

PROJET

BIG DATA

MovieLens - Système de recommandation de films avec
Hadoop et Hive

JEATHUSAN ET RANIA
M1 IBD à université paris 8

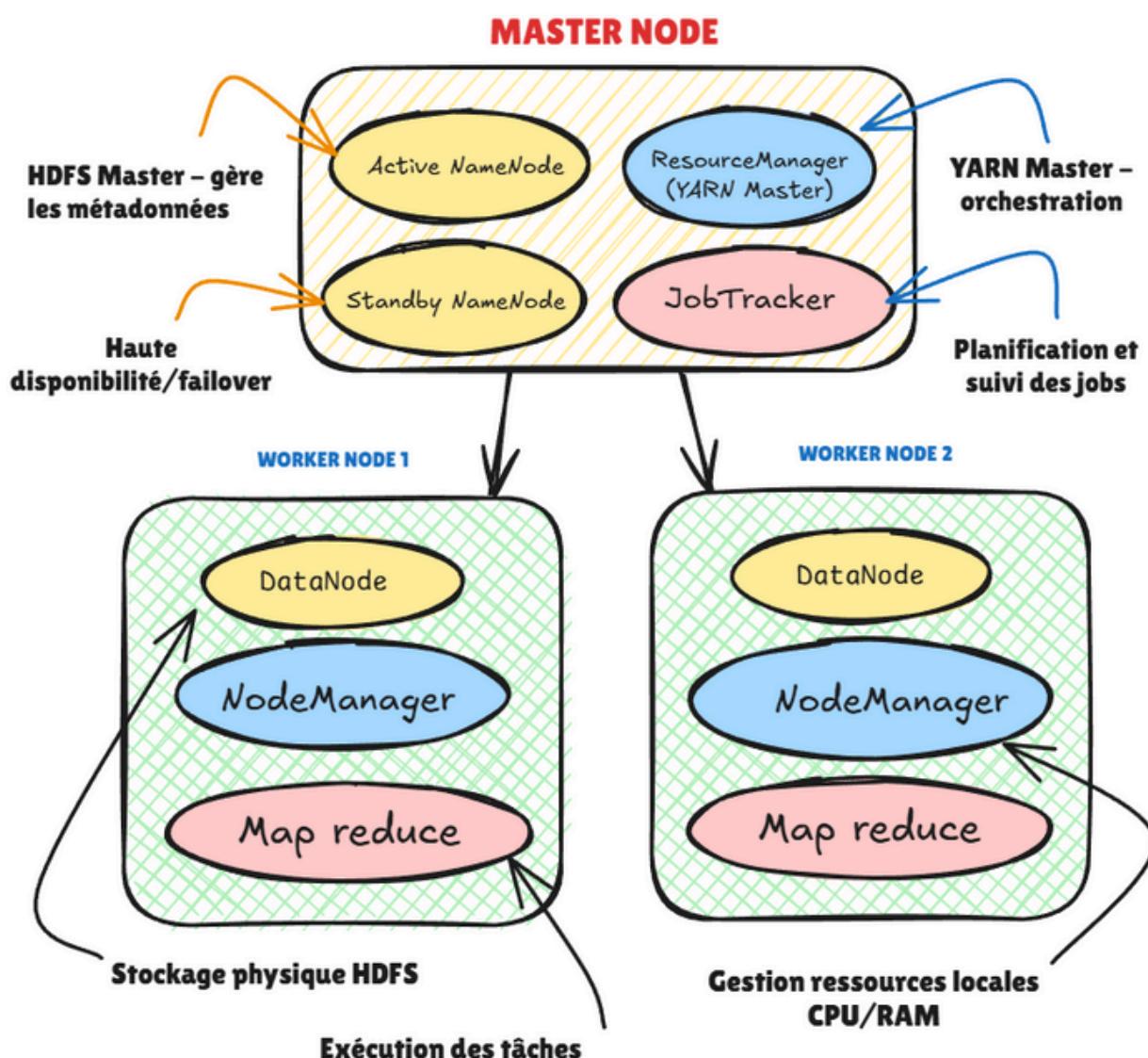
MODULE

Cadre logiciel pour big data

Ce projet consiste à créer un système de recommandation de films en utilisant Hadoop et Hive. L'objectif est de traiter une grande quantité de données de films (le dataset MovieLens) pour identifier les tendances et suggérer des films aux utilisateurs. Nous stockons d'abord les données dans HDFS, puis nous les analysons avec MapReduce pour trouver les films les mieux notés et les préférences des utilisateurs. Ensuite, nous utilisons Hive pour affiner ces analyses et construire un système de recommandation simple basé sur les goûts des utilisateurs. Le tout est déployé sur un cluster Hadoop distribué dans le cloud via création des vm pour profiter d'une architecture réellement distribuée.

Les résultats montrent que notre approche fonctionne bien et permet d'obtenir des recommandations pertinentes à partir d'une grande base de données de films. Cela prouve qu'une architecture Big Data est adaptée pour ce type de projet.

- *Le Master CONTRÔLE mais ne STOCKE PAS les données*
- *Les Workers font le VRAI travail : stockage + traitement distribué*
- *Résilience : si un worker tombe, le cluster continue*



Objectif de ce projet

L'objectif de ce projet est de développer et d'analyser une solution de recommandation dans le domaine du streaming, en mobilisant les compétences acquises durant notre formation et appliquées ici concrètement. Nous avons utilisé l'écosystème Hadoop (HDFS, MapReduce, YARN) ainsi que Hive pour analyser le jeu de données public MovieLens.

Ce projet vise à démontrer nos compétences auprès des entreprises pour des stages ou alternances, afin de prouver notre capacité à occuper des postes tels que Data Engineer, Data Scientist ou Data Analyst.

Pour ce faire, nous avons travaillé dans un cadre simulant l'environnement d'une entreprise de streaming comme Netflix, Amazon Prime ou Disney+, à travers une étude de cas typique d'un Data Engineer. L'enjeu était de développer un système d'analyse basé sur les évaluations de films données par les utilisateurs.

Au-delà de l'enrichissement de notre expérience, ce projet permet aussi de montrer aux entreprises l'importance des métiers de la data et leur utilité pour mieux cibler les utilisateurs et améliorer la stratégie marketing. Il illustre ainsi la valeur ajoutée de ces compétences pour la croissance de l'entreprise. Tel était notre principal objectif en lançant cette initiative.

Les liens utiles à consulter :

- **Lien Github du projet :** <https://github.com/Jeathu/MovieLens-Big-Data-project>
- **Vidéo du tout les procédure du projet avec les VMs Azure 1 à 7 :**
<https://e.pcloud.link/publink/show?code=kZAQnNZvzWdFnVGhhRb4O0EnUyWtjw6SlfV>
- **Démo :** <https://e.pcloud.link/publink/show?code=XZByPNZx4f2dGjcKym5Uh77DElljCiU7Wk>
- **Avec GCP CLI les création des cluster :**
<https://e.pcloud.link/publink/show?code=kZ9yPNZ5VcjN9xHheuYBBWauN0Qhhtw0Dly>

