Creation d’une toolchain custom

Table des matières

[Création de l’environnement de travail sur le poste local 3](#_Toc529286001)

[Initialisation de l’environnement de travail sur le poste local 3](#_Toc529286002)

[Création du code source 3](#_Toc529286003)

[Création d'un projet sur GitHub 7](#_Toc529286004)

[Clone sur le poste/serveur local 7](#_Toc529286005)

[Préparation de l’environnement de build 8](#_Toc529286006)

[(Optionnel) tester le fonctionnement du code dans un Docker 9](#_Toc529286007)

[Initialisation d’une instance de tests 9](#_Toc529286008)

[Compilation du projet 9](#_Toc529286009)

[Préparation de l’environnement Docker 9](#_Toc529286010)

[Installation de Jenkins et paramétrage de GitHub et Maven 11](#_Toc529286011)

[Installation de Jenkins 11](#_Toc529286012)

[Création d’un token d’authentification dans GitHub 12](#_Toc529286013)

[Configuration du Webhook GitHub dans Jenkins 13](#_Toc529286014)

[Création du repo Nexus 14](#_Toc529286015)

[Installation de Nexus 14](#_Toc529286016)

[Création de repository 14](#_Toc529286017)

[Compilation et publication via Jenkins 16](#_Toc529286018)

[Tests Sonar 17](#_Toc529286019)

[Installation de Sonar 17](#_Toc529286020)

[Exécution des tests via Jenkins 17](#_Toc529286021)

[dependency-check (OWASP) 18](#_Toc529286022)

[Installation du plugin dans Jenkins 18](#_Toc529286023)

[Installation du plugin dans Sonar 18](#_Toc529286024)

[Exécution des tests via Jenkins 19](#_Toc529286025)

[Tests unitaires 20](#_Toc529286026)

[Configuration des tests unitaires 20](#_Toc529286027)

[Exécution des tests via Jenkins 20](#_Toc529286028)

[Affichage des tests unitaires dans Sonar 20](#_Toc529286029)

[Build des images Docker par Jenkins 22](#_Toc529286030)

[Création d'une registry Docker 23](#_Toc529286031)

[Création de la registry en http 23](#_Toc529286032)

[Création de la registry en https (requis pour Kubernetes) 23](#_Toc529286033)

[Gestion des images Docker et de la registry 24](#_Toc529286034)

[Déploiement dans Kubernetes 26](#_Toc529286035)

[Installation de Rancher et Kubernetes 26](#_Toc529286036)

[Installation manuelle d’un master Kubernetes et d’un Node séparé 27](#_Toc529286037)

[Quelques commandes du CNI Weave 29](#_Toc529286038)

[Diverses commandes Kubernetes 29](#_Toc529286039)

[Installation d’un Ingress Controller Kubernetes-Nginx 29](#_Toc529286040)

[Création d’une configuration de déploiement Kubernetes 29](#_Toc529286041)

[Stage Jenkins de déploiement dans Kubernetes 31](#_Toc529286042)

[Spécifier une registry Docker 32](#_Toc529286043)

[Installation Kubernetes via conjure-up sur Ubuntu 32](#_Toc529286044)

# Création de l’environnement de travail sur le poste local

## Initialisation de l’environnement de travail sur le poste local

1. Créer le répertoire du projet sur le disque local (exemple : « *C:\Jenkins-Master-slave-SimpleAPI »* sous Windows)
2. Créer la structure de répertoires suivantes :
   * Jenkins-Master-slave-SimpleAPI
     + src
       - main
         * java

hello

* + - * test
        + java

hello

## Création du code source

Créer les 4 fichiers suivants utilisés pour le code source :

* Jenkins-Master-slave-SimpleAPI/pom.xml
* Jenkins-Master-slave-SimpleAPI/src/main/java/hello/Greeting.java
* Jenkins-Master-slave-SimpleAPI/src/main/java/hello/GreetingController.java
* Jenkins-Master-slave-SimpleAPI/src/main/java/hello/Application.java
* Jenkins-Master-slave-SimpleAPI/src/test/java/hello/ApplicationTests.java

Jenkins-Master-slave-SimpleAPI/**pom.xml** :

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>org.springframework</groupId>

<artifactId>Jenkins-Master-slave-SimpleAPI</artifactId>

<version>0.1.0-SNAPSHOT</version>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.0.5.RELEASE</version>

</parent>

<!-- Nexus repositories configuration -->

<distributionManagement>

<snapshotRepository>

<id>API-javaSpringboot-snapshots</id>

<name>API-javaSpringboot snapshot repository</name>

<url>http://localhost:8081/nexus/content/repositories/API-javaSpringboot-snapshots</url>

</snapshotRepository>

<repository>

<id>API-javaSpringboot-release</id>

<name>API-javaSpringboot release repository</name>

<url>http://localhost:8081/nexus/content/repositories/API-javaSpringboot-release</url>

