INTRODUCTION AUX SYSTÈMES DISTRIBUÉS

INF36307 – SYSTÈMES DISTRIBUÉS

AGENDA – COURS 4 – SYSTÈMES DE QUEUE

1

Near Realtime

L'anti-pod des batch job

2

Distribution de charge

Mise à l'échelle horizontal

3

Stateless

Faciliter la gestion des queues multiples

4

Consommateurs multiples

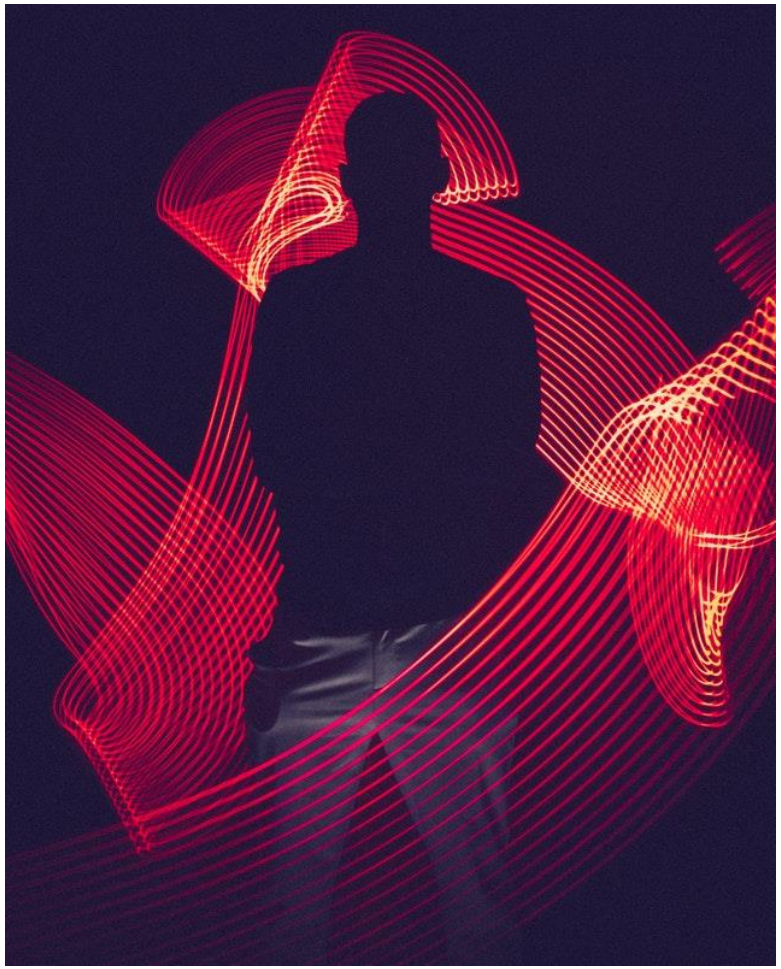
Plusieurs groupes peuvent consommer la même donnée

COMMENT C'ÉTAIT AVANT

- **Contexte :** Grosse compagnie de télécommunication
- Toute les transactions de la journée sont accumulées dans des bases de données sans être traitées
- La nuit de lourdes et complexes batch jobs traitent toutes les informations de la dernière journée
  - Ex: Commande client, facturation
  - Technologie: *Statistical Analysis System* (SAS)
- Héritage d'un passé ou les ordinateurs était le goulot d'étranglement et ou l'on devait attendre la nuit

La facturation chez Telus

AUJOURD'HUI – NEAR REAL TIME



- Aujourd'hui un client s'attend à commander un service et commencer à l'utiliser la journée même:
  - Ex:
    - Abonnement Tou.tv
    - Changement de chaine dans un forfait TV
- Les entreprises qui attendent la nuit pour faire leur traitement sont dépassées
- Les systèmes de queue ont remplacé le traitement batch

