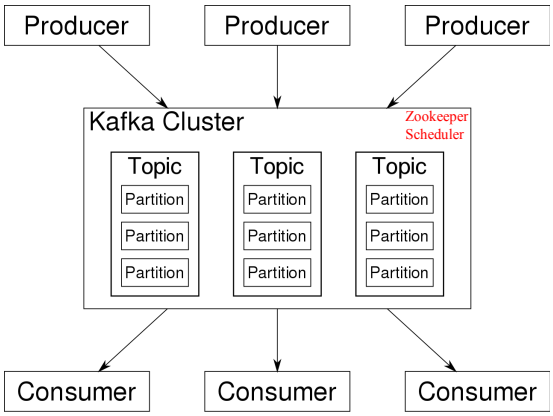
Apache Kafka

Ce document vous donnent les bases que vous devez connaître pour avoir une bonne compréhension de Kafka. Apache Kafka a été créé par LinkedIn et maintenant un projet open source majoritairement géré par Confluent. C'est un transporteur de données distribués résilients tolèrent aux pannes. Il peut être facilement mis en échelle (Scalability) jusqu'à 100 Courtiers (Brokers) et des millions de messages par seconde gérant ainsi des données en temps réel avec un temps de latence de moins de 10 ms. Apache Kafka est actuellement utilisé par plus de 2000 entreprises. Netflix utilise Kafka pour son système de recommandation en temps réel, Uber l'utilise pour collecter les données sur les véhicules, utilisateurs et les trajets en temps réel, LinkedIn l'utilise pour la gestion des Spams et les interactions des utilisateurs, etc. La meilleure façon de tester Kafka est probablement d'utiliser un service cloud comme, Confluent Cloud.



Événements. Presque tous les programmes que vous avez déjà écrits répondent à des événements de quelque nature que ce soit: le déplacement de la souris, une entrée saisie, les formulaires Web soumis, etc. Kafka vous encourage à voir le monde comme des séquences d'événements, qu'il modélise comme des paires de clés-valeurs. Les événements sont immuables, car il est impossible de changer le passé.

Sujets (Topics)Kafka. Parce que le monde est rempli de tant d'événements, Kafka nous donne un moyen de les organiser et de les maintenir en ordre: des sujets. Un sujet est un journal ordonné des événements. Lorsqu'un système externe écrit un événement dans Kafka, il est ajouté à la fin d'un sujet. Par défaut, les messages ne sont pas supprimés des sujets tant qu'un délai configurable n'est pas écoulé, même s'ils ont été lus. Les sujets sont des journaux , pas des files d'attente; ce sont des enregistrements durables, répliqués et tolérants aux pannes des événements qui y sont stockés. Les journaux sont une structure de données très pratique qui est efficace à stocker et à maintenir. Vous ne pouvez pas vraiment analyser un journal, ni l'interroger, nous devrons donc ajouter des fonctionnalités pour le rendre plus accessible.

Partitionnement. Les sujets sont stockés sous forme de fichiers journaux sur le disque et les disques sont limite de taille. Ce ne serait pas bon si notre capacité à stocker des événements était limitée aux disques sur un seul serveur, ou si notre capacité à publier de nouveaux événements sur un sujet ou notre accès à des mises à jour sur ce sujet était limitée aux capacités d'E/S d'un seul serveur. Pour pouvoir évoluer et se mettre à l'échelle (Scalability), Kafka nous donne la possibilité de diviser les sujets en partitions. Les partitions sont un moyen systématique de diviser le fichier journal d'un sujet en plusieurs journaux, chacun d'entre eux pouvant être hébergé sur un serveur distinct. Cela nous donne en principe la capacité de redimensionner les sujets à l'infini. On utilise ainsi un numéro de séquence (offSet) pour identifier les messages dans une partition. Donc, pour accéder a une message, il faut trois information, Topic-Partition-OffSet.

Courtiers (Brokers) Kafka. Kafka est une infrastructure de données distribuée, ce qui implique qu'il existe des nœuds qui peuvent être dupliqués sur un réseau de sorte que l'ensemble de tous ces nœuds fonctionne ensemble comme un seul cluster Kafka. Ce nœud s'appelle un courtier. Un courtier peut s'exécuter sur du matériel bas niveau, une instance cloud, dans un conteneur géré par Kubernetes, dans Docker sur votre ordinateur portable, ou partout où les processus JVM peuvent s'exécuter. Les courtiers Kafka sont intentionnellement maintenus très simples. Ils sont responsables de l'écriture de nouveaux événements sur les partitions, de l'exécution des lectures sur les partitions existantes et de la réplication des partitions entre elles. Ils ne font aucun calcul sur les messages ou le routage des messages entre les sujets.

