TP : Installation des outils et création d'un pipeline CI/CD intégrant DevSecOps

Objectifs du TP :

L'objectif principal de ce TP est de vous initier au monde de l'intégration continue et de la livraison continue (CI/CD) en intégrant des pratiques de sécurité dès le début du cycle de développement (DevSecOps). Vous serez guidé à travers l'installation et la configuration des outils essentiels, puis aidé à créer un pipeline de base intégrant des contrôles de sécurité pour une application simple.

Plus précisément, à la fin de ce TP, vous devriez être capable de :

- **Installer et configurer les outils nécessaires pour un pipeline CI/CD intégrant la sécurité**, incluant Git pour la gestion sécurisée du code source, Jenkins comme serveur d'automatisation intégrant des étapes de sécurité, et Docker pour la conteneurisation sécurisée de votre application. Comprendre comment installer et configurer correctement ces outils est fondamental pour établir une base solide pour votre pipeline DevSecOps.
- **Créer un pipeline CI/CD basique intégrant des étapes de sécurité automatisées** pour une application simple. Vous apprendrez à définir une série d'étapes automatisées au sein de Jenkins qui construiront, testeront (y compris des tests de sécurité) et potentiellement prépareront le déploiement sécurisé de votre application.
- **Comprendre le workflow du pipeline CI/CD avec une emphase sur la sécurité**, de l'intégration continue à la livraison continue sécurisée. Vous saisirez le flux logique d'un pipeline CI/CD, en commençant par l'intégration continue (fusion fréquente des changements de code et automatisation des builds et des tests, y compris des analyses de sécurité) et en progressant vers la livraison continue (automatisation de la préparation du déploiement en assurant la sécurité de l'artefact).

En résumé, l'utilité de ce TP est d'établir les fondations pratiques pour comprendre et mettre en œuvre les principes de base du CI/CD avec une intégration précoce et continue de la sécurité (DevSecOps) en utilisant des outils standard de l'industrie comme Git, Jenkins et Docker.

Prérequis :

Avant de commencer ce TP, assurez-vous d'avoir les connaissances et les outils suivants :

- **Notions de base en ligne de commande :** Une familiarité avec l'utilisation du terminal ou de l'invite de commandes est nécessaire pour exécuter les commandes d'installation et de configuration de Git et potentiellement Docker.
- **Une machine avec Docker installé** (optionnel : VirtualBox ou WSL si nécessaire) : Docker est un outil puissant pour créer et gérer des conteneurs de manière isolée, ce qui contribue à la sécurité. Son utilisation simplifie grandement l'installation de Jenkins et la conteneurisation sécurisée de votre application. Si vous n'avez pas Docker installé nativement sur votre système d'exploitation, vous pouvez utiliser une machine virtuelle via VirtualBox ou le Sous-système Windows pour Linux (WSL) si vous êtes sous Windows.

- **Accès à un IDE (comme VS Code ou IntelliJ) :** Un Environnement de Développement Intégré (IDE) facilitera la consultation et la modification des fichiers de code de votre projet, y compris ceux liés à la sécurité.
- **Accès à internet :** Une connexion internet est indispensable pour télécharger les outils et cloner des dépôts Git.

Plan du TP :

Le TP est divisé en trois parties principales :

Partie 1 : Préparation et installation sécurisée des outils

Cette première partie est cruciale car elle met en place les fondations de votre pipeline CI/CD intégrant la sécurité.

1. **Installer Git et sécuriser la gestion du code source**
    - **Téléchargez et installez Git :** Rendez-vous sur le site git-scm.com et téléchargez la version de Git correspondant à votre système d'exploitation. Suivez les instructions d'installation. Git est un système de contrôle de version distribué, essentiel pour suivre les modifications de votre code source et collaborer avec d'autres développeurs de manière auditable.
    - **Configurez Git :** Une fois installé, ouvrez votre terminal ou invite de commandes et exécutez les commandes suivantes en remplaçant "Votre Nom" et "Votre Email" par vos informations personnelles :
    - `git config --global user.name "Votre Nom"`
    - `git config --global user.email "Votre Email"`

      Ces commandes configurent votre nom d'utilisateur et votre adresse e-mail qui seront associés à vos commits (enregistrements de modifications), permettant de suivre l'historique des changements et d'identifier les auteurs en cas de problèmes de sécurité. L'option `--global` applique cette configuration à tous vos projets Git sur cette machine.

