

Etude comparative entre les outils d'automatisation et d'orchestration des tâche ML: Airflow vs. Luigi vs. Argo vs. MLFlow vs. KubeFlow



Apache
Airflow

*Visualisation et fouille de big Data
2ITE 3eme année*

Réalisé par : ARJANE Khadija

Supervisé par : Prof. Fahd KALLOUBI

29/12/2020

Table des matières

I. Objectif	3
II. Configuration de l'environnement.....	3
1. Étape 1 : Installation du sous-système Linux (Ubuntu)	3
a) Activer le mode développeur.....	3
b) Activer le sous-système Linux.....	4
2. Étape 2 : Installation de PIP.....	4
3. Étape 3 : Installation de Airflow.....	5
III. Airflow	7
1. Exemple de base du airflow	7
2. Architecture de base du flux d'air.....	8
IV. Manipulation	9
1. Scheduling and running Jupyter-Notebook files through Parpermill+Airflow	9
a. Installer jupyter notebook.....	9
b. Installer Parpermill sous airflow	9
c. Créer un notebook.....	10
d. Créer un DAG de airflow	10
e. Exécuter la tâche de airflow	11
f. Afficher le bloc-notes de sortie	11
2. Scheduling and running two Jupyter-Notebooks through Airflow	12
a) Lancer le serveur de jupiter	12
b) Importer les deux notebooks iris1 et iris2	12
c) Créer un dag de airflow pour synchroniser l'exécution des deux notebooks.....	12
d) Exécuter le dag de airflow	13
e) Afficher la sortie	13
V. Conclusion	15

I. Objectif

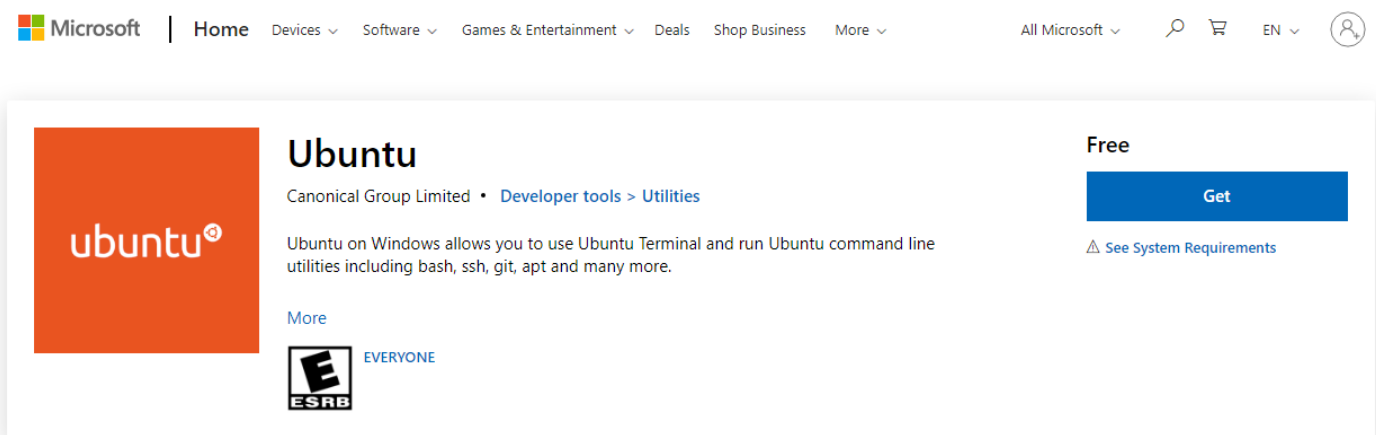
Airflow est une plate-forme créée par la communauté pour créer, planifier et surveiller les flux de travail par programmation. L'objectif de cette manipulation est d'exécuter directement le fichier un notebook de jupyter-notebook via airflow et papermill, pour rendre le déploiement et l'exécution des notebooks .ipynb simples et efficaces.

II. Configuration de l'environnement

Nous allons installer d'Apache **Airflow** sur une machine Windows 10 à l'aide d'Ubuntu.

1. Étape 1 : Installation du sous-système Linux (Ubuntu)

Vous pouvez trouver une copie gratuite d'Ubuntu dans le Microsoft Store [ici](#).

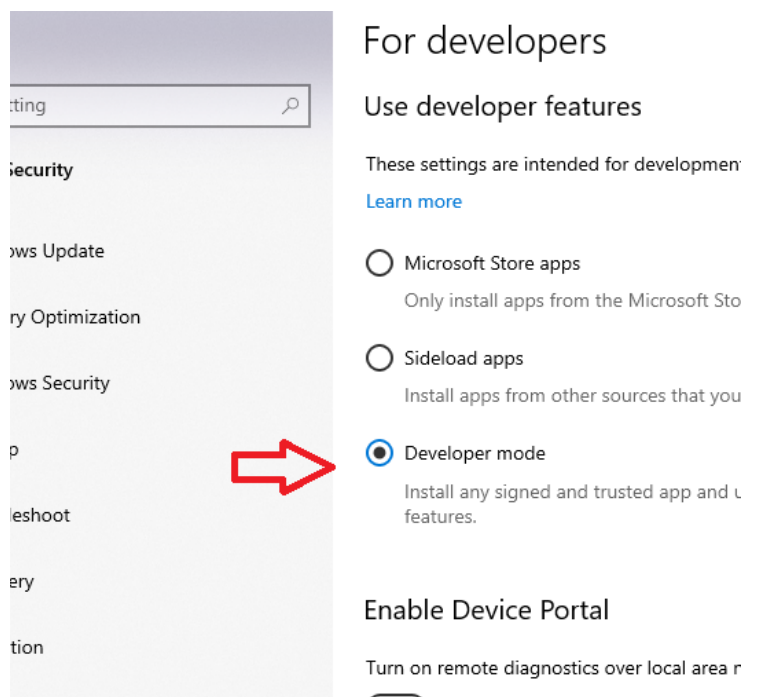


Téléchargez et installez Ubuntu. Avant d'ouvrir le programme, il y a quelques éléments de ménage dont nous devons d'abord nous occuper.

a) Activer le mode développeur

Il faut activer le mode développeur de Windows 10.

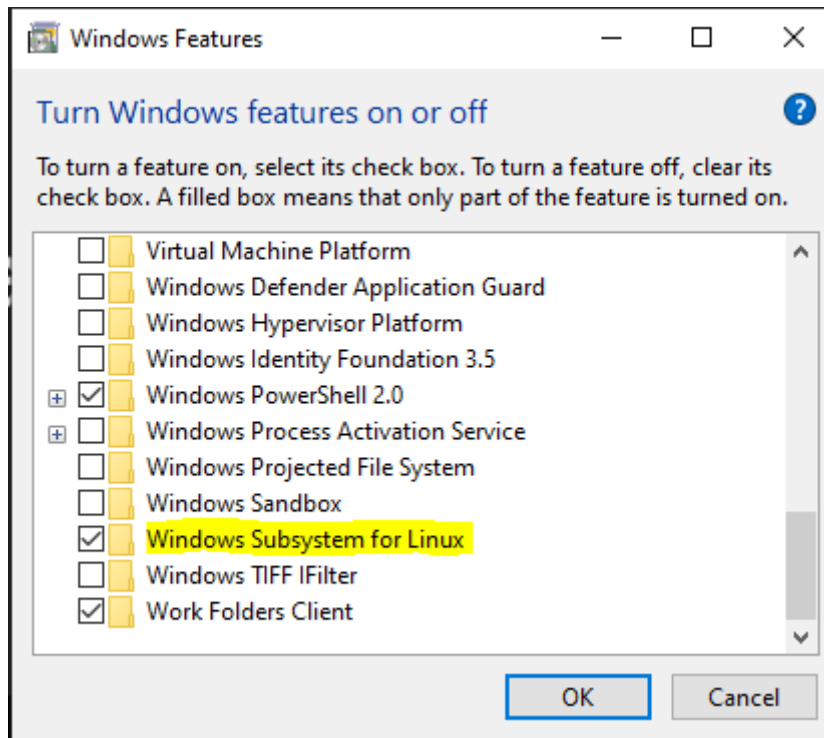
- Tapez « Developer » dans la barre de recherche de Windows et sélectionnez l'option qui dit « Paramètres du développeur ».
- Dans la page qui apparaît, sélectionnez la bulle à côté de l'option « Mode développeur ».



b) Activer le sous-système Linux

Dans la barre de recherche Windows, tapez « Fonctionnalité Windows » et sélectionnez l'option « Activer ou désactiver une fonctionnalité Windows ».

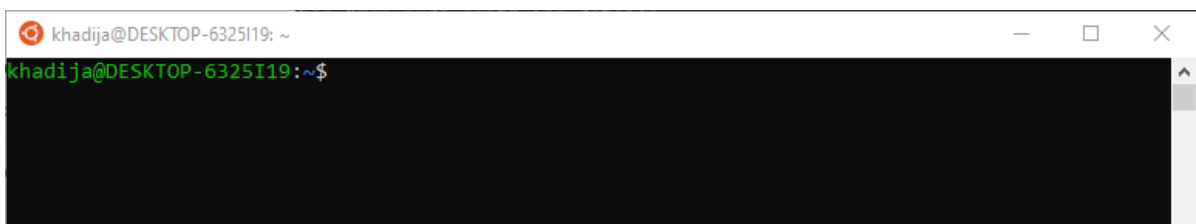
Cochez la case en regard de Sous-système Windows pour Linux et cliquez sur OK.



Remarque : Cela nécessitera un redémarrage pour prendre effet.

Après ces étapes, notre ménage est terminé ! Une fois que vous avez lancé Ubuntu, il commencera son processus d'installation initial qui ne prend généralement qu'une minute ou deux.

Ubuntu vous demandera alors un nom d'utilisateur et un mot de passe. Définissez-les sur les informations d'identification souhaitées et appuyez sur Entrée. Lorsque vous arrivez à une invite comme celle ci-dessous, vous êtes prêt à passer à l'étape deux.



2. Étape 2 : Installation de PIP

Pour installer Airflow, nous devons nous assurer que pip est installé. Exécutez la commande suivante pour installer la dernière version de pip.

```
sudo apt update && sudo apt upgrade  
sudo apt-get install software-properties-common  
sudo apt-add-repository universe  
sudo apt-get update  
sudo apt-get install python3-pip
```

3. Étape 3 : Installation de Airflow

Exécutez les 2 commandes suivantes pour installer Airflow :

```
export SLUGIFY_USES_TEXT_UNIDECODE=yes
pip3 install apache-airflow
```

Exécutez `sudo nano /etc/wsl.conf` , insérez le bloc ci-dessous, enregistrez et quittez avec `ctrl+s ctrl+x`.

```
[automount]
root = /
options = "métadonnées"
```

Exécutez `nano ~/.bashrc` , insérez la ligne ci-dessous, enregistrez et quittez avec `ctrl+s ctrl+x`

```
export AIRFLOW_HOME=/c/Users/khadija/AirflowHome
```

Remarque : (khadija c'est mon nom utilisateur que j'ai ajouté lors de la configuration de Ubuntu, alors vous devez ajouter votre no d'utilisateur de Ubuntu, le vôtre).

Ouvrez un nouveau terminal Ubuntu. (J'ai été surpris, mais cela semblait nécessaire).

Initier le DB de Airflow par cette commande :

```
airflow db init
```

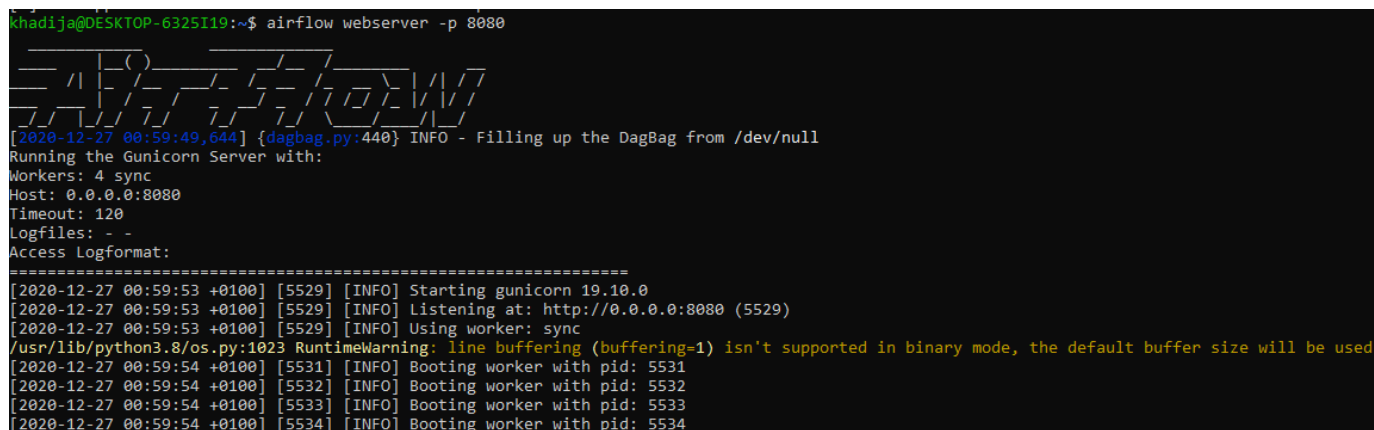
Assurez-vous d'avoir créé un utilisateur administrateur :

```
airflow users create -r Admin -u admin -e admin@acme.com -f admin -l user -p admin
```

Nous pouvons maintenant démarrer le serveur Web et le planificateur d'airflow .

Exécutez la commande et laissez-la s'exécuter :

```
airflow webserver -p 8080
```



```
khadija@DESKTOP-6325I19:~$ airflow webserver -p 8080
[2020-12-27 00:59:49,644] {dagbag.py:440} INFO - Filling up the DagBag from /dev/null
Running the Gunicorn Server with:
Workers: 4 sync
Host: 0.0.0.0:8080
Timeout: 120
Logfiles: - -
Access Logformat:
[2020-12-27 00:59:53 +0100] [5529] [INFO] Starting gunicorn 19.10.0
[2020-12-27 00:59:53 +0100] [5529] [INFO] Listening at: http://0.0.0.0:8080 (5529)
[2020-12-27 00:59:53 +0100] [5529] [INFO] Using worker: sync
/usr/lib/python3.8/os.py:1023 RuntimeWarning: line buffering (buffering=1) isn't supported in binary mode, the default buffer size will be used
[2020-12-27 00:59:54 +0100] [5531] [INFO] Booting worker with pid: 5531
[2020-12-27 00:59:54 +0100] [5532] [INFO] Booting worker with pid: 5532
[2020-12-27 00:59:54 +0100] [5533] [INFO] Booting worker with pid: 5533
[2020-12-27 00:59:54 +0100] [5534] [INFO] Booting worker with pid: 5534
```

Ouvrez une nouvelle fenêtre de terminal et exécutez la commande :

- `airflow scheduler`

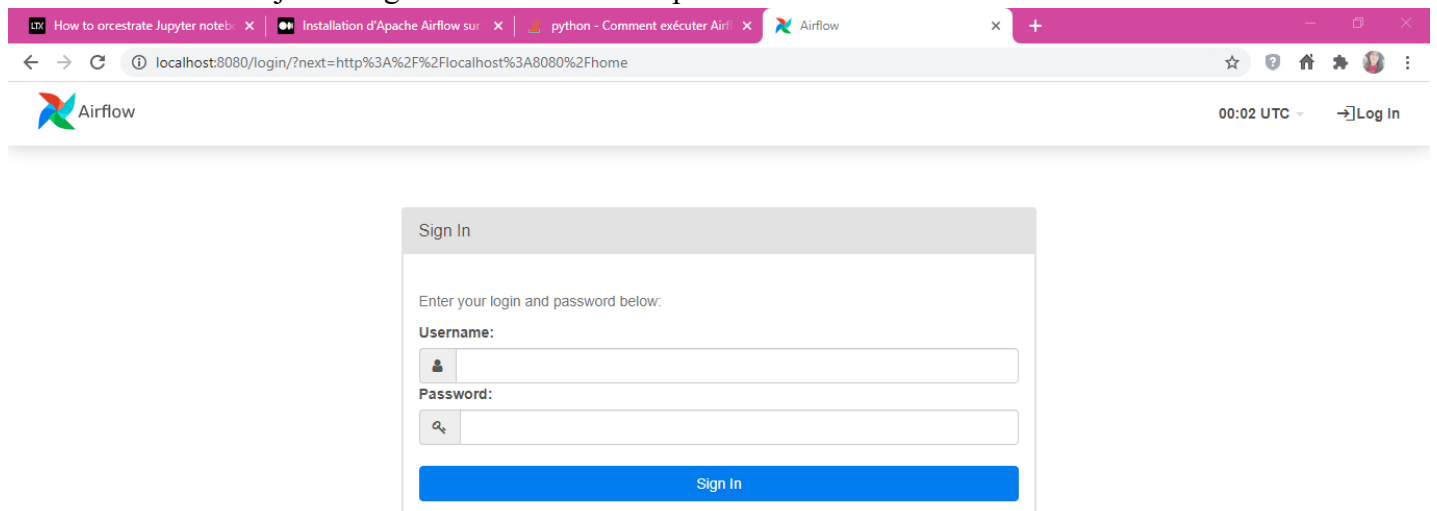
```
Sélection khadija@DESKTOP-6325I19: ~
khadija@DESKTOP-6325I19:~$ airflow scheduler

[2020-12-27 01:01:07,842] {scheduler_job.py:1241} INFO - Starting the scheduler
[2020-12-27 01:01:07,843] {scheduler_job.py:1246} INFO - Processing each file at most -1 times
[2020-12-27 01:01:07,870] {dag_processing.py:250} INFO - Launched DagFileProcessorManager with pid: 5560
[2020-12-27 01:01:07,872] {scheduler_job.py:1751} INFO - Resetting orphaned tasks for active dag runs
[2020-12-27 01:01:07,884] {scheduler_job.py:1764} INFO - Marked 1 SchedulerJob instances as failed
[2020-12-27 01:01:07,886] {settings.py:52} INFO - Configured default timezone Timezone('UTC')
[2020-12-27 01:01:07,909] {dag_processing.py:515} WARNING - Because we cannot use more than 1 thread (parsing_processes = 2 ) when using sqlite. So we set parallelism to 1.
```

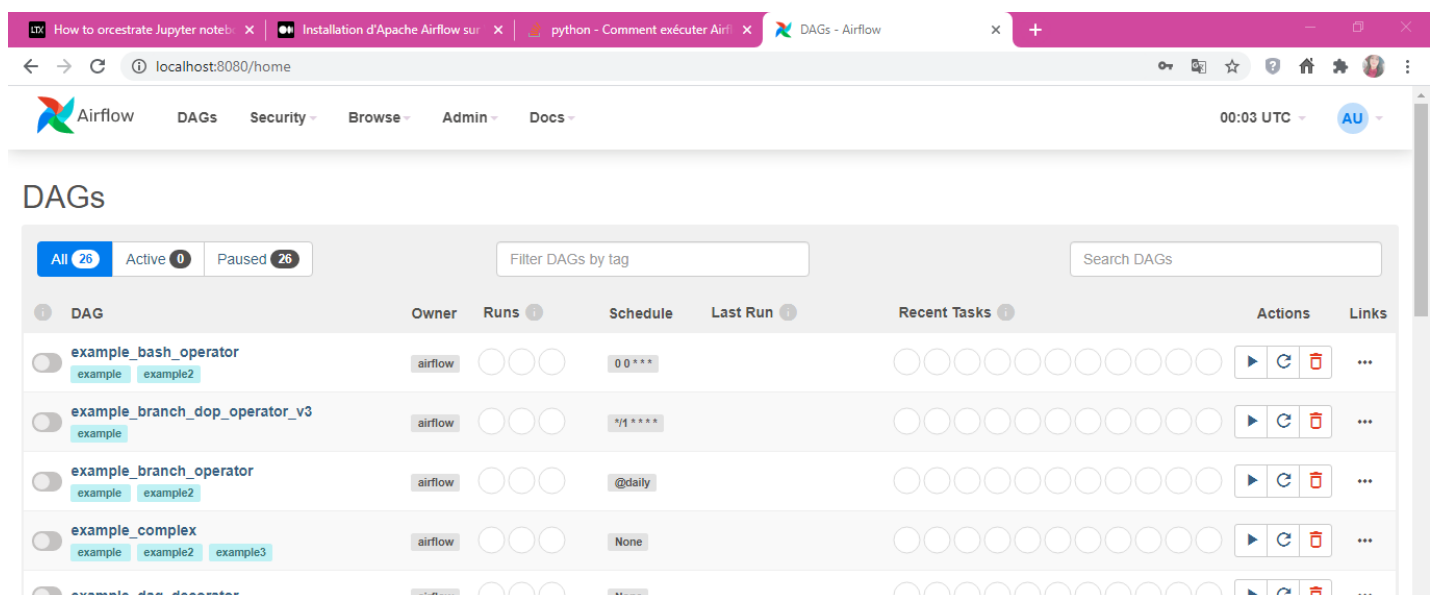
Une fois les deux exécutés avec succès, ouvrez un onglet dans le navigateur de votre choix (Chrome, Firefox, etc.) et entrez ce qui suit dans la barre d'URL : <http://localhost:8080>

L'authentification :

- Vous devez ajouter login : **admin** et mot de passe : **admin**



- Lorsque vous appuyez sur Entrée, la page suivante devrait se charger :



III. Airflow

1. Exemple de base du airflow

Voici quelques commandes qui déclencheront quelques instances de tâches. Vous devriez être en mesure de voir le statut des travaux changer dans le `example_bash_operator` DAG lorsque vous exécutez les commandes ci-dessous.