1. Création de l'architecture distribuée Hadoop

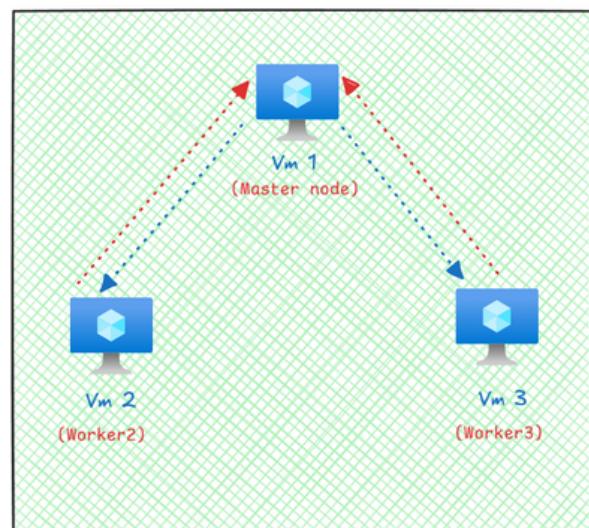
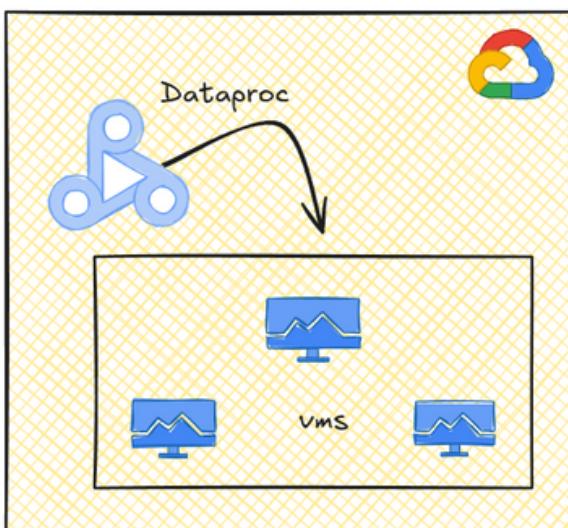
1.1 Analyse des différentes approches de choix de démarrage

Pour ce projet, nous avons testé plusieurs méthodes afin d'identifier celles qui étaient à la fois fiables et adaptées à un cadre pédagogique. Trois possibilités s'offraient à nous :

- **Machine locale** : nous ne l'avons pas retenue, car il s'agit d'une machine unique, ce qui ne permet pas une architecture distribuée nécessaire au big data.
- **VMs type Cloudera** : cette option s'est avérée trop lourde et nécessitait au minimum deux machines virtuelles configurées sur le même réseau, ce qui ajoutait de la complexité. Nous ne l'avons donc pas choisie.
- **Cloud** : solution idéale, car elle permet de créer facilement plusieurs machines virtuelles, adaptées à un projet big data.

Nous avons donc travaillé sur différents clouds, en utilisant également des méthodes variées. Au départ, nous avons choisi de créer des clusters via les interfaces en ligne de commande (CLI) des fournisseurs cloud. Cette approche était rapide (moins de 15 minutes pour déployer un cluster). Dans le contexte cloud, on peut faire avec le CLI moins de 10 minutes ou monter des VMs et faire les configurations. Nous, on a choisi les deux pour tester d'abord leur fiabilité. Pour le test CLI, on a choisi de le faire avec Google Cloud Platform (GCP) en créant un cluster Dataproc. L'objectif était de faire un premier essai de démarrage d'un cluster. Cette approche visait à : comparer une solution cloud managée à une architecture distribuée de VMs, simplifier le déploiement et l'administration de l'infrastructure Big Data, et permettre un démarrage plus rapide.

Mais d'un point de vue pédagogique, elle n'était pas idéale : tout étant géré automatiquement par le cloud, nous ne comprenions pas le fonctionnement sous-jacent. C'est pourquoi nous avons finalement opté pour une configuration manuelle : **créer des machines virtuelles et mettre en place une architecture distribuée Hadoop nous-mêmes**. Cela nous a permis d'approfondir nos connaissances et de maîtriser les métriques des outils utilisés.



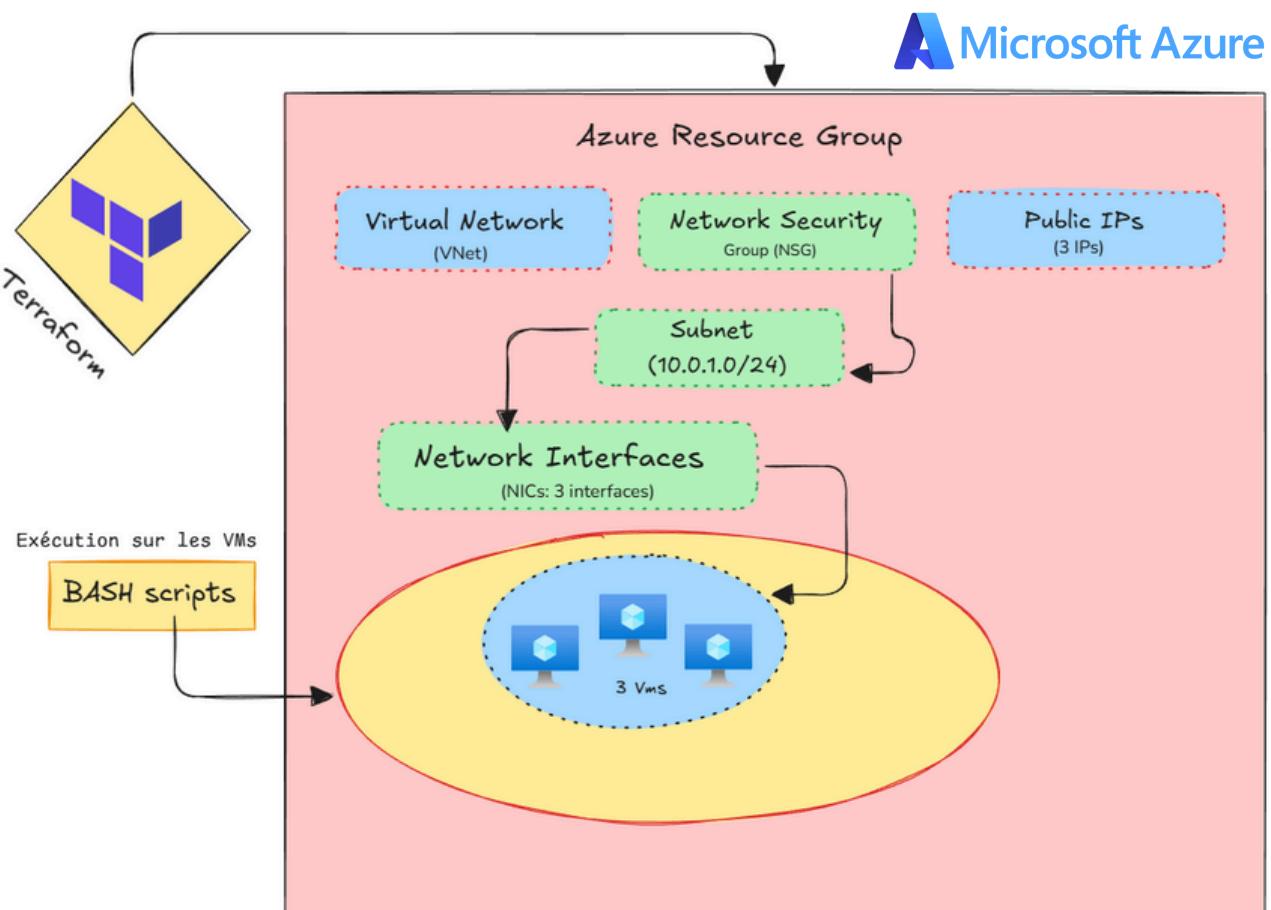
1.2. Utilisation des outils DevOps et leur architecture (perspective data engineer)

Lorsque nous avons commencé à utiliser le cloud pour ce projet, un problème récurrent est apparu : la gestion des coûts. En effet, si les machines virtuelles (VMs) restent allumées en permanence, les coûts peuvent rapidement augmenter de manière significative. Nous étions donc contraints de les arrêter régulièrement, ce qui nous obligeait à reconfigurer l'environnement à chaque redémarrage, une tâche longue et répétitive.

Face à cette situation, j'ai adopté une approche inspirée des pratiques en entreprise, en utilisant des outils DevOps pour automatiser ces processus. J'ai ainsi utilisé Terraform pour créer les VMs de manière automatisée. Terraform est un outil d'infrastructure en tant que code (IaC) qui permet de créer, allouer et gérer des ressources informatiques dans le cloud de façon déclarative et reproductible.

Parallèlement, j'ai développé des scripts Bash automatisés pour installer et configurer Hadoop, Hive et leurs dépendances. Cette double automatisation nous a permis de reconstruire rapidement et à volonté un environnement de travail entièrement fonctionnel, tout en maîtrisant strictement les coûts liés à l'infrastructure.

