PROJET BIG DATA

Présentation du système NoSQL COUCHBASE

Université Paris Cité

Ilyes ABBASSI
Said HARBANE
Yanis SEGHDAU

Master MIAGE

2024 — 2025

Table des matières

1. Introduction

Ce projet a été réalisé dans le cadre de notre Master MIAGE. L'objectif principal était de découvrir et comprendre le fonctionnement d'un système de base de données NoSQL. Nous avons choisi Couchbase, une solution orientée documents, connue pour sa performance, sa scalabilité et sa souplesse. Concrètement, notre travail s'est organisé autour de plusieurs axes :

- D'abord, une présentation théorique de Couchbase et de son architecture.
- Ensuite, la mise en place d'un cluster Couchbase à l'aide de Docker.
- Puis, l'importation d'un jeu de données réel (données FIFA depuis Kaggle) dans le système.
- Et enfin, une démonstration de la réplication inter-cluster (XDCR) entre deux instances Couchbase.

Ce rapport suit une structure logique. On commence par l'architecture générale de Couchbase, suivie de la présentation du dataset et des manipulations réalisées pour l'importer. Ensuite, on développe la partie XDCR avec une démonstration concrète. On termine par une conclusion qui revient sur les apprentissages du projet.

2. Architecture Générale

2.1. Structure et Composants Principaux

Couchbase Server adopte une architecture distribuée de type partagé-nul, où chaque nœud est indépendant. Les principaux composants, détaillés dans la documentation officielle (Architecture Overview), incluent :

- **Data Service** : Gère le stockage et la récupération des données par clé.
- **Query Service** : Exécute des requêtes SQL++ pour interroger les données.
- **Index Service** : Crée et maintient des index pour accélérer les recherches.
- **Search Service** : Offre des capacités de recherche textuelle, par exemple, cherchant « beauties » inclut « beauty » et « beautiful ».
- **Analytics Service** : Supporte des opérations analytiques intensives, comme les jointures et agrégations.
- **Eventing Service** : Gère les changements de données en temps réel, exécutant du code sur des mutations de documents ou des temporisations.
- **Backup Service** : Planifie et exécute des sauvegardes complètes ou incrémentales pour les buckets.

Chaque nœud peut exécuter un ou plusieurs services, permettant une escalade multidimensionnelle. Par exemple, un cluster de développement peut avoir un nœud par service, tandis qu'un environnement de production peut ajuster pour des charges plus lourdes sur Data et Index.

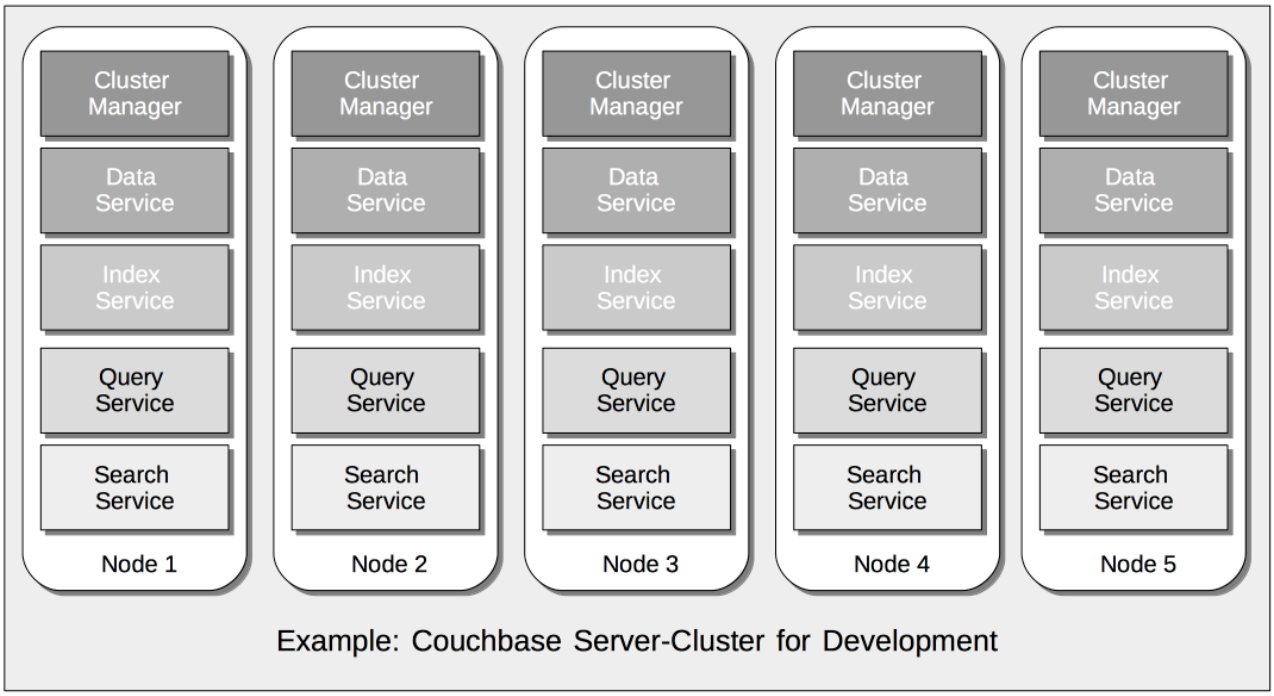

Fig. 1. – Exemple de cluster Couchbase pour le développement Couchbase Server-Cluster for Development.

Cette figure montre un cluster de 5 nœuds, chacun exécutant tous les services (Cluster Manager, Data, Index, Query, Search), typique d'un environnement de développement où la séparation des services n'est pas critique.

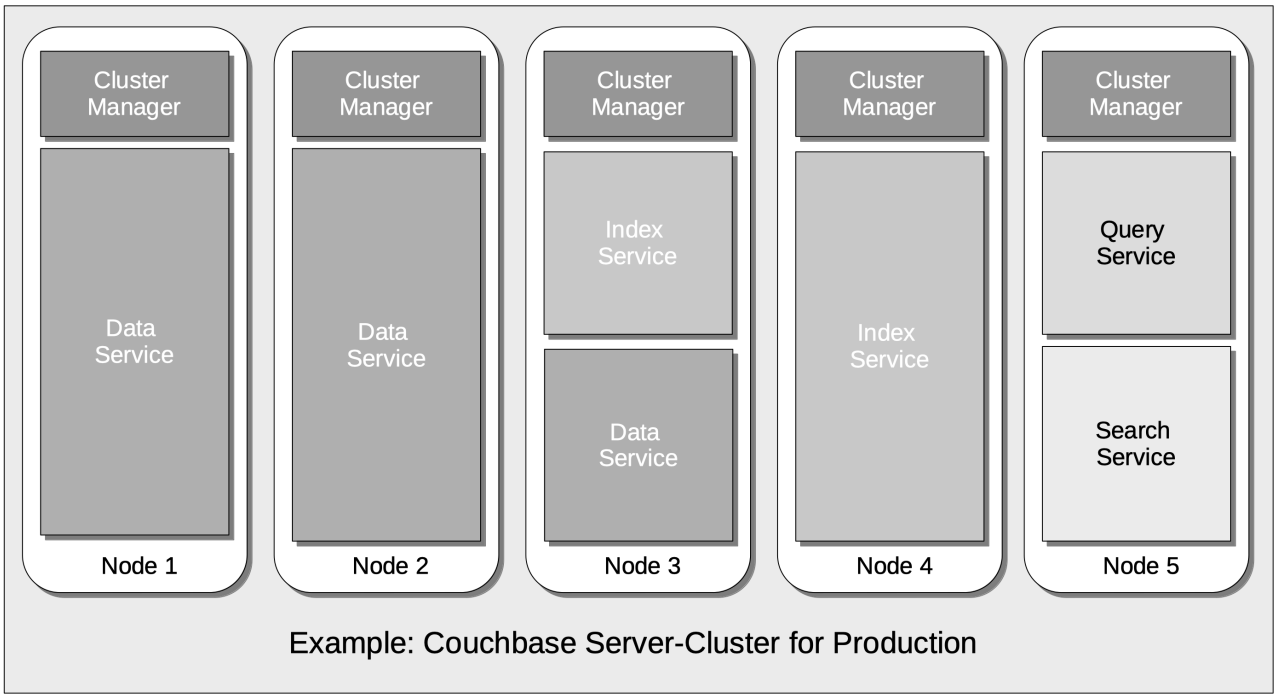

Fig. 2. – Exemple de cluster Couchbase pour la production Couchbase Server-Cluster for Production.

Dans un environnement de production, les services sont répartis pour optimiser les performances. Par exemple, le nœud 3 est dédié à l'Index Service, et le nœud 4 au Query Service, réduisant les interférences entre les charges de travail.

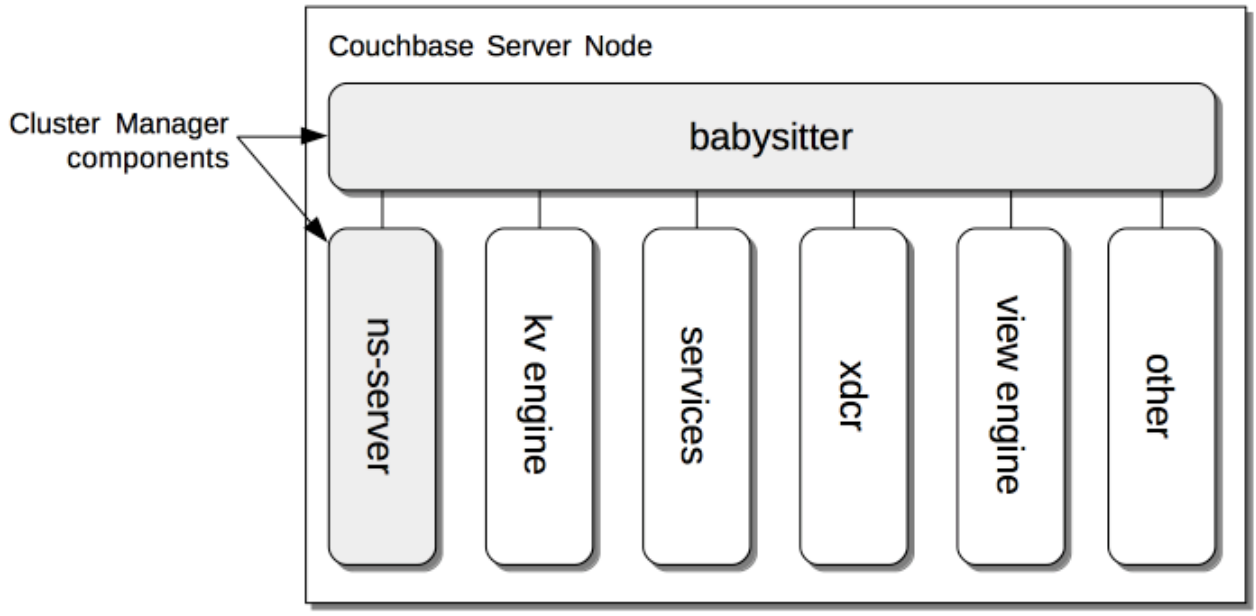

Fig. 3. – Composants d'un nœud Couchbase Couchbase Server Node

Cette figure illustre les composants internes d'un nœud, incluant le Cluster Manager (babysitter), le KV Engine pour les opérations clé-valeur, le View Engine pour les vues, et d'autres services comme XDCR pour la réplication.

