# Introduction

Ce document présente une proposition de feuille de route pour la mise en place d'un processus de développement et de déploiement industrialisé, inspiré de la méthodologie Devops.

Ce plan est applicable dans le cadre d'une startup, et peut être mis en place de manière progressive.

Contexte technologique :

- Applications écrites en Java avec le framework Spring
- Base de données NoSQL MongoDB

# Objectifs

- **Développer avec efficience** : selon le besoin utilisateur, des systèmes extensibles mais en évitant l’ « over-engineering »

- **Être réactif aux évolutions** : pouvoir déployer souvent, périodiquement ou sur demande

- **Développer « en confiance »**, sans crainte de détériorer l’existant, pour améliorer la productivité

# Étapes de mise en place

- Choix d’un fonctionnement organique de développement
- Choix d’un écosystème technologique
- Mise en place d’un système de gestion de versions
- Création d’infrastructures logicielles
- Élaboration de processus de contrôles qualité automatisés
- Élaboration de processus de déploiement automatisés et répétables
- Création d’un environnement de développement robuste et extensible
- Mise en place d’une solution de recueil de retours d’expérience et de support
- Mise en place d’une solution de surveillance d’infrastructure

# Fonctionnement organique

**Objectifs :**

- Prendre en compte les besoins et les retours d’expérience au plus tôt
- Développer en petites itérations, pour éviter l’effet « tunnel »
- Communiquer sur les avancées du projet et se concerter sur la direction à suivre

# Fonctionnement organique

**Propositions :**

Utilisation de la méthodologie Agile avec sprints de deux semaines, et toutes les deux semaines:

- Démonstration des avancées, recueil des réactions :  ~1h
- Sprint review : bilan du sprint écoulé ~1h
- Sprint planning : planification et découpage des tâches à venir ~2h

# Écosystème technologique

Composants du système où est développé et déployé le produit final.

**Cet écosystème doit :**

- Permettre une autonomie totale : possibilité de support et de mise en production externalisés, mais possibilité de déploiement en interne au besoin
- Répondre aux besoins de l'entreprise et de ses clients : coûts d'entretien, compétences actuelles, confidentialité, ...
- Être fiable et sécurisé
- Être évolutif, doit supporter l'utilisation de technologies hétérogènes
- Doit pouvoir être installé et manipulé de manière automatisable
- Être documenté et de préférence populaire

# Écosystème technologique

**Propositions architecturales:**

- Utilisation au maximum de systèmes Open Source ou libres
- Architecture en micro-services :
  - Système résilient et « scalable »
  - Compatible toutes technologies
- Automatisation de déploiement avec processus d’ « Infrastructure as Code » et automatisation de configuration
  - Outils de Cloud Providers ou Terraform
  - Ansible

# Écosystème technologique

**Propositions système:**

- Ubuntu Server LTS en tant qu'OS serveur : support commercial disponible et sous licence libre, bien documenté, largement supporté et sécurisé
- Ubuntu Desktop LTS en tant qu'OS de développement, pour être au plus proche des conditions de production
- Conteneurisation Docker pour les applications : permet d'utiliser tous types de technologies sur une même machine
- Kubernetes en tant qu'orchestrateur de conteneurs : permet de gérer un grand nombre d'applications et leurs interactions

# Gestion de versions

Choix d’un service de gestion de versions disponible avec interface web.

**Fonctionnalités attendues :**

- Création de dépôts privés sans limitations
- Interface de gestion des modifications et de partage autour de ces modifications sous forme de Merge requests
- Possibilité de déclenchement d’actions automatisées : pipelines, jobs périodiques, etc …
- Possibilité d’héberger la plateforme

# Gestion de versions

Importance du système de **Merge Request** pour augmenter la qualité du code produit:

*Un développeur soumet des modifications, elles doivent être validées par un autre développeur*

Permet d'éviter les erreurs, et permet à tous de donner un avis sur une modification

*Mais surtout permet à tous d'apprendre des modifications en cours, sur l'évolution du système et sur les meilleures pratiques à mettre en œuvre*

# Gestion de versions

**Propositions :**

Gitlab CE, hébergé en interne ou externalisé :

- Bonne interface de gestion des modifications de code (Merge Request)
- Bonnes possibilités d’intégration continue / de déploiement continu
- Système de permissions élaboré

# Infrastructures logicielles

**Besoins:**

- Avoir une infrastructure à destination des utilisateurs finaux:
  - Exécutant un code stable et fiable
  - « Scalable » et sans point faibles (Single Point of Failure)
- Avoir une ou plusieurs infrastructures où :
  - Les développeurs peuvent tester les nouvelles fonctionnalités
  - Le code peut être testé de manière automatisée avant déploiement

