

Rapport du Projet d'IPBD

Tessa DEPAOLI

Xavier DE WEERD

Alexandre FORT

Table des matières

Analyse des tendances sur les animés	2
1. Objectifs et mise en place	2
Description des données et objectif	2
Dépôt du projet et paquets	2
Les données	3
2. Nettoyage des données et pré-analyse	6
Environnement Python	6
Nettoyage des données	6
Premières visualisations	7
3. Hadoop et Hive	7
Démarrage	7
Import des données dans HDFS	8
Création de la database du Hive	8
Résolution du problème de connexion à Hive	11
4. Utilisation d'Apache Superset	11
Premier essai : version dev	12
Deuxième essai : version non-dev	12
Connexion à Hive	12
Création de Dashboards	13
5. Pour aller plus loin	13
6. Google Cloud	13
Chargement des données	14

Analyse des tendances sur les animés

1. Objectifs et mise en place

La consigne du projet est la suivante : Construction d'une plateforme Big-Data sur la base de données issues d'une bibliothèque Open Data choisie par chaque groupe d'étudiants et proposition d'un ensemble d'analyses représentatives du jeu de données de la plateforme

Description des données et objectif

Nous avons cherché des données d'au moins 2 Go, sans images car le traitement n'est pas le même.

Le dataset que nous avons choisi fait 8 Go et porte sur les animés, trop bien : <https://www.kaggle.com/datasets/dataset/>. (Dans les faits on a utilisé que 3 fichiers, donc on est plutôt sur 2/3 Go mais c'est bien quand même!)

Il s'agit de données sur les animés, les profils d'utilisateurs et les notations des utilisateurs de la plateforme [MyAnimeList](#). MyAnimeList permet de constituer sa liste de visionnage (animes déjà vus, en cours, abandonnées,...) et de les noter.

Notre but est d'analyser ces données afin de trouver des tendances ou/et corrélation dans les données.

Pour cela, on utilisera les outils suivants :

- Python : pandas, seaborn
- Docker
- Hadoop
- Hive
- Superset
- Google Cloud

Dépôt du projet et paquets

On travaille d'abord sur la VM. On s'assure d'avoir les paquets nécessaires :

```
1 sudo dnf install git wget unzip python3
```

Pour les utilisateurs :

On récupère le git du projet. Ce dernier contient les scripts pour télécharger les données et lancer le cluster Hadoop et Hive :

```
1 cd ~  
git clone https://git.iens.net/de-weerd2022/projet-ipbd24.git
```

Pour les développeurs :

Afin de pouvoir travailler et effectuer des changements sur le dépôt du projet depuis la VM, on ajoute une clé ssh de déploiement sur gitlab dans "Settings>Repository>Deploy keys".

On doit pour cela générer une clé ssh, qu'on copie ensuite pour chaque utilisateur en prenant garde à donner les bonnes permissions.

```

sudo - su
2 ssh-keygen -t rsa -b 4096
  cp /root/.ssh/id_rsa* /home/xavier/.ssh
4 chown xavier:ensiie /home/xavier/.ssh/*
  cp /root/.ssh/id_rsa* /home/tezza/.ssh
6 chown tezza:ensiie /home/tezza/.ssh/*
  cp /root/.ssh/id_rsa* /home/alexandre/.ssh
8 chown alexandre:ensiie /home/alexandre/.ssh/*

```

On peut désormais cloner le dépôt en utilisant ssh, et push des modifications (pour les développeurs, en tant qu'utilisateur du dépôt le lien du git https suffit) :

```
git clone git@git.iiens.net:de-weerd2022/projet-ipbd24.git
```

La version accélérée du déploiement est faisable en lisant uniquement la dernière section TL;DR. Un grand nombre de script incluant les commandes qui vont suivre permettant cette accélération.

Les données

Afin de télécharger les données sur kaggle, on a obtenu une commande `wget` permettant de télécharger le fichier à l'aide d'une extension de navigateur. Kaggle n'offrant pas de lien direct de téléchargement.

