Etats des lieux et recommandations

Table des matières

Etat des lieux

Parcs applicatifs

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
  - BD vers BD permettant la synchronisation entre applications
  - 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
- Modèle d'autorisation ACL
- Classes du domaine (Structure hiérarchique banque)
- Reporting facilité de déploiement des rapports ou édition avec JasperReport
- Workflow : Machine à état permettant de décrire les statuts et transition d'un dossier
- Génération de code ExtJs
- Interface utilisateur pour le monitoring et le déclenchement de Job

Retours :

- 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

Autres technologies envisagées :

- Angular
- JMIX (SpringBoot + Vaadin)

Organisation des équipes

Le service est divisé en 4 pôles :

- Statistiques
- SIB (Sysytème d'informations bancaires)
- Comptabilité
- 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 que 1 développeur est responsable de 1 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

Retours :

- Le travail de développeur est souvent interrompu par des urgences :
  - Correction de bug
  - Spécifications fonctionnelles peu précises (ajustement de la fonctionnalité)
  - Décisions politiques

=> Il est donc difficile de rester concentrer sur de longues tâches, grosse évolution, migration

Pilotage projet

Démarrage de projet avec réunions d'échange avec le métier.

Éventuellement, rédaction d'un cahier des charges initial au format bureautique

Planning ? point de contrôle ? Méthodologie pas formalisée

Pas d'outil de gestion des évolutions, chaque projet s'organise (Excel, PowerPoint, Google Drive)

Gestion des sources

Dépôts de référence géré par Git EA mais fonctionnalités de collaboration inutilisées (Issues, Pull Request, Milestone)

Bonne pratique de git

Pas de workflow de collaboration définie.

Utilisation de branches différentes pour chaque projet

Environnement de développement

IDEs utilisés :

- Eclipse
- Intellij version ultimate envisagé

Plugins :

- Equivalence 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 installer localement son service de support (BD)
2 config possible , ; bd, ldap, keycloak

Intégration continue

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 un dépôt Archiva. Un version SNAPSHOT s'ajoute aux précédents artefacts
- L'équipe Infra récupère le dernier SNAPSHOT et le déploie sur un environnement de recette accédé et validé par les utilisateurs finaux.

Fichier properties externe au jar
Fourni par l'équipe de dév, surchargé par devos et par la production

Qualité

Pas d'analyse statique de code

Pas de revue de code

Pas d'analyse de vulnérabilités

Régulièrement une équipe de détection de vulnérabilités dédié effectue des analyse statique et dynamique des applications déployées 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

Mail d’incidents

Observabilité

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

Recommandations

Augmenter la qualité

Beacoup de bugs relevés

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

Adopter une approche TDD

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

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 particpants d’un projet développeeur, métiers, 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 qux imprévus

Mettre en place des cérémonies

Stand up meetings

Présentation des projets

Favoriser l’adoption des nouvelles technologies, rester à jour

Adopter architecture micro-services

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 :

Infrastructure existante

Automatiser la production de release et de release notes

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