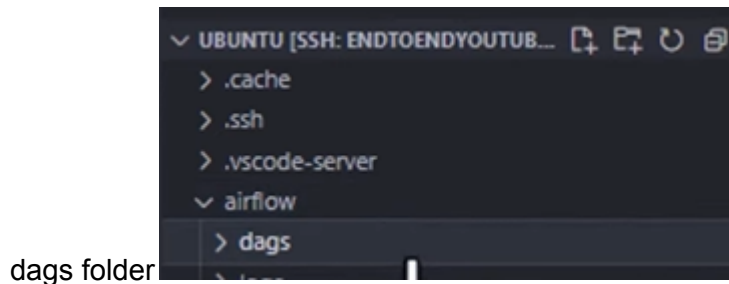
25. Here we can see the files in our ec2 instance



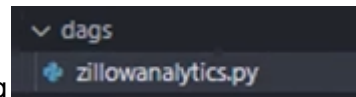
26. Lets look at airflow.cfg



27. Dags folder is the folder where our dags will live..so in our ec2 instance ..we create a



dags folder

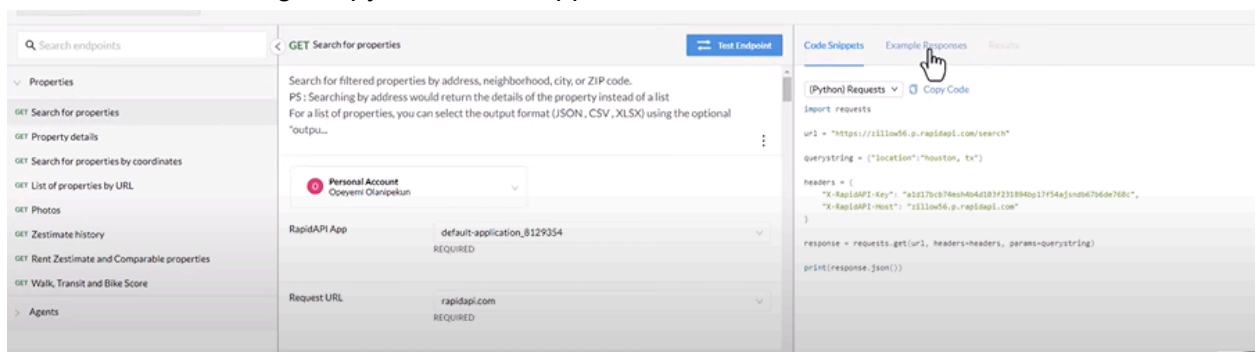


28. Now we will create a .py file inside our dag

29. Next we will use rapidAPI website.. It will provide us data for zillow dataset

30. We'll sign up in rapidAPI..next we will search for zillow...now we have to subscribe for zillow test point

31. Next we'll be selecting the python code snippet



32. Now if we want to connect with this and extract data..we need

```
url = "https://zillow56.p.rapidapi.com/search"
```

33. Here in our query string we have default address of houston,txwhich when passed to response..gives us the data of that location

```
import requests

url = "https://zillow56.p.rapidapi.com/search"

querystring = {"location": "houston, tx"}

headers = {
    "X-RapidAPI-Key": "a1d17bcb74msh4b4d103f231894bp17f54ajsndb67b6de768c",
    "X-RapidAPI-Host": "zillow56.p.rapidapi.com"
}

response = requests.get(url, headers=headers, params=querystring)

print(response.json())
```

Task1 (get data from api)

1. Now here our first task is to get data



2. So lets write python code..that extracts data from api

```
File Edit Selection View Go Run Terminal Help
EXPLORER
UBUNTU [SSH: ENDTOENDYOUTUBE-YM-EC2]
> .cache
> .ssh
> .vscode-server
airflow
  dags
    zillowanalytics.py
airflow.cfg
zillowanalytics.py
airflow > dags > zillowanalytics.py
1 from airflow import DAG
2 from datetime import timedelta, datetime
```

```
default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2023, 8, 1),
    'email': ['myemail@domain.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 2,
    'retry_delay': timedelta(seconds=15)
}
```

3. Then we have default args

Let's explore each argument:

- **'owner': 'airflow':** This sets the DAG owner to "airflow". The owner field is used for tracking and auditing purposes.
- **'depends_on_past': False:** This indicates that tasks in the DAG don't rely on the success of tasks in previous DAG runs. If set to `True`, a task won't start until the corresponding task in the prior DAG run finishes successfully.
- **'start_date': datetime(2023, 8, 1):** This sets the start date for the DAG to August 1st, 2023. This is the earliest date the DAG can be scheduled to run.
- **'email': ['myemail@domain.com']:** This defines the list of email addresses to notify in case of DAG failures. Here, it's set to notify `myemail@domain.com`.
- **'email_on_failure': False:** Disables email notifications for DAG failures.
- **'email_on_retry': False:** Disables email notifications for retries.
- **'retries': 2:** Sets the number of times a task should be retried if it fails. In this case, a task will be rerun up to two times after the initial failed attempt.
- **'retry_delay': timedelta(seconds=15):** This defines the delay between retries. Here, there's a 15-second wait time between retries.

