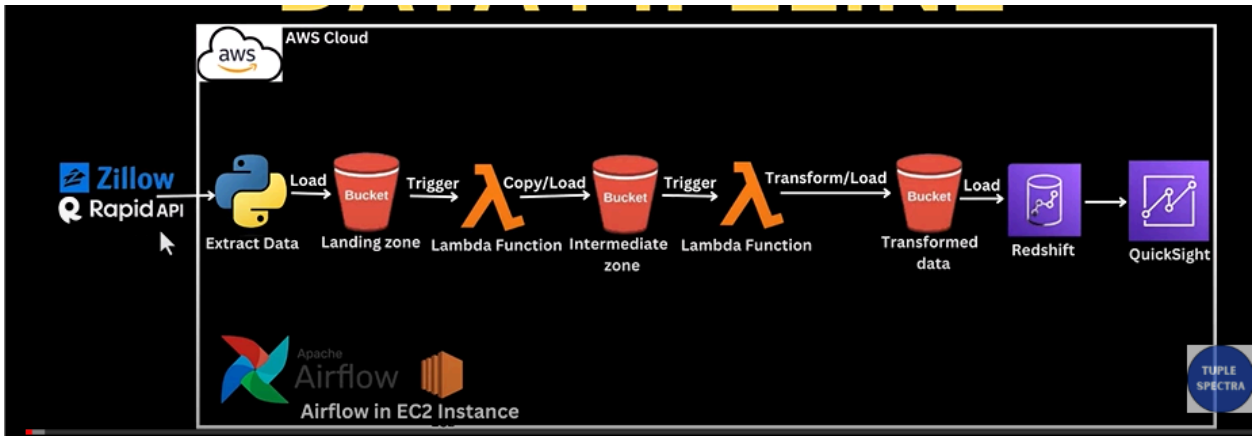
PipeLine Architecture

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

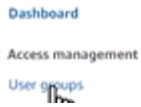
# DAG TO BUILD


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

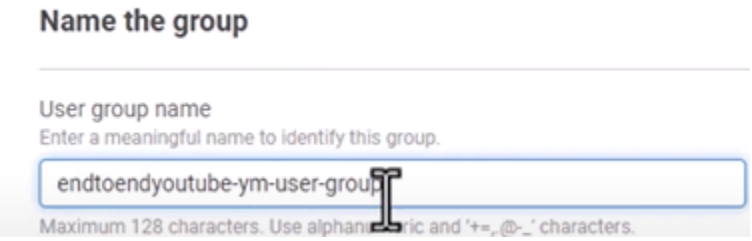
AWS - Handson(creating user groups and user)

