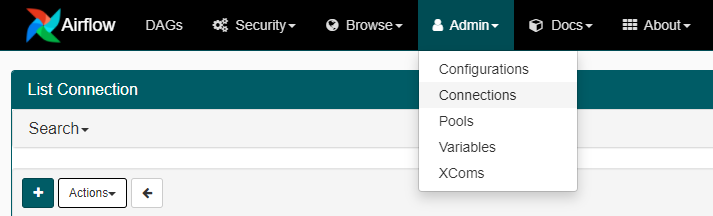
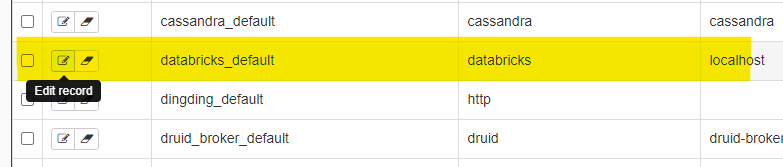
**Run notebooks/scripts/jars in Databricks using Airflow**

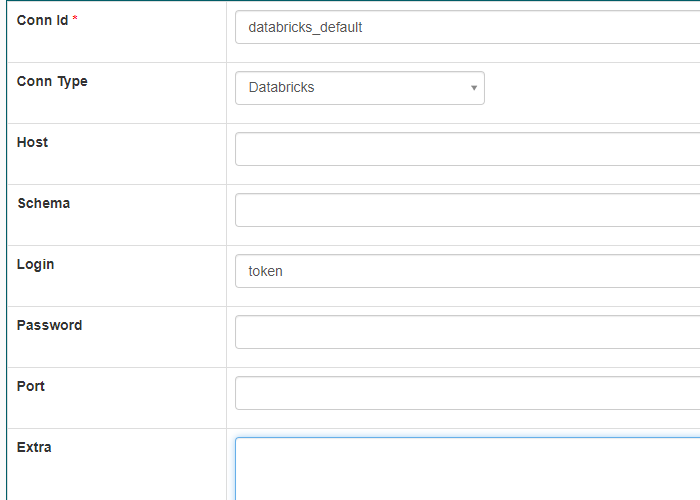
1. Create a Databricks connection in airflow

Go to Admin - > Connections 

Find **databricks\_default** and click edit (A new connection can also be created)



Fill in the required details



Host : https://<region>.azuredatabricks.net/

Login : token

Password : Generated user token

Extra : {"token": "<user\_token>","host":"<your\_host>"}

Eg: {"token": "abcdefghijklmn12345","host":"https://eastus2.azuredatabricks.net/"}

Leave the other fields empty and click save

1. Create a DAG with below code to run the notebook

We have two operators to use

**DatabricksRunNowOperator** to run an existing job

**DatabricksSubmitRunOperator** to create a one time run. Runs submitted via this don’t display in the databricks UI

Below example uses DatabricksSubmitRunOperator to trigger a notebook.

To submit a run, we need a notebook and a cluster config. We can also use an existing cluster instead of creating a new job cluster. To use an existing cluster, avoid new\_cluster\_conf and use existing\_cluster\_id instead of new\_cluster in the notebook\_task\_params

For other config options please check the jobs API link given in reference section

import airflow

from airflow import DAG

from airflow.operators.bash\_operator import BashOperator

from airflow.contrib.operators.databricks\_operator import DatabricksSubmitRunOperator

# default arguments for all the tasks

default\_args = {

    'owner': 'airflow',

    'email': ['airflow@example.com'],

    'depends\_on\_past': False,

    'start\_date': airflow.utils.dates.days\_ago(0)

}

# create a DAG definition

dag = DAG(dag\_id='databricks\_test', default\_args=default\_args, schedule\_interval=None)

# create a simple task that prints todays date

date\_task = BashOperator(

    task\_id='print\_date',

    bash\_command='date',

    dag=dag,

)

# create a cluster config

new\_cluster\_conf = {

    'spark\_version': '6.6.x-scala2.11',

    'node\_type\_id': 'Standard\_DS3\_v2',

    'autoscale' : {

        'min\_workers': 1,

        'max\_workers': 8

    },

    'spark\_conf': {

        'spark.databricks.delta.preview.enabled': 'true',

        'spark.sql.crossJoin.enabled': 'true',

    },

    'spark\_env\_vars': {

        'PYSPARK\_PYTHON': '/databricks/python3/bin/python3'

