**Ce quoi Devops ?**

DevOps est un ensemble de pratiques qui unissent le développement de logiciels (Dev) et les opérations informatiques (Ops) dans le but d'améliorer la collaboration et la productivité des équipes.

**Les outils et pratiques suivants :**

* **Jenkins** : Outil d'intégration continue qui automatise la compilation et les tests des applications.
* **Docker** : Permet d'empaqueter des applications et leurs dépendances dans des conteneurs, facilitant ainsi le déploiement.
* **Git** : Système de gestion de versions décentralisé, essentiel pour le suivi des modifications de code.
* **Nexus** : Gestionnaire de référentiels pour organiser et stocker les artefacts de développement.
* **JUnit** : Framework pour effectuer des tests unitaires en Java, garantissant que le code fonctionne comme prévu.
* **SonarQube** : Outil d'analyse de la qualité du code, permettant une revue continue du code source.
* **Grafana et Prometheus** : Outils de visualisation et de surveillance des performances des applications.

**Concepts Clés :**

* **Environnements de Développement** :
  + **DEV** : Environnement où le code est écrit et testé par les développeurs.
  + **INT** : Environnement d'intégration pour vérifier la compatibilité des différents composants.
  + **UAT (User Acceptance Testing)** : Tests effectués pour s'assurer que l'application répond aux besoins des utilisateurs.
  + **QUALIF** : Tests de conformité par rapport à des normes définies.
  + **PREPROD** : Tests finaux avant le déploiement en production.
  + **PROD** : Environnement opérationnel où l'application est utilisée par les utilisateurs finaux.
* **Culture DevOps** : Favorise la collaboration entre les équipes de développement et d'opérations pour transformer les tensions en coopération productive.
* **CI/CD** :
  + **Intégration Continue** : Processus où chaque modification de code lance des tests automatiques pour détecter rapidement les erreurs.
  + **Livraison Continue** : Automatisation du processus de déploiement d'une application à tout moment.
  + **Déploiement Continu** : Mises à jour automatiques en production sans intervention manuelle.

**Avantages de DevOps**

* **Réduction du Cycle de Livraison** : Accélération du temps entre le développement et le déploiement.
* **Optimisation des Ressources** : Utilisation efficace des ressources humaines et techniques.
* **Amélioration de la Qualité** : Détection précoce des bugs et validation continue de la qualité du code.
* **Human-Centric** : Met l'accent sur l'importance des équipes humaines dans le processus de développement et de déploiement.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Qu'est-ce que Vagrant ?**

Vagrant est un outil open-source qui permet de créer, configurer et gérer des machines virtuelles via une interface de commande. Il facilite le développement collaboratif en garantissant que tous les membres de l'équipe travaillent sur la même configuration. Vagrant est un outil puissant et essentiel pour la création et la gestion d'environnements de développement.

**Architecture de Vagrant**

* **Vagrantfile** : Fichier de configuration où sont définies toutes les configurations nécessaires pour la machine virtuelle.
* **Vagrant Cloud** : Un service où les utilisateurs peuvent partager des "boxes" (images de machines virtuelles pré-configurées) telles qu'Ubuntu ou CentOS.

**Installation**

1. **Installation de VirtualBox** : Nécessaire comme outil de virtualisation. Téléchargez-le à partir de [VirtualBox](https://www.virtualbox.org/wiki/Downloads).
2. **Installation de Vagrant** : Téléchargez Vagrant depuis [Vagrant](https://www.vagrantup.com/downloads) et installez-le. Vérifiez l'installation avec la commande vagrant --version.

**Création et Configuration d'une Machine Virtuelle**

1. Créez un dossier pour vos projets Vagrant, par exemple D:\Vagrant.
2. Créez un sous-dossier nommé « ubuntu » et ouvrez PowerShell.
3. Générer un Vagrantfile: vagrant init
4. Modifiez le Vagrantfile pour définir la configuration de la machine virtuelle.
5. Créer et démarrer la machine virtuelle : vagrant up
6. Accéder à la machine : vagrant ssh

**Gestion des Machines Virtuelles**

* **Arrêter une machine** : vagrant halt
* **Suspendre une machine** : vagrant suspend
* **Redémarrer une machine** : vagrant reload
* **Supprimer une machine** : vagrant destroy

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Résumé : Maven**

**Définition de Maven**

Maven est un outil essentiel pour le développement Java, facilitant la gestion des projets et des dépendances.

