Rapport de SF - DEVOPS Partie 2

Dans cette deuxième partie de la session de formation DevOps, les participants ont abordé des sujets tels que les tests unitaires et fonctionnels, les exigences de sécurité, les outils DevOps et les stratégies de branching. Les discussions ont porté sur l'importance des tests pour garantir la qualité du code, les pratiques de sécurité pour protéger les applications et les données, et les outils et méthodes pour automatiser les processus de développement et de déploiement.

Les points important qui ont été abordés sont les suivants :

- **Compte Développeur iOS** :
    - **Importance** : Posséder un compte développeur iOS est crucial pour toute personne souhaitant publier des applications sur l'App Store d'Apple. Ce compte donne accès aux outils de développement d'Apple, à des ressources techniques et à la possibilité de tester et de distribuer des applications à un public mondial.
    - **Expérience Pratique** : En essayant de publier une application, les développeurs acquièrent une compréhension des exigences de la plateforme, des procédures de soumission, et des standards de qualité imposés par Apple.
- **Framework DevOps pour Tests Unitaires Flutter** :
    - **DevOps** : DevOps est une combinaison de pratiques, de méthodologies et d'outils qui augmentent la capacité d' une organisation à fournir des applications et des services à un rythme élevé.
    - **Tests Unitaires** : Les tests unitaires sont des tests automatisés qui vérifient le bon fonctionnement des plus petites unités de code, souvent des fonctions ou des méthodes. Pour Flutter, des frameworks comme `flutter_test` permettent de créer et exécuter ces tests facilement.
    - **Avantages** : Les tests unitaires permettent de détecter les bugs dès le début du cycle de développement, ce qui réduit les coûts de correction et améliore la qualité du code.
- **Tests Fonctionnels Flutter** :
    - **End-to-End Testing** : Ce type de test vérifie le fonctionnement complet d'une application, de l'interface utilisateur à la base de données. Il simule les interactions des utilisateurs et valide les flux de travail.
    - **Multiplateforme** : Exécuter les tests sur plusieurs navigateurs et environnements permet de s'assurer que l' application fonctionne correctement partout, ce qui est crucial pour une bonne expérience utilisateur.
    - **Outils** : Utilisation d'outils comme `flutter_driver` ou `integration_test` pour écrire et exécuter des tests end-to-end.

Azure Koloka

- **User Story**
    - **Définition** : Une user story est une description simple d'une fonctionnalité du point de vue de l'utilisateur final. Elle est utilisée dans les méthodes agiles pour capturer les besoins de l'utilisateur.
    - **Approche Utilisateur** : Toujours se mettre à la place de celui qui va recevoir la fonctionnalité pour mieux comprendre ses besoins et attentes.
    - **Sprint Planning** : Processus de planification dans lequel l'équipe décompose les user stories en tâches techniques et les planifie pour un sprint spécifique. Cela aide à définir clairement les objectifs et à allouer les ressources efficacement.
- **Estimation des Story Points**
    - **Peer-Programming** : Estimation collaborative où deux développeurs travaillent ensemble pour évaluer les user stories. Bien qu'efficace pour les petites tâches, cette méthode peut être limitée pour les grands projets.
    - **Planning Poker** : Méthode où chaque membre de l'équipe utilise des cartes pour proposer une estimation. Les différences d'estimation sont discutées pour atteindre un consensus.
    - **"Ideal Day"** : Concept où une journée idéale est utilisée comme unité de mesure, sans interruptions. Cela aide à établir une référence pour les estimations futures.
    - **Division des Sous-Tâches** : Diviser les user stories en sous-tâches plus petites rend l'estimation plus précise et facilite la gestion du travail.
- **Test API** :
    - **JMeter** : Outil open-source utilisé pour tester les performances des applications. JMeter peut simuler une charge élevée sur un serveur, un groupe de serveurs, un réseau ou un objet pour tester sa force ou analyser les performances globales sous différents types de charge.
- **Tests de Sécurité**
    - **SCA (Software Composition Analysis)** : Analyse des composants logiciels pour détecter les vulnérabilités dans les bibliothèques et les frameworks tiers utilisés dans le code.
    - **Conteneurs et Sécurité** : Utilisation de conteneurs sécurisés pour isoler les applications et limiter les risques. Outils comme Hadolint (https://github.com/hadolint) sont utilisés pour analyser les Dockerfiles.
    - **OWASP Dependency Check** : Outil qui identifie les dépendances connues pour être vulnérables et suggère des actions correctives.
    - **IAST (Interactive Application Security Testing)** : Combine les tests statiques et dynamiques pour une analyse en profondeur du code pendant son exécution.
    - **DAST (Dynamic Application Security Testing)** : Tests effectués sur une application en cours d'exécution pour détecter les vulnérabilités que les utilisateurs peuvent exploiter.
    - **SAST (Static Application Security Testing)** : Analyse du code source pour détecter les vulnérabilités avant que le code ne soit exécuté.
- **CI (Continuous Integration)**
    - **Tests Unitaires** : Exécution automatique des tests unitaires à chaque commit pour s'assurer que chaque modification du code ne casse pas les fonctionnalités existantes.

    - **Pipeline CI** : Intégration des tests dans le pipeline CI pour automatiser le processus de build et de test, garantissant que le code reste toujours dans un état déployable.
    - **Tests Statics** : Analyse statique planifiée pour s'exécuter régulièrement, souvent la nuit, afin de détecter les problèmes de sécurité et de qualité du code sans perturber le cycle de développement quotidien.
    - **Tests Dynamiques** : Tests qui prennent plus de temps et qui sont exécutés dans un pipeline séparé pour ne pas ralentir le développement. Ils simulent les interactions utilisateur pour détecter les failles en conditions réelles.

- **Sprints**

    - **Time Boxing** : Technique de gestion du temps qui implique de fixer une durée prédéterminée pour une activité. En contexte Agile, cela signifie que chaque sprint a une durée fixe, assurant une cadence régulière et une planification prévisible.

Outils DevOps

1. **Slides partagées par David Wannier** : Ressources fournies par le formateur, contenant des informations détaillées sur les outils et pratiques DevOps.
2. **OWASP ZAP Scanner** : Outil open-source pour l'analyse de sécurité des applications web, utilisé pour détecter les vulnérabilités.
3. **OWASP Dependency Check** : Outil de gestion des dépendances et de détection des vulnérabilités connues dans les bibliothèques tierces.
4. **Sonarcloud** : Plateforme de qualité du code qui fournit des outils d'analyse statique pour identifier les bugs, les vulnérabilités et les mauvaises pratiques de programmation.

Branching Strategy

- **Exécution des Tests** : Les tests unitaires et de sécurité (hacking) sont effectués sur un serveur dédié pour s'assurer que le code reste stable et sécurisé avant d'être fusionné dans la branche principale.
- **Schéma Excalidraw de Joiakim** : Référence à un schéma visuel modifié pendant la session de formation pour illustrer la stratégie de branching et les meilleures pratiques de gestion des branches.