**Compte-rendu Hbase**

# Table des matières

[Chapitre 1 Table des matières 1](#_Toc101526419)

[I. Introduction 3](#_Toc101526420)

[II. Modèle de données 3](#_Toc101526421)

[1. Lexique : 4](#_Toc101526422)

[2. Attention : 4](#_Toc101526423)

[III. Architecture et fonctionnement 5](#_Toc101526424)

[1. Composants de l’architecture Hbase 5](#_Toc101526425)

[a. RegionServers : 5](#_Toc101526426)

[b. HBase HMaster : 6](#_Toc101526427)

[c. Zookeeper : 6](#_Toc101526428)

[2. Fonctionnement de Hbase 7](#_Toc101526429)

[a. Split des régions 8](#_Toc101526430)

[b. Hot-Spotting 8](#_Toc101526431)

[c. Solution au Hot-spotting 9](#_Toc101526432)

[d. Compaction 9](#_Toc101526433)

[IV. Opérations de base dans Hbase 9](#_Toc101526434)

# 

# Introduction

* HBase est Base de données **NoSQL** **orientée colonne**
* Une base de données NoSQL (Not Only SQL) est non-relationnel et peut être
  + Orientée colonne comme **HBase** ou Cassandra
  + Orientée clé-valeur comme Aerospike, Redis et Riak
  + Orientée document comme MongoDB
  + Orientée graphe comme InfiniteGraph et Neo4j
* C’est une base de données **distribuées** au basée sur HDFS
* HBase est inspiré de BigTable de Google (GCP)
* HBase peut être assimilé à une base de données **clé-valeur** par le stockage des données dans les HFiles
* Fait partie de l’écosystème Hadoop et fonctionne dans un cluster
* **Hautement scalable** horizontalement
* HBase fournit **l’accès en temps réel** des données que ce soit en écriture ou en lecture
* HBase est un outil de gestion et de stockage de référentiel de données
  + Un **référentiel de données** désigne un ensemble de données isolées à exploiter pour la déclaration et l'analyse des données.
  + C’est un catalogue de données qui bouge beaucoup
  + Son Intérêt :
    - Use case : remonter en temps réel les informations des tickets de caisse dans la grande distribution
    - On ne peut pas remonter les données issues des tickets car trop volumineux
    - On remonte plutôt les identifiants des produits, clients, magasins
    - Par la suite, arrivées au niveau de la base de données, il y aura enrichissement du flux par les référentiels dans HBase via les identifiants
    - Pour faire des KPIs par exemple

# Modèle de données

* Sachant que HBase est similaire à une base de données clé-valeur, le modèle de données peut être décrite comme une une map (clé ROW KEY) + valeur : une map (clé COLUMN FAMILY) + valeur : une map (clé QUALIFIERS) + valeur : une map (clé : version -> timestamp) + valeur : la cellule contenant la ou les données non typées et sans schéma prédéfini
* La clé des cellules dans une table HBase est donc la combinaison de RK, CF, CQ, Timestamp

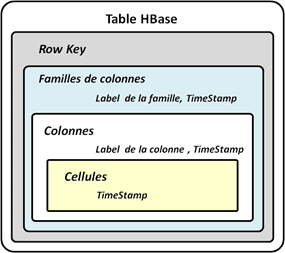


Figure 1: représentation d'une table hbase

## Lexique :

* + **Table** : on crée des tables et les données sont organisées en table dans HBase
  + **Row Key** (RK) : identifie les ligne d’une table
  + **Column Family** (CF) : groupement de colonnes en fonction du schéma de la table
  + **Column qualifiers** (CQ) : identifie les colonnes dans une CF
  + **Cell** ou cellule : c’est la donnée
  + **Version** : chaque donnée de même clé écrite est historisée par un Timestamp

## Attention :

* + Pour créer une table, seul le nom de la table et les CF doivent être définis
  + Il est déconseillé de créer beaucoup de CF (max 3) pour de raison de performance
  + Les versions de données sont par défaut limitées à 5
  + 1 use case par table

# Architecture et fonctionnement

## Composants de l’architecture Hbase

Physiquement, HBase est composé de trois types de serveurs de type Master/Slave.

### a. RegionServers :

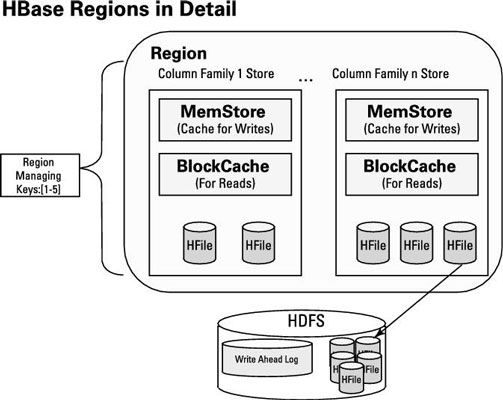
* Services qui s’opèrent au niveau des nœuds du cluster
* Permettent de fournir les données pour lectures et écritures.
* Pour accéder aux données, les clients communiquent avec les RegionServers directement.
* Dans chaque région serveur il y a plusieurs régions qui contiennent

Figure 2: Fonctionnement d'un region-serveur

* **MemStore :** 
  + C’est une mémoire cache des écritures (stocke les dernières requêtes de création et d’insertion de manière temporaire) pour avoir un Log
  + Lors du flush (vidage du cache) elles sont ensuite transférées sur HDFS sur disque de manière permanente dans le WAL (Write Ahead Log)
  + Pour pouvoir retracer l’état des tables
* **BlockCache :** 
  + C’est une mémoire cache des lectures (mémorise les dernières requêtes et résultat de lecture)
  + Pour gagner en rapidité de réponse. Souvent ce sont les mêmes requêtes qui se répètent (on dit que 80% des requêtes concernent les 20% de dernières données enregistrées)

### b. HBase HMaster :

* + Gère l'affectation des régions, les opérations de création et suppression de la table.
  + Les données dans les Hfiles sont stockées sous forme de Clé Valeur :
    - Clé : (row key, column family, column qualifier, timestamp)
    - Valeur : valeur de la cellule
    - Les données dans le HFile sont stockées par ordre croissant des clés (mais séquentiel car le Hfile ne peut pas être modifié)
  + Les clés des Hfiles d’une région appartiennent à une même plage
  + Attention :
    - Vue que les tables sont dans des Hfiles dans HDFS, en réalité il n’y a pas de suppression
    - Un marqueur est ajouté sur la clé de la ligne indiquant que celle-ci n’est plus pris en compte

### c. Zookeeper :

* Permet de maintenir le cluster en état.
* C’est un quorum (système de votants impaire à 3 ou plus) dans le cluster
* Chaque service tient un arbre éphémère de l’état des régions serveurs et du HMaster du cluster grâce aux Heart-Beat (signe de régulière envoyé comme un ping)

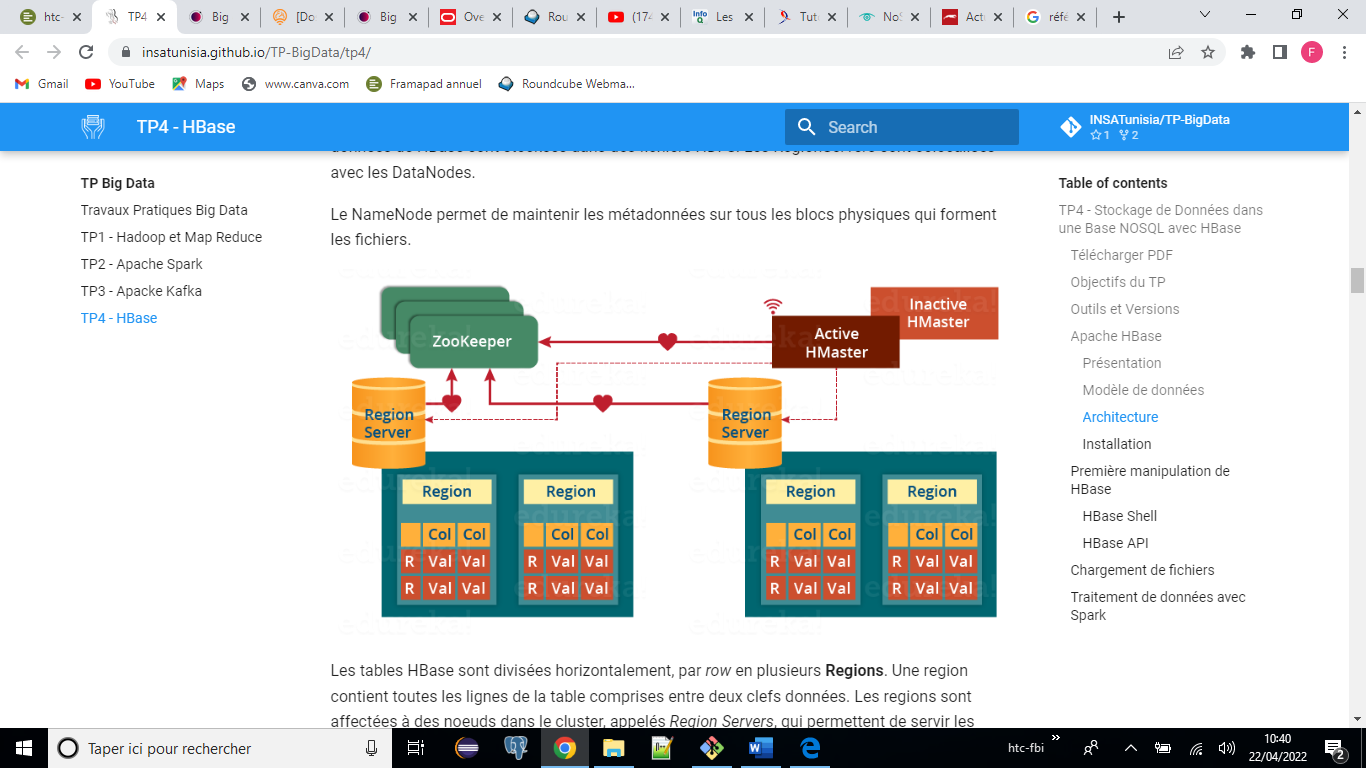


Figure 3: illustration du fonctionnement de zookeeper

## Fonctionnement de Hbase

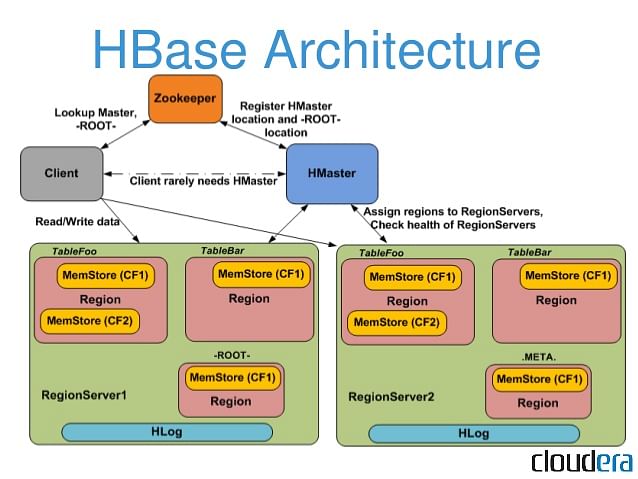


Figure 4: architecture hbase

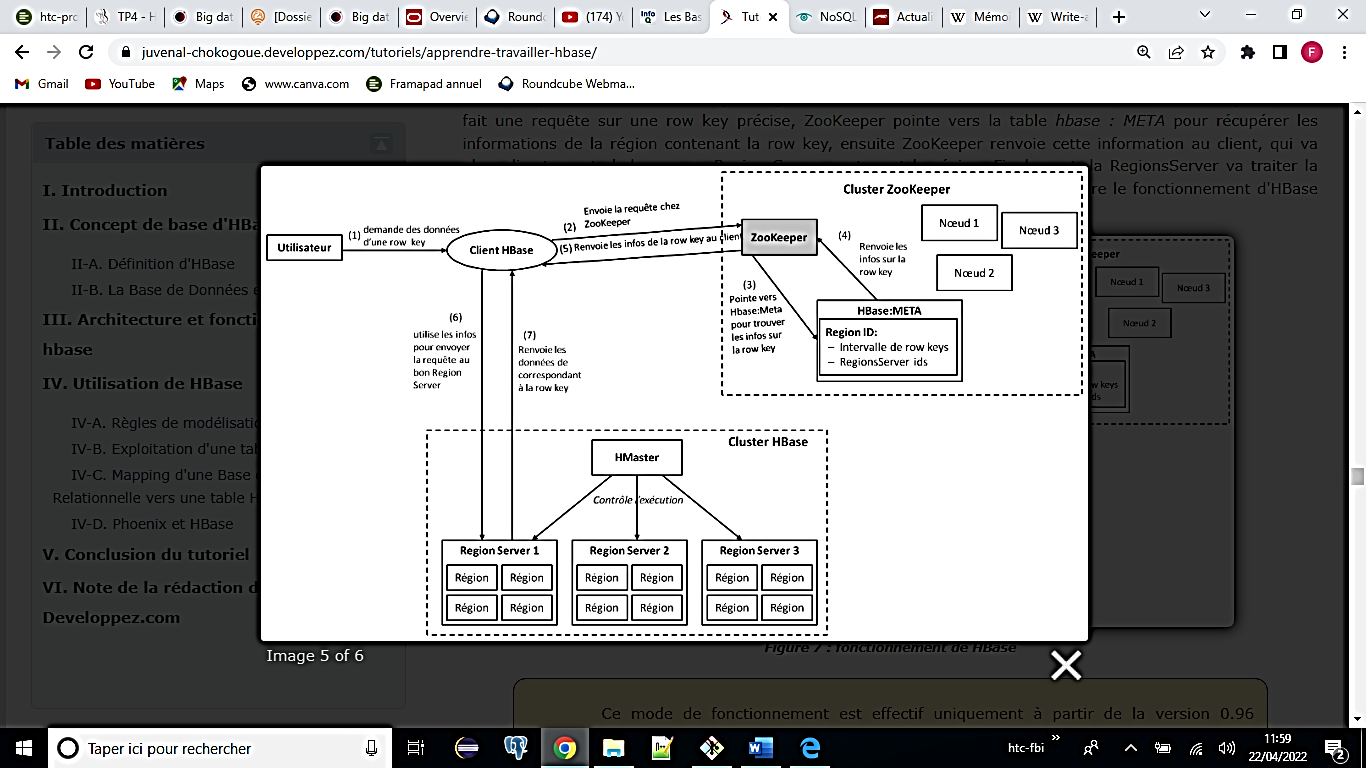


Figure 5: architecture de hbase commenté

### a. Split des régions

* Le région server divise une région dès que sa taille atteint un certain seuil
* La taille de split est configurable
* Chaque région fille s’occupe de la moitié de la plage des clés de la région mère
* Les régions filles sont ouvertes (pour écriture et lecture) simultanément sur le même server
* Des load-balancing (équilibrage des charges) de région d’un région server à un autre peut être déclenché par le HMaster

### b. Hot-Spotting

* Le Hot-Spotting arrive quand la distribution des clés n’est pas uniforme sur toutes les régions
* Conséquence d’une erreur de conception de la table comme une clé mal choisie du style auto-incrémenté ou timestamp)
* Les dernières données sont écrites dans la même région
* Le hot-spotting provoque des Splits trop intempestifs de régions qui va également bouffer de la ressource
* On sollicite les ressources d’un seul nœud et les autres ne font rien dans le cluster qui est contraire à l’esprit même d’un système distribué

