



Big Data Master DSBD

Table des matières:

[C'est quoi Big data](#)
[Pour quoi Big data](#)
[Système d'information](#)
[Les Propriétés de Big data](#)
[Type des Données](#)
[Architecture Générale Big data](#)
[Les Systèmes Distribués](#)
[informatique distribuée & Traitement des Données](#)
[Big data Ecosystem](#)
[Hadoop \[HDFS/MapReduce/Yarn\]](#)
[HDFS commandes](#)
[Hive](#)
[Hive commandes](#)
[HBase](#)
[HBase commandes](#)
[Spark](#)
[Hadoop & Spark](#)
[Hive & HBase](#)

▼ C'est quoi Big data

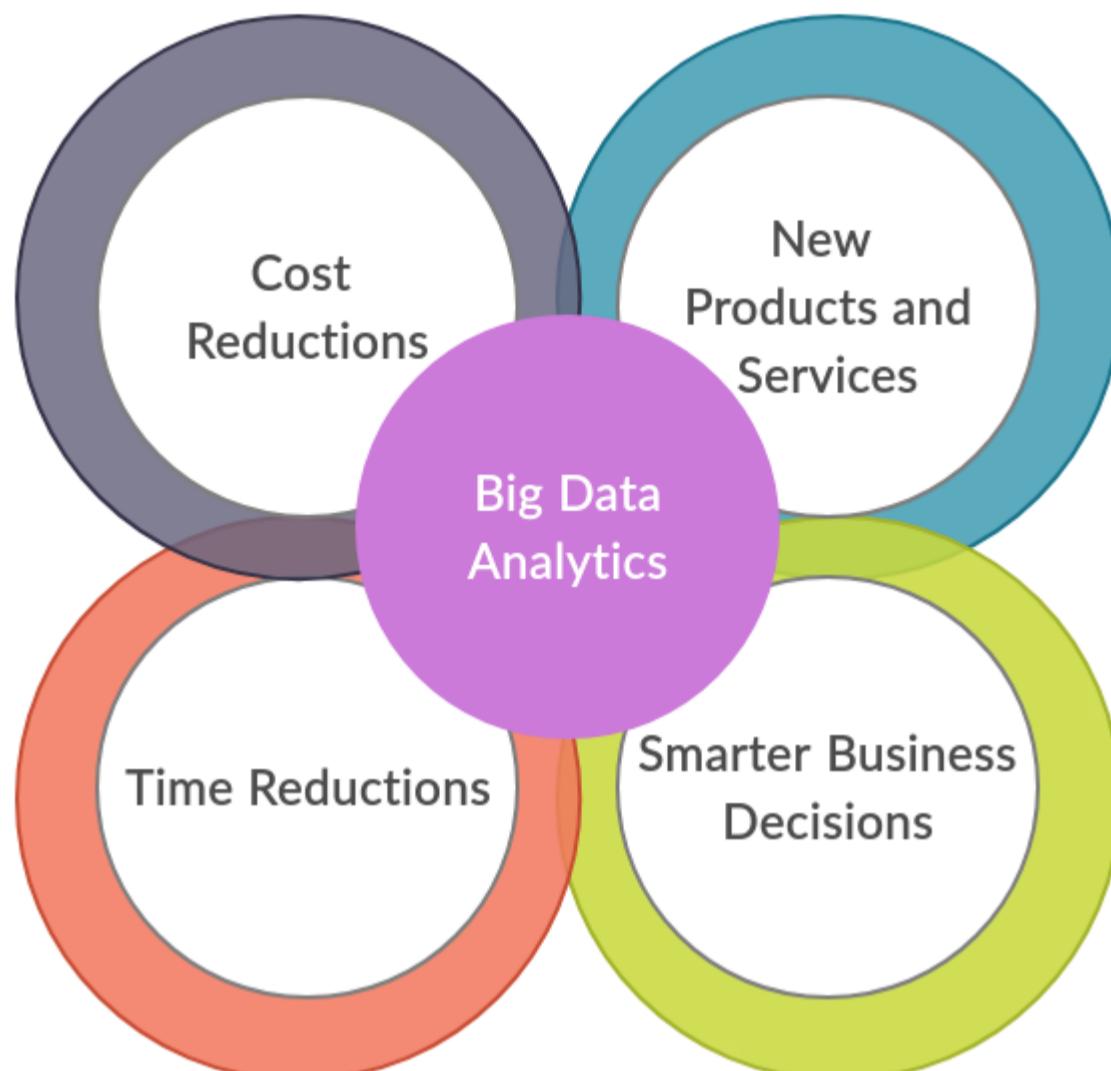
Le big data fait référence à la gestion et à l'analyse de vastes ensembles de données qui sont trop volumineux, complexes ou rapides pour être traités par des méthodes de traitement de données traditionnelles. Les caractéristiques clés du big data sont souvent résumées par les "3V" : Volume (le volume massif des données), Velocity (la vitesse à laquelle les données sont générées et traitées), et Variety (la diversité des types de données, qu'elles soient structurées, semi-structurées ou non structurées).

En résumé, le big data implique la manipulation de grandes quantités de données provenant de diverses sources, avec l'objectif d'en extraire des informations significatives pour prendre des décisions éclairées, découvrir des tendances et améliorer les performances.



▼ Pour quoi Big data

Le big data aide les entreprises à gérer des quantités massives de données, découvrant des tendances pour des décisions intelligentes. Cela favorise l'innovation en personnalisant les expériences clients, améliorant ainsi la compétitivité. En résumé, le big data transforme la façon dont les entreprises utilisent les données pour de meilleurs résultats.



Le big data est crucial pour plusieurs raisons, notamment :

1. **Gestion de volumes massifs de données** : Les organisations génèrent et accumulent dénormes quantités de données provenant de diverses sources telles que les médias sociaux, les capteurs, les transactions en ligne, etc. Le big data offre les outils nécessaires pour stocker, traiter et analyser ces volumes massifs de données.
2. **Identification de tendances et de modèles** : En analysant de vastes ensembles de données, le big data permet d'identifier des tendances, des modèles et des corrélations significatives. Ces informations peuvent être utilisées pour prendre des décisions éclairées, anticiper les changements du marché et améliorer les performances opérationnelles.
3. **Analyse en temps réel** : Dans de nombreux secteurs, la capacité à traiter les données en temps réel est cruciale. Le big data fournit des solutions pour analyser les flux de données en temps réel, ce qui est essentiel pour les applications telles que la surveillance de la santé, la détection de fraudes, et la prise de décisions en temps réel.
4. **Amélioration de la prise de décision** : En fournissant des informations exploitables et basées sur des données, le big data contribue à améliorer la prise de décision au sein des organisations. Les dirigeants peuvent utiliser ces informations pour ajuster leurs stratégies, identifier de nouvelles opportunités et résoudre des problèmes.
5. **Innovation et compétitivité** : Les entreprises qui tirent parti du big data sont souvent plus innovantes et compétitives. En utilisant l'analyse avancée, elles peuvent découvrir de nouvelles idées, améliorer leurs produits et services, et rester en phase avec les évolutions du marché.
6. **Personnalisation des expériences** : Le big data permet aux entreprises de comprendre les préférences individuelles des clients en analysant leurs données. Cela conduit à des expériences personnalisées, que ce soit dans le domaine du commerce électronique, des services en ligne ou du marketing.
7. **Recherche scientifique et médicale** : Dans le domaine de la recherche scientifique et médicale, le big data est utilisé pour analyser dénormes ensembles de données génomiques, de données cliniques et d'autres données médicales. Cela conduit à des avancées significatives dans la compréhension des maladies et le développement de nouveaux traitements.

En résumé, le big data joue un rôle essentiel dans la transformation numérique, la prise de décision informée et l'innovation, impactant divers secteurs de l'économie et de la société.

▼ Système d'information

Un système d'information (SI) est un ensemble organisé de ressources, telles que des personnes, des processus, des données, des technologies et des infrastructures, qui recueille, stocke, traite et distribue des informations au sein d'une organisation pour faciliter la prise de décision et le contrôle des activités.



Un système d'information (SI) typique comprend cinq composants principaux :

1. Matériel (Hardware) :

- *Définition* : Dispositifs physiques et équipements utilisés pour collecter, traiter, stocker et présenter des données.
- *Exemples* : Ordinateurs, serveurs, dispositifs réseau, dispositifs de stockage.

2. Logiciel (Software) :

- *Définition* : Programmes et applications qui fournissent des instructions pour que le matériel exécute des tâches spécifiques.
- *Exemples* : Systèmes d'exploitation, systèmes de gestion de bases de données, logiciels d'application.

3. Données :

- *Définition* : Faits bruts et chiffres qui sont traités par le système pour générer des informations significatives.
- *Exemples* : Bases de données, feuilles de calcul, documents, fichiers multimédias.

4. Procédures :

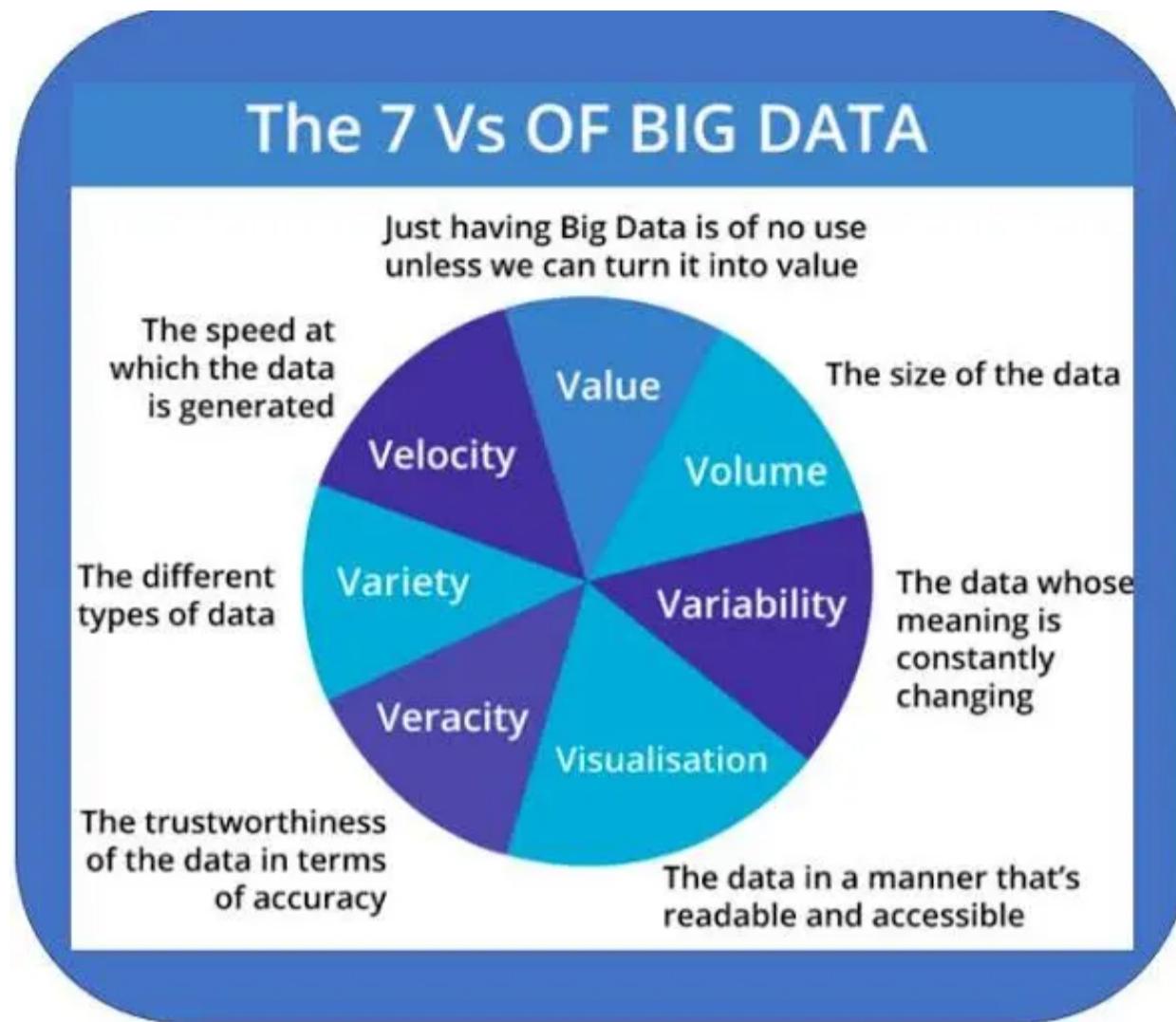
- *Définition* : Ensemble d'instructions ou de règles qui guident la manière dont le système traite et gère les données.
- *Exemples* : Procédures de saisie des données, protocoles de sécurité, processus de sauvegarde et de récupération.

5. Personnes :

- *Définition* : Utilisateurs, administrateurs et autres personnels impliqués dans le fonctionnement et l'utilisation du système d'information.
- *Exemples* : Utilisateurs finaux, personnel de support informatique, administrateurs système, analystes de données.

Ces cinq composants travaillent ensemble pour collecter, traiter, stocker et diffuser des informations au sein d'une organisation, facilitant la prise de décision et soutenant divers processus métier. L'intégration et la coordination efficaces de ces composants contribuent au succès global et à l'efficacité du système d'information.

▼ Les Propriétés de Big data



Les propriétés clés du big data, souvent résumées par les "3V" (Volume, Vélocité, Variété), peuvent être étendues à d'autres "V" et caractéristiques importantes. Voici quelques-unes des propriétés essentielles du big data :

1. Volume :

- Définition** : Fait référence à la quantité massive de données générées et stockées.
- Exemple** : Téraoctets ou pétaoctets de données provenant de sources telles que les réseaux sociaux, les capteurs et les transactions en ligne.

2. Vélocité :

- Définition** : Fait référence à la rapidité à laquelle les données sont générées, collectées et traitées.
- Exemple** : Flux constant de données en temps réel à partir de capteurs, médias sociaux et autres sources.

3. Variété :

- Définition** : Fait référence à la diversité des types de données, y compris les données structurées, semi-structurées et non structurées.
- Exemple** : Données provenant de bases de données relationnelles, fichiers XML ou JSON, médias tels que textes, images et vidéos.

4. Véracité :

- Définition** : Fait référence à la qualité et à la fiabilité des données.
- Exemple** : La précision des données peut varier, en particulier dans des sources telles que les médias sociaux.

5. Valeur :

Bien sûr, je vais aligner les "3V", "5V", et "7V" du big data pour que vous puissiez mieux les comprendre :

1. "3V" du Big Data :

- Volume (Volume)** : Quantité massive de données générées et stockées.
- Vélocité (Velocity)** : Rapidité à laquelle les données sont générées, traitées et mises à jour.
- Variété (Variety)** : Diversité des types de données, qu'elles soient structurées, semi-structurées ou non structurées.

2. "5V" du Big Data (Ajout de Véracité et Valeur) :

- Véracité (Veracity)** : Qualité, fiabilité et précision des données.
- Valeur (Value)** : Capacité à extraire des informations utiles et significatives des données.

3. "7V" du Big Data (Ajout de Variabilité et Volatilité) :

- Variabilité (Variability)** : Fluctuation dans la génération et le traitement des données.
- Volatilité (Volatility)** : Durée de validité des données.

Ces "7V" fournissent une vue plus complète des défis et des caractéristiques du big data, en mettant l'accent sur la qualité des données (Véracité), la valeur des informations extraites (Valeur), la variabilité des données générées, et la volatilité dans la durée de validité des données. Ces dimensions supplémentaires reflètent la complexité du paysage du big data et la diversité des sources et des applications de ces données massives.

- *Définition* : Fait référence à la capacité d'extraire des informations significatives et de valeur à partir des données.
- *Exemple* : L'analyse du big data permet de découvrir des tendances, des modèles et des informations utiles pour la prise de décision.

6. Variabilité :

- *Définition* : Fait référence à la fluctuation dans la génération et le traitement des données.
- *Exemple* : Les pics de trafic sur un site web pendant des promotions peuvent entraîner une variabilité importante dans les données générées.

7. Volatilité :

- *Définition* : Fait référence à la durée de validité des données.
- *Exemple* : Les données liées à un événement spécifique peuvent devenir obsolètes rapidement, nécessitant une mise à jour fréquente.

Ces propriétés illustrent la complexité et la diversité des défis associés à la gestion et à l'analyse du big data. Les entreprises et les organisations cherchent à exploiter ces propriétés pour obtenir des avantages concurrentiels, prendre des décisions informées et innover dans leurs processus.

▼ Type des Données

En big data, les types de données peuvent être extrêmement variés en raison de la diversité des sources de données. Voici quelques types de données courants rencontrés dans le contexte du big data :

1. Données Structurées :

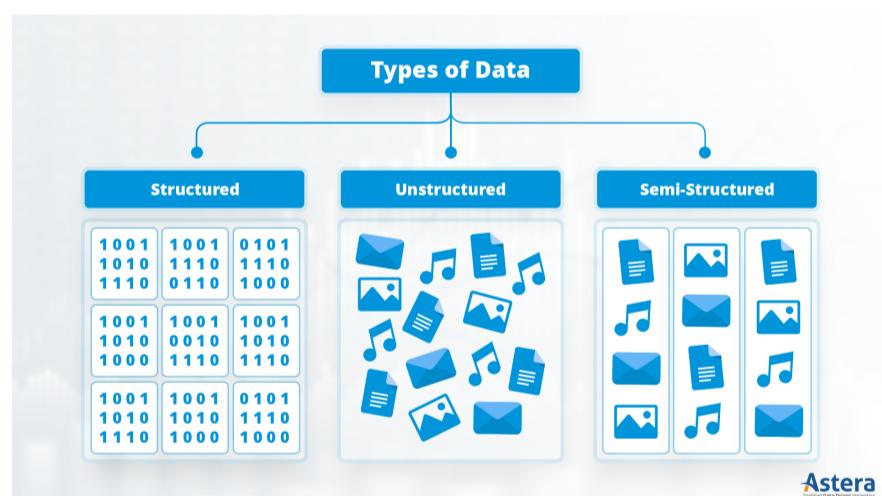
- *Définition* : Des données organisées selon un modèle fixe, généralement stockées dans des bases de données relationnelles.
- *Exemple* : Tableaux de bases de données SQL.

2. Données Non Structurées :

- *Définition* : Des données qui n'ont pas de structure prédéfinie, ce qui peut inclure des textes, des images, des vidéos, etc.
- *Exemple* : Fichiers texte, documents PDF, images.

3. Données Semi-Structurées :

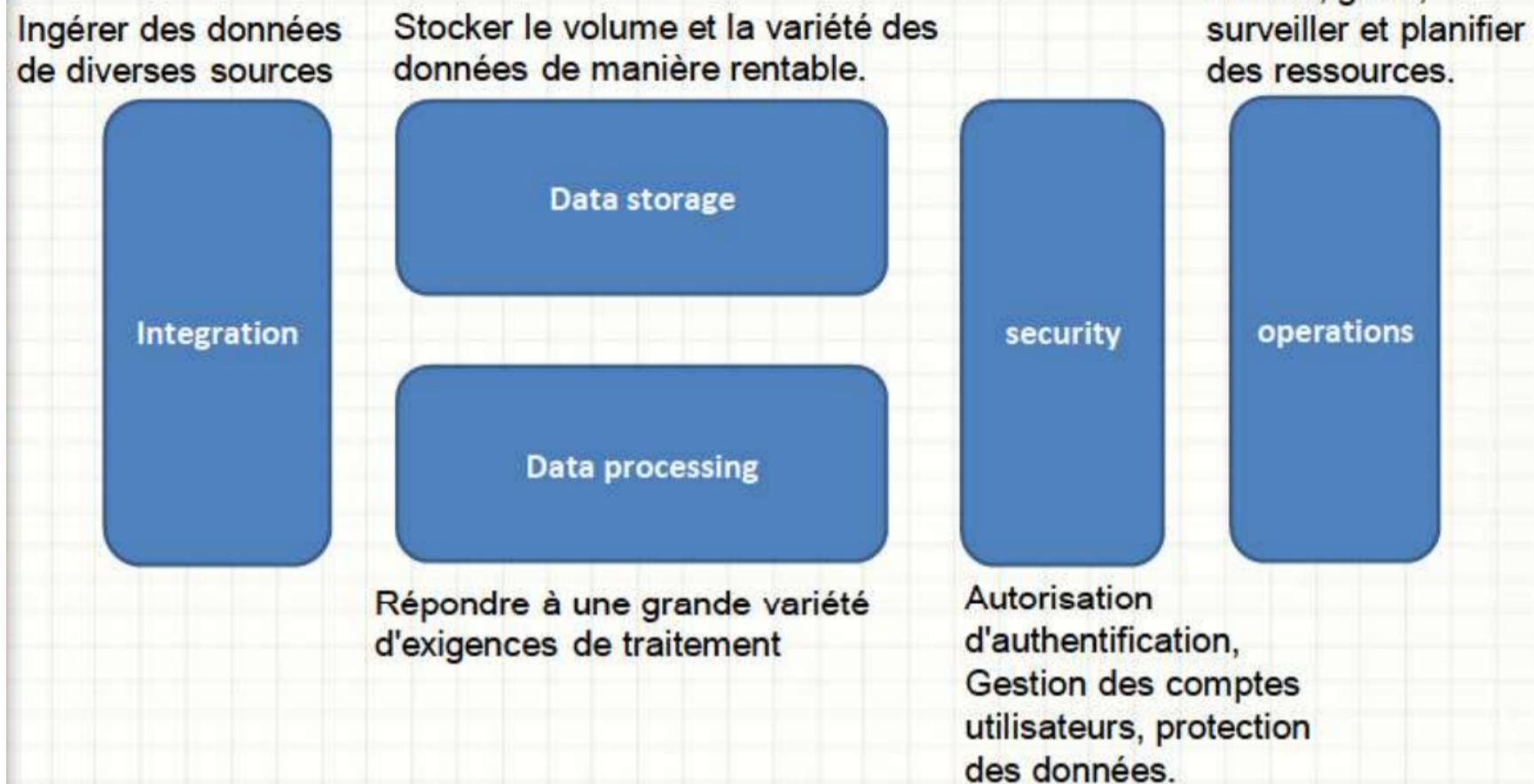
- *Définition* : Des données qui ne sont pas strictement structurées mais qui ont une certaine organisation, généralement au format JSON, XML.
- *Exemple* : Documents JSON, fichiers XML.



Ces types de données reflètent la diversité inhérente au big data, où la capacité à traiter et à analyser des données provenant de sources variées est essentielle pour en tirer des informations utiles.

▼ Architecture Générale Big data

Principaux composants d'une architecture Big Data



L'architecture générale du big data est conçue pour traiter, stocker et analyser de grandes quantités de données, en tirant parti de divers composants et technologies. Voici une vue d'ensemble des principaux éléments de l'architecture big data :

1. Sources de Données :

- Définition* : Les endroits d'où proviennent les données, tels que les applications, les capteurs, les médias sociaux, les bases de données, etc.

2. Ingestion des Données (Data Ingestion) :

- Fonction* : La collecte et l'importation initiales des données depuis les sources vers le système de traitement big data.
- Composants* : Apache Kafka, Apache NiFi, AWS Kinesis.

3. Stockage des Données :

- Fonction* : Le stockage durable des données brutes ou transformées.
- Composants* : Hadoop Distributed File System (HDFS), Amazon S3, Google Cloud Storage.

4. Traitement des Données (Data Processing) :

- Fonction* : L'analyse, la transformation et le nettoyage des données pour en extraire des informations utiles.
- Composants* : Apache Spark, Apache Flink, Hadoop MapReduce, Hive.

5. Analyse et Exploration des Données :

- Fonction* : L'exploration des données pour découvrir des tendances, des modèles et des informations significatives.
- Composants* : Apache Hive, Apache Pig, Apache Zeppelin.

6. Bases de Données NoSQL :

- Fonction* : Stockage de données non structurées ou semi-structurées, offrant une flexibilité et une évolutivité importantes.
- Composants* : MongoDB, Cassandra, Couchbasen, HBase.

7. Entrepôts de Données (Data Warehouses) :

- Fonction* : Stockage centralisé et structuré de données pour l'analyse et la génération de rapports.
- Composants* : Amazon Redshift, Google BigQuery, Snowflake.

8. Moteurs de Recherche :

- Fonction* : Indexation et recherche rapides des données.

- *Composants* : Elasticsearch, Apache Solr.

9. Systèmes de Gestion de Cluster :

- *Fonction* : La gestion et l'orchestration de l'ensemble du système big data.
- *Composants* : Apache Hadoop YARN, Apache Mesos, Kubernetes.

10. Sécurité et Gouvernance :

- *Fonction* : La protection des données, la gestion des accès et la conformité aux réglementations.
- *Composants* : Apache Ranger, Apache Knox, solutions de chiffrement.

11. Visualisation des Données :

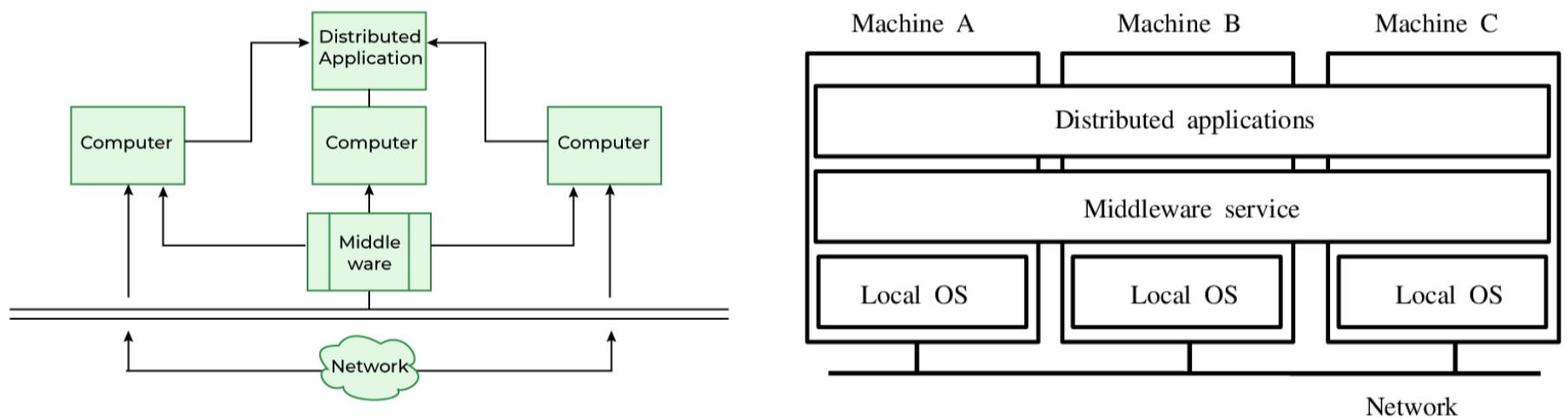
- *Fonction* : La représentation graphique des résultats de l'analyse pour une compréhension facile.
- *Composants* : Tableau, Power BI, Apache Superset.

12. Intégration avec des Outils d'Entreprise :

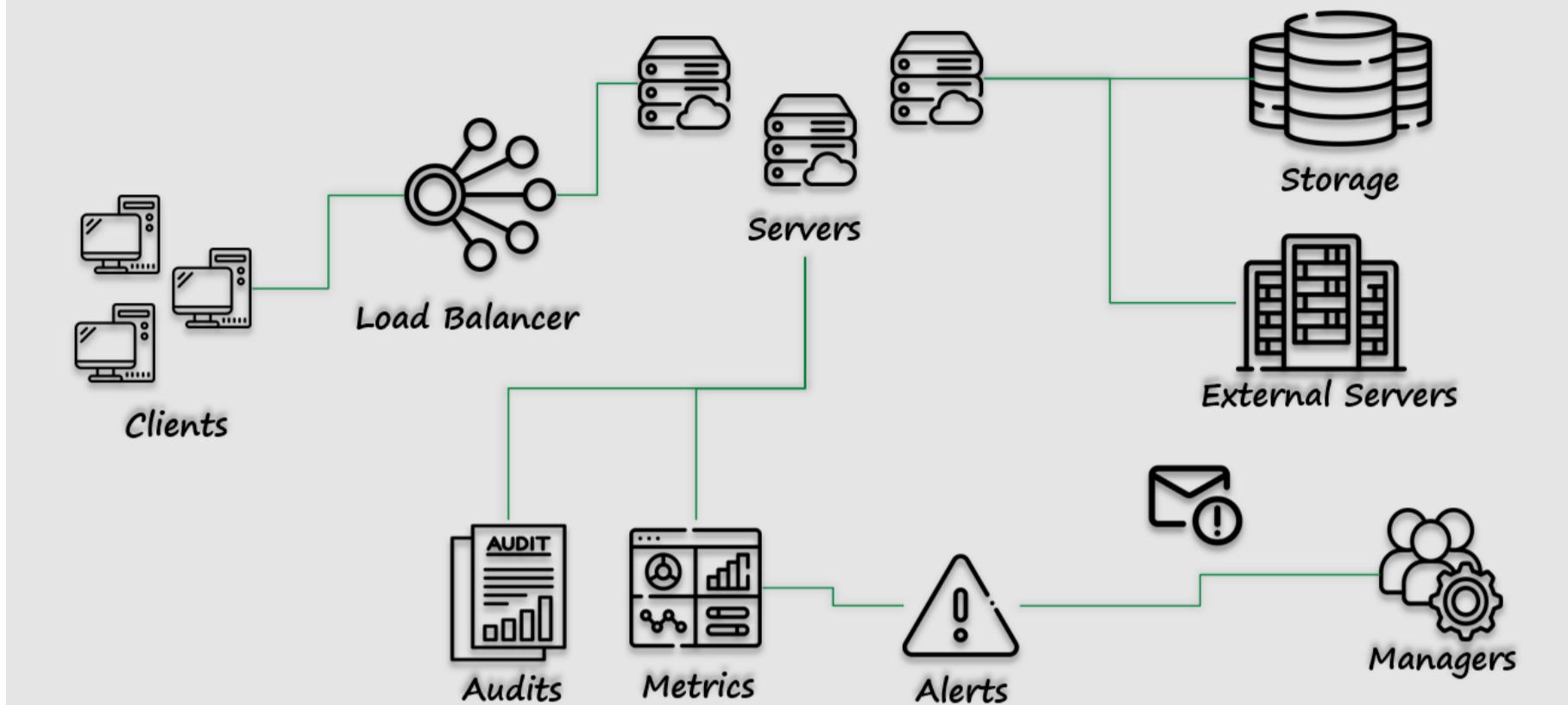
- *Fonction* : L'interopérabilité avec d'autres systèmes et outils déjà présents dans l'entreprise.
- *Composants* : Connecteurs et API personnalisés.

Cette architecture big data peut varier en fonction des besoins spécifiques de l'entreprise et des technologies choisies. Elle est conçue pour offrir une scalabilité, une flexibilité et une efficacité élevées dans le traitement de grandes quantités de données.

▼ Les Systèmes Distribués



System Architecture



des systèmes distribués. Un système distribué est un ensemble d'ordinateurs interconnectés qui travaillent ensemble pour accomplir une tâche commune. Chaque nœud du système a sa propre mémoire et son propre processeur, et la coordination entre ces nœuds permet d'atteindre un objectif global. Voici quelques aspects clés des systèmes distribués :

1. Communication entre Nœuds :

- *Définition* : Les nœuds dans un système distribué communiquent entre eux pour partager des informations et coordonner leurs activités.
- *Exemple* : Protocoles de communication tels que HTTP, RPC (Remote Procedure Call), et frameworks comme Apache Kafka pour la diffusion de messages.

2. Coordination et Consistance :

- *Définition* : Garantir que tous les nœuds du système arrivent à un état cohérent malgré les éventuelles défaillances et les retards de communication.
- *Exemple* : Algorithmes de consensus comme Paxos et Raft, bases de données distribuées utilisant le modèle CAP (Consistency, Availability, Partition Tolerance).

3. Gestion des Pannes :

- *Définition* : La capacité du système à fonctionner correctement même en présence de nœuds défaillants.
- *Exemple* : Réplication des données, détection de pannes, et mécanismes de reprise sur incident.

4. Synchronisation :

- *Définition* : Assurer que les différentes parties du système sont synchronisées dans le temps malgré la nature distribuée des nœuds.
- *Exemple* : Services de synchronisation du temps comme NTP (Network Time Protocol).

5. Scalabilité :

- *Définition* : La capacité du système à s'adapter à une augmentation du nombre de nœuds sans sacrifier les performances.
- *Exemple* : Partitionnement horizontal des données, élasticité dans les systèmes cloud.

6. Répartition de la Charge :

- *Définition* : La distribution équitable des charges de travail entre les nœuds du système.
- *Exemple* : Équilibrage de charge dans les serveurs web, distribution de tâches dans les clusters de traitement de données.

7. Intégrité et Sécurité :

- *Définition* : Garantir que les données et les opérations du système sont sécurisées et ne sont pas compromises.
- *Exemple* : Mécanismes de chiffrement, authentification, et autorisation.

Les systèmes distribués sont omniprésents dans de nombreuses applications modernes, des infrastructures cloud aux réseaux sociaux, en passant par les bases de données distribuées. La conception et la gestion de ces systèmes nécessitent une compréhension approfondie des défis liés à la distribution des ressources et des opérations.

Amélioration et mises à niveau du système:

	Vertical Scaling	Horizontal Scaling
Définition	Augmenter les ressources d'un serveur existant, comme la RAM, le processeur, ou le stockage.	Ajouter davantage de serveurs au système pour répartir la charge.
Avantages	Simplicité, pas de complexité liée à la gestion de plusieurs serveurs.	Meilleure évolutivité, coût potentiellement inférieur.
Inconvénients	Coût élevé, limite physique sur la capacité d'extension.	Complexité de la gestion des clusters, nécessite une conception distribuée.

les Contraintes de Conception de Système :

Les contraintes de conception de système sont des limitations qui influent sur la façon dont un système est développé. Elles peuvent inclure :

1. Disponibilité / Availability :

- *Définition* : Assurer que le système est opérationnel et accessible lorsque nécessaire.
- *Exemple* : 99,99% d'uptime.

2. Fiabilité [Tolérance aux Pannes - Redondance] / Reliability[fault tolerance-redundancy] :

- *Définition* : La capacité à maintenir le fonctionnement même en cas de défaillance d'un composant.
- *Exemple* : Utilisation de redondances, sauvegardes régulières.

3. Débit / through put:

- *Définition* : La quantité de travail qu'un système peut accomplir par unité de temps.
- *Exemple* : Transactions par seconde (TPS).

4. Latence / latency:

- *Définition* : Le temps qu'il faut pour exécuter une opération ou pour qu'une donnée traverse le système.
- *Exemple* : Latence réseau, latence d'accès aux données.

Trade-Off Temps, Coût :

• Temps vs. Coût :

- *Trade-Off* : L'accélération du développement peut entraîner une augmentation des coûts.
- *Exemple* : L'utilisation de frameworks préconstruits peut accélérer le développement mais peut nécessiter des ajustements coûteux.

Suggestions pour l'Amélioration du Système :

1. Optimisation des Requêtes :

- *Approche* : Analyser et optimiser les requêtes de base de données.
- *Avantages* : Amélioration des performances sans nécessiter de changements majeurs dans l'infrastructure.

2. Mise en Cache :

- *Approche* : Utilisation de mécanismes de mise en cache pour stocker temporairement des résultats fréquemment utilisés.
- *Avantages* : Réduction de la latence pour les requêtes répétitives.

3. Équilibrage de Charge :

- *Approche* : Distribuer équitablement la charge de travail entre plusieurs serveurs.

- **Avantages** : Amélioration de la disponibilité et de la fiabilité.

4. Optimisation des Images :

- **Approche** : Compression d'images pour réduire la charge réseau et le temps de chargement.
- **Avantages** : Amélioration des performances du front-end.

5. Mise à Niveau Matérielle :

- **Approche** : Mise à niveau du matériel des serveurs pour augmenter les ressources.
- **Avantages** : Accroissement des capacités sans révolutionner l'architecture.

6. Évolutivité Horizontale :

- **Approche** : Ajout de serveurs pour gérer une augmentation de la charge.
- **Avantages** : Meilleure gestion des pics de trafic et meilleure évolutivité.

7. Optimisation du Code :

- **Approche** : Analyse et amélioration du code pour des performances optimales.
- **Avantages** : Réduction des goulets d'étranglement et de la consommation de ressources.

Chaque suggestion dépend des besoins spécifiques du système et des contraintes opérationnelles. L'équilibre entre ces différentes considérations est crucial pour obtenir un système performant, fiable et évolutif.

RDBMS (Système de Gestion de Bases de Données Relationnelles) vs. NoSQL (Système de Gestion de Bases de Données Non Relationnelles) :

Aspect	RDBMS	NoSQL
Modèle de Données	Structurellement tabulaire avec des relations définies.	Divers modèles, y compris documents, clé-valeur, colonnes, graphes, etc.
Schéma	Schéma fixe et rigide défini à l'avance.	Schéma flexible, permettant l'ajout de champs sans modification globale.
Transactions ACID	Prise en charge des transactions ACID (Atomicité, Cohérence, Isolation, Durabilité).	Souvent moins strictes sur les garanties ACID, privilégiant la disponibilité et la tolérance aux partitions.
Évolutivité Verticale	Extensible par le biais d'une mise à niveau matérielle (augmentation des ressources sur un serveur).	Extensible horizontalement en ajoutant des nœuds au cluster.
Exemples de Bases de Données	MySQL, PostgreSQL, Oracle.	MongoDB, Cassandra, Redis.

System Design Aspect (Aspect de Conception Système) in Both:

Les aspects de conception système varient selon qu'il s'agit d'un RDBMS ou d'un NoSQL. Certains points communs incluent :

Aspects de Conception Système	RDBMS	NoSQL
1. Évolutivité :	Souvent extensible verticalement, nécessitant des mises à niveau matérielles.	Préférence pour l'évolutivité horizontale, avec l'ajout de nœuds.
2. Considérations de Schéma :	Schéma fixe avec des relations prédéfinies.	Schéma flexible, adaptatif aux changements de données.
3. Transactions :	Transactions ACID pour garantir la cohérence des données.	Moins strict sur les transactions ACID pour privilégier la disponibilité et la partition tolérante.

Cette table résume les aspects de conception système liés à l'évolutivité, aux considérations de schéma et aux transactions pour les systèmes RDBMS et NoSQL.

Replication / Sharding:

Stratégies de Distribution des Données	RDBMS	NoSQL
1. RéPLICATION (RéPLICATION) :	Utilisé pour la tolérance aux pannes et la disponibilité. Les données sont copiées sur plusieurs serveurs.	Couramment utilisé pour améliorer la disponibilité. Chaque nœud peut contenir une copie des données.

Stratégies de Distribution des Données	RDBMS	NoSQL
2. Fragmentation (Sharding) :	Moins courant, mais certaines bases de données relationnelles supportent le sharding pour répartir les données.	Fréquemment utilisé pour distribuer les données sur plusieurs serveurs, améliorant l'évolutivité horizontale.

Cette table résume les stratégies de distribution des données, notamment la réplication et le sharding, pour les systèmes RDBMS et NoSQL.

CAP Theorem (Théorème CAP) :

Le théorème CAP stipule que dans un système distribué, on ne peut garantir simultanément les trois propriétés suivantes : Consistency (Cohérence), Availability (Disponibilité), et Partition Tolerance (Tolérance aux Partitions).

1. **RDBMS** : Souvent axé sur la Cohérence et la Tolérance aux Partitions, sacrifiant parfois la Disponibilité en cas de partition.
2. **NoSQL** : Souvent axé sur la Disponibilité et la Tolérance aux Partitions, sacrifiant parfois la Cohérence en cas de partition.

Object Storage / Message Queues :

1. Object Storage (Stockage d'Objets) :

- Stocke les données sous forme d'objets, offrant une évolutivité élevée et une gestion flexible des données non structurées.
- *Exemple* : Amazon S3, Google Cloud Storage.

2. Message Queues (File d'Attente de Messages) :

- Facilite la communication asynchrone entre les composants du système, permettant une meilleure extensibilité et la gestion des charges de travail.
- *Exemple* : RabbitMQ, Apache Kafka.

▼ informatique distribuée & Traitement des Données

Concepts Principaux du Calcul Distribué dans le Big Data

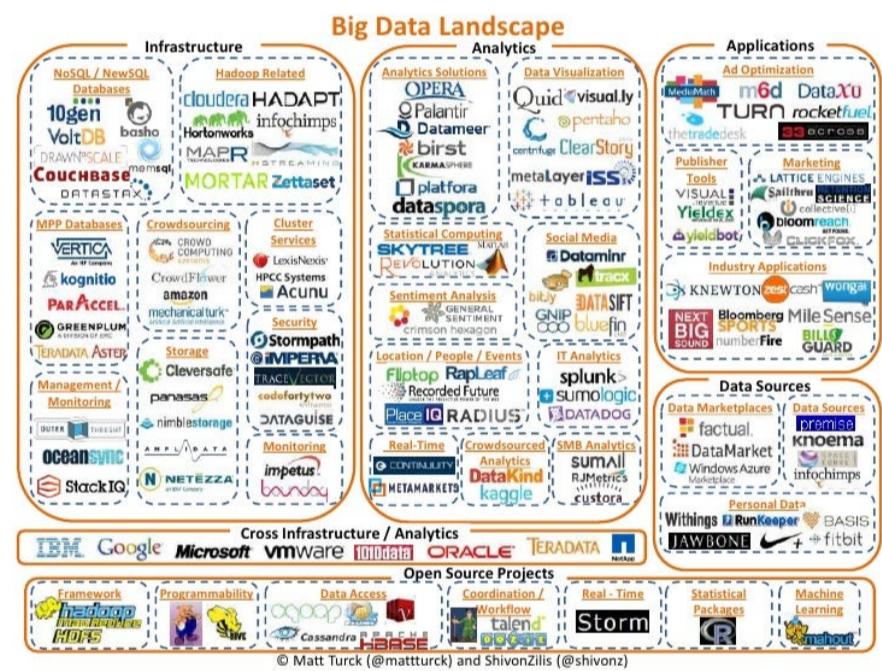
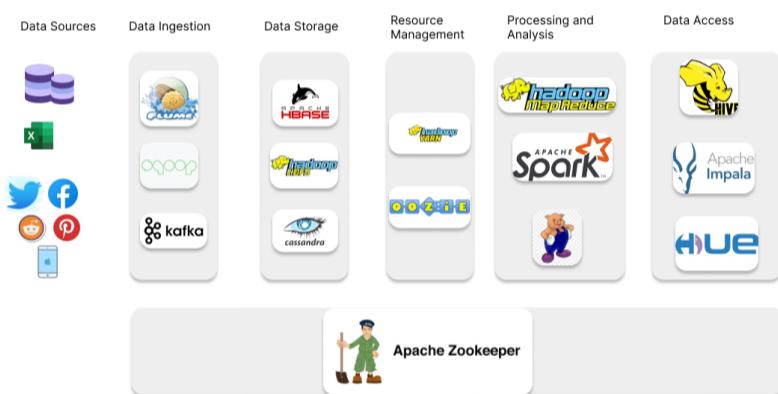
Aspect	Description
1. Parallel Processing	Dans un environnement de calcul distribué, les tâches sont divisées en sous-tâches traitées indépendamment sur différentes machines simultanément.
2. Communication Between Nodes	Les nœuds communiquent pour partager des informations, coordonner des tâches et échanger des données, assurant une synchronisation appropriée.
3. Scalabilité	Le calcul distribué permet de s'étendre horizontalement en ajoutant plus de machines, gérant ainsi des charges de travail plus importantes.
4. Tolérance aux Pannes	Des mécanismes de tolérance aux pannes intègrent la redondance, la réplication des données et des stratégies de récupération.
5. Équilibrage de Charge	Assure une répartition uniforme des charges de travail entre les nœuds, évitant les goulets d'étranglement et maximisant l'efficacité.
6. Distribution et Réplication des Données	Les données sont réparties et répliquées sur plusieurs nœuds, améliorant la tolérance aux pannes et garantissant la disponibilité des données.
7. Contrôle Centralisé vs. Décentralisé	Les systèmes distribués peuvent avoir un contrôle centralisé ou décentralisé, influençant la prise de décision.
8. Exemples de Systèmes Distribués	De nombreuses applications modernes, y compris les plateformes de cloud computing et les frameworks comme Apache Spark et Apache Hadoop.
9. Défis	Le calcul distribué pose des défis liés à la cohérence des données, à la synchronisation et aux surcharges de communication, nécessitant des algorithmes efficaces.

Traitement dans le Monde du Big Data

Aspect	Description
1. Manipulation	Changement de la structure ou du format des données pour les rendre adaptées à l'analyse ou à la présentation.
2. Transformation	Conversion des données d'une forme à une autre, incluant l'agrégation, le filtrage ou le tri des données.
3. Analyse	Examen des données pour identifier des modèles, des tendances, des corrélations ou des enseignements, souvent avec l'application de techniques statistiques ou mathématiques.
4. Calcul	Réalisation de calculs ou d'opérations mathématiques sur les données pour dériver de nouvelles informations ou métriques.
5. Stockage	Gestion du stockage des données, y compris la sauvegarde des données traitées pour une utilisation future.
6. Récupération	Accès aux données stockées pour permettre aux utilisateurs de récupérer des informations spécifiques au besoin.
7. Présentation	Affichage des données traitées sous une forme compréhensible et significative pour les utilisateurs, souvent à travers des graphiques, des rapports ou d'autres visualisations.

Dans le contexte des grands frameworks de traitement de données comme Apache Spark, le traitement se réfère à l'exécution parallèle de tâches sur des ensembles de données distribués. Spark fournit un moteur d'analyse unifié pour le traitement efficace de grands volumes de données en répartissant la charge de travail sur un cluster d'ordinateurs.

▼ Big data Ecosystem



L'écosystème du big data est composé d'un ensemble varié de technologies, outils et frameworks qui travaillent de concert pour collecter, stocker, traiter, analyser et visualiser de grandes quantités de données. Voici un aperçu des principaux éléments de l'écosystème du big data :

1. Stockage de Données :

- Hadoop Distributed File System (HDFS)** : Système de fichiers distribué conçu pour stocker de grandes quantités de données sur des clusters de serveurs.
- Amazon S3, Google Cloud Storage** : Services de stockage objet dans le cloud, offrant une scalabilité élevée et une facilité d'accès aux données.

2. Traitement et Analyse :

- Apache Spark** : Moteur de traitement de données en mémoire, permettant le traitement batch et le traitement en temps réel.
- Apache Flink** : Framework de traitement de flux pour l'analyse en temps réel avec des caractéristiques de faible latence.

3. Bases de Données NoSQL :

- **MongoDB, Cassandra, Couchbase** : Bases de données NoSQL adaptées au stockage de données non structurées et à l'évolutivité horizontale.
- **Amazon DynamoDB** : Service de base de données NoSQL entièrement géré dans le cloud.

4. Bases de Données SQL :

- **MySQL, PostgreSQL, Oracle** : Bases de données relationnelles qui peuvent également être utilisées dans des environnements big data pour des besoins spécifiques.
- **Amazon Redshift, Google BigQuery** : Bases de données de type entrepôt de données adaptées à l'analyse de données massives.

5. Outils de Traitement de Flux et Messagerie :

- **Apache Kafka** : Plateforme de streaming d'événements distribuée pour la gestion de flux de données en temps réel.
- **RabbitMQ, Apache Pulsar** : Brokers de messages pour la communication asynchrone et la gestion de files d'attente.

6. Frameworks de Machine Learning :

- **TensorFlow, PyTorch** : Bibliothèques populaires pour la création de modèles d'apprentissage automatique.
- **Scikit-Learn** : Bibliothèque Python pour l'apprentissage automatique et l'analyse de données.

7. Outils de Visualisation :

- **Tableau, Power BI** : Outils de visualisation de données pour créer des tableaux de bord interactifs.
- **D3.js** : Bibliothèque JavaScript pour la création de visualisations de données dynamiques sur le web.

8. Orchestration de Conteneurs :

- **Kubernetes** : Système d'orchestration de conteneurs pour automatiser le déploiement, la mise à l'échelle et la gestion.

9. Sécurité et Gestion des Identités :

- **Apache Ranger, Apache Knox** : Outils pour la gestion de la sécurité dans les environnements big data.
- **AWS Identity and Access Management (IAM)** : Service de gestion des identités dans le cloud.

10. Outils de Surveillance et Journalisation :

- **Prometheus, Grafana** : Outils de surveillance et de visualisation de métriques.
- **ELK Stack (Elasticsearch, Logstash, Kibana)** : Solution pour la gestion des journaux et l'analyse de données de journal.

Cet écosystème dynamique continue d'évoluer avec l'introduction de nouvelles technologies et solutions. L'utilisation appropriée de ces outils dépend des besoins spécifiques du projet et des caractéristiques des données à traiter.

▼ Hadoop [HDFS/MapReduce/Yarn]

Présentation Générale de Hadoop :

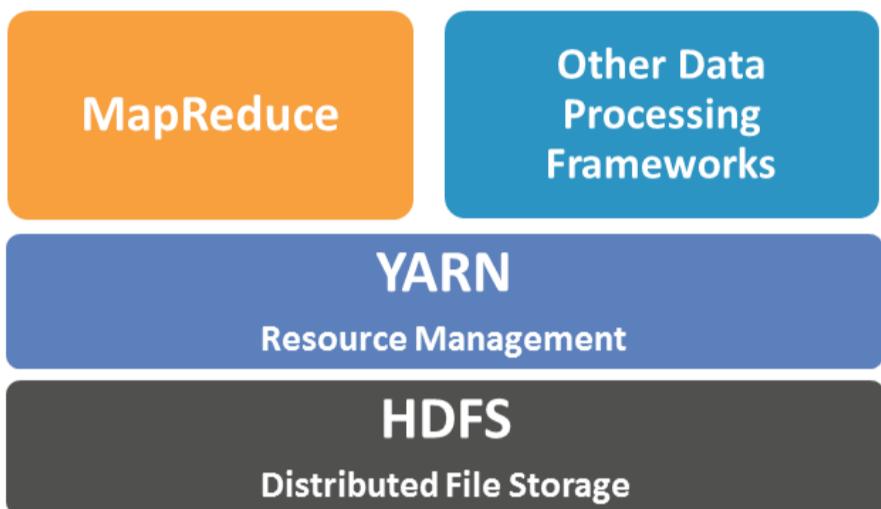
Hadoop est un framework open source conçu pour le stockage et le traitement distribués de grands ensembles de données à travers des grappes(clusters) d'ordinateurs. Il répond aux défis liés à la manipulation de big data en fournissant une solution évolutive, tolérante aux pannes et économique.



Hadoop v1.0



Hadoop v2.0

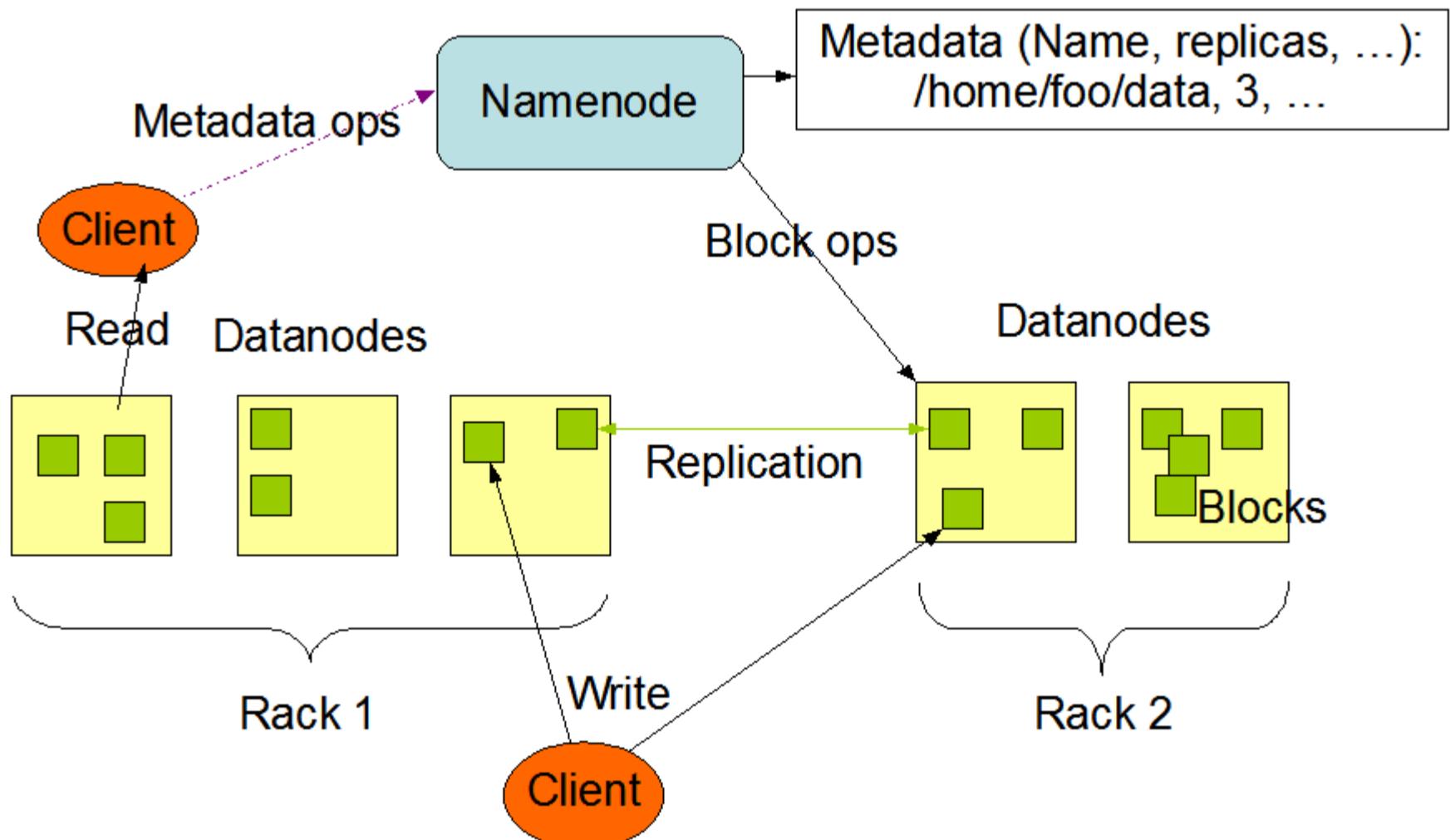


Architecture de Hadoop :

1. Système de Fichiers Distribué Hadoop (HDFS) :

- **Aperçu :** HDFS est le composant de stockage de Hadoop. Il divise de grands fichiers en blocs plus petits (généralement de 128 Mo ou 256 Mo) et les distribue sur les nœuds d'une grappe(cluster) Hadoop.

HDFS Architecture



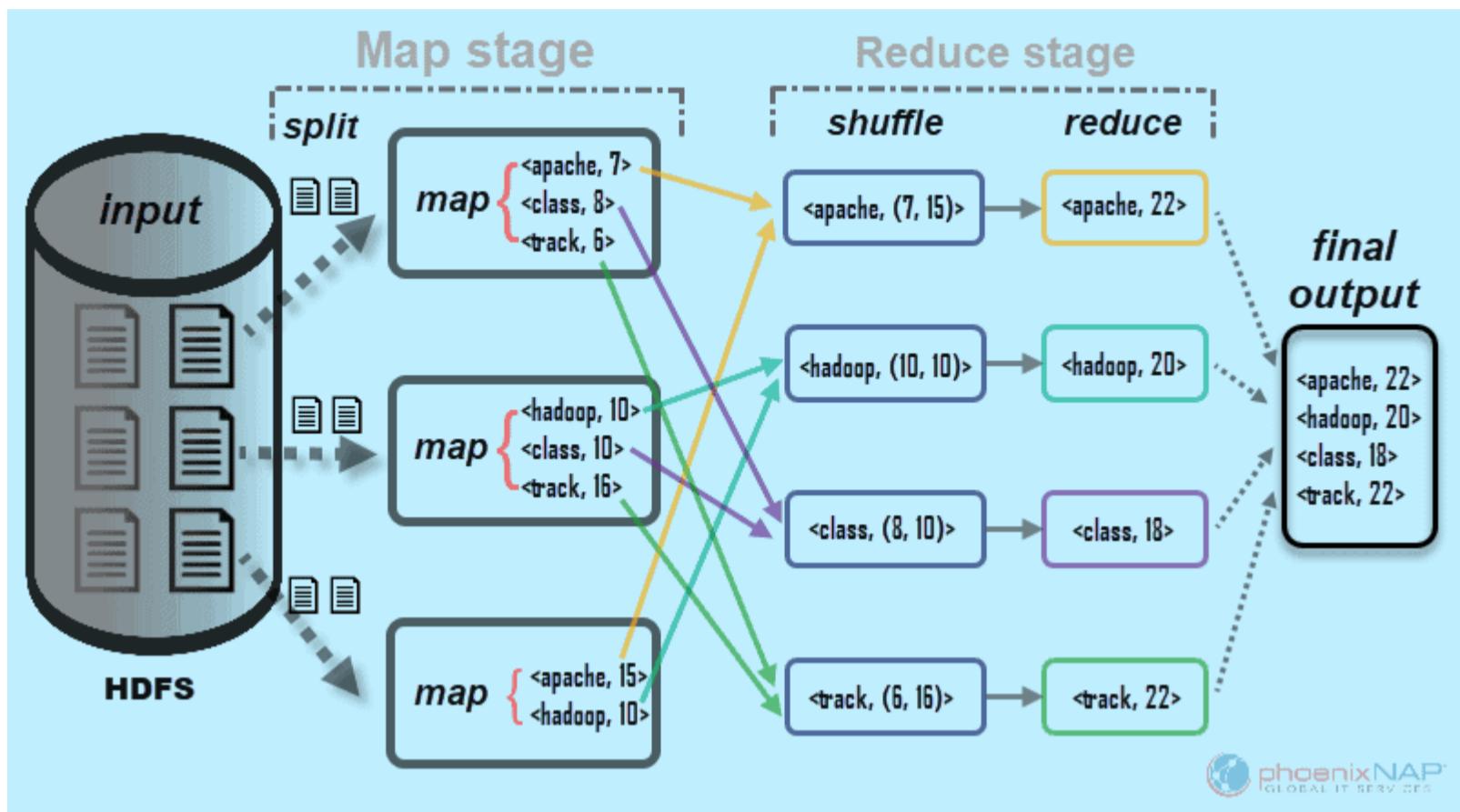
• Fonctionnalités :

- **NameNode** : Gère les métadonnées et garde une trace de l'emplacement des blocs de données.

- **DataNode** : Stocke les blocs de données réels et communique avec le NameNode.

2. MapReduce :

- **Aperçu** : MapReduce est un modèle de programmation et un moteur de traitement pour le traitement parallèle et distribué de grands ensembles de données.

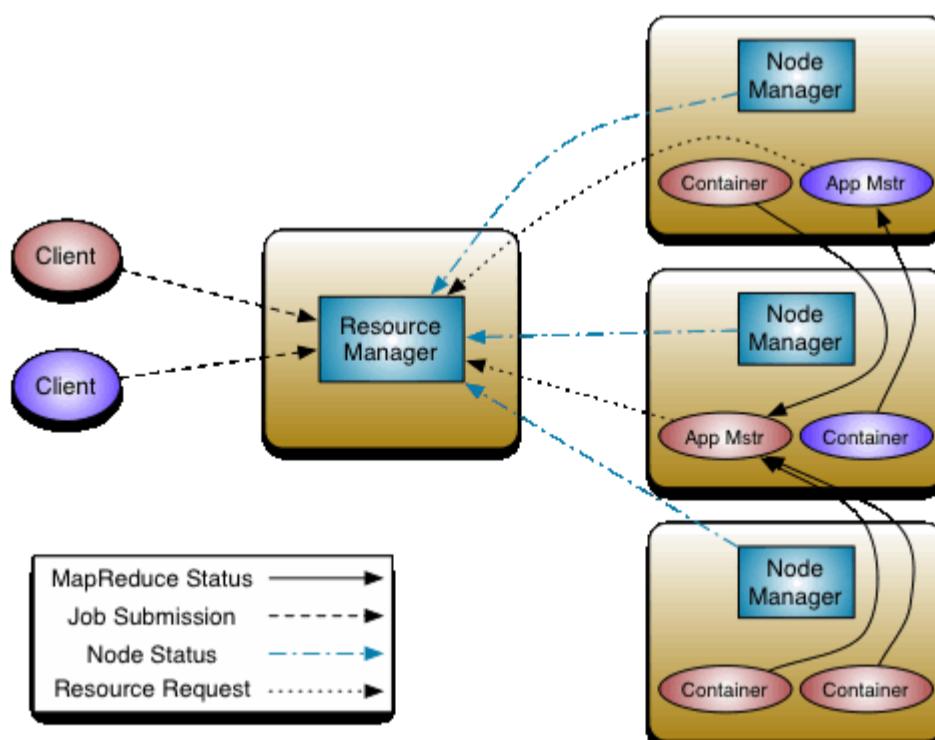


• Fonctionnalités :

- **Phase de Map** : Divise les données d'entrée en morceaux plus petits, les traite indépendamment et produit des paires clé-valeur intermédiaires.
- **Phase de Shuffle et de Tri** : Groupe et trie les paires clé-valeur intermédiaires.
- **Phase de Reduce** : Agrège les paires clé-valeur triées, produisant la sortie finale.

3. Hadoop YARN (Yet Another Resource Negotiator) :

- **Aperçu** : YARN est la couche de gestion des ressources dans Hadoop. Il permet à plusieurs moteurs de traitement de données de partager et d'allouer dynamiquement des ressources.



• Fonctionnalités :

- **ResourceManager/Master** : Gère les ressources à travers la grappe(cluster) et planifie les applications.
- **NodeManager/Slave** : Gère les ressources sur les nœuds individuels et surveille l'exécution des conteneurs.

Modules Fonctionnels Hadoop :

1. Hadoop Common :

- **Aperçu** : Contient des bibliothèques, des utilitaires et des APIs qui fournissent une base pour les autres modules Hadoop.
- **Fonctionnalités** : Inclut des outils et des bibliothèques essentiels pour Hadoop, tels que les bibliothèques du Système de Fichiers Distribué Hadoop (HDFS).

2. Hadoop MapReduce :

- **Aperçu** : Implémente le modèle de programmation MapReduce pour le traitement de grands ensembles de données.
- **Fonctionnalités** : Fournit les outils nécessaires et l'environnement d'exécution pour exécuter des travaux MapReduce sur une grappe Hadoop.

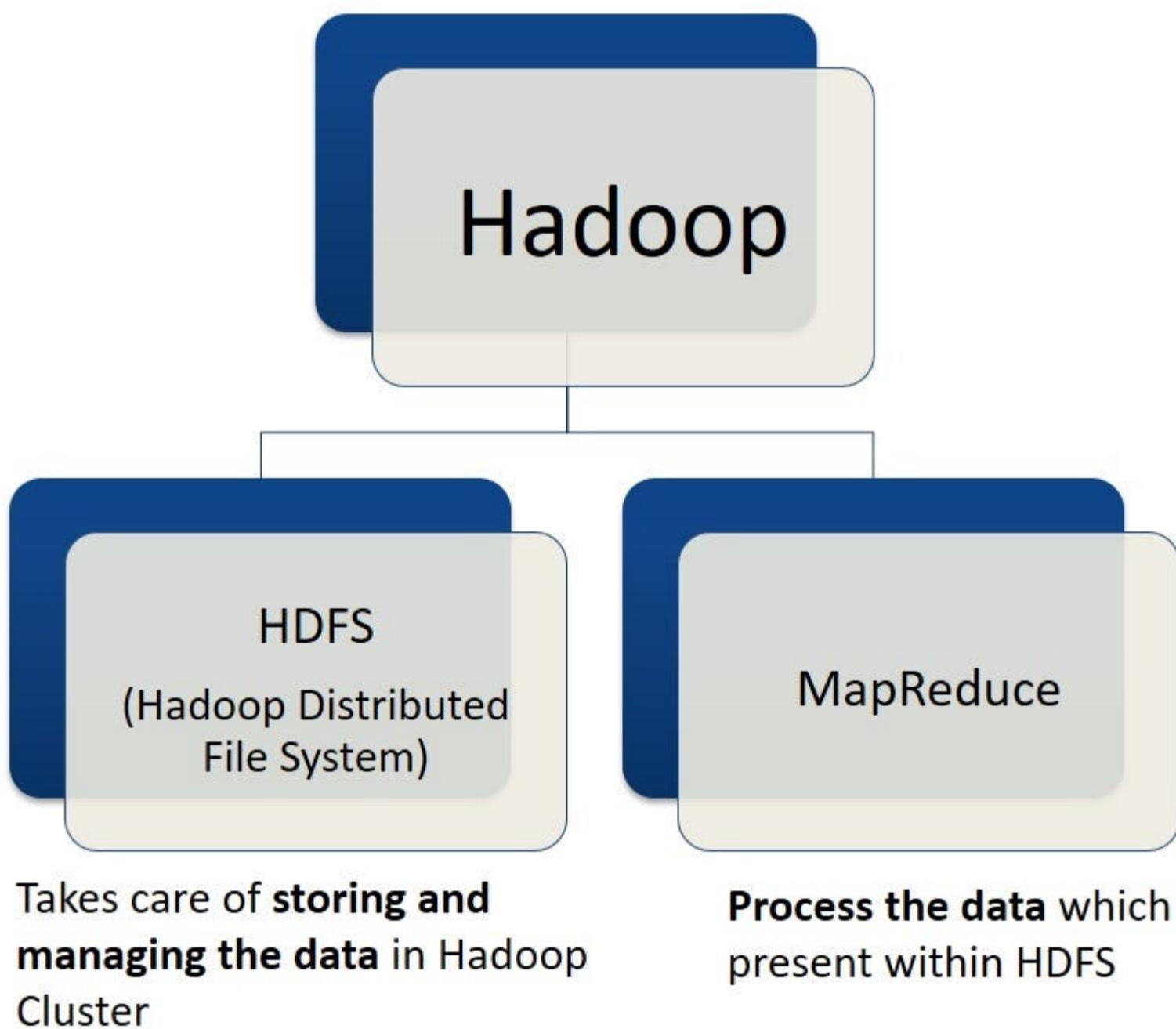
3. Système de Fichiers Distribué Hadoop (HDFS) :

- **Aperçu** : Un système de fichiers distribué conçu pour stocker de grands fichiers sur plusieurs nœuds dans une grappe Hadoop.
- **Fonctionnalités** : Facilite le stockage fiable et efficace de grands ensembles de données en les divisant en blocs et en les distribuant dans la grappe.

4. Hadoop YARN :

- **Aperçu** : YARN est une couche de gestion des ressources qui permet le partage et l'allocation de ressources entre plusieurs applications dans une grappe Hadoop.
- **Fonctionnalités** : Gère les ressources, planifie les applications et supervise l'exécution des tâches dans un environnement distribué.

HDFS(distributed storage), MapReduce(distributed processing), YARN(Ressource Manager) :



- **HDFS** : Fournit un stockage fiable, scalable et tolérant aux pannes pour de grands ensembles de données. Il divise les données en blocs, les réplique sur les nœuds et assure une haute disponibilité.
- **MapReduce** : Permet le traitement de vastes quantités de données en parallèle en divisant les tâches en phases de map et de reduce. Il convient au traitement par lots de données à grande échelle.
- **YARN** : sépare la gestion des ressources et la planification des tâches dans Hadoop. Il permet à plusieurs applications de partager dynamiquement et efficacement des ressources, prenant en charge divers moteurs de traitement au-delà de MapReduce.

L'architecture modulaire et distribuée de Hadoop en fait un framework puissant pour le traitement et l'analyse de grands ensembles de données, permettant aux organisations de tirer des enseignements du big data de manière évolutive et rentable.

▼ HDFS commandes

Les Commandes de HDFS définie par [`hdfs dfs`] ou [`hadoop -fs`]

Dans Hadoop, la commande `hdfs dfs` est une interface en ligne de commande permettant d'interagir avec le Hadoop Distributed File System (HDFS). Elle permet aux utilisateurs d'effectuer diverses opérations sur les fichiers et les répertoires, telles que la création, la liste, le déplacement, la copie et la suppression de fichiers et de répertoires au sein de HDFS. Cette commande est couramment utilisée par les administrateurs et les développeurs Hadoop pour gérer et travailler avec les données stockées dans HDFS. Voici quelques commandes `hdfs dfs` courantes et leurs explications :

Commande <code>hdfs dfs</code> / <code>hadoop -fs</code>	Description
<code>hdfs dfs -ls [chemin]</code>	Liste les fichiers et répertoires au chemin spécifié dans HDFS.
<code>hdfs dfs -mkdir [chemin]</code>	Crée un nouveau répertoire dans HDFS au chemin spécifié.
<code>hdfs dfs -copyFromLocal [source-local] [destination-hdfs]</code>	Copie un fichier ou répertoire du système de fichiers local vers HDFS.
<code>hdfs dfs -copyToLocal [source-hdfs] [destination-local]</code>	Copie un fichier ou répertoire de HDFS vers le système de fichiers local.
<code>hdfs dfs -mv [source] [destination]</code>	Déplace ou renomme un fichier ou répertoire dans HDFS.
<code>hdfs dfs -rm [chemin]</code>	Supprime un fichier dans HDFS.
<code>hdfs dfs -rmdir [chemin]</code>	Supprime un répertoire vide dans HDFS.
<code>hdfs dfs -rm -r [chemin]</code>	Supprime récursivement des fichiers et répertoires.
<code>hdfs dfs -put [source-local] [destination-hdfs]</code>	Téléverse un fichier du système de fichiers local vers HDFS (similaire à <code>copyFromLocal</code>).
<code>hdfs dfs -get [source-hdfs] [destination-local]</code>	Télécharge un fichier de HDFS vers le système de fichiers local (similaire à <code>copyToLocal</code>).
<code>hdfs dfs -cat [source-hdfs]</code>	Affiche le contenu d'un fichier dans HDFS.
<code>hdfs dfs -chmod [permissions] [chemin]</code>	Modifie les autorisations d'un fichier ou répertoire dans HDFS.
<code>hdfs dfs -chown [propriétaire:groupe] [chemin]</code>	Modifie le propriétaire et le groupe d'un fichier ou répertoire dans HDFS.
<code>hdfs dfs -du [chemin]</code>	Affiche l'utilisation du disque d'un fichier ou répertoire dans HDFS.
<code>hdfs dfs -count [chemin]</code>	Affiche les informations sur l'utilisation du quota pour un fichier ou répertoire.
<code>hdfs dfs -setQuota [nombre] [chemin]</code>	Définit un quota sur le nombre de noms (fichiers et répertoires) dans un répertoire.

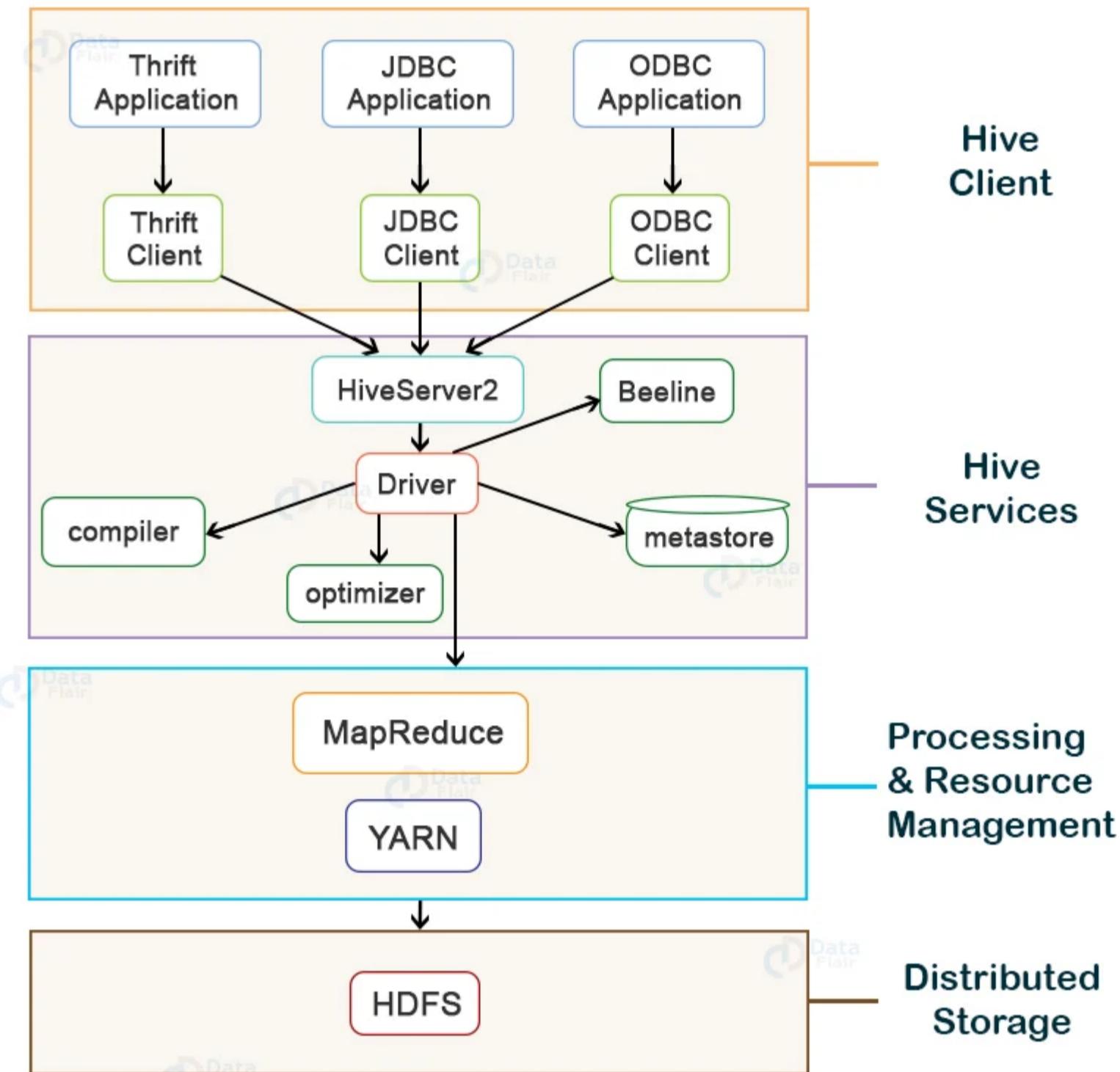
Ce sont quelques-unes des commandes `hdfs dfs` les plus courantes. Elles permettent aux utilisateurs de gérer les fichiers et répertoires dans HDFS depuis la ligne de commande. HDFS est un composant fondamental de Hadoop, et ces commandes sont essentielles pour travailler avec des données dans un cluster Hadoop.

▼ Hive

Présentation Générale de Hive :

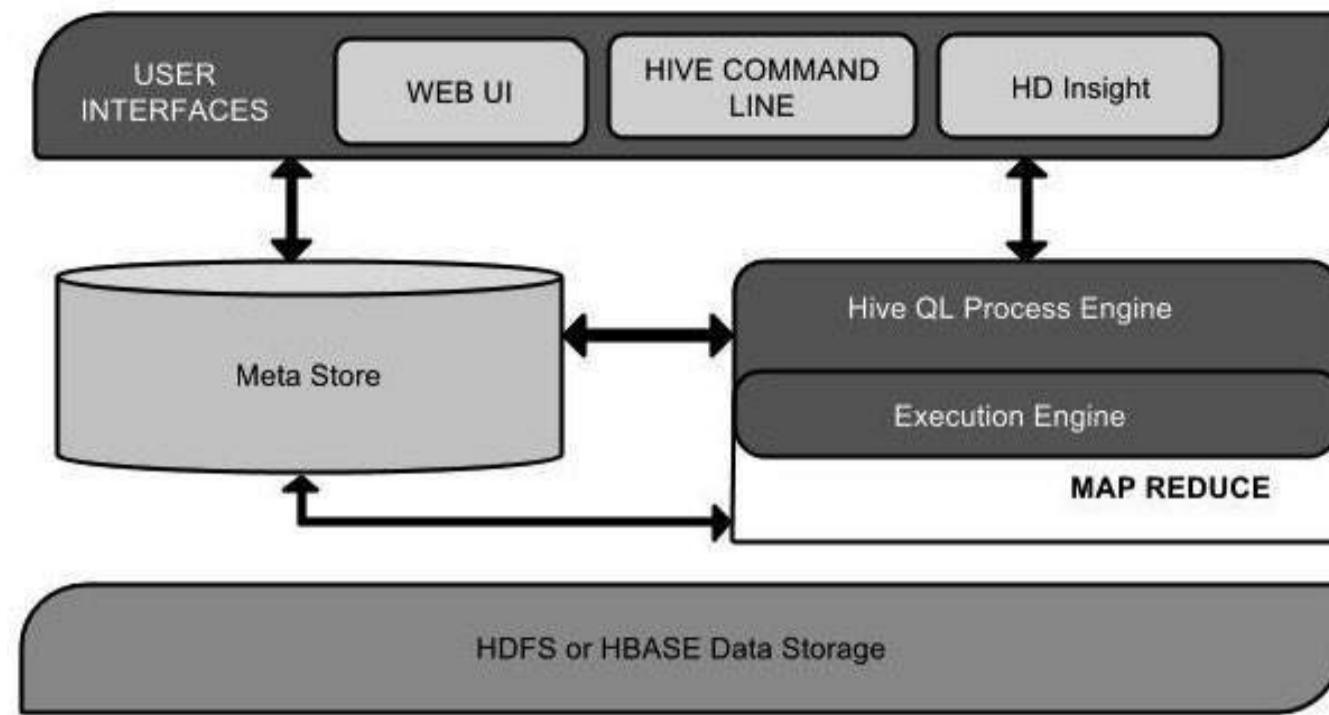
Hive est une plateforme open source construite au-dessus de Hadoop, conçue pour faciliter le traitement et l'analyse de grands ensembles de données en utilisant une syntaxe similaire au langage SQL. Il permet aux utilisateurs de traiter les données stockées dans Hadoop sans avoir à écrire des programmes complexes.

Hive est une infrastructure de data warehouse et un langage de requête construit sur Hadoop. Il offre une abstraction de haut niveau et un langage similaire à SQL appelé HiveQL pour interroger et analyser de grands ensembles de données stockés dans le Distributed File System (HDFS) de Hadoop. Hive fait partie de l'écosystème Hadoop et est conçu pour simplifier l'interaction avec de gros volumes de données sans nécessiter une connaissance approfondie de langages de programmation complexes.



Hive Architecture & Its Components

Architecture de Hive :



1. Métastore Hive :

- **Aperçu :** Le Métastore Hive stocke les métadonnées, telles que les schémas de table, les types de données, et les emplacements physiques des données.
- **Fonctionnalités :**
 - **Stockage des Métadonnées** : Contient des informations sur la structure et l'emplacement des données.
 - **Interopérabilité** : Facilite l'interopérabilité avec d'autres outils et applications.

2. Hive Query Processor :

- **Aperçu :** Le Hive Query Processor analyse les requêtes Hive, les optimise, puis les traduit en un ensemble de tâches MapReduce.
- **Fonctionnalités :**
 - **Analyse et Optimisation** : Analyse les requêtes pour générer des plans d'exécution optimisés.
 - **Traduction en MapReduce** : Convertit les requêtes en tâches MapReduce pour le traitement distribué.

3. Hive Execution Engine :

- **Aperçu :** Le moteur d'exécution Hive prend en charge l'exécution des tâches MapReduce générées à partir des requêtes Hive.
- **Fonctionnalités :**
 - **Exécution Distribuée** : Gère la distribution des tâches sur le cluster Hadoop.
 - **Intégration avec Hadoop** : S'intègre nativement avec le framework Hadoop.

Modules Fonctionnels :

1. HiveQL :

- **Aperçu :** Hive Query Language (HiveQL) est un langage de requête similaire à SQL utilisé pour interagir avec Hive.
- **Fonctionnalités :** Permet aux utilisateurs d'écrire des requêtes SQL-like pour traiter et analyser les données stockées dans Hadoop.

2. Hive Thrift Server :

- **Aperçu :** Le serveur Hive Thrift permet aux applications externes d'exécuter des requêtes Hive via une interface basée sur Thrift.
- **Fonctionnalités :** Fournit une interface programmatique pour l'accès aux fonctionnalités de Hive à partir d'autres applications.

3. Hive Drivers :

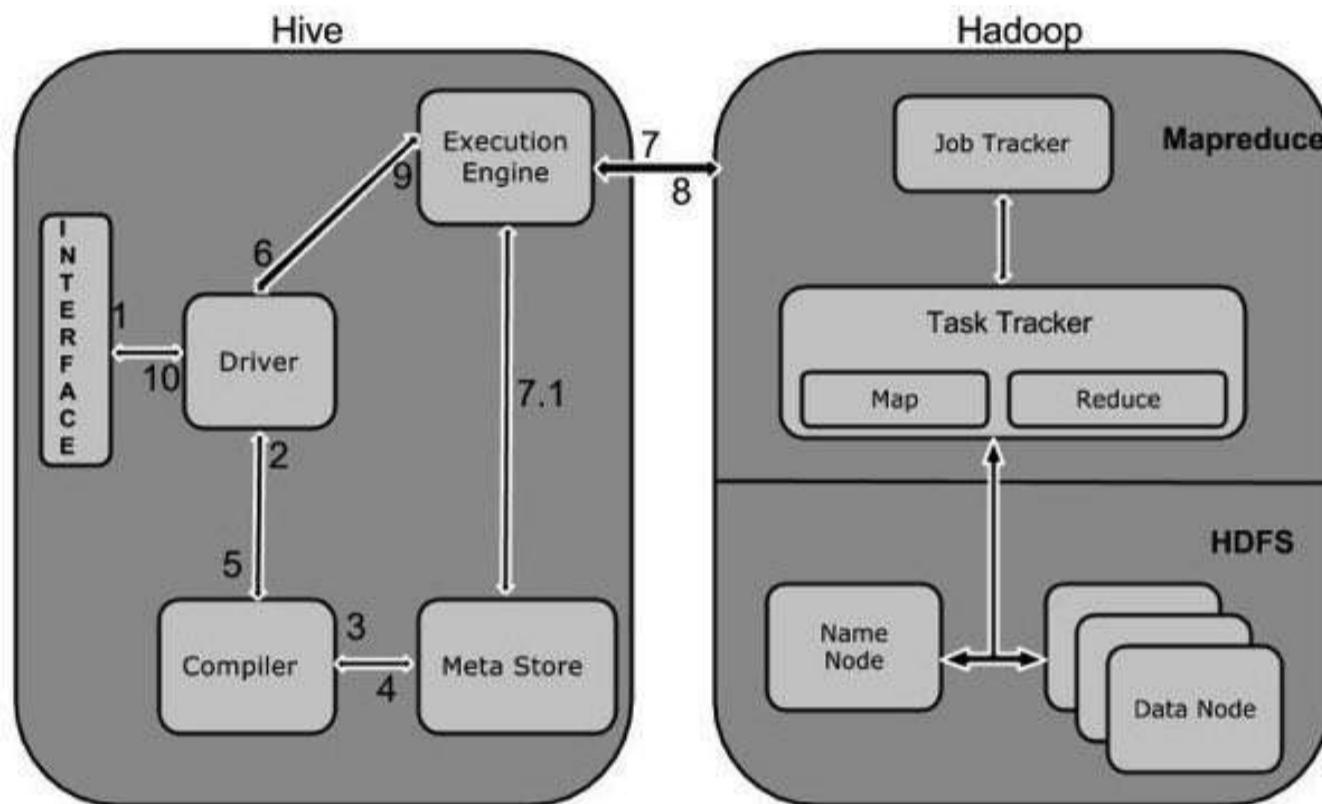
- **Aperçu :** Les pilotes Hive permettent la connexion et l'interaction avec Hive à partir de différents clients et applications.

- **Fonctionnalités** : Facilite la communication entre les applications et le serveur Hive.

4. Hive UDF (User-Defined Functions) :

- **Aperçu** : Les fonctions définies par l'utilisateur (UDF) permettent aux utilisateurs d'étendre les fonctionnalités de Hive en écrivant leurs propres fonctions.
- **Fonctionnalités** : Offre une flexibilité pour créer des fonctions personnalisées répondant aux besoins spécifiques des utilisateurs.

Fonctionnement de Hive :



- **HiveQL** : HiveQL permet aux utilisateurs d'écrire des requêtes SQL-like pour interroger et analyser les données stockées dans Hadoop.
- **Traitement Distribué** : Hive utilise le framework Hadoop pour le traitement distribué, divisant les tâches en tâches MapReduce exécutées sur le cluster.
- **Optimisation des Requêtes** : Le Hive Query Processor analyse et optimise les requêtes pour améliorer les performances d'exécution.
- **Interopérabilité** : Hive offre une interopérabilité avec d'autres outils Hadoop, ce qui facilite l'intégration dans l'écosystème Hadoop.

Hive simplifie l'analyse de données dans l'écosystème Hadoop en offrant une interface SQL-like, permettant aux utilisateurs d'exploiter les capacités de traitement distribué pour les données stockées dans Hadoop.

▼ Hive commandes

0. Base de données :

1. Crédation d'une base de données Hive :

```
-- Crédation d'une base de données Hive

CREATE DATABASE IF NOT EXISTS mydatabase
COMMENT 'Une base de données Hive d'exemple'
WITH DBPROPERTIES (
    'proprietaire' = 'John Doe',
    'date_creation' = '2024-02-02'
)
LOCATION '/user/hive/warehouse/mydatabase.db';
```

Cet exemple crée une base de données Hive nommée `mydatabase` avec des paramètres optionnels tels qu'un commentaire et des propriétés de base de données. La clause `LOCATION` spécifie l'emplacement HDFS où la base de données stockera ses données.

1. Suppression d'une base de données Hive :

```
-- Suppression d'une base de données Hive  
  
DROP DATABASE IF EXISTS mydatabase CASCADE;
```

Cet exemple supprime la base de données Hive `mydatabase`. L'option `CASCADE` est utilisée pour supprimer la base de données ainsi que toutes ses tables et données. Utilisez cette option avec précaution, car elle supprime définitivement la base de données et son contenu.

1. Modification d'une base de données Hive :

```
-- Modification d'une base de données Hive  
  
ALTER DATABASE mydatabase  
SET DBPROPERTIES ('nouvelle_propriete' = 'nouvelle_valeur');
```

Cet exemple modifie la base de données Hive `mydatabase` en ajoutant ou en modifiant une propriété de base de données. Dans ce cas, il définit une nouvelle propriété appelée `nouvelle_propriete` avec la valeur `nouvelle_valeur`. Vous pouvez utiliser `SET` pour modifier des propriétés existantes ou en ajouter de nouvelles.

N'oubliez pas de remplacer le nom de la base de données, les propriétés et les valeurs par vos besoins spécifiques. Ces exemples illustrent la syntaxe de base pour la création, la suppression et la modification d'une base de données Hive.

1. Création de Tables dans Hive :

La création de tables dans Hive se fait avec la syntaxe suivante, avec des options pour les tables internes et externes :

```
CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [nom_base.] nom_table  
  
[(nom_colonne type_colonne [COMMENTAIRE col_commentaire], ...)]  
[COMMENTAIRE table_commentaire]  
[ROW FORMAT format_ligne]  
[STORED AS format_fichier]
```

Dans Hive, on peut créer deux types de tables : les tables internes et les tables externes.

Table Interne :

Une **table interne** est une **table gérée par Hive**, ce qui signifie que Hive est responsable de la gestion de la table ainsi que de ses données. Lorsque vous créez une table interne, Hive stocke les données dans son propre répertoire dans HDFS. Si vous supprimez la table, Hive supprime également les données associées.

Exemple :

```
-- Créer une table interne  
CREATE TABLE table_interne (  
    colonne1 type1,  
    colonne2 type2,  
    ...  
)
```

Dans cet exemple, `table_interne` est une table interne avec les colonnes spécifiées et les types de données associés. Les données de cette table seront stockées dans le répertoire géré par Hive dans HDFS.

Table Externe :

Une **table externe** est une **table où les données sont stockées à l'extérieur du répertoire géré par Hive dans HDFS**. Cela signifie que Hive n'est pas responsable de la gestion des données, et si vous supprimez la table externe, les données sous-jacentes restent dans le système de fichiers. Vous spécifiez l'emplacement des données lors de la création de la table externe.

Exemple :

```
-- Créer une table externe
CREATE EXTERNAL TABLE table_externe (
    colonne1 type1,
    colonne2 type2,
    ...
)
LOCATION 'chemin_hdfs';
```

Ici, `table_externe` est une table externe avec les colonnes spécifiées et les types de données associés. L'emplacement `'chemin_hdfs'` indique à Hive où trouver les données dans HDFS. Si vous supprimez cette table, les données resteront dans le chemin spécifié. Les tables externes sont utiles lorsque vous avez besoin de partager des données entre plusieurs outils ou lorsque vous ne voulez pas que Hive gère la suppression des données.

```
CREATE EXTERNAL TABLE IF NOT EXISTS mydb.mytable (
    column1 INT,
    column2 STRING,
    column3 DOUBLE
)
COMMENT 'A sample external table'
PARTITIONED BY (partition_column STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n'
STORED AS TEXTFILE
LOCATION '/user/hive/warehouse/mydab/mytable'
TBLPROPERTIES (
    'created_by' = 'John Doe',
    'created_on' = '2024-02-02',
    'file_format' = 'text'
);
```

2. Spécifier les Types de Données dans Hive :

Hive prend en charge divers types de données, y compris les types primitifs et les types complexes. Voici des exemples :

Types primitifs :

```
CREATE TABLE table_exemple (
    colonne_entier INT,
    colonne_chaine STRING,
    colonne_flottant FLOAT,
    colonne_booleen BOOLEAN
);
```

Types complexes :

```
CREATE TABLE table_complexe (
    colonne_tableau ARRAY<INT>,
    colonne_map MAP<STRING, INT>,
    colonne_structure STRUCT<champ1: INT, champ2: STRING>
);
```

3. Partitionnement dans Hive :

Le partitionnement dans Hive est essentiel pour optimiser les performances des requêtes. Il consiste à diviser une table en parties plus petites et gérables en fonction de certains critères.

```
-- Créer une table partitionnée
CREATE TABLE table_partitionnee (
    colonne1 type1,
    colonne2 type2,
    ...
)
PARTITIONED BY (colonne_partition type);

-- Insérer des données dans une partition spécifique
INSERT OVERWRITE TABLE table_partitionnee PARTITION (colonne_partition=valeur)
SELECT * FROM table_source;
```

Partitionnement de Table dans Hive

Supposons que vous ayez une table appelée `mytable` avec une colonne nommée `partition_column` que vous souhaitez utiliser pour le partitionnement. Voici comment vous pouvez créer une table partitionnée :

```
-- Création d'une table partitionnée dans Hive

CREATE TABLE mydatabase.mytable (
    column1 INT,
    column2 STRING,
    partition_column STRING
)
PARTITIONED BY (partition_column STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\\\\t'
LINES TERMINATED BY '\\\\n'
STORED AS TEXTFILE
LOCATION '/user/hive/warehouse/mydatabase.db/mytable';
```

Cela crée une table avec une colonne de partition nommée `partition_column`. Maintenant, supposons que vous ayez un fichier volumineux et que vous souhaitez charger les données dans cette table et les partitionner en fonction des valeurs de `partition_column`. Vous pouvez utiliser la commande suivante :

```
-- Chargement des données dans la table partitionnée
LOAD DATA LOCAL INPATH '/chemin/vers/votre/bigfile.txt' INTO TABLE mydatabase.mytable;
```

Après le chargement des données, Hive créera automatiquement des partitions en fonction des valeurs distinctes de `partition_column`. Par exemple, si `partition_column` a des valeurs telles que 'valeur1', 'valeur2' et 'valeur3', Hive créera des répertoires séparés pour chaque partition :

```
/user/hive/warehouse/mydatabase.db/mytable/partition_column=valeur1
/user/hive/warehouse/mydatabase.db/mytable/partition_column=valeur2
/user/hive/warehouse/mydatabase.db/mytable/partition_column=valeur3
```

De cette manière, les données sont physiquement stockées dans des répertoires séparés en fonction des valeurs de la colonne de partition, facilitant ainsi la requête et la gestion de sous-ensembles spécifiques des données.

