**République Algérienne Démocratique et Populaire**

**Ministère de l’Enseignement Supérieur et de la Recherche Scientifique**

**École Supérieure en Science et**

**Technologie De**

**l’informatique et du**

**numérique**



# **Apache cassandra : Guide Complet du SGBD NoSQL Distribué**



**Membres :**

* Slimani Wahid
* Benali Amine
* Lallouche Abdelghani
* Koudia yacine

**Groupe**: 01 IADS

**Supervisé par :**

Pr. Sebaa Abderrazak  
Mme Haroune Asma

## Table des Matières

1. [Introduction](#introduction)
2. [Architecture et Fonctionnement](#architecture-et-fonctionnement)
3. [Caractéristiques Principales](#caracteristiques-principales)
4. [Avantages et Inconvénients](#avantages-et-inconvenients)
5. [Comparaison avec d’autres SGBD](#comparaison-avec-dautres-sgbd)
6. [Démonstration / Exemples Illustratifs](#demonstration)
7. [Conseils pour les Projets](#conseils)
8. [Cas d’Usage dans le Monde Réel](#cas-dusage)
9. [Conclusion et Perspectives](#conclusion-et-perspectives)

## 1. Introduction

Un **Système de Gestion de Base de Données (SGBD)** permet de stocker, organiser et manipuler des données. Apache Cassandra est un **SGBD NoSQL distribué** conçu pour gérer de vastes volumes de données sur plusieurs serveurs avec une haute disponibilité.

**Exemple** : - **Netflix** utilise Cassandra pour les données de visionnage de plus de 200 millions d’utilisateurs, avec jusqu’à **1 million d’écritures par seconde**.

* **Origine** : Développée par Facebook en 2008 pour améliorer la recherche dans les messages.
* **Open Source** : Devenue un projet Apache en 2009.
* **Utilisation** : Adoptée par des entreprises comme Apple, Uber, et Spotify.
* Cassandra a été créée pour surmonter les limites des bases de données relationnelles, en offrant une meilleure scalabilité et une haute disponibilité.

## 2. Architecture et Fonctionnement

### 2.1 Type de SGBD

Apache Cassandra est classé comme **Wide-Column Store**. Contrairement aux bases relationnelles qui ont des schémas fixes, Cassandra offre une **modélisation flexible** adaptée aux charges massives de lecture/écriture distribuées.

**Uber** : Utilise Cassandra pour stocker les données des trajets, où chaque colonne peut varier selon le contexte (ex: driver\_rating, surge\_pricing).

### 2.2 Langage CQL

Cassandra utilise le **Cassandra Query Language (CQL)**, similaire à SQL mais optimisé pour son architecture distribuée.

**Exemple de Requête** :



### 2.3 Approche Distribuée et Réplication

Cassandra adopte une approche décentralisée où les données sont réparties par un processus de hachage et répliquées à travers plusieurs nœuds pour assurer tolérance aux pannes.

Diagramme de l’architecture Cassandra

### 2.4 Modèle de Données et Cohérence

Cassandra utilise un **modèle de colonnes larges**, où chaque ligne peut contenir un nombre variable de colonnes. Les données sont distribuées sur plusieurs nœuds selon un **hashing de la clé primaire**, assurant un équilibrage efficace.

La **réplication** garantit la **tolérance aux pannes** en dupliquant les données sur plusieurs nœuds en fonction du **facteur de réplication** défini dans le keyspace. Par exemple, avec replication\_factor = 3, chaque donnée est stockée sur trois nœuds.

Un **keyspace** est un conteneur logique qui organise les tables, types, fonctions et vues matérialisées. Il définit également la stratégie de réplication ainsi que d’autres paramètres, comme le **TTL (Time-To-Live)**. Le TTL permet d’attribuer une durée de vie aux données, exprimée en secondes, après laquelle elles sont automatiquement supprimées sans intervention manuelle.

**Exemple :**



## 3. Caractéristiques Principales

* **Modèle de Cohérence** : BASE (Basically Available, Soft State, Eventual Consistency)
* **Tolérance aux Pannes** : Système décentralisé résistant à la défaillance de plusieurs nœuds.
* **Scalabilité** : Horizontale, facilitant l’ajout de nœuds selon la demande.

