Architectures des Systèmes Distribués et Big Data

Master 1 - Semestre 1

Organisation

- 5 séances de cours
  - A partir du 06 Janvier 2025
- Des cas pratique
- 1 contrôle continu
- 1 Projet avec une présentation
- 1 examen final

Références

- "Distributed Systems: Principles and Paradigms" par **Andrew S. Tanenbaum et Maarten Van Steen.**
- "Designing Data-Intensive Applications" **par Martin Kleppmann**.
- "Big Data: Principles and best practices of scalable realtime data systems" **par Nathan Marz et James Warren.**
- Analyse d'articles sur les tendances actuelles en systèmes distribués, tels que les systèmes sans serveur et l'edge computing.
- "Hadoop: The Definitive Guide" par Tom White.
- "Spark: The Definitive Guide" par Bill Chambers et Matei Zaharia.
- "Learning Spark: Lightning-Fast Big Data Analysis" par Holden Karau, Andy Konwinski, Patrick Wendell, Matei Zaharia.
- "Stream Processing with Apache Flink" par Fabian Hueske, Vasiliki Kalavri.
- "Storm: Distributed Real-time Computation Framework" par Taylor Gautier.
- "Architecting the Cloud" par Michael J. Kavis.
- "Cloud Computing: Concepts, Technology & Architecture" par Thomas Erl.

Sommaire

Historique : Origine du concept de Big Data 1/3

- **Définition :** Le terme ***Big Data*** a été utilisé pour la première fois dans les années 1990 pour décrire les défis liés à la gestion d'énormes volumes de données. Ce terme fait référence à des ensembles de données si volumineux et complexes qu'ils nécessitent des technologies spécifiques pour les collecter, les stocker, les gérer et les analyser.
- **Origine du terme :**
  Le concept est apparu dans les années 1990. John Mashey, un scientifique en informatique, est souvent crédité de l'utilisation du terme dans ses discours sur les défis liés à la gestion des données massives.
- **Les trois "V" du Big Data :**
  En 2001, Doug Laney, analyste chez Gartner, a introduit le modèle des **3 "V"** pour caractériser les Big Data :
  - **Volume :** Quantité massive de données générées chaque seconde.
  - **Vélocité :** Rapidité à laquelle ces données sont créées, collectées et traitées.
  - **Variété :** Différents formats (structurés, semi-structurés, non structurés).

Historique : Origine du concept de Big Data 2/3

La révolution des bases de données (années 1970-1980)

- **Apparition des SGBD relationnels :**
  - Dans les années 1970, Edgar F. Codd, mathématicien anglais chez IBM, a proposé le modèle relationnel qui a conduit au développement des systèmes de gestion de bases de données relationnelles (SGBDR) comme Oracle, IBM DB2, et Sybase.
  - Les bases de données relationnelles ont transformé la manière dont les données étaient stockées et récupérées. Le développement des systèmes de gestion de bases de données relationnelles (SGBDR) tels que IBM's System R et Oracle a permis une gestion plus structurée et efficace des données.
  - Ces systèmes ont permis d'organiser et de manipuler de grandes quantités de données structurées.
- **Langage SQL (Structured Query Language) :**
  - SQL (Structured Query Language) a été introduit, dans les années 1980, pour permettre aux utilisateurs de structurer et d'interroger les données facilement. SQL est devenu le langage standard pour interagir avec les bases de données relationnelles, facilitant l'extraction et l'analyse des données.

Historique : Origine du concept de Big Data 3/3

Le développement des entrepôts de données (années 1990)

- **Concept :**
  - Les entrepôts de données(*data warehouse*) ont été développés pour centraliser les données provenant de diverses sources. Ces entrepôts permettaient de stocker et d'analyser de grandes quantités de données historiques, facilitant ainsi la prise de décisions informées. Des entreprises comme Teradata et IBM ont joué un rôle clé dans cette évolution.
  - Ce modèle a marqué une évolution significative vers l'analyse décisionnelle.
- **Technologies et outils associés :**
  - Lancements d'outils tels que **Informatica** et **SAP BW**.
  - Apparition du concept de *OLAP (Online Analytical Processing)* pour l'analyse multidimensionnelle des données.
- **Exemples de mise en œuvre :**
  - Secteur bancaire : Utilisation des entrepôts de données pour le suivi des transactions et la détection des fraudes.
  - Secteur du commerce : Analyse des comportements d'achat à l'aide des systèmes de gestion des données des points de vente.

Historique : Les phases de l'évolution du Big data - 3V

- **Volume**
  - L'augmentation massive des données générées par les entreprises et les consommateurs a été l'une des premières caractéristiques du Big Data. Cette explosion des données a été alimentée par l'adoption croissante des technologies numériques et d'Internet.
  - Exemples :
    - Facebook générait des téraoctets de données utilisateur dès ses premières années.
    - Les grandes entreprises accumulaient d'énormes quantités de données CRM (Customer Relationship Management) et transactionnelles.
- **Vélocité**
  - La capacité de traiter et d'analyser les données en temps réel est devenue cruciale pour les entreprises cherchant à obtenir des insights rapides et à prendre des décisions instantanées. Les données de streaming et les transactions en ligne ont nécessité des solutions de traitement en temps réel. Le besoin de traiter les données en temps réel a émergé pour répondre à des défis comme :
    - **Traitement des transactions bancaires instantanées.**
    - **Gestion des flux de données des capteurs IoT (Internet of Things).**
  - Développement de systèmes capables de collecter et de traiter les données à une vitesse inégalée, par exemple avec l'émergence du *streaming data*.

Historique : Les phases de l’évolution du Big data - 3V

- **Variété**
  - Les données ne se limitaient plus aux formats structurés comme les tableaux de bases de données. Elles incluaient désormais des données non structurées et semi-structurées telles que des images, des vidéos, des textes, des logs, etc. Cela a nécessité de nouvelles approches pour le stockage et l'analyse de ces types variés de données. Les données au format non structuré :
    - **Images, vidéos, audio** provenant des réseaux sociaux, caméras de sécurité, capteurs, etc.
    - **Texte brut, emails, logs d’application, données de navigation web.**
  - Apparition de nouveaux défis pour stocker et interroger ces types de données.

Historique : Technologies disruptives 1/2

- **Traitement distribué avec Hadoop et Spark**
  - **Hadoop (2006) :** Hadoop a révolutionné le traitement des données en introduisant un cadre open-source pour le stockage et le traitement distribué de grandes quantités de données à l'aide de clusters d'ordinateurs grâce au modèle *MapReduce*.
    - *HDFS (Hadoop Distributed File System)* a permis un stockage fiable et distribué.

  - **Apache Spark (2009) :** Spark a apporté des améliorations significatives en termes de vitesse de traitement et de capacités d'analyses en mémoire.
    - Complément ou alternative à Hadoop.
    - Offrant une vitesse de traitement jusqu'à 100 fois supérieure en mémoire.
    - Permet l'analyse en temps réel avec des bibliothèques dédiées comme *MLlib* (machine learning).

Historique : Technologies disruptives 2/2

- **Bases de données NoSQL :** Pour répondre aux besoins de flexibilité et de scalabilité, des bases de données NoSQL comme MongoDB, Cassandra, et Couchbase ont été développées. Elles ont permis de gérer efficacement des données non structurées et de fournir des performances élevées pour des applications nécessitant de grandes capacités de traitement.
  - **MongoDB :** MongoDB est une base de données NoSQL orientée documents, connue pour sa capacité à gérer de grandes quantités de données non structurées. Elle permet une scalabilité horizontale et offre une flexibilité dans la modélisation des données.
  - **Cassandra :** Développé par Apache, Cassandra est une base de données NoSQL distribuée conçue pour gérer de grandes quantités de données à travers plusieurs serveurs. Elle est particulièrement adaptée pour les applications nécessitant une haute disponibilité et une scalabilité massive.
  - Ces bases NoSQL ont permis une flexibilité accrue par rapport aux SGBD traditionnels.
- **Cas d'usage clés**
  - Commerce en ligne : Recommandations personnalisées grâce à l'analyse de comportements en temps réel.
  - Santé : Analyse prédictive pour la recherche biomédicale et le diagnostic rapide.

Historique : IA et IoT

- **Fusion entre Big Data et intelligence artificielle (IA)**
  - Les algorithmes d'apprentissage automatique s'appuient sur des volumes massifs de données pour affiner leurs modèles.
  - Développement de technologies comme les *deep learning frameworks* (TensorFlow, PyTorch) pour tirer parti des données Big Data.
- **Applications dans l'Internet des Objets (IoT)**
  - Les objets connectés produisent d'énormes volumes de données exploitables en temps réel.
  - Exemples :
    - **Voitures autonomes :** Traitement des flux en direct pour naviguer.
    - **Maisons intelligentes :** Analyse des données des appareils pour une optimisation énergétique.
- **Décisions stratégiques basées sur les Big Data**
  - **Secteur financier :** Analyse des risques en temps réel.
  - **Secteur logistique :** Optimisation des chaînes d'approvisionnement grâce à l'analyse prédictive et au suivi des marchandises en temps réel.

Historique : Le Big Data - Importance Stratégiques

- Le Big Data est devenu un levier crucial pour la compétitivité des entreprises et la gouvernance des institutions publiques.
- **Pour les entreprises :**
  - Analyse approfondie des comportements des consommateurs pour personnaliser les services et augmenter la satisfaction client.
  - Optimisation des processus internes grâce à l'automatisation et à l'analyse prédictive (maintenance prédictive, optimisation logistique).
  - Décisions basées sur des données : Adoption généralisée de la *data-driven culture* pour orienter les stratégies commerciales.
  - Exemples :
    - **Amazon :** Exploite les données pour la gestion des stocks et les recommandations produits.
    - **Tesla :** Analyse les données des voitures connectées pour améliorer la conduite autonome.

Historique : Le Big Data - Importance Stratégiques

- **Pour les gouvernements :**
  - Utilisation des données pour la planification urbaine (trafic, pollution, infrastructures).
  - Surveillance et anticipation des crises (sanitaires, climatiques, économiques).
  - Exemple :
    - **Chine :** Utilisation des Big Data dans les villes intelligentes pour optimiser les transports publics et surveiller les émissions de CO_2.

Historique : Évolution des grands V - Véracité

- **Années 2000 – L'émergence des 3V (avec les autres grands "V") :** Les 3 "**V**" (Volume, Vélocité, Variété) sont souvent complétés par d'autres dimensions, reflétant les défis et opportunités du Big Data dans les années 2000. Voici une exploration détaillée des autres "V" :
- **Véracité**
  - **Définition :**
    - Désigne la qualité et la fiabilité des données collectées.
    - Les entreprises doivent s'assurer que leurs données ne contiennent pas d'erreurs, de duplications ou d'informations biaisées.
  - **Exemples de défis liés à la véracité :**
    - **Réseaux sociaux :** Les données générées par les utilisateurs (commentaires, likes) peuvent être bruyantes, incomplètes ou biaisées.
    - **Capteurs IoT :** Les appareils peuvent parfois transmettre des données erronées en raison de pannes ou d'interférences.
  - **Solutions développées :**
    - Techniques de nettoyage des données (ETL – Extraction, Transformation, Chargement).
    - Détection des anomalies pour identifier les incohérences.

Historique : Évolution des grands V - Valeur

- **Valeur**
  - **Définition :** Les données massives n'ont de sens que si elles apportent une valeur ajoutée concrète. Cela implique de transformer les données brutes en informations exploitables pour la prise de décision.
  - **Exemples d'application de la valeur des données :**
    - **Commerce en ligne :** Les données sur les comportements d'achat permettent de créer des recommandations personnalisées.
    - **Marketing :** Analyse des segments clients pour concevoir des campagnes ciblées et efficaces.
    - **Finance :** Analyse prédictive pour anticiper les fluctuations du marché.
  - **Outils pour extraire la valeur :**
    - Analytique avancée avec Python, R ou des outils spécialisés (Tableau, Power BI).
    - Techniques de *data mining* et de *machine learning*.

Historique : Évolution des grands V - Variabilité

- **Variabilité**
  - **Définition :**
    - Ce terme met en lumière les fluctuations des données en termes de formats, de types et de structures.
    - La variabilité peut également se référer à l'incertitude dans l'interprétation des données en raison de leur complexité.
  - **Exemples :**
    - **Données saisonnières :** Les tendances peuvent changer drastiquement selon la période (ex. : données e-commerce à Noël vs hors saison).
    - **Langages naturels :** Analyse des sentiments dans les tweets, où une même phrase peut être interprétée différemment selon le contexte.
  - **Défis et solutions :**
    - Développer des algorithmes capables de gérer les données contextuelles et ambiguës.
    - Utilisation de technologies comme le traitement automatique du langage naturel (NLP).

Historique : Évolution des grands V - Visibilité

- **Visibilité (ou Visualisation)**
  - **Définition :**
    - Capacité à présenter des données complexes sous une forme compréhensible et exploitable via des visualisations interactives et significatives.
  - **Exemples d'outils de visualisation :**
    - Tableau, Power BI, D3.js.
    - Visualisation de tendances dans des ensembles de données massifs pour détecter des anomalies ou des opportunités.
  - **Impact sur la prise de décision :**
    - Facilite la communication des idées aux décideurs non techniques.
    - Permet d'identifier rapidement des corrélations ou des anomalies.

