Creation d’une toolchain totalement serverless

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 accédant aux résultats des différents tests effectués à tout moment et en gardant l'historique si besoin.

Les tests réalisés ici sont les tests unitaires, OWASP et Sonar. L'importance de cette toolchain est de l'éprouver avec différents outils de tests ayant des contraintes spécifiques :

* Les tests unitaires et OWASP sont exécutés depuis Maