

Pour ce chapitre nous avons vu le framework **Airflow** et comment les jobs sont planifiés et exécutés à travers son core **DAG** qui est une collection de tâches à exécuter sur un ordre donné.

Pour pouvoir gérer un workflow avec **Airflow**, l'ensemble des tâches à exécuter sont définies dans le fichier DAG qui est un script python contenant l'ensemble des dépendances et tâches à exécuter ainsi que l'ordre d'exécution des tâches.

Pour cela nous avons la possibilité d'utiliser workflow avec le container Docker ou de l'installer en local pour ensuite l'utiliser.

L'utilisation avec Docker nous a causé des soucis, après avoir pull l'image de workflow sur Docker Hub nous n'arrivions pas à voir le dossier airflow pour y mettre nos fichiers dag.

Donc nous avons utilisé la deuxième option qui est d'installer Airflow afin de pouvoir l'utiliser.

Pour cela, nous avons créé un environnement Python avec anaconda et ensuite activé ce dernier à travers la commande **conda activate**.

Ensuite nous avons installé les dépendances de Airflow avec la commande **apt-get install** et enfin nous avons utilisé la commande **pip** pour installer Airflow.

Ainsi nous avons créé un sous dossier dag, dans le répertoire airflow, qui va contenir l'ensemble de fichiers dag et script bash, python, ...

Nous avons fait un premier exemple assez simple à travers le fichier **first_dag.py** permettant juste d'afficher la date actuelle à travers une commande bash et le dag a été exécuté avec succès sur l'interface web de Airflow comme le montre l'image ci-dessous:

The screenshot shows the Airflow web interface with the URL `127.0.0.1:8080/taskinstance/list/?flt_3_dag_id=pramod_airflow_dag&flt_3_state=success`. The page displays a table of task instances for the DAG 'pramod_airflow_dag' in a 'success' state. The table has 15 columns: State, Dag Id, Task Id, Run Id, Map Index, Logical Date, Operator, Start Date, End Date, Duration, Job Id, Hostname, Unixname, Priority, and Weight. There are 4 records shown, all with a 'success' state and a 'BashOperator' operator.

State	Dag Id	Task Id	Run Id	Map Index	Logical Date	Operator	Start Date	End Date	Duration	Job Id	Hostname	Unixname	Priority	Weight
success	pramod_airflow_dag	print_date	scheduled__2022-11-04T00:00:00+00:00		2022-11-04, 00:00:00	BashOperator	2022-11-06, 17:14:18	2022-11-06, 17:14:18	<1s	6	momo	momo	2	
success	pramod_airflow_dag	print_date	scheduled__2022-11-05T00:00:00+00:00		2022-11-05, 00:00:00	BashOperator	2022-11-06, 17:14:19	2022-11-06, 17:14:19	<1s	7	momo	momo	2	
success	pramod_airflow_dag	sleep	scheduled__2022-11-04T00:00:00+00:00		2022-11-04, 00:00:00	BashOperator	2022-11-06, 17:14:20	2022-11-06, 17:14:20	5s	8	momo	momo	1	
success	pramod_airflow_dag	sleep	scheduled__2022-11-05T00:00:00+00:00		2022-11-05, 00:00:00	BashOperator	2022-11-06, 17:14:27	2022-11-06, 17:14:32	5s	9	momo	momo	1	

Contrairement à notre deuxième dag où seule la première tâche a été exécutée avec succès au moment où l'exécution de la deuxième tâche a échoué comme le montre l'image ci-dessous:

The screenshot shows the Airflow web interface with the URL `127.0.0.1:8080/taskinstance/list/?flt_3_dag_id=test_AI_model_dag&flt_3_state=failed`. The page displays a table of task instances for the DAG 'test_AI_model_dag' in a 'failed' state. The table has 15 columns: State, Dag Id, Task Id, Run Id, Map Index, Logical Date, Operator, Start Date, End Date, Duration, Job Id, Hostname, Unixname, Priority, and Weight. There are 2 records shown, both with a 'failed' state and a 'BashOperator' operator.

State	Dag Id	Task Id	Run Id	Map Index	Logical Date	Operator	Start Date	End Date	Duration	Job Id	Hostname	Unixname	Priority	Weight
failed	test_AI_model_dag	print_date	scheduled__2022-11-05T00:00:00+00:00		2022-11-05, 00:00:00	BashOperator	2022-11-06, 17:51:21	2022-11-06, 17:51:21	<1s	16	momo	momo	2	
failed	test_AI_dag	print_date	scheduled__2022-11-04T00:00:00+00:00		2022-11-04, 00:00:00	BashOperator	2022-11-06, 18:07:50	2022-11-06, 18:07:50	<1s	19	momo	momo	2	

Sur cette image nous voyons bien que l'exécution de la tâche `print_date`, qui consiste à exécuter un script python testant un modèle IA, a échoué. Notre problème ici c'est de ne pas être en mesure de voir le problème à travers les logs sur l'interface web mais plutôt nous sommes obligé d'aller sous dossier logs du répertoire airflow. Ainsi après avoir consulté les logs il se trouve qu'il y avait un problème avec le chemin vers notre modèle (No such file or directory error) et après correction la tâche `print_date` arrive à s'exécuter.