Chapitre2 KAFKA

Mr DIATTARA Ibrahima

Consultant Senior Big Data

Professeur Cloud Data Engineer à l'Université Paris Sorbonne

Objectifs

Data Streaming

Systèmes de messageries distribués

ELK

Airflow

Sommaire

1. Introduction
2. Contexte
3. Broker
4. Topic
5. Partition
6. Segment
7. Offset
8. Producer

Big Data : Principe des « 4V » (Volume, Variety, Velocity et Valeur)

Le Big Data est une nouvelle génération de technologies et d'architectures pour extraire de la valeur à partir de très large volumes et variétés (+sources , +types(structurées , semi structuré et non structurées) de données avec une grande vélocité de collecte , de transformation et d'analyse

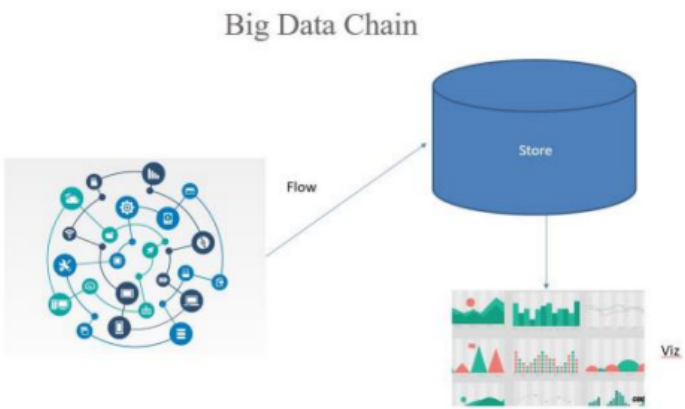
Data-Lake est une zone logique de données ou se déverse l'ensemble des données de l'entreprise pour répondre aux besoins dits Big-Data (agrégation, corrélation, traitement visualisation ..).

Data warehouses: Les données ne sont pas chargées dans l'entrepôt de données tant que leur utilisation n'a pas été définie.

Cluster: ensemble de machine avec master et une ou plusieurs esclaves

Datacenter :lieu où se trouvent différents équipements électroniques, des ordinateurs, des systèmes **de** stockage (rack, …)

Rack/Baie une armoire technique qui centralise des éléments (Disk, …)



Introudctiopn

- Apache Kafka est une plateforme distribuée de streaming de données initié en 2009 chez Linkedin puis rendu à la

  fondation Apache en 2012

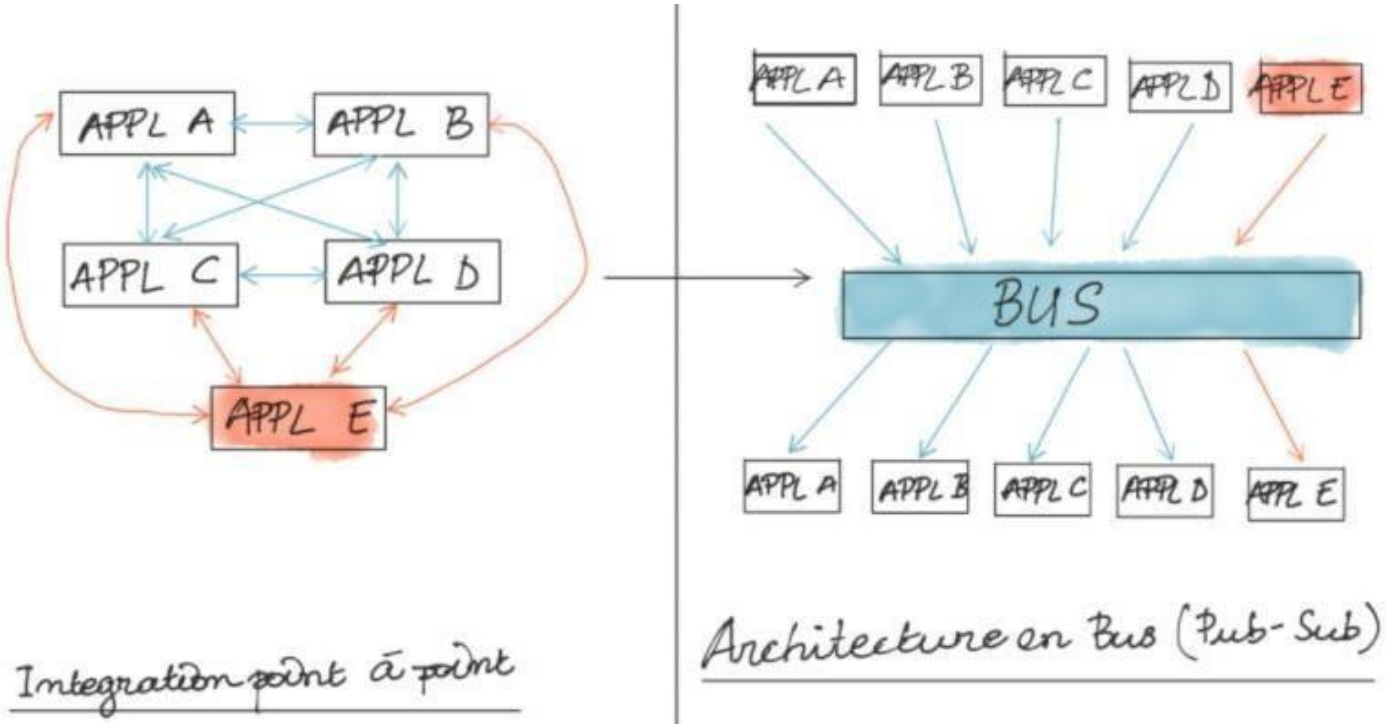
- Il est essentiellement utilisé comme un broker de messages (bus de messages).
- Il a été conçu pour des objectifs de performance (millions de messages par seconde) et de résilience