Il est important de noter que le partitionnement est une fonctionnalité puissante, mais le choix de la colonne de partitionnement doit être basé sur votre cas d'utilisation spécifique et les modèles de requête.

4. Chargement de Données dans Hive (INSERT | LOAD DATA) :

Dans Hive, le chargement de données dans des tables à partir de sources externes, telles que des fichiers dans HDFS, peut être effectué avec les commandes `INSERT` ou `LOAD DATA`. Voici des exemples :

```
-- Charger des données dans une table existante depuis HDFS
LOAD DATA INPATH 'chemin_hdfs' INTO TABLE table_name;
```

```
-- Charger des données dans une table existante depuis HDFS/Locale
LOAD DATA [LOCAL] INPATH 'chemin_fichier' [OVERWRITE] INTO TABLE table_name
[PARTITION (partcol1=val1, partcol2=val2 ...)]

-- Charger des données dans une table existante depuis commande insert
INSERT INTO TABLE table_name VALUES (valeur1, valeur2, ...);
```

5. Opérations de Manipulation de Données :

Les opérations de manipulation de données telles que `INSERT`, `UPDATE`, `DELETE`, et `SELECT` en `HiveQL` sont similaires à celles de SQL standard.

- **INSERT** : Insère des données dans une table. Les exemples ci-dessous illustrent différentes méthodes d'insertion :

```
-- Insertion de valeurs dans une table
INSERT INTO TABLE table_interne VALUES (valeur1, valeur2, ...);

-- Insertion à partir d'une requête SELECT
INSERT INTO TABLE table_interne SELECT * FROM autre_table;
```

- **UPDATE** : Met à jour les valeurs d'une table. Hive ne prend en charge que la mise à jour des tables gérées par ACID.

```
-- Exemple d'utilisation de la clause UPDATE (pour les tables gérées par ACID)
UPDATE table_geree_par_acid SET colonne1 = nouvelle_valeur WHERE condition;
```

- **DELETE** : Supprime des lignes d'une table. Comme pour la mise à jour, Hive ne prend en charge que la suppression des tables gérées par ACID.

```
-- Exemple d'utilisation de la clause DELETE (pour les tables gérées par ACID)
DELETE FROM table_geree_par_acid WHERE condition;
```

- **SELECT** : Sélectionne des données à partir d'une table. Utilisez la clause `SELECT` pour récupérer des informations.

```
-- Exemple de requête SELECT
SELECT colonne1, colonne2 FROM table_name WHERE condition;
```

- **DROP** : Supprime une table ou une base de données. Attention, cela supprime également les données si la table est gérée par Hive.

```
-- Exemple de suppression de table
DROP TABLE table_name;
```

Ces opérations vous permettent de manipuler les données dans Hive de manière similaire à SQL standard. Veillez à comprendre les implications de chaque opération, notamment en termes de gestion des données dans un environnement distribué comme Hive.

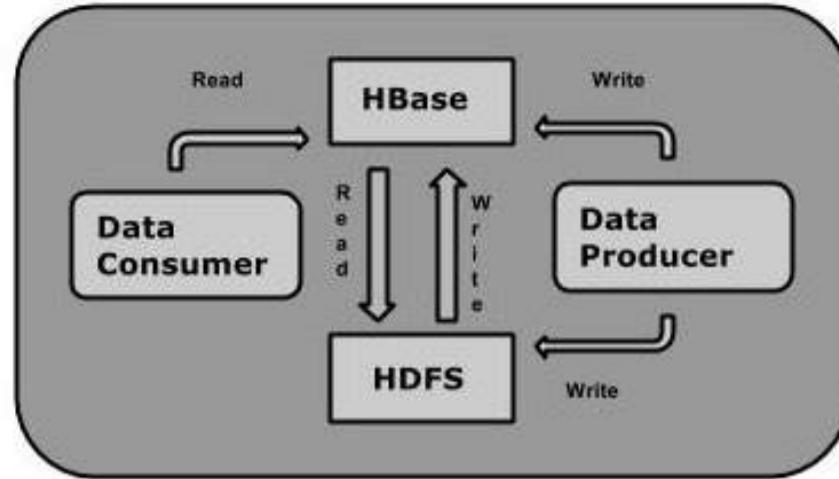
Ces principes sont en accord avec les pratiques courantes de SQL, mais adaptés aux capacités de traitement distribué et de big data de Hive. Gardez à l'esprit que HiveQL est insensible à la casse et que vous travaillerez souvent avec Hive via son interface en ligne de commande ou des scripts.

▼ HBase

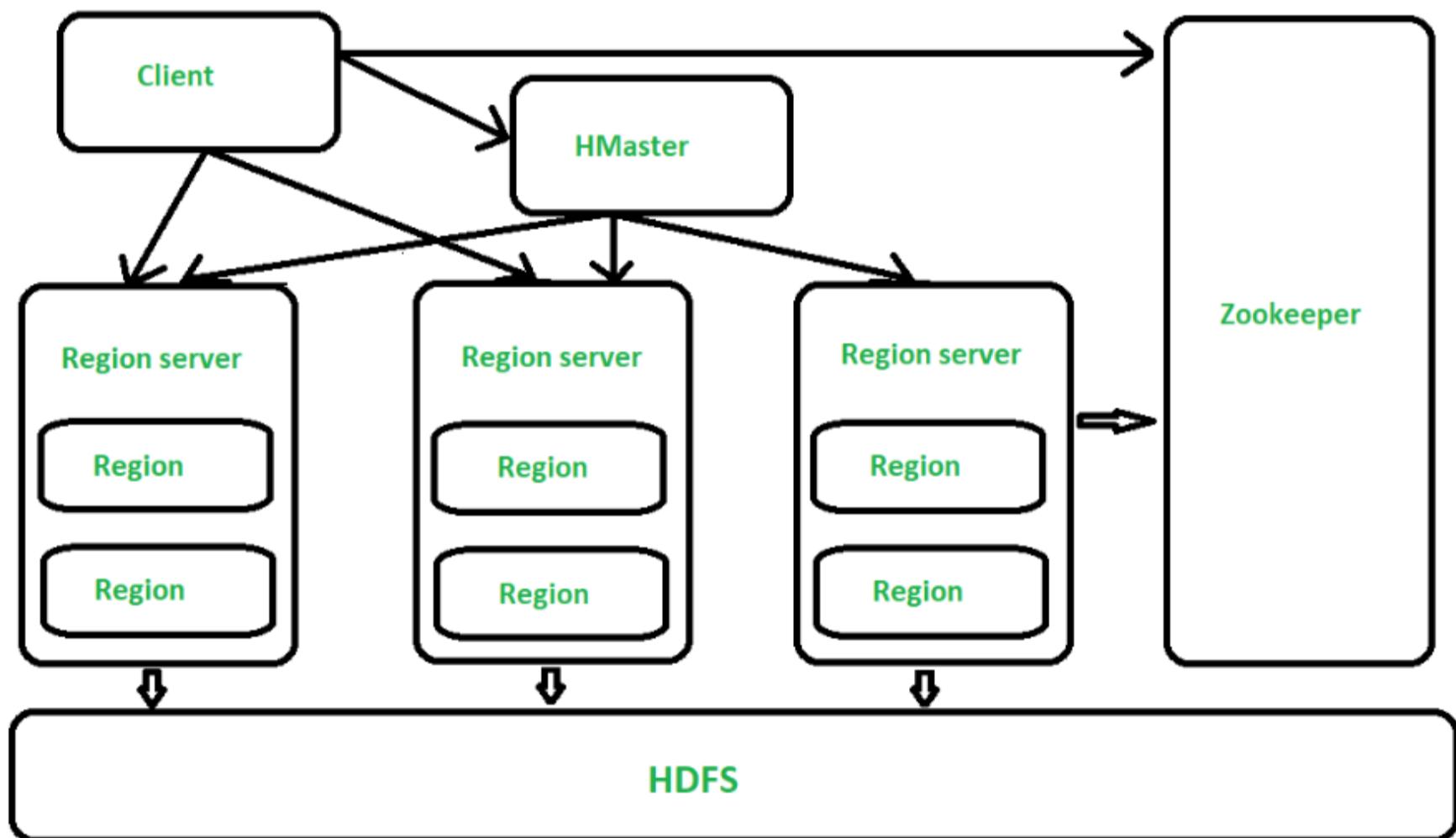
Présentation Générale de HBase :

HBase est une base de données NoSQL (Not Only SQL) construite sur Hadoop qui offre une capacité de stockage et de traitement distribuée pour des volumes massifs de données. Conçue pour être évolutive et hautement disponible, HBase est adaptée aux applications nécessitant un accès aléatoire rapide à de vastes ensembles de données.

HBase stocke ses données dans HDFS sous forme de HFiles, fournissant une solution de stockage distribuée et tolérante aux pannes pour le traitement des données en temps réel à grande échelle.



Architecture de HBase :



1. HBase Region Server :

- **Aperçu :** Les Region Servers stockent et gèrent les données HBase. Chaque Region Server gère un certain nombre de régions.
- **Fonctionnalités :**
 - **Stockage des Données** : Responsable du stockage des données dans les régions.
 - **Gestion des Régions** : Gère la division et la fusion automatiques des régions.

2. HBase Master Server :

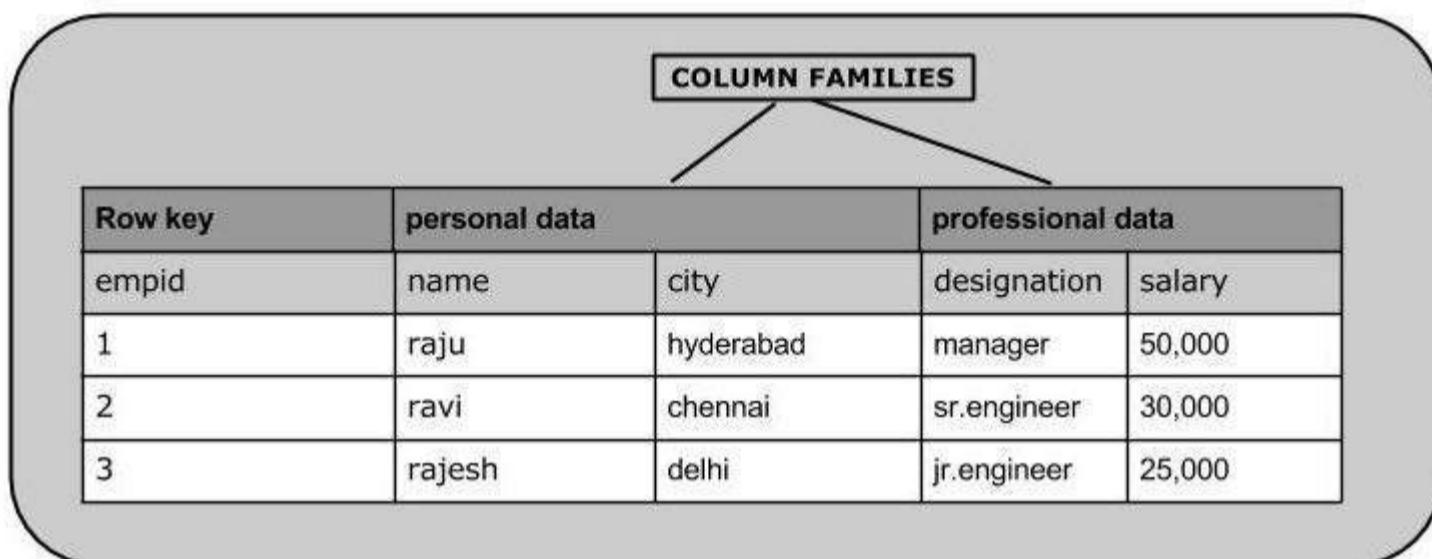
- **Aperçu :** Le Master Server coordonne et supervise les opérations HBase, y compris la gestion des régions.
- **Fonctionnalités :**
 - **Allocation des Régions** : Alloue les régions aux Region Servers.
 - **Gestion Globale** : Surveille l'état global du cluster HBase.

3. HBase ZooKeeper :

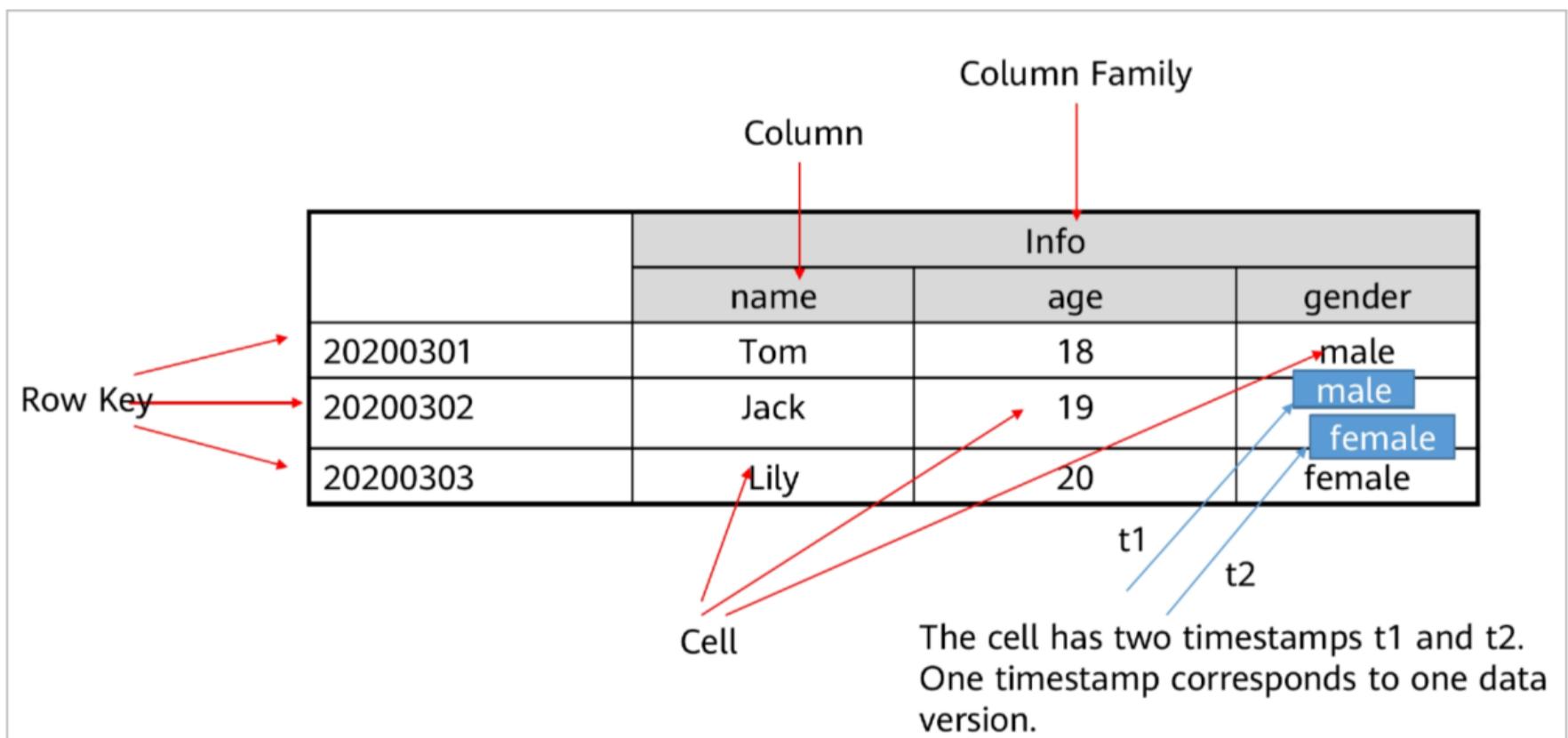
- **Aperçu :** ZooKeeper est un service de coordination utilisé par HBase pour la gestion des membres du cluster et la détection de pannes.
- **Fonctionnalités :**
 - **Coordination :** Assure la coordination entre les différents composants HBase.
 - **Détection de Pannes :** Déetecte les pannes et facilite la reprise en cas de défaillance.

Modules Fonctionnels :

1. HBase Data Model :



- **Aperçu :** Le modèle de données HBase est basé sur une structure de table avec des rangées, **des colonnes et des familles de colonnes**.
- **Fonctionnalités :** Permet un stockage flexible des données avec une prise en charge de colonnes dynamiques.



2. HBase Shell :

- **Aperçu :** La HBase Shell est une interface en ligne de commande permettant aux utilisateurs d'interagir avec HBase.
- **Fonctionnalités :** Facilite la création de tables, l'insertion de données et l'exécution de requêtes.

3. HBase Filters :

- **Aperçu :** Les filtres HBase permettent de restreindre les résultats des requêtes en fonction de critères spécifiques.

- **Fonctionnalités** : Offre une flexibilité pour récupérer des données basées sur des conditions prédéfinies.

4. HBase Coprocessors :

- **Aperçu** : Les coprocesseurs HBase permettent l'exécution de logique personnalisée au sein des serveurs HBase.
- **Fonctionnalités** : Étend les fonctionnalités de HBase en permettant aux utilisateurs d'ajouter leur propre logique de traitement.

Fonctionnement de HBase :

- **Stockage Distribué** : HBase distribue les données sur plusieurs Region Servers, assurant un stockage et un accès rapides à des volumes massifs de données.
- **Gestion Automatique des Régions** : Les Region Servers gèrent automatiquement la division et la fusion des régions pour assurer l'évolutivité.
- **Coordination avec ZooKeeper** : ZooKeeper assure la coordination entre les composants HBase, facilitant la gestion des membres du cluster et la détection des pannes.
- **Modèle de Données Flexible** : Le modèle de données HBase permet un stockage flexible des données avec des colonnes dynamiques, adapté à divers types de données.