By default, these arguments will be applied to all tasks within the DAG unless explicitly overridden by individual task definitions.

```
with DAG('zillow_analytics_dag',
        default_args=default_args,
        schedule_interval = '@daily',
        catchup=False) as dag:
```

4. Next we define our DAG

- A DAG is defined using the `DAG` constructor, specifying arguments like:
 - `dag_id` : A unique identifier for the DAG.
 - `schedule_interval` : How often the DAG should run (e.g., `'@daily'`, `'@hourly'`).
 - `default_args` : Default arguments applied to tasks within the DAG (refer to previous explanation).
 - `start_date` : The earliest date the DAG can be scheduled to run.
 - `catchup` : Whether to run missed tasks from previous intervals (often set to `False`).

5. Next we define python operator and call our `extract_zillow_data` function

```
with DAG('zillow_analytics_dag',
        default_args=default_args,
        schedule_interval = '@daily',
        catchup=False) as dag:

    extract_zillow_data_var = PythonOperator(
        task_id= 'tsk_extract_zillow_data_var',
        python_callable=extract_zillow_data,
        op_kwargs={'url': 'https://zillow56.p.rapidapi.com/search', 'querystring': {"location":"houston, tx"},
    })
```

6. Sample code explaining use of python operator

```
from airflow import DAG
from airflow.operators.python import PythonOperator

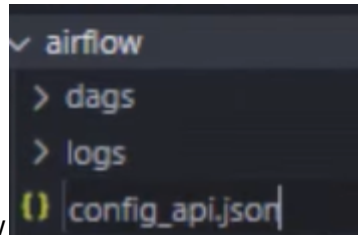
def my_custom_function(name):
    """Greets the provided name"""
    print(f"Hello, {name}!")

with DAG(
    dag_id="my_dag_with_python",
    start_date=datetime(2024, 4, 5),
    schedule_interval=None,
) as dag:

    greet_task = PythonOperator(
        task_id="greet",
        python_callable=my_custom_function,
        op_args=["Airflow"], # Pass "Airflow" as an argument to the function
    )
```

7. SO here we dont want to expose the API keys...

```
headers = {
    "X-RapidAPI-Key": "a1d17bcb74msh4b4d103f231894bp17f54ajsndb67b6de768c",
    "X-RapidAPI-Host": "zillow56.p.rapidapi.com"
```



8. So we create a json file inside airflow next we will copy our key and host to this file

```
airflow > {} config_api.json > ...
1
2 "X-RapidAPI-Key": "a1d17bcb74msh4b4d103f231894bp17f54ajsndb67b6de768c",
3 "X-RapidAPI-Host": "zillow56.p.rapidapi.com"
4
```

9. Next we'll load this json file in our zillowanalytics.py file

```
# Load JSON config file
with open('/home/ubuntu/airflow/config_api.json', 'r') as config_file:
    api_host_key = json.load(config_file)
```

```
extract_zillow_data_var = PythonOperator(
    task_id= 'tsk_extract_zillow_data_var',
    python_callable=extract_zillow_data,
```

10. Next we will concentrate on python callable function

```
def extract_zillow_data(**kwargs):
    url = kwargs['url']
    headers = kwargs['headers']
    querystring = kwargs['querystring']
    dt_string = kwargs['date_string']
    # return headers
    response = requests.get(url, headers=headers, params=querystring)
    response_data = response.json()

    # Specify the output file path
    output_file_path = f"/home/ubuntu/response_data_{dt_string}.json"
    file_str = f'response_data_{dt_string}.csv'

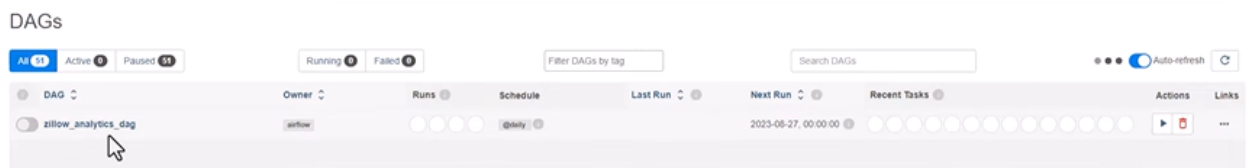
    # Write the JSON response to a file
    with open(output_file_path, "w") as output_file:
        json.dump(response_data, output_file, indent=4) # indent for pretty formatting
    output_list = [output_file_path, file_str]
    return output_list
```