Architecture Terraform - Azure Hadoop/Hive Cluster



1.3. Vizualisation des machine virtuelles sur (AZURE)

Machines virtuelles Commencer

+ Créer Réservations Gérer l'affichage Actualiser Exporter au format CSV Ouvrez une requête Attribuer des balises ... Aucun regroupement

(i) Vous affichez une nouvelle version de l'expérience de navigation. Cliquez ici pour accéder à l'ancienne expérience.

Filtrer un champ... Abonnement est égal à tout Type est égal à tout Groupe de ressources est égal à tout Emplacement est égal à tout + Ajouter un filtre

Nom ↑	Abonnement	Groupe de ress...	Emplacement	État	Système d'explor...	Taille	Adresse IP pub...	Disques
hadoop-master	Azure for Stud...	MovieLens-proj...	France Central	Exécution	Linux	Standard_B2as_v2	4.233.121.182	1
hadoop-worker1	Azure for Stud...	MovieLens-proj...	France Central	Exécution	Linux	Standard_B2s_v2	4.211.254.177	1
hadoop-worker2	Azure for Stud...	MovieLens-proj...	France Central	Exécution	Linux	Standard_B2s_v2	4.233.72.163	1



MovieLens-projet Groupe de ressources

Rechercher + Créer Gérer l'affichage ... Aucun regroupement

Vue d'ensemble Bases Vue JSON

Ressources Recommandations

Filtrer un champ... Type est égal à tout Emplacement est égal à tout + Ajouter un filtre

Nom ↑	Type	Emplacement
hadoop-master	Machine virtuelle	France Central
hadoop-master_disk1_c0bb85a4ad194a	Disque	France Central
hadoop-nsg	Groupe de sécurité	France Central
hadoop-vnet	Réseau virtuel	France Central
hadoop-worker1	Machine virtuelle	France Central
hadoop-worker1_disk1_ba440bd87f8a41	Disque	France Central
hadoop-worker2	Machine virtuelle	France Central
hadoop-worker2_disk1_034a4e7ab39f4c	Disque	France Central
master-nic	Interface réseau	France Central

Affichage de 1 – 10 sur 14. Afficher le nombre : auto < 1 2 >

Ajouter ou supprimer des favoris en appuyant sur Ctrl+Shift+F Envoyer des commentaires

1.4. Vizualisation des information des Datanode et Cluster Hadoop

The screenshot shows the Hadoop Cluster Metrics interface. On the left, there's a sidebar with 'Cluster' and 'Tools' sections. The main area displays 'Cluster Metrics' with tabs for 'Cluster Metrics', 'Cluster Nodes Metrics', and 'Scheduler Metrics'. Under 'Cluster Metrics', there are sections for 'Apps Submitted', 'Apps Pending', 'Apps Running', 'Apps Completed', 'Containers Running', and 'Used Resources'. Below these are 'Cluster Nodes Metrics' and 'Scheduler Metrics' sections. A table lists applications with columns for ID, User, Name, Application Type, Application Tags, Queue, Application Priority, Start Time, Launch Time, Finish Time, State, and Size. One entry is shown: application_1766444888571_0001, hadoop, QuasiMonteCarlo, MAPREDUCE, default, 0, Tue Dec 23 00:08:54 +0100 2025, Tue Dec 23 00:08:56 +0100 2025, Tue Dec 23 00:09:13 +0100 2025, FINISHED, Size. At the bottom, it says 'Showing 1 to 1 of 1 entries'.

Datanode Information

The screenshot shows the Datanode Information page. At the top, there are status icons: In service (green checkmark), Down (red circle), Decommissioning (green circle), Decommissioned (yellow circle), Decommissioned & dead (red circle), Entering Maintenance (green circle), In Maintenance (yellow circle), and In Maintenance & dead (red circle). Below this is a 'Datanode usage histogram' chart showing disk usage of each DataNode (%) from 0 to 100. A single green bar at approximately 2% is circled in red. The 'In operation' section lists data nodes with columns for Node, Http Address, Last contact, Last Block Report, Used, Non DFS Used, Capacity, Blocks, Block pool used, and Version. Two nodes are listed: /default-rack/hadoop-worker1:9866 (10.0.1.5:9866) and /default-rack/hadoop-worker2:9866 (10.0.1.6:9866). Both have 2s last contact, 9m last block report, 319.33 KB used, 4.29 GB non DFS used, 123.87 GB capacity, 3 blocks, and 319.33 KB block pool used. They are both version 3.3.6. The table shows 'Showing 1 to 2 of 2 entries'.

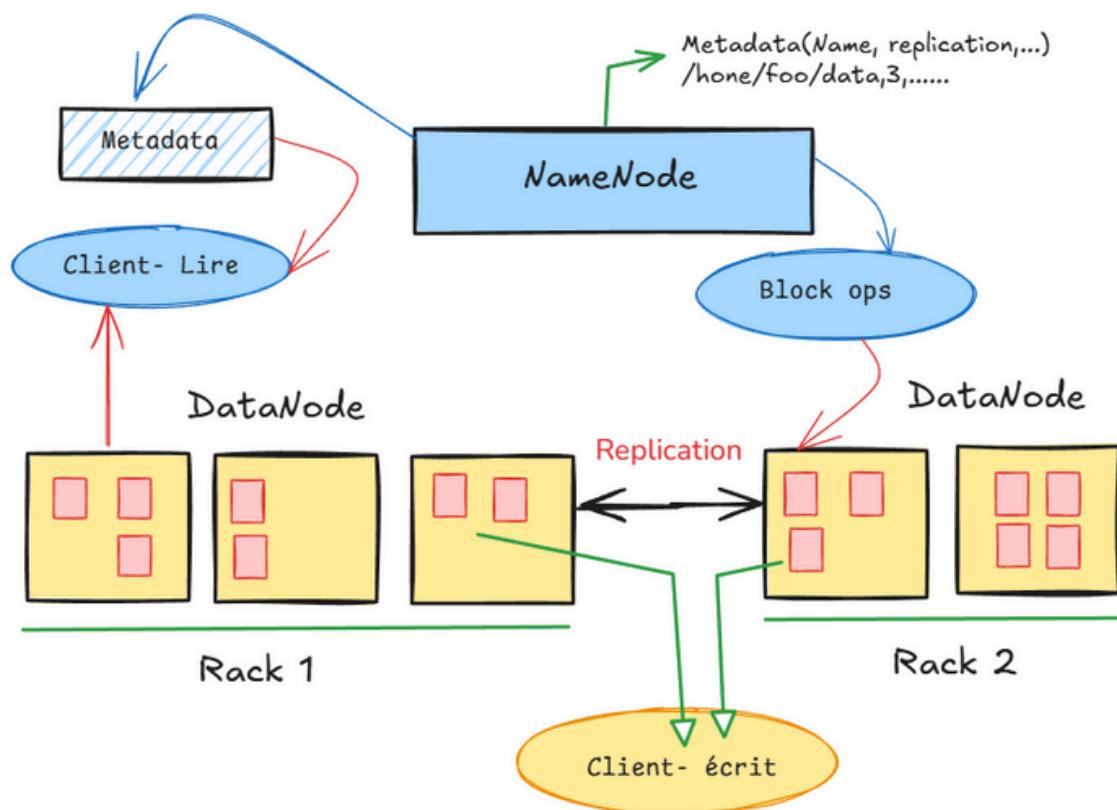
Comme on peut le voir sur l'image, le datanode contient des informations réparties sur **2 workers**. L'intérêt de cette architecture est de stocker les données de manière clusterisée. Pour illustrer, imaginons un entrepôt de données : si toutes les données sont stockées dans un seul entrepôt et qu'un incendie s'y déclare, toutes les données seraient perdues sans possibilité de récupération.

En revanche, si l'on réplique les données en plusieurs exemplaires dans différents entrepôts, même si l'un d'eux est détruit, une copie reste disponible ailleurs, évitant ainsi la perte des données. Dans Hadoop, c'est le même principe : le datanode stocke les données sur les noeuds workers avec un mécanisme de réplication qui dépend du nombre de nœuds disponibles, le tout étant supervisé par le noeud maître (master) qui contrôle ces workers.

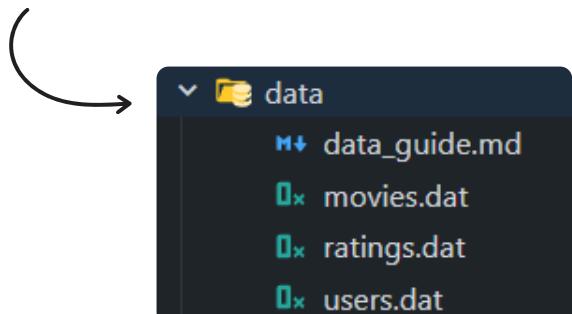
2. Implémentation des écosystèmes Hadoop et Hive

2.1. Manipulation avec HDFS (Hadoop Distributed File System)

HDFS divise les fichiers en blocs et stocke chaque bloc sur un DataNode. Les différents DataNodes sont reliés dans le cluster. Le NameNode distribue ensuite les répliques de ces blocs de données sur le cluster. Il indique à l'utilisateur ou à l'application où localiser les informations voulues.



Pour mon projet, j'ai utilisé le dataset **MovieLens 1M movie ratings**, qui est composé de **1 million d'évaluations** provenant de **6 000 utilisateurs** sur **4 000 films**. Dans un premier temps, notre objectif est de transférer nos données locales vers HDFS à l'aide des commandes **hdfs**. Vous trouverez les étapes détaillées dans la partie suivante.



On a commencé par créer **trois répertoires** dans HDFS pour y répartir les **trois jeux de données principaux** (movies, ratings et users). Pour cela, j'ai utilisé la commande HDFS suivante :

```
hdfs dfs -mkdir -p /user/movielens/ratings  
hdfs dfs -mkdir -p /user/movielens/movies  
hdfs dfs -mkdir -p /user/movielens/users
```

```
hadoop@hadoop-master:~$ hdfs dfs -ls /  
Found 2 items  
drwxrwxrwx  - hadoop supergroup      0 2025-12-28 18:18 /tmp  
drwxr-xr-x  - hadoop supergroup      0 2025-12-28 18:16 /user  
hadoop@hadoop-master:~$ hdfs dfs -mkdir -p /user/movielens/ratings  
hadoop@hadoop-master:~$ hdfs dfs -mkdir -p /user/movielens/movies  
hadoop@hadoop-master:~$ hdfs dfs -mkdir -p /user/movielens/users  
hadoop@hadoop-master:~$ hdfs dfs -ls /  
Found 2 items  
drwxrwxrwx  - hadoop supergroup      0 2025-12-28 18:18 /tmp  
drwxr-xr-x  - hadoop supergroup      0 2025-12-28 18:24 /user  
hadoop@hadoop-master:~$ hdfs dfs -ls -R /user/  
drwxr-xr-x  - hadoop supergroup      0 2025-12-28 18:16 /user/hive  
drwxrwxrwx  - hadoop supergroup      0 2025-12-28 18:16 /user/hive/warehouse  
drwxr-xr-x  - hadoop supergroup      0 2025-12-28 18:24 /user/movielens  
drwxr-xr-x  - hadoop supergroup      0 2025-12-28 18:24 /user/movielens/movies  
drwxr-xr-x  - hadoop supergroup      0 2025-12-28 18:24 /user/movielens/ratings  
drwxr-xr-x  - hadoop supergroup      0 2025-12-28 18:24 /user/movielens/users  
hadoop@hadoop-master:~$
```

Comme on peut le voir, nos données sont stockées dans le HDFS (Hadoop Distributed File System).

```
hadoop@hadoop-master:~$ sudo chown hadoop:hadoop ~/*.dat  
hadoop@hadoop-master:~$ ls -lh ~/*.dat  
-rwxr-xr-x 1 hadoop hadoop 168K Dec 28 18:42 /home/hadoop/movies.dat  
-rwxr-xr-x 1 hadoop hadoop 24M Dec 28 18:42 /home/hadoop/ratings.dat  
-rwxr-xr-x 1 hadoop hadoop 132K Dec 28 18:42 /home/hadoop/users.dat  
hadoop@hadoop-master:~$
```

```
hdfs dfs -put ~/ratings.dat /user/movielens/ratings/  
hdfs dfs -put ~/movies.dat /user/movielens/movies/  
hdfs dfs -put ~/users.dat /user/movielens/users/
```

```
hadoop@hadoop-master:~$ hdfs dfs -put ~/ratings.dat /user/movielens/ratings/  
hadoop@hadoop-master:~$ hdfs dfs -put ~/movies.dat /user/movielens/movies/  
hadoop@hadoop-master:~$ hdfs dfs -put ~/users.dat /user/movielens/users/  
hadoop@hadoop-master:~$ hdfs dfs -ls -R /user/movielens/  
drwxr-xr-x  - hadoop supergroup      0 2025-12-28 18:43 /user/movielens/movies  
-rw-r--r--  2 hadoop supergroup  171308 2025-12-28 18:43 /user/movielens/movies/movies.dat  
drwxr-xr-x  - hadoop supergroup      0 2025-12-28 18:43 /user/movielens/ratings  
-rw-r--r--  2 hadoop supergroup  24594131 2025-12-28 18:43 /user/movielens/ratings/ratings.dat  
drwxr-xr-x  - hadoop supergroup      0 2025-12-28 18:43 /user/movielens/users  
-rw-r--r--  2 hadoop supergroup  134368 2025-12-28 18:43 /user/movielens/users/users.dat  
hadoop@hadoop-master:~$
```

Pour une **bonne organisation du projet**, nous avons placé les données dans chaque répertoire correspondant. **Une organisation claire est essentielle** car, en tant que data engineer ou data analyst en entreprise, on travaille souvent avec des commandes la plupart du temps. Ainsi, une **bonne organisation peut nous aider à accélérer les tâches**. C'est pourquoi nous avons pris cette initiative.

2.2. Application et manipulation avec Hive

Dans un premier temps, j'ai commencé par créer une base de données **MovieLens**, puis j'ai également créé trois tables : **movies**, **ratings** et **users**. Comme on peut le voir sur l'image...

```
hive> SHOW TABLES;
OK
movies
ratings
users
Time taken: 0.016 seconds, Fetched: 3 row(s)
hive> 
```

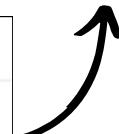
```
hive> SELECT * FROM ratings LIMIT 5;
OK
1      1193    5      978300760
1      661     3      978302109
1      914     3      978301968
1      3408    4      978300275
1      2355    5      978824291
Time taken: 0.097 seconds, Fetched: 5 row(s)
```

userid	movieid	rating	timestamp
1	1193	5	978300760
1	661	3	978302109
1	914	3	978301968
1	3408	4	978300275
1	2355	5	978824291



```
hive> SELECT * FROM movies LIMIT 5;
OK
1      Toy Story (1995)      Animation|Children's|Comedy
2      Jumanji (1995)       Adventure|Children's|Fantasy
3      Grumpier Old Men (1995) Comedy|Romance
4      Waiting to Exhale (1995)   Comedy|Drama
5      Father of the Bride Part II (1995) Comedy
Time taken: 0.133 seconds, Fetched: 5 row(s)
hive> 
```

ID Film	Titre du film	Genres
1	Toy Story (1995)	Animation, Children's, Comedy
2	Jumanji (1995)	Adventure, Children's, Fantasy
3	Grumpier Old Men (1995)	Comedy, Romance
4	Waiting to Exhale (1995)	Comedy, Drama
5	Father of the Bride Part II (1995)	Comedy



2.3. La magie du partitionnement dans Hive

Le partitionnement dans Hive permet d'organiser de grands ensembles de données en parties plus petites et plus faciles à gérer, en fonction des valeurs d'une ou plusieurs colonnes .