La sélétion des chaines TV chez Cogéco

RÉEL VS. QUASI RÉEL

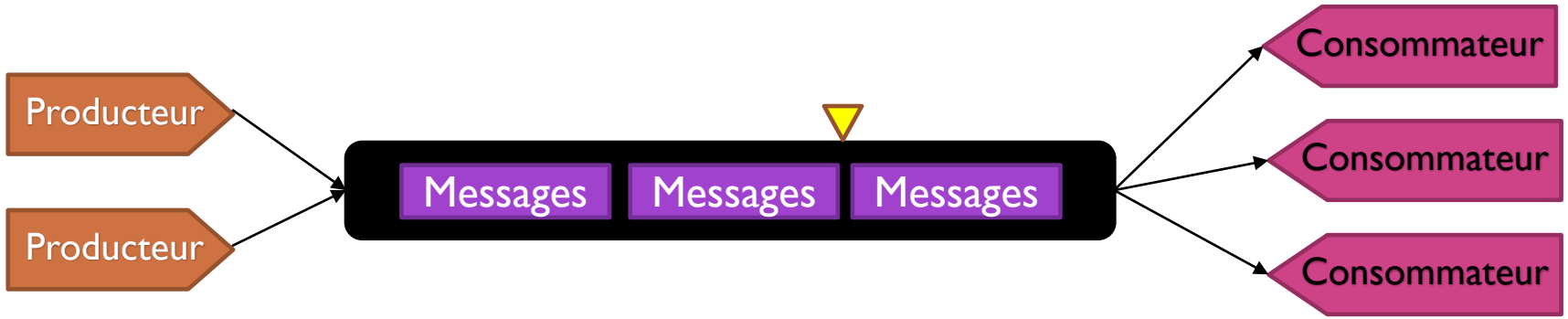
Temps réel

- Le demandeur attends confirmation de complétion d'une tâche avant de poursuivre
- Requête bloquante
  - Traitement **synchrone**

Temps quasi réel

- Le demandeur attend confirmation que la tâche a été comprise avant de poursuivre
- Requête non-bloquante en attendant la completion
- La tâche sera éventuellement complète
  - Traitement **asynchrone**

QU'EST-CE QU'UN SYSTÈME DE QUEUE?



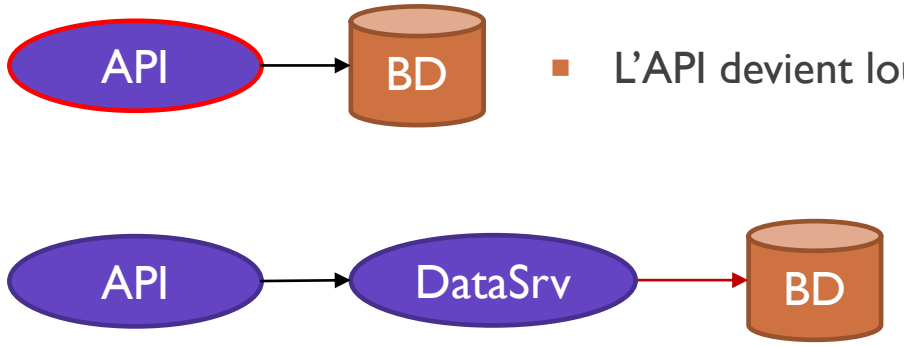
- Un accumulateur de données à être consommées plus tard
- **Producteur:** une application qui écrit dans la queue
- **Consommateur** : une application qui lit dans la queue
- **Curseur**: position de lecture dans la queue d'un **consommateur**
  - Propre à chaque **consommateur**

PROGRESSION

- **Temps quasi réel**
- **Distribution de la charge**
- **Stateless**
- **Groupe de consommation**

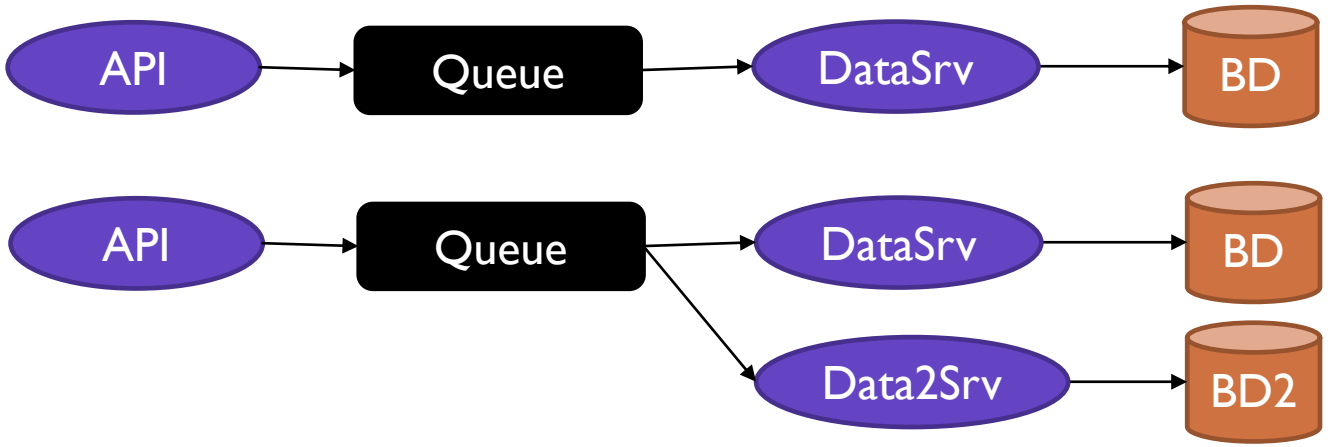
UTILISATION DE QUEUE – CAS TYPIQUE

Temps réel



- L'API devient lourd et on décide de sortir la logique d'affaire
- La BD ne fourni pas et on perd des informations

