Faculté des sciences et techniques, Beni Mellal

Dashboard Crypto Temps Réel avec Kafka et MongoDB

Réalisé par :
- Bel-assal Mohamed
- El Boubkraoui Farid

Encadré par :
- Pr. Chakour Imane

Année académique : 2024-2025

1 Introduction

Les cryptomonnaies, telles que le Bitcoin et l'Ethereum, ont révolutionné le paysage financier mondial en introduisant des formes de monnaie numérique décentralisées, sécurisées par la cryptographie et fonctionnant sur des technologies de registre distribué comme la blockchain. Depuis la première transaction en Bitcoin en 2010, où 10 000 BTC ont été échangés contre deux pizzas, la valeur et l'adoption des cryptomonnaies ont connu une croissance exponentielle, atteignant des capitalisations de marché de plusieurs trillions de dollars.
Cette évolution rapide a engendré une explosion des données générées par les transactions, les échanges et les activités des utilisateurs dans l'écosystème des cryptomonnaies. Pour exploiter efficacement ces flux de données en temps réel, il est essentiel de disposer d'une architecture robuste capable de collecter, traiter, stocker et visualiser ces informations de manière efficiente. Ce projet vise à concevoir et implémenter une telle architecture, en s'appuyant sur Apache Kafka pour la diffusion en temps réel des données, MongoDB pour le stockage flexible et évolutif des informations, et une application web interactive pour la visualisation des données. Cette approche permettra de surveiller les tendances du marché des cryptomonnaies, d'analyser les comportements des utilisateurs et de prendre des décisions éclairées basées sur des données actualisées en continu.

2 Apache Kafka : Fondements et Architecture

Apache Kafka est une plateforme de diffusion d'événements distribuée conçue pour gérer des flux de données en temps réel avec une haute performance, une faible latence et une tolérance aux pannes. Initialement développé par LinkedIn et maintenant maintenu par la Fondation Apache, Kafka est largement utilisé pour construire des pipelines de données en continu et des applications de streaming.

2.1 Architecture et Composants

L'architecture de Kafka repose sur quelques composants fondamentaux. Les producteurs sont les entités qui publient des messages dans des canaux appelés topics. Chaque topic est subdivisé en partitions, qui assurent une répartition parallèle des données pour un traitement plus rapide et plus scalable.
Les brokers sont les serveurs qui reçoivent et stockent les messages. Un cluster Kafka est constitué de plusieurs brokers, ce qui permet de distribuer la charge et de garantir la résilience du système. Pour assurer la disponibilité, chaque partition peut être répliquée sur plusieurs brokers. Un leader gère la lecture et l'écriture de chaque partition, tandis que les autres serveurs (appelés réplicas) assurent la redondance.
Du côté de la consommation, les consommateurs lisent les messages des topics. Ils peuvent être regroupés en groupes de consommateurs pour permettre une lecture parallèle sans doublon.

Kafka suit la position de lecture de chaque consommateur à l'aide d'un identifiant appelé offset, ce qui permet de reprendre le traitement en cas de redémarrage.
Jusqu'à la version 2.8, Kafka s'appuyait sur ZooKeeper pour la gestion des métadonnées du cluster. Aujourd'hui, Kafka peut fonctionner sans ce composant grâce à une architecture dite « KRaft », qui simplifie le déploiement et l'administration du système.