Ouvrez un nouveau terminal Ubuntu.

```
# run your first task instance
```

```
airflow tasks run example_bash_operator runme_0 2015-01-01
```

```
# run a backfill over 2 days
```

airflow dags backfill example_bash_operator \

```
--start-date 2015-01-01 \
```

```
--end-date 2015-01-02
```

Airflow

DAGsSecurityBrowseAdminDocs

00:24 UTCAU

DAGs

All 26Active 0Paused 26

Filter DAGs by tag

Search DAGs

DAG	Owner	Runs	Schedule	Last Run	Recent Tasks	Actions	Links
<input type="checkbox"/> example_bash_operator example example2	airflow	2	00***	2015-01-02, 00:00:00	[Success]	[Run][Refresh][Delete] ...	
<input type="checkbox"/> example_branch_dop_operator_v3 example	airflow		*/* ****			[Run][Refresh][Delete] ...	
<input type="checkbox"/> example_branch_operator example example2	airflow		@daily			[Run][Refresh][Delete] ...	

☒ DAG: example_bash_operator

success schedule: 00***

Tree ViewGraph ViewTask DurationTask TriesLanding TimesGanttDetailsCode

2015-01-02T00:00:01ZRuns 25Runbackfill__2015-01-02T00:00:00+00:00LayoutLeft > RightUpdateFind Task...

BashOperatorDummyOperatorqueuedrunningSUCCESSfailedup_for_retryup_for_rescheduleupstream_failedskippedscheduledno_status

Auto-refresh

```
graph LR; runme_0 --> also_run_this; runme_1 --> run_after_loop; runme_2 --> run_after_loop; also_run_this --> run_this_last; run_after_loop --> run_this_last;
```

☒ DAG: example_bash_operator

scheduled: 00***

Tree ViewGraph ViewTask DurationTask TriesLanding TimesGanttDetailsCode

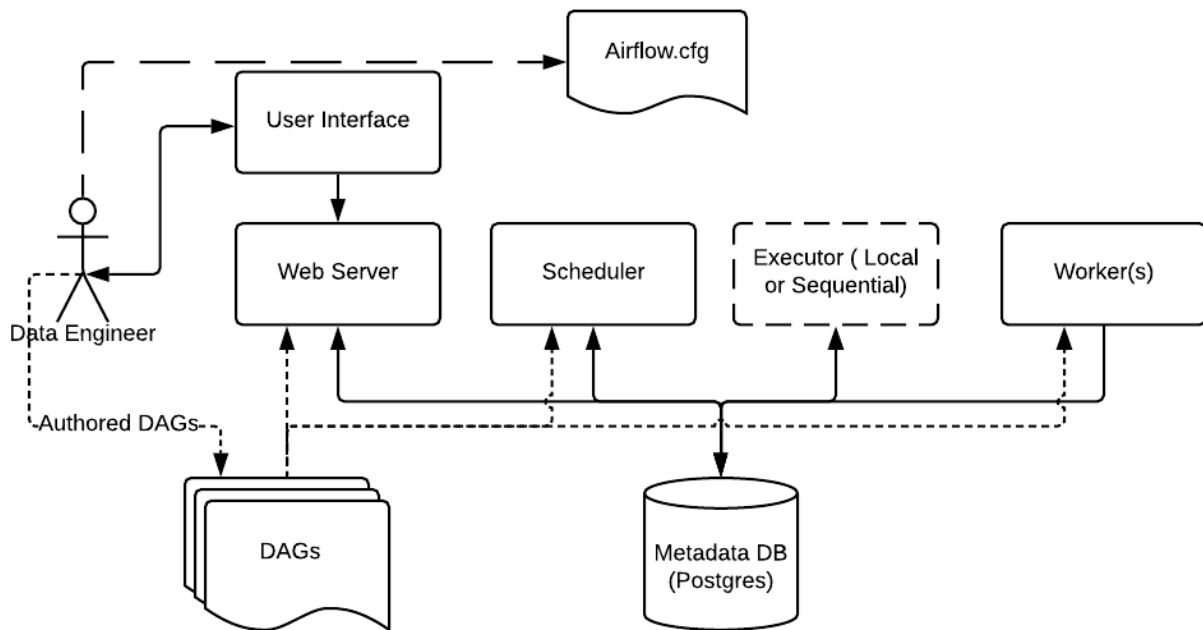
2015-01-02T00:00:01ZRuns 25Runbackfill__2015-01-02T00:00:00+00:00Update

runme_0
runme_1
runme_2
also_run_this
run_after_loop
run_this_last

Task	Start Time	End Time	Status
runme_0	00:10:40	00:10:41	Completed
runme_1	00:10:46	00:10:47	Completed
runme_2	00:10:54	00:10:55	Completed
also_run_this	00:11:02	00:11:03	In Progress
run_after_loop	00:11:08	00:11:09	In Progress
run_this_last	00:11:14	00:11:15	In Progress

2. Architecture de base du flux d'air

Principalement destinée au développement, l'architecture de base d'Airflow avec les exécuteurs Local et Séquentiel est un excellent point de départ pour comprendre l'architecture d'Apache Airflow.



Il y a quelques éléments à noter :

- **Base de données de métadonnées** : Airflow utilise une base de données SQL pour stocker des métadonnées sur les pipelines de données en cours d'exécution. Dans le diagramme ci-dessus, cela est représenté par Postgres qui est extrêmement populaire avec Airflow. Les autres bases de données prises en charge avec Airflow incluent MySQL.
- **Serveur Web et planificateur** : Le serveur Web et le planificateur Airflow sont des processus distincts exécutés (dans ce cas) sur la machine locale et interagissent avec la base de données mentionnée ci-dessus.
- L' **exécuteur** est présenté séparément ci-dessus, car il est généralement discuté dans Airflow et dans la documentation, mais en réalité, il ne s'agit PAS d'un processus distinct, mais exécuté dans le planificateur.
- Le ou les **travailleurs** sont des processus distincts qui interagissent également avec les autres composants de l'architecture Airflow et du référentiel de métadonnées.
- **airflow.cfg** est le fichier de configuration Airflow auquel accèdent le serveur Web, le planificateur et les travailleurs.
- **Les DAG font** référence aux fichiers DAG contenant du code Python, représentant les pipelines de données à exécuter par Airflow. L'emplacement de ces fichiers est spécifié dans le fichier de configuration Airflow, mais ils doivent être accessibles par le serveur Web, le planificateur et les travailleurs.

IV. Manipulation

1. *Scheduling and running Jupyter-Notebook files through Parpermill+Airflow*

Exécutez les fichiers Jupyter-Notebook (.ipynb) via Parpermill + Airflow

Lors du traitement des données et du code de science des données, nous utilisons généralement jupyter-notebook pour le développement. Nous pouvons développer de nombreux fichiers notebook (.ipynb), et il peut encore y avoir des dépendances entre les fichiers notebook. Il est plus difficile de gérer plus de notebooks, et c'est un grand défi pour la planification en ligne et les tâches.

Parpermill est un outil qui peut exécuter directement des notebooks en fonction des dépendances entre les blocs-notes, sans convertir les blocs-notes en fichiers python, et peut transmettre des paramètres.

Airflow est un système de planification open source, qui fournit les opérateurs Parpermill. Avec ces deux outils, le déploiement et le lancement du notebook devient simple et rapide.