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 

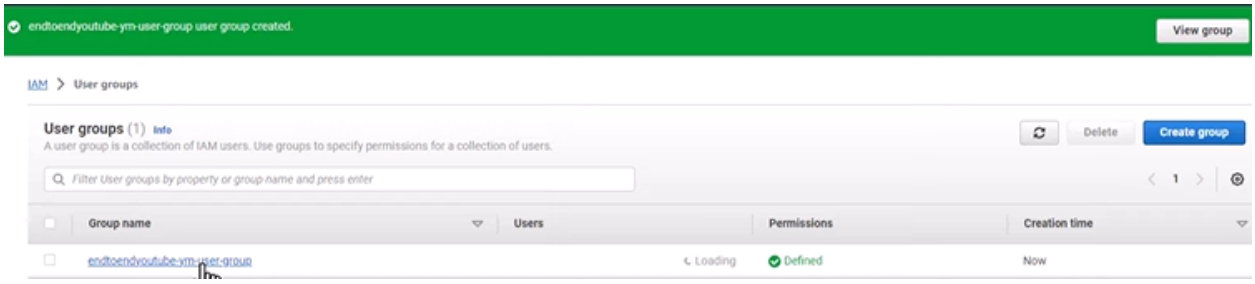
6. The click on user groups 

then click on create group

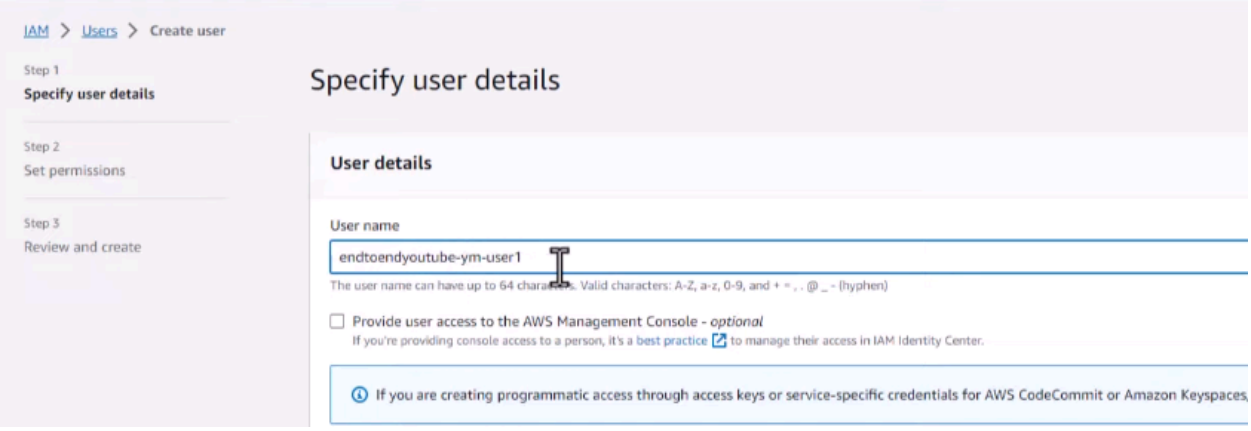
give a name to group

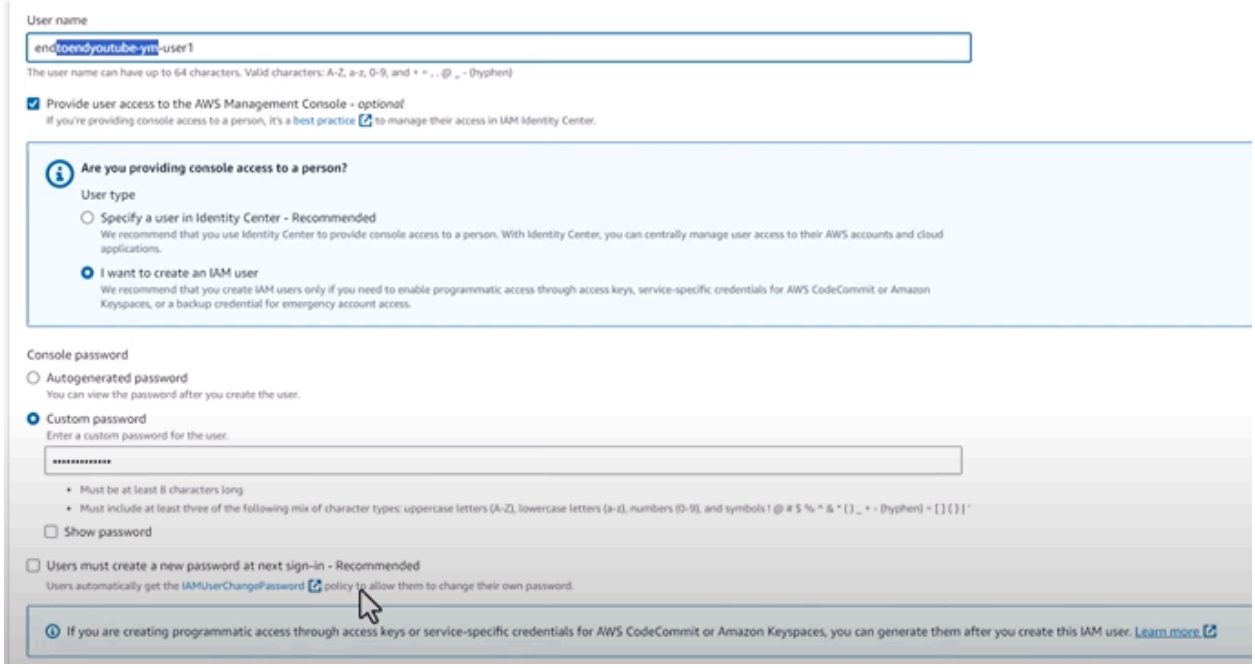



7. Next we'll attach administrator access to our user_group..and then we click on create group

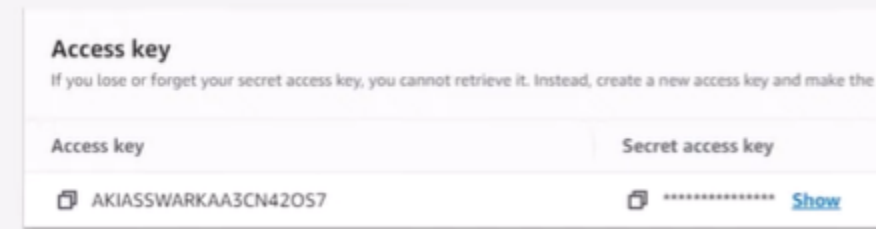



8. Next we'll create user



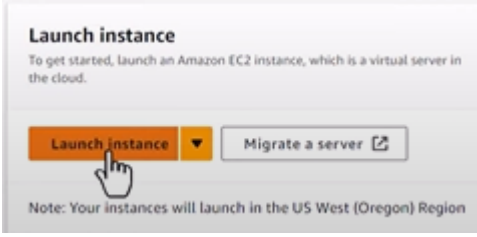
9. 