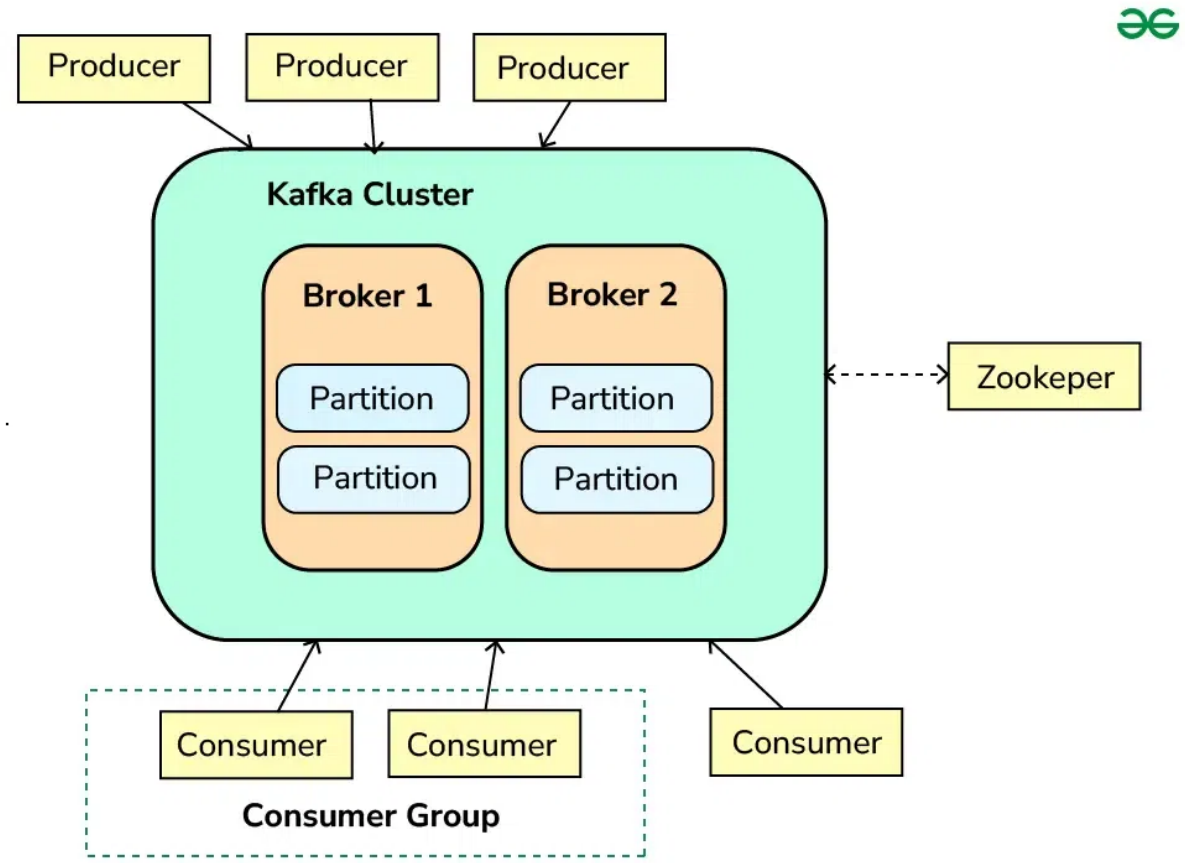

FIGURE 1 – Schéma de l'Architecture Kafka (Geeksforgeeks.org)

2.2 Fonctionnement Général

Le processus d'échange de données dans Kafka peut être résumé comme suit : les producteurs envoient des données vers des topics définis, lesquelles sont distribuées et stockées dans les partitions par les brokers. Ces messages sont ensuite consommés par les applications abonnées aux topics, permettant ainsi un traitement en continu ou différé.
Ce modèle découplé entre producteurs et consommateurs permet à Kafka d'être extrêmement flexible. Dans notre projet, cela est essentiel pour traiter en temps réel les flux de données issus des marchés de cryptomonnaies, tout en assurant une persistance fiable via MongoDB.

3 Intégration de Kafka avec MongoDB

Pour réaliser l'intégration de données en temps réel dans notre projet de suivi des cryptomonnaies, nous avons développé une architecture basée sur Apache Kafka en tant que système de streaming, MongoDB comme base de données persistante.

3.1 Producteur Kafka

Le producteur Kafka est responsable de la collecte des données de prix et volume des cryptomonnaies à partir de l'API CoinMarketCap, et de la publication périodique de ces informations sur le topic Kafka crypto-prices.
Nous utilisons la bibliothèque Python Kafka pour initialiser un KafkaProducer avec sérialisation JSON. Les données sont récupérées toutes les 5 minutes grâce à des requêtes sécurisées et robustes à l'API, avec gestion des erreurs et tentatives de retry.
Une logique spécifique permet de maintenir un historique glissant sur 1 heure des prix et volumes pour chaque crypto, afin de calculer des métriques dynamiques telles que le prix haut/bas sur 1 heure et le volume échangé.

```
producer = KafkaProducer(
    bootstrap_servers=['localhost:9092'],
    value_serializer=lambda v: json.dumps(v).encode('utf-8'),
    retries=5,
    request_timeout_ms=30000
)
```

Cette approche garantit une ingestion continue et fiable des données dans le système Kafka.