    },

}

notebook\_task\_params = {

    'new\_cluster': new\_cluster\_conf,

    'notebook\_task': {

    'notebook\_path': '<add\_path\_here>’,

    }

}

# create a task to run a notebook using above config

notebook\_task = DatabricksSubmitRunOperator(

    task\_id='notebook\_task',

    dag=dag,

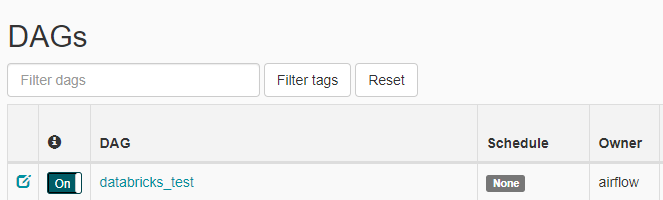
    json=notebook\_task\_params,

    do\_xcom\_push = True

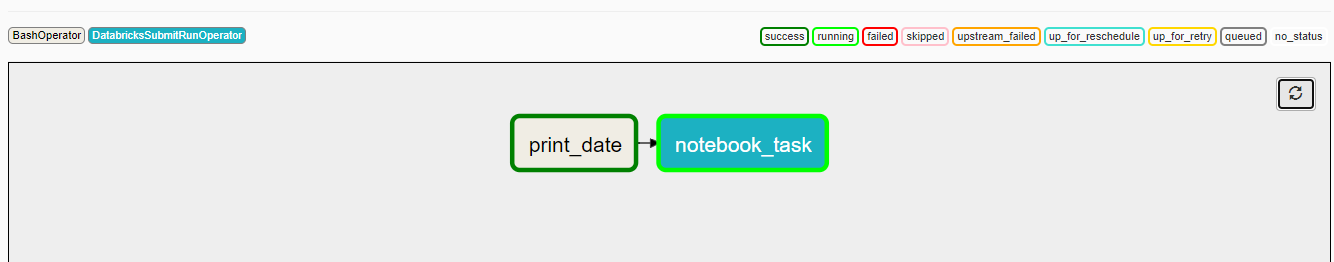
)

# set the order of tasks

date\_task >> notebook\_task

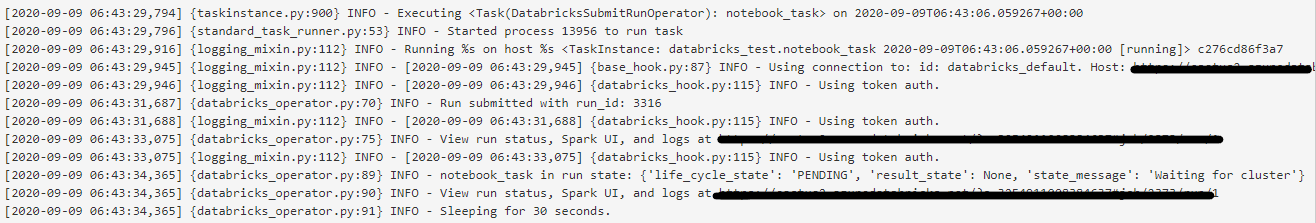
Save the DAG file and check the UI to see if the DAG is created

Turn ON the DAG and trigger it manually

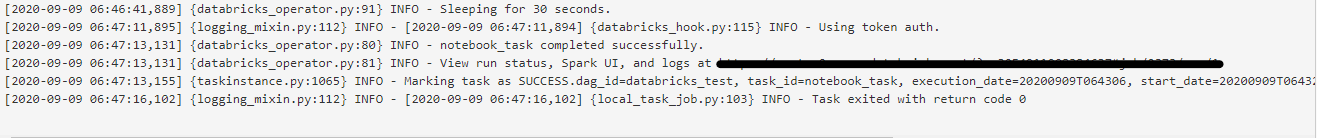


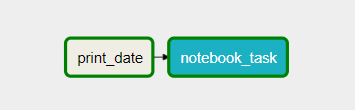
Click on the notebook task and view\_log

Check the logs if the connection is valid and the operator is able to start a cluster

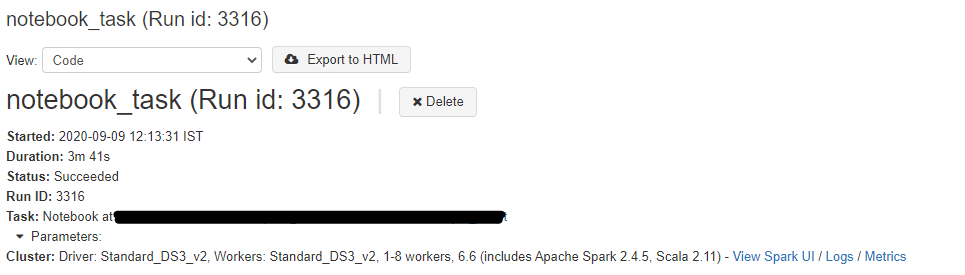


DatabricksSubmitRunOperator submits a job and waits for it to complete.





You can also check the run details in Databricks workspace if needed using the link printed in the logs



This is a just a sample with only a task to trigger a notebook. We can run multiple notebooks and tasks like checking the availability of files, DQC etc. Add upstream and downstream tasks with conditional triggers as needed and create complex pipelines. Airflow UI can be used to monitor the pipelines and we can also send mail/slack alerts on success/failure of DAG runs.

References:

<https://docs.databricks.com/dev-tools/api/latest/jobs.html>