

TRAVAUX PRATIQUE TRANSVERSAL

PLATEFORME COLLABORATIVE D'AIDE A L'INDUSTRIALISATION DES MODELES PREDICTIFS

Réalisés par :

NITEKA Lys Ciella

DIALLO Elhadj Mamadou Foula

PAJANY CARPIN CAOUNDIN Allan

Master 2 - Mobilité, Base de Données et Intégration Système



Tuteur projet :

Mr. HAMMOUDI Alaaeddine

Chef de projet SI - Responsable d'activité data science & IA
chez PRO BTP Groupe



Tuteurs universitaires :

Mr. MOPOLLO-MOKE Gabriel

Mr. GALLI Grégory

Avril 2021

TABLE DES MATIERES

REMERCIEMENTS	3
INTRODUCTION GENERALE	4
1. PRESENTATION	5
1.1. L'ENTREPRISE	5
1.2. LE PROJET ET L'EQUIPE	5
2. TECHNOLOGIE	7
2.1. AIRFLOW	7
2.1.1. Présentation	7
2.1.2. Architecture et Composants	8
2.1.3. Concepts	9
2.2. SPRING BATCH	10
3. TRAVAUX REALISES	12
3.1. PROBLEMATIQUE	12
3.2. ARCHITECTURE PROPOSEE	12
3.3. ENVIRONNEMENTS	13
3.3.1. Docker	13
3.3.2. Environnement d'Airflow	14
3.4. IMPLEMENTATION	18
4. GESTION DE PROJET	20
4.1. METHODOLOGIE ET ORGANISATION	20
4.2. PLANNING	20
4.3. RISQUES ET ACTIONS	21
5. BILAN PERSONNEL	22
CONCLUSION	23
REFERENCES	24



REMERCIEMENTS

3

Nous tenons tout d'abord à remercier Mr. **HAMMOUDI Alaeddine**, notre tuteur projet pour son encadrement, son suivi et sa pédagogie tout au long de ce TPT.

Nous remercions également Mr. **MOPOLO-MOKE Gabriel**
Mr. **GALLI Grégory**, nos tuteurs universitaires pour leur suivi et leurs conseils.

INTRODUCTION GENERALE

4

La montée exponentielle de la quantité de données a poussé les entreprises à rechercher des outils adaptés pour le stockage et l'exploitation de ces données. La variété des données exploitées et l'exigence des performances au niveau des traitements ont demandé que ces outils soient assez puissants pour faire face aux besoins des entreprises. De plus, la majorité des applications des entreprises s'appuient sur des tâches qui sont exécutés automatiquement.

Le Batching (traitement par lots) répond à ce besoin en permettant d'exécuter découper ces tâches de façon périodique. Il va permettre ainsi un gain de performance non négligeable en fonction de la quantité de données à traiter.

Nous allons donc dans ce rapport présenter notre projet né d'un besoin de mise en place d'un système de Batch.

Nous présenterons d'abord l'entreprise ainsi que le projet, ensuite les technologies utilisées. Ensuite nous détaillerons les tâches mises en œuvre dans ce projet afin de travailler dans un environnement adéquat et terminerons sur la partie gestion de projet ainsi que nos perspectives.

1. PRESENTATION

5

1.1. L'entreprise

Créé en 1993, Pro BTP est un groupe de protection sociale au service du secteur professionnel du Bâtiment et Travaux Publics. Il s'agit d'un groupe géré par les représentants des entreprises et des salariés du BTP. Ayant un statut d'association à but non lucratif, les actionnaires détenteurs de fonds propre ne sont pas rémunérés.

Ses domaines d'activités sont :

- La retraite complémentaire
- La prévoyance
- La santé
- L'épargne
- Les assurances individuelles
- L'action sociale

1.2. Le projet et l'équipe

Le but de ce projet était de mettre en place une plateforme permettant d'accélérer la mise en production et la mise à disposition des travaux de :

- Data Visualisation
- Produits liés à Intelligence Artificielle
- Data en général, extraction/transformation

Cette plateforme devait permettre d'optimiser/standardiser le pré-traitement de données (collecter, agréger et auditer), faciliter la mise en production, mise à jour ainsi que la consommation de données d'un modèle prototype satisfaisant.

En raison d'un manque de temps pour la réalisation de cette plateforme, nous nous sommes orientés vers la mise en place d'un environnement de base et de la compréhension du système de Batching.



L'équipe est composée de :

- ✓ NITEKA Lys Ciella
- ✓ DIALLO Elhadj Mamadou Foula
- ✓ PAJANY CARPIN CAOOUNDIN Allan

6

N'étant pas familiarisés avec l'environnement Airflow, nous avons généralement travaillé ensemble sur les majeures parties du projet en évitant une trop grande répartition des tâches entre membres, car voulant tous comprendre et s'adapter à l'environnement.

2. TECHNOLOGIE

7

2.1. Airflow

2.1.1. Présentation

Airflow a été créé en 2014 par Maxime Beauchemin lorsqu'il travaillait chez Airbnb. C'est une plate-forme de gestion de flux de travail open source. Au moment de sa création la plateforme de l'action de vacances connaissait une énorme croissance et avait de plus en plus de difficulté à gérer l'énorme volume de données qui ne cessait de s'accroître.

Airbnb recrutait durant cette période des Data Scientists, Data Analystes et des Data Engineers pour essayer de mettre en place des solutions d'automatisations de processus en écrivant des batch planifiés. C'est ainsi que grâce à l'architecte et ingénieur en Business Intelligence, Maxime Beauchemin, Airflow vit le jour.

Il va permettre principalement aux équipes de développement de créer, surveiller les pipelines de données sous forme de lot et de manière itérative. Airflow s'imposera comme un standard dans le domaine du Data Engineering

Quelques éléments clés :

- En 2016 le projet a rejoint le programme d'incubation de la fondation Apache, d'où d'Apache Airflow.
- Intégré à AWS (Amazon Web Services)
- Inspiré d'outils internes de Facebook
- Utilisé par plus de 300 entreprises
- Plus de 800 collaborateurs (Airbnb, Facebook, Yahoo, Slack, ...)

