Problématique :

La problématique consiste à moderniser une application monolithique de simulation de structures offshores interagissant avec la glace en mer, tout en mettant en place une chaîne d'outils automatisée pour l'intégration continue (CI) et le déploiement continu (CD).

Cette modernisation doit prendre en compte les spécificités de l'entreprise, notamment la forte séparation entre les équipes scientifiques et de production, ainsi que la diversité internationale des équipes de recherche.

L'objectif est de transformer l'application en une architecture modulaire avec un cœur de moteur physique et des services de post-traitement, en introduisant notamment un module de visualisation.

De plus, il est nécessaire de concevoir une solution pour gérer efficacement les données massives générées par le simulateur, tout en respectant les contraintes de performance en lecture/écriture. La plateforme de sortie visée est Windows 11.

Ajout au Monolithe des Services pour l'Application de Simulation en Milieu Arctique :

Fonctionnalités Spécifiques :

Identifier les fonctionnalités clés de l'application, telles que la modélisation de la dynamique des structures offshores en interaction avec la glace, le traitement des résultats, la génération de champs de glace.

Transformer chaque fonctionnalité en un service indépendant avec sa propre logique, en prenant en compte les spécificités liées à la simulation en milieu arctique.

Contrats et Communication :

Il est essentiel d'établir des accords précis entre les différents services, en tenant compte des exigences spécifiques liées à la simulation. Cela garantit que chaque service comprend clairement ses responsabilités et ses interactions avec les autres composants du système. En parallèle, l'adoption de protocoles de communication standard comme REST ou gRPC assure une uniformité dans l'échange de données, tout en prenant en considération les impératifs de performance propres à la simulation. Ces protocoles permettent une transmission efficace et fiable des informations entre les services, contribuant ainsi à la cohérence et à la robustesse de l'ensemble du système.

Containérisation avec Docker adaptée à l'Application de Simulation :

Containérisation des Services :

Chaque service doit être encapsulé dans un conteneur Docker afin de garantir qu'il puisse être facilement déployé sur différentes plateformes et qu'il fonctionne de manière isolée, sans interférer avec d'autres services.

Pour ce faire, nous utilisons des fichiers appelés Dockerfiles pour décrire les composants nécessaires à chaque service, y compris les bibliothèques spécifiques utilisées pour la simulation. Ces Dockerfiles définissent les dépendances et les configurations requises pour chaque service, ce qui facilite la gestion et le déploiement de l'ensemble du système de simulation.

Gestion des Dépendances :

Pour gérer efficacement les dépendances entre les différents conteneurs Docker, nous utilisons Docker Compose. Cette solution nous permet de définir et de coordonner les relations entre les différents services, tout en garantissant l'intégrité des bibliothèques de simulation. En parallèle, il est crucial de documenter de manière claire et détaillée les dépendances entre les services, en mettant particulièrement l'accent sur les bibliothèques et les composants spécifiques à la simulation. Cette documentation aide les développeurs et les opérateurs à comprendre les interactions entre les différents services, facilitant ainsi la maintenance et le dépannage du système de simulation.

Orchestration avec Kubernetes ou Swarm adapté à l'Application de Simulation :

Pour orchestrer de manière efficace les services liés à la simulation en milieu arctique, il est crucial d'utiliser des outils comme Kubernetes ou Swarm. Ces plateformes permettent de

simplifier le déploiement, la mise à l'échelle et la gestion des services, en garantissant une disponibilité élevée et une performance optimale. Grâce à une gestion efficace des conteneurs, ces solutions assurent que les services sont toujours accessibles et répondent aux exigences de performance de la simulation.

Pour garantir la flexibilité et l'évolutivité du système, Kubernetes ou Swarm permettent le déploiement de chaque service de manière indépendante. Ces outils exploitent des concepts comme les pods, les services et les déploiements pour assurer une évolutivité adaptée aux besoins spécifiques de la simulation. Ainsi, les charges de travail variables sont gérées de manière efficace, permettant au système de s'adapter aux fluctuations de la demande sans compromettre la qualité de la simulation. Expliquer le pourquoi des pods (evite les crash ou la surutilisation ...)

Chaîne CI/CD Automatisée pour le Développement de l'Application de Simulation :

Pour développer l'application de simulation de manière efficace, il est crucial de mettre en place une chaîne CI automatisée. Cela permet une séparation claire entre les équipes scientifiques et de production, tout en prenant en compte les particularités de la simulation en milieu arctique.

Une approche modulaire dans la chaîne CI/CD est essentielle pour permettre aux équipes de travailler de manière indépendante. Des outils comme Jenkins ou GitLab CI(parler des difference) facilitent cette séparation tout en assurant une intégration continue adaptée à la simulation.

