**Code d’honneur**

En répondant à cet examen à domicile ("take-home") de LOG3000, je m’engage à :

- avoir complété cet examen par moi-même, sans l’aide d’autres personnes ;

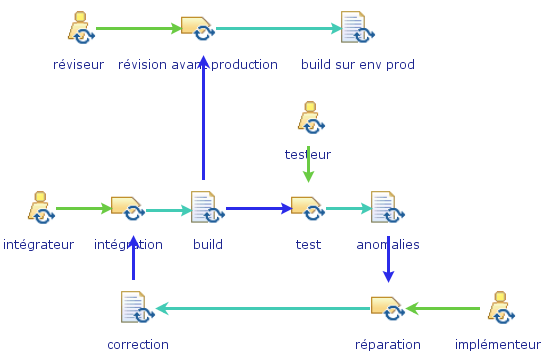
- ne remettre que mes propres travaux, qui ne contiennent aucun matériel plagié ;

- ne pas divulguer les questions posées dans cet examen et les réponses aux questions à quiconque ne suit pas LOG3000 cette session ;

- ne pas participer à des activités malhonnêtes visant à améliorer mes résultats ou à nuire aux résul- tats d’autres personnes.

Signature :

Date et lieu : 1 mai 2022

**1**

1. Le concept de traçabilité implique qu’il faut toujours avoir des records sur toutes les tâches faites comme le code, les artéfacts, les tests etc. dans un cycle de vie d’un logiciel pour améliorer la performance où toutes les membres de l’équipe connaissent leur rôle et ceux des autres. Les équipes vont avoir une meilleure communication et cela aide également à la détection et résolution des problèmes.
2. Un manque de traçabilité peut introduire des conflits pour les différentes équipes qui travaillent sur des tâches différentes. S’il y a une séquence de tâches où chaque tâche dépend de son prédécesseur, l’équipe va avoir beaucoup de difficultés à retracer la tâche qui est la source d’un problème qui affecte les tâches qui la suivent. Un manque de traçabilité peut aussi affecter le travail des autres équipes. Par exemple, si une équipe de développement fait des changements, l’équipe de testeur doit également adapter leur travail. Une bonne traçabilité va permettre de facilement détecter l’impact des changements et adapter l’architecture, le code, les tests, etc.
3. L’entreprise peut commencer par faire un plan pour ajouter une nouvelle technologie pour enregistrer les erreurs de build ou d’intégration. Ils peuvent également introduire un logiciel pour afficher des métriques comme l’usage du CPU, l’usage de la mémoire, le trafic utilisateur sur les servers, etc. Ils peuvent aussi essayer d’introduire plus de technologies d’automatisation pour les tâches d’intégration, de build, de déploiement qui offre déjà des services pour la gestion des métriques.

**2**

1. S’il y a un problème dans le déploiement, les opérateurs doivent signaler les développeurs, car ils ont une meilleure connaissance sur l’architecture du code et l’implémentation. De plus, comme l’environnement de production est différente de l’environnement de développement, les développeurs doivent travailler avec les opérateurs pour voir si le produit va bien fonctionner dans l’environnement de production. Les développeurs et les opérateurs doivent faire beaucoup de collaboration pour livrer un produit fonctionnel.
2. Les caissières ne sont pas les opérateurs, car elles n’ont pas à surveiller les produits et enlever les produits avec des défauts et restocker les produits qui est comme le travail à faire pour vérifier que les produits sont bien organisés pour les clients pour construire l’environnement de production. Donc la mise en production c’est quand les clients vont commencer à acheter les produits de IGA.
3. Adopter la méthodologie DevOps va fortement aider « DATA 4 SERVICES » et les équipes qui travaillent sur le produit. Premièrement, la méthodologie va permettre d’accélérer et d’augmenter la fréquence de déploiement, car plusieurs tâches vont être automatisés ce qui permet d’introduire plus de mise à jour pour les nouvelles fonctionnalités et la réparation des bogues ce qui va attirer plus de clients. Deuxièmement, la méthodologie va améliorer la qualité des produits, l’équipe des développeurs et des opérateurs vont pouvoir recevoir rapidement les commentaires des utilisateurs ce qui va les donner une meilleure piste pour leur travail pour répondre aux besoins des clients. Finalement, les erreurs et les problèmes vont être détecté rapidement. L’automatisation de plusieurs tâches va permettre de faire des tests rapides pour éliminer une grande majorité des problèmes et retravailler dessus.
4. Cette vision est erronée, la méthodologie de DevOps demande plus qu’embaucher un ingénieur DevOps pour ajouter des technologies d’automatisation. Il faut que toutes les équipes concernées adoptent des changements. Par exemple, les développeurs et les opérateurs doivent comprendre que les responsabilités sont partagées. Ils doivent s’entraider et travailler ensemble.

