FOREX DATA PIPELINE USING APACHE AIRFLOW

Naveen Raju Sreerama Raju Govinda Raju
naveenraju100@gmail.com | +1 224 706 7718 | +91 9886157101 |
https://www.linkedin.com/in/naveen-raju-s-g-bb1486124
https://naveenrajusg.github.io/Portfolio/

Table of Contents

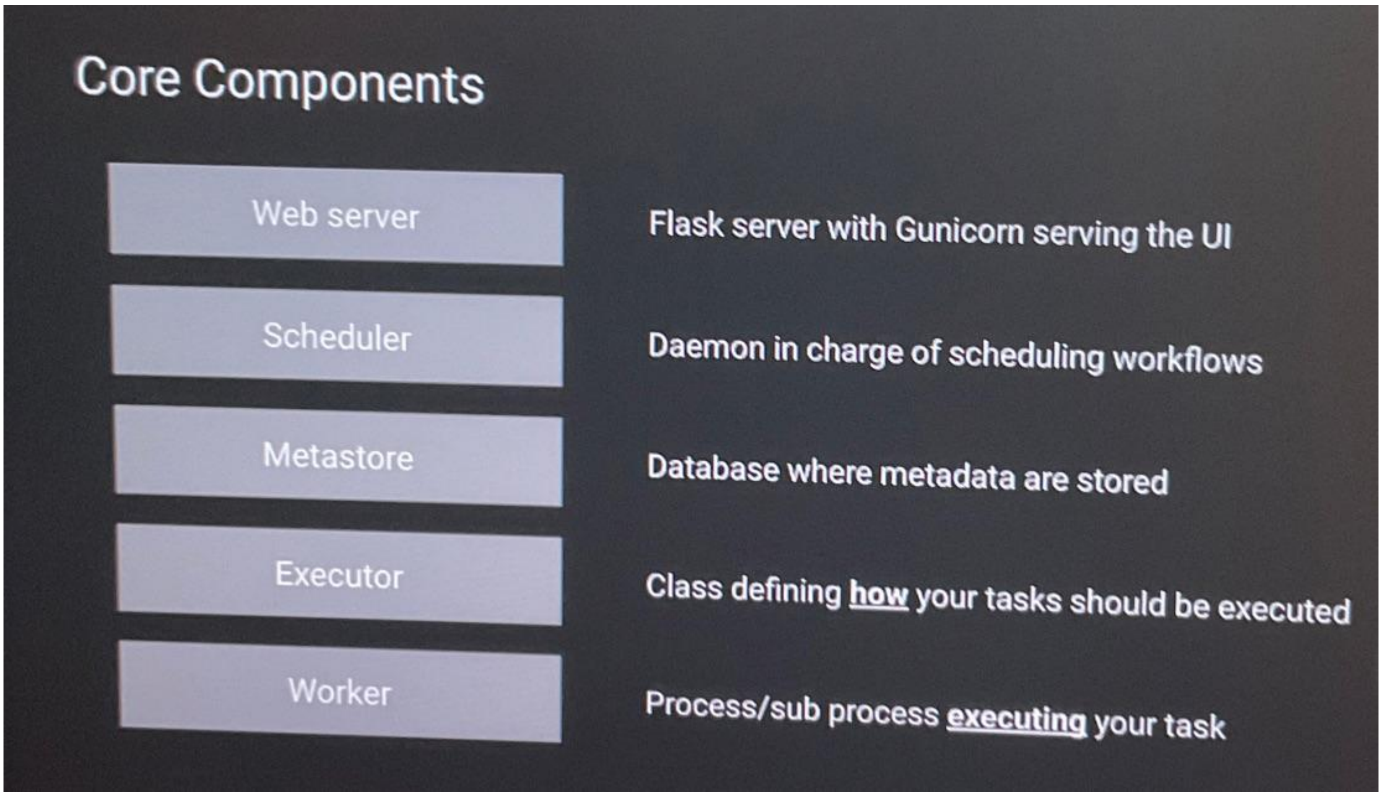
1) **Overview**

Airflow :

Apache Airflow is an open-source platform used for orchestrating complex workflows and data pipelines.

Airflow allows you to define, schedule, and monitor workflows as Directed Acyclic Graphs (DAGs). Each DAG represents a workflow that consists of a set of tasks and their dependencies. Tasks can be anything from simple data processing steps to more complex operations, such as data extraction, transformation, loading, and model training.

Core Components of Airflow :



Core Components

Web server	Flask server with Gunicorn serving the UI
Scheduler	Daemon in charge of scheduling workflows
Metastore	Database where metadata are stored
Executor	Class defining **how** your tasks should be executed
Worker	Process/sub process **executing** your task

Directed Acyclic Graph:

"DAG" stands for "Directed Acyclic Graph." It is a fundamental concept used to represent workflows as a collection of tasks and their dependencies.

Directed: This means that there is a defined direction or flow between tasks.
Acyclic: A graph is acyclic if there are no cycles, which means there are no closed loops of dependencies.

A typical DAG represents a workflow with multiple tasks that need to be executed in a specific order, where the output of one task is often used as input for subsequent tasks. Each node in the DAG represents a task, and the edges between nodes represent the dependencies between tasks.

Operator:

Operator is an object that encapsulates the task, the job that we want to execute. For example, if we want to connect to a database and insert data we use special operator to do that. An object encapsulates the tasks.

Categories of Operators : Action operator, Transfer operator, Sensor operator

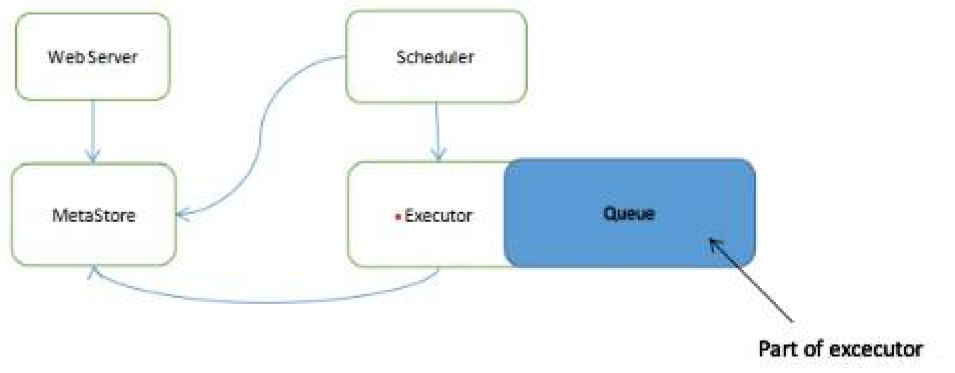
Task Instance:

When operator run on a dag then it is an instance.

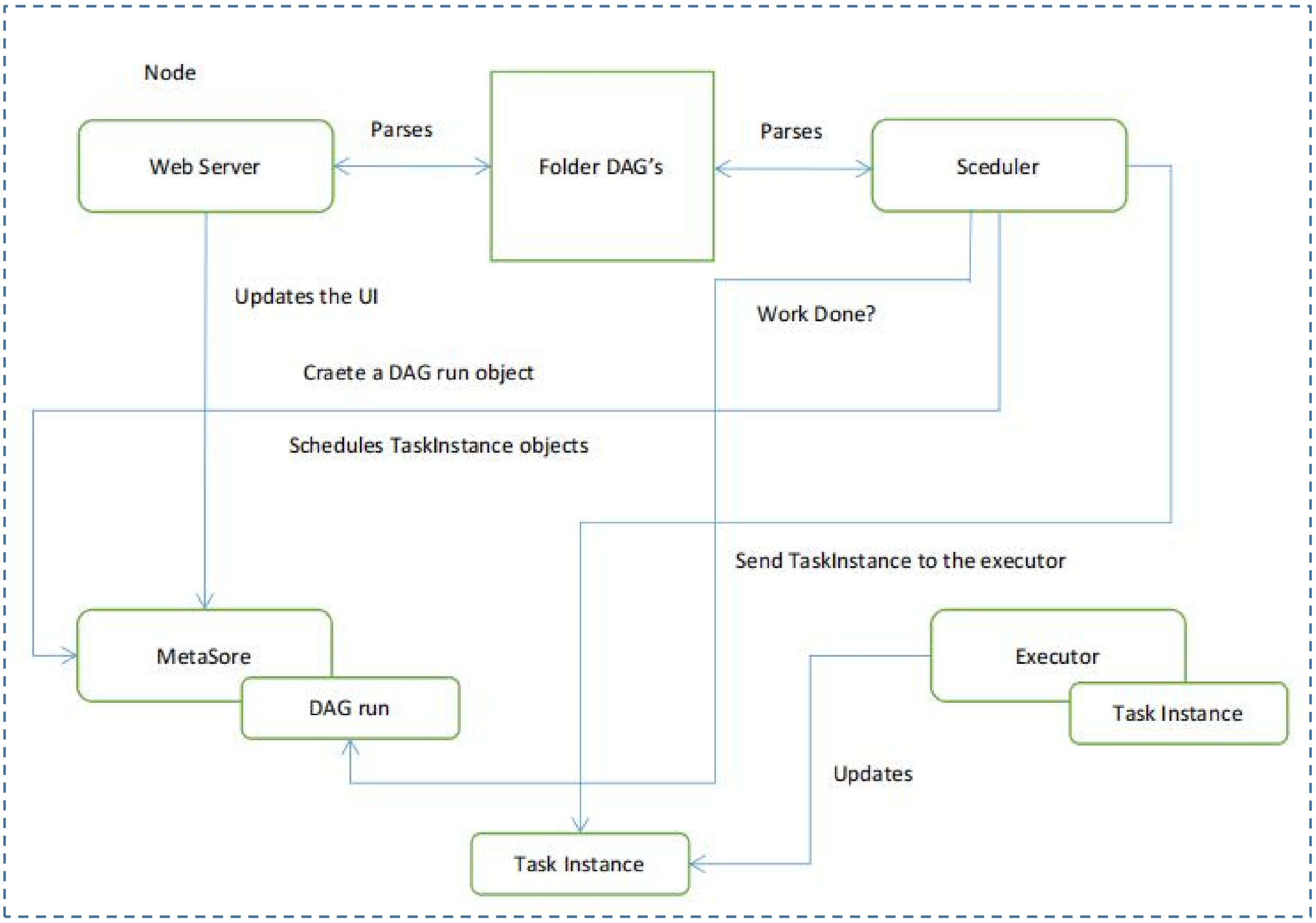
One - node Architecture:

First the Web server fetches some metadata from the meta database in order to display information corresponding to Dag's, your task instances or your users on the user interface. Next, this category interacts with the meta database and the executor in order to trigger your Dag's,in order to trigger your tasks.

Finally, the executor interacts also with the meta in order to update the tasks that just have been completed. One thing to remember is that this scheduler,executor and the Web interact together with help of meta database. The executor has an internal queue and this is part of the executor. If you use the local executor, for example, and this is how your tasks are executed in the right order because there is a queue (Queue can be RabitMQ or Reddis) in the executor, that's why its executes tasks one after the other.By default we get sequential executor. And if we want to start setting up local executor where our tasks are executed in sub processes with both executors there is a queue in it and that's how your tasks are executed in order.



If we need to scale this to multi node then we can go with multi-node Celery executor or Kubernetes executor.



<u>Steps of DAG run in Apache Airflow:</u>

- When new DAG is in DAG's folder both Web Server and Scheduler will parse the DAG.
- Scheduler checks if DAG is ready to be triggered if so DAG run object is created. (DAG run object is nothing but instance of our DAG running at a given time). This DAG run object is stored in MetaStore of Airflow with status running.
- If there is a task ready to be triggered in your DAG in that case scheduler creates TaskInstance objects corresponding to the task with the status scheduled in MetaStore of Airflow.
- Then Scheduler sends TaskInstance object to Executor with status queued.
- Once executor is ready to run the task this time TaskInstance object has status running and now Executor updates status of task in MetaStore.
- As soon as task is completed the Executor updates status of task in MetaStore.
  After this Scheduler verifies if there is no more task to run in DAG, if so DAG run object will now have status complete.
- Lastly Web Server updates UI.

2) <u>Install Docker for Windows</u>

Follow the steps in the following link for installation https://docs.docker.com/desktop/install/windows-install/

3) <u>Install Airflow steps</u>

Installing Airflow
docker run -it --rm -p 8080:8080 python:3.8-slim /bin/bash
* Create and start a docker container from the Docker image python:3.8-slim and execute the command /bin/bash in order to have a shell session

```
python -V
```
* Print the Python version

```
export AIRFLOW_HOME=/usr/local/airflow
```
* Export the environment variable AIRFLOW_HOME used by Airflow to store the dags folder, logs folder and configuration file

```
env | grep airflow
```
* To check that the environment variable has been well exported

```
apt-get update -y && apt-get install -y wget libczmq-dev curl libssl-dev git inetutils-telnet bind9utils
freetds-dev libkrb5-dev libsasl2-dev libffi-dev libpq-dev freetds-bin build-essential default-libmysqlclient-
dev apt-utils rsync zip unzip gcc && apt-get clean
```
* Install all tools and dependencies that can be required by Airflow

```
useradd -ms /bin/bash -d ${AIRFLOW_HOME} airflow
```
* Create the user airflow, set its home directory to the value of AIRFLOW_HOME and log into it

```
cat /etc/passwd | grep airflow
```
* Show the file /etc/passwd to check that the airflow user has been created

```
pip install --upgrade pip
```
* Upgrade pip (already installed since we use the Docker image python 3.5)

```
su - airflow
```
* Log into airflow

```
python -m venv .sandbox
```
* Create the virtual env named sandbox

```
source .sandbox/bin/activate
```
* Activate the virtual environment sandbox

```
wget https://raw.githubusercontent.com/apache/airflow/constraints-2.0.2/constraints-3.8.txt
```
* Download the requirement file to install the right version of Airflow’s dependencies

```
pip install "apache-airflow[crypto,celery,postgres,cncf.kubernetes,docker]"==2.0.2 --
constraint ./constraints-3.8.txt
```

* Install the version 2.0.2 of apache-airflow with all subpackages defined between square brackets. (Notice that you can still add subpackages after all, you will use the same command with different subpackages even if Airflow is already installed)

airflow db init
* Initialise the metadatabase

airflow scheduler &
* Start Airflow's scheduler in background

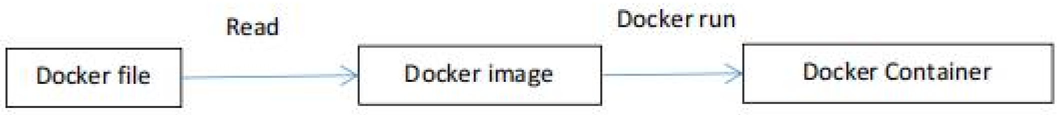
airflow webserver &
* Start Airflow's webserver in background

docker build -t airflow-basic .
* Build a docker image from the Dockerfile in the current directory (airflow-materials/airflow-basic) and name it airflow-basic

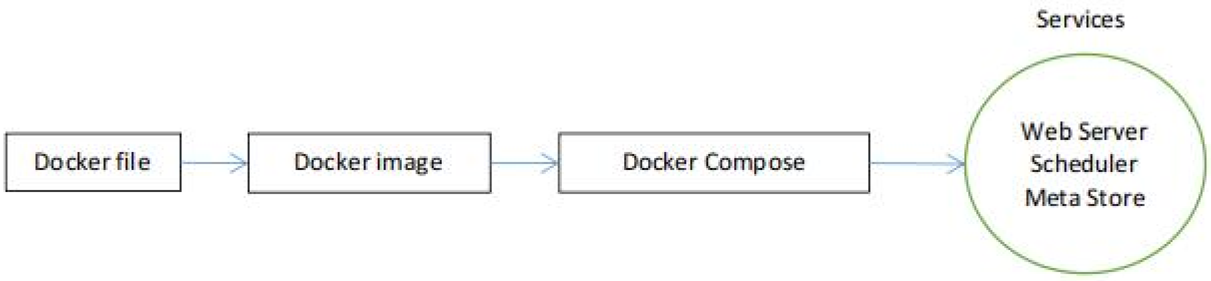
docker image ls
*to list all docker image

docker run --rm -d -p 8080:8080 airflow-basic
*Airflow UI is running inside docker container, thus to access Airflow UI we should connect Docker container port 8080 to our machine port 8080

4) <u>Docker</u>



Docker image is created from Docker file. Docker Image is like an application compiled. Docker file consists of all dependency installation commands.

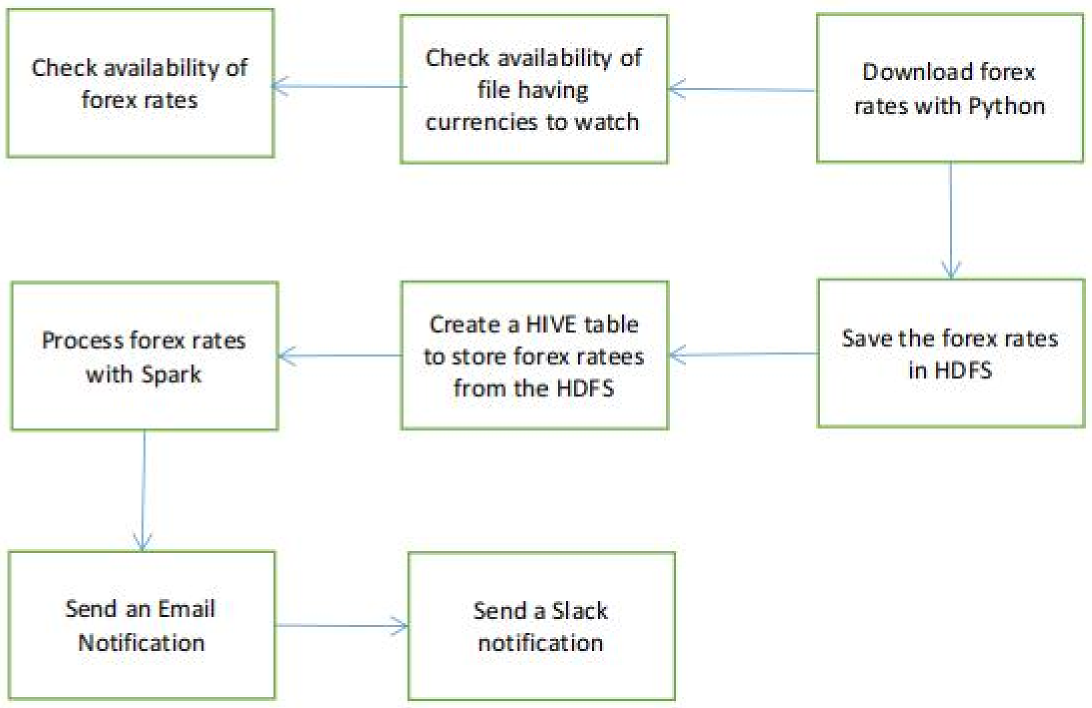


<u>Docker Compose</u> : is a tool provided by Docker that allows to define and manage multi containers. Networks and volumes as a single application, making it easier to orchestrate and deploy complex containerized applications.

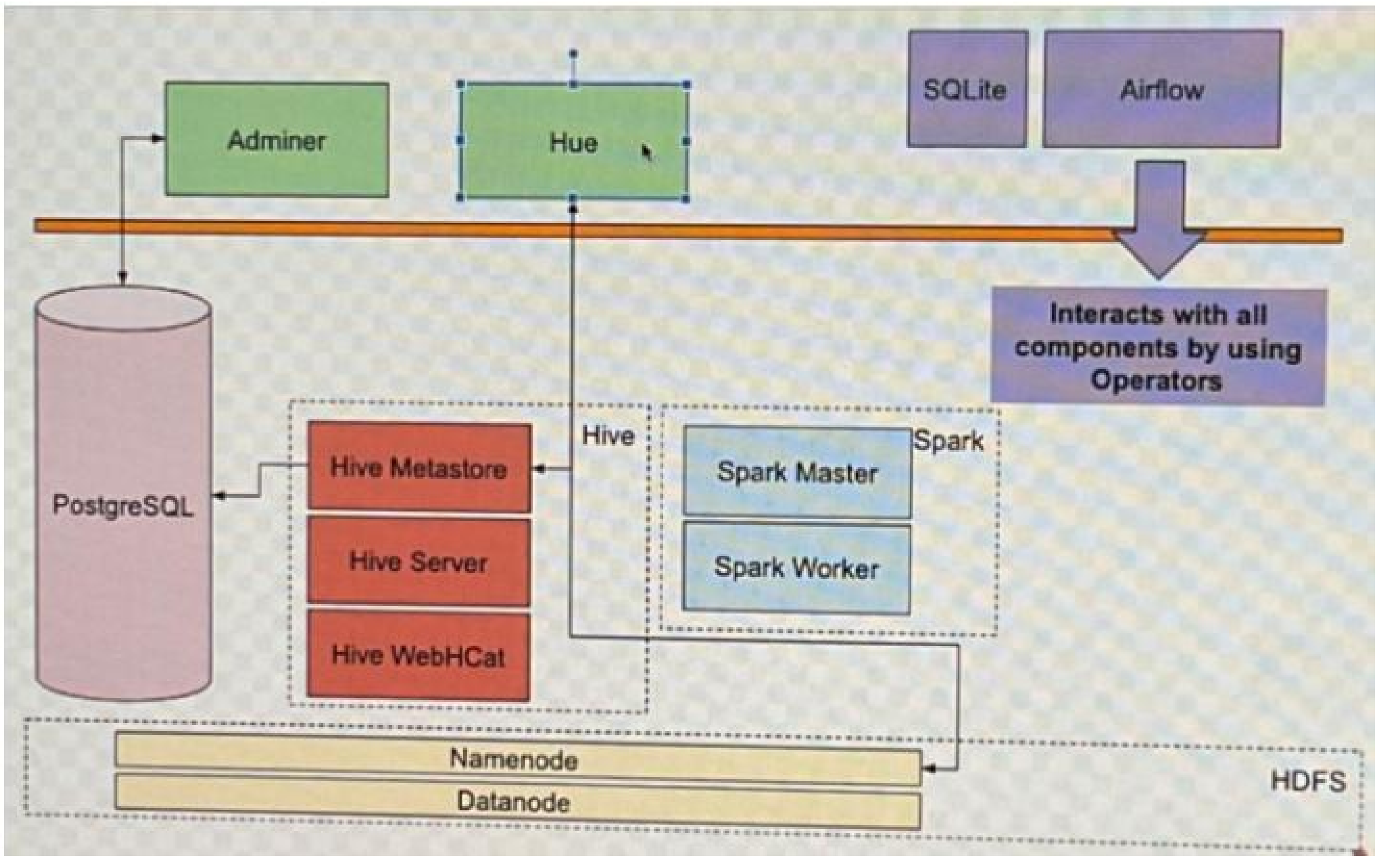
In Airflow we have 3 main components MetaStore, Scheduler and Web server where each run in 3 different container managed by docker compose as single application.

All 3 containers will run inside same network and so each container will able to communicate with the other.

5) Flow chart of Forex Data pipeline


Check availability of forex rates
Check availability of file having currencies to watch
Download forex rates with Python
Save the forex rates in HDFS
Create a HIVE table to store forex ratees from the HDFS
Process forex rates with Spark
Send an Email Notification
Send a Slack notification

6) Architecture back end details



HDFS : This is where Forex data is stored

Admirer : is a too to interact with PostgreSQL

Spark module : This consists of Spark Master and Spark Worker, is used to process Forex data
Hive module : Which consists of Hive Metastore, Hive server and Hive WebHCat, is used to query Forex data using SQL like statements, and also it uses PostgreSQL.

PostgreSQL :

PostgreSQL, often referred to as Postgres, is an open-source relational database management system (RDBMS) known for its robustness, reliability, and extensive feature set. It is a powerful and scalable database system that offers high performance, data integrity, and advanced capabilities for handling complex data.

PostgreSQL supports various operating systems, including Windows, macOS, Linux and BSD, making it a versatile choice for different environments. It adheres to SQL standards and provides support for a wide range of data types, indexing techniques, and advanced features such as triggers, stored procedures, and full-text search

Some notable features and capabilities of PostgreSQL include:

- ACID Compliance: PostgreSQL ensures Atomicity, Consistency, Isolation, and Durability,providing transactional integrity and reliability

- Extensibility: It supports user-defined data types, operators, functions, and extensions,allowing developers to customize and extend the database functionality.

- Concurrency: PostgreSQL employs multi version concurrency control (MVCC), enabling multiple transactions to access the database simultaneously without blocking each other

- Replication and High Availability: It offers various replication options, including and synchronous replication, for data redundancy and high availability.

- Full-Text Search: PostgreSQL provides robust full-text search capabilities, enabling efficient searching and indexing of textual data.

- JSON and NoSQL Support: It includes native support for JSON data, allowing storage, retrieval, and querying of JSON documents. It also provides support for NoSQL-like functionality

- Scalability: It supports scaling horizontally through sharding and can handle large volumes of data and high traffic workloads.

7) <u>Pipeline code:</u>

<u>A) Create a Gist in our Github account</u>

Create a Gist in our Github account with name ‘api_forex_exchange.json’ with 3 Json files in it.

```
api_forex_exchange.json
{
"rates":{"CAD":1.31,"HKD":7.82,"ISK":121.32,"PHP":50.76,"DKK":6.73,"GBP":0.76,"JPY":108.56,"CH
F":0.98,"EUR":0.90,"NZD":1.52,"USD":1.0,"SGD":1.35,"AUD":1.45},
"base":"USD",
"date":"2021-01-01"
}
```

```
api_forex_exchange_eur.json
{
"rates":{"CAD":1.21,"GBP":0.36,"JPY":101.89,"USD":1.13,"NZD":1.41,"EUR":1.0},
"base":"EUR",
"date":"2021-01-01"
}
```

```
api_forex_exchange_usd.json
{
"rates":{"CAD":1.31,"GBP":0.76,"JPY":108.56,"EUR":0.90,"NZD":1.52,"USD":1.0},
"base":"USD",
"date":"2021-01-01"
}
```

B) Define DAG

```
 from airflow import DAG
from datetime import datetime, timedelta

default_args = {
   "owner": "airflow",
   "email_on_failure": False,
   "email_on_retry": False,
   "email": "admin@localhost.com",
   "retries": 1,   # On failure retry for one time
   "retry_delay": timedelta(minutes=5)  # During the retry maximum processing time is 5minutes
}

with DAG("forex_data_pipeline", start_date=datetime(2021, 1 ,1),
  schedule_interval="@daily", default_args=default_args, catchup=False) as dag:



# forex_data_pipeline - is name of DAG
# schedule_interval="@daily" - Execute DAG at midnight (00:00) every day
# catchup = False - Don't run DAG for older dates (I.e dates between current date and start date
mentioned)
```

C) Create a task named "is_forex_rates_available" - HttpSensor to check if the API is available and start a docker container airfow airflow-section-3

Write "is_forex_rates_available" code in our DAG

```
   is_forex_rates_available = HttpSensor(
       task_id="is_forex_rates_available",
       http_conn_id="forex_api",
       endpoint="naveenrajusg/497e10579edfe65fdf1c3d60a387fa20",
       response_check=lambda response: "rates" in response.text,
       poke_interval=5,
       timeout=20
   )
```

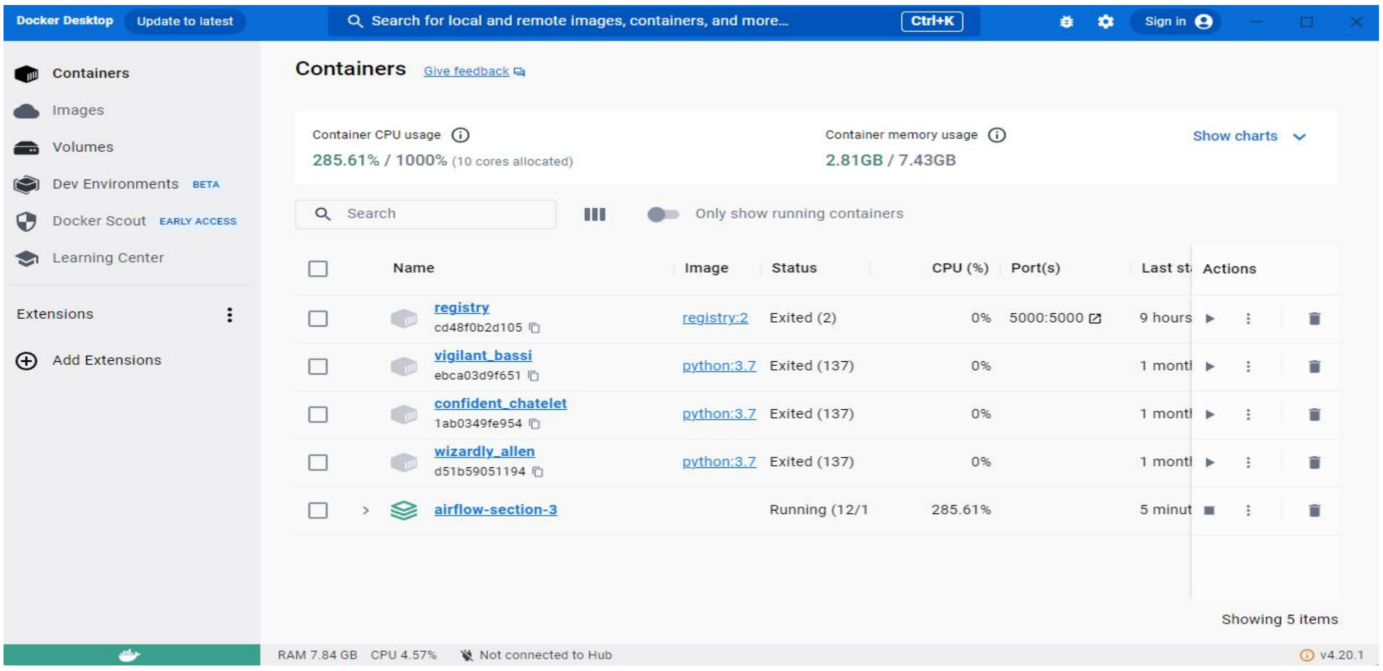
In current working directory of the pipeline code all command prompt code should be run:

```
In cmd:
start.sh
```

start.sh

```
#!/bin/bash

# Build the base images from which are based the Dockerfiles
# then Startup all the containers at once
docker build -t hadoop-base docker/hadoop/hadoop-base && \
docker build -t hive-base docker/hive/hive-base && \
docker build -t spark-base docker/spark/spark-base && \
docker-compose up -d --build




# docker build -t hadoop-base docker/hadoop/hadoop-base: This command builds a Docker image named "hadoop-base" using the Dockerfile located in the "docker/hadoop/hadoop-base" directory. This Docker image likely contains the base Hadoop installation.

# docker build -t hive-base docker/hive/hive-base: This command builds a Docker image named "hive-base" using the Dockerfile located in the "docker/hive/hive-base" directory. This Docker image likely contains the base Hive installation.

# docker build -t spark-base docker/spark/spark-base: This command builds a Docker image named "spark-base" using the Dockerfile located in the "docker/spark/spark-base" directory. This Docker image likely contains the base Spark installation.

# docker-compose up -d --build: This command brings up the Docker containers defined in the Docker Compose configuration file and rebuilds the containers if necessary. The -d flag runs the containers in detached mode, meaning they will run in the background. The --build flag ensures that any changes made to the Docker images are applied before starting the containers.
```

We can see a Docker container named airflow-section-3 is created



Containers

Container CPU usage 285.61% / 1000% (10 cores allocated)

Container memory usage 2.81GB / 7.43GB

Name	Image	Status	CPU (%)	Port(s)	Last st
registry cd48f0b2d105	registry:2	Exited (2)	0%	5000:5000	9 hours
vigilant_bassi ebca03d9f651	python:3.7	Exited (137)	0%		1 montl
confident_chatelet 1ab0349fe954	python:3.7	Exited (137)	0%		1 montl
wizardly_allen d51b59051194	python:3.7	Exited (137)	0%		1 montl
airflow-section-3		Running (12/1	285.61%		5 minut

Showing 5 items

RAM 7.84 GB CPU 4.57% Not connected to Hub v4.20.1

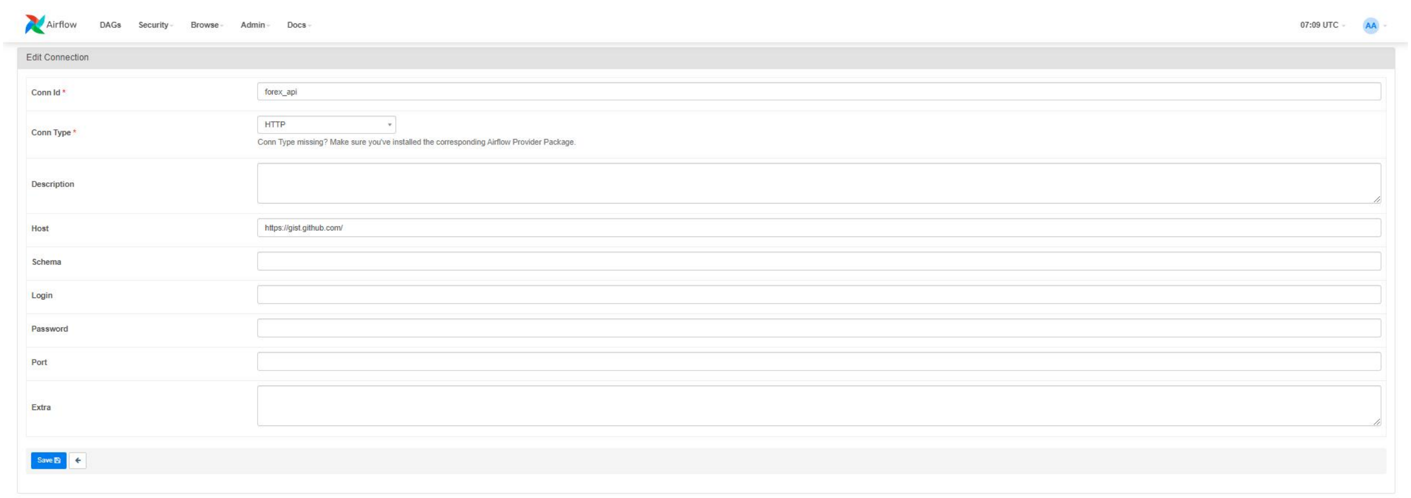
In browser type locathost:8080

Username : airflow
Password : airflow

On Airflow UI navigate to Admin - > Connections -> +

Conn Id * : forex_api
Conn Type * : HTTP
Host : https://gist.github.com/

Save

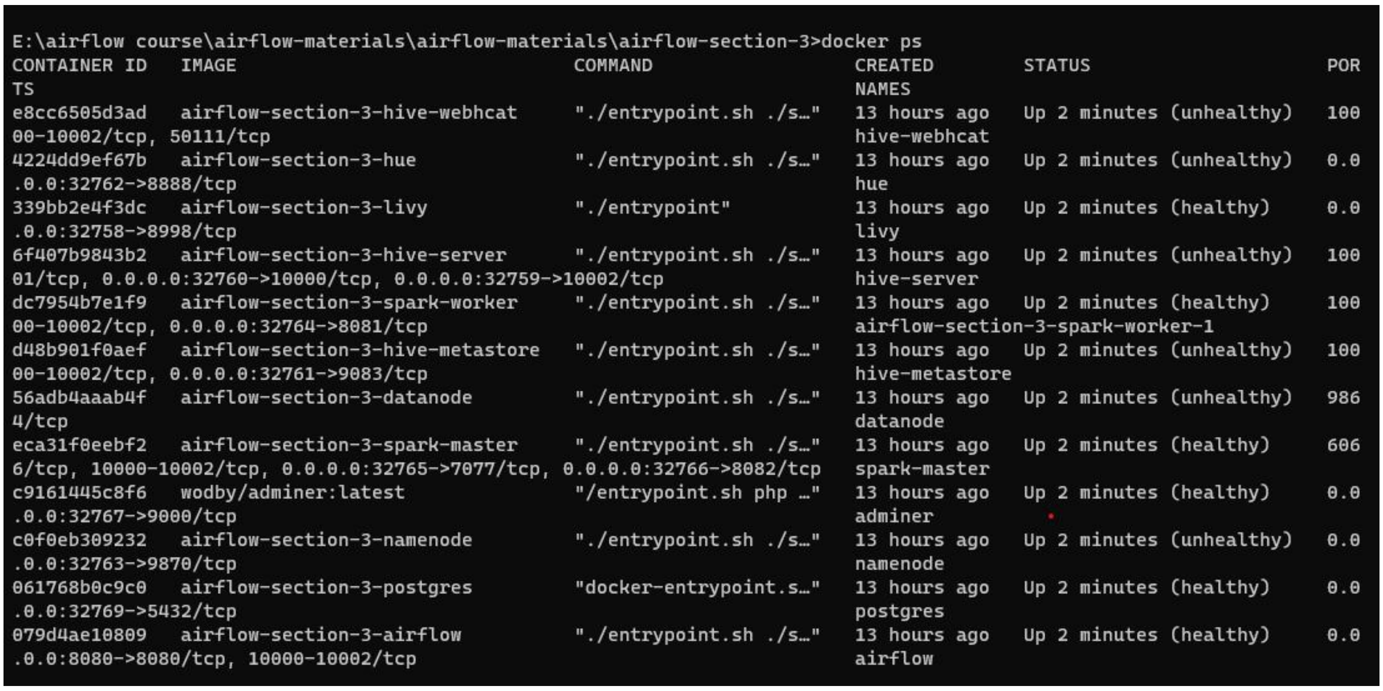


Edit Connection

Conn Id *	forex_api
Conn Type *	HTTP Conn Type missing? Make sure you've installed the corresponding Airflow Provider Package.
Description	
Host	https://gist.github.com/
Schema	
Login	
Password	
Port	
Extra	