Temps quasi réel

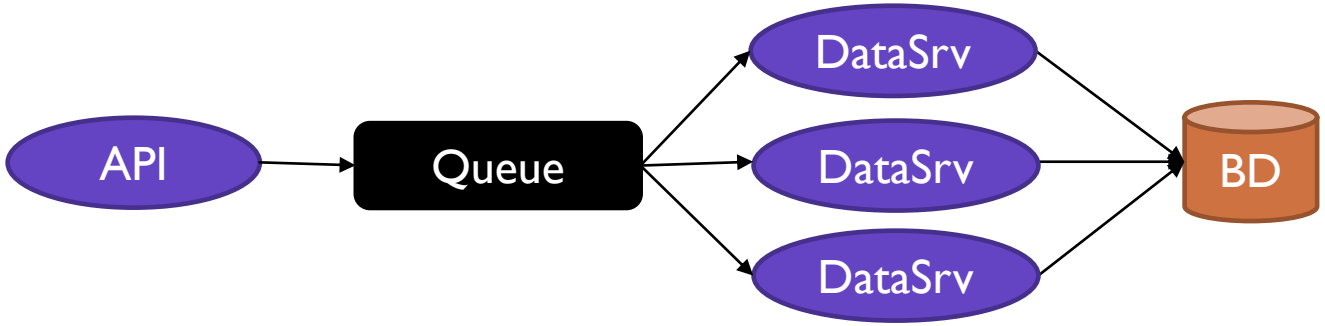


- Réessaye en cas d'erreur
- La donnée est stockée dans différent stockage

GoToNetworkTest results ingestor

ÉCART AVEC LE TEMPS REEL - LAG

- **Lag** : écart entre le temps réel et le curseur d'un **consommateur**
- Dans un système temps réel on vise à minimiser le **lag**
- Lorsque le **lag** augmente, une réponse possible est la mise à l'échelle horizontale



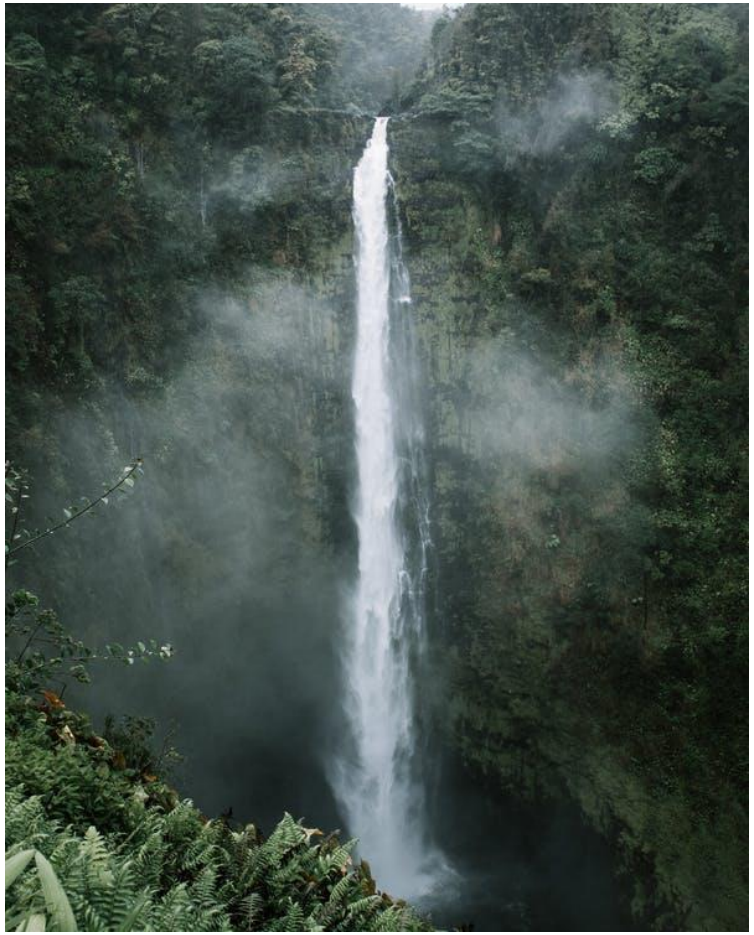
Alerte lorsque le lag > 2 minutes

PROGRESSION

- Temps quasi réel
- Distribution de la charge
- **Stateless**
- Groupe de consommation