Dans notre cas, nous partitionnons les données MovieLens selon trois axes d'accès fréquents - **la note (rating)**, **le genre (genre)** et **l'année d'évaluation (year_rated, extraite de l'horodatage)** - afin de réduire les E/S, d'accélérer les requêtes (par élagage des partitions), de faciliter les analyses démographiques et temporelles, et de permettre un archivage ainsi qu'une gestion de la rétention par année.

```
CREATE TABLE IF NOT EXISTS ratings_partitioned (
    user_id INT,
    movie_id INT,
    timestamp_val BIGINT
)
PARTITIONED BY (rating INT)
STORED AS TEXTFILE;
```

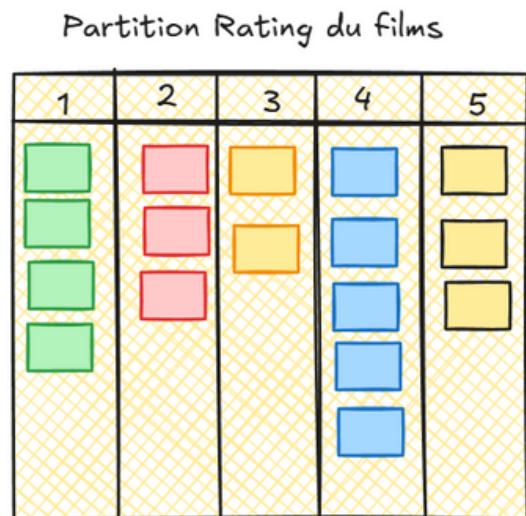
```
hive> SHOW PARTITIONS ratings_partitioned;
OK

rating=1
rating=2
rating=3
rating=4
rating=5
Time taken: 0.065 seconds, Fetched: 5 row(s)
```

La **partition par rating** permet de diviser l'ensemble des ratings de 1 à 5, c'est-à-dire que tous les films ayant un rating 1 sont regroupés, et ainsi de suite jusqu'au rating 5.

```
hive> SELECT COUNT(*) as excellent_ratings
    > FROM ratings_partitioned
    > WHERE rating = 5;
OK
226310
Time taken: 0.248 seconds, Fetched: 1 row(s)
hive>
```

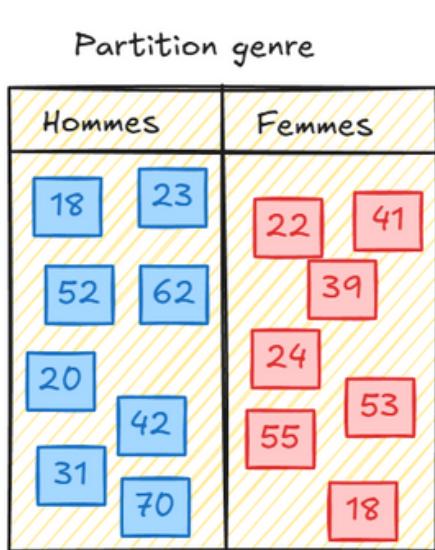
Comme on peut voir on a **226310 rating 5 stars**



Au lieu de chercher parmi toutes les données de manière désordonnée, le partitionnement permet de trouver directement ce dont on a besoin dans la partie qui nous intéresse !

(Exemple : si je veux les films notés 5, je cherche uniquement dans la partition "rating 5" – pas besoin de parcourir les autres.)

Pour le partitionnement par genre, les données sont d'abord séparées entre hommes et femmes, puis regroupées par tranche d'âge. Comme on peut le voir dans cet exemple, j'ai choisi les utilisatrices et leurs différents âges.



```

SELECT*
CASE age
WHEN 1 THEN 'Under 18'
WHEN 18 THEN '18-24'
WHEN 25 THEN '25-34'
WHEN 35 THEN '35-44'
WHEN 45 THEN '45-49'
WHEN 50 THEN '50-55'
ELSE '56+'
END as age_group,
COUNT(*) as count
FROM users_partitioned
WHERE gender = 'F'
GROUP BY age
ORDER BY age;
    
```

```

MapReduce Jobs Launched:
Stage-Stage-1: Map: 1   Reduce: 1   Cumulative CPU: 3.15 sec   HDFS Read: 41214 HDFS Write: 280 SUCCESS
Stage-Stage-2: Map: 1   Reduce: 1   Cumulative CPU: 2.57 sec   HDFS Read: 8093 HDFS Write: 241 SUCCESS
Total MapReduce CPU Time Spent: 5 seconds 720 msec
OK
Under 18          78
18-24            298
25-34            558
35-44            338
45-49            189
50-55            146
56+              102
Time taken: 40.999 seconds, Fetched: 7 row(s)
hive> 
```

On voit la répartition des **1 809 utilisatrices** par tranche d'âge, avec la tranche **25-34 ans** qui est la plus nombreuse (588 femmes), suivie des **35-44 ans** (338) puis des **18-24 ans** (288).

Dans un contexte d'entreprise, cette analyse permet de **cibler précisément les utilisateurs**. Par exemple, si l'on constate une forte proportion d'abonnés jeunes (18-24 ans) et majoritairement des femmes, on pourra adapter les **recommandations de films** en fonction des préférences de cette catégorie d'âge et de genre. Ainsi, indirectement, cette approche de recommandation ciblée permet non seulement d'attirer davantage d'utilisatrices, mais aussi d'augmenter le chiffre d'affaires.

Cette valorisation des données ne se limite pas aux recommandations internes : elle permet aussi de cibler des entreprises souhaitant faire de la publicité. En identifiant précisément les segments d'audience (comme les femmes de certaines tranches d'âge), on peut proposer aux annonceurs (par exemple, des marques de cosmétique, de mode, ou de bien-être) des espaces publicitaires adaptés aux préférences des utilisatrices.

2.4. La magie du bucketing dans Hive

Le bucketing consiste à diviser les données en un nombre fixe de fichiers (appelés « buckets ») de taille équilibrée. Cette répartition est basée sur la valeur de hachage d'une colonne spécifique. Il est généralement utilisé pour uniformiser la distribution des données dans un nombre déterminé de fichiers, ce qui peut améliorer les performances dans certaines opérations, comme les jointures sur de grands volumes de données.

On a appliqué le bucketing sur les tables ratings et users en utilisant user_id comme colonne de clustering, créant ainsi une structure optimisée pour les requêtes de jointure entre utilisateurs et leurs évaluations.

Pour le Bucket **ratings_bucketed**, on organise toutes les évaluations de films en 10 groupes selon l'identifiant utilisateur. Chaque évaluation d'un utilisateur donné est toujours stockée dans le même bucket. Cela permet de :

- Regrouper toutes les évaluations d'un même utilisateur
- Faciliter l'analyse du comportement par utilisateur
- Optimiser les jointures avec la table des utilisateurs

```
CREATE TABLE IF NOT EXISTS ratings_bucketed (
    user_id INT,
    movie_id INT,
    rating INT,
    timestamp_val BIGINT
)
CLUSTERED BY (user_id) INTO 10 BUCKETS
STORED AS TEXTFILE;
```

Exemple de fonctionnement :

Données d'entrée:

Table ratings_bucketed		
user_id	movie_id	rating
101	501	5
205	502	4
312	503	3
101	504	5 ← même user
425	505	2
...		

Fonction de Hash
hash(user_id) % 10
Résultat: 0 à 9

10 Buckets créés:



Calcul du bucket:

```
hash(101) = X + Y % 10 = 1 → Bucket 1
hash(205) = Y + Z % 10 = 5 → Bucket 5
hash(312) = Z + W % 10 = 2 → Bucket 2
hash(101) = X + X % 10 = 1 → Bucket 1 (même!)
hash(425) = W + W % 10 = 5 → Bucket 5
```

Visualisation physique sur HDFS

Fichiers physiques créés par Hive, chacun correspondant à un bucket. Seuls les buckets 1, 2 et 5 contiennent des données, illustrant la répartition des enregistrements selon le hachage du user_id.

```
ratings_bucketed/
├── 000000_0 # Bucket 0 - vide
└── 000001_0 # Bucket 1 - 2 lignes (user_id=101)
├── 000002_0 # Bucket 2 - 1 ligne (user_id=312)
├── 000003_0 # Bucket 3 - vide
├── 000004_0 # Bucket 4 - vide
└── 000005_0 # Bucket 5 - 2 lignes (user_id=205,425)
├── 000006_0 # Bucket 6 - vide
├── 000007_0 # Bucket 7 - vide
├── 000008_0 # Bucket 8 - vide
└── 000009_0 # Bucket 9 - vide
```

```
SELECT COUNT(*) FROM ratings_bucketed TABLESAMPLE(BUCKET 1 OUT OF 10 ON user_id);
```

```
MapReduce Total cumulative CPU time: 5 seconds 70 msec
Ended Job = job_1766945762376_0013
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 5.07
Total MapReduce CPU Time Spent: 5 seconds 70 msec
OK
114383
Time taken: 18.986 seconds, Fetched: 1 row(s)
hive> 
```

Cette commande compte le nombre de lignes du bucket #1 pour vérifier que les données sont correctement distribuées dans la structure bucketée.

