**Install Airflow on Linux/Ubantu:**

Apache Airflow is open-source platform for developing, scheduling and monitoring batch-oriented workflows. It is built on Python framework, so setting up Airflow needs Python as prerequisite.

**Install Python:**

* Install pip for version 3 of Python.
  + sudo apt install python3-pip
* Check list of packages
  + pip3 list
* Optionally, we can install Airflow on Python Virtual environment.
  + sudo apt install python3.9-venv
  + create folder for virtual env
    - python3 -m venv venv
  + To activate virtual env
    - source venv/bin/activate OR . ./venv/bin/activate

**Install Airflow:**

* Install Airflow using pip
  + sudo pip3 install apache-airflow
  + **Note**:
    - To install specific version of Airflow, we can use following command.
      * pip install apache-airflow==2.6.1 cryptography mysql-connector-python
    - If you run into issue with compatibility with specific packages, we reinstall them with specific compatible versions.
      * Ex: pip install pydantic==1.10.0, pip install apache-airflow-providers-amazon==3.4.0
* Backend of Airflow needs a database to talk to. We execute the below command to setup sqlite database (Default database for Airflow)
  + airflow db init

**Note**:

* sqlite db is useful for a Test and Dev environments but not for Prod as it doesn’t support multiple connections which restricts the Airflow to execute multiple tasks in parallel.
* In contrast, databases like PostgreSQL and MySQL support multiple connections, allowing multiple workers to interact with the database concurrently.
* Setup for MySQL DB can be found here.
* Pre-requisites for setting up MySQL as the backend database for Airflow.
  + A package that provides necessary files for compiling and linking against MySQL client Library.
    - sudo apt-get install libmysqlclient-dev
  + The libraries that allows Python applications to connect to MySQL database.
    - pip install mysql-connector-python
    - pip install mysqlclient
* Configure Airflow webserver with default port 8080
  + airflow webserver -p 8080

**About Webserver**: Itis the User Interface (UI) of Airflow, which can be used to get an overview of the overall health of various Directed Acyclic Graphs (DAG) as well as assist in visualizing different components and states of each DAG. The Airflow Webserver responds to HTTP requests and allows users to interact with it. The Airflow WebServer also allows you to manage users, roles, and various configurations for the Airflow setup.

* The default port is 8080 for airflow webserver, but this can be changed as needed in airflow.cfg.
* Airflow webserver can be accessed using the <http://servername:8080/home>
* Ex: <http://bi-airflow01:8080/home>
* When the webserver is up, it shows the default DAGBAG where it looks for the dags
* The default dagbag path is --> /root/airflow/dags (or /home/test/airflosdags)
* This can be updated in config file of airflow --> airflow.cfg (dags\_folder = /root/airflow/dags)
* To connect to webserver as a Deamon to run in the backend.
  + airflow webserver -D

**About Scheduler**: The Airflow scheduler monitors all tasks and DAGs, then triggers the task instances once their dependencies are complete. Behind the scenes, the scheduler spins up a subprocess, which monitors and stays in sync with all DAGs in the specified DAG directory. Once per minute, by default, the scheduler collects DAG parsing results and checks whether any active tasks can be triggered.

* To connect to scheduler as a Deamon to run in the backend.
  + airflow scheduler -D

**Create Airflow Admin User:**

* Navigate to the ‘airflow’ folder created on the server
* Execute below command with your email and user name
  + airflow users create -u admin -f bala -l vakkantham -r Admin -e [bala.vakkantham@intapp.com](mailto:bala.vakkantham@intapp.com)
  + Enter a password
  + This creates the user ‘admin’ with the password defined for UI interaction.

**MySQL Database setup on Ubuntu**:

* Install My SQL server
  + sudo apt install mysql-server
* Check the status if the MySQL is installed.
  + sudo service mysql status
* You will see the service would be running already.
  + To stop the service: sudo service mysql stop
  + To start the service: sudo service mysql start
  + To restart the service: sudo service mysql restart
  + To check the version of mysql installed: mysql –version

**Note**: With mysql 5.7 or more, we can login to mysql with root without pwd

* Setup Login for MySQL
  + sudo mysql
  + sudo mysql\_secure\_installation
  + ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql\_native\_password BY '\*\*\*\*\*\*\*\*';
  + Ex: If your server is ubuntu, the default user is 'ubuntu'@'localhost'
  + Exit using the ‘exit’ command.
  + We can reset the password using the command ‘sudo mysql\_secure\_installation’
* Login to MySQL to create **new** user that can be configured for Airflow.
  + - mysql -u root -p
    - CREATE USER 'mysqladmin'@'localhost' IDENTIFIED BY '\*\*\*\*\*\*\*\*';
  + Grant all the necessary privileges for this user to interact with MySQL DB.

Note: ‘\*’ grants all the privileges including Create, Drop, Insert, Delete, Update.

* + - GRANT ALL PRIVILEGES ON \*.\* to 'mysqladmin'@'localhost' WITH GRANT OPTION;
  + To free up any memory that the server cached because of the preceding CREATE USER and GRANT statements
    - FLUSH PRIVILEGES;
  + Create a database called ‘airflow’ (can be named anything)
    - CREATE DATABASE airflow;
    - GRANT ALL PRIVILEGES ON airflow.\* to 'mysqladmin'@'localhost';
  + Now exit MySQL using ‘exit’

**Connect Airflow with MySQL**:

* Navigate to airflow folder and edit ‘airflow.cfg’ file to change ‘sql\_alchemy\_conn’ property.
  + sql\_alchemy\_conn=mysql+mysqlconnector://mysqladmin:\*\*\*\*\*\*@localhost:3306/airflow
* Reexecute airflow db init to make the changes reflect
  + airflow db init

**Troubleshooting with Airflow**:

* To check what jobs are running on Airflow server
  + htop
* Check the active list of Airflow jobs running
  + ps aux | grep airflow
* Check the server for issues if you see UI is down
  + Check the disk space (command: df)
  + Check memory (command: free)
  + Clean up logs in airflow folder if needed in airflow/logs.
  + Check free space on server df -h
* To see if we are able to connect to lets say ec2 server from Airflow:
  + ping ip
  + Telnet ip ssh (telnet 10.127.8.222 ssh)

**Important Configurations:**

Configurations reside in a config file called Airflow.cfg in root/airflow folder:

* The executor class that airflow should use. Choices include
  + ``SequentialExecutor``, ``LocalExecutor``, ``CeleryExecutor``, ``DaskExecutor``,

