Docs -

Docker Commands -

```
docker compose build
docker compose up
docker compose down -v
```

Things to do -

- Checks such that level code should have any interger or not
- sectorcode departmencode ministrycode projectcode datagranuatyid kpiid instancecode frequencyid
- from and to date and datadate (YYYY-MM-DD HH:MM:SS) `imp`
- level1 to level5 checks
- token optimization (global var)

Airflow task order -

1. Data comes through this api - http://10.23.124.59/getTableData
2. Data is then sent to this api change port - http://localhost:2999/getData flask_app.py
3. to test for encryptionKey use this api and put proper token - http://10.23.124.59:2222/getEncryptionKey?token=G4VF4B7fy70jAEcPndsKmKxVShmNTEFfIO0dlq89udE=
4. Data is validated through etl_m12.py

Docker debugs -

to remove all containers and volumes

```
docker compose down --volumes --rmi all
```

Both of these changes are for POSTGRES:15

to use `POSTGRES_DB: postgres` -

```
AIRFLOW__DATABASE__SQL_ALCHEMY_CONN: postgresql+psycopg2://airfl
```

to use `POSTGRES_DB: airflow` -

```
AIRFLOW__DATABASE__SQL_ALCHEMY_CONN: postgresql+psycopg2://airfl
```

Import Errors -

The error you're encountering is likely because the Python interpreter cannot find the necessary modules for Apache Airflow. This issue can occur when the Airflow environment is not properly configured or when the required packages are not installed.

To resolve this issue, please follow these steps:

1. Make sure you have Apache Airflow installed. If you haven't installed Airflow yet, you can install it using pip:

```
Copy
pip install apache-airflow
```

1. After installing Airflow, you need to initialize the Airflow database by running the following command:

```
Copy
airflow db init
```

1. Next, you need to configure Airflow to use the PostgreSQL provider. Open the `airflow.cfg` file (typically located in `~/airflow/airflow.cfg` on Unix-based

systems or `%AIRFLOW_HOME%\airflow.cfg` on Windows) and uncomment the following line under the `[core]` section:

```
Copy
load_default_connections = True
```

This will enable Airflow to load the default connections, including the PostgreSQL connection.

1. If the issue persists, try restarting the Airflow webserver and scheduler services:

```
Copy
airflow webserver --reload
airflow scheduler
```

1. If you're still encountering the issue, ensure that the PostgreSQL provider is installed. You can install it using pip:

```
Copy
pip install apache-airflow-providers-postgres
```

After following these steps, your DAG file should be able to import the necessary Airflow modules and operators without any issues.

Note: If you're running Airflow in a virtual environment or a container, make sure you've activated the virtual environment or installed the required packages in the container before running the commands mentioned above.

Airflow learn -

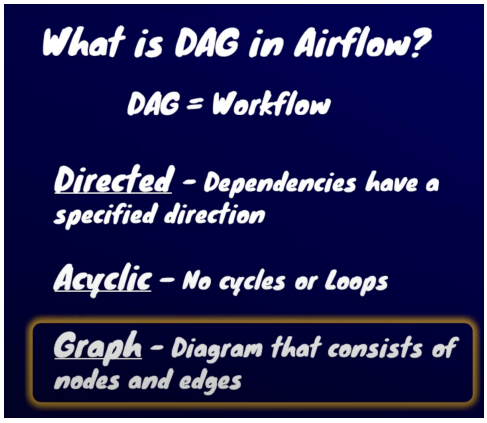
How to run things -

- For DAGs to run we need to start airflow scheduler
- To run locally - no need of redis,

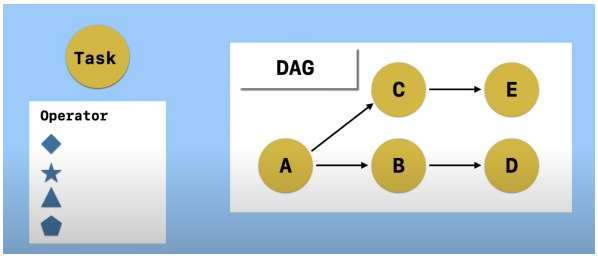
```
  AIRFLOW__CELERY__RESULT_BACKEND: db+postgresql://airflow:airfl
    AIRFLOW__CELERY__BROKER_URL: redis://:@redis:6379/0
```

