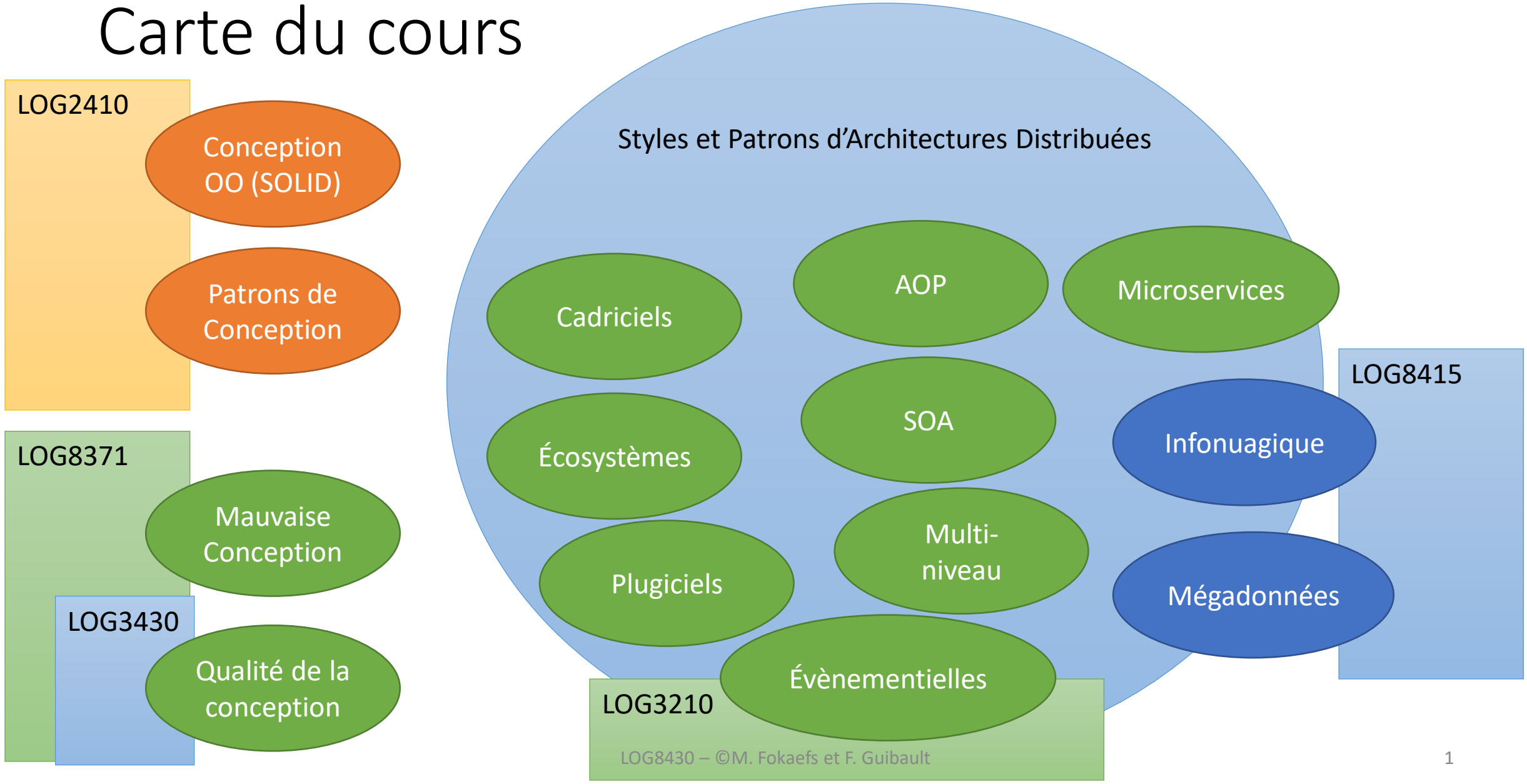
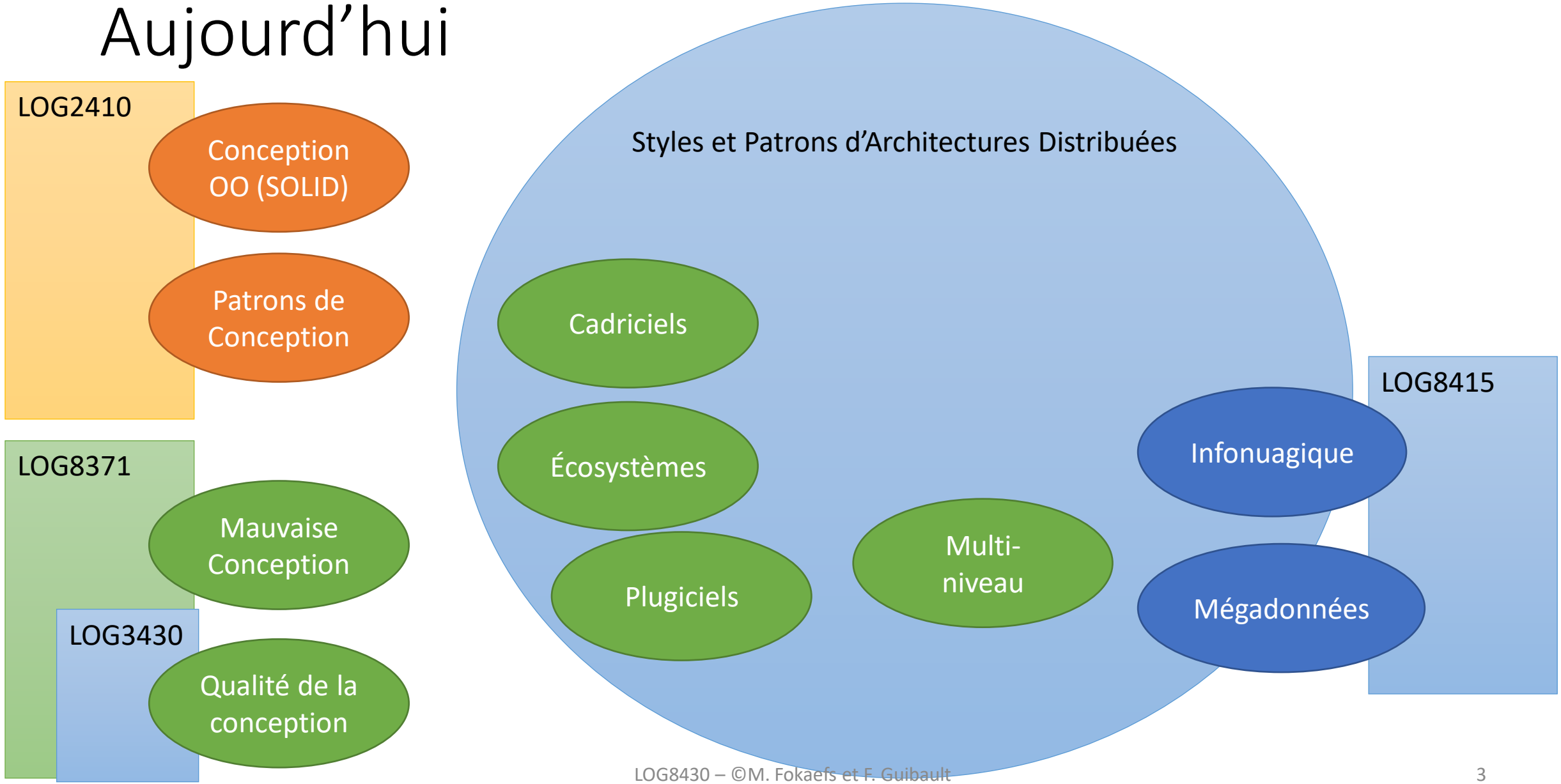
Carte du cours