Historique : Tendances actuelles - Edge Computing

- **Edge Computing**
  - **Définition :** Traitement des données à la périphérie du réseau, près de leur source, plutôt que dans un centre de données centralisé.
  - **Bénéfices :**
    - Réduction de la latence (temps réel).
    - Moins de dépendance vis-à-vis des infrastructures cloud.
  - **Exemples :**
    - Voitures autonomes traitant les données des capteurs en local pour des décisions immédiates.
    - Caméras de sécurité intelligentes avec traitement intégré.

Historique : Tendances actuelles - Data Lakes

- **Data Lakes**
  - **Définition :** Un référentiel centralisé qui stocke des données dans leur format brut, structuré ou non structuré.
  - **Avantages :**
    - Capacité de gérer différents types de données sans avoir à définir un schéma au préalable.
    - Flexibilité pour l'analyse exploratoire et l'apprentissage automatique.
  - **Exemples :**
    - Entreprises utilisant des *data lakes* comme Amazon S3 pour centraliser les données issues de multiples systèmes.

Historique : Tendances actuelles - Data Warehouse

- **Data Warehouse**
  - **Définition :** Un **Data Warehouse** est un système centralisé conçu pour stocker des données structurées provenant de multiples sources. Optimisé pour les requêtes complexes et la génération de rapports, il est largement utilisé dans les processus décisionnels.
  - **Avantages :**
    - **Performance élevée :** Conçu pour des requêtes analytiques rapides sur de grandes volumétries.
    - **Fiabilité :** Données nettoyées, intégrées, et conformes pour une prise de décision cohérente.
    - **Scalabilité :** S'adapte aux besoins croissants des entreprises grâce aux solutions modernes dans le Cloud.
  - **Exemples :**
    - **Amazon Redshift :** Solution Cloud pour des analyses à grande échelle.
    - **Google BigQuery :** Analyse rapide et serverless pour des données massives.
    - **Snowflake :** Plateforme Cloud hybride supportant plusieurs formats de données.

Historique : Tendances actuelles - Cloud Computing

- **Cloud Computing**
  - **Définition :** Utilisation de plateformes cloud pour stocker, gérer et analyser les Big Data à grande échelle.
  - **Avantages :**
    - Évolutivité quasi illimitée.
    - Réduction des coûts d'infrastructure interne.
    - Solutions prêtes à l'emploi pour l'analyse Big Data (AWS, Azure, Google Cloud).
  - **Exemples :**
    - Entreprises adoptant des solutions cloud pour des projets IA comme la reconnaissance faciale ou la modélisation prédictive.

Sommaire

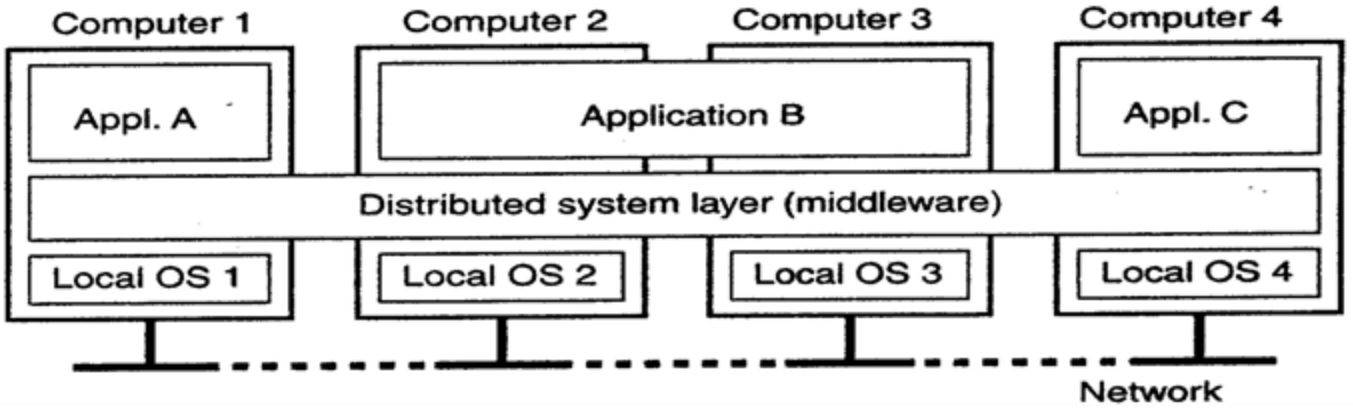
Introduction : Définition

Objectifs :

- Comprendre les concepts fondamentaux des systèmes distribués.
- Identifier les caractéristiques distinctives des systèmes distribués.
- Appréhender les applications pratiques et les défis associés à ces systèmes.
- **Définitions :**
  - Un système distribué est un ensemble de composants matériels et logiciels connectés via un réseau, qui interagissent pour apparaître comme un système unique à l'utilisateur et coordonnés pour réaliser une tâche commune. Contrairement aux systèmes centralisés, les systèmes distribués n'ont pas de mémoire partagée et communiquent uniquement via des messages. (réf : *Distributed Systems: Principles and Paradigms*)
  - Un système distribué est un modèle de traitement où plusieurs ordinateurs indépendants (nœuds) travaillent ensemble pour atteindre un objectif commun. Ces systèmes sont conçus pour partager des ressources, comme des fichiers ou des périphériques, et pour permettre une communication transparente entre les nœuds.
  - **Exemple :** Une application de messagerie instantanée comme WhatsApp, où plusieurs serveurs à travers le monde gèrent les communications des utilisateurs de manière transparente.

Schéma

- Un système distribué organisé comme un intergiciel. La couche d'intergiciel s'étend sur plusieurs machines et offre à chaque application la même interface.



Caractéristiques 1/2

- **Transparence :**
  - Transparence d'accès : L'utilisateur interagit avec le système sans savoir où les ressources sont situées.
  - Transparence de localisation : Les ressources peuvent être situées à divers endroits mais semblent localisées de manière uniforme pour l'utilisateur.
  - Exemple : Accéder à un fichier stocké sur le cloud sans savoir dans quel centre de données il se trouve.
- **Concurrence :** La gestion des processus concurrents, l'importance de la synchronisation, et les techniques pour éviter les interblocages.
- **Performance :**
  - Optimisation des ressources pour minimiser la latence et maximiser le débit.
  - Exemple concret : Utilisation de caches locaux pour réduire la latence d'accès aux données fréquemment demandées.

Caractéristiques 2/2

- **Fiabilité :**
  - Les systèmes distribués sont conçus pour continuer à fonctionner même si certains nœuds échouent.
  - Tolérance aux pannes à travers la redondance et la récupération.
  - Exemple : Réplication des données dans une base de données distribuée pour assurer la disponibilité même en cas de panne d'un nœud.
- **Évolutivité et Scalabilité :**
  - Capacité d'ajouter des ressources pour gérer l'augmentation de la charge sans dégrader les performances.
  - Exemple : Ajout de serveurs dans un cluster Hadoop pour gérer plus de données.
- **Hétérogénéité** : Les nœuds peuvent avoir des architectures matérielles et des systèmes d'exploitation différents.
- **Exemple** : Internet est l'exemple le plus connu de système distribué, où des millions de serveurs et de clients interagissent.

Historique et Technologies

Historique :

- Des premiers systèmes centralisés, où un ordinateur central effectuait toutes les tâches, aux systèmes modernes de cloud et edge computing qui distribuent les tâches à travers plusieurs nœuds pour améliorer la performance et la disponibilité.
- *Exemple :* Transition des mainframes aux microservices hébergés sur des infrastructures cloud.

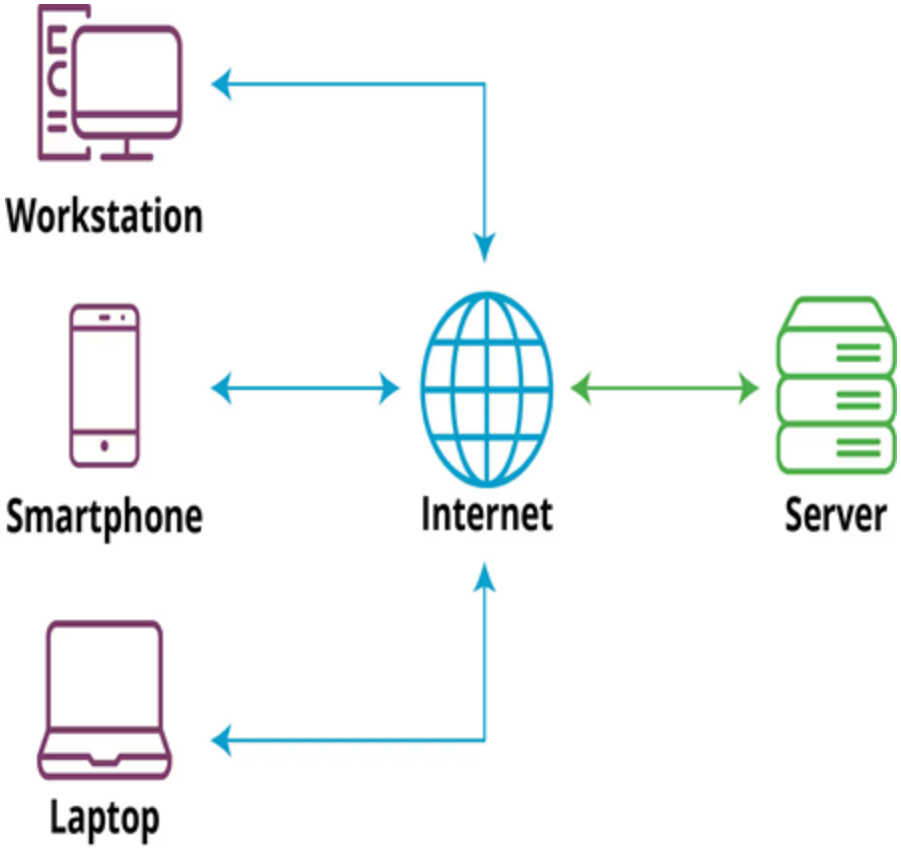
Technologies influentes :

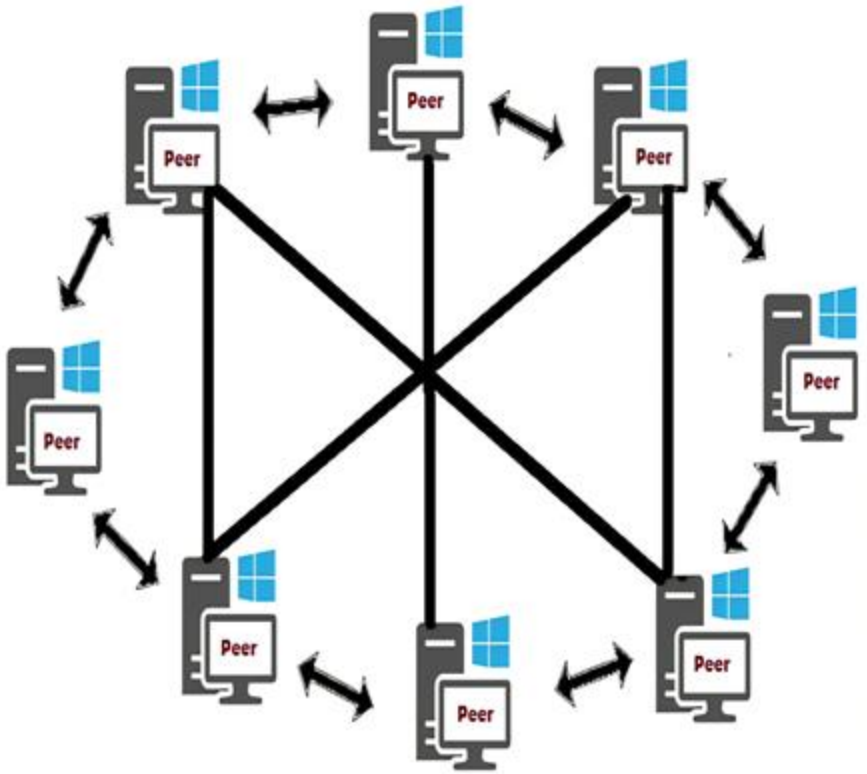
- **LAN et WAN :**
  - Les réseaux locaux connectent des ordinateurs dans une même zone, alors que les réseaux étendus connectent des ordinateurs à travers des zones géographiques larges.
  - *Exemple :* Utilisation d'un réseau LAN pour connecter les serveurs dans un data center et un WAN pour connecter différents data centers.
- **Internet :**
  - Base des systèmes distribués modernes, permettant la communication à grande échelle.
  - *Exemple :* Applications comme Netflix qui exploitent l'Internet pour distribuer du contenu à des utilisateurs dans le monde entier.

