ESILV
ENGINEERING SCHOOL
DE VINCI PARIS

IoT -- Cloud

Thème 1 -- Ingestion

Pierre COURBIN
pierre.courbin@devinci.fr

Références et sources

Si vous repérez une erreur dans les affirmations suivantes, n'hésitez pas à me l'indiquer en commentaire de ce Deck !

- Sur chaque image, vous trouverez le lien vers sa source
- Suite à chaque citation, vous trouverez le lien vers sa source
- De façon générale, je n'ai aucun lien avec les différentes sources listées, ce sera sinon indiqué clairement.
  J'ai parfois simplement trouvé leur image particulièrement pertinente...

À propos de ce cours

Ce cours est donné aux étudiants de Master2 (3ème année d'école d'ingénieur) qui ont choisi l'option "IoT".

Cette option est dédiée aux étudiants des Majeures IBO et NE de l'ESILV.
Les étudiants y étudient l'aspect "Cloud" de l'IoT (ce cours), l'IA dans l'IoT, une partie sur les objets eux même et leur conception et enfin un cours sur la sécurité et la responsabilité sociétale.

À propos de l'enseignant

Informations professionnelle sur LinkedIn et sur l'aspect recherche sur ResearchGate.

À propos de l'ESILV

L'ESILV, Ecole Supérieure d'Ingénieurs Léonard de Vinci est une école d'ingénieurs généraliste au cœur des technologies du numérique.

Elle recrute principalement au niveau Baccalauréat (S et STI2D) et forme en 5 ans des ingénieurs opérationnels s'insérant parfaitement dans le monde professionnel.

Le projet pédagogique de l'ESILV s'articule autour **des sciences et des technologies numériques** combinées à 5 thématiques (Informatique, Finance, Mécanique, Energie/Ville, Santé) développées dans **9 grandes spécialisations** : IOCS (Informatique, objets connectés et sécurité), DIA (Data et intelligence artificielle), IF (Ingénierie financière), ACT (Actuariat), FIN (Fintech), MMN (Modélisation et mécanique numérique), IND (Industrie 4.0), EVD (Énergie et villes durables), SB (Santé biotech).

L'ESILV s'appuie aussi sur une **approche pédagogique transverse** avec 20% de son cursus en commun avec une école de management (EMLV) et une école du digital (IIM) dont un parcours Ingénieur Manager en 5 ans, double diplômant.

Elle propose également un Bachelor Ingénierie Numérique et un Bachelor Technology & Management.

2600 élèves. Labellisée EESPIG, l'ESILV est membre de la CGE, de l'UGEI, de la CDEFI, de Campus France, de Talents du Numérique et de LearningLab Network.

ESILV
ENGINEERING SCHOOL
DE VINCI PARIS

Plan du cours

1. **Objectifs de l'Ingestion**
   - Pourquoi ?
   - Principe de Pub/Sub
   - Messages Brokers
2. **MQTT, AMQP ou Kafka ?**
   - Cas d'usages : MQTT/AMQP
   - Cas d'usages : RabbitMQ/Kafka
3. **Focus sur RabbitMQ**
   - Processus global, queues et droit d'accès
   - Publishers et Exchanges

Pourquoi ?

L'intérêt d'un "Message Oriented Middleware"

Principes de base

- Un programme émet/publie un message,
- Un "broker" a la charge de le diffuser,
- Des programmes consomment des messages.

=> **L'émetteur ne remet pas directement le message au consommateur.**

Quelques spécificités

- Communication **Asynchrone** (comme pour un email),
- L'émetteur n'a pas besoin de savoir à qui servira le message, ni quand ni même s'il servira,
- Le consommateur peut s'intéresser à un sous ensemble de messages sans savoir d'où ils viennent,
- Le broker devient central pour s'assurer que chaque programme pourra faire son travail,
- Le broker n'est pas là pour enrichir le message ou le transformer fortement,

Cas d'usage (référence) :

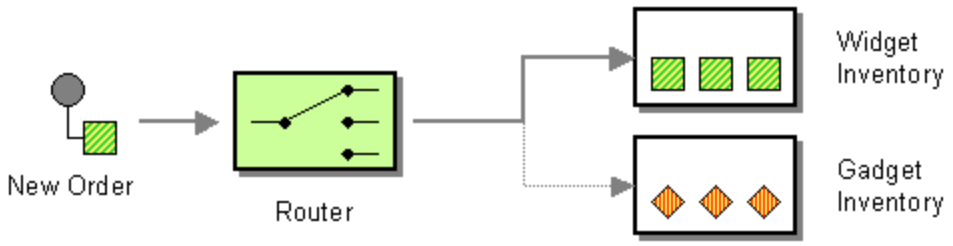
- Collecte d'informations venant d'objets connectés,
- Transmission de métriques liées à des applications (nombre d'utilisateurs, consommation CPU, disponibilité, charge etc),
- Envoi de notifications (mails, SMS etc),
- Messageries instantanées,
- etc.