  (tolérance à la panne)

- Il est codé en Scala

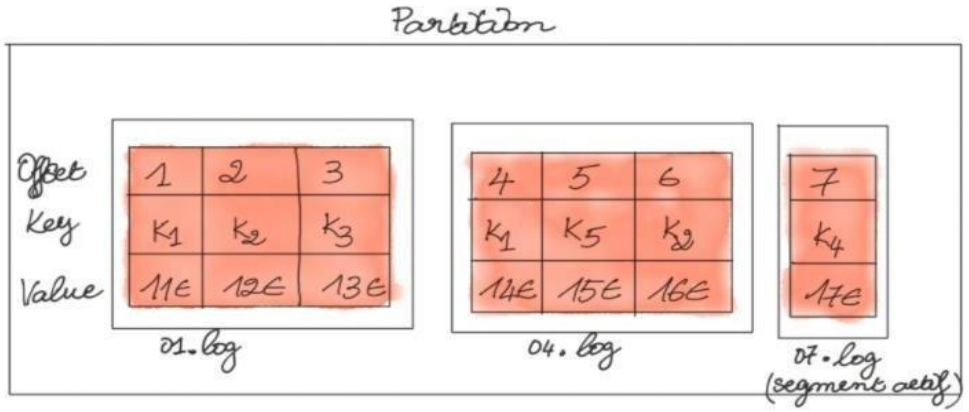
Contexte

- L'architecture bus a pour but d'éviter les intégrations point à point entre les différentes applications d'un système d'information.
- Les applications publient des messages vers un bus ou broker et toute autre application peut se connecter au bus pour les récupérer



Définitions

- **Broker**(Courtier) est techniquement une instance de votre application Apache Kafka. Vous pouvez démarrer plusieurs instances du broker pour former un cluster
- **Topic** : c' est la façon de catégoriser ou de regrouper les messages. Il faut le voir comme un rangement spécial dans le bus
- **Partition** : Un topic peut être découpé en plusieurs partitions. Une partition est définie comme une séquence immuable et ordonnée à laquelle on peut ajouter continuellement des messages à la fin (append-only).
- **Segmentent**: Fichier qui contient la data
- Un **offse**t est un indicateur qui permet de retrouver le premier message non lu pour chaque consumer.

