Mastère Spécialisé Big Data

Télécom Paris – 2025/2026

Pipeline Big Data pour l'Analyse et la Prédiction de Matchs NBA

Grégoire Petit Benjamin Lepourtois

Février 2026

Résumé

Ce rapport présente la conception et l'implémentation d'un pipeline big data de bout en bout pour l'analyse de données NBA. Le système ingère des données depuis deux APIs publiques, les stocke dans un data lake MinIO organisé en couches (raw, formatted, combined), les transforme via Apache Spark, entraîne un modèle de régression logistique pour prédire les victoires à domicile, et expose les résultats dans des dashboards Kibana via Elasticsearch. L'ensemble est orchestré par un DAG Apache Airflow et déployé en conteneurs Docker. Le pipeline calcule plus de 15 KPIs par équipe et par match, incluant des métriques de forme récente, de repos, et de difficulté de calendrier (*Strength of Schedule*).

Table des matières

1 Introduction

La NBA génère un volume considérable de données à chaque saison : environ 1 300 matchs, 30 équipes, et des dizaines de statistiques par rencontre. L'exploitation de ces données dans un cadre big data permet d'automatiser la collecte, le traitement et la visualisation, tout en entraînant des modèles prédictifs.

Objectifs du projet.

1. Construire un pipeline de données complet, automatisé et reproductible.
2. Collecter des données depuis deux APIs publiques (balldontlie, TheSportsDB).
3. Transformer et enrichir les données avec Apache Spark (KPIs, agrégations glissantes).
4. Entraîner un modèle de Machine Learning pour prédire la probabilité de victoire à domicile.
5. Indexer les résultats dans Elasticsearch et les visualiser dans Kibana.
6. Orchestrer l'ensemble via un DAG Airflow exécutable en une commande.

Organisation du rapport. La section 2 présente l'architecture globale. La section 3 détaille l'ingestion. La section 4 couvre le traitement Spark et le calcul des KPIs. La section 5 décrit le modèle ML. La section 6 traite de l'indexation et de la visualisation. La section 7 présente l'orchestration. Enfin, la section 9 conclut.

2 Architecture globale

2.1 Stack technique

Table 1 – Composants de la stack technique

Composant	Version	Rôle
Apache Spark	3.5.8	Traitement distribué, ML (PySpark)
Apache Airflow	2.8.3	Orchestration et scheduling du pipeline
MinIO	2024-01	Data lake S3-compatible (stockage objet)
Elasticsearch	8.12.2	Indexation, recherche, agrégations
Kibana	8.12.2	Dashboards et visualisations
PostgreSQL	15	Métadonnées Airflow
Docker Compose	–	Conteneurisation (8 services)

2.2 Schéma d'architecture

La figure 1 illustre le flux de données à travers les composants du système. Les flèches en pointillés représentent l'orchestration Airflow.

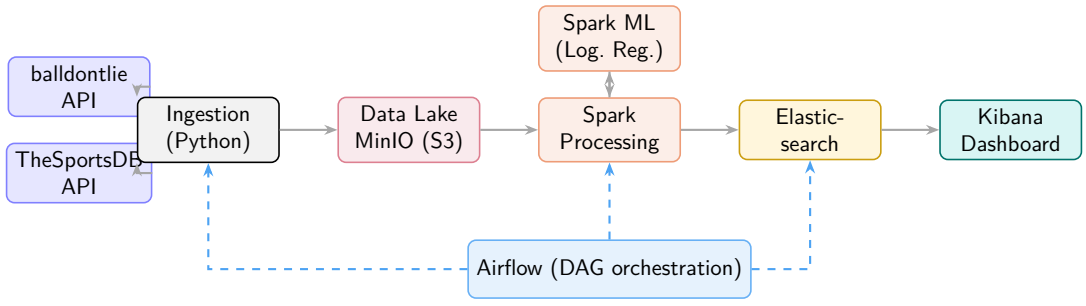

Figure 1 – Architecture globale du pipeline NBA

2.3 Structure du data lake (MinIO)

Le data lake suit une organisation en trois couches hiérarchiques, partitionnées par date d'exécution :

