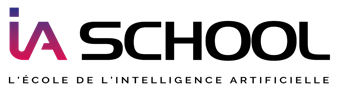


PROJET BIG DATA

Semestre 1



Porteurs du projet : Groupe 6

Aboubacar Sidiki SIDIBÉ

Amadou lamarana DIALLO

Arnaud TCHIDA

Kindi DIALLO

SOMMAIRE

[INTRODUCTION 1](#_Toc73958253)

[1. ARCHITECTURE DE LA SOLUTION 2](#_Toc73958254)

[a) Ingestion de données (collecte et stockage) 2](#_Toc73958255)

[b) Pourquoi Couchbase ? 3](#_Toc73958256)

[c) Déploiement du cluster Couchbase 3](#_Toc73958257)

[d) Simulation d’un flux de données et upsertion dans la base de stockage 4](#_Toc73958258)

[2. TRAITEMENT DE DONNÉES 7](#_Toc73958259)

[a) Pourquoi Spark ? 7](#_Toc73958260)

[b) Configuration du connecteur Couchbase 8](#_Toc73958261)

[c) Analyse de données 8](#_Toc73958262)

[3. AFFICHAGE DES RESULTATS 9](#_Toc73958263)

[a) Nombre de transaction par type de carte 9](#_Toc73958264)

[b) Nombre de transaction effectuée par mois 10](#_Toc73958265)

[c) Nombre de type de carte par montant 10](#_Toc73958266)

[CONCLUSION 11](#_Toc73958267)

# **INTRODUCTION**

Le contexte inédit de la crise sanitaire a renforcé le besoin de mobiliser des sources de données « en temps réel », c’est‐à‐dire très rapidement disponibles, suffisamment représentatives et détaillées afin de pouvoir décrire les hétérogénéités des situations durant la crise et après. À cet effet, quoi de plus mieux que d’avoir des données bancaires qui sont particulièrement représentatifs des comportements des ménages. Ces données permettent d’étudier les comportements d’une bonne partie de la population en termes de consommation, d’épargne à travers leurs retraits et dépôts mais aussi et surtout le type d’achat le plus sollicité durant cette période. Ou encore les jours ou dates qui engendrent plus de fluctuations monétaires.

Notre démarche a pour but de répondre à la problématique évoquée ci-dessus (cad collecter, stocker et traité des données bancaires en temps réel et les visualiser…).

# **ARCHITECTURE DE LA SOLUTION**

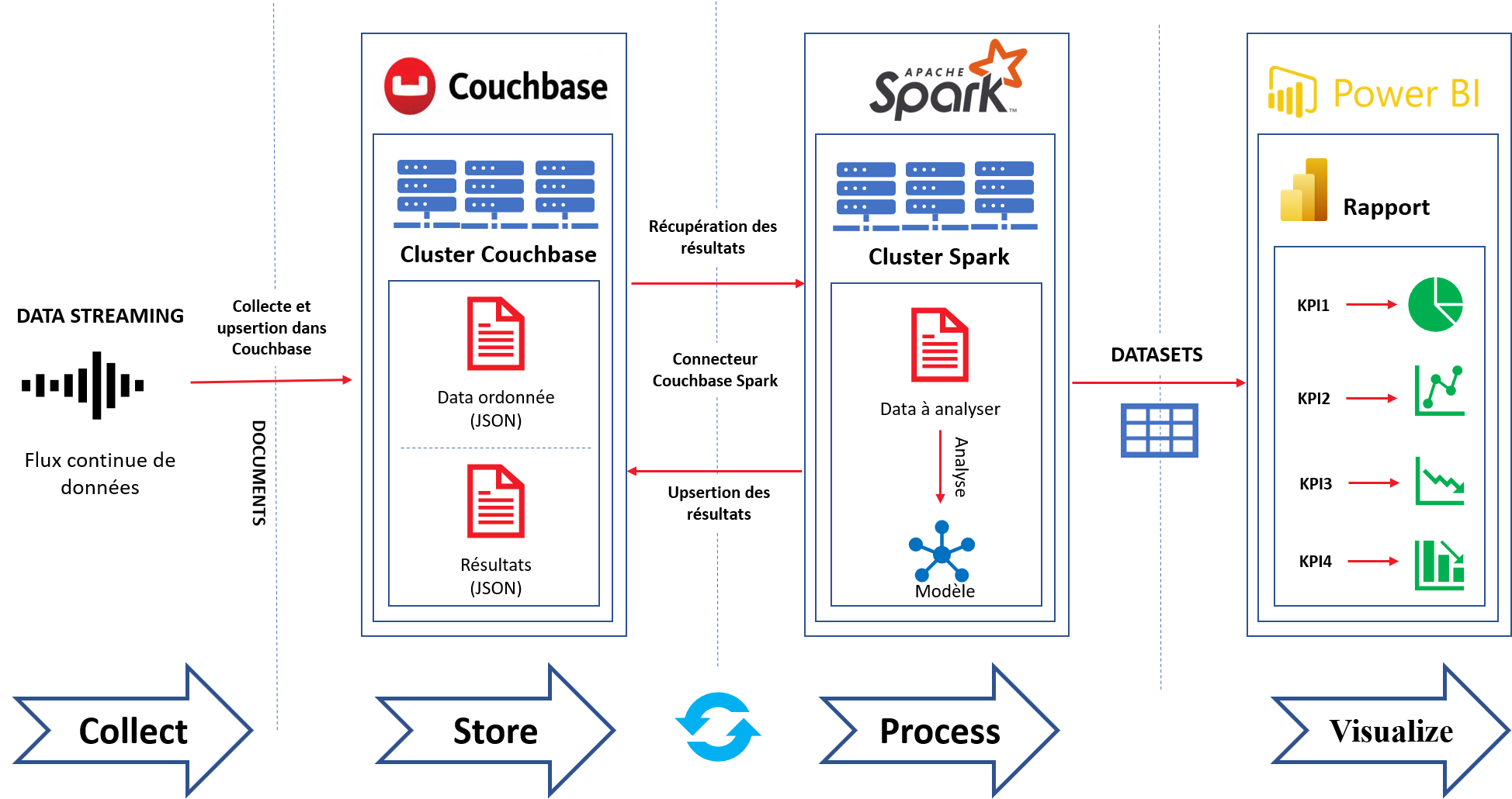


Figure 1 : Architecture générale de la solution

**COLLECT** : on simulera un flux qui sera traité en streaming

**STORE** : les données seront insérées de manière pérenne dans Couchbase notre base de stockage distribuée

**PROCESS** : la couche processing Spark détectera les mutations au sein de la base de stockage, traitera les données et puis restituera les résultats à la volée en les injectant dans Couchbase

**VISUALIZE** : les résultats peuvent être visualisés en pseudo temps réel

## **Ingestion de données (collecte et stockage)**

Dans la cadre notre projet nous avons opté pour une collecte de données bancaire arrivant en temps réels(streaming) à travers un script python ensuite ces données seront insérées de manière pérenne dans Couchbasse (dont nous expliquerons le choix plus tard) notre base de stockage distribuée.

## **Pourquoi Couchbase ?**

C’est une base de données NoSQL orientée document qui envoie et reçoit des données en s'appuyant sur le protocole JSON (JavaScript Object Notation). Le JSON, qui a donc la réputation d'être facile à comprendre et à utiliser, permet d'échanger des informations entre deux systèmes. « Ce protocole est particulièrement utile pour les applications qui envoient et reçoivent beaucoup de données périodiquement comme dans le cas notre projet.

En outre c’est une base de stockage distribuée qui est axée sur une architecture unique et centrée sur la mémoire, scalable, haut débit au niveau des flux, et surtout basée sur du NoSQL : tous ces critères rassemblés sont idéals pour faire du stockage en temps réel. Comparé à une base de données relationnelle classique, le NoSQL est bien plus scalable et délivre des performances bien supérieures. En plus, si vous voulez analyser un large volume de données dont la structure peut changer, Couchbase est votre allié.

## **Déploiement du cluster Couchbase**

Nous avons utilisé 3 machines dans le même sous réseau sur lesquelles nous avons déployé notre cluster Couchbase (voir figure 2 ci-dessous).

Cette configuration scalable nous permettra de supporter le flux de données entrant et facilitera le stockage en temps réel. Avec cette configuration, on n’aura pas besoin de nœuds maître et esclaves, tous les nœuds sont égaux, et les données sont répliquées automatiquement pour éviter un single point of failure. Ainsi, si un nœud n’est plus opérationnel, les données seront toujours accessibles.

On aura donc une base données distribuée qui concilie performance, scalabilité et disponibilité.

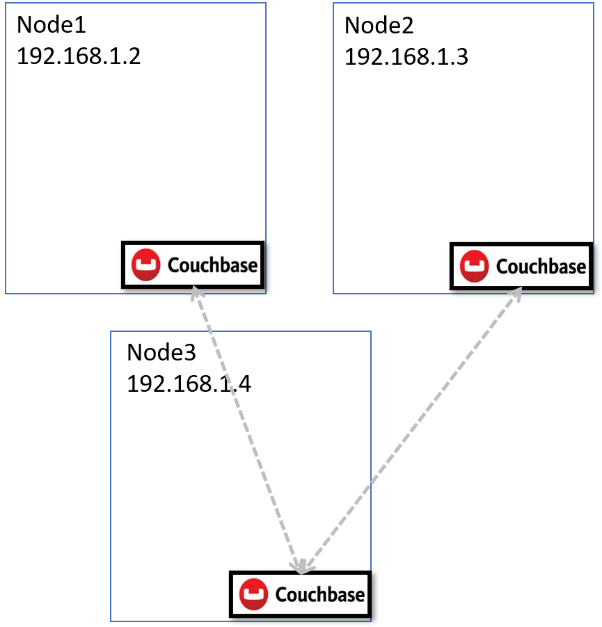


Figure 2 : cluster couchbase

## **Simulation d’un flux de données et upsertion dans la base de stockage**

Dans le cadre de ce projet, le flux sera généré automatiquement avec un script python, mais le même principe peut être appliqué à un flux provenant d’une API, datalake, etc. Nous avons simulé des transactions bancaires qui auront le même format : Transaction id, Card id, Card type, Name, Amount, @timestamp

Par exemple :

Transaction id: 123456789

Card id: 4815162342

Card type: Amex

Amount: 100.97$

@timestamp : 20-06-2017 13:50:23.345676

Les transactions seront générées manière non périodique, ce qui constituera notre flux de données, puis chaque transaction sera convertie dans un format JSON pour être insérée dans le cluster Couchbase.

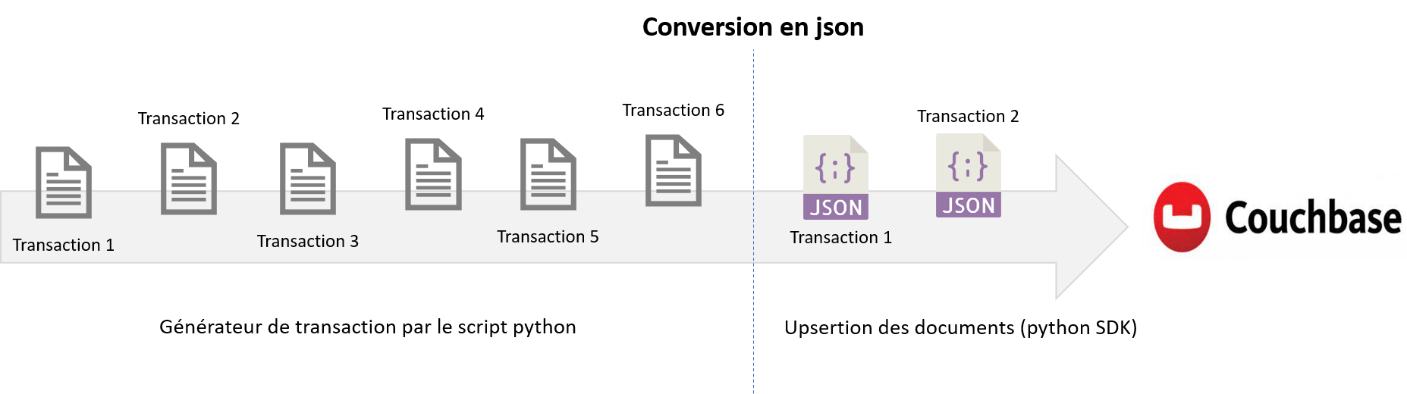


Figure 3 : schéma de génération et de conversion des transactions en json

Par exemple, la transaction montrée plus haut sera convertie au format JSON avant d’être insérée dans Couchbase :

