DAGS:

SimpleDAG
TriggerDAG
BackfillDAG
DynamicDAG
Data Quality Checks DAG:
ParallelDAG
GroupDAG

Micro Pipelines
Dataset
Executer
Celery
Flower
Queue
Grouping
Xcom
Branching
Trigger

Micro Pipelines

Trigger Based on Data
Dataset

Define a dataset:

From airflow import Dataset
myfile = Dataset(“/tmp/file.txt”)

Scheduling based in Dataset

@task(outlets= [myfile])

Create a File Using One dag
Read Usiing Another dag : Schedule the read with dataset Outlet

Schedule = [myfile]

Executer

Local Executer : Multiple Tasks in single Machine
Sequential Executor : One Task at a time in single Machine
Celery Executer : Multiple Machines on Celery Cluster
Kubernetes Executor : Multiple Clusters at Multiple Machines

Celery Understanding:
Web
Scheduler -

Queue- Worker

Monitoring using Flower :

docker-compose down

docker-compose - -profile flower up -d

Localhost:5555

Queue:

Task - Worker Distribution

Duplicate airflow-worker

Command : celery worker -q high_cpu

Define Task high_cpu:

queue:'high_cpu'

Grouping:

```
from airflow.utils.task_group import TaskGroup
  with(TaskGroup("group", tooltip="Tasks")) as group:
```

Return group

XCOM:

Sqllite : 2GB
Mysql : 64 KB

Ti.xcom_push(key, value)
Ti.xcom_pull(key, task_id)

Trigger Rules:

All_success,

all_failed,

all_done,

one_failed,

one_success,

none_failed,

none_failed_min_one_success

Branching:

BranchPythonOperator

Choose Your next Task

```
from airflow import DAG, Dataset
from airflow.decorators import task
from datetime import datetime

my_file = Dataset("/tmp/my_file.txt")

with DAG(
   dag_id="source",
   schedule ="@daily",
   start_date=datetime()(2022, 1, 1),
   catchup=False
):
   @task(outlets=[my_file])
   def update_dataset():
       with open(my_file.url,"a+") as f:
           print(f.write("Source"))
   update_dataset()
```

```
from airflow import DAG, Dataset
from airflow.decorators import task
from datetime import datetime
```

```
my_file = Dataset("/tmp/my_file.txt")

with DAG(
   dag_id="dest",
   schedule =[my_file],
   start_date=datetime()(2022, 1, 1),
   catchup=False
):
   @task
   def read_dataset():
       with open(my_file.uri,"r") as f:
           print(f.read())

   read_dataset()
```

Dynamic Dag:

```
from airflow import DAG
from datetime import datetime
from airflow.operators.python_operator import PythonOperator

def generate_dag():
  dag = DAG(
     'dynamic_dag',
     description='A dynamic DAG',
     schedule_interval='@daily',
     start_date=datetime(2023, 3, 28)
  )

  for i in range(1, 4):
     task = PythonOperator(
        task_id='task_{}'.format(i),
        python_callable=lambda: print('Task {}'.format(i)),
        dag=dag
     )

  return dag

dynamic_dag = generate_dag()
```

Prll Dags:

```
from airflow import DAG
from datetime import datetime
from airflow.operators.bash_operator import BashOperator

dag = DAG(
    'parallel_dag',
    description='A parallel DAG',
    start_date=datetime(2023, 3, 28),
    schedule_interval=None
)

task1 = BashOperator(
    task_id='task1',
    bash_command='echo "Task 1"',
    dag=dag
)

task2 = BashOperator(
    task_id='task2',
    bash_command='echo "Task 2"',
    dag=dag
)

task3 = BashOperator(
    task_id='task3',
    bash_command='echo "Task 3"',
    dag=dag
)

task4 = BashOperator(
    task_id='task4',
    bash_command='echo "Task 4"',
    dag=dag
)

task1 >> [task2, task3] >> task4
```

Check FIle Architecture:

```
from airflow import DAG
from airflow.operators.bash import BashOperator

from datetime import datetime

with DAG('group_dag', start_date=datetime(2022, 1, 1),
    schedule_interval='@daily', catchup=False) as dag:

    download_a = BashOperator(
        task_id='download_a',
        bash_command='sleep 10'
    )

    download_b = BashOperator(
        task_id='download_b',
        bash_command='sleep 10'
    )

    download_c = BashOperator(
        task_id='download_c',
        bash_command='sleep 10'
    )

    check_files = BashOperator(
        task_id='check_files',
        bash_command='sleep 10'
    )

    transform_a = BashOperator(
        task_id='transform_a',
        bash_command='sleep 10'
    )

    transform_b = BashOperator(
        task_id='transform_b',
        bash_command='sleep 10'
    )

    transform_c = BashOperator(
        task_id='transform_c',
        bash_command='sleep 10'
    )

    [download_a, download_b, download_c] >> check_files >> [transform_a, transform_b,
transform_c]
```