2.2. Fonctionnement Global du Système

Le fonctionnement repose sur un cluster de nœuds, coordonné par un Cluster Manager, offrant une interface unifiée aux clients via des SDK, CLI, REST APIs, et une console web (Couchbase Server Introduction). Les données sont partitionnées en 1024 vBuckets par bucket, répartis équitablement entre les nœuds Data Service. Chaque vBucket est répliqué via le Database Change Protocol sur d'autres nœuds, assurant la redondance. Les clients accèdent aux données par clé, avec routage automatique, et les requêtes SQL++ sont traitées par le Query Service, utilisant les index du Index Service.

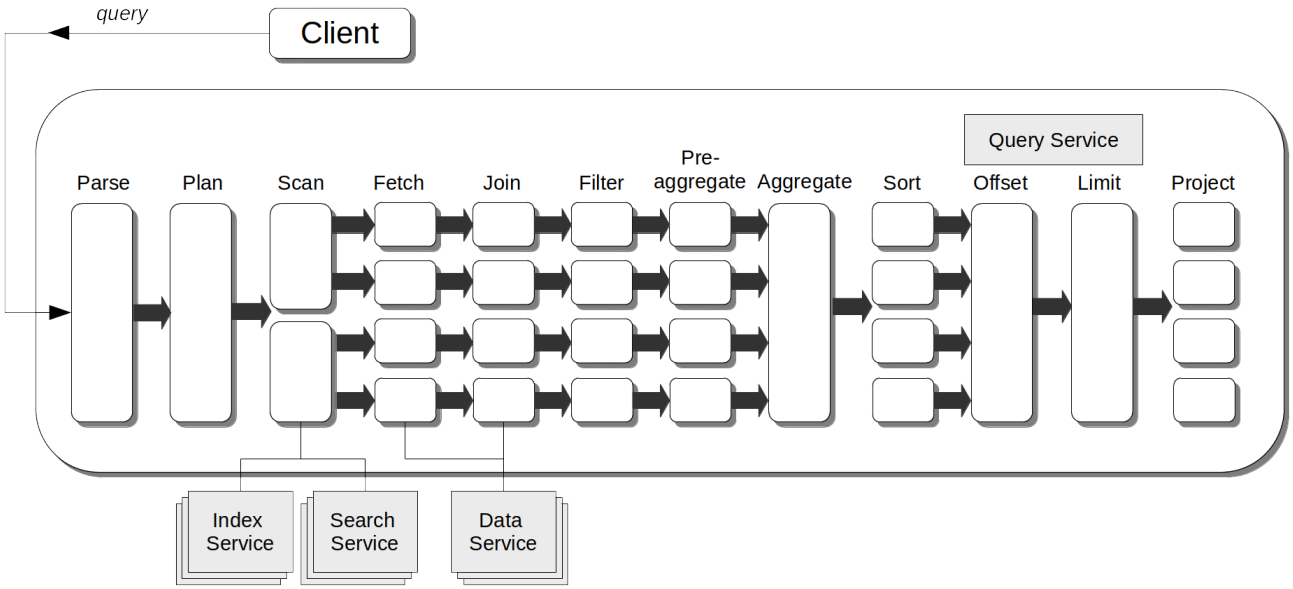

Fig. 4. – Processus de traitement des requêtes dans Couchbase Query Processing

Cette figure montre le flux de traitement d'une requête SQL++ : de la phase de parsing à la projection finale, en passant par le scan des index, le fetch des données, et les opérations comme le filtrage et l'agrégation. Le Query Service coordonne avec l'Index, Search, et Data Services pour exécuter la requête efficacement.

2.3. Modèle de Données

Couchbase est une base de données orientée document, stockant les données comme des documents JSON, chacun avec une clé unique (Data in Couchbase Server). Cela permet une flexibilité de schéma, où les documents peuvent avoir des structures variées, adaptées aux besoins des applications modernes. Les valeurs peuvent aussi être binaires, mais l'accent est mis sur les documents JSON pour leur versatilité.

2.4. Modèle de Cohérence des Données

Couchbase utilise principalement une cohérence éventuelle, où les mises à jour se propagent aux réplicas avec un léger délai, mais il offre des options pour une cohérence plus forte. Pour les accès documentaires (key-value), les écritures sont fortement cohérentes au sein d'un seul vBucket, avec des lectures immédiates des dernières valeurs (Performance and Consistency). Pour les requêtes, des niveaux comme `not_bounded` (cohérence éventuelle) ou `at_plus` (cohérence bornée) peuvent être spécifiés, équilibrant performance et exactitude. Les transactions ACID, introduites récemment, permettent une cohérence forte pour des opérations multi-documents, avec des options de durabilité ajustables (Data Consistency Models).

2.5. Types de Charges Supportées

Couchbase est optimisé pour des charges nécessitant des accès rapides à des documents individuels ou des ensembles, avec des capacités de requête SQL++ pour des analyses complexes (Enterprise Database Server). Il supporte des applications à haut débit, comme les applications web nécessitant des réponses en sous-milliseconde, et des charges IoT avec synchronisation edge-cloud. Il n'est pas conçu pour des charges SQL traditionnelles nécessitant des jointures complexes sur des tables relationnelles, mais il excelle dans les scénarios nécessitant une faible latence et une haute disponibilité.