**3**

1. SaaS est un service où une grande majorité des responsabilités est données au fournisseur. L’application, les librairies, le système d’exploitation, les serveurs, le stockage, etc. est fourni par le fournisseur. L’utilisateur a simplement besoin de fournir une configuration personnalisée et les données. Par exemple, l’application « google docs ».

PaaS est un service où on demande à l’utilisateur de fournir l’application et les données. Le reste comme les librairies, le système d’exploitation va être fourni par le fournisseur. Par exemple, si un utilisateur créer une application, il peut utiliser le service « heroku » pour héberger son application.

IaaS est un service où l’utilisateur a la plus grande liberté. Il doit fournir l’application, les librairies, les données et le système d’exploitation. Le fournisseur va offrir la virtualisation, les réseaux, les serveurs et le stockage. Par exemple, l’utilisation de « amazon web services ».

1. L’application « Teams » est un service SaaS, l’utilisateur a juste besoin de fournir une configuration personnalisée et les données. L’application, les librairies, le système d’exploitation, les serveurs, le stockage, etc. est fourni par le fournisseur.
2. L’approche « IaC » consiste à transformer l’infrastructure en code. En d’autres mots, c’est d’automatiser la gestion des dépendants entre les paquets et la gestion des environnements virtualisés. Le travail manuel pour la configuration du matériel physique va être remplacer par des scripts qui vont rouler automatiquement.
3. CI consiste à faire une intégration continue, l’automatisation des builds, des tests et de la fusion des changements. Les changements des développeurs sont fusionnés dans un référentiel de code central partagé. Les changements dans le référentiel vont automatiquement générer des builds et déclencher une série de tests pour vérifier si les changements n’introduit pas de problèmes au code existant. Le lien avec l’environnement est que ces concepts sont seulement possibles avec un environnement de production où l’implémentation de l’automatisation de l’intégration, des tests, du déploiement et de la livraison a été fait.
4. L’utilisation des conteneurs est plus légère, ils ne nécessitent pas un système d’exploitation pour l’exécution de chaque application spécifique. Ils ont besoins de moins de ressources pour s’exécuter les images sont partagées et le système d’exploitation du matériel est réutilisé. La machine virtuelle donne plus de flexibilité pour la personnalisation de l’environnement logiciel. Chaque a un système d’exploitation dédié à son exécution.

**4**

1. Le déploiement est quand le produit est prêt mais les clients n’ont pas encore accès au produit. La mise en production est quand le produit est disponible aux clients et ils peuvent commencer à utiliser le produit.
2. La distinction entre les deux permet de mieux organiser les tâches dans les processus. Par exemple, la gestion de risque va être améliorer. Si des problèmes sont détectés après le déploiement, on peut rapidement les corriger sans nuire à l’expérience des utilisateurs.
3. On peut voir que le problème central est que pour compléter les tests d’intégration, de système et de performance une durée de 15 jours est nécessaire. On peut utiliser la méthodologie de DevOps pour déclencher les tests automatiquement et sauver beaucoup de temps. On peut également utiliser la stratégie de Rolling upgrades, où on commence par mettre la nouvelle version sur une ensemble de serveurs et on redirige le trafic utilisateur vers d’autres ensembles de serveur et par la suite on peut faire les tests. Si des problèmes s’introduit, on peut remettre l’ancienne version et redistribuer le trafic utilisateur.
4. Comme le test consiste à savoir si les étudiants préfèrent avoir le plugin sur le tableau de bord et si c’est mieux de le placer à droite ou à gauche. En utilisant la stratégie de A/B testing, on peut séparer chacun en version différente et distribuer le trafic utilisateur sur les différents ensembles de serveur. À la suite de l’analyse des métriques, on peut choisir de garder une version.
5. Premièrement, il faut bien définir les buts qui est de définir si les utilisateurs aiment un plugin sur le tableau de bord et s’ils aiment, déterminer s’ils préfèrent avoir le plugin à gauche ou à droite du tableau de bord. Deuxièmement, il faut trouver des questions qui va nous aider à atteindre les buts. Par exemple, « Est-ce que les utilisateurs aiment le plugin sur le tableau de bord ? », « Est-ce que les utilisateurs préfèrent avoir le plugin à gauche ? », « Est-ce que les utilisateurs préfèrent avoir le plugin à droite ? ». Troisièmement, il faut déterminer les données à recueillir pour répondre aux questions. Par exemple, le pourcentage d’utilisateurs qui accèdent le nouveau plugin, le nombre d’utilisateurs qui accèdent le plugin à droite par jour et le nombre d’utilisateurs qui accèdent le plugin à gauche par jour. Finalement, il faut expérimenter avec des vrais utilisateurs. Par exemple, si 76% des utilisateurs accèdent le plugin sur le tableau de bord on peut assumer qu’ils sont satisfaits d’avoir le plugin sur le tableau de bord. Si en moyenne 50 utilisateurs accèdent le plugin à gauche chaque jour et 35 utilisateurs accèdent celui à droite on peut assumer qu’ils préfèrent avoir le plugin à gauche du tableau de bord.

**5**

1. Le concept de « Feature toggles » consiste à implémenter toutes les nouvelles fonctionnalités dans des fonctions et avoir une fonction « toggle » pour contrôler l’utilisation des fonctions pour pouvoir basculer entre l’ancienne fonctionnalité et la nouvelle fonctionnalité pour tester et si la nouvelle fonctionnalité passe, on peut nettoyer la fonction « toggle ». Tous les développeurs doivent être conscients de ce que les autres font et avoir une connaissance détaillée de leur code. Le concept de « portes de qualité́ » consiste à avoir des cas de tests à chaque étape pour valider les changements avant de passer à la prochaine étape. Si les changements passent toutes les étapes on va accepter la fusion des changements.
2. Pour commencer, il faut choisir les contributeurs externes et organiser une rencontre pour se mettre d’accord sur qui va développer quelle fonctionnalité. Par la suite il faut séparer les fonctionnalités à faire par fonction et créer les fonctions « toggle » pour chacun. Pour l’implémentation des fonctionnalités les contributeurs peuvent utiliser une technologie comme « VS code live share » pour que tous les contributeurs puissent coder en même temps sur une même version et connaitre les changements fait par les autres contributeurs ou trouver une alternative pour s’assurer que tous les contributeurs connaissent le code détaillé des autres. À la fin de la phase de développement, je pourrais tester chaque fonctionnalité individuellement en basculant sur la nouvelle fonctionnalité pour voir si l’intégration des changements ne causent pas de problème et juste garder et nettoyer les fonctions « toggle » qui marchent et laisser les autres sur l’ancienne fonctionnalité.
3. L’approche de « portes de qualité́ » manuelle consiste à faire les vérifications des changements manuellement par des reviewers et ils peuvent mettre des commentaires pour proposer des changements. Pour intégrer les changements dans le projet, il faut faire un vote pour l’acceptation. L’approche de « portes de qualité́ » automatique consiste à utiliser des outils qui vont automatiquement déclencher des tests pour vérifier si les changements n’introduit pas des problèmes dans la version précédente du projet. Par exemple les tests de « coverage » et les tests de dépendances. Personnellement je vais adopter l’approche de « portes de qualité́ » automatique, car si beaucoup de contributeurs externes décident de participer les tests automatisés vont sauver beaucoup de temps. La vérification de tout le contributeur va prendre trop de temps.
4. La première bonne pratique est qu’ils doivent faire des « pull » fréquents pour s’assurer que leur projet est toujours à jour et régler les problèmes. La deuxième bonne pratique est qu’ils doivent faire des « pull request » fréquent dans la branche « master » pour déclencher les tests automatiques et régler les problèmes détectés.
5. La première pratique à éviter est de faire un « pull request » seulement à la fin de plusieurs changements. Cela risque d’introduire beaucoup de problèmes en même temps et les contributeurs vont gaspiller beaucoup de temps à régler des problèmes. La deuxième pratique à éviter est de pas faire de « pull » fréquents de la branche de « master ». Cela risque d’introduire plusieurs changements qui peuvent briser le code implémenté par un contributeur.

**6**

1. Le concept de « ingénierie du chaos » consiste à introduire des conditions turbulentes intentionnellement dans le système pour trouver des pistes d’amélioration au système ce qui va permettre d’améliorer la fiabilité du système, car la majorité des gros problèmes en production sont déjà réglées.
2. Une première bonne pratique est qu’il faut créer une norme pour les « log » des métriques pour faciliter l’analyser. Si toutes les équipes font de leur propre façon ils ne vont pas comprendre les « log » des autres équipes ce qui va gaspiller beaucoup de temps. Une deuxième bonne pratique est qu’il faut créer des alertes pour signaler quand il y a des changements dans les métriques par rapport à la référence ce qui va permettre de rapidement détecter les défauts dans le système.

**7** J’ai bien aimé la présentation sur les conteneurs et les déploiements, car c’est des concepts assez modernes dans l’industrie. Je pense que le cours peut donner des tp avec des étapes « screenshot » pour guider les étudiants à coder les fichiers de configurations « .yml ».