État des lieux et recommandations

Table des matières

I. État des lieux

I.I Parcs applicatifs

Types d'applications

Applications à destination des employés internes à la BCEAO (8 pays), quelques unes ouvertes sur l'extérieur

- Applications Web monolithique construites au dessus d'un socle technique maison déployées sous forme de *war* sur des serveurs Tomcat
- Jobs schédulés par du cron construit au dessus de *Talend/Mule*
  - Synchronisation entre applications (BD vers BD)
  - Importation de données externes
- Application mobile en cours de définition et développement

Socle technique

Construit au dessus des framework Spring4 et ExtJs et d'une BD relationnelle (Oracle, MySQL, Postgresql)
Langage Java : 1.7, 1.8, 11

2 versions du socle existantes qui apportent :

- Composant d'authentification, 3 configuration possible :
  - Realm bd : Utilisé pour le développement
  - Realm Ldap : Solution legacy
  - Realm Keycloak : En cours d'intégration dans les application
- Modèle d'autorisation ACL
  - Gérer par un processus d'approbation d'affectation de rôle et l'application métier RAFIA.
    Les applications se synchronisent régulièrement à RAFIA via des batch et mettent à jour leur propre modèle d'ACL.
- Classes du domaine (Structure hiérarchique banque)
- Reporting : facilité de déploiement des rapports conçus avec JasperReport
- Workflow : Machine à état permettant de décrire les statuts et transition d'un dossier
- Génération d'interface utilisateur avec du code ExtJs
- Interface utilisateur pour le monitoring et le déclenchement de Job

<u>Retours :</u>

- Pour certains projet, nécessité de surcharger le modèle de domaine
- Pour certains projet, certaines fonctionnalités ne sont pas utilisées, le modèle de domaine n'est pas repris dans son intégralité
- Difficile de faire évoluer le socle car cela touche tous les projets en production

A l'heure actuelle, l'utilisation de ce socle est en grand frein à la migration vers des technologies plus modernes

Évolutions envisagées

- Spring Boot 3 (Java 17, 21)
- JMIX (SpringBoot + Vaadin), achat en cours de Licenses
- Angular

I.2 Organisation des équipes

Le service est divisé en 5 pôles :

- Statistiques
- SIB (Système d'informations bancaires)
- Comptabilité et administration
- RH
- DevOps : Équipe transverse qui s'occupe de l'environnement d'intégration et des déploiements

Dans chaque, ce sont des profils développeur sans plus de distinction.

Les membres de chaque pôle sont des contractuels BCEAO ou des personnes en régie

Chaque pôle, autre que DevOps, est responsable d'un portefeuille d'applications.
Le nombre d'applications vis à vis du nombre de membre du pôle fait qu'un seul développeur est responsable et a la connaissance de une ou plusieurs applications

Pour chaque application, il y a également 1 responsable métier, 1 responsable admin pour la mise en production.

Chaque développeur est évalué avec un indicateur de performance

<u>Retours :</u>

- Le travail de développeur est souvent interrompu par des urgences :
  - Correction de bug
  - Spécifications fonctionnelles peu précises (ajustement de la fonctionnalité)
  - Décisions politiques urgentes
    => **Il est donc difficile de rester concentrer sur de longues tâches, grosse évolution, migration**
- La connaissance d'une application est peu partagée. L'absence d'un développeur peut générer des retards car lui seul connaît les applications dont il a la charge
- Peu de documentation fonctionnelle.
  Le cahier de charges initial évolue et les évolutions ne sont pas tracées

I.3 Gestion de projet

Pilotage

Lors du démarrage d'un projet des réunions d'échange avec le métier permettent de définir le cahier de charge initial au format bureautique

Il n'y a pas de documentation technique produite :

- Architecture

- Modèle de données
- User Story

Pas de méthodologie formalisée :

- Scrum, Agile
- Planning, Sprint, jalons, point de contrôle, suivi d'avancement

Pas d'outil de pilotage projet : chaque projet s'organise de façon différente : Excel, PowerPoint, Google Drive

Gestion des sources

Les sources sont publiés dans des dépôts Git gérés par Git EA mais les fonctionnalités de collaboration de l'outil sont inutilisées (Issues, Pull Request, Milestone)

L'équipe a une bonne pratique de git et est capable de faire des opérations avancées (*merge, rebase, cherry-pick*)

Il n'y pas de workflow de collaboration défini autour de notion comme les Merge ou Pull Request.

Chaque projet a une utilisation des branches Git différentes

I.4 Environnement de développement

2 IDEs utilisés :
- Eclipse
- Intellij version ultimate envisagé

Plugins d'assistance de code :
- AI : Copilot envisagé / IA assistance
- SonarLint : Peu utilisé car beaucoup de bruit sur projet existant

Services de support :
Pour travailler sur un projet, le développeur doit au préalable installer localement les services de support (comme une BD par exemple)

I.5 Intégration continue

La responsabilité de l'équipe s'arrête à l'environnement d'intégration et la production d'un artefact dans le dépôt Archiva. L'équipe d'infra reprend le relais pour déployer les artefacts dans les environnements de recette et de production

Un changement de code (nouvelle fonctionnalité, évolution, correction de bug) suit le cycle suivant :

- Le code est développé localement
- Il est ensuite déployé sur un serveur d'intégration provisionné par l'équipe DevOps. Le serveur est partagé par plusieurs application.
  Le déploiement s'effectue en exécutant un script manuellement
- Des tests de validation sont effectués manuellement par le développeur
- Un job Jenkins permet de construire l'application et de stocker l'artefact dans le dépôt Archiva. Un version SNAPSHOT s'ajoute aux précédents artefacts
- Lorsque les tests sont concluants, l'équipe Infra récupère le dernier SNAPSHOT et le déploie sur un environnement de recette accédé et validé par les utilisateurs finaux.

La configuration applicative est gérée par un fichier properties externe à l'artefact qui est fourni par les développeurs.
Certaines propriétés sont surchargées

- Par l'équipe DevOps pour l'environnement d'intégration
- Par l'équipe Infra pour l'environnement de production

Qualité

Pas d'analyse statique de code

Pas de revue de code

Pas d'analyse de vulnérabilités si ce n'est que régulièrement une équipe dédiée tente de détecter des vulnérabilités par des analyses statiques ou des analyses dynamiques avec l'outil Burp

Releasing

Pas de gestion de version, Pas de processus de release formalisée, les versions SNAPSHOTS sont utilisées pour le déploiement

Tests d'intégration et tests de recette

Pas d'équipe de tests dédié

Pas de tests automatisés

Tests d'intégration effectués par le développeur

Test de recette effectués par le métier

Maintenance applicative

Outil de gestion d'incidents EasyVista

Des incidents sont souvent reportés par l’équipe métier via de simple mail

Observabilité

Aucun retour sur la prod, si ce n’est de temps à temps des fichiers de logs

II. Recommandations

Augmenter la qualité

Beaucoup de bugs relevés ou d’incidents en prod (application en rade, BD surchargée, …), comment diminuer ce type d’incidents

Adopter une approche TDD

Instaurer des revue de code (1 mainteneur + 1 ou plusieurs développeurs)

Répliquer l’environnement de production dans un environnement de staging y effectuer des tests d’utilisation

Se doter de plugins d’aide au développement SonarLint, Copilot

Augmenter la collaboration et la documentation

Définir des workflows de collaboration autour des branches pour chaque projet

Se douter d’outils permettant la collaboration entre les participants d’un projet : développeurs, métier, infra

Gérer l’historique du projet avec des Release Notes identifiant les apports de chaque déploiement

Avoir des plannings revus en fonction des événements, prévoir des sécurité pour faire face aux imprévus

Mettre en place des cérémonies

- Stand up meetings
- Présentation des projets aux collaborateurs

Favoriser l’adoption des nouvelles technologies, rester à jour

Adopter architecture micro-services afin de permettre une meilleure évolutivité et une centralisation des fonctionnalités métier

Pousser l’infra afin de bénéficier des apports des orchestrateurs de conteneurs

Progressivement, améliorer les pipeline d’intégration continue

Intégrer des tests unitaires et d’intégration automatisé

Automatiser les déploiements en intégration, renforcer l ‘étanchéïté entre applications

Migrer vers des déploiements jar avec serveur embarqué plutôt que war Infrastructure existante

Automatiser la production de release et de release notes, déployer dans archiva des release plutôt que des snapshots

Améliorer les synchronisations entre applicatifs

Synchronisation temps réel via message broker plutôt que par des batchs de synchronisation

Améliorer l’observabilité des applications en production

Actuator, prometheus et Grafana