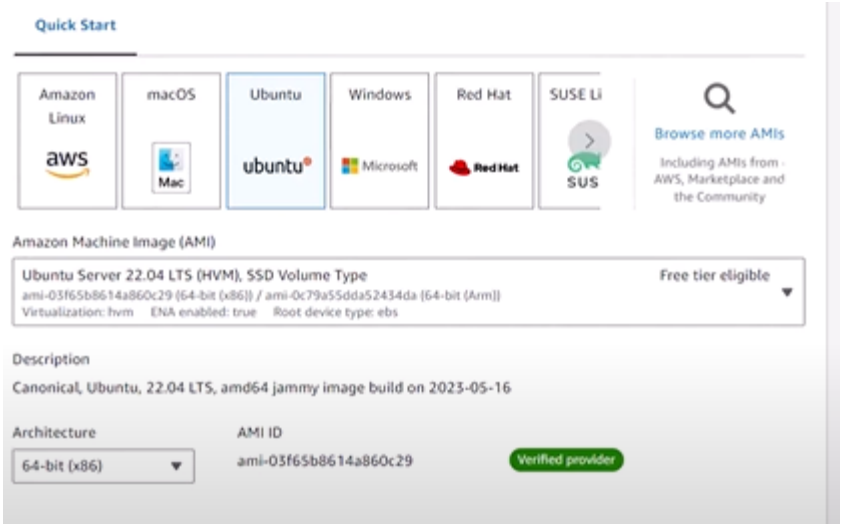
10. Now we'll add this user to the group
11. Next we'll create access keys to our users



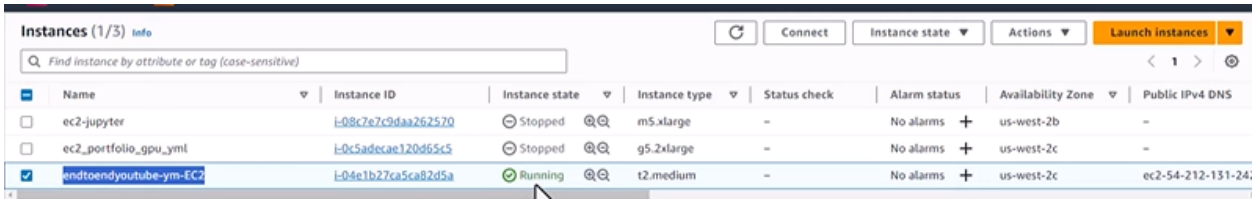

12. Now we'll logout of root account and sign in with user account
13. This user will be responsible for this project

AWS EC2(user1)

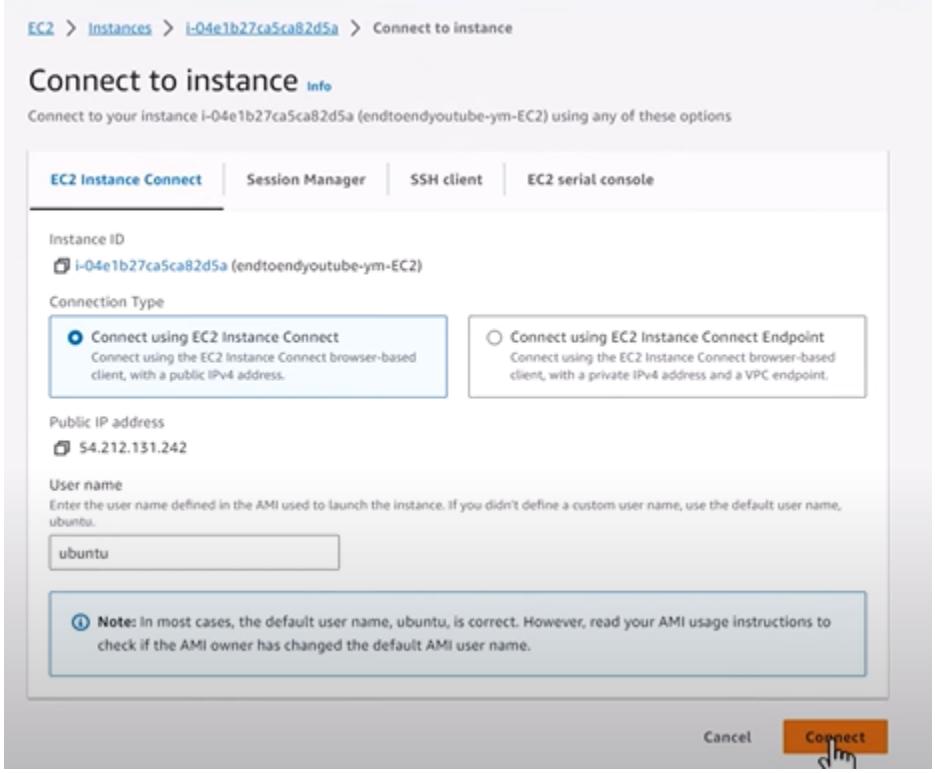
1. Here we need EC2 instance to create and run our project
2. We click on launch instance 