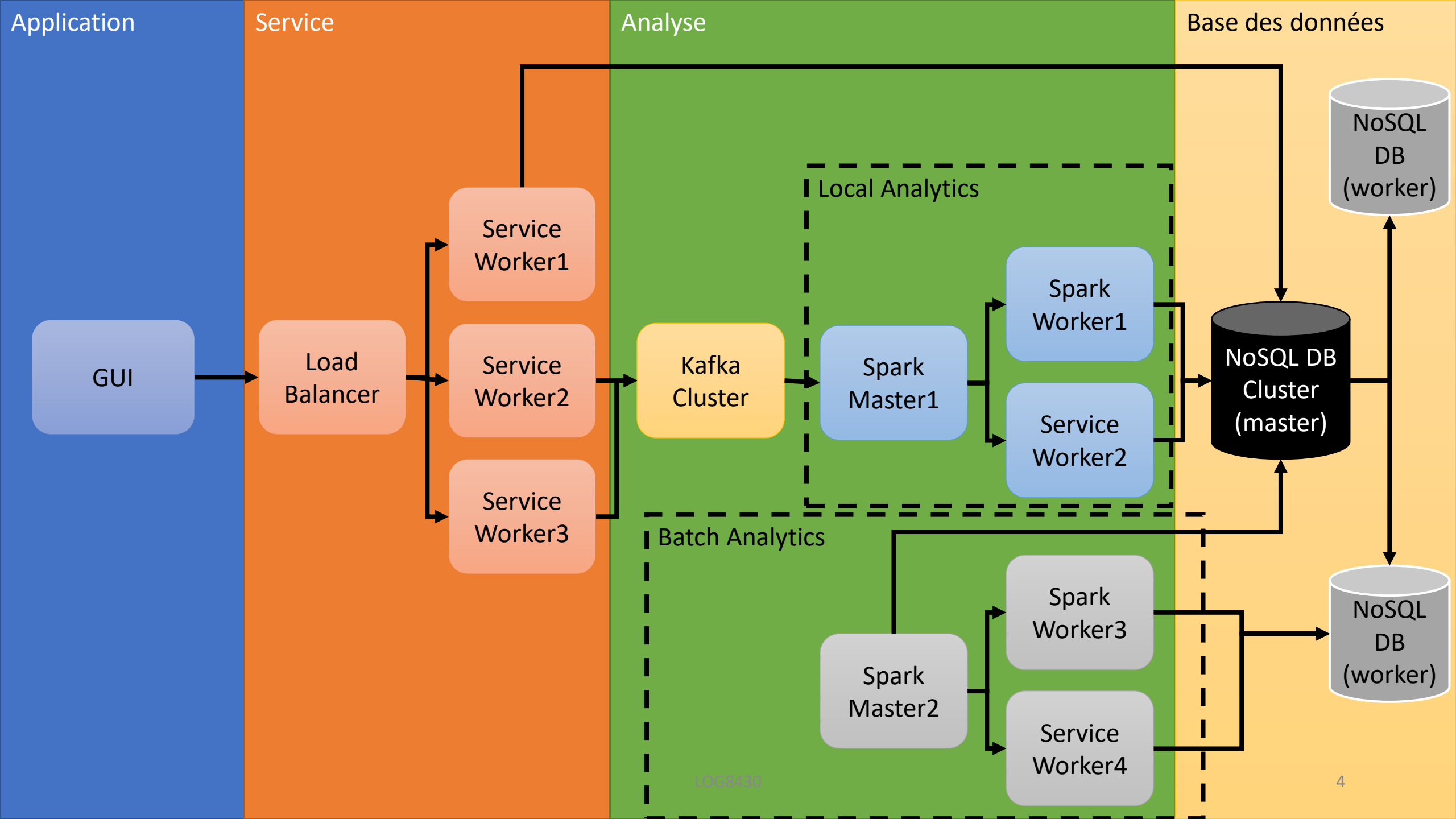


LOG8430 : Architectures des Megadonnées

Entrée et analyse de données

Aujourd'hui


LOG2410
Conception OO (SOLID)
Patrons de Conception
LOG8371
Mauvaise Conception
LOG3430
Qualité de la conception
Styles et Patrons d'Architectures Distribuées
Cadriciels
Écosystèmes
Plugiciels
Multi-niveau
Infonuagique
Mégadonnées
LOG8415


Application
Service
Analyse
Base des données
GUI
Load Balancer
Service Worker1
Service Worker2
Service Worker3
Kafka Cluster
Local Analytics
Spark Master1
Spark Worker1
Service Worker2
Batch Analytics
Spark Master2
Spark Worker3
Service Worker4
NoSQL DB Cluster (master)
NoSQL DB (worker)
NoSQL DB (worker)

Analyses et données dans la nouvelle ère

Accès aux données

- Services
- Encapsulation
- Gestion de l'entrée des données
- Intergiciel orienté messages

Analyses de données

- Algorithmes
- Architectures d'analyse des mégadonnées

Systèmes de mégadonnées

- Bases des données NoSQL

Un petit problème…

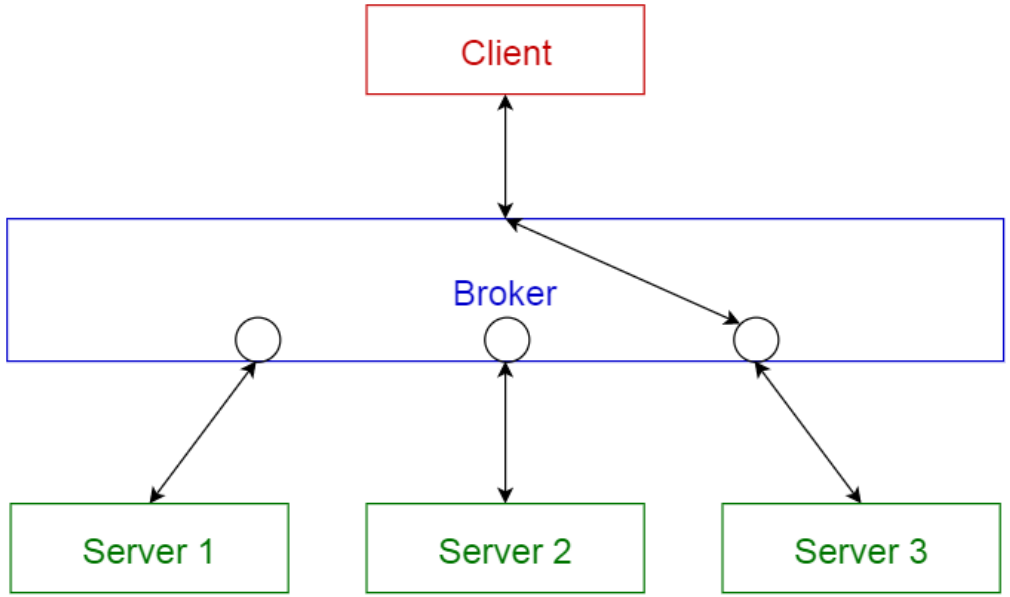
- Supposons qu'on a un bâtiment « intelligent » qui peut contrôler ses conditions environnementales (température, humidité, luminosité). Les capteurs et l'équipement viennent de différents fabricants et avec des spécifications différentes. On n'a qu'un seul système de contrôle et d'analyse, qui va accepter les données.
- Quels sont les défis d'implémentation pour ce système?

Un petit problème…

- Plusieurs sources, une seule destination.
  - Divers formats de données.
  - Interopérabilité devient difficile.
- Pour rendre les analyses efficaces, il faut avoir assez de données.
  - On ne peut pas traiter chaque élément d'information individuellement.
  - On doit attendre d'avoir une quantité suffisante de données.
- Les capteurs n'ont pas beaucoup d'intelligence.
  - Doit-on implémenter un service par capteur ou par type de capteur pour accepter les données?
  - Qui a la responsabilité de gérer l'entrée des données?
- Les services web ne sont pas une solution suffisante (pas fausse, mais insuffisante).
  - Ils requièrent une connexion directe entre le producteur et le consommateur.
  - Ils requièrent (parfois) une interaction synchrone.

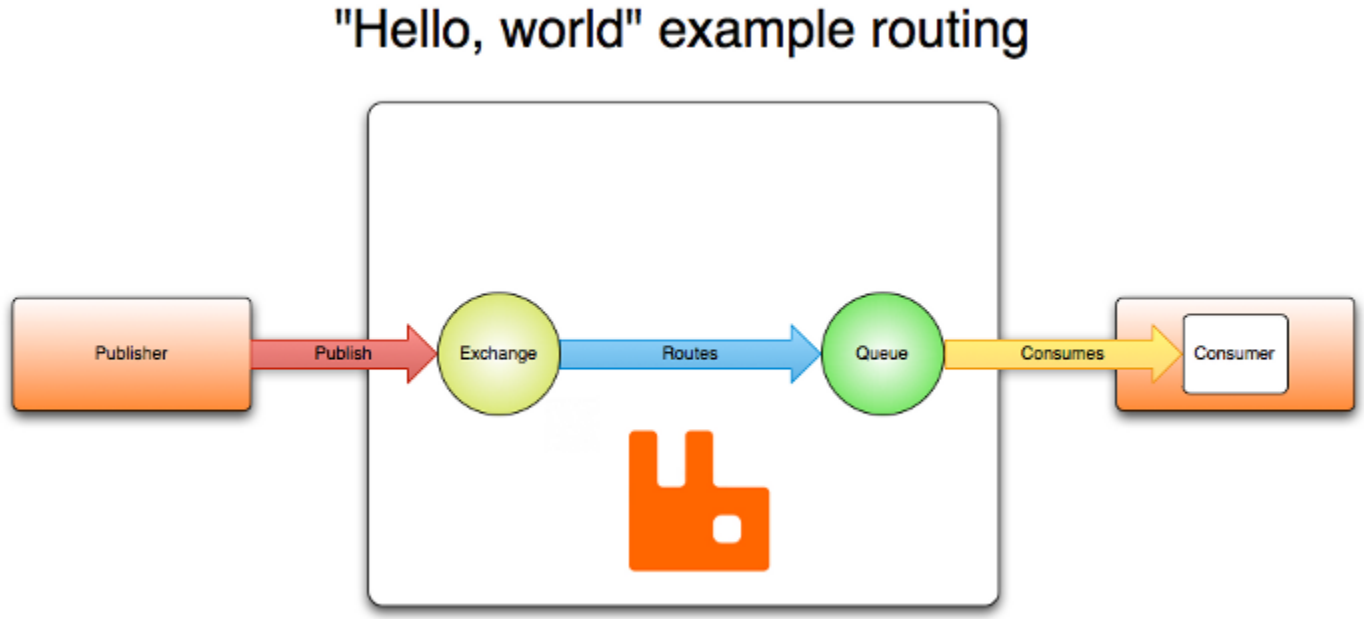
Intergiciels orientés messages

- Dans une architecture orientée message, les modules sont vraiment découplés et interopérables.
- L'entité principale de l'architecture est le message.
- Les messages correspondent aux données qui doivent être communiquées entre les modules.
- Les modules se séparent entre producteurs et consommateurs de messages.
- Le courtier est l'intergiciel.