Pourquoi ?

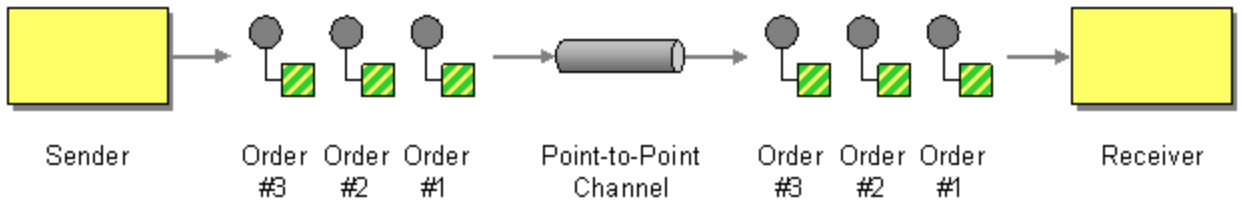
L'intérêt d'un "Message Oriented Middleware"

Différents patterns d'intégration :

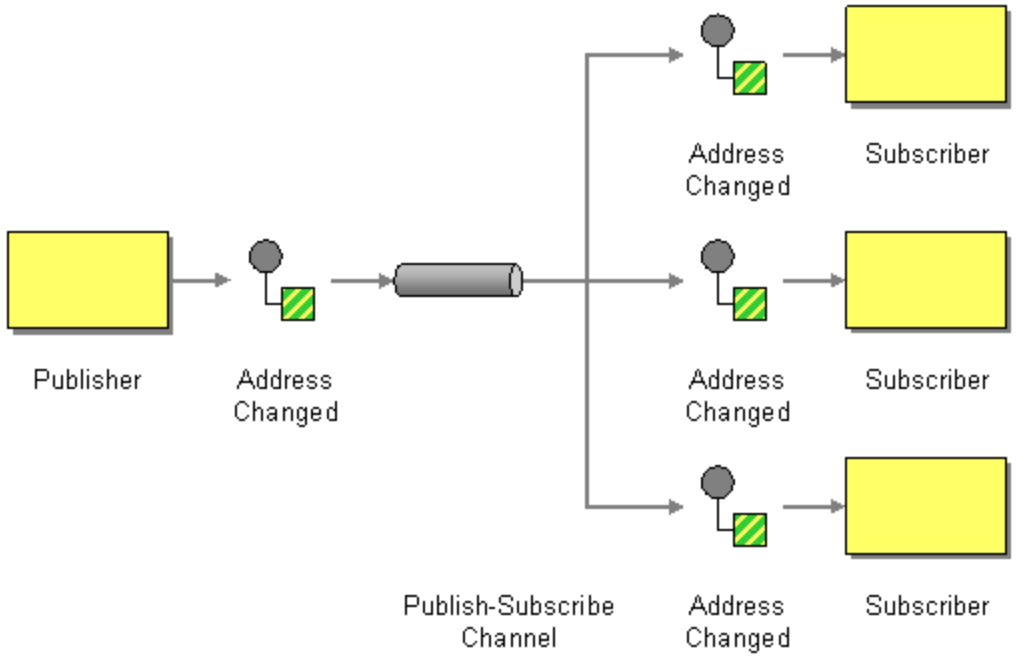
- Content-based Router
- Point-to-Point Channel
- Publish-Subscribe Channel



“ Use a Content-Based Router to route each message to the correct recipient based on message content.



“ Send the message on a Point-to-Point Channel, which ensures that only one receiver will receive a particular message.



“ Send the event on a Publish-Subscribe Channel, which delivers a copy of a particular event to each receiver.

Principe de Pub/Sub

Objectifs et fonctionnement

> *In software architecture, publish–subscribe is a messaging pattern where **senders of messages, called publishers, do not program the messages to be sent directly to specific receivers, called subscribers**, but instead **categorize published messages into classes** without knowledge of which subscribers, if any, there may be. Similarly, **subscribers express interest in one or more classes** and only receive messages that are of interest, without knowledge of which publishers, if any, there are.*
>
> Publish–subscribe pattern, Wikipedia

Filtrage des messages

- **topic-based** : messages publiés dans des *topics*. Les *subscribers* s'abonnent à des *topics*.
  Le *publisher* doit envoyer son message au bon *topic*.
- **content-based** : messages délivrés au *subscribers* seulement si un attribut du message correspond à ce qu'il attend.
  Le *subscriber* doit définir la règle qui lui permet de recevoir les bons messages.

Topologie

Nous nous concentrons sur les topologies **utilisant un *broker*** (courtier/négocier).
Les *publishers* et les *subscribers* s'enregistrent sur un *broker* et **lui délèguent la tâche de filtrer et organiser les messages**.
Ce *broker* pourra aussi prioriser les messages, les organiser en fil d'attente (queues) etc.

