**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**
2. **Install Docker for Windows**
3. **Install Airflow steps**
4. **Docker**
5. **Flow chart of Forex Data pipeline**
6. **Architecture back end details**

**7) Pipeline code**

**A) Create a Gist in our Github account**

1. **Define DAG**

**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**

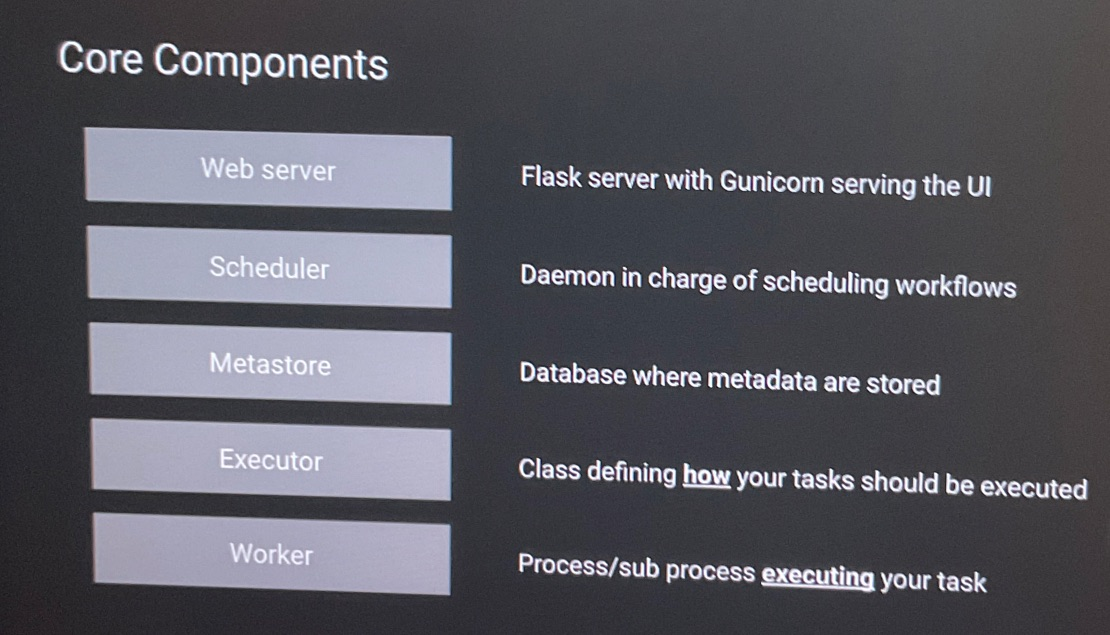
1. **Create task named “if currency file is available” - File sensor and check**
2. **Create task named “downloading\_rates” to download the forex rates from API - Python operator and check**
3. **Create a task named “forex\_rates” - Bash operator to save the forex rates into HDFS and check it**
4. **Create a task named “creating\_forex\_rates\_table” - Hive operator and check it**
5. **Process the forex rates with Spark - Spark submit operator**
6. **Send Email notifications - Email Operator**
7. **Send Slack notifications - SlackWebHookOperator**
8. **Add dependency between tasks**
9. **Trigger a DAG from airflow UI:**
10. **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 :



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.

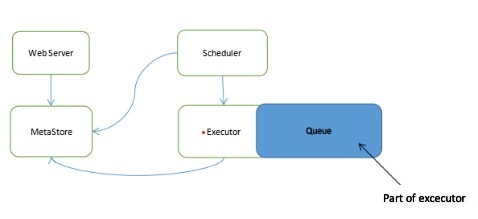
One - node Architecture:

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.

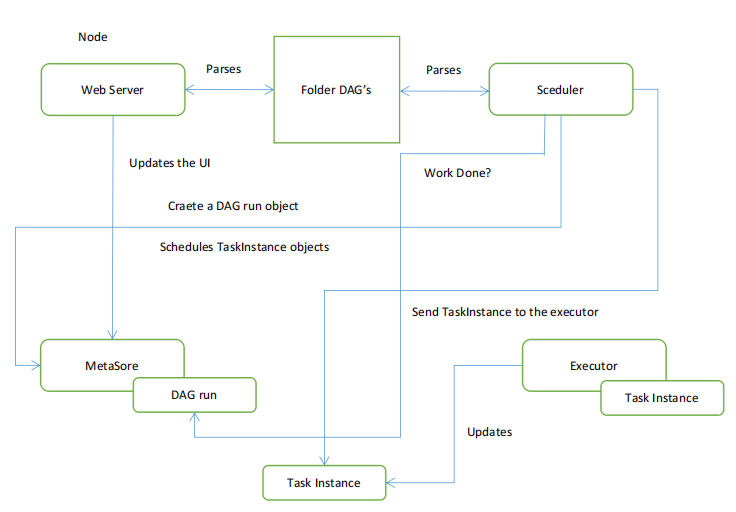
Finally, the executor interacts also with the meta in order to update the tasks that just have been completed.

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.



Steps of DAG run in Apache Airflow:

* 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.

1. **Install Docker for Windows**

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

1. **Install Airflow steps**

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

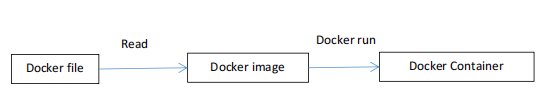
docker image ls

\*to list all docker image

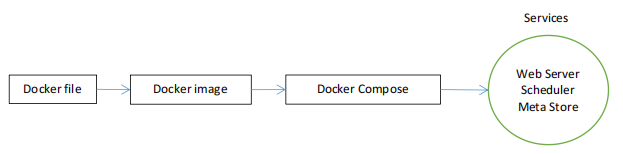
docker run --rm -d -p 8080:8080 airflow-basic

\*Airflow UI is running inside docker container, thus to access Airflow UI we should connect Docker container port 8080 to our machine port 8080

1. **Docker**



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

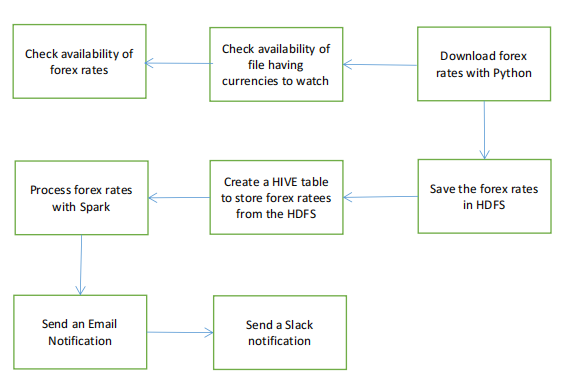


Docker Compose : is a tool provided by Docker that allows to define and manage multi containers. Networks and volumes as a single application, making it easier to orchestrate and deploy complex containerized applications.

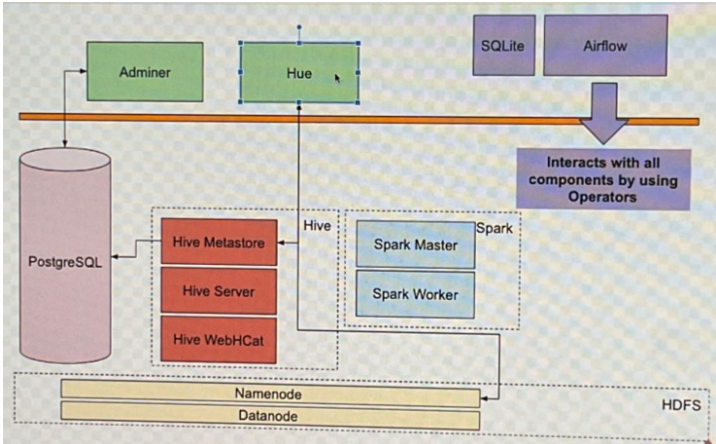
In Airflow we have 3 main components MetaStore, Scheduler and Web server where each run in 3 different container managed by docker compose as single application.

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

1. **Flow chart of Forex Data pipeline**



1. **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.

1. **Pipeline code:**

**A) Create a Gist in our Github account**

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,"CHF":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  ) |

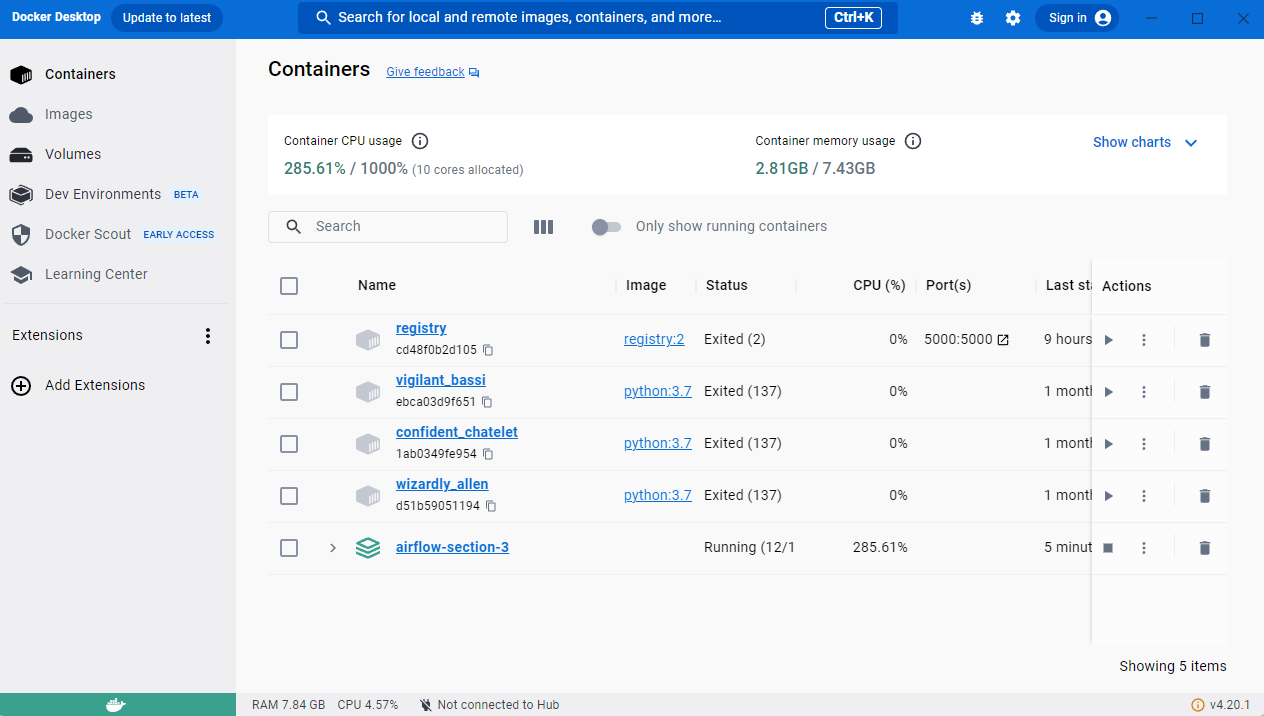
In current working directory of the pipeline code all command prompt code should be run:

