Creation d’une toolchain serverless avec installation de Sonar dans EFS

Table des matières

[Introduction 3](#_Toc559767)

[Fonctionnement de l'ensemble 3](#_Toc559768)

[Déploiement de la toolchain 3](#_Toc559769)

[Exécution de la toolchain 3](#_Toc559770)

[Création de l’environnement de travail sur le poste local 4](#_Toc559771)

[Initialisation de l’environnement de travail sur le poste local 4](#_Toc559772)

[Création du code source 4](#_Toc559773)

[Création d'un projet sur GitHub 12](#_Toc559774)

[Clone sur le poste/serveur local 12](#_Toc559775)

[(Optionnel) tester le fonctionnement du code dans un Docker 14](#_Toc559776)

[Initialisation d’une instance de tests 14](#_Toc559777)

[Compilation du projet 15](#_Toc559778)

[Création du filesystem EFS contenant Sonar 16](#_Toc559779)

[Provisioning du point de montage EFS 16](#_Toc559780)

[Installation manuelle de Sonar 16](#_Toc559781)

[Création de l'image Docker customisée pour Codebuild 18](#_Toc559782)

[Création de la Registry ECR 18](#_Toc559783)

[Création de l'image Docker custom 18](#_Toc559784)

[Enregistrement de l'image Docker dans la registry ECS 19](#_Toc559785)

[(Optionnel) Récupération de l'image Docker dans la registry ECS 20](#_Toc559786)

[Création du pipeline 21](#_Toc559787)

[Fichiers de build utilisés par CodeBuild 21](#_Toc559788)

[Template CloudFormation 25](#_Toc559789)

[Déploiement de la configuration réseau et du pipeline complets 25](#_Toc559790)

[Déploiement de la stack 42](#_Toc559791)

# Introduction

L'objectif est de créer une toolchain serverless permettant de faire des tests unitaires, test OWASP et tests de vulnérabilité, tout en gardant l'historique dans Sonar si besoin.

Sonar est installé dans un filesystem hébergé par EFS. A chaque build de l'application, ce filesystem est monté par 'image Docker utilisée par CodeBuild afin d'utiliser les paramètres de Sonar et préserver l'historique. Ce filesystem peut également être monté sur une EC2 afin de modifier les configurations de Sonar et consulter l'historique des tests.

## Fonctionnement de l'ensemble

### Déploiement de la toolchain

* **Pré-requis** :
  + Le **filesystem EFS** doit déjà avoir été créé **avec le répertoire "/opt/sonar"** (cf modop dans les chapitres plus bas). Car pas de possibilité d'importer des données directement depuis S3 vers EFS via une Lambda (montage EFS impossible).
  + Le **repository** de l'image Docker de build customisée pour CodeBuild doit exister ainsi que **l'image de build** en question (cf modop dans les chapitres plus bas).
* Un seul template permet de créer toutes les ressources réseau et la toolchain. Afin de monter un volume EFS sur l'image Docker de build utilisée par CodeBuild, il faut déclarer un VPC, subnet privé et Security Group dans la configuration de CodeBuild. CodeBuild n'accepte de récupérer les sources auprès que GitHub que depuis un subnet privé via une NAT Gateway (utilisation d'une Internet Gateway impossible).
* Une fois le template déployé, le filesystem EFS doit être modifié pour être dans le même VPC et le même subnet privé que CodeBuild.

### Exécution de la toolchain

* *Note : le premier build échoue en attendant que l'utilisateur configure EFS.*
* Les sources sont récupérées via GitHub.
* L'image de build customisée est exécutée par CodeBuild et le cache Maven est récupéré depuis S3 (pas au premier lancement : cache vide).(toutes les étapes sont détaillées dans le buildspec):
  + Dans la phase de **pre-build**, le filesystem NFS est monté et Sonar est démarré en arrière plan (requière 1 minute pour se lancer).
  + Un script vérifie si l'image Docker de base de l'application existe dans son repository. A la première exécution ce n'est pas le cas : elle est donc recréée à partie du dockerfile "dockerfile\_basis" directement dans la registry locale. C'est fait en tâche de fond. Durant les builds suivant l'image sera dispo dans le repo, elle sera donc 'pullée' en tâche de fond.
  + Dans la phase de **build**, Maven compile le code, puis exécute les tests unitaires et les tests OWASP. Au premier build c'est très long car il télécharge toutes les dépendances.
  + *Les étapes de démarrage de Sonar, création (ou récupération) de l'image de base et compilation/tests de maven sont exécutés en parallèle.*
  + Une fois les tests terminés et Sonar démarré, les tests Sonar sont exécutés.
  + Dans la phase de **Post-build**, l'image Docker est buildée à partir de l'image de base (reconstruire ou téléchargée) afin de gagner du temps (les 9/10ème des layer étant déjà prêts). L'image est alors poussée dans le Repo ECR de l'application.
  + Enfin, toutes les dépendances et plugins Maven téléchargés sont uploadés dans S3 afin d'être récupérées au prochain build et réduire largement le temps de build.

Note : EFS ne permet de mettre à disposition des points de montage que dans un VPC. C'est pourquoi l'image de build customisée pour CodeBuild est déployée par CodeBuild dans un VPC (cf configuration de CodeBuild dans les templates CloudFormation). **Le temps de provisioning de l'image de Builds de CodeBuild est rallongé d'une quinzaine de secondes lorsqu'on spécifie un VPC par rapport au provisioning sans VPC**.

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

hello

* + - * test
        + java

hello

## Création du code source

Créer les 4 fichiers suivants utilisés pour le code source :

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

serverlessEFSToolchainJava/**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>serverlessefstoolchainjava</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>

<!-- Possibilité de choisir le format du rapport généré : ALL, JSON, CSV, HTML, etc.