3. We select this machine 

4. And in instance type we choose t2.medium




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
```

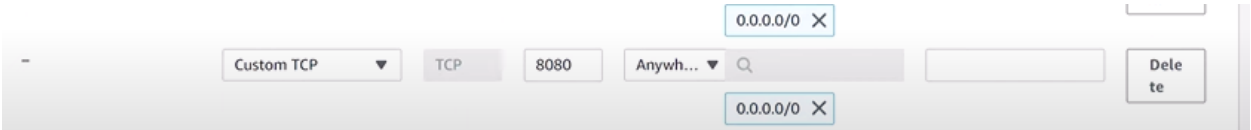
14. Next we'll initiate the airflow using airflow standalone

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

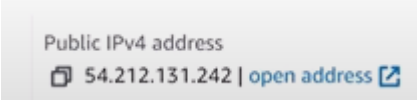
15. We'll get this output

```
Airflow is ready
Login with username: admin  password: Ty5bEwN5PHhwwXC4
Airflow Standalone is for development purposes only. Do not use this in production!
```

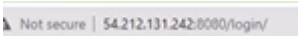
16. Next we need to go to airflow UI
17. To go there first we have change our inbound rules in security group attached to ec2 instance




and click on savw

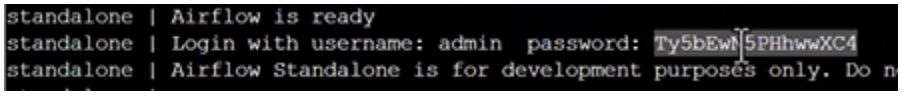
18. To go to airflow UI copy 

or copy this