In cmd type : docker ps

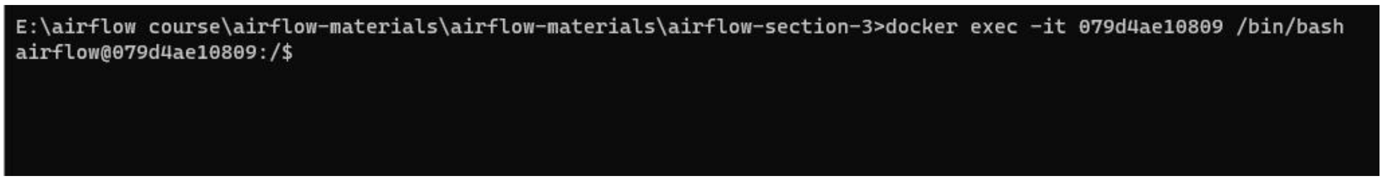
Copy container id of airflow-section-3-airflow



```
E:\airflow course\airflow-materials\airflow-materials\airflow-section-3>docker ps
CONTAINER ID   IMAGE                              COMMAND                  CREATED        STATUS                      POR
TS                                                                                         NAMES
e8cc6505d3ad   airflow-section-3-hive-webhcat     "./entrypoint.sh ./s…"   13 hours ago   Up 2 minutes (unhealthy)    100
00-10002/tcp, 50111/tcp                                                                    hive-webhcat
4224dd9ef67b   airflow-section-3-hue              "./entrypoint.sh ./s…"   13 hours ago   Up 2 minutes (unhealthy)    0.0
.0.0:32762->8888/tcp                                                                       hue
339bb2e4f3dc   airflow-section-3-livy             "./entrypoint"           13 hours ago   Up 2 minutes (healthy)      0.0
.0.0:32758->8998/tcp                                                                       livy
6f407b9843b2   airflow-section-3-hive-server      "./entrypoint.sh ./s…"   13 hours ago   Up 2 minutes (unhealthy)    100
01/tcp, 0.0.0.0:32760->10000/tcp, 0.0.0.0:32759->10002/tcp                                 hive-server
dc7954b7e1f9   airflow-section-3-spark-worker     "./entrypoint.sh ./s…"   13 hours ago   Up 2 minutes (healthy)      100
00-10002/tcp, 0.0.0.0:32764->8081/tcp                                                      airflow-section-3-spark-worker-1
d48b901f0aef   airflow-section-3-hive-metastore   "./entrypoint.sh ./s…"   13 hours ago   Up 2 minutes (unhealthy)    100
00-10002/tcp, 0.0.0.0:32761->9083/tcp                                                      hive-metastore
56adb4aaab4f   airflow-section-3-datanode         "./entrypoint.sh ./s…"   13 hours ago   Up 2 minutes (unhealthy)    986
4/tcp                                                                                      datanode
eca31f0eebf2   airflow-section-3-spark-master     "./entrypoint.sh ./s…"   13 hours ago   Up 2 minutes (healthy)      606
6/tcp, 10000-10002/tcp, 0.0.0.0:32765->7077/tcp, 0.0.0.0:32766->8082/tcp                   spark-master
c9161445c8f6   wodby/adminer:latest               "/entrypoint.sh php …"   13 hours ago   Up 2 minutes (healthy)      0.0
.0.0:32767->9000/tcp                                                                       adminer
c0f0eb309232   airflow-section-3-namenode         "./entrypoint.sh ./s…"   13 hours ago   Up 2 minutes (unhealthy)    0.0
.0.0:32763->9870/tcp                                                                       namenode
061768b0c9c0   airflow-section-3-postgres         "docker-entrypoint.s…"   13 hours ago   Up 2 minutes (healthy)      0.0
.0.0:32769->5432/tcp                                                                       postgres
079d4ae10809   airflow-section-3-airflow          "./entrypoint.sh ./s…"   13 hours ago   Up 2 minutes (healthy)      0.0
.0.0:8080->8080/tcp, 10000-10002/tcp                                                       airflow
```

In cmd type : docker exec -it 079d4ae10809 /bin/bash

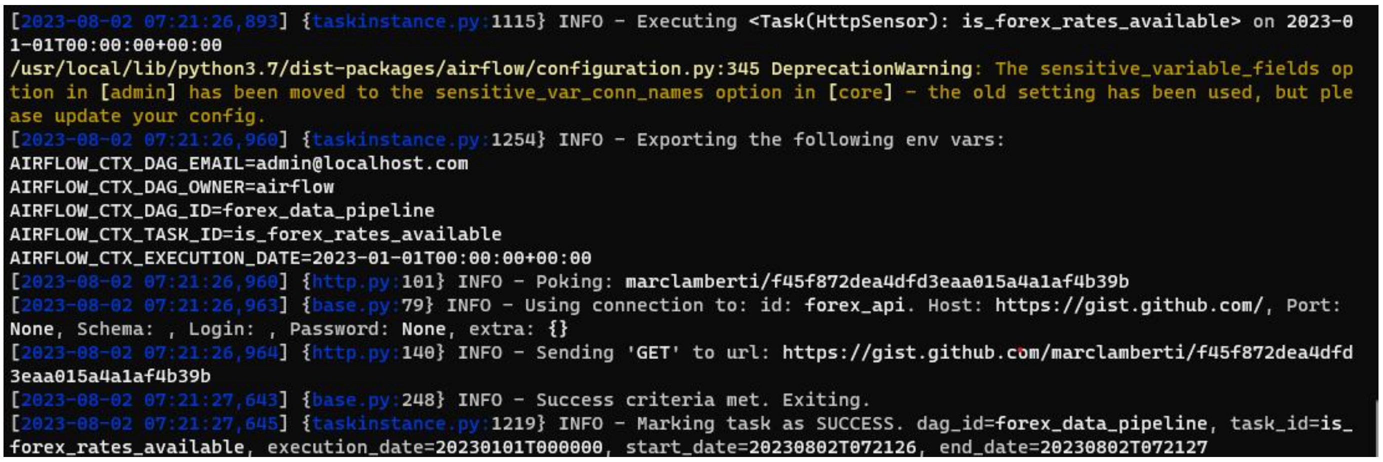
This command is used to open a bash terminal to access airflow CLI running inside container



```
E:\airflow course\airflow-materials\airflow-materials\airflow-section-3>docker exec -it 079d4ae10809 /bin/bash
airflow@079d4ae10809:/$
```

<u>Check if task "is_forex_rates_available" of the DAG "forex_data_pipeline" is working</u>

In cmd type: airflow tasks test forex_data_pipeline is_forex_rates_available 2023-01-01



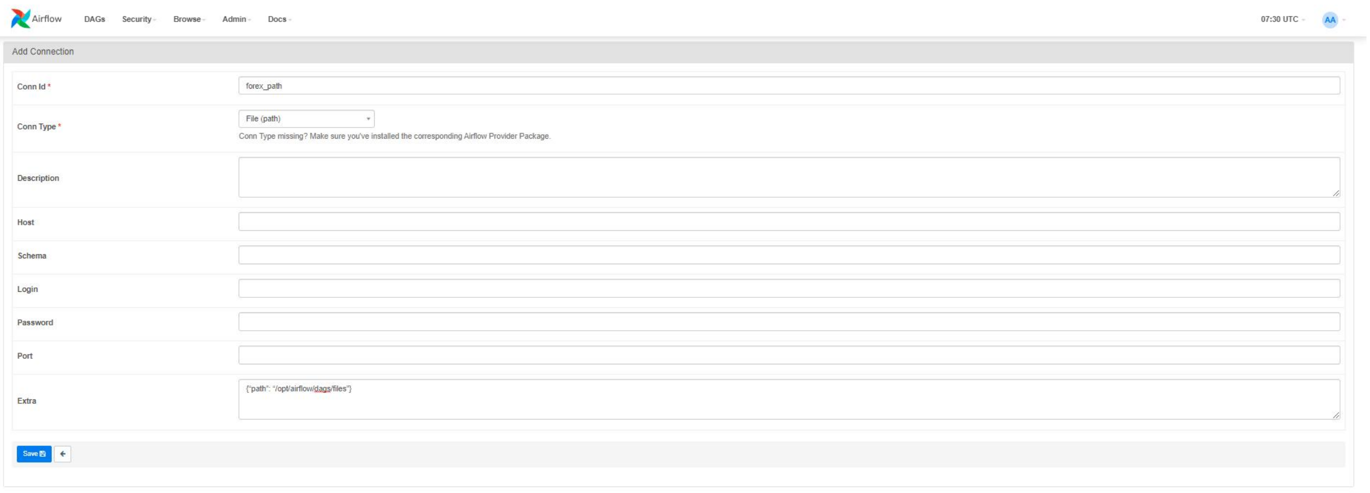
```
[2023-08-02 07:21:26,893] {taskinstance.py:1115} INFO - Executing <Task(HttpSensor): is_forex_rates_available> on 2023-0
1-01T00:00:00+00:00
/usr/local/lib/python3.7/dist-packages/airflow/configuration.py:345 DeprecationWarning: The sensitive_variable_fields op
tion in [admin] has been moved to the sensitive_var_conn_names option in [core] - the old setting has been used, but ple
ase update your config.
[2023-08-02 07:21:26,960] {taskinstance.py:1254} INFO - Exporting the following env vars:
AIRFLOW_CTX_DAG_EMAIL=admin@localhost.com
AIRFLOW_CTX_DAG_OWNER=airflow
AIRFLOW_CTX_DAG_ID=forex_data_pipeline
AIRFLOW_CTX_TASK_ID=is_forex_rates_available
AIRFLOW_CTX_EXECUTION_DATE=2023-01-01T00:00:00+00:00
[2023-08-02 07:21:26,960] {http.py:101} INFO - Poking: marclamberti/f45f872dea4dfd3eaa015a4a1af4b39b
[2023-08-02 07:21:26,963] {base.py:79} INFO - Using connection to: id: forex_api. Host: https://gist.github.com/, Port:
None, Schema: , Login: , Password: None, extra: {}
[2023-08-02 07:21:26,964] {http.py:140} INFO - Sending 'GET' to url: https://gist.github.com/marclamberti/f45f872dea4dfd
3eaa015a4a1af4b39b
[2023-08-02 07:21:27,643] {base.py:248} INFO - Success criteria met. Exiting.
[2023-08-02 07:21:27,645] {taskinstance.py:1219} INFO - Marking task as SUCCESS. dag_id=forex_data_pipeline, task_id=is_
forex_rates_available, execution_date=20230101T000000, start_date=20230802T072126, end_date=20230802T072127
```

<u>D) Create task named "if currency file is available" - File sensor and check:</u>

On Airflow UI navigate to Admin - > Connections -> +

Conn Id * : forex_path
Conn Type * : file(path)
Extra : {"path": "/opt/airflow/dags/files"} # this is where we will be looking if file exists.

Save



Add Connection

Conn Id *	forex_path
Conn Type *	File (path) Conn Type missing? Make sure you've installed the corresponding Airflow Provider Package.
Description	
Host	
Schema	
Login	
Password	
Port	
Extra	{"path": "/opt/airflow/dags/files"}

<u>Write File sensor code in the DAG</u>

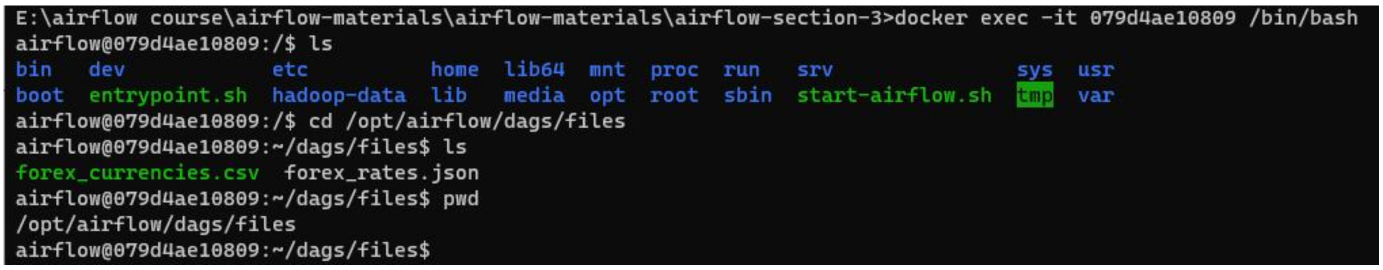
```
from airflow.sensors.filesystem import FileSensor

    is_forex_currencies_file_available = FileSensor(
        task_id="is_forex_currencies_file_available",
        fs_conn_id="forex_path",
        filepath="forex_currencies.csv",
        poke_interval=5,
        timeout=20
    )
```

'poke_interval' specifies the time interval (in seconds) at which the FileSensor checks for the existence of the specified file on the filesystem.

'timeout' defines the maximum time (in seconds) the FileSensor will wait for the file to become available on the filesystem.

We may wonder where is the following path "/opt/airflow/dags/files", this is inside a docker container where Airflow is running.

```
In cmd:
docker exec -it 079d4ae10809 /bin/bash
ls
cd /opt/airflow/dags/files
ls
pwd
```

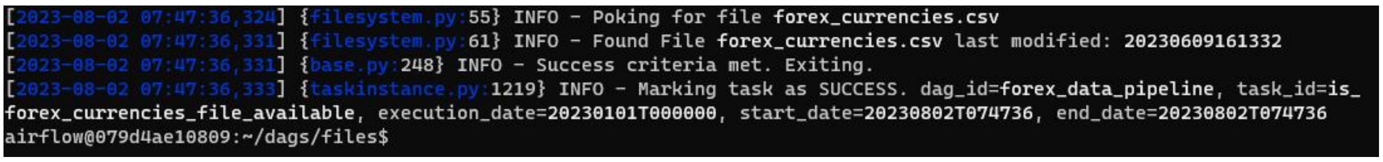


```
E:\airflow course\airflow-materials\airflow-materials\airflow-section-3>docker exec -it 079d4ae10809 /bin/bash
airflow@079d4ae10809:/$ ls
bin   dev            etc           home  lib64  mnt  proc  run   srv               sys  usr
boot  entrypoint.sh  hadoop-data   lib   media  opt  root  sbin  start-airflow.sh  tmp  var
airflow@079d4ae10809:/$ cd /opt/airflow/dags/files
airflow@079d4ae10809:~/dags/files$ ls
forex_currencies.csv  forex_rates.json
airflow@079d4ae10809:~/dags/files$ pwd
/opt/airflow/dags/files
airflow@079d4ae10809:~/dags/files$
```

Any files we keep in our local file system in "mnt/airflow/dags" will be in airflow container "/opt/airflow/dags/".

