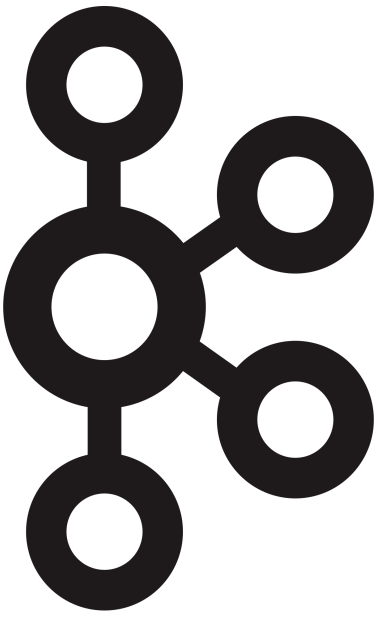
BEDJAOUI Khalil
DELANOE Rémi
DIMROCI Anatole

FISA Informatique

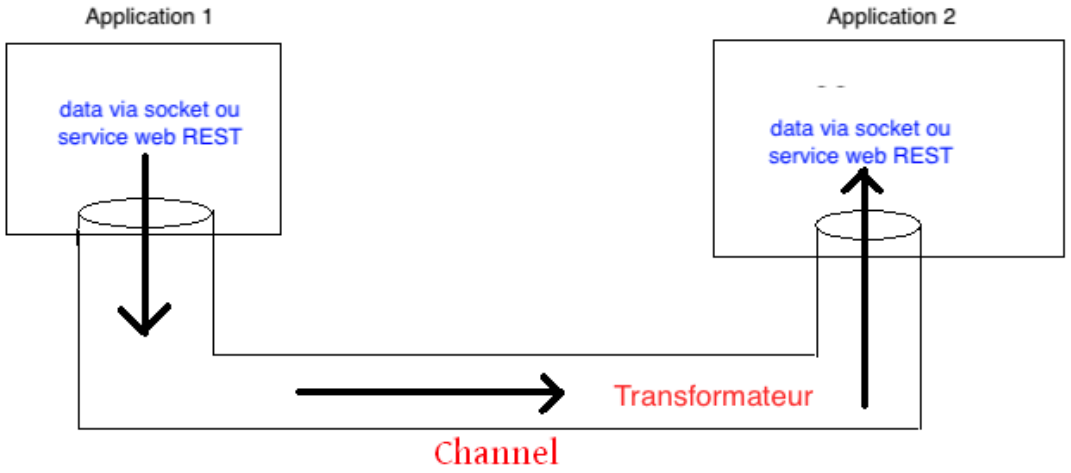
Projet Kafka



1) Dans l'hypothèse où l'on ne connaît pas les avantages et bénéfices des bus kafka, il est possible d'utiliser un EAI.
Mirth Connect pourrait effectivement agir comme MiddleWare et voici un schéma simple montrant ce qu'il fait :

Pour détailler, Mirth Connect reçoit les données via des sockets ou des services web REST, les transformer (ou pas) et les renvoyer via des sockets ou services web REST.
Cette architecture est possible via les "canaux" (il faut les configurer) Mirth Connect permettant la réception et l'envoi des données via sockets et services web.
Pour ce qui est de la transformation (optionnelle) des messages, cela est possible grâce aux transformateurs Mirth Connect.

On aurait donc une architecture plus simple à mettre en place mais moins évolutive en termes de volumes de messages à traiter et de systèmes à intégrer.



2) Mirth Connect est un outil plus facile à appréhender pour les développeurs ayant une expérience en intégration d'applications. En effet, il utilise des concepts familiers tels que les flux de messages et les transformations de données, ce qui peut faciliter la compréhension et l'utilisation de l'outil.

En revanche, Kafka est une technologie plus complexe qui nécessite une compréhension plus approfondie des concepts tels que les topics, les partitions et les offsets. Cependant, une fois la technologie maîtrisée, Kafka offre une grande flexibilité et une haute performance pour la gestion de flux de données en temps réel.

En termes de déploiement et de maintenance, Mirth Connect peut être installé facilement sur un serveur et peut être configuré via une interface graphique utilisateur. Cependant, Kafka nécessite une configuration plus avancée et une

attention particulière doit être portée à la mise à l'échelle et à la haute disponibilité.

En résumé, Mirth Connect peut être plus simple à mettre en place et à maintenir pour les développeurs ayant une expérience en intégration d'applications, tandis que Kafka peut être plus complexe à comprendre mais offre une plus grande flexibilité et des performances élevées pour la gestion de flux de données en temps réel.

3) Dans un premier temps, les possibilités pour sécuriser les échanges de messages sont :
- SSL/TLS pour chiffrer les données échangées entre les différents composants Kafka.
- Une authentification des clients via SASL (Simple Authentication and Security Layer) pour authentifier les clients qui se connectent au bus Kafka
- Les ACL (Access Control Lists) qui permettent la gestion des autorisations d'accès des différents clients aux différents topics de Kafka.
- Un chiffrage (autre que SSL/TLS) des données pour en garantir la confidentialité, l'intégrité et l'authenticité.

Si on choisit SSL/TLS, voici le principe de mise en place :
- Générer des certificats SSL/TLS pour chaque broker Kafka et pour chaque client Kafka. Ces certificats peuvent être auto-signés ou émis par une autorité de certification (CA) de confiance.
- Configurer les brokers Kafka pour utiliser SSL/TLS. Cela implique de configurer le fichier "server.properties" pour inclure les informations du certificat SSL/TLS.
- Configurer les clients Kafka pour utiliser SSL/TLS et SASL. Cela implique de configurer les fichiers de propriétés du client pour inclure les informations du certificat SSL/TLS et les informations d'authentifications SASL.
- Définir les règles d'autorisation pour les clients Kafka et les brokers Kafka. Cela peut être fait en utilisant des fichiers de configuration ou en utilisant des API pour définir les règles d'autorisation.
- Tester la sécurité en envoyant des messages en clients et brokers Kafka et en vérifiant que les messages sont bien chiffrés et que seuls les clients autorisés peuvent y accéder.

Vous remarquerez qu'il n'existe aucune classe pour le producer 2. Elle existait mais finalement ne sert à rien car l'envoi des messages dans le topic se fait par CommandConsumer.