{

     transaction\_id : 123456789,

     card\_id : 4815162342,

     card\_type : “Amex”,

     amount : 100.97,

     @timestamp : “20-06-2017 13:50:23.345676”

}

Les données seront générées sur un nœud distant ou interne à notre cluster.

Pour se faire, nous avons installé la SDK Python de Couchbase sur la machine où nous avons généré les données.

curl -O https://packages.couchbase.com/releases/couchbase-release/couchbase-release-1.0-amd64.deb

sudo dpkg -i ./couchbase-release-1.0-amd64.deb

sudo apt-get update

sudo apt-get install couchbase-server

Extrait de code source générateur de transactions

DEFAULT\_USER = 'admin'

PASSWORD = 'password'

URL = "couchbase://127.0.0.1"

cluster = Cluster(URL, ClusterOptions(PasswordAuthenticator(DEFAULT\_USER, PASSWORD)))

bucket = cluster.bucket('default')

transaction\_id = 0

card\_id = 0

card\_type\_list = ["Amex", "CB", "Visa", "Mastercard"]

amount = 0

timestamp\_event = datetime.datetime.now()

i=0

while(i<10000):

    i+=1

    if(random.randint(0, 10)>3):

        delay = random.uniform(0, 0.05)

    else:

        delay = random.uniform(0, 3)

    timestamp\_event += datetime.timedelta(seconds=delay)

    transaction\_id += 1

    card\_id = random.randint(0,10000)

    card\_type = card\_type\_list[random.randint(0,3)]

    amount = random.uniform(0, 1000)

     # create json

    transaction = {}

    transaction['transaction\_id'] = transaction\_id

    transaction['card\_id'] = card\_id

    transaction['card\_type'] = card\_type

    transaction['amount'] = amount

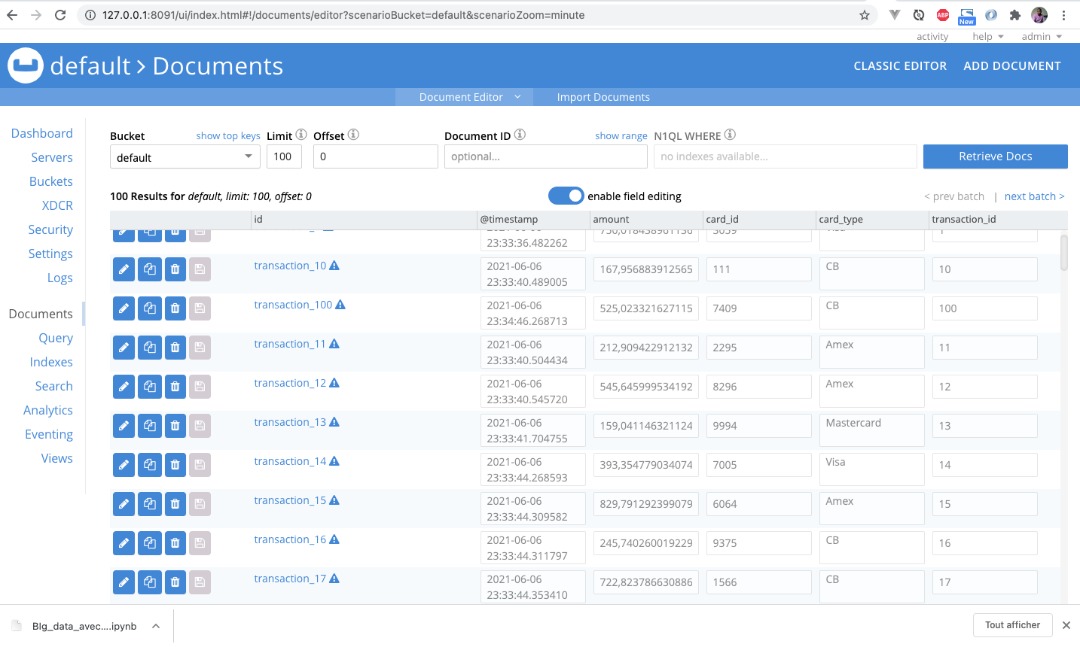
    transaction['@timestamp'] = str(timestamp\_event)

    print("Transsaction: "+ str(i)+" ", transaction)

     # upsert json in couchbase bucket

    bucket.upsert("transaction\_"+str(transaction\_id), transaction)

    time.sleep(delay)



Capture 1 : sauvegarde des transactions

# **TRAITEMENT DE DONNÉES**

Une fois ingérées dans Couchbase, nous utiliserons un cluster spark pour l’analyse de données. La couche processing Spark détectera les mutations au sein de la base de stockage, traitera les données et puis restituera les résultats à la volée en les injectant dans Couchbase et déployer le résultat sur un Dashboard pour la visualisation.

## **Pourquoi Spark ?**

Spark, dont la réputation n’est plus à refaire, permet des vitesses de traitement ultra-rapides (Map-reduce) et combine SQL, Streaming et analyses Machine Learning pour traiter les données. Grâce à sa librairie « ML », il est possible d’entrainer des modèles. La librairie « Spark streaming » combinée avec le connecteur Couchbase – Spark permet de récupérer les nouvelles données provenant de Couchbase en temps réel et de les confronter au modèle Spark. De la même manière que les données peuvent être récupérées, on pourra insérer les résultats dans Couchbase depuis Spark. L’autre principal avantage de spark est sa vitesse, puisqu’il permet de lancer des programmes 100 fois plus rapidement que Hadoop MapReduce in-memory, et 10 fois plus vite sur disque.

Comme nous l’avons souligné plus haut, d’autres outils peuvent être des alternatifs à spark notamment hadoop, MapReduce et bien d’autres mais tous accumulent du retard par rapport à spark aussi en termes de vitesse mais également d’offre.

## **Configuration du connecteur Couchbase**

Pour pouvoir faire l’analyse sur spark, un connecteur Couhbase devrait être configuré au préable afin qu’on puisse récupérer les données disponibles dans couchbase.

Nous avons donc utilisé la librairie pyspark.streaming pour assurer cette connexion.

## **Analyse de données**

Script d’analyse du flux de données issues de couchbase.

if \_\_name\_\_=='\_\_main\_\_':

    #spark = SparkSession.builder.config('couchbase://127.0.0.1','8091').getOrCreate()

    conf = SparkConf()

    conf.setMaster("local").setAppName("ProjetBigData")

    sc = SparkContext(conf=conf)

    sc.setLogLevel("ERROR")

    DEFAULT\_USER = 'admin'

    PASSWORD = 'password'

    URL = "couchbase://127.0.0.1"

    cluster = Cluster(URL, ClusterOptions(PasswordAuthenticator(DEFAULT\_USER, PASSWORD)))

    bucket = cluster.bucket('default')

    #log4jLogger = sc.\_jvm.org.apache.log4j

    #log = log4jLogger.LogManager.getLogger(\_\_name\_\_)

    #log.warn("Hello World!")

    #print(sc.version)

    ssc = StreamingContext(sc, 1)

    transactions = ssc.socketTextStream(URL, 8091)

    print(transactions.count().count())

    words = transactions.flatMap(lambda line: line.split(","))

    #print(words.count())

    # Count each word in each batch

    pairs = words.map(lambda word: (word, 1))

    wordCounts = pairs.reduceByKey(lambda x, y: x + y)

    # Print the first ten elements of each RDD generated in this DStream to the console

    wordCounts.pprint()

    ssc.start()

    ssc.awaitTermination()

Ce code permet de détecter les mutations au sein de la base de stockage puis d’en récupérer le contenu. Ensuite nous filtrons ce dont nous avons besoin (transaction\_id, amount, card\_type, @timestamp) pour le processer par la suite en temps réel. La récupération du contenu des documents n’est pas forcément explicite dans la documentation Couchbase, en effet seule la détection des mutations est traitée. Avec ce code nous avons pu récupérer le contenu des documents JSON qui sont déversés en temps réel dans la base de stockage distribuée.

