Rapport de l’expérimentation

POC micro-services SICA

Table des matières

Présentation

6 jours ont été consacrés à l'élaboration d'un POC permettant d'identifier les problématiques liées à la migration des applications métiers BCEAO vers des architectures micro-service évolutives, permettant la haute-disponibilité et la sécurisation des systèmes.

Les agents BCEAO, répartis en équipes de 1 à 3 personnes, avaient la responsabilité d'un micro-service.
Cette expérimentation pratique faisaient suite à des jours de formation ayant présentés les apports du framework SpringBoot3 ainsi que les design patterns des architectures micro-services

Les objectifs de ce POC étaient :

- Valider les acquis Spring Boot 3 et Design Patterns pour Micro-services des agents
- Évaluer la solution JMix pour la construction d'applications BCEAO (interface utilisateur et API Rest) et son intégration dans un architecture micro-services
- Identifier les problématiques propre au développement de micro-service:
  - Organisationnel :
    - Workflow de collaboration à l'intérieur d'un projet
    - Degré d'autonomie des micro-services
    - Organisation de la coordination entre micro-services
  - Techniques
    - Services et infrastructure nécessaire
    - Intégration au gestionnaire de sécurité Keycloak
    - Format de packaging des projet
    - Identification des pipelines CI/CD

Ce projet a comporté différentes phases :

- Identification de la problématique métier, identification des UserStory
- Analyse et décomposition en micro-service
- Pour chaque micro-service :
  - Définition et publication des APIs
  - Implémentation et Tests des API
  - Intégration Service de discovery Eureka et configuration centralisée
  - Implémentation / Tests des interaction entre microservies
  - Intégration de keycloak pour l'authentification et l'autorisation

Décomposition en micro-services

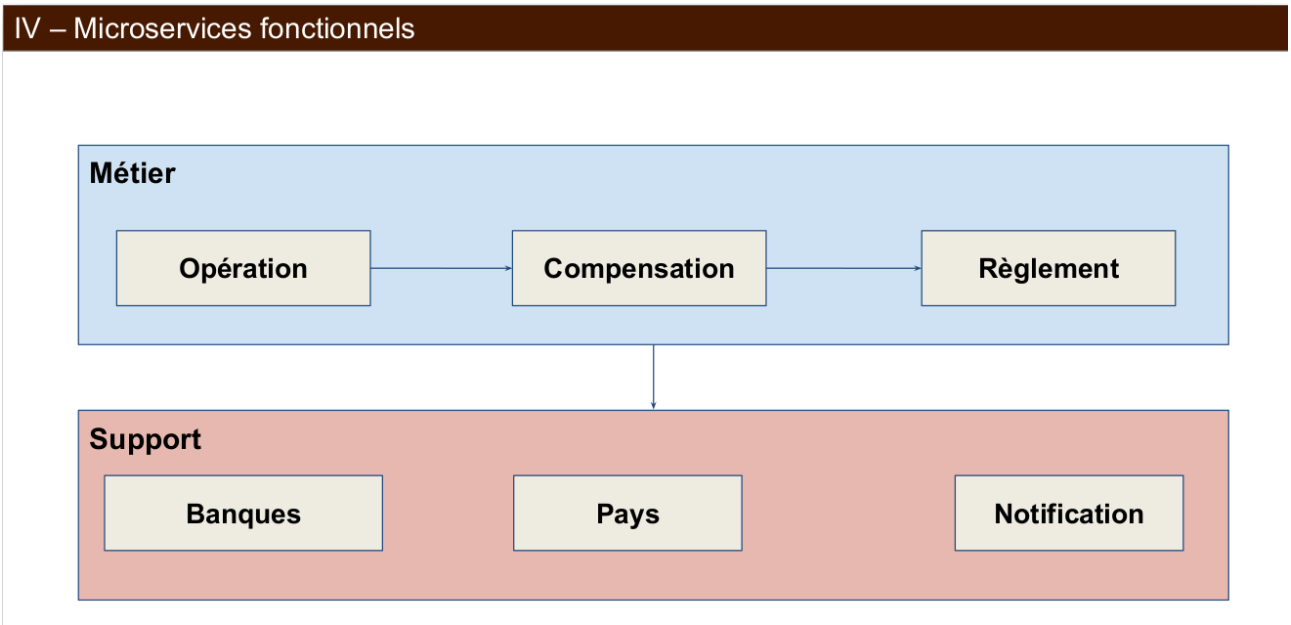
Une analyse préalable effectuée par les agents BCEAO ont permis de proposer une décomposition en micro-services de la problématique lié au logiciel SICA.

