BNP – TEC

Expression de besoin d’un POC sur Apache Camel – Flux de paiement

Date : 09/10/2025

# 1. Contexte du projet

Dans le cadre de la migration de notre architecture d’intégration, BNP – TEC souhaite évaluer les capacités de la plateforme Apache Camel (Red Hat Integration) pour la mise en œuvre de flux de paiements financiers complexes. L’objectif de ce POC est de valider la pertinence fonctionnelle, technique et opérationnelle de Camel pour des cas d’usage concrets liés au traitement des messages de paiement interbancaires.

# 2. Objectif du POC

Le POC a pour but de démontrer la capacité d’Apache Camel à :

* Intégrer des messages de paiement provenant de systèmes hétérogènes (IBM MQ, Kafka, CFT, http etc.) ;
* Valider et transformer des messages selon les standards du domaine financier (ISO20022, SWIFT, etc.) ;
* Réaliser des transformations métier (enrichissement, règles de conversion, formatage, etc.) ;
* Distribuer les messages transformés vers plusieurs systèmes cibles ;
* Assurer le monitoring, la traçabilité et l’audit de bout en bout des traitements.
* Démontrer la capacité d'Apache Camel à structurer les flux sous forme de composants métier réutilisables et modulaires, facilitant leur maintenance et leur réutilisation dans d'autres contextes d'intégration.
* Évaluer l'outil Kaoto pour la conception, la visualisation et la maintenance des routes Camel, en mettant l'accent sur la productivité des développeurs et la simplicité d'évolution des flux.
* Définir et valider une bibliothèque de templates de flux Camel réutilisables pour standardiser le développement des futures intégrations.
* Démontrer comment Kaoto peut faciliter l'utilisation et la personnalisation de ces templates via son interface visuelle.
* Évaluer la réduction du time-to-market pour le développement de nouveaux flux grâce à cette approche par templates.
* Démontrer la capacité d'Apache Camel à gérer une configuration externalisée et dynamique adaptée à des déploiements multi-environnements (dev, qualif, prod) sur OpenShift.
* Valider les mécanismes de rechargement de configuration à chaud sans redémarrage de l'application.
* Évaluer l'intégration avec l'écosystème OpenShift (ConfigMaps, Secrets, Service Discovery) pour une gestion cloud-native de la configuration.
* Évaluer les performances et la scalabilité de la solution Apache Camel sous charge réaliste de messages de paiement.
* Mesurer le débit maximal (throughput) et la latence du flux de bout en bout.
* Valider l'efficacité de la consommation des ressources (CPU, mémoire) sur OpenShift.

# 3. Principes Architecturaux et de Conception

Pour garantir la maintenabilité et l'évolutivité de la plateforme, le POC devra impérativement respecter les principes suivants :

* **Modularité des Routes :**
  + Les routes Camel doivent être découpées en segments logiques et autonomes (micro-routes). Par exemple : une route pour la "Réception et Validation", une autre pour la "Transformation Métier", etc.
  + Ces modules doivent communiquer entre eux via des canaux asynchrones (seda, direct-vm) ou des endpoints de messagerie (Kafka), permettant leur réutilisation et leur orchestration indépendante.
* **Réutilisabilité des Composants :**
  + Les transformations de données communes (ex: XML <-> CDM) doivent être encapsulées dans des **beans Java réutilisables** ou des **templates Camel**.
  + Les logiques métier fréquentes (enrichissement, validation spécifique) doivent être développées une fois et exposées comme des services réutilisables au sein de l'espace d'intégration.
* **Abstraction des Connecteurs :**
  + Les configurations spécifiques aux systèmes (URL MQ, topics Kafka, schémas de base de données) doivent être externalisées dans des fichiers de configuration (.properties ou via ConfigMap/Secret si déployé sur Kubernetes), et non codées en dur dans les routes.
* **Gestion de Configuration Cloud-Native**

Le POC doit mettre en œuvre une stratégie de gestion de configuration adaptée aux contraintes OpenShift :

* **Externalisation Complète :**
  + Tous les paramètres configurables (URLs de connexion, credentials, noms de files, topics Kafka, seuils métier) doivent être externalisés hors du code.
  + Utilisation de profils Spring Boot (application-dev.yml, application-prod.yml) couplée avec les mécanismes OpenShift.
* **Configuration Hiérarchique :**
  + **Niveau 1 :** Configurations par défaut dans le JAR
  + **Niveau 2 :** ConfigMaps OpenShift pour les paramètres environnementaux
  + **Niveau 3 :** Secrets OpenShift pour les données sensibles
  + **Niveau 4 :** Variables d'environnement pour les surcharges runtime
* **Configuration Dynamique :**
  + Mise en place de mécanismes de rechargement à chaud utilisant :
    - camel-kubernetes pour surveiller les changements de ConfigMaps
    - Spring Cloud Config pour une gestion centralisée
    - Camel's PropertiesComponent avec rechargement périodique
* **Observabilité et Gouvernance :**
  + Chaque module doit exposer des métriques standardisées (via Micrometer) et générer des traces de log structurées pour un monitoring granulaire.