2.6. Durabilité et Persistance des Données

La durabilité est assurée par des buckets Couchbase, stockés à la fois en mémoire et sur disque, avec réplication pour protéger contre les pannes de nœud (Buckets, Memory, and Storage). Les buckets éphémères, en revanche, existent uniquement en mémoire, sans persistance. La compression peut être configurée, et un TTL (Time To Live) peut être défini pour expirer les documents, rendant les données inaccessibles après un délai. La réplication croisée des centres de données (XDCR) offre une sauvegarde ou un accès multi-géo.

2.7. Capacités de Montée en Charge

Couchbase supporte une escalade horizontale en ajoutant des nœuds au cluster sans interruption, avec rééquilibrage automatique des données (Clusters and Availability). Chaque nœud peut être ajouté ou retiré dynamiquement, avec une amélioration linéaire des performances et de la capacité. L'escalade multidimensionnelle permet d'ajuster indépendamment les services, par exemple, augmenter les nœuds Data Service pour des ensembles de données plus grands, ou Query Service pour des charges de travail intensives.

2.8. Distribution de la Charge

La charge est distribuée via les vBuckets, avec 1024 vBuckets par bucket répartis équitablement entre les nœuds Data Service. Lors de l'ajout ou de la suppression de nœuds, les vBuckets sont rééquilibrés pour maintenir une distribution uniforme, assurant une charge équilibrée et une utilisation optimale des ressources (Cluster Management).

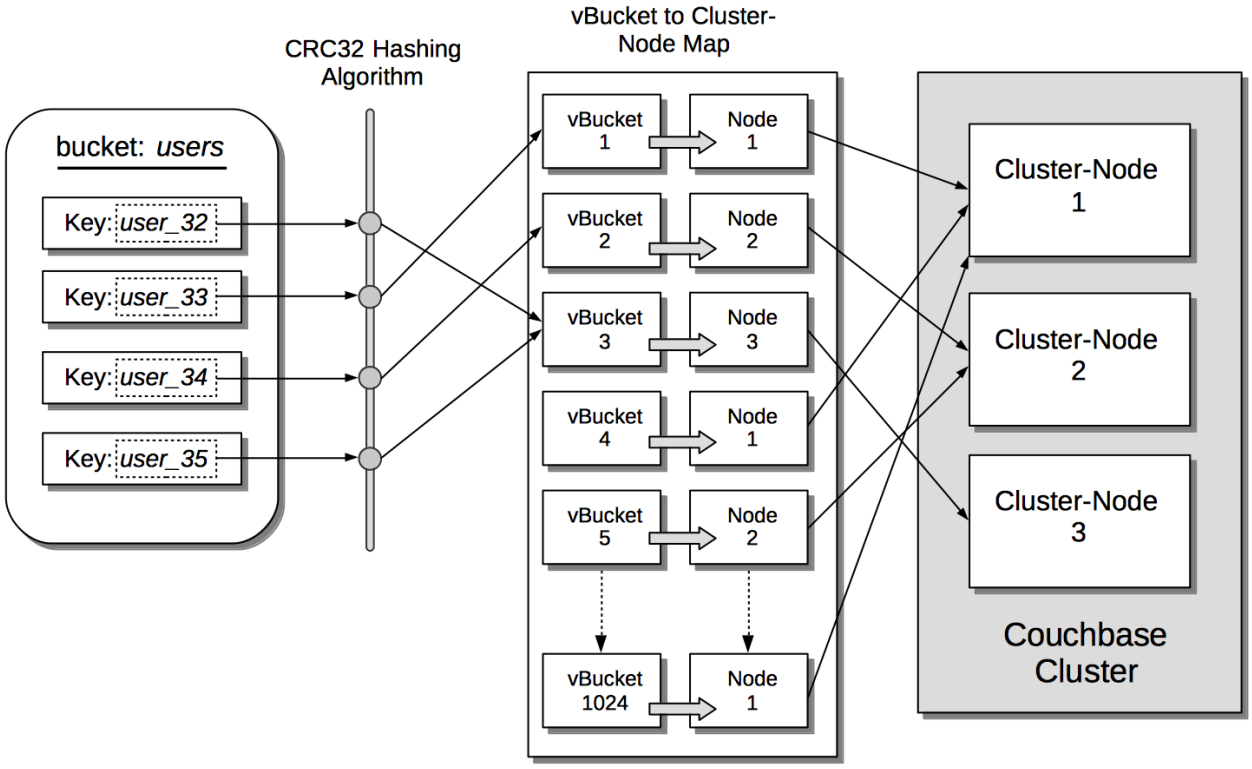

Fig. 5. – Mappage des vBuckets aux nœuds dans un cluster Couchbase vBucket to Cluster-Node Map

Cette figure illustre comment les clés d'un bucket sont mappées à des vBuckets via un algorithme de hachage CRC32, puis assignées à des nœuds dans le cluster. Par exemple, vBucket 1 est mappé au Cluster-Node 1, et vBucket 2 au Cluster-Node 2, assurant une distribution équilibrée.

2.9. Sécurité

Couchbase inclut des fonctionnalités de sécurité robustes, telles que :

- **Authentification :** Tous les administrateurs, utilisateurs, et applications doivent s'authentifier, supportant des registres locaux ou externes, avec SCRAM et TLS (Authentication).
- **Autorisation :** Utilise le contrôle d'accès basé sur les rôles (RBAC) pour associer les utilisateurs à des rôles avec privilèges définis (Authorization).
- **Audit :** Permet de suivre les actions pour assurer une gestion appropriée du système (Auditing).
- **Chiffrement :** Assure la confidentialité et l'intégrité des données, avec des options pour le stockage et la transmission sécurisés (Encryption).