### c. Solution au Hot-spotting

* Clé aléatoire (hachage avant envoie)
* Salting de la clé : Préfixe + clé aléatoire
  + Préfixe : pour regrouper les données de même « famille »
  + Clé-aléatoire : pour uniformiser
* Inverser les clés numériques, timestamp ou les noms de domaine

### d. Compaction

* Compaction mineure (compression des fichiers)
  + Combine plusieurs petits Hfiles pour créer un Hfile plus grand.
  + Ces Hfile garde les fichiers supprimés avec lui.
  + Pour augmenter l'espace en mémoire.
  + Ce processus utilise le tri par fusion
* Compaction majeure (réécriture des fichiers)
  + Les données présentes par famille de colonnes dans une région sont accumulées dans un Hfile.
  + Tous les fichiers supprimés ou les cellules expirées (les vieilles versions de plus de 5) sont supprimés définitivement, au cours de ce processus.
  + Augmenter les performances de lecture du fichier Hfile nouvellement créé.
  + Il accepte beaucoup d'I/O.
  + Possibilités de congestion du trafic (à programmer au moment des heures creuses car un processus lourd).
  + Peut-être décrit comme un processus d'amplification d'écriture.

# Opérations de base dans Hbase

* GET Récupérer une ligne
* PUT Insérer une donnée dans une ligne (ajout ou mise à jour)
* SCAN Récupérer une plage de clés
* DELETE Supprimer une ligne ou une plage de clés