Avec le bucket **users_bucketed**, on crée une table organisée en 10 buckets selon l'identifiant utilisateur pour optimiser les performances des jointures futures.

```
CREATE TABLE IF NOT EXISTS users_bucketed (
    user_id INT,
    gender STRING,
    age INT,
    occupation INT,
    zipcode STRING
)
CLUSTERED BY (user_id) INTO 10 BUCKETS
STORED AS TEXTFILE;
```

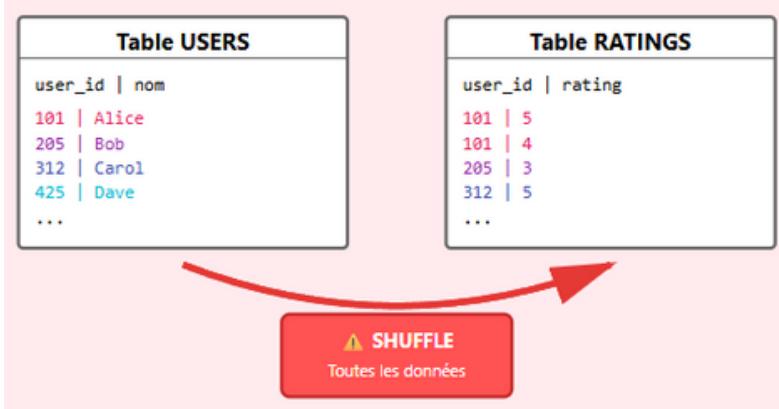
```
Total MapReduce CPU Time Spent: 8 seconds 580 msec
OK
F      1      3.616290925569276      8827
F      18     3.4531446056310124     45427
F      25     3.6067002408583315     91340
F      35     3.6596527398783176     49473
F      45     3.663044379925342     24110
F      50     3.7971102745792735     18064
F      56     3.915534297206218     9199
M      1      3.5174608355091386     18384
M      18     3.525476254262937     138109
M      25     3.52678031398743     304216
M      35     3.604433892864308     149530
M      45     3.627942140013104     59523
M      50     3.687098078124426     54426
M      56     3.720327237077854     29581
Time taken: 51.242 seconds, Fetched: 14 row(s)
hive> 
```

```
SELECT
    u.gender,
    u.age,
    AVG(r.rating) as avg_rating,
    COUNT(*) as num_ratings
FROM ratings_bucketed r
JOIN users_bucketed u ON r.user_id = u.user_id
GROUP BY u.gender, u.age
ORDER BY u.gender, u.age;
```

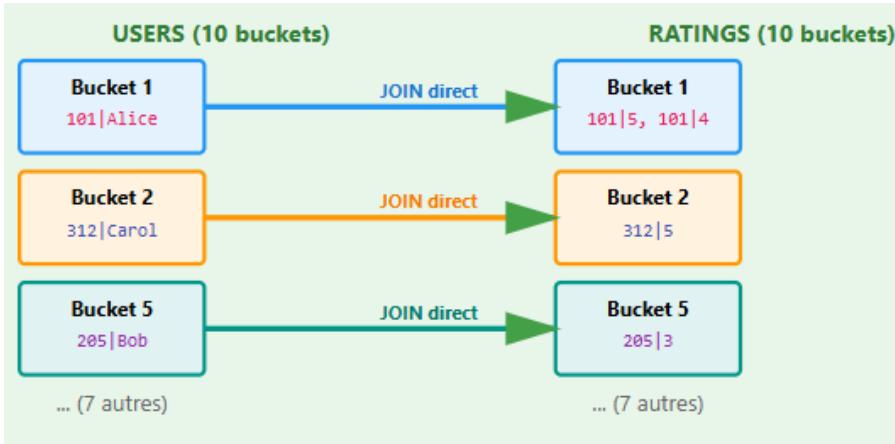
Cette requête calcule la note moyenne et le nombre total de notes par genre et par âge. Elle joint les tables **ratings_bucketed** et **users_bucketed** sur **user_id** (join très rapide grâce au bucketing), puis regroupe les résultats par genre et age pour calculer les statistiques. Le résultat final est trié par genre puis par âge, et montre par exemple que, les femmes de 18 ans donnent 3.45/5 (45427 notes) et les hommes de 18 ans donnent 3.53/5 (138109 notes), etc.....

Exemple de fonctionnement de jointure avec le backting hive :

Jointure sans bucketing



Jointure Avec bucketing



Sans bucketing (en haut) : Lors d'un JOIN classique, Hive doit mélanger toutes les données des deux tables via un SHUFFLE. Toutes les lignes de **USERS** et **RATINGS** sont transférées sur le réseau pour trouver les correspondances sur **user_id**. C'est très lent et consomme beaucoup de ressources réseau.