Ces fonctionnalités garantissent la conformité et la protection des données, essentielles pour les déploiements d'entreprise. Couchbase est une solution NoSQL versatile, adaptée aux besoins des applications modernes nécessitant une faible latence, une haute disponibilité, et une évolutivité flexible. Sa capacité à gérer des charges variées, combinée à des options de cohérence ajustables et à des fonctionnalités de sécurité robustes, en fait un choix.

3. Présentation du jeu de données

Le dataset que nous avons choisi d'étudier fournit des informations détaillées sur environ 17 000 joueurs de football présents dans le jeu FIFA. Ces données ont été extraites du site SoFIFA.com et sont accessibles à l'adresse suivante : https://www.kaggle.com/datasets/maso0dahmed/football-players-data/data. Ce dataset est au format csv, il regroupe un large éventail d'informations sur les joueurs, incluant notamment leurs noms, nationalités, clubs, classements, potentiels, positions, âges et une variété d'attributs liés à leurs compétences. Il s'adresse aux amateurs de football, aux analystes de données et aux chercheurs qui souhaitent mener des analyses approfondies, des études statistiques ou des projets en apprentissage automatique portant sur les performances, les caractéristiques ou l'évolution de carrière des joueurs.

3.1. Procédure de collecte et d'import des données dans le système

Nous allons maintenant voir comment importer notre dataset dans couchbase via docker. Les prérequis sont d'avoir docker installé sur la machine et de posséder le dataset.

3.1.1. Etape 1

La première étape est de lancer la commande suivante afin de démarrer le cluster couchbase :

```
docker run -d --name couchbase-server -p 8091-8096:8091-8096 -p 11210:11210
couchbase
```

3.1.2. Etape 2

Nous avons maintenant lancé le serveur, nous pouvons à présent mettre en place un cluster et créer un bucket où insérer nos données.

```
docker exec -it couchbase-server couchbase-cli cluster-init --cluster-username
admin --cluster-password password --services data,index,query --cluster-ramsize
1024 --cluster-index-ramsize 256
```

```
docker exec -it couchbase-server couchbase-cli bucket-create --cluster
http://127.0.0.1:8091 --username admin --password password --bucket fifa_players
--bucket-type couchbase --bucket-ramsize 256
```

Ces deux commandes utilisent le cli fourni par couchbase afin de communiquer avec le serveur, la documentation complète est disponible en ligne, nous l'utiliserons fréquemment pour faire des manipulations : https://docs.couchbase.com/server/current/cli/cbcli/couchbase-cli.html

3.1.3. Etape 3

Afin d'insérer les données, nous devons préalablement les importer dans le conteneur, pour celà il faut utiliser la commande suivante :

```
docker cp fifa_players.csv couchbase server:/tmp/fifa_players.csv
```

Puis nous pouvons réutiliser le cli couchbase pour insérer le fichier dans notre bucket :

```
docker exec -it couchbase-server cbimport csv --cluster couchbase://127.0.0.1
--username admin --password password --bucket my_bucket --dataset file:///tmp/
fichier.csv --infer-types --generate-key %#UUID#%
```

3.1.4. Etape 3

Les données seront stockées dans notre bucket au format json, à savoir qu'une clé unique est généré pour chaque donnée insérée. Pour vérifier le bon déroulement de notre insertion, nous pouvons interroger le bucket via une requête N1QL, nous lancerons le mode requête avec le cli puis nous lancerons notre requête :

```
docker exec -it couchbase-server cbq --engine=http://127.0.0.1:8091 --u admin --
p password
```

```
SELECT `playersfifa_bucket`.`name`,
       `playersfifa_bucket`.`overall_rating`
FROM `playersfifa_bucket`
ORDER BY `playersfifa_bucket`.`overall_rating` DESC
LIMIT 10;
```

Cette requête récupère les 10 meilleurs joueurs selon leur note générale. Voici les résultats renvoyés :


```
{
    "requestID": "1a1e4e59-4377-4da3-90fa-c563ef7dfb74",
    "signature": {
        "name": "json",
        "overall_rating": "json"
    },
    "results": [
    {
        "name": "Cristiano Ronaldo",
        "overall_rating": 94
    },
    {
        "name": "L. Messi",
        "overall_rating": 94
    },
    {
        "name": "Neymar Jr",
        "overall_rating": 92
    },
    {
        "name": "E. Hazard",
        "overall_rating": 91
```

```
    },
    {
        "name": "De Gea",
        "overall_rating": 91
    },
    {
        "name": "L. Modrić",
        "overall_rating": 91
    },
    {
        "name": "K. De Bruyne",
        "overall_rating": 91
    },
    {
        "name": "L. Suárez",
        "overall_rating": 91
    },
    {
        "name": "Sergio Ramos",
        "overall_rating": 90
    },
    {
        "name": "G. Chiellini",
        "overall_rating": 90
    }
    ],
    "status": "success",
    "metrics": {
        "elapsedTime": "747.024975ms",
        "executionTime": "746.474933ms",
        "resultCount": 10,
        "resultSize": 665,
        "serviceLoad": 2,
        "sortCount": 17954
    }
}
```

4. La réplication inter-cluster (XDCR)

La réplication inter-cluster, ou Cross Data Center Replication (XDCR), est une fonctionnalité avancée de Couchbase permettant la synchronisation des données entre plusieurs clusters, souvent déployés dans des centres de données géographiquement distincts. Cette capacité est essentielle pour assurer la haute disponibilité, la redondance géographique, et l'optimisation des performances dans des environnements distribués à grande échelle.