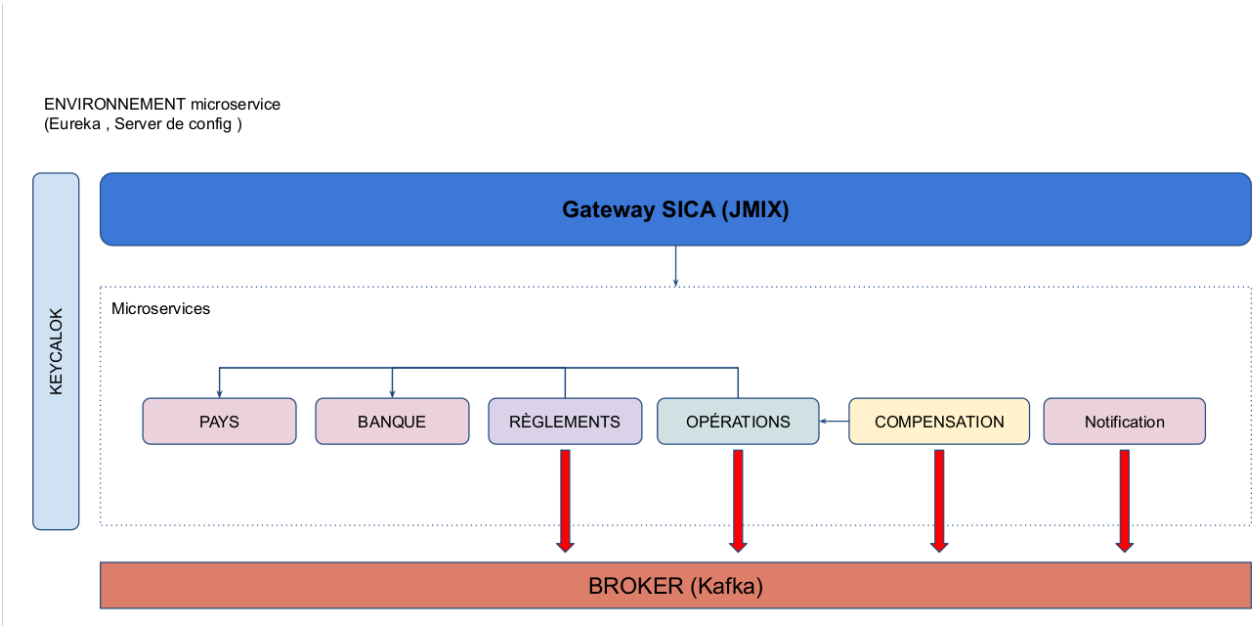
Les user stories visées étaient :

- Un utilisateur doit pouvoir accéder à la liste des journées de compensation, ouvrir ou fermer une journée
- Un utilisateur doit pouvoir accéder à la liste des opérations d'une journée.
- Un utilisateur doit pouvoir accéder à la liste des règlements d'une journée.

La décomposition a identifié :

- 3 micro-services de support :
  - Gestion des Banques,
  - Gestion des Pays
  - Service de notification
- 3 micro-servies métier :
  - Opérations : Traitement des entrées en provenance des banques
  - Compensation : Service de compensation pour une journée
  - Règlement : Service de crédit/débit de compte
- 1 application Gateway proposant une interface utilisateur permettant les User Stories et s’appuyant sur les précédents miro-services





Bilan de la décomposition

La décomposition semble adéquate et n’a pas provoqué trop de dépendances entre micro-services. Chaque micro-service a pu évoluer de façon relativement indépendante.