11.
12. Here in our code **kwargs are

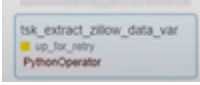
```
op_kwargs={'url': 'https://zillow56.p.rapidapi.com/search', 'querystring': ("location": "houston, tx"), 'headers': api_host_key, 'date_string': dt_row_string
})
```

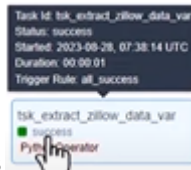
13. We have used date_string ...it is used to give naming for our extracted data...see pic and code
14. Now lets test our DAG

15. If we save our code..and refresh the DAGs...then we can see our dag in airflow UI

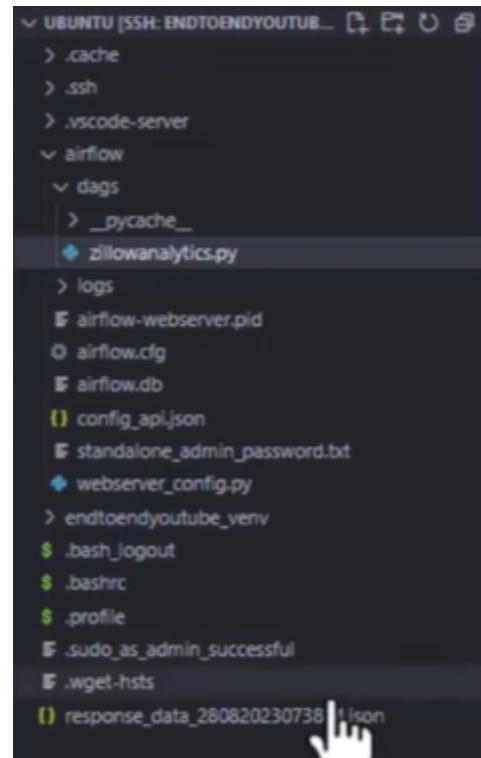


16. Now we can trigger our DAG using 

17.  here we have received an error..and we solve it



18. And again we run it ..we receive success



19. Here we can see the data been created

```

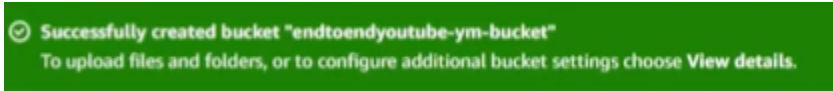
{
  "results": [
    {
      "bathrooms": 3.0,
      "bedrooms": 4.0,
      "city": "Houston",
      "country": "USA",
      "currency": "USD",
      "daysOnZillow": 0,
      "homeStatus": "FOR_SALE",
      "homeStatusForHDP": "FOR_SALE",
      "homeType": "SINGLE_FAMILY",
      "imgSrc": "https://photos.zillowstatic.com/fp/2c0f5492cfe6eb36641b2871931ebe1",
      "isFeatured": false,
      "isNonOwnerOccupied": true,
      "isPreforeclosureAuction": false,
      "isPremierBuilder": false,
      "isShowcaseListing": false,
      "isUnmappable": false,
      "isZillowOwned": false,
      "latitude": 29.984749,
      "listing_sub_type": {
        "is_FSBA": true
      },
      "livingArea": 2712.0,
      "longitude": -95.510376,
      "lotAreaUnit": "sqft",
      "lotAreaValue": 9239.076,
      "price": 299900.0,
      "priceForHDP": 299900.0,
      "rentZestimate": 2377,
      "shouldHighlight": false,
      "state": "TX",
      "streetAddress": "13310 Glen Erica Dr",
      "taxAssessedValue": 284876.0,
      "zestimate": 300300,
      "zipcode": "77069",
      "zpid": 28138835
    }
  ]
}

```

20. Here we have completed the task 1

Task2

1. Here first we need to create a s3 bucket



2. For loading this data to s3..we use bash operator..we need to import bash importer

```

load_to_s3 = BashOperator(
    task_id='tsk_load_to_s3',
    bash_command='aws s3 mv {{ ti.xcom_pull("tsk_extract_zillow_data_var")[0]}} s3://endtoendyoutube-ym-bucket/'
)

```


3. `Ti.xcom = task_instance.xcom_pull`

In this case, the command is:

```
mv ({ti.xcom_pull("tsk_extract_zillow_data_var")}[0]}) s3://endtoendyoutube-ym-bucket/
```

This command appears to be moving a file, the output from the `tsk_extract_zillow_data_var` task, to an S3 bucket.