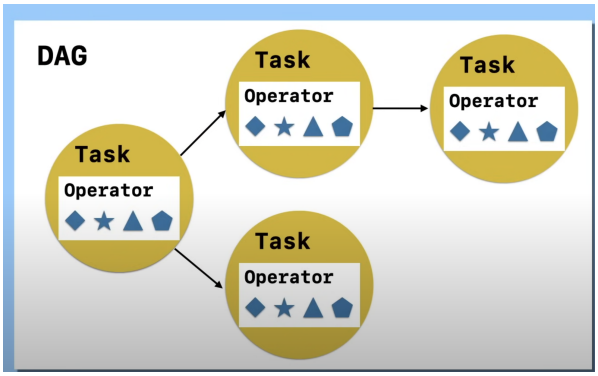
- these lines are also removed for local
- no need of celeryworker and flower
- These things run after a docker compose up
  - airflow_webserver
  - airflow_scheduler
  - airflow_Executor
  - airflow_worker
  - postgres

What is airflow ?

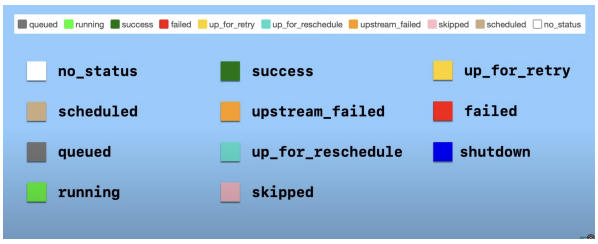


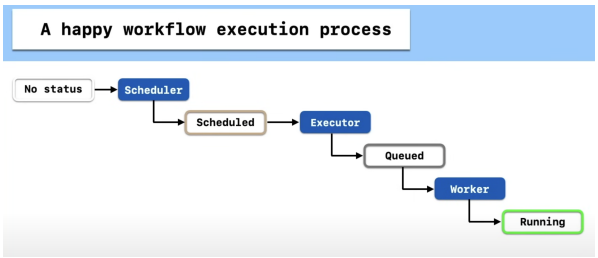
- used to manage workflow
- sequence of task
- DAG - a sequence of task a,b,c,d and e (Directed Async Cycle)
- Task - unit of work
- method used to achieve a task = method





- IMP - nodes represent tasks and edges represent dependencies





Running DAGs ?

```
AIRFLOW__CORE__LOAD_EXAMPLES: 'true'
```

to stop exmaple dags from loading turn this to false

> To run airflow locally and to actually import it we use virtual enivroments

> To create a virtual enviorment in python do this

```
Creating a virtual environment on macOS is very similar to the g

1. Open Terminal.
```

````
2. Navigate to your project directory:
   ```
   cd /path/to/your/project
   ```

3. Create the virtual environment:
   ```
   python3 -m venv myenv
   ```
   Replace `myenv` with your preferred environment name.

4. Activate the virtual environment:
   ```
   source myenv/bin/activate
   ```

5. You'll see `(myenv)` at the beginning of your prompt, indicat

6. To deactivate when you're done:
   ```
   deactivate
   ```

Some macOS-specific notes:

- Macs typically come with Python 2 pre-installed. For Python 3,
- If you have multiple Python versions, you can specify which to
  ```
  python3.9 -m venv myenv
  ```
- If you're using Homebrew's Python, the command might be:
  ```
  python3 -m venv myenv
  ```
````

```
Remember, once activated, any Python packages you install using

Would you like to know how to install packages in your new virtu
```

Different Operators -

1. Bash Operator -

- To run any bask commands we use this to define this -

```
from airflow.operators.bash import BashOperator

with DAG(
    dag_id='our_first_dag_v16',
    default_args=default_args,
    description='This is our first dag that we write',
    start_date=datetime(2021, 1, 1),
    schedule_interval='@daily'
) as dag:
    task1 = BashOperator(
        task_id='first_task',
        bash_command="echo hello world, this is the first task!'
    )
    task2 = BashOperator(
        task_id="second_task",
        bash_command="echo second task!!!!!"
    )
    task3 = BashOperator(
        task_id="third_task",
        bash_command="echo third task!!!!!"
    )


    task1 >> [task2, task3]  # this line means that task1 ka dow
```