2.1.2. Architecture et Composants

8

Ci-dessous l'architecture de base d'Airflow :

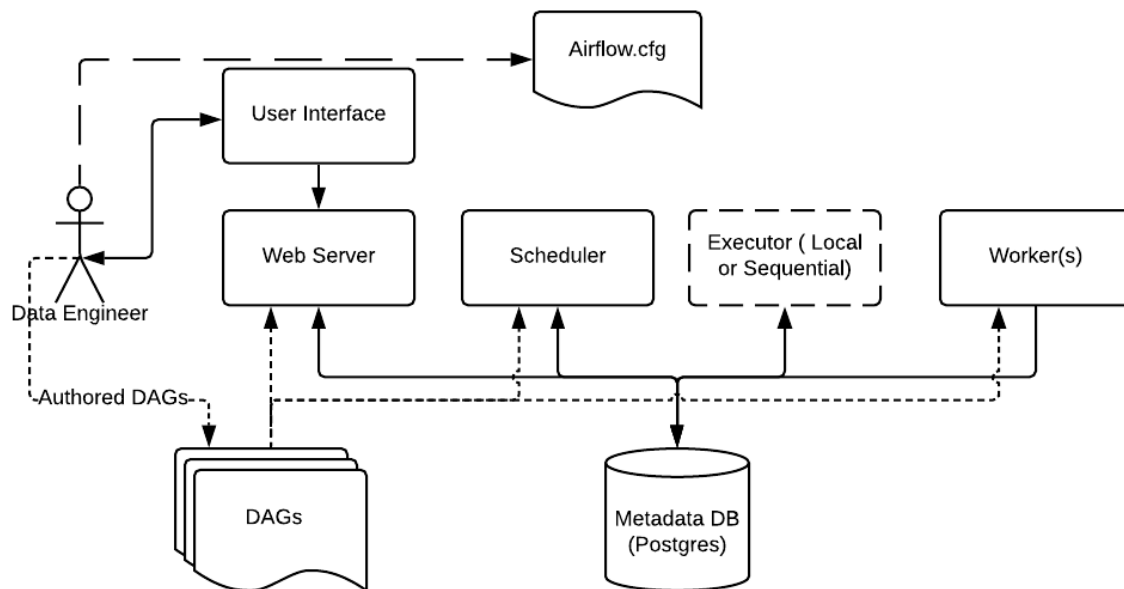


Image 1: Architecture Airflow

- **Web server** : Interface graphique permettant de :
 - Monitorer les données liées aux DAGs
 - Vérifier que nos tâches s'exécutent et se terminent correctement
 - Configurer des connexions vers des systèmes externes (par exemple base de données)
- **Scheduler** : Permet de planifier des tâches de nos DAGs ainsi que leurs dépendances
- **Executor** : Option permettant l'exécution des tâches (selon la plateforme), lié au Scheduler et permettant de déterminer le worker (processus) qui exécutera chaque tâche.
- **Worker** : Processus exécutant les tâches indiquées par l'executor
- **Metadata** : Base de données stockant les données liées aux DAGs, l'interface utilisateur.
- **Airflow.cfg** : Fichier de configuration d'Airflow contenant tous ses paramètres, options... C'est dans ce fichier qu'on peut par exemple choisir le type d'Executor utilisé.
- **DAGs** : Fichiers de codes en Python contenant la définition et le cheminement de nos tâches. Ces fichiers sont reconnus automatiquement par Airflow et sont exécutable via l'interface graphique.

2.1.3. Concepts

9

Airflow s'exécute dans environnement Python et se base sur des concepts fondamentaux :

- ✓ **DAG & Tasks** : Partie mère d'Airflow, les DAG (Directed Acyclic Graph pour Graphe Orienté Acyclique, un graphe dont aucun retour ou cycle n'est possible) sont des graphes qui vont nous permettre de définir l'ordre d'exécution de nos tâches (parallèle ou séquentielle), le tout sans avoir de boucle infinie.

Les DAGs se composent de Tasks (tâches) reliés entre elles de la façon suivante

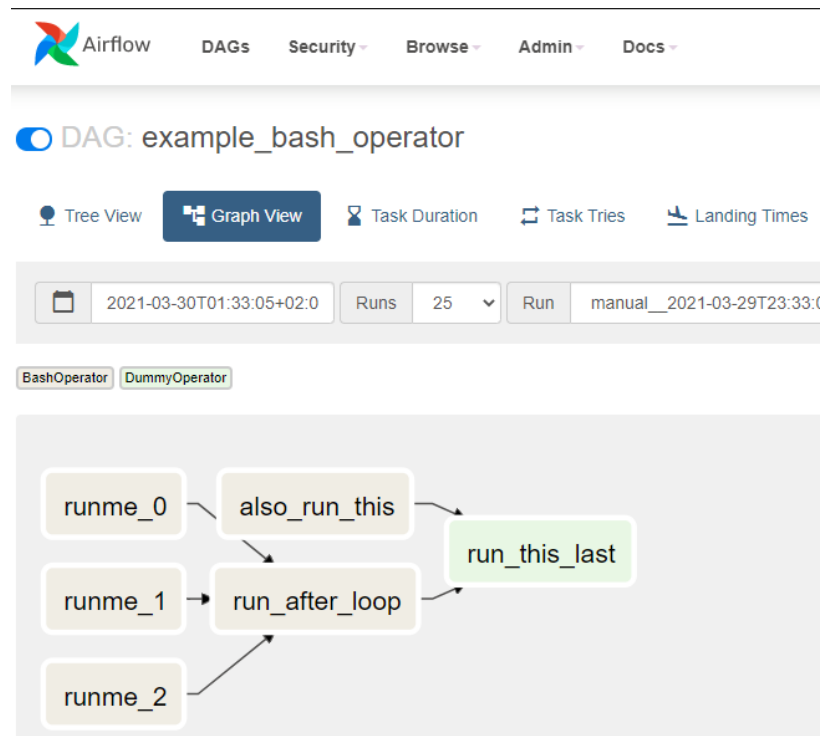


Image 2 : Interface graphique - DAGs

Un DAG se lance de manière régulière, lorsqu'il est en cours d'exécution, on l'appelle **DagInstance** et ses tâches **TaskInstances**

- ✓ **Operators** : Un Operator définit le type de tâche à exécuter. Nous retrouvons par exemple parmi ces tâches :
 - BashOperator : Exécute une commande Bash
 - PythonOperator : Exécute une fonction Python
 - EmailOperator : Effectue l'envoi d'un mail
 - DockerOperator : Exécute une commande dans un container Docker
 - HttpOperator : Effectue des requêtes http

➤ **MysqlOperator** : Permet de gérer une base de données MySQL

Il en existe bien évidemment une panoplie d'Operators.