Les étapes de la chaîne CI doivent être automatisées pour garantir un processus fluide. Cela inclut la compilation, les tests unitaires, la création de conteneurs Docker et le déploiement, en prenant en compte les bibliothèques et les composants spécifiques à la simulation.

Il est également important d'intégrer des scripts pour la gestion des versions et la documentation automatique. Cela met l'accent sur la traçabilité des changements liés à la simulation, ce qui est crucial pour assurer la transparence et la cohérence du processus de développement. GIT TAG

Tests Automatisés pour l'Application de Simulation :

Pour assurer la qualité de l'application de simulation, il est essentiel d'intégrer des tests automatisés à chaque étape du processus CI. Cela inclut des tests unitaires, d'intégration et de performance, en tenant compte des exigences de précision propres à la simulation. Nous utilisons des frameworks de test appropriés(LESQUELS ?) pour le langage de programmation utilisé dans le contexte de la simulation, ce qui garantit une couverture complète et précise des fonctionnalités.

Pour maintenir la stabilité des services de simulation, nous mettons en place des tests de régression automatisés. Ces tests permettent de détecter les régressions potentielles lors des mises à jour. De plus, nous intégrons des tests de charge spécifiques à la simulation pour évaluer les performances dans des conditions réalistes.

Surveillance et Logging pour l'Application de Simulation :

Pour assurer un suivi efficace de l'application de simulation, nous utilisons des outils de surveillance tels que Prometheus et Grafana. Ces outils nous permettent de surveiller en temps réel les performances des services de simulation, en nous concentrant sur les métriques critiques pour la modélisation en milieu arctique. Nous configurons des alertes pour réagir rapidement, en prenant en compte des seuils spécifiques à la simulation.

Pour faciliter l'identification rapide des problèmes dans le contexte de la simulation, nous mettons en place des mécanismes de logging cohérents dans chaque service. Nous utilisons des agrégateurs de logs tels qu'ELK (Elasticsearch, Logstash, Kibana) pour centraliser et analyser les logs. Cela nous permet de corréler les événements liés à la simulation et de diagnostiquer rapidement les problèmes en production, ce qui est essentiel pour assurer la disponibilité et la fiabilité du système.

Kubernetes :

Origine : Kubernetes a été développé par Google et est un projet open-source hébergé par la Cloud Native Computing Foundation (CNCF).

Architecture : Il suit une architecture maître-esclave, où un maître (control plane) gère l'ensemble du cluster tandis que les nœuds de travail (nodes) exécutent les applications.

Orchestration : Kubernetes offre une orchestration avancée avec des fonctionnalités telles que le déploiement déclaratif, l'équilibrage de charge, la découverte de service et l'auto-scaling.

Définition des Applications : Il utilise des fichiers YAML pour décrire l'état souhaité de l'application (Pods, Services, etc.).

Évolutivité : Conçu pour gérer des clusters massifs avec une extensibilité élevée, pouvant supporter des milliers de nœuds.

Taille des Clusters : Convient aux clusters de taille importante.

Adoption : Très populaire et largement adopté dans l'industrie.

Écosystème : Bénéficie d'un écosystème riche et de nombreuses intégrations avec d'autres outils.

Docker Swarm :

Origine : Docker Swarm est développé par Docker, la même société derrière le logiciel de conteneurisation Docker.

Architecture : Il suit également une architecture maître-esclave, avec un gestionnaire de Swarm (Swarm manager) et des nœuds de travail qui exécutent les conteneurs.

Orchestration : Docker Swarm fournit une orchestration plus simple et plus accessible, adaptée aux cas d'utilisation moins complexes.

Définition des Applications : Il utilise également des fichiers YAML pour la définition des services.

Évolutivité : Convient pour des déploiements plus simples, bien qu'il puisse être extensible, il peut ne pas être aussi adapté que Kubernetes pour des scénarios très massifs.

Taille des Clusters : Bien adapté aux petits et moyens clusters.

Adoption : Moins utilisé que Kubernetes, surtout pour les déploiements complexes.

Écosystème : Bien que moins étendu que Kubernetes, Docker Swarm offre un écosystème suffisant pour des besoins de déploiements plus simples.

Conclusion :

Si vous recherchez une solution puissante, hautement évolutive, avec une riche fonctionnalité d'orchestration et un large écosystème, Kubernetes serait un meilleur choix étant open source.

Docker Swarm peut être plus approprié pour des scénarios plus simples ou lorsque vous avez déjà une expérience significative avec Docker et que vous préférez une solution intégrée.

Jenkins :

Fonction : Jenkins est un outil d'intégration continue open-source.