STATELESS

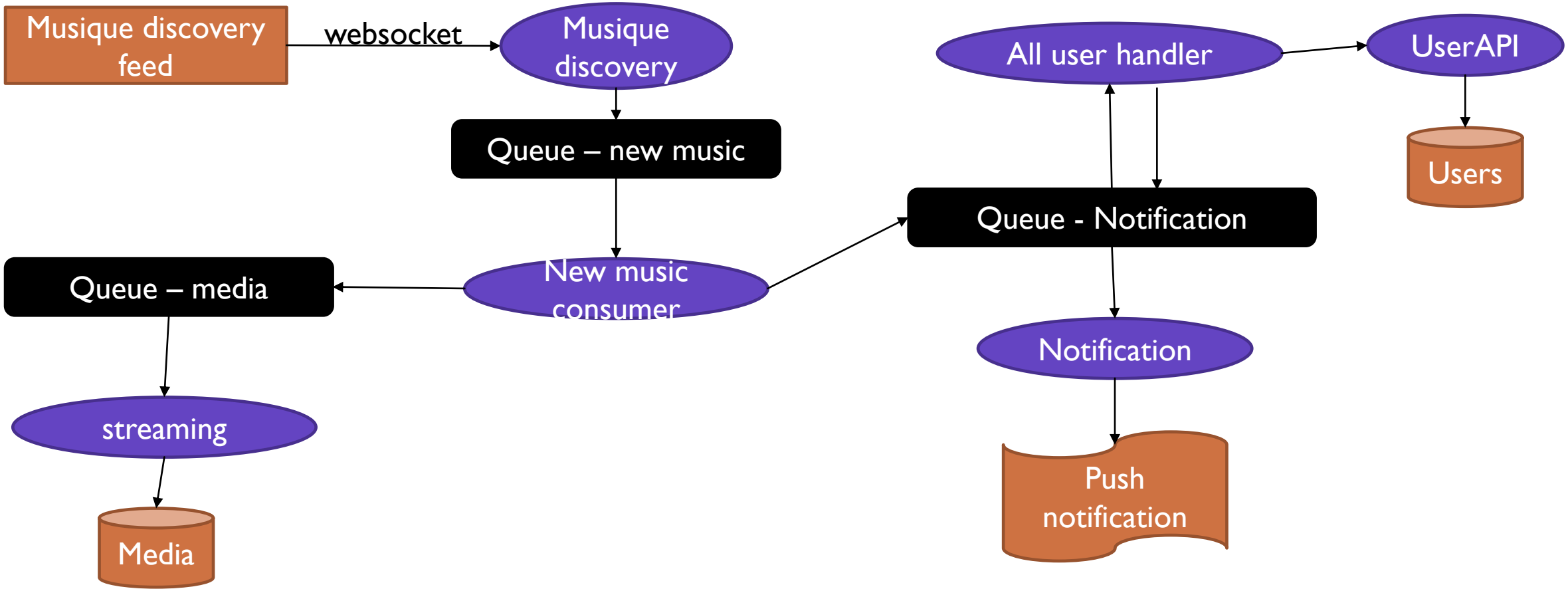
- Il est beaucoup plus simple de gérer des queues de messages qui n'ont pas d'état
- La dépendance sur une machine à états centralisée créer plusieurs problèmes
  - Accès à un stockage centralisé pour le traitement des messages est souvent lent
  - Certains messages peuvent être bloqués dans la queue en attendant un changement d'état qui ne viendra jamais
- La cible est de tenter de découper votre processus en chaines de queues qui s'imbriquent l'une dans l'autre séquentiellement ou parallèle sans avoir besoin d'état



Prendre un système existent et le découper en plusieurs queues. Dessiner l'architecture avant et après. Ex: inscription UQAR