✓ **Executors** : Les executors gèrent l'exécution des tâches dans Airflow. L'executor peut se servir de trois différentes options pour exécuter une tâche :

➤ **SequentialExecutor** : Option par défaut, les tâches sont lancées sur la machine local (worker) et en série.

➤ **LocalExecutor** : Lance plusieurs tâches en parallèle toujours sur la machine locale. Plus performant que le SequentialExecutor.

➤ **CeleryExecutor** : Permet le lancement de tâches en parallèle, mais cette fois-ci sur un ensemble de machines (pool de worker).

2.2. Spring Batch

Spring Batch est un Framework open source Java utilisé pour le traitement par lots. Il s'agit d'une solution légère et complète conçue pour permettre le développement d'applications par lots.

Il permet de palier à des problèmes récurrents lors de développement de batches :

- Productivité
- Gestion de gros volumes de données
- Fiabilité
- Réinvention de la roue.

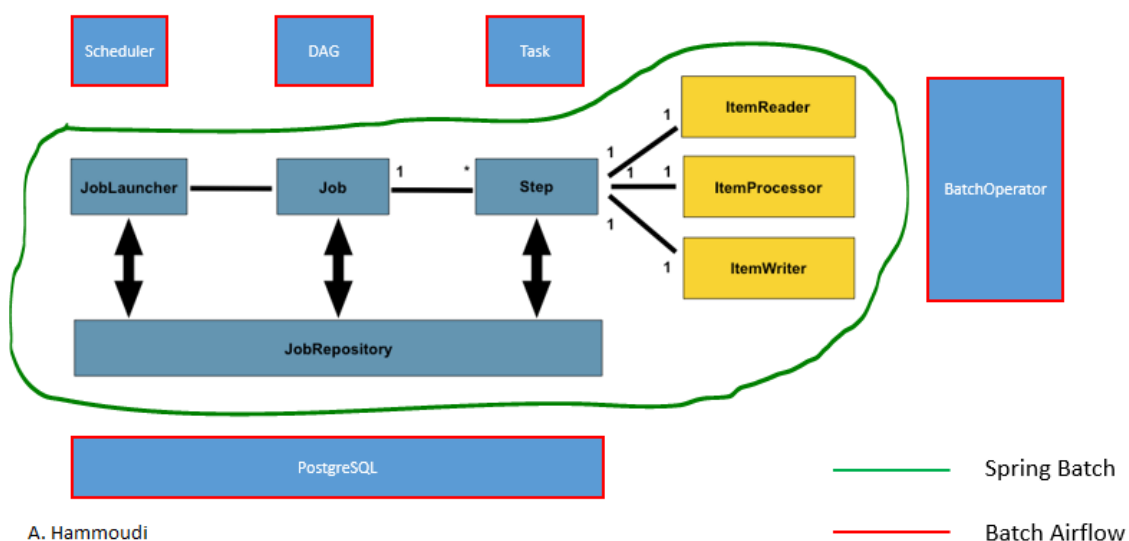


Image 3 : Architecture Spring Batch et Batch Airflow

Nous identifions différentes parties :

- **JobLauncher** : Permet de lancer le programme de trainement de lot (Batch). Il peut être configuré être déclenché de manière automatique ou manuel (après un évènement). Ici il se charge d'exécuter un Job
- **Job** : Représente la tâche à laquelle on délèguera la responsabilité d'un besoin métier qui sera traité dans le programme. Il se charge de lancer séquentiellement une ou plusieurs Step.
- **Step** : Se compose du besoin métier à traiter, il sera de définir les trois composants suivants :
 - **ItemReader** : Lecture de données d'entrées à traiter (csv, xml, ...)
 - **ItemProcessor** : Transforme les données lues.
 - **ItemWriter** : Sauvegarde les données transformées par l'ItemProcessor dans différents types de conteneurs (base de données, cloud, csv, ...)
- **JobRepository** : Enregistre les statistiques de monitoring (JobLauncher, le Job, Step) lors de chaque exécution.

Ci-dessous un tableau de représentation de Spring Batch dans Airflow :

Tableau 1 : Spring Batch et Airflow

Spring Batch	Airflow
JobLauncher	Scheduler
Job	DAG
Step	Task
ItemReader ItemProcessor ItemWriter	BashOperator
JobRepository	PostgreSQL

3. TRAVAUX REALISES

12

3.1. Problématique

Lorsqu'une erreur se produit dans le process d'exécution de Batch d'Airflow, il a la possibilité de recommencer à partir du Step qui a cassé et non recommencé tout le Batch.

Par exemple, lors de l'édition d'une facture où l'on doit insérer 10 produits, un Batch « **EditerFacture** » est mis en place et dans ce Batch nous retrouvons des Step permettant l'édition par lot de la facture.

Si une erreur se produit dans un Step donné, par exemple un Step « **insérer_produits** », Airflow pourra détecter quel Step a cassé et le recommencer. Cependant, ce qu'Airflow ne peut pas faire, c'est de reprendre à la ligne où l'erreur s'est produite. Si l'erreur se produit lors de l'insertion du produit 6, Airflow ne pourra pas recommencer à la ligne d'insertion 6, comme il recommencera le Step « **insérer_produits** », nous reviendrons donc à le relancer entièrement.

Sur un petit volume de donnée, cela reste négligeable, mais lorsque nous disposons de milliers voir de millions de lignes, nous risquons de perdre rapidement en productivité.

Comment permettre à Airflow de reprendre à partir d'une ligne de Step ?