Organisation des équipes

Chaque équipe est organisée autour d'un projet Gitlab appartenant à la même organisation.
Un projet config contient les fichiers partagés par tous les micro-services (propriétés de configuration de Kafka, Eureka, …)

Organisation : https://gitlab.com/sdi-poc/sica-microservices

Les agents BCEAO ont le profil développeur, David THIBAU est mainteneur de projet.

La collaboration s'effectue comme suit :

- Des issues sont créées par David THIBAU, décrivant le travail à réaliser
- Un membre de l'équipe créé une merge request à partir de l'issue lorsqu'il prend en main la tâche. Une branche du nom de l'issue est créée sur le dépôt distant
- Le développeur récupère la branche dans son environnement et y effectue un ensemble de commit
- Lorsque la tâche est prête, il le signale sur le site du projet en activant le lien Mark As Ready
- David THIBAU effectue une Revue de code et accepte ou nom la merge request
- Lorsque la Merge Request est acceptée, la branche associée est intégrée dans la branche principale main.

Bilan de la collaboration

- La procédure de collaboration décrite ci dessus n'a pas toujours été respecté ; ce qui a pu apporter un peu de confusion sur le projet.
- Les fonctionnalités de collaboration offertes par Gitlab ont été peu exploitées.
- Les problématiques de fusion de branche et de résolution de conflit n'ont pas toujours été prises en charge par les développeurs.

Progression des projets

Les étapes de développement consistaient en :

1. Définition et publication de l'API
2. Implémentation et test des fonctionnalités sans les interactions
3. Intégration Eureka et Configuration centralisée
4. Implémentation des interactions entre les micro-services (REST ou Message Kafka)
5. Intégration Keycloak et mise en place de la sécurité

	API	Implémentation/ Test API	Intégration Eureka et config centralisée	Implémentation / Test des interactions	Sécurité
Banque					
Compensation					
Gateway-sica					
Operation					
Pays				Non applicable	
Règlement					

Bilan et problèmes rencontrés

<u>API :</u>

- Les échanges par messages n'ont pas été spécifiés (On aurait pu utiliser AsyncAPI),
- Les échanges REST ont eux été spécifié par OpenAPI et une interface Swagger a été générée.
- Il n'y avait pas de pont central dans l'organisation de publication des APIs
- Pas de gestion de version dans l'API

<u>Implémentation et test</u>

- L'équipe n'a pas l'habitude d'écrire des tests et connaît peu les librairies et mécanisme SpringBoot utilisés (MockMvc, JsonPath, @Transactional)
- De nombrux tests avaient des dépendances de séquencement entre eux.
- Le support de persistance n'était pas initialisé dans un état connu. Les projets qui n'utilisaient Hibernate et H2 ont eu plus de difficulté à initialiser la base avant les tests

<u>Intégration Eureka et config centralisée</u>

- 2 mécanismes différents ont été utilisés :
  - Projet non jmix : Utilisation du starter bootstrap et du fichier bootstrap.yml
  - Projets Jmix : Jmix génère des fichiers de configuration au format .properties. Donc beaucoup de propriétés ont été positionnés au niveau local et non pas au niveau du serveur de configuration
    => Pas une vision claire, de quelle propriétés sont générés par l'outil et quelles autres sont gérées de façon centralisée dans le serveur de configuration
- Le mécanisme de merge request pour modifier les configurations d'un projet est trop lourd

<u>Implémentation et test des interactions</u>

- Les interactions avec le broker Kafka ont été les plus facile sà tester. Un fichier docker-compose comprenant Config, Eureka, Kafka, Redpanda et les services concernés est mis au

point. Il doit être lancé avant l'exécution des tests soit manuellement, soit avec le starter *docker-compose*.

- La SAGA Operation → Compensation → Règlement → Compensation n'a pas pu être testé
- Les interactions REST ont été plus difficiles à tester. Cela a nécessité la mise au point d'un docker-compose pour démarrer Config et Eureka, un démarrage manuelle du service appelé (soit après récupération des source, soit en utilisant une image docker, soit en générant un service à partir de sa définition OpenAPI). La disparité des mécanismes rend difficile l'automatisation dans le cadre d'une pipeline DevOps.
  C'est l'un des points les plus importants à résoudre lors des prochains incréments.