# Infrastructures logicielles

**Propositions :**

- Création de deux infrastructures distinctes :
  - Une infra de développement, avec déploiement automatisé à chaque modification du code source principal
  - Une infra de production avec déploiement suite à intervention humaine toutes les deux semaines
- Hébergement:
  - Utilisation d'IaaS ou d'un prestataire traditionnel pour l'infra de production
  - Auto hébergement de l'infra de dev pour économie de coûts
- Utilisation exclusive de logiciels « clusterisables », même si ils ne sont pas clusterisés dans un premier temps

# Contrôles qualité

**Objectifs :**

Exécuter des tests pour assurer le fonctionnement du code et éviter les régressions :

- Tests unitaires : testent le fonctionnement de petites portions de code
- Tests hauts niveau (intégration, système, « end to end ») : testent le fonctionnement de l'application

Auditer le code régulièrement pour consultation:

- Analyse de couverture de tests
- Analyse de qualité de code

Les clefs de la réussite de ces contrôles sont:

- Création de rapports clairs et disponibilité d'outils de suivi
- Automatisation de toutes les étapes
- Fiabilité d'exécution
- Rapidité, dans la mesure du possible

# Contrôles qualité

**Propositions :**

- Mise en place d'un pipeline Gitlab d'exécution de tests unitaires, étiquetés comme tels avec Junit 5. Ce pipeline est exécuté automatiquement avant chaque fusion dans le code principal, et bloque la fusion si les tests échouent.
- Exécution de tests de haut niveau, exécutés avec le code source principal au moins une fois avant chaque déploiement en production :
  - Tests d'intégration étiquetés comme tels avec Junit 5
  - Tests end-to-end avec Cypress (alternative à Sélénium)
- Audit de code:
  - Analyse de couverture de tests unitaires avec Jacoco, rapports exportés à chaque exécution des tests
  - Analyse de code avec Sonarqube après chaque modification du code principal, mise en évidence de mauvaises pratiques et failles potentielles

# Déploiement automatisé

**Objectifs :**

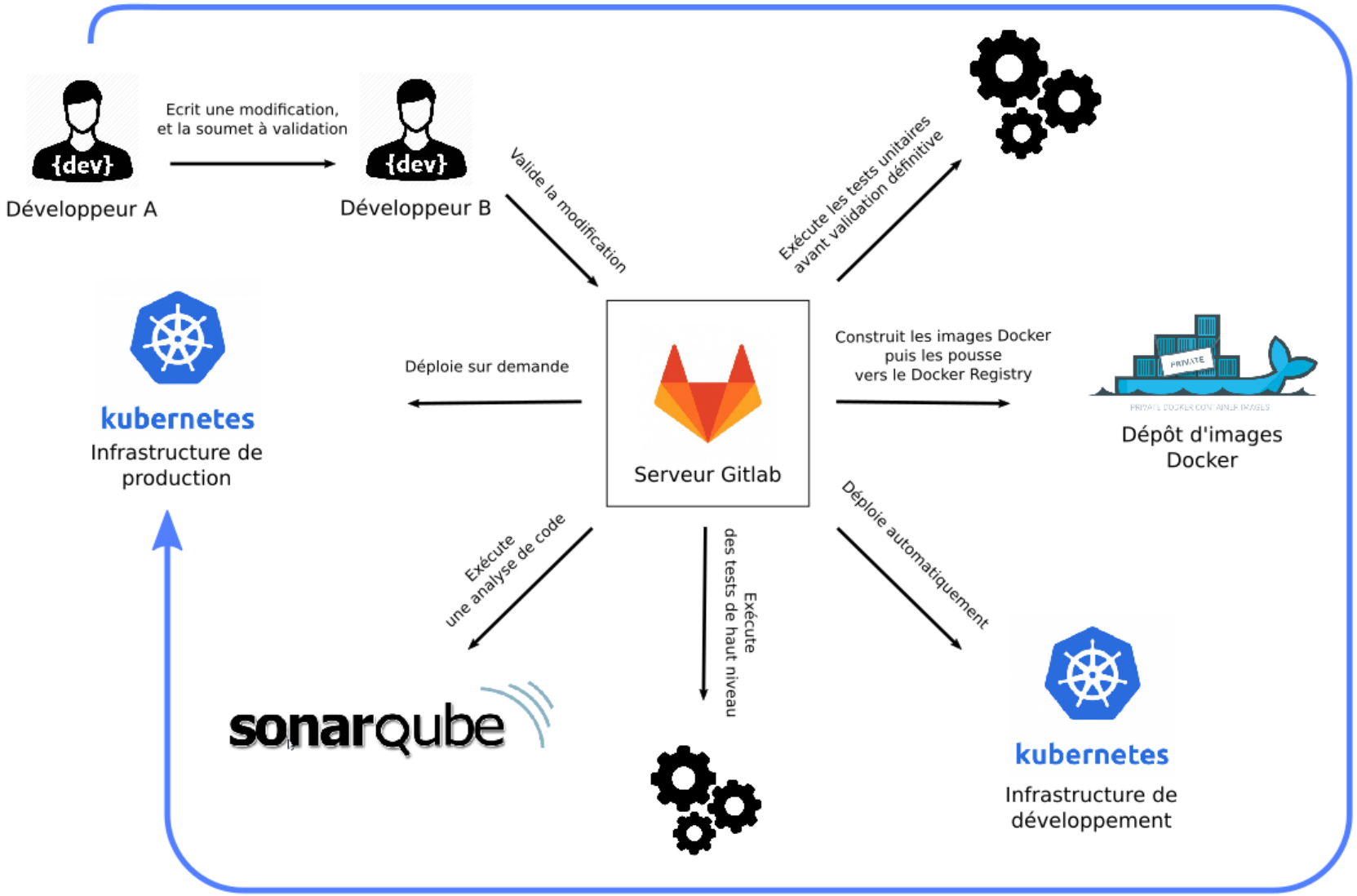
- Déployer rapidement les nouvelles fonctionnalités, de manière fiable, périodiquement et à la demande
- Avoir la possibilité de réparer et de redéployer rapidement du code en production en cas de problème
- Déployer sur plusieurs environnements distincts, de manière extensible