3.2 Consommateur Kafka

Le consommateur Kafka, également écrit en Python, s'abonne au topic crypto-prices, désérialise les messages JSON et procède à leur insertion dans MongoDB.

```
consumer = KafkaConsumer(
        'crypto-prices',
        bootstrap_servers=['localhost:9092'],
        value_deserializer=lambda x: json.loads(x.decode('utf
            -8')),
        auto_offset_reset='earliest',
        group_id='crypto-consumer-group'
    )
```

MongoDB est configuré avec une base de données dédiée (crypto_db) et une collection prices. Chaque document stocké contient un horodatage précis ainsi que les données et métriques relatives aux différentes cryptomonnaies.

Un mécanisme de backfill permet aussi de récupérer l'historique des 24 dernières heures depuis l'API CoinMarketCap, afin de combler les éventuelles lacunes en données historiques.

```
for coin in coins:
    url = f"https://pro-api.coinmarketcap.com/v1/coins/{coin
       }/market_chart/range"
    params = {
        'vs_currency': 'usd',
        'from': int(start_time.timestamp()),
        'to': int(end_time.timestamp())
    }
    while True:
        response = requests.get(url, params=params)
        if response.status_code != 429:
            break
        print(" Rate limited. Sleeping 10s...")
        time.sleep(10)
    response.raise_for_status()
    data = response.json()

    for ts_ms, price in data['prices']:
        ts = int(ts_ms // 1000)
        current_time = datetime.fromtimestamp(ts)
        if ts not in merged_data:
            merged_data[ts] = {"prices": {}, "timestamp":
               current_time}
        merged_data[ts]["prices"][coin] = {"usd": price}
```

Cette chaîne assure la fiabilité, la cohérence et la persistance des données dans un environnement distribué.

3.3 Synthèse

L'implémentation concrète à l'aide des librairies Python Kafka et pymongo montre comment Kafka peut servir de colonne vertébrale pour un pipeline de données cryptographiques temps réel. Le producteur garantit une collecte résiliente et structurée, tandis que le consommateur assure un stockage fiable et exploitable via MongoDB, prêt à alimenter la couche de visualisation.
Ce choix architectural permet d'équilibrer performance, scalabilité, et simplicité de maintenance dans un contexte où la réactivité et la précision des données sont critiques.

4 Application web

L'application web développée permet la visualisation en temps réel des prix de plusieurs cryptomonnaies, notamment le Bitcoin, l'Ethereum et le Ripple. Elle combine une interface utilisateur réactive basée sur React et la bibliothèque Chart.js pour afficher un graphique en ligne dynamique représentant l'évolution des prix sur des plages temporelles sélectionnables, soit une heure ou vingt-quatre heures.

4.1 Frontend : interface et interactions utilisateur

Le frontend, développé avec React, offre une interface intuitive et responsive qui facilite la navigation et la sélection des paramètres par l'utilisateur. Les composants React sont conçus pour gérer efficacement les états liés aux données reçues, notamment la cryptomonnaie choisie et la période d'affichage.
Le graphique est rendu via Chart.js, qui permet une visualisation claire et dynamique des données en temps réel. Les contrôles de sélection, tels que les boutons pour choisir la cryptomonnaie ou la plage temporelle, déclenchent des requêtes vers le backend pour mettre à jour le graphique sans rechargement de la page, assurant une expérience fluide.
De plus, un panneau latéral affiche en temps réel les informations clés comme le prix actuel, les variations de prix, les valeurs minimales et maximales, ainsi que le volume échangé sur la période sélectionnée, permettant à l'utilisateur d'avoir une vue complète en un coup d'œil.