Test if the task "is_forex_currencies_file_available" is working:

```
In cmd type: airflow tasks test forex_data_pipeline is_forex_currencies_file_available 2023-01-01
```



```
[2023-08-02 07:47:36,324] {filesystem.py:55} INFO - Poking for file forex_currencies.csv
[2023-08-02 07:47:36,331] {filesystem.py:61} INFO - Found File forex_currencies.csv last modified: 20230609161332
[2023-08-02 07:47:36,331] {base.py:248} INFO - Success criteria met. Exiting.
[2023-08-02 07:47:36,333] {taskinstance.py:1219} INFO - Marking task as SUCCESS. dag_id=forex_data_pipeline, task_id=is_
forex_currencies_file_available, execution_date=20230101T000000, start_date=20230802T074736, end_date=20230802T074736
airflow@079d4ae10809:~/dags/files$
```

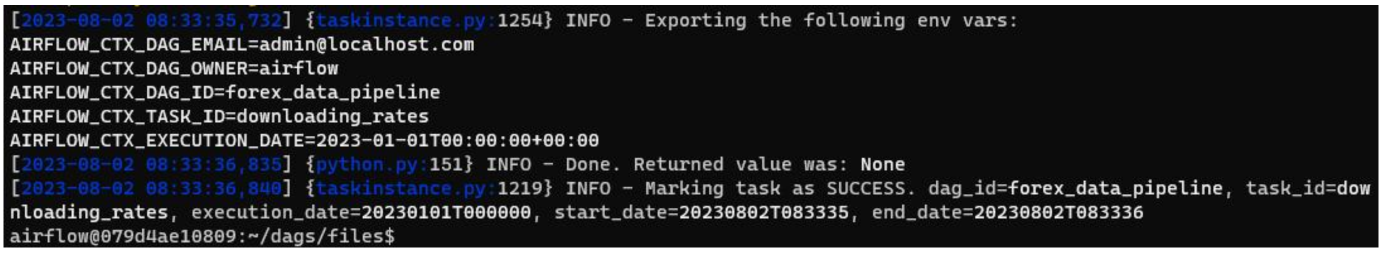
E) Create task named "downloading_rates" to download the forex rates from API - Python operator and check

```
from airflow.operators.python_operator import PythonOperator
import csv
import requests
import json
```

```
def download_rates():
    BASE_URL =
"https://gist.githubusercontent.com/naveenrajusg/497e10579edfe65fdf1c3d60a387fa20/raw/"
    ENDPOINTS = {
        'USD': 'api_forex_exchange_usd.json',
        'EUR': 'api_forex_exchange_eur.json'
    }
    with open('/opt/airflow/dags/files/forex_currencies.csv') as forex_currencies:
        reader = csv.DictReader(forex_currencies, delimiter=';')
        for idx, row in enumerate(reader):
            base = row['base']
            with_pairs = row['with_pairs'].split(' ')
            indata = requests.get(f"{BASE_URL}{ENDPOINTS[base]}").json()
            outdata = {'base': base, 'rates': {}, 'last_update': indata['date']}
            for pair in with_pairs:
                outdata['rates'][pair] = indata['rates'][pair]
            with open('/opt/airflow/dags/files/forex_rates.json', 'a') as outfile:
                json.dump(outdata, outfile)
                outfile.write('\n')
# Inside the DAG definition add the below code


    downloading_rates = PythonOperator(
            task_id="downloading_rates",
            python_callable=download_rates
    )
```

Test if the task "downloading_rates" is working:

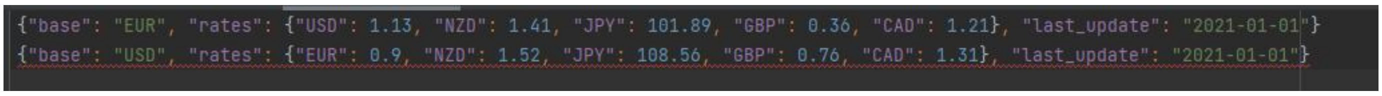
```
In cmd type: airflow tasks test forex_data_pipeline downloading_rates 2023-01-01
```



```
[2023-08-02 08:33:35,732] {taskinstance.py:1254} INFO - Exporting the following env vars:
AIRFLOW_CTX_DAG_EMAIL=admin@localhost.com
AIRFLOW_CTX_DAG_OWNER=airflow
AIRFLOW_CTX_DAG_ID=forex_data_pipeline
AIRFLOW_CTX_TASK_ID=downloading_rates
AIRFLOW_CTX_EXECUTION_DATE=2023-01-01T00:00:00+00:00
[2023-08-02 08:33:36,835] {python.py:151} INFO - Done. Returned value was: None
[2023-08-02 08:33:36,840] {taskinstance.py:1219} INFO - Marking task as SUCCESS. dag_id=forex_data_pipeline, task_id=dow
nloading_rates, execution_date=20230101T000000, start_date=20230802T083335, end_date=20230802T083336
airflow@079d4ae10809:~/dags/files$
```

Now we can see file named forex_rates.json created in local file system "/mnt/airflow/dags/files/forex_rates.json" the same will be created inside Airflow docker container.

File contents:



```
{"base": "EUR", "rates": {"USD": 1.13, "NZD": 1.41, "JPY": 101.89, "GBP": 0.36, "CAD": 1.21}, "last_update": "2021-01-01"}
{"base": "USD", "rates": {"EUR": 0.9, "NZD": 1.52, "JPY": 108.56, "GBP": 0.76, "CAD": 1.31}, "last_update": "2021-01-01"}
```

F) Create a task named "forex_rates" - Bash operator to save the forex rates into HDFS and check it

In real time if files are huge we need to save them in HDFS.

Bash operator is an operator that allows us to execute a bash command or a script as a task within an Airflow DAG. It is one of the core operator provided by Airflow.

Define bash operator in the DAG, here we are trying to save forex_rates.json created in last step into HDFS.

HUE :

Hue (Hadoop User Experience) is an open-source web-based interface that provides a graphical user interface (GUI) for interacting with Apache Hadoop and its ecosystem components. It is designed to simplify and enhance the user experience for working with Hadoop and related tools. Hue offers a wide range of features and capabilities that make it easier for users to interact with Hadoop clusters, perform data analysis, and develop workflows. Some key features of

Hue include:

- File Browser: Allows users to navigate and manage files stored in Hadoop Distributed File System (HDFS) or other compatible file systems.
- Query Editors: Provides interactive editors for writing and executing queries in languages like Hive, Impala, Pig, and Spark SQL it includes features like syntax highlighting, auto-completion, and result visualization.
- Job Designer: Enables users to visually design and schedule workflows using tools like Oozie and Apache Workflow Scheduler (AWS). It simplifies the creation and management of data pipelines.
- Data Browsing and Visualization Allows users to explore and analyze data stored in Hadoop using tools like Apache Hive, Apache Impala, and Apache Solr. It provides Interactive visualizations and data exploration capabilities.
- Security and User Management Offers features for managing user access, authentication, and authorization to Hadoop resources. It integrates with security mechanisms like and LDAP

Enables users to create and share dashboards for data visualization and reporting purposes. It supports various charting libraries and allows customization of dashboards

- Job Monitoring Provides monitoring and tracking capabilities for jobs running on the cluster. It allows users view job status, logs, and performance metrics.

```
from airflow.operators.bash_operator import BashOperator

    saving_rates = BashOperator(
        task_id="saving_rates",
        bash_command="""
            hdfs dfs -mkdir -p /forex && \   # creates folder named forex in HDFS
            hdfs dfs -put -f $AIRFLOW_HOME/dags/files/forex_rates.json /forex       # Copy file from Airflow
container to HDFS folder
            """
    )
```

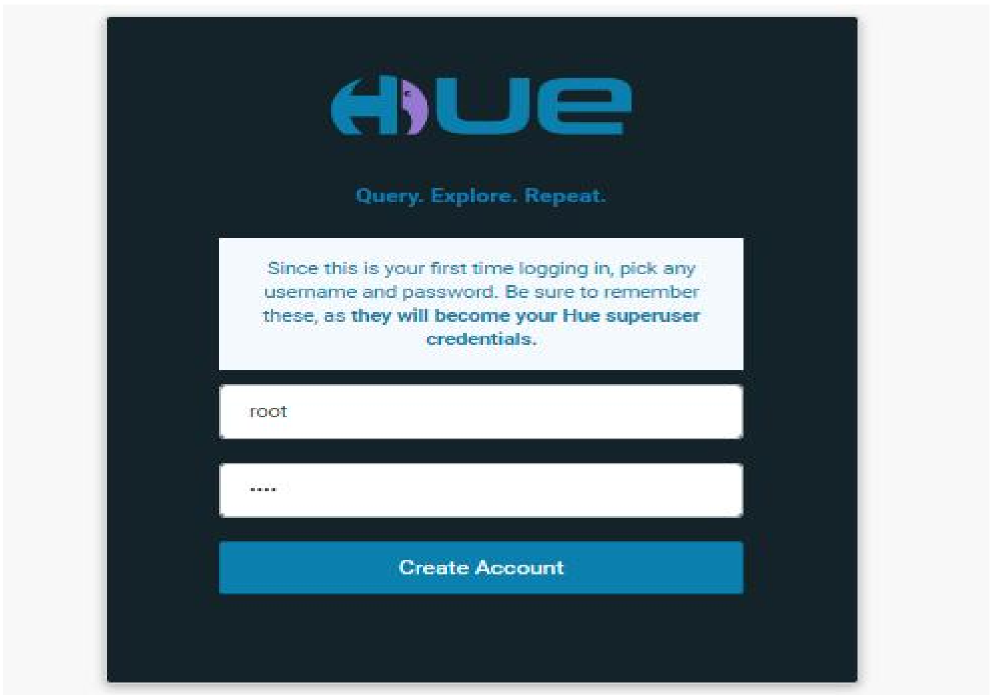
In browser type : http://localhost:32762/ (It takes to HUE login)
Username : root
Password : root

Restart docker container if HUE port is not working

```
In cmd:
restart.sh
```

restart.sh

```
#!/bin/bash

./stop.sh
./start.sh
```



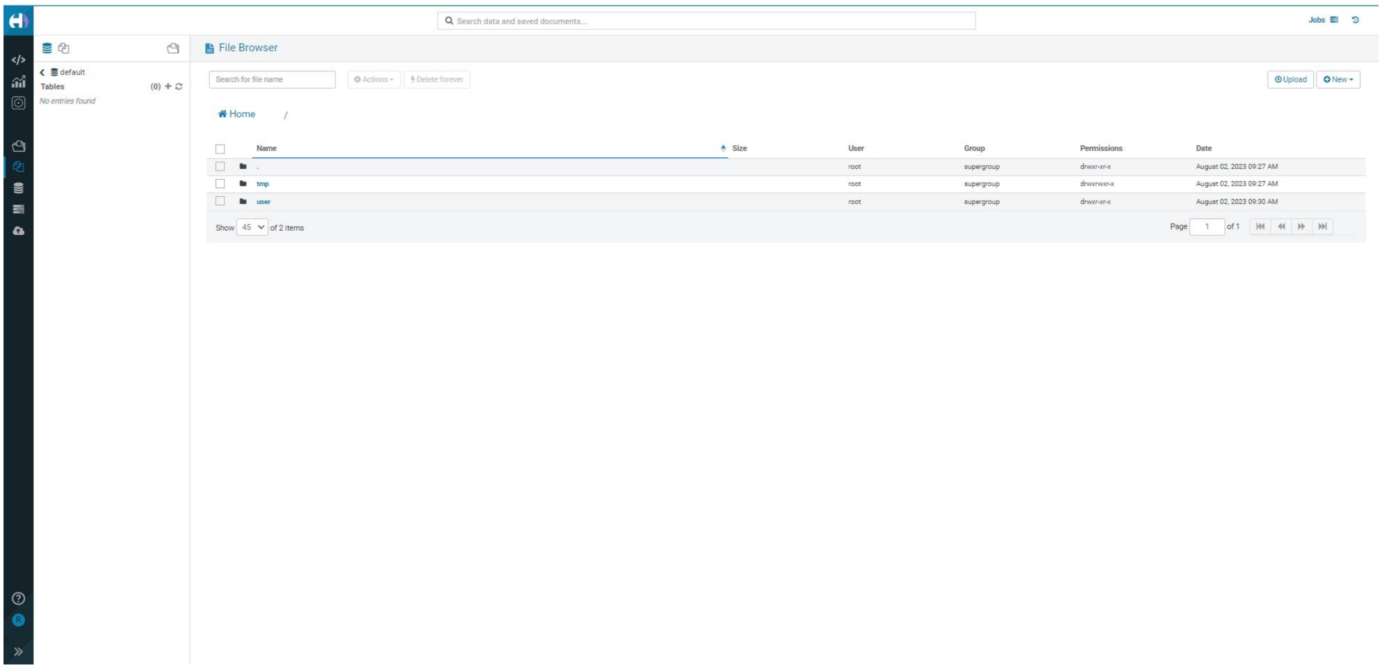
Query. Explore. Repeat.

Since this is your first time logging in, pick any username and password. Be sure to remember these, as **they will become your Hue superuser credentials.**

root

••••

In HUE UI navigate to files to see files in HDFS



	Name	Size	User	Group	Permissions	Date
☐	.		root	supergroup	drwxr-xr-x	August 02, 2023 09:27 AM
☐	tmp		root	supergroup	drwxrwxr-x	August 02, 2023 09:27 AM
☐	user		root	supergroup	drwxr-xr-x	August 02, 2023 09:30 AM

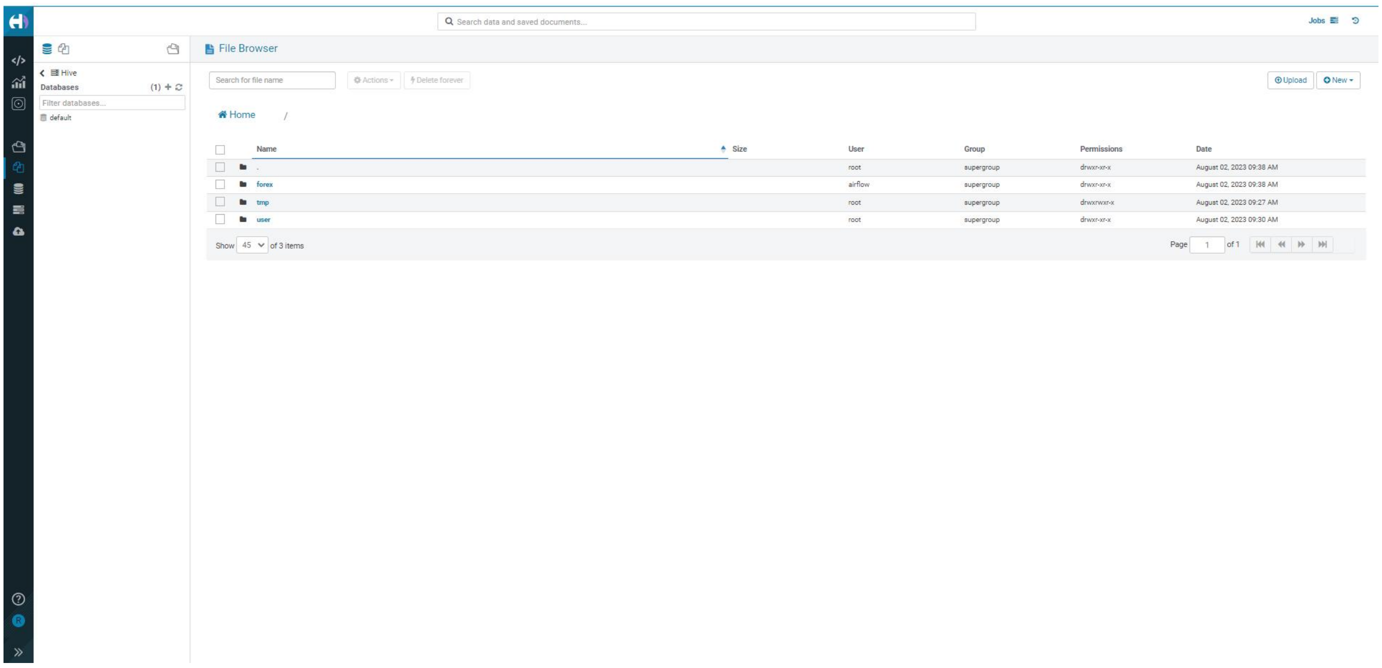
Show 45 of 2 items Page 1 of 1

```
In cmd:

docker ps # copy container id of airflow-section-3-airflow

docker exec -it 510f580614fb /bin/bash
```

<u>Test if the task “saving_rates” is working:</u>

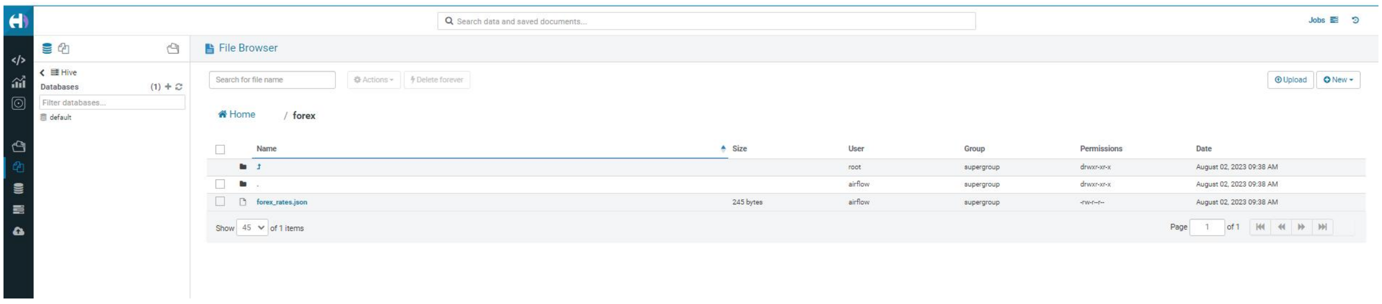
```
In cmd type: airflow tasks test forex_data_pipeline saving_rates 2023-01-01
```

Now refresh the HUE UI, now we can see forex folder created with a file forex_rates.json in it.