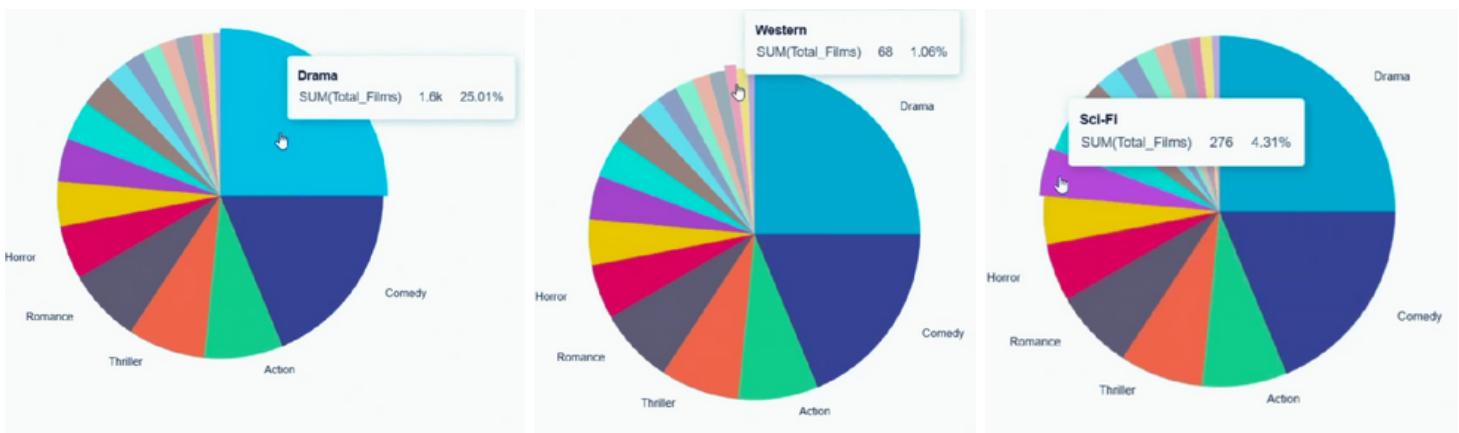
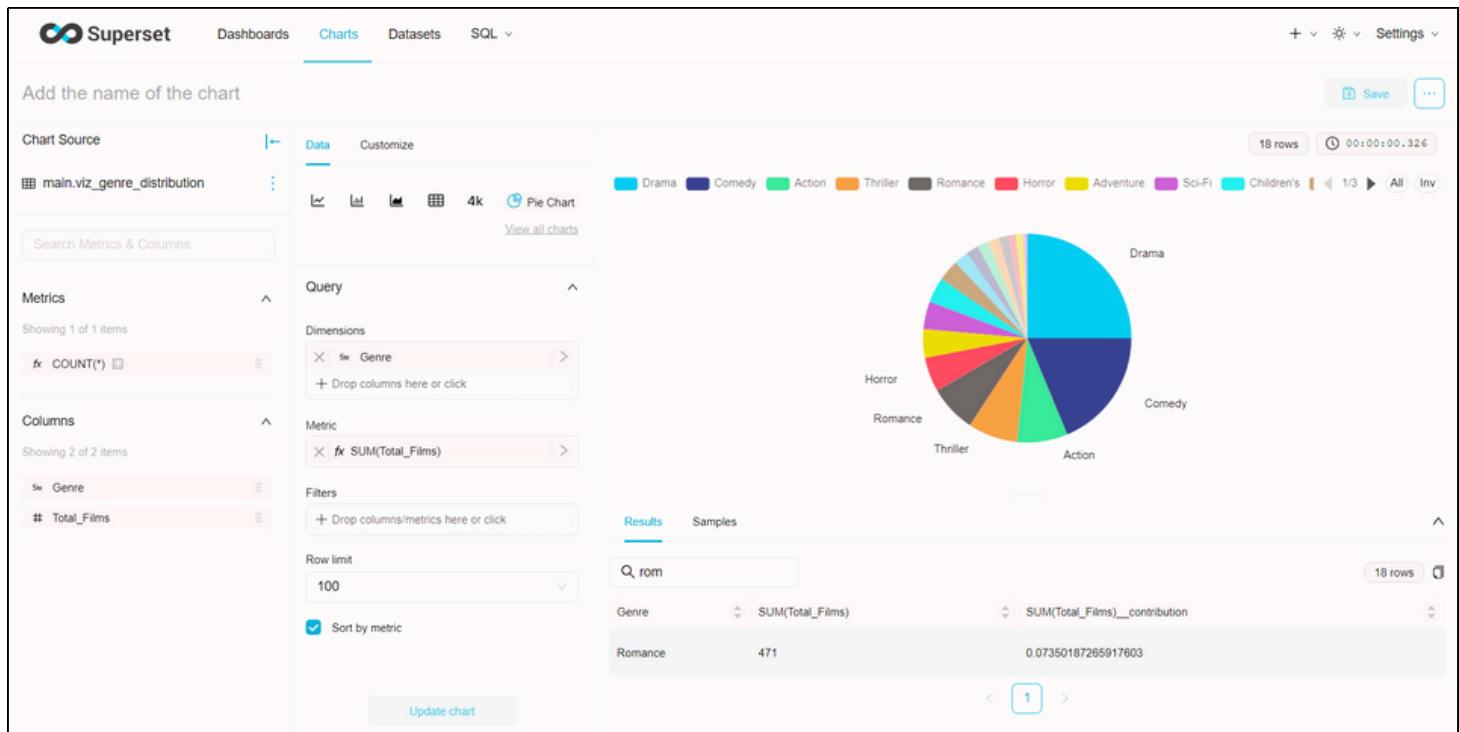
Avec bucketing (en bas) : Les deux tables sont divisées en 10 buckets basés sur **user_id**. Hive effectue un JOIN direct entre les buckets correspondants : Bucket 1 de **USERS** rejoint uniquement Bucket 1 de **RATINGS**, Bucket 2 avec Bucket 2, etc. Comme tous les **user_id** identiques sont déjà dans les mêmes buckets, aucun shuffle n'est nécessaire. Chaque bucket se joint indépendamment en parallèle, ce qui rend l'opération beaucoup plus rapide (10 à 100 fois plus rapide selon la taille des données).

3. Vizualisation du avec Apache superset

Dans un contexte professionnel, ce type d'analyse relève du travail d'un **data analyst**. À l'aide d'outils tels qu'**Apache Superset**, il est possible de visualiser et d'explorer efficacement de grands volumes de données.

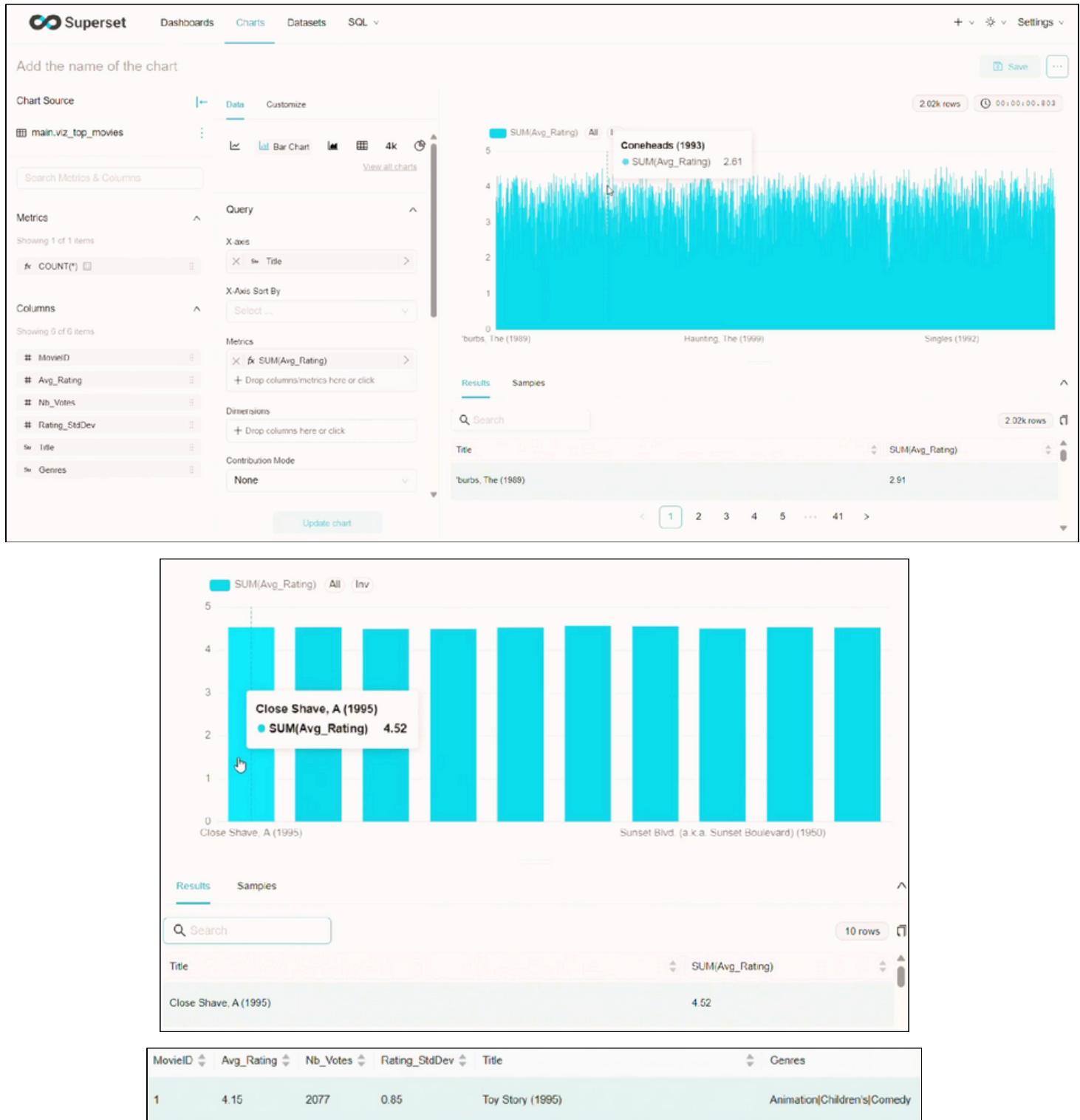
Lien de démonstration à consulter :