<format>JSON</format>

-->

<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>

<version>3.0.0-M3</version>

<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>

serverlessEFSToolchainJava/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/serverlessEFSToolchainJava

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

Par exemple : serverlessEFSToolchainJava

c) Copier le lien https du repository :

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

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

1. Simplifier le push de code dans GitHub :
   1. Créer un fichier C:\serverlessEFSToolchainJava\**Git Push.bat**:

echo off

SET nom\_Projet\_Git=serverlessEFSToolchainJava

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

SET Repertoire\_Local\_Projet=C:\awsLambda\java\serverlessEFSToolchainJava

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: ~/serverlessEFSToolchainJava/...)*
2. Se placer dans le répertoire contenant le POM du projet (exemple : *« ~/serverlessEFSToolchainJava »*).
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/serverlessEFSToolchainJava-0.1.0-SNAPSHOT.jar »*).
2. Tester le fonctionnement de l’application : exécuter l'application via Spring Boot :

$ java -jar target/serverlessEFSToolchainJava-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!"}

# Création du filesystem EFS contenant Sonar

## Provisioning du point de montage EFS

Dans la console AWS EFS, provisionner un point de montage EFS en spécifiant un VPC, subnet et un Security Group identique à l'EC2 qui sera utilisée pour installer Sonar.

Récupérer le nom DNS d'EFS ($EFS\_DNS)

## Installation manuelle de Sonar

Dans la console EC2, provisionner une instance EC2 avec une Ubuntu 18.04 (type t3.medium ou t3.large recommandée car Sonar est gourmand).

*Selon le tuto : https://www.alibabacloud.com/blog/easy-guide-to-install-sonarqube-on-ubuntu-16-04\_593923*

1. Installer de quelques packages pre-requis :

$ apt-get install -y software-properties-common nfs-common

1. Créer le répertoire sonar :

$ mkdir /opt/sonar

1. Monter le volume EFS :

$ mount -t nfs4 -o nfsvers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600,retrans=2 $EFS\_DNS:/ /opt/sonar

1. Installer Sonar :

$ mkdir /opt/sonar

$ cd /opt/sonar

$ wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-7.3.zip

$ unzip sonarqube-7.3.zip

1. Créer l'utilisateur Sonar et lui donner les droit sur le répertoire de Sonar:

$ adduser sonar

$ chown -R sonar:sonar /opt/sonar

1. *Vérifier l'UID et le GUID de l'utilisateur Sonar : il faudra reporter les même dans le buildspec.yml afin d'éviter les problèmes de droit lors du montage avec EFS.*
2. Modifier le script d'exécution de Sonar pour que Sonar soit lancé via l'utilisateur 'Sonar' (problèmes d'exécution avec 'root).

$ vi /opt/sonar/sonarqube-7.3/bin/linux-x86-64/sonar.sh

*- Décommenter la ligne "#RUN\_AS\_USER" pour la remplacer par "RUN\_AS\_USER=sonar"*

1. Démarrer Sonar et vérifier qu'il se lance bien en se connectant à la console de Sonar : (http://<IP publique instance EC2>:9000)

$ /opt/sonar/sonarqube-7.6/bin/linux-x86-64/sonar.sh start

# Création de l'image Docker customisée pour Codebuild

## Création de la Registry ECR

Dans la console AWS ECR, créer un nouveau repository ' [codebuild-custom-java](https://eu-west-1.console.aws.amazon.com/ecr/repositories/codebuild-custom-java/?region=eu-west-1)'

Aller dans les 'Permissions' de ce repository et créer les règles suivantes en JSON afin que CodeBuild ait la permission de récupérer des images de ce repo :

{

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

"Statement": [

{

"Sid": "CodeBuildAccess",

"Effect": "Allow",

"Principal": {

"Service": "codebuild.amazonaws.com"

},

"Action": [

"ecr:BatchCheckLayerAvailability",

"ecr:BatchGetImage",

"ecr:GetDownloadUrlForLayer"

]

}

]

}

## Création de l'image Docker custom

Dans la console EC2, provisionner une instance EC2 avec une Ubuntu 18.04 (type t2.micro) avec un **rôle ayant tous les droits sur ECR**.

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/serverlessEFSToolchainJava-0.1.0-SNAPSHOT.jar ~/docker-test

$ cd ~/docker-test

1. Se placer dans ce répertoire et créer le dockerfile *"****dockerfile\_docker-custom-codebuild****"* avec le contenu suivant :

FROM ubuntu:18.04

RUN apt-get update

RUN apt-get install -y tzdata

RUN ln -fs /usr/share/zoneinfo/Europe/Paris /etc/localtime

RUN dpkg-reconfigure --frontend noninteractive tzdata

RUN apt-get install curl apt-transport-https software-properties-common -y

RUN curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-key add -

RUN add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable"

RUN apt-get update

RUN apt-cache policy docker-ce

RUN apt-get install docker-ce -y

RUN apt-get install openjdk-8-jdk -y

RUN apt-get install nfs-common -y

RUN echo "JAVA\_HOME=\"/usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java\"" >> /etc/environment

RUN apt-get install maven -y

RUN apt-get install python2.7 -y

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

RUN python2.7 get-pip.py

RUN pip install awscli

RUN adduser --disabled-password --gecos "" sonar \

&& mkdir /opt/sonar \

&& chown sonar:sonar /opt/sonar

RUN apt-get purge -y curl \

software-properties-common \

python2.7

RUN apt-get clean \

&& rm -rf /var/lib/apt/lists/\*

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

***Note*** *: l'écriture de ce fichier est largement perfectible, déjà en limitant le nombre de 'RUN'*

1. Créer l’image Docker :

$ docker build -f dockerfile\_docker-custom-codebuild -t <URI\_repo\_ECR>:<version> .

