

kafka

Sommaire

1. **Concepts fondamentaux**
2. **Architecture d'une plateforme kafka**
3. **Gestion d'un cluster kafka**
4. **kafka Connect**
5. **Streaming**
6. **Sécuriser un cluster kafka**
7. **Optimiser les performances d'un cluster kafka**
8. **Gérer un cluster kafka en production**

Introduction kafka

Introduction kafka

Origine

Apache kafka est un système de messagerie distribué open-source qui a été initialement développé par LinkedIn en 2011. Il a été conçu pour répondre aux besoins croissants de LinkedIn en matière de traitement de flux de données en temps réel. kafka a été rapidement adopté par la communauté open-source après sa contribution à la Apache Software Foundation en tant que projet de niveau supérieur.

Disponible sous Github : <https://github.com/apache/kafka>

Introduction kafka

Origine

- Le projet intègre l'incubateur Apache Incubator le 23 octobre 2012. En novembre 2014, plusieurs ingénieurs créateurs de kafka chez LinkedIn créent une nouvelle société nommée Confluent avec pour axe le logiciel **kafka**.
- **kafka** est utilisé principalement pour la mise en place de « data pipeline » temps réel mais ce n'est pas sa seule application possible dans le monde de l'entreprise.
- Il est aussi de plus en plus utilisé dans les architectures micro services comme système d'échange, dans la supervision temps réel et dans l'IOT.
- **kafka** apporte sa capacité à ingérer et diffuser une grande quantité de données, couplé à un framework de data stream processing, il permet le traitement complexe et en temps réel des données.

Introduction kafka

But

- Le but principal de kafka est de fournir une plateforme robuste, scalable, et efficace pour le traitement de flux de données en temps réel.
- kafka est spécialement conçu pour gérer de hauts volumes de données en offrant une haute disponibilité et une durabilité
- Il permet de publier, de stocker et de traiter des flux de données entre différentes applications et systèmes de manière asynchrone.

Introduction kafka

Fonctionnalités

kafka combine trois capacités principales :

1. **Système de messagerie:** Il permet la transmission de messages entre producteurs (qui publient les données) et consommateurs (qui reçoivent les données).
2. **Stockage de données:** kafka stocke un grand volume de messages de manière durable et sécurisée.
3. **Traitement de flux:** Il peut traiter les données en temps réel à mesure qu'elles arrivent, ce qui est idéal pour les analyses en temps réel et le monitoring.

Introduction kafka











Cas d'utilisation

- Pour traiter les paiements et les transactions financières en temps réel, comme dans les bourses, les banques et les assurances.
- Pour suivre et surveiller les voitures, les camions, les flottes et les expéditions en temps réel, par exemple dans la logistique et l'industrie automobile.
- Pour capturer et analyser en continu les données des capteurs des appareils IoT ou d'autres équipements, comme dans les usines et les parcs éoliens.
- Pour collecter et réagir immédiatement aux interactions et aux commandes des clients, par exemple dans le commerce de détail, l'hôtellerie et le voyage, ainsi que dans les applications mobiles.
- Surveiller les patients hospitalisés et prévoir les changements de leur état afin de garantir un traitement rapide en cas d'urgence.
- Connecter, stocker et mettre à disposition les données produites par les différentes divisions d'une entreprise.
- Servir de base aux plateformes de données, aux architectures basées sur les événements et aux microservices.

Introduction kafka

Ils l'utilisent...

- Avec plus de 1 000 cas d'utilisation de **kafka**, certains avantages courants sont la création de pipelines de données, l'exploitation de flux de données en temps réel, l'activation de mesures opérationnelles et l'intégration de données à travers d'innombrables sources.
- Aujourd'hui, **kafka** est utilisé par des milliers d'entreprises, dont plus de 80 % des sociétés Fortune 100. Parmi celles-ci figurent Box, Goldman Sachs, Target, Cisco, Intuit, etc.

 <p>Apache Kafka est utilisé chez AddThis pour collecter les événements générés par notre réseau de données et transmettre ces données à nos clusters d'analyse et à notre plateforme d'analyse Web en temps réel.</p>	 <p>adidas utilise Kafka comme cœur de la plateforme Fast Data Streaming, intégrant les systèmes sources et permettant aux équipes de mettre en œuvre le traitement des événements en temps réel pour les solutions de surveillance,</p>	 <p>Apache Kafka alimente l'épine dorsale du pipeline de données d'Agoda avec des milliards d'événements diffusés quotidiennement dans plusieurs centres de données. La majorité des événements sont destinés aux systèmes analytiques et influencent directement les décisions commerciales sur l'une des plateformes de réservation de voyages en ligne à la croissance la</p>	 <p>AGORA est une plateforme de données pour Apache Kafka qui encourage la collaboration autour de la transparence et de la propriété des</p>
 <p>Aiven est une plateforme cloud pour les technologies open source. Nous fournissons Apache Kafka en tant que service géré sur des cloud publics et l'utilisons en interne pour exécuter et surveiller notre plateforme de dizaines de</p>	 <p>AllegroGraph et Kafka sont utilisés ensemble en tant que plateforme Entity Event Knowledge Graph dans divers contextes tels que les centres d'appels, les hôpitaux, les compagnies d'assurance, les organisations aéronautiques et les sociétés financières. En couplant AllegroGraph à Kafka, les utilisateurs peuvent créer un moteur de décision en temps réel qui produit des flux d'événements en temps réel basés sur des calculs déclenchant des actions</p>	 <p>Apache Kafka est utilisé à la fois pour le traitement des données en temps réel et par lots, et constitue la technologie de journal d'événements choisie pour les applications de streaming basées sur les microservices Amadeus. Kafka est également utilisé pour des cas d'utilisation opérationnels tels que la</p>	 <p>Utilisé dans notre pipeline d'événements, le suivi des exceptions et plus encore à venir.</p>
 <p>Kafka est utilisé comme pipeline de traitement des journaux d'événements pour fournir des produits et des services mieux personnalisés à nos clients.</p>		 <p>Altair Panopticon(TM) intègre Kafka dans son moteur de traitement de flux. Le logiciel permet aux utilisateurs professionnels de créer des applications de traitement de flux qui s'abonnent aux entrées de données en streaming, y compris les sujets Kafka et d'autres files d'attente de messages en temps réel, d'extraire des bases de données SQL, NoSQL et de</p>	

Concepts fondamentaux

Concepts fondamentaux

1. Producers
2. Consumers
3. Messages
4. Core APIs

Concepts fondamentaux

Producers

- Les producteurs (producers) sont des applications ou des systèmes qui envoient (publient) des messages à Kafka.
- Ils envoient des données à Kafka en les écrivant dans des topics spécifiques.
- Chaque message peut être vu comme une unité de données et est stocké dans Kafka jusqu'à ce qu'il soit consommé.

Concepts fondamentaux

Consumers

- Les consommateurs (consumers) sont des applications ou des systèmes qui lisent (consomment) les messages à partir de Kafka.
- Ils s'abonnent à un ou plusieurs topics et traitent les données reçues.
- Les consommateurs peuvent travailler en groupes pour se répartir les messages d'un topic, permettant ainsi un traitement parallèle des données.

Concepts fondamentaux

Messages

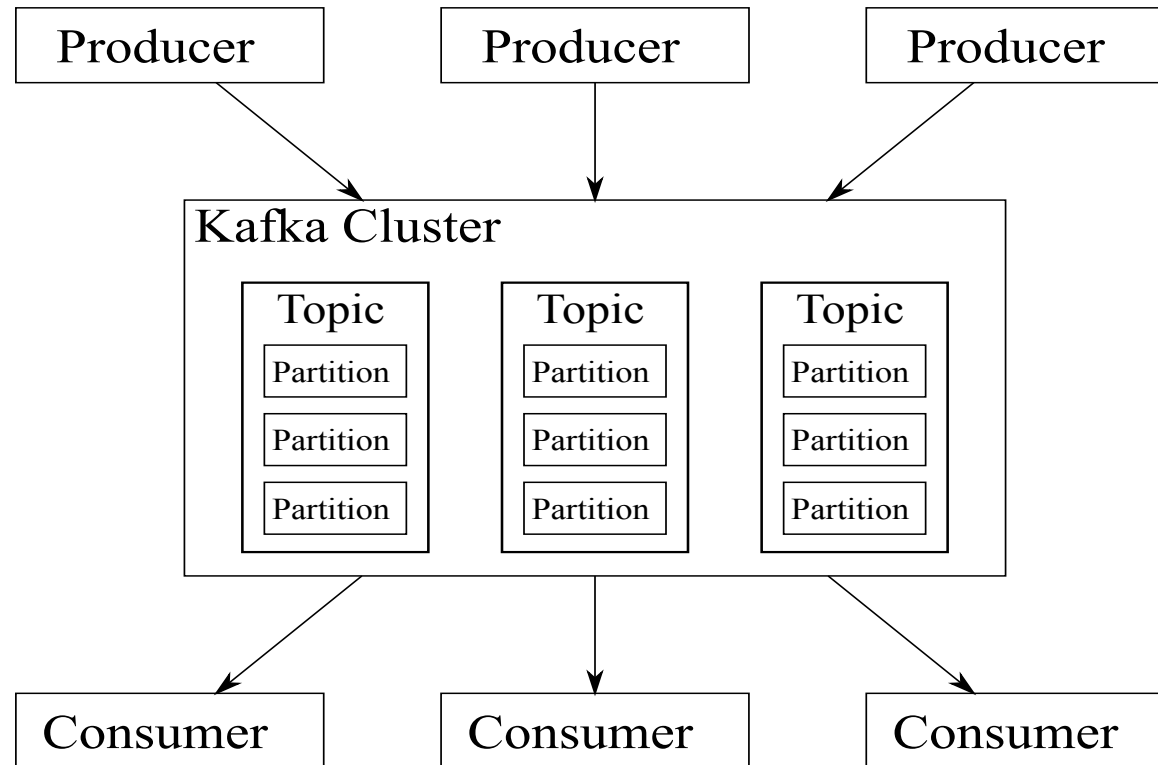
- Les messages sont les unités de données échangées via Kafka.
- Un message est constitué d'une clé (facultative) et d'une valeur.
- Kafka stocke ces messages dans des topics, qui sont des journaux partitionnés et distribués.

Concepts fondamentaux

5 Core APIs

- **Producteur API:** Va permettre à une application de publier des données dans un topic Kafka.
- **Consumer API:** Permet à une application de lire les données d'un ou plusieurs topics kafka.
- **Streams API:** Permet à une application d'interagir comme un stream, de lire des données d'un ou plusieurs, les transformer et les écrire dans un ou plusieurs topics de sorties.
- **Connector API:** Permet de connecter Kafka avec le monde extérieur, par exemple une base de données, HDFS, S3, teradata ...

Concepts fondamentaux



Architecture d'une plateforme Kafka

Architecture d'une plateforme Kafka

- Brokers
- Zookeeper
- Kafka Connect
- Kafka Streams
- Schema Registry
- KSQLDB
- Rest Proxy

Architecture d'une plateforme Kafka

Broker

- Un **broker** est un serveur au sein du cluster Kafka qui a pour rôle de gérer le stockage et le transit des messages.
- Chaque **broker** est une instance individuelle dans un réseau de brokers et chaque broker peut gérer des terabytes de messages sans dégrader les performances.

Architecture d'une plateforme Kafka

Fonctionnement et Responsabilités d'un Broker

1. *Stockage des Messages:*

- Les brokers stockent les messages dans des topics.
- Un topic est une catégorie ou un canal de messages qui permet à Kafka de gérer efficacement le flux de données.
- Les topics sont divisés en partitions, chacune stockée sur un ou plusieurs brokers du cluster.
- Chaque partition a un broker leader et peut avoir zéro ou plusieurs brokers répliques qui assurent la redondance et la haute disponibilité des données

Architecture d'une plateforme Kafka

Fonctionnement et Responsabilités d'un Broker

2. Répartition des Charges:

- Les brokers répartissent la charge de manière équilibrée dans le cluster.
- Kafka utilise ZooKeeper pour coordonner les informations entre les brokers et pour élire les leaders des partitions.
- Le broker leader gère toutes les écritures et lectures pour sa partition, tandis que les brokers répliques synchronisent les données en continu pour maintenir les répliques à jour.