<https://e.pcloud.link/publink/show?code=XZByPNZx4f2dGjcKym5Uh77DEEljCiU7Wk>

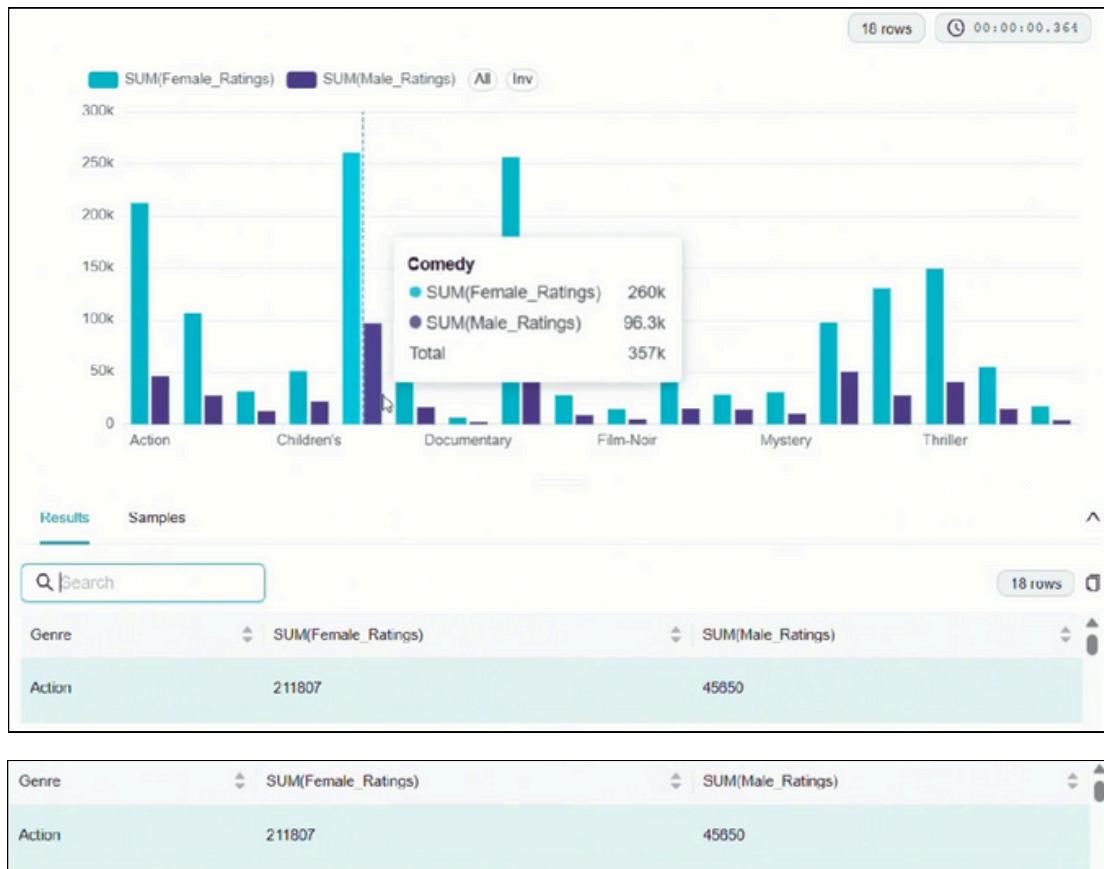
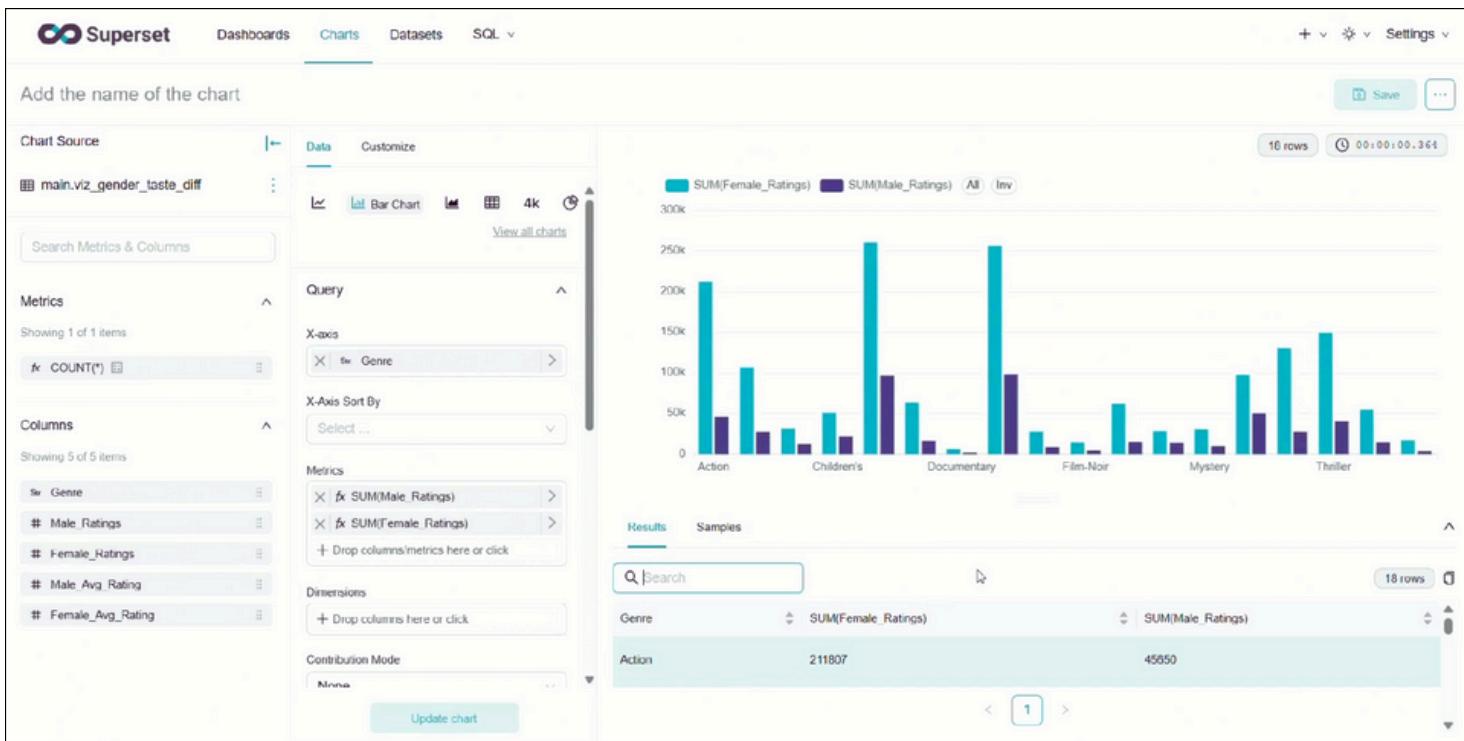


Comme le montre le diagramme, on peut visualiser le nombre total de films par genre. On y constate que les films de **drame** représentent **25,01 %** des titres, ce qui en fait la catégorie la plus importante du jeu de données. Viennent ensuite les films de **science-fiction** avec **4,31 %**, tandis que les **westerns** arrivent en dernière position avec seulement **1,06 %**.

Nous pouvons observer ici une analyse **des meilleurs films basée sur leurs notes**. Grâce à Apache Superset, nous avons tenté de visualiser à la fois le nombre de votes reçus par chaque film et sa note moyenne. Cette approche permet, dans le cadre d'un travail de data analyst, d'identifier les films ayant obtenu les meilleures évaluations.



Pour une visualisation plus claire, j'ai limité l'affichage à un échantillon de 10 films. D'après l'analyse, on peut observer que Close Shave (1995) obtient la meilleure note moyenne avec 4,52, suivi de Toy Story (1995) qui atteint 4,15. À l'inverse, Coneheads (1993) enregistre la note la plus basse de l'échantillon avec 2,91.



Ici, nous cherchons à identifier ***les différences de préférences cinématographiques entre les femmes et les hommes***. On peut remarquer que, pour ***les films de comédie***, ***les femmes sont plus favorables que les hommes, tandis que c'est l'inverse pour les films d'action***, où ***les hommes montrent une préférence plus marquée***. Cela démontre l'utilité des outils d'analyse de données pour mieux cibler les personnes selon leurs goûts.

En ce qui concerne le nombre de votes, on observe que, pour les comédies, les femmes ont attribué 260 000 votes, contre 96 300 pour les hommes. Dans le cas des films d'action, les femmes en ont compté 211 807, alors que les hommes en totalisent 456 500.