3.2. Architecture proposée

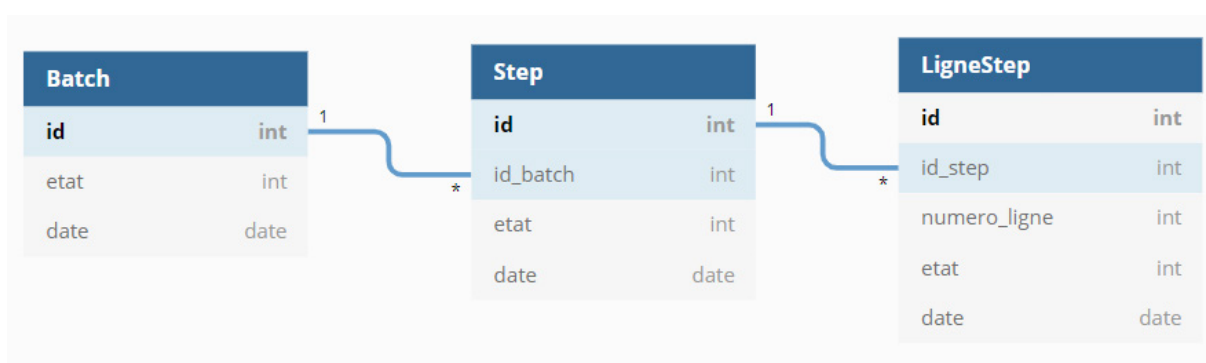


Image 4 : Architecture proposée

Dans cette architecture, nous mettons en œuvre trois tables et de manière hiérarchique, nous avons d'abord la table Batch, ensuite Step liée au Batch et enfin LigneStep liée à un Step en particulier. Chacune de ces tables possède une colonne « etat », cette colonne se chargera de détecter si le traitement d'une ligne s'est bien déroulé. Ces états sont dépendant entre eux,

i.e. si dans une LigneStep l'état correspond à un échec, cet état sera remonté dans le Step et le Batch correspondant pour en avertir le système et lui permettre de reprendre à la ligne où l'état est mis en échec ou bien pour des raisons de maintenance et d'optimisation afin de trouver la raison de cet échec.

Un des problèmes qui a été soulevé par Mr. GALLI Grégory avec cette architecture est l'insertion des données dans les tables et donc les performances ainsi que le stockage. N'insérer que la ligne où l'erreur s'est produite serait beaucoup moins coûteux en temps et en espace.

3.3. Environnements

Nous parlerons dans cette partie de la mise en place ainsi que de la prise en main de l'environnement Airflow. Nous aborderons les points sur lesquels nous avons majoritairement travaillé. Le sujet et les concepts étant nouveau pour nous, nous avons pris un certain temps avant de pouvoir bien nous adapter cette technologie.

3.3.1. Docker

Docker est un logiciel open source permettant de lancer des applications dans des conteneurs spécifiques, rassemblant ainsi les éléments nécessaires à leur bon fonctionnement (outils système, bibliothèques, ...). Il facilite donc l'exécution de plusieurs processus et applications séparément les uns des autres afin d'optimiser leur utilisation dans un environnement qui leur seront propre. Nous pourrons par la suite communiquer avec nos conteneurs où les faire communiquer ensemble à l'aide de ports réseau.

Afin de pouvoir travailler avec Airflow, nous avons utilisé un fichier de commande (DockerFile) fournis par Apache Airflow que nous avons retravaillé pour obtenir un environnement adapté à nos besoins.

Nous avons en outre :

- ✓ Créé des volumes (répertoires) pour nous permettre d'accéder à partir de la machine locale à nos fichiers conteneurisés
 - **Dags** : Dossier contenant script python définissant nos DAGs
 - **Données** : Contient les fichiers de données à traiter

- **Résultats** : Contient les fichiers de sorties des résultats de nos traitements
 - **Logs** : Dossier stockant les fichier logs
 - **Mysql** : Dossier stockant les informations liées à notre base de données
- ✓ Remplacé le conteneur de base de données par défaut (PostgreSQL) par un conteneur MySQL et modifié certains paramètres de connexion pour prendre en compte le nouveau SGBD.

Après l'exécution du DockerFile nous obtenons ceci dans Docker :

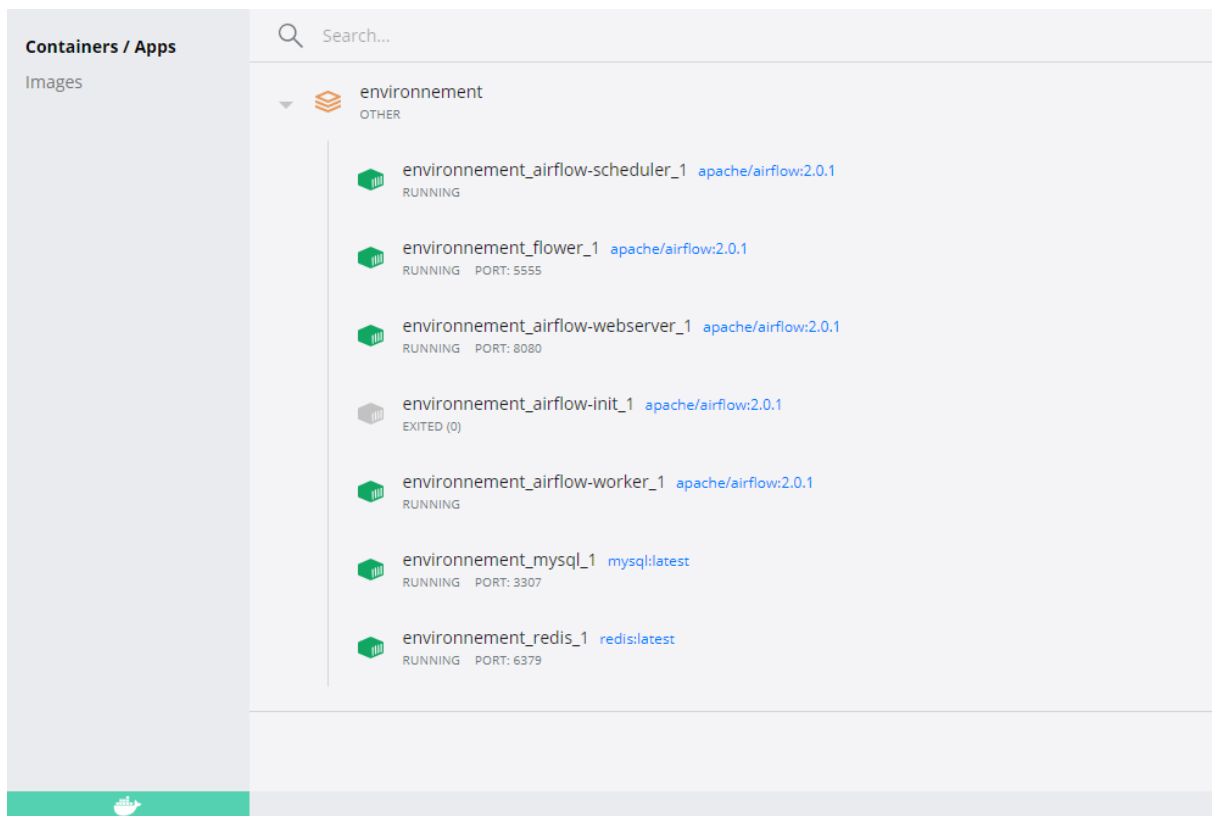
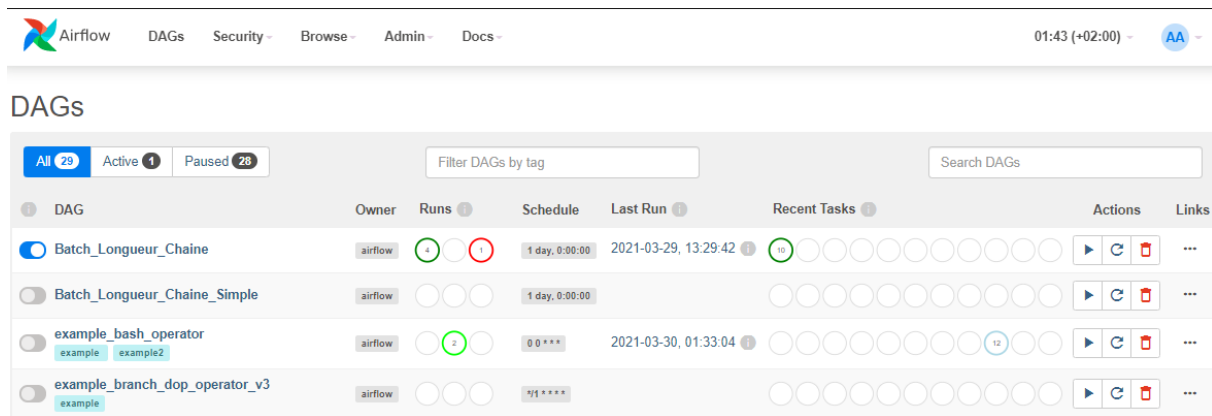