4.1. Fonctionnement de XDCR

XDCR opère en réplicant les données d'un bucket source vers un bucket cible dans un autre cluster. Le processus est initié par la configuration d'une réplication entre les clusters, où le cluster source détecte les modifications (insertions, mises à jour, suppressions) et les transmet au cluster cible via une connexion sécurisée. La réplication est effectuée au niveau des vBuckets, les unités de partitionnement des données dans Couchbase, permettant un traitement parallèle et efficace. Chaque vBucket est mappé à un nœud spécifique, comme illustré dans la figure 5, et les modifications sont propagées indépendamment pour chaque vBucket, optimisant ainsi les performances. Le flux de réplication peut être décomposé comme suit :

- **Détection des Modifications :** Les changements dans le bucket source sont capturés via le Database Change Protocol (DCP), un mécanisme interne de Couchbase.
- **Transmission des Données :** Les modifications sont envoyées au cluster cible à travers des threads de travail dédiés à XDCR, qui opèrent en parallèle pour chaque vBucket.
- **Application des Changements :** Le cluster cible applique les modifications reçues, mettant à jour ses propres copies des données.

4.2. Particularités de XDCR

XDCR se distingue par plusieurs caractéristiques techniques qui en font une solution robuste pour la réplication distribuée :

- **Modes de Réplication :** XDCR supporte la réplication unidirectionnelle, où les données sont copiées d'un cluster source vers un cluster cible, et la réplication bidirectionnelle, où les deux clusters synchronisent mutuellement leurs données. La réplication bidirectionnelle est particulièrement utile pour les applications nécessitant des mises à jour simultanées dans plusieurs régions.
- **Résolution de Conflits :** En cas de modifications concurrentes sur le même document dans différents clusters, XDCR utilise une politique de résolution de conflits. Par défaut, il privilégie la version avec le numéro de séquence le plus élevé ou l'horodatage le plus récent. Cette politique peut être personnalisée pour répondre à des besoins spécifiques, par exemple en priorisant un cluster particulier.
- **Gestion des Interruptions :** XDCR implémente des points de contrôle pour assurer la résilience. En cas d'interruption (panne réseau ou basculement de cluster), la réplication reprend à partir du dernier point de contrôle, minimisant les pertes de données et évitant une resynchronisation complète.

- **Optimisation des Performances :** XDCR est conçu pour minimiser l'impact sur les performances. Il peut compresser les données pendant la transmission (à partir de Couchbase Server 5.5), réduisant l'utilisation de la bande passante. Cependant, XDCR consomme des ressources supplémentaires, notamment en termes de CPU et de bande passante, nécessitant un dimensionnement adéquat des clusters pour éviter les goulots d'étranglement.
- **Filtrage Avancé :** XDCR permet de filtrer les données à répliquer grâce à des expressions régulières. Par exemple, il est possible de ne répliquer que les documents dont les clés commencent par un préfixe spécifique, comme « client_123 », optimisant ainsi l'utilisation des ressources réseau.

Exemple d'Utilisation : Dans une application de réservation de vols, un cluster à New York et un autre à Sydney peuvent utiliser XDCR pour synchroniser les données. Une réservation effectuée à New York est répliée vers Sydney en quelques secondes, permettant à un utilisateur à Sydney de voir la mise à jour en temps réel. En cas de panne du cluster de New York, le cluster de Sydney peut prendre le relais, assurant la continuité du service.