|  |
| --- |
| 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



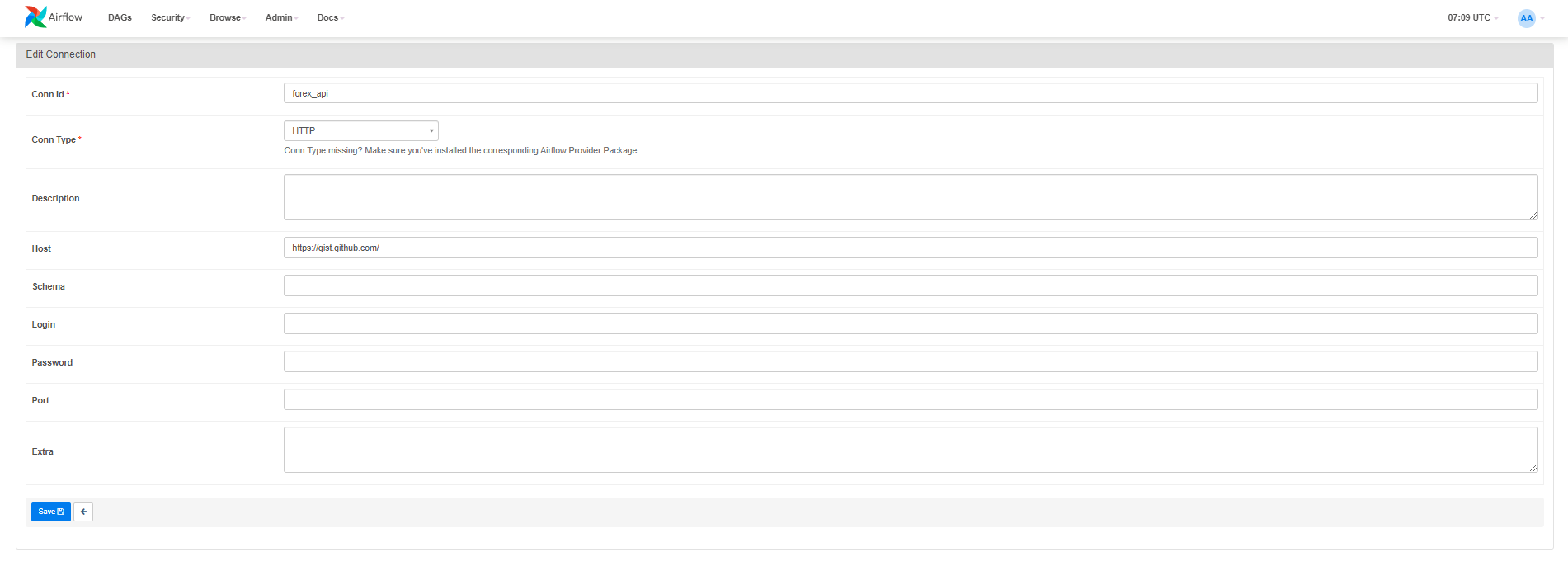
In browser type locathost:8080

Username : airflow

Password : airflow

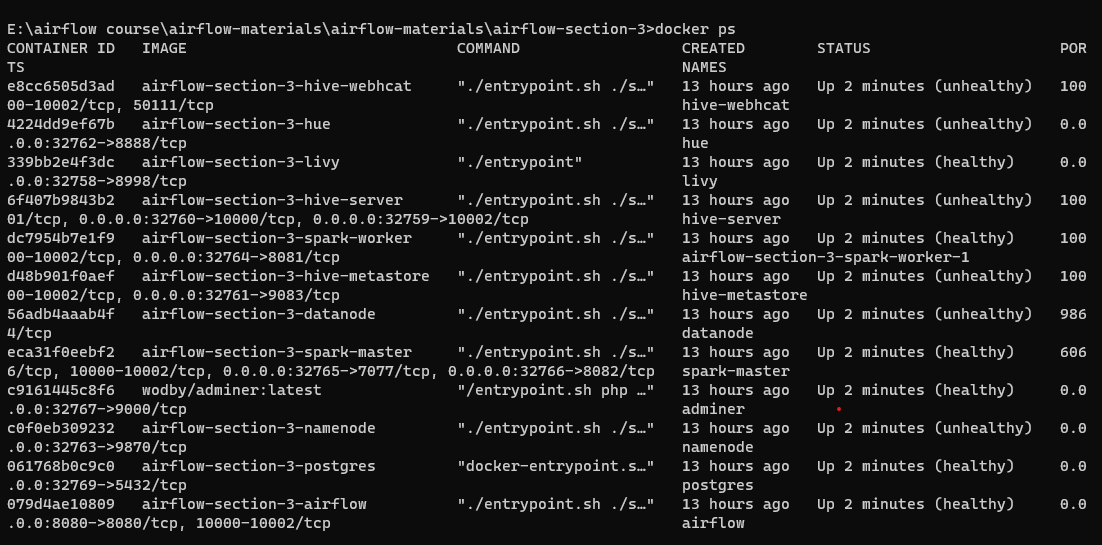
On Airflow UI navigate to Admin - > Connections -> +

|  |
| --- |
| Conn Id \* : forex\_api  Conn Type \* : HTTP  Host : <https://gist.github.com/>  Save |



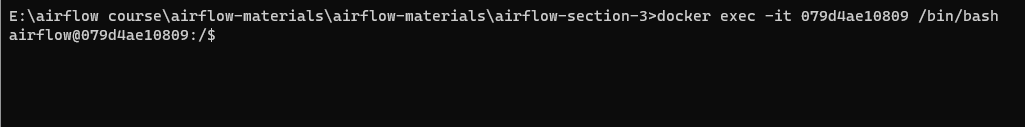
|  |
| --- |
| In cmd type : docker ps |

Copy container id of airflow-section-3-airflow



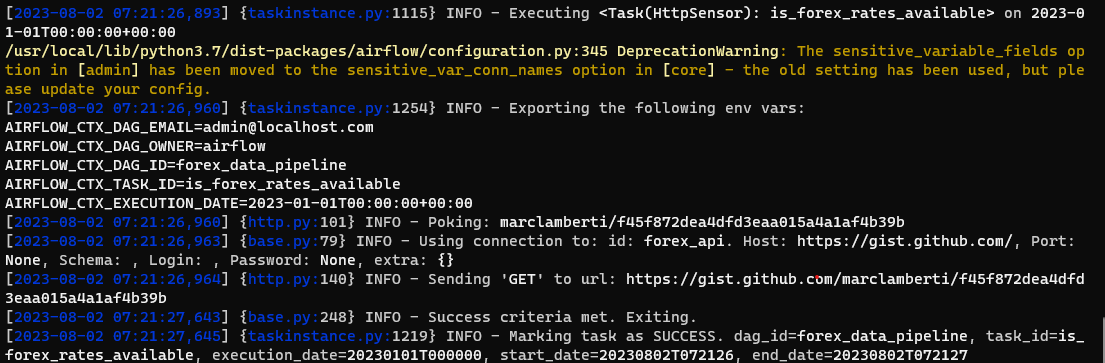
|  |
| --- |
| In cmd type : docker exec -it 079d4ae10809 /bin/bash |

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



Check if task “is\_forex\_rates\_available” of the DAG “forex\_data\_pipeline” is working

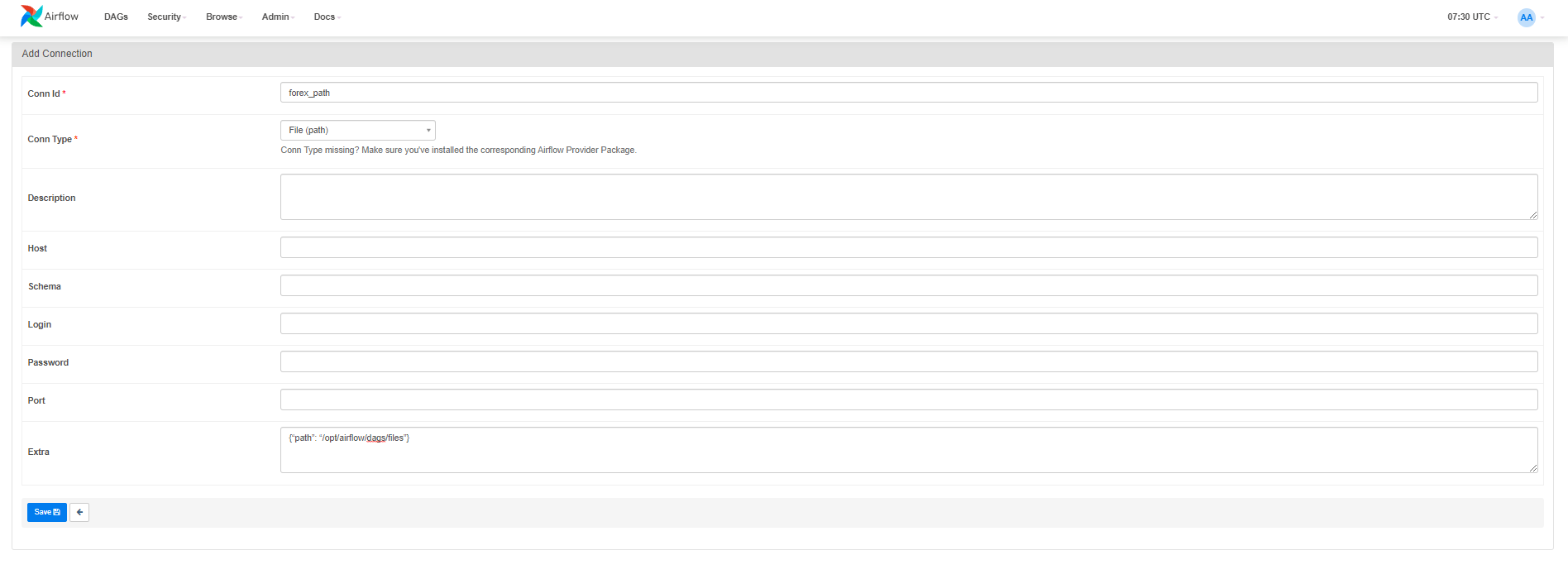
|  |
| --- |
| In cmd type: airflow tasks test forex\_data\_pipeline is\_forex\_rates\_available 2023-01-01 |