Le processus de déploiement et de lancement devient :

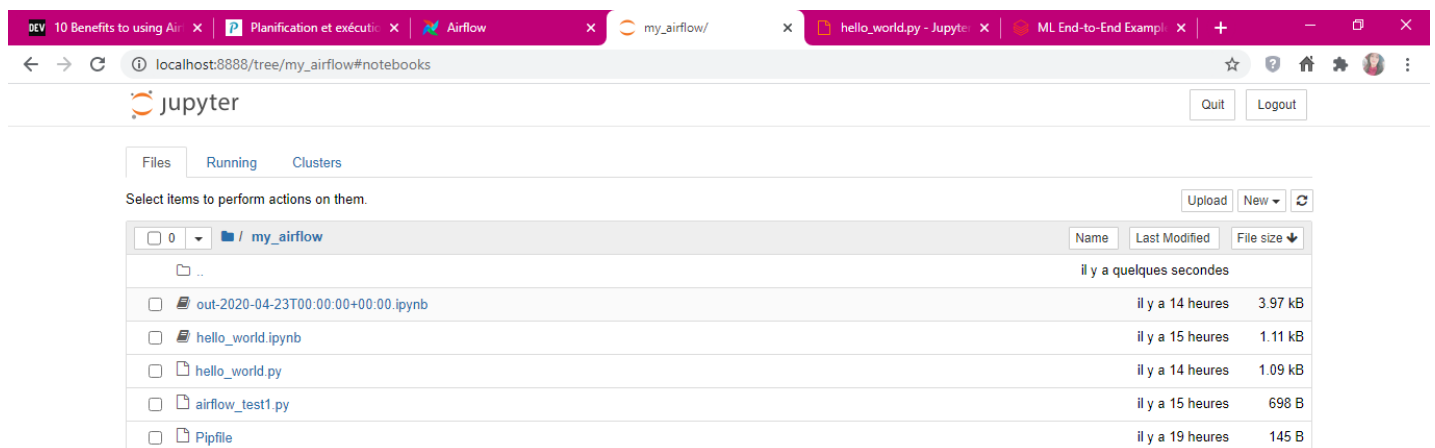
- La première étape : développer du code sur jupyter.
- La deuxième étape : J'ai utilisé l'opérateur Parpermill sur airflow pour configurer les dépendances et les paramètres entre les notebooks.
- La troisième étape : exécuter la planification du flux d'air.

a. Installer jupyter notebook

```
sudo pip3 install jupyter
jupyter notebook --generate-config
```

Lancer le serveur de jupiter

```
jupyter notebook --no-browser --port=8888
```



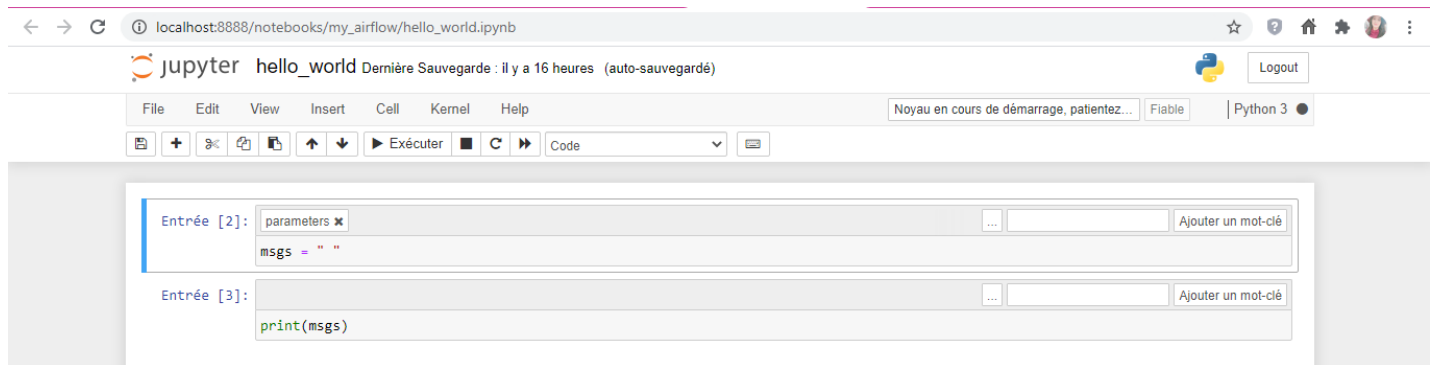
Name	Last Modified	File size
..	il y a quelques secondes	
out-2020-04-23T00:00:00+00:00.ipynb	il y a 14 heures	3.97 kB
hello_world.ipynb	il y a 15 heures	1.11 kB
hello_world.py	il y a 14 heures	1.09 kB
airflow_test1.py	il y a 15 heures	698 B
Pipfile	il y a 19 heures	145 B

b. Installer Parpermill sous airflow

```
pip3 install 'apache-airflow[papermill]'
pip3 install kubernetes
```

c. Créer un notebook

Créez un notebook via jupyter-notebook, nommé : `hello_world.ipynb`, son contenu est le suivant:



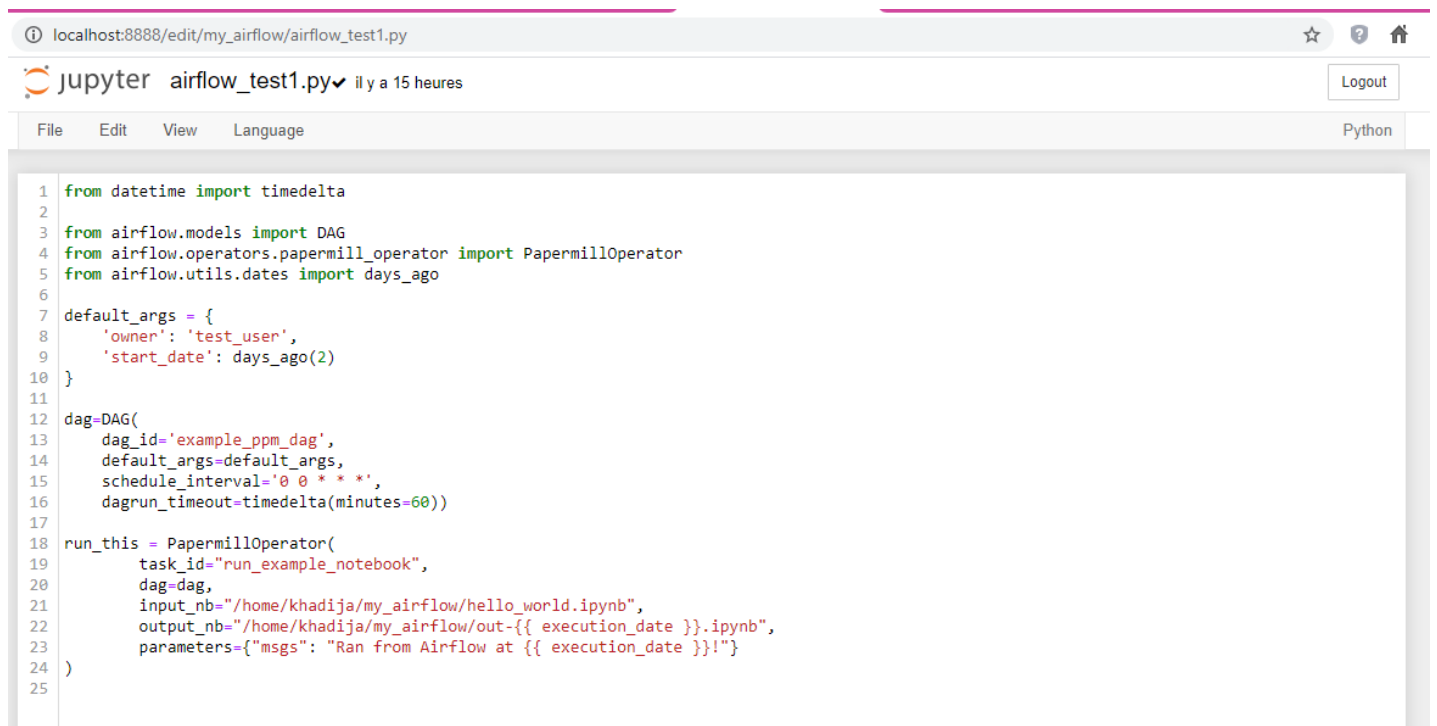
Remarque : vous trouverez le fichier dans le dossier ressources.