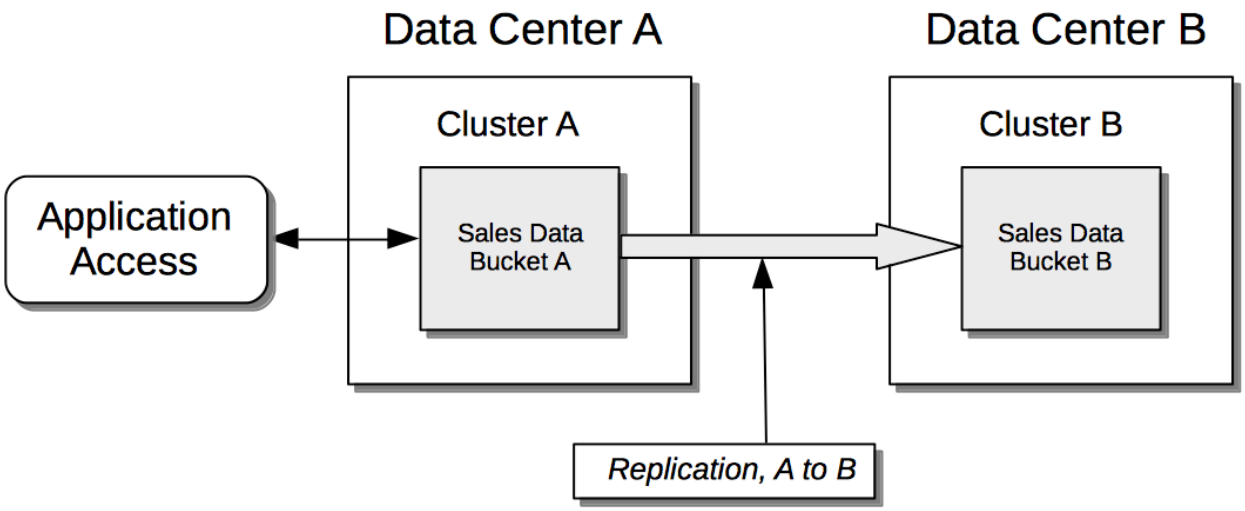

Fig. 6. – Schéma de la réplication inter-cluster unidirectionnelle

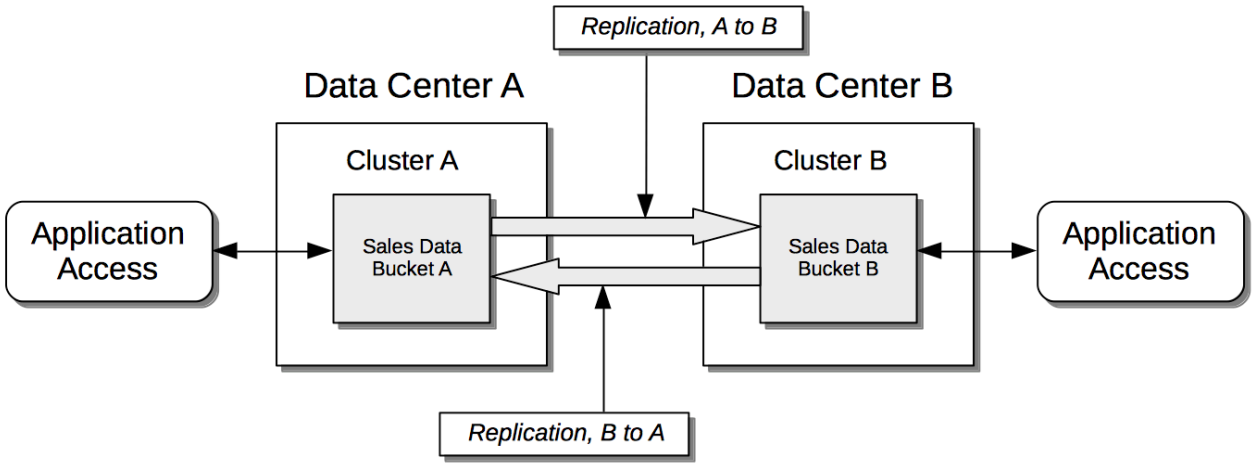

Fig. 7. – Schéma de la réplication inter-cluster bidirectionnelle

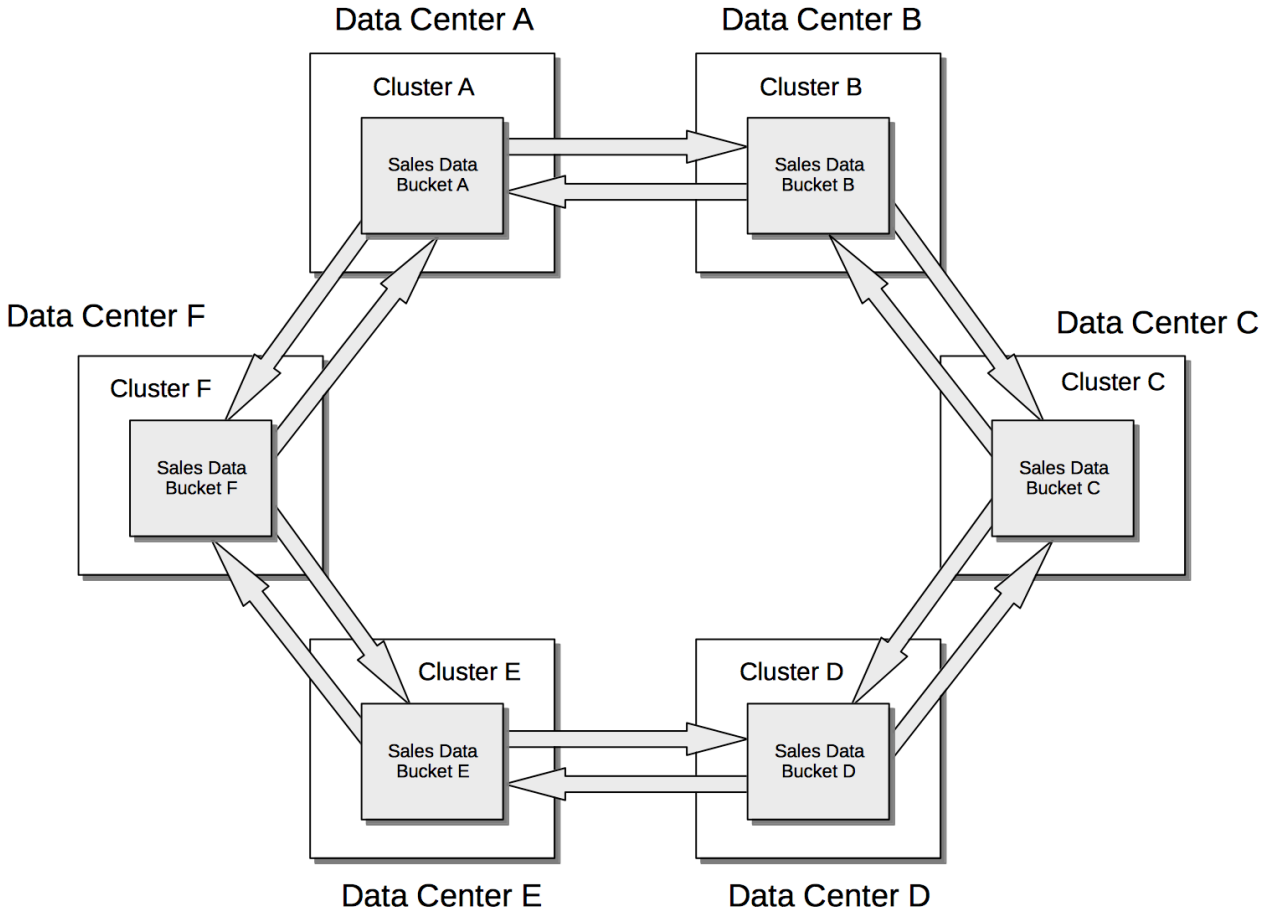

Fig. 8. – Schéma de la combinaison de clusters bidirectionnels

Considérations Techniques

- **Ressources :** XDCR peut nécessiter 1 à 2 cœurs CPU supplémentaires par flux de réplication, ainsi qu'une bande passante réseau suffisante. Un dimensionnement inadéquat peut entraîner des latences.
- **Surveillance :** Il est recommandé de surveiller les journaux XDCR pour détecter les erreurs, comme les documents non-UTF-8, qui sont automatiquement filtrés.
- **Sécurité :** XDCR utilise des connexions sécurisées (TLS) pour protéger les données en transit entre les clusters.