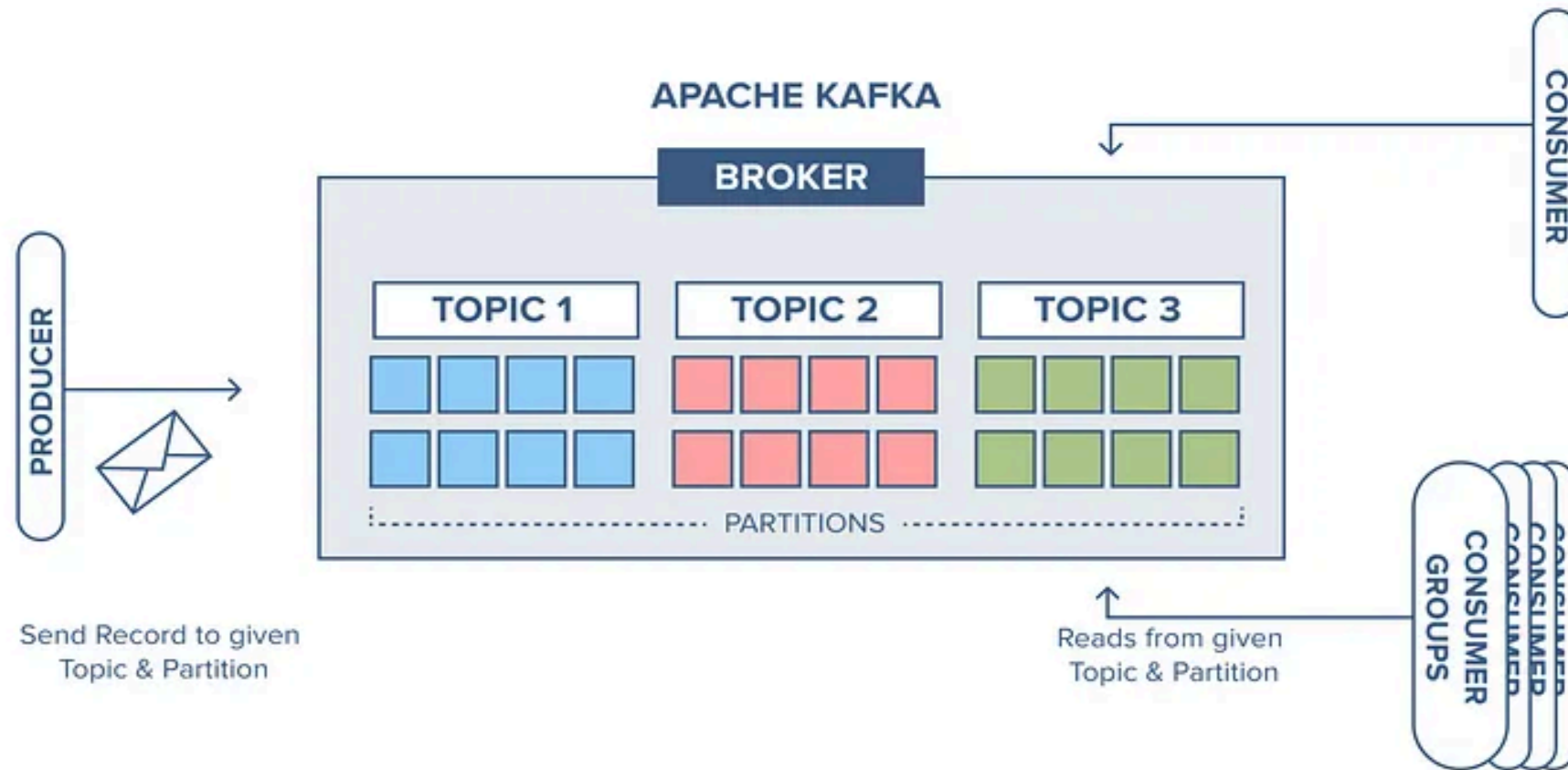
Architecture d'une plateforme Kafka

Fonctionnement et Responsabilités d'un Broker

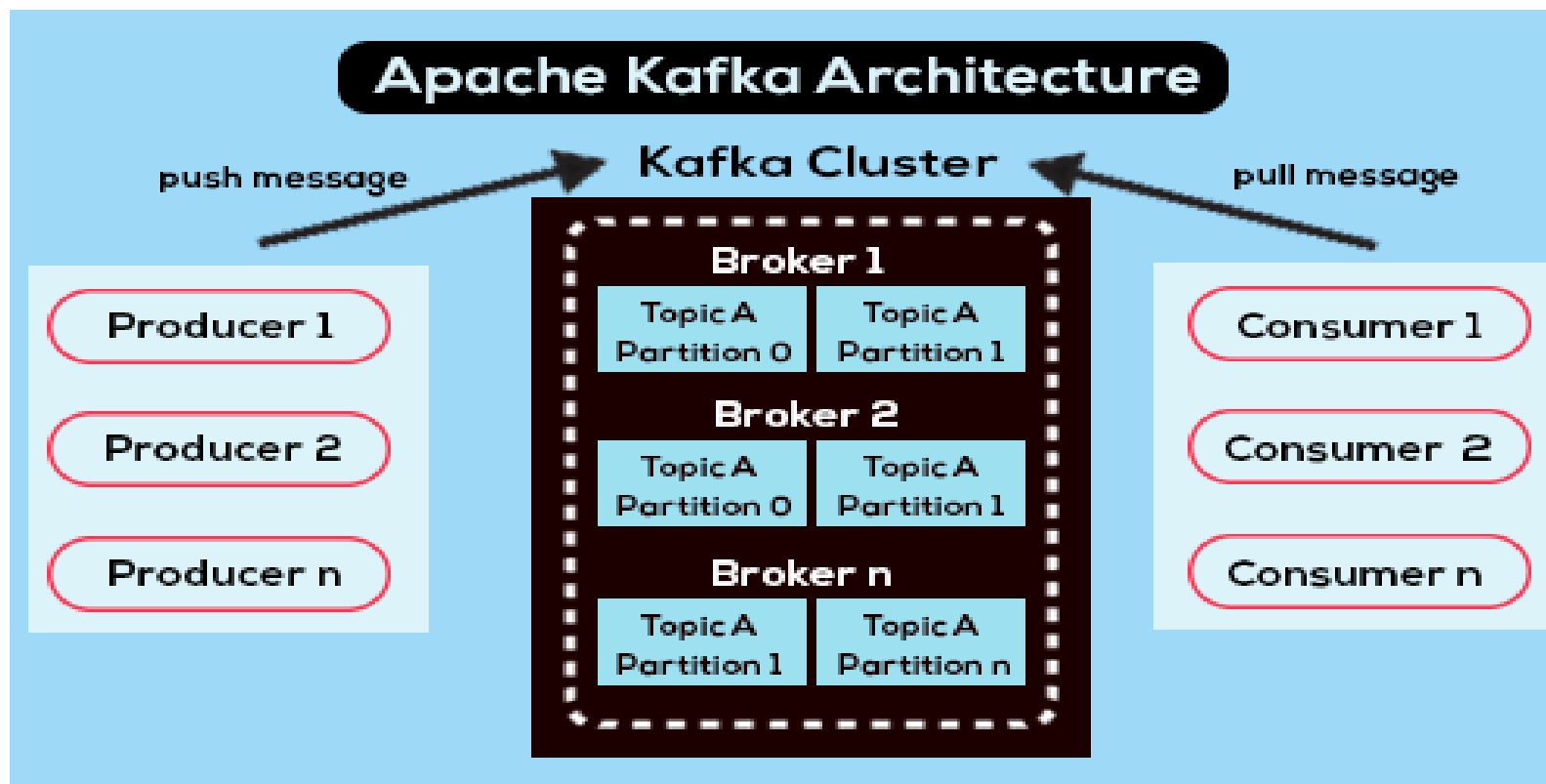
3. *Traitement des Requêtes:*

- Les brokers traitent et répondent aux requêtes des clients (producteurs et consommateurs).
- Les producteurs envoient des messages au broker leader de la partition appropriée, et les consommateurs lisent les messages à partir de ces partitions.
- Les brokers gèrent aussi les offsets de consommation, qui sont des pointeurs indiquant jusqu'où un consommateur a lu dans un topic.

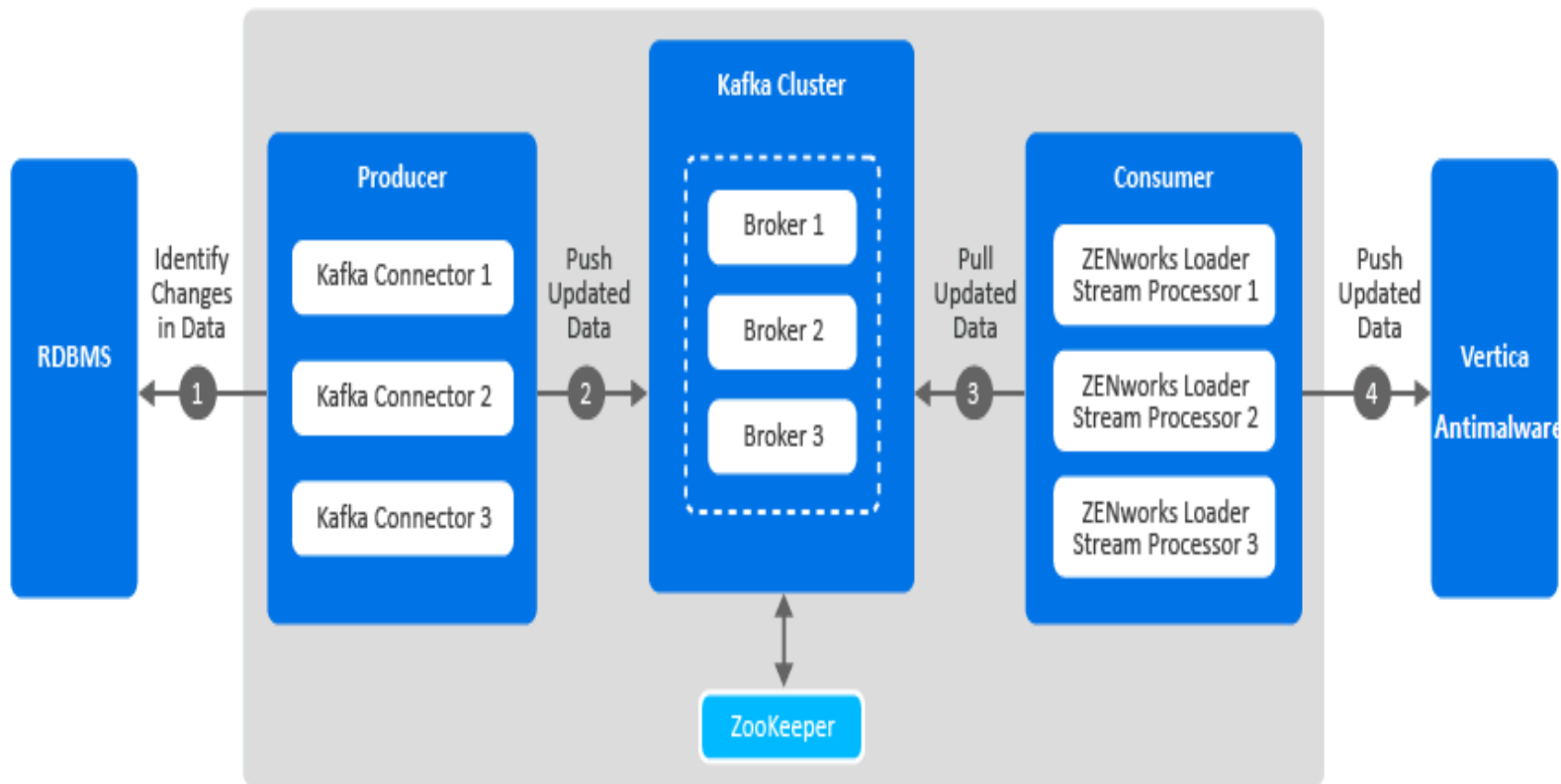
Architecture d'une plateforme Kafka



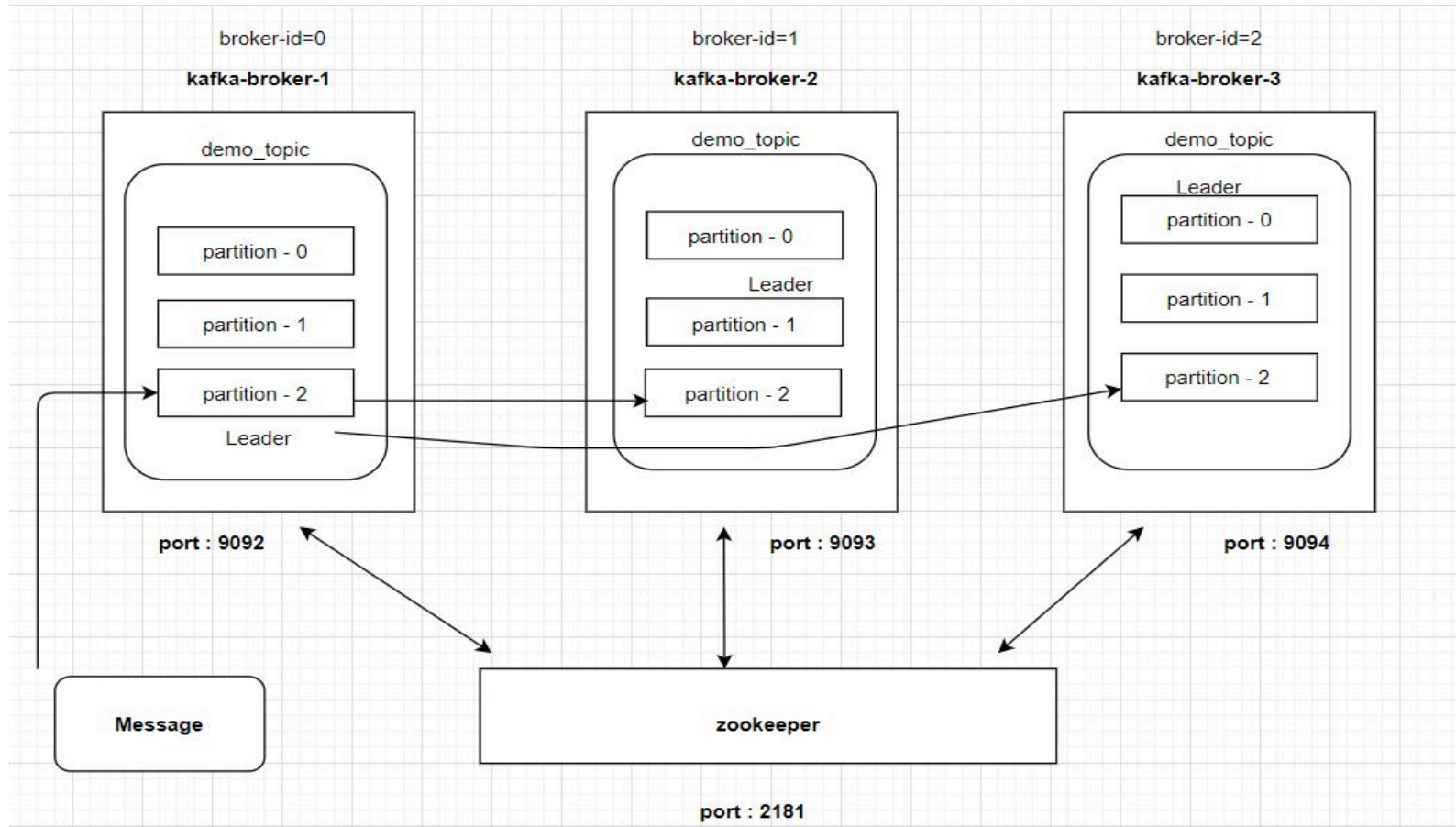
Architecture d'une plateforme Kafka



Architecture d'une plateforme Kafka



Architecture d'une plateforme Kafka



Architecture d'une plateforme Kafka

Les composantes d'un Broker

1. Topics et Partitions

- **Topics** : Un topic est une catégorie ou un flux de messages auxquels les producteurs envoient des messages et à partir desquels les consommateurs lisent. Chaque message publié dans un topic est conservé par le système Kafka pour une durée configurable.
- **Partitions** : Chaque topic est divisé en partitions. La partition est l'unité de base du parallélisme dans Kafka, et chaque partition est ordonnée, immuable et unique. Les partitions permettent à plusieurs consommateurs de lire le topic en parallèle, augmentant ainsi les performances.

Architecture d'une plateforme Kafka

Les composants d'un Broker

2. Logs

- Chaque partition dans **Kafka** est représentée par un ensemble de logs, où un log est un fichier sur le disque.
- Les **logs** contiennent les enregistrements des messages dans l'ordre chronologique.
- Kafka écrit tous les messages entrants dans ces **logs** pour assurer la persistance des données.

Architecture d'une plateforme Kafka

Les composantes d'un Broker

3. Segments de Log

- Un log est composé de plusieurs segments de fichier.
- Lorsqu'un segment atteint une certaine taille ou une certaine durée, il est fermé et un nouveau segment est créé.
- Les anciens segments peuvent être supprimés ou archivés en fonction de la politique de rétention du topic

Architecture d'une plateforme Kafka

Les composantes d'un Broker

4. Index

- Pour chaque segment de log, Kafka maintient un index qui permet de localiser rapidement un message dans le segment sans avoir à scanner tout le fichier.
- L'index stocke l'offset de message et la position correspondante dans le fichier de log, permettant ainsi des accès directs rapides aux données.

Architecture d'une plateforme Kafka

Les composantes d'un Broker

5. Réplicas

- Les **partitions** de chaque topic peuvent être répliquées sur plusieurs **brokers** pour garantir la redondance des données et une haute disponibilité.
- Chaque **réplique** d'une partition est un clone complet de toutes les données de cette partition.
- Un des **brokers** sera le leader pour une partition donnée et gèrera toutes les écritures et lectures, tandis que les autres brokers agiront comme des followers et se synchroniseront avec le **leader**.

Architecture d'une plateforme Kafka

Les composantes d'un Broker

6. Offset Manager

- Chaque broker maintient un suivi des offsets de consommation pour chaque consommateur, ce qui indique jusqu'où un consommateur a lu dans un topic.
- Ces informations sont cruciales pour assurer que les consommateurs peuvent reprendre la lecture à partir du bon endroit après une déconnexion ou un redémarrage.

Architecture d'une plateforme Kafka

Zookeeper

Apache ZooKeeper est un service centralisé pour la gestion de la configuration, la synchronisation des données et la fourniture de services de nommage distribués.

ZooKeeper permet aux applications distribuées de fonctionner avec une coordination et une configuration uniformes, grâce à une interface simple et robuste.

Architecture d'une plateforme Kafka

Rôle de ZooKeeper dans Kafka

1. Gestion de la Configuration:

ZooKeeper stocke la configuration de l'ensemble du cluster Kafka. Cela inclut les informations sur les topics, les partitions, les répliques, les brokers, les quotas de clients, et plus encore.

2. Synchronisation et Coordination:

ZooKeeper sert de mécanisme de coordination entre les brokers Kafka. Il aide à maintenir le cluster synchronisé en termes d'état et de configuration, assurant que tous les brokers ont une vision cohérente de l'état du cluster.

Architecture d'une plateforme Kafka

Rôle de ZooKeeper dans Kafka

3. Élection des Leaders:

- Pour chaque partition de topic, un **broker** doit agir en tant que leader, tandis que les autres servent de répliques.
- **ZooKeeper** aide à gérer l'élection des leaders pour les partitions.
- En cas de défaillance d'un broker leader, **ZooKeeper** coordonne l'élection d'un nouveau leader parmi les brokers répliques disponibles.

Architecture d'une plateforme Kafka

Rôle de ZooKeeper dans Kafka

4. Suivi des Brokers:

- **ZooKeeper** surveille l'état des brokers dans le cluster Kafka.
- Chaque **broker** s'enregistre auprès de **ZooKeeper** à son démarrage.
- Si un **broker** cesse de répondre, **ZooKeeper** en informe les autres brokers, qui peuvent alors réagir en conséquence, par exemple en rééquilibrant les partitions ou en élisant de nouveaux leaders.

Architecture d'une plateforme Kafka

Rôle de ZooKeeper dans Kafka

5. Enregistrement des Clients et Gestion des Offsets:

Les **consommateurs** utilisent ZooKeeper pour gérer les offsets de consommation des messages, bien que les versions plus récentes de Kafka gèrent cela directement via le broker (déplacement de cette fonctionnalité hors de ZooKeeper pour améliorer les performances et la scalabilité).

Architecture d'une plateforme Kafka

Les limites de Zookeeper

Scalabilité:

- ZooKeeper peut devenir un goulot d'étranglement lorsqu'il s'agit de gérer de très grands clusters Kafka.
- Comme ZooKeeper doit gérer toutes les transactions de métadonnées de manière séquentielle, sa capacité à échelonner horizontalement est limitée, ce qui peut ralentir les performances avec l'augmentation du nombre de brokers et de clients.

Complexité de Maintenance:

- La gestion de ZooKeeper nécessite une surveillance et une maintenance supplémentaires.
- Les administrateurs de Kafka doivent également être compétents dans la gestion de ZooKeeper, ce qui ajoute une couche de complexité opérationnelle.

Architecture d'une plateforme Kafka

Les limites de Zookeeper

Résilience:

- Bien que ZooKeeper soit conçu pour être résilient, la défaillance de ZooKeeper peut avoir un impact significatif sur la disponibilité du cluster Kafka.
- Un cluster ZooKeeper doit être configuré correctement avec un nombre impair de serveurs pour éviter les scénarios de split-brain.

Performance:

- Les performances de ZooKeeper peuvent être un facteur limitant dans les environnements où la lecture et l'écriture des métadonnées sont fréquentes.
- Cela est particulièrement vrai dans les cas où les brokers et les consommateurs sont très dynamiques, ce qui augmente le volume des opérations que ZooKeeper doit gérer.

Architecture d'une plateforme Kafka

Une alternative : KRaft

- **Apache Kafka Raft (KRaft)** est le protocole de consensus qui a été introduit dans KIP-500 pour supprimer la dépendance d'Apache Kafka à l'égard de zooKeeper pour la gestion des métadonnées.
- Cela simplifie considérablement l'architecture de Kafka en consolidant la responsabilité des métadonnées en Kafka elle-même, plutôt que de la diviser entre deux systèmes différents: le zooKeeper et la kafka.
- Le mode KRaft utilise un nouveau service de contrôleur de quorum dans Kafka qui remplace le contrôleur précédent et utilise une variante basée sur des événements du protocole de consensus Raft.

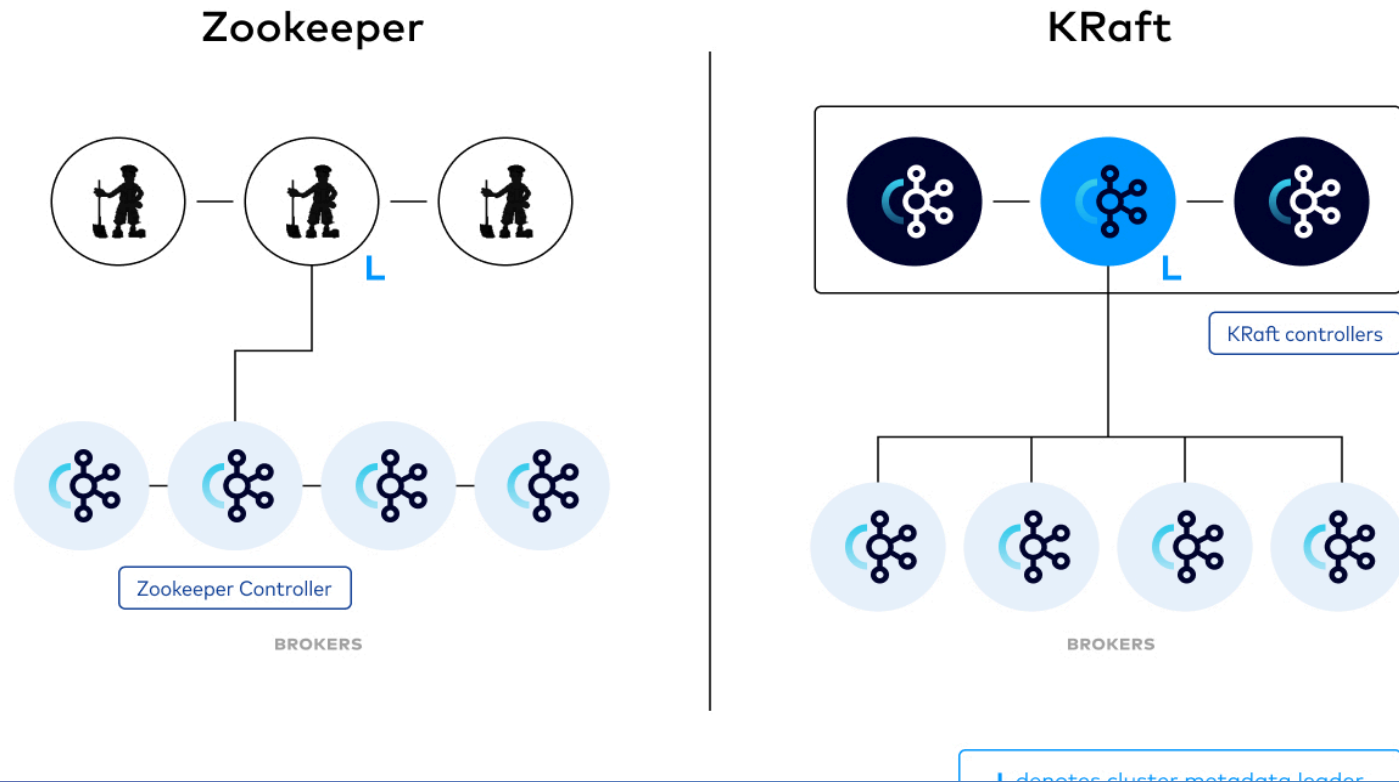
Architecture d'une plateforme Kafka

Avantages de KRaft

- **Simplification de l'architecture** : Réduit la complexité en n'ayant pas à gérer un système externe pour les métadonnées.
- **Amélioration des performances** : Diminue la latence liée à la gestion des métadonnées en intégrant le processus directement dans Kafka.
- **Scalabilité améliorée** : Permet à Kafka de mieux gérer les clusters de grande taille sans être limité par les performances de ZooKeeper.
- **Autonomie accrue** : Rend Kafka plus autonome et simplifie la configuration et la maintenance pour les utilisateurs.

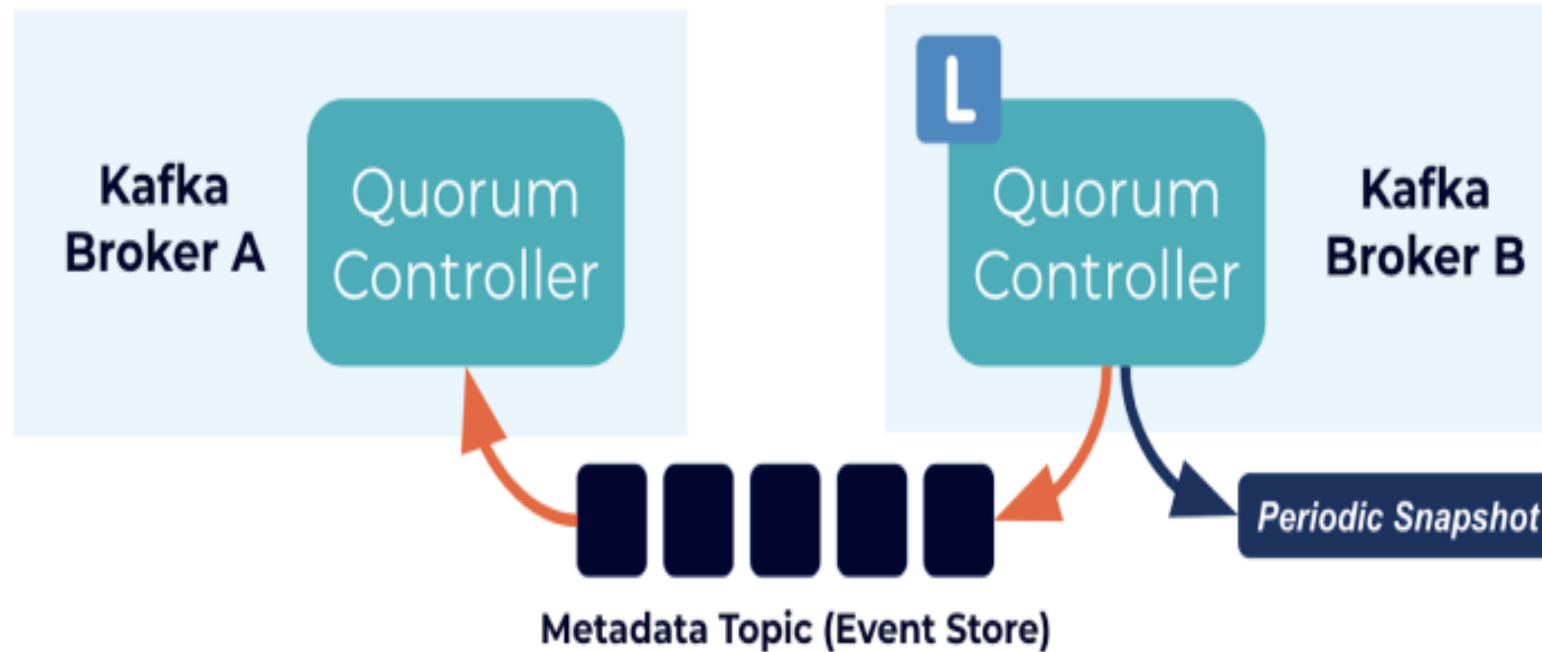
Architecture d'une plateforme Kafka

Avantages de KRaft



Architecture d'une plateforme Kafka

Avantages de KRaft



Architecture d'une plateforme Kafka

Kafka Connect

- **Kafka Connect** est un framework intégré à Apache Kafka qui permet de simplifier et d'automatiser le transfert de données entre Kafka et d'autres systèmes, comme des bases de données, des systèmes de fichiers, des services de stockage en nuage, etc.
- Il est conçu pour être évolutif, fiable et facile à configurer et à déployer.
- **Kafka Connect** est particulièrement utile pour les intégrations de données en temps réel et les charges de travail de streaming de données.

Cette partie sera abordé dans une section à part entière, un peu plus loin.

Architecture d'une plateforme Kafka

Schema Registry

- **Schema Registry** est un système de gestion de schémas centralisé pour les données structurées qui transitent dans Apache Kafka.
- Il permet de garantir que les messages envoyés à travers **Kafka** suivent un schéma prédéfini, ce qui facilite la validation, l'enrichissement, et la compatibilité des données au sein des systèmes qui produisent et consomment des messages.
- **Schema Registry** est particulièrement utile dans les environnements de données complexes où plusieurs applications et services échangent des données, car il aide à maintenir la discipline des formats de données et réduit le risque d'erreurs dues à des incompatibilités de schéma.

Architecture d'une plateforme Kafka

Les fonctionnalités Schema Registry

1. Gestion Centralisée des Schémas :

- **Schema Registry** stocke et gère tous les schémas dans un emplacement central, permettant un accès facile et une gestion cohérente des schémas à travers les applications.
- Cela inclut l'ajout, la modification et la suppression de schémas ainsi que le versioning des schémas pour gérer les évolutions.

2. Validation de Schéma :

- Lorsque les producteurs envoient des messages à Kafka, **Schema Registry** peut être configuré pour valider ces messages par rapport aux schémas enregistrés.
- Cela garantit que les données produites sont conformes aux attentes des consommateurs, améliorant ainsi la qualité des données.

Architecture d'une plateforme Kafka

Les fonctionnalités Schema Registry

3. Support du Versioning :

- **Schema Registry** gère les versions des schémas, ce qui permet aux applications de continuer à fonctionner même si le schéma des données évolue.
- Cela aide à gérer la compatibilité vers l'arrière (**backward compatibility**) et vers l'avant (**forward compatibility**) pour les applications consommatrices.

4. Compatibilité de Schéma :

- Il offre des politiques de compatibilité pour contrôler comment les schémas peuvent évoluer, par exemple en empêchant des changements qui casseraient la compatibilité vers l'arrière.
- Cela permet aux développeurs de s'assurer que les applications qui dépendent d'un schéma particulier continuent de fonctionner correctement après des modifications du schéma.

Architecture d'une plateforme Kafka

Les fonctionnalités Schema Registry

5. Intégration avec Kafka :

- Schema Registry est souvent utilisé avec des bibliothèques client comme **Apache Avro**, qui permettent de sérialiser les données de manière compacte et efficace en utilisant les schémas définis.
- Cela réduit la taille des messages Kafka et améliore les performances de transmission.

Architecture d'une plateforme Kafka

Schema Registry : Avro

- **Apache Avro** est un système de sérialisation de données développé dans le cadre de la fondation Apache.
- Il est conçu pour la sérialisation de structures de données complexes exprimées en JSON.
- Avro est largement utilisé dans des écosystèmes tels que Apache Hadoop, où il est souvent utilisé pour la sérialisation de données dans des systèmes de stockage distribué et pour des tâches de traitement de données.

Architecture d'une plateforme Kafka

```
{
  "type": "record",
  "name": "Order",
  "namespace": "com.example.ecommerce",
  "fields": [
    {
      "name": "orderId",
      "type": "int"
    },
    {
      "name": "orderDate",
      "type": "string"
    },
    {
      "name": "items",
      "type": {
        "type": "array",
        "items": {
          "type": "record",
          "name": "Item",
          "fields": [
            {
              "name": "itemId",
              "type": "int"
            },
            {
              "name": "quantity",
              "type": "int"
            },
            {
              "name": "price",
              "type": "float"
            }
          ]
        }
      }
    }
  ]
}
```

```
// Importation des classes nécessaires pour travailler avec Avro.
import org.apache.avro.Schema;
import org.apache.avro.generic.GenericData;
import org.apache.avro.generic.GenericRecord;
import org.apache.avro.io.DatumWriter;
import org.apache.avro.io.DatumReader;
import org.apache.avro.io.Encoder;
import org.apache.avro.io.EncoderFactory;
import java.io.ByteArrayOutputStream;

// Déclaration de la classe principale.
public class AvroOrderExample {
    // Point d'entrée principal du programme.
    public static void main(String[] args) throws Exception {
        // Chargement du schéma Avro à partir d'un fichier .avsc.
        Schema schema = new Schema.Parser().parse(new java.io.File("Order.avsc"));

        // Création d'un enregistrement générique pour une commande en utilisant le schéma chargé.
        GenericRecord order = new GenericData.Record(schema);
        // Ajout de l'ID de la commande et de la date de la commande à l'enregistrement.
        order.put("orderId", 123456);
        order.put("orderDate", "2021-09-23");

        // Création du premier article de la commande.
        GenericRecord item1 = new GenericData.Record(schema.getField("items").schema().getElementType());
        item1.put("itemId", 987);
        item1.put("quantity", 2);
        item1.put("price", 19.99f);

        // Création du deuxième article de la commande.
        GenericRecord item2 = new GenericData.Record(schema.getField("items").schema().getElementType());
        item2.put("itemId", 654);
        item2.put("quantity", 1);
        item2.put("price", 149.99f);

        // Création d'une liste pour stocker les articles de la commande.
        List<GenericRecord> items = new ArrayList<>();
        items.add(item1);
        items.add(item2);

        // Ajout de la liste des articles à l'enregistrement de la commande.
        order.put("items", items);

        // Préparation d'un flux pour écrire les données sérialisées.
        ByteArrayOutputStream out = new ByteArrayOutputStream();
        // Création d'un écrivain de données pour sérialiser l'enregistrement de la commande.
        DatumWriter<GenericRecord> writer = new GenericDatumWriter<>(schema);
        // Création d'un encodeur pour écrire les données sérialisées.
        Encoder encoder = EncoderFactory.get().binaryEncoder(out, null);
        // Sérialisation de l'enregistrement de la commande dans le flux de sortie.
        writer.write(order, encoder);
        // Vidage de l'encodeur pour s'assurer que toutes les données sont écrites.
        encoder.flush();
        // Fermeture du flux de sortie.
        out.close();

        // Récupération des données sérialisées sous forme d'un tableau de bytes et affichage.
        byte[] serializedBytes = out.toByteArray();
        System.out.println("Serialized order data as byte array: " + Arrays.toString(serializedBytes));
    }
}
```

Architecture d'une plateforme Kafka

KSQLDB et KSQL

- 1. **ksqlDB** est une base de données de streaming distribuée qui est construite sur Apache Kafka.
 - Elle est conçue pour permettre des requêtes continues sur des flux de données, ce qui la rend particulièrement utile pour des scénarios où il est nécessaire d'analyser et de réagir aux données en temps réel.
- 2. **KSQL** est un outil de requête de streaming pour Apache Kafka développé par **Confluent**.
 - Il est conçu pour permettre aux utilisateurs d'effectuer des requêtes SQL en temps réel sur les données Kafka.
 - KSQL simplifie le processus d'écriture de pipelines de traitement de données en streaming, en utilisant une syntaxe SQL familière, ce qui le rend accessible même pour ceux qui ne sont pas experts en programmation de streaming complexe.

Architecture d'une plateforme Kafka

KSQLEDB et KSQL

- 1. **ksqleDB** est une base de données de streaming distribuée qui est construite sur Apache Kafka.
 - Elle est conçue pour permettre des requêtes continues sur des flux de données, ce qui la rend particulièrement utile pour des scénarios où il est nécessaire d'analyser et de réagir aux données en temps réel.
- 2. **KSQL** est un outil de requête de streaming pour Apache Kafka développé par **Confluent**.
 - Il est conçu pour permettre aux utilisateurs d'effectuer des requêtes SQL en temps réel sur les données Kafka.
 - KSQL simplifie le processus d'écriture de pipelines de traitement de données en streaming, en utilisant une syntaxe SQL familière, ce qui le rend accessible même pour ceux qui ne sont pas experts en programmation de streaming complexe.

Cette partie sera abordée dans une section à part entière, un peu plus loin.

Architecture d'une plateforme Kafka

Kafka REST Proxy

- Le **Kafka REST Proxy** est un composant de l'écosystème Confluent Kafka qui permet d'interagir avec un cluster Kafka via une interface RESTful.
- Cela rend Apache Kafka accessible à partir de n'importe quel langage de programmation qui peut faire des requêtes HTTP, sans nécessiter de clients Kafka spécifiques.
- Le **REST Proxy** est particulièrement utile pour intégrer des systèmes qui ne supportent pas nativement les clients Kafka, ou pour simplifier l'architecture en réduisant la nécessité de bibliothèques spécifiques.

Architecture d'une plateforme Kafka

Fonctionnalités principales du REST Proxy

1. Production de messages :

- Le REST Proxy permet de publier des messages dans des topics Kafka via des requêtes HTTP POST.
- Cela inclut le support pour la publication de messages individuels ou par lot, ce qui offre flexibilité et efficacité dans la gestion du trafic réseau.

2. Consommation de messages :

- Il permet également de consommer des messages de Kafka en créant des instances de consommateurs via l'API et en lisant les messages à travers des requêtes HTTP.
- Les consommateurs peuvent être configurés pour lire depuis un point spécifique dans le topic et peuvent être assignés à des groupes de consommateurs pour une consommation équilibrée.

Architecture d'une plateforme Kafka

Fonctionnalités principales du REST Proxy

3. Gestion des métadonnées :

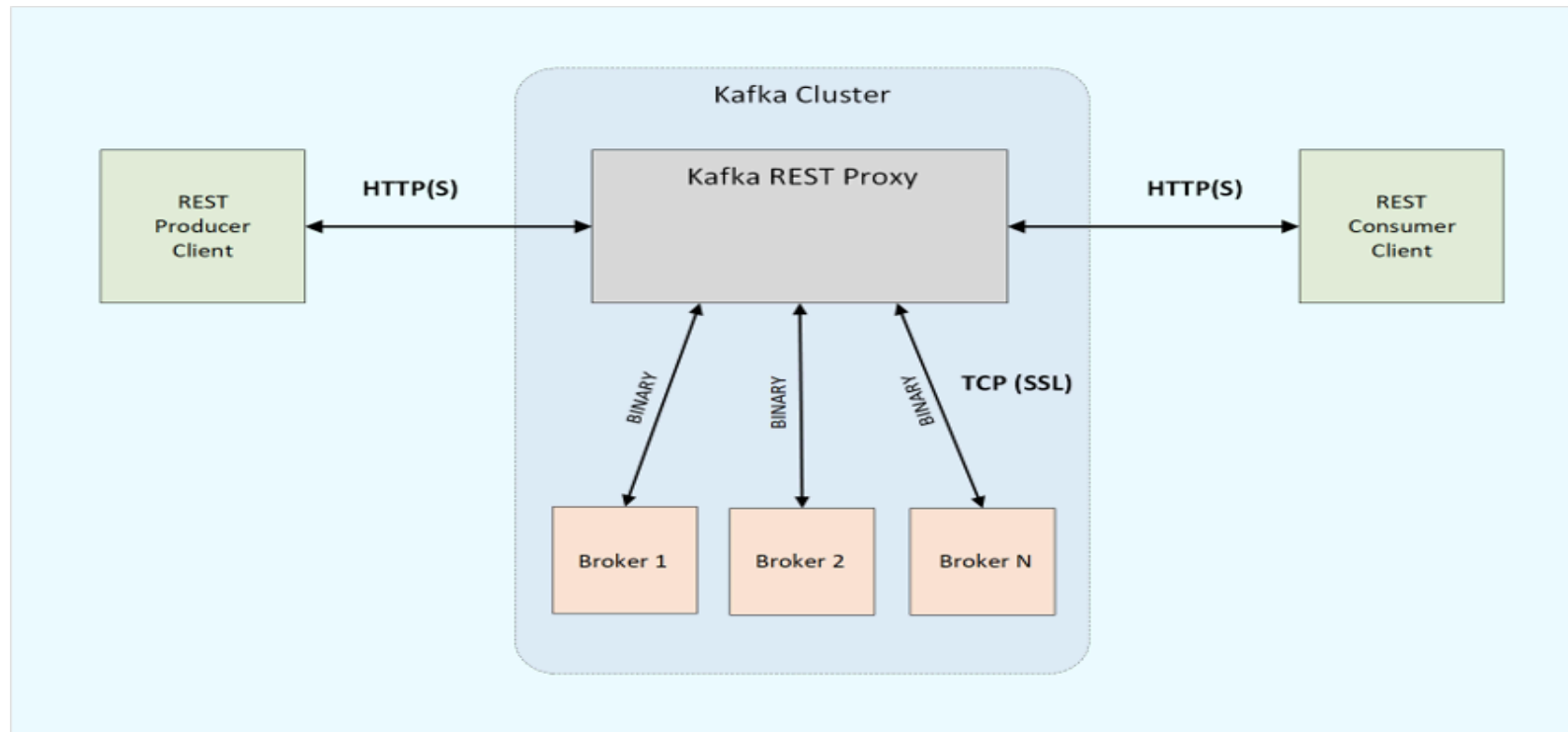
- Le REST Proxy fournit des endpoints pour récupérer des métadonnées sur les clusters Kafka, y compris des informations sur les topics, les partitions, les offsets, les brokers, et d'autres configurations.

4. Sécurité :

- Il prend en charge les configurations de sécurité de Kafka, y compris SSL/TLS pour les connexions cryptées et peut être configuré pour utiliser l'authentification via HTTP Basic ou via des mécanismes d'authentification plus complexes intégrés à des systèmes d'entreprise.

Architecture d'une plateforme Kafka

Fonctionnalités principales du REST Proxy



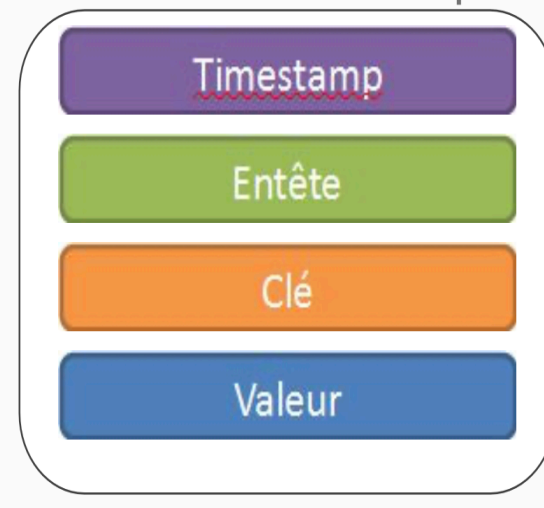
Focus : Topic - Replication - Leader - Follower - Producer - Consumer - Ack - Offset

Focus : Topic - Message - Partitions

Message

- Un **message** ou un **record** dans Kafka est l'unité de données fondamentale qui est produite, stockée et consommée.
- Chaque **message** est enregistré dans un topic Kafka spécifique.
- **Key (Clé)** - La clé est facultative et peut être utilisée pour déterminer la partition au sein d'un topic où le record doit être envoyé. L'utilisation de clés garantit que tous les records avec la même clé seront envoyés à la même partition, ce qui est crucial pour le maintien de l'ordre des messages dans certains scénarios d'usage.
- **Value (Valeur)** - La valeur est le contenu réel du message. Cela peut être une simple chaîne de caractères, un nombre, ou un objet plus complexe sous forme de données sérialisées, telles que JSON, Avro, ou un format binaire personnalisé.

- Message ou record = plus petite unité de données
- Taille maximal de 1Mo par default (message.max.bytes)



RECORD = Tableau d'octets

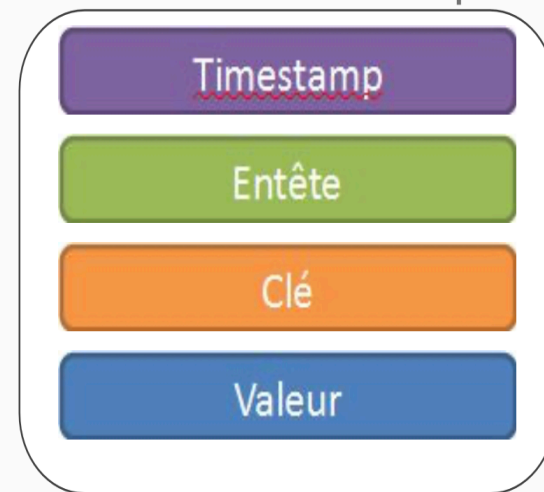
- Formats: string, json, avro ...

Focus : Topic - Message - Partitions

Message

- **Timestamp (Horodatage)** - Chaque record peut inclure un horodatage qui indique le moment de sa création. Kafka peut configurer automatiquement cet horodatage lorsque le record est ajouté au topic, ou il peut être spécifié par le producteur.
- **Headers (En-têtes)** - Kafka permet également d'attacher des en-têtes aux records, qui peuvent être utilisés pour inclure des métadonnées supplémentaires avec chaque message. Les en-têtes sont facultatifs et peuvent être utilisés pour des fonctionnalités avancées telles que le routage ou le filtrage des messages.

- Message ou record = plus petite unité de données
- Taille maximal de 1Mo par default (message.max.bytes)



RECORD = Tableau d'octets

- Formats: string, json, avro ...

Focus : Topic - Message - Partitions

Topic

- Dans Apache Kafka, un **topic** est une catégorie ou un canal de messages par lequel les données sont stockées et organisées.
- Les **topics** sont le composant clé de Kafka où les messages sont écrits, lus et stockés.
- Ils jouent un rôle central dans la gestion de la distribution et de la persistance des messages dans le système Kafka.

Focus : Topic - Message - Partitions

Fonctions principales d'un topic

1. Organisation des Données :

- Les topics permettent de catégoriser les flux de données.
- Par exemple, un topic peut être dédié aux données de transactions, tandis qu'un autre peut contenir des logs d'événements.

2. Distribution de Messages :

- Les messages envoyés à Kafka sont publiés dans des topics spécifiques.
- Les producteurs écrivent des messages dans ces topics et les consommateurs s'abonnent à des topics pour lire ces messages.

3. Scalabilité et Performance :

- Les topics sont divisés en partitions, permettant une écriture et une lecture parallèles, ce qui augmente les performances et la scalabilité du système.
- Chaque partition peut être répliquée sur plusieurs nœuds (brokers) pour assurer la redondance et la fiabilité des données.

Focus : Topic - Message - Partitions

Caractéristiques d'un topic

Partitionnement :

- Chaque topic est partitionné, ce qui signifie qu'il est divisé en plusieurs partitions.
- Les partitions permettent à Kafka de paralléliser le traitement en distribuant les données sur plusieurs brokers.
- Chaque partition est ordonnée, et chaque message au sein d'une partition se voit attribuer un identifiant séquentiel unique appelé offset.

Réplication :

- Pour garantir la disponibilité et la durabilité, les partitions d'un topic peuvent être répliquées sur plusieurs brokers.
- Cela signifie que chaque message publié dans une partition peut être copié sur plusieurs brokers, ce qui protège contre la perte de données en cas de défaillance d'un broker.

Facteur de réplication :

- Le facteur de réplication détermine le nombre de copies de chaque partition.
- Un facteur de réplication plus élevé augmente la tolérance aux pannes mais nécessite plus de capacité de stockage et de bande passante réseau.

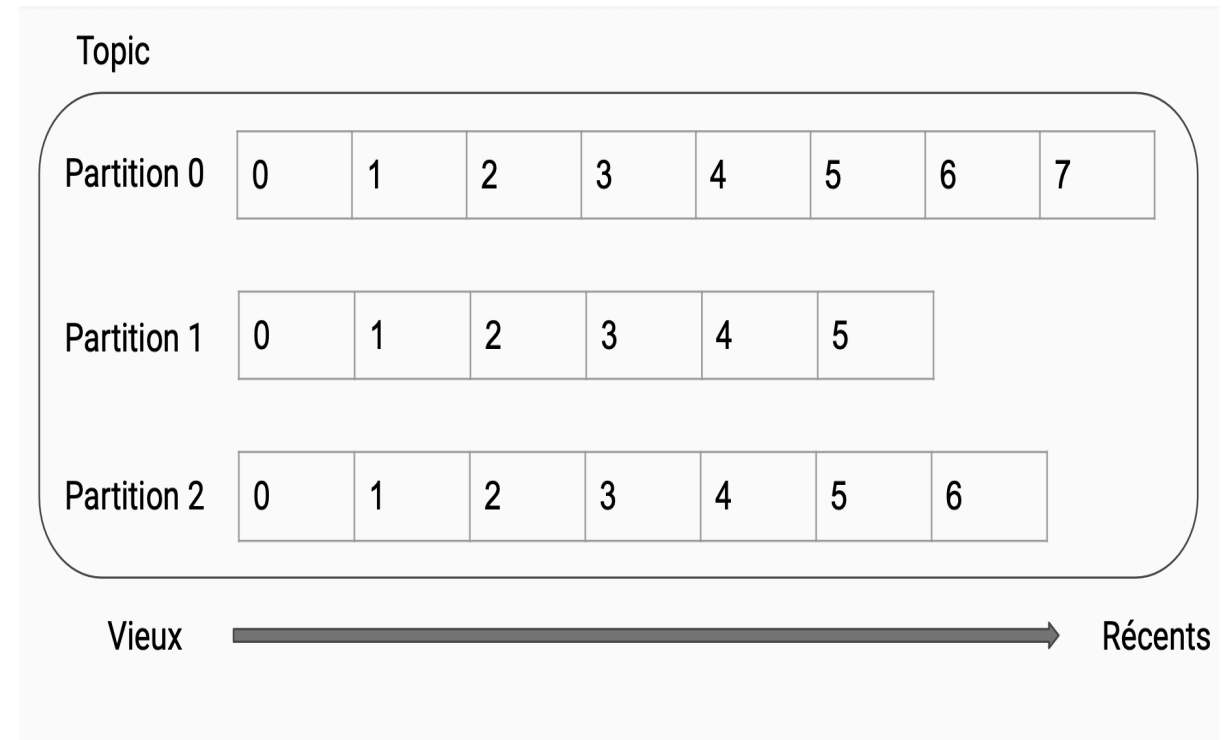
Durée de rétention :

- Les messages dans un topic sont stockés sur le disque et peuvent être configurés pour être conservés pendant une période définie ou jusqu'à ce que le topic atteigne une certaine taille.
- Après cela, les messages anciens sont purgés pour faire de la place à de nouveaux messages.

Focus : Topic - Message - Partitions

Partitions

- Les partitions dans Apache Kafka jouent un rôle crucial dans le fonctionnement et la performance de ce système de messagerie distribuée.
- Elles permettent à Kafka de paralléliser le traitement des données en divisant physiquement les données d'un topic sur plusieurs brokers.



Focus : Topic - Message - Partitions

Fonctionnalité des Partitions

1. Distribution des Données :

- Chaque topic dans Kafka peut être divisé en plusieurs partitions.
- Cette division permet de distribuer les données sur plusieurs serveurs (brokers), ce qui augmente la capacité de stockage et la balance de charge entre les serveurs.

2. Parallélisme :

- Les partitions permettent de traiter les données en parallèle, car chaque partition peut être lue et écrite indépendamment.
- Cela permet à Kafka de gérer de gros volumes de messages en simultané, augmentant ainsi la performance et la scalabilité du système.

3. Redondance et Durabilité :

- Kafka offre la possibilité de répliquer les partitions sur plusieurs brokers.
- Si un broker tombe en panne, une autre copie de cette partition sur un autre broker peut prendre le relais, garantissant ainsi la haute disponibilité et la résilience du système.

Focus : Topic - Message - Partitions

Caractéristiques et Fonctionnement des partitions

- **Offsets :**

- Chaque **message** dans une partition se voit attribuer un identifiant séquentiel unique connu sous le nom **d'offset**.
- **L'offset** permet aux consommateurs de garder une trace de quels messages ont été lus et de recommencer la lecture à partir du dernier message non lu.

- **Leaders et Réplicas :**

- Chaque **partition** a un broker leader et zéro ou plusieurs réplicas.
- Le **leader** gère toutes les demandes de lecture et d'écriture pour la **partition**, tandis que les **réplicas** maintiennent des copies des données pour la redondance.
- En cas de défaillance du broker **leader**, l'un des réplicas peut automatiquement devenir le nouveau **leader**.

- **Équilibrage de charge :**

- Kafka tente de distribuer les **partitions** uniformément entre tous les brokers dans le cluster pour équilibrer la charge.
- Cela inclut non seulement l'équilibrage du nombre de partitions mais aussi l'équilibrage de l'espace disque utilisé et du trafic réseau.

Focus : Facteur de replication

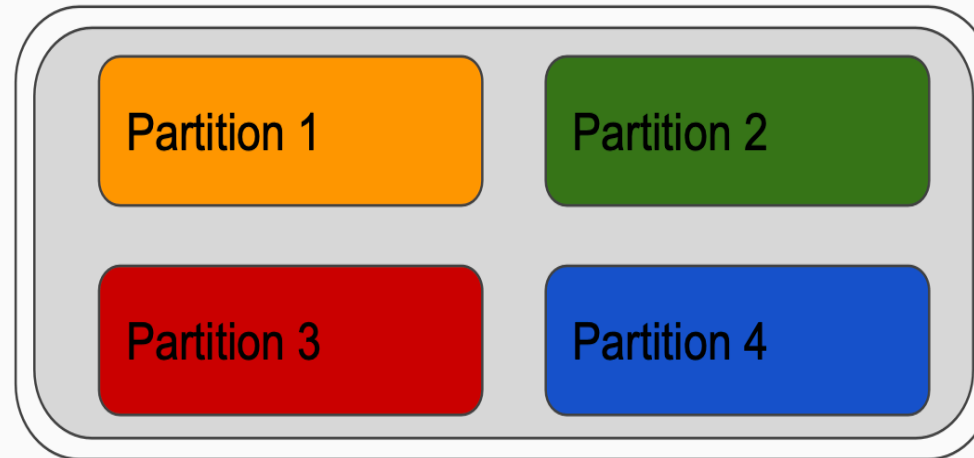
Définition

- Le **facteur de réplication** dans Apache Kafka est une caractéristique cruciale pour assurer la durabilité et la haute disponibilité des données.
- Il se réfère au nombre de copies de chaque **partition de topic** qui sont créées et maintenues sur différents brokers dans le cluster Kafka.
- Ce mécanisme de réplication est essentiel pour protéger les données contre les défaillances des **brokers** et permettre une récupération rapide sans perte de données.

Focus : Facteur de replication

1 Brokers, 1 Topic, 4 partitions,

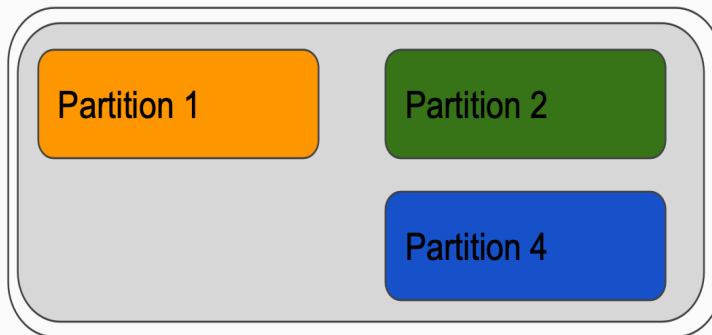
Broker 1



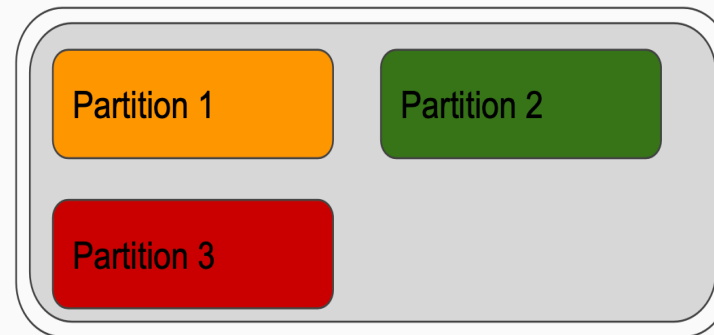
Focus : Facteur de replication

4 Brokers, 1 Topic, 4 partitions, Facteur de réplication: 3

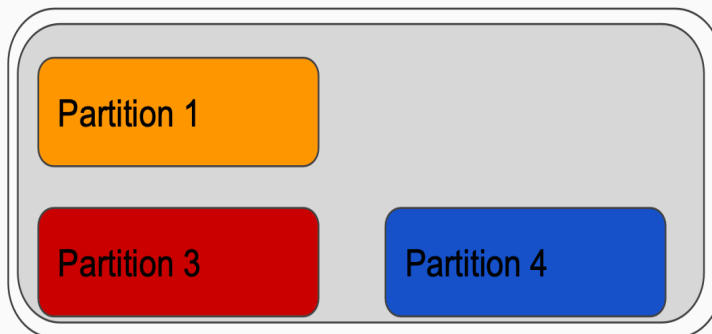
Broker 1



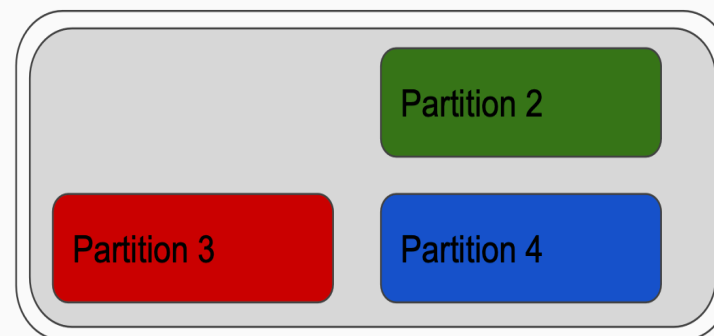
Broker 2



Broker 3



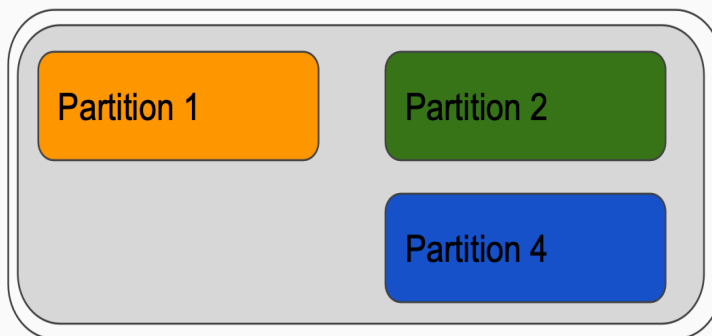
Broker 4



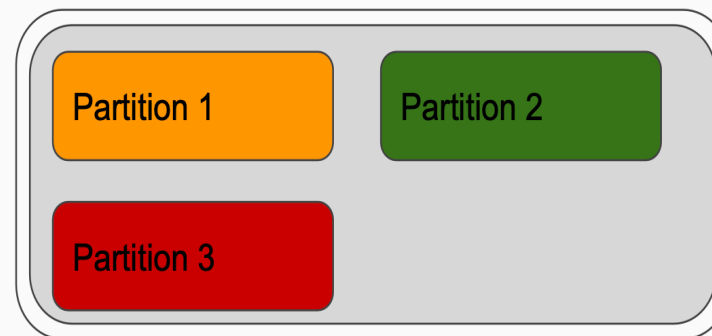
Focus : Facteur de replication

4 Brokers, 1 Topic, 4 partitions, Facteur de réplcation: 3

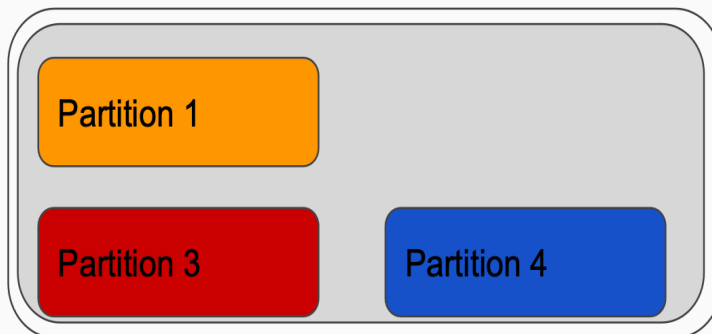
Broker 1



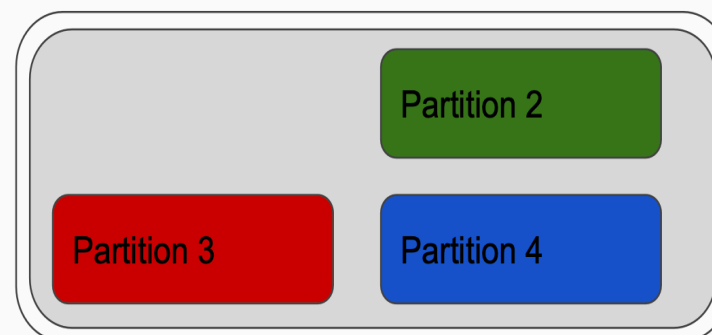
Broker 2



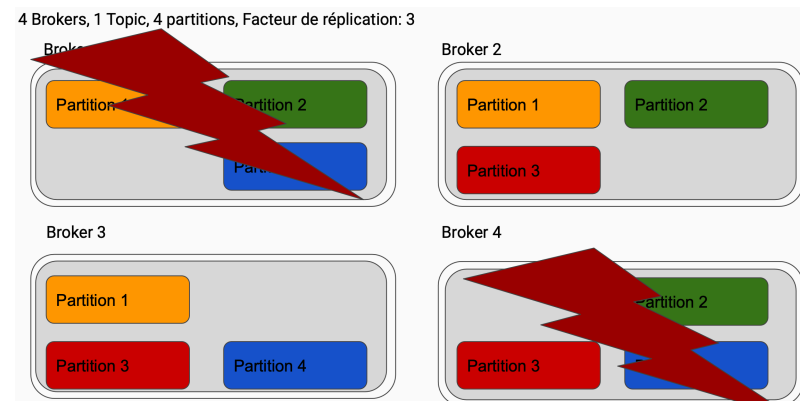
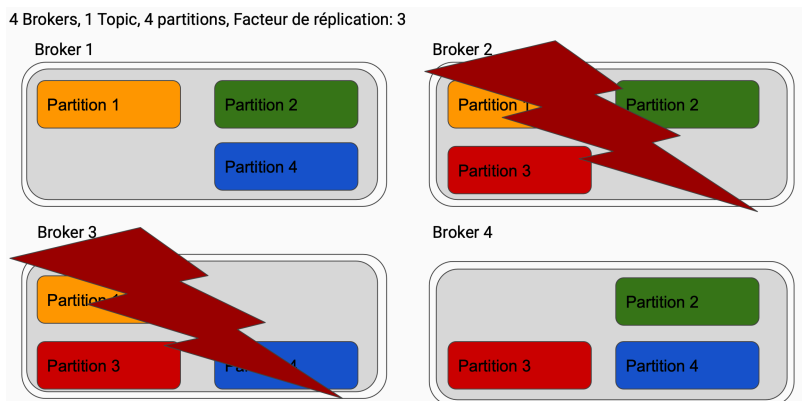
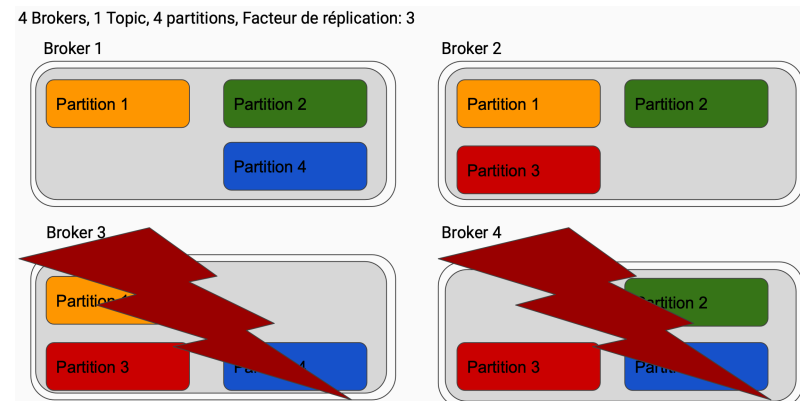
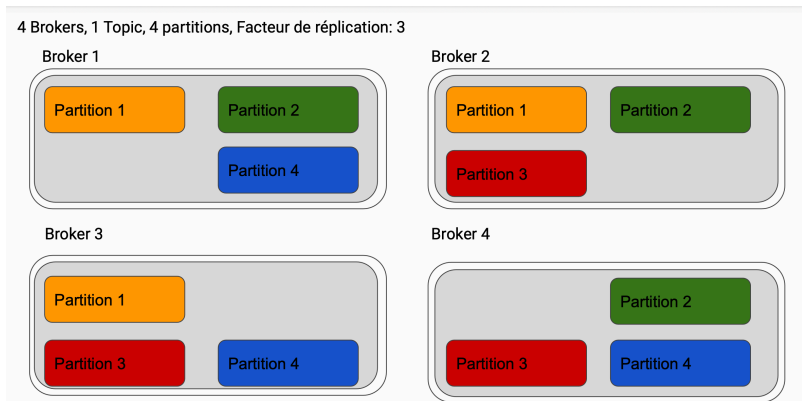
Broker 3