Advanced Message Queuing Protocol (AMQP)

- AMQP est un standard ISO pour la spécification d'intergiciels orientés messages.
- AMQP définit les entités d'un système de messages et leur mode de communication.
- Les entités selon l'AMQP:
  - Courtier
  - Producteur
  - Consommateur
  - Échange
  - Queue
  - Message



AMQP : Courtier

- Le courtier implémente l'interopérabilité et il fournit et enlève plusieurs responsabilités aux modules.
- Le courtier permet aussi l'asynchronicité : le producteur soumet un message et continue son travail; le consommateur peut recevoir le message lorsqu'il est prêt.
- Le courtier a la responsabilité d'acheminer les messages entre les producteurs et les consommateurs.
- Le courtier peut aussi transformer les messages. Cela aide le traitement des données en divers formats.
- Le courtier est responsable de garantir la sécurité, la performance, la fiabilité et l'évolutivité du système.

AMQP : Producteurs et Échanges

- Les producteurs soumettent des messages au courtier et en particulier aux échanges.
- Les échanges acheminent les messages aux queues selon un algorithme :
  - **Direct** : L'échange achemine le message à une queue spécifiée par une clé.
  - « **Fanout** » : L'échange achemine le message à toutes les queues liées à l'échange. La clé est ignorée.
  - « **Thème** » : Le message est acheminé selon la clé et un paramètre supplémentaire pour permettre la multidiffusion.
  - « **En-tête** » : Un échange en-tête achemine le message à une queue spécifié par plusieurs paramètres qu'une clé.
- Outre l'algorithme, les échanges ont aussi des propriétés :
  - Nom
  - Durabilité : L'échange survit à une relance du courtier.
  - Suppression automatique : lorsque il n'y a pas de queues liées à l'échange.
  - Arguments supplémentaires : utilisés par les extensions ou les plugins.

AMQP : Consommateurs et Queues

- Les consommateurs consomment les messages en s'enregistrant sur les queues.
- Les consommateurs peuvent retirer des messages de la queue ou pousser des messages sur la queue.
- La queue fonctionne comme un stockage temporaire pour les messages.
- Les queues peuvent être durables ou pas. Une queue durable est stockée sur disque et elle survit à une relance du courtier.
- Une queue peut être exclusive à une connexion.
- Les queues sont liées aux échanges par des *liaisons*.
  - Les liaisons représentent des règles selon lesquelles les échanges acheminent les messages aux queues.

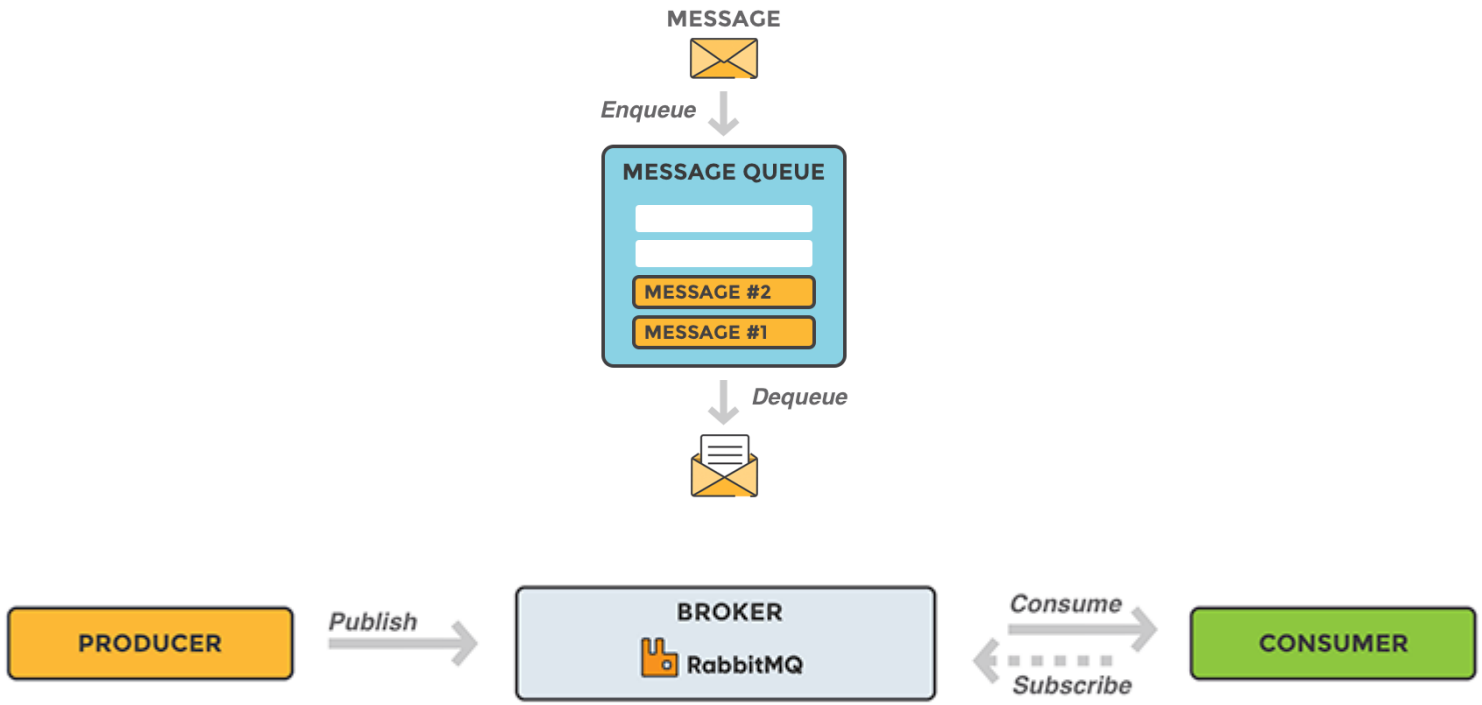
AMQP : Messages

- Les messages sont définis par leurs données et leurs attributs.
- Le contenu (les données) du message est optionnel et il est ignoré par le courtier.
- Le message peut être déclaré persistent, ce qui implique que le message est stocké sur disque et il peut survivre à une relance.
- Les attributs du message incluent:
  - Type et encodage du contenu
  - Clé d'acheminement
  - Persistent ou non
  - Priorité
  - ID du producteur
  - Horodatage et temps d'expiration du message

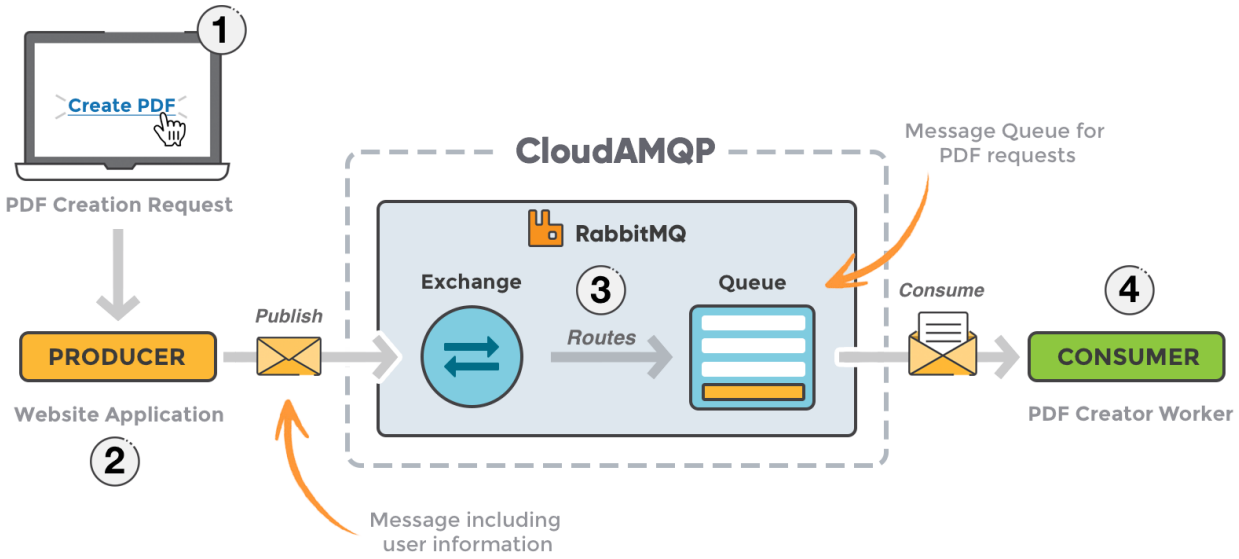
Implémentations d’AMQP

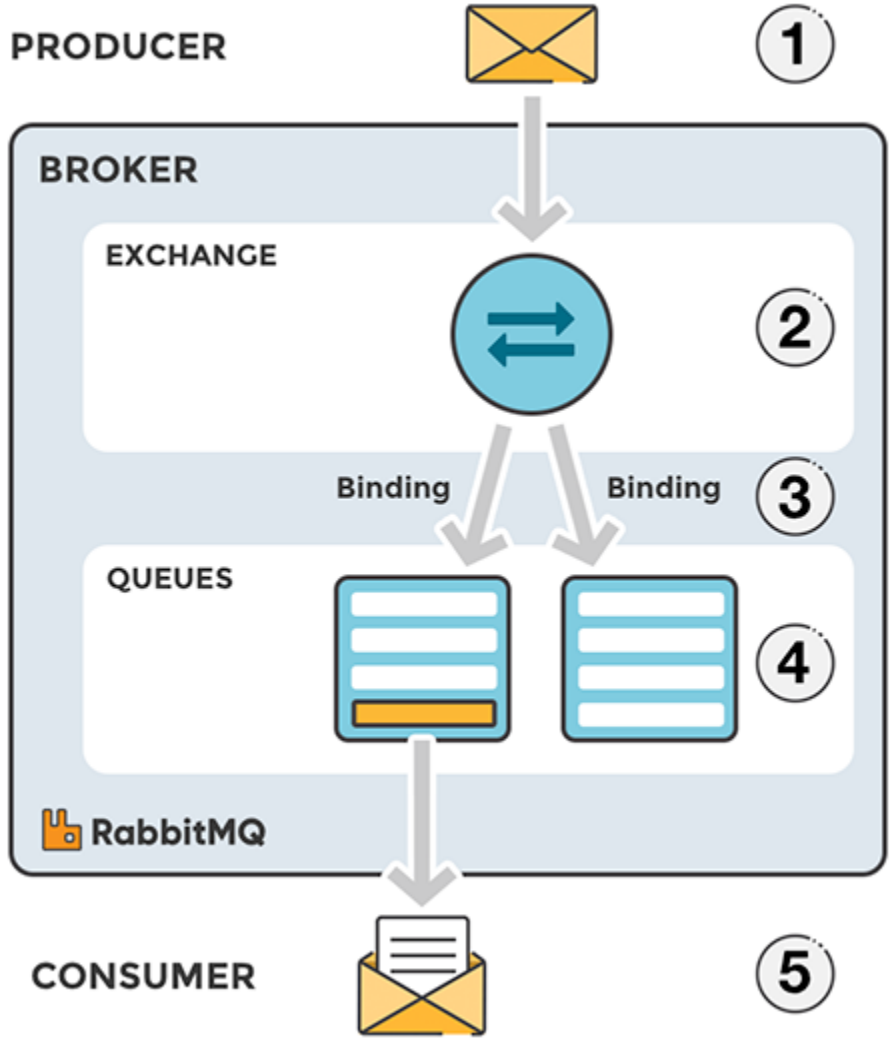
RabbitMQ

- RabbitMQ est une implémentation directe d'AMQP.
  - Tous les éléments discutés précédemment sont spécifiés dans RabbitMQ.

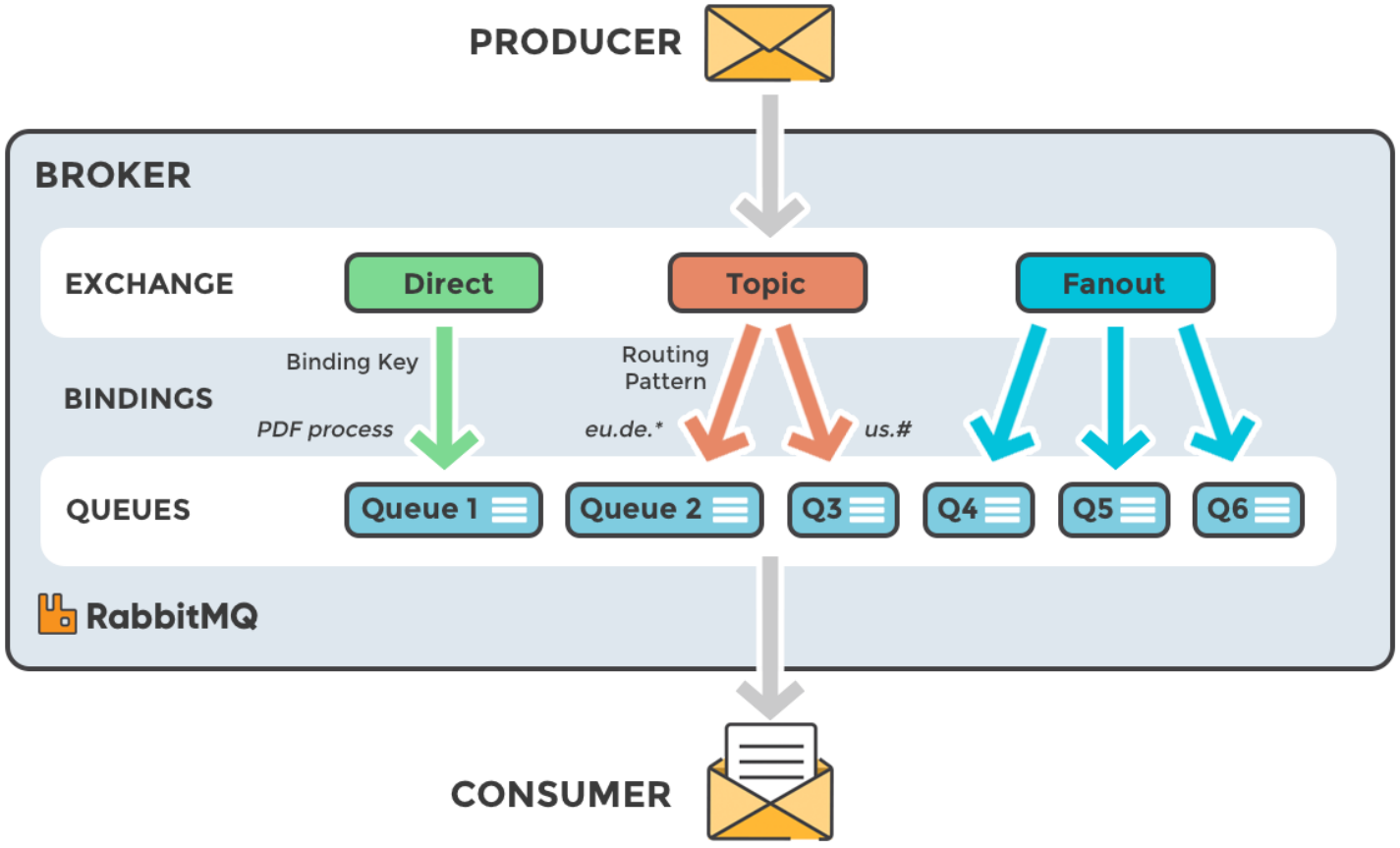


RabbitMQ : Un exemple



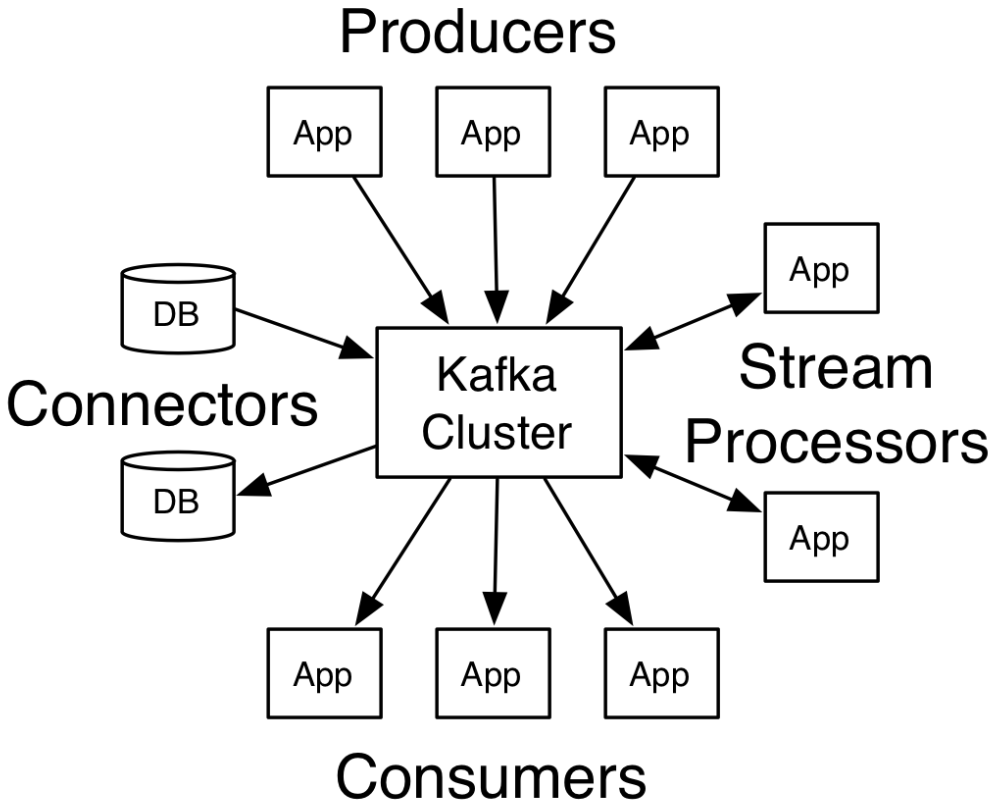


RabbitMQ : Types d'échange



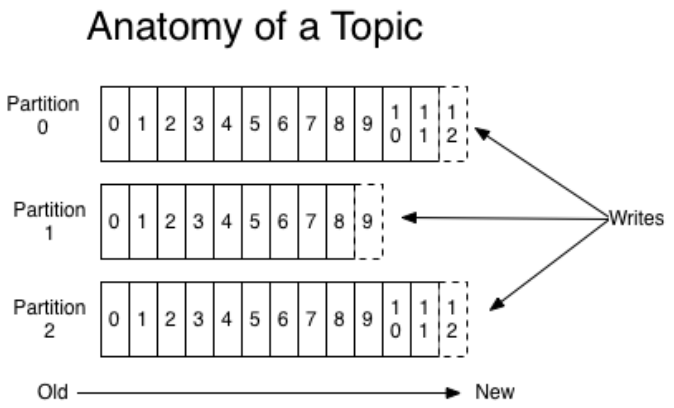
Apache Kafka

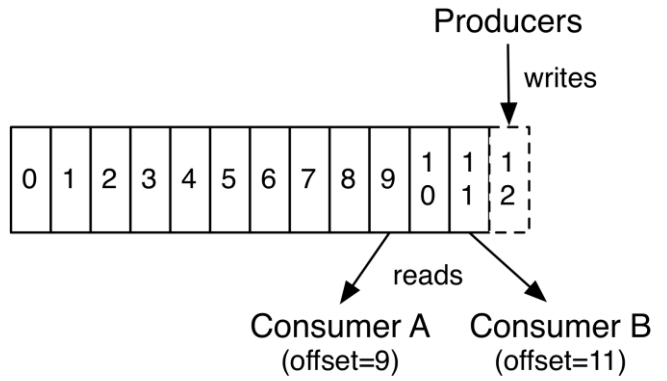
- Kafka est défini comme un service distribué de streaming.
- Les streams sont comme les queues mais avec quelques petites différences.
- Les streams permettent la diffusion des données et pas juste la multidiffusion.
- Kafka utilise la transmission selon des thèmes.
  - Les consommateurs s'enregistrent sur des thèmes et attendent les données en temps réel.
- Toutes les données sont persistantes sur disque.

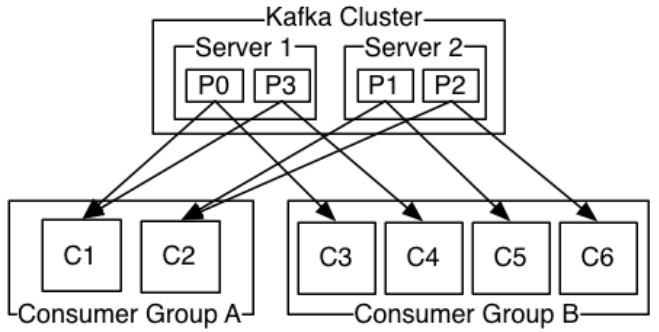


Kafka : Thèmes

- L’ordre des messages est dicté par les producteurs, mais les consommateurs peuvent contrôler l’ordre de lecture.
- Grâce aux thèmes et au groupement des consommateurs, Kafka offre les avantages du parallélisme et de la diffusion.
- Des messages peuvent être lus par plusieurs consommateurs.

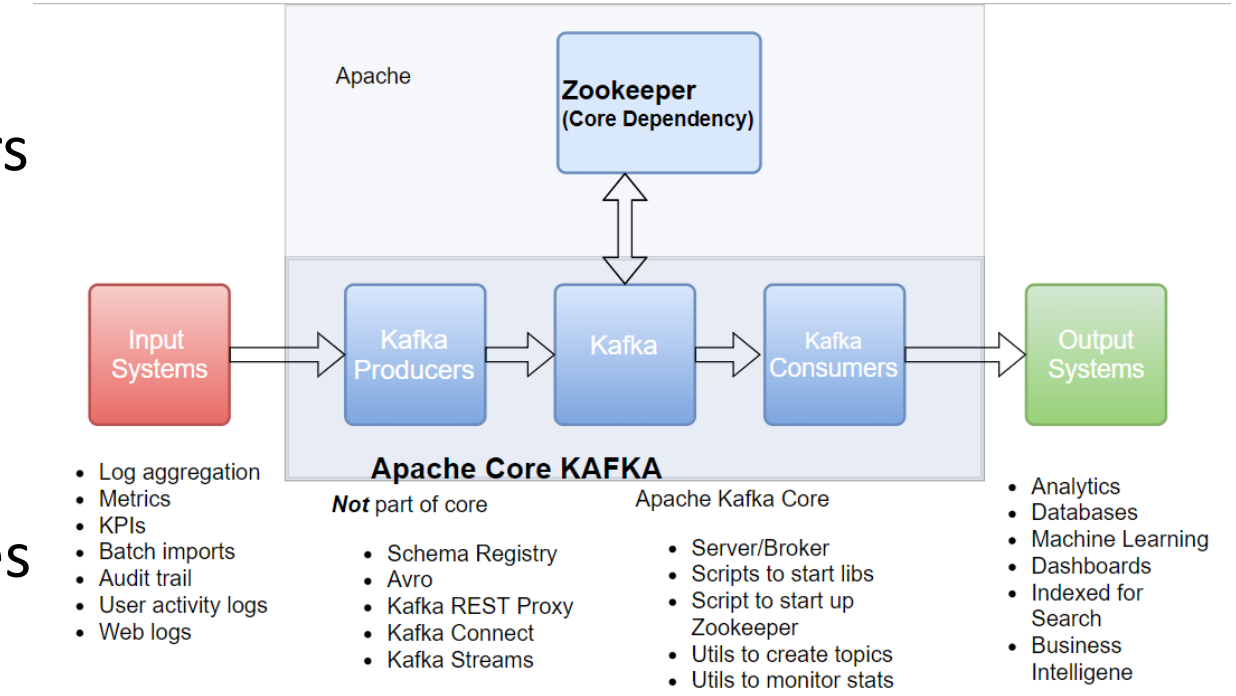






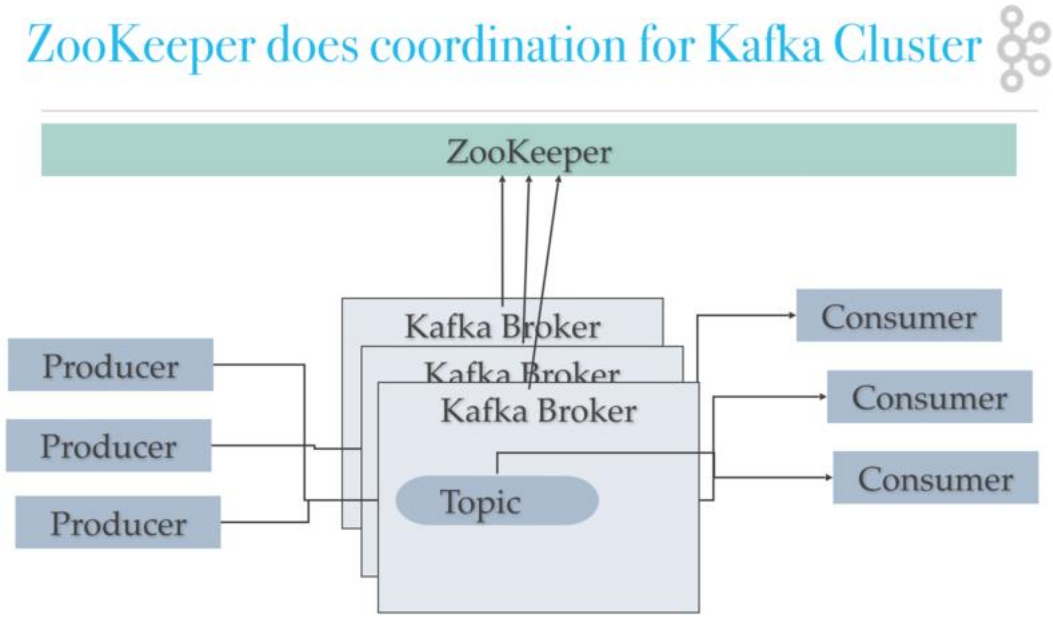
Kafka : Architecture de base

- « Input Systems » = Producteurs
- « Output Systems » = Consommateurs
- « Kafka Producers » = Échanges
- « Kafka Consumers » = Queues
- « Kafka » = Courtier
- Pour Kafka, les échanges et les queues sont l'équivalent des stubs pour les services.
  - Ce sont des classes locales qui seront appelées par les systèmes externes.



Kafka : ZooKeeper

- Kafka peut fonctionner en groupe de serveurs/courtiers.
- Cela aide la performance et la fiabilité.
  - Les messages peuvent être répliqués entre les courtiers pour augmenter la tolérance aux pannes.
- ZooKeeper est un service de découverte qui peut gérer le groupe.
  - Les clients communiquent avec le service ZooKeeper pour se connecter à un courtier.



RabbitMQ vs Kafka

RabbitMQ

- Courtier intelligent/Consommateurs simples
- Fiabilité et découplage sont les priorités.
- Plusieurs types d'échange sont supportés.
- Transmission directe ou multidiffusion.
- Sécurité est augmentée.
- La haute performance est possible mais avec des ressources significatives.

Kafka

- Courtier simple/Consommateurs intelligents
- Streaming permet l'utilisation d'une seule connexion avec un taux d'entrée augmentée.
- Seul l'échange par thème est supporté.
- Transmission par diffusion.
- Sécurité peut être un problème.
- La performance est augmentée.