Principe de Pub/Sub

Objectifs et fonctionnement

*“ Message queues allow each component to **perform its tasks independently** - it allows components **to remain completely autonomous** and unaware of each other. A change in one service shouldn't requires a change in the other services. It is the process of separating services so that their functionality is more self-contained.*

Decoupling in IoT, CloudAMQP

Avantages

- **Asynchrone**
  Les messages n'ont pas à être traités tout de suite, ni au même endroit.
  Si un service de traitement est occupé, la file d'attente augmente mais on ne perd rien.
- **Faible couplage**
  Découplage les *publishers* et *subscribers*.
  Les producteurs ou consommateurs de messages n'ont pas besoin de savoir que les autres existent.
  Dans le cas "topic-based", ils n'ont même pas besoin de connaitre la topologie du système.
- **Scalabilité**
  Plusieurs options pour améliorer la *scalability* par rapport à un traditionnel client-serveur : cache de messages, traitements parallèles, etc

Limites

- **Assurance de livraison** de messages
  Les mécanismes du *broker* et le design de l'architecture ne permettre pas forcément de garantir qu'une donnée est bien livrée, éventuellement dans un temps contraint...

Message Brokers

Objectifs et fonctionnement

Principe et fonctionnalités

- Utilise généralement le Pub/Sub
- Propose une architecture et des protocoles pour, par exemple :
  - stocker ponctuellement des messages,
  - redonder les messages,
  - garantir une livraison,
  - organiser les topics/filtres,
  - etc.

Quels protocoles ?

Différents protocoles existent (par exemple MQTT -- Message Queuing Telemetry Transport, ou AMQP -- Advanced Message Queuing Protocol), ils permettent de :

- rendre un *broker* interchangeable par un autre suivant le même protocole
- faciliter l'interconnexion entre les services et l'évolutivité/compatibilité ascendante

Exemple de *brokers*

- RabbitMQ (propose notamment du MQTT et du AMQP)
- Mosquitto (MQTT)
- Apache Kafka (protocole spécifique)

ESILV
ENGINEERING SCHOOL
DE VINCI PARIS

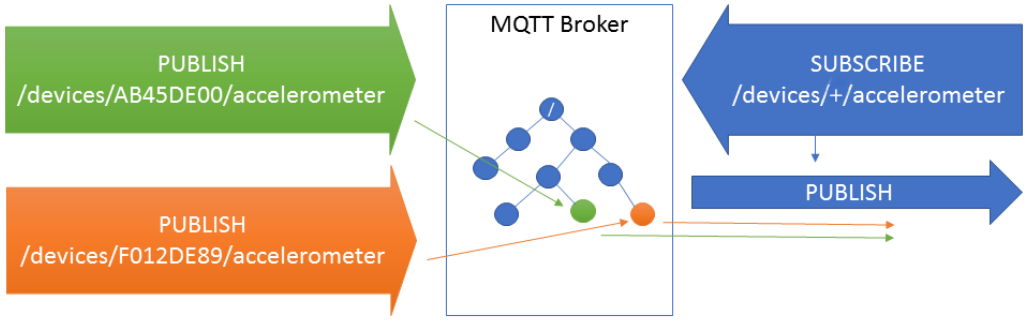
Plan du cours

1. **Objectifs de l'Ingestion**
   - Pourquoi ?
   - Principe de Pub/Sub
   - Messages Brokers
2. **MQTT, AMQP ou Kafka ?**
   - Cas d'usages : MQTT/AMQP
   - Cas d'usages : RabbitMQ/Kafka
3. **Focus sur RabbitMQ**
   - Processus global, queues et droit d'accès
   - Publishers et Exchanges

Cas d'usages : MQTT/AMQP

Cas du MQTT

> *“* ***MQTT*** *started as a* ***lightweight*** *companion protocol to IBM’s MQ messaging middleware for allowing* ***easy integration of industrial equipment with each other****, and with backend enterprise systems (via MQ). It has a* ***strong focus on minimal wire footprint****. The “TT” stands for “telemetry transfer” and that’s still, in very many use-cases, one of its key purposes.*
>
> From MQTT to AMQP and back, Clemens Vasters