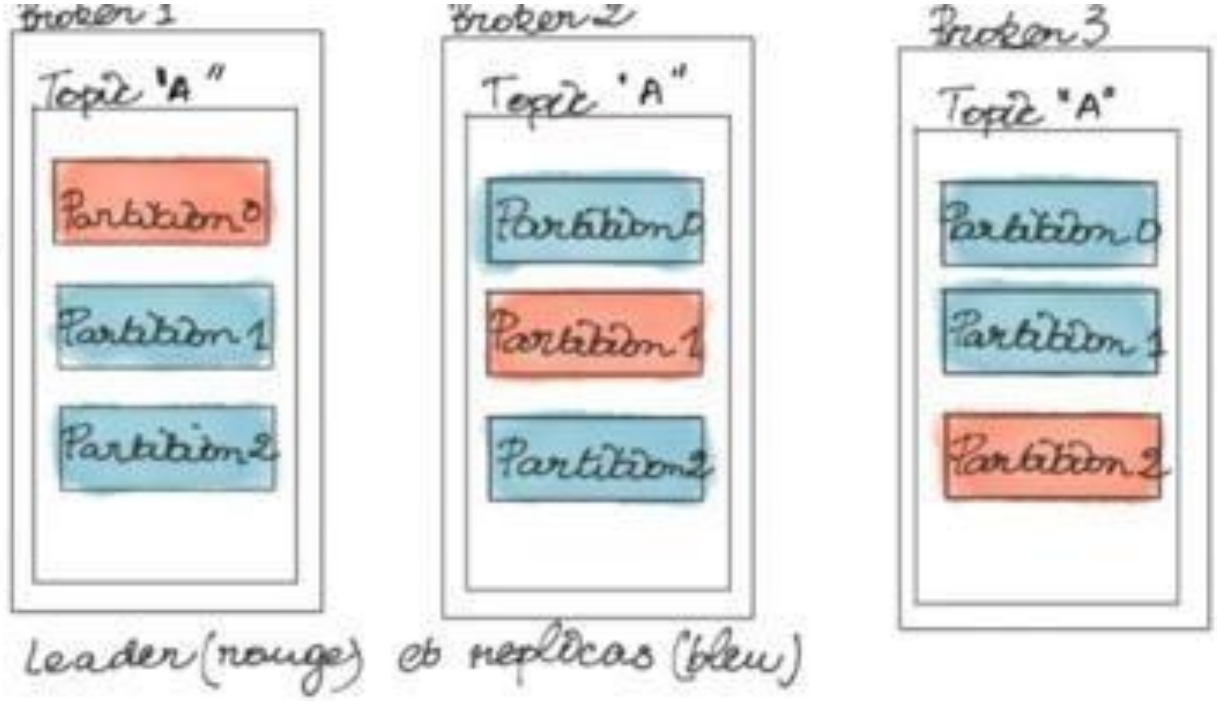


Partitions

Les partitions permettent de gérer:

Parallélisation

Tolérant à la panne



Purge

1 Mo par msg par defaut
Rétention 7 jours

- Kafka dispose d'un mécanisme interne qui permet de **purger** les anciens messages pour libérer de l'espace.

- La politique de rétention peut être basée sur le temps ou la taille des segments. Grâce à ce mécanisme les anciens segments sont progressivement supprimés.

Create Topic

./kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 3 --topic diattara

Create a topic

```
kafka/bin/kafka-topics.sh --create \
--zookeeper localhost:2181 \
--replication-factor 2 \
--partitions 3 \
--topic unique-topic-name
```

- log.retention.hours=1
- log.retention.check.interval.ms=30000

Producer

- Un producer est tout système qui **publie** dans un topic Kafka
- Un producer ne pourra ajouter un nouveau message qu'à la **fin de la séquence** d'une partition mais ne pourra pas **supprimer** ou **réordonner** les messages existants.
- Le producer n'écrit que dans la partition leader. Kafka se charge de synchroniser cette dernière avec les autres partitions réplicas.
- Un producer, en publiant vers un topic qui a plusieurs partitions, voit son **écriture parallélisée**
- On ne **peut pas écrire simultanément** sur la même partition
- Quand un producer publie un message vers un topic avec plusieurs partitions, les messages sont distribués de façon **aléatoire** sur les partitions.
- Pours'assurer qu'un type de message donné se retrouve toujours sur la même partition, il faut leur assigner la **même clé**

Producer: Partitionner

- Si les messages ont une **clé non-nulle,** le partitionner va répartir les messages entre les différentes partitions en fonction de leur clé et va garantir que des messages d'une même clé seront envoyés vers la même partition.
- Si la **clé est null** alors le partitionner va répartir ces messages entre différentes partitions de manière équilibrée

Donc pour garantir l'ordre des messages on doit utiliser le cas avec clé non null

Warning on peut avoir des hot spot pour le cas clé non null

Kafka producer delivery semantics Producer:Ack

- 0 : envoie le messagesans se soucier de l'acquittement du broker ⇔ **Best Effort**
- 1 : envoie le messageet ce dernierest acquittéune foisqu'il est écrit sur la partitionleader ⇔ **Guarantee Single Node Delivery**
- -1 ou all : envoie le messageet ce dernierest acquitté quandles partitions ISRont tous reçu le message ⇔ **Guarantee Replicated Delivery**

Acks impact

The table below summarizes the impact of acks property on latency, throughput, and durability.