Intégration Keycloak et mise en place de la sécurité

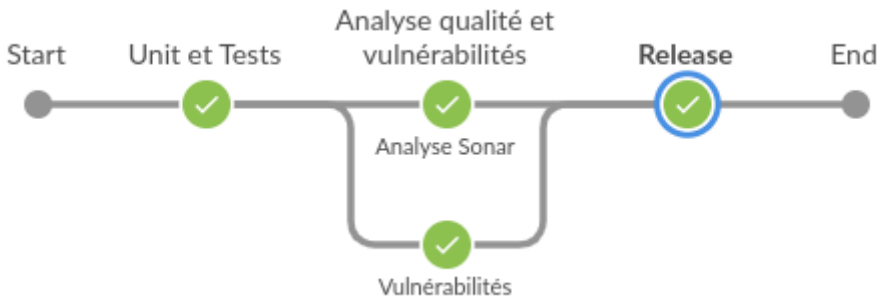
- Seul le projet règlement est arrivé à intégrer Keycloak pour protéger sa couche Web et Service. Les mécanismes sont fonctionnesls et pourraient être affinés
- Les aspects authentification utilisateur final n'ont pas pu être abordés
- Les aspects propagation de jeton ou obtention de jeton par client crédentiels n'ont pas pu être abordés

Approche continue

Des pipelines CI/CD ont été mis en place. Au début, 2 environnements étaient disponibles (Gitlab CI et Jenkins), l'environnement Gitlab s'est arrêté après le 1er jour car il nécessitait des achats. L'environnement Jenkins n'était malheureusement pas accessible des agents

La pipeline était constitué de 3 étapes :

- Exécution et publication des résultats de tests
- Analyse statique comprenant :
  - Analyse qualité Sonar
  - Analyse OWASP des vulnérabilités des dépendances
- Étape de publication de release au format Docker



Bilan

Phase de tests

- Les tests n'ont pas eu être maintenus dans un état opérationnel jusqu'à la fin du poc.
- Après la phase d'intégration de Config et Eureka, les tests nécessitaient le démarrage d'un docker-compose. Ce qui rendait difficile, l'exécution simultanée de pipeline dans l'infrastructure minimale que l'on disposait.
- Le fait de ne pas disposer d'un environnement d'intégration ou les micro-services auraient pu être déployés nous a empêché des tester les interactions entre micro-services en live