Spécificités

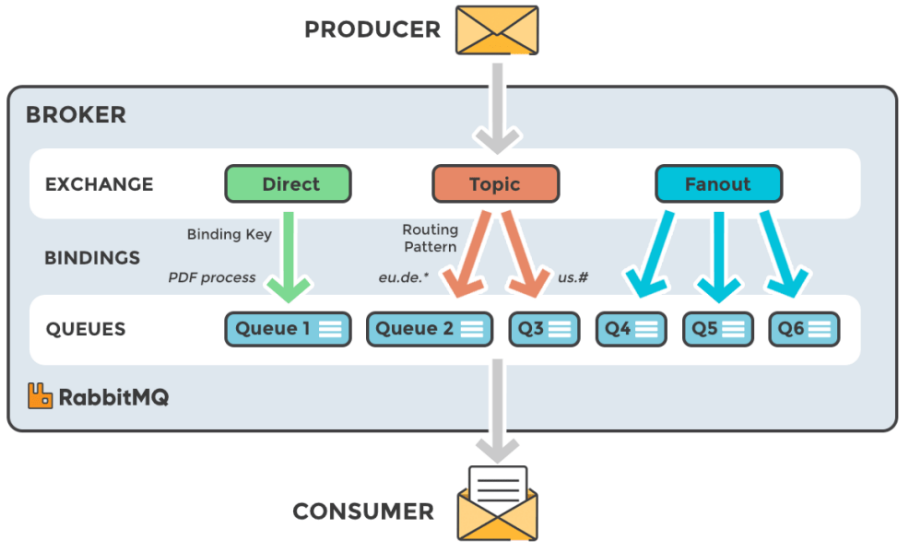
- Les topics sont partagés au niveau du *broker* et forment un "*topic graph*",
- Un consommateur peut s'abonner à des topics avec des filtre conditionnel,
- Les producteurs peuvent créer des topics/arborescences à la volée,
- Légèreté en terme d'emprunte mémoire et réseau (fonctionne directement sur des systèmes embarqués),
- Simplicité de connexion, de création d'arborescence...
- Focus sur la latence (*latency*) plutôt que sur la fiabilité (*reliability*)

Cas d'usage

- Publier des informations de télémétrie vers une infrastructure de calcul
- Envoi ponctuel de petite quantité de données avec contrainte de connexion réseau
- Envoi de données à partir d'un objet avec peu de ressources (puissance de calcul, énergie etc)

Cas d'usages : MQTT/AMQP

Cas du AMQP



> *“* ***AMQP*** *is a* ***general-purpose*** *message transfer protocol suitable for a broad range of messaging-middleware infrastructures, and also for peer-to-peer data transfer. It’s a* ***symmetric and bi-directional protocol*** *that allows either party on an existing connection to initiate links and transfers, and has* ***rich extensibility and annotation features*** *at practically all layers.*
>
> From MQTT to AMQP and back, Clemens Vasters

Spécificités

- Possibilité de mettre en place une architectures complexe/flexible grâce à la séparation *Exchange*/*Queues* et aux différents types d'*exchanges*,
- Le consommateur doit avoir une idée de l'architecture pour savoir à quel *topic* s'abonner,
- La flexibilité le rend "verbeux", notamment pour initier une connexion, mais messages assez court ensuite (par rapport à MQTT, pas de rappel du *topic* dans le message).
- Bi-directionnel (le client ou le serveur peuvent initier une communication)
- Focus sur la fiabilité (*reliability*) plutôt que sur la latence (*latency*)

Cas d'usage

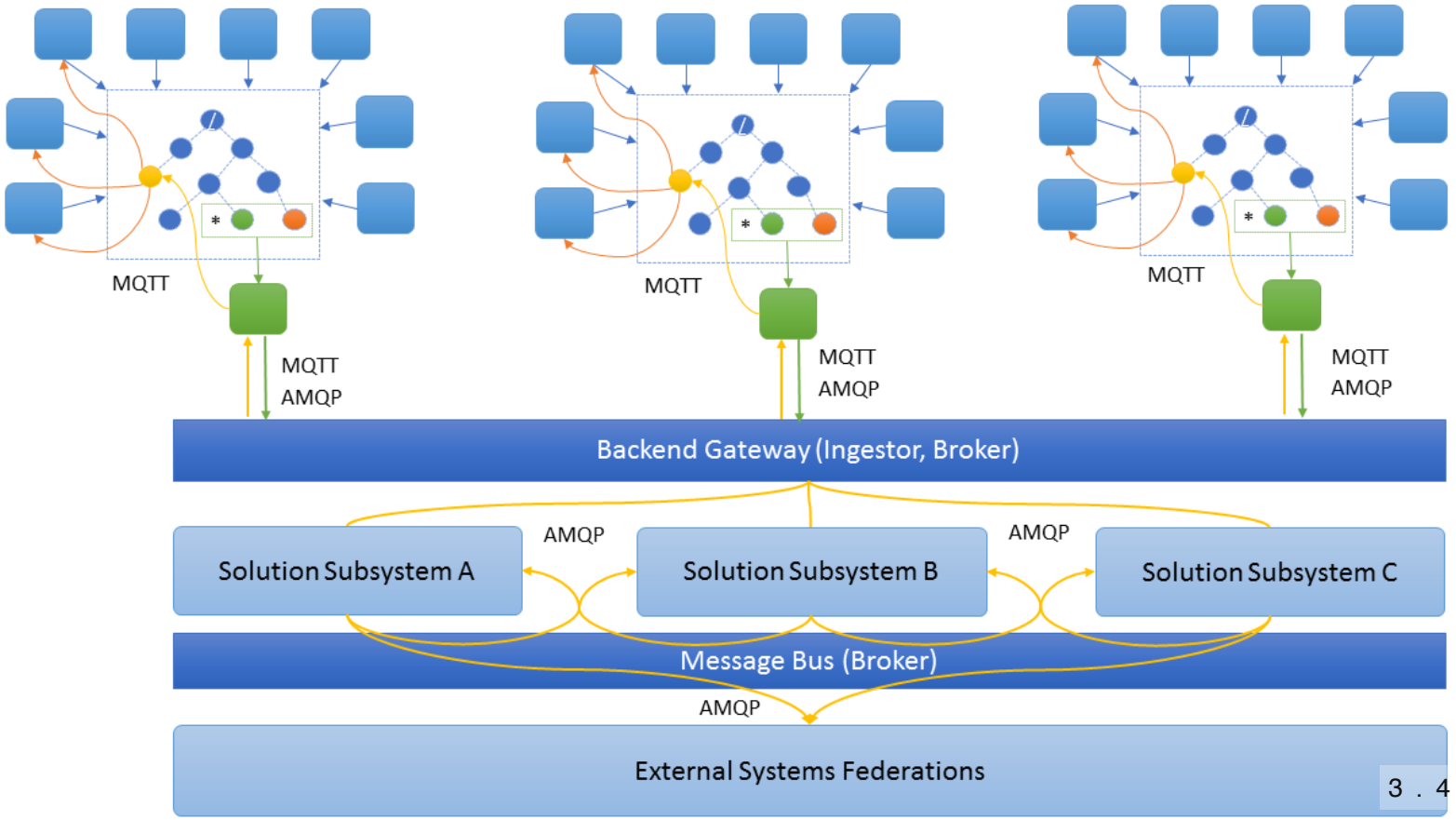
- Envoi régulier d'informations de taille importante (batch de données temporelles par exemple)
- Envoi de messages avec contraintes spécifiques (routage complexe vers différents type de consommateurs)