Notez que la première cellule (la cellule qui a des paramètres et veut changer la valeur de paramètre en externe) doit ajouter une balise (étiquette) et la nommer : `parameters`.

La méthode pour ajouter un tag est : sélectionnez cette cellule, sélectionnez : `"View" -> "Cell Tool" -> "tags"`, entrez `parameters` dans la zone de saisie, puis cliquez sur `"Add Tag"`.

d. Créer un DAG de airflow

Créez un fichier python nommé : `airflow_test1.py`, le code est le suivant :



Remarque : vous trouverez le fichier dans le dossier ressources.

e. Exécuter la tâche de airflow

Avant d'exécuter des tâches de flux d'air, vous devez copier `airflow_test1.py` dans le répertoire dags sous le répertoire d'installation airflow. S'il n'y a pas de dags, vous pouvez en créer un.

Ajouter un dossier dags à `/c/Users/$user/AirflowHome/` avec :

```
cd /c/Users/maite/AirflowHome
mkdir dags
```

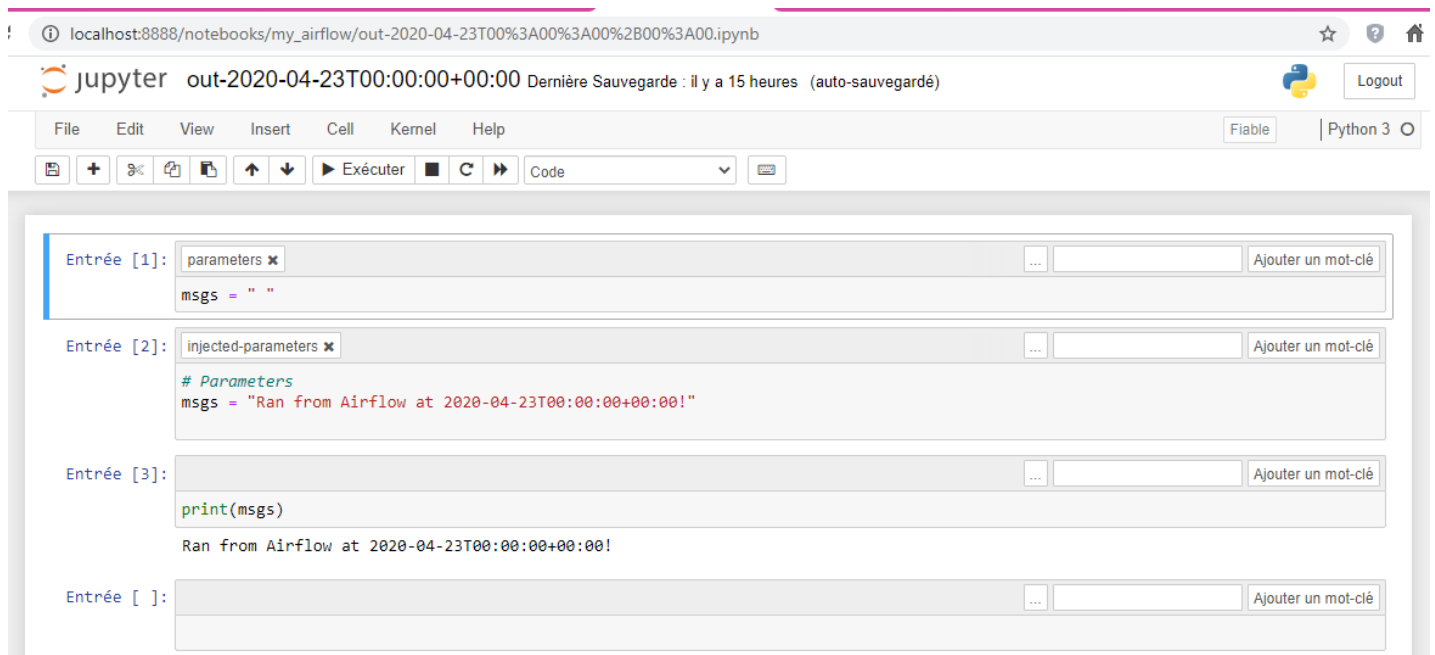
On copie le fichier `airflow_test.py` au répertoire de `AirflowHome/dags` avec cette commande :

```
cp /home/khadija/my_airflow/airflow_test1.py /c/Users/maite/AirflowHome/dags
cd /c/Users/maite/AirflowHome/dags
python3 airflow_test1.py
airflow tasks run example_ppm_dag run_example_notebook 20200423
```

f. Afficher le bloc-notes de sortie

Selon l'opérateur Airflow PapermillOperator, nous avons configuré le nom du notebook de sortie comme : `out - {{execution_date}}`. Ipynb, qui est `out- <temps d'exécution> .ipynb`, vérifiez s'il y a le notebook dans le répertoire `/home/$user/my_airflow/`. Si tel est le cas, l'exécution a réussi.

Lorsque vous ouvrez le notebook dans jupyter-notebook, le résultat devrait ressembler à ceci :



The screenshot shows a Jupyter Notebook interface with the following elements:

- Browser address bar: `localhost:8888/notebooks/my_airflow/out-2020-04-23T00%3A00%3A00%2B00%3A00.ipynb`
- Jupyter logo and title: `out-2020-04-23T00:00:00+00:00`
- Menu bar: File, Edit, View, Insert, Cell, Kernel, Help
- Toolbar: Buttons for file operations, execution, and code editing.
- Input fields:
 - Entrée [1]: `parameters`
 - Entrée [2]: `injected-parameters`
 - Entrée [3]: (empty)
 - Entrée []: (empty)
- Output of the third cell:

```
print(msgs)

Ran from Airflow at 2020-04-23T00:00:00+00:00!
```

Comme vous pouvez le voir, le paramètre est remplacé par le paramètre `msgs` dans l'opérateur de flux d'air, indiquant que l'exécution a réussi.

2. Scheduling and running two Jupyter-Notebooks through Airflow

Nous allons lancer l'exécution de deux notebooks en ordre avec airflow dags.

a) Lancer le serveur de jupyter

jupyter notebook --no-browser --port=8888

b) Importer les deux notebooks iris1 et iris2



The screenshot shows the JupyterLab interface with the 'Files' tab selected. The breadcrumb path is '/ my_airflow'. Below the breadcrumb, there is a table of files:

	Name	Last Modified	File size
	..	il y a quelques secondes	
<input type="checkbox"/>	hello_world.ipynb	il y a un jour	1.11 kB
<input type="checkbox"/>	Iris1.ipynb	il y a 5 heures	98.6 kB
<input type="checkbox"/>	Iris2.ipynb	il y a 5 heures	28.6 kB

Remarque : vous trouverez les fichiers dans le dossier ressources.

c) Créer un dag de airflow pour synchroniser l'exécution des deux notebooks.