    - **Vérification :** Pour vous assurer que Git est correctement installé, exécutez la commande suivante :
    - `git --version`

      Ceci devrait afficher la version de Git installée sur votre système. L'utilisation de branches pour isoler les correctifs de sécurité et les revues de code via les pull requests sont des pratiques importantes pour intégrer la sécurité dès la gestion du code source.

2. **Installer Jenkins et renforcer sa sécurité**
    - **Installez Jenkins via Docker :** L'utilisation de Docker est une manière simple et rapide de déployer Jenkins dans un environnement potentiellement isolé. Si Docker est installé sur votre machine, exécutez les commandes suivantes dans votre terminal :
    - `docker pull jenkins/jenkins:lts`
    - `docker run -d -p 8080:8080 -p 50000:50000 --name jenkins jenkins/jenkins:lts`

        - `docker pull jenkins/jenkins:lts` : Cette commande télécharge l'image Docker officielle de Jenkins avec la désignation `lts` (Long Term Support), ce qui signifie qu'il s'agit d'une version stable avec des mises à jour de sécurité régulières.
        - `docker run -d -p 8080:8080 -p 50000:50000 --name jenkins jenkins/jenkins:lts` : Cette commande exécute un nouveau conteneur basé sur l'image `jenkins/jenkins:lts`.
            - `-d` : Exécute le conteneur en arrière-plan (detached mode).
            - `-p 8080:8080` : Mappe le port 8080 de votre machine hôte au port 8080 du conteneur Jenkins. Vous pourrez accéder à l'interface web de Jenkins via `http://localhost:8080`.
            - `-p 50000:50000` : Mappe le port 50000 de votre machine hôte au port 50000 du conteneur Jenkins. Ce port est utilisé par les agents Jenkins pour communiquer avec le serveur principal.
            - `--name jenkins` : Donne le nom "jenkins" à votre conteneur, ce qui facilite sa gestion.
            - `jenkins/jenkins:lts` : Spécifie l'image Docker à utiliser.
    - **Vérification :**
        - **Accédez à Jenkins via votre navigateur :** Ouvrez votre navigateur web et naviguez vers l'adresse `http://localhost:8080`. Si Jenkins est correctement démarré, vous devriez voir une page de configuration initiale.
        - **Suivez les instructions pour l'initialisation :** Jenkins vous demandera un mot de passe administrateur initial. Ce mot de passe est généralement stocké dans les logs Docker du conteneur Jenkins. Vous pouvez le récupérer en exécutant la commande suivante dans un autre terminal :
        - `docker logs jenkins`

          Recherchez une ligne similaire à `**********` qui contient le mot de passe administrateur. Copiez ce mot de passe et collez-le dans le champ approprié de la page web de Jenkins. Suivez ensuite les instructions à l'écran pour terminer l'installation. **Il est crucial de configurer un utilisateur administrateur fort et de sécuriser l'accès à Jenkins dès cette étape.** Lors de l'installation des plugins, considérez ceux qui peuvent améliorer la sécurité de votre pipeline.

3. **Cloner un dépôt Git et examiner les pratiques de sécurité du code**
    - **Clonez un dépôt Git existant (ou créez-en un nouveau) :** Pour ce TP, vous aurez besoin d'un projet de code source géré par Git. Vous pouvez soit cloner un dépôt existant hébergé sur une plateforme comme GitHub, GitLab ou Bitbucket, soit créer un nouveau dépôt localement et y ajouter quelques fichiers. **Pensez à la manière dont les droits d'accès sont gérés sur votre dépôt et à l'importance des revues de code pour la sécurité.**
    - **Exemple de commande :** Si vous souhaitez cloner un dépôt existant, utilisez la commande `git clone` suivie de l'URL du dépôt :
    - `git clone https://github.com/example/repository.git`
    - `cd repository`

La première commande clone le dépôt dans un nouveau dossier nommé "repository" dans votre répertoire courant. La deuxième commande vous déplace à l'intérieur de ce dossier.

Partie 2 : Création du pipeline CI/CD intégrant la sécurité

Maintenant que les outils sont installés, nous allons configurer un pipeline CI/CD de base intégrant des étapes de sécurité dans Jenkins.