Cette commande est dans le script `download_dataset.sh` :

```

wget --header="Host: storage.googleapis.com" --header="User-Agent:
  Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
  (KHTML, like Gecko) Chrome/124.0.0.0 Safari/537.36"
--header="Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp
--header="Accept-Language:
  fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7,zh-TW;q=0.6,zh-CN;q=0.5,zh;q=0.4"
--header="Referer: https://www.kaggle.com/"
"https://storage.googleapis.com/kaggle-data-sets/3384322/6207733/bundle/arc
-c -O 'dataset.zip'

```

Les scripts en questions sont disponibles sur le git. Cette requête `wget` dépend d'une données d'authentification qui n'est pas permanente et ne fonctionnera donc pas indéfiniment. Il faudrait faire usage du git contenant les scripts de récupération des données directement si on souhaite une méthode intemporelle.

On télécharge l'archive des données avec la requête `wget` puis on l'extrait :

```

1 cd ~/projet-ipbd24
  ./download_dataset.sh

```

Les données sont désormais dans le dossier `dataset/`, on compte 6 csv :

- `anime-dataset-2023.csv`
- `anime-filtered.csv`
- `final_animatedataset.csv`

- `user-filtered.csv`
- `users-details-2023.csv`
- `users-score-2023.csv`

Les fichiers `*-filtered.csv` et `final_*.csv` permettent d'être plus rapidement prêts à l'emploi en fonction de l'analyse qu'on compte faire dessus.

On a fait le choix de se limiter aux tables `anime-dataset-2023`, `user-details-2023` et `user-filtered`.

On supprime les autres csv.

```
cd ~/projet-ipbd24/files/dataset
2 rm anime-filtered.csv final_animatedataset.csv users-details-2023.csv
```

Voici une description de nos tables :

- `anime-dataset-2023.csv`
 - `anime_id` : ID unique pour chaque anime.
 - `Name` : Le nom de l'anime dans sa langue originelle.
 - `English name` : Le nom anglais de l'anime.
 - `Other name` : Nom d'origine ou titre de l'anime (peut être en japonais, chinois ou coréen).
 - `Score` : Le score ou la note attribuée à l'anime.
 - `Genres` : Les genres de l'anime, séparés par des virgules.
 - `Synopsis` : Brève description ou résumé de l'intrigue de l'anime.
 - `Type` : Le type de l'anime (par exemple, série télévisée, film, OVA, etc.).
 - `Episodes` : Le nombre d'épisodes de l'anime.
 - `Aired` : Les dates de diffusion de l'anime.
 - `Premiered` : La saison et l'année de la première de l'anime.
 - `Status` : Le statut de l'anime (par exemple, Fini de diffuser, En cours de diffusion, etc.).
 - `Producers` : Sociétés de production ou producteurs de l'anime.
 - `Licensors` : Les concédants de licence de l'anime (par exemple, les plateformes de diffusion en continu).
 - `Studios` : Les studios d'animation qui ont travaillé sur l'anime.
 - `Source` : Le matériel source de l'anime (par exemple, manga, light novel, original).
 - `Duration` : La durée de chaque épisode.
 - `Rating` : La classification par âge de l'anime.
 - `Rank` : Le rang de l'anime en fonction de sa popularité ou d'autres critères.
 - `Popularity` : Le rang de popularité de l'anime.
 - `Favorites` : Le nombre de fois où l'anime a été marqué comme favori par les utilisateurs.
 - `Scored By` : Le nombre d'utilisateurs qui ont noté l'anime.
 - `Members` : Le nombre de membres qui ont ajouté l'anime à leur liste sur la plateforme.
 - `Image URL` : URL de l'image ou de l'affiche de l'anime. L'ensemble de données fournit des informations précieuses pour analyser et comprendre les caractéristiques, les classements, la popularité et l'audience de diverses émissions d'anime. En utilisant cet ensemble de données, il est possible d'effectuer un large éventail d'analyses, y compris l'identification des animes les mieux notés, l'exploration des genres les plus populaires, l'examen de la distribution des notes et l'obtention d'informations sur les préférences et les tendances des téléspectateurs. En outre, l'ensemble de données facilite la création de

systèmes de recommandation, l'analyse des séries temporelles et le regroupement pour approfondir les tendances des anime et le comportement des utilisateurs.