</repository>

</distributionManagement>

<!-- /Nexus repositories configuration -->

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>com.jayway.jsonpath</groupId>

<artifactId>json-path</artifactId>

<scope>test</scope>

</dependency>

<!-- Unit tests declaration -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

<scope>test</scope>

</dependency>

<!-- /Unit tests declaration -->

</dependencies>

<properties>

<java.version>1.8</java.version>

<dependency.check.report.dir>${project.build.directory}</dependency.check.report.dir>

<sonar.dependencyCheck.htmlReportPath>${project.build.directory}/dependency-check-report.html</sonar.dependencyCheck.htmlReportPath>

</properties>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

<!-- Sonar tests -->

<plugin>

<groupId>org.sonarsource.scanner.maven</groupId>

<artifactId>sonar-maven-plugin</artifactId>

<version>3.5.0.1254</version>

</plugin>

<!-- /Sonar tests -->

<!-- OWASP Dependecy check -->

<plugin>

<groupId>org.owasp</groupId>

<artifactId>dependency-check-maven</artifactId>

<!-- <version>3.3.2</version> -->

<version>5.0.0-M3</version>

<configuration>

<!-- <failBuildOnCVSS>8</failBuildOnCVSS> -->

</configuration>

<executions>

<execution>

<goals>

<goal>check</goal>

</goals>

</execution>

</executions>

</plugin>

<!-- /OWASP Dependecy check -->

<!-- Unit tests declaration -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<configuration>

<useSystemClassLoader>false</useSystemClassLoader>

</configuration>

</plugin>

<!-- Code coverage reporting to Sonar -->

<plugin>

<groupId>org.jacoco</groupId>

<artifactId>jacoco-maven-plugin</artifactId>

<version>0.7.7.201606060606</version>

<executions>

<execution>

<goals>

<goal>prepare-agent</goal>

</goals>

</execution>

<execution>

<id>report</id>

<phase>prepare-package</phase>

<goals>

<goal>report</goal>

</goals>

</execution>

</executions>

</plugin>

<!-- /Code coverage reporting to Sonar -->

<!-- /Unit tests declaration -->

</plugins>

</build>

<!-- Nexus repositories declaration -->

<repositories>

<repository>

<id>spring-releases</id>

<url>https://repo.spring.io/libs-release</url>

</repository>

</repositories>

<pluginRepositories>

<pluginRepository>

<id>spring-releases</id>

<url>https://repo.spring.io/libs-release</url>

</pluginRepository>

</pluginRepositories>

<!-- /Nexus repositories declaration -->

</project>

Jenkins-Master-slave-SimpleAPI/src/main/java/hello/**GreetingController.java** :

package hello;

import java.util.concurrent.atomic.AtomicLong;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.RestController;

@RestController

public class GreetingController {

private static final String template = "Salut, %s!";

private final AtomicLong counter = new AtomicLong();

@RequestMapping("/greeting")

public Greeting greeting(@RequestParam(value="name", defaultValue="World") String name) {

return new Greeting(counter.incrementAndGet(),

String.format(template, name));

}

}

Jenkins-Master-slave-SimpleAPI/src/main/java/hello/**Application.java** :

package hello;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class Application {

public static void main(String[] args) {

SpringApplication.run(Application.class, args);

}

}

Jenkins-Master-slave-SimpleAPI/src/test/java/hello/**ApplicationTests.java**

package hello;

import java.util.Map;

import org.junit.Test;

import org.junit.runner.RunWith;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.boot.test.web.client.TestRestTemplate;

import org.springframework.boot.web.server.LocalServerPort;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.test.context.TestPropertySource;

import org.springframework.test.context.junit4.SpringRunner;

import static org.assertj.core.api.BDDAssertions.then;

/\*\*

\* Basic integration tests for service demo application.

\*

\* @author Dave Syer

\*/

@RunWith(SpringRunner.class)

@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM\_PORT)

@TestPropertySource(properties = {"management.port=0"})

public class ApplicationTests {

@LocalServerPort

private int port;

@Value("${local.management.port:${local.server.port}}")

private int mgt;

@Autowired

private TestRestTemplate testRestTemplate;

@Test

public void shouldReturn200WhenSendingRequestToController() throws Exception {

@SuppressWarnings("rawtypes")

ResponseEntity<Map> entity = this.testRestTemplate.getForEntity(

"http://localhost:" + this.port + "/greeting", Map.class);

then(entity.getStatusCode()).isEqualTo(HttpStatus.OK);

}

}

## Création d'un projet sur GitHub

a) Se connecter sur GitHub et créer un nouveau Repository :

https://github.com/Jenkins-Master-slave-SimpleAPI

b) Dans la fenêtre de création d'un nouveau repository :

Par exemple : Jenkins-Master-slave-SimpleAPI

c) Copier le lien https du repository :