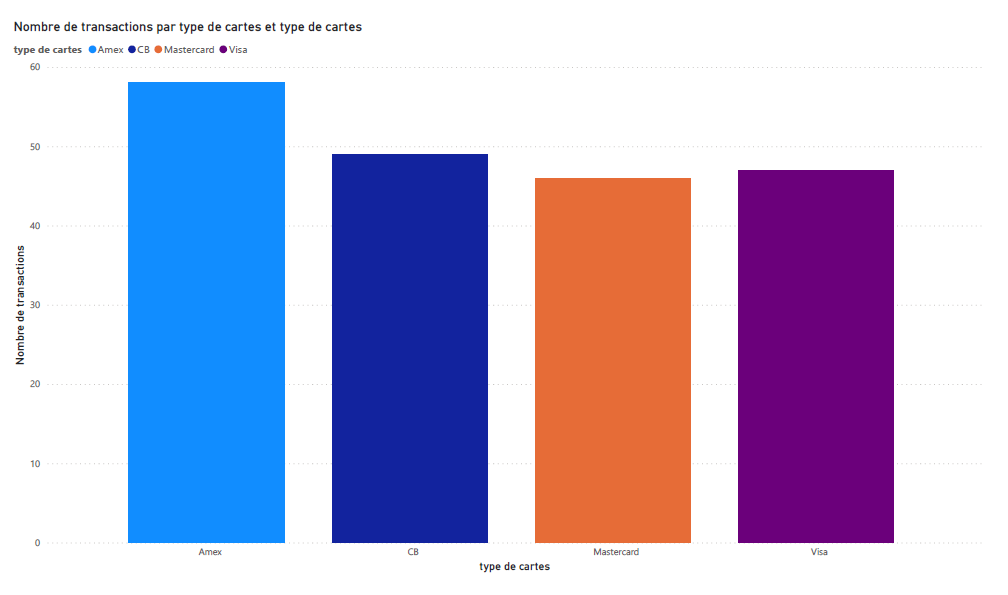
# **AFFICHAGE DES RESULTATS**

Pourla visualisation des données, nous avons utilisé Powerbi. Power BI est une solution d'analyse de données de Microsoft. Il permet de créer des visualisations de données personnalisées et interactives avec une interface suffisamment simple pour que les utilisateurs finaux créent leurs propres rapports et tableaux de bord.

Une connexion a été réalisée au préable afin de permet à powerbi de lire de manière automatique le dataset issue de l’analyse.

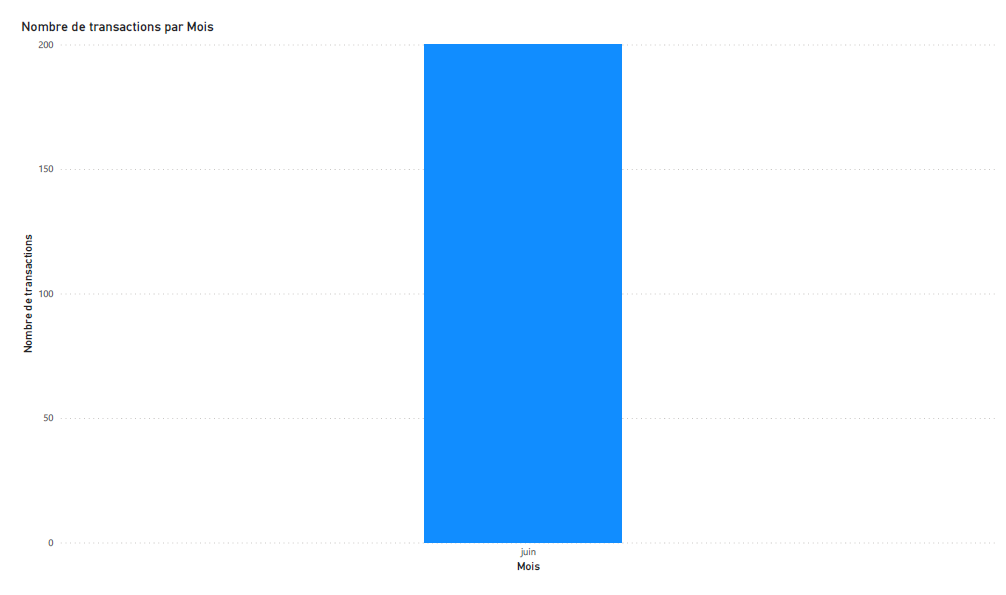
Pour ce projet, trois KPIs (Key Performance Indicator) ont été choisies pour visualiser les données issues de l’analyse qui sont :