Platform event stateful : stockage en cluster redis + End-2-End call aggregation

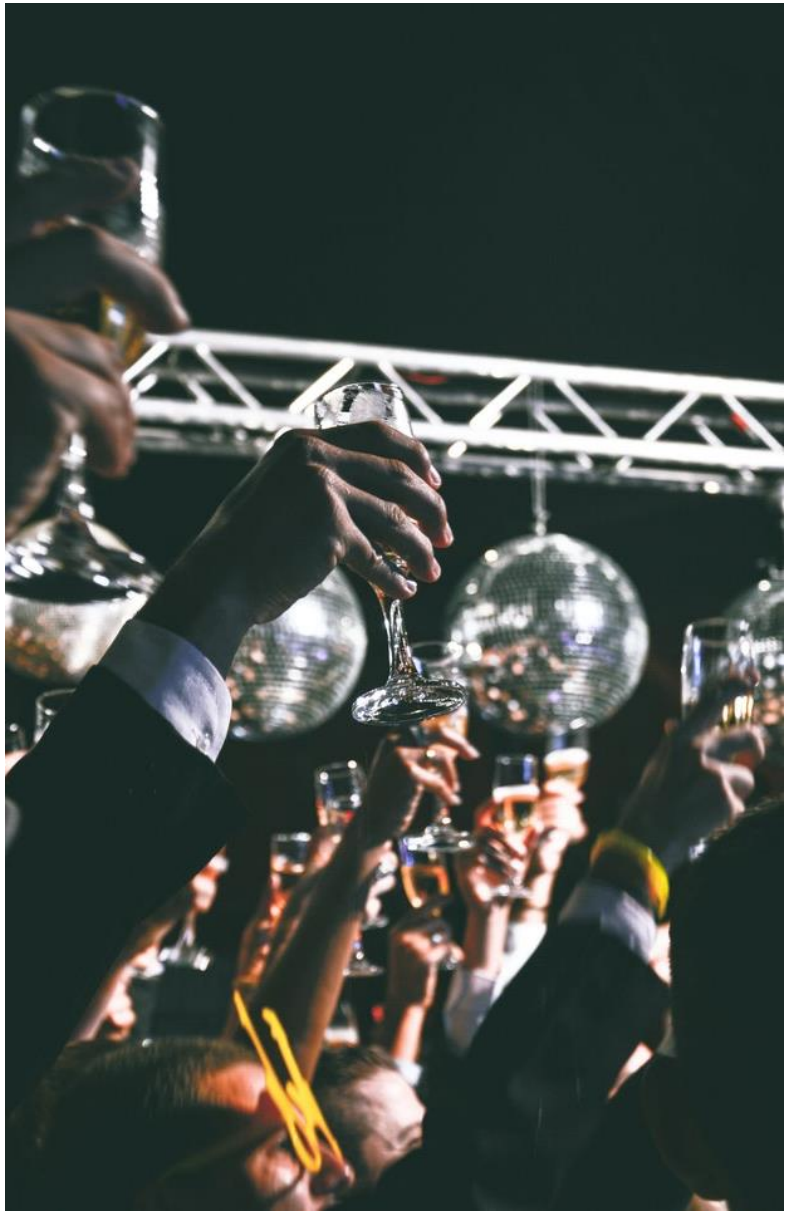
SPOTIFY – RENDRE LA MUSIQUE DU MONDE DISPONIBLE AUX USAGERS



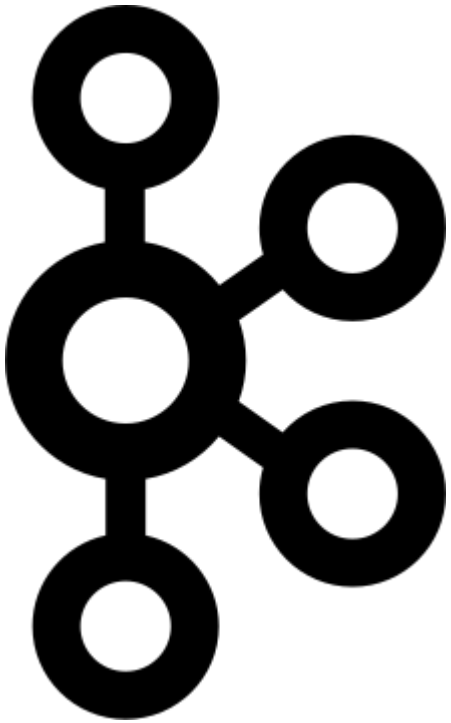
QUEUE COMME SYSTÈME D'ÉVÈNEMENTS

- Contexte
  - Un web service doit maintenir une cache de tous les usagers en mémoire
  - Le service vérifie dans la base de donnée à toutes les 5 minutes pour détecter un changement dans la liste d'usager
  - Plein d'équipes dans l'entreprise ont des besoins similaires et la base de donnée devient surcharger par des requêtes de vérification de changements dans la liste des usagers
- Une queue peut être utilisée pour produire des évènements de changements dans la liste des usagers
  - La cache n'a qu'a écouter les évènements et adapté ça cache
  - Quand même un besoin de chargement initiale (warmup) de la cache