Par exemple : https://github.com/maddoudou22/Jenkins-Master-slave-SimpleAPI.git

## Clone sur le poste/serveur local

1. Installer Git Bash sous Windows et l’exécuter :
   1. Se placer dans le répertoire du projet (exemple : « *C:\Jenkins-Master-slave-SimpleAPI »*).
   2. Exécuter les commandes suivantes :
      * $ git init
      * $ git config --global user.email [you@example.com](mailto:you@example.com)
      * $ git config --global push.default simple
      * $ git config --global credential.helper wincred

*Note : cette commande permet d’éviter de renseigner renseigner le username et mot de passe du compte GitHub chaque push.*

* + - $ git clone [https://github.com/maddoudou22/Jenkins-Master-slave-SimpleAPI.git](https://github.com/maddoudou22/API-javaSpringboot.git)
    - $ git add \*
    - $ git commit -m "First push !"
    - $ git push --set-upstream https://github.com/maddoudou22/Jenkins-Master-slave-SimpleAPI.git master

*Note : pour les push suivants : $ git push https://github.com/maddoudou22/Jenkins-Master-slave-SimpleAPI.git*

1. Simplifier le push de code dans GitHub :
   1. Créer un fichier Jenkins-Master-slave-SimpleAPI/**Git Push.bat**:

echo off

SET nom\_Projet\_Git=Jenkins-Master-slave-SimpleAPI

SET URL\_Projet\_Git=https://github.com/maddoudou22/Jenkins-Master-slave-SimpleAPI.git

SET Repertoire\_Local\_Projet=C:\Jenkins-Master-slave-SimpleAPI

cd %Repertoire\_Local\_Projet%

echo "Git Add ..."

git add -A

echo "Git Commit ..."

git commit -m "latest push from windows"

echo "Git Push ..."

git push %URL\_Projet\_Git%

*Note : à chaque modification d’un fichier du code source, exécuter ce script afin de pousser l’ensemble des modifications dans GitHub.*

## Installation du master Jenkins

*Note : provisionner une instance EC2 qui accueillera le master Jenkins et le serveur Sonar pour les tests de qualité de code (provisionner une EC2 t3.medium de préférence pour éviter les problème de mémoire de JVM lors des tests Sonar).*

Sonar est exécuté sur la master car ses rapports doivent être accessibles à tout moment et il ne supporte pas les architectures distribuées, ni l’import de rapports dans sa base.

1. Installer la version 8 de Java .

$ apt-get install openjdk-8-jdk

1. Sélectionner la version de Java à utiliser par défaut :

$ update-alternatives --config java

1. Récupérer le chemin de la JRE voulue (ex : « */usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java »*) et ajouter la ligne suivante à la fin du fichier *"/etc/environment"* :

$ JAVA\_HOME="/usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java"

1. Sourcer le fichier *"/etc/environment"* :

$ source /etc/environment

1. Installer Maven :

$ apt-get install maven

# (Optionnel) tester le fonctionnement du code dans un Docker

## Initialisation d’une instance de tests

1. Provisionner une image EC2 Ubuntu dans AWS
2. Changer la timezone de l’instance :

$ apt-get update

$ dpkg-reconfigure tzdata

## Compilation du projet

1. Se placer dans le répertoire contenant le POM du projet (exemple : *« ~/Jenkins-Master-slave-SimpleAPI »*).
2. Compiler le projet :

$ mvn clean package

1. Vérifier l’absence d’erreurs et la présence de l’archive .jar dans le dossier target (exemple : *« ~/Jenkins-Master-slave-SimpleAPI/target/Jenkins-Master-slave-SimpleAPI-0.1.0.jar »*).
2. Tester le fonctionnement de l’application : exécuter l'application via Spring Boot :

$ java -jar target/Jenkins-Master-slave-SimpleAPI-0.1.0.jar

1. Tester avec l'URL suivante dans un navigateur :

http://<IP publique instance EC2>:8080/greeting

La réponse doit être :

{"id":1,"content":"Hello, World!"}

1. Tester avec l'URL suivante :

http://<IP publique instance EC2>:8080/greeting?name=User

La réponse doit être :

{"id":2,"content":"Hello, User!"}

## Préparation de l’environnement Docker

1. Installer Docker :

$ apt-get install docker.io

1. Créer un répertoire de déploiement (exemple : « ~/docker-tests »).
2. Se placer dans ce répertoire et créer le dockerfile *"dockerfile-exec-jar"* avec le contenu suivant :

FROM java:8

WORKDIR /root/docker-tests

ADD Jenkins-Master-slave-SimpleAPI-0.1.0.jar Jenkins-Master-slave-SimpleAPI-0.1.0.jar

EXPOSE 8080

CMD java -jar Jenkins-Master-slave-SimpleAPI-0.1.0.jar

*(****Attention*** *: laisser une ligne vide à la fin du fichier)*

1. Créer l’image Docker :

$ docker build -f dockerfile-exec-jar -t Image-Jenkins-Master-slave-SimpleAPI /root/docker-tests/

1. Vérifier la bonne création de l'image :

$ docker images

Doit retourner la liste des images, dont l’une doit être « Image-Jenkins-Master-slave-SimpleAPI ».