### Modèle de Cohérence : ACID vs BASE

Cassandra adopte le modèle **BASE** (Basically Available, Soft State, Eventual Consistency), contrairement aux bases relationnelles qui suivent **ACID** (Atomicité, Cohérence, Isolation, Durabilité).

🔹 **ACID** (utilisé par MySQL, PostgreSQL...) garantit :

* **Atomicité** : Une transaction est entièrement exécutée ou annulée.
* **Cohérence** : La base passe d’un état valide à un autre.
* **Isolation** : Une transaction en cours n'affecte pas les autres.
* **Durabilité** : Une transaction validée reste enregistrée, même après une panne.

**Exemple ACID** : Lors d’un virement bancaire, le débit du compte A et le crédit du compte B doivent être confirmés ensemble ou annulés.

🔹 **BASE** (adopté par Cassandra) favorise :

* **Basically Available** : Le système reste accessible même en cas de panne partielle.
* **Soft State** : L'état des données peut évoluer temporairement en raison de la réplication.
* **Eventual Consistency** : Les mises à jour se propagent progressivement et deviennent cohérentes après un certain délai.

**Exemple BASE** : Un post sur un réseau social peut mettre quelques secondes à apparaître partout.

Cassandra privilégie **BASE** pour garantir **haute disponibilité** et **scalabilité**, au détriment de la cohérence immédiate.

### Tolérance aux Pannes et Réplication

Cassandra est conçue pour continuer à fonctionner même en cas de défaillance d’un ou plusieurs nœuds.

**Exemple** :

* **Apple iCloud** : Les photos sont répliquées sur 3 nœuds géographiquement distants. Si un datacenter tombe en panne, les données restent accessibles.

### Scalabilité : Horizontale vs Verticale

🔹 **Scalabilité Horizontale** : Ajouter des **nœuds** au cluster pour répartir la charge (ex : Cassandra).  
🔹 **Scalabilité Verticale** : Augmenter les **ressources** (CPU, RAM) d’un serveur unique (ex : MySQL).

**Exemple** : Netflix ajoute des nœuds à son cluster Cassandra lors des pics de demande, comme la sortie d’une nouvelle série, pour assurer des performances optimales.

### Sécurité

* **Authentification** : Intégration avec LDAP ou Kerberos.
* **Chiffrement** : Données chiffrées en transit (SSL/TLS) et au repos (AES-256).
* **RBAC (Role-Based Access Control)** : Restreindre l’accès à des tables spécifiques.

**Exemple** :

* **Banque XYZ** : Utilise Cassandra pour stocker les historiques de transactions, avec un chiffrement AES-256 et un accès restreint aux auditeurs.

### Interaction avec Apache Cassandra via une API REST

L’intégration d’Apache Cassandra avec une API REST permet de manipuler des données distribuées de manière efficace et évolutive. Une connexion est établie avec un cluster Cassandra via un driver spécifique, permettant l’exécution de requêtes en CQL (Cassandra Query Language). L’API REST expose des endpoints pour insérer, récupérer, mettre à jour et supprimer des données, facilitant ainsi leur gestion via des requêtes HTTP standard. Grâce à son architecture distribuée et sans point de défaillance unique, Cassandra assure une haute disponibilité et une scalabilité horizontale, adaptées aux applications nécessitant un traitement rapide et fiable des données massives.

## 4. Avantages et Inconvénients

### Avantages

* **Scalabilité Illimitée** : Ajoutez des nœuds à la volée (ex: **eBay** pendant le Black Friday).
* **Haute Disponibilité** : Aucun point de défaillance unique.
* **Performance en Écriture** : Optimisée pour les opérations massives (ex: logs système).

**Exemple** :

* **Spotify** : Utilise Cassandra pour stocker les playlists et les historiques de lecture, avec une haute disponibilité et une performance optimale.

### Inconvénients

* **Pas de Jointures** : Impossible de lier deux tables directement. Solution : Stocker les données dénormalisées.
* **Complexité de Mise en Œuvre** : Requiert une expertise pour gérer les clusters distribués.
* **Latence en Lecture** : En raison de la cohérence éventuelle.