Créez un fichier python nommé : hello_world.py, le code est le suivant :



The screenshot shows the JupyterLab code editor with the file 'hello_world.py' open. The code is as follows:

```
1 from datetime import datetime
2 from airflow import DAG
3 from airflow.operators.dummy_operator import DummyOperator
4 from airflow.operators.python_operator import PythonOperator
5 from airflow.operators.bash_operator import BashOperator
6 from airflow.operators.papermill_operator import PapermillOperator
7 from airflow.utils.dates import days_ago
8
9 # Define default args
10 default_args = {
11     'owner': 'user',
12     'on_failure_callback': lambda context: True,
13 }
14 def my_func():
15     print('Notebook runned successfully!')
16
17
18 # Define DAG setting
19 with DAG('python_dag', description='Python DAG', default_args=default_args, schedule_interval='*/5 * * * *', start_date=datetime(2018, 11, 1), catchup=False) as dag:
20     first_nb = PythonOperator(
21         task_id='first_nb',
22         python_callable=my_func,
23         provide_context=True,
24         op_kwargs={'path': '/home/khadija/my_airflow/iris1.ipynb'})
25     second_nb = PythonOperator(
26         task_id='second_nb',
27         python_callable=my_func,
28         provide_context=True,
29         op_kwargs={'path': '/home/khadija/my_airflow/iris2.ipynb'},)
30
31     first_nb >> second_nb
```

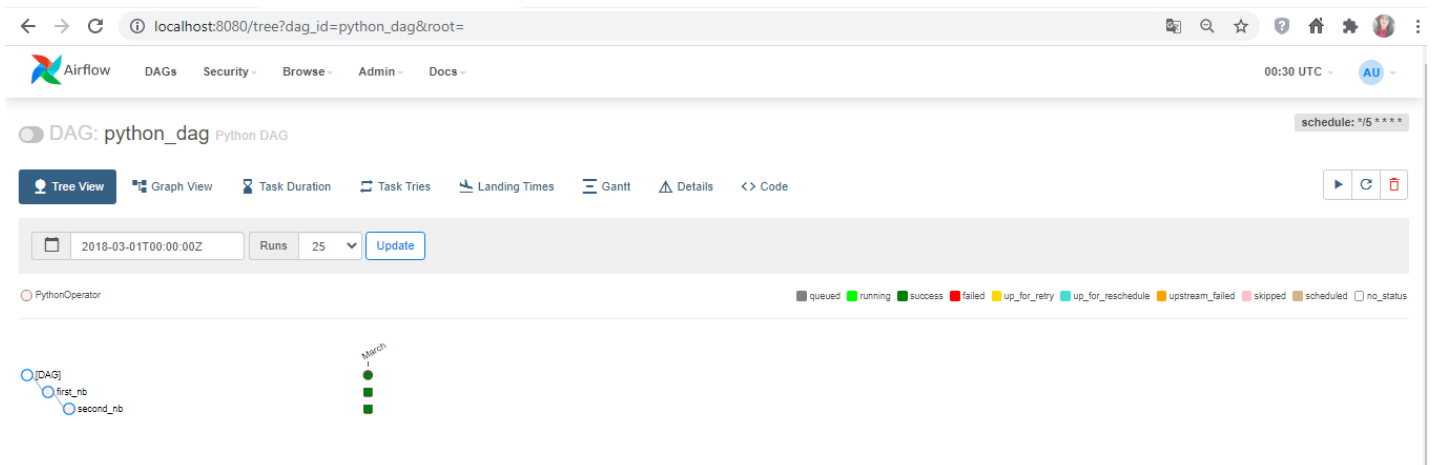
Remarque : vous trouverez le fichier dans le dossier ressources.

d) Exécuter le dag de airflow

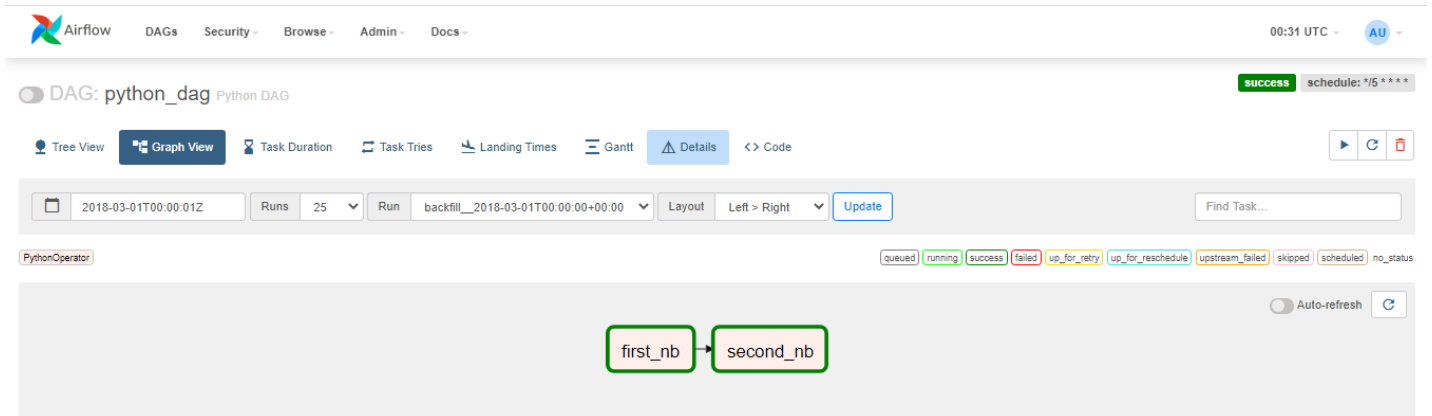
```
cd /c/Users/maite/AirflowHome
cp /home/khadija/my_airflow/hello_world.py /c/Users/maite/AirflowHome/dags
cd /c/Users/maite/AirflowHome/dags
python3 hello_world.py
airflow dags backfill python_dag -s 2018-03-01
```

e) Afficher la sortie

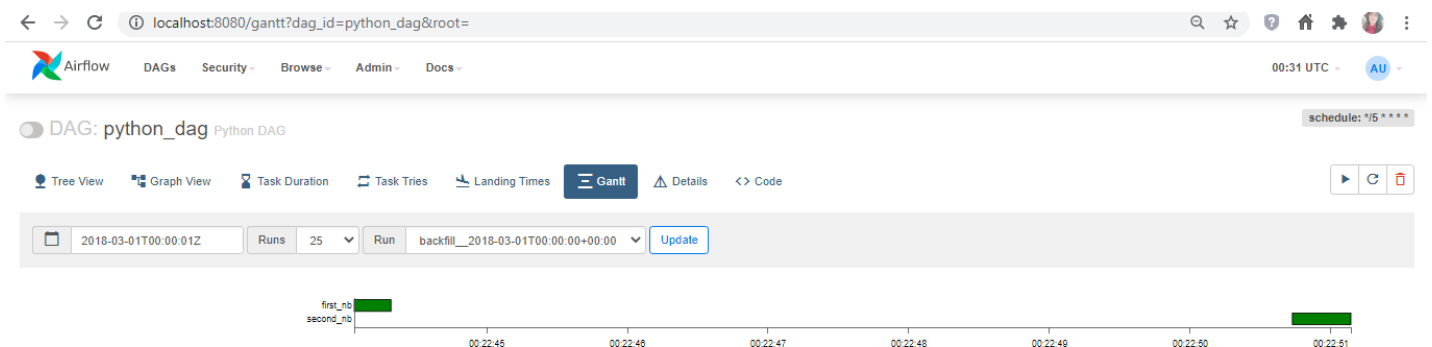
Nous avons synchronisé le lancement de premier notebook en premier et le deuxième après 5 seconds sans aucune erreur.