HBase est une solution robuste pour le stockage et le traitement distribué de grands volumes de données avec un accès aléatoire rapide, ce qui en fait un choix idéal pour les applications nécessitant une extensibilité horizontale et une disponibilité élevée.

▼ HBase commandes

	Row Key	Prénom	Personnel	Age	téléphone	Contact	ville
00001	Juvénal <i>timestamp:</i> 10/09/2011 13:05:17:09		22 <i>timestamp:</i> 10/09/2011 13:05:17:09		06 90 98 76 52 <i>timestamp:</i> 10/09/2011 13:05:17:09	Douala <i>timestamp:</i> 10/09/2011 13:05:17:09	
			25 <i>timestamp:</i> 20/09/2013 15:00:05:09			Lille <i>timestamp:</i> 20/09/2013 15:00:05:09	
						Paris <i>timestamp:</i> 30/10/2016 16:18:50:10	
00002	Paul <i>timestamp:</i> 10/09/2011 13:10:05:09		30 <i>timestamp:</i> 10/09/2011 13:10:05:09		07 90 94 86 52 <i>timestamp:</i> 10/09/2011 13:10:05:09	Nancy <i>timestamp:</i> 10/09/2011 13:10:05:09	
00003	Jean <i>timestamp:</i> 12/09/2011 11:30:20:09		34 <i>timestamp:</i> 12/09/2011 11:30:20:09		06 74 98 76 25 <i>timestamp:</i> 12/09/2011 11:30:20:09	Marseille <i>timestamp:</i> 12/09/2011 11:30:20:09	

Voici quelques-unes des commandes couramment utilisées dans HBase pour effectuer diverses opérations, telles que la création de tables, l'insertion de données, la recherche, la suppression, etc.

1. Créer une Table :

en peut créer initialement le tableau avec "family columns"

```
create 'nom_table', 'famille_de_colonnes'

create 'emp', 'personal data', 'professional data'
```

ou créer "family columns" avec "columns"

```
create 'employee',
{NAME => 'personal_data', VERSIONS => 3},
{NAME => 'professional_data', VERSIONS => 3},
'personal_data:name', 'personal_data:address', 'professional_data:salary',
'professional_data:position'
```

L'attribut `VERSIONS` d'une famille de colonnes spécifie le nombre de versions d'une cellule (données associées à une colonne et timestamp) **`VERSIONS = 1 (default)`**

2. Insérer des Données :

```
put 'nom_table', 'row_key', 'famille_de_colonnes:colonne', 'valeur'  
put 'emp', 'row1', 'personal data:name', 'John Doe'
```

3. Lire des Données :

```
get 'nom_table', 'row_key'  
get 'emp', 'row1'
```

4. Scanner la Table (`selecte * from table_name`):

```
scan 'nom_table'  
scan 'emp'
```

5. Scanner avec Filtre (`select * from emp where personal data → name = 'John Doe'`) :

```
scan 'nom_table', {FILTER => "SingleColumnValueFilter('famille_de_colonnes', 'colonne',  
=, 'valeur')"}  
scan 'emp', {FILTER => "SingleColumnValueFilter('personal data', 'name', =, 'John Doe')"}  
This command scans the 'emp' table applying a filter to show only the rows where the 'name' column of the 'personal data' family is equal to 'John Doe'.
```

6. Supprimer une Ligne :

```
delete 'nom_table', 'row_key'  
delete 'emp', 'row1'
```

7. Supprimer une Colonne :

```
delete 'nom_table', 'row_key', 'famille_de_colonnes:colonne'  
delete 'emp', 'row1', 'personal data:name'
```

Cette commande supprime la colonne 'name' de la famille de colonnes 'personal data' pour la ligne avec la clé 'row1' dans la table 'emp'.

8. Supprimer une Table :

```
drop 'nom_table'  
drop 'emp'
```

9. Lister les Tables :

```
hbase(main):001:0> list
```

```
<TABLE_NAMES>
```

```
...
```

10. Décrire une Table :

```
describe 'nom_table'
```

Cette commande affiche la description de la structure de la table 'emp'.

11. Compter les Lignes d'une Table :

```
count 'nom_table'
```

```
count 'emp'
```

12. Ajouter une Nouvelle Famille de Colonnes à une Table Existante :

```
alter 'nom_table', {NAME => 'nouvelle_famille_de_colonnes'}
```

```
alter 'emp', {NAME => 'contact_data'}
```

13. Supprimer une Famille de Colonnes d'une Table Existante :

```
alter 'nom_table', 'delete' => 'nom_famille_de_colonnes'
```

```
alter 'emp', 'delete' => 'professional_data'
```

14. Modifier le TTL (Time-To-Live) d'une Colonne :

```
alter 'nom_table', {NAME => 'famille_de_colonnes:colonne', TTL => ttl_en_seconds}
```

```
alter 'emp', {NAME => 'personal data:name', TTL => 2592000}
```

Cette commande modifie le TTL de la colonne 'name' de la famille de colonnes 'personal data' à 2592000 secondes (30 jours).

15. Modifier le TTL de la Table :

```
alter 'nom_table', {TTL => ttl_en_secondes}
```

```
alter 'emp', {TTL => 604800}
```

Cette commande modifie le TTL de la table 'emp' à 604800 secondes (7 jours).

16. Désactiver une Table :

```
disable 'nom_table'
```

```
disable 'emp'
```

Cette commande désactive la table spécifiée. Lorsqu'une table est **désactivée**, elle **ne peut pas être modifiée ou lue**. Cela peut être utile lors de la réalisation de maintenance ou de modifications structurelles sur la table.

17. Activer une Table :

```
enable 'nom_table'
```

```
enable 'emp'
```

Cette commande *active* une table qui a été précédemment désactivée. Une fois **activée, la table redevient disponible pour les opérations de lecture et de modification.**

Ces commandes de base vous permettront de commencer à travailler avec HBase. Assurez-vous de personnaliser les noms de table, les familles de colonnes, les colonnes, les valeurs, etc., en fonction de votre configuration spécifique. N'oubliez pas que les commandes HBase sont généralement exécutées dans l'interface de ligne de commande HBase (shell HBase).

▼ Spark

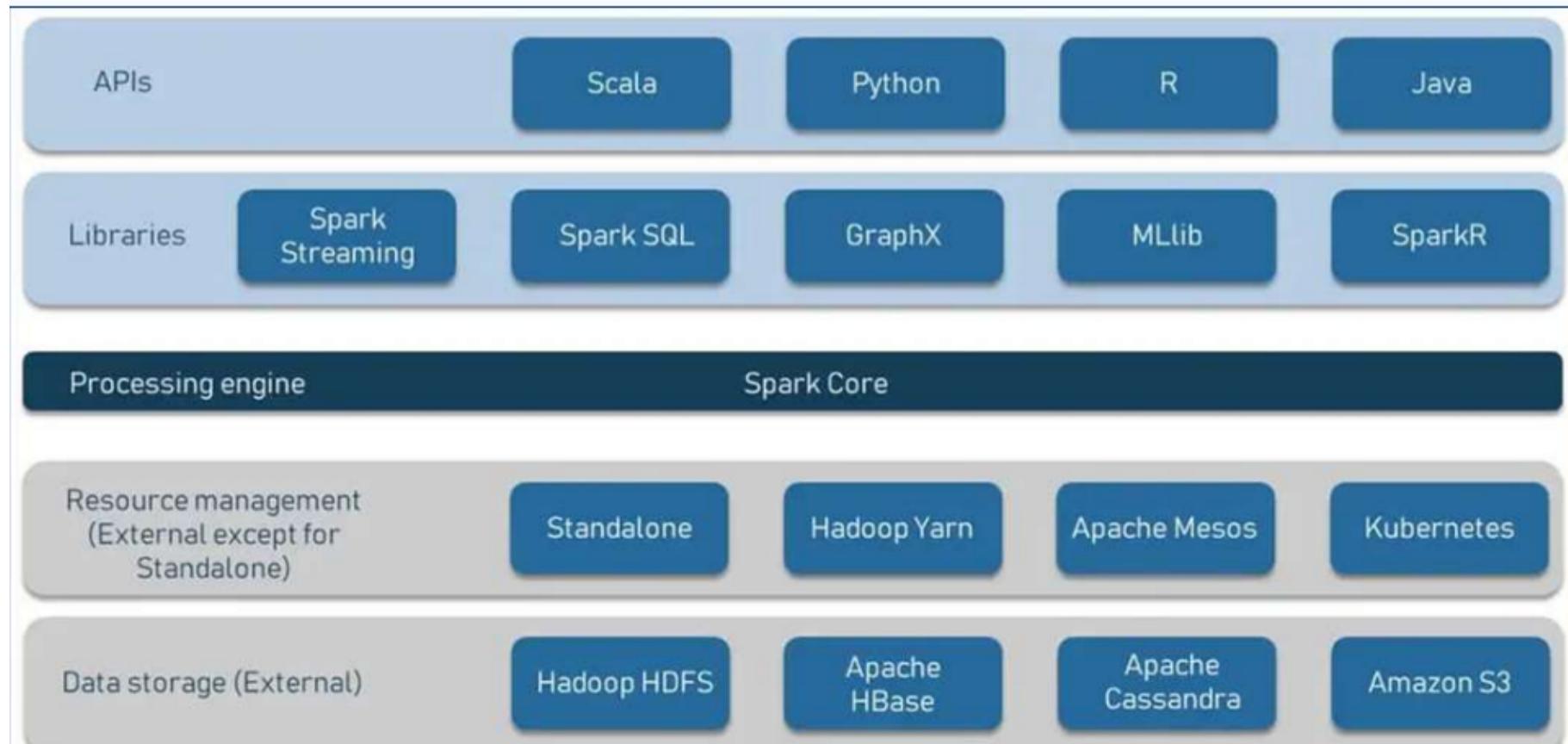
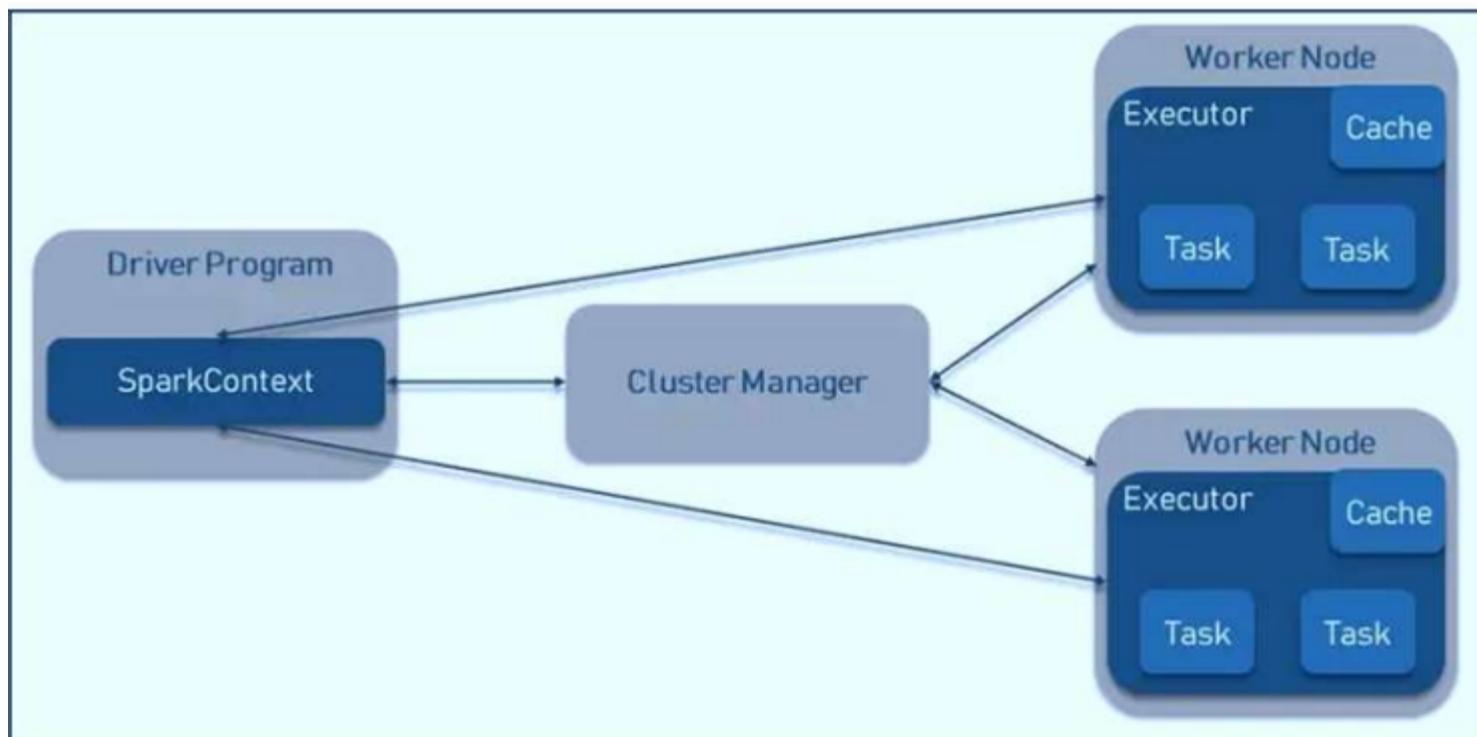
Présentation Générale de Apache Spark :

Apache Spark est un framework de traitement de données open source, rapide et polyvalent, conçu pour l'analyse de données distribuée. Il offre une interface unifiée pour le traitement de données en batch, interactif, et en temps réel, avec des performances optimisées grâce à son architecture de traitement en mémoire.

L'idée clé de Spark est celle des ensembles de données distribués résilients (RDD) ; il prend en charge le calcul du traitement en mémoire. Cela signifie qu'il stocke l'état de la mémoire en tant qu'objet entre les tâches et que l'objet est partageable entre ces tâches. Le partage de données en mémoire est 10 à 100 fois plus rapide que le réseau et Disque

Si la mémoire distribuée (RAM) n'est pas suffisante pour stocker les résultats intermédiaires (état du JOB), alors elle stockera ces résultats sur le disque.

Architecture de Apache Spark :



1. Spark Core :

- **Aperçu :** Le cœur de Spark, Spark Core, fournit les fonctionnalités fondamentales et les API pour le traitement distribué des données.
- **Fonctionnalités :**
 - **Abstraction de Données RDD** : Resilient Distributed Datasets (RDD) représentent la principale abstraction de données en Spark.
 - **Planification de Tâches** : La planification des tâches distribue les opérations sur les RDD sur un cluster.

2. Spark SQL :

- **Aperçu :** Spark SQL permet le traitement de données structurées en utilisant le langage SQL au-dessus de Spark.
- **Fonctionnalités :**
 - **Interrogation SQL** : Permet d'exécuter des requêtes SQL sur les données distribuées.
 - **Intégration avec RDD** : Offre une intégration transparente avec les RDD de Spark.

3. Spark Streaming :



- **Aperçu :** Spark Streaming permet le traitement en temps réel des données, avec des fenêtres temporelles pour l'analyse continue.
- **Fonctionnalités :**
 - **Traitements en Continu** : Traite les flux de données en temps réel en utilisant des fenêtres temporelles.
 - **Intégration avec Spark Core** : S'intègre de manière transparente avec les fonctionnalités de Spark Core.

4. Spark MLlib :

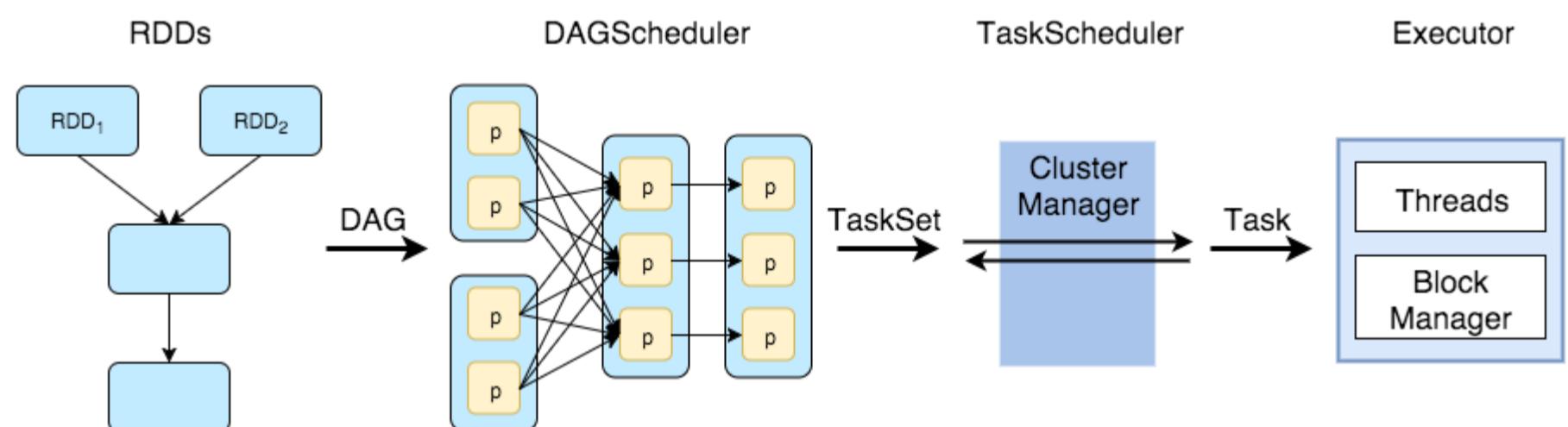
Machine Learning Algorithms in Apache Spark

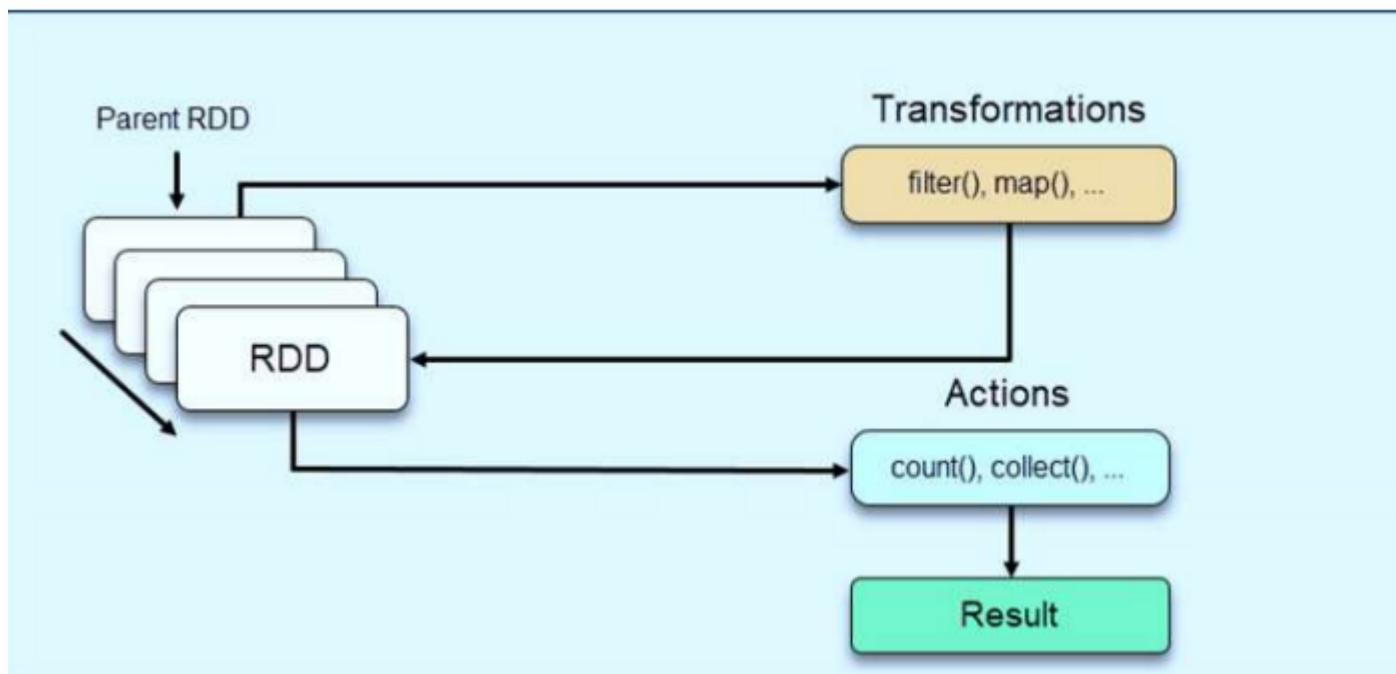


- **Aperçu :** Spark MLlib est la bibliothèque de machine learning de Spark, offrant des outils pour la construction de modèles de machine learning distribués.
- **Fonctionnalités :**
 - **Algorithmes ML Distribués** : Fournit des algorithmes de machine learning distribués pour l'analyse de données distribuée.
 - **Intégration avec Spark Core** : Intégration avec le cœur de Spark pour le traitement distribué.