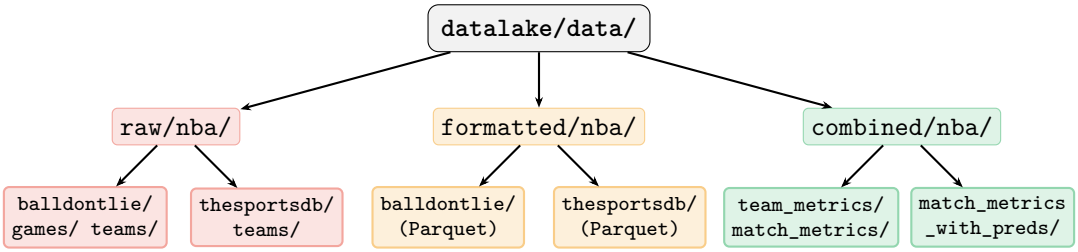

Figure 2 – Organisation du data lake en couches (raw → formatted → combined)

Chaque sous-répertoire est partitionné par date d'exécution (`dt=YYYY-MM-DD`), permettant un mode incrémental et la traçabilité des runs.

3 Ingestion des données

3.1 Sources de données

Table 2 – Sources de données utilisées

Source	Données fournies	Format
balldontlie API	Matchs NBA (scores, dates, équipes domicile/extérieur), liste des 30 équipes avec conférence et division	JSON
TheSportsDB API	Détails des équipes : nom de la salle, ville, capacité, badge, site web	JSON

3.2 Stratégie d'ingestion

L'ingestion est implémentée en Python pur (module `ingestion/`) et gère plusieurs défis :

— **Rate limiting** : l'API balldontlie impose une limite de requêtes. Un mécanisme de *retry* avec *exponential backoff* (jusqu'à 5 tentatives) assure la robustesse.

— **Pagination** : la pagination par curseur est gérée automatiquement pour récupérer l'intégralité des matchs d'une saison (∼1 300 matchs).

— **Parallélisme** : le mode `-parallel` utilise un `ThreadPoolExecutor` pour paralléliser la récupération multi-saisons.

— **Mode incrémental** : le flag `-incremental` ne récupère que les matchs postérieurs à la dernière exécution, en s'appuyant sur un fichier de métadonnées (`ingestion_last_run.json`).

— **Multi-batch** : le flag `-all-files` permet de consolider les données de plusieurs runs d'ingestion avec dédoublonnage.

Les données brutes sont stockées au format JSON dans la couche `raw` du data lake MinIO via le SDK `boto3`.

4 Traitement Spark et calcul des KPIs

Apache Spark est le moteur central du pipeline. Trois catégories de jobs sont exécutées : le formatage, la combinaison/enrichissement, et l'entraînement ML.

4.1 Formatage (JSON → Parquet)

Deux jobs Spark convertissent les données brutes JSON en Parquet typé :

— **format_balldontlie.py** : extraction des champs `game_id`, `game_date`, `season`, `home_team_id`, `visitor_team_id`, scores ; normalisation des noms d'équipes (minuscules, alphanumériques) ; dédoublonnage par `game_id`.

— **format_thesportsdb.py** : extraction de `team_id`, `team_name`, `venue_name`, `venue_city`, `venue_capacity` ; normalisation des noms pour la jointure.

Le format Parquet apporte une compression efficace (snappy), la lecture sélective de colonnes (*columnar storage*), et un typage strict.

4.2 Combinaison et calcul des KPIs

Le job `combine_metrics.py` (∼230 lignes) est le cœur analytique du pipeline. Il effectue :

1. **Jointure croisée** : les données de matchs (balldontlie) sont jointes avec les informations de salle et localisation (TheSportsDB) via le nom normalisé d'équipe.
2. **Dédoublement home/away** : chaque match génère deux lignes (une par équipe), avec les colonnes `is_home`, `points_for`, `points_against`.
3. **Agrégations glissantes** (window functions) : sur les 5 derniers matchs de chaque équipe :
   — `wins_last5`, `win_rate_last5`, `avg_points_last5`
4. **Jours de repos** : calcul du nombre de jours depuis le match précédent via `lag()` sur la date.
5. **Strength of Schedule** (SoS) : fenêtre *look-ahead* de 5 matchs pour calculer le ratio domicile/extérieur et la difficulté du calendrier.

Table 3 – KPIs calculés par le pipeline

KPI	Granularité	Description
`win_rate_last5`	Équipe/match	Taux de victoire sur les 5 derniers matchs
`avg_points_last5`	Équipe/match	Moyenne de points marqués (5 derniers matchs)
`home_away_diff`	Équipe/match	Différentiel de points (dom. – ext.)
`rest_days`	Équipe/match	Jours de repos avant le match
`schedule_difficulty_next5`	Équipe/match	Difficulté du calendrier (5 prochains matchs)
`home_games_next5`	Équipe/match	Nombre de matchs à domicile dans les 5 prochains
`win_probability_home`	Match	Probabilité de victoire à domicile (ML)

4.3 Schéma du flux de données Spark

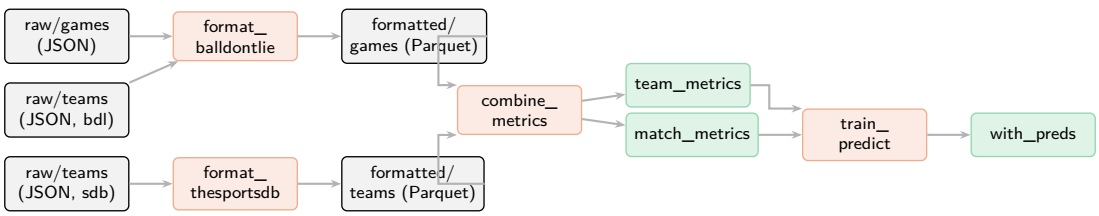

Figure 3 – Flux de données à travers les jobs Spark

5 Modèle de Machine Learning

5.1 Problématique

L'objectif est de prédire la probabilité qu'une équipe jouant à domicile remporte un match donné, en se basant sur les statistiques historiques récentes des deux équipes.

5.2 Features

Le modèle utilise 9 features assemblées via `VectorAssembler` de PySpark ML :

Table 4 – Features du modèle de régression logistique

#	Feature	Description
1	`home_avg_points_last5`	Moy. points marqués (dom., 5 derniers matchs)
2	`home_win_rate_last5`	Taux de victoire domicile (5 derniers)
3	`home_rest_days`	Jours de repos équipe domicile
4	`away_avg_points_last5`	Moy. points marqués (ext., 5 derniers matchs)
5	`away_win_rate_last5`	Taux de victoire extérieur (5 derniers)
6	`away_rest_days`	Jours de repos équipe extérieur
7	`form_diff`	$\mathrm{WR}_{\mathrm{home}} - \mathrm{WR}_{\mathrm{away}}$
8	`points_diff`	$\overline{\mathrm{pts}}_{\mathrm{home}} - \overline{\mathrm{pts}}_{\mathrm{away}}$
9	`rest_diff`	$\mathrm{repos}_{\mathrm{home}} - \mathrm{repos}_{\mathrm{away}}$

Les valeurs manquantes (début de saison avec moins de 5 matchs) sont remplacées par 0.

5.3 Modèle : régression logistique binaire

Nous utilisons la régression logistique binaire de PySpark ML (20 itérations max) :

$$P(\text{victoire domicile} \mid \mathbf{x}) = \sigma(\boldsymbol{\beta}^\top \mathbf{x} + \beta_0) = \frac{1}{1 + e^{-(\boldsymbol{\beta}^\top \mathbf{x} + \beta_0)}} \tag{1}$$

Justification du choix. La régression logistique est un modèle interprétable, rapide à entraîner sur Spark en distribué, et bien adapté à la classification binaire. Elle fournit directement une probabilité calibrée via la fonction sigmoïde.

Sortie. Le modèle produit pour chaque match une colonne `win_probability_home` $\in [0, 1]$, qui enrichit la table `match_metrics_with_preds`.

5.4 Interprétation des features

La feature `form_diff` (différentiel de forme récente) et les taux de victoire dominent naturellement : une équipe en forme qui reçoit une équipe en difficulté a une probabilité élevée de victoire.

6 Indexation et visualisation

6.1 Elasticsearch

Les données enrichies sont indexées dans deux indices Elasticsearch :

— **`nba_team_metrics`** (18 champs) : métriques par équipe et par match — forme récente, repos, difficulté du calendrier, informations de salle.