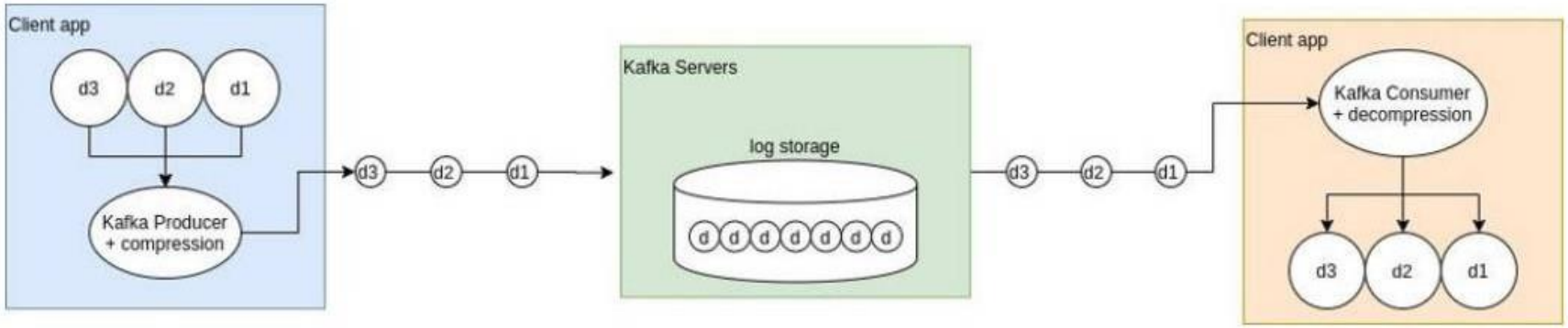
Acks	Latency	Throughput	Durability
0	Low	High	No guarantee
1	Medium	Medium	Leader only
All	High	Low	All Replicas

Compression

Les producer peuvent compresser la data
La compression joue deux roles:

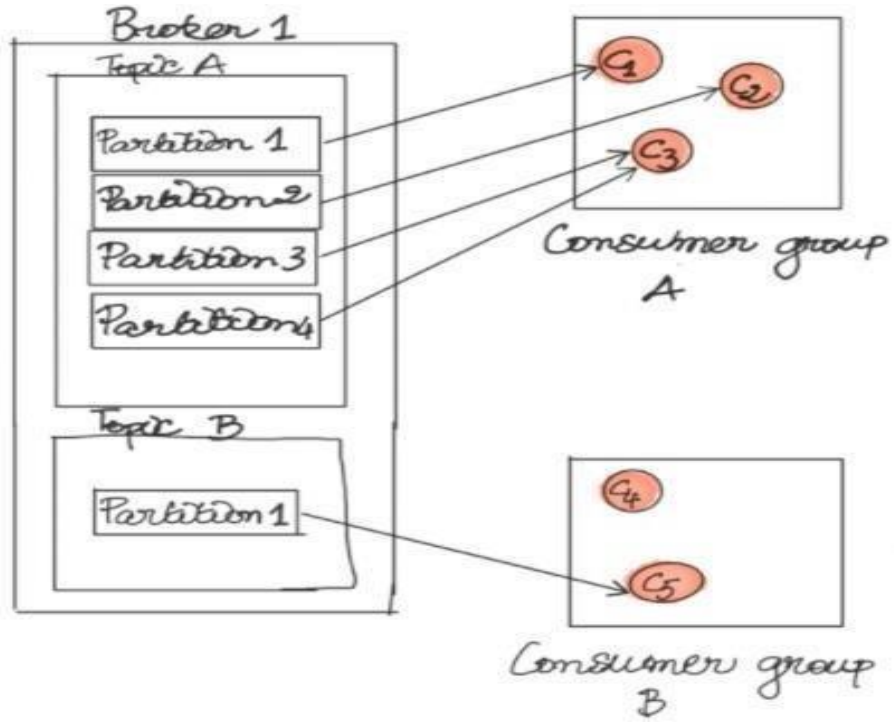
- Reduire le network bandwidth .
- Réduire l'espace Disk sur les Brokers