FIGURE 2 – Interface utilisateur affichant le graphique des prix en temps réel

4.2 Architecture technique et communication

Le backend repose sur un serveur WebSocket Node.js qui établit une communication persistante et bidirectionnelle entre la base de données MongoDB et le frontend. Lors de la connexion d'un client, le serveur interroge MongoDB pour récupérer les données historiques correspondant à la

plage temporelle demandée et les transmet au client. Il écoute également les requêtes du client visant à modifier cette plage et adapte dynamiquement l'envoi des données.

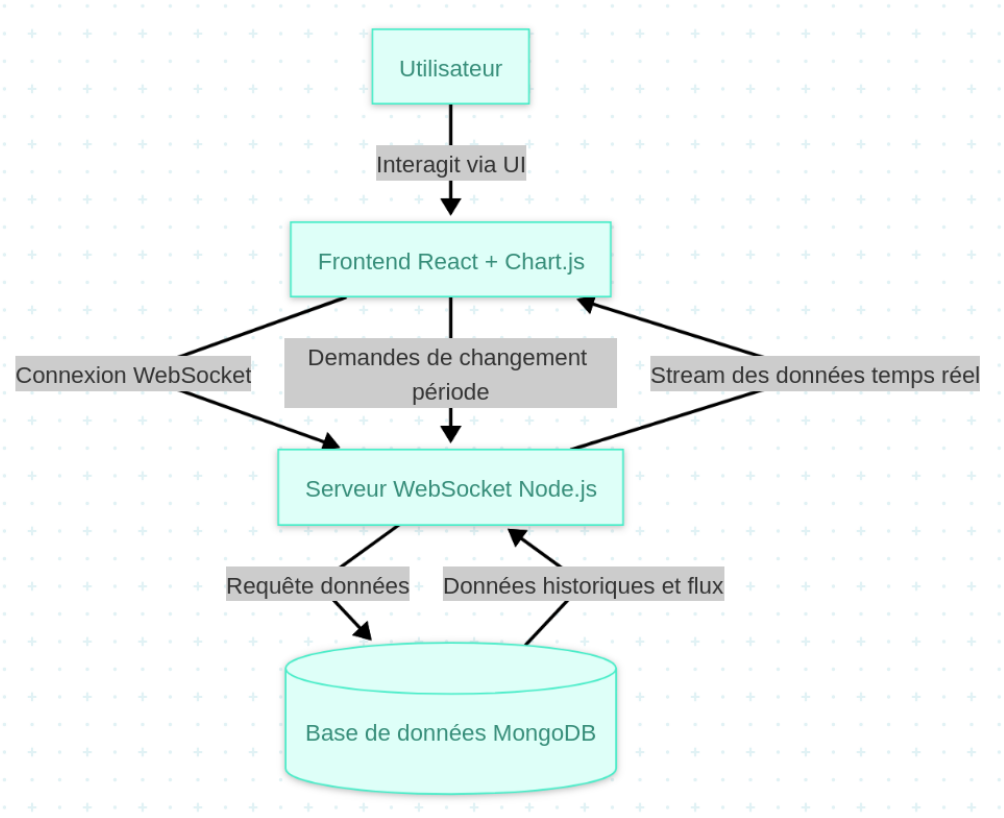

FIGURE 3 – Schéma de l'architecture : communication entre WebSocket, MongoDB et frontend

Cette architecture garantit une faible latence dans la transmission des données, en combinant l'efficacité des requêtes MongoDB avec la réactivité offerte par WebSocket. Le frontend filtre et trie les données reçues afin d'afficher uniquement les points pertinents pour la période active, assurant ainsi une performance optimale et une clarté visuelle.

5 Conclusion

L'application web développée illustre une intégration efficace entre une interface utilisateur réactive et une architecture backend robuste basée sur WebSocket et MongoDB. Cette solution permet d'afficher des données financières en temps réel, offrant ainsi une expérience utilisateur fluide et pertinente. La modularité de l'architecture facilite l'évolution future, notamment l'ajout de nouvelles cryptomonnaies ou l'amélioration des indicateurs financiers. Ce projet met en lumière l'importance d'une communication bidirectionnelle optimisée entre client et serveur pour les applications critiques où la latence et la mise à jour instantanée sont des exigences essentielles. En somme, ce travail ouvre la voie à des systèmes plus complexes et scalables, adaptés aux besoins actuels du marché financier numérique.