Intergiciels orientés messages

Avantages

- Flexibilité
- Découplage/Interopérabilité
- Fiabilité
- Sécurité
- Évolutivité

Désavantages

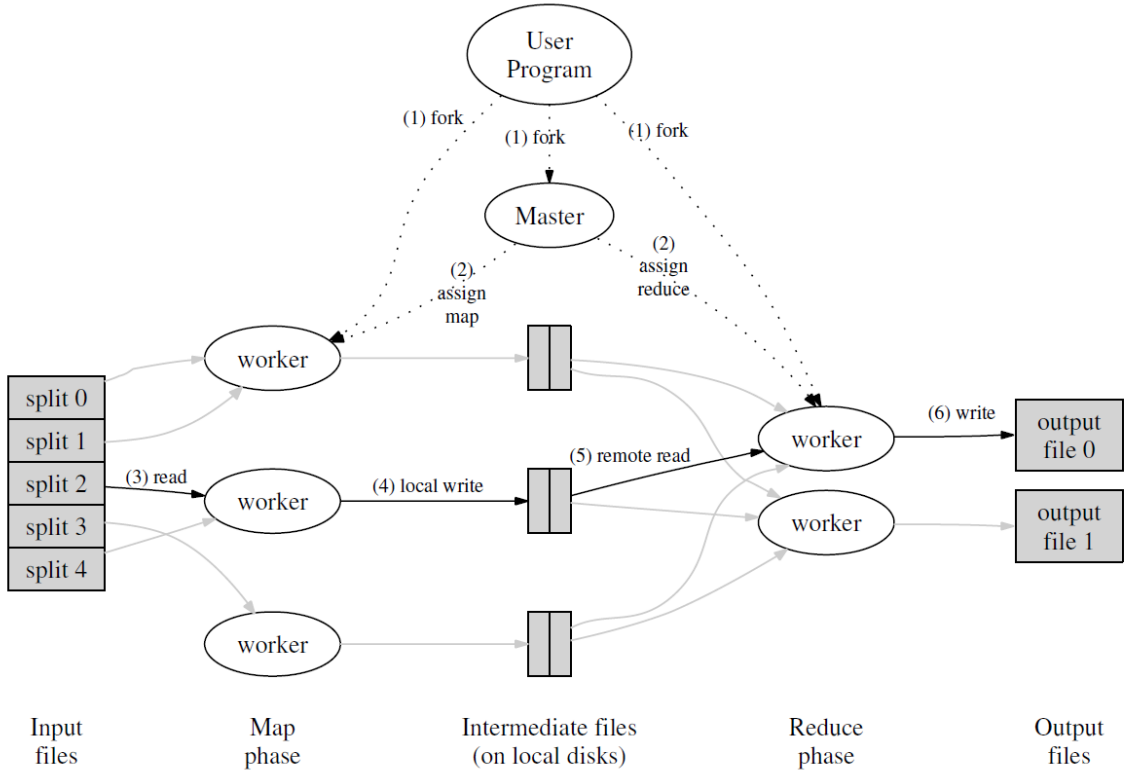
- Performance
- Complexité de l'infrastructure et de l'architecture
- Découplage n'est pas toujours désiré.

Analyse : MapReduce

- MapReduce est un modèle de programmation pour l'analyse distribuée des mégadonnées.
- Le modèle contient un algorithme, une implémentation et une infrastructure.
- Dans une implémentation MapReduce, le développeur doit définir deux tâches :
  - Une tâche *map* qui représente un travail d'analyse (p.ex., un tri, une addition, une cherche etc.)
  - Une tâche *reduce* qui représente un travail de sommation, qui va agréger les résultats des tâches *map* individuelles.
- Les tâches peuvent être exécutées sur plusieurs nœuds de traitement et elles requièrent plusieurs nœuds de stockage pour faciliter le traitement des données et pour stocker les résultats finaux si nécessaire.
  - Systèmes de données distribués qui sont traditionnellement utilisés incluent GFS (Google File System) et HDFS (Hadoop Distributed File System)

MapReduce : Processus

- Les données sont définies et organisées en pairs de clés-valeurs.
- Les données sont partitionnées automatiquement ou explicitement par le développeur.
  - Par taille, par distribution géographique ou temporelle.
- Les données sont partagées parmi les ouvriers selon les clés.
- Une copie des tâches (map et reduce) est transmise à chaque ouvrier.
- Les ouvriers exécutent les tâches *map* en utilisant des données d'entrée selon les clés et ils stockent les résultats intermédiaires.
- D'autres ouvriers agrègent les résultats map en exécutant les tâches *reduce.*
- Les résultats sont stockés dans un ou plusieurs fichiers.
- Les développeurs peuvent traiter les fichiers ou les donner à un autre tâche MapReduce.



MapReduce : Architecture

- Le modèle suit une architecture « master-slave ».
- Le master est responsable pour la distribution des tâches.
- Les ouvriers peuvent accepter des tâches *map* aussi bien que des tâches *reduce*.
- Le master est responsable pour balancer la charge entre les ouvriers.
- Le master contrôle périodiquement le bon fonctionnement des ouvriers. Cela garantit la fiabilité du système et la haute disponibilité du service.
- Le master de son côté est un point de faiblesse.
  - De nouvelles versions résolvent ce problème.
- Le système distribué des fichiers suit aussi une architecture « master-slave ».

MapReduce : Avantages et Désavantages

Avantages

- Efficacité grâce au parallélisme.
- Disponibilité.
- Tolérance aux pannes.
- Évolutivité.
- Mobilité minimale des données.
  - Le traitement des données peut se produire sur le serveur qui stocke les données.

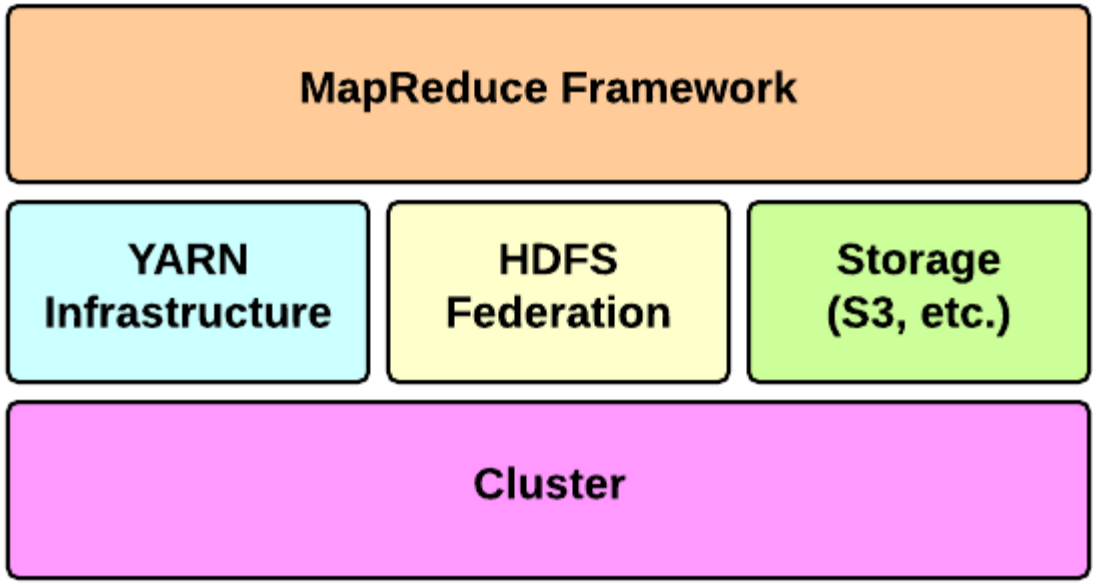
Désavantages

- « Region hotspotting »
  - L'utilisation des clés pour partitionner et partager les données parmi les ouvriers peut charger un ouvrier particulier.
  - On peut résoudre cette situation en utilisant le hachage sur les clés.
- Ce ne sont pas tous les types de traitement qui peuvent être implémentés par MapReduce.

Implémentations de MapReduce

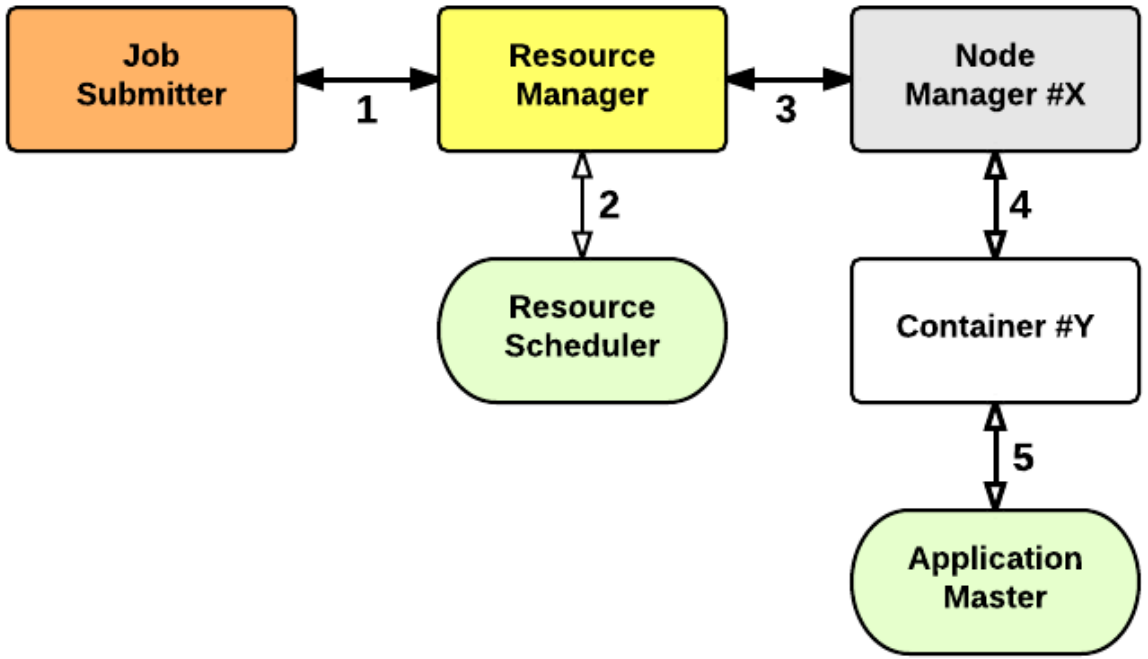
Hadoop : Architecture de Base

- Hadoop est une implémentation directe de MapReduce.
- Elle supporte les mêmes modules principaux.
- Le « Cluster » représente l'infrastructure physique (ou virtuelle) pour le traitement de données.
- YARN est le système de gestion du cluster.
- HDFS est le système de fichiers distribué.
- S3 est le stockage physique où le HDFS existe.
- MapReduce est l'implémentation du modèle.



