|  |  |  |  |
| --- | --- | --- | --- |
| charge_de_emploi_CMJN |  |  |  |
| **Dossier-Projet** | | |
|  | | |
|  |  | ◢ |
|  |  |  |

|  |  |  |
| --- | --- | --- |
|  |  |  |
| *Nom de naissance* | ▶ | FÉLIX |
| *Nom d’usage* | ▶ |  |
| *Prénom* | ▶ | Jérémy |
| *Adresse* | ▶ | 70 chemin de la Foyère, 73800 CRUET, Savoie |
|  |  |

|  |
| --- |
| **Certification professionnelle visée** |
|  |
| **Exploiter les outils de développement de la chaîne DevOps** |
|  |

|  |
| --- |
| **Sommaire** |
| 1) Remerciements  2) Résumé du projet en anglais  3) [Liste des compétences du référentiel](#_heading=h.1fob9te)  4) Introduction : contexte du projet, résumé exécutif  5) Partie 1 : compréhension du besoin et traduction technique de la réponse apportée  6) Partie 2 : enjeux de la mise en œuvre du projet : justification des choix et arbitrages réalisés, problèmes rencontrés et solutions apportés  7) Partie 3 : bilan de projet et améliorations envisagées  8) Conclusion : apprentissages, perspectives pour le projet professionnel envisagé |

|  |
| --- |
| 1 - Remerciements |
|  |
|  |
|  |

|  |
| --- |
| 2 – Résumé du projet en anglais |
|  |
| Copier collé de l’autre résumé en anglais |

|  |
| --- |
| 3 - Liste des compétences du référentiel |
|  |
| **C1. Assurer le versionnement d’un code source d’une application organisée en fonctionnalités et lots à l’aide d’un logiciel de contrôle de version de manière à garantir la fiabilité du code source dans un environnement multi-contributeurs**   * Les sources sont versionnées avec Git * Git en ligne de commande * Respect d’un gitflow * Utilisation de Github ou Gitlab par exemple   **C2. Contrôler l'exécution du code source à l’aide de tests et d’outils d’analyses statiques du code source afin de minimiser le risque d’erreur dans un contexte de livraison continue**   * Utilisation d’un *linter (natif IDE ou externe)* * Environnement de test (virtuel ou conteneurisé par exemple) * Au moins des tests unitaires d’intégrés (pas de minimum de *coverage*) * Au moins des tests fonctionnels d’intégrés (pas de minimum de *coverage*) * Savoir récupérer la valeur de la couverture du code par les tests * Exécuter les tests * Interpréter les résultats et les erreurs   **C3. Automatiser les phases de tests unitaires et d’analyses statiques du code source lors du partage des sources à l’aide d’un outil d’intégration continue\* de manière à prévenir les erreurs potentielles**   * Configurer l’intégration continue, avec Github Actions ou Gitlab CI/CD * Paramétrer les phases d'exécution des tests dans l’environnement de test à chaque *push* (sur la branche concernée)   **C4. Concevoir un processus de livraison continue à l’aide d’outils d’automatisation de manière à l’intégrer au processus de développement**   * À chaque *push* (sur la branche concernée) paramétrer les phases de build pour un environnement de pré-production * Paramétrer les phases de livraison des builds en environnement de pré-production   **C5. Développer l’architecture d’une application en micro-services à l’aide d’outils et de bibliothèques logicielles adaptées afin de réduire la complexité globale du système**   * Décomposer une application monolithique en plusieurs composants et services * Utiliser un service de conteneurisation pour tous les environnements : dev, test, prod, etc * Adapter toute la chaîne DevOps à cette nouveauté   **C6. Concevoir un système de veille technologique permettant la collecte, la classification, l’analyse et la diffusion de l’information aux différents acteurs de l’organisation afin d’améliorer la prise de décisions techniques**   * définir des objectifs à sa veille ou encore des thématiques de veille * planifier les temps de veille : durée, fréquence * organiser sa collecte d’information : outils de curation, outils de sauvegarde, etc * organiser le partage voir la diffusion des informations pertinantes   **C7. Accompagner les collaborateurs au sein de l’équipe projet dans la sensibilisation et l’acculturation des méthodes d’organisation et de production DevOps de manière à optimiser le cycle de livraison d’un projet**   * expliquer et partager la culture DevOps * expliquer et partager la méthode DevOps : les rôles, les outils, les leviers, etc * analyser un processus Devops * préconiser des améliorations à un processus DevOps |