Modèles d’architecture

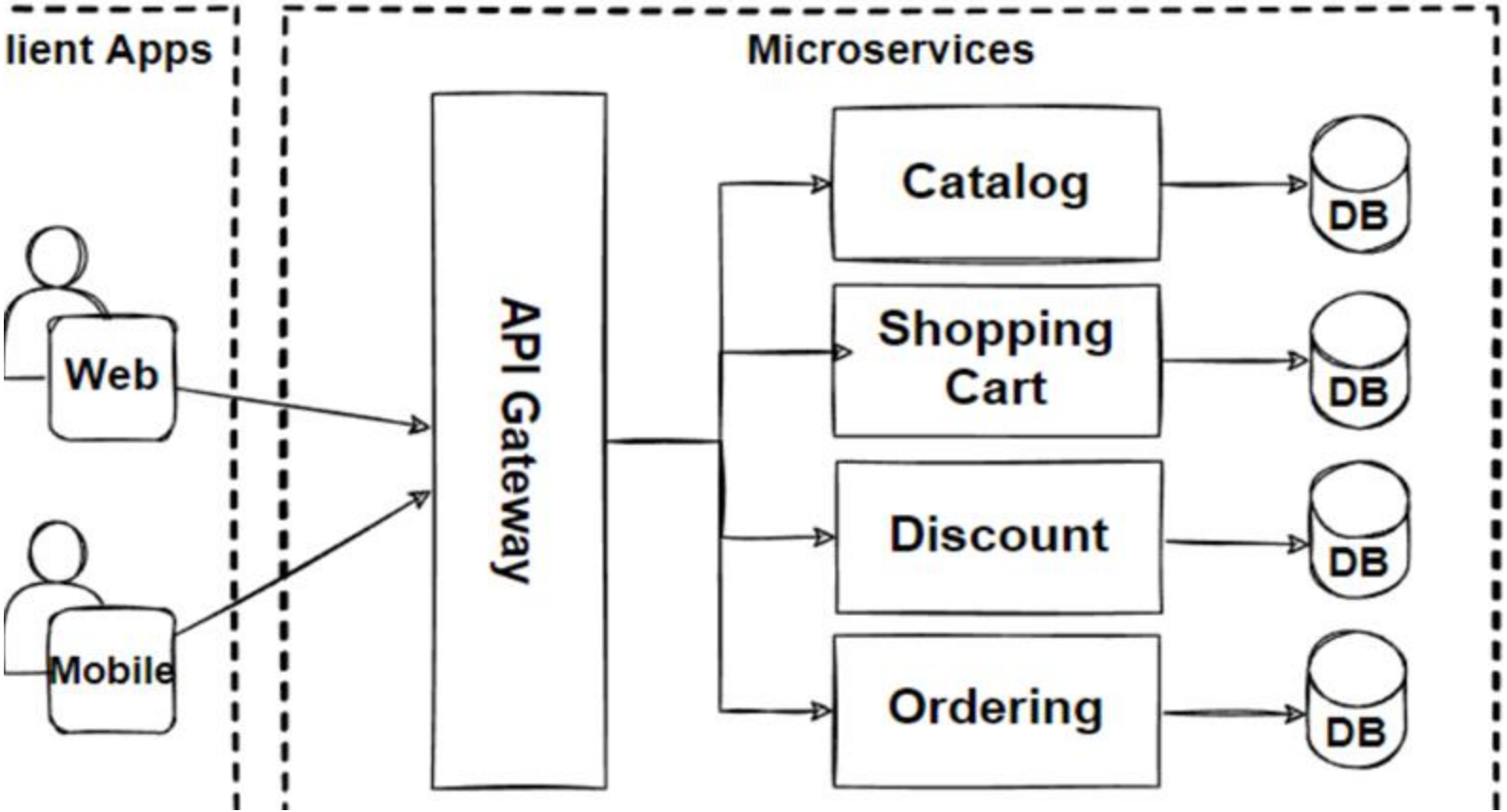
- **Client-serveur :**
  - Le client demande un service que le serveur fournit.
  - *Exemple :* Une application web où le navigateur agit comme un client et le serveur web fournit les pages demandées.
- **Peer-to-peer (P2P) :**
  - Les nœuds agissent comme clients et serveurs.
  - *Exemple :* Réseau de partage de fichiers comme BitTorrent.
- **Microservices :**
  - Applications décomposées en services légers, chaque service ayant sa propre logique métier.
  - *Exemple :* Une plateforme e-commerce où chaque fonctionnalité (paiement, recherche, etc.) est un microservice distinct.
- **Architectures basées sur les événements :**
  - Les composants communiquent en émettant et en écoutant des événements.
  - *Exemple :* Utilisation de Kafka pour gérer les flux de données en temps réel dans un système de recommandation.

Architecture client/serveur et peer-to-peer





Architectures micro-services



Exemples de systèmes distribués

- **Cloud Computing :**
  - Plateformes comme AWS, Azure, et Google Cloud offrent des services de stockage, de calcul et de réseau à travers un modèle distribué.
  - *Exemple :* Déploiement d'une application mobile utilisant AWS Lambda pour l'exécution de fonctions sans gérer de serveurs.
- **Bases de données distribuées :**
  - Systèmes comme Cassandra et MongoDB répartissent les données à travers plusieurs nœuds pour améliorer la disponibilité et la performance.
  - *Exemple :* Utilisation de Cassandra pour gérer une base de données utilisateur massive dans un réseau social.
- **Streaming de données :**
  - Plateformes comme Apache Kafka et Apache Flink traitent les flux de données en temps réel pour des applications comme l'analyse de transactions financières ou la détection de fraudes.
  - *Exemple :* Utilisation de Kafka pour ingérer et traiter les données de capteurs dans une usine intelligente.

Applications pratiques

- **Finance :** Utilisation de systèmes distribués pour gérer les transactions en temps réel et assurer la disponibilité continue des services bancaires en ligne.
- **Santé :** Systèmes distribués pour stocker et analyser les données des patients de manière sécurisée et conforme aux réglementations.
- **E-commerce :** Plateformes qui utilisent des architectures distribuées pour gérer les fortes charges de trafic, les paiements et la gestion des stocks.
- **Réseaux sociaux :** Systèmes distribués pour gérer des millions d'utilisateurs simultanés, stocker des données multimédias, et fournir des recommandations personnalisées.

Installation et Configuration : Apache Hadoop

- **Objectif :** Donner une expérience pratique de l'installation et de la configuration d'un cluster Hadoop en utilisant des machines virtuelles (VM).
- **Étapes Détaillées :**
- **Préparation : Choisir un Framework - Apache Hadoop**
  - **Choix du Framework :** Apache Hadoop est un framework open-source permettant de gérer des systèmes de stockage distribués et de traitement de données volumineuses.
- **Installation : Suivre les instructions pour installer Hadoop**
  - **Prérequis :**
  - Système d'exploitation : Ubuntu 20.04 LTS (ou toute distribution Linux)
  - Java JDK : Version 8 ou supérieure
- **Création et Configuration des Machines Virtuelles :**
  - Créer trois machines virtuelles (une pour le maître et deux pour les esclaves).
  - Assurez-vous que toutes les VM sont sur le même réseau.

Cas pratique

Instructions

Installation de Java sur toutes les VM :

```
sudo apt update
sudo apt install openjdk-8-jdk -y
java -version
```

Téléchargement et Installation de Hadoop :

```
wget https://downloads.apache.org/hadoop/common/hadoop-3.3.4/hadoop-3.3.4.tar.gz
tar -xzvf hadoop-3.3.4.tar.gz
sudo mv hadoop-3.3.4 /usr/local/hadoop
```

Configuration des Variables d'Environnement : Ajouter les lignes suivantes à ~/.bashrc :

```
export HADOOP_HOME=/usr/local/hadoop
export PATH=$PATH:$HADOOP_HOME/bin
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
source ~/.bashrc
```

Configuration des noeuds pour former un cluster

- **Modification des fichiers de configuration Hadoop :**

core-site.xml :

```
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://master-node:9000</value>
  </property>
</configuration>
```

Configuration des noeuds pour former un cluster

- **hdfs-site.xml** :

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>2</value>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>file:/usr/local/hadoop/hdfs/namenode</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>file:/usr/local/hadoop/hdfs/datanode</value>
  </property>
</configuration>
```

Configuration des noeuds pour former un cluster

- **mapred-site.xml** :

```
<configuration>
 <property>
  <name>mapreduce.framework.name</name>
  <value>yarn</value>
 </property>
</configuration>
```

- **yarn-site.xml** :

