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:\serverlessToolchainJava »* sous Windows)
2. Créer la structure de répertoires suivantes :
   * serverlessToolchainJava
     + src
       - main
         * java

hello

* + - * test
        + java

hello

## Création du code source

Créer les 4 fichiers suivants utilisés pour le code source :

* serverlessToolchainJava/pom.xml
* serverlessToolchainJava/src/main/java/hello/Greeting.java
* serverlessToolchainJava/src/main/java/hello/GreetingController.java
* serverlessToolchainJava/src/main/java/hello/Application.java
* serverlessToolchainJava/src/test/java/hello/ApplicationTests.java

serverlessToolchainJava/**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>serverlessToolchainJava</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>

<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>

*Test avec un pom.xml simplifié :* ***serverlessToolchainJava/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>serverlessToolchainJava</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>*

*<properties>*

*<maven.compiler.target>1.8</maven.compiler.target>*

*<maven.compiler.target>1.8</maven.compiler.target>*

*<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>*

*</properties>*

*<dependencies>*

*<dependency>*

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

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

*</dependency>*

*</dependencies>*

*<build>*

*<plugins>*

*<plugin>*

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

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

*</plugin>*

*<plugin>*

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

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

*<version>3.1.0</version>*

*<configuration>*

*<createDependencyReducedPom>false</createDependencyReducedPom>*

*</configuration>*

*<executions>*

*<execution>*

*<phase>package</phase>*

*<goals>*

*<goal>shade</goal>*

*</goals>*

*</execution>*

*</executions>*

*</plugin>*

*</plugins>*

*</build>*

*</project>*

serverlessToolchainJava/src/main/java/hello/**Greeting.java** :

package hello;

public class Greeting {

private final long id;

private final String content;

public Greeting(long id, String content) {

this.id = id;

this.content = content;

}

public long getId() {

return id;

}

public String getContent() {

return content;

}

}

serverlessToolchainJava/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));

}

}

serverlessToolchainJava/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);

}

}

serverlessToolchainJava/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/serverlessToolchainJava

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

Par exemple : serverlessToolchainJava

c) Copier le lien https du repository :

Par exemple : https://github.com/maddoudou22/serverlessToolchainJava.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:\*serverlessToolchainJava*»*).
   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/serverlessToolchainJava.git
    - $ git add \*
    - $ git commit -m "First push !"
    - $ git push --set-upstream https://github.com/maddoudou22/serverlessToolchainJava.git master

*Note : pour les push suivants : $ git push https://github.com/maddoudou22/serverlessToolchainJava.git*

1. Simplifier le push de code dans GitHub :
   1. Créer un fichier API-javaSpringboot/**Git Push.bat**:

echo off

SET nom\_Projet\_Git= serverlessToolchainJava

SET URL\_Projet\_Git=https://github.com/maddoudou22/serverlessToolchainJava.git

SET Repertoire\_Local\_Projet=C:\serverlessToolchainJava

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.*

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

## Initialisation d’une instance de tests

1. Créer un rôle qui sera endossé par l'instance EC2 pour pousser/récupérer des conteneurs dans la registry ECR d'AWS :
   1. La policy associée à ce rôle sera la suivante :

{

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

"Statement": [

{

"Effect": "Allow",

"Action": [

"ecr:\*"

],

"Resource": [

"\*"

]

}

]

}

1. Provisionner une image EC2 Ubuntu dans AWS et endossant le rôle créé précédemment.
2. Changer la timezone de l’instance :

$ apt-get update

$ dpkg-reconfigure tzdata

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

1. Installer la CLI AWS (sera utilisé epour pousser/récupérer des images Docker sur la registry ECR):

$ apt-get install python2.7

$ curl -O https://bootstrap.pypa.io/get-pip.py

$ python2.7 get-pip.py

$ pip install awscli

$ aws --version

*-> Doit retourner le numéro de version de AWS CLI*

## Compilation du projet

1. Recréer l'arborescence du projet sur l'instance EC2 *(ex: ~/serverlessToolchainJava/...)*
2. Se placer dans le répertoire contenant le POM du projet (exemple : *« ~/serverlessToolchainJava »*).
3. Compiler le projet :

$ apt-get install maven

$ mvn clean package

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

$ java -jar target/serverlessToolchainJava-0.1.0-SNAPSHOT.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. Copier l'archive JAR dans ce nouveau répertoire docker-test et s'y placer

$ cp target/serverlessToolchainJava-0.1.0-SNAPSHOT.jar ~/docker-test

$ cd ~/docker-test

1. Se placer dans ce répertoire et créer le dockerfile *"dockerfile-exec-jar"* avec le contenu suivant :

FROM java:8

WORKDIR /root/docker-tests

ADD serverlessToolchainJava-0.1.0-SNAPSHOT.jar serverlessToolchainJava-0.1.0-SNAPSHOT.jar

EXPOSE 8080

CMD java -jar serverlessToolchainJava-0.1.0-SNAPSHOT.jar

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

1. Créer l’image Docker :

$ docker build -f dockerfile-exec-jar -t serverlesstoolchainjava /root/docker-tests/

1. Vérifier la bonne création de l'image :

$ docker images

Doit retourner la liste des images, dont l’une doit être « serverlesstoolchainjava ».

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

$ docker run -d -p 8088:8080 serverlesstoolchainjava

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>*

## Enregistrement de l'image Docker dans la registry ECS

1. Vérifier que l'instance EC2 à partir de laquelle on pousse l'image endosse un rôle lui permettant de pousser des images dans ECR.
2. Faire une demande de login auprès d'ECR via la commande suivante :

$ aws ecr get-login --no-include-email

Cette commande renvoie une commande de loggin. Faire un copier/coller et l'exécuter (commande de type *"docker login -u AWS -p password https://aws\_account\_id.dkr.ecr.region.amazonaws.com")*

*Essayer tout simplement avec la commande suivante :*

$(aws ecr get-login --no-include-email --region $AWS\_DEFAULT\_REGION)

1. Dans la console AWS, créer un repository en spécifiant ' [serverlesstoolchainjava](https://eu-west-1.console.aws.amazon.com/ecr/repositories/serverlesstoolchainjava/?region=eu-west-1)' comme nom de repository. Récupérer l'URI du repository *(<Account ID>.dkr.ecr.<region>.amazonaws.com/serverlesstoolchainjava*)
2. Récupérer l'ID de l'image Docker buildée précédemment avec la commande 'docker images'

$ docker images

1. Ajouter un tag à une image existante (image ID = 518a41981a6a, dans la future registry registry ECR, nom de l'image = serverlesstoolchainjava, tag = 0.1.0)

$ docker tag 518a41981a6a <AccountID>.dkr.ecr.<region>.amazonaws.com/serverlesstoolchainjava:0.1.0

1. Pousser l'image du conteneur créé dans le repository ECR :

$ docker push <AccountID>.dkr.ecr.<region>.amazonaws.com/serverlesstoolchainjava:0.1.0

1. hj

## Récupération de l'image Docker dans la registry ECS

1. Répéter les étapes 1) et 2) du chapitre "Enregistrement de l'image Docker dans la registry ECS" pour se logger auprès d'ECR.
2. Récupérer l'image serverlesstoolchainjava avec le tag 0.1.0 depuis le repository ECR :

$ docker pull <AccountID>.dkr.ecr.<region>.amazonaws.com/serverlesstoolchainjava:0.1.0

# Création du pipeline

## xxxx

1. xxxx