Cas d'usages : MQTT/AMQP

MQTT et/ou AMQP ?

Combiner les avantages de chacun...

- Utilisation de MQTT sur chaque site physique
  -- Des objets locaux peuvent avoir besoin de données des autres objets pour fonctionner : latence > fiabilité.
- Un concentrateur sur chaque site filtre et sélectionne/agrège les données importantes pour un usage local et/ou pour les transmettre à un système central.
- Le système central a une vue globale et transmets les informations à des sous-systèmes, partenaires etc en suivant des règles complexes grâce à AMQP.

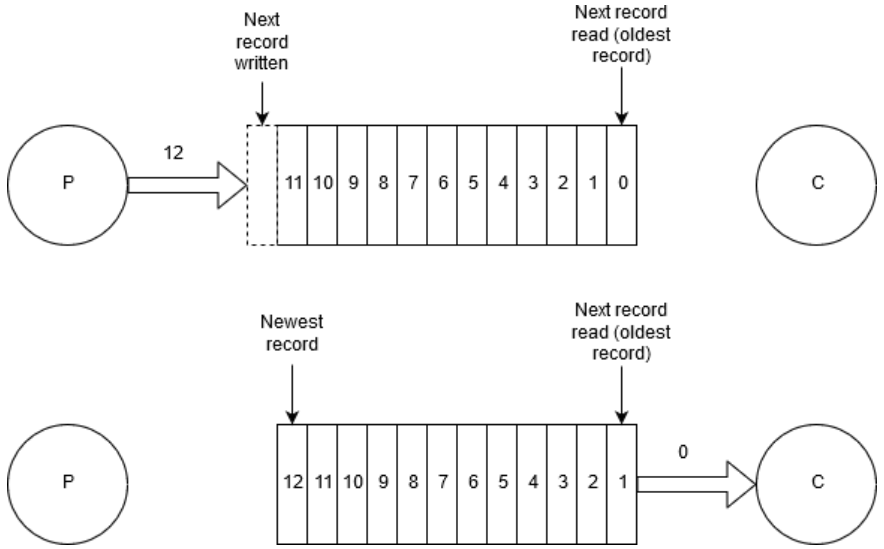


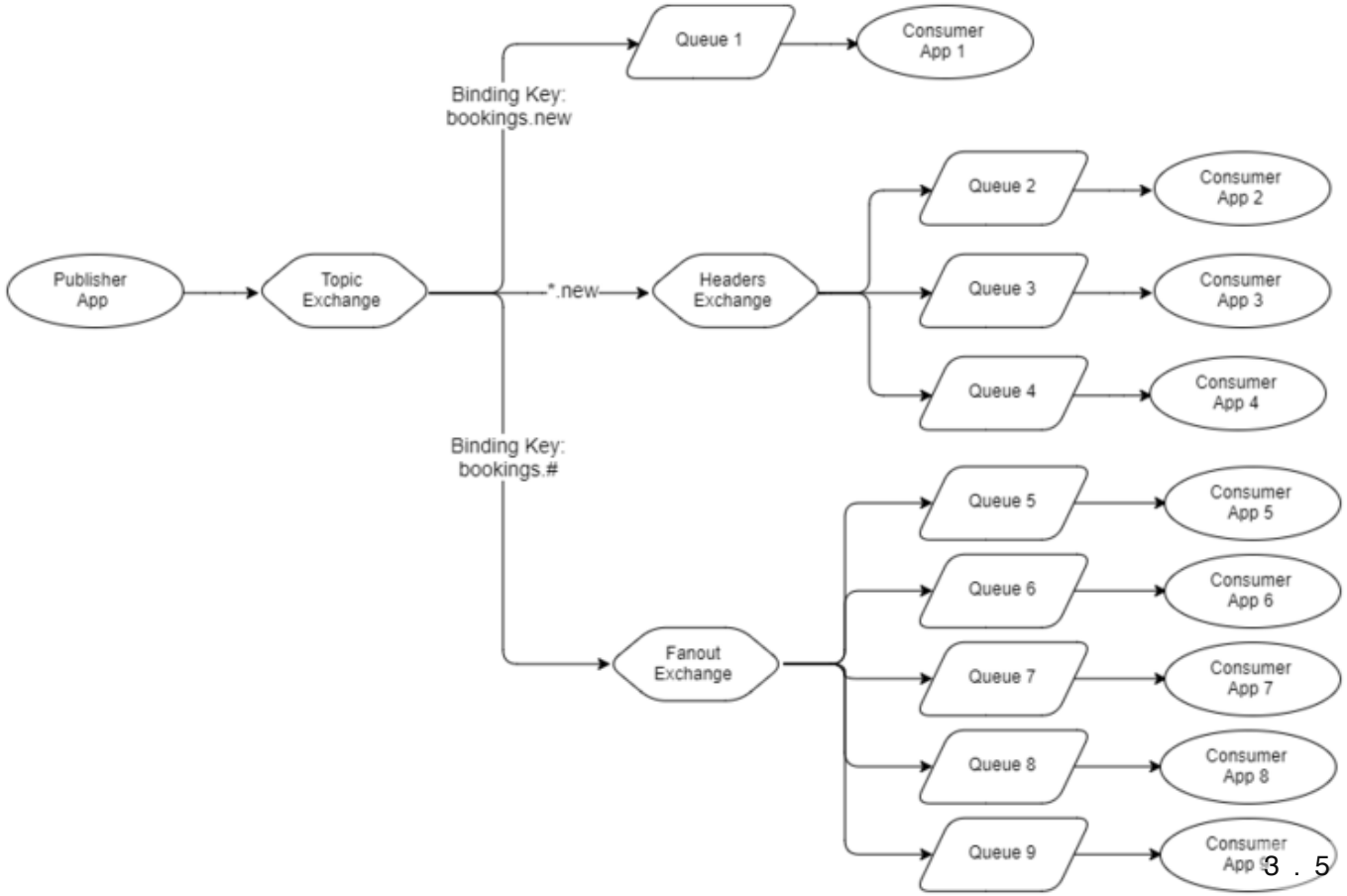
Cas d'usages : RabbitMQ/Kafka

Cas de RabbitMQ

Généralités

- Détenu par Pivotal Software depuis 2013,
- Propose une passerelle pour différents protocoles (AMQP, HTTP, STOMP, et MQTT),
- Développé en Erlang,
- Open Source (Mozilla Public License)
- Grâce à AMQP, possibilité d'avoir des architectures complexes,
- Utilisé par les entreprises comme Instagram, Google, Mozilla, NASA, BBC, OpenStack...