- **users-details-2023.csv**

- Mal ID : ID unique pour chaque utilisateur. (MAL = MyAnimeList)
- Username : Le nom d'utilisateur de l'utilisateur.
- Gender : Le sexe de l'utilisateur.
- Birthday : La date de naissance de l'utilisateur (au format ISO).
- Location : La localisation ou le pays de l'utilisateur.
- Joined : La date à laquelle l'utilisateur a rejoint la plateforme (au format ISO).
- Days Watched : Le nombre total de jours que l'utilisateur a passé à regarder des animes.
- Mean Score : La note moyenne attribuée par l'utilisateur aux animes qu'il a regardés.
- Watching : TLe nombre d'anime actuellement regardés par l'utilisateur.
- Completed : Le nombre d'anime terminés par l'utilisateur.
- On Hold : Le nombre d'anime en attente par l'utilisateur.
- Dropped : Le nombre d'anime abandonnés par l'utilisateur..
- Plan to Watch : Le nombre d'animes que l'utilisateur prévoit de regarder dans le futur.
- Total Entries : Le nombre d'animes que l'utilisateur prévoit de regarder dans le futur.
- Rewatched : Le nombre d'anime revus par l'utilisateur.
- Episodes Watched : Le nombre total d'épisodes regardés par l'utilisateur. L'ensemble de données User Details fournit des informations précieuses pour l'analyse du comportement et des préférences des utilisateurs sur la plateforme d'anime. En examinant les scores moyens et les genres d'anime, vous pouvez obtenir des informations sur les préférences des utilisateurs. Les utilisateurs peuvent être segmentés en différents groupes en fonction de leur comportement de visionnage, comme les utilisateurs actifs et les spectateurs occasionnels. Des systèmes de recommandation personnalisés peuvent être mis en place à partir des listes de films achevés et de ceux que les utilisateurs ont l'intention de regarder. L'analyse basée sur la localisation révèle la popularité des animes et l'engagement des utilisateurs dans différents pays. Il est possible d'identifier les tendances en matière de comportement de visionnage, de fidélisation des utilisateurs et de différences entre les sexes en ce qui concerne les préférences en matière d'anime. En outre, vous pouvez explorer les habitudes de relecture et effectuer des analyses de séries chronologiques pour comprendre les schémas d'engagement des utilisateurs au fil du temps.

- **users-filtered.csv**

- mal_id : ID unique pour chaque utilisateur.
- anime_id : Identifiant unique pour chaque anime.
- rating : La note attribuée par l'utilisateur à l'anime. L'ensemble de données sur les notes attribuées par les utilisateurs permet d'effectuer diverses analyses et d'obtenir des informations sur les interactions entre les utilisateurs et les dessins animés. En examinant les notes attribuées par les utilisateurs à différents titres d'animes, vous pouvez identifier les animes les mieux notés et les plus populaires parmi les utilisateurs. En outre, vous pouvez explorer les préférences des utilisateurs et les habitudes de visionnage pour des titres d'anime spécifiques. Cet ensemble de données constitue également la base de la création de systèmes de recommandation basés sur les évaluations des utilisateurs, ce qui permet de suggérer des animes qui correspondent aux goûts de chacun. En outre, vous pouvez effectuer un filtrage collaboratif et une analyse de similarité pour découvrir des modèles d'intérêts similaires chez les utilisateurs. Dans l'ensemble, cet en-

semble de données offre des informations précieuses pour comprendre l'engagement et les préférences des utilisateurs sur la plateforme d'anime.

2. Nettoyage des données et pré-analyse

Environnement Python

Pour mettre en place cette partie, nous avons décidé d'utiliser la librairie **pandas** sur Python afin d'extraire des données et faire des analyses comme la tendance sur les genres d'animés les plus visionnés par exemple. La librairie python **seaborn** permettra un affichage des données.

Pour ce faire nous avons procédé à toute la partie mise en place de python sur la VM dans un environnement virtuel :

```
cd ~
2 # Création du v-env
  python3 -m venv mon-venv
4 source mon-venv/bin/activate
  # Installation des modules
6 pip install --upgrade pip
  python3 -m pip install pandas seaborn
```

On exécute ensuite le code. (également disponible sur le git)

```
1 (mon-venv)$ python3 treat.py
```

Nettoyage des données

- Nettoyage et conversion des données :

Les colonnes numériques du DataFrame `df` (comme `Score`, `Popularity`, `Favorites`, etc.) qui contiennent des valeurs “UNKNOWN” sont remplacées par -1.

Ces colonnes sont ensuite converties en types de données appropriés (float64 pour les scores et int64 pour les autres).

Séparation de la colonne “Premiered” :

La colonne `Premiered` est divisée en deux nouvelles colonnes : `Premiere_year` et `Premiere_season`.