Objectif : Automatise le processus d'intégration, de déploiement et de tests dans le cycle de vie du développement logiciel.

Caractéristiques : Facilite l'intégration régulière du code, la construction automatisée, le déploiement continu, et la gestion des workflows.

Avantages : Améliore l'efficacité du développement, réduit les erreurs d'intégration, et permet des déploiements plus fréquents et fiables.

Utilisation Courante : Adapté à tout projet nécessitant une automatisation du processus de développement et de déploiement, des petits projets aux grandes entreprises.

Logiciel proposés :

Les logiciels que je recommanderais pour répondre à vos besoins spécifiques dans le cadre de la modernisation de cette application de simulation en milieu arctique, tout en tenant compte des spécificités de la culture d'entreprise et des exigences du projet, sont les suivants :

Containerisation avec Docker :

Docker : Leader incontesté dans le domaine de la containerisation, offrant une solution complète pour la création, le déploiement et l'exécution d'applications dans des conteneurs. Avec une large adoption industrielle et une vaste communauté de développeurs, Docker est un choix solide.

Chaîne CI/CD Automatisée :

Jenkins : Outil d'intégration continue open-source largement utilisé, offrant une grande flexibilité et extensibilité pour automatiser les processus d'intégration, de déploiement et de tests.

GitLab CI : Intégré à l'infrastructure GitLab, GitLab CI offre une intégration étroite avec les dépôts Git et une interface utilisateur conviviale.

Tests Automatisés :

JUnit (pour Java) ou PyTest (pour Python) : Ces frameworks de test sont largement utilisés pour écrire et exécuter des tests unitaires et d'intégration dans leurs langages respectifs, offrant une grande flexibilité et facilité d'utilisation.

Selenium : Populaire pour les tests d'interface utilisateur automatisés, permettant de tester l'application à travers différents navigateurs et plates-formes.

(si besoin d'autres environnement)

Surveillance et Logging :

Prometheus : Solution populaire pour la surveillance des performances, offrant une collecte de métriques en temps réel, des requêtes flexibles et des alertes intégrées.

Grafana : Souvent utilisé avec Prometheus pour visualiser les données de surveillance et créer des tableaux de bord interactifs.

ELK Stack (Elasticsearch, Logstash, Kibana) : Solution robuste pour le logging, avec Elasticsearch pour l'indexation et le stockage des logs, Logstash pour l'ingestion des logs et Kibana pour la visualisation et l'analyse des logs.

Ces logiciels sont largement adoptés dans l'industrie, offrant une base solide de fonctionnalités et compatibles avec la plateforme visée, Windows 11. Leur adoption facilitera la collaboration entre les équipes scientifiques et de production, tout en répondant aux exigences spécifiques du projet.

Probleme de communication :

Pour surmonter les problèmes humains et favoriser une collaboration entre les équipes scientifiques et de production dans le cadre du projet de modernisation de l'application de simulation en milieu arctique, voici quelques mesures recommandées :

Communication et collaboration renforcées :

Organiser des réunions quotidiennes (daily scrums) pour permettre aux membres des deux équipes de discuter des progrès et des besoins spécifiques du projet.

Formation et sensibilisation :

Offrir des formations aux équipes scientifiques sur les outils et processus utilisés dans le projet.

Impliquer activement les membres des deux équipes dans toutes les phases de la modernisation de l'application.

Développer une culture de collaboration :

Promouvoir une culture d'entreprise où la collaboration et le travail d'équipe sont valorisés.

Reconnaître et récompenser les contributions des membres des deux équipes à la réussite du projet.

Transparence et partage des responsabilités :

Clarifier les rôles et les attentes de chaque équipe.

Assurer une répartition claire des responsabilités entre les équipes scientifiques et de production.

Feedback et amélioration continue :

Encourager les membres des deux équipes à partager leurs feedbacks, suggestions et préoccupations.

Utiliser ces retours d'expérience pour identifier les domaines d'amélioration et ajuster les stratégies en conséquence.

En mettant en œuvre ces mesures, vous favoriserez une meilleure collaboration entre les équipes, améliorant ainsi l'efficacité et le succès du projet de modernisation de votre application de simulation en milieu arctique.

Sources :

https://kubernetes.io

https://docs.docker.com/engine/swarm

https://grafana.com/

https://openclassrooms.com/fr/courses/1750566-optimisez-la-securite-informatique-grace-au-monitoring/7145268-decouvrez-la-stack-elk/

https://www.jenkins.io/

https://docs.pytest.org/en/8.0.x/

https://junit.org/junit5/

écris par moi et reformuler et corrigé par gpt 3