`KubernetesExecutor``, ``CeleryKubernetesExecutor``

* + Set LocalExecutor if we donot have Kubernetes setup for Prod environment
    - **executor = LocalExecutor**
* The folder where your airflow pipelines live, most likely a subfolder in a code repository. This path must be absolute.
  + **dags\_folder = /root/airflow/dags**
* The SqlAlchemy connection string to the metadata database. SqlAlchemy supports many different database engines. More information here: <http://airflow.apache.org/docs/apache-airflow/stable/howto/set-up-database.html#database-uri>
  + sql\_alchemy\_conn = sqlite:////root/airflow/airflow.db
  + Note: This will be replaced with the MySQL or postgres connections we need to configure as a backend
* The encoding for the databases
  + **sql\_engine\_encoding = utf-8**
* If SqlAlchemy should pool database connections.
  + **sql\_alchemy\_pool\_enabled = True**
* The SqlAlchemy pool size is the maximum number of database connections in the pool. 0 indicates no limit.
  + **Ex: sql\_alchemy\_pool\_size = 5**
* The SqlAlchemy pool recycle is the number of seconds a connection can be idle in the pool before it is invalidated. This config does not apply to sqlite. If the number of DB connections is ever exceeded, a lower config value will allow the system to recover faster.
  + **sql\_alchemy\_pool\_recycle = 1800**
* The maximum number of active DAG runs per DAG. The scheduler will not create more DAG runs if it reaches the limit. This is configurable at the DAG level with ``max\_active\_runs``, which is defaulted as ``max\_active\_runs\_per\_dag``.
  + **max\_active\_runs\_per\_dag = 1**
* Whether to load the default connections that ship with Airflow. It's good to get started, but you probably want to set this to ``False`` in a production environment
  + **load\_default\_connections = True**
* Path to the folder containing Airflow plugins
  + **plugins\_folder = /root/airflow/plugins**
* The maximum number of task instances allowed to run concurrently in each DAG. To calculate the number of tasks that is running concurrently for a DAG, add up the number of running tasks for all DAG runs of the DAG. This is configurable at the DAG level with ``max\_active\_tasks``, which is defaulted as ``max\_active\_tasks\_per\_dag``.
* An example scenario when this would be useful is when you want to stop a new dag with an early start date from stealing all the executor slots in a cluster.
  + **max\_active\_tasks\_per\_dag = 16**
* This defines the maximum number of task instances that can run concurrently in Airflow regardless of scheduler count and worker count. Generally, this value is reflective of the number of task instances with the running state in the metadata database.
  + **parallelism = 16**
* Path to custom XCom class that will be used to store and resolve operators results.
  + Example: xcom\_backend = path.to.CustomXCom

**xcom\_backend = airflow.models.xcom.BaseXCom**

**[logging]**

* The folder where airflow should store its log files. This path must be absolute.
* There are a few existing configurations that assume this is set to the default.
* If you choose to override this you may need to update the dag\_processor\_manager\_log\_location and dag\_processor\_manager\_log\_location settings as well.
  + **base\_log\_folder = /root/airflow/logs**
* Airflow can store logs remotely in AWS S3, Google Cloud Storage or Elastic Search. Set this to True if you want to enable remote logging.
  + **remote\_logging = False**
* Users must supply an Airflow connection id that provides access to the storage location.
  + **remote\_log\_conn\_id = aws\_default**
* Storage bucket URL for remote logging
  + S3 buckets should start with "s3://"
  + Cloudwatch log groups should start with "cloudwatch://"
  + GCS buckets should start with "gs://"
  + WASB buckets should start with "wasb" just to help Airflow select correct handler
  + Stackdriver logs should start with "stackdriver://"
    - **remote\_base\_log\_folder = "s3://intapp-snowflake/aiflow\_logs/"**
* Use server-side encryption for logs stored in S3
  + **encrypt\_s3\_logs = False**
* Logging level can be defined as ``CRITICAL``, ``ERROR``, ``WARNING``, ``INFO``, ``DEBUG``.
  + **logging\_level = INFO**
* Formatting for how airflow generates file names/paths for each task run.
  + **log\_filename\_template = {{ ti.dag\_id }}/{{ ti.task\_id }}/{{ ts }}/{{ try\_number }}.log**
* Formatting for how airflow generates file names for log
  + **log\_processor\_filename\_template = {{ filename }}.log**
* When you start an airflow worker, airflow starts a tiny web server subprocess to serve the workers local log files to the airflow main web server, who then builds pages and sends them to users. This defines the port on which the logs are served. It needs to be unused, and open visible from the main web server to connect into the workers.
  + **worker\_log\_server\_port = 8793**
* Default timezone to display all dates in the UI, can be UTC, system, or any IANA timezone string (e.g. Europe/Amsterdam). If left empty the default value of core/default\_timezone will be used
  + Example: default\_ui\_timezone = America/New\_York
  + **default\_ui\_timezone = UTC**
* The ip specified when starting the web server
  + **web\_server\_host = 0.0.0.0**
* The port on which to run the web server
  + **web\_server\_port = 8080**
* Paths to the SSL certificate and key for the web server. When both are provided SSL will be enabled. This does not change the web server port.
  + **web\_server\_ssl\_cert =**
* Paths to the SSL certificate and key for the web server. When both are provided SSL will be enabled. This does not change the web server port.
  + **web\_server\_ssl\_key =**
* The type of backend used to store web session data, can be 'database' or 'securecookie' Example: session\_backend = securecookie
  + **session\_backend = database**
* Expose hostname in the web server
  + **expose\_hostname = True**
* Expose stacktrace in the web server
  + **expose\_stacktrace = True**
* Default DAG view. Valid values are: ``tree``, ``graph``, ``duration``, ``gantt``, ``landing\_times``
  + **dag\_default\_view = tree**
* Default DAG orientation. Valid values are: ``LR`` (Left->Right), ``TB`` (Top->Bottom), ``RL`` (Right->Left), ``BT`` (Bottom->Top)
  + **dag\_orientation = LR**