Une fonction `yr_season` est définie pour gérer cette séparation et assigner “UNKNOWN” lorsque nécessaire.

- Encodage des genres :

Les genres sont convertis en format one-hot encoding. Chaque genre devient une nouvelle colonne avec des valeurs 0 ou 1 indiquant la présence de ce genre dans chaque anime.

Une fonction `split_genre` est utilisée pour mettre à jour chaque ligne avec les valeurs one-hot encodées.

- Rafraîchissement et sauvegarde des données :

Le DataFrame est sauvegardé dans un nouveau fichier CSV `anime-dataset-2023-refined.csv`, puis rechargé avec `anime_id` comme index.

- Agrégation des données par année :

Les données sont regroupées par année de première diffusion (`Premiere_year`), calculant le score moyen (`mean_score`) et le nombre de shows (`show_count`) par année.

Une somme totale des genres par année est également calculée (`genre_all_sum`).

Premières visualisations

- Evolution de la proportion des genres par année

Ici on va vouloir utiliser la librairie `seaborn` pour pouvoir afficher une “heatmap”

Proportion des genres par années

- Nombre d'utilisateurs par pays

Ici la chose à faire n'était pas si compliquée, il fallait différencier 3 cas dans les données de localisation (en splitant le string via le caractère de la virgule) :

On va faire alors une nouvelle colonne “COUNTRY” composée de : - le string vide [””] -> on remplace par “UNKNOWN” - le string composé d'un pays [“France”], -> on vient prendre le premier élément - le string composé de la ville ET du pays [“Evry, France”] -> on vient prendre le deuxième (indice -1)

Après ceci on vient faire une requête en groupby sur les country, on compte, puis affichage sous forme de barre.

```
sns.barplot(x='User_count', y='Country', data=top_15_countries,
            palette="viridis", ax=ax)
```

Nombre d'utilisateurs par pays

3. Hadoop et Hive

Pour repartir de 0 (l'option `-v` supprime les volumes) :

```
# Arrêt des docker-compose
2 docker compose -f
  ~/projet-ipbd24/docker-files/hadoop/docker-compose.yml -v down
docker compose -f ~/superset/docker-compose.yml -v down
4 # Suppression de tous les conteneurs
docker rm -f $(docker ps -a -q)
6 # Suppression de tous les volumes restants
docker volume rm $(docker volume ls -q)
```

Démarrage

On lance tout d'abord le cluster Hadoop avec les scripts de démarrage. Ces scripts sont disponibles dans le dossier `~/projet-ipbd24/docker-files/hadoop/`.

Il s'agit de ceux vus en cours modifiés. Les scripts de démarrage ont été légèrement factorisés ; avec l'introduction d'une variable pour définir le fichier `docker-compose` qui a été utilisée. De plus, les conteneurs spark ont été enlevés car inutilisés et le nom du cluster est désormais `project` et non `test`.

On lance le cluster :

```
cd ~/projet-ipbd24/docker-files/hadoop/  
2 ./start-hadoop.sh
```

Dans un souci de lisibilité, j'ai ajouté des volumes nommés afin de pouvoir savoir quel volume est lié à quel conteneur. J'utilise la commande suivante pour déterminer les volumes montés de tous les conteneurs qui n'ont pas encore de nom lisible.

```
docker container inspect $(docker ps -q) | grep -A5 Mounts
```

Le seul volume à ajouter fut `hadoop_metastore_postgresql`.

La liste des volumes est à présent limpide, pour les conteneurs actifs.

```
docker volume ls
```

Import des données dans HDFS

On se connecte au conteneur `namenode` et on vérifie que HDFS fonctionne correctement :

```
docker exec -it namenode bash  
2 # Le volume contenant les données  
cd /data/hdfs/files/dataset  
4 # Vérification du bon fonctionnement d'HDFS  
hdfs dfs -df -h
```

On crée ensuite les répertoires dans HDFS et on ajoute nos données :

```
1 hdfs dfs -mkdir /dataset  
hdfs dfs -put /data/hdfs/files/dataset/*.csv /dataset  
3 # Pour vérifier :  
hdfs dfs -ls /dataset
```

On peut également vérifier via l'interface graphique : <http://162.19.124.170:9870/explorer.html#/>

Une fois le cluster Hadoop/Hive/Yarn lancé et les données mises sur HDFS, on peut passer sur Hive.