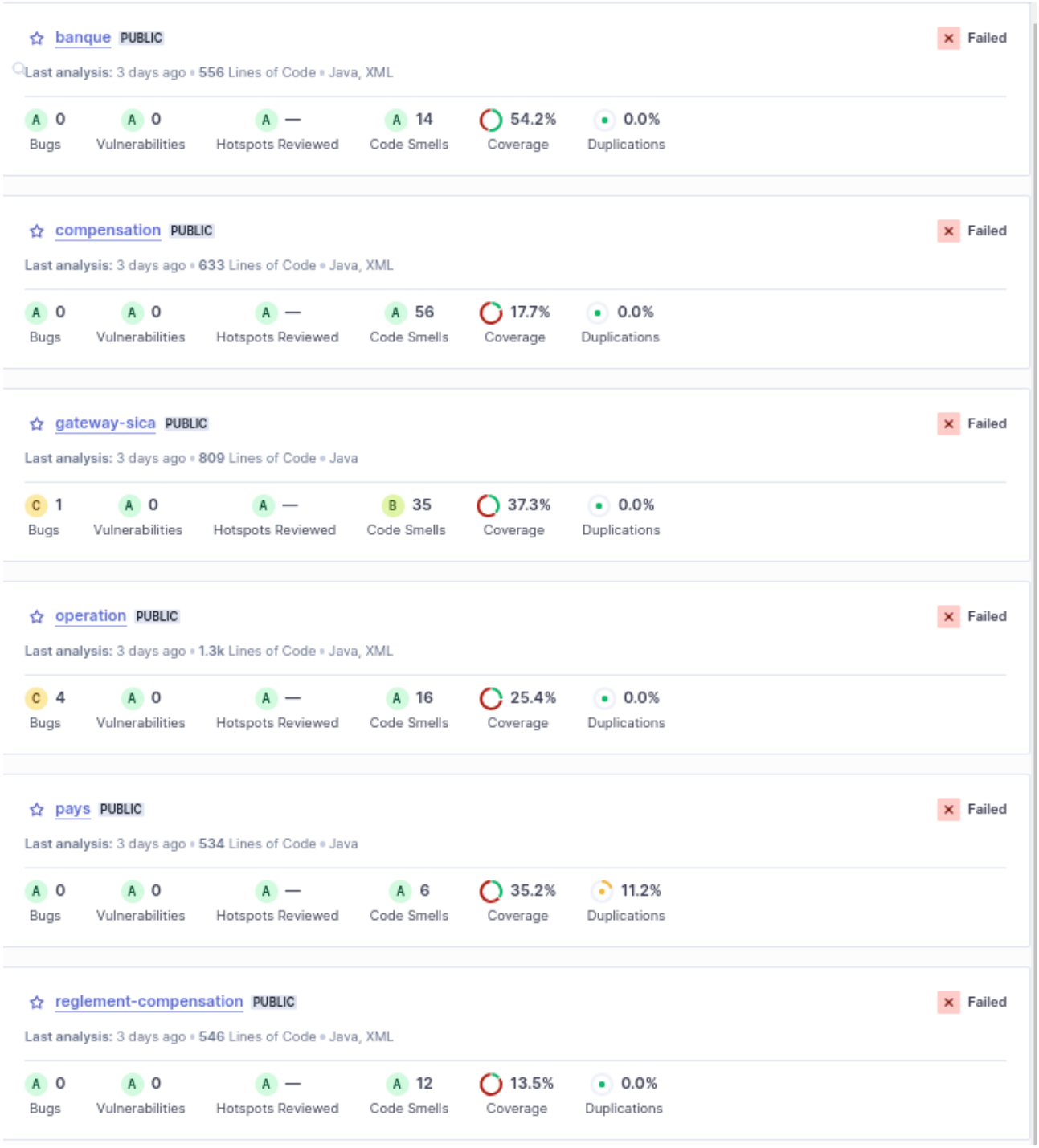
= > Recommandation : Séparer les tests en autonomie et les tests d'intégration nécessitant une plateforme de déploiement des micro-services

Analyse qualité

- Les critères habituels de qualité ont du être abaissés afin certains projets puissent passer les portes qualité
- Les agent n'avaient pas facilement accès aux résultats de l'analyse et n'ont donc pas été sensibilisés à la qualité du code

=> Recommandation : Installer des plugins assistance à la qualité comme SonarLint dans les IDEs. Avoir accès aux résultats des analyses Sonar

Résultats analyses :



Analyse vulnérabilités

- Un rapport a été généré pour les projets Jmix mais les données n’ont pas été exploitées

Release

- Les n° de versions n’ont pas été gérés
- Le projet Gateway n’a pas pu être packagé au format Docker

Conclusion

Tous les objectifs initiaux du poc n’ont pas été atteint, en particulier l’intégration avec le serveur Keycloak.

Cependant, les 6 jours ont permis :

- d’avoir une vision plus claire des problématiques liés à cette migration
- de valider la faisabilité technique de l’intégration de Jmix dans une architecture micro-services
- de faire monter en compétences les agents bceao sur les problématiques liés à cette évolution : Configuration centralisée, interactions en micro-services, tests, approche DevOps, utilisation d’images Docker
- de valider la décomposition fonctionnelle préalable

Il serait intéresser de pouvoir continuer cette expérience afin d’aborder les problématiques liés à la sécurité.

Dans le cas, d’un 2ème Sprint sur cette méthodique les points suivants devront être corrigés :

- Mise à disposition d’une infrastructure d’intégration permettant le déploiement des micro-services
- Mise à disposition d’une plateforme d’intégration continue incluant les analyses qualités
- Redéfinition d’une pipeline CI/CD distinguant les tests unitaires ou de composants (test d’un micro-service en isolation) des tests d’intégration ou end 2 end nécessitant un environnement de développement