**D) Create task named “if currency file is available” - File sensor and check:**

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 |



Write File sensor code in the DAG

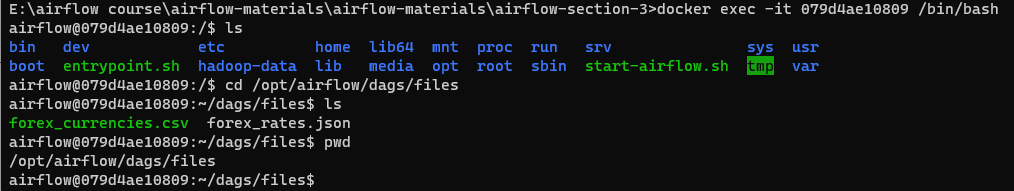
|  |
| --- |
| from airflow.sensors.filesystem import FileSensor  is\_forex\_currencies\_file\_available = FileSensor(  task\_id="is\_forex\_currencies\_file\_available",  fs\_conn\_id="forex\_path",  filepath="forex\_currencies.csv",  poke\_interval=5,  timeout=20  ) |

‘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.

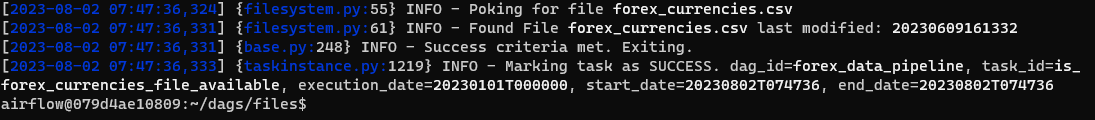
|  |
| --- |
| In cmd:  docker exec -it 079d4ae10809 /bin/bash  ls  cd /opt/airflow/dags/files  ls  pwd |



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 |

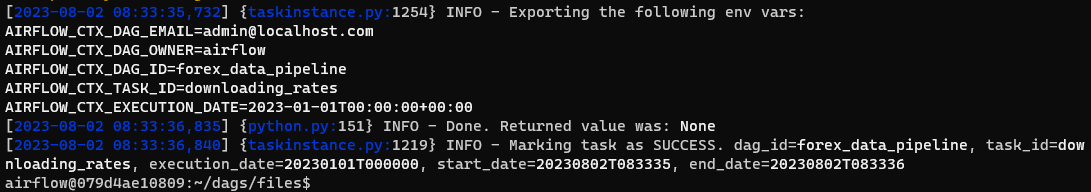


**E) Create task named “downloading\_rates” to download the forex rates from API - Python operator and check**

|  |
| --- |
| 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  ) |

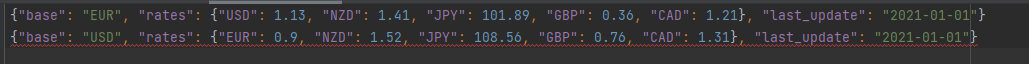
Test if the task “downloading\_rates” is working:

|  |
| --- |
| In cmd type: airflow tasks test forex\_data\_pipeline downloading\_rates 2023-01-01 |



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:



**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

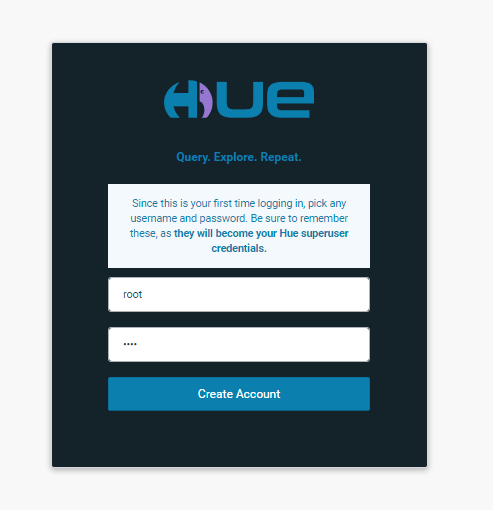
Password : root

Restart docker container if HUE port is not working

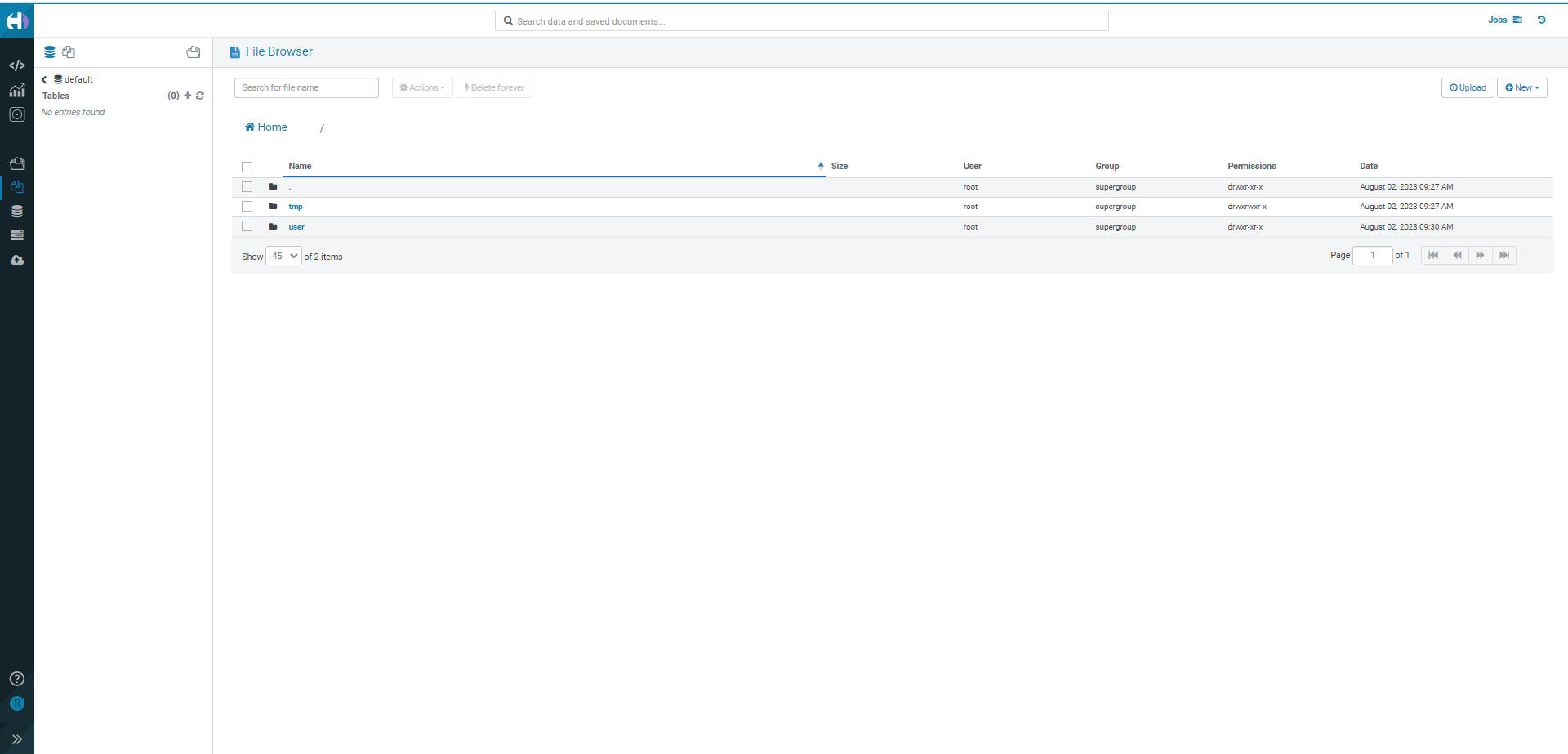
|  |
| --- |
| In cmd:  restart.sh |

restart.sh

|  |
| --- |
| #!/bin/bash  ./stop.sh  ./start.sh |



In HUE UI navigate to files to see files in HDFS

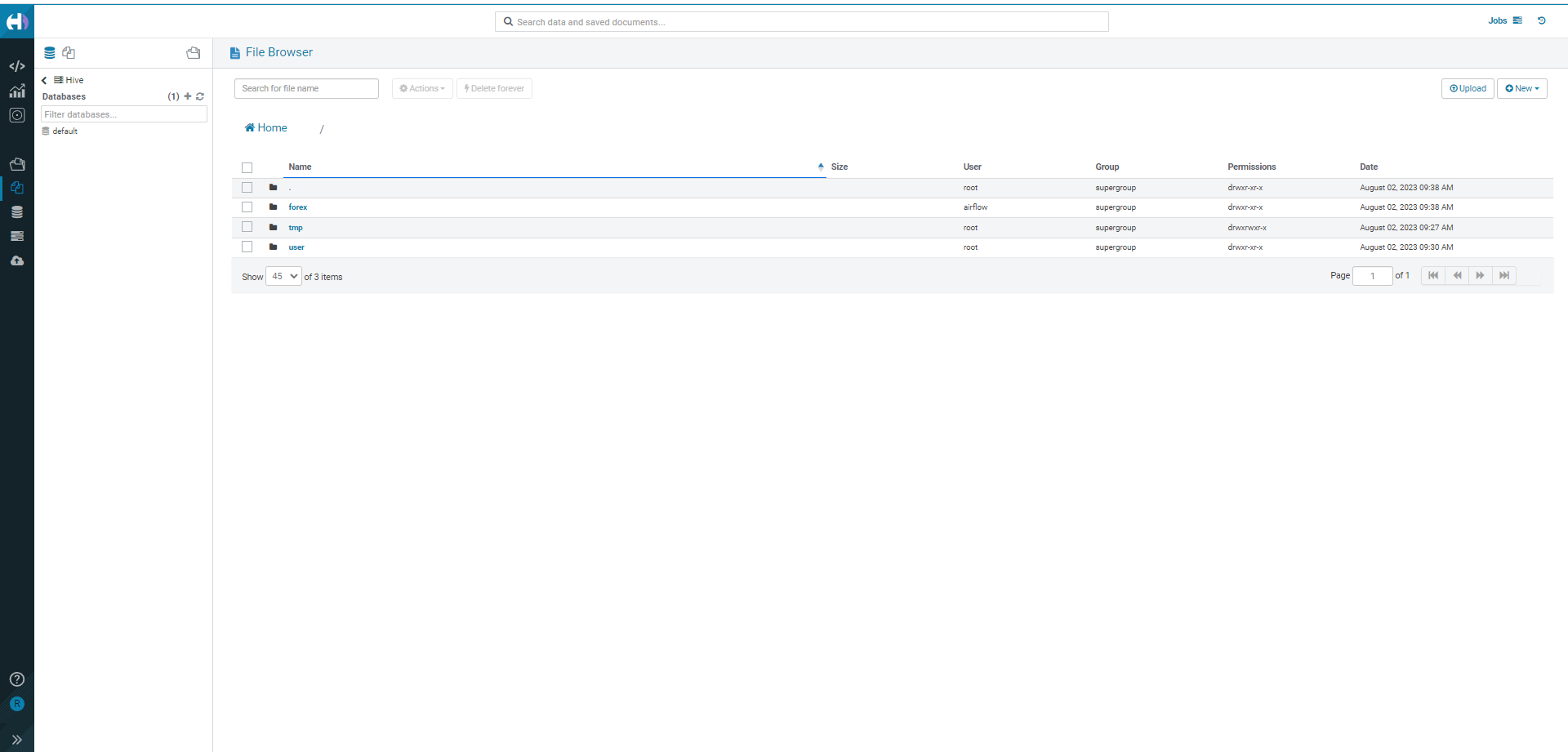


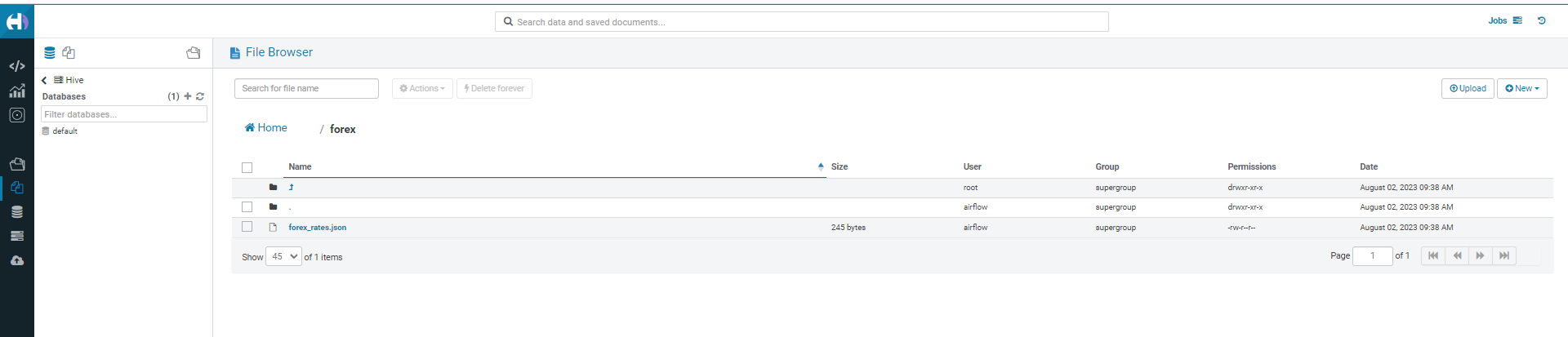
|  |
| --- |
| In cmd:  docker ps # copy container id of airflow-section-3-airflow  docker exec -it 510f580614fb /bin/bash |

Test if the task “saving\_rates” is working:

|  |
| --- |
| 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.





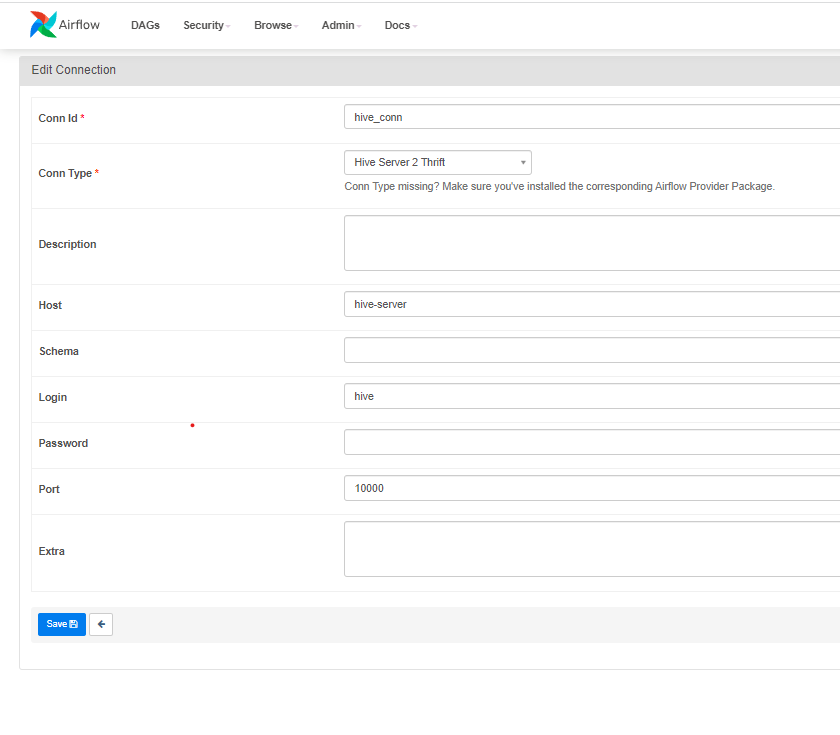
1. **Create a task named “creating\_forex\_rates\_table” - Hive operator and check it**

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  """  ) |

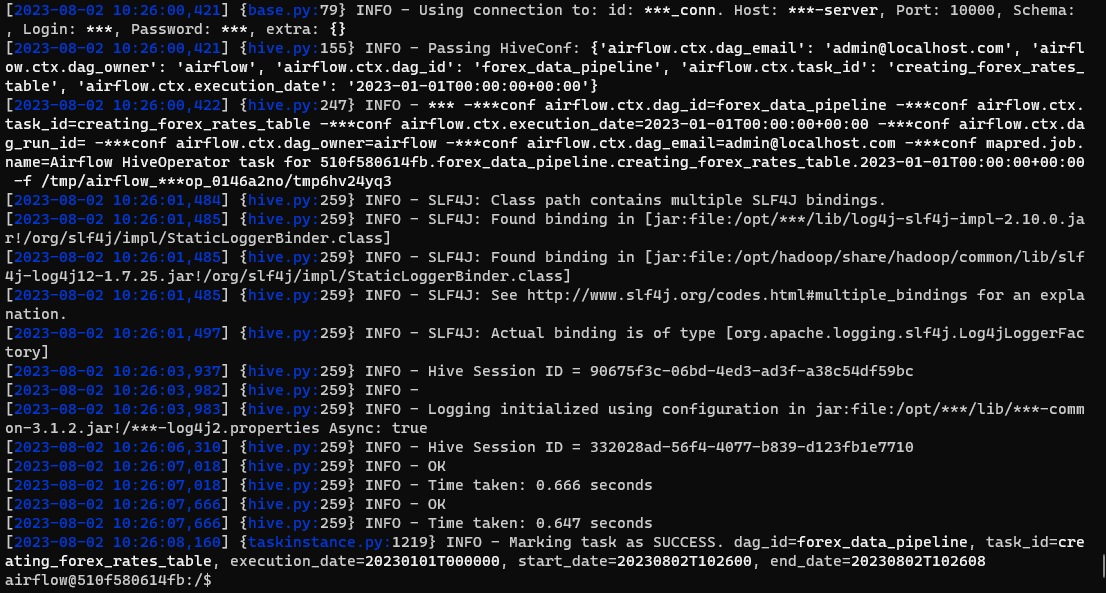
In Airflow UI page navigate to Admin - > connections - > +

|  |
| --- |
| Conn Id \* : hive\_conn  Conn Type \* : Hive Server 2 Thrift  Host : hive-server  Login : hive  Password : hive  Port : 10000  Save |

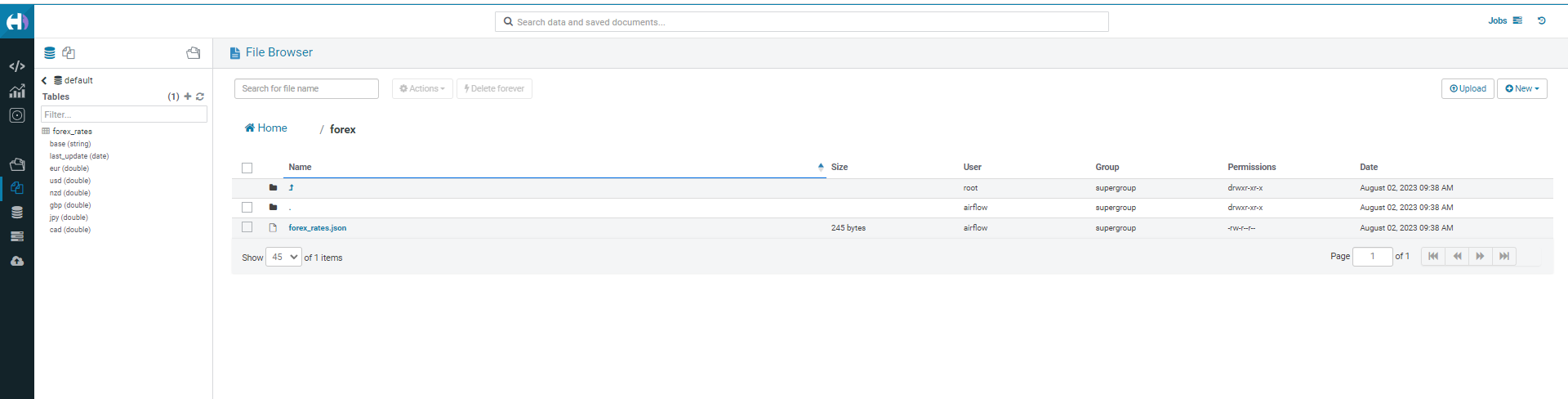


Test the task:

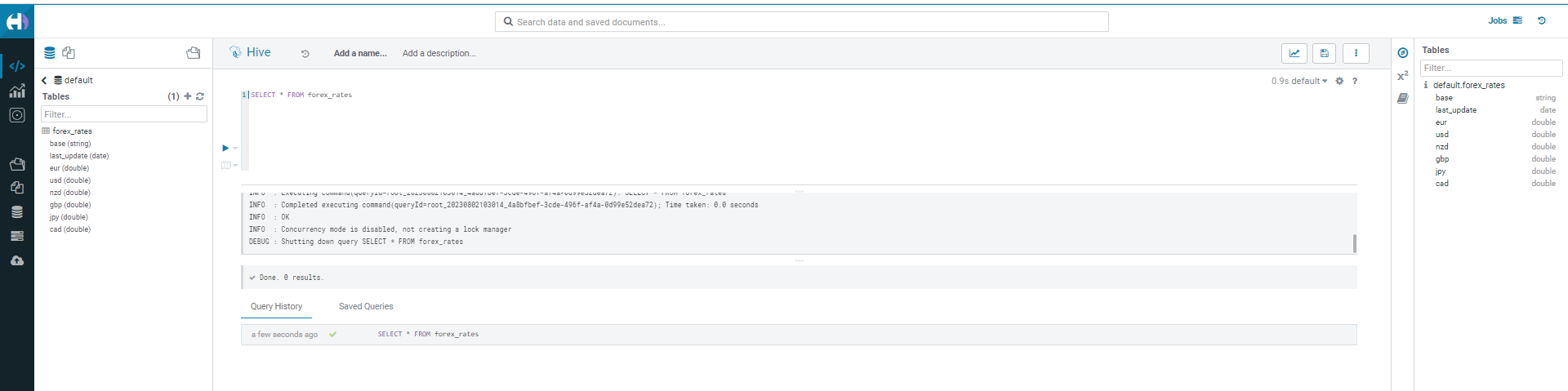
|  |
| --- |
| airflow tasks test forex\_data\_pipeline creating\_forex\_rates\_table 2023-01-01 |



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



No we try to write query in HUE



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

1. **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  ) |

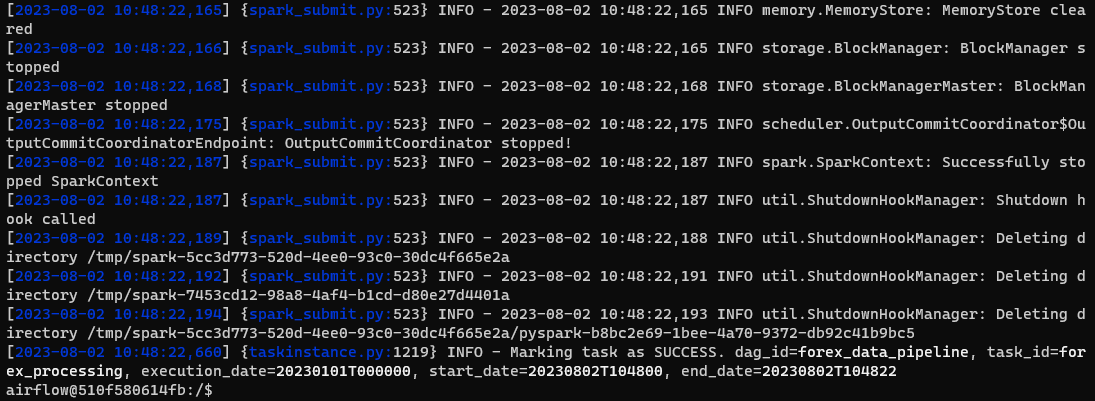
The below program basically creates a Spark session, read forex\_rates.json file and then do some pre processing on it and the finally insert it into forex\_rates table.

|  |
| --- |
| 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 - > +

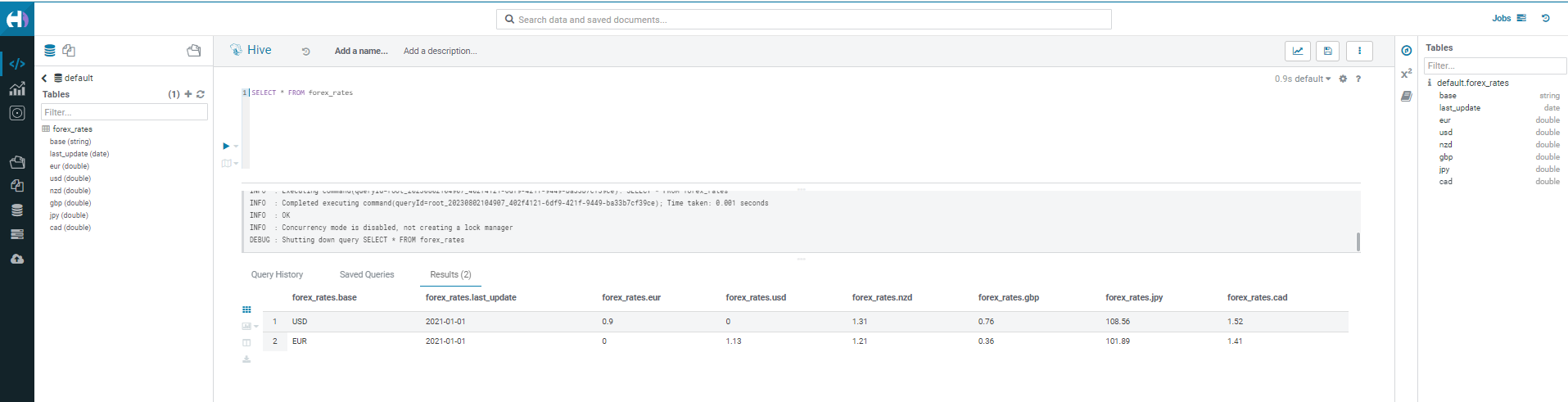
|  |
| --- |
| Conn Id \* : spark\_conn  Conn Type \* : Spark  Host : spark://spark-master  Port : 7077  Save |

|  |
| --- |
| In cmd:  airflow tasks test forex\_data\_pipeline forex\_processing 2023-01-01 |