**Exemple** :

* Une startup a dû embaucher un ingénieur spécialisé en Cassandra pour optimiser son cluster après avoir rencontré des problèmes de latence.

## 5. Comparaison avec d’autres SGBD

|  | Wide-Column (Cassandra) | Documents (MongoDB) | Relationnel | Clé-Valeur |
| --- | --- | --- | --- | --- |
| Modèle de Données | Flexible | JSON | Structuré | Simple |
| Scalabilité | Horizontale | Horizontale | Variable | Horizontale |
| Cohérence | BASE | Variable | ACID | Éventuelle |
| Performance | Élevée (écriture) | Élevée (lecture) | Équilibrée | Élevée |
| Complexité | Modérée | Modérée | Élevée | Faible |

* **Cassandra** : Idéal pour les applications nécessitant des écritures massives (ex: logs système).
* **MongoDB** : Meilleur pour les requêtes complexes sur des documents JSON (ex: profils utilisateur).
* **CockroachDB** : Idéal pour les applications nécessitant des transactions ACID (ex: systèmes bancaires).
* **DynamoDB** : Idéal pour les applications serverless et les modèles clé-valeur simples (ex: paniers d’achat).

## 6. Démonstration / Exemples Illustratifs

### 6.1 Création d’un Keyspace et d’une Table



### 6.2 Insertion et Sélection de Données



### 6.3 Utilisation d’Index Secondaires



## 7. Conseils pour les Projets

* **Commencez Simple** : Utilisez Docker pour déployer un cluster Cassandra local.
* **Utilisez des Données Réelles** : Testez avec des jeux de données publics (ex. données météorologiques, tweets).
* **Évitez les Pièges Courants** : Choisissez des clés de partition équilibrées et configurez correctement la réplication.

### (a) Commencez Simple

* Utilisez **Docker** pour déployer un cluster Cassandra local. Cela vous permet de tester et de développer sans avoir besoin de matériel coûteux.
* **Exemple** :
* docker run --name cassandra -d cassandra:latest
* Cette commande lance un conteneur Docker avec Cassandra.

### (b) Utilisez des Données Réelles

* Testez Cassandra avec des jeux de données publics pour mieux comprendre ses performances.
* **Exemple de Jeux de Données** :
  + **Données Météo** : Stockez des données météorologiques historiques pour analyser les tendances.
  + **Tweets** : Utilisez l’API Twitter pour collecter et stocker des tweets en temps réel.

### (c) Évitez les Pièges Courants

* **Clés de Partition Mal Choisies** : Évitez de surcharger une partition en choisissant une clé de partition équilibrée. Par exemple, utilisez user\_id plutôt qu’une colonne avec peu de valeurs distinctes.
* **Réplication Mal Configurée** : Assurez-vous que le replication\_factor est adapté à votre cas d’usage. Pour un petit projet, replication\_factor=2 est souvent suffisant.

## 8. Cas d’Usage dans le Monde Réel

### Netflix : Stocke les données de visionnage en temps réel.

### Apple : Gère des données iCloud avec répartition sur de nombreux nœuds.

### Uber : Optimise le stockage des trajets en temps réel.

### Spotify : Gère les playlists et historiques de lecture pour des millions d’utilisateurs.

## 9. Conclusion et Perspectives

Apache Cassandra reste une solution incontournable pour des applications nécessitant scalabilité et haute disponibilité. Son adoption par des géants tels que Netflix et Apple témoigne de sa fiabilité.

### Perspectives

* **Modèles de Cohérence Améliorés** : Recherche de solutions hybrides combinant cohérence forte et haute disponibilité.
* **Intégration IA/ML** : Exploiter l’analyse en temps réel pour optimiser les performances.
* **Automatisation des Clusters** : Utilisation de Kubernetes et d’opérateurs dédiés pour simplifier la gestion.
* **Optimisation des Requêtes** : Amélioration par des index secondaires et optimisations analytiques.
* **Interopérabilité Cloud-Native** : Adaptation aux environnements multi-cloud pour une meilleure évolutivité.