Maven est un outil de gestion de projets développé par la fondation Apache, permettant l'automatisation de la construction et de la gestion des dépendances d'un projet. Il offre plusieurs fonctionnalités clés :

* **Automatisation des tâches** : Simplifie les processus récurrents.
* **Construction et compilation** : Gère la création des projets.
* **Gestion des dépendances** : Facilite l'intégration de bibliothèques et de modules externes.
* **Génération des livrables** : Produit des fichiers JAR (Java Archive) et WAR (Web Archive).
* **Documentation et rapports** : Génère des documents de projet.
* **Déploiement d'applications** : Utilise la commande mvn deploy pour déployer les applications.

**Installation de Maven**

Maven peut être installé de deux manières :

1. **En tant que plugin** : Inclus par défaut dans des IDE comme Eclipse ou IntelliJ.
2. **Standalone** : Installé via l'invite de commandes. Pour les utilisateurs d'Ubuntu, la commande est sudo yum install maven.

**Gestion des Dépendances**

* **POM.XML** : Le fichier de configuration principal contenant des informations sur le projet et ses dépendances.
* **Repositories** : Les dépendances et livrables sont stockés dans des repositories qui peuvent être :
  + **Local** : Situé dans le répertoire utilisateur.
  + **Central** : Repository public accessible en ligne.
  + **Distant** : Autres repositories configurés pour des besoins spécifiques.

**Gestion des Versions**

La version des projets suit une convention de numérotation :

* **Release majeure** : Changements significatifs.
* **Release mineure** : Ajouts de fonctionnalités sans rupture.
* **Patch** : Corrections de bugs.

**Commandes Maven**

Maven utilise des commandes pour gérer le cycle de vie des projets :

* **mvn compile :** Compile les fichiers source.
* **mvn test :** Exécute les tests unitaires.
* **mvn package :** Crée le livrable dans le dossier target.
* **mvn install :** Installe le livrable dans le repository local.
* **mvn deploy :** Déploie le livrable sur un repository distant.
* **mvn clean :** Supprime le contenu du dossier target.
* **mvn site :** Génère un site web avec des rapports sur l'application.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Résumé : Outil de travail collaboratif (Git)**

**Introduction**

Git est un système de contrôle de version décentralisé, développé par Linus Torvalds. Il permet aux équipes de développement de gérer les modifications du code source au fil du temps, facilitant ainsi le travail collaboratif et l'accélération des processus de développement.

**Systèmes de Contrôle de Version**

* **Définition** : Outils logiciels qui aident à gérer les changements apportés au code source.
* **Avantages** : Historique complet des modifications, possibilité de branches et de merges, et traçabilité des changements.
* **Utilisateurs** : Utilisé non seulement par les développeurs, mais aussi par les architectes, administrateurs et équipes d'infrastructure pour gérer des fichiers de configuration.

**SVN vs Git**

* **SVN (Apache Subversion)** : Système de versioning centralisé avec une architecture client-serveur. Nécessite une connexion réseau pour les mises à jour.
* **Git** : Système décentralisé où chaque développeur a une copie complète du projet. Permet un travail hors ligne et une meilleure gestion des modifications.

**Installation et Configuration de Git**

* **Installation** : Pour vérifier l'installation, utilise git --version.
* **Configuration GitHub** : Collaborer sur des projets et gérer le versioning sans utiliser la ligne de commande.

**Commandes Git**

* **Initialisation** **d’un dépôt :** git init
* **Ajouter des fichiers au staging area :** git add
* **Commit pour enregistrer les modifications :** git commit
* **Gestion des branches** : Création et gestion des branches pour permettre des développements parallèles.
* **Push pour envoyer des modifications vers un dépôt distant :** git push
* **Pull pour récupérer des mises à jour :** git pull

**Gestion des Conflits**

Les conflits peuvent survenir lorsque plusieurs développeurs modifient le même contenu. Git permet de gérer ces conflits lors des merges, en encourageant le travail dans des branches isolées.