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

SELECT \* FROM forex\_rates



1. **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

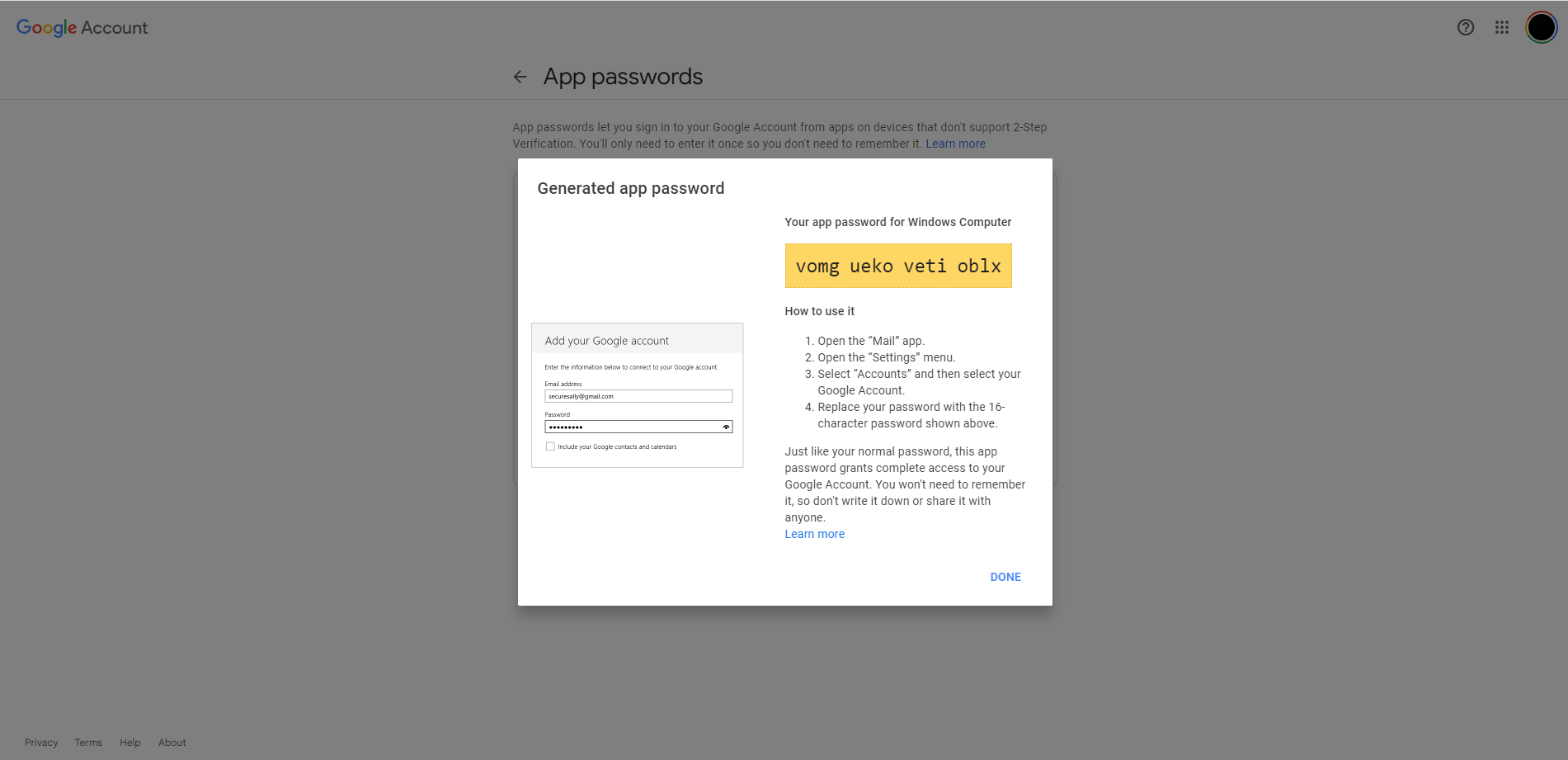
App passwords let you sign in to your Google Account from apps on devices that don't support 2-Step Verification.

Select App : Mail

Select device : Windows computer

Click on Generate

Click on Done



Configure SMTP in airflow.cfg located in “/mnt/airflow/”

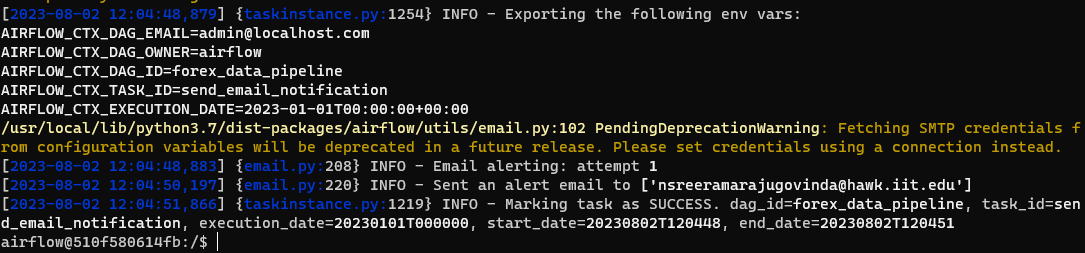
|  |
| --- |
| [smtp]  # If you want airflow to send emails on retries, failure, and you want to use  # the airflow.utils.email.send\_email\_smtp function, you have to configure an  # smtp server here  smtp\_host = smtp.gmail.com  smtp\_starttls = True  smtp\_ssl = False  # Example: smtp\_user = airflow  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 |

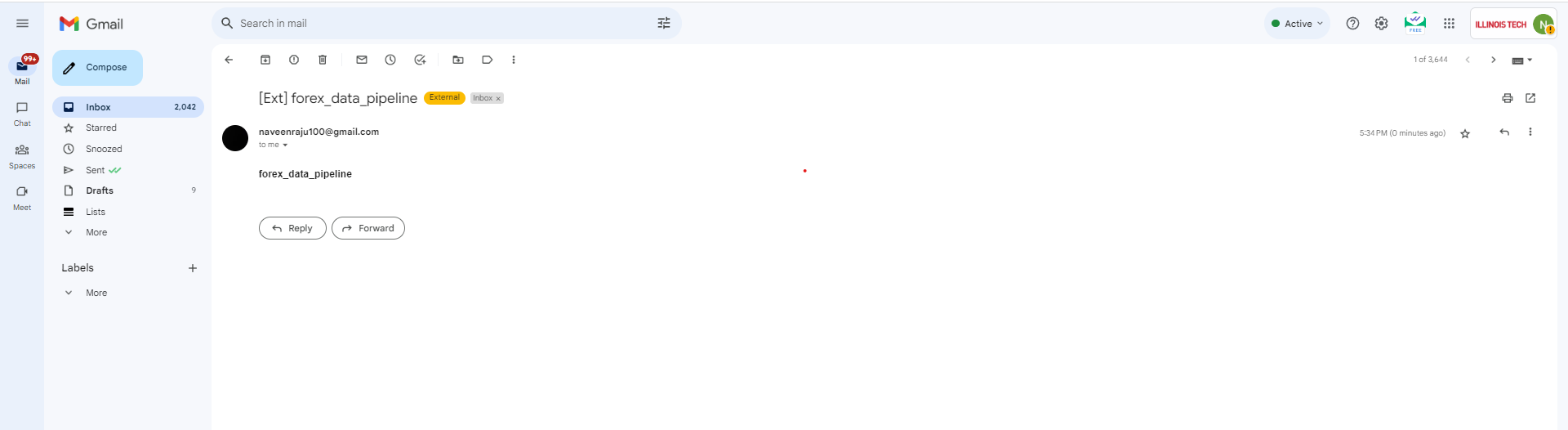
Restart Docker conainer

|  |
| --- |
| 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 |





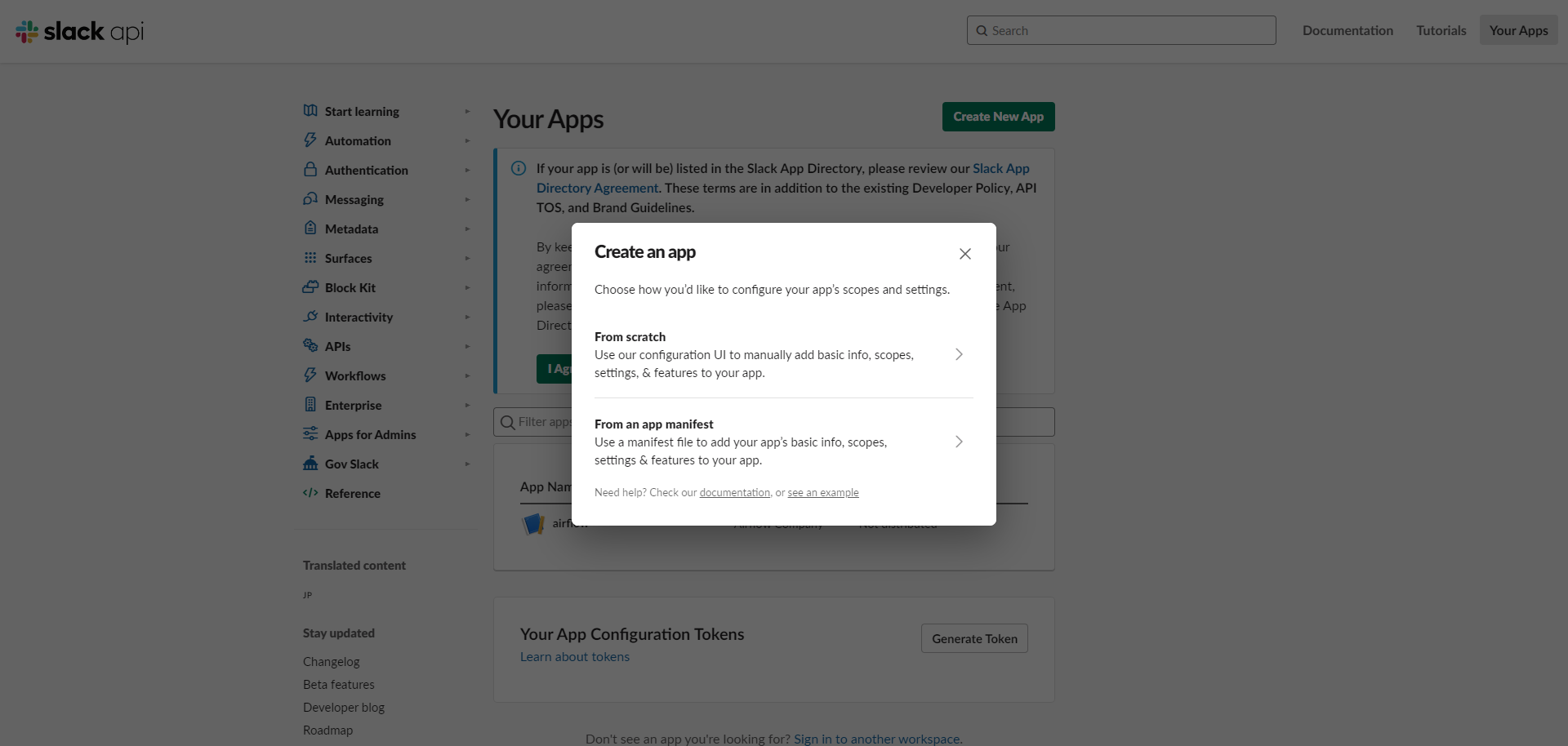
1. **Send Slack notifications - SlackWebHookOperator**