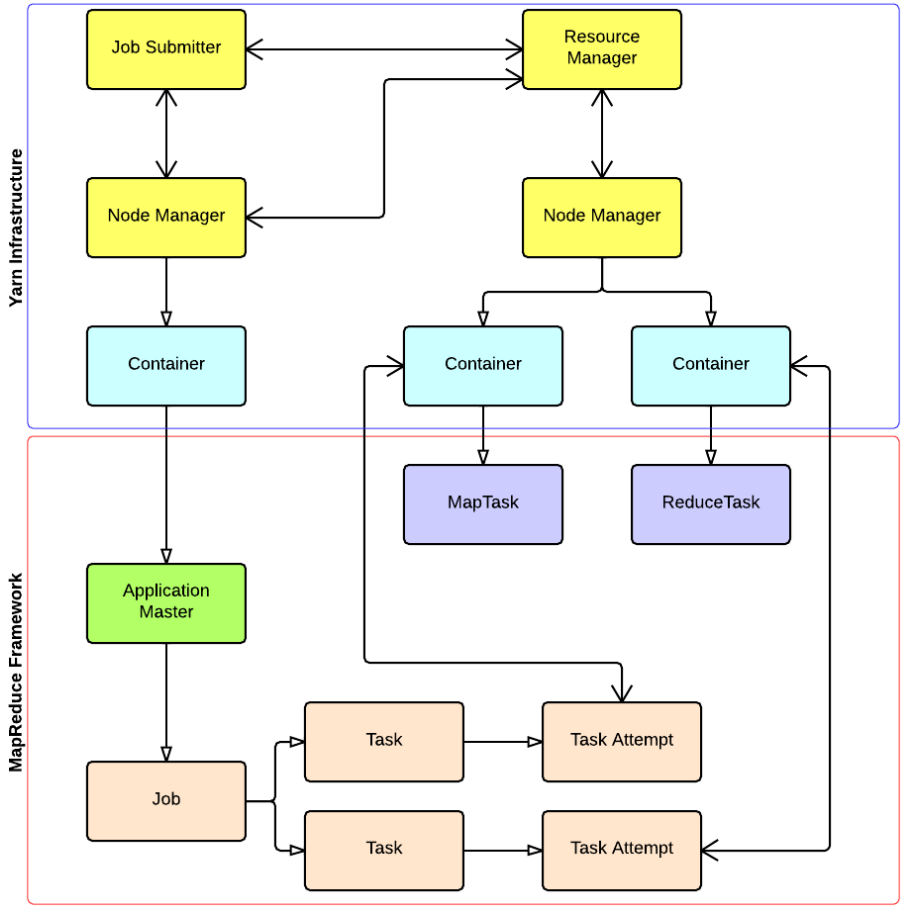
YARN : Flux de travail

- Un client soumit un travail pour une application.
- Le gestionnaire de ressources (GR) cherche les ressources disponibles.
- Le GR contacte le gestionnaire de nœuds (GN).
- Le GN lance un conteneur avec les ressources disponibles.
- Le conteneur exécute le master de l'application



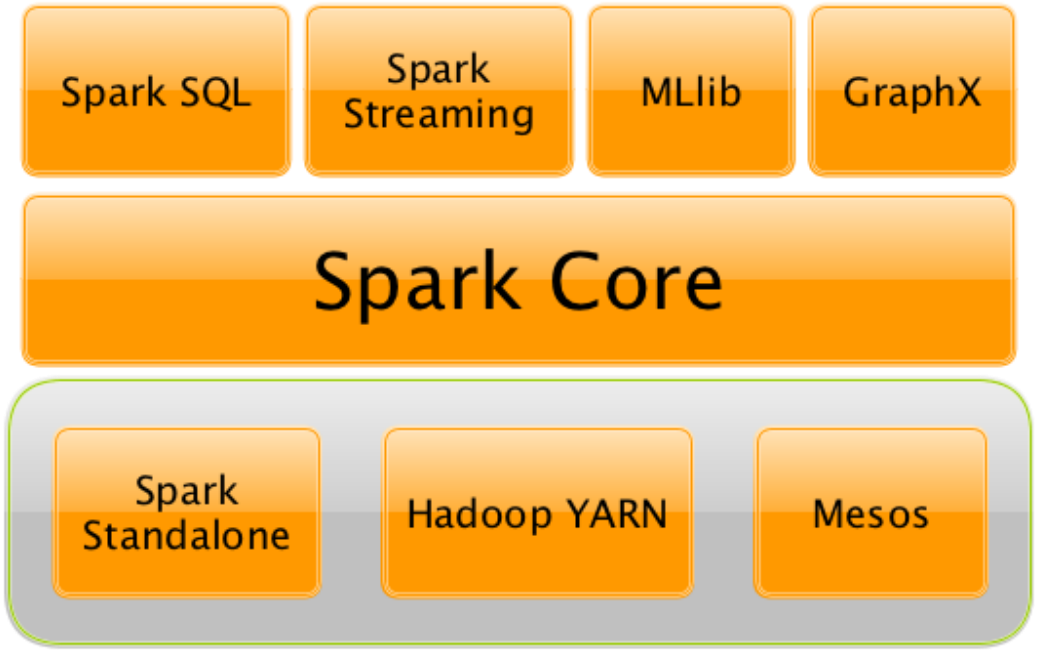
Hadoop : MapReduce Architecture

- Les conteneurs contiennent le master et les ouvriers.
- Le master de l'application lance le travail et le travail est responsable de définir les tâches (map et reduce).
- Les tâches sont déployées dans les conteneurs pour s'exécuter.
- Les ressources nécessaires sont déterminées et fournies par YARN.



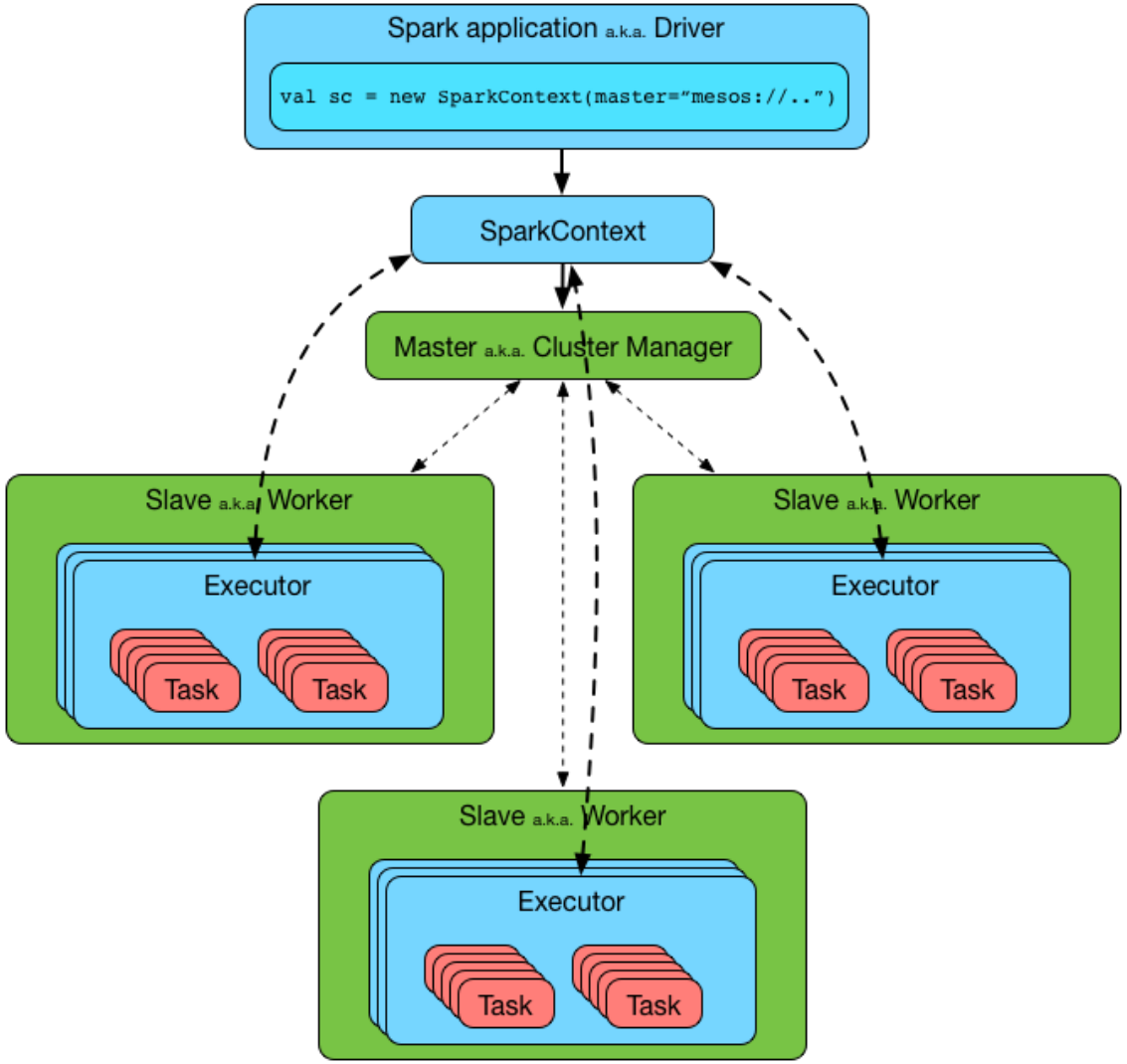
Spark : La plateforme

- Spark est une autre implémentation de MapReduce (Spark Core).
- Elle fonctionne aussi en groupe de ressources.
- Elle peut fonctionner avec plusieurs gestionnaires de ressources, mais elle fournit son propre gestionnaire aussi.
- Spark fournit aussi plusieurs capacités, outre MapReduce.
  - Spark SQL, pour les opérations avec les bases des données relationnelles.
  - Spark Streaming
  - MLlib, pour l'apprentissage automatique.
  - GraphX, pour l'analyse des graphes.