Image 5 : Nos conteneurs docker pour notre environnement Airflow

3.3.2. Environnement d'Airflow

Lorsque l'on lance le web server, nous accédons à une page web d'accueil d'Airflow nous permettant d'effectuer d'exécuter, gérer, monitorer nos DAGs.

Nous arrivons tout d'abord sur une vue d'ensemble de tous nos DAGs



15

Image 6 : Airflow - Page d'accueil

Ici nous nous pouvons obtenir les informations générales liées à tous les DAGs créés dans notre environnement tel que :

- **DAG** : Nom du DAG
- **Owner** : Propriétaire
- **Runs** : Statuts des précédentes exécutions de nos DAGs. Trois types d'états des exécutions de nos DAGs ainsi que le nombre d'exécutions.
 - Success (vert foncé)
 - Running (vert clair)
 - Failed (rouge)
- **Schedule** : Le type de planification
- **Last Run** : Dernière date de lancement
- **Récents Tasks** : Statut des tâches des DAGs en cours d'exécution ou de la dernière exécution.
- **Actions** : Lancement, rafraichissement et suppression des DAGs.
- **Links** : Redirection vers un des outils de monitoring choisi (Tree view, Graph view, ...).

Nous avons utilisé différents outils de monitoring afin de gérer nos DAGs :

- ✓ **Graph View** : Permet de visualiser les dépendances entre tâches d'un DAG et leur état actuel pour une exécution spécifique. Chaque nœud d'un DAG représente une tâche. Chacun des nœuds est défini par un opérateur (Operator) spécifique.

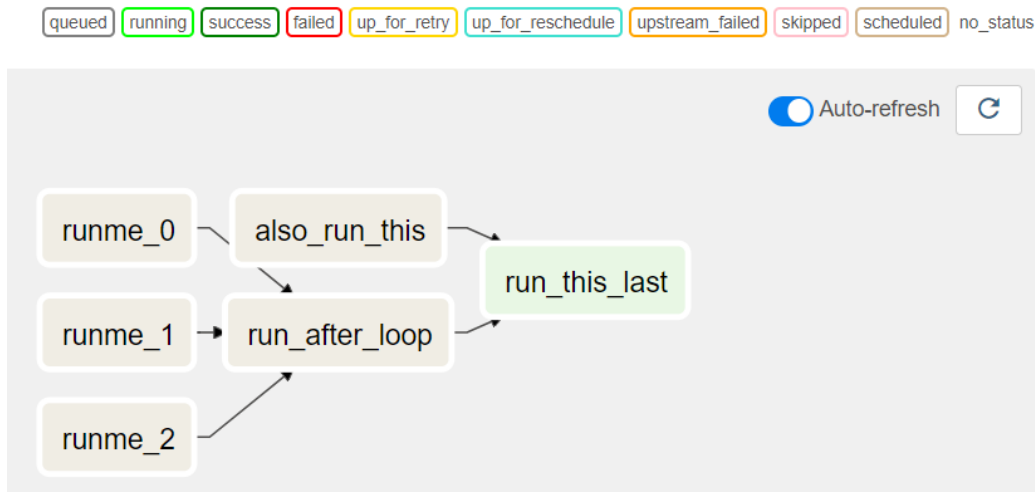


Image 7 : Airflow - Graphe View

- ✓ **Tree View** : Représente l'arborescence de DAG sous forme d'arbre et qui s'étend dans le temps.