Création de la database du Hive

On se connecte au docker du serveur Hive après être revenu sur la VM (quitter le docker avec Ctrl+D).

```
docker exec -it hive-server bash
```

On initialise ensuite la database `myanimelist_db` ainsi que ses tables. Toutes les commandes sql nécessaires sont dans le fichier `files/scripts/init_db.sql` du dépôt. Il suffit d'exécuter ce fichier sur Hive, il se trouve dans le volume du docker `/data/hive/files/`.


```
/opt/hive/bin/beeline -u jdbc:hive2://hive-server:10000 -f
/data/hive/files/scripts/init_db.sql
```

Un extrait de ce fichier SQL :

```
1 CREATE DATABASE IF NOT EXISTS myanimelist_db;
3 USE myanimelist_db;
5
CREATE EXTERNAL TABLE IF NOT EXISTS score (
7     user_id INT,
    anime_id INT,
9     rating INT
)
11 ROW FORMAT DELIMITED
    FIELDS TERMINATED BY ','
13 LINES TERMINATED BY '\n'
    STORED AS TEXTFILE
15 LOCATION '/dataset/';
17 [...]
```

Si aucune erreur de survient, tout va bien. On peut tout de même se connecter notre database sur Hive et vérifier que les tables sont correctes :

```
1 /opt/hive/bin/beeline -u
    jdbc:hive2://hive-server:10000/myanimelist_db
SHOW TABLES;
3 SELECT * FROM anime LIMIT 3;
SELECT * FROM users LIMIT 10;
5 SELECT * FROM score LIMIT 10;
```

On voit plusieurs problème : - le header de chaque fichier csv a été importé. - les groupement créé par les apostrophes ne sont pas prises en compte, en plus de cela ces groupements peuvent comprendre des retours à la ligne. - au delà du problème des guillemets, il y a beaucoup trop de valeurs NULL.

Pour le problème de guillemet et de header on remplace :

```
ROW FORMAT DELIMITED
2 FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
4 STORED AS TEXTFILE
    LOCATION '/dataset/score'
```

Par :

```
1 ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
    WITH SERDEPROPERTIES (
```

```

3     "separatorChar" = ',',
     "quoteChar"      = '\\"',
5     "escapeChar" = "\\\"
)
7 STORED AS TEXTFILE
  LOCATION '/dataset/score'
9 TBLPROPERTIES ("skip.header.line.count"="1");

```

L'ancien fichier `files/scripts/init_db.sql` s'appelle désormais `wrong_init_db.sql` pour voir les différences.

Pour régler la trop grande présence de valeurs nulles, on crée un répertoire par dossier csv. Actuellement, toutes les tables ont la même `LOCATION` ce qui peut poser problème.

Pour pouvoir traiter les retours à la lignes dans le csv, on écrit un script python `files/scripts/replace_newlines.py` qui remplace les retours à la ligne dans les colonnes du csv par `\\n` afin que Hive puisse l'importer. On fournit les csv à traiter en ligne de commande, le script utilise les libraires `pandas`, `os` et `sys`.

```

source ~/mon-venv/bin/activate
2 cd ~/projet-ipbd24/files/
  ./scripts/replace_newlines.py dataset/*.csv
4 deactivate

```

Les fichiers traités on pour nomenclature `*_modified.csv`.

On recommence l'importation de 0 avec les csv modifiés :

```

docker exec -it namenode bash
2 # Suppression
  hdfs dfs -rm -r -f /dataset/*
4 # Création des dossiers
  hdfs dfs -mkdir /dataset/{anime,users,score}
6 # Importation des données sur HDFS
  hdfs dfs -put
    /data/hdfs/files/dataset/anime-dataset-2023_modified.csv
    /dataset/anime/anime-dataset-2023.csv
8 hdfs dfs -put /data/hdfs/files/dataset/users-score-2023_modified.csv
    /dataset/users/users-score-2023.csv
  hdfs dfs -put /data/hdfs/files/dataset/user-filtered_modified.csv
    /dataset/score/user-filtered.csv
10 # Pour vérifier :
  hdfs dfs -ls /dataset/*
12 # <C-D> quitter le conteneur

```

On supprime les tables, et on importe à nouveau dans Hive :

```

docker exec -it hive-server bash
2 /opt/hive/bin/beeline -u
  jdbc:hive2://hive-server:10000/myanimelist_db
DROP TABLE anime;