**[smtp]**

* If you want airflow to send emails on retries, failure, and you want to use the airflow.utils.email.send\_email\_smtp function, you have to configure an smtp server here.
  + **smtp\_host = 10.147.1.1**
  + **smtp\_starttls = True**
  + **smtp\_ssl = False**
  + Example: smtp\_user = airflow
    - **smtp\_user =** [**data\_business\_intelligence@intapp.com**](mailto:data_business_intelligence@intapp.com)
    - smtp\_password =
    - **smtp\_port = 25**
    - **smtp\_mail\_from =** [**data\_business\_intelligence@intapp.com**](mailto:data_business_intelligence@intapp.com)
    - **smtp\_timeout = 30**

**[triggerer]**

* How many triggers a single Triggerer will run at once, by default.
  + default\_capacity = 1000

**DAG Examples**:

1. To run one process when upstream succeeds and other when upstream fails.

A screenshot of a computer

Description automatically generated with medium confidence

Code for above DAG:

**import** **airflow**

**from** **airflow.models** **import** DAG

**from** **airflow.operators.bash\_operator** **import** BashOperator

**from** **airflow.utils.dates** **import** days\_ago

**from** **datetime** **import** datetime, timedelta

**from** **airflow.contrib.operators.ssh\_operator** **import** SSHOperator

**from** **airflow.operators.python\_operator** **import** PythonOperator

args = {

"owner": "airflow",

"provide\_context": **True**,

'email': ['balasubramanyam.vakkanthamramamurthy@intapp.com'],

'email\_on\_failure': **True**,

'email\_on\_retry': **False**,

'retries': 1,

'retry\_delay': timedelta(seconds=30)

}

**def** print\_xcom(dag\_run):

ti = dag\_run.get\_task\_instance("wait\_for\_flag")

print("printing the flag:", ti.xcom\_pull(task\_ids="wait\_for\_flag"))

**with** DAG(dag\_id="qa\_op\_test\_hist\_api\_flagcheck", schedule\_interval='50 \* \* \* \*', start\_date=days\_ago(1), default\_args=args, catchup=**False**, tags=["QA|OP"]) **as** dag:

start = BashOperator(

task\_id="start",

bash\_command='exit 0'

)

check\_for\_fullload = SSHOperator(

task\_id="check\_for\_fullload",

ssh\_conn\_id='aws\_etl\_server',

command='`/usr/bin/python3.8 /home/ec2-user/deploy/oneplace\_dev/pyscripts/OP\_Incremental\_Check.py >> /home/ec2-user/deploy/oneplace\_dev/log/op\_hourly\_load.log`',

)

HistTest = SSHOperator(

task\_id="HistTest",

ssh\_conn\_id='aws\_etl\_server',

command='`/usr/bin/python3.8 /home/ec2-user/deploy/oneplace\_dev/pyscripts/OP\_Hist\_Test.py >> /home/ec2-user/deploy/oneplace\_dev/log/op\_hourly\_load.log`',

trigger\_rule = 'all\_success'

)