1. Exécuter un conteneur basé sur cette image (exécution en arrière-plan) :

$ docker run -d -p 8088:8080 Image-Jenkins-Master-slave-SimpleAPI

1. Vérifier le bon fonctionnement du conteneur :

$ docker ps

Doit retourner la liste des conteneurs en cours d’exécution. Le port 8088 du serveur hôte est mappé sur le port 8080 du conteneur.

1. Tester avec l'URL suivante dans un navigateur :

http://<IP publique instance EC2>:8088/greeting

La réponse doit être :

{"id":1,"content":"Hello, World!"}

*A ajouter dans le pom.xml pour que Maven créé l’archive .jar dans un autre endroit que ‘/target’ :*

*<plugin>*

*<groupId>org.apache.maven.plugins</groupId>*

*<artifactId>maven-jar-plugin</artifactId>*

*<version>2.3.1</version>*

*<configuration>*

*<outputDirectory>${targetRepo}</outputDirectory>*

*</configuration>*

*</plugin>*

## Installation de Jenkins

1. Installer Jenkins :

$ wget -q -O - https://pkg.jenkins.io/debian/jenkins-ci.org.key | sudo apt-key add -

$ echo deb http://pkg.jenkins.io/debian-stable binary/ | sudo tee /etc/apt/sources.list.d/jenkins.list

$ sudo apt-get update

$ apt-get install jenkins

1. Remplacer le port d’écoute par défaut de Jenkins (8080) dans le fichier *« /etc/default/jenkins »* -> Le remplacer par exemple par 9090.
2. Démarrer Jenkins :

$ systemctl start Jenkins

$ systemctl status jenkins

1. Récupérer le mot de passe de Jenkins depuis le fichier *« /var/lib/jenkins/secrets/initialAdminPassword »*.
2. Se connecter à l'URL d'admin de Jenkins (http://<IP publique instance EC2>:9090/) et coller le mot de passe.
3. Laisser l’installation des plugins par défaut.
4. Dans la console d'admin web de **Jenkins**, aller dans « Administrer Jenkins », puis « Gestion des plugins ».
5. Dans l’onglet *‘Disponible’*, sélectionner les plugins *‘Blue Ocean’, ‘pipeline-utility-steps’, ‘Slave SetupPlugin’, 'Amazon EC2' et 'EC2 fleet',* puis cliquer sur le bouton *‘Installer sans redémarrer’*.
6. Créer un fichier '*Jenkinsfile*' à la racine du répertoire de l'application à côté du '*pom.xml'*. Ce fichier contiendra les Stage à exécuter :

Note : ce pipeline est complet ici. On n’utilise pas Nexus dans ce cas car aucun intérêt dans ce tutorial.

***Attention****: bien noter la valeur du* ***label*** *donné à l’agent tout au debut dans la section ‘Agent’ car il faudra l’utiliser dans la configuration du plugin ‘EC2 SpotFleet’.*

Jenkins-Master-slave-SimpleAPI/**Jenkinsfile :**

## pipeline {

## agent {

## node {

## //label '!master' // Le '!master' ne doit pas etre positionne dans le Jenkinsfile ou la flotte EC2 ne sera jamais mise a jour par le plugin.