☒ DAG: example_bash_operator

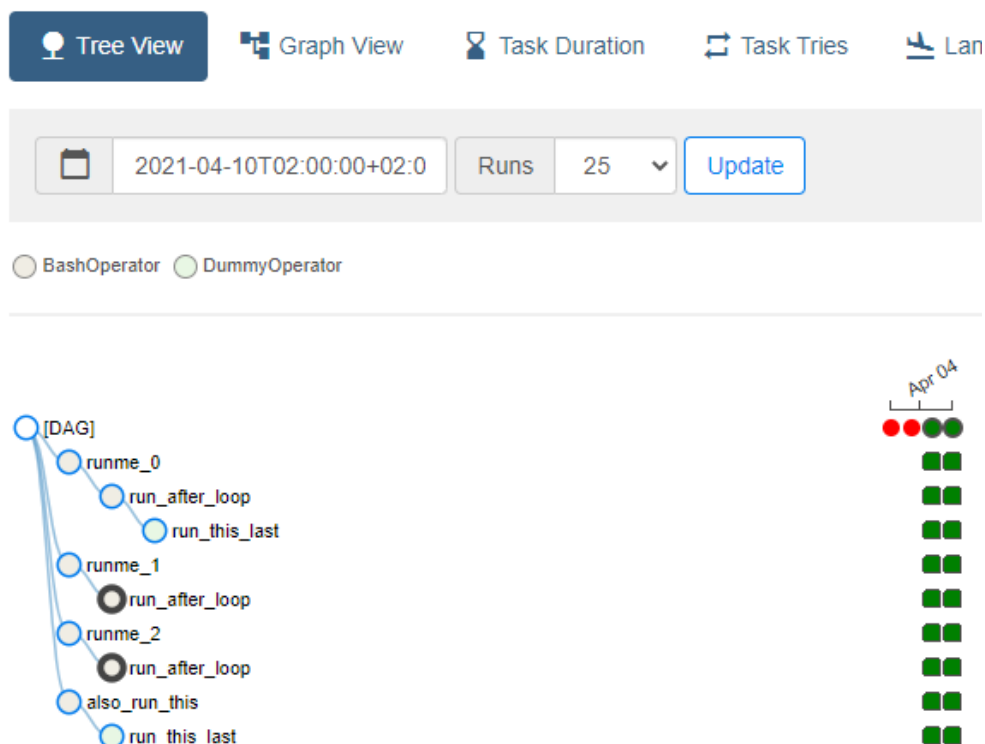
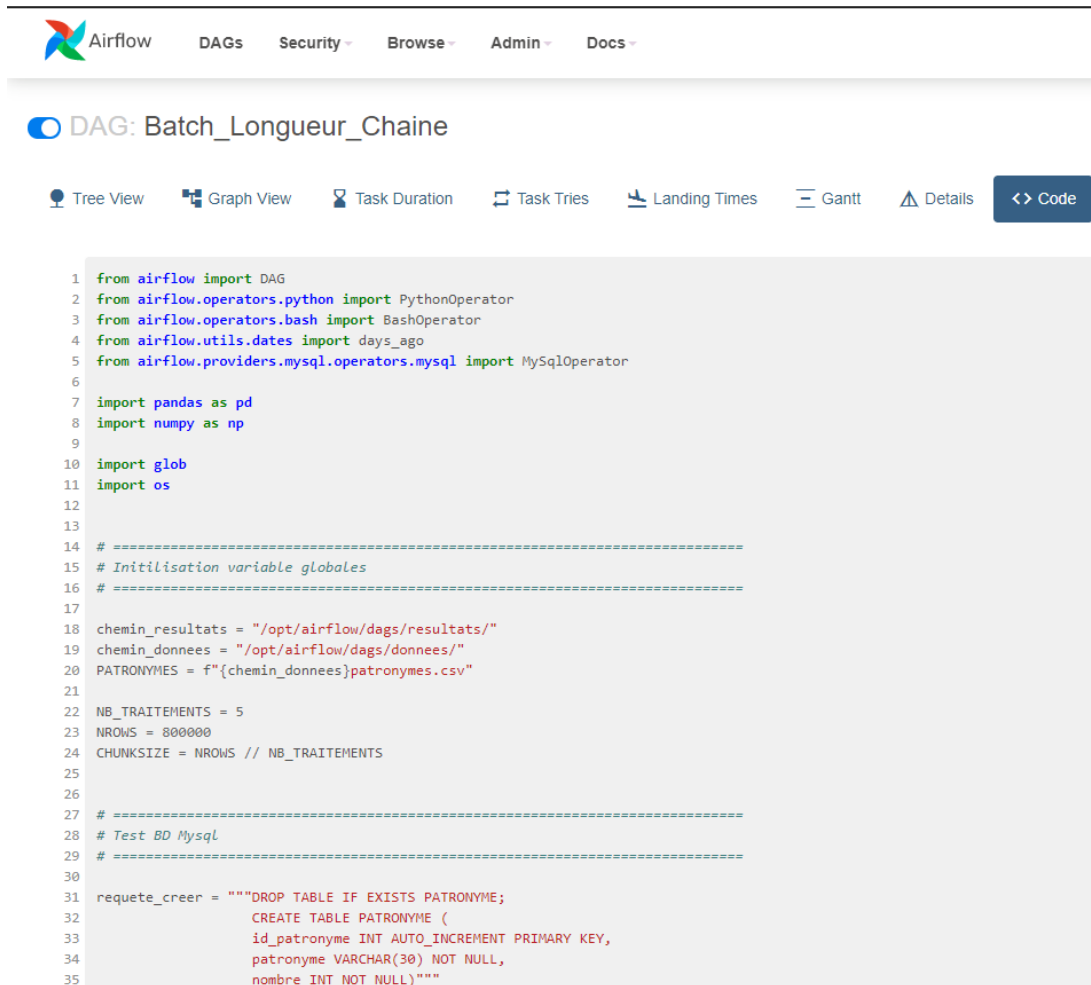


Image 8 : Airflow - Tree View

- ✓ **Code** : Permet d'afficher rapidement le code source d'un DAG (Python). Il permet notamment de vérifier que nos nouvelles modifications ont bien été prises avant la réexécution d'un DAG.



```
1 from airflow import DAG
2 from airflow.operators.python import PythonOperator
3 from airflow.operators.bash import BashOperator
4 from airflow.utils.dates import days_ago
5 from airflow.providers.mysql.operators.mysql import MySQLOperator
6
7 import pandas as pd
8 import numpy as np
9
10 import glob
11 import os
12
13
14 # =====
15 # Initilisation variable globales
16 # =====
17
18 chemin_resultats = "/opt/airflow/dags/resultats/"
19 chemin_donnees = "/opt/airflow/dags/donnees/"
20 PATRONYMES = f"{chemin_donnees}patronymes.csv"
21
22 NB_TRAITEMENTS = 5
23 NROWS = 800000
24 CHUNKSIZE = NROWS // NB_TRAITEMENTS
25
26
27 # =====
28 # Test BD Mysql
29 # =====
30
31 requete_creer = """DROP TABLE IF EXISTS PATRONYME;
32                     CREATE TABLE PATRONYME (
33                         id_patronyme INT AUTO_INCREMENT PRIMARY KEY,
34                         patronyme VARCHAR(30) NOT NULL,
35                         nombre INT NOT NULL)"""
```