Gestion des évênements GDPR

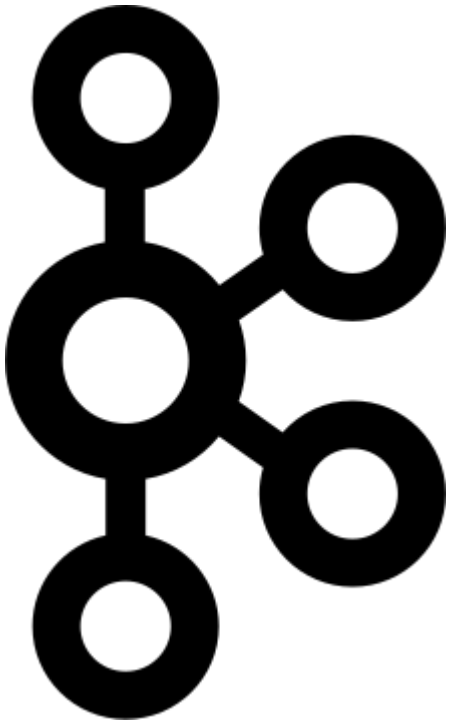


KAFKA - INTRODUCTION



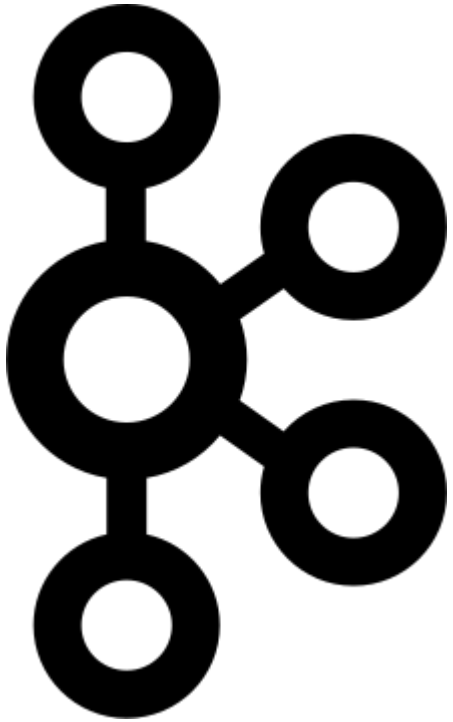
- Projet Open Source depuis 2011
- Gestion d'information par flux de message
- Développé par LinkedIn
- Facilite le développement d'application temps réel

KAFKA – INTRODUCTION – VIDÉO



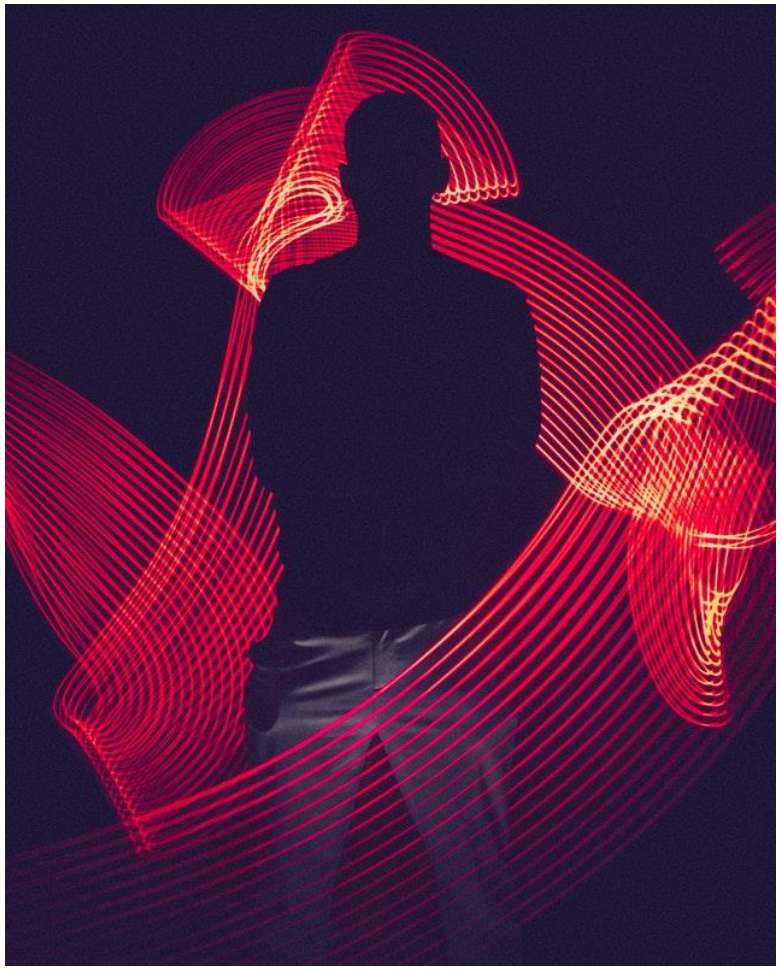
Source : https://kafka.apache.org/intro

KAFKA



- **Topic:** Une queue
- **Consumer group**
  - Un ensemble de un ou plusieurs consommateurs avec un nom
  - Un **curseur** unique dans un topic
- **Partition**
  - Découpe un topic en ensembles distincts (Sharding dans le monde de Base de Données)
  - Clef utilisée pour les partitions
  - Nombre de partitions vs. mise à l'échelle horizontale
- Stockage sur disque
  - Durée de rétention
  - Taille maximale
- **Consumer**
  - Membre d'un **consumer group**
  - Consomme une ou plusieurs **partitions** d'un **topic**