RunWhenFailed = SSHOperator(

task\_id="RunWhenFailed",

ssh\_conn\_id='aws\_etl\_server',

command='`/usr/bin/python3.8 /home/ec2-user/deploy/oneplace\_dev/pyscripts/OP\_Hist\_Test.py >> /home/ec2-user/deploy/oneplace\_dev/log/op\_hourly\_load.log`',

trigger\_rule = 'one\_failed'

)

check\_for\_fullload.set\_upstream(start)

HistTest.set\_upstream(check\_for\_fullload)

RunWhenFailed.set\_upstream(check\_for\_fullload)

1. DAG with dependencies along with Task Group to run a set in parallel.

A screenshot of a computer

Description automatically generated with medium confidence

**import** **airflow**

**from** **airflow.models** **import** DAG

**from** **airflow.operators.bash\_operator** **import** BashOperator

**from** **airflow.utils.dates** **import** days\_ago

**from** **datetime** **import** datetime, timedelta

**from** **airflow.contrib.operators.ssh\_operator** **import** SSHOperator

**from** **airflow.utils.task\_group** **import** TaskGroup

**from** **airflow.providers.amazon.aws.operators.redshift** **import** RedshiftSQLOperator

args = {

"owner": "airflow",

"provide\_context": **True**,

'email': ['jorge.herrera@intapp.com','balasubramanyam.vakkanthamramamurthy@intapp.com','andrei.vasiliuk@intapp.com'],

'email\_on\_failure': **True**,

'email\_on\_retry': **False**,

'retries': 1,

'retry\_delay': timedelta(seconds=30)

}

**with** DAG(dag\_id="prod\_30min\_op\_loads", schedule\_interval='28,56 \* \* \* \*', start\_date=datetime(2022, 5, 6, 22, 15), default\_args=args, catchup=**False**, tags=["PROD|OP"]) **as** dag:

start = BashOperator(

task\_id="start",

bash\_command='exit 0'

)

refresh\_view1 = RedshiftSQLOperator(

task\_id='refresh\_materialized\_view\_1',

sql="REFRESH MATERIALIZED VIEW analytics.fact\_op\_forecast\_calls;",

)

refresh\_view2 = RedshiftSQLOperator(

task\_id='refresh\_materialized\_view\_2',

sql="REFRESH MATERIALIZED VIEW analytics.fact\_sales\_production\_forecast;",

)

**with** TaskGroup(group\_id="jobs") **as** jobs:

ProjectComponent = SSHOperator(

task\_id="ProjectComponent",

ssh\_conn\_id='aws\_etl\_server',

command='`python3.8 /home/ec2-user/deploy/oneplace\_prod/pyscripts/OP\_Extract.py ProjectComponent >> /home/ec2-user/deploy/oneplace\_prod/log/op\_30min\_load.log`',

)

ForecastWeekly = SSHOperator(

task\_id="ForecastWeekly",

ssh\_conn\_id='aws\_etl\_server',

command='`sleep 5; python3.8 /home/ec2-user/deploy/oneplace\_prod/pyscripts/OP\_Extract.py ForecastWeekly >> /home/ec2-user/deploy/oneplace\_prod/log/op\_daily\_load.log`',

)

ForecastQuarterly = SSHOperator(

task\_id="ForecastQuarterly",

ssh\_conn\_id='aws\_etl\_server',

command='`sleep 10; python3.8 /home/ec2-user/deploy/oneplace\_prod/pyscripts/OP\_Extract.py ForecastQuarterly >> /home/ec2-user/deploy/oneplace\_prod/log/op\_daily\_load.log`',

)

Inbound = SSHOperator(

task\_id="Inbound",

ssh\_conn\_id='aws\_etl\_server',

command='`sleep 15; python3.8 /home/ec2-user/deploy/oneplace\_prod/pyscripts/OP\_Extract.py Inbound >> /home/ec2-user/deploy/oneplace\_prod/log/op\_daily\_load.log`',

)

start >> jobs >> refresh\_view1 >> refresh\_view2