**Commandes Avancées**

* **Annuler des Commits** : git reset
* **Différences pour voir les différences entre le dernier commit et le répertoire de travail** : git diff HEAD

Une image contenant texte, capture d’écran, diagramme, ligne

Description générée automatiquement

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Résumé : Chapitre 2 - CI-Agents Jenkins**

Jenkins est un outil incontournable pour l'intégration continue (CI), la livraison continue (CD) et le déploiement continu. Il est basé sur une architecture distribuée qui utilise un modèle maître-esclave pour gérer les builds de manière efficace.

**Architecture Distribuée de Jenkins**

* **Maître-Esclave** : Jenkins fonctionne avec un serveur principal (master) qui coordonne les tâches et des agents (slaves) qui exécutent les builds. Cette architecture permet de répartir la charge de travail et d'exécuter plusieurs builds simultanément.

**Rôles des Composants**

* **Master** : Gère l'ordonnancement des tâches, surveille les agents, et exécute des builds.
* **Slave** : Exécute les tâches de build envoyées par le maître.

**Avantages de l'Architecture Distribuée**

* **Optimisation des Ressources** : Répartition de la charge de travail entre plusieurs agents.
* **Scalabilité** : Adaptation rapide aux variations des charges de travail.
* **Isolation des Pannes** : Si un agent échoue, cela n'affecte pas les autres builds.

**Terminologie Clé**

* **Executors** : Nombre de tâches pouvant être exécutées simultanément sur un agent.
* **Job** : Ensemble de tâches pour construire un projet.

**Types d'Agents**

1. **Agents Statics** : Installés sur des machines physiques ou virtuelles.
2. **Agents Dynamiques** : Provisionnés à la demande, souvent sur des conteneurs Docker ou des clusters Kubernetes.

**Connexion Master-Slave**

Jenkins utilise plusieurs méthodes pour connecter le maître aux agents :

* **SSH** : Connexion standard pour les environnements Unix/Linux.
* **JNLP** : Utilisé pour démarrer des agents sur divers systèmes d'exploitation.
* **Docker** : Les conteneurs peuvent servir d'agents Jenkins.

**Configuration de l'Environnement Distribué**

La configuration implique :

1. Installation de Jenkins (master).
2. Configuration des agents (slaves) sur différents environnements.
3. Établissement de la connexion entre le maître et les agents.

**Distribution des Builds**

Les builds peuvent être associés à des agents spécifiques, permettant une gestion efficace des ressources et une allocation optimale des tâches.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Résumé : Module DevOps – Jenkins**

**La vie avant Jenkins**

* Avant Jenkins, les développeurs devaient attendre la fin du développement pour tester le code, ce qui ralentissait le processus de livraison.

**Jenkins - Définition**

* Jenkins est un serveur open source écrit en Java, conçu pour automatiser l'intégration continue et la livraison continue. Il s'interface avec des systèmes de gestion de versions comme Git et Subversion et prend en charge des outils de construction comme Apache Maven et Ant.

**Avantages de Jenkins**

* **Plugins** : Large éventail de plugins disponibles pour étendre les fonctionnalités.
* **Architecture distribuée** : Permet d'accélérer les builds en répartissant le travail sur plusieurs machines.

**Installation de Jenkins**

1. Vérifier l'installation : java -version
2. **Installation de JDK 17** : sudo apt install openjdk-11-jdk -y.
3. **Installation de Maven** : sudo apt install maven -y.
4. **Installation de Jenkins** : sudo apt install jenkins -y.
5. **Installation de Git** : sudo apt install git

**Configuration d’un projet avec Jenkins**

* **Jobs** : Les jobs représentent les processus de build dans Jenkins. Les étapes principales incluent :
  1. Création du job.
  2. Configuration des étapes du build.
  3. Lancement du build.
* **Méthodes de configuration** :
  1. **Freestyle** : Configuration via un formulaire.
  2. **Pipeline** : Utilisation de scripts (Groovy) pour définir le processus de build.