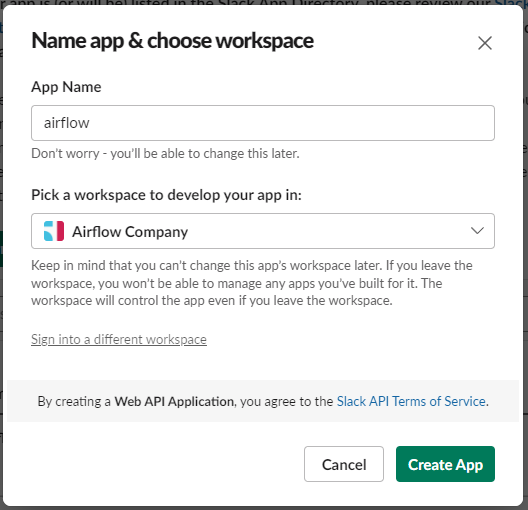
Create a new a new Slack Workspace named “Airflow Company”

In the browser browse <https://api.slack.com/apps> and click “Create an App”

Select “From an app manifest”

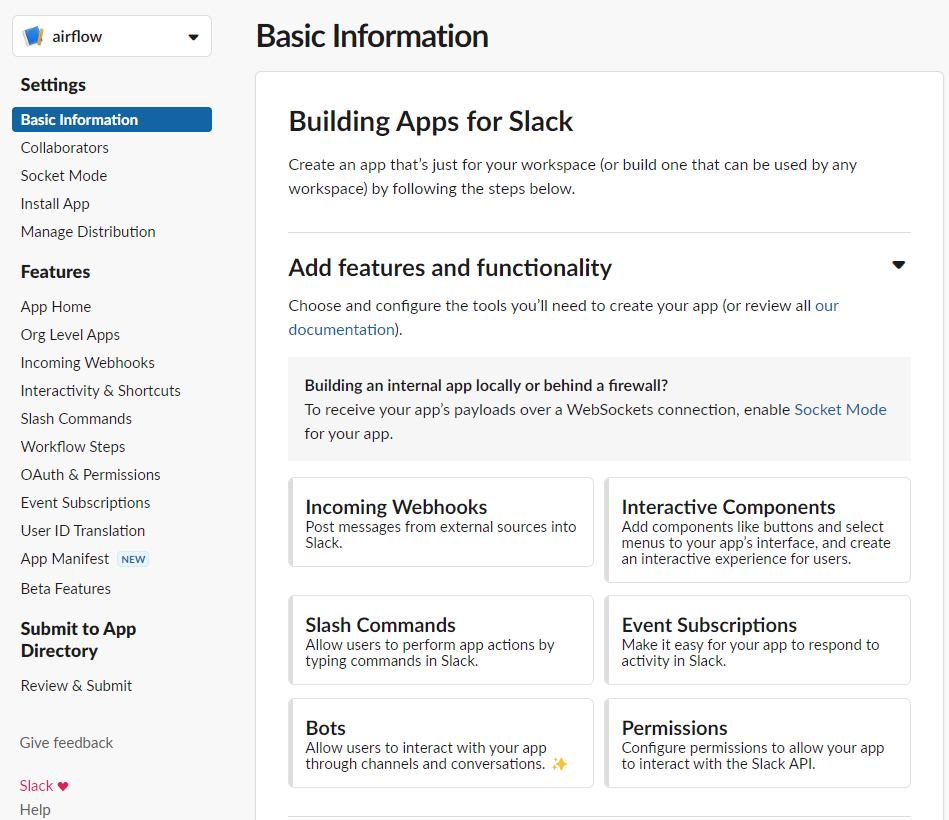
Select “From Scratch”





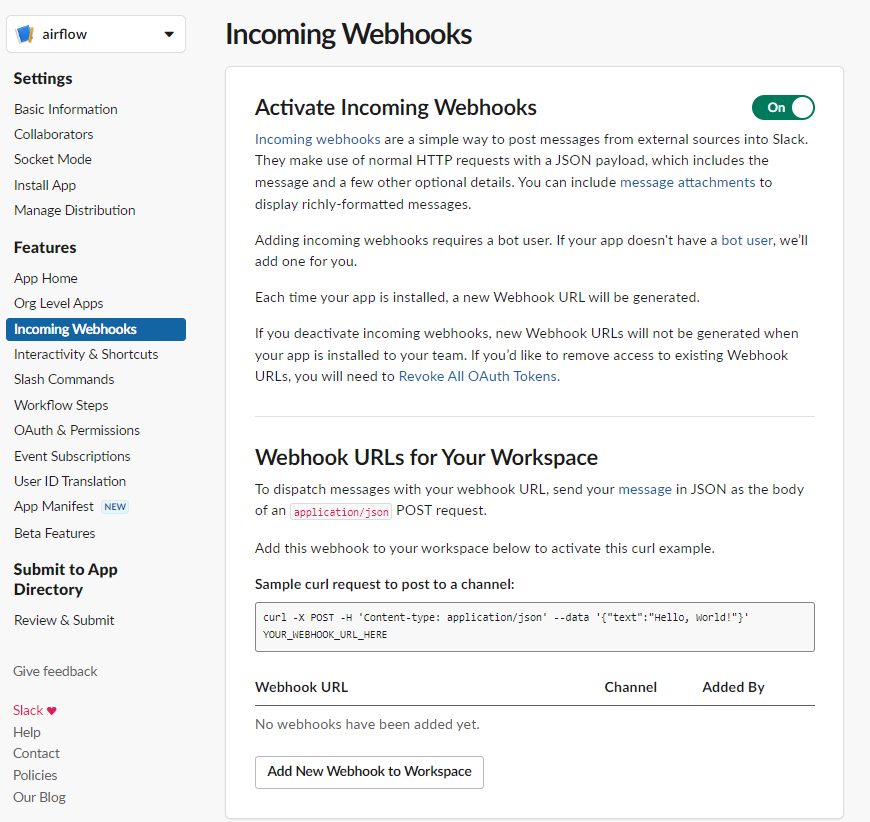
Click on “Incoming Webhooks”

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

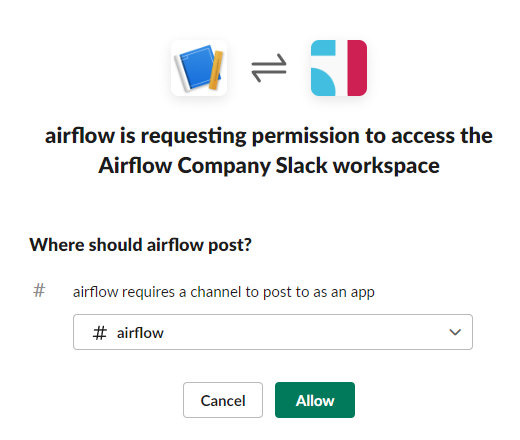


Activate “Incoming Webhooks”

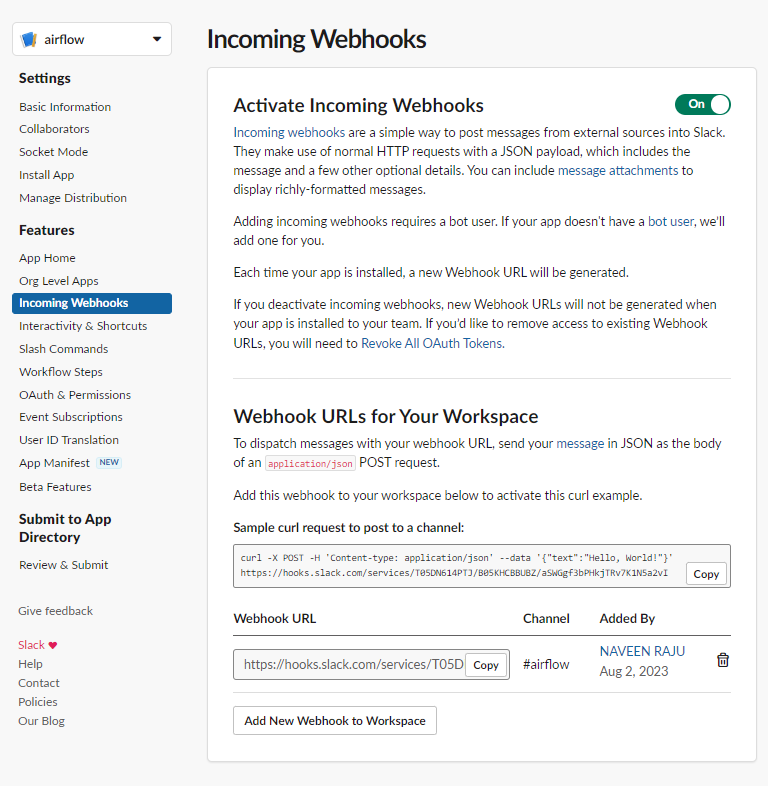
Click on “Add New Webhook to Workspace”