Image 9 : Airflow - Code source d'un DAG

✓ Log : Affichage de log d'exécutions d'une tâche d'un DAG

```
*** Reading local file: /opt/airflow/logs/Batch_Longueur_Chaine/concatener_data/2021-03-30T13:36:34.684505+00:00/1.log
[2021-03-30 13:36:55,677] {taskinstance.py:851} INFO - Dependencies all met for <TaskInstance: Batch_Longueur_Chaine.concatener_data 2021-03-30T13:36:34.684505+00:00 [
[2021-03-30 13:36:55,690] {taskinstance.py:851} INFO - Dependencies all met for <TaskInstance: Batch_Longueur_Chaine.concatener_data 2021-03-30T13:36:34.684505+00:00 [
[2021-03-30 13:36:55,690] {taskinstance.py:1042} INFO -
-----
[2021-03-30 13:36:55,690] {taskinstance.py:1043} INFO - Starting attempt 1 of 1
[2021-03-30 13:36:55,691] {taskinstance.py:1044} INFO -
-----
[2021-03-30 13:36:55,707] {taskinstance.py:1063} INFO - Executing <Task(BashOperator): concatener_data> on 2021-03-30T13:36:34.684505+00:00
[2021-03-30 13:36:55,712] {standard_task_runner.py:52} INFO - Started process 67 to run task
[2021-03-30 13:36:55,716] {standard_task_runner.py:76} INFO - Running: ['airflow', 'tasks', 'run', 'Batch_Longueur_Chaine', 'concatener_data', '2021-03-30T13:36:34.684
[2021-03-30 13:36:55,716] {standard_task_runner.py:77} INFO - Job 67: Subtask concatener_data
[2021-03-30 13:36:55,767] {logging_mixin.py:104} INFO - Running <TaskInstance: Batch_Longueur_Chaine.concatener_data 2021-03-30T13:36:34.684505+00:00 [running]> on hos
[2021-03-30 13:36:55,819] {taskinstance.py:1257} INFO - Exporting the following env vars:
AIRFLOW_CTX_DAG_OWNER=airflow
AIRFLOW_CTX_DAG_ID=Batch_Longueur_Chaine
AIRFLOW_CTX_TASK_ID=concatener_data
AIRFLOW_CTX_EXECUTION_DATE=2021-03-30T13:36:34.684505+00:00
AIRFLOW_CTX_DAG_RUN_ID=manual__2021-03-30T13:36:34.684505+00:00
[2021-03-30 13:36:55,820] {bash.py:135} INFO - Tmp dir root location:
/tmp
[2021-03-30 13:36:55,822] {bash.py:158} INFO - Running command: cat /opt/airflow/dags/resultats/*.csv >> /opt/airflow/dags/resultats/patronymes_tailles.csv
[2021-03-30 13:36:55,837] {bash.py:169} INFO - Output:
[2021-03-30 13:36:55,966] {bash.py:177} INFO - Command exited with return code 0
[2021-03-30 13:36:55,994] {taskinstance.py:1166} INFO - Marking task as SUCCESS. dag_id=Batch_Longueur_Chaine, task_id=concatener_data, execution_date=20210330T133634,
[2021-03-30 13:36:56,027] {taskinstance.py:1220} INFO - 1 downstream tasks scheduled from follow-on schedule check
[2021-03-30 13:36:56,048] {local_task_job.py:146} INFO - Task exited with return code 0
```

Image 10 : Airflow - Logs d'une tâche

Vous pouvez retrouver sur ce [lien](#) une version d'Airflow 1.10 pour vous donner un aperçu de l'environnement et de l'interface graphique. Le temps d'exécution d'un DAG y est assez long.

3.4. Implémentation

18

Avant de pouvoir réaliser l'implémentation totale de notre architecture, nous avons mis en place une implémentation orienté fichiers (aucun travail dans la base de données) pour d'abord commencer par simuler les étapes de travail sur nos données et bien comprendre ce qu'est le traitement par lots.

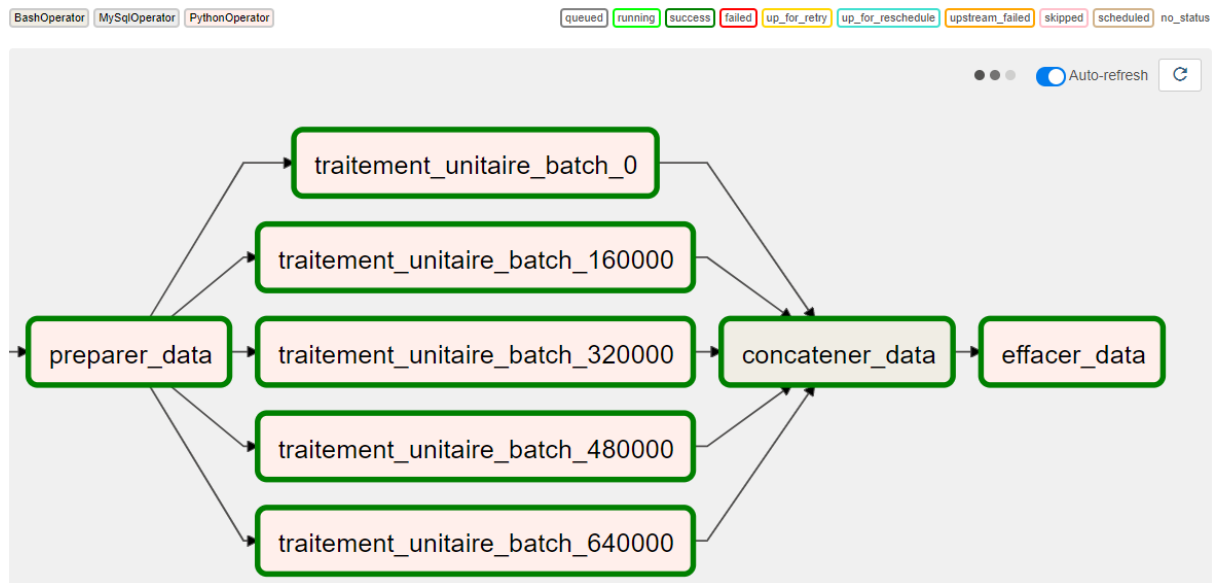


Image 11 : Implémentation finale

Pour cela, nous avons travaillé sur un fichier contenant plus de 800 000 lignes de patronymes. Le fichier est défini ainsi :

Tableau 2 : Tableau descriptif de la source de donnée "patronyme.csv"

ATTRIBUT	TYPE	DESCRIPTION	EXEMPLE
patronyme	Caractères	Le patronyme	Jackass, Duchemin, ...
count	Numérique	Le nombre de fois que ce patronyme est compté (évite les doublons)	2, 15, ...

