Atelier 10 : Examen blanc pratique

Objectifs de l'atelier :

- **Prendre une application simple** fournie sur un dépôt Git.
- **Écrire un Dockerfile** pour containeriser l'application.
- **Créer un Jenkinsfile** pour configurer un pipeline Jenkins qui :
  - Teste l'application.
  - Construit l'image Docker.
  - Pousse l'image sur un **Docker registry**.
- **Écrire un playbook Ansible** pour déployer un serveur Docker.
- **Lancer l'application conteneurisée** à l'aide de ce playbook Ansible.

Scénario complet :

1. **Cloner l'application depuis le dépôt Git**

   Objectif : Commencez par cloner l'application depuis le dépôt Git fourni pour avoir une base de travail.

   - Exécutez la commande suivante pour cloner l'application :

```
git clone https://github.com/nom-utilisateur/app-simple.git
cd app-simple
```

   Note : L'application peut être une simple application web en Python, Node.js, ou autre technologie.

2. **Écrire un Dockerfile**

   Objectif : Créer un fichier **Dockerfile** pour containeriser l'application.

   - **Exemple pour une application Node.js** :

```
# Étape 1 : Utiliser une image de base officielle
FROM node:14

# Étape 2 : Créer un répertoire pour l'application
WORKDIR /usr/src/app

# Étape 3 : Copier les fichiers de l'application dans le conteneur
COPY package*.json ./

# Étape 4 : Installer les dépendances
RUN npm install

# Étape 5 : Copier le reste des fichiers
COPY . .
```

```
# Étape 6 : Exposer le port utilisé par l'application
EXPOSE 3000

# Étape 7 : Démarrer l'application
CMD [ "npm", "start" ]
```

- Si l'application est en Python, voici un exemple de **Dockerfile** pour une application Flask :

```
FROM python:3.9

WORKDIR /usr/src/app

COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 5000

CMD [ "python", "app.py" ]
```

Instructions :

- Remarquez que le **Dockerfile** doit prendre en compte le langage et les dépendances de votre application.
- Veuillez adapter le **Dockerfile** en fonction de l'application fournie.

3. **Écrire un Jenkinsfile**

Objectif : Créer un **Jenkinsfile** qui va orchestrer le pipeline CI/CD de l'application.

Exemple de **Jenkinsfile** pour Docker :

```
pipeline {
    agent any

    environment {
        DOCKER_REGISTRY = 'mydockerregistry.com'
        IMAGE_NAME = 'app-simple'
    }

    stages {
        stage('Test') {
            steps {
                echo 'Running tests...'
                sh 'npm test'  // Changez ceci selon le framework de
test utilisé
            }
        }

        stage('Build Docker Image') {
```

```
            steps {
                script {
                    docker.build("${IMAGE_NAME}:${BUILD_NUMBER}")
                }
            }
        }

        stage('Push to Docker Registry') {
            steps {
                script {
                    docker.withRegistry('https://${DOCKER_REGISTRY}',
'docker-credentials') {

docker.image("${IMAGE_NAME}:${BUILD_NUMBER}").push()
                    }
                }
            }
        }
    }
}
```

Explication :

- Le pipeline commence par exécuter les tests.
- Ensuite, il construit l'image Docker à l'aide du **Dockerfile**.
- Enfin, l'image est poussée vers un **Docker Registry**.

Note : Veuillez remplacer les parties spécifiques à votre application et ajuster les commandes de test selon le langage et le framework que vous utilisez.

4. **Créer un Playbook Ansible pour déployer un serveur Docker**

Objectif : Utiliser **Ansible** pour déployer un serveur Docker sur une machine distante.

Exemple de **playbook Ansible** (`deploy-docker.yml`) pour installer Docker sur une machine distante :

```
---
- name: Installer Docker sur la machine distante
  hosts: all
  become: yes

  tasks:
    - name: Mise à jour des paquets
      apt:
        update_cache: yes

    - name: Installer Docker
      apt:
        name: docker.io
        state: present

    - name: Démarrer Docker et l'activer au démarrage
```

```
    service:
      name: docker
      state: started
      enabled: yes
```

Ce playbook permet d'installer **Docker** sur une machine distante. Une fois Docker installé, vous pourrez y déployer l'application conteneurisée.

5. **Déployer l'application conteneurisée avec Ansible**

   Objectif : Lancer l'application conteneurisée en utilisant Docker et Ansible.

   Ajoutez une nouvelle tâche dans le playbook Ansible pour déployer l'application :

```
- name: Lancer l'application conteneurisée
  docker_container:
    name: app-simple
    image: "{{ docker_image_name }}:latest"
    state: started
    restart_policy: always
```

   Explication :

   - Cette tâche déploie l'image Docker construite dans le pipeline Jenkins sur le serveur distant via Ansible.
   - Le paramètre `restart_policy: always` garantit que le conteneur sera relancé automatiquement en cas de redémarrage du serveur.

   Variable à définir dans le playbook :

```
vars:
  docker_image_name: 'mydockerregistry.com/app-simple'
```

Résumé des étapes de l'atelier :

Étape	Description
1. Cloner l'application	Récupérer le code source de l'application fournie sur GitHub
2. Créer le Dockerfile	Containeriser l'application en écrivant un Dockerfile adapté
3. Créer le Jenkinsfile	Mettre en place un pipeline Jenkins pour tester, construire et pousser l'image Docker
4. Créer un playbook Ansible	Automatiser l'installation de Docker via Ansible
5. Lancer l'application avec Ansible	Déployer l'application en utilisant Docker et Ansible

Conclusion :

Cet atelier permet de combiner plusieurs outils DevOps pour créer un pipeline complet de **CI/CD** avec **Docker**, **Jenkins**, et **Ansible**. À la fin de cet exercice, les participants devraient être en mesure de :

- **Construire une image Docker** à partir d'un Dockerfile.
- **Mettre en place un pipeline Jenkins** pour tester, construire et pousser l'image vers un registre Docker.
- **Utiliser Ansible** pour automatiser le déploiement d'un serveur Docker et lancer l'application dans un conteneur.