En conclusion, XDCR offre une solution robuste pour la réplication inter-cluster, garantissant la haute disponibilité et la redondance géographique. Ses capacités de filtrage, de résolution de conflits, et de gestion des interruptions en font un outil puissant pour les déploiements distribués (XDCR Overview).

4.3. Démonstration de la réplication XDCR avec Couchbase et Docker entre deux clusters

4.3.1. Pré-requis

- Docker
- Docker Compose
- Fichier CSV (`fifa_players.csv`)

4.3.2. Etape 1

Nous créons tout d'abord un réseau privé sur docker pour permettre à nos 2 clusters de communiquer entre eux.

```
docker network create couchbase-network
```

4.3.3. Etape 2

Nous avons rédigé ce fichier `docker-compose.yml`, il permet d'orchestrer le lancement de nos 2 serveurs couchbase sous le même network avec une configuration précise pour les ports etc. Nous pouvons lancer nos conteneurs simultanément via la commande :

```
docker-compose up -d
```

```
version: '3'
services:
  couchbase1:
    image: couchbase:latest
    container_name: couchbase1
    ports:
      - "8091:8091"
      - "11210:11210"
    networks:
      - couchbase-network
  couchbase2:
    image: couchbase:latest
    container_name: couchbase2
    ports:
      - "9191:8091"
      - "21210:11210"
    networks:
      - couchbase-network
networks:
  couchbase-network:
    driver: bridge
```

Nos conteneurs font maintenant tourner nos serveurs couchbase en arrière plan, ils sont à l'écoute sur les ports `8091` et `9091`.

4.3.4. Etape 3

Comme pour la démonstration de l'intégration de données, nous allons initialiser les clusters dans nos serveurs via la commande (il faut répéter l'opération pour couchbase 2):

```
docker exec -it couchbase1 couchbase-cli cluster-init -c http://127.0.0.1:8091
--cluster-username admin --cluster-password password --services data,index,query
--cluster-ramsize 512 --cluster-index-ramsize 256
```

4.3.5. Etape 4

Nous allons maintenant créer un bucket dans chacun de nos clusters via la commande :

```
docker exec -it couchbase1 couchbase-cli bucket-create -c http://127.0.0.1:8091
-u admin -p password --bucket playersfifa_bucket --bucket-type couchbase --bucket-
ramsize 128
```

4.3.6. Etape 5

Nous allons maintenant activer la réplication de nos buckets, afin de connecter couchbase2 à partir de couchbase1, il faut récupérer l'adresse IP de couchbase2 :

```
docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}'
couchbase2
```

Les commandes suivant vont servir dans un premier temps à créer un pont entre les 2 serveurs puis de lier les buckets à répliquer :

```
docker exec -it couchbase1 couchbase-cli xdcr-setup -c http://127.0.0.1:8091
-u admin -p password --create --xdcr-cluster-name cluster2 --xdcr-hostname
<IP_couchbase2>:8091 --xdcr-username admin --xdcr-password password
```

```
docker exec -it couchbase1 couchbase-cli xdcr-replicate -c http://127.0.0.1:8091
-u admin -p password --create --xdcr-cluster-name cluster2 --xdcr-from-bucket
playersfifa_bucket --xdcr-to-bucket playersfifa_bucket
```

Nous pouvons maintenant insérer les données dans l'un des buckets comme dans la première démonstration et constater la réplication en interrogeant les 2 clusters. A savoir que nous avons configuré une réplication bidirectionnelle.

5. Conclusion

Ce projet nous a permis de mettre en pratique des notions clés sur les bases NoSQL et les architectures distribuées. On a appris à déployer un cluster Couchbase avec Docker, à manipuler des buckets, à importer des données JSON, et à faire de la réplication entre deux instances.On a aussi mieux compris comment fonctionne Couchbase de l'intérieur : ses services, son modèle de données, sa gestion de la charge, et ses capacités de montée en charge. Enfin, on a vu concrètement comment gérer un dataset, écrire des requêtes avec SQL++, et mesurer les performances du système. Globalement, ce projet nous a permis d'approfondir nos connaissances sur les bases de données modernes et d'expérimenter des outils qu'on pourrait retrouver en entreprise.

6. Table des figure

7. Bibliographie

[1] A. Messina, « Architectural Overview of Couchbase Server », janv. 2018. [En ligne]. Disponible sur: https://intranet.icar.cnr.it/wp-content/uploads/2018/06/RT-ICAR-PA-2018-02.pdf

[2] R. Mayuram, « Ending the Relational vs NoSQL Debate, Once and for All ». [En ligne]. Disponible sur: https://www.couchbase.com/blog/ending-the-relational-vs-nosql-debate-once-and-for-all/

[3] A. Yudovin et C. Gutierrez, « NoSQL Comparison 2021: Couchbase Server, MongoDB, and Cassandra (DataStax) ». [En ligne]. Disponible sur: https://www.altoros.com/blog/nosql-comparison-2021-couchbase-server-mongodb-and-cassandra-datastax/

[4] Couchbase-Team, « Cross Data Center Replication (XDCR) ». 2025. [En ligne]. Disponible sur: https://docs.couchbase.com/server/current/learn/clusters-and-availability/xdcr-overview.html