*(exemple : build -f dockerfile\_docker-custom-codebuild -t 962109799108.dkr.ecr.eu-west-1.amazonaws.com/codebuild-custom-java:0.3)*

1. Vérifier la bonne création de l'image :

$ docker images

Doit retourner la liste des images, dont l’une doit être « *962109799108.dkr.ecr.eu-west-1.amazonaws.com/codebuild-custom-java* ».

## 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. Installer la CLI AWS :

$ apt-get install python2.7 -y

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

$ python2.7 get-pip.py

$ pip install awscli

1. Faire une demande de login auprès d'ECR via la commande suivante :

$(aws ecr get-login --no-include-email --region eu-west-1)

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

$ docker push <AccountID>.dkr.ecr.<region>.amazonaws.com/<URI\_repo\_ECR>:<version>

*(exemple : docker push 962109799108.dkr.ecr.eu-west-1.amazonaws.com/codebuild-custom-java:0.3)*

1. Vérifier que l'image est disponible dans ECR.

## (Optionnel) Récupération de l'image Docker dans la registry ECS

1. Faire une demande de login auprès d'ECR via la commande suivante :

$(aws ecr get-login --no-include-email --region eu-west-1)

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

$ docker pull <AccountID>.dkr.ecr.<region>.amazonaws.com/<URI\_repo\_ECR>:<version>

*(exemple : docker pull 962109799108.dkr.ecr.eu-west-1.amazonaws.com/codebuild-custom-java:0.3)*

# Création du pipeline

## Fichiers de build utilisés par CodeBuild

1. A la racine du répertoire du projet, créer le fichier "buildspec.yml" qui sera utilisé par CodeBuild pour exécuter les commande requises pour la compilation, la création du conteneur et son enregistrement dans la registry ECR (CodeBuild lui-même déploie un conteneur puis exécute les commandes spécifiées dans ce fichier).

Ce fichier ne contient que des variables d'environnement transmises par CodeBuild : il n'a jamais besoin d'être modifié quelque soit la version du projet utilisée ou même l'instance de projet *(pas sûr pour cette version ...).*

serverlessEFSToolchainJava/**buildspec.yml** :

version: 0.2

phases:

pre\_build:

commands:

- echo Pre-Build started on `date`

- echo "Demarrage de Docker necessaire sur cette image"

- service docker start

- echo "Verification que les UID et GID de Sonar sont bien identiques entre l'image CodeBuild et l'instance EC2 via laquelle Sonar a ete installe, sinon problemes de droits en montage et Sonar ne se lancera pas correctement ..."

- usermod -u 1001 sonar

- groupmod -g 1001 sonar

- echo "Montage du volume EFS contenant le repertoire de Sonar"

- mount -t nfs4 -o nfsvers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600,retrans=2 $EFS\_DNS:/ $EFS\_DIR

- echo "Supression d un eventuel PID subsistant"

- rm -f /opt/sonar/sonarqube-7.3/bin/linux-x86-64/./SonarQube.pid

- echo "demarrage de Sonar"

- /opt/sonar/sonarqube-7.3/bin/linux-x86-64/sonar.sh start

- echo "Recuperation des metadonnees de l application (version et nom du package)"

- PACKAGE\_VERSION=$(grep -m 1 '<version>' pom.xml | awk -F">" '{print $2}' | awk -F"<" '{print $1}')

- APPLICATION\_NAME=$(grep -m 1 '<artifactId>' pom.xml | awk -F">" '{print $2}' | awk -F"<" '{print $1}')

- echo "Recuperation de la precedente image de l application (base initiale) des maintenant pour gagner du temps dans la phase post-build. En arriere plan afin de paralleliser cette action et le demarrage de Sonar"

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

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

- chmod +x build-docker.sh

- ./build-docker.sh $IMAGE\_REPO\_NAME $DOCKER\_CACHE\_IMAGE\_VERSION dockerfile\_basis $AWS\_REGION $AWS\_ACCOUNT\_ID

build:

commands:

- echo Build started on `date`

- export MAVEN\_OPTS="-Dmaven.repo.local=/opt/sonar/sonarqube-7.3/cache" # Trop lent avec EFS : a ne pas utiliser

- echo "Compilation du code source ..."

- mvn -T 1C -Dmaven.test.skip=true clean package

- echo "Demarrage des tests unitaires ..."

- mvn -T 1C test

- echo "Demarrage des tests OWASP ..."

- mvn dependency-check:check

- echo "Attente du demarrage de Sonar ..."

# Si l'image est une Alpine : "timeout -t 60 ...". Si l'image est une Ubuntu : "timeout 60 ..."

- timeout 60 sh -c 'until ps aux | grep [o]rg.sonar.ce.app.CeServer; do sleep 1; done'

- echo "Demarrage des tests Sonar ..."

- mvn sonar:sonar #-Duser.home=/opt/sonar/sonarqube-7.3/cache/

- cp target/$APPLICATION\_NAME-$PACKAGE\_VERSION.jar .

post\_build:

commands:

- echo Post-Build started on `date`

- echo "Verification de la presence de l'image Docker dans la registry locale (elle a du avoir le temps de se reconstruire ou se telecharger)"

# Si l'image est une Alpine : "timeout -t 60 ...". Si l'image est une Ubuntu : "timeout 60 ..."

- timeout 60 sh -c 'until docker images | grep $IMAGE\_REPO\_NAME | grep $DOCKER\_CACHE\_IMAGE\_VERSION; do sleep 1; done'

- echo "Modification du dockerfile pour y indiquer l'image de base a utiliser pour le build afin de beneficier des layer mis en cache localement"

- sed -i.bak "s/BASIS\_IMAGE/$AWS\_ACCOUNT\_ID.dkr.ecr.$AWS\_REGION.amazonaws.com\/$IMAGE\_REPO\_NAME:$DOCKER\_CACHE\_IMAGE\_VERSION/g" dockerfile

- echo "Build de l'image Docker de l'application a partir de l'image de base pour accelerer le processus - Les 3/4 des donnees sont deja en cache"

- docker build --cache-from $AWS\_ACCOUNT\_ID.dkr.ecr.$AWS\_REGION.amazonaws.com/$IMAGE\_REPO\_NAME:$DOCKER\_CACHE\_IMAGE\_VERSION --build-arg PACKAGE\_VERSION=$PACKAGE\_VERSION --build-arg APPLICATION\_NAME=$APPLICATION\_NAME -t $AWS\_ACCOUNT\_ID.dkr.ecr.$AWS\_REGION.amazonaws.com/$IMAGE\_REPO\_NAME:$PACKAGE\_VERSION .