Spark : Architecture

- Une exécution de Spark est divisée en deux types de processus, un *driver* et des *executors*.
- Le *driver* est le travail/application et il définit un contexte.
- Le contexte spécifie la configuration du travail.
- Le travail définit les tâches et lance les *executors*, un pour chaque tâche.



Spark : Resilient Distributed Dataset

- Les RDDs expliquent la puissance de Spark.
- Même si Spark peut fonctionner avec un système de fichiers comme HDFS, grâce aux RDDs, Spark fonctionne en mémoire.
- Cela le rend plus rapide et plus efficace que Hadoop. (100x plus rapide)
- Mais, si les données sont extrêmement nombreuses, le système est forcé d'utiliser le disque. Toutefois, Spark est encore plus rapide. (10x plus rapide)
- Les RDDs peuvent être récupérés en cas d'un échec des serveurs.
- Cela augmente la tolérance aux pannes.

Hadoop vs Spark

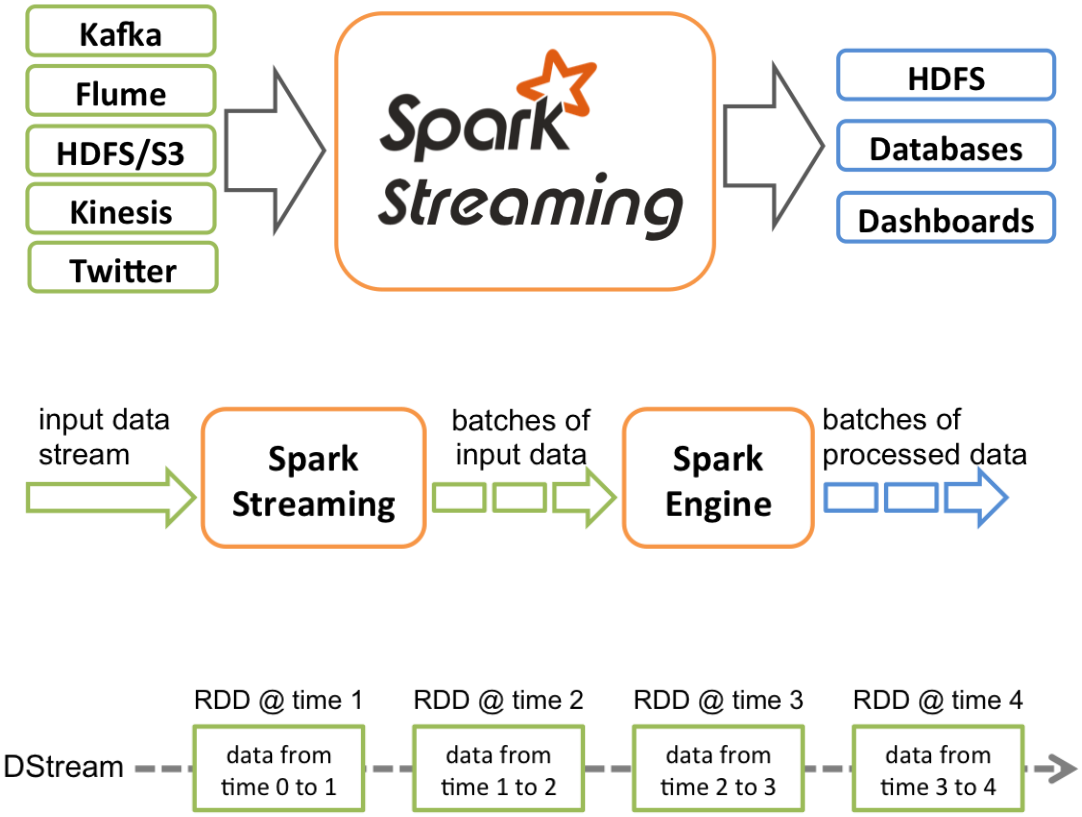
	Hadoop	Spark
Performance	o	+
Utilisabilité/Portabilité	+	++
Frais	o	-/+
Compatibilité	+	+
Traitement de données	-	+
Tolérance aux pannes	+	++
Évolutivité	++	+
Sécurité	+	o

Streaming Analytics

- Des données peuvent être analysées en lots ou en flux.
- La version de MapReduce qu'on a vue analyse les données en lots.
  - Une grande quantité de données entrent dans le système et elles sont toutes analysées lors d'une exécution du système.
- En cas de flux, la connexion entre le producteur des données et le système d'analyse reste toujours ouverte. Les données sont analysées en morceau comme elles viennent.
- Cela permet l'analyse de données en temps réel, ce qui est important pour des systèmes tels que les systèmes d'alarmes ou financiers.

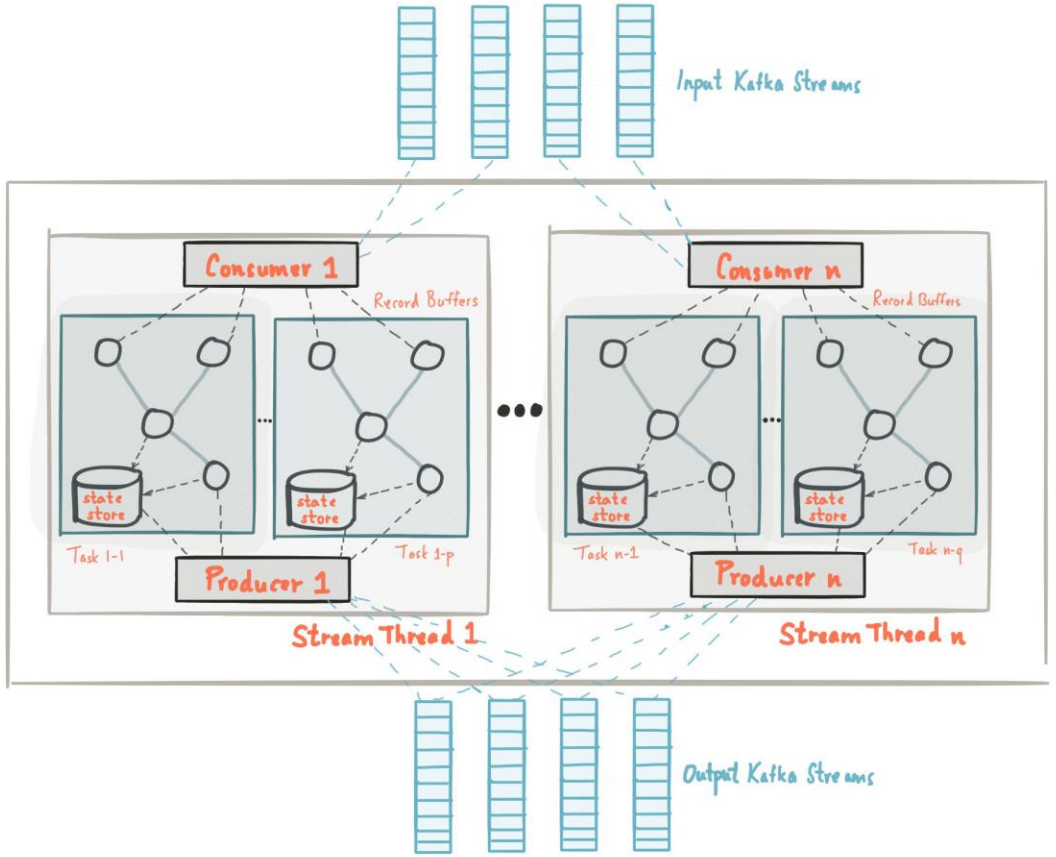
Spark Streaming

- Spark streaming accepte des flux de données de plusieurs sources.
- À intervalles fréquents, Spark extrait des données et les divise en petits lots, qui sont insérés dans le système principal.
- Les flux de données sont représentés comme DStreams (streams discrets), qui sont des fils de RDDs.



Kafka Streaming

- Pour Kafka, chaque thème correspond à un flux.
- Kafka utilise les filières pour paralléliser le traitement des flux.
- Kafka stocke l'état local pour chaque partition (stateful operation).



Spark Streaming vs Kafka Streaming

- La définition des flux et l'entrée des données est basée sur le temps en Spark et sur les évènements en Kafka.
- Spark est une fausse analyse stream, car en réalité il analyse les données en petits lots.
- Kafka est plus efficace et meilleur pour réagir aux évènements.
- Spark est plus stable et portable, et il est compatible avec toutes les technologies de Hadoop.

Carte du cours