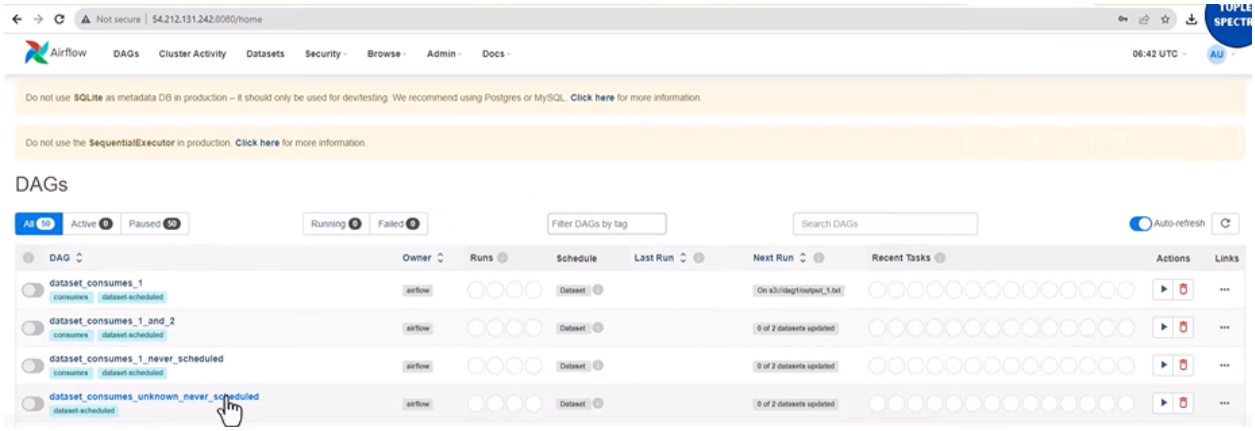

19. Now enter ipaddress + 8080 in the url 

20. And we get airflowUI now


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



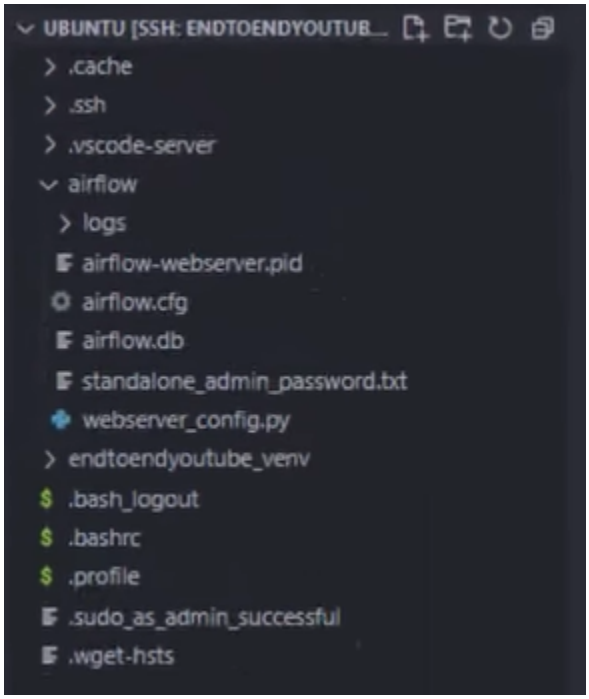

22. After logging in using our credentials..we can see our airflow UI



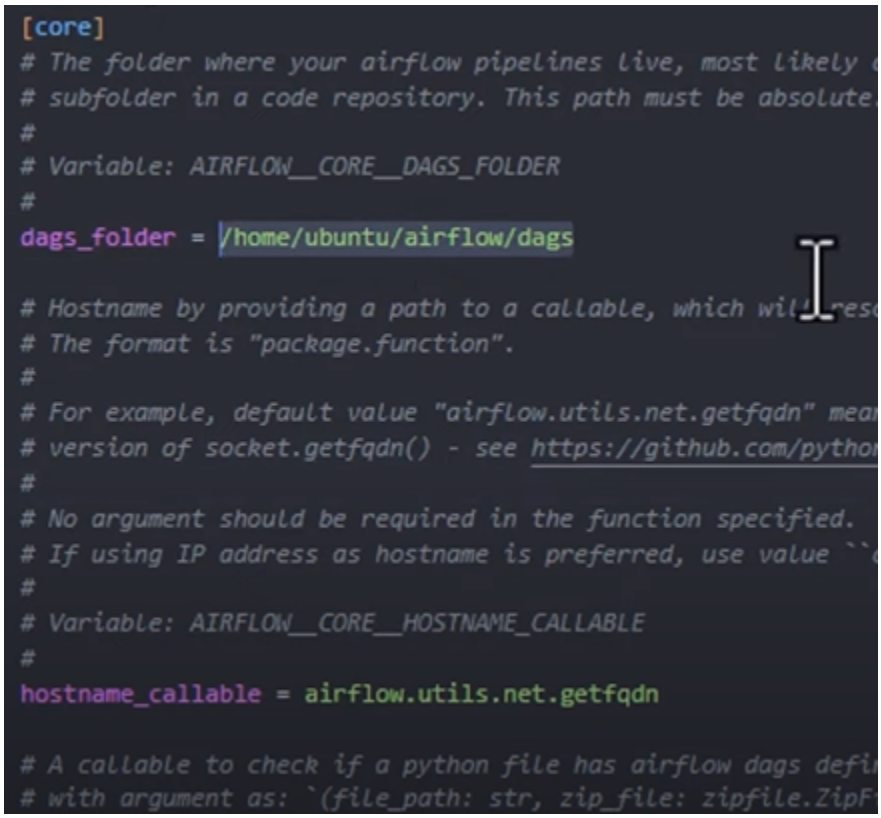

23. if we dont want to see this samples dags ..then we can set load_examples = False and restart our airflow

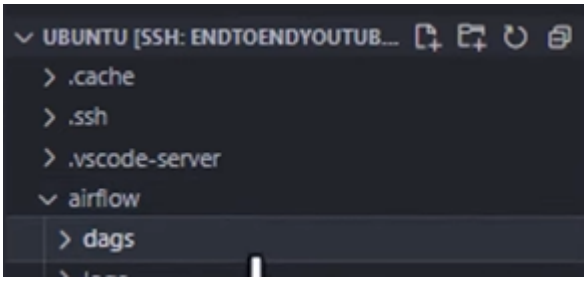
24. Now we'll connect our ec2 to vs code…(tutorial is in the same channel)

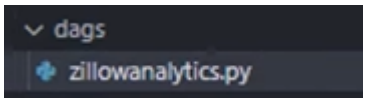
25. Here we can see the files in our ec2 instance



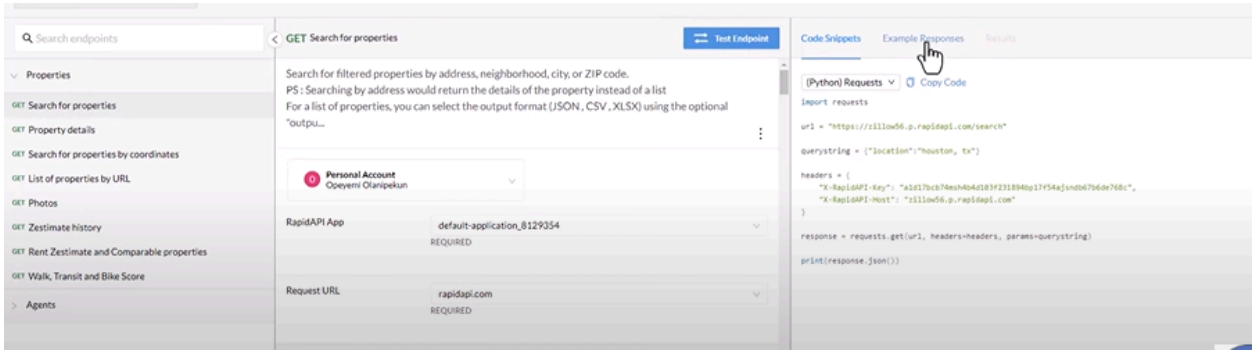

26. Lets look at airflow.cfg



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 signup 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"
```