# 4. Périmètre fonctionnel attendu

Le scénario du POC porte sur le traitement d’un message de paiement ISO20022 (exemple : pain.008.001.02 – prélèvement SEPA). Le flux se déroule comme suit :  
1. Réception d’un message XML via un connecteur MQ Series.  
2. Validation du message selon le schéma XSD ISO20022 standard d’un pacs 008 (xsd en attaché).  
3. Transformation du message vers un modèle générique interne en XML (CDM) (xsd en atatché).  
4. Application d’une règle métier simple d’enrichissement ou de transformation.  
5. Reproduction du message au format target.  
6. Distribution du message vers plusieurs destinations avec des formats différents (exemple : un moteur de paiement via MQ et un système de reporting reporting via Kafka).  
7. Enregistrement des traces et audits dans une base PostgreSQL (le traitement des messages doit être asynchrone et non sur le chemin critique)

8. Configuration Externalisée :

* Les noms de files MQ (poc.bnp.input.queue) sont configurés via ConfigMap
* Les topics Kafka (reporting.events) sont paramétrés dynamiquement
* Les règles métier (ex: seuil de montant = ${business.amount.threshold}) sont externalisées

9. Gestion des Secrets :

* Les credentials de connexion aux systèmes sont injectés via Secrets OpenShift
* Les certificats SSL/TLS sont gérés via les mécanismes OpenShift

10. Rechargement à chaud :

* Démonstration de la modification d'un paramètre (ex: seuil métier) dans un ConfigMap et rechargement sans redémarrage du pod

# 5. Résultats attendus

Le POC doit démontrer :  
- La simplicité de mise en œuvre de flux d’intégration complexes avec Apache Camel ;  
- La gestion native des standards financiers (validation XSD, XML, ISO20022) ;  
- La souplesse du mapping et de la transformation de données ;  
- La capacité de Camel à orchestrer plusieurs systèmes (MQ, Kafka, base de données) ;  
- La gestion des audits et événements techniques dans une architecture distribuée.

# 6. Environnements et technologies concernés

Les technologies suivantes sont envisagées pour le POC (à confirmer par Red Hat)

|  |  |  |
| --- | --- | --- |
| Domaine | Technologie envisagée | Objectif |
| Intégration / ESB | Apache Camel (Red Hat Integration / Camel Quarkus) | Moteur d’intégration |
| Messaging | IBM MQ Series | Système source / destination |
| Streaming | Apache Kafka | Distribution d’événements / reporting |
| Base de données | PostgreSQL | Stockage des traces et audits |
| Formats de message | ISO20022 (pain/pacs), SWIFT | Formats financiers standards |
| Format pivot | CDM (Common Data Model) | Format de données générique |
| Outillage & Productivité | Kaoto | Conception visuelle, gestion et maintenance des routes Apache Camel. |
| Modularité | Camel Components, Beans Spring/Quarkus | Création de briques logicielles réutilisables. |
| Orchestration de modules | Camel SEDA, Direct VM | Communication asynchrone et découplage entre les modules. |
| Gestion de Config | OpenShift ConfigMaps & Secrets | Stockage externalisé et sécurisé de la configuration |
| Configuration Dynamique | camel-kubernetes, Spring Cloud Config | Rechargement à chaud de la configuration |
| Sécurité | OpenShift Service Accounts & RBAC | Accès sécurisé aux ressources cluster |

# 7. Livrables attendus

À l’issue du POC, les éléments suivants sont attendus :

1. Une architecture de référence proposée par Red Hat ;
2. Une implémentation de démonstration illustrant le scénario de bout en bout ;
3. Une documentation expliquant les choix d’architecture et de conception ;
4. Un retour d’expérience sur les bonnes pratiques et recommandations d’industrialisation.
5. Catalogue de Composants Réutilisables : Documentation et code source des briques métier et techniques conçues pour être réutilisées (ex: connecteur MQ standardisé, transformateur CDM, service d'audit).
6. Modèle d'Architecture Modulaire : Schéma détaillant le découpage modulaire et les interactions entre les composants
7. Retour d'expérience sur Kaoto : Évaluation de l'outil sur les critères de productivité, de visualisation des flux complexes, et de facilité de refactoring.
8. Modèle de Configuration Multi-environnements : Arbre de configuration type avec séparation dev/qualif/prod
9. Chart Helm / Template OpenShift : Packaging de l'application avec gestion des ConfigMaps et Secrets
10. Stratégie de déploiement GitOps : Documentation pour la gestion de la configuration via Git (avec ArgoCD ou équivalent)
11. Procédures de rechargement à chaud : Documentation opérationnelle pour modifier la configuration en production
12. Retour d'expérience et les recommandations sur les aspects performance

# 7. Attentes vis-à-vis de Red Hat

L’équipe Red Hat est attendue sur les points suivants :

* Proposer la meilleure approche technique pour répondre au besoin ;
* Identifier les bonnes pratiques de performance, sécurité et observabilité ;
* Fournir des recommandations d’architecture cible et d’évolution post-POC.
* Démontrer les bonnes pratiques OpenShift pour le déploiement d'applications Camel
* Proposer une stratégie de configuration industrialisable pour des centaines de flux
* Fournir des bonnes pratiques de conception de templates modulaires et paramétrables
* Démontrer comment Kaoto supporte nativement les Kamelets et templates personnalisés
* Démontrer comment apache camel gère les aspects de performance et multithreading

# 8. Prochaines étapes

Les prochaines étapes proposées sont :  
1. Organisation d’un atelier de cadrage entre BNP – TEC et Red Hat pour préciser le périmètre technique si le besoin n’est pas clair ;   
2. Réalisation du POC par Red Hat avec points d’étape réguliers ;  
3. Présentation des résultats, retour d’expérience et recommandations pour une éventuelle phase d’industrialisation.