```
<configuration>
 <property>
  <name>yarn.nodemanager.aux-services</name>
  <value>mapreduce_shuffle</value>
 </property>
</configuration>
```

Configuration des noeuds pour former un cluster

Configurer le Fichier slaves : Ajouter les noms ou les adresses IP des nœuds esclaves.

slave1

slave2

Exécution : Lancer un job simple pour vérifier le bon fonctionnement du cluster

- **Démarrage des Services Hadoop :**

start-dfs.sh

start-yarn.sh

- **Vérification du Statut des Nœuds :**
  - Accéder à l'interface web de HDFS : http://master-node:9870/
  - Accéder à l'interface web de YARN : http://master-node:8088/
- **Lancer un Job MapReduce Exemple :**

hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.3.4.jar wordcount input_dir output_dir

- **Vérification du Résultat :**

hdfs dfs -cat output_dir/part-r-00000

Sommaire

Technologies et Plateformes Big Data

Technologies de traitement du Big data : Hadoop

- **Composants de Base :**
  - **HDFS (Hadoop Distributed File System) :** Système de fichiers distribué qui stocke des fichiers en les divisant en blocs et en les distribuant sur plusieurs nœuds pour assurer la tolérance aux pannes et la haute disponibilité.
    - **Exemple :** Stockage de données météorologiques historiques.
  - **YARN (Yet Another Resource Negotiator) :** Gestionnaire de ressources qui permet à plusieurs applications de traiter des données en parallèle.
    - **Exemple :** Planification de tâches MapReduce sur un cluster.
  - **MapReduce :** Modèle de programmation pour le traitement distribué des grandes quantités de données en deux étapes (Map et Reduce).
    - **Exemple :** Analyse de journaux de serveurs pour des schémas de trafic.
- **Fonctionnement et Cas d'Utilisation Typiques :**
  - **Processus :** Les données sont stockées dans HDFS, traitées en parallèle par MapReduce et les résultats sont stockés à nouveau dans HDFS.
  - **Exemples :** Analyse des logs de serveur pour identifier les tendances et les anomalies.

Technologies de traitement du Big data : Hadoop

- **Limites de Hadoop :**
  - **Performance :** Lent pour les tâches itératives.
  - **Complexité :** Nécessite une gestion et une maintenance complexes.
  - **Transition :** Migration vers des frameworks plus modernes comme Apache Spark pour le traitement en mémoire et le traitement en temps réel.

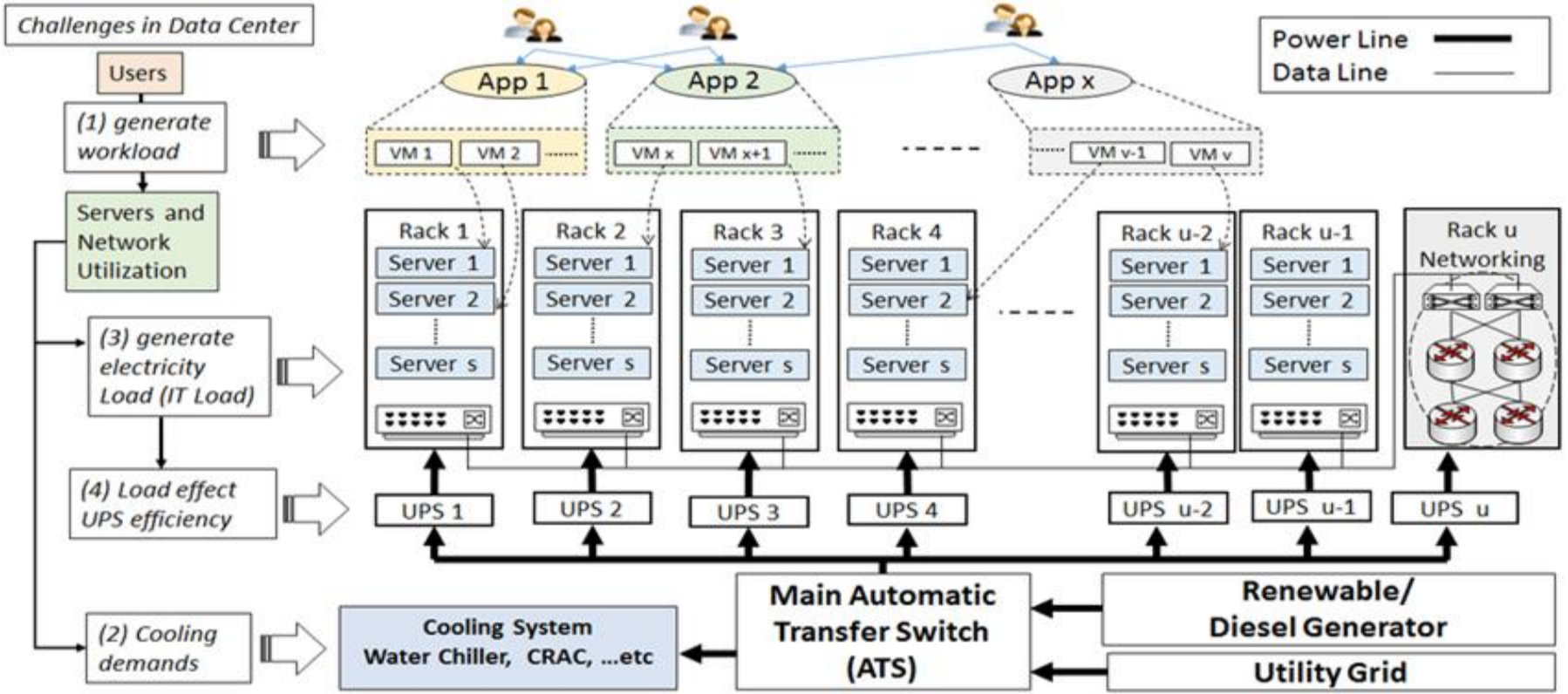
Technologies de traitement du Big data : Spark

- **Fonctionnalités de base :**
  - **API pour le traitement de données en mémoire:** Traitement en mémoire pour des performances plus rapides.
  - **MLlib pour le machine learning :** Librairie ML intégrée qui permet de construire des modèles de machine learning à grande échelle.
  - **Spark Streaming :** Composant pour le traitement de flux de données en temps réel.
- **Comparaison entre Spark et MapReduce :**
  - **Performance :** Spark est beaucoup plus rapide que MapReduce car il utilise le traitement en mémoire.
  - **Facilité d'utilisation :** Spark offre des API haut niveau en Java, Scala, Python et R, ce qui facilite son utilisation pour les développeurs.
  - **Exemple :** Traitement en temps réel des données de capteurs dans une usine.

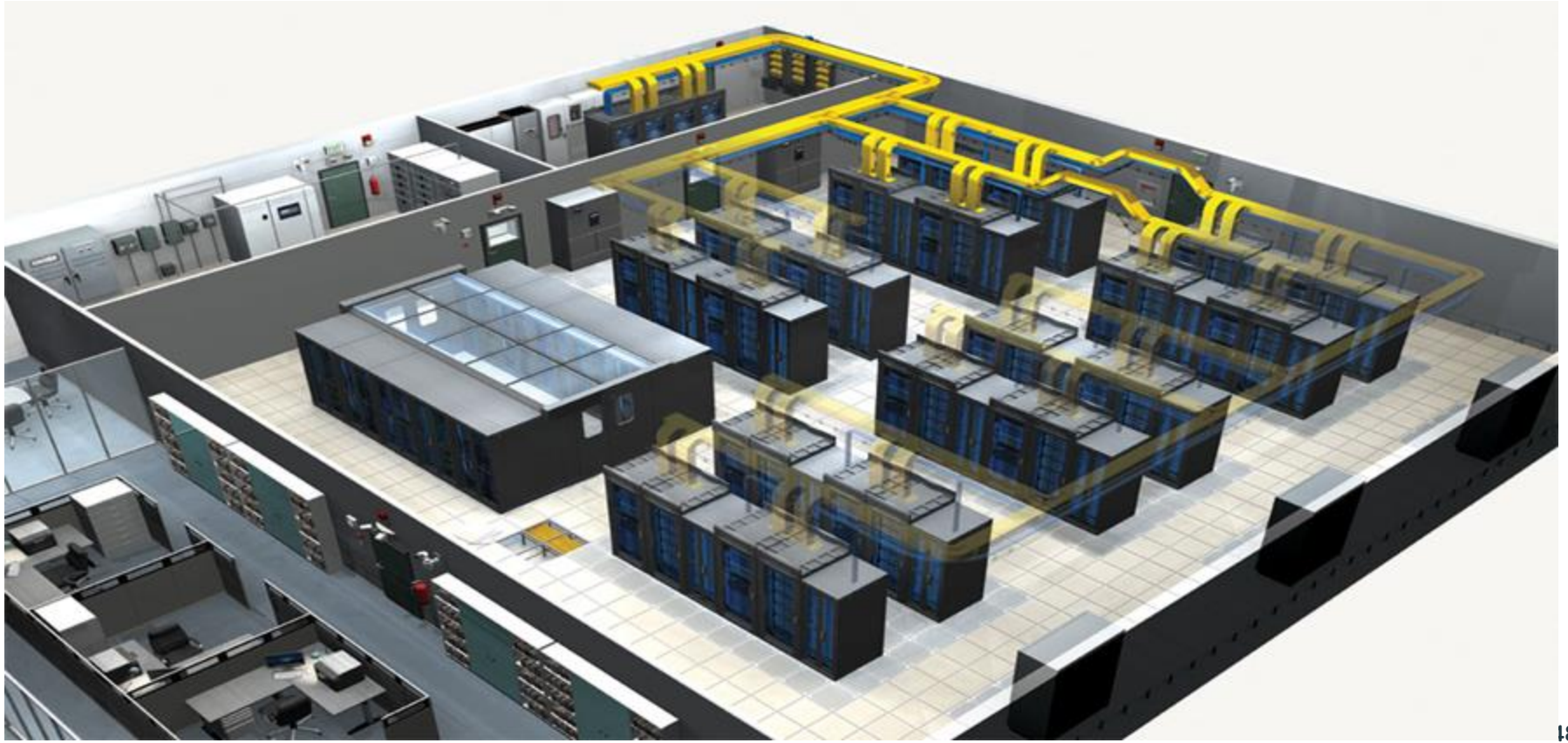
Technologies de traitement du Big data : Flink & Storm

- **Différences entre Flink et Spark :**
  - **Traitement de flux natif :** Flink est conçu pour le traitement de flux en temps réel avec une latence très faible et une haute disponibilité.
  - **Gestion de l'état :** Flink offre une gestion avancée de l'état, permettant de garder l'état des applications en temps réel.
  - *Exemple :* Surveillance en temps réel des transactions financières pour détecter des fraudes immédiatement.
- **Storm :**
  - **Utilisation de Storm pour le traitement de flux distribués :** Storm permet de traiter des flux de données en temps réel en utilisant des "topologies" composées de "spouts" (sources de données) et de "bolts" (unités de traitement).
  - **Mise en œuvre et cas pratiques :** Configuration d'une topologie Storm pour ingérer et traiter des données en temps réel.
    - *Exemple :* Analyse des clics utilisateurs en temps réel sur un site web pour personnaliser les recommandations.

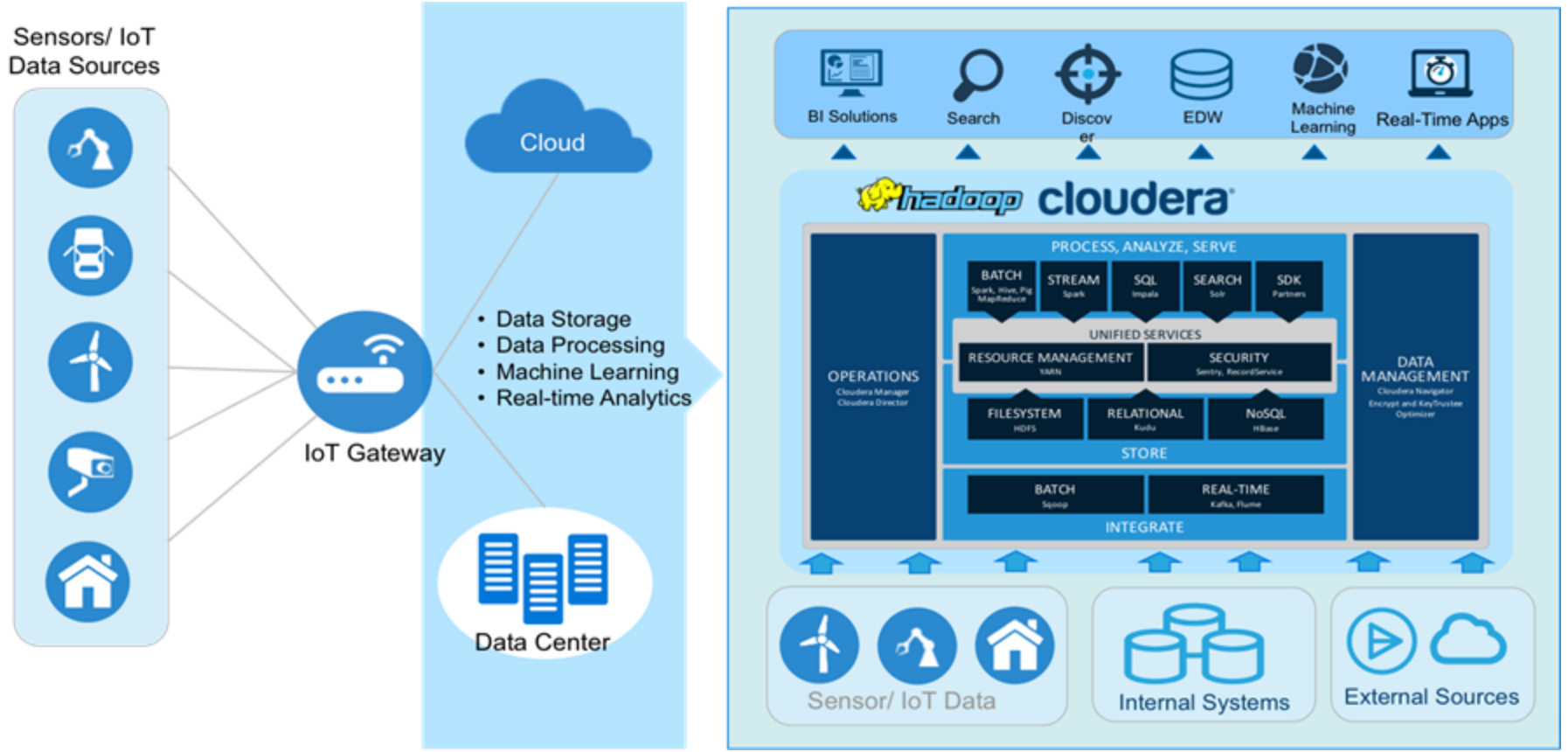
Data center High Level Design



Data center HLD : 3G



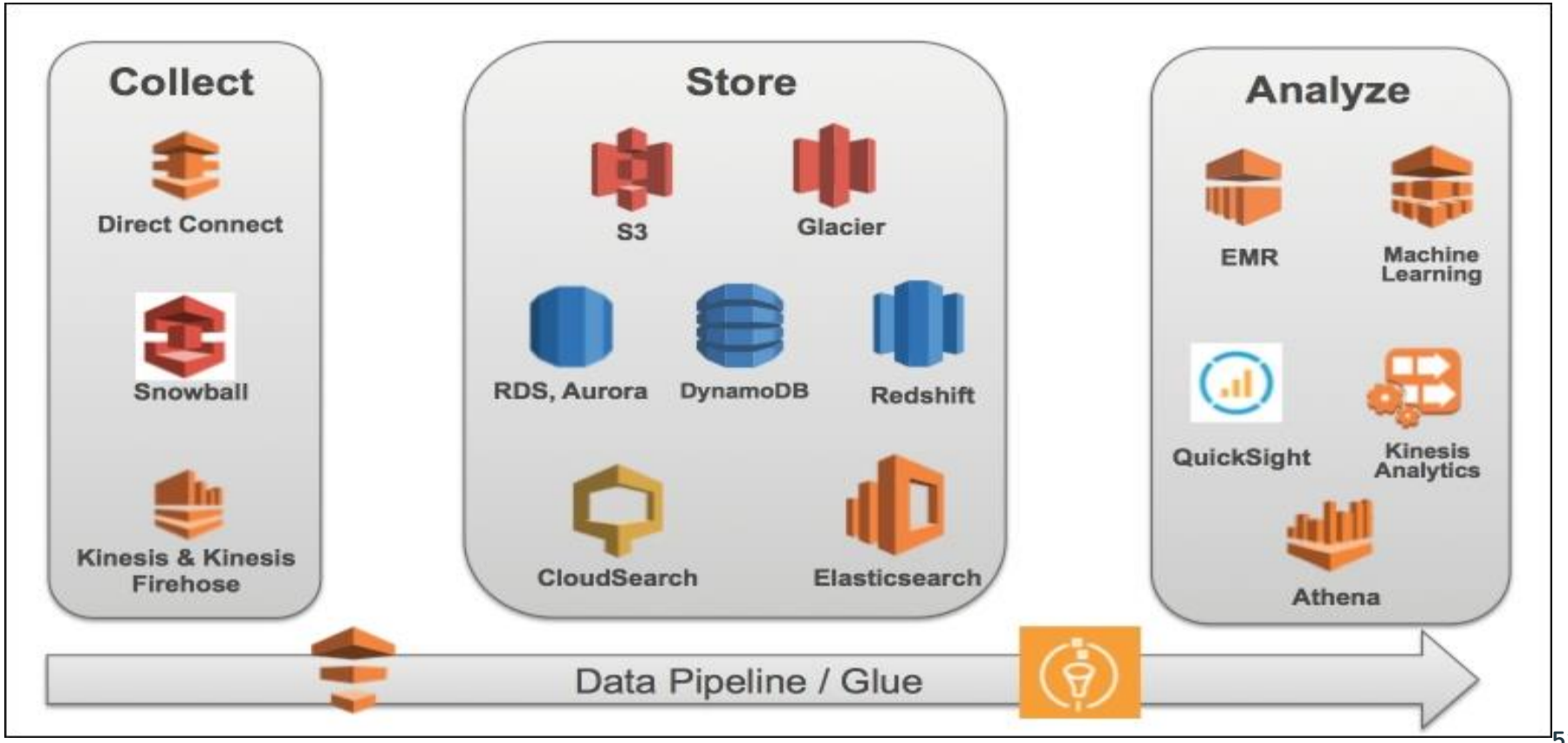
Cloudera HLD Design



Amazon Web Services AWS

- **EMR (Elastic MapReduce) :** Service de traitement de données basé sur Hadoop et Spark, permettant d'exécuter des clusters temporaires pour analyser de grandes quantités de données.
  - **Exemple :** Analyser les journaux de clics d'une boutique en ligne pour détecter les tendances d'achat.
- **Redshift :** Est un service de Data warehouse en cloud rapide, scalable et entièrement géré qui permet l'analyse des données en utilisant SQL.
  - **Exemple :** Analyse des ventes et des performances marketing pour une entreprise.
- **Kinesis :** Permet de collecter, traiter et analyser des flux de données en temps réel.
  - **Exemple :** Surveiller les transactions de cartes de crédit en temps réel pour détecter des fraudes potentielles.
- **Étude de cas : Netflix**
  - **Utilisation d'EMR :** Netflix utilise EMR pour traiter d'énormes volumes de données sur les habitudes de visionnage des utilisateurs afin de fournir des recommandations personnalisées.
  - **Utilisation de Redshift :** Est utilisé pour stocker les logs de streaming et les données analytiques, facilitant ainsi l'analyse rapide des comportements des utilisateurs.
  - **Utilisation de Kinesis :** Aide à surveiller et analyser les flux de données en temps réel, permettant une prise de décision rapide et informée.

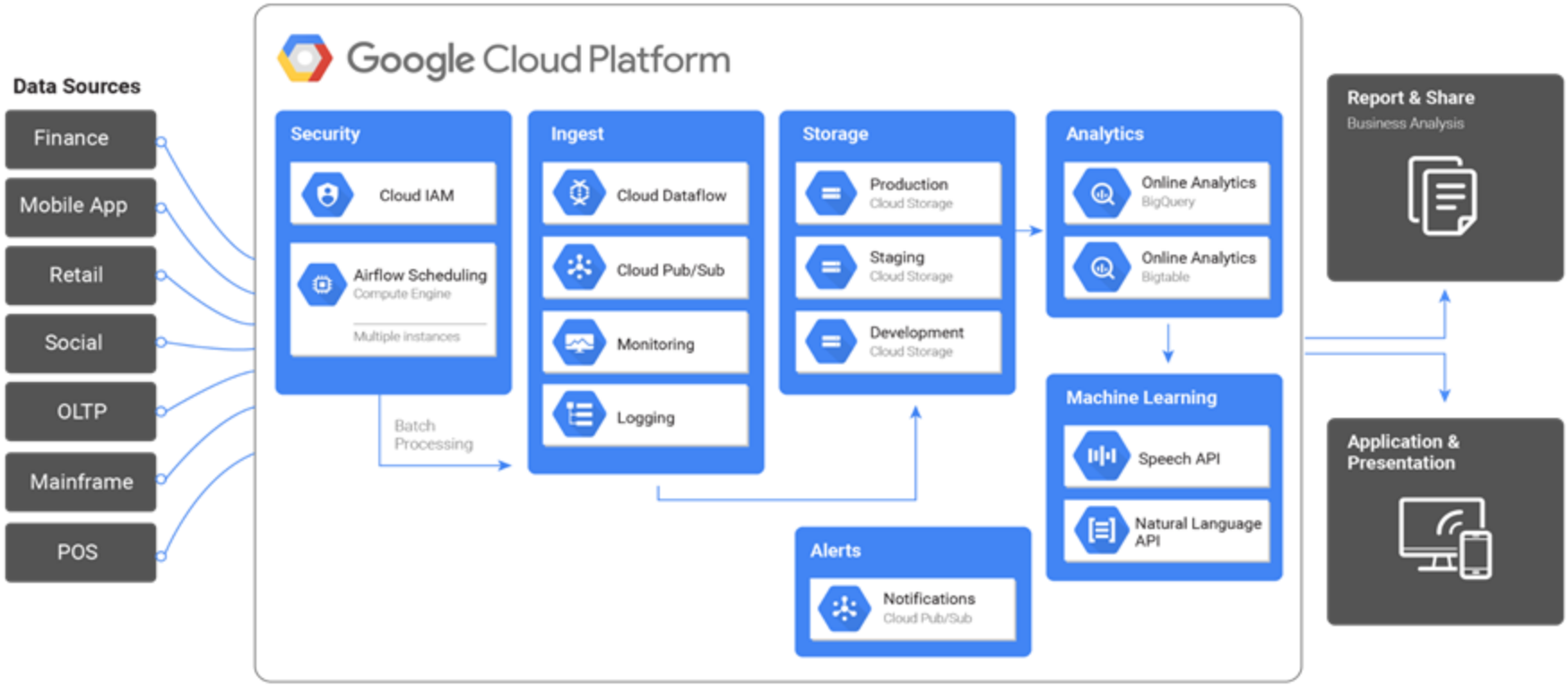
AWS HLD


Collect
Direct Connect
Snowball
Kinesis & Kinesis Firehose
Store
S3
Glacier
RDS, Aurora
DynamoDB
Redshift
CloudSearch
Elasticsearch
Analyze
EMR
Machine Learning
QuickSight
Kinesis Analytics
Athena
Data Pipeline / Glue

Google Cloud Plateform GCP

- **BigQuery :** Service de data warehouse sans serveur et scalable permettant des analyses SQL rapides sur de grandes bases de données.
  - **Exemple :** Utilisation de BigQuery pour analyser les données de trafic d'un site web et optimiser les campagnes marketing.
- **Dataflow :** Est un service géré pour le traitement de données en mode batch et stream, basé sur Apache Beam. Création rapide de clusters, intégration avec GCP Storage et BigQuery, support pour Hadoop, Spark, Pig et Hive.
  - **Exemple :** Pipeline de traitement des transactions en temps réel pour une entreprise de e-commerce.
- **Dataproc :** Simplifie la gestion des clusters Hadoop et Spark, permettant des opérations Big Data rapides et faciles.
  - **Exemple :** Utilisation de Dataproc pour analyser les données de capteurs de température dans un réseau de serres agricoles.
- **Comparaison avec AWS :**
  - **Coûts :** GCP propose un modèle de tarification flexible, souvent compétitif par rapport à AWS, surtout pour les utilisateurs ayant des charges de travail fluctuantes.
  - **Performances :** BigQuery est connu pour sa rapidité d'exécution des requêtes par rapport à Redshift.
  - **Fonctionnalités :** AWS propose une plus grande variété de services, mais GCP se distingue par la simplicité et la rapidité d'intégration de ses services Big Data.

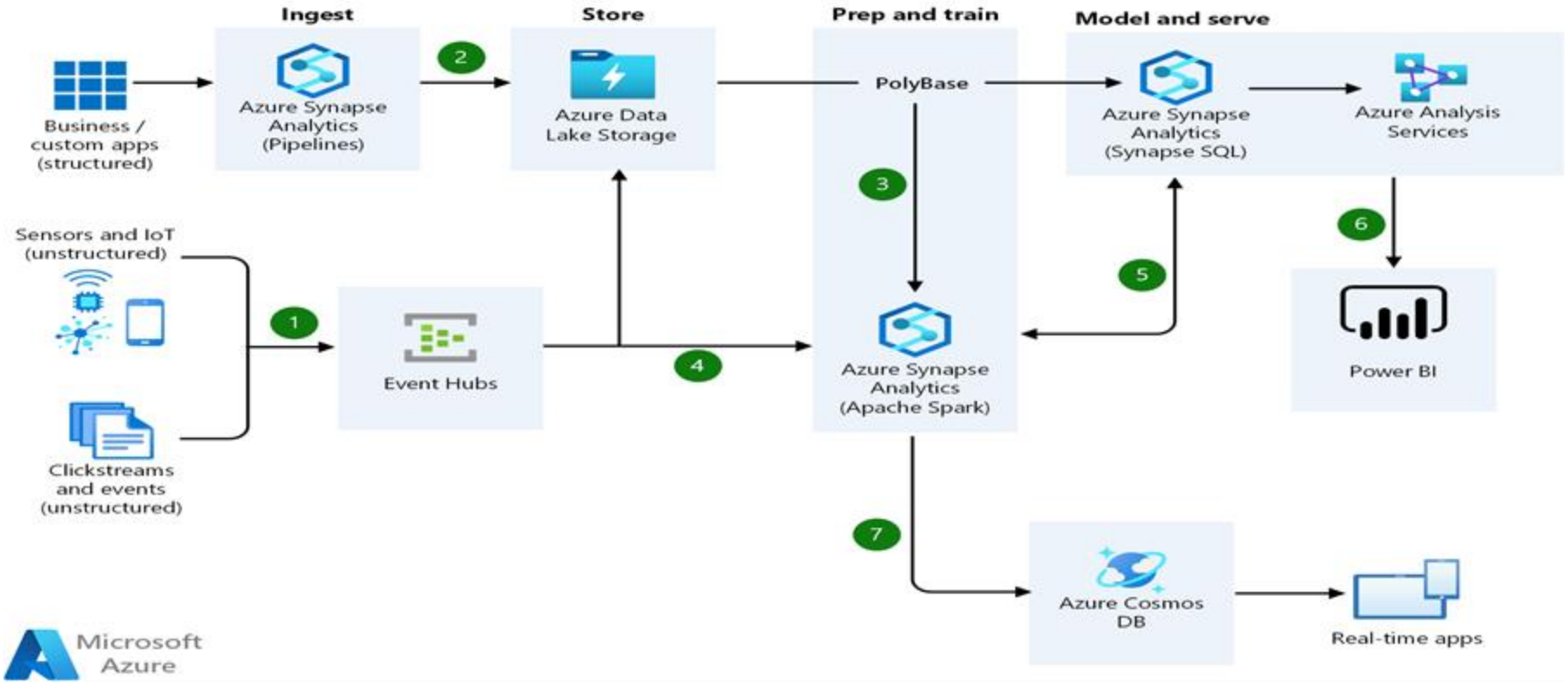
GCP HLD


Data Sources
Finance
Mobile App
Retail
Social
OLTP
Mainframe
POS
Google Cloud Platform
Security
Cloud IAM
Airflow Scheduling
Compute Engine
Multiple instances
Batch Processing
Ingest
Cloud Dataflow
Cloud Pub/Sub
Monitoring
Logging
Storage
Production
Cloud Storage
Staging
Cloud Storage
Development
Cloud Storage
Analytics
Online Analytics
BigQuery
Online Analytics
Bigtable
Machine Learning
Speech API
Natural Language API
Alerts
Notifications
Cloud Pub/Sub
Report & Share
Business Analysis
Application & Presentation

Microsoft Azure

- **HDInsight :** Est un service de traitement et analyse de données à grande échelle, intégration avec Azure Blob Storage et Data Lake, basé sur des frameworks comme Hadoop, Spark, Hive, et Kafka.
  - **Exemple :** Utilisation pour un projet de data science impliquant des algorithmes d'apprentissage automatique sur un grand jeu de données.
- **Azure Data Lake :** Est un service de stockage massif et de traitement de données, intégration avec U-SQL pour le traitement, conçu pour les applications analytiques et Big Data.
  - **Exemple :** Stockage des données de capteurs d'une entreprise de logistique et exécuter des analyses pour optimiser les routes de livraison.