**Exemples de Configuration**

* **Freestyle Job** : Comprend la récupération du code, la compilation, et le lancement de tests.
* **Pipeline** : Permet d'écrire le processus de build dans un fichier Jenkinsfile, facilitant la gestion des versions et des modifications.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Résumé : Introduction à Docker**

**Virtualisation**

* **Définition** : Technologie permettant de créer des machines virtuelles (VM) sur une machine physique. Docker facilite le déploiement et la gestion d’applications dans des conteneurs, offrant flexibilité et efficacité dans le développement logiciel. Pour toute question, vous pouvez contacter le département informatique.
* **Types** :
  + **Virtualisation lourde** : Chaque VM a son propre système d'exploitation, entraînant un gaspillage de ressources.
  + **Virtualisation légère** : Utilise des conteneurs qui partagent le noyau de l'hôte, offrant un environnement isolé et portable.

**Conteneurisation**

* Les conteneurs emballent une application et ses dépendances, permettant une portabilité entre différents environnements.

**Docker**

* **Définition** : Outil pour créer et gérer des conteneurs.
* **Versions** :
  + **Docker EE** : Version payante avec support avancé.
  + **Docker CE** : Version gratuite pour les utilisateurs individuels.

**Avantages de Docker**

* **Flexibilité** : Supporte des applications complexes.
* **Légèreté** : Utilisation efficace des ressources.
* **Portabilité** : Moins de dépendances avec la machine hôte.
* **Faible couplage** : Conteneurs autonomes pouvant être remplacés sans perturber le système.

**Inconvénients de Docker**

* **Portabilité** : Difficulté à gérer les conteneurs sur différents systèmes d'exploitation.
* **Gestion complexe** : Difficulté à gérer plusieurs conteneurs simultanément.
* **Sécurité** : Partage du même système d'exploitation peut présenter des failles.

**Composants de Docker**

* **Docker Engine** : Logiciel central pour gérer les conteneurs.
* **Docker Daemon** : Service en arrière-plan qui gère les opérations.
* **Docker CLI** : Interface en ligne de commande pour interagir avec Docker.
* **Docker Hub** : Registre public pour stocker et partager des images Docker.

**Installation de Docker**

1. Mise à jour du système : sudo apt update
2. Installation des prérequis.
3. Ajout de la clé GPG de Docker.
4. Ajout du dépôt Docker à la liste des sources APT.
5. Installation de Docker avec : sudo apt install docker-ce

**Commandes de Base**

* **Vérifier la version de Docker** : docker -v
* **Exécuter une image** : docker run hello-world
* **Créer un conteneur** : docker run "nom de l’image"
* **Lister les images** : docker images
* **Supprimer une image** : docker rmi "nom de l’image"
* **Gérer les conteneurs** : docker ps, docker rm

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Résumé : Module DevOps – Nexus**

**Introduction**

* Après l'implémentation et les tests des modules d'une application, il est crucial de déployer l'application dans les environnements appropriés.
* L'objectif est d'automatiser la gestion des livrables et l'alimentation des environnements par les artefacts correspondants.

**Livrable**

* **Définition :** Un livrable est le résultat d'un projet, comprenant le code source, la compilation, et l'archivage dans un dépôt distant pour le déploiement.

**Release vs Snapshot**

* **Release :** Version fixe d'un projet avec toutes les fonctionnalités requises.
* **Snapshot :** Version en développement, pouvant avoir plusieurs itérations pour divers usages (tests, validations, etc.).

**Nexus - Définition**

* Nexus est un outil de gestion de dépôts permettant d'héberger des artefacts (ex. : jar, war).
* Développé par Sonatype en Java, il existe en version gratuite (Community) et payante.

**Types de Dépôts dans Nexus**

1. **Hosted :** Dépôts contenant les livrables créés par les utilisateurs.
   * Releases : Artefacts stables.
   * Snapshots : Artefacts en développement.
2. **Proxy :** Relais d'un dépôt distant pour optimiser les builds.
3. **Virtual :** Adaptateur pour structurer des dépôts.
4. **Group :** Regroupement de dépôts sous une même URL.