- echo "Pushing the Docker image..."

- docker push $AWS\_ACCOUNT\_ID.dkr.ecr.$AWS\_REGION.amazonaws.com/$IMAGE\_REPO\_NAME:$PACKAGE\_VERSION

cache:

paths:

- '/root/.m2/\*\*/\*'

artifacts:

files: imagedefinitions.json

1. A la racine du répertoire du projet, créer le fichier "dockerfile" qui spécifie les modalité de construction du conteneur Docker de l'application à builder.

serverlessEFSToolchainJava/**dockerfile** :

FROM BASIS\_IMAGE

ARG PACKAGE\_VERSION

ARG APPLICATION\_NAME

ENV PACKAGE\_VERSION ${PACKAGE\_VERSION}

ENV APPLICATION\_NAME ${APPLICATION\_NAME}

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

EXPOSE 8880

CMD ["sh", "-c", "java -jar ${APPLICATION\_NAME}-${PACKAGE\_VERSION}.jar"]

1. A la racine du répertoire du projet, créer le fichier "dockerfile\_basis" qui spécifie les modalité de construction de l'image de base de l'application afin d'accélérer le build de l'image Docker de l'application pour les build suivants (utilisation des layers existants) - Surtout intéressant si l'image de base est grosse, pas comme dans cet exemple.

serverlessEFSToolchainJava/**dockerfile\_basis** :

FROM java:8

1. A la racine du répertoire du projet, créer le script "build-docker.sh" qui sera appelé par le fichier 'buildspec.yml' afin de vérifier si l'image de base existe dans le repo ECR ou si elle doit être construite.

serverlessEFSToolchainJava/**build-docker.sh** :

#!/bin/sh

# Script utilise pour le build de l'image Docker:

# recupere en arguments 1&2 le nom et la version de l'image docker pouvant servir de cache pour la construction de l'image cible, et en argument 2 l'image de base a utiliser si l'image n'est pas accessible

cache\_image\_name=$1

cache\_image\_version=$2

dockerfile\_basis=$3

region=$4

accountid=$5

if ! aws ecr list-images --region $region --registry-id $accountid --repository-name $cache\_image\_name | grep $cache\_image\_version

then

echo "Image cache non trouvee. Reconstruction ..."

docker build -f $dockerfile\_basis -t $accountid.dkr.ecr.$region.amazonaws.com/$cache\_image\_name:$cache\_image\_version .

docker push $accountid.dkr.ecr.$region.amazonaws.com/$cache\_image\_name:$cache\_image\_version &

else

docker pull $accountid.dkr.ecr.$region.amazonaws.com/$cache\_image\_name:$cache\_image\_version &

echo "Image cache trouvee. Telechargement ..."

fi

## Template CloudFormation

### Déploiement de la configuration réseau et du pipeline complets

1. A la racine du répertoire du projet, créer un répertoire 'CFN templates' le template CloudFormation qui déploiera les ressources réseau requises pour CodeBuild dans un VPC (VPC, subnet, SecurityGroup, NAT), ainsi que les ressources CodePipeline, CodeBuild et la registry ECR.

*Note : tous les paramètres peuvent être laissés par défaut à l'exception de* ***GitHubToken*** *et du* ***point de montage EFS*** *qui doivent impérativement être renseignés.*

serverlessToolchainJava/CFN templates/**Serverless-EFStoolchain-Network-CodeBuild.yml** :

AWSTemplateFormatVersion: '2010-09-09'

Description: serverless EFS Toolchain - DAN

Parameters:

# Network parameters -------------------------------------------------

VPCName:

Description: Nom du VPC a creer

Type: String

Default: 'serverlessEFSToolchain\_VPC\_CFN'

VPCCidrBlock:

Type: String

Description: 'Plage IP allouee au VPC en /16 (Exemple: 172.30.0.0/16)'

MinLength: '9'

MaxLength: '18'

Default: 172.30.0.0/16

AllowedPattern: '(\d{1,3})\.(\d{1,3})\.0\.0/16'

ConstraintDescription: Doit etre un block CIDR valide en /16

AvailabilityZone:

Type: String

Description: 'Zone de dispo dans laquelle les subnets seront crees'

Default: 'eu-west-1b'

AllowedValues: [eu-west-1a, eu-west-1b, eu-west-1c]

PublicSubnetName:

Description: Nom du subnet Public sur lequel la NAT Gateway est attachee.

Type: String

Default: 'serverlessEFSToolchain\_PublicSubnet\_CFN'

PublicSubnetCIDR:

Type: String

Description: 'Plage IP allouee au subnet public en /24 (Exemple: 172.30.1.0/24)'

MinLength: '9'

MaxLength: '18'

Default: 172.30.1.0/24

AllowedPattern: '(\d{1,3})\.(\d{1,3})\.(\d{1,3})\.0/24'

ConstraintDescription: Doit etre un block CIDR valide en /24

PrivateSubnetName:

Description: Nom du subnet Prive dans lequel Codebuild instancie son image Docker pour le build et subnet d'acces au moint de montage EFS.