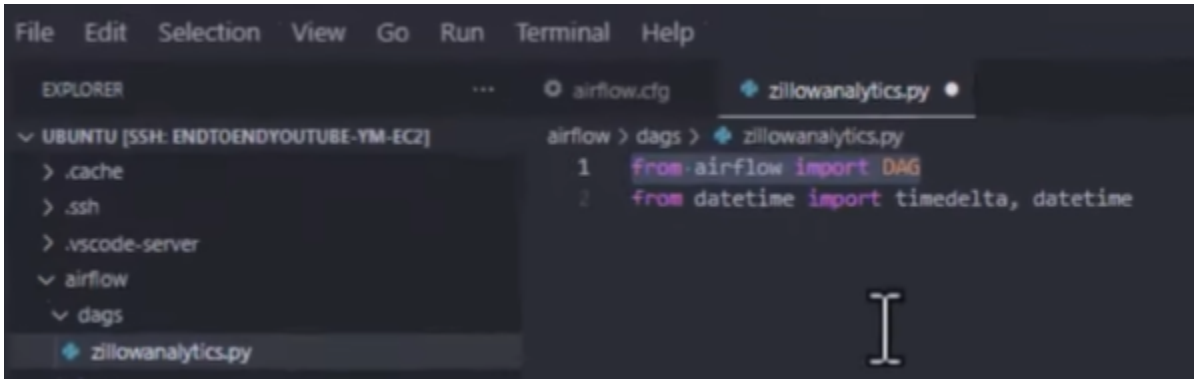
33. Here in our query string we have default address of houston,tx ….which when passed to response..gives us the data of that location

```
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



3. Then we have default args

```
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)
}
```

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.

4. Next we define our DAG

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

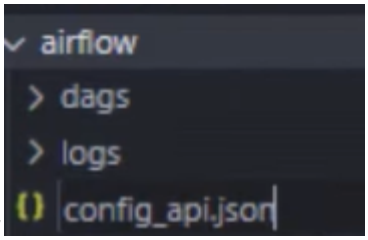
```
with DAG('zillow_analytics_dag',
        default_args=default_args,
        schedule_interval = '@daily',
        catchup=False) as dag:

        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

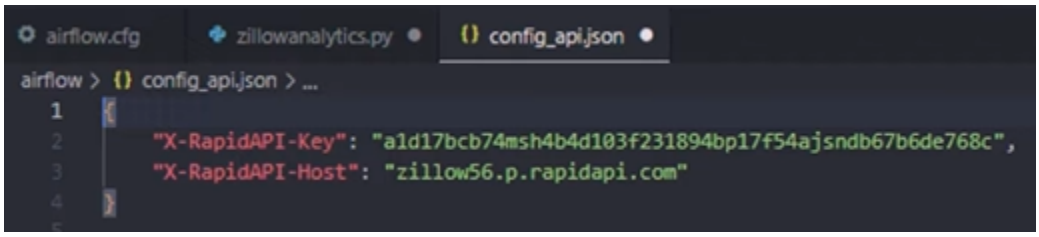
```
from airflow import DAG
from airflow.operators.python import PythonOperator

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




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)
```

10. Next we will concentrate on python callable function

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

11.