## label 'Jenkins-slave-forSpot' // Label de flotte specifie dans la configuration du plugin EC2 SpotFleet

## }

## }

## 

## environment {

## package\_version = readMavenPom().getVersion()

## dockerRegistry = "962109799108.dkr.ecr.eu-west-1.amazonaws.com"

## DOCKER\_CACHE\_IMAGE\_VERSION = "latest"

## dockerRepo = "jenkins-master-slave-simpleapi"

## applicationName = 'Jenkins-Master-slave-SimpleAPI' // Same as artifactId in pom.xml

## AWS\_REGION = "eu-west-1"

## AWS\_ACCOUNT\_ID = "962109799108"

## SONAR\_ENDPOINT = "http://34.250.131.213:9000"

## EC2\_LOCAL\_MAVEN\_DEPENDENCIES\_DIRECTORY = "/home/ubuntu/.m2"

## S3\_BUCKET\_MAVEN\_DEPENDENCIES = "s3://jenkinsspotfleetmavencache/Jenkins-Master-slave-SimpleAPI/.m2/"

## }

## 

## stages {

## stage('Download dependencies from S3') {

## steps {

## echo 'Get the cached maven dependencies from an S3 bucket ...'

## //sh 'mkdir -p $EC2\_LOCAL\_MAVEN\_DEPENDENCIES\_DIRECTORY'

## sh 'aws s3 sync $S3\_BUCKET\_MAVEN\_DEPENDENCIES $EC2\_LOCAL\_MAVEN\_DEPENDENCIES\_DIRECTORY'

## }

## }

## 

## stage('Prepa baking') {

## steps {

## echo 'Getting previous image ...'

## sh 'echo \"Si l\'image cache n\'existe pas dans le repo ECR elle est reconstruire, sinon elle est telechargee\"'

## sh 'chmod +x build-docker.sh'

## sh './build-docker.sh $dockerRepo $DOCKER\_CACHE\_IMAGE\_VERSION dockerfile\_basis $AWS\_REGION $AWS\_ACCOUNT\_ID'

## }

## }

## stage('Build') {

## steps {

## echo 'Building ...'

## //sh 'mvn -T 1C -Dmaven.test.skip=true dependency:purge-local-repository clean package'

## sh 'mvn -T 1C -Dmaven.test.skip=true clean package'

## }

## }

## 

## stage('Unit test') {

## steps {

## echo 'Unit testing ...'

## sh 'mvn -T 1C test'

## }

## }

## stage('OWASP - Dependencies check') {

## steps {

## echo 'Check OWASP dependencies ...'

## 

## //sh 'mvn dependency-check:purge'

## sh 'mvn dependency-check:check'

## }

## }

## 

## stage('Sonar - Code Quality') {

## steps {

## echo 'Check Code Quality ...'

## sh 'mvn sonar:sonar -Dsonar.host.url=$SONAR\_ENDPOINT' // -Dsonar.dependencyCheck.reportPath=target/dependency-check-report.xml'

## }

## }

## stage('Bake') {

## steps {

## echo 'Building Docker image ...'

## sh '$(aws ecr get-login --no-include-email --region $AWS\_REGION)'

## sh 'docker build --build-arg PACKAGE\_VERSION=${package\_version} --build-arg APPLICATION\_NAME=${applicationName} -t ${dockerRegistry}/${dockerRepo}:${package\_version} .'

## echo 'Publishing Docker image into the private registry ...'

## sh 'docker push ${dockerRegistry}/${dockerRepo}:${package\_version}'

## }

## }

## 

## stage('Dependencies sync') {

## steps {

## echo 'Copying the maven dependencies to an S3 bucket ...'

## sh 'aws s3 sync $EC2\_LOCAL\_MAVEN\_DEPENDENCIES\_DIRECTORY $S3\_BUCKET\_MAVEN\_DEPENDENCIES'

## }

## }

## }

## 

## }Création d’un token d’authentification dans GitHub

1. Dans le menu général de GitHub, aller dans *‘Settings’*, puis *‘Developer Settings’,* puis *‘Personal access tokens’*
2. Cliquer sur le bouton ‘*Generate New Token’* et saisir le nom « *Jenkins-token* » dans ‘*Token Description*’.
3. Cliquer sur le bouton ‘*Generate New Token’* et saisir le nom « *Jenkins-token* » dans ‘*Token Description*’.
4. Sélectionner les scopes suivants :

* repo (tout)
* admin:repo\_hook (tout)
* gist
* user (tout)

1. Clicker sur "*Generate token*" et noter le token généré (exemple : 2*2ede6581035a0cb521b306bb76e187217e68c76*)

## Configuration du Webhook GitHub dans Jenkins

1. Dans la console d'admin web de **Jenkins**, aller dans « Administrer Jenkins », puis « Configurer le système ».
2. Rechercher le paramètre « URL de Jenkins » et remplacer le port d’écoute par celui modifié précédemment (ex : 9090).

*Note : c’est cette URL que GitHub appellera à chaque commit.*

1. Dans la console Web de **GitHub**, aller dans le repository créé et cliquer sur son bouton *'Settings'*, puis sur *'Webhooks'* dans le menu de gauche. Cliquer sur le bouton *'Add Webhook'*.
2. Dans le champ *'Payload URL'*, coller l'URL de Jenkins , suivie de "*github-webhook*" (exemple : http://52.210.141.11:9090/github-webhook/).
3. Dans "*Content type*", choisir "*application/json*", puis cliquer sur le bouton "*Add Webhook*".
4. Dans la console d'admin web de **Jenkins**, créer un nouvel item.
5. Saisir un nom au projet, puis sélectionner « *Pipeline* ». Cliquer sur le bouton « *Ok* ».
6. Dans la page qui apparaît, dans l’onglet ‘*Général’*, cocher la case ‘*GitHub Projet*’ et saisir l’URL du projet (exemple : *« https://github.com/maddoudou22/Jenkins-Master-slave-SimpleAPI/ »*).
7. Dans la section « *Build triggers* », cocher « *GitHub hook trigger for GITScm polling* ».
8. Dans la section « *Pipeline* », dans ‘*Definition’* sélectionner *‘Pipeline script from SCM’* :
   1. Dans *‘SCM’*, Sélectionner *‘Git’*. Dans ‘Repositories’ :
      1. Dans ‘Repository URL’, saisir l’URL du repo GitHub (exemple : [*https://github.com/maddoudou22/Jenkins-Master-slave-SimpleAPI*](https://github.com/maddoudou22/API-javaSpringboot)).
      2. Dans ‘*Credentials’*, laisser ‘*aucun’* et cliquer sur le bouton ‘*Ajouter’*. Dans ‘*Type’* choisir ‘*Secret text’* et coller le token généré par GitHub (exemple : 2*2ede6581035a0cb521b306bb76e187217e68c76*).
9. Cliquer sur le bouton « *Sauver* ».
10. Dans le menu Jenkins du pipeline créé, cliquer sur le bouton *‘Lancer un build’* pour tester la configuration.
11. Tester le Webhook : modifier une partie du code source sur le poste de travail et exécuter le batch « *Git Push.bat* ».