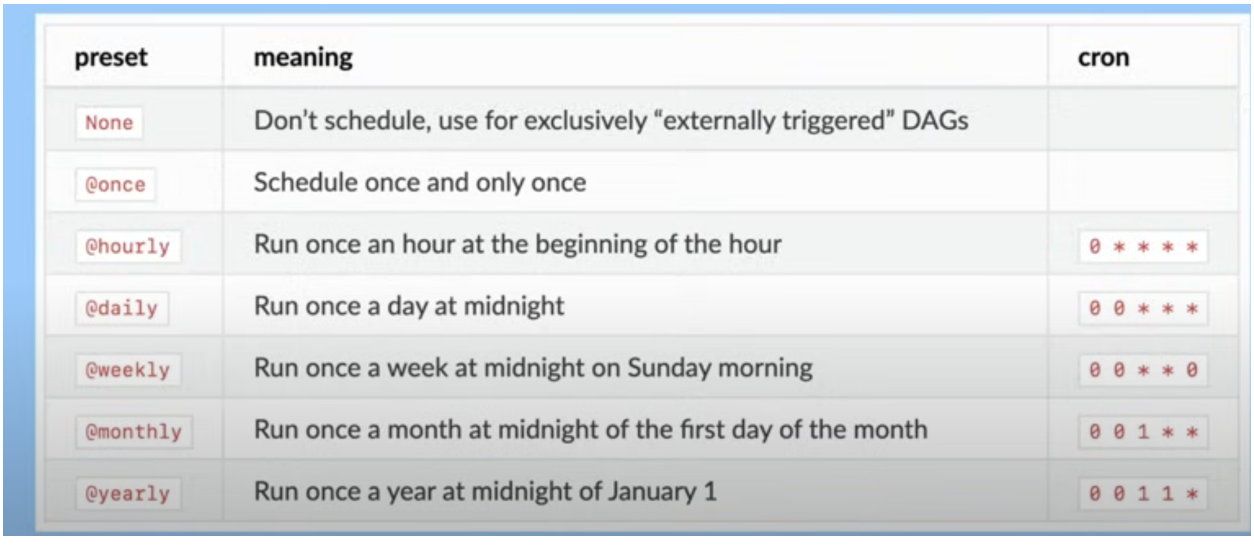
```
                                                                    # i.e t
```

2. Python Operators -

```
# the python_callable or bask_command are keywords of python and
from airflow import DAG
from datetime import datetime, timedelta
from airflow.operators.python import PythonOperator


default_args = {
    "owner": "mehul",
    "retries": "5",
    "retry_delays": timedelta(minutes=5)  #this line sets 2 minu
}

def greet():
    print("Good morning to yaa")


with DAG(
    dag_id='create_dag_with_python_operator_v1',
    default_args=default_args,
    description="This is dags uses python operators",
    start_date=datetime(2024, 6, 17),
    schedule_interval='@daily'
) as dag:
    task1 = PythonOperator(
        task_id="greet",
        python_callable=greet
    )
    task1
```

```
#this lets us to pass in inputs to the function using op_kwags
def greet(name,age):
    print("Good morning to yaa")
    print("my name is ",name," and my age is ",age)

with DAG(
    dag_id='create_dag_with_python_operator_v2',
    default_args=default_args,
    description="This is dags uses python operators",
    start_date=datetime(2024, 6, 17),
    schedule_interval='@daily'
) as dag:
    task1 = PythonOperator(
        task_id="greet",
        python_callable=greet,
        op_kwargs={"name":"mehul", "age":19}
    )
    task1
```

Cron expressions -



preset	meaning	cron
None	Don't schedule, use for exclusively "externally triggered" DAGs	
@once	Schedule once and only once	
@hourly	Run once an hour at the beginning of the hour	0 * * * *
@daily	Run once a day at midnight	0 0 * * *
@weekly	Run once a week at midnight on Sunday morning	0 0 * * 0
@monthly	Run once a month at midnight of the first day of the month	0 0 1 * *
@yearly	Run once a year at midnight of January 1	0 0 1 1 *

Crontab.guru - The cron schedule expression generator

An easy to use editor for crontab schedules.

https://crontab.guru/#0_0_*_*_*