Le but ici était de charger ce fichier de patronyme et de rajouter une nouvelle colonne correspondant à la taille d'un patronyme, le tout en essayant de respectant au mieux le concept de traitements par lots.

Nous avons donc mis en place les tâches suivantes :

19

Tableau 3 : Tableau descriptif des tâches

TACHE	OPERATOR	DESCRIPTION
préparer_data	PythonOperator	Chargement des données
traitement_unitaire_batch_i	PythonOperator	Récupère les données par lot de « n », calcul la taille de chaque patronyme et crée un nouveau fichier temporaire correspondant au lot contenant la nouvelle colonne « taille ». Dans notre exemple, chaque lot possède 160 000 lignes
concatener_data	BashOperator	Commande Bash permettant de concaténer dans un seul fichier de résultat, les nouveaux fichiers générés par le traitement_unitaire_batch_i
effacer_data	PythonOperator	Supprime les fichiers temporaires créés par les traitement_unitaire_batch_i

Notre DAG commence donc par charger le fichier, ensuite nous appliquons par lot de n , un traitement sur les n lignes de ce lot en calculant la taille de chaque patronyme et en enregistrant dans différents fichiers temporaire les résultats du traitement (l'ajout d'une colonne comptant la taille du patronyme), nous concaténons ensuite chacun de ces fichiers dans un seul et même fichier résultat et enfin nous supprimons les fichiers créés lors du traitement.

Nous avons mis en place par la suite une liaison entre Airflow et MySQL afin de pouvoir travailler avec des bases de données, cependant par manque de temps nous n'avons malheureusement pas pu aller plus loin dans notre implémentation.

4. GESTION DE PROJET

20

4.1. Méthodologie et Organisation

Concernant cette partie, nous faisons avec notre tuteur de projet des réunions hebdomadaires généralement les lundis de 13h à 14h pour faire le point sur les exigences qui ont été réalisés, établir les objectifs pour la séance suivant. Notre tuteur profitait également de ces réunions pour répondre à nos interrogations et nous présenter les différents concepts utilisés pour la bonne réussite de nos tâches.

Nous avons utilisé GitHub pour héberger notre projet et garder une traçabilité de l'évolution de ce dernier.

4.2. Planning

Ci-dessous un planning résumant les tâches effectuées par séances :

Tableau 4 : Planning des tâches réalisées

EXIGENCES	SEANCE 1-2	SEANCE 3-4	SEANCE 5	SEANCE 6-8	SEANCE 9-10
Mise en place de l'environnement Airflow					
Prise en main Airflow					
Etude architecture					
Test de nos différentes implémentations (découpage Tasks, traitement des données, ...)					
Ajout serveur MySQL, travaille sur les présentations/ Rapport/Documentation du projet					

4.3. Risques et Actions

21

Ci-après les risques anticipés ainsi que les actions misent en place :

Tableau 5 : Risques et Actions

Risques	Actions
Technologies non adaptées aux matériels	Mise en place d'environnements légers et portable (Docker)
Ne pas tenir les délais de réalisation d'un besoin	Travail sur notre temps personnel
Objectifs irréalisables	Découpage selon l'avancement avec descriptifs détaillés du travail à réaliser
Mauvaise communication	Création d'un groupe de travail collaboratif (Discord, Hangouts)

5. BILAN PERSONNEL

22

Tout au long du TPT, nous nous sommes impliqués dans notre projet pour avoir un résultat fini dans les temps impartis, et ce malgré le faible nombre de séances dont nous disposions. Nous avons également pu garder une très bonne communication avec Mr. Hammoudi afin d'avoir un encadrement et des suivis réguliers. Et enfin, nous avons pu mettre en place une bonne synergie pour le travail en équipe.

Cependant, nous n'avions pas pu utiliser pleinement les outils de suivi de gestion de projet fournis par nos tuteurs universitaires, tel que Space (plateforme de collaboration pour les développements logiciels et gestion de projet, d'équipes et de communication).

CONCLUSION

23

Malgré le manque de temps, grâce à ce projet nous avons pu découvrir de nouvelles technologies, notamment Airflow avec son système de Batch, utiliser des container Docker d'une manière plus avancé afin de mettre en place un environnement robuste et exécutable sur de multiple plateforme.

Cela nous a également sensibilisé sur l'avantage du découpage de tâches afin de les rendre plus scalable, maintenable et performant.



REFERENCES

- <https://docs.spring.io/spring-batch/docs/current/reference/html/step.html>
- <https://airflow.apache.org/docs/apache-airflow/stable/index.html>
- <https://www.kaherecode.com/tutorial/commencer-la-programmation-avec-spring-batch>
- https://airflow.apache.org/docs/apacheairflow/stable/_api/airflow/operators/index.html
- <https://airflow.apache.org/ecosystem/>
- <https://www.astronomer.io/guides/airflow-ui>
- <https://airflow-tutorial.readthedocs.io/en/latest/first-airflow.html>
- <https://medium.com/@itunpredictable/apache-airflow-on-docker-for-complete-beginners-cf76cf7b2c9a>
-

Table des images

Image 1: Architecture Airflow	8
Image 2: Interface graphique - DAGs	9
Image 3: Architecture Spring Batch et Batch Airflow	10
Image 4: Architecture proposée	12
Image 5: Nos conteneurs docker - Airflow	14
Image 6 : Airflow - Page d'accueil	15
Image 7: Airflow - Graphe View	16
Image 8: Airflow - Tree View	16
Image 9: Code lié au DAG- Airflow	17
Image 10 : Logs d'une tâche - Airflow	17
Image 11: Implémentation finale	18

Table des tableaux

Tableau 1 : Spring Batch et Airflow	11
Tableau 2 : Tableau descriptif des tâches	19
Tableau 3 : Planning des tâches réalisées	20
Tableau 4 : Risques et Actions	21