**Installation de Nexus**

* **Utilisation d'une image Docker :** 
  + Télécharger l'image : docker pull sonatype/nexus3
  + Créer le conteneur : docker run -d -p 8081:8081 --name nexus sonatype/nexus3
  + Vérifier le conteneur : docker container ls -a
* **Accéder à Nexus via l'adresse IP de la machine virtuelle.**

**Configuration de Nexus**

* Connexion avec les identifiants par défaut (admin) et changement de mot de passe.
* Déploiement des livrables via :
  + Nexus Artifact Uploader ou
  + Maven :
    - Modifier settings.xml pour ajouter les informations de connexion.
    - Configurer pom.xml pour pointer vers le repository Nexus.

**Configuration avec Jenkins**

* Intégration de Nexus avec Jenkins pour automatiser la construction et le déploiement des livrables.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Résumé : Module DevOps – SonarQube**

**Introduction**

* Les applications peuvent présenter des problèmes comme du code dupliqué, des méthodes non commentées, et des violations des bonnes pratiques.
* Les erreurs peuvent résulter de bugs, de tests unitaires mal développés, ou de classes trop volumineuses.

**Tests Dynamiques et Tests Statiques**

* **Tests dynamiques** : Réalisés pendant l'exécution de l'application pour détecter des dysfonctionnements (ex. : tests unitaires).
* **Tests statiques** : Analyzent le code source avant exécution, permettant de détecter les violations de bonnes pratiques. SonarQube est un outil pour ces tests.

**SonarQube - Définition**

* SonarQube est un outil open source de test statique pour analyser la qualité du code source selon des règles prédéfinies.
* Il permet une inspection continue de la qualité du code.

**SonarQube - Caractéristiques**

* Supporte de nombreux langages (Java, .Net, Python, etc.).
* Détecte divers défauts de codage (code non utilisé, dupliqué, etc.).
* Permet de choisir les règles d'analyse.

**SonarQube - Fonctionnalités**

* Détection rapide des erreurs, amélioration de la productivité, réduction des coûts, et évaluation de la couverture des tests unitaires.
* Intégration avec des outils comme Jenkins et Maven pour des analyses automatisées.

**SonarQube - Rapports**

* Génère des rapports sur l'architecture, les tests unitaires, les bugs potentiels, le code complexe, et les règles de programmation.

**SonarQube - Couverture du Code**

* Indique le nombre de tests unitaires, le taux de succès, et la couverture de code.
* Une couverture faible peut signaler des lacunes dans les tests.

**SonarQube - Architecture**

* Composée d'un exécuteur pour les analyses, de plugins pour étendre les fonctionnalités, d'un serveur pour stocker et visualiser les résultats, et d'une base de données (H2 par défaut, mais peut être MySQL, PostgreSQL, etc.).

**SonarQube - Installation**

* Accessible via un plugin pour IDE ou une image Docker.
* Installation via Docker : docker pull sonarqube:8.9.7-community et docker run -d -p 9000:9000 sonarqube:8.9.7-community.

**Utilisation de SonarQube**

* Accès via l'URL http://<ip>:9000 avec les identifiants « admin/admin ».
* Changement du mot de passe recommandé.
* Pour l'intégration avec Jenkins, récupérer un projet depuis Git et analyser le code avec Maven.

**Analyse des Résultats**

* Visualiser les résultats d'analyse sur l'interface de SonarQube.
* Évaluer la qualité du code et identifier les bugs, vulnérabilités, et "code smells".

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Résumé : Les tests dans DevOps**

**Introduction**

* **Équipe Scrum** : Composée de développeurs et d'ingénieurs qualité.
* **Rôle de l'ingénieur qualité** : Surveiller la qualité tout au long des itérations et communiquer avec l'équipe.