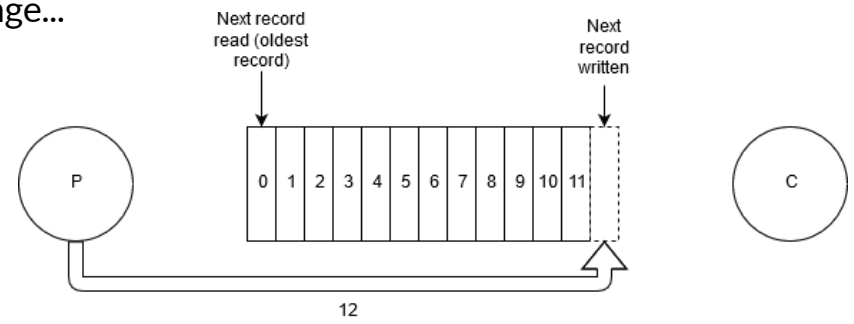


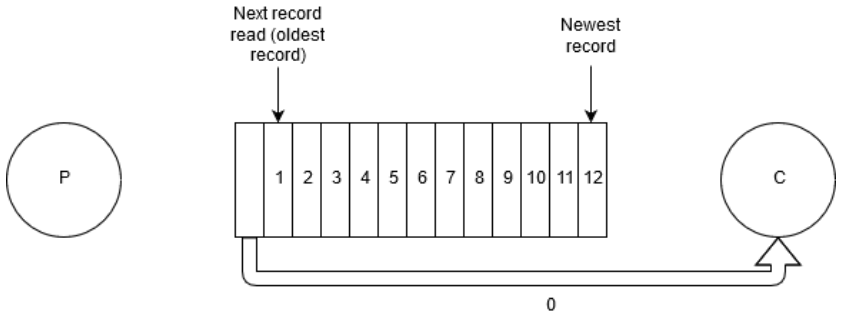
Cas d'usages : RabbitMQ/Kafka

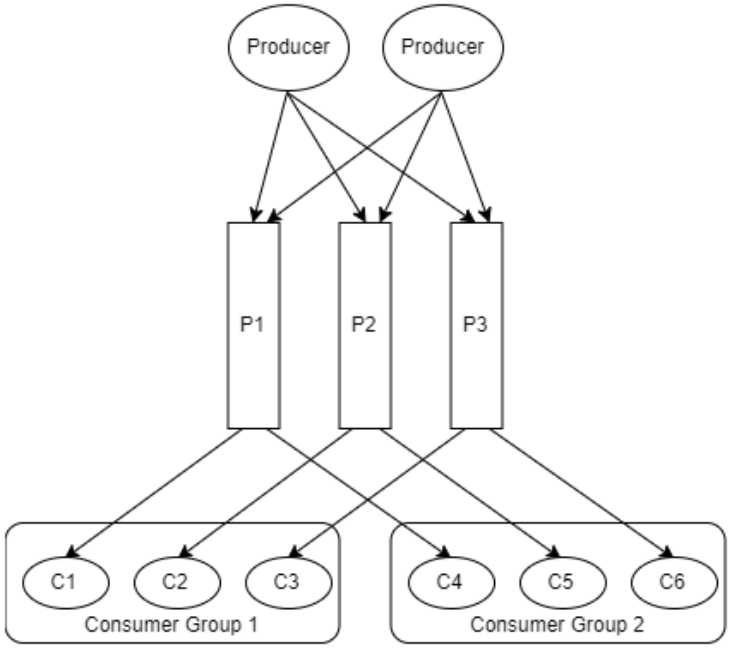
Cas de Kafka

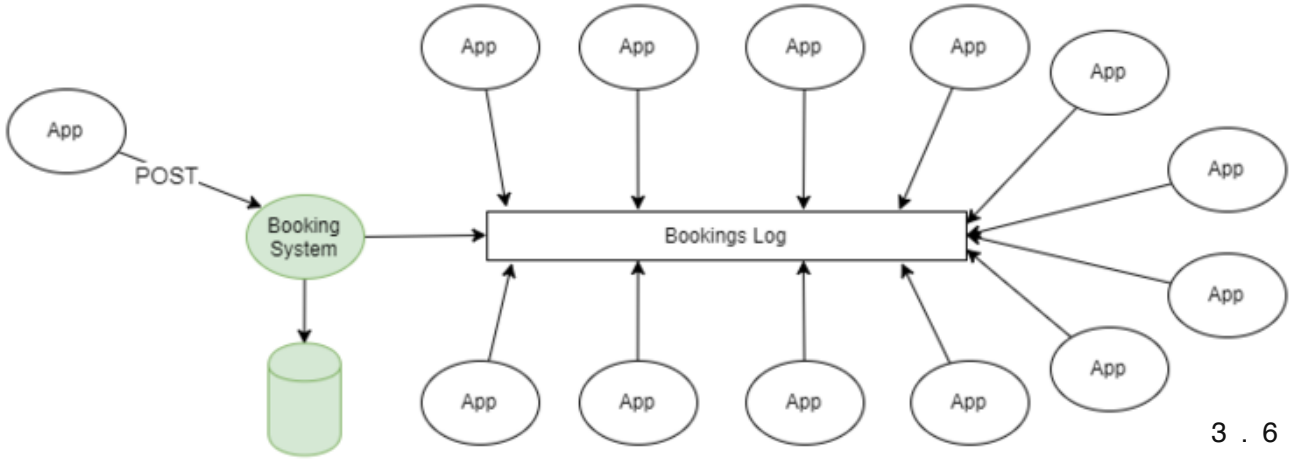
Généralités

- Développé par l'Apache Software Foundation et LinkedIn, code ouvert depuis 2011.
- Propose une architecture et un protocole spécifique
- Développé en Scala et Java
- Open Source (Apache License 2.0)
- Utilisé par les entreprises comme LinkedIn, Netflix, Spotify, Uber, Orange...