Type: String

Default: 'serverlessEFSToolchain\_PrivateSubnet\_CFN'

PrivateSubnetCIDR:

Type: String

Description: 'Plage IP allouee au subnet prive en /24 (Exemple: 172.30.2.0/24)'

MinLength: '9'

MaxLength: '18'

Default: 172.30.2.0/24

AllowedPattern: '(\d{1,3})\.(\d{1,3})\.(\d{1,3})\.0/24'

ConstraintDescription: Doit etre un block CIDR valide en /24

PublicRouteTableName:

Description: Nom de la table de routage associee au subnet public 'CodebuildPublicSubnet'.

Type: String

Default: 'serverlessEFSToolchain\_PublicRouteTable\_CFN'

PrivateRouteTableName:

Description: Nom de la table de routage associee au subnet prive 'CodebuildPrivateSubnet'.

Type: String

Default: 'serverlessEFSToolchain\_PrivateRouteTable\_CFN'

InternetGatewayName:

Description: Nom de l'Internet Gateway associee au VPC.

Type: String

Default: 'serverlessEFSToolchain\_IGW\_CFN'

NATGatewayName:

Description: Nom de la NAT Gateway associee au VPC.

Type: String

Default: 'serverlessEFSToolchain\_NAT\_CFN'

PrivateSGName:

Description: Nom du Security Group associe a CodeBuild pour la recuperation des sources Git en HTTPS.

Type: String

Default: 'serverlessEFSToolchain\_CodeBuild-SG\_CFN'

# CodeBuild parameters -------------------------------------------------

ProjectName:

Description: Nom du projet utilise pour tagger les ressources creees

Type: String

Default: "serverlessEFSToolchainJava"

ECRRepositoryURL:

Description: URL du repository cree dans ECR hebergeant les images Docker generee durant l'etape de Post-Build de CodeBuild

Type: String

Default: "962109799108.dkr.ecr.eu-west-1.amazonaws.com/"

ApplicationImageRepository:

Description: Nom de l'image Docker generee durant l'etape de Post-Build de CodeBuild

Type: String

Default: "serverlessefstoolchainjava"

DockerCodebuildImage:

Description: Image Docker (et version) utilisee par CodeBuild pour builder l'application

Type: String

Default: "codebuild-custom-java:0.3"

DockerCacheImageVersion:

Description: Version de l'image Docker utilisee comme base initiale pour la construction de l'image Docker de l'application.

Type: String

Default: "basis"

bucketS3:

Description: "Bucket contenant les artefacts de l'application pour leur integration et leur deploiement (codes source et imagedefinitions des conteneurs requis)."

Type: 'String'

Default: "serverlessefstoolchainjava"

AllowedPattern: '[a-z0-9]+'

EFSDIR:

Description: "Repertoire de Sonar a monter sur l'image executee par CodeBuild."

Type: 'String'

Default: "/opt/sonar"

EFSDNS:

Description: "EndPoint EFS requis pour monter le filesystem."

Type: 'String'

Default: ""

CodeBuildCacheLocation:

Description: "Bucket contenant le cache des dependances et plugins de Maven pour le build"

Type: 'String'

Default: "/cache/archives"

EncryptionKeyARN:

Description: ARN de la cle CMK KMS utilisee pour chiffrer le bucket S3 (imperatif pour l'utilisation avec CodeBuild). Si la cle par defaut "arn:aws:kms:eu-west-1:962109799108:alias/aws/s3" n'existe pas, elle sera creee automatiquement par S3.

Type: String

Default: "arn:aws:kms:eu-west-1:962109799108:alias/aws/s3"

GitHubUser:

Description: Nom d'utilisateur GitHub.

Type: String

Default: "maddoudou22"

GitHubRepository:

Description: Nom du projet declare dans GitHub.

Type: String

Default: "serverlessEFSToolchainJava"

GitHubBranch:

Description: Branch utilisee dans GitHub.

Type: String

Default: "master"

GitHubOAuthToken:

Description: Token Oauth de GitHub. (A generer et recuperer depuis GitHub dans Settings -> Developer Settings -> Personnal Access Tokens)

Type: String

Default: ""

#--------------------------------------------------------------------------------

# INTERFACE

#--------------------------------------------------------------------------------

Metadata:

AWS::CloudFormation::Interface:

ParameterGroups:

- Label:

default: Configuration reseau

Parameters:

- VPCCidrBlock

- PublicSubnetCIDR

- PrivateSubnetCIDR

- AvailabilityZone

- Label:

default: Configuration de Codebuild

Parameters:

- ECRRepositoryURL

- ApplicationImageRepository

- DockerCodebuildImage

- GitHubOAuthToken

- EFSDNS

ParameterLabels:

ECRRepositoryURL:

default: "URL du repository cree dans ECR hebergeant les images Docker generee durant l'etape de Post-Build de CodeBuild"

ApplicationImageRepository:

default: "Nom de l'image Docker generee durant l'etape de Post-Build de CodeBuild"

DockerCodebuildImage:

default: "Image Docker (et version) utilisee par CodeBuild pour builder l'application"

GitHubOAuthToken:

default: "Token GitHub utilise par CodeBuild pour la recuperation des sources"

EFSDNS:

default: "Endpoint (nom DNS) du filesystem EFS hebergeant les repertoire de Sonar)"

VPCCidrBlock:

default: Plage IP du VPC

PublicSubnetCIDR:

default: Plage IP du subnet public

PrivateSubnetCIDR:

default: Plage IP du subnet prive

AvailabilityZone:

default: "Zone de dispo dans laquelle les subnets seront crees. ATTENTION: devra etre declaree dans EFS !"