1. **Configurer Jenkins pour un projet avec une approche DevSecOps**
    - **Créer un Job dans Jenkins :**
        - Ouvrez votre navigateur et accédez à l'interface web de Jenkins (`http://localhost:8080`). Connectez-vous avec les identifiants que vous avez créés lors de l'initialisation.
        - Sur le tableau de bord de Jenkins, cliquez sur le bouton "New Item".
        - Entrez un nom descriptif pour votre job (par exemple, "mon-projet-devsecops-ci").
        - Choisissez l'option "Freestyle Project" et cliquez sur le bouton "OK". Les projets Freestyle offrent une grande flexibilité pour définir des étapes de build et de sécurité personnalisées.
    - **Configurer le dépôt Git :**
        - Dans la page de configuration de votre nouveau job, recherchez la section "Source Code Management".
        - Sélectionnez l'option "Git".
        - Dans le champ "Repository URL", entrez l'URL du dépôt Git que vous avez cloné précédemment.
        - Si votre dépôt Git nécessite des identifiants (par exemple, s'il s'agit d'un dépôt privé), vous devrez configurer des "Credentials" dans Jenkins et les sélectionner ici. **Assurez-vous de gérer ces identifiants de manière sécurisée dans Jenkins.** Vous pouvez ajouter des identifiants en cliquant sur le bouton "Add" à côté du champ "Credentials".
2. **Ajout d'étapes de build et de sécurité au pipeline**

    Dans la section "Build" de la configuration du job, vous allez définir les différentes étapes de votre pipeline, en y intégrant des contrôles de sécurité. Cliquez sur le bouton "Add build step" pour ajouter chaque étape.

    - **Étape 1 : Build et analyse de code statique (SAST)**
        - Sélectionnez l'option "Execute shell" (ou une option similaire si vous utilisez d'autres outils de build comme Maven et des plugins SAST).
        - Dans la zone de texte "Command", entrez la commande nécessaire pour compiler votre application, suivie de la commande pour exécuter un outil d'analyse de code statique (SAST). Par exemple, si votre application utilise Maven, vous pourriez intégrer un plugin comme SpotBugs ou SonarQube (via un plugin Jenkins ou une exécution en ligne de commande) après la compilation :
        - `mvn clean package`
        - `# Exemple avec SpotBugs`
        - `mvn spotbugs:check`

```
# Ou pour une analyse SonarQube (nécessite une configuration préalable de SonarQube)
# mvn sonar:sonar
```

L'analyse de code statique permet de détecter des vulnérabilités potentielles dans le code source sans l'exécuter.

- **Étape 2 : Tests unitaires et analyse de la composition des logiciels (SCA)**
  - Cliquez à nouveau sur "Add build step" et sélectionnez "Execute shell" (ou une option spécifique à votre outil de test et d'analyse SCA).
  - Dans la zone de texte "Command", entrez la commande pour exécuter vos tests unitaires, suivie de la commande pour exécuter un outil d'analyse de la composition des logiciels (SCA). Les outils SCA analysent les dépendances open source de votre application à la recherche de vulnérabilités connues. Par exemple, avec Maven, vous pourriez utiliser un plugin comme OWASP Dependency-Check :

```
mvn test
# Exemple avec OWASP Dependency-Check
mvn dependency-check:check
```

Les tests unitaires assurent la fonctionnalité du code, tandis que l'analyse SCA aide à identifier les risques liés aux bibliothèques externes.

- **Étape 3 : Docker : Build et scan de l'image Docker**
  - Si votre application est conteneurisée avec Docker, ajoutez une nouvelle étape de build en sélectionnant "Execute shell".
  - Dans la zone de texte "Command", entrez les commandes Docker nécessaires pour construire l'image, suivies d'une commande pour scanner l'image à la recherche de vulnérabilités :

```
docker build -t mon-app .
# Exemple d'un scan de vulnérabilités avec un outil comme Trivy (doit être installé)
trivy image mon-app
```

Cette commande construit une image Docker nommée `mon-app` en utilisant le Dockerfile présent dans le répertoire de votre projet. **Le scan de l'image Docker permet de détecter les vulnérabilités dans les couches de l'image et les logiciels installés dans le conteneur.** Vous pourriez également envisager de pousser l'image vers un registre Docker sécurisé après le scan :

```
# docker login -u votre_nom_utilisateur -p votre_mot_de_passe
# docker tag mon-app votre_nom_utilisateur/mon-app:latest
# docker push votre_nom_utilisateur/mon-app:latest
```

3. **Exécution du pipeline et gestion des alertes de sécurité**
   - Une fois que vous avez configuré les étapes de votre pipeline, cliquez sur le bouton "Save" en bas de la page de configuration du job.
   - Pour lancer l'exécution du pipeline, retournez à la page du job et cliquez sur le bouton "Build Now".