Cas d'usages : RabbitMQ/Kafka

Comparaison argumentée

voir aussi cet article de Nokia Bell Labs

Source	RabbitMQ	Kafka
Stockage des messages	Fonctionne mieux quand les messages sont consommés rapidement (files presque vides). Stockage des messages en mémoire puis sur disque. Message détruit une fois consommé et que l'a.	Architecture pour consommer des messages à différentes échelles, donc accumulation possible. Stockage des messages sur disque, sous forme de logs. Message détruit après un délai spécifique (retention policy).
Logique de routage	Très flexible (voir AMQP) en utilisant des clés de routage dynamiques et l'entête des messages.	Basique, fondé sur la notion de topic.
Garantie de livraison	"At Least Once" Mécanisme d'acquittement entre les clients (producteurs/consommateurs) et le broker.	"At Least Once" Acquittement entre le producteur et le broker. Les messages peuvent ensuite être consommés jusqu'à expiration.
Conservation de l'ordre des messages	Conservé pour les messages issus d'un même channel.	Garanti au niveau d'une partition (un topic peut être décomposé en plusieurs partitions)
Multidiffusion	Multicast possible en connectant plusieurs files à un même exchange (fanout) et un consommateur à chaque file.	Multicast géré au niveau des groupes de consommateurs : message envoyé qu'à un seul membre du groupe. Pour multidiffuser, il faut autant de groupes de consommateurs que de consommateurs.
Disponibilité	Réplication des composants entre les instances RabbitMQ d'un cluster. Messages consommés sur la file maîtresse, mais publié sur n'importe quel file.	Facteur de réplication paramétrable (topics subdivisés en partitions, chaque partition peut être répliquée plusieurs fois.)

RabbitMQ

ESILV
ENGINEERING SCHOOL
DE VINCI PARIS

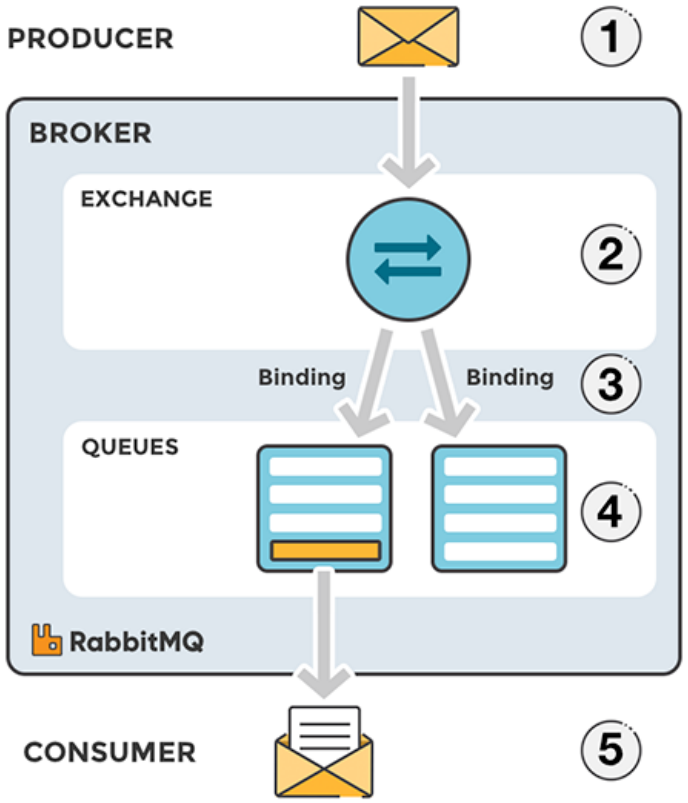
Plan du cours

1. **Objectifs de l'Ingestion**
   - Pourquoi ?
   - Principe de Pub/Sub
   - Messages Brokers
2. **MQTT, AMQP ou Kafka ?**
   - Cas d'usages : MQTT/AMQP
   - Cas d'usages : RabbitMQ/Kafka
3. **Focus sur RabbitMQ**
   - Processus global, queues et droit d'accès
   - Publishers et Exchanges

Processus global, queues et droit d'accès

Processus global

1. *Producer* publie un message vers un *exchange* (voir les types à la suite).
2. L'*exchange* a la responsabilité de router le message vers la/les bonne(s) *queue(s)* en fonction des attributs du message et du type d'*exchange*.
3. Les *bindings* correspondent aux liens entre les *exchanges* et les *queues*.
4. Le message reste dans la *queue* tant qu'il n'est pas lu (voir aussi notion de TTL)
5. Le *consumer* récupère le message et valide la réception auprès de la *queue* qui supprime le message.



Quelques notions importantes

- Les *queues* peuvent être organisées dans des "*virtual hosts*" pour segmenter/cloisonner les messages.
- On peut fixer des droits d'accès associés à un identifiant sur :
  - Un *virtual host*,
  - Une *queue*,
  - un *exchange*

Publishers et Exchanges

Les types d'*Exchanges*

Différents types d'*Exchanges*, d'après cet article :

- **Direct**: The message is routed to the queues whose binding key exactly matches the routing key of the message. For example, if the queue is bound to the exchange with the binding key pdfprocess, a message published to the exchange with a routing key pdfprocess is routed to that queue.
- **Fanout**: A fanout exchange routes messages to all of the queues bound to it.
- **Topic**: The topic exchange does a wildcard match between the routing key and the routing pattern specified in the binding.
- **Headers**: Headers exchanges use the message header attributes for routing.