Projet DevOps : Automatisation du Déploiement d'une Application Web Java sur AWS

NACERE Mohammed

27 octobre 2025

Table des matières

1 Objectif du projet

Le projet **DevOps-project** a pour objectif de concevoir une chaîne complète d'intégration et de déploiement continu (CI/CD) afin d'automatiser le **build, le test et le déploiement** d'une application web Java sur une instance **AWS EC2**.

L'approche repose sur une architecture DevOps, combinant les aspects développement (Maven, Java) et opérations (automatisation, scripts, services Linux).

2 Architecture globale du projet

2.1 Chaîne CI/CD

```
Développeur → GitHub → CodeBuild → CodeDeploy → EC2
                                        ↓
                                  Tomcat + HTTPD
```

L'architecture comprend :

- Une application web Java packagée avec **Maven**.
- Une phase de build via **AWS CodeBuild**.
- Une phase de déploiement via **AWS CodeDeploy**.
- Une instance EC2 hébergeant les services **Tomcat** (application Java) et **HTTPD** (reverse proxy).

3 Partie développement : Application et Maven

3.1 Application Java Web

L'application est constituée d'un fichier `index.jsp` et d'un descripteur de déploiement `web.xml` :

```
<web-app>
  <display-name>Archetype Created Web Application</display-name>
</web-app>
```

L'application est packagée sous forme de fichier **.war** (Web Archive) grâce à Maven.

3.2 Configuration Maven : pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.nextwork.app</groupId>
  <artifactId>nextwork-web-project</artifactId>
  <packaging>war</packaging>
  <version>1.0-SNAPSHOT</version>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
```

```
      <scope>test</scope>
    </dependency>
  </dependencies>
  <build>
    <finalName>nextwork-web-project</finalName>
  </build>
</project>
```

Ce fichier définit :
- Les métadonnées du projet (nom, version, artefact).
- Les dépendances (ici JUnit pour les tests).
- Le packaging final sous forme de fichier `.war`.

3.3 Fichier settings.xml

```
<settings>
<servers>
  <server>
    <id>nextwork-nextwork-devops-cicd</id>
    <username>aws</username>
    <password>${env.CODEARTIFACT_AUTH_TOKEN}</password>
  </server>
</servers>
<profiles>
  <profile>
    <id>nextwork-nextwork-devops-cicd</id>
    <activation>
      <activeByDefault>true</activeByDefault>
    </activation>
    <repositories>
      <repository>
        <id>nextwork-nextwork-devops-cicd</id>
        <url>https://nextwork-820450995345.d.codeartifact.eu-north-1.amazonaws.com/ma
      </repository>
    </repositories>
  </profile>
</profiles>
</settings>
```

Ce fichier permet à Maven d'accéder au dépôt privé **AWS CodeArtifact** pour télécharger ou publier des dépendances.

4 Partie intégration continue (CI) – CodeBuild

Le fichier `buildspec.yml` définit la procédure de build automatique :

```
version: 0.2

phases:
```

```
  install:
    runtime-versions:
      java: corretto17
    commands:
      - echo "Setting up Maven & CodeArtifact..."
      - export CODEARTIFACT_AUTH_TOKEN=${CODEARTIFACT_AUTH_TOKEN}
  build:
    commands:
      - mvn -s settings.xml clean package
artifacts:
  files:
    - target/*.war
    - appspec.yml
    - scripts/*
```

CodeBuild compile le projet avec Maven, génère le fichier `.war` et le stocke avec les scripts et fichiers de configuration nécessaires au déploiement.

5 Partie déploiement continu (CD) – CodeDeploy

5.1 Fichier appspec.yml

```
version: 0.0
os: linux
files:
  - source: /target/nextwork-web-project.war
    destination: /usr/share/tomcat/webapps/
hooks:
  ApplicationStop:
    - location: scripts/stop_server.sh
      timeout: 300
      runas: root
  BeforeInstall:
    - location: scripts/install_dependencies.sh
      timeout: 300
      runas: root
  ApplicationStart:
    - location: scripts/start_server.sh
      timeout: 300
      runas: root
```

Les étapes réelles d'exécution sont :

1. **ApplicationStop** : arrêt des services Tomcat et HTTPD.
2. **BeforeInstall** : installation des dépendances et configuration du serveur.
3. **Copie du WAR** dans le répertoire Tomcat.
4. **ApplicationStart** : démarrage des services.

6 Scripts Bash d'automatisation

6.1 install_dependencies.sh

```
#!/bin/bash
sudo yum install tomcat -y
sudo yum -y install httpd
sudo tee /etc/httpd/conf.d/tomcat_manager.conf >/dev/null <<'EOF'
<VirtualHost *:80>
  ServerAdmin root@localhost
  ServerName app.nextwork.com
  ProxyRequests Off
  ProxyPreserveHost On
  ProxyPass / http://localhost:8080/nextwork-web-project/
  ProxyPassReverse / http://localhost:8080/nextwork-web-project/
</VirtualHost>
EOF
```

Ce script installe les services nécessaires (Tomcat et Apache HTTPD) et configure un **VirtualHost** qui redirige les requêtes du port 80 vers le port 8080. HTTPD joue ici le rôle de **reverse proxy**, tandis que Tomcat exécute l'application Java.

6.2 start_server.sh

```
#!/bin/bash
sudo systemctl start tomcat.service
sudo systemctl enable tomcat.service
sudo systemctl start httpd.service
sudo systemctl enable httpd.service
```

Ce script démarre et active les services Tomcat et HTTPD au démarrage de la machine. Il garantit que l'application est accessible après chaque redéploiement ou redémarrage de l'instance.

6.3 stop_server.sh

```
#!/bin/bash
isExistApp="$(pgrep httpd)"
if [[ -n $isExistApp ]]; then
  sudo systemctl stop httpd.service
fi
isExistApp="$(pgrep tomcat)"
if [[ -n $isExistApp ]]; then
  sudo systemctl stop tomcat.service
fi
```

Ce script arrête proprement les services avant le déploiement d'une nouvelle version afin d'éviter les conflits.

7 Explication de la relation Tomcat / HTTPD

— **Tomcat** : exécute l'application Java (.war) sur le port 8080.
— **HTTPD** : agit comme un serveur web frontal (port 80) et redirige les requêtes vers Tomcat via un reverse proxy.

Pourquoi utiliser les deux :

— Séparer les rôles (Tomcat = application, HTTPD = web/public).
— Améliorer la sécurité (Tomcat non exposé directement).
— Faciliter l'ajout de fonctionnalités (HTTPS, logs, compression, cache).

8 Conclusion

Ce projet met en œuvre un pipeline DevOps complet intégrant :
— La **compilation** automatisée via CodeBuild (Maven).
— Le **déploiement** automatisé via CodeDeploy (scripts Bash).
— L'**exécution** de l'application sur Tomcat, exposée au public via HTTPD.

Il démontre la compréhension de concepts clés : **CI/CD, scripting, automatisation, infrastructure as code et orchestration de services**. L'application est livrée de manière fiable, traçable et sans intervention manuelle.