```

```
4 DROP TABLE users;
  DROP TABLE score;
6 # <C-C> quitter Hive, pas le conteneur
```

On importe à nouveau, en étant dans le conteneur `hive-server` :

```
/opt/hive/bin/beeline -u jdbc:hive2://hive-server:10000 -f
/data/hive/files/scripts/init_db.sql
```

Et on vérifie à nouveau :

```
1 /opt/hive/bin/beeline -u
  jdbc:hive2://hive-server:10000/myanimelist_db
  SHOW TABLES;
3 SELECT * FROM anime LIMIT 3;
  SELECT * FROM users LIMIT 5;
5 SELECT * FROM score LIMIT 5;
```

Finalement, après toutes ces manipulations en aller-retour pour vérifier quelle modification fonctionne, on a nos données proprement importées dans Hive ! Plus qu'à faire de belles visualisations dans Superset

Résolution du problème de connexion à Hive

Fréquemment la connexion à Hive ne fonctionne plus, la seconde commande retournant une erreur :

```
1 docker exec -it hive-server bash
  /opt/hive/bin/beeline -u jdbc:hive2://hive-server:10000
```

L'erreur retournée témoigne que la database par défaut `default` n'existe pas, ce qui rend la connexion impossible.

```
$ docker logs hive-server
2 FAILED: SemanticException [Error 10072]: Database does not exist:
  default
```

Pour résoudre ce problème, on arrête les deux containers du metastore et on les relance :

```
docker compose down hive-metastore
2 docker compose down hive-metastore-postgresql
  cd ~/projet-ipbd24/docker-files/hadoop/
4 ./start-hadoop.sh
```

4. Utilisation d'Apache Superset

Apache Superset est une plateforme de visualisation et d'exploration de données moderne. Elle inclut entre autres une interface no-code mais supporte également des requêtes SQL pour des requêtes avancées.

```
git clone --depth=1 https://github.com/apache/superset.git
2 cd superset
```

L'entièreté de cette partie, git clone compris mais hors création de Dashboard, peut être faite en lançant simplement le script suivant :

```
./files/superset/docker-compose-superset-git.sh
```

Pour vivre toutes péripéties et problèmes rencontrés, continuer avec la partie qui suit.

Premier essai : version dev

On a effectué un premier lancement avec la version de développement, qui est supposé se mettre à jour lors de changement de données locales. Cependant l'interface en ligne ne fonctionnait (<http://162.19.124.170:8088/>, identifiants : `admin`, `admin`).

UI Web de la version dev en erreur

La commande néanmoins :

```
docker compose up -f docker-compose.yml -d
```

Quelque chose était possiblement mal configuré, je n'ai pas spécialement cherché plus loin et j'ai installé la version `non-dev`. Il fallait d'abord désinstaller la version `dev`, ce qui a posé quelques problèmes car le network `superset_default` était encore up et ne voulait pas être arrêté. Je pense que cela était dû à un arrêt du lancement mal géré lors d'un Ctrl+D.

J'ai donc inspecté ce réseau, ce dernier avait encore un `endpoint` actif ce qui l'empêchait d'être arrêté. Après déconnexion de l'endpoint, c'était bon.

```
docker network inspect superset_default
2 docker network disconnect -f superset_default superset_init
docker compose down
```

Deuxième essai : version non-dev

La version dev ne fonctionnant pas, nous avons lancé la version non-dev qui nous conviendrait tout autant.

La version `non-dev` prenant une image immutable une fois `up`, c'est-à-dire que toute modification des fichiers de configuration locaux ne sont pas visibles sans un redémarrage.

```
docker compose -f docker-compose-non-dev.yml up -d
```

On se connecte ensuite sur l'interface en ligne <http://162.19.124.170:8088/> avec les identifiants par défaut `admin`, `admin`.

Tout fonctionne.

UI Web non-dev fonctionnel

Connexion à Hive

On souhaite désormais accéder aux données gérées par Hive.

La première étape est d'installer les drivers nécessaires pour Hive sur le conteneur de Superset. On doit l'ajouter dans la configuration et recréer l'image docker.

```
cd ~/superset
2 touch ./docker/requirements-local.txt
  echo "pyhive" >> ./docker/requirements-local.txt
4 docker compose -f docker-compose-non-dev.yml up -d
```

Deuxièmement, sur l'interface web (<http://162.19.124.170:8088/>, admin, admin) on va dans "Settings > Database Connections > + Database > SUPPORTED DATABASES" et on sélectionne Apache Hive. On ajoute ici l'URI SQLAlchemy de Hive :

```
hive://hive@162.19.124.170:10000/myanimelist_db
```

On teste la connexion à la db en cliquant sur le bouton **TEST CONNECTION**. La connexion est bonne.