#--------------------------------------------------------------------------------

# RESSOURCES

#--------------------------------------------------------------------------------

Resources:

#--------------------------------------------------------------------------------

# Configuration du reseau

#--------------------------------------------------------------------------------

# VPC ---------------------

VPC:

Type: "AWS::EC2::VPC"

Properties:

EnableDnsSupport: "true"

EnableDnsHostnames: "true"

CidrBlock: !Ref VPCCidrBlock

Tags:

- Key: "Name"

Value: !Sub ${VPCName}

- Key: "Application"

Value: !Ref 'AWS::StackId'

Value: !Sub ${VPCName}

# Subnets ---------------------

PublicSubnet:

Type: AWS::EC2::Subnet

Properties:

VpcId: !Ref VPC

AvailabilityZone: !Sub ${AvailabilityZone}

CidrBlock: !Ref PublicSubnetCIDR

MapPublicIpOnLaunch: false

Tags:

- Key: "Name"

Value: !Sub ${PublicSubnetName}

- Key: "Application"

Value: !Ref 'AWS::StackId'

Value: !Sub ${PublicSubnetName}

PrivateSubnet:

Type: AWS::EC2::Subnet

Properties:

VpcId: !Ref VPC

AvailabilityZone: !Sub ${AvailabilityZone}

CidrBlock: !Ref PrivateSubnetCIDR

MapPublicIpOnLaunch: false

Tags:

- Key: "Name"

Value: !Sub ${PrivateSubnetName}

- Key: "Application"

Value: !Ref 'AWS::StackId'

Value: !Sub ${PrivateSubnetName}

# Internet Gateway ---------------------

InternetGateway:

Type: AWS::EC2::InternetGateway

Properties:

Tags:

- Key: "Name"

Value: !Sub ${InternetGatewayName}

- Key: "Application"

Value: !Ref 'AWS::StackId'

Value: !Sub ${InternetGatewayName}

InternetGatewayAttachment:

Type: AWS::EC2::VPCGatewayAttachment

Properties:

InternetGatewayId: !Ref InternetGateway

VpcId: !Ref VPC

# NAT Gateway ---------------------

NatGateway:

Type: AWS::EC2::NatGateway

Properties:

AllocationId: !GetAtt NatGatewayEIP.AllocationId

SubnetId: !Ref PublicSubnet

Tags:

- Key: "Name"

Value: !Sub ${NATGatewayName}

- Key: "Application"

Value: !Ref 'AWS::StackId'

Value: !Sub ${NATGatewayName}

NatGatewayEIP:

Type: AWS::EC2::EIP

DependsOn: InternetGatewayAttachment

Properties:

Domain: vpc

# Route Tables ---------------------

PublicRouteTable:

Type: AWS::EC2::RouteTable

Properties:

VpcId: !Ref VPC

Tags:

- Key: "Name"

Value: !Sub ${PublicRouteTableName}

- Key: "Application"

Value: !Ref 'AWS::StackId'

Value: !Sub ${PublicRouteTableName}

DefaultPublicRoute:

Type: AWS::EC2::Route

DependsOn: InternetGatewayAttachment

Properties:

RouteTableId: !Ref PublicRouteTable

DestinationCidrBlock: 0.0.0.0/0

GatewayId: !Ref InternetGateway

PublicSubnetRouteTableAssociation:

Type: AWS::EC2::SubnetRouteTableAssociation

Properties:

RouteTableId: !Ref PublicRouteTable

SubnetId: !Ref PublicSubnet

PrivateRouteTable:

Type: AWS::EC2::RouteTable

Properties:

VpcId: !Ref VPC

Tags:

- Key: "Name"

Value: !Sub ${PrivateRouteTableName}

- Key: "Application"

Value: !Ref 'AWS::StackId'

Value: !Sub ${PrivateRouteTableName}

DefaultPrivateRoute:

Type: AWS::EC2::Route

DependsOn: InternetGatewayAttachment

Properties:

RouteTableId: !Ref PrivateRouteTable

DestinationCidrBlock: 0.0.0.0/0

NatGatewayId: !Ref NatGateway

PrivateSubnetRouteTableAssociation:

Type: AWS::EC2::SubnetRouteTableAssociation

Properties:

RouteTableId: !Ref PrivateRouteTable

SubnetId: !Ref PrivateSubnet

# Security Group ---------------------

CodeBuildSecurityGroup:

Type: AWS::EC2::SecurityGroup

Properties:

VpcId: !Ref 'VPC'

GroupDescription: !Sub ${PrivateSGName}

SecurityGroupIngress:

# NFS:

- IpProtocol: tcp

FromPort: '2049'

ToPort: '2049'

CidrIp: 0.0.0.0/0

- IpProtocol: tcp

FromPort: '111'

ToPort: '111'

CidrIp: 0.0.0.0/0

- IpProtocol: udp

FromPort: '2049'

ToPort: '2049'

CidrIp: 0.0.0.0/0

- IpProtocol: udp

FromPort: '111'

ToPort: '111'

CidrIp: 0.0.0.0/0

# Pour tests : A SUPPRIMER :

# - IpProtocol: tcp

# FromPort: 0

# ToPort: 65535

# CidrIp: 0.0.0.0/0

SecurityGroupEgress:

# HTTPS:

- IpProtocol: tcp

FromPort: '443'

ToPort: '443'

CidrIp: 0.0.0.0/0

- IpProtocol: tcp

FromPort: '80'

ToPort: '80'

CidrIp: 0.0.0.0/0

# NFS:

- IpProtocol: tcp

FromPort: '2049'

ToPort: '2049'

CidrIp: 0.0.0.0/0

- IpProtocol: tcp

FromPort: '111'

ToPort: '111'

CidrIp: 0.0.0.0/0

- IpProtocol: udp

FromPort: '2049'

ToPort: '2049'

CidrIp: 0.0.0.0/0

- IpProtocol: udp

FromPort: '111'

ToPort: '111'

CidrIp: 0.0.0.0/0

# Pour tests : A SUPPRIMER :

# - IpProtocol: tcp

# FromPort: 0

# ToPort: 65535

# CidrIp: 0.0.0.0/0

Tags:

Tags:

- Key: "Name"

Value: !Sub ${PrivateSGName}

- Key: "Application"

Value: !Ref 'AWS::StackId'