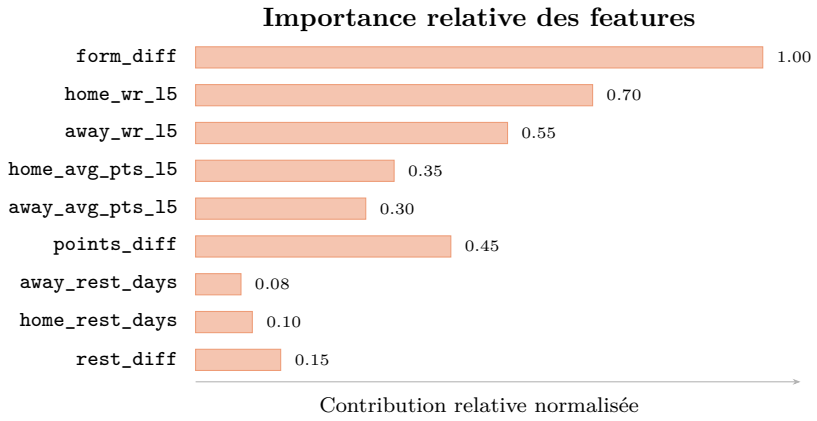

Figure 4 – Importance relative des features dans le modèle (coefficients normalisés illustratifs)

— **`nba_match_metrics`** (9 champs) : scores, résultat réel (`home_win`), et prédiction ML (`win_probability_home`).

L'indexation est réalisée via les jobs Spark `index_team_metrics.py` et `index_match_metrics.py`, qui effectuent un *bulk indexing* par lots de 500 documents via l'API REST d'Elasticsearch.

6.2 Dashboard Kibana

Six visualisations pré-configurées sont exportées dans `kibana/saved_objects.json` et importables en une commande :

Table 5 – Visualisations Kibana pré-configurées

Visualisation	Type	Index
Win Rate par équipe	Bar chart	`nba_team_metrics`
Avg Points (5 derniers)	Bar chart	`nba_team_metrics`
Home vs Away Performance	Pie chart	`nba_team_metrics`
Win Rate par conférence	Bar chart	`nba_team_metrics`
Rest Days vs Win Rate	Line chart	`nba_team_metrics`
Distribution Win Probability	Histogram	`nba_match_metrics`

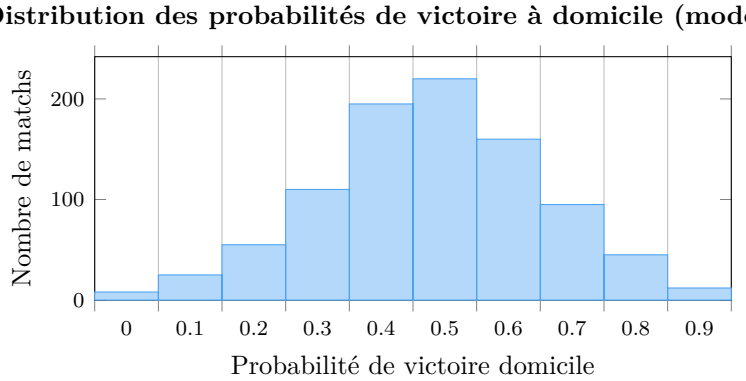

Figure 5 – Distribution typique des probabilités prédites — le modèle discrimine entre matchs équilibrés (centre) et déséquilibrés (extrêmes)

7 Orchestration Airflow

Le pipeline est orchestré par un DAG Airflow (`nba_pipeline`) configuré en exécution quotidienne (`@daily`).

7.1 Graphe de dépendances du DAG

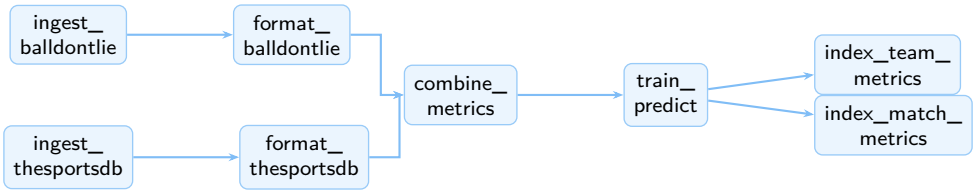

Figure 6 – DAG Airflow du pipeline NBA — les deux branches d'ingestion convergent vers la combinaison