- Vous pouvez observer l'avancement de chaque étape du pipeline en cliquant sur le numéro du build dans l'historique des builds, puis en cliquant sur "Console Output". Cela affichera les logs de chaque étape, vous permettant de suivre l'exécution et d'identifier d'éventuelles erreurs ou alertes de sécurité.
- **Si une étape d'analyse de sécurité détecte des vulnérabilités au-delà d'un certain seuil, configurez votre pipeline pour que le build échoue.** Examinez attentivement les logs pour comprendre la cause de l'alerte de sécurité et corrigez le problème dans votre code ou vos dépendances. **Mettez en place des mécanismes de feedback pour informer les développeurs des problèmes de sécurité détectés le plus tôt possible.**

Partie 3 : Validation et discussion sur l'amélioration continue de la sécurité du pipeline

Cette dernière partie vous encourage à vérifier le succès de votre pipeline intégrant la sécurité et à réfléchir à son amélioration continue en matière de sécurité.

1. **Vérification de la réussite du pipeline et des contrôles de sécurité**
   - **Assurez-vous que toutes les étapes du pipeline CI (build, tests unitaires et analyses de sécurité) sont réussies :** Dans la vue du build Jenkins, vérifiez que toutes les étapes sont marquées en vert, indiquant qu'elles se sont terminées sans erreur et sans détection de vulnérabilités critiques selon vos seuils configurés.
   - **Vérifiez l'image Docker générée et les résultats du scan de sécurité (si utilisée) :** Si votre pipeline inclut une étape Docker, vous pouvez vérifier si l'image a été créée localement en exécutant la commande `docker images` dans votre terminal. Examinez également les logs de l'étape de scan de l'image pour comprendre les éventuelles vulnérabilités détectées.
2. **Analyse et optimisation du pipeline avec un focus sur la sécurité**
   - **Quels sont les points forts du pipeline créé en termes de sécurité ?** Réfléchissez aux avantages d'avoir automatisé l'intégration de contrôles de sécurité à chaque modification du code. Par exemple, l'automatisation assure une exécution cohérente des analyses de sécurité et une identification précoce des vulnérabilités.
   - **Quelles étapes supplémentaires pourraient être ajoutées pour améliorer davantage la sécurité du pipeline ?** Pensez à d'autres étapes qui pourraient renforcer la sécurité dans un pipeline CI/CD plus complet (DevSecOps). Voici quelques exemples :
     - **Tests de sécurité dynamiques (DAST) :** Déployer l'application dans un environnement de test et utiliser des outils pour simuler des attaques et identifier les faiblesses au niveau de l'application en cours d'exécution.
     - **Analyse de la configuration de l'infrastructure (IaC Security) :** Si vous utilisez des outils d'Infrastructure as Code, intégrer des analyses pour détecter les erreurs de configuration qui pourraient introduire des risques de sécurité.
     - **Tests de pénétration automatisés :** Intégrer des outils qui effectuent des tests d'intrusion de manière automatisée.
     - **Gestion des secrets :** S'assurer que les informations sensibles (clés d'API, mots de passe) sont gérées de manière sécurisée et ne sont pas exposées dans le code ou les logs.

    - **Politiques de sécurité et seuils d'acceptation :** Définir clairement les politiques de sécurité et les seuils de sévérité des vulnérabilités qui doivent entraîner l'échec du pipeline.
    - **Notifications et rapports de sécurité :** Configurer des alertes (par e-mail, Slack, etc.) en cas de détection de vulnérabilités et générer des rapports de sécurité réguliers.

Livrables :

Pour valider votre travail, vous devrez fournir les éléments suivants :

- **Capture d'écran du pipeline Jenkins en exécution :** Une capture d'écran montrant les étapes de votre job Jenkins, y compris les étapes d'analyse de sécurité, et leur statut (succès ou échec).
- **Log du succès ou de l'échec des étapes :** Les sorties de la console de chaque étape de votre pipeline, montrant qu'elles se sont déroulées comme prévu (ou les erreurs et alertes de sécurité rencontrées). Fournissez en particulier les logs des outils d'analyse de sécurité (SAST, SCA, scan Docker).
- **Un fichier texte ou PDF résumant les améliorations possibles pour le pipeline en matière de sécurité :** Une courte description des étapes de sécurité supplémentaires que vous avez identifiées pour améliorer le pipeline, en expliquant brièvement leur intérêt pour renforcer la sécurité de l'application et du processus de développement.