Value: !Sub ${PrivateSGName}

#--------------------------------------------------------------------------------

# Configuration de CodeBuild

#--------------------------------------------------------------------------------

# Creation du bucket lui-meme :

ArtefactsBucket:

Type: AWS::S3::Bucket

Properties:

BucketName: !Ref bucketS3

AccessControl: Private

# Policy associee au bucket :

S3BucketPolicy:

Type: AWS::S3::BucketPolicy

#Si la Policy est créée avant le bucket la création de la stack échoue :

DependsOn: ArtefactsBucket

Properties:

Bucket: !Ref bucketS3

PolicyDocument:

Statement:

-

Action:

- s3:\*

Effect: Allow

Resource:

- !Sub arn:aws:s3:::${bucketS3}

- !Sub arn:aws:s3:::${bucketS3}/\*

Principal:

Service:

- codepipeline.amazonaws.com

- codebuild.amazonaws.com

# Role endosse par CodeBuild pour la creation du projet Codebuild :

BuildProjectRole:

Type: "AWS::IAM::Role"

Properties:

RoleName: !Sub ${ProjectName}-CodeBuildRole-CFN

AssumeRolePolicyDocument:

Version: "2012-10-17"

Statement:

#

- Effect: "Allow"

Principal:

Service:

- codebuild.amazonaws.com

Action:

- "sts:AssumeRole"

Path: "/"

Policies:

-

PolicyName: !Sub ${ProjectName}-CodeBuildPolicy-CFN

PolicyDocument:

Version: "2012-10-17"

Statement:

-

Effect: Allow

Action:

- s3:PutObject

- s3:GetBucketPolicy

- s3:GetObject

- s3:GetObjectVersion

- s3:ListBucket

Resource:

- !Join ['',['arn:aws:s3:::',!Ref bucketS3, '/\*']]

- !Join ['',['arn:aws:s3:::',!Ref bucketS3]]

-

Effect: Allow

Action:

- kms:\*

Resource: !Ref EncryptionKeyARN

-

Effect: Allow

Action:

- EC2:\*

# Tentative d'affinage des droits : erreur "VPC\_CLIENT\_ERROR: Unexpected EC2 error: UnauthorizedOperation" sans plus d'explications ...

Resource: "\*"

-

Effect: Allow

Action:

- logs:CreateLogGroup

- logs:CreateLogStream

- logs:PutLogEvents

Resource: !Sub "arn:aws:logs:${AWS::Region}:${AWS::AccountId}:log-group:/aws/codebuild/\*"

-

Effect: Allow

Action:

- ecr:GetAuthorizationToken

Resource: "\*"

-

Effect: Allow

Action:

- ecr:BatchCheckLayerAvailability

- ecr:CompleteLayerUpload

- ecr:InitiateLayerUpload

- ecr:PutImage

- ecr:UploadLayerPart

- ecr:GetDownloadUrlForLayer

- ecr:BatchGetImage

- ecr:DescribeImages

- ecr:BatchDeleteImage

- ecr:GetRepositoryPolicy

- ecr:ListImages

Resource: !GetAtt ECRrepository.Arn

# Repository ECR :

ECRrepository:

Type: AWS::ECR::Repository

Properties:

RepositoryName: !Ref ApplicationImageRepository

# Projet Codebuild :

BuildProject:

Type: AWS::CodeBuild::Project

Properties:

Name: !Sub ${ProjectName}-CodeBuild-CFN

Description: !Sub Projet CodeBuild cree automatiquement pour le pipeline ${ProjectName}-pipeline-CFN

EncryptionKey: !Ref EncryptionKeyARN

ServiceRole: !GetAtt BuildProjectRole.Arn

Source:

Type: CODEPIPELINE

Artifacts:

Type: CODEPIPELINE

Environment:

Type: linuxContainer

ComputeType: BUILD\_GENERAL1\_MEDIUM

Image: !Sub '${ECRRepositoryURL}${DockerCodebuildImage}'

PrivilegedMode: true # Obligatoire pour l'utilisation des commandes Docker

EnvironmentVariables:

- Name: IMAGE\_REPO\_NAME

Value: !Ref ApplicationImageRepository

- Name: S3\_KEY

Value: !Ref EncryptionKeyARN

- Name: AWS\_ACCOUNT\_ID

Value: !Ref "AWS::AccountId"

- Name: DOCKER\_CACHE\_IMAGE\_VERSION

Value: !Ref DockerCacheImageVersion

- Name: EFS\_DIR

Value: !Ref EFSDIR

- Name: EFS\_DNS

Value: !Ref EFSDNS

VpcConfig:

SecurityGroupIds:

- !Ref CodeBuildSecurityGroup

Subnets:

- !Ref PrivateSubnet

VpcId: !Ref VPC

Cache:

Type: S3

Location: !Sub '${bucketS3}${CodeBuildCacheLocation}'

TimeoutInMinutes: 10

Tags:

- Key: Name

Value: !Ref ProjectName

#- CodePipeline ---------------------------------------------------------------------------------------------

# Role endosse par CodePipeline pour la creation du pipeline :

PipelineRole:

DependsOn: S3BucketPolicy

Type: AWS::IAM::Role

Properties:

RoleName: !Sub ${ProjectName}-codepipeline-role-CFN

AssumeRolePolicyDocument:

Version: 2012-10-17

Statement:

-

Effect: Allow

Principal:

Service:

- codepipeline.amazonaws.com

Action:

- sts:AssumeRole

Path: /

Policies:

-

PolicyName: !Sub ${ProjectName}-CodePipelinePolicy-CFN

PolicyDocument:

Version: "2012-10-17"

Statement:

-

Effect: Allow

Action:

- codepipeline:\*

- iam:ListRoles

- iam:PassRole

- codecommit:GetCommit

- codecommit:UploadArchive

- codebuild:BatchGetBuilds

- codebuild:StartBuild

Resource:

- "\*"

-

Effect: Allow

Action:

- ecs:DescribeServices

- ecs:DescribeTaskDefinition

- ecs:DescribeTasks

- ecs:ListTasks

- ecs:RegisterTaskDefinition

- ecs:UpdateService

Resource:

- "\*"

-

Effect: Allow

Action:

- kms:Decrypt

Resource: !Ref EncryptionKeyARN

-

Effect: Allow

Action:

- s3:PutObject

- s3:GetBucketPolicy

- s3:GetObject

- s3:GetObjectVersion

- s3:ListBucket

Resource:

- !Join ['',['arn:aws:s3:::',!Ref bucketS3, '/\*']]

- !Join ['',['arn:aws:s3:::',!Ref bucketS3]]

# Pipeline

Pipeline:

Type: AWS::CodePipeline::Pipeline

Properties:

RoleArn: !GetAtt PipelineRole.Arn

Name: !Sub ${ProjectName}-pipeline-CFN #!Ref AWS::StackName

Stages:

-

Name: Source

Actions:

-

Name: GitHub

ActionTypeId:

Category: Source

Owner: ThirdParty

Version: 1

Provider: GitHub

Configuration:

Owner: !Ref GitHubUser

Repo: !Ref GitHubRepository

Branch: !Ref GitHubBranch

OAuthToken: !Ref GitHubOAuthToken

OutputArtifacts:

- Name: SCCheckoutArtifact

RunOrder: 1

-

Name: Build

Actions:

-

Name: Build

ActionTypeId:

Category: Build

Owner: AWS

Version: 1

Provider: CodeBuild

Configuration:

ProjectName: !Ref BuildProject

RunOrder: 1

InputArtifacts:

- Name: SCCheckoutArtifact

OutputArtifacts:

- Name: BuildOutput

ArtifactStore:

Type: S3

Location: !Ref bucketS3

EncryptionKey:

Id: !Ref EncryptionKeyARN

Type: KMS

#--------------------------------------------------------------------------------

# OUTPUTS

#--------------------------------------------------------------------------------

# Déploiement de la stack

**Les étapes de déploiement seront donc les suivantes** :

1. Exécution du template *"* *Serverless-EFStoolchain-Network-CodeBuild.yml"* via CloudFormation.
   * Le pipeline sera entièrement créé avec succès et le projet sera buildé automatiquement.
   * La phase de déploiement échouera car EFS ne sera pas disponible dans le subnet de Codebuild (pas important).
2. Paramétrage d'EFS pour fournir un point de montage dans le subnet de CodeBuild.
   * Dans la gestion de accès d'EFS, déclarer le VPC, subnet privé et Security Group automatiquement déployés par le template (name='serverless...').
   * Attendre que le pont de montage passe de 'Creating' à 'available' (peut prendre quelques minutes).
3. Exécution d'une 'Release Change' dans CodePipeline (ou via GitHub) afin de rebuilder le projet.
   * Cette fois, le déploiement se déroulera jusqu'au bout. C'est très long car il faut récupérer toutes les dépendances.
4. Nouvelle exécution d'une 'Release Change' dans CodePipeline (ou via GitHub) afin de rebuilder le projet.
   * Cette fois, le déploiement est beaucoup plus rapide grâce au cache Maven récupéré dans S3 et au cache de l'image Docker.

# Suppression de la stack

**Les étapes de suppression seront donc les suivantes** :

1. Suppression de la Mount Target d'EFS :
   * Dans la console EFS, supprimer le mount target pointant sur le subnet déployé automatiquement **(pas le File System en lui-même !)**. Sinon CloudFormation ne pourra pas supprimer le subnet privé.
2. Suppression de toutes les images Docker du Repo ECR.
   * Dans la console ECR, supprimer toutes les images Docker du Repo ECR créé automatiquement (contenant les images Docker de l'application). Sinon CloudFormation ne pourra pas supprimer le repo ECR.
3. Vidage du bucket.
   * Dans la console S3, supprimer tous les objets du bucket S3 de la toolchain. Sinon CloudFormation ne pourra pas supprimer le bucket S3.
4. Suppression de la stack créée dans CloudFormation.

# Optimisation de l'image custom

La taille de l'image de build custom est déterminant pour le temps de provisioning de l'image.

L'objectif est de minimiser sa taille le plus possible. J'ai tenté avec l'image Alpine suivante qui ne fait que 170Mo au lieu de 360Mo. Le problème est qu'elle ne fonctionne pas : le démon Docker ne s'exécute pas avec une erreur *" Error starting daemon: Error initializing network controller: error obtaining controller instance: failed to create NAT chain DOCKER: Iptables not found "*

**dockerfile** :

FROM docker:latest

RUN apk --no-cache add openjdk8 \

maven \

tzdata \

python \

shadow \

nfs-utils \

curl \

&& adduser --disabled-password --gecos "" sonar \

&& mkdir -p /opt/sonar \

&& chown sonar:sonar /opt/sonar

&& cp /usr/share/zoneinfo/Europe/Paris /etc/localtime \

&& echo "Europe/Paris" > /etc/timezone \

&& echo "Europe/Paris" > /etc/TZ \

&& unset TZ \

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

&& python2.7 get-pip.py \

&& pip install awscli

Avec ajout des lignes suivantes dans le **buildspec.yaml** :

- nohup /usr/local/bin/dockerd --host=unix:///var/run/docker.sock --host=tcp://127.0.0.1:2375 --storage-driver=overlay2&

- timeout 15 sh -c "until docker info; do echo .; sleep 1; done"

- docker build -f dockerfile\_basis -t coucou/coucou:1.0 .

Pour le moment pas de solution ...

**Cette image reste intéressante pour utiliser Maven (compilation, exécution de tests, etc.) et la CLI AWS, mais sans commandes Docker.**