KAFKA DANS KUBERNETES



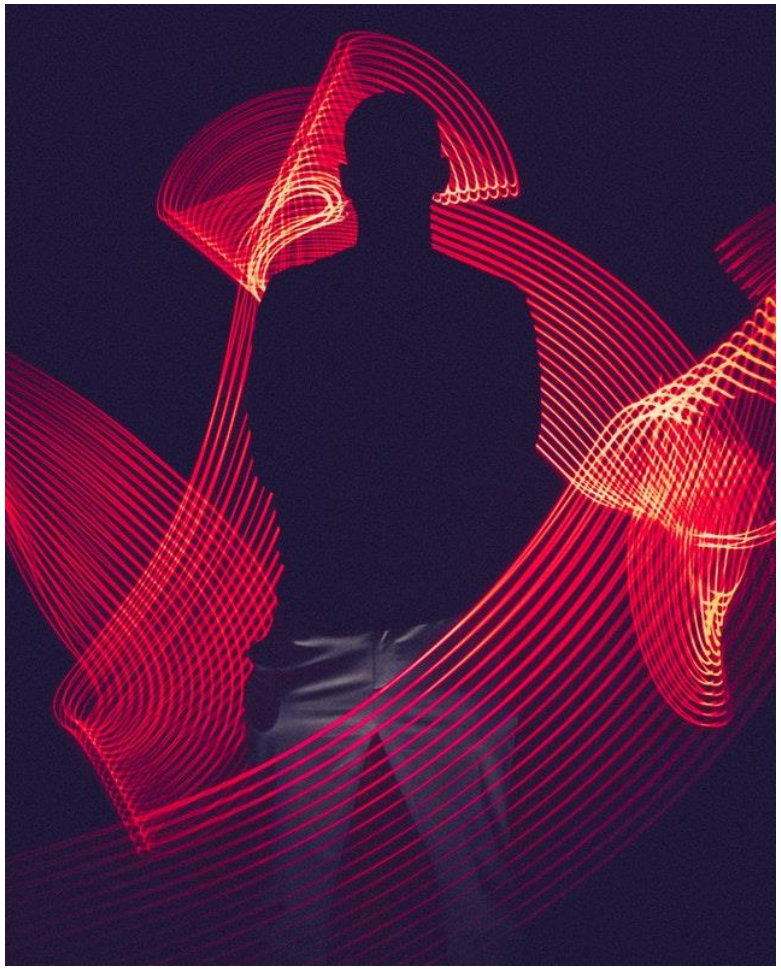
- Il y a plusieurs composants dans kafka et l'objectif du cours n'est pas d'apprendre les rouages interne de kafka, mais plutôt sont utilisation

Deployer kafka dans minikube

- Ajouter kafka-manager
- Créer un topic
- Lire et écrire dans un topic via la ligne de commande

docker-compose pour local testing

KAFKA UTILISATION CONCRÈTE



- Quel sont les avantage si l'on met une queue kafka entre votre service REST et la Base de donnée MySQL?

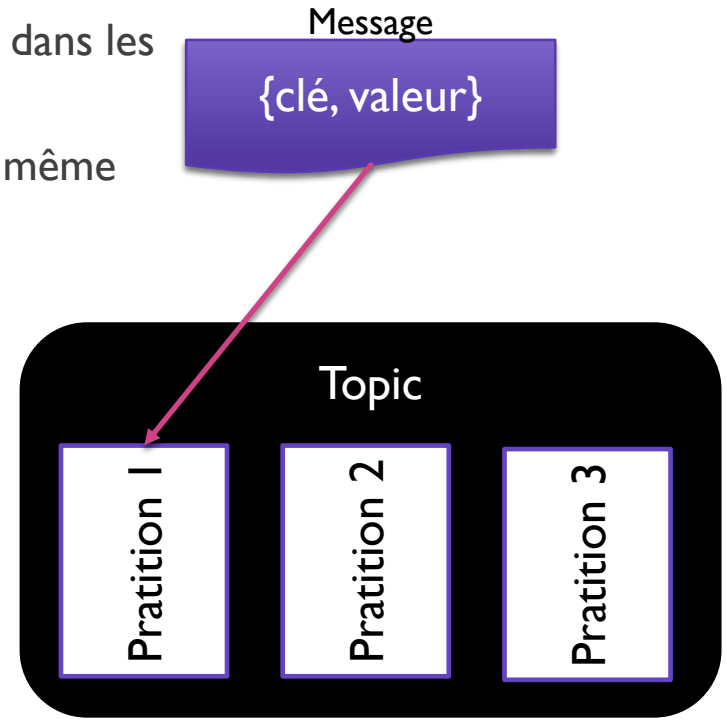
Implémenter la possibilité d'ajouter un étudiants

- Modifier le service REST existent
- Utiliser une queue kafka

- Que se passe-t-il si la Base de Donnée est temporairement inaccessible?