Name	Size	User	Group	Permissions	Date
.		root	supergroup	drwxr-xr-x	August 02, 2023 09:38 AM
forex		airflow	supergroup	drwxr-xr-x	August 02, 2023 09:38 AM
tmp		root	supergroup	drwxrwxr-x	August 02, 2023 09:27 AM
user		root	supergroup	drwxr-xr-x	August 02, 2023 09:30 AM

Show 45 of 3 items Page 1 of 1



Name	Size	User	Group	Permissions	Date
↥		root	supergroup	drwxr-xr-x	August 02, 2023 09:38 AM
.		airflow	supergroup	drwxr-xr-x	August 02, 2023 09:38 AM
forex_rates.json	245 bytes	airflow	supergroup	-rw-r--r--	August 02, 2023 09:38 AM

Show 45 of 1 items Page 1 of 1

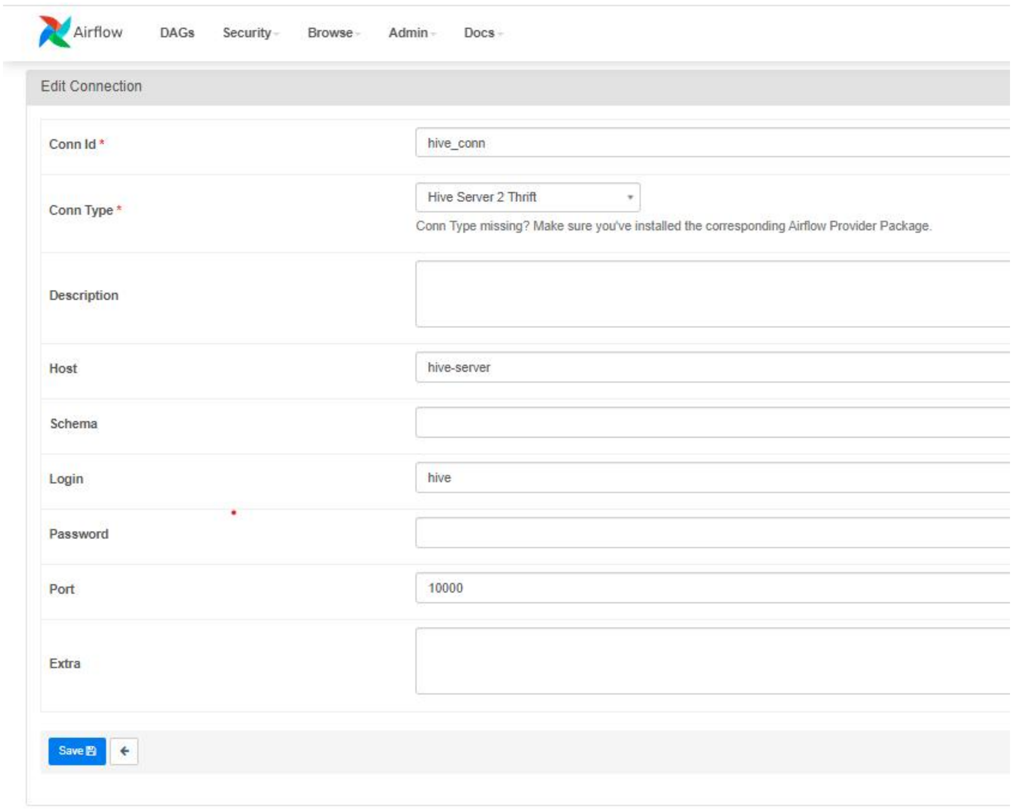
G) <u>Create a task named “creating_forex_rates_table” - Hive operator and check it</u>

Now we need to create a HIVE table for forex data so we can query it.

```
from airflow.providers.apache.hive.operators.hive import HiveOperator

    creating_forex_rates_table = HiveOperator(
        task_id="creating_forex_rates_table",
        hive_cli_conn_id="hive_conn",
        hql="""
        CREATE EXTERNAL TABLE IF NOT EXISTS forex_rates(
            base STRING,
            last_update DATE,
            eur DOUBLE,
            usd DOUBLE,
            nzd DOUBLE,
            gbp DOUBLE,
            jpy DOUBLE,
            cad DOUBLE
            )
        ROW FORMAT DELIMITED
        FIELDS TERMINATED BY ','
        STORED AS TEXTFILE
    """
)
```

In Airflow UI page navigate to Admin - > connections - > +

```
Conn Id * : hive_conn
Conn Type * : Hive Server 2 Thrift
Host : hive-server
Login : hive
Password : hive
Port : 10000

Save
```

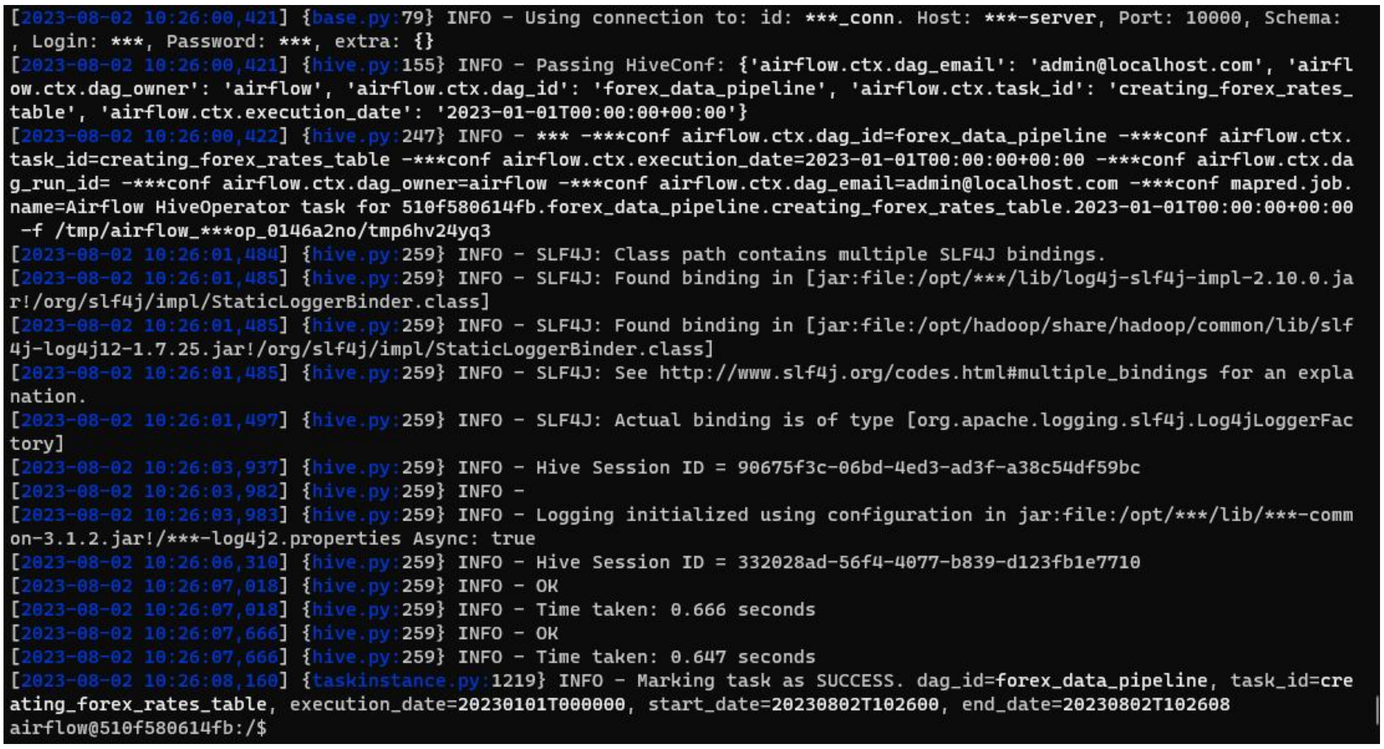


Edit Connection

Conn Id *	hive_conn
Conn Type *	Hive Server 2 Thrift Conn Type missing? Make sure you've installed the corresponding Airflow Provider Package.
Description	
Host	hive-server
Schema	
Login	hive
Password	
Port	10000
Extra	

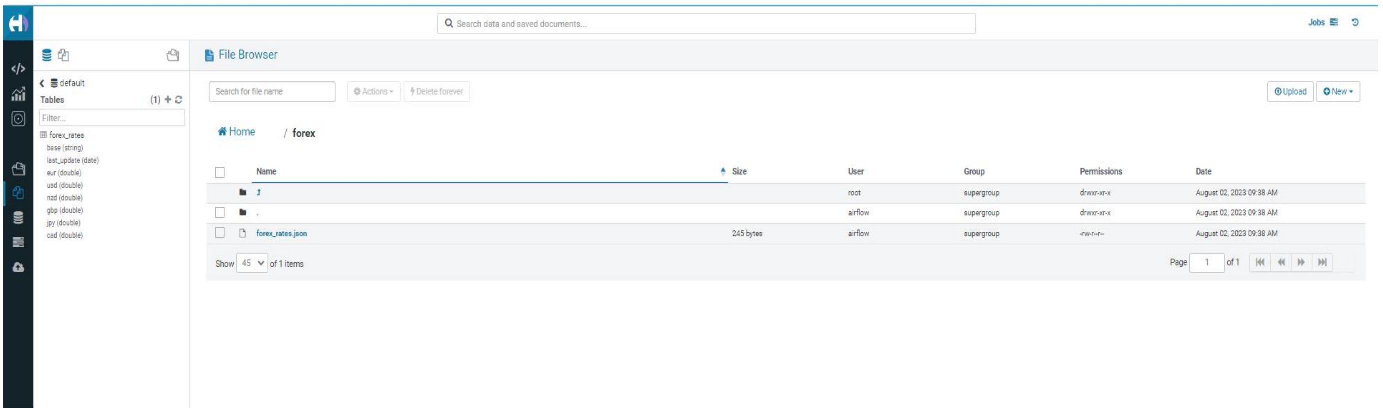
<u>Test the task:</u>

```
airflow tasks test forex_data_pipeline creating_forex_rates_table 2023-01-01
```



```
[2023-08-02 10:26:00,421] {base.py:79} INFO - Using connection to: id: ***_conn. Host: ***-server, Port: 10000, Schema:
, Login: ***, Password: ***, extra: {}
[2023-08-02 10:26:00,421] {hive.py:155} INFO - Passing HiveConf: {'airflow.ctx.dag_email': 'admin@localhost.com', 'airfl
ow.ctx.dag_owner': 'airflow', 'airflow.ctx.dag_id': 'forex_data_pipeline', 'airflow.ctx.task_id': 'creating_forex_rates_
table', 'airflow.ctx.execution_date': '2023-01-01T00:00:00+00:00'}
[2023-08-02 10:26:00,422] {hive.py:247} INFO - *** -***conf airflow.ctx.dag_id=forex_data_pipeline -***conf airflow.ctx.
task_id=creating_forex_rates_table -***conf airflow.ctx.execution_date=2023-01-01T00:00:00+00:00 -***conf airflow.ctx.da
g_run_id= -***conf airflow.ctx.dag_owner=airflow -***conf airflow.ctx.dag_email=admin@localhost.com -***conf mapred.job.
name=Airflow HiveOperator task for 510f580614fb.forex_data_pipeline.creating_forex_rates_table.2023-01-01T00:00:00+00:00
 -f /tmp/airflow_***op_0146a2no/tmp6hv24yq3
[2023-08-02 10:26:01,484] {hive.py:259} INFO - SLF4J: Class path contains multiple SLF4J bindings.
[2023-08-02 10:26:01,485] {hive.py:259} INFO - SLF4J: Found binding in [jar:file:/opt/***/lib/log4j-slf4j-impl-2.10.0.ja
r!/org/slf4j/impl/StaticLoggerBinder.class]
[2023-08-02 10:26:01,485] {hive.py:259} INFO - SLF4J: Found binding in [jar:file:/opt/hadoop/share/hadoop/common/lib/slf
4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
[2023-08-02 10:26:01,485] {hive.py:259} INFO - SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an expla
nation.
[2023-08-02 10:26:01,497] {hive.py:259} INFO - SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFac
tory]
[2023-08-02 10:26:03,937] {hive.py:259} INFO - Hive Session ID = 90675f3c-06bd-4ed3-ad3f-a38c54df59bc
[2023-08-02 10:26:03,982] {hive.py:259} INFO -
[2023-08-02 10:26:03,983] {hive.py:259} INFO - Logging initialized using configuration in jar:file:/opt/***/lib/***-comm
on-3.1.2.jar!/***-log4j2.properties Async: true
[2023-08-02 10:26:06,310] {hive.py:259} INFO - Hive Session ID = 332028ad-56f4-4077-b839-d123fb1e7710
[2023-08-02 10:26:07,018] {hive.py:259} INFO - OK
[2023-08-02 10:26:07,018] {hive.py:259} INFO - Time taken: 0.666 seconds
[2023-08-02 10:26:07,666] {hive.py:259} INFO - OK
[2023-08-02 10:26:07,666] {hive.py:259} INFO - Time taken: 0.647 seconds
[2023-08-02 10:26:08,160] {taskinstance.py:1219} INFO - Marking task as SUCCESS. dag_id=forex_data_pipeline, task_id=cre
ating_forex_rates_table, execution_date=20230101T000000, start_date=20230802T102600, end_date=20230802T102608
airflow@510f580614fb:/$
```

In HUE we can see the HIVE table would have been created

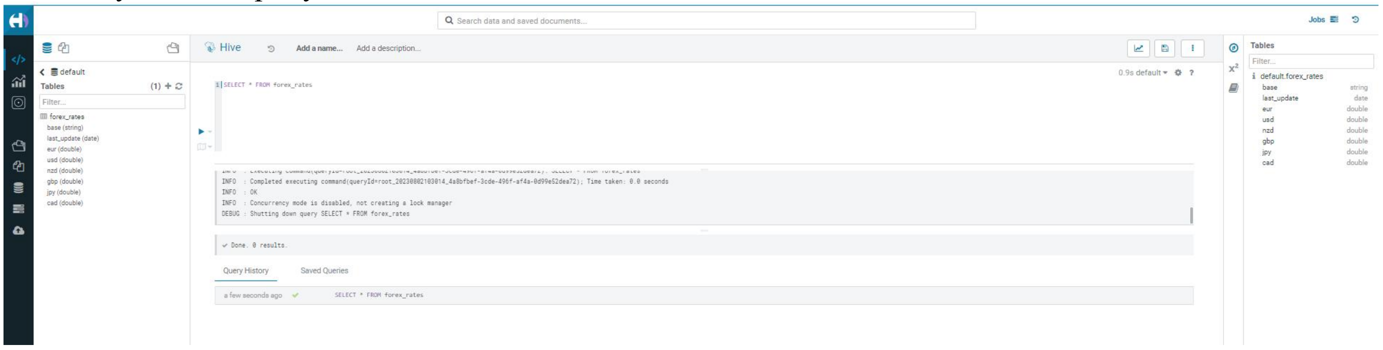


forex_rates
base (string)
last_update (date)
eur (double)
usd (double)
nzd (double)
gbp (double)
jpy (double)
cad (double)

Name	Size	User	Group	Permissions	Date
↥		root	supergroup	drwxr-xr-x	August 02, 2023 09:38 AM
.		airflow	supergroup	drwxr-xr-x	August 02, 2023 09:38 AM
forex_rates.json	245 bytes	airflow	supergroup	-rw-r--r--	August 02, 2023 09:38 AM

Show 45 of 1 items   Page 1 of 1

No we try to write query in HUE



```
SELECT * FROM forex_rates
```

```
INFO  : Completed executing command(queryId=root_20230802103014_4a8bfbef-3cde-496f-af4a-0d99e52dea72); Time taken: 0.0 seconds
INFO  : OK
INFO  : Concurrency mode is disabled, not creating a lock manager
DEBUG : Shutting down query SELECT * FROM forex_rates
```

✓ Done. 0 results.

a few seconds ago ✓ SELECT * FROM forex_rates

default.forex_rates

base	string
last_update	date
eur	double
usd	double
nzd	double
gbp	double
jpy	double
cad	double

We cannot see results as there is no data in table.

H) Process the forex rates with Spark - Spark submit operator

```
from airflow.providers.apache.spark.operators.spark_submit import SparkSubmitOperator

    forex_processing = SparkSubmitOperator(
        task_id="forex_processing",
        application="/opt/airflow/dags/scripts/forex_processing.py",
        conn_id="spark_conn",
        verbose=False
    )
```

The below program basically creates a Spark session, read forex_rates.json file and then do some pre processing on it and the finally insert it into forex_rates table.

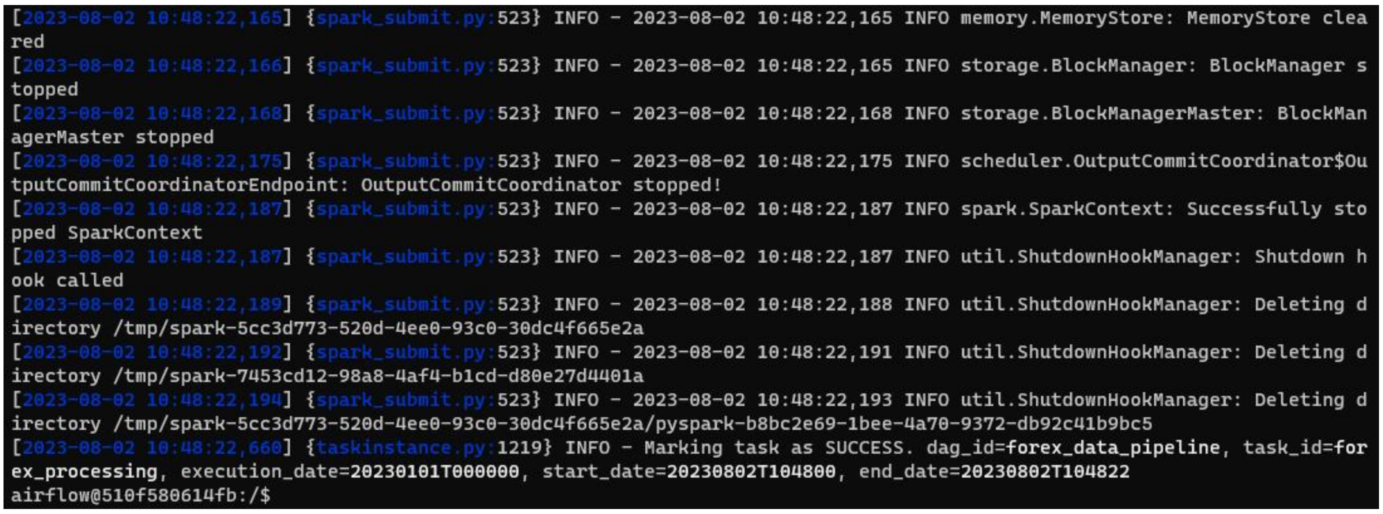
```
forex_processing.py

from os.path import expanduser, join, abspath

from pyspark.sql import SparkSession
from pyspark.sql.functions import from_json

warehouse_location = abspath('spark-warehouse')

# Initialize Spark Session

#warehouse_location?

spark = SparkSession \
    .builder \
    .appName("Forex processing") \
    .config("spark.sql.warehouse.dir", warehouse_location) \
    .enableHiveSupport() \
    .getOrCreate()

# Read the file forex_rates.json from the HDFS
df = spark.read.json('hdfs://namenode:9000/forex/forex_rates.json')

# Drop the duplicated rows based on the base and last_update columns
forex_rates = df.select('base', 'last_update', 'rates.eur', 'rates.usd', 'rates.cad', 'rates.gbp', 'rates.jpy',
'rates.nzd') \
    .dropDuplicates(['base', 'last_update']) \
    .fillna(0, subset=['EUR', 'USD', 'JPY', 'CAD', 'GBP', 'NZD'])

# Export the dataframe into the Hive table forex_rates
forex_rates.write.mode("append").insertInto("forex_rates")
```