Les deux taches ont été bien exécutés.



La tâche first_nb et par la suite second_nb.



Un intervalle de 5 seconds entre les deux tâches.

The screenshot shows the Airflow web interface for the DAG 'python_dag'. The task 'first_nb' is selected, and its log is displayed. The log shows the task starting at 2018-03-01 00:00:00+00:00. The log content includes the following lines:

```
*** Reading local file: /c/Users/maite/AirflowHome/logs/python_dag/first_nb/2018-03-01T00:00:00+00:00/1.log
[2020-12-28 01:22:44,059] [taskinstance.py:826] INFO - Dependencies all met for <TaskInstance: python_dag.first_nb 2018-03-01T00:00:00+00:00 [queued]>
[2020-12-28 01:22:44,067] [taskinstance.py:826] INFO - Dependencies all met for <TaskInstance: python_dag.first_nb 2018-03-01T00:00:00+00:00 [queued]>
[2020-12-28 01:22:44,067] [taskinstance.py:1017] INFO -
-----
[2020-12-28 01:22:44,067] [taskinstance.py:1018] INFO - Starting attempt 1 of 1
[2020-12-28 01:22:44,068] [taskinstance.py:1019] INFO -
-----
[2020-12-28 01:22:44,090] [taskinstance.py:1038] INFO - Executing <Task(PythonOperator): first_nb> on 2018-03-01T00:00:00+00:00
[2020-12-28 01:22:44,136] [standard_task_runner.py:51] INFO - Started process 13882 to run task
[2020-12-28 01:22:44,155] [standard_task_runner.py:75] INFO - Running: ['airflow', 'tasks', 'run', 'python_dag', 'first_nb', '2018-03-01T00:00:00+00:00', '--job-id', '41', '--pool', 'default_pool', '--raw', '--subdir', 'DAGS_FOLDER', 'python_dag.py', 'first_nb']
[2020-12-28 01:22:44,161] [standard_task_runner.py:76] INFO - Job 41: Subtask first_nb
[2020-12-28 01:22:44,239] [logging_mixin.py:103] INFO - Running <TaskInstance: python_dag.first_nb 2018-03-01T00:00:00+00:00 [running]> on host DESKTOP-6325I19.localdomain
[2020-12-28 01:22:44,313] [taskinstance.py:1230] INFO - Exporting the following env vars:
AIRFLOW_CTX_DAG_OWNER=user
AIRFLOW_CTX_DAG_ID=python_dag
AIRFLOW_CTX_TASK_ID=first_nb
AIRFLOW_CTX_EXECUTION_DATE=2018-03-01T00:00:00+00:00
AIRFLOW_CTX_DAG_RUN_ID=backfill_2018-03-01T00:00:00+00:00
[2020-12-28 01:22:44,315] [logging_mixin.py:103] INFO - Notebook_ran Successfully
[2020-12-28 01:22:44,315] [python.py:118] INFO - Done. Returned value was: None
[2020-12-28 01:22:44,320] [taskinstance.py:1135] INFO - Marking task as SUCCESS. dag_id=python_dag, task_id=first_nb, execution_date=20180301T000000, start_date=20201228T002244, end_date=20201228T002244
[2020-12-28 01:22:44,350] [taskinstance.py:1195] INFO - 0 downstream tasks scheduled from follow-on schedule check
```

Dans le log de `first_nb` on peut voir le return de la fonction `my_func` lors que la tâche a été bien exécuté.

The screenshot shows the Airflow web interface for the DAG 'python_dag'. The task 'first_nb' is selected, and its log is displayed. The log shows the task starting at 2018-03-01 00:00:00+00:00. The log content includes the following lines:

```
*** Reading local file: /c/Users/maite/AirflowHome/logs/python_dag/first_nb/2018-03-01T00:00:00+00:00/1.log
[2020-12-28 01:22:44,059] [taskinstance.py:826] INFO - Dependencies all met for <TaskInstance: python_dag.first_nb 2018-03-01T00:00:00+00:00 [queued]>
[2020-12-28 01:22:44,067] [taskinstance.py:826] INFO - Dependencies all met for <TaskInstance: python_dag.first_nb 2018-03-01T00:00:00+00:00 [queued]>
[2020-12-28 01:22:44,067] [taskinstance.py:1017] INFO -
-----
[2020-12-28 01:22:44,067] [taskinstance.py:1018] INFO - Starting attempt 1 of 1
[2020-12-28 01:22:44,068] [taskinstance.py:1019] INFO -
-----
[2020-12-28 01:22:44,090] [taskinstance.py:1038] INFO - Executing <Task(PythonOperator): first_nb> on 2018-03-01T00:00:00+00:00
[2020-12-28 01:22:44,136] [standard_task_runner.py:51] INFO - Started process 13882 to run task
[2020-12-28 01:22:44,155] [standard_task_runner.py:75] INFO - Running: ['airflow', 'tasks', 'run', 'python_dag', 'first_nb', '2018-03-01T00:00:00+00:00', '--job-id', '41', '--pool', 'default_pool', '--raw', '--subdir', 'DAGS_FOLDER', 'python_dag.py', 'first_nb']
[2020-12-28 01:22:44,161] [standard_task_runner.py:76] INFO - Job 41: Subtask first_nb
[2020-12-28 01:22:44,239] [logging_mixin.py:103] INFO - Running <TaskInstance: python_dag.first_nb 2018-03-01T00:00:00+00:00 [running]> on host DESKTOP-6325I19.localdomain
[2020-12-28 01:22:44,313] [taskinstance.py:1230] INFO - Exporting the following env vars:
AIRFLOW_CTX_DAG_OWNER=user
AIRFLOW_CTX_DAG_ID=python_dag
AIRFLOW_CTX_TASK_ID=first_nb
AIRFLOW_CTX_EXECUTION_DATE=2018-03-01T00:00:00+00:00
AIRFLOW_CTX_DAG_RUN_ID=backfill_2018-03-01T00:00:00+00:00
[2020-12-28 01:22:44,315] [logging_mixin.py:103] INFO - Notebook_ran Successfully
[2020-12-28 01:22:44,315] [python.py:118] INFO - Done. Returned value was: None
[2020-12-28 01:22:44,320] [taskinstance.py:1135] INFO - Marking task as SUCCESS. dag_id=python_dag, task_id=first_nb, execution_date=20180301T000000, start_date=20201228T002244, end_date=20201228T002244
[2020-12-28 01:22:44,350] [taskinstance.py:1195] INFO - 0 downstream tasks scheduled from follow-on schedule check
[2020-12-28 01:22:44,360] [local_task_job.py:118] INFO - Task exited with return code 0
```

Dans le log de `first_nb` on peut voir le return de la fonction `my_func` lors que la tâche a été bien exécuté.

V. Conclusion

Cette manipulation décrit comment exécuter directement le fichier .ipynb de jupyter-notebook via airflow et papermill. Cela rend le déploiement et l'exécution des notebooks .ipynb simples et efficaces.