On clique ensuite sur "Connect", la db est créée sur Superset bien qu'une erreur s'affiche.

Création de Dashboards

5. Pour aller plus loin

On a également essayé de créer des dashboards à partir de nos datasets, mais l'import des données depuis hive a été mal géré par superset (conversion en VARCHAR de tous les types de données). Il faudrait réessayer en important les données de la table externe dans une table gérée directement par Hive (format orc par exemple). Il faut également faire usage du prétraitement, et donc modifier les requêtes SQL afin de correspondre aux changements.

Superset est actuellement toujours configuré en mode développement. Mettre en place un environnement de production digne de ce nom selon les indications de la documentation nécessite un certain nombre de changement.

La documentation utile pour cela : - <https://superset.apache.org/docs/configuration/configuring-superset/#setting-up-a-production-metadata-database> - <https://superset.apache.org/docs/installation/docker-compose> - <https://minikube.sigs.k8s.io/docs/start/>

Le logiciel minikube est un kubernetes pour un seul host (VM).

6. Google Cloud

En parallèle de la démarche faite sur apache superset, nous avons tentés de faire une analyse des données via les services google : Big Query et Google Looker. Ces deux outils permettent de visualiser des données ainsi que d'effectuer des requêtes sur celles-ci. Il y a un fort esprit d'automatisation, cet outil est donc adapté au grand public, mais pas très pratique quand les données ne sont pas formatées exactement comme il le faut. Comme ce n'est pas un outil opensource mais bien un service payant, vous ne pourrez pas retracer les démarches (même si elles sont simplissimes).

Le plus compliqué est surtout la compréhension des outils et des API liées, il y a énormément de ressources sur google cloud. Après avoir déterminé que j'allais utiliser BigQuery et Looker, il faut paramétrer une BD.

On commence ici, sur la [console du cloud](#)

Chargement des données

Voici un [document avec tous les screens de la démarche](#)

1. On crée un nouveau projet, qu'on appelle ipbd00
2. On sélectionne BigQuery dans le volet latéral gauche une fois qu'on a bien sélectionné le bon projet
3. Une fois sur la console de bigQuery, on sélectionne "+Ajouter"
4. On importe les données qu'on a localement
5. On définit la table (fichier correspondant et projet). Il faut aussi définir un ensemble de données pour le projet.
6. On appelle notre ensemble de données "data_originelle", et on sélectionne une région proche et peu consommatrice (vive l'environnement). On ne sélectionne pas d'autres options.
7. on définit le nom de la table et on active la détection automatique de la table (promis ça fonctionne plutôt bien dans notre cas)
8. On coche les options "Nouvelles lignes entre guillemets" pour gérer les sauts de lignes ; et "Lignes irrégulières" puisque c'est le cas. On ne coche rien d'autre
9. On réitère les étapes 3 à 8 pour définir la table users
10. On répète l'étape 3 pour la table score, mais au lieu de sélectionner un fichier local, on va importer les données d'abord sur google cloud storage
11. On parcourt nos buckets. On en a pas : il faut en créer un.
12. On va sur la console du service Google Cloud Storage
13. On crée un bucket
14. On l'appelle scoreaaa parce qu'il faut qu'il soit unique.
15. On sélectionne région, et on choisit Paris
16. On choisit la classe standard, parce que cela correspond à nos besoins
17. On sélectionne un accès aux données uniforme
18. On a des options de protection des données, ici peu importe (je les décoche)
19. On a un bucket ! Il faut maintenant mettre des données dedans
20. On importe les données
21. On crée une table de la même façon que pour les autres, mais avec les fichiers d'origine du cloud (ceux qu'on vient d'ajouter)
22. On retourne sur la console BigQuery et on clique sur "Canevas de données"
23. On active le canevas
24. On autorise les API
25. On a un canevas de données où on peut ajouter des requêtes SQL, qui sont ensuite visualisables
26. Tada !

On peut voir également qu'il y a plein de services ajoutables, ce qui a l'air vraiment prometteur