In Airflow UI navigate to Admin - > Connections - > +

```
Conn Id * : spark_conn
Conn Type * : Spark
Host : spark://spark-master
Port : 7077

Save
```

In cmd:

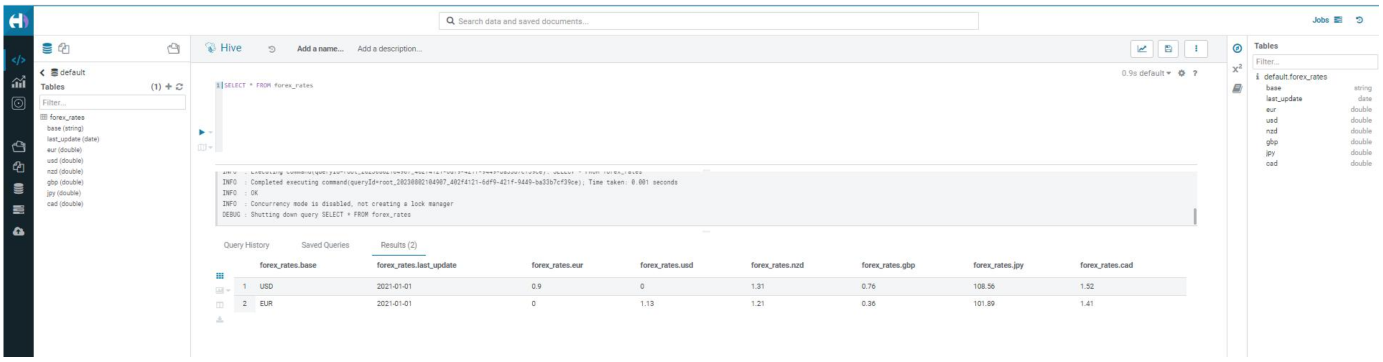
```
airflow tasks test forex_data_pipeline forex_processing 2023-01-01
```



```
[2023-08-02 10:48:22,165] {spark_submit.py:523} INFO - 2023-08-02 10:48:22,165 INFO memory.MemoryStore: MemoryStore clea
red
[2023-08-02 10:48:22,166] {spark_submit.py:523} INFO - 2023-08-02 10:48:22,165 INFO storage.BlockManager: BlockManager s
topped
[2023-08-02 10:48:22,168] {spark_submit.py:523} INFO - 2023-08-02 10:48:22,168 INFO storage.BlockManagerMaster: BlockMan
agerMaster stopped
[2023-08-02 10:48:22,175] {spark_submit.py:523} INFO - 2023-08-02 10:48:22,175 INFO scheduler.OutputCommitCoordinator$Ou
tputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
[2023-08-02 10:48:22,187] {spark_submit.py:523} INFO - 2023-08-02 10:48:22,187 INFO spark.SparkContext: Successfully sto
pped SparkContext
[2023-08-02 10:48:22,187] {spark_submit.py:523} INFO - 2023-08-02 10:48:22,187 INFO util.ShutdownHookManager: Shutdown h
ook called
[2023-08-02 10:48:22,189] {spark_submit.py:523} INFO - 2023-08-02 10:48:22,188 INFO util.ShutdownHookManager: Deleting d
irectory /tmp/spark-5cc3d773-520d-4ee0-93c0-30dc4f665e2a
[2023-08-02 10:48:22,192] {spark_submit.py:523} INFO - 2023-08-02 10:48:22,191 INFO util.ShutdownHookManager: Deleting d
irectory /tmp/spark-7453cd12-98a8-4af4-b1cd-d80e27d4401a
[2023-08-02 10:48:22,194] {spark_submit.py:523} INFO - 2023-08-02 10:48:22,193 INFO util.ShutdownHookManager: Deleting d
irectory /tmp/spark-5cc3d773-520d-4ee0-93c0-30dc4f665e2a/pyspark-b8bc2e69-1bee-4a70-9372-db92c41b9bc5
[2023-08-02 10:48:22,660] {taskinstance.py:1219} INFO - Marking task as SUCCESS. dag_id=forex_data_pipeline, task_id=for
ex_processing, execution_date=20230101T000000, start_date=20230802T104800, end_date=20230802T104822
airflow@510f580614fb:/$
```

Run the query in HUE, and we can see results as we have load data into the table

SELECT * FROM forex_rates



```
SELECT * FROM forex_rates
```

```
INFO  : Completed executing command(queryId=root_20230802104907_402f4121-6df9-421f-9449-ba33b7cf39ce); Time taken: 0.001 seconds
INFO  : OK
INFO  : Concurrency mode is disabled, not creating a lock manager
DEBUG : Shutting down query SELECT * FROM forex_rates
```

	forex_rates.base	forex_rates.last_update	forex_rates.eur	forex_rates.usd	forex_rates.nzd	forex_rates.gbp	forex_rates.jpy	forex_rates.cad
1	USD	2021-01-01	0.9	0	1.31	0.76	108.56	1.52
2	EUR	2021-01-01	0	1.13	1.21	0.36	101.89	1.41

I) Send Email notifications - Email Operator

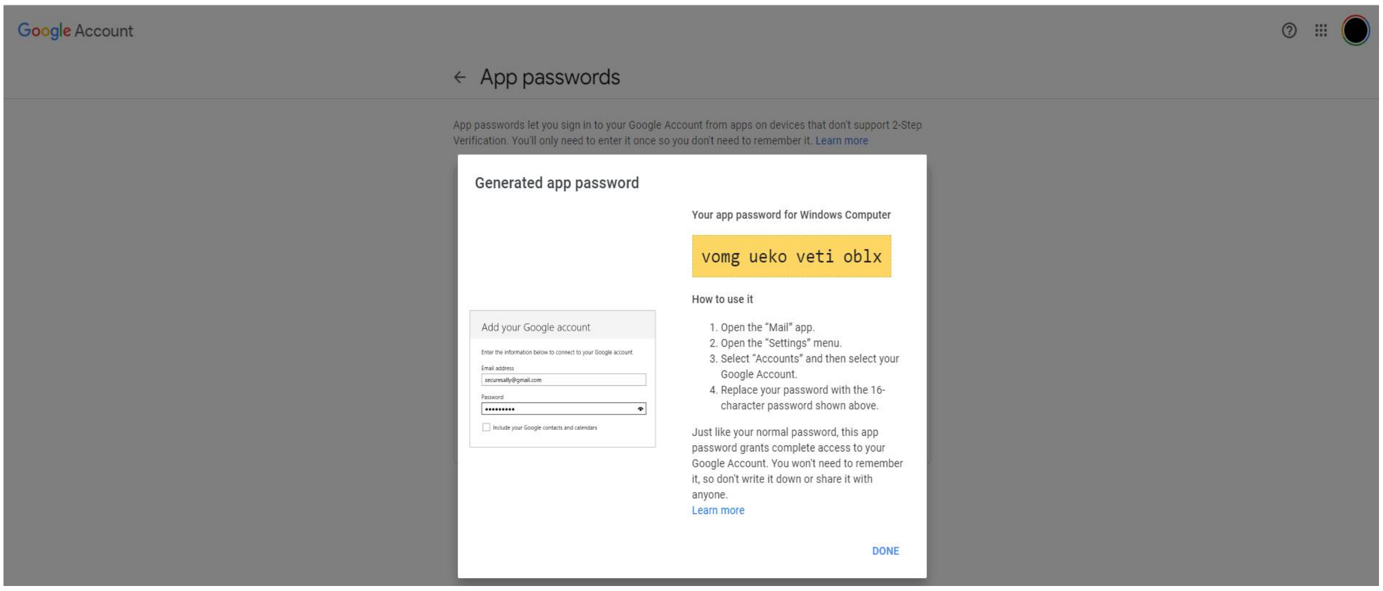
Now we need to configure our email provider so we can send email from our data pipeline by using our email address.

In the browser browse for https://security.google.com/settings/security/apppasswords and sign in to the gmail account.

App passwords
App passwords let you sign in to your Google Account from apps on devices that don't support 2-Step Verification.

Select App : Mail
Select device : Windows computer

Click on Generate
Click on Done


Google Account

App passwords

App passwords let you sign in to your Google Account from apps on devices that don't support 2-Step Verification. You'll only need to enter it once so you don't need to remember it. Learn more

Generated app password

Add your Google account

Enter the information below to connect to your Google account.

Email address: securesally@gmail.com

Password: ••••••••••

Include your Google contacts and calendars

Your app password for Windows Computer

`vomg ueko veti oblx`

How to use it

1. Open the "Mail" app.
2. Open the "Settings" menu.
3. Select "Accounts" and then select your Google Account.
4. Replace your password with the 16-character password shown above.

Just like your normal password, this app password grants complete access to your Google Account. You won't need to remember it, so don't write it down or share it with anyone.
Learn more

DONE

Configure SMTP in airflow.cfg located in "/mnt/airflow/"

```
[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 = smtp.gmail.com
smtp_starttls = True
smtp_ssl = False
# Example: smtp_user = airflow
smtp_user = naveenraju100@gmail.com
# Example: smtp_password = airflow
smtp_password = vomguekovetioblx
smtp_port = 587
smtp_mail_from = naveenraju100@gmail.com
smtp_timeout = 30
smtp_retry_limit = 5
```

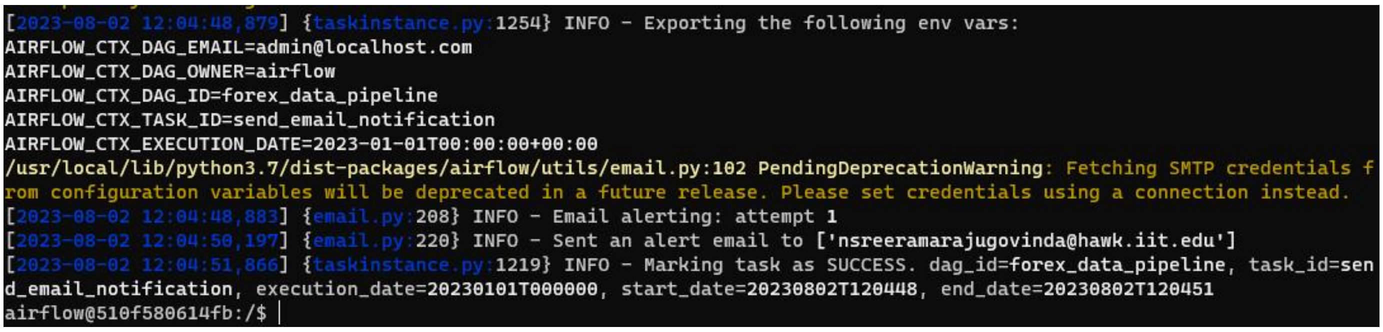
Restart Docker conainer

```
In cmd

docker-compose restart airflow
```

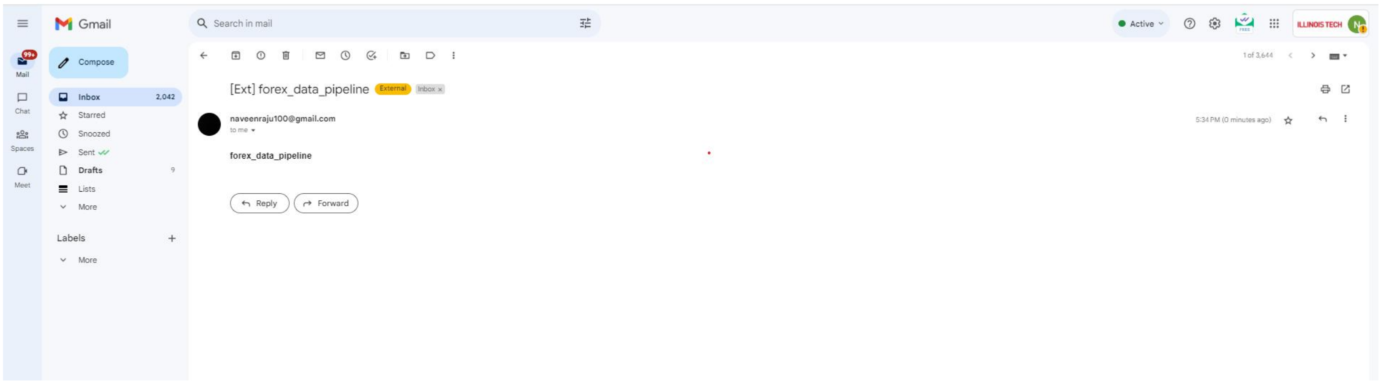
```
from airflow.operators.email import EmailOperator

    send_email_notification = EmailOperator(
        task_id="send_email_notification",
        to="nsreeramarajugovinda@hawk.iit.edu",
        subject="forex_data_pipeline",
        html_content="<h3>forex_data_pipeline</h3>"
    )
```

```
In cmd : docker ps

# Copy the container id of airflow-section-3-airflow
```

```
docker exec -it 510f580614fb /bin/bash

airflow tasks test forex_data_pipeline send_email_notification 2023-01-01
```



```
[2023-08-02 12:04:48,879] {taskinstance.py:1254} INFO - Exporting the following env vars:
AIRFLOW_CTX_DAG_EMAIL=admin@localhost.com
AIRFLOW_CTX_DAG_OWNER=airflow
AIRFLOW_CTX_DAG_ID=forex_data_pipeline
AIRFLOW_CTX_TASK_ID=send_email_notification
AIRFLOW_CTX_EXECUTION_DATE=2023-01-01T00:00:00+00:00
/usr/local/lib/python3.7/dist-packages/airflow/utils/email.py:102 PendingDeprecationWarning: Fetching SMTP credentials f
rom configuration variables will be deprecated in a future release. Please set credentials using a connection instead.
[2023-08-02 12:04:48,883] {email.py:208} INFO - Email alerting: attempt 1
[2023-08-02 12:04:50,197] {email.py:220} INFO - Sent an alert email to ['nsreeramarajugovinda@hawk.iit.edu']
[2023-08-02 12:04:51,866] {taskinstance.py:1219} INFO - Marking task as SUCCESS. dag_id=forex_data_pipeline, task_id=sen
d_email_notification, execution_date=20230101T000000, start_date=20230802T120448, end_date=20230802T120451
airflow@510f580614fb:/$ 
```



[Ext] forex_data_pipeline

naveenraju100@gmail.com
to me

5:34 PM (0 minutes ago)

forex_data_pipeline

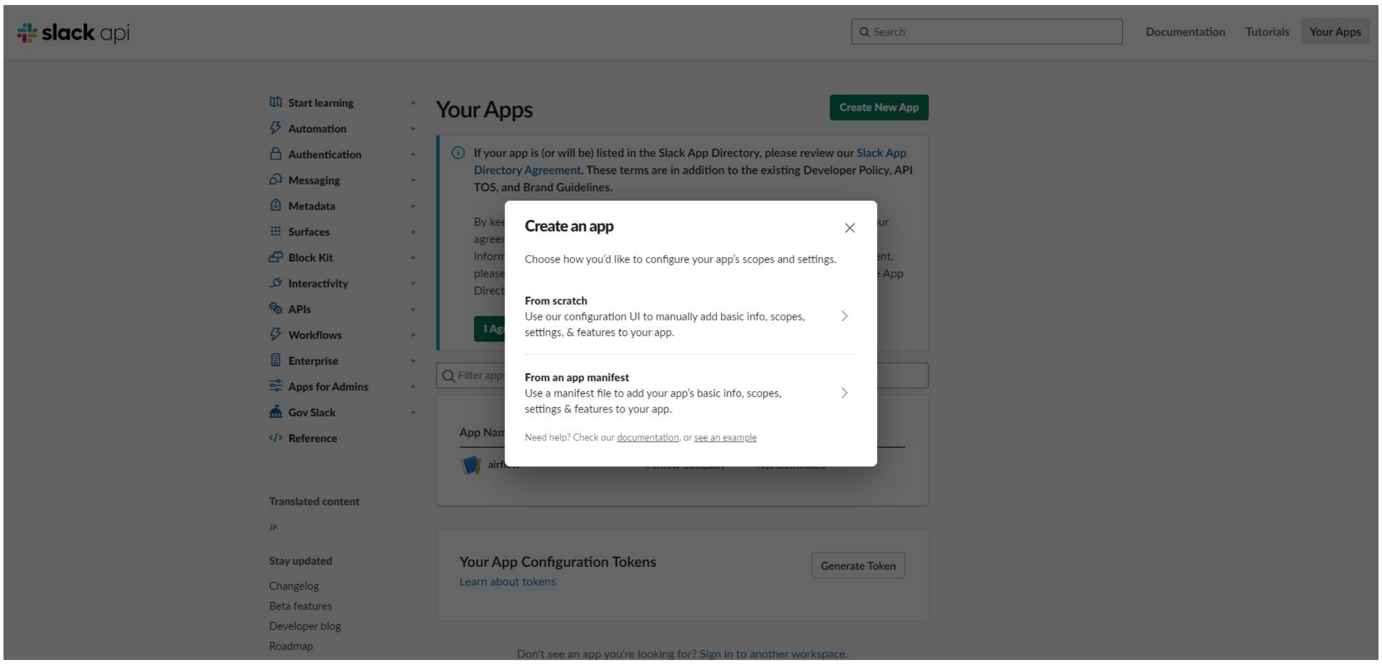
J) Send Slack notifications - SlackWebHookOperator

Create a new a new Slack Workspace named "Airflow Company"

In the browser browse https://api.slack.com/apps and click "Create an App"

Select "From an app manifest"

Select "From Scratch"



Create an app

Choose how you'd like to configure your app's scopes and settings.

From scratch
Use our configuration UI to manually add basic info, scopes, settings, & features to your app.

From an app manifest
Use a manifest file to add your app's basic info, scopes, settings & features to your app.