- **Stream Analytics :** Est un moteur de traitement de flux en temps réel entièrement géré. Avec des requêtes SQL pour le traitement de flux, intégration avec d'autres services Azure comme **Event Hubs** et **IoT Hub**.
  - **Exemple :** Déploiement de Stream Analytics pour analyser les flux de données d'une chaîne de montage et prévenir les pannes de machines.
- **Étude de cas : eBay**
  - **Utilisation d'HDInsight :** Utiliser pour analyser de vastes ensembles de données transactionnelles et améliorer ses modèles de recommandation.
  - **Utilisation d'Azure Data Lake :** Permet à eBay de stocker des données historiques et de les analyser à des fins de business intelligence.
  - **Utilisation de Stream Analytics :** Stream Analytics aide eBay à traiter et analyser les transactions en temps réel pour détecter des fraudes potentielles.

Azure HLD


Ingest
Store
Prep and train
Model and serve
Business / custom apps (structured)
Azure Synapse Analytics (Pipelines)
2
Azure Data Lake Storage
PolyBase
Azure Synapse Analytics (Synapse SQL)
Azure Analysis Services
3
6
Sensors and IoT (unstructured)
1
Event Hubs
4
5
Azure Synapse Analytics (Apache Spark)
Power BI
Clickstreams and events (unstructured)
7
Azure Cosmos DB
Real-time apps
Microsoft Azure

Base de Données NoSQL : MongoDB

- **Objectifs :** Comprendre les caractéristiques des bases de données NoSQL, leurs cas d'utilisation, et savoir les comparer pour sélectionner la meilleure solution selon les besoins spécifiques d'un projet.

MongoDB: Caractéristiques Principales de:

- **Flexibilité du Schéma :** permet de stocker des documents JSON (ou BSON) qui peuvent avoir des structures différentes, ce qui facilite l'évolution et l'adaptation des schémas de données sans avoir à restructurer la base de données.
- **Requêtes Riches :** MongoDB offre un langage de requête puissant qui permet de filtrer, trier, agréger et des index multiples les données de manière complexe.
- **Utilisation :** Idéal pour les applications nécessitant une structure de données dynamique comme les applications web, IoT.
- **Exemple :** Déploiement de MongoDB pour une application e-commerce qui nécessite un stockage flexible pour différents types de produits avec des attributs variés.

Base de Données NoSQL : Cassandra

Cassandra : Avantages en Termes de Scalabilité et Tolérance aux Pannes :

- **Architecture Décentralisée :** Pas de single point of failure, permettant une disponibilité élevée.
- **Scalabilité Horizontale :** Cassandra est conçue pour gérer de grandes quantités de données et peut facilement évoluer horizontalement en ajoutant de nouveaux nœuds sans interruption de service.
  - *Exemple* **:** Une plateforme de réseaux sociaux qui doit stocker des milliards de messages et de connexions utilisateurs.
- **Tolérance aux pannes :** Cassandra réplique les données sur plusieurs nœuds et centres de données pour assurer la disponibilité et la résilience en cas de défaillance matérielle.
  - *Exemple :* Un service de streaming vidéo qui doit garantir la disponibilité des vidéos, même en cas de panne de certains serveurs.
- **Cas d'Utilisation :** Idéal pour les applications qui nécessitent un haut débit d'écriture, comme les systèmes de gestion de logs et les plateformes de médias sociaux.

Base de Données NoSQL : HBase

- **Description :** HBase est une base de données NoSQL distribuée et évolutive conçue pour stocker et traiter des données massives en temps réel, s'intégrant parfaitement avec l'écosystème Hadoop.
- **Stockage en Colonnes :** Conçu pour les grandes tables avec des milliards de lignes et des millions de colonnes.
- **Faible Latence :** Optimisé pour les applications nécessitant un accès rapide aux données.
- **Cas d'Utilisation :** Utilisation de HBase pour stocker les données de navigation et les historiques d'achat des utilisateurs, permettant une génération rapide de recommandations personnalisées, recherches analytiques en temps réel, systèmes de gestion de la fraude.
- **Exemple :**
  - Implémentation de HBase pour une application de surveillance des transactions financières en temps réel.
  - Un système de recommandation en temps réel pour une plateforme de e-commerce.

Elasticsearch

Utilisation pour la Recherche et l'Analyse en Temps Réel :

- **Description :** Elasticsearch est un moteur de recherche et d'analyse distribué capable d'indexer et de rechercher des données en temps réel.
- **Indexation et Recherche Full-Text :** Performant pour les recherches textuelles et la navigation par facettes.
- **Cas d'Utilisation :** Surveillance en temps réel, applications de recherche web, tableaux de bord analytiques.
- **Exemple :** Déploiement d'Elasticsearch pour fournir une fonctionnalité de recherche rapide sur un site de commerce électronique.

Comparaison des technologies et plateformes

Facteurs de choix :

- **Coût :** Certains systèmes NoSQL sont open-source et gratuits (comme Cassandra et MongoDB), tandis que les solutions gérées dans le cloud peuvent avoir des coûts variables en fonction de l'utilisation (comme AWS DynamoDB ou Azure Cosmos DB).
- **Complexité de mise en œuvre :** La facilité d'installation, de configuration et de gestion des bases de données varie. Par exemple, MongoDB est souvent considéré comme facile à démarrer, tandis que Cassandra peut nécessiter plus de connaissance en termes de mise en cluster et de gestion de la tolérance aux pannes.
- **Évolutivité :** Il est important de choisir une solution qui peut évoluer horizontalement pour gérer la croissance des données. Cassandra et HBase sont connus pour leur haute évolutivité.
- **Support communautaire :** Une grande communauté peut fournir de la documentation, des bibliothèques tierces et de l'aide en cas de problèmes. MongoDB et Elasticsearch ont de grandes communautés actives.

Comparaison des technologies et plateformes

Exemples de scénarios :

- **Sélection de technologies et de plateformes en fonction des besoins spécifiques d'une organisation :**
  - **Taille des données :** Pour une entreprise qui prévoit de stocker des pétaoctets de données, Cassandra ou HBase pourrait être plus approprié en raison de leur évolutivité.
  - **Fréquence d'accès :** Pour des applications nécessitant un accès fréquent et rapide aux données, MongoDB ou Elasticsearch pourrait être choisi pour leur performance en lecture rapide.
  - **Budget :** Pour une startup avec un budget limité, une solution open-source comme MongoDB ou Cassandra pourrait être plus adaptée.

Sommaire

- Histoire et évolution du Big Data
- Introduction aux Systèmes Distribués
- Technologies et Plates-formes pour le Big Data
- **Traitement de Flux de Données**
- Stockage et Gestion de Données à Grande Échelle
- Sécurité et Confidentialité dans les Systèmes Distribués
- Conception et Déploiement de Systèmes Distribués
- Étude de Cas et Projets Pratiques

Traitements de Flux de Données

Introduction au Traitement de Flux de Données

Objectifs :

- Comprendre les concepts fondamentaux du traitement de flux de données.
- Explorer les technologies et outils utilisés pour le traitement en temps réel.
- Appliquer les concepts à des cas d'usage pratiques dans l'industrie.

Traitement de Flux vs Traitement par Lots : Le traitement de flux de données traite les données en continu, au fur et à mesure qu'elles sont reçues, contrairement au traitement par lots qui accumule les données avant de les traiter.

- **Concepts importants :**
  - **Flux de Données :** Séquence continue de données générées par des sources comme des capteurs, des logs de serveur, des clics d'utilisateurs.
  - **Fenêtres Temporelles :** Méthode de découpage des flux de données en intervalles de temps pour permettre leur agrégation et analyse. Types de fenêtres : fixes, glissantes, et par sessions.
  - **Transformations de Flux :** Opérations appliquées aux flux de données, comme les filtrages, les regroupements, et les jointures.

Architecture du Traitement de Flux

Modèle Pub/Sub (Publication/Abonnement) :

- **Définition :** Modèle de communication dans lequel les producteurs (publishers) envoient des messages à des sujets (topics), et les consommateurs (subscribers) reçoivent ces messages via des abonnements.
- **Avantages :** Permet une scalabilité horizontale et une gestion asynchrone des messages.

Concept de Fenêtrage :

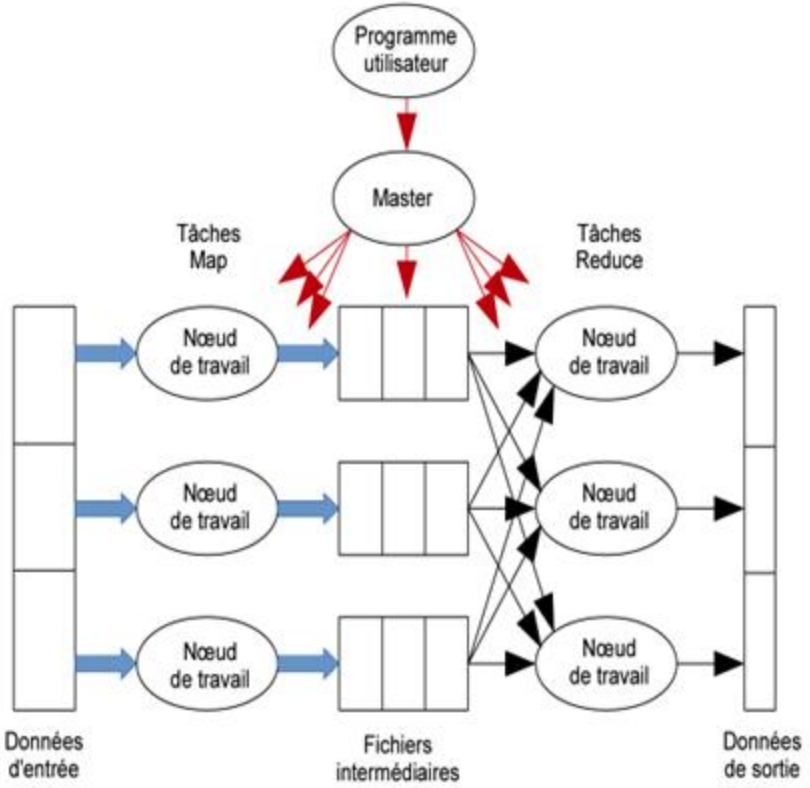
- **Types de Fenêtres :**
  - **Fenêtres Fixes :** Segments de temps fixes (e.g., chaque minute).
  - **Fenêtres Glissantes :** Fenêtres qui se chevauchent sur une période définie.
  - **Fenêtres de Session :** Définies par des périodes d'inactivité entre les événements.

Techniques de Gestion :

- **En Mémoire :** Utilisé pour des performances élevées, mais limité par la taille de la mémoire disponible.
- **Sur Disque :** Permet de gérer de grands volumes de données, au prix d'une latence accrue.

Traitement de flux Hadoop : MapReduce

- L'ensemble de données à traiter est découpé en fragments (***chunks***).
- Chaque tâche ***Map*** est assignée à un nœud de calcul qui reçoit un ou plusieurs fragments que la tâche *Map* transforme en une séquence de paires **[clé, valeur]**.
- Chaque tâche ***Reduce*** est associée à une ou plusieurs **clés** et est assignée à un nœud de calcul.
- Les paires (clé, valeur) produites par les *Map* sont groupées par clés et stockées sur les nœuds de calcul qui exécuteront les tâches *Reduce* respectives **(étape *shuffle*)**.
- Chaque tâche *Reduce* combine, pour chaque clé qui lui est associée, les valeurs des paires [clé, valeur] avec cette clé ; les résultats sont stockés et constituent le résultat du traitement.



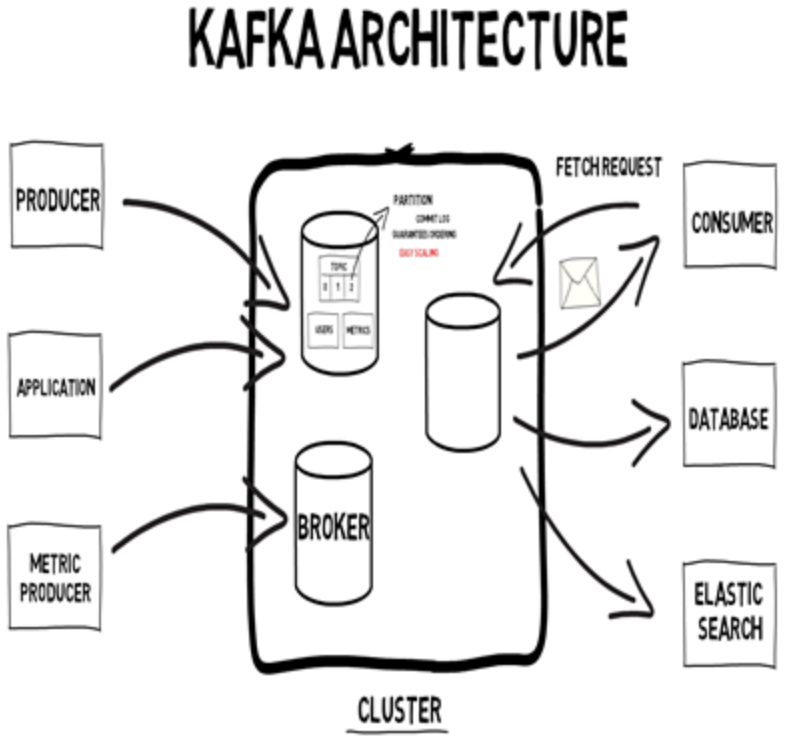
Apache Kafka

Fonctionnalités :

- **Système de Messagerie Distribué :** Permet la transmission de messages entre producteurs et consommateurs.
- **Scalabilité :** Peut évoluer horizontalement en ajoutant des brokers pour gérer plus de flux de données.
- **Persistance des Messages :** Stocke les messages sur disque, permettant de les relire si nécessaire.

Architecture et Composantes :

- **Topic** : Catégorie logique dans laquelle les messages sont publiés par les producteurs.
- **Producteur (Producer)** : Envoie des messages aux topics Kafka.
- **Consommateur (Consumer)** : Lit les messages des topics Kafka.
- **Broker :** Serveur Kafka qui gère le stockage et la distribution des messages.



Apache Flink & Storm

Apache Flink :

- **Introduction :** Plateforme puissante pour le traitement de données en temps réel, avec un support natif pour les flux de données.
- **Comparaison avec Apache Spark Streaming :**
  - **Latence :** Flink offre une latence plus faible grâce à son traitement natif des flux.
  - **Gestion des États :** Flink excelle dans la gestion des états complexes, ce qui le rend adapté pour les applications nécessitant un suivi d'état avancé.

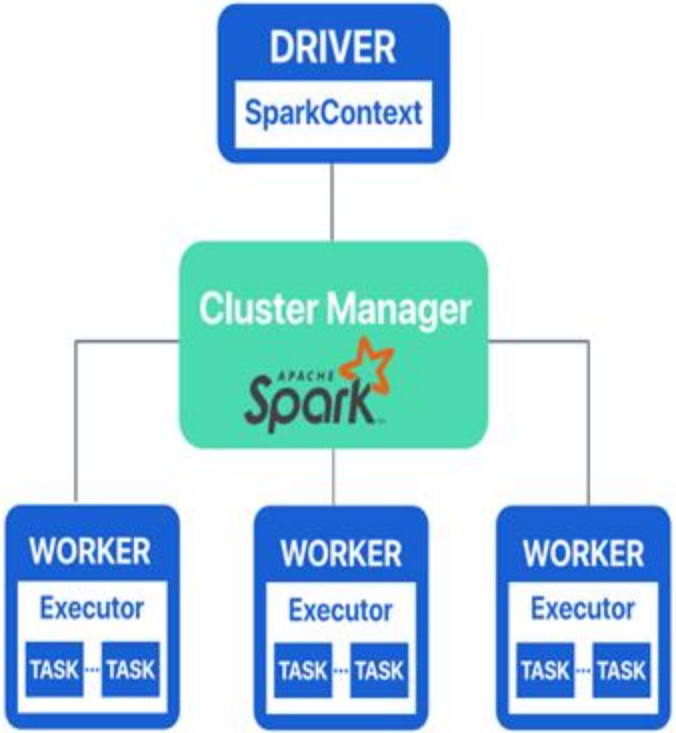
Apache Storm :

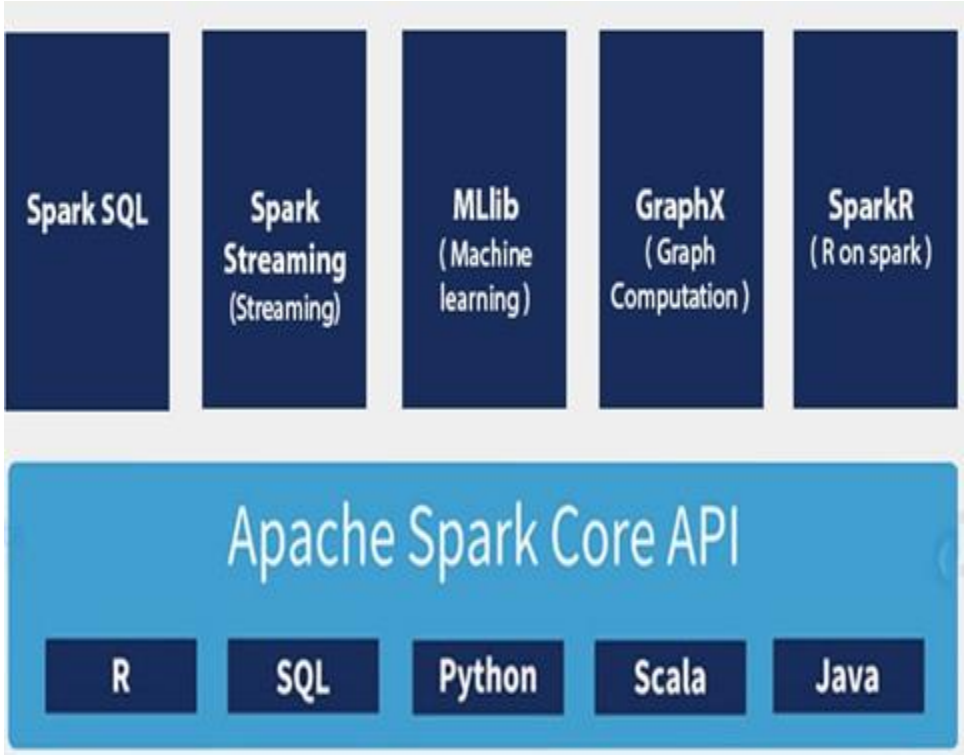
- **Fonctionnalités :**
  - **Traitement de Flux Distribués :** Conçu pour un traitement rapide et fiable des flux de données.
  - **Tolérance aux Pannes :** Assure la continuité du traitement même en cas de défaillance de composants.
- **Exemples d'Utilisation :** Surveillance en temps réel, analyses financières, systèmes de recommandation.

Apache Spark : Composant principaux de l'Architecture

- **Driver** : Est le cœur de l'application Spark. Il est responsable de la planification, de la distribution et de la surveillance des tâches sur le cluster.
- **Cluster Manager :** Est responsable de la gestion des ressources dans le cluster. Spark peut fonctionner avec différents gestionnaires de cluster, tels que Standalone, Apache Mesos, Hadoop YARN, ou Kubernetes.
- **Executors** : Sont les processus qui exécutent les tâches sur les nœuds de travail. Chaque application Spark a ses propres Executors, qui sont lancés par le Cluster Manager.
- **DAG Scheduler** : (Directed Acyclic Graph Scheduler) est responsable de la planification des tâches dans Spark. Il divise le code de l'application en un graphe acyclique dirigé (DAG) de tâches.
- **Task Scheduler** : Est responsable de l'assignation des tâches aux Executors. Il prend les tâches planifiées par le DAG Scheduler et les distribue aux Executors disponibles.
- **Block Manager** : Est responsable de la gestion des données en cache dans Spark. Il stocke les données intermédiaires et les résultats des tâches dans la mémoire des nœuds de travail.

Apache Spark : Composant principaux de l'Architecture





Apache Spark : Fonctionnement

- **Soumission d'une Application** : Lorsqu'une application Spark est soumise, le Driver initialise le SparkContext et se connecte au Cluster Manager.
- **Création du DAG** : Le code de l'application Spark est traduit en un DAG de tâches par le DAG Scheduler.
- **Exécution des Tâches** : Le Task Scheduler prend les tâches planifiées par le DAG Scheduler et les distribue aux Executors disponibles.
- **Gestion des Échecs** : En cas de défaillance d'un nœud de travail ou d'un Executor, le Task Scheduler ré-assigne les tâches échouées à d'autres Executors. Le DAG Scheduler peut également re-planifier les tâches en fonction des dépendances du DAG.

Optimisation des Performances

- **Caching** : En stockant les données intermédiaires en mémoire, Spark réduit les lectures et écritures de données sur le disque, ce qui accélère l'exécution des tâches.
- **Partitionnement** : Spark divise les données en partitions et distribue ces partitions aux Executors pour un traitement parallèle.
- **Shuffle** : Est une opération coûteuse en termes de performance, car elle implique le transfert de données entre les nœuds de travail. Spark optimise le shuffle en minimisant les transferts de données et en utilisant des algorithmes efficaces pour le tri et l'agrégation des données.
- **Broadcast Variables** : Elles permettent de réduire les transferts de données en envoyant une seule copie des données à chaque nœud de travail.

Défis et Solutions dans le Traitement de Flux de Données

Défis :

- **Latence :** La latence est le délai entre l'arrivée des données dans le système et leur traitement. Dans les systèmes de traitement de flux, la latence doit être minimale pour permettre des analyses en temps réel.
- **Débit :** Le débit est le volume de données pouvant être traité par le système par unité de temps. Un débit élevé est nécessaire pour gérer des flux de données massifs.

Solutions :

- **Traitement en Mémoire :** Utilisation de frameworks comme Apache Spark Streaming et Apache Flink, qui traitent les données en mémoire, réduisant ainsi la latence.
- **Parallélisme et Partitionnement :** Répartition des tâches sur plusieurs nœuds pour augmenter le débit et équilibrer la charge.
- **Optimisation des Algorithmes :** Algorithmes de traitement optimisés pour minimiser les temps de calcul et les opérations de lecture/écriture.

Tolérance aux Pannes dans le Traitement de Flux de Données

Défis :

- **Pannes de Nœuds :** Perte de données ou d'état si un nœud du cluster tombe en panne.
- **Continuité de Service :** Maintenir le service opérationnel malgré des pannes partielles.

Solutions :

- **Réplication des Données :** Copie des données sur plusieurs nœuds pour assurer la disponibilité en cas de défaillance.
- **Journalisation des Transactions (WAL - Write-Ahead Logging) :** Enregistrement des opérations avant leur exécution pour permettre une reprise après une panne.
- **Supervision et Reprise Automatique :** Utilisation de systèmes comme ZooKeeper pour surveiller les nœuds et redémarrer automatiquement les services défaillants.

Sommaire

- Histoire et évolution du Big Data
- Introduction aux Systèmes Distribués
- Technologies et Plates-formes pour le Big Data
- Traitement de Flux de Données
- **Stockage et Gestion de Données à Grande Échelle**
- Sécurité et Confidentialité dans les Systèmes Distribués
- Conception et Déploiement de Systèmes Distribués
- Étude de Cas et Projets Pratiques

Stockage et Gestion de Données à grande Échelle

Introduction au Stockage de Données

- **Définition :** Réfère à la capacité de stocker des volumes massifs de données, souvent en pétaoctets ou plus, sur des systèmes distribués. C'est crucial pour les applications de Big Data qui génèrent d'énormes quantités de données.

Différences avec les Systèmes Traditionnels :

- Les **systèmes de stockage traditionnels** (comme les bases de données relationnelles) sont conçus pour des volumes de données plus modestes et une structure de données fixe.
- Les **systèmes de stockage Big Data** (comme HDFS, NoSQL) sont conçus pour gérer des volumes massifs de données non structurées ou semi-structurées, avec une scalabilité horizontale.

Problématiques Courantes :

- **Scalabilité** : Capacité d'un système à s'étendre pour gérer des volumes croissants de données.
- **Performance** : Capacité d'un système à traiter les données rapidement, en minimisant la latence.
- **Coût** : Coûts associés au stockage et au traitement des données, y compris les coûts matériels et opérationnels.
- **Gestion de la redondance** : Réplication des données pour assurer la tolérance aux pannes et la disponibilité.

Hadoop Distributed File System HDFS

- HDFS est un système de fichiers distribué conçu pour stocker de grandes quantités de données en les divisant en blocs et en les distribuant sur plusieurs nœuds. Il assure une haute disponibilité et une tolérance aux pannes en répliquant les blocs de données.
- **Fonctionnalités principales** :
  - **Scalabilité** : Peut stocker des pétaoctets de données en ajoutant de nouveaux nœuds.
  - **Tolérance aux pannes** : Réplication des données pour assurer la continuité des services.
  - **Intégration** : S'intègre bien avec d'autres outils de l'écosystème Hadoop comme MapReduce, Hive et HBase.
- **Architecture :**
  - **NameNode :** Gère le système de fichiers et les métadonnées des fichiers.
  - **DataNodes :** Stockent les blocs de données réels et servent les requêtes de lecture/écriture.
  - **Réplication :** Chaque bloc de données est répliqué sur plusieurs DataNodes pour assurer la tolérance aux pannes.

Bases de Données Distribuées : Redis & Memcached

- **Redis :** Est une base de données en mémoire open source, souvent utilisée comme cache ou broker de messages. Elle prend en charge plusieurs structures de données comme les chaînes, les listes, les ensembles, et les hashes.
  - **Avantages :**
    - **Performance Élevée :** Temps de réponse en millisecondes grâce au stockage en mémoire.
    - **Persistante :** Option pour écrire les données sur disque à intervalles réguliers pour assurer la durabilité.
- **Memcached :** Système de cache en mémoire distribué conçu pour accélérer les applications web en réduisant la charge sur les bases de données en backend.
  - **Avantages :**
    - **Simplicité :** Facile à configurer et à intégrer.
    - **Performance :** Réduit la latence en stockant les résultats des requêtes fréquentes.
    - **Scalabilité** : Capacité à évoluer horizontalement en ajoutant plus de nœuds.

Systèmes de Stockage Distribués

MongoDB : MongoDB est une base de données NoSQL orientée documents qui stocke les données en JSON (ou BSON : une version binaire de JSON) flexible, ce qui facilite le stockage et la gestion des données semi-structurées.

- **Avantages :**
  - **Schéma Flexible :** Permet de gérer des données non structurées ou semi-structurées.
  - **Indexation Avancée :** Améliore les performances des requêtes.

Cassandra :

- **Architecture peer-to-peer** : Tous les nœuds de Cassandra sont égaux, il n'y a pas de nœud maître unique, ce qui améliore la scalabilité et la tolérance aux pannes.
- **Modèle de données orienté colonnes** : Conçu pour les écritures rapides et les requêtes en temps réel à grande échelle

HBase :

- **Système de stockage basé sur les colonnes** : Est un système de base de données NoSQL qui utilise un modèle de données orienté colonnes, idéal pour les opérations de lecture/écriture en vrac.
- **Utilisé avec HDFS** : HBase fonctionne au-dessus de HDFS pour fournir des opérations de lecture/écriture en temps réel sur des données massives.

Techniques de Gestion de Données Massives : Partitionnement

Stratégies :

- **Partitionnement Horizontal :** Diviser les données en plusieurs tables ou partitions plus petites, où chaque partition contient un sous-ensemble de lignes de données. Chaque partition est stockée sur un nœud différent du cluster
- **Partitionnement Vertical :** Diviser les données en plusieurs tables ou partitions plus petites, où chaque partition contient un sous-ensemble de colonnes de données. Les lignes restent les mêmes dans chaque partition.

Avantages et Inconvénients :

- **Partitionnement horizontal :**
  - *Avantages :* Scalabilité, tolérance aux pannes, facilité de gestion des volumes de données.
  - *Inconvénients :* Complexité de la requête distribuée, besoin de mécanismes de répartition des shards.
- **Partitionnement vertical :**
  - *Avantages :* Optimisation des requêtes spécifiques, amélioration des performances de lecture.
  - *Inconvénients :* Difficile à maintenir si les schémas de données changent fréquemment, nécessite des jointures fréquentes.

Indexation & Compression

Indexation :

- **Objectif :** Améliorer la vitesse de récupération des données.
- **Types d'Index :**
  - **B-trees :** Structure d'index équilibrée utilisée dans les bases de données relationnelles pour permettre des insertions, suppressions et recherches rapides.
  - **Index Inversés :** Utilisés dans les moteurs de recherche comme Elasticsearch pour permettre des recherches textuelles rapides.

Compression :

- **Compression :** Réduction de la taille des données stockées pour économiser de l'espace et améliorer les performances de lecture/écriture
- **Techniques :**
  - **Sans Perte :** ZIP, Snappy, etc., qui réduisent la taille des données sans en perdre.
  - **Avec Perte :** Utilisé dans le multimédia pour réduire davantage les tailles de fichiers.

Optimisation, Performances, Sécurité et Gestion des Accès

Caching : Réduire le temps d'accès aux données fréquemment utilisées.

- ➢ **Redis :** Stockage clé-valeur en mémoire, utilisé pour des réponses rapides.
- ➢ **Memcached :** Stockage en mémoire destiné à alléger la charge de base de données.

Load Balancing :

- ➢ Répartition de charge entre plusieurs serveurs pour garantir l'équilibre de l'utilisation des ressources.
- ➢ Implémentation de load balancers dans les environnements cloud pour gérer les flux massifs de données.

Contrôle d'Accès :

- ❖ **RBAC (Role-Based Access Control) :** Stratégie de contrôle d'accès où les utilisateurs se voient attribuer des rôles qui déterminent leurs permissions.
- ❖ **Politiques de Sécurité :** Définition de règles pour accéder aux données sensibles.

Cryptage des Données :

- ❖ **Au Repos :** Chiffrement des données stockées sur disque.
- ❖ **En Transit :** Utilisation de protocoles de cryptage comme SSL/TLS pour protéger les données lorsqu'elles sont transmises entre les systèmes.

Sommaire

- Histoire et évolution du Big Data
- Introduction aux Systèmes Distribués
- Technologies et Plates-formes pour le Big Data
- Traitement de Flux de Données
- Stockage et Gestion de Données à Grande Échelle
- **Sécurité et Confidentialité dans les Systèmes Distribués**
- Conception et Déploiement de Systèmes Distribués
- Étude de Cas et Projets Pratiques

Sécurité et Confidentialité dans les Systèmes Distribués

Introduction à la sécurité dans les systèmes distribués

Objectifs :

- Comprendre les principaux défis de sécurité et de confidentialité dans les systèmes distribués.
- Explorer les techniques et les outils de sécurité pour protéger les données et les communications dans les systèmes distribués.
- Analyser les aspects de la confidentialité des données et les implications réglementaires.

Défis :

- Multiplicité des points de vulnérabilité
- Gestion complexe des autorisations

Risques courants :

- **Attaques par déni de service (DDoS)** : Saturation des ressources pour rendre le système indisponible.
- **Attaques de l'homme du milieu (MITM)** : Interception des communications pour voler ou modifier des données.
- **Compromis des nœuds** : Prise de contrôle des nœuds pour exécuter des actions malveillantes.

Cryptographie pour les systèmes distribués

- **Chiffrement des données :**
  - **Algorithmes symétriques (AES)** : Clé unique pour le chiffrement et le déchiffrement.
  - **Algorithmes asymétriques (RSA)** : Paire de clés publique et privée pour sécuriser les communications.
- **Signature numérique et certificats :**
  - **Signatures numériques** : Assurer l'intégrité des données et l'authenticité de l'expéditeur.
  - **Certificats** : Délivrés par des autorités de certification (CA) pour sécuriser les échanges.
- **Protocole SSL/TLS :**
  - **Fonctionnement** : SSL/TLS assure la sécurisation des communications réseau par le chiffrement des données en transit.
  - **Mise en œuvre** : Utilisé dans des systèmes distribués pour protéger les données lors de leur transmission.

Authentification et gestion des accès

Modèles d'authentification :

- **JWT (JSON Web Token)** : Utilisé pour la transmission sécurisée d'informations entre parties sous forme de jeton.
- **OAuth 2.0** : Protocole d'autorisation qui permet de fournir un accès délégué à des ressources.
- **Kerberos** : Protocole réseau pour l'authentification sécurisée.

Contrôle d'accès :

- **RBAC (Role-Based Access Control)** : Autorisations basées sur les rôles des utilisateurs dans le système.
- **ABAC (Attribute-Based Access Control)** : Les autorisations sont basées sur des attributs tels que le rôle, l'emplacement, l'heure, etc.

Gestion des identités et des accès (IAM) :

- **IAM** : Gestion centralisée des identités pour contrôler l'accès aux ressources distribuées.
- **Plateformes cloud** : AWS, Azure, et GCP fournissent des services IAM pour sécuriser les environnements cloud.

Sécurité des communications

Sécurisation des protocoles réseau :

- **HTTP sécurisé (HTTPS)** : Chiffrement des données en transit à l'aide de TLS pour assurer la confidentialité et l'intégrité.
- **FTP sécurisé (SFTP)** : Utilisation de SSH pour sécuriser les transferts de fichiers.
- **WebSocket sécurisé (WSS)** : Communication bidirectionnelle sécurisée via TLS pour les applications en temps réel.
- **VPN et réseaux privés virtuels** : Création de tunnels chiffrés entre nœuds pour sécuriser les communications.

Pare-feux et systèmes de détection d'intrusion (IDS/IPS) :

- **Pare-feux** : Filtrage des paquets réseau pour bloquer les connexions non autorisées.
- **IDS/IPS** : Détection et prévention des tentatives d'intrusion en surveillant les flux réseau pour identifier des comportements suspects.

Isolation des réseaux :

- **Virtual Private Clouds (VPCs)** : Segmenter les ressources dans un environnement cloud pour limiter la portée des attaques.
- **Segmentation du réseau** : Isoler différentes parties du réseau pour minimiser la propagation des menaces.

Protection des données, Audit et conformité

Anonymisation et pseudonymisation :

- **Anonymisation** : Suppression des informations d'identification pour protéger la vie privée.
- **Pseudonymisation** : Remplacement des données identifiantes par des alias.

Réglementations sur la confidentialité :

- **APDP et les lois connexes**

Gestion des incidents de sécurité :

- **Processus de gestion des incidents** : Détection, évaluation de l'impact, confinement, éradication, récupération.
- **Outils de surveillance** : SIEM (Security Information and Event Management) pour la détection des menaces.

Audits de sécurité :

- **Importance des audits** : Évaluer la posture de sécurité, identifier les vulnérabilités, et assurer le respect des politiques internes.
- **Techniques d'audit** : Scans de vulnérabilités (Nessus, OpenVAS), examens manuels des configurations et des logs.

Conformité aux standards de sécurité :

- **ISO 27001** : Norme de gestion de la sécurité de l'information.
- **PCI DSS** : Norme de sécurité pour les transactions par carte de crédit.
- **SOC 2** : Critères de gestion des données pour protéger les informations des clients.

Sommaire

- Histoire et évolution du Big Data
- Introduction aux Systèmes Distribués
- Technologies et Plates-formes pour le Big Data
- Traitement de Flux de Données
- Stockage et Gestion de Données à Grande Échelle
- Sécurité et Confidentialité dans les Systèmes Distribués
- **Conception et Déploiement de Systèmes Distribués**
- Étude de Cas et Projets Pratiques

Conception et Déploiement de Systèmes Distribués

Introduction à la Conception de systèmes distribués

Objectifs :

- Apprendre à concevoir des systèmes distribués robustes, évolutifs et tolérants aux pannes.
- Comprendre les méthodologies de déploiement de systèmes distribués.
- Acquérir les compétences pratiques pour implémenter et gérer des systèmes distribués.

Principes :

- Modularité, Cohérence, Disponibilité, Partitionnement.

Modèle CAP :

- Cohérence (Consistency), Disponibilité (Availability), Tolérance au Partitionnement (Partition Tolerance), Implications : Choix entre cohérence forte et disponibilité en fonction des besoins de l'application.

Choix architecturaux :

- **Architecture monolithique vs microservices** :
  - *Monolithique* : Simplicité de déploiement mais moins flexible pour la scalabilité.
  - *Microservices* : Flexibilité et évolutivité, avec des services indépendants.
- **Protocole de communication** :
  - *REST* : Basé sur HTTP, stateless, et largement utilisé.
  - *gRPC* : Protocole plus performant, supportant le streaming bidirectionnel.
  - *Message Brokers* : Utilisation de systèmes comme Kafka ou RabbitMQ pour la communication asynchrone.

Modèles de conception

Partitionnement des données :

- Partitionnement horizontal;
- Partitionnement vertical.
- Considérations : Latence, volume des données, et nature des requêtes.

Tolérance aux pannes et redondance :

- Réplication de données;
- Redondance de services;
- Techniques de consensus :
  - *Raft* : Simplicité et efficacité pour maintenir un leader unique.
  - *Paxos* : Plus complexe, mais robuste pour atteindre un consensus distribué.

Équilibrage de charge :

- **Importance** : Prévenir les goulots d'étranglement et améliorer la performance globale.
- **Stratégies d'équilibrage** :
  - *Round-robin* : Distribution égale des requêtes.
  - *Least connections* : Rediriger les requêtes vers les nœuds avec le moins de connexions actives.
  - *Random* : Sélection aléatoire des nœuds pour répartir la charge.

Déploiement de systèmes distribués

Orchestration de conteneurs :

- **Introduction à Docker et Kubernetes :**
  - *Docker* : Plateforme de conteneurisation permettant de packager des applications avec leurs dépendances dans des conteneurs.
  - *Kubernetes* : Système d'orchestration pour déployer, gérer et faire évoluer des applications conteneurisées.
- **Utilisation de Kubernetes :** Déploiement de microservices (pods), gestion de la découverte de services(avec DNS) et scaling automatique (avec l'utilisation autoscaler horizontal).

Infrastructure as Code (IaC) :

- **Utilisation de Terraform et Ansible :**
  - *Terraform* : Outil pour construire, modifier et versionner l'infrastructure en toute sécurité et efficacement.
  - *Ansible* : Outil d'automatisation pour provisionner l'infrastructure, déployer des applications et gérer la configuration.

Stratégies de déploiement :

- **Approches de déploiement continu (CI/CD) :**
  - Mise en œuvre d'intégrations et de déploiements continus pour automatiser le processus de build, test, et déploiement.
- **Blue-Green Deployments :**
  - Maintenir deux environnements identiques, un "bleu" (production actuelle) et un "vert" (nouvelle version), permettant des mises à jour sans interruption.
- **Canary Releases :** Déployer une nouvelle version à un sous-ensemble d'utilisateurs pour tester avant un déploiement

Gestion des états et de la persistance

Stateless vs Stateful Services :

- **Services Stateless** :
  - *Définition* : Services ne stockant aucune donnée de session entre les requêtes.
  - *Exemple* : Idéal pour les services web, API, et microservices nécessitant une scalabilité rapide.
- **Services Stateful** :
  - *Définition* : Services qui maintiennent des informations de session ou d'état entre les requêtes.
  - *Exemple* : Pour les applications nécessitant une session utilisateur continue, comme les bases de données ou les sessions de chat.

Bases de données distribuées :

- **Déploiement et gestion** :
  - *Cassandra* : Base de données NoSQL hautement disponible et partitionnée.
  - *MongoDB* : Base de données orientée document
  - *Redis* : Utilisé pour le caching.
- **Considérations de conception** :
  - *Transactions distribuées* : Gestion des transactions couvrant plusieurs nœuds ou bases de données.
  - *Cohérence éventuelle* : Modèle où les mises à jour sont propagées progressivement à tous les nœuds.

Performance, Scalabilité, Monitoring et Maintenance

Scalabilité horizontale vs verticale :

- **Scalabilité horizontale** : Ajouter plus de machines ou de nœuds pour répartir la charge.
- **Scalabilité verticale** : Augmenter la puissance d'une seule machine (CPU, RAM, stockage).
- **Techniques de scalabilité** : Sharding (Partitionner), Caching (Redis), Réplication.

Optimisation des performances :

- **Réduction de la latence et amélioration du débit** : Basculement des charges et Minimisation des goulots d'étranglement.
- **Profilage et surveillance** :
  - *Profilage* : Analyse des performances des applications pour identifier les sections de code ou les requêtes lentes.
  - *Outils* : Utilisation de traceurs d'application comme Jaeger, et outils de profiling comme YourKit.

Surveillance des systèmes distribués :

- **Outils de monitoring** :
  - *Prometheus* : Collecte et stockage de métriques sous forme de séries temporelles.
  - *Grafana* : Visualisation des métriques collectées pour une analyse facile.
  - *ELK Stack (Elasticsearch, Logstash, Kibana)* : Analyse des journaux et gestion centralisée des logs.
- **Mise en place de métriques clés** : Disponibilité (uptime), Utilisation des ressources (CPU, RAM) Alertes.

Maintenance et mise à jour :

- **Stratégies de mise à jour** : Mises à jour continues, Canary releases.
- **Gestion des versions et migrations de données** : Contrôle des versions, Migrations de données