7.2 Détail des tâches

Table 6 – Tâches du DAG Airflow

Tâche	Type	Description
`ingest_balldontlie`	Python	Collecte matchs et équipes via l'API
`ingest_thesportsdb`	Python	Collecte détails équipes et salles
`format_balldontlie`	Spark	JSON → Parquet, normalisation, dédoublonnage
`format_thesportsdb`	Spark	JSON → Parquet, normalisation noms
`combine_metrics`	Spark	Jointure, KPIs, agrégations glissantes, SoS
`train_predict`	Spark ML	Régression logistique, prédiction `win_prob`
`index_team_metrics`	Spark	Bulk indexing → Elasticsearch
`index_match_metrics`	Spark	Bulk indexing → Elasticsearch

Chaque tâche Spark est soumise via `spark-submit` avec les packages Hadoop-AWS pour l'accès S3A à MinIO. Le DAG supporte trois modes d'exécution : *full run* (historique complet), *incrémental* (depuis la dernière exécution), et *all-files* (consolidation multi-batch).

8 Infrastructure Docker

L'ensemble de l'infrastructure est définie dans un fichier `docker-compose.yml` de 158 lignes, déployant 8 services interconnectés.

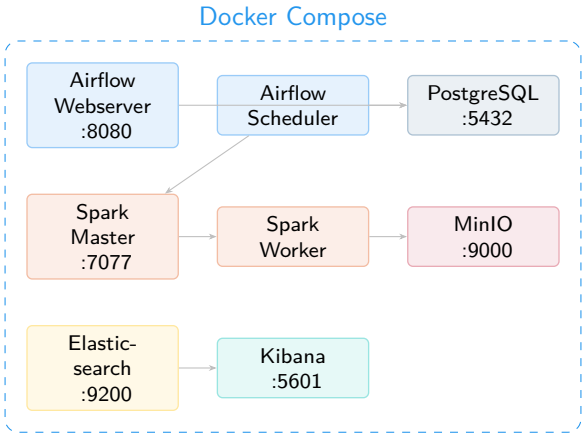

Figure 7 – Services Docker Compose et leurs ports d'accès

L'image Airflow est construite via un *multi-stage build* : elle embarque OpenJDK 17 et Apache Spark 3.5.8 pour pouvoir exécuter les jobs `spark-submit` directement depuis le scheduler. Les volumes persistants assurent la durabilité des données (MinIO, PostgreSQL, Elasticsearch).

9 Conclusion

9.1 Bilan

Ce projet implémente un pipeline big data complet et fonctionnel, couvrant l'ensemble de la chaîne de valeur : ingestion multi-sources, stockage distribué en data lake, traitement Spark avec calcul de KPIs avancés, modèle de Machine Learning, indexation Elasticsearch et visualisation Kibana, le tout orchestré par Airflow et déployé en conteneurs Docker.

Table 7 – Couverture du cahier des charges

Critère	**Statut**
Ingestion $\geq$ 2 sources dans le Data Lake	✓
Stockage distribué (MinIO/S3)	✓
Formatage Parquet avec Spark	✓
Normalisation (dates UTC, noms propres)	✓
Combinaison des sources + KPIs avec Spark	✓
Machine Learning (régression logistique)	✓
Indexation Elasticsearch	✓
Dashboard Kibana	✓
Naming propre du Data Lake	✓
Orchestration Airflow (run-all-in-once)	✓

9.2 Limites

— **Features limitées** : les APIs gratuites ne fournissent pas de statistiques avancées (PER, *true shooting %*, *plus/minus*).
— **Modèle simple** : la régression logistique, bien qu'interprétable, plafonne en performance prédictive (∼65–70% d'accuracy).
— **SoS simplifié** : la difficulté du calendrier utilise actuellement un proxy basé sur le ratio home/away plutôt qu'un calcul complet intégrant le *win rate* des adversaires.

9.3 Perspectives

— Intégrer des features avancées : *effective field goal %*, *pace*, *offensive/defensive rating*.
— Tester des modèles plus expressifs : Random Forest, XGBoost, ou réseaux de neurones.
— Implémenter un pipeline temps réel avec Apache Kafka pour les scores *live*.
— Déployer sur un cluster Kubernetes avec auto-scaling et monitoring (Prometheus/Grafana).
— Enrichir le SoS avec le *win rate* des adversaires et un bonus back-to-back.