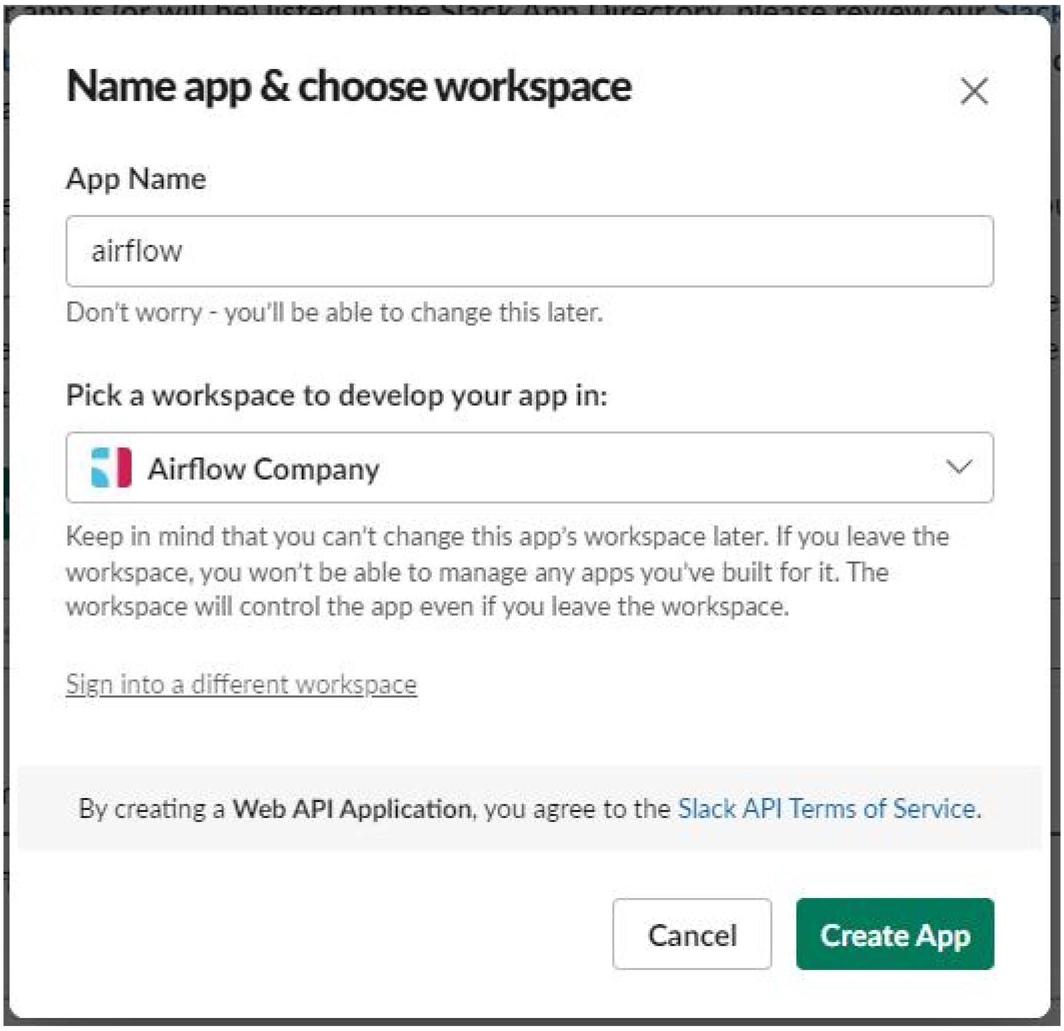
Need help? Check our documentation, or see an example

Your Apps

If your app is (or will be) listed in the Slack App Directory, please review our Slack App Directory Agreement. These terms are in addition to the existing Developer Policy, API TOS, and Brand Guidelines.

Your App Configuration Tokens
Learn about tokens

Don't see an app you're looking for? Sign in to another workspace.



Name app & choose workspace

App Name

airflow

Don't worry - you'll be able to change this later.

Pick a workspace to develop your app in:

Airflow Company

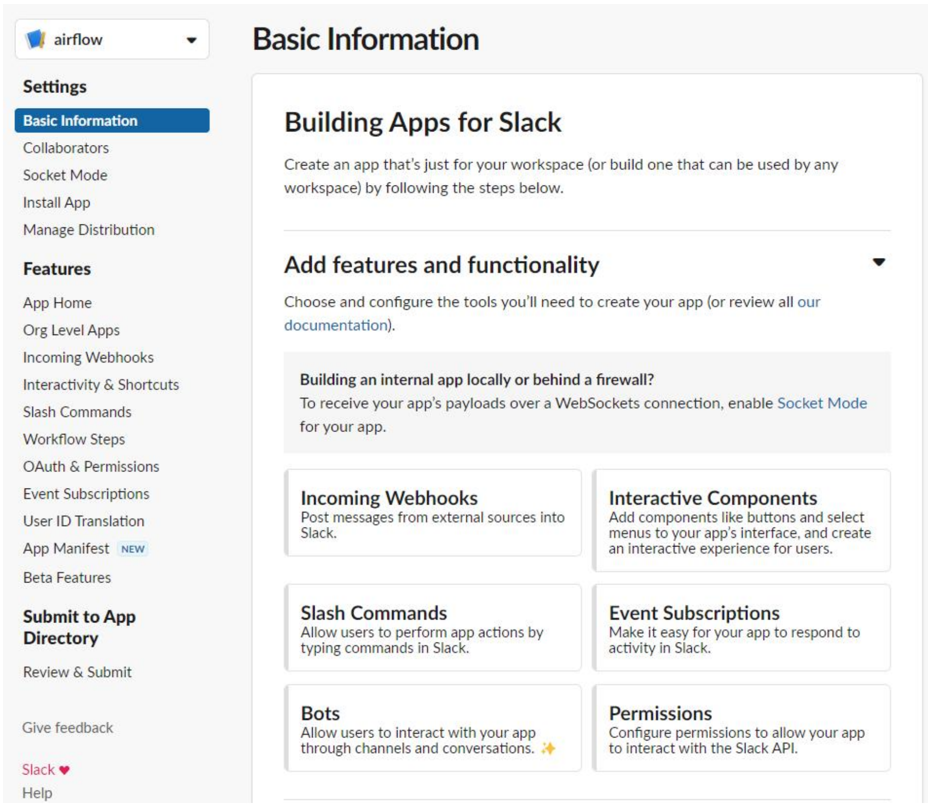
Keep in mind that you can't change this app's workspace later. If you leave the workspace, you won't be able to manage any apps you've built for it. The workspace will control the app even if you leave the workspace.

Sign into a different workspace

By creating a **Web API Application**, you agree to the Slack API Terms of Service.

Click on "Incoming Webhooks"

Incoming webhooks are a simple way to post messages from external sources into Slack.



Basic Information

Building Apps for Slack

Create an app that's just for your workspace (or build one that can be used by any workspace) by following the steps below.

Add features and functionality

Choose and configure the tools you'll need to create your app (or review all our documentation).

Building an internal app locally or behind a firewall?
To receive your app's payloads over a WebSockets connection, enable Socket Mode for your app.

Incoming Webhooks
Post messages from external sources into Slack.

Interactive Components
Add components like buttons and select menus to your app's interface, and create an interactive experience for users.

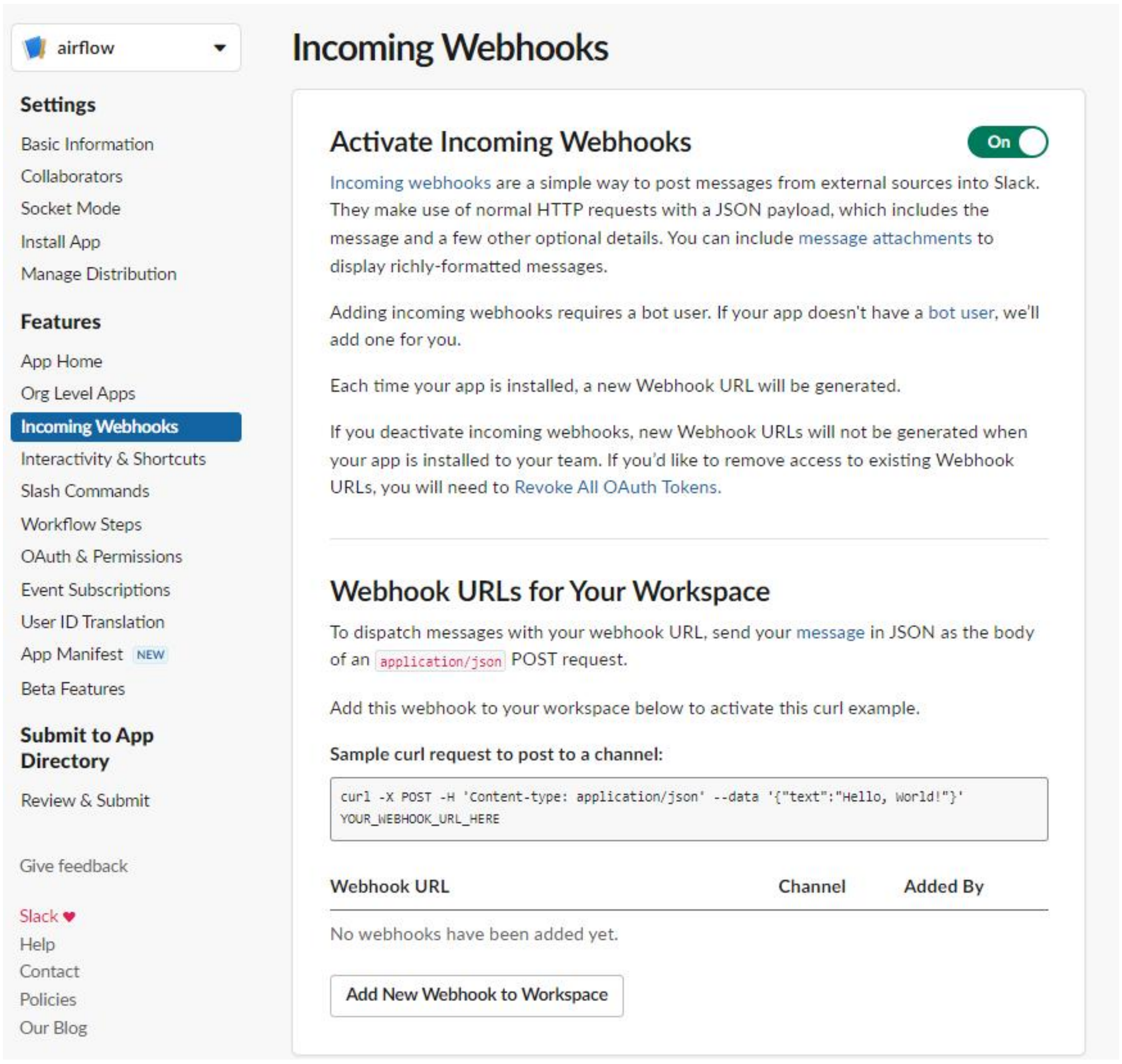
Slash Commands
Allow users to perform app actions by typing commands in Slack.

Event Subscriptions
Make it easy for your app to respond to activity in Slack.

Bots
Allow users to interact with your app through channels and conversations.

Permissions
Configure permissions to allow your app to interact with the Slack API.

Activate "Incoming Webhooks"
Click on "Add New Webhook to Workspace"



Incoming Webhooks

Activate Incoming Webhooks — On

Incoming webhooks are a simple way to post messages from external sources into Slack. They make use of normal HTTP requests with a JSON payload, which includes the message and a few other optional details. You can include message attachments to display richly-formatted messages.

Adding incoming webhooks requires a bot user. If your app doesn't have a bot user, we'll add one for you.

Each time your app is installed, a new Webhook URL will be generated.

If you deactivate incoming webhooks, new Webhook URLs will not be generated when your app is installed to your team. If you'd like to remove access to existing Webhook URLs, you will need to Revoke All OAuth Tokens.

Webhook URLs for Your Workspace

To dispatch messages with your webhook URL, send your message in JSON as the body of an `application/json` POST request.

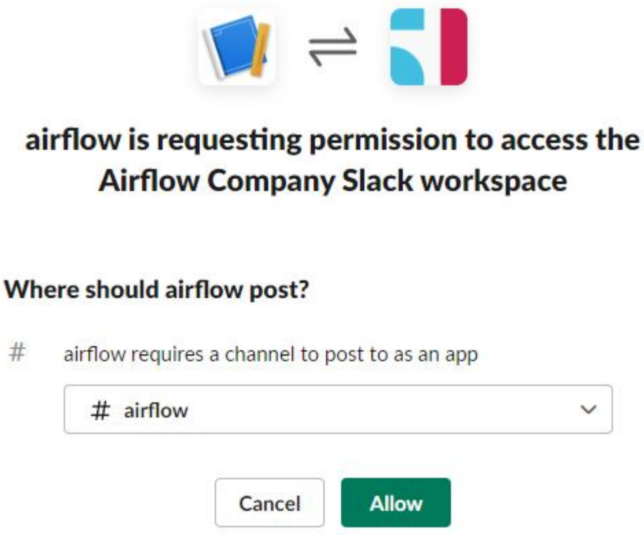
Add this webhook to your workspace below to activate this curl example.

Sample curl request to post to a channel:

```
curl -X POST -H 'Content-type: application/json' --data '{"text":"Hello, World!"}'
YOUR_WEBHOOK_URL_HERE
```

Webhook URL	Channel	Added By
No webhooks have been added yet.		

#airflow is workspace created before

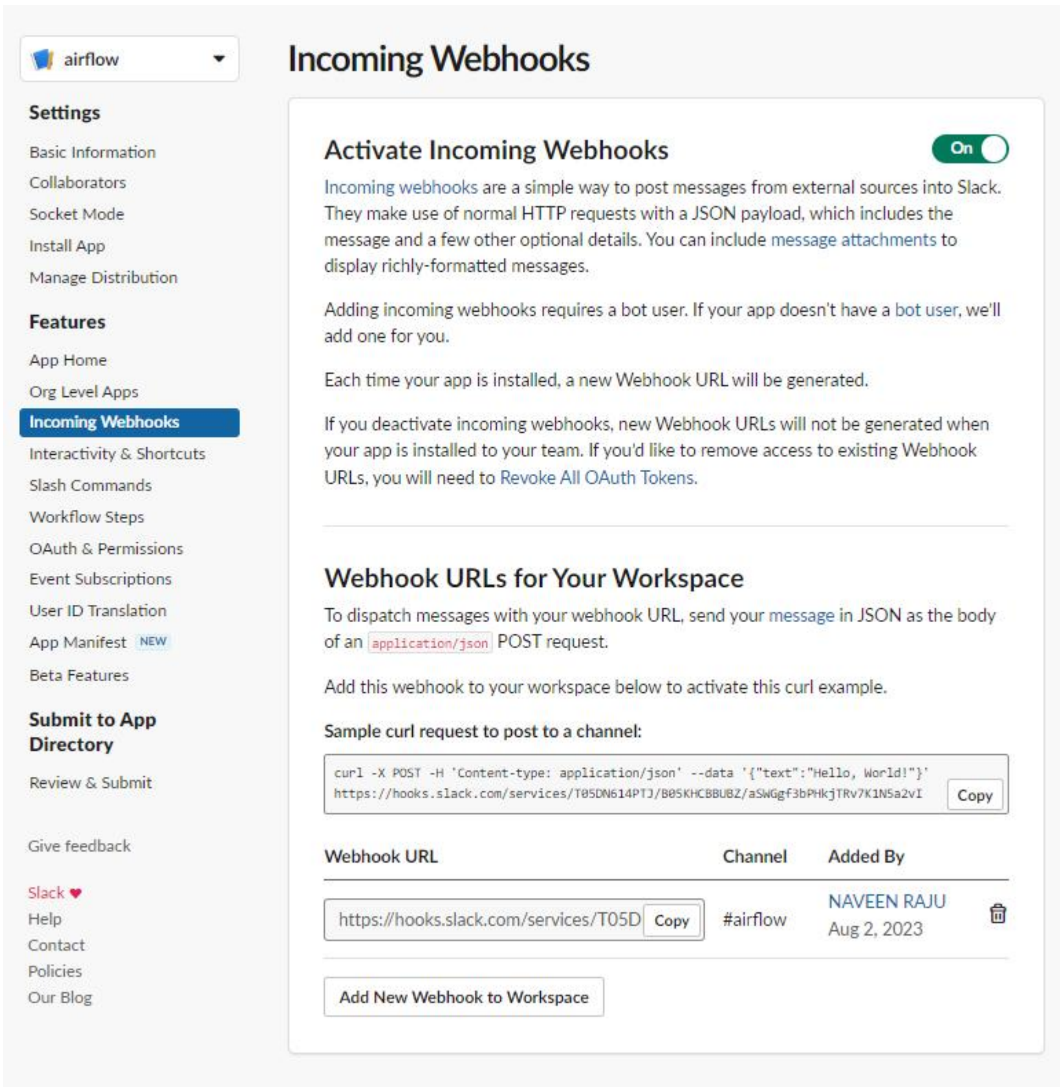

airflow is requesting permission to access the Airflow Company Slack workspace

Where should airflow post?

airflow requires a channel to post to as an app

# airflow

Copy the Webhook URL that is created



Incoming Webhooks

Activate Incoming Webhooks

On

Incoming webhooks are a simple way to post messages from external sources into Slack. They make use of normal HTTP requests with a JSON payload, which includes the message and a few other optional details. You can include message attachments to display richly-formatted messages.

Adding incoming webhooks requires a bot user. If your app doesn't have a bot user, we'll add one for you.

Each time your app is installed, a new Webhook URL will be generated.

If you deactivate incoming webhooks, new Webhook URLs will not be generated when your app is installed to your team. If you'd like to remove access to existing Webhook URLs, you will need to Revoke All OAuth Tokens.

Webhook URLs for Your Workspace

To dispatch messages with your webhook URL, send your message in JSON as the body of an `application/json` POST request.

Add this webhook to your workspace below to activate this curl example.