# Déploiement automatisé

**Proposition:**

- Déployer les applications sous forme de charts Helm (Kubernetes)
- Construire un pipeline Gitlab de déploiement basé sur des projets Git et des branches de code:
  - Déploiement par projet Git
  - Une branche de code par environnement : dev, prod/env1, prod/env2, ...
  - Déploiement automatisé sur l'infra de dev, déclenché à chaque mise à jour du code principal
  - Déploiement après confirmation humaine sur l'infra de production
  - Possibilité d'ajouter d'autres environnements au besoin en ajoutant d'autres branches Git
  - Possibilité de déployer plusieurs projets indépendamment
- Toutes les étapes de déploiement font l'objet d'un rapport succinct dans un chat ou par mail

# Cycle de déploiement en CI/CD (se lit dans le sens des aiguilles d'une montre)



# Environnement de développement

Permet aux développeurs de simuler l'infrastructure de production en local. Conditionne en grande partie la productivité des développeurs.

**Caractéristiques souhaitées :**

- Similaire à l'infrastructure de production
- Rapide et debuggable
- Simple à installer
- Simple à comprendre et à modifier
- Utilisable en fonctionnement automatisé, pour les pipelines de test de haut niveau par exemple

# Environnement de développement

**Propositions :**

- Ubuntu Desktop LTS comme OS de développement, accessible et plus proche des conditions de déploiement en production
- Minikube ou Docker Compose comme support système des applications et services
- Mise en place de ports de debug JDWP
- Outil d'aide au développement maison « batteries-included »
  - Installation fiable et reproductible de l'environnement de développement
  - Aide à la reconstruction et au redémarrage de l'environnement, ainsi qu'à l'initialisation de jeux de données
  - Aide aux tests et au déploiement local

# Estimation du temps de mise en place

Temps estimé de mise en place des composants, dans le cas où ils sont tous hébergés (chaque déploiement faisant l'objet d'un playbook Ansible*) :

- Serveur Gitlab : déployé en Docker Compose, certificats TLS, création de comptes : 3 jours
- Docker registry : déployé en Docker compose, certificats TLS, authentification : 3 jours
- Cluster Kubernetes de développement, certificats TLS et proxy : de 3 à 4 jours
- Mise en place du pipeline de tests unitaires: 3 jours
- Mise en place du pipeline de déploiement en dev: 5 jours pour la V1
- Cluster Kubernetes de production : 2 jours
- Serveur Sonarqube : 3 jours
- Mise en place du pipeline et des méthodologies de tests de haut niveau : 5 jours pour la V1
- Mise en place de l'environnement de développement local : 5 jours pour la V1

(*L'utilisation d'Ansible permet d'automatiser la configuration de serveurs, par exemple pour restaurer l'état d'un composant rapidement après un accident)

# Parties non traitées

Deux parties nécessitent plus de recherches pour trouver des plateformes open source et manipulables par de petites équipes :

- Monitoring d'infrastructure : la plupart des plateformes nécessitant beaucoup d'investissement en temps (ex : Nagios/Centreon, Prométheus, ...)
- Recueil de retours d'expérience, support et planning (JIRA?)