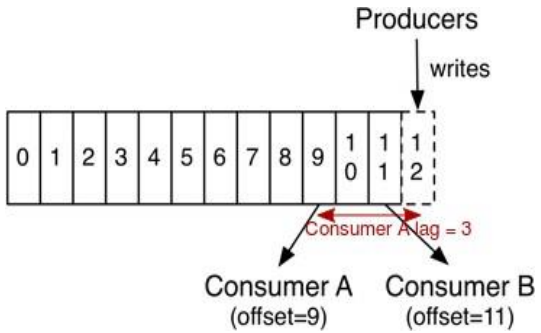
Le seul compromis avec ces deux avantages de la compression est une utilisation CPU légèrement plus élevée



Consumer

Un **consumer** est tout système qui lit des messages dans les topics Kafka. Plusieurs consumers peuvent lire des messages dans le même topic à des rythmes différents.

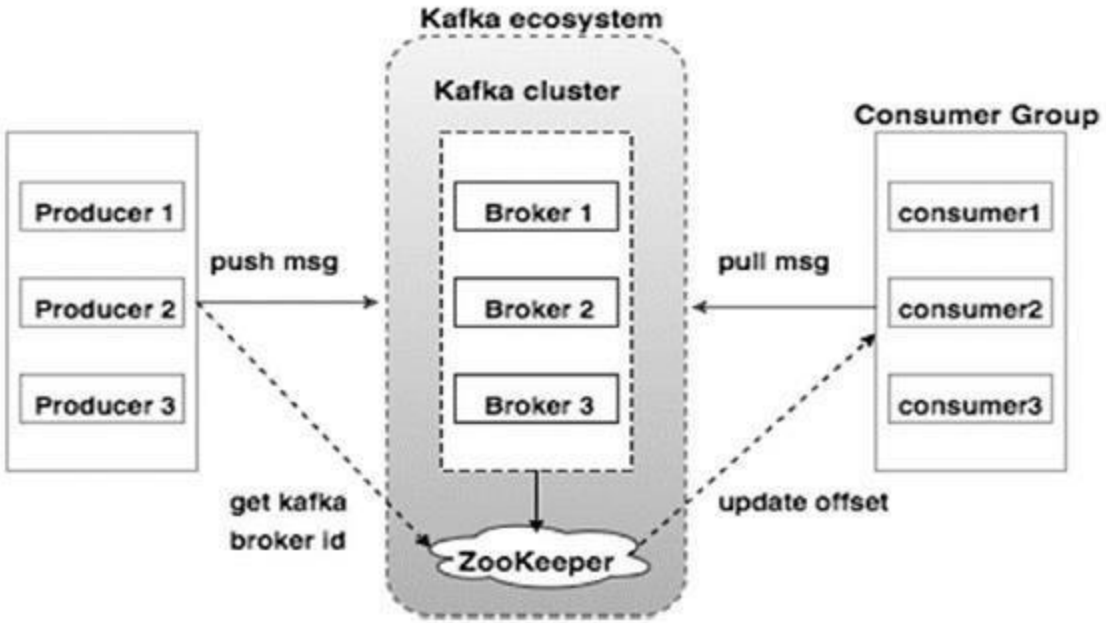


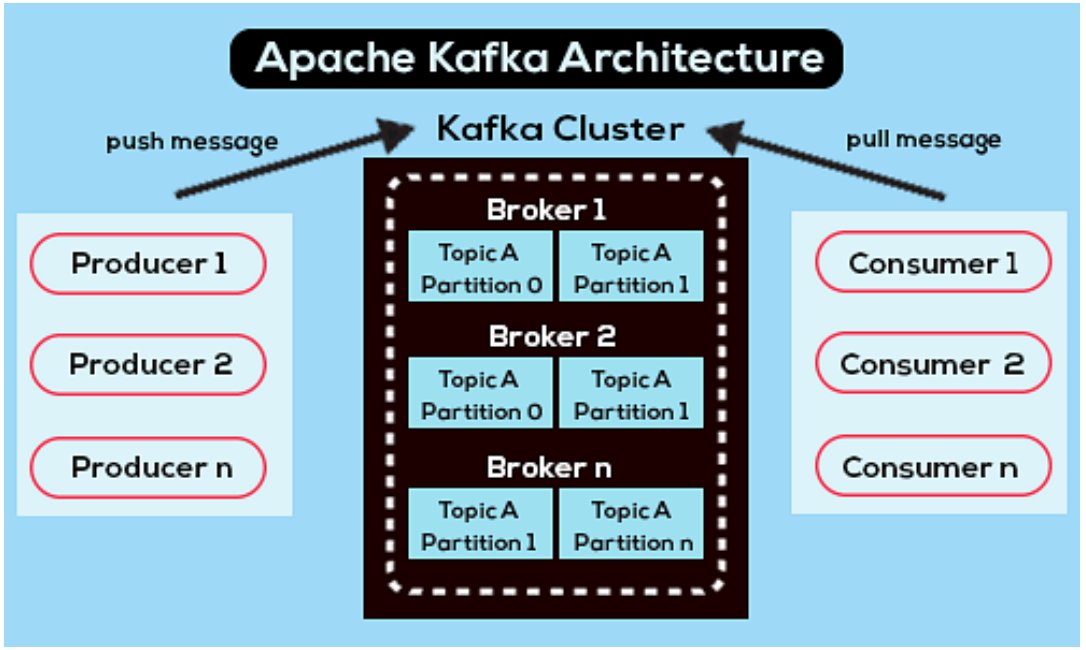


Consumer

- **earliest** Automatically reset the offset to the earliest offset
- **latest** Automatically reset the offset to the latest offset
- **none**

Kafka Architecture





Tunning:Disk

(avg-msg-size) x (msgs-per-day) x (retention-period-days) x (replication-factor)

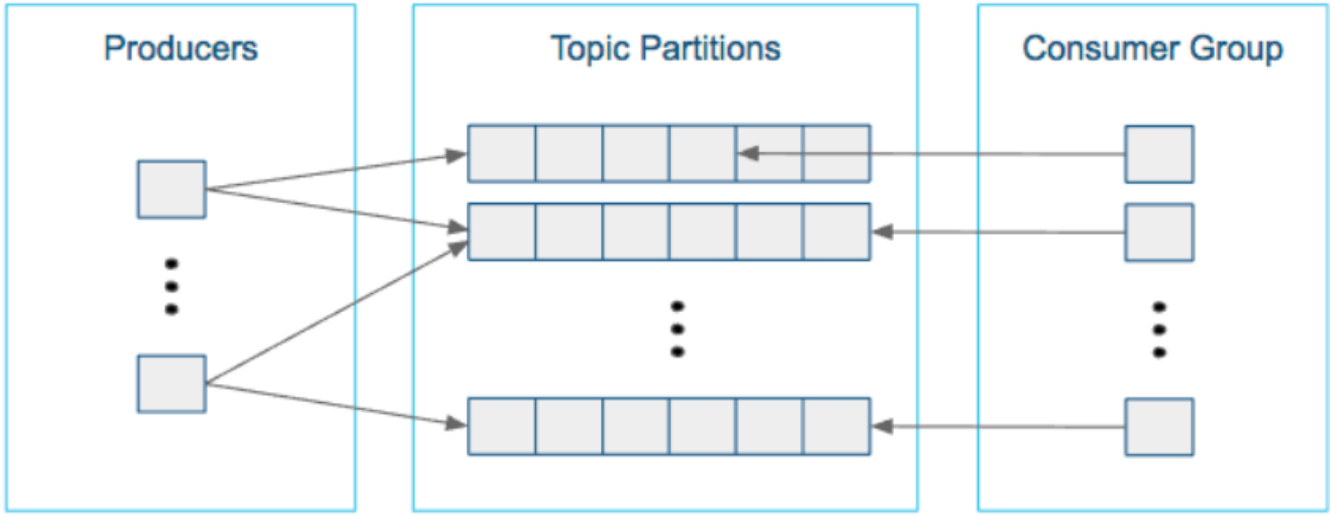
For example, let's use these numbers as an example:

- Average message size is 10kb
- Messages per day is 1,000,000
- Retention period is 5 days
- Replication factor is 3

Using our disk space utilization formula:

```
10 x 1000000 x 5 x 3 = 150,000,000 kb = 146484 MB = 143 GB
```

Tunning :Choosing the number of partitions for a topic(avoid hot spots)?




```
#Partitions = max(NP, NC)
```

where:

- N_P is the number of required producers determined by calculating: T_T/T_P
- N_C is the number of required consumers determined by calculating: T_T/T_C
- T_T is the total expected throughput for our system
- T_P is the max throughput of a single producer to a single partition
- T_C is the max throughput of a single consumer from a single partition

Exo1

https://github.com/idiattara/Spark_DIATTARA/blob/main/KafkaProducer.py

https://github.com/idiattara/Spark_DIATTARA/blob/main/KafkaConsume.py