Modules Fonctionnels :

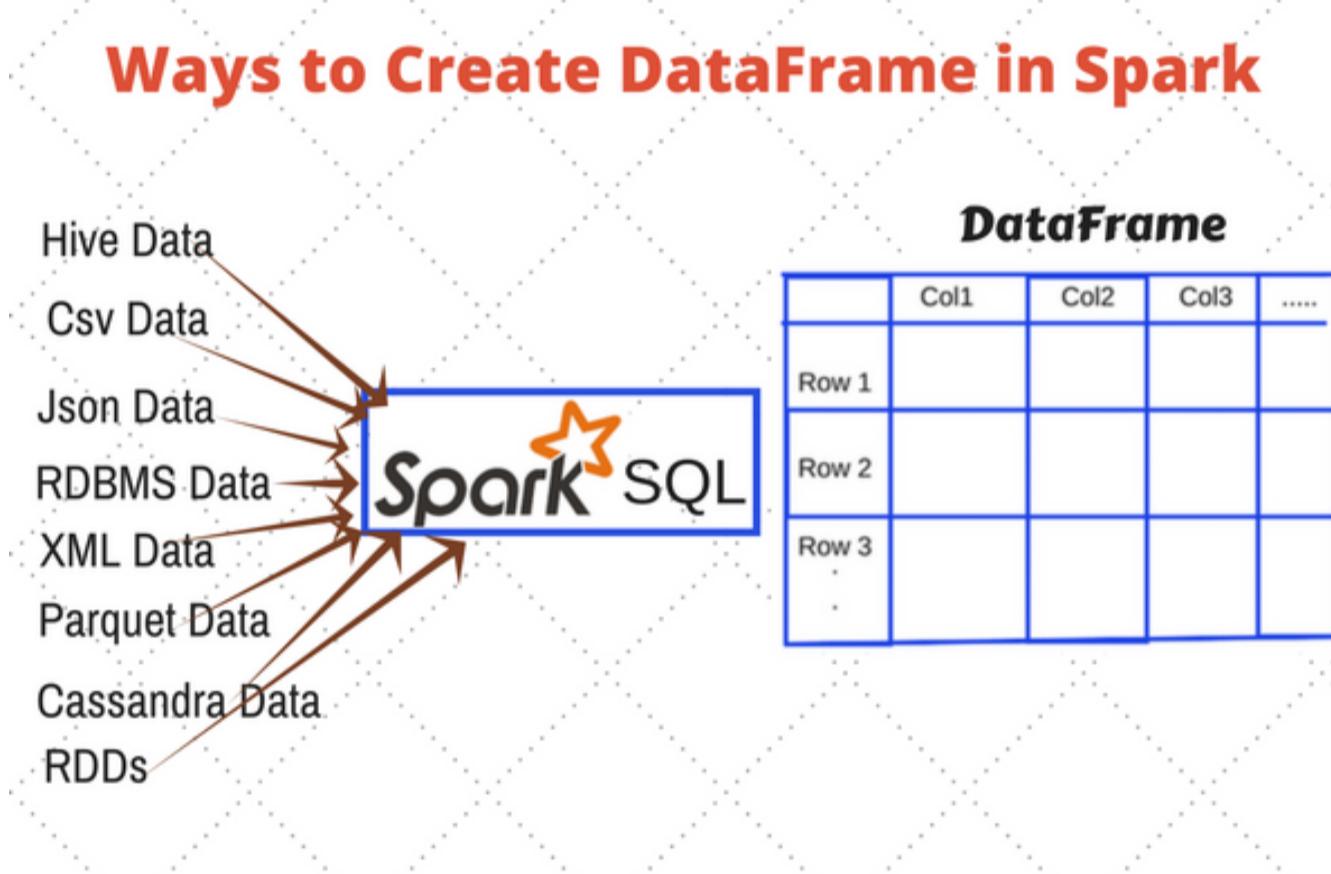
1. Spark RDD :





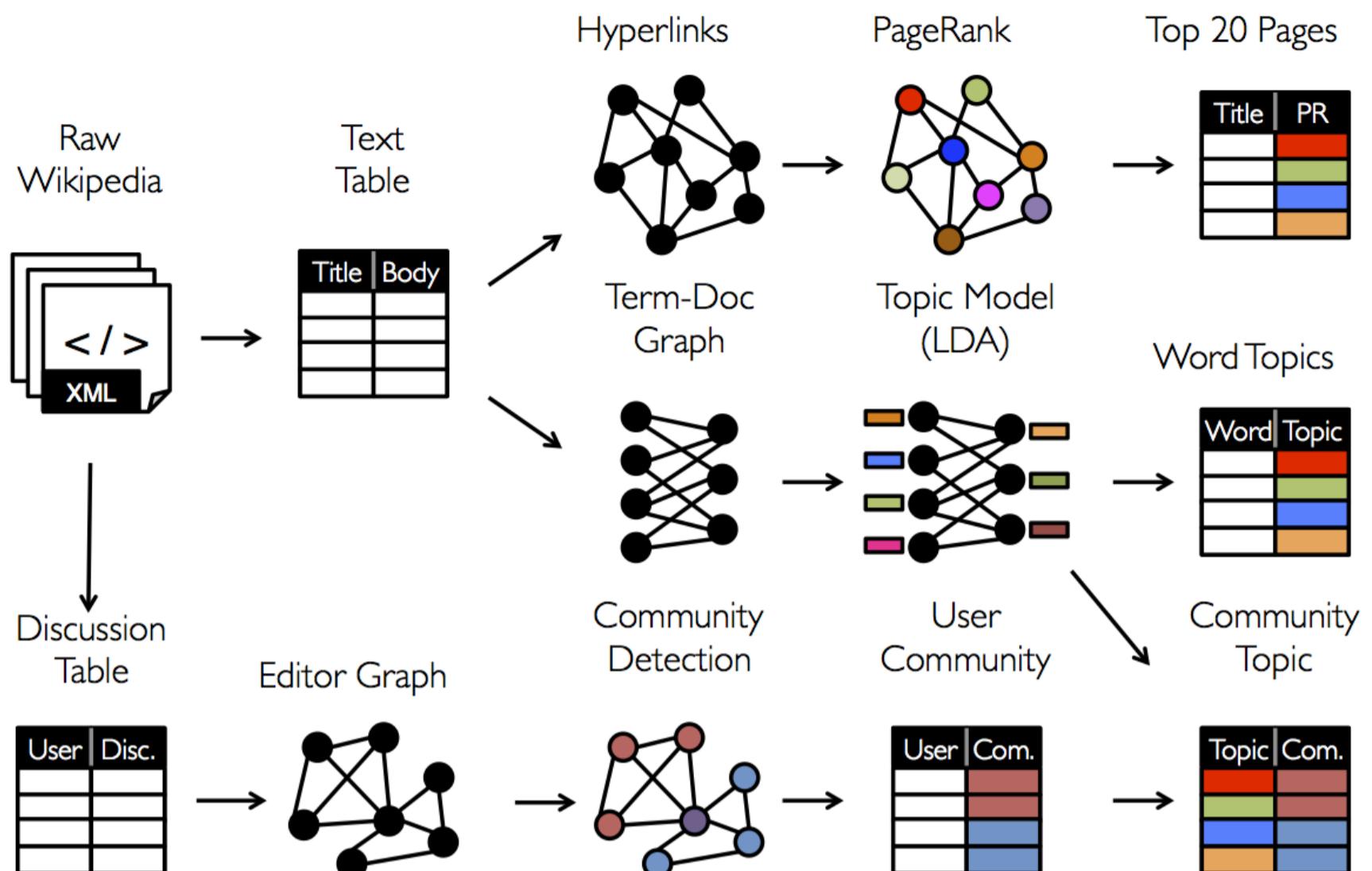
- Aperçu :** Les Resilient Distributed Datasets (RDD) sont la principale abstraction de données en Spark, offrant une résilience et une distribution transparentes.
- Fonctionnalités :** Permet la manipulation distribuée de données en mémoire.

2. Spark DataFrame :



- Aperçu :** Les DataFrames Spark fournissent une abstraction de données structurées similaire à un tableau dans une base de données relationnelle.
- Fonctionnalités :** Offre un traitement plus efficace des données structurées avec une API similaire à SQL.

3. Spark GraphX :

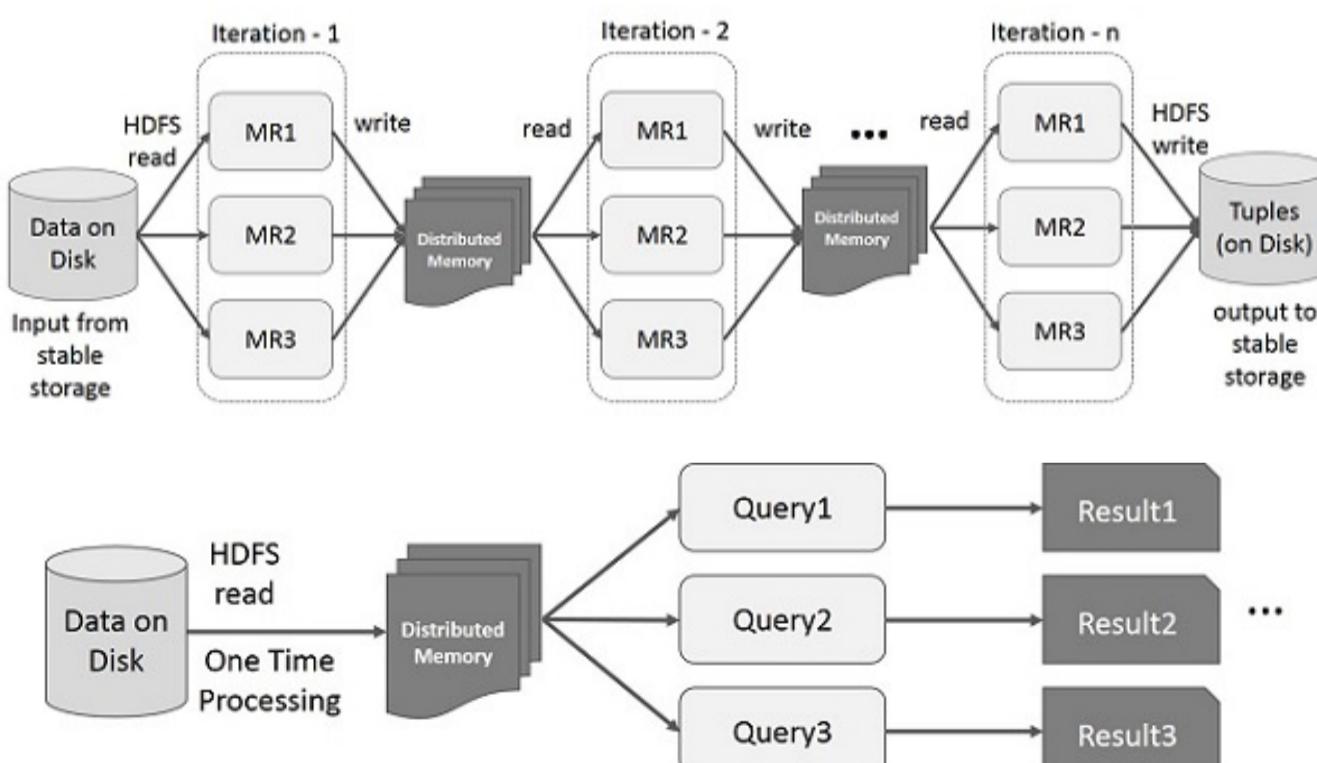


- **Aperçu** : Spark GraphX est la bibliothèque de traitement de graphes de Spark, permettant l'analyse distribuée des graphes.
- **Fonctionnalités** : Offre des outils pour l'analyse de graphes distribués et le calcul parallèle.

4. Spark Submit :

- **Aperçu** : Spark Submit est un outil en ligne de commande pour soumettre des applications Spark à un cluster.
- **Fonctionnalités** : Facilite le déploiement et l'exécution d'applications Spark sur un cluster.

Fonctionnement de Apache Spark :



- **Traitement en Mémoire** : Spark effectue le traitement en mémoire, ce qui accélère les opérations par rapport aux systèmes basés sur le stockage disque.
- **Abstraction de Données** : Les RDD offrent une abstraction de données résiliente et distribuée pour le traitement distribué des données.

- **Traitements Multi-Mode** : Spark prend en charge le traitement de données en mode batch, interactif et en temps réel au sein d'un même framework.
- **Intégration des Bibliothèques** : Spark offre une intégration transparente avec des bibliothèques spécialisées telles que Spark SQL, Spark Streaming et Spark MLlib pour répondre à divers besoins d'analyse de données.

Apache Spark est largement utilisé pour l'analyse de données distribuée en raison de sa rapidité, de sa polyvalence et de son support intégré pour divers types de traitement de données, de la simple analyse en batch à l'analyse en temps réel.

▼ Hadoop & Spark

Apache Spark et Hadoop servent à des fins différentes et adoptent des approches différentes en matière de stockage de données.

Hadoop :

- Hadoop est un cadre qui comprend le Hadoop Distributed File System (HDFS) pour le stockage distribué.
- Dans Hadoop, les données sont généralement stockées dans HDFS, conçu pour stocker et gérer de grands fichiers sur un cluster de machines.
- HDFS divise de grands fichiers en blocs, les réplique sur des nœuds du cluster pour la tolérance aux pannes et permet le traitement parallèle des données en utilisant le modèle de programmation MapReduce.

Spark :

- Spark, d'autre part, est un cadre de calcul distribué qui se concentre sur le traitement des données en mémoire de manière plus souple que Hadoop MapReduce.
- Spark ne possède pas son propre système de stockage distribué dédié comme HDFS. Il peut plutôt lire des données à partir de diverses sources, y compris HDFS, et effectuer un traitement distribué sur ces données.
- Bien que Spark puisse utiliser HDFS pour le stockage, il n'est pas limité à cela. Spark peut travailler avec d'autres systèmes de stockage tels que Amazon S3, Apache Cassandra, HBase, et plus encore.

En résumé, Hadoop fournit un système de fichiers distribué dédié (HDFS) pour stocker de grands fichiers, tandis que Spark est un cadre de calcul qui peut fonctionner avec divers systèmes de stockage et se concentre sur le traitement distribué des données en mémoire. Spark est souvent utilisé en conjonction avec des systèmes de stockage tels que HDFS, mais il n'a pas sa propre couche de stockage dédiée.

▼ Hive & HBase

Hive et HBase font partie de l'écosystème Apache Hadoop, mais remplissent des rôles distincts dans le traitement des données volumineuses.

Hive :

- **Rôle : Entrepôt de données** : Hive facilite le traitement de grands ensembles de données via un langage de requête SQL, offrant une abstraction pour les utilisateurs familiers avec SQL.
- **Fonctionnalités clés :**
 1. **SQL-Like (HiveQL)** : Permet des requêtes similaires à SQL.
 2. **Schéma à la lecture** : Structure des données définie lors de la requête.
 3. **Intégration Hadoop** : S'intègre avec HDFS pour le stockage des données.
- **Cas d'utilisation** : Adaptée aux requêtes ad hoc, à l'analyse de données pour l'intelligence d'affaires.

HBase :

- **Rôle : Base NoSQL** : HBase offre un accès en temps réel à de grands ensembles de données, basée sur un modèle de stockage par colonnes.
- **Fonctionnalités clés :**
 1. **Stockage par colonnes** : Optimisé pour une récupération efficace des colonnes.
 2. **Évolutivité** : Extension horizontale pour traiter de grandes quantités de données.

3. **Cohérence** : Assure une forte cohérence pour les opérations en temps réel.
 4. **Accès aléatoire** : Convient aux applications nécessitant un accès rapide à des enregistrements spécifiques.
- **Cas d'utilisation** : Idéale pour un accès en temps réel à d'importantes quantités de données, comme les systèmes de recommandation ou l'IoT.

Résumé :

Hive est utilisée pour des requêtes ad hoc et l'analyse de données volumineuses, offrant une interface SQL. HBase, base NoSQL, excelle dans l'accès en temps réel à des données massives. Ces outils complètent l'écosystème Hadoop, chacun répondant à des besoins spécifiques.