Sample curl request to post to a channel:

```
curl -X POST -H 'Content-type: application/json' --data '{"text":"Hello, World!"}'
https://hooks.slack.com/services/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
```
Copy

Webhook URL	Channel	Added By
https://hooks.slack.com/services/T05D Copy	#airflow	NAVEEN RAJU Aug 2, 2023

Add New Webhook to Workspace

https://hooks.slack.com/services/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

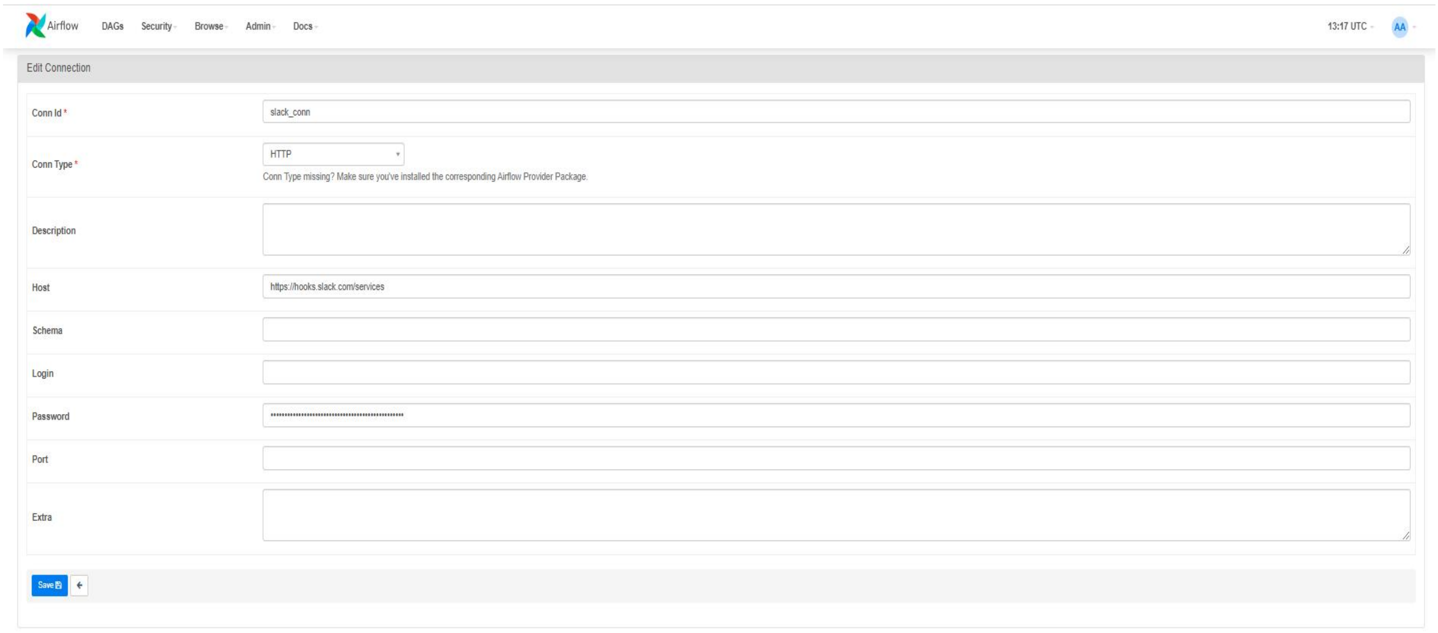
```
from airflow.providers.slack.operators.slack_webhook import SlackWebhookOperator
from airflow.hooks.base_hook import BaseHook

def _get_message() -> str:
  return "Hi from forex_data_pipeline"

send_slack_notification = SlackWebhookOperator(
        task_id="send_slack_notification",
        http_conn_id="slack_conn",
        message=_get_message(),
        channel="#airflow"
  )
```

In Airflow UI navigate to Admin - > Connections - > +

Conn Id * : slack_conn
Conn Type * : HTTP
Host : https://hooks.slack.com/services

Password : T05DN614PTJ/B05KHCBBUBZ/aSWGgf3bPHkjTRv7K1N5a2vI

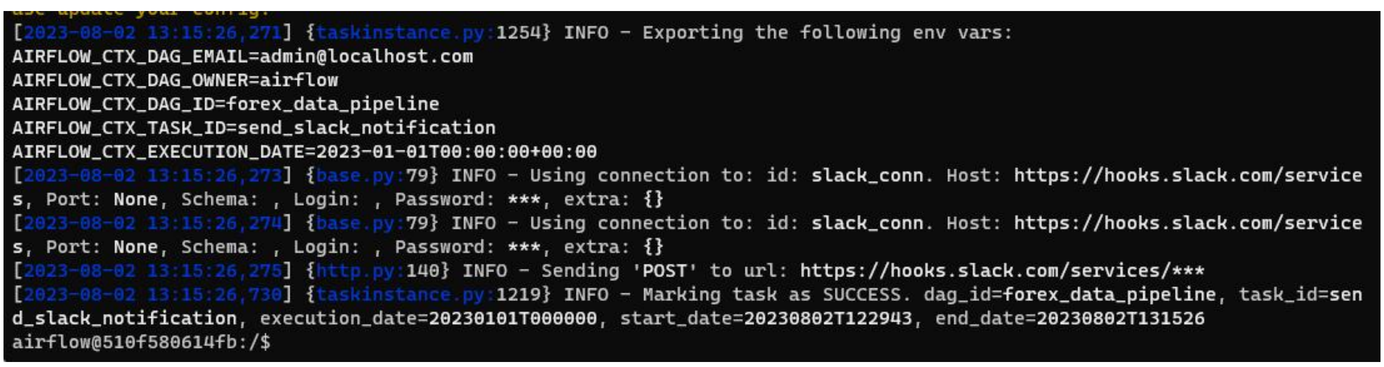
Save



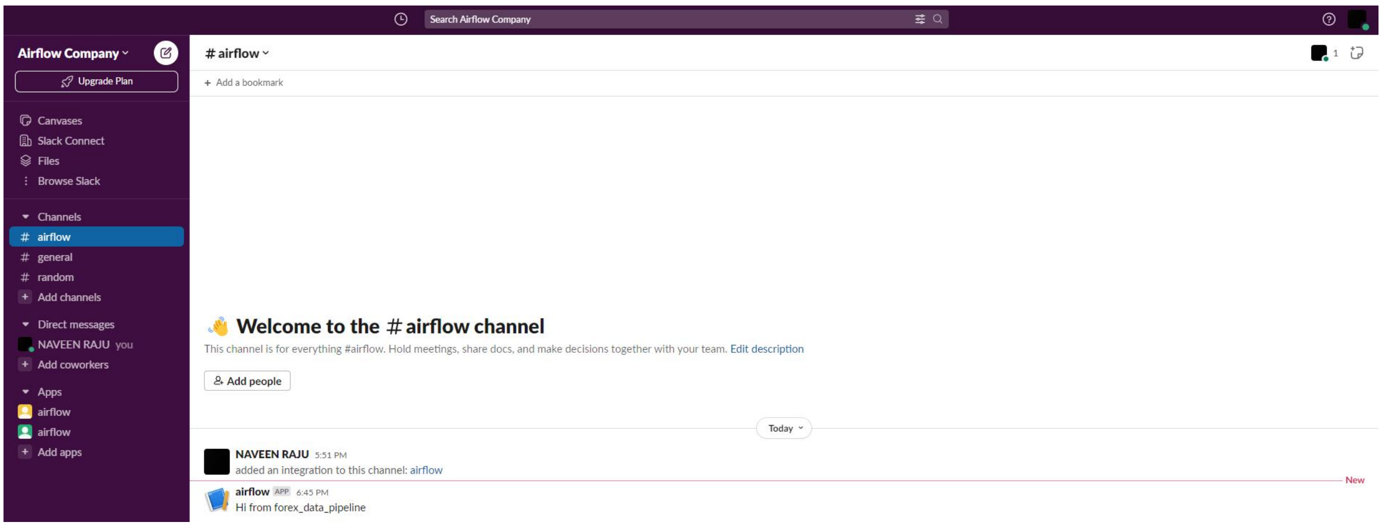
Conn Id *	slack_conn
Conn Type *	HTTP Conn Type missing? Make sure you've installed the corresponding Airflow Provider Package.
Description	
Host	https://hooks.slack.com/services
Schema	
Login	
Password	••
Port	
Extra	

In cmd:

```
airflow tasks test forex_data_pipeline send_slack_notification 2023-01-01
```



```
[2023-08-02 13:15:26,271] {taskinstance.py:1254} INFO - Exporting the following env vars:
AIRFLOW_CTX_DAG_EMAIL=admin@localhost.com
AIRFLOW_CTX_DAG_OWNER=airflow
AIRFLOW_CTX_DAG_ID=forex_data_pipeline
AIRFLOW_CTX_TASK_ID=send_slack_notification
AIRFLOW_CTX_EXECUTION_DATE=2023-01-01T00:00:00+00:00
[2023-08-02 13:15:26,273] {base.py:79} INFO - Using connection to: id: slack_conn. Host: https://hooks.slack.com/service
s, Port: None, Schema: , Login: , Password: ***, extra: {}
[2023-08-02 13:15:26,274] {base.py:79} INFO - Using connection to: id: slack_conn. Host: https://hooks.slack.com/service
s, Port: None, Schema: , Login: , Password: ***, extra: {}
[2023-08-02 13:15:26,275] {http.py:140} INFO - Sending 'POST' to url: https://hooks.slack.com/services/***
[2023-08-02 13:15:26,730] {taskinstance.py:1219} INFO - Marking task as SUCCESS. dag_id=forex_data_pipeline, task_id=sen
d_slack_notification, execution_date=20230101T000000, start_date=20230802T122943, end_date=20230802T131526
airflow@510f580614fb:/$
```



👋 Welcome to the #airflow channel

This channel is for everything #airflow. Hold meetings, share docs, and make decisions together with your team. Edit description

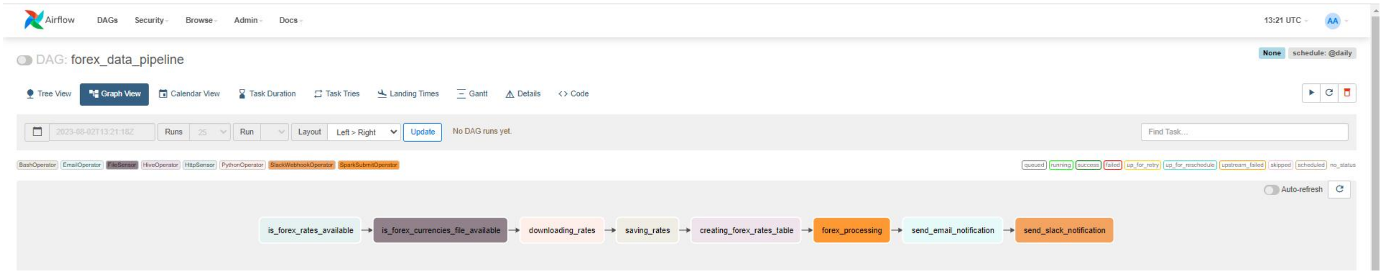
NAVEEN RAJU 5:51 PM
added an integration to this channel: airflow

airflow APP 6:45 PM
Hi from forex_data_pipeline

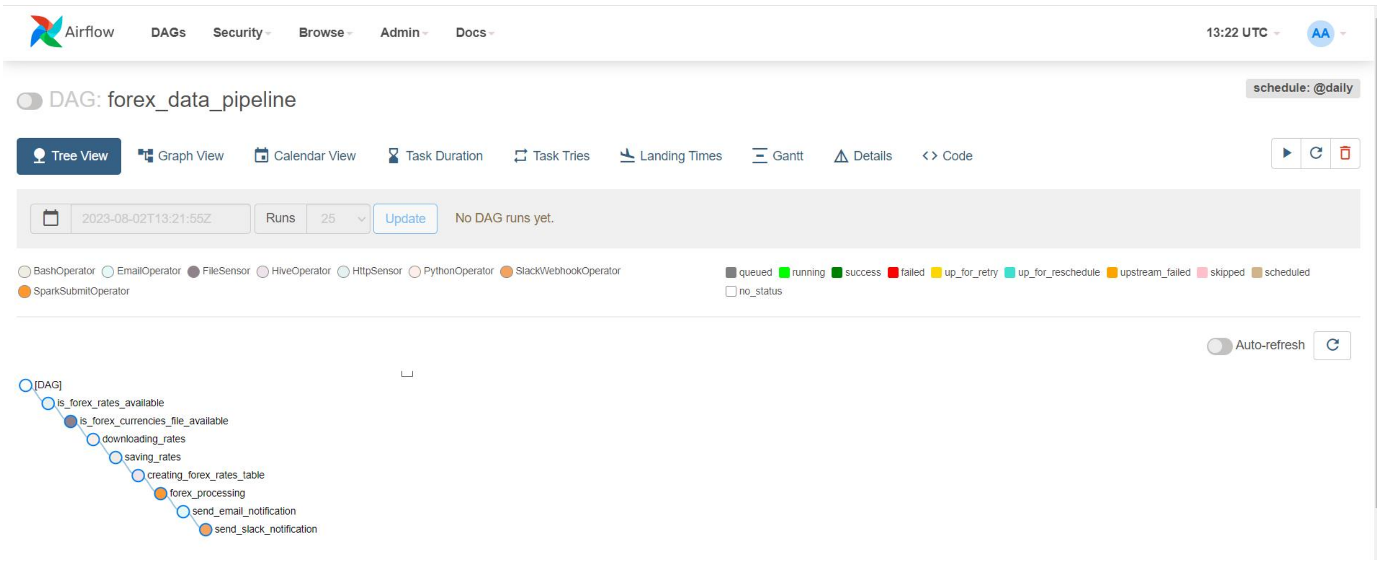
K) Add dependency between tasks

```
is_forex_rates_available >> is_forex_currencies_file_available >> downloading_rates >> saving_rates
saving_rates >> creating_forex_rates_table >> forex_processing
forex_processing >> send_email_notification >> send_slack_notification
```

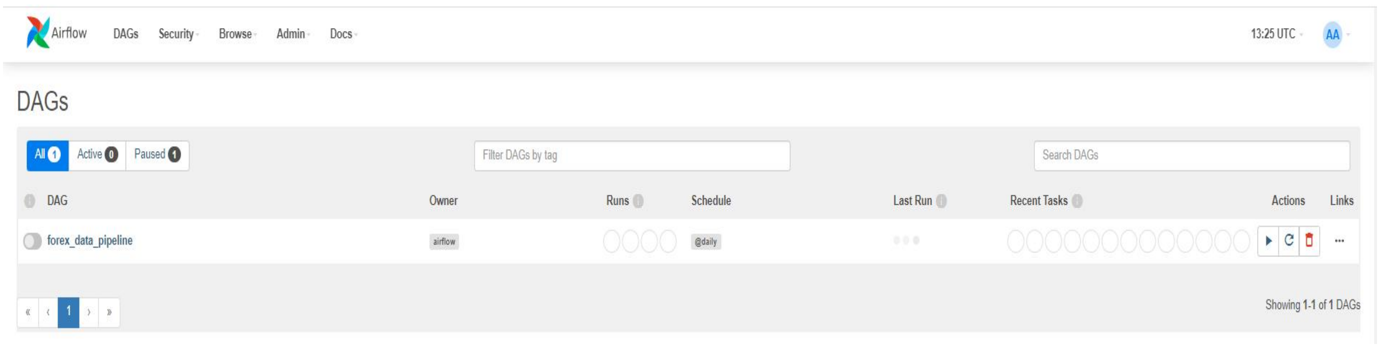
Graph view :

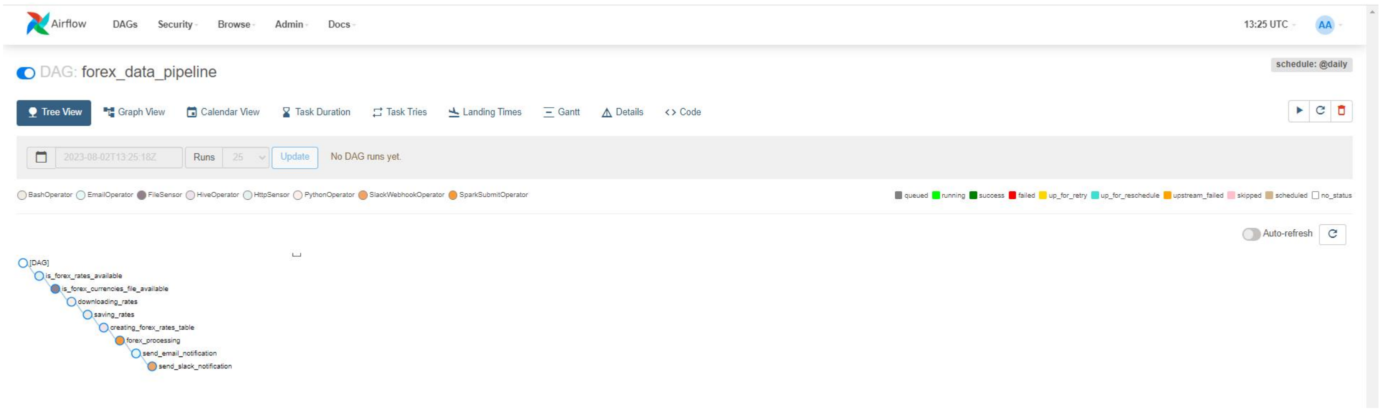


Tree view:



L) Trigger a DAG from airflow UI:





DAG: forex_data_pipeline

2023-08-02T13:25:18Z Runs 25 Update No DAG runs yet.

○ BashOperator ○ EmailOperator ● FileSensor ○ HiveOperator ○ HttpSensor ○ PythonOperator ● SlackWebhookOperator ● SparkSubmitOperator

▨ queued 🟩 running 🟩 success 🟥 failed 🟨 up_for_retry 🟦 up_for_reschedule 🟧 upstream_failed ▨ skipped ▨ scheduled ☐ no_status

```
○ [DAG]
  ○ is_forex_rates_available
    ● is_forex_currencies_file_available
      ○ downloading_rates
        ○ saving_rates
          ○ creating_forex_rates_table
            ● forex_processing
              ○ send_email_notification
                ● send_slack_notification
```