|  |
| --- |
| 4 – Introduction : contexte du projet, résumé exécutif |
|  |
| **Dans le corps de l’explication, mettre des captures d ‘écran et rentrer dans le détail des outils utilisés APRÈS :**  Pourquoi est-ce que l’on utilise une méthodologie en DevOps ?  - Ancienne méthode en Silo ()  **> image de cycle en « V »**  - Les équipes étaitent div en silos de comp où ils y a avait d’un coté les dev (production du code) et de l’autre côté, les Ops (déployemennt du code) propre à un une méthodologie de cycle en « V »  Vers le début du 21ème siècle, le manifeste Agile a été rédigé pour élaborer une nouvelle manière de travailler dans le developpement web, dont voici les 4 grands piliers :  -  -  -  -  De ces piliers découlent 12 principes sous-jacents :  -  -  -  -  -  -  -  -  -  -  Ce qui a donné lieu, la méthode de cycles itératifs :  **> schéma de cycles itératifs**  \_\_\_\_\_  Avec la démocratisation d’Internet (arrivée de Google en 2003) et la multiplication des données consommées, de nouveaux standards industriels sont apparues, notamment le SRE (***Site Reliability Engineering***) dont les principaux objectifs sont de créer des systèmes logiciels évolutifs et fiables.  Avec le DevOps, les développeurs et les Operatifs (« Ops ») se sont rapprochés pour (...) colaborer ensemble.  - Automatisation  - Cloud  - IasS (« Infrastructure As a Service ») ; pour le projet, j’ai utilisé l’IasS « Néon » pour le stockage des données en ligne.  - etc,  \_\_\_\_\_  **> Pour choisir les outils de devOps, j’ai comparé les différentes techno existantes sur le marché, voici une liste argumentée des outils et de la raison de leur choix :**  - Kanban ()  - Gest (pour les tests)  - Docker ()  - Linter (voir la veille techno que j’ai faite)  - Git ()  - GitHub (avec « Action » et le fichier yalm)  - etc.  Les enjeux du DevOps d’aujourd’hui est d’assurer la qualité et le suivi du code et de l’application dans un environnement multi-contributeurs,  Pour cela, il faut configurer le dépôt gitHub, des règles de Pull Request et des conventions de nommage (vérifiés par EsLint) qui sont les suivantes :  - CamelCase pour (,,,)  - KebabCase pour les URL  -  \_\_\_\_  Pour s’assurer de la qualité du code, il faut testerson application avec 1) des tests unitaires et 2) des tests e2e.  Voici quelques exemples de tests unitaires que j’ai effectués (avec Gest) :  > Code coverage  \_\_\_\_\_  Pour le build de l’application,  Pour assurer le déploiement de l’application, il a fallu mettre créer des images, c’est-à-dire (,,,) à l’aide de l’outil Docker pour (...)  Puis déposer ses images (une pour le backend et une pour le frontend) dans un répertoire sur la plateforme DockerHub (pareillement que le code source sur gitHub).  Le DevOps comprend aussi l’intégration continue (CI) avec l’automatisation du linter, des tests (+ code coverage) et la création / dépot automatisées des images sur gitHub  **> Schéma d’intégration continue**  Par convention, beaucoup d’outils d’intégration continuent utilise des fichiers yalm (sur gitHub Action).  \_\_\_\_  Or, le devOps n’est pas qu’une chaine d’outils automatisées, mais aussi des habitudes de communications entre les membres d’une équipe  Parler de :  - Kanban  - Code Review  - etc,  \_\_\_\_\_\_  A partir des tickets définis sur Kanban, j’ai créé des branches de fonctionnalité à partir de la branche de développement pour travailler, et valider (via des pull requests) des petits ensembles logiques de code.  Pour pourvoir revenir facilement en arrière lorsqu’un bug survient, il est préférable d’avoir un historique de sauvegardes en faisant concrètement des commits (nommés de façon explicite) régulièrement à chaque fois que l’on rajoute un bout de code fonctionnel et cohérent ; pour m’y faire penser, j’ai lancé un timmer en arrière-plan qui émet un « bip » tous les 15 minutes.  Le site gitmoji.dev m’a permit d’illustrer le nommage de mes commit de façon visuelle.  Voici la procédure que j’ai appliquée pour envoyer une branche de fonctionnalité sur gitHub afin d’effectuer la validation collective : |

|  |
| --- |
| 5 – Partie 1 : Compréhension du besoin et traduction technique de la réponse apportée |
|  |
|  |

|  |
| --- |
| 6 – Partie 2 : Enjeux de la mise en œuvre du projet : justification des choix et arbitrage réalisés, problèmes rencontrés et solutions apportées |
|  |
|  |

|  |
| --- |
| 7 – Partie 3 : bilan de projet et améliorations envisagées |
|  |
|  |

|  |
| --- |
| 8 - Conclusions : apprentissages, perspectives pour le projet professionnel envisagé |
|  |
|  |