Dans le projet créé, Jenkins doit détecter le build dans l’historique des builds.

## Installation de Sonar sur le master

1. Dans l’instance EC2 du master, télécharger et démarrer une image Docker de SonarQube :

$ docker pull sonarqube

$ docker run -d --name sonarqube -p 9000:9000 -p 9092:9092 sonarqube

1. Accéder à la console de Sonar : http://<IP publique instance EC2>:9000
2. Se logger avec admin/admin, puis dans l’écran de bienvenue :
   1. Générer un token d’authentification avec un nom (exemple : ***2a2cd66cab7710c20b0f91b258c7e94b8ed40228***)
   2. Sélectionner le langage *Java* pour l’analyse

*Sonar affiche la commande à coller dans un nouveau staging Jenkins pour lancer le scan.*

* 1. Cliquer sur le bouton *‘Finish this tutorial’*.

## dependency-check (OWASP)

### Installation du plugin dans Jenkins

1. Dans la console d'administration de Jenkins, installer le plugin *"OWASP Dependency-Check Plugin"* puis redémarrer Jenkins.

### Installation du plugin dans Sonar

1. Dans l’instance EC2, récupérer le plugin *"*dependency-check-sonar-plugin *"* de Steve Sprigett depuis GitHub afin d'afficher le rapport des vulnérabilités OWASP directement dans Sonar :

$ wget https://github.com/stevespringett/dependency-check-sonar-plugin/releases/download/1.1.1/sonar-dependency-check-plugin-1.1.1.jar

1. Le copier dans le conteneur exécutant Sonar, puis redémarrer Sonar :

$ cp sonar-dependency-check-plugin-1.1.1.jar <ID du conteneur Sonar>:/opt/sonarqube/extensions/plugins/

$ docker restart <ID du conteneur Sonar>

## Build des images Docker par Jenkins

**Vraiment nécesasire sur le Jenkins master dans ce tutotriel ??**

1. Ajouter Jenkins au groupe Docker afin qu’il ait le droit d’utiliser Docker, puis redémarrer Jenkins :

$ usermod -aG docker jenkins

$ service jenkins restart

1. Créer le fichier 'dockerfile' à la racine du projet sur le poste local (sera exporté dans le workspace de Jenkins via GitHub) avec le contenu suivant :

Jenkins-Master-slave-SimpleAPI/**dockerfile :**

FROM java:8

ARG PACKAGE\_VERSION

ARG APPLICATION\_NAME

COPY /target/${APPLICATION\_NAME}-${PACKAGE\_VERSION}.jar ${APPLICATION\_NAME}-${PACKAGE\_VERSION}.jar

EXPOSE 8080

CMD java -jar ${APPLICATION\_NAME}-${PACKAGE\_VERSION}.jar

*Note: laisser une ligne vide à la fin du fichier*

## Création d'une registry Docker dans AWS ECR

1. Dans la console AWS, aller dans le service ECR et cliquer sur le bouton ‘Create Repository’.
2. Indiquer le nom du repository tel qu’il a été défini dans le jenkinsfile (exemple : ‘[jenkins-master-slave-simpleapi](https://eu-west-1.console.aws.amazon.com/ecr/repositories/jenkins-master-slave-simpleapi/?region=eu-west-1)’).

## Création d’une image Slave Jenkins

1. Instancier une EC2 de type ‘t3.medium’ avec une image Ubuntu 18.08 LTS dans le même subnet que le master. Avec une **adresse publique**, car cette instance devra récupérer les sources depuis GitHub.
2. Installer software-properties-common et apt-transport-https :

# apt-get update

# apt install software-properties-common apt-transport-https -y

1. Créer un utilisateur Jenkins :

# useradd -m -s /bin/bash jenkins

# passwd jenkins

-> Saisir un mot de passe

1. Authoriser une connexion via mot de passe (pour le POC uniquement) :

# vi /etc/ssh/sshd\_config

-> Mettre le paramètre ‘PasswordAuthentication’ à 'yes'

1. Redémarrer sshd :

# systemctl restart sshd

1. Installer Docker :

# apt-get install docker.io -y

1. Donner les droits à Jenkins et à l’utilisateur Ubuntu (c’est lui lui qui sera utilisé par le master pour se connecter au slave) d'utiliser Docker :