Here's a breakdown of the code:

- `mv` : This is the Unix move command.
- `({ti.xcom_pull("tsk_extract_zillow_data_var")}[0]})` : This part of the code appears to be using Airflow's XComs to pull the output from the `tsk_extract_zillow_data_var` task. XComs are a way for tasks in an Airflow DAG to communicate with each other by passing key/value pairs.
- `s3://endtoendyoutube-ym-bucket/` : This is the destination for the file; an S3 bucket.

4. Now we need to connect the ec2 with our s3...to connect...we create a role and provide `amazons3fullaccess` policy

The screenshot shows the AWS IAM console interface for creating a new role. The left sidebar shows the navigation menu with 'IAM' selected. The main content area is titled 'Name, review, and create' and shows the 'Role details' section. The 'Role name' field is populated with 'ec2severalAccess-08232023'. The 'Description' field contains the text 'Allows EC2 instances to call AWS services on your behalf.' Below this, the 'Step 1: Select trusted entities' section shows a JSON policy document for an EC2 instance profile role. The policy allows the 'sts:AssumeRole' action for the principal 'ec2.amazonaws.com'. The 'Step 2: Add permissions' section is visible at the bottom, showing a table with the 'AmazonS3FullAccess' policy attached as a 'Permissions policy'.

Step 1: Select trusted entities

```
1 {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Effect": "allow",
6       "Action": [
7         "sts:AssumeRole"
8       ],
9       "Principal": {
10        "Service": [
11          "ec2.amazonaws.com"
12        ]
13      }
14    }
15  ]
16 }
```

Step 2: Add permissions

Policy name	Type	Attached as
AmazonS3FullAccess	AWS managed	Permissions policy

5. Later after connecting our ec2 with s3...we give task priority and save our program

```
# Write the JSON response to a file
with open(output_file_path, "w") as output_file:
    json.dump(response_data, output_file, indent=4) # Indent for pretty formatting
output_list = [output_file_path, file_str]
return output_list

default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2023, 8, 1),
    'email': ['myemail@domain.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 2,
    'retry_delay': timedelta(seconds=15)
}

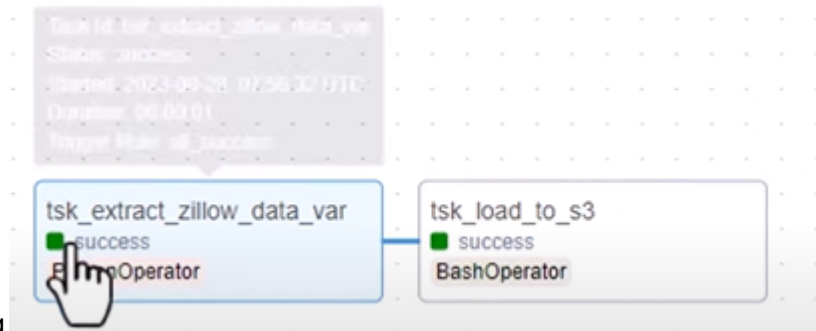
with DAG('zillow_analytics_dag',
        default_args=default_args,
        schedule_interval = '@daily',
        catchup=False) as dag:

    extract_zillow_data_var = PythonOperator(
        task_id='tsk_extract_zillow_data_var',
        python_callable=extract_zillow_data,
        op_kwargs={'url': 'https://zillow56.p.rapidapi.com/search', 'querystring': {'location': 'houston, tx'}, 'headers': api_host_key, 'date_string': dt_now_string})

    load_to_s3 = BashOperator(
        task_id='tsk_load_to_s3',
        bash_command='aws s3 mv {{ ti.xcom_pull("tsk_extract_zillow_data_var") }} s3://endoendyoutube-ym-bucket/'
    )

    extract_zillow_data_var >> load_to_s3
```

6. Next we trigger the dag



- 7.

Amazon S3 > Buckets > endoendyoutube-ym-bucket

endoendyoutube-ym-bucket [info](#)

Objects | Properties | Permissions | Metrics | Management | Access Points

Objects (1)

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](#)

[Refresh](#) [Copy S3 URI](#) [Copy URL](#) [Download](#) [Open](#) [Delete](#) [Actions](#) [Create folder](#) [Upload](#)

Find objects by prefix

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	response_data_28082023075632.json	json	August 28, 2023, 00:56:38 (UTC-07:00)	601.0 KB	Standard

8. Here we can see our data got dropped to s3 bucket