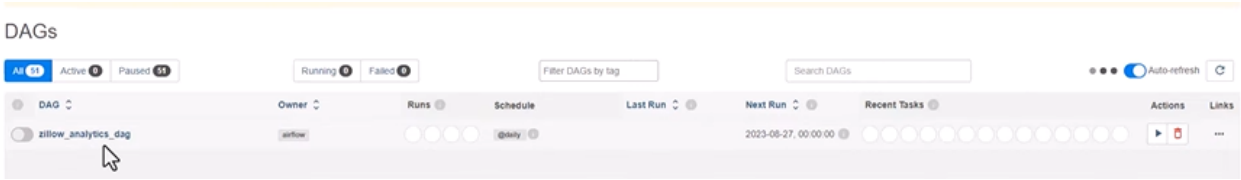
```
def extract_zillow_data(**kwargs):
    url = kwargs['url']
    headers = kwargs['headers']
    querystring = kwargs['querystring']
    dt_string = kwargs['date_string']
    # return headers
    response = requests.get(url, headers=headers, params=querystring)
    response_data = response.json()


    # 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
```

12. Here in our code **kwargs are

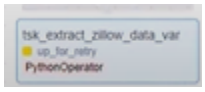
```
op_kwargs={'url': 'https://zillow56.p.rapidapi.com/search', 'querystring': {"location":"houston, tx"}, 'headers': api_host_key, 'date_string':dt_now_string}
)
```

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

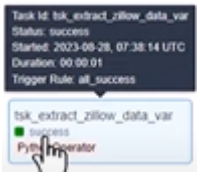
15. If we save our code..and refresh the DAGs…then we can see our dag in airflow UI