Broker 4



Focus : Facteur de replication



Focus : Leader et Follower

Rôle des Leaders et des Followers

- Chaque partition d'un topic Kafka possède un broker désigné comme leader et un ou plusieurs brokers agissant comme followers.
- Ces rôles sont attribués pour gérer la réplication et assurer l'intégrité des données dans le cluster.

Focus : Leader et Follower

Leader

Traitement des Requêtes :

- Le **leader** est le point de contact principal pour toutes les opérations de lecture et d'écriture pour la partition.
- Toutes les requêtes de production (écriture de messages) et de consommation (lecture de messages) passent par le broker **leader** de la partition.

Responsabilité de la Réplication :

- Le **leader** est également responsable de la réplication des données vers tous les followers de la partition.
- Lorsqu'un message est produit sur une partition, il est d'abord écrit par le **leader**, puis répliqué vers les **followers**.

Focus : Leader et Follower

Follower

Réplication Passive :

- Les **followers** ne gèrent pas directement les requêtes des clients.
- Leur rôle principal est de copier de manière passive les données du **leader**.
- Ils consomment continuellement les messages écrits sur le **leader** et mettent à jour leur propre copie de la partition.

Sécurité et Redondance :

- En maintenant des copies des données, les **followers** contribuent à la redondance et à la résilience du système.
- Si le leader tombe en panne, l'un des **followers** peut être promu en tant que nouveau **leader**, minimisant ainsi les interruptions de service.

Focus : Leader et Follower

Élection du Leader

ZooKeeper et Coordination :

- Kafka utilise **ZooKeeper** pour gérer l'état du cluster, y compris l'élection des **leaders** des partitions.
- Lorsqu'un broker **leader** actuel devient indisponible ou échoue, ZooKeeper coordonne l'élection d'un nouveau leader parmi les followers disponibles et synchronisés.

Critères de Sélection :

- Pour qu'un **follower** soit éligible à devenir **leader**, il doit avoir une copie complète et à jour des données de la partition.
- Cette condition garantit que toutes les données récentes ne sont pas perdues lors du passage à un nouveau **leader**.

Focus : Leader et Follower

Élection du Leader

ZooKeeper et Coordination :

- Kafka utilise **ZooKeeper** pour gérer l'état du cluster, y compris l'élection des **leaders** des partitions.
- Lorsqu'un broker **leader** actuel devient indisponible ou échoue, ZooKeeper coordonne l'élection d'un nouveau leader parmi les followers disponibles et synchronisés.

Critères de Sélection :

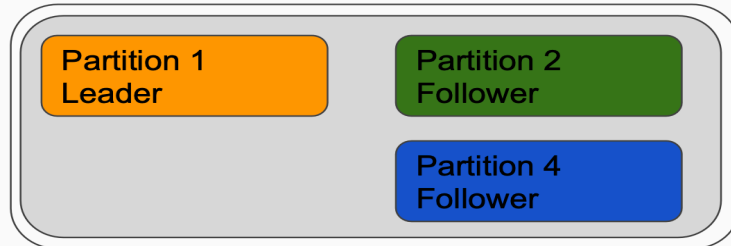
- Pour qu'un **follower** soit éligible à devenir **leader**, il doit avoir une copie complète et à jour des données de la partition.
- Cette condition garantit que toutes les données récentes ne sont pas perdues lors du passage à un nouveau **leader**.

Focus : Leader et Follower

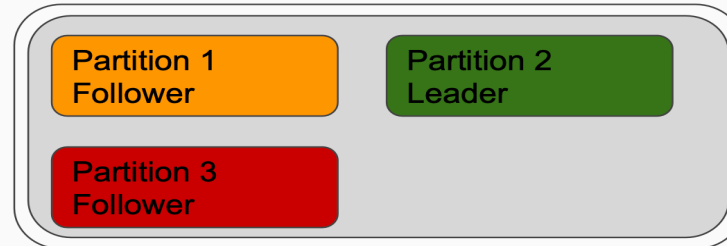
Schema Leader et Follower

4 Brokers, 1 Topic, 4 partitions, Facteur de réplication: 3

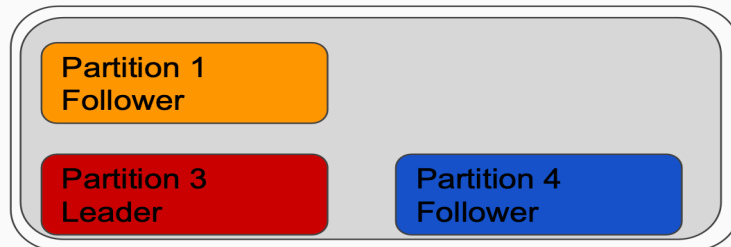
Broker 1



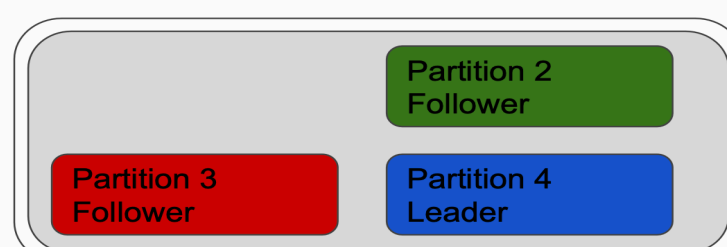
Broker 2



Broker 3

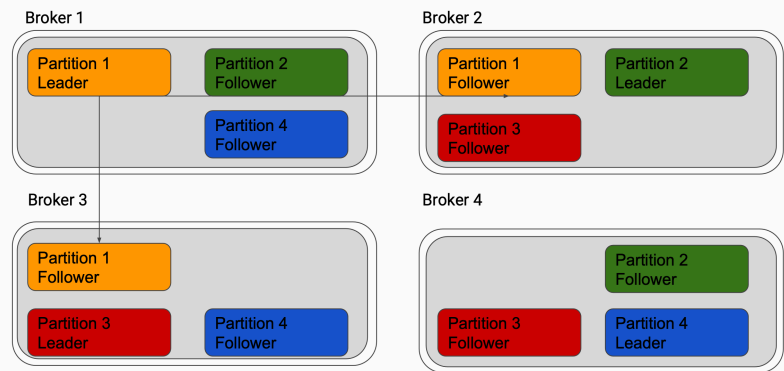


Broker 4

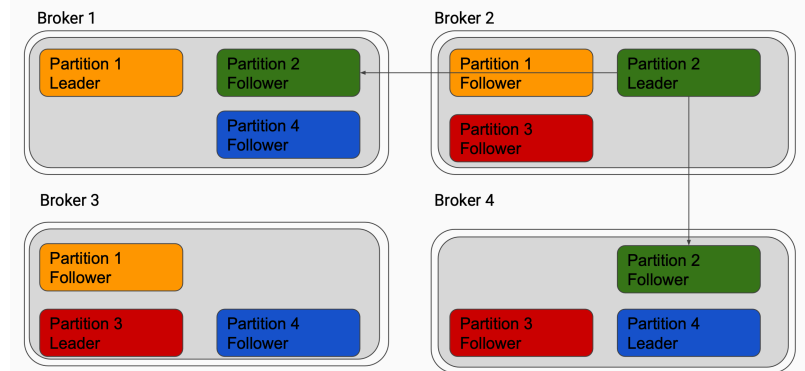


Focus : Leader et Follower

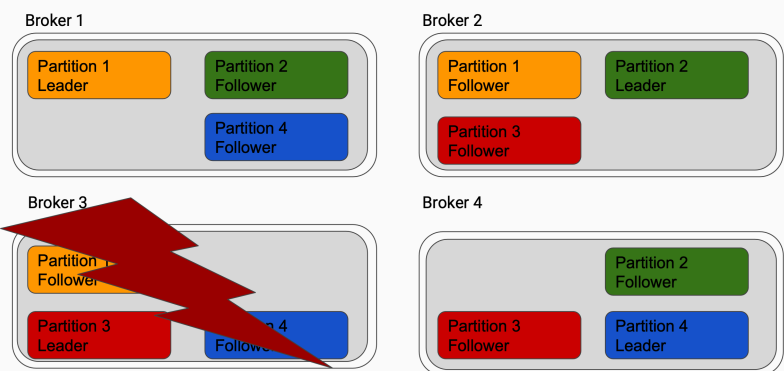
4 Brokers, 1 Topic, 4 partitions, Facteur de réplication: 3



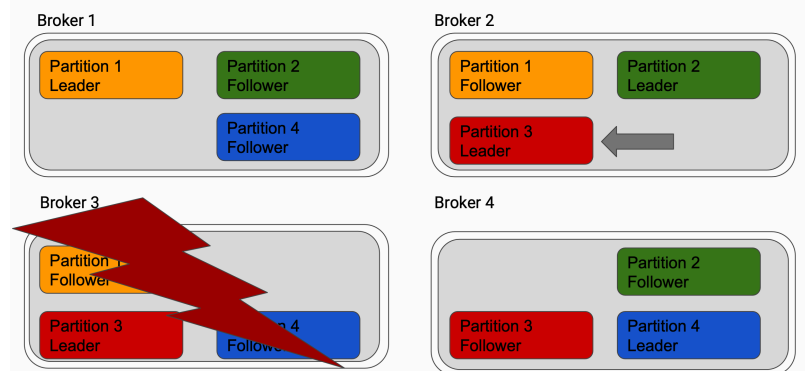
4 Brokers, 1 Topic, 4 partitions, Facteur de réplication: 3



4 Brokers, 1 Topic, 4 partitions, Facteur de réplication: 3



4 Brokers, 1 Topic, 4 partitions, Facteur de réplication: 3



Focus : Producer - Consumer

Consumer

- Un **consumer** est une application ou un processus qui lit des messages d'un ou plusieurs topics Kafka.
- Chaque **consumer** se connecte à un cluster **Kafka**, s'abonne à des topics spécifiques, et commence à recevoir des messages de ces topics.

Focus : Producer - Consumer

Consumer

Lecture de messages :

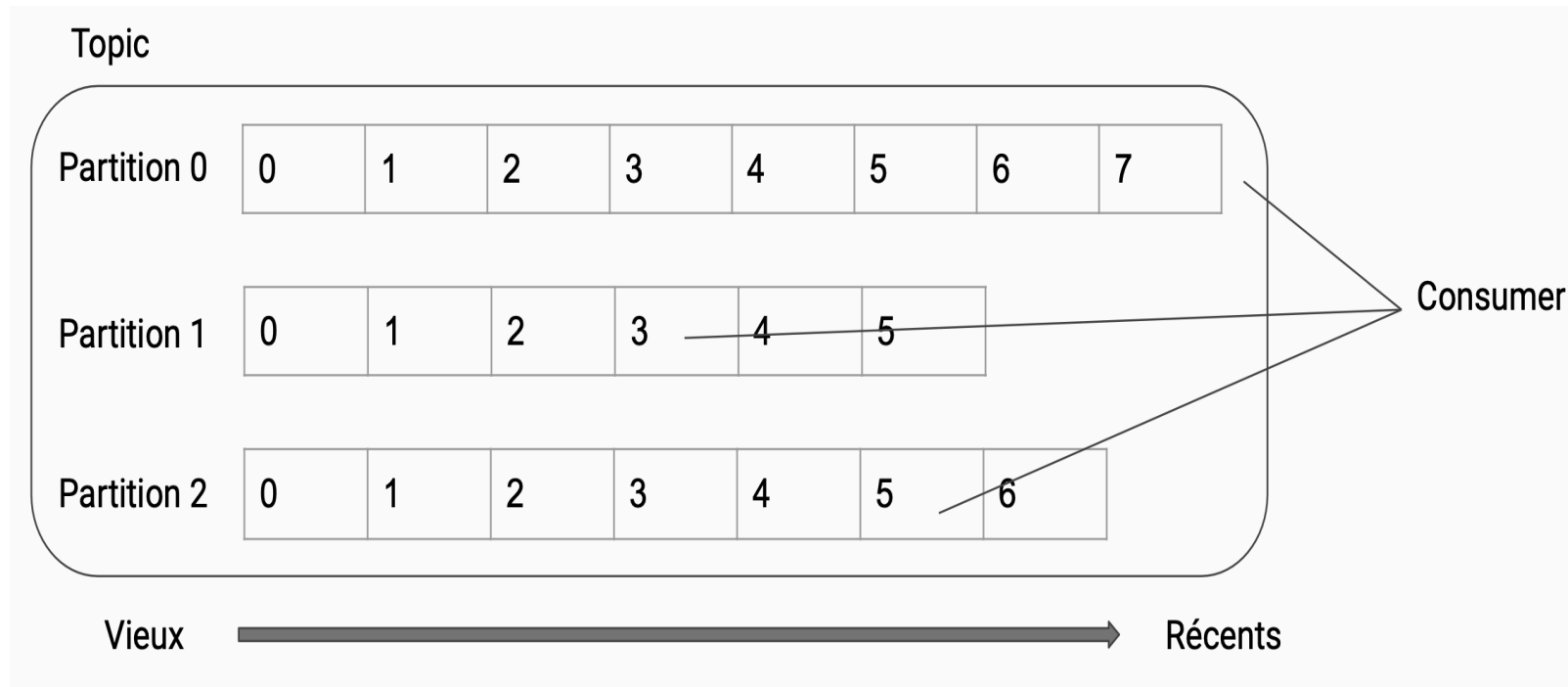
- Un **consumer** lit les messages à partir de la position où il s'est arrêté lors de la dernière consommation, connue sous le nom d'offset.
- Kafka stocke ces offsets pour chaque consumer, permettant à chaque session de reprendre la lecture là où elle s'était arrêtée.

Indépendance :

- Chaque **consumer** fonctionne indépendamment des autres consumers.
- Il peut parcourir les messages à son propre rythme et selon ses propres besoins.

Focus : Producer - Consumer

Consumer



Focus : Producer - Consumer

Consumer Group

Un consumer group est un ensemble de un ou plusieurs consumers qui collaborent pour consommer un topic ou un ensemble de topics.

Focus : Producer - Consumer

Consumer Group

Répartition de charge :

- Les **consumer groups** permettent de distribuer la charge de traitement des messages parmi plusieurs consumers.
- Chaque **consumer** dans un groupe lit les messages de une ou plusieurs partitions uniques du topic auquel il est abonné, ce qui permet de paralléliser le traitement.

Scalabilité :

- Ajouter des **consumers** à un **consumer group** permet d'échelonner la consommation d'un topic.
- Si le nombre de partitions d'un topic est plus grand que le nombre de **consumers** dans le groupe, certains **consumers** liront les messages de plusieurs partitions.
- Si le nombre de **consumers** est supérieur au nombre de partitions, certains **consumers** resteront inactifs jusqu'à ce qu'un autre **consumer** quitte le groupe.

Fiabilité :

- En utilisant des **consumer groups**, Kafka assure que chaque message d'une partition est consommé par

Focus : Producer - Consumer

Fonctionnement des Consumer Groups

Assignment des partitions :

- Lorsqu'un consumer rejoint un groupe ou qu'un nouveau topic est ajouté aux abonnements du groupe, le coordinateur de groupe (un broker choisi dans le cluster) réassigne les partitions parmi les consumers actifs du groupe.

Gestion des offsets :

- Kafka maintient l'offset de lecture pour chaque partition au sein d'un consumer group.
- Cela permet à chaque consumer de reprendre la lecture là où il s'est arrêté, même en cas de redémarrage du consumer ou de défaillance.

Heartbeats et rebalancing :

- Les consumers envoient régulièrement des signaux de présence (heartbeats) au coordinateur de groupe pour signaler qu'ils sont actifs et fonctionnent correctement.
- Si un consumer cesse d'envoyer des heartbeats, le coordinateur déclenche un rééquilibrage des partitions entre les consumers restants.

Focus : Producer - Consumer

Producer

- Un **producer** dans Kafka est une entité ou un processus responsable de la publication de messages dans les topics Kafka.
- Les **producers** envoient des données à Kafka en choisissant explicitement le topic et, optionnellement, la partition au sein de ce topic où les messages doivent être écrits.

Focus : Producer - Consumer

Fonctionnalités clés d'un Producer

1. Choix de la Partition :

- Les producers peuvent spécifier la partition au sein d'un topic pour envoyer un message.
- Si aucune partition n'est spécifiée, Kafka peut assigner une partition basée sur une clé fournie avec le message ou de manière round-robin si aucune clé n'est fournie.

2. Sérialisation des Messages :

- Les données envoyées par les producers doivent être sérialisées en un format binaire avant d'être envoyées à Kafka. Kafka supporte plusieurs sérialisateurs pour des formats tels que String, Byte array, Avro, JSON, etc.

Focus : Producer - Consumer

Fonctionnalités clés d'un Producer

3. Fiabilité et Durabilité :

- Kafka offre différentes configurations pour garantir la fiabilité et la durabilité des messages envoyés.
- Par exemple, les producers peuvent configurer l'acknowledgment (acks) pour s'assurer que les messages ont été correctement reçus et enregistrés par le cluster.

4. Compression :

- Les messages peuvent être compressés par le producer avant l'envoi pour économiser de la bande passante et améliorer les performances.
- Kafka supporte plusieurs algorithmes de compression comme GZIP, Snappy, LZ4, et Zstd.

kafka Connect

kafka Connect

Introduction à Kafka Connect

1. Qu'est-ce que Kafka Connect ?

- Kafka Connect est une couche d'intégration dédiée au sein de l'écosystème Apache Kafka.
- Elle permet d'automatiser le transfert des données entre Kafka et divers systèmes externes comme les bases de données, les systèmes de fichiers, et les services de données cloud.
- Son but est de simplifier et rationaliser les pipelines de données sans nécessiter de codage intensif, en utilisant des connecteurs préexistants ou personnalisés.

kafka Connect

Introduction à Kafka Connect

2. Rôle et utilisation

- Le rôle principal de Kafka Connect est de faciliter l'importation (source) et l'exportation (sink) de données.
- En mode source, il extrait les données de systèmes externes vers Kafka.
- En mode sink, il envoie les données de Kafka vers d'autres systèmes.
- Cette dualité permet une intégration bidirectionnelle et dynamique adaptée aux environnements de données modernes.

kafka Connect

Introduction à Kafka Connect

3. Architecture de Kafka Connect

- **Connecteurs** : Ce sont des plugins qui implémentent la logique spécifique pour interagir avec les systèmes externes. Ils encapsulent le code nécessaire pour extraire ou charger les données.
- **Tâches** : Les connecteurs peuvent se décomposer en plusieurs tâches qui sont des unités de travail distribuables. Cela permet une parallélisation efficace des opérations de données.
- **Workers** : Ce sont des processus qui exécutent les tâches. Ils peuvent être configurés en mode standalone pour des scénarios de développement ou de test, ou en mode distribué pour des environnements de production, offrant ainsi scalabilité et résilience.

kafka Connect

Introduction à Kafka Connect

4. Configuration et gestion via API REST

- Kafka Connect est géré via une API REST, ce qui permet de configurer, de surveiller et de modifier les connecteurs dynamiquement.
- L'API fournit des points d'accès pour :
 - Ajouter ou supprimer des connecteurs.
 - Vérifier l'état des connecteurs et des tâches.
 - Réajuster les configurations sans interruption de service.

kafka Connect

Introduction à Kafka Connect

5. Transformation des messages (Single Message Transforms - SMT)

- Kafka Connect supporte des transformations simples des messages (SMT) qui permettent de modifier les données à la volée lorsqu'elles passent à travers les connecteurs.
- Ceci est utile pour des tâches telles que le filtrage de données, la modification de schémas, ou l'ajout de métadonnées.

kafka Connect

Introduction à Kafka Connect

5. Transformation des messages (Single Message Transforms - SMT)

- Kafka Connect supporte des transformations simples des messages (SMT) qui permettent de modifier les données à la volée lorsqu'elles passent à travers les connecteurs.
- Ceci est utile pour des tâches telles que le filtrage de données, la modification de schémas, ou l'ajout de métadonnées.

Streaming

Kafka Streaming

Introduction à Kafka Streams

1. Qu'est-ce que Kafka Streams ?

- Kafka Streams est une bibliothèque de traitement de flux de données légère, entièrement intégrée à Kafka, conçue pour des applications Java.
- Elle permet de développer des applications robustes, scalables et facilement déployables, qui traitent les flux de données en continu.

Kafka Streaming

Introduction à Kafka Streams

2. Caractéristiques principales

- **Simplicité** : Utilise la simplicité de Kafka pour le traitement de flux sans nécessiter un cluster séparé.
- **Intégration profonde avec Kafka** : Conçue spécifiquement pour Apache Kafka, elle bénéficie d'une intégration optimale pour la gestion des états, la répartition de la charge et la reprise après incident.
- **Déploiement flexible** : Peut être déployé de manière indépendante ou en tant que microservice dans un conteneur, sans nécessiter d'infrastructure de gestion de cluster dédiée.

Kafka Streaming

Introduction à Kafka Streams

3. Concepts fondamentaux

- **Topologies de traitement** : Les applications Kafka Streams sont construites autour de topologies de traitement, qui sont des graphes acycliques dirigés (DAG) composés de sources, de processeurs et de sinks.
- **Stream et Table** : Deux abstractions principales, où un stream représente un journal immuable de données et une table représente un état évolutif.
- **Time Windowing** : Supporte différentes fenêtrisations (tumbling, hopping, sliding, session) pour le traitement des données basé sur le temps.

Kafka Streaming

Introduction à Kafka Streams

4. Architecture de Kafka Streams

- **Stream Threads et Stream Tasks** : Le parallélisme est géré par des threads de flux, qui peuvent exécuter une ou plusieurs tâches de flux.
- **State Stores** : Utilisés pour stocker des données temporelles et fournir une capacité de requête stateful.
- **Processor API et DSL** : Deux API qui permettent la construction de topologies. DSL (Domain Specific Language) pour des opérations de haut niveau, et Processor API pour des contrôles de bas niveau.

Kafka Streaming

Introduction à Kafka Streams

```
StreamsBuilder builder = new StreamsBuilder();
KStream<String, String> textLines = builder.stream("input-topic");
KTable<String, Long> wordCounts = textLines
    .flatMapValues(textLine -> Arrays.asList(textLine.toLowerCase().split("\\W+")))
    .groupBy((key, word) -> word)
    .count();
wordCounts.toStream().to("output-topic", Produced.with(Serdes.String(), Serdes.Long()));
KafkaStreams streams = new KafkaStreams(builder.build(), props);
streams.start();
```

Sécuriser un cluster kafka

Sécuriser un cluster kafka

1. Introduction à la Sécurité dans kafka

Les enjeux de la sécurité dans les systèmes distribués incluent :

- **Confidentialité** : Assurer que les données sensibles ne soient accessibles qu'aux entités autorisées.
- **Intégrité** : Garantir que les données ne soient pas altérées durant leur transmission ou stockage.
- **Disponibilité** : Maintenir le service accessible pour les utilisateurs légitimes, même en cas d'attaques ou de pannes.

Sécuriser un cluster kafka

1. Introduction à la Sécurité dans kafka

Importance de la Sécurité pour les Systèmes Distribués

- Dans les architectures modernes, les systèmes distribués jouent un rôle crucial en permettant le traitement et le stockage de données à grande échelle.
- Apache kafka, en tant que plateforme de streaming distribuée, facilite la transmission de données en temps réel entre différents composants d'une application.
- La nature distribuée de ces systèmes implique que les données transitent sur différents réseaux et serveurs, ce qui les expose à diverses vulnérabilités et attaques.
- La sécurisation de ces données devient donc impérative pour protéger l'intégrité, la confidentialité et la disponibilité de l'information

Sécuriser un cluster kafka

1. Introduction à la Sécurité dans kafka

Vue d'ensemble des Menaces Courantes sur les Clusters kafka :

- **Écoutes clandestines (Eavesdropping)** : Des attaquants peuvent intercepter des données non chiffrées en transit entre les producteurs, les brokers et les consommateurs, compromettant ainsi la confidentialité des données échangées.
- **Usurpation d'identité (Spoofing)** : Des individus malveillants peuvent se faire passer pour des utilisateurs légitimes ou des services, permettant l'accès non autorisé à des données sensibles ou la possibilité d'injecter des données malveillantes.
- **Attaques par Rejeu (Replay Attacks)** : Si un attaquant capture des données authentiques, il peut tenter de les renvoyer au système pour déclencher des actions indésirables.

Sécuriser un cluster kafka

1. Introduction à la Sécurité dans kafka

Vue d'ensemble des Menaces Courantes sur les Clusters kafka :

- **Attaques par Déni de Service (DoS)** : Les attaquants peuvent essayer de saturer le réseau ou les serveurs kafka avec un volume élevé de données, rendant le service indisponible pour les utilisateurs légitimes.
- **Modification non autorisée** : Sans contrôles d'accès adéquats, des utilisateurs non autorisés peuvent modifier les configurations du cluster ou les données stockées, altérant l'intégrité des données.
- **Manque de conformité et de gouvernance** : L'absence de politiques de sécurité et de mécanismes de surveillance adéquats peut entraîner des violations de conformité réglementaire et des risques de gouvernance des données.

Sécuriser un cluster kafka

2. Chiffrement

Le principal objectif du chiffrement dans kafka est de protéger les données contre les écoutes clandestines, assurant ainsi que les données en transit et au repos ne peuvent être lues ou compromises par des acteurs non autorisés.

Sécuriser un cluster kafka

2.1 Chiffrement en Transit

Le chiffrement en transit est essentiel pour sécuriser les données lorsqu'elles se déplacent à travers le réseau, entre les producteurs, les courtiers (brokers), et les consommateurs de kafka.

Sécuriser un cluster kafka

2.1 Chiffrement en Transit

Présentation de SSL/TLS

SSL (Secure Socket Layer) et TLS (Transport Layer Security) sont des protocoles cryptographiques qui fournissent des communications sécurisées sur un réseau informatique. Dans kafka, TLS (la version successeur de SSL) est couramment utilisé pour chiffrer les données en transit, ce qui empêche les écoutes clandestines et garantit l'intégrité des données.

Sécuriser un cluster kafka

2.1 Chiffrement en Transit

Configuration de kafka pour utiliser SSL/TLS

La configuration de kafka pour utiliser SSL/TLS implique plusieurs étapes clés :

- La génération de certificats et de clés.
- La configuration des courtiers kafka.
- La configuration des clients kafka.
- La validation de la configuration SSL/TLS.

Sécuriser un cluster kafka

2.1 Chiffrement en Transit

Génération de Certificats et de Clés

1. **Créer une Autorité de Certification (CA)** : Pour émettre des certificats, vous pouvez soit utiliser une CA existante soit en créer une pour votre organisation.

```
openssl req -new -x509 -keyout ca-key -out ca-cert -days 365
```

Sécuriser un cluster kafka

2.1 Chiffrement en Transit

Génération de Certificats et de Clés

2. Créer des Clés et des Certificats pour les Courtiers : Chaque courtier kafka (Broker) doit avoir sa propre paire de clés et certificat signé par la CA.

```
openssl req -new -keyout kafka-broker-key -out kafka-broker-req -days 365  
openssl x509 -req -in kafka-broker-req -CA ca-cert -CAkey ca-key -CAcreateserial -out kafka-broker-cert -days 365
```

Sécuriser un cluster kafka

2.1 Chiffrement en Transit

Génération de Certificats et de Clés

3. **Importer les Certificats et Clés dans un Keystore Java** : kafka utilise Java Keystores pour gérer les certificats et les clés.

```
keytool -keystore kafka.server.keystore.jks -alias localhost -validity 365 -genkey
```

Sécuriser un cluster kafka

2.1 Chiffrement en Transit

Configuration des Courtiers kafka

Modifier le fichier `server.properties` pour inclure les configurations SSL/TLS, par exemple :

```
listeners=SSL://your.broker.host.name:9093
ssl.keystore.location=/path/to/kafka.server.keystore.jks
ssl.keystore.password=yourkeystorepassword
ssl.key.password=yourkeypassword
ssl.truststore.location=/path/to/kafka.server.truststore.jks
ssl.truststore.password=yourtruststorepassword
```


Sécuriser un cluster kafka

2.1 Chiffrement en Transit

Configuration des Clients kafka

Les clients producteurs et consommateurs doivent également être configurés pour utiliser SSL/TLS. Cela implique la spécification des chemins vers les keystores et truststores, ainsi que des mots de passe correspondants dans leurs configurations.

```
props.put("security.protocol", "SSL");  
props.put("ssl.truststore.location", "/path/to/client.truststore.jks");  
props.put("ssl.truststore.password", "yourtruststorepassword");
```

Sécuriser un cluster kafka

2.1 Chiffrement en Transit

Validation de la Configuration SSL/TLS

Pour vérifier que la configuration SSL/TLS fonctionne comme prévu, vous pouvez démarrer kafka et tenter une communication chiffrée entre un client et le serveur. Les logs doivent indiquer une connexion sécurisée établie, et les données transmises ne devraient pas être lisibles en clair sur le réseau.

Sécuriser un cluster kafka

2.2 Chiffrement au Repos

Le chiffrement au repos fait référence à la pratique de chiffrer les données stockées sur un disque ou tout autre support de stockage. L'objectif est de garantir que les données soient inaccessibles ou illisibles sans les clés de chiffrement appropriées, même si un attaquant parvient à accéder physiquement au stockage.

Sécuriser un cluster kafka

2.2 Chiffrement au Repos

- Le **chiffrement au repos** fait référence à la pratique de chiffrer les données stockées sur un disque ou tout autre support de stockage. L'objectif est de garantir que les données soient inaccessibles ou illisibles sans les clés de chiffrement appropriées, même si un attaquant parvient à accéder physiquement au stockage.
- Dans kafka, le **chiffrement au repos** n'est pas fourni nativement par la plateforme. kafka se concentre principalement sur le passage et le traitement des messages, laissant le chiffrement au repos être géré par l'infrastructure sous-jacente ou par des intégrations avec des outils tiers.

Sécuriser un cluster kafka

2.1 Chiffrement au Repos

Options pour le Chiffrement au Repos dans kafka

1. Intégration avec des Systèmes de Fichiers Chiffrés :

- Utiliser des solutions de système de fichiers chiffrés comme Linux Unified Key Setup (LUKS) ou des systèmes de fichiers comme EncFS pour chiffrer les volumes de données où kafka stocke ses journaux (logs) et données.
- Cette approche assure que toutes les données écrites sur le disque par kafka sont automatiquement chiffrées par le système de fichiers.

Sécuriser un cluster kafka

2.1 Chiffrement au Repos

Options pour le Chiffrement au Repos dans kafka

2. Intégration avec des Solutions de Gestion des Clés de Chiffrement (KMS) :

- Utiliser un service de gestion des clés de chiffrement comme AWS KMS, Azure Key Vault, ou Google Cloud KMS pour gérer les clés de chiffrement utilisées pour chiffrer les données au repos.
- Les clés peuvent être utilisées pour chiffrer les données avant qu'elles ne soient stockées sur le disque ou pour chiffrer les volumes de stockage au niveau du système d'exploitation ou du cloud.

Sécuriser un cluster kafka

2.1 Chiffrement au Repos

Options pour le Chiffrement au Repos dans kafka

3. Utilisation de Disques Auto-chiffnants :

- Certains disques durs et SSDs offrent des fonctionnalités de chiffrement matériel intégré, connues sous le nom de disques auto-chiffnants (SEDs).
- Ces dispositifs peuvent être utilisés pour fournir une couche supplémentaire de sécurité pour les données stockées par kafka, avec une gestion des clés intégrée dans le matériel.

Sécuriser un cluster kafka

3. Authentification

- L'authentification dans Apache kafka est cruciale pour garantir que seuls les utilisateurs et systèmes autorisés peuvent accéder et interagir avec le cluster kafka.
- L'authentification par SSL/TLS, en particulier, joue un rôle clé dans la sécurisation de l'accès au cluster.
- Elle permet non seulement de chiffrer les données en transit entre les clients et les serveurs mais aussi d'authentifier les parties impliquées grâce à l'utilisation de certificats numériques.

Sécuriser un cluster kafka

3.1 Authentification par SSL/TLS

Principe de l'Authentification Mutuelle SSL

- L'authentification dans Apache kafka est cruciale pour garantir que seuls les utilisateurs et systèmes autorisés peuvent accéder et interagir avec le cluster kafka.
- L'authentification par SSL/TLS, en particulier, joue un rôle clé dans la sécurisation de l'accès au cluster.
- Elle permet non seulement de chiffrer les données en transit entre les clients et les serveurs mais aussi d'authentifier les parties impliquées grâce à l'utilisation de certificats numériques.

Sécuriser un cluster kafka

3.2 Authentification SASL

L'authentification SASL (Simple Authentication and Security Layer) joue un rôle crucial dans la sécurisation des communications entre les clients et les serveurs, y compris dans des systèmes distribués comme Apache kafka. SASL agit comme une couche d'abstraction qui prend en charge divers mécanismes d'authentification sans requérir des modifications au protocole sous-jacent.

Sécuriser un cluster kafka

3.2 Authentification SASL

Présentation de SASL

- SASL est un framework d'authentification qui permet d'ajouter des mécanismes d'authentification aux protocoles basés sur une connexion.
- Il fournit un mécanisme standard pour les applications client et serveur pour s'authentifier mutuellement.
- SASL supporte plusieurs mécanismes d'authentification, permettant ainsi une flexibilité considérable en termes de sécurité et de compatibilité.

Sécuriser un cluster kafka

3.2 Authentification SASL

Différents mécanismes SASL pris en charge par kafka

- **PLAIN** : Un mécanisme simple qui transmet le nom d'utilisateur et le mot de passe en clair (non chiffré) sur le réseau. Bien que facile à configurer, il est recommandé de l'utiliser seulement avec une connexion SSL/TLS pour assurer la confidentialité des informations d'authentification.
- **SCRAM (Salted Challenge Response Authentication Mechanism)** : SCRAM améliore la sécurité en ajoutant un échange de challenge-response pour éviter l'envoi des mots de passe en clair. kafka supporte SCRAM-SHA-256 et SCRAM-SHA-512, qui se réfèrent aux algorithmes de hash utilisés.
- **GSSAPI (Kerberos)** : GSSAPI est un mécanisme d'authentification basé sur des tickets pour authentifier les services. kafka peut utiliser Kerberos pour authentifier les clients et les serveurs, offrant une méthode robuste pour gérer l'authentification dans des environnements d'entreprise.

Sécuriser un cluster kafka

3.2 Authentification SASL

Configuration de kafka pour utiliser SASL

- 1. Choisir le mécanisme SASL adapté à votre environnement et à vos exigences de sécurité.
- 2. Configurer les brokers kafka pour activer SASL et spécifier le mécanisme choisi.
- 3. Cela comprend la modification du fichier `server.properties` pour inclure les paramètres SASL appropriés.
- 4. Configurer les clients kafka pour utiliser SASL lors de la connexion aux brokers.
- 5. Cela implique souvent la spécification de propriétés dans le fichier de configuration du client ou dans le code client lui-même.
- 6. (Optionnel) Configurer un canal sécurisé (SSL/TLS) pour protéger les données en transit, surtout si vous utilisez le mécanisme PLAIN.

Sécuriser un cluster kafka

3.2 Authentification SASL

Sécurisation de la configuration SASL

Pour sécuriser davantage la configuration SASL de kafka, vous pouvez prendre les mesures suivantes :

- Utiliser des mécanismes SASL plus sécurisés comme SCRAM ou GSSAPI/Kerberos au lieu de PLAIN, sauf si vous êtes sur un réseau sécurisé.
- Activer SSL/TLS pour chiffrer les données en transit, particulièrement important si vous utilisez le mécanisme PLAIN.
- Gérer soigneusement les identifiants en évitant de stocker des mots de passe en clair et en utilisant des systèmes de gestion des secrets.
- Mettre en place des politiques de sécurité strictes autour de l'accès aux brokers kafka et aux données qu'ils gèrent.

Sécuriser un cluster kafka

3.2 Authentification SASL

Pratique : Configuration de kafka pour utiliser SASL/PLAIN

Sur les brokers kafka:

1. Modifier le fichier server.properties:

```
#Activez SASL/PLAIN en ajoutant ou en modifiant la ligne suivante  
listeners=SASL_PLAINTEXT://your.broker.hostname:9092  
#Spécifiez le mécanisme SASL comme PLAIN:  
sasl.mechanism.inter.broker.protocol=PLAIN  
sasl.enabled.mechanisms=PLAIN  
#Ajoutez les configurations de sécurité et d'authentification:  
security.inter.broker.protocol=SASL_PLAINTEXT
```

Sécuriser un cluster kafka

3.2 Authentification SASL

Pratique : Configuration de kafka pour utiliser SASL/PLAIN

Sur les brokers kafka:

2. *Créer un fichier JAAS pour configurer le fournisseur d'authentification:*

- Créez un fichier kafka_server_jaas.conf avec le contenu suivant:

```
kafkaServer {  
  org.apache.kafka.common.security.plain.PlainLoginModule required  
  username="admin"  
  password="admin-secret"  
  user_admin="admin-secret"  
  user_kafka="kafka-secret";  
};
```


Sécuriser un cluster kafka

3.2 Authentification SASL

Pratique : Configuration de kafka pour utiliser SASL/PLAIN

Sur les brokers kafka:

2. *Créer un fichier JAAS pour configurer le fournisseur d'authentification:*

- Configurez l'environnement du serveur pour utiliser ce fichier JAAS en ajoutant à votre kafka-server-start.sh ou au fichier de démarrage de kafka:

```
export kafka_OPTS="-Djava.security.auth.login.config=/path/to/kafka_server_jaas.conf"
```

Sécuriser un cluster kafka

3.2 Authentification SASL

Pratique : Configuration de kafka pour utiliser SASL/PLAIN

Sur les clients kafka:

1. *Configurer le fichier JAAS pour le client:*

- Créez un fichier kafka_client_jaas.conf similaire au fichier serveur, mais adapté pour l'authentification client.
- Configurez l'environnement du client pour utiliser ce fichier JAAS.

Sécuriser un cluster kafka

3.2 Authentification SASL

Pratique : Configuration de kafka pour utiliser SASL/PLAIN

Sur les clients kafka:

2. Configurer les propriétés du client pour inclure l'authentification SASL :

- Dans les configurations de votre client kafka (producteur ou consommateur), ajoutez :

```
security.protocol=SASL_PLAINTEXT  
sasl.mechanism=PLAIN
```

Sécuriser un cluster kafka

3.2 Authentification SASL

Pratique : Configuration pour SASL/SCRAM

Les mêmes étapes que pour SASL/PLAIN, mais avec quelques modifications :

- Dans `server.properties`, utilisez `sasl.enabled.mechanisms=SCRAM-SHA-256` (ou `SCRAM-SHA-512` selon votre préférence).
- Le fichier JAAS devra utiliser `ScramLoginModule` au lieu de `PlainLoginModule`.
- Vous devrez également créer des utilisateurs SCRAM via `kafka-configs.sh` :

```
bin/kafka-configs.sh --zookeeper your.zookeeper.host:2181 --alter --add-config 'SCRAM-SHA-256=[password="user-password"]' --entity-type users --entity-name your-username
```

Sécuriser un cluster kafka

3.2 Authentification SASL

Pratique : Configuration pour Kerberos/GSSAPI

La configuration pour GSSAPI est plus complexe et nécessite un environnement Kerberos déjà configuré :

- Configurez server.properties et les fichiers JAAS comme indiqué dans la documentation officielle de kafka pour GSSAPI, y compris la définition des propriétés listeners, sasl.enabled.mechanisms, et sasl.mechanism.inter.broker.protocol pour utiliser GSSAPI.
- Assurez-vous que tous les brokers et clients kafka ont accès à un service Kerberos fonctionnel et sont configurés pour l'utiliser.

Sécuriser un cluster kafka

4. Autorisation

L'autorisation dans Apache kafka est un composant crucial pour sécuriser votre cluster en contrôlant l'accès aux ressources pour les utilisateurs et les systèmes authentifiés. Nous verrons les aspects essentiels de la gestion des autorisations dans kafka, en mettant l'accent sur les Listes de Contrôle d'Accès (ACLs) et l'utilisation du kafka Authorizer.

Sécuriser un cluster kafka

4.1 Présentation des ACLs (Listes de Contrôle d'Accès)

Concept des ACLs dans kafka

- Dans kafka, les ACLs sont utilisées pour gérer les autorisations au niveau des ressources.
- Une ACL est essentiellement une règle qui définit et contrôle l'accès aux ressources du cluster pour les utilisateurs ou les groupes d'utilisateurs.
- Les ressources peuvent inclure des sujets (topics), des groupes de consommateurs, et d'autres éléments de kafka. Les ACLs déterminent qui (le principal) est autorisé à effectuer quelle action (par exemple, lire, écrire) sur une ressource donnée.

Sécuriser un cluster kafka

4.1 Présentation des ACLs (Listes de Contrôle d'Accès)

Gestion des ACLs

La gestion des ACLs se fait via l'interface de ligne de commande (CLI) de kafka ou par programmation via l'API d'administration. Les commandes `kafka-acls.sh` permettent de lister, ajouter ou supprimer des ACLs.

Sécuriser un cluster kafka

4.1 Présentation des ACLs (Listes de Contrôle d'Accès)

Création et suppression des ACLs

Pour créer une ACL, vous utiliseriez la commande kafka-acls avec les options appropriées pour spécifier le principal, l'hôte, l'opération et la ressource. Pour supprimer une ACL, vous spécifieriez des critères similaires pour identifier l'ACL à supprimer.

Sécuriser un cluster kafka

4.1 Présentation des ACLs (Listes de Contrôle d'Accès)

Principe du moindre privilège

Il est recommandé d'appliquer le principe du moindre privilège, c'est-à-dire de ne donner aux utilisateurs que les droits strictement nécessaires à l'accomplissement de leurs tâches. Cela aide à minimiser les risques de sécurité.

Sécuriser un cluster kafka

4.1 Présentation des ACLs (Listes de Contrôle d'Accès)

Exemples de configurations d'ACLs pour différents scénarios d'utilisation

- **Scénario de production:** Accorder à un utilisateur le droit d'écrire sur un sujet spécifique.
- **Scénario de consommation:** Accorder à un groupe de consommateurs le droit de lire à partir d'un sujet spécifique.
- **Administration:** Accorder à un utilisateur des droits d'administration sur le cluster.

Sécuriser un cluster kafka

4.1 Présentation des ACLs (Listes de Contrôle d'Accès)

Pratique : Étapes pour gérer les ACLs dans kafka

1. **Identification des besoins d'accès:** Déterminez quelles opérations (lire, écrire, etc.) sont nécessaires pour chaque utilisateur ou service sur les différentes ressources (topics, groupes de consommateurs, etc.).
2. **Utilisation de la CLI kafka-acls.sh:** kafka fournit l'outil kafka-acls.sh pour la gestion des ACLs. Cet outil permet de créer, lister, et supprimer des ACLs.
3. **Création d'une ACL:** Pour créer une ACL, vous devez spécifier le principal (l'utilisateur ou le service), l'hôte, l'opération, et la ressource concernée.
4. **Vérification des ACLs existantes:** Après avoir créé des ACLs, vous pouvez les lister pour vérifier que vos configurations sont correctes.
5. **Suppression d'une ACL:** Si une ACL n'est plus nécessaire ou si elle a été créée par erreur, vous pouvez la supprimer.

Sécuriser un cluster kafka

4.1 Présentation des ACLs (Listes de Contrôle d'Accès)

Pratique : Exemple concret

1. **Identification des besoins d'accès:** Déterminez quelles opérations (lire, écrire, etc.) sont nécessaires pour chaque utilisateur ou service sur les différentes ressources (topics, groupes de consommateurs, etc.).
2. **Utilisation de la CLI kafka-acls.sh:** kafka fournit l'outil kafka-acls.sh pour la gestion des ACLs. Cet outil permet de créer, lister, et supprimer des ACLs.
3. **Création d'une ACL:** Pour créer une ACL, vous devez spécifier le principal (l'utilisateur ou le service), l'hôte, l'opération, et la ressource concernée.
4. **Vérification des ACLs existantes:** Après avoir créé des ACLs, vous pouvez les lister pour vérifier que vos configurations sont correctes.
5. **Suppression d'une ACL:** Si une ACL n'est plus nécessaire ou si elle a été créée par erreur, vous pouvez la supprimer.