## **Nombre de transaction par type de carte**



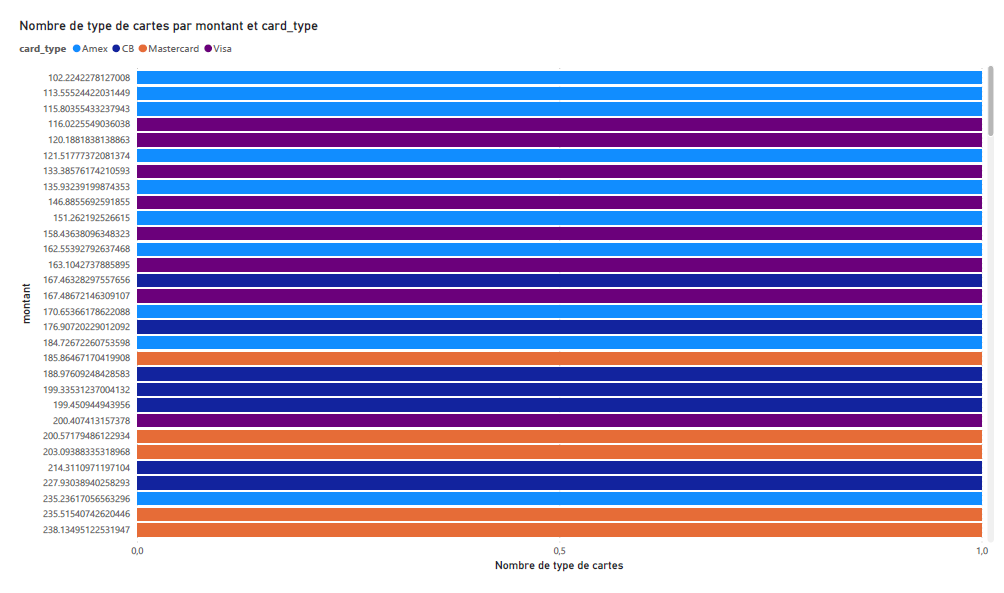
On remarque que le type de carte « amex » est beaucoup plus utilisé par les clients comparés aux autres cartes répertoriées.

## **Nombre de transaction effectuée par mois**



On remarque que toutes les transactions ont été effectuées pendant le mois de juin ce qui s’explique par le fait que le flux de données généré par notre script python a été exécuté que pendant ce mois-ci. Le graphe aurait été différent si le générateur était lancé sur deux, trois ou plusieurs mois.

## **Nombre de type de carte par montant**



On remarque que sur ce graphe aucun client n’a effectué aucune transaction avec le même montant.

# **CONCLUSION**

Comme nous l’avons vu précédemment, nous avons réussi à mettre en place une architecture temps réel, l’architecture Couchbase / Spark qui est capable de collecter des données en streaming, de sauvegarder les flux de données dans une base de données distribuée (Couchbase), de traiter et analyser ces données en temps réel et de visualiser par la suite ces données via des graphes sur powerbi.

On pouvait créer notre propre modèle machine learning grâce à la librairie ML de Spark ce qui nous permettra d’avoir une architecture beaucoup plus complète.