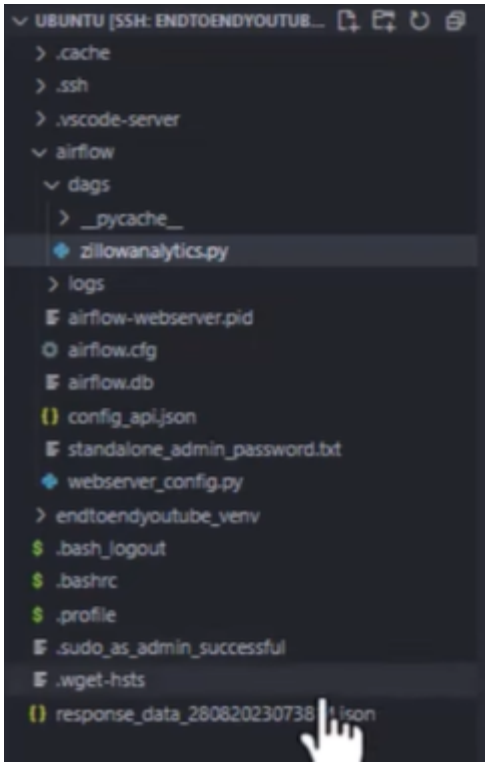

16. Now we can trigger our DAG using 

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

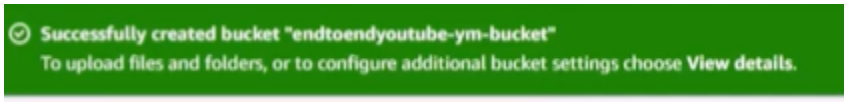


```
{
    "results": [
        {
            "bathrooms": 3.0,
            "bedrooms": 4.0,
            "city": "Houston",
            "country": "USA",
            "currency": "USD",
            "daysOnZillow": 0,
            "homeStatus": "FOR_SALE",
            "homeStatusForHDP": "FOR_SALE",
            "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:

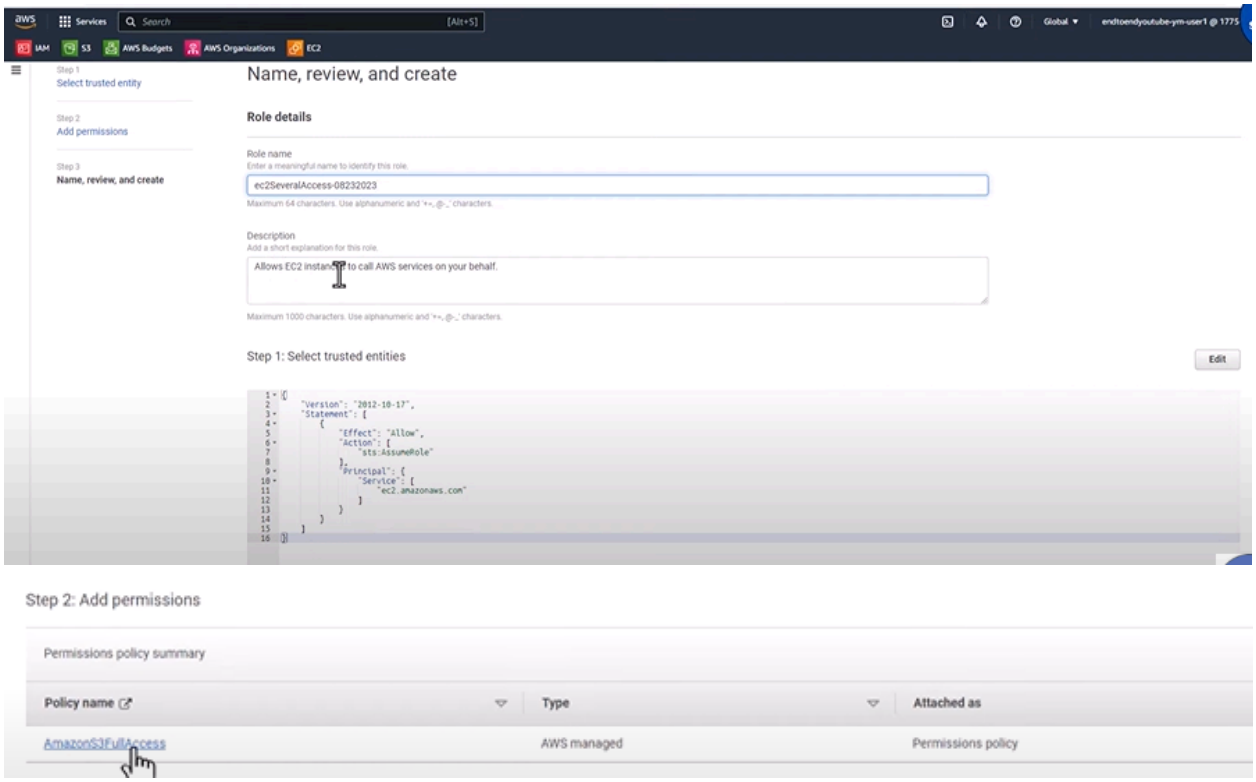
```
mv ({ti.xcom_pull("tsk_extract_zillow_data_var")[0]}) s3://endtoendyoutube-ym-bucke
```

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.

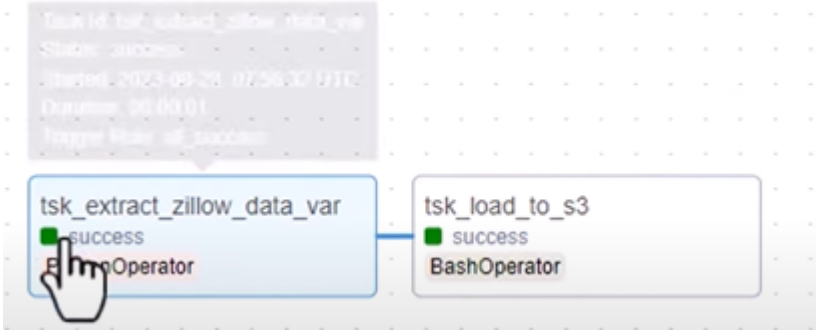
4. Now we need to connect the ec2 with our s3…to connect…we create a role and provide amazons3fullaccess policy



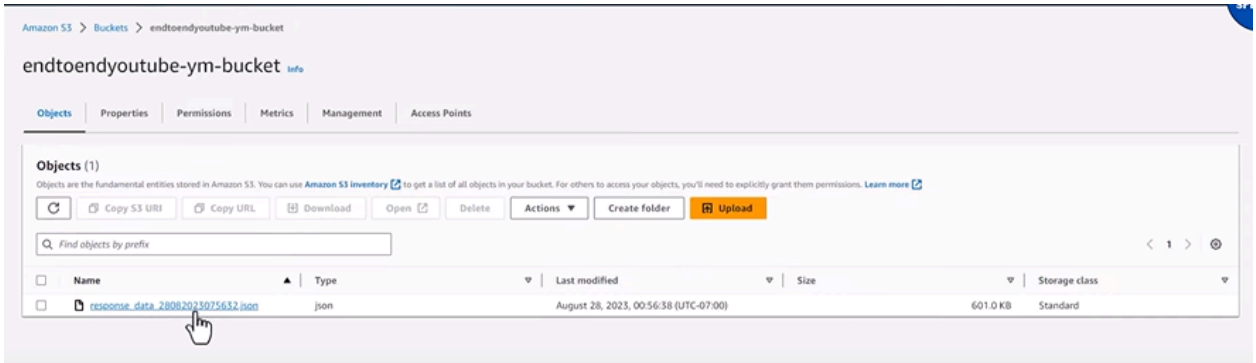
5. Later after connecting our ec2 with s3…we give task priority and save our program

```python
    # 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")[0]}} s3://endtoendyoutube-ym-bucket/',
        )

        extract_zillow_data_var >> load_to_s3
```

6. Next we trigger the dag




7.




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