**Quelques Types de Tests**

* **Tests d'intégration** : Vérifient le bon fonctionnement de plusieurs composants.
* **Tests de régression** : Vérifient que les modifications n'ont pas introduit de nouveaux bugs.
* **Tests de montée en charge** : Évaluent la capacité du système à gérer un nombre croissant d'utilisateurs.
* **Tests de sécurité** : Identifient les vulnérabilités et protègent les données.

**Les Tests dans le Pipeline CI/CD**

* **Intégration Continue** : Inclut des tests unitaires et de qualité du code.
* **Livraison Continue** : Nécessite des tests d'intégration avancés pour garantir la qualité.
* **Déploiement Continu** : Exige des tests fonctionnels dans un environnement identique à la production.

**Tests Manuels vs Tests Automatiques**

* **Tests manuels** : Fastidieux et sujets à erreurs humaines.
* **Tests automatiques** : Plus rapides, peuvent être exécutés en parallèle, et nécessitent un temps de mise en place.

**Test Unitaire**

* **Définition** : Test d'une seule unité de code pour vérifier son bon fonctionnement.
* **Phases** : Comprend l'initialisation, l'application d'un stimulus, et l'observation du comportement (AAA - Arrange, Act, Assert).
* **Outils** : Utilisation de frameworks comme JUnit pour automatiser les tests.

**JUnit**

* **Définition** : Framework pour le développement et l'exécution de tests unitaires.
* **Annotations** : Utilisation d'annotations pour définir des tests et gérer leur ordre d'exécution.
* **Assertions** : Méthodes pour vérifier les résultats des tests.

**Mockito**

* **Utilisation** : Permet de simuler des dépendances pour les tests unitaires, isolant ainsi le code testé.
* **Vérification** : Utilisation de la méthode verify() pour s'assurer que les interactions avec les mocks se produisent comme prévu.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Résumé : Docker Compose**

**Introduction**

* **Objectif** : Intégrer une application Spring Boot dans une chaîne DevOps complète (CI/CD).
* **Livraison Continue (CD)** : Assurer le bon fonctionnement de l'application dans différents environnements (machines physiques, virtuelles, ou conteneurs Docker).

**Docker Compose**

* **Définition** : Outil pour définir et exécuter des applications Docker multi-conteneurs à l'aide de fichiers YAML.
* **Fonctionnement** :
  1. Définir l'environnement avec un Dockerfile.
  2. Décrire les services dans un fichier docker-compose.yml.
  3. Exécuter docker compose up pour démarrer tous les services.

**Utilisation de Docker Compose**

* **Commandes Principales** :
  + docker-compose up -d : Lancer les services en arrière-plan.
  + docker-compose logs : Vérifier les logs des conteneurs.
  + docker-compose down : Arrêter et supprimer les conteneurs.

**Docker Volume**

* **Fonctionnalité** : Permet la persistance des données générées par les conteneurs.
* **Avantages** : Les volumes facilitent la réutilisation des données entre plusieurs conteneurs et assurent leur intégrité même après la suppression des conteneurs.

**Docker et Jenkins**

* **Automatisation** : Utiliser Jenkins pour automatiser la création et le déploiement d'images Docker.
* **Étapes** :
  1. Installer le plugin Docker Pipeline.
  2. Créer un Dockerfile pour le BackEnd.
  3. Configurer les stages pour construire et déployer l'image sur DockerHub.
  4. Lancer le fichier Docker Compose via Jenkins.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Résumé : Monitoring avec Prometheus et Grafana**

**Introduction**

* **Supervision des applications (Monitoring)** : Processus de collecte de données pour suivre la disponibilité, les performances et les anomalies des applications.
* **Outils de surveillance** : Fournissent des alertes en cas d'anomalies et permettent de suivre les événements.

**Prometheus**

* **Définition** : Système de monitoring avec une base de données de séries chronologiques, conçu pour surveiller divers cibles (serveurs, bases de données, etc.).
* **Caractéristiques** :
  + Basé sur le langage Go.
  + Utilise un modèle de scraping continu.
  + Offre un langage de requêtage (PromQL).

**Installation**

* Utiliser Docker :

bash

Copier

docker pull prom/prometheus

docker run -d --name prometheus -p 9090:9090 prom/prometheus

**Configuration**

* Accéder à l'interface web : http://@IP:9090
* Configurer la supervision de Jenkins en ajoutant un job dans prometheus.yml.

**Grafana**

* **Définition** : Outil open source de visualisation et d'analyse permettant de créer des dashboards basés sur diverses sources de données.
* **Installation** :

bash

Copier

docker pull grafana/grafana

docker run -d --name grafana -p 3000:3000 grafana/grafana

* **Configuration** :
  + Accéder à l'interface web : http://@IP:3000 (admin/admin par défaut).
  + Ajouter une source de données (Prometheus).

**Différence entre conteneur et image**

* **Image** : Modèle immuable contenant tout ce qui est nécessaire pour exécuter une application.
* **Conteneur** : Instance en cours d'exécution d'une image, dynamique et isolée.

**Différence entre conteneur et machine virtuelle (VM)**

* **Conteneur** : Partage le noyau de l'OS, léger, démarre rapidement.
* **VM** : Exécute un OS complet, lourd, démarre lentement, isolation matérielle.

**Différence entre Prometheus et Grafana**

* **Prometheus** : Système de monitoring et d'alerte, stocke des métriques.
* **Grafana** : Plateforme de visualisation de données, crée des tableaux de bord interactifs.

**Différence entre CI et CD**

* **CI (Continuous Integration)** : Intégration fréquente du code avec tests automatisés.
* **CD (Continuous Delivery/Deployment)** : Livraison automatique des builds en staging ou production.

**Qu'est-ce que DevOps ?**

* **DevOps** : Culture et pratiques unifiant développement et opérations pour des livraisons rapides et continues.

**Qu'est-ce qu'un Docker Compose ?**

* **Docker Compose** : Outil pour définir et gérer des applications multi-conteneurs avec un fichier YAML.

**Qu'est-ce qu'un pipeline ?**

* **Pipeline** : Série de processus automatisés pour livrer des logiciels, incluant intégration, tests et déploiement.

**Pourquoi utiliser Nexus et quel est son rôle ?**

* **Nexus** : Gestionnaire de référentiel pour stocker et distribuer des artefacts de build, facilitant le partage et la réutilisation.

**Différence entre test statique et test dynamique**

* **Test statique** : Analyse du code sans exécution, détecte erreurs syntaxiques et vulnérabilités.
* **Test dynamique** : Exécution du code, vérifie fonctionnalités et performance en temps réel.

**Qu'est-ce qu'un Coverage ?**

* **Coverage (couverture de code)** : Mesure du pourcentage de code testé, aide à identifier les parties non testées pour améliorer la qualité.
* **Jenkins** : Outil d'intégration continue qui automatise la compilation et les tests des applications.
* **Docker** : un outil de **conteneurisation:** Permet d'empaqueter des applications et leurs dépendances dans des conteneurs, facilitant ainsi le déploiement.
* **Git** : Système de gestion de versions décentralisé, essentiel pour le suivi des modifications de code.
* **Nexus** : Gestionnaire de référentiels pour organiser et stocker les artefacts de développement.
* **JUnit** : Framework pour effectuer des tests unitaires en Java, garantissant que le code fonctionne comme prévu.
* **Mockito** est une bibliothèque effectue des **tests unitaires** en Java, en utilisant le framework **JUnit**. Permet de simuler des dépendances pour les tests unitaires, isolant ainsi le code testé.
* **JaCoCo** (Java Code Coverage) est un outil de couverture de code:Il permet de mesurer le pourcentage de code source qui est exécuté lors de l'exécution de tests unitaires: aide à identifier les parties de votre code qui ne sont pas suffisamment testées.
* **SonarQube** : Outil d'analyse de la qualité du code, permettant une revue continue du code source.
* **Grafana et Prometheus** : Outils de visualisation et de surveillance des performances des applications.
* **Prometheus** est un système de surveillance et de gestion des séries temporelles open-source conçu pour collecter, stocker, et analyser des métriques
* **Grafana** Outil open source de visualisation et d'analyse permettant de créer des dashboards basés sur diverses sources de données.(et metrics)