#airflow is workspace created before



Copy the Webhook URL that is created

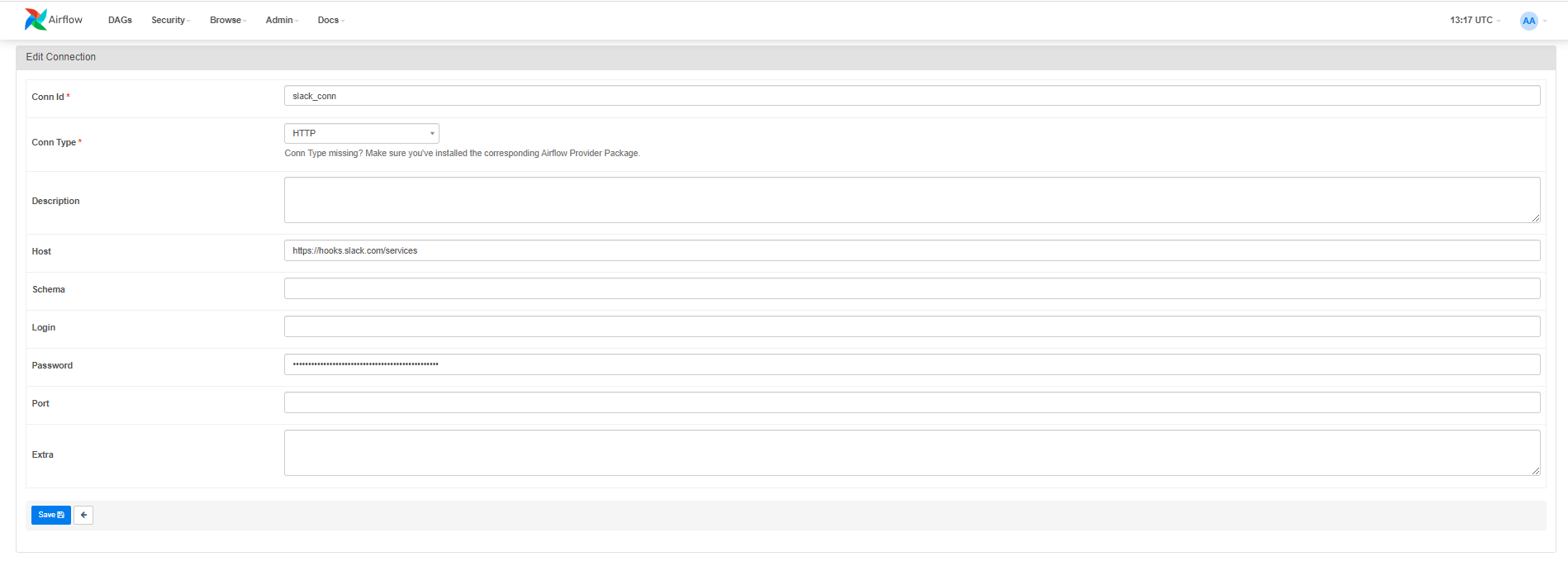


<https://hooks.slack.com/services/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>

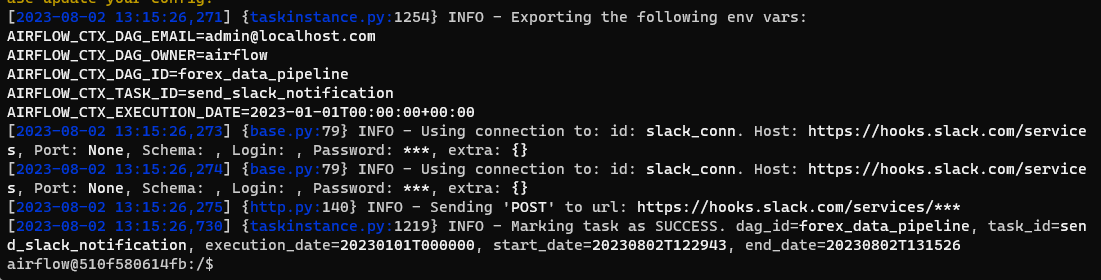
|  |
| --- |
| from airflow.providers.slack.operators.slack\_webhook import SlackWebhookOperator  from airflow.hooks.base\_hook import BaseHook  def \_get\_message() -> str:  return "Hi from forex\_data\_pipeline"  send\_slack\_notification = SlackWebhookOperator(  task\_id="send\_slack\_notification",  http\_conn\_id="slack\_conn",  message=\_get\_message(),  channel="#airflow"  ) |

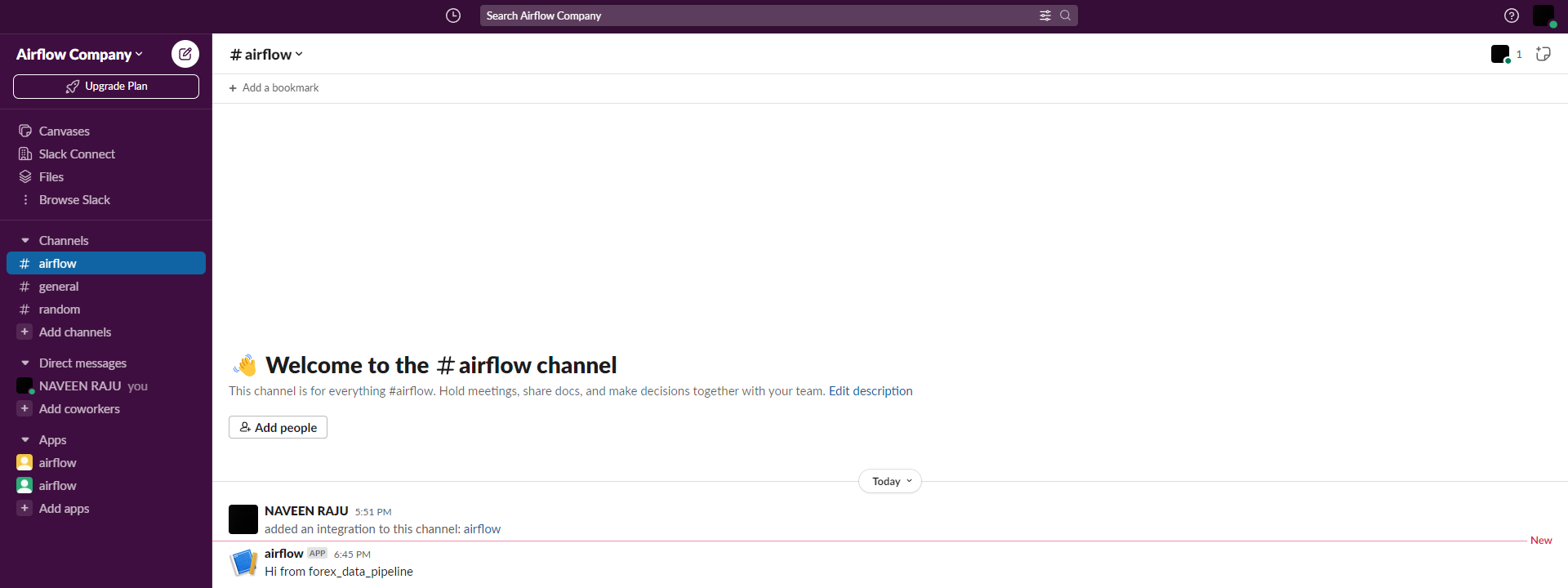
In Airflow UI navigate to Admin - > Connections - > +

|  |
| --- |
| Conn Id \* : slack\_conn  Conn Type \* : HTTP  Host : https://hooks.slack.com/services  Password : [T05DN614PTJ/B05KHCBBUBZ/aSWGgf3bPHkjTRv7K1N5a2vI](https://hooks.slack.com/services/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)  Save |



|  |
| --- |
| In cmd:  airflow tasks test forex\_data\_pipeline send\_slack\_notification 2023-01-01 |





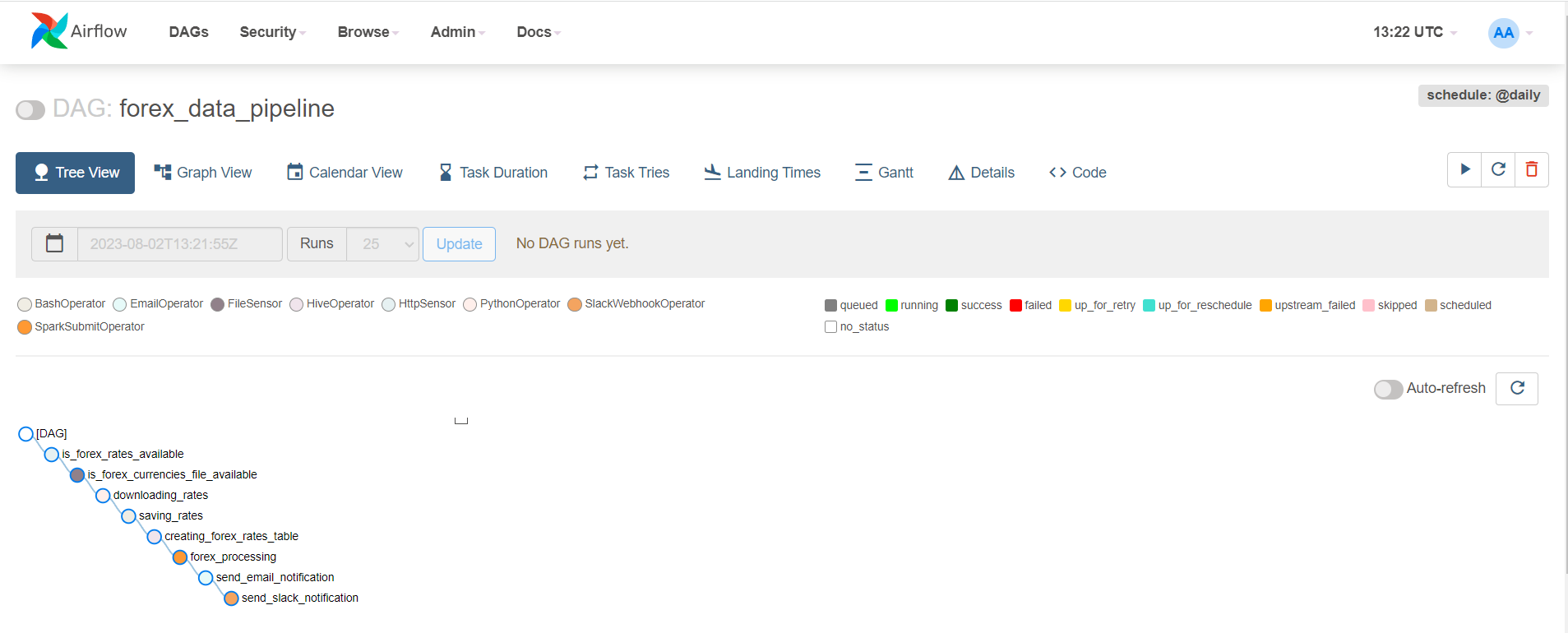
1. **Add dependency between tasks**

|  |
| --- |
| 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:



1. **Trigger a DAG from airflow UI:**