# usermod -aG docker jenkins

# usermod -aG docker ubuntu

# chmod 777 /var/run/docker.sock

1. Installer la CLI AWS :

# apt install awscli

1. Créer une AMI de l’instance créée (exemple : Name : ‘Jenkins-slave-Spot-autoscalling’)
2. Dans AWS IAM, créer une policy endossée par Jenkins pour la gestion des instances EC2 ( ex : ‘[jenkins-EC2-management-policy](https://console.aws.amazon.com/iam/home?region=eu-west-1#/policies/arn%3Aaws%3Aiam%3A%3A962109799108%3Apolicy%2Fjenkins-EC2-management-policy)’) :

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"ec2:DescribeInstances",

"ec2:TerminateInstances",

"ec2:RequestSpotInstances",

"ec2:DeleteTags",

"ec2:CreateTags",

"ec2:DescribeRegions",

"ec2:RunInstances",

"ec2:DescribeSpotInstanceRequests",

"ec2:StopInstances",

"ec2:DescribeSecurityGroups",

"ec2:GetConsoleOutput",

"ec2:DescribeSpotPriceHistory",

"ec2:DescribeImages",

"ec2:CancelSpotInstanceRequests",

"ec2:ModifySpotFleetRequest",

"ec2:StartInstances",

"ec2:DescribeAvailabilityZones",

"ec2:DescribeSubnets",

"ec2:DescribeKeyPairs",

"ec2:\*"

],

"Resource": "\*"

},

{

"Effect": "Allow",

"Action": [

"iam:ListRoles",

"iam:ListInstanceProfiles",

"iam:PassRole"

],

"Resource": "\*"

}

]

}

*Note : revoir les permissions pour se passer de "ec2:\*".*

1. Dans AWS IAM, créer un rôle qui sera endossé par le Master Jenkins pour la gestion des instances EC2 ( ex : ‘[jenkins-EC2-management-role](https://console.aws.amazon.com/iam/home?region=eu-west-1#/policies/arn%3Aaws%3Aiam%3A%3A962109799108%3Apolicy%2Fjenkins-EC2-management-policy)’). Déclarer la policy précédemment créée à ce rôle.
2. Dans le menu AWS EC2, sélectionner l’instance Jenkins Master, puis cliquer sur le bouton ‘Actions’ -> Instance Settings -> Attach/Replace IAM Role.
   * Dans « IAM Role », sélectionner le rôle précédemment créé et cliquer sur ‘Save’.
3. Dans AWS S3, créer un bucket ‘jenkinsspotfleetmavencache’ : il servira à stocker le cache des dépendances Maven pour les différentes instances.
4. Sélectionner le bucket créé, puis cliquer sur l’onglet ‘Permissions’, puis sélectionner ‘Bucket Policy’. L’exemple suivant é été créé pour un bucket précédemment créé portant le nom ‘jenkinsspotfleetmavencache’ : remplacer si besoin. Dans ‘Principale’, indiquer le’ARN du rôle créé pour les instances Jenkins slave (exemple : ‘jenkins-EC2-spotinstance-assumed-role’).
   * Coller la policy suivante :

{

"Version": "2008-10-17",

"Statement": [

{

"Sid": "AllowPublicRead",

"Effect": "Allow",

"Principal": {

"AWS": "arn:aws:iam***::<AWS account Number>***:role/jenkins-EC2-spotinstance-assumed-role"

},

"Action": "s3:\*",

"Resource": [

"arn:aws:s3:::jenkinsspotfleetmavencache",

"arn:aws:s3:::jenkinsspotfleetmavencache/\*"

]

}

]

}

1. Dans AWS IAM, créer un utilisateur endossant la policy précédemment créée ( ex : ‘[jenkins-EC2-management-policy](https://console.aws.amazon.com/iam/home?region=eu-west-1#/policies/arn%3Aaws%3Aiam%3A%3A962109799108%3Apolicy%2Fjenkins-EC2-management-policy)’).
   * . Name : 'jenkins-spot-fleet'
   * . Access Type : sélectionner 'Programmatic Access'
   * Cliquer sur le bouton 'Next: permission'
   * . Cliquer sur le bouton 'Attach existing policies directly' et choisir la policy précédemment créée ('jenkins-EC2-management-policy')
   * Cliquer sur les boutons 'Next:Tags', 'Next: Review' et 'Create User'.

-> Noter L'Access Key ID et le Secret Access Key de l'utilisateur créé.

1. Dans AWS IAM, créer la policy qui sera endossée par les slave Jenkins ( ex : ‘[jenkins-EC2-management-policy](https://console.aws.amazon.com/iam/home?region=eu-west-1#/policies/arn%3Aaws%3Aiam%3A%3A962109799108%3Apolicy%2Fjenkins-EC2-management-policy)’). Le slave aura les droits pour :
   1. S3 : pour la synchronisation des dependances Maven avec un Bucket via la commande aws s3 sync.
   2. ECR : pour l ajout d images Docker dans un repo ECR.
   * L’exemple suivant é été créé pour un bucket précédemment créé portant le nom ‘jenkinsspotfleetmavencache’ : remplacer si besoin.