PARTITION ET CLÉ KAFKA

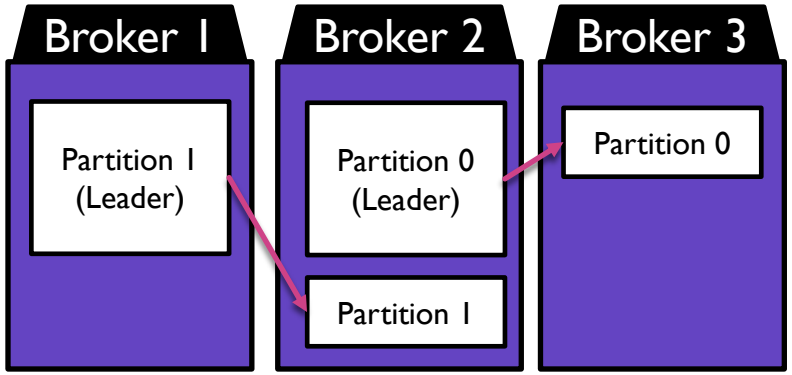
- Lorsqu'un **message** est produit dans kafka une **clé** est associée à chaque **message**
- La clé peut être fournie par le **producteur** ou générée automatiquement
- Les clés générées automatiquement vise une répartition égale des **messages** dans les **partitions** via round-robin
- Si plusieurs **messages** utilisent la même **clé**, ils seront tous contenus dans la même **partition**
- Utilité:
  - Lire des **messages** dans l'ordre
  - Lire des **messages** par un seul serveur
- Problème:
  - Débalancement des **partitions**
    - Surcharge d'un serveur, pendant de les autres serveurs ne font rien



system_test_code comme clé kafka

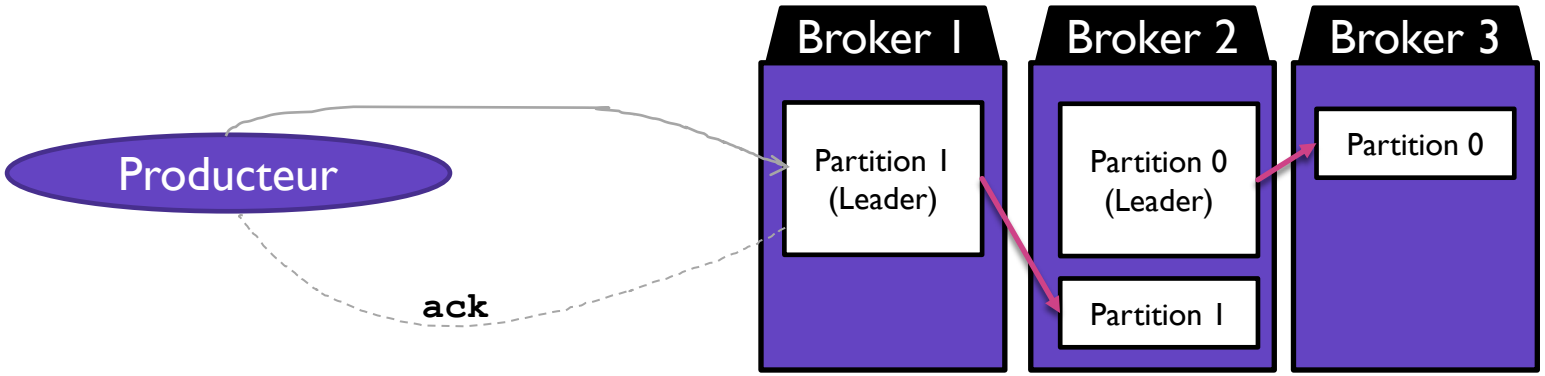
FACTEUR DE REPLICATION

- Lors de la création d'un **topic** le paramètre **replaction factor** doit être spécifié
- Ce paramètre détermine de nombre de copie des **messages** écrit dans le **topic**
- Ceci permet d'avoir un système résilient au pannes
- Un **leader** parmi les **broker** qui hébergent le **topic** est élu et sauvegarder dans **zookeeper**
- C'est le **leader** qui répond à toute les requêtes, les autres **broker** conservent une copie à jours de tous les **messages**
- Lorsque un **leader** ne répond plus, un nouveau **leader** est élu



ÉCRITURE ET CONFIRMATION

- Lorsqu'un producteur publie un message dans un topic, il dispose de 3 options de réponse:
  - Acknowledge=0 : L'écriture est faite sans attente de confirmation
  - Acknowledge=leader: L'écriture est complétée lorsque le leader confirme que l'information est sauvegardée
  - Acknowledge=all: L'écriture est complétée lorsque le leader et toutes les réplicas confirment que l'information est sauvegardée



DEVOIR

LIRE SUR LES SYSTEMS DE GESTION DE CACHE