Réplication. En tant que composant responsable de l'infrastructure de données, Kafka fournit un stockage répliqué des partitions des sujets. Chaque sujet a un facteur de réplication configurable qui détermine le nombre total de ces copies dans le cluster. L'une des répliques est élue pour être le leader, et c'est sur cette réplique que toutes les écritures sont produites et à partir de laquelle toutes les lectures sont probablement consommées. Les autres répliques sont appelées suiveurs (followers), et leur travail est de rester à jour avec le leader et être éligible à l'élection en tant que nouveau leader si le courtier hébergeant le leader actuel tombe en panne.

Producteurs Kafka. Une fois que le cluster Kafka est opérationnel avec son ensemble de fonctionnalités minimales, nous devons pouvoir le connecter au monde extérieur. Un producteur est une application externe qui écrit des messages sur un cluster Kafka, communiquant avec le cluster à l'aide du protocole réseau de Kafka. Ce protocole réseau est une chose publiquement documentée. Apache Kafka fournit une bibliothèque Java et Confluent qui prend en charge les bibliothèques en Python, C/C ++, .NET, etc. La bibliothèque du producteur gère toute la communication réseau non triviale entre votre programme client et le cluster et prend également des décisions telles que la façon d'attribuer de nouveaux messages aux partitions de sujet. La bibliothèque du producteur est complexe avec ses composants internes multiples, mais l'API pour la tâche de base d'écrire un message dans un sujet est très simple.

Consommateurs Kafka. Le consommateur est une application externe qui lit les messages des sujets Kafka et travaille avec eux, pour les filtrer, les agréger ou les enrichir avec d'autres sources d'informations. Comme le producteur, il s'appuie sur une bibliothèque cliente pour gérer l'interface réseau de bas niveau en plus d'autres fonctionnalités assez sophistiquées. Un consommateur peut être une seule instance ou plusieurs instances du même programme: un groupe de consommateurs. Les groupes de consommateurs sont extensibles de manière élastique.

Écosystème Kafka. Avec le partitionnement, la production et la consommation, d'autres besoins sont nécessaires. Vous avez besoin d'une intégration de données, d'une gestion des schémas et d'options pour le traitement des flux. La communauté Kafka et la communauté Confluent ont résolu la majorité de ces problèmes. Comme la planification avec Zookeeper.

Kafka Connect. Kafka Connect est un système permettant de connecter des systèmes non-Kafka à Kafka de manière déclarative, sans vous obliger à écrire un tas de code d'intégration pour vous connecter aux mêmes systèmes auxquels le reste du monde se connecte. Connect s'exécute en tant que cluster évolutif et tolérant aux pannes de machines externes au cluster Kafka. Plutôt que d'écrire du code sur mesure pour lire des données à partir d'une base de données ou écrire des messages dans Elasticsearch, vous déployez des connecteurs prédéfinis à partir du vaste écosystème de connecteurs existants et les configurez avec JSON. Connect lit ensuite les données des systèmes sources et les écrit automatiquement sur les systèmes récepteurs.

Flux Kafka. Produire des messages à Kafka est souvent assez simple: les messages proviennent d'une source, soit lus à partir d'une entrée, soit calculés à partir d'un état antérieur, et ils vont dans un sujet. Mais la lecture se complique très vite. L'API Kafka Streams existe pour fournir une couche d'abstraction au-dessus du consommateur. Il s'agit d'une API Java qui fournit une vue fonctionnelle des primitives de traitement de flux typiques dans les consommateurs complexes: filtrage, regroupement, agrégation, jointure, etc. Il fournit une abstraction non seulement pour les flux, mais pour les flux transformés en tables, et un mécanisme pour interroger ces tables également.

ksqlDB. Écrire des applications de traitement de flux en Java est une bonne chose à faire si vous utilisez Kafka Streams, et si vous utilisez Java, et s'il est judicieux de marier la fonctionnalité de traitement de flux avec l'application elle-même. Cependant, ksqlDB est une base de données de traitement de flux orientée application pour Kafka. Un petit cluster de nœuds ksqlDB exécute des requêtes de traitement de flux continu écrites en SQL, consommant constamment des événements d'entrée et produisant des résultats dans Kafka. Il expose les mêmes abstractions de flux et de table que Kafka Streams et rend les tables interrogeables via une API JSON.