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"ecr:\*"

],

"Resource": "\*"

},

{

"Effect": "Allow",

"Action": [

"s3:GetObject",

"s3:List\*",

"s3:ListBucket",

"s3:PutObject",

"s3:PutObjectAcl",

"s3:DeleteObject"

],

"Resource": [

"arn:aws:s3:::jenkinsspotfleetmavencache",

"arn:aws:s3:::jenkinsspotfleetmavencache/\*"

]

}

]

}

. Name : ‘jenkins-slave-spot-autoscalling-assumed-policy'

1. Dans AWS IAM, créer un rôle qui sera endossée par les slave Jenkins ( ex : ‘jenkins-slave-spot-autoscalling-assumed-role’).
   * Sélectionner AWS Service, puis EC2.
   * Cliquer sur le bouton 'Next: permission'
   * Cliquer sur le bouton 'Attach existing policies directly' et choisir la policy précédemment créée (‘jenkins-slave-spot-autoscalling-assumed-role’)
   * Cliquer sur les boutons 'Next:Tags', 'Next: Review' et 'Create Role.
2. Dans la console AWS, dans EC2, aller dans l'onglet 'Spot Requests' et cliquer sur le bouton 'Request Spot Instances' :

* Sélectionner 'Flexible workloads'
* Dans AMI, sélectionner l'AMI du Slave Jenkins précédemment créée (ex : ‘Jenkins-slave-Spot-autoscalling’).
* Choisir le type d'instance à utiliser (ex : ‘t3a.medium’).
* Sélectionner les paramètres réseau (VPC et Subnet) et la keypair à utiliser.
* Dans 'Additional Configurations' :
  + Sélectionner le Security Group à utiliser
  + S’assurer que ‘Auto-assign IPv4 public IP’ est à ‘Enabled’
  + IAM instance profile**;** Sélectionner le role à endosser par les instances Spot (ex : 'jenkins-slave-spot-autoscalling-assumed-role')
* Mettre la ‘Total Target Capacity' à 0
* Cocher la case 'Maintain target capacity'
* Cliquer sur 'Launch'

1. Dans la GUI Jenkins du Master, aller dans 'Administrer Jenkins' -> 'Configurer le système'. Dans la section 'Cloud', cliquer sur le bouton 'Ajouter un nouveau Cloud' et sélectionnez 'Amazon SpotFleet'.

* 'Name' : saisir un nom (exemple : 'Jenkins - EC2 FleetCloud')
* 'Amazon EC2 Credentials' : sélectionner “Add”, puis “Jenkins”
  + “Type” : “AWS Credentials”
  + ID : indiquer un nom à cet identifiant (ex : ‘aws\_keypair\_jenkins’)
  + Remplir les champs “Access Key ID” et “Secret Access Key” de l'utilisateur créé dans IAM.
  + Cliquer sur 'Ajouter'.
* 'Region' : Sélectionner la région (ex : 'eu-west-1')
* 'Spot Fleet' : sélectionner la flotte créée dan AWS dans la liste déroulante (peut prendre plusieurs minutes et même une erreur 400 en attendant que la flotte soit accessible. Attendre au moins 5 minutes).
* 'Launcher' : cliquer sur le bouton 'Ajouter' et sélectionner 'Jenkins'
  + Type : choisir 'SSH Username with private key'
  + 'Key Pair's Private Key' : Coller la clé privée PEM avec laquelle le slave Jenkins sera provisionné.
  + 'username' : saisir le nom de l'utilisateur pour se connecter à une instance (ex : 'ubuntu')
  + 'ID' : saisir un identifiant de clé (ex : le nom de la clé)
  + 'Passphrase' : saisir la passphrase associée à la clé
* Cliquer sur le bouton ‘Avancé’ :
  + Maximum Number of Retries : 3
  + Seconds To Wait Between Retries : 5
* Laisser la case 'Connect using private IP' décochée.
* Label : saisir le label qui a été indiqué dans le Jenkinsfile pour indiquer l'Agent à exécuter (ex : 'Jenkins-slave-forSpot')
  + Jenkins fielsystem root : laisser vide
  + Max Idle Minutes Before Scaledown : 1
  + Minimum Cluster Size : 0
  + Maximum Cluster Size : 2
* Cliquer sur le bouton 'Enregistrer'

1. Tester la configuration en lançant un build depuis Jenkins et vérifier que la Desired Capacity dans la console de AWS Spot passe à ‘1’.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Note pour la purge du cache des dépendances dans S3 : la commande aws s3 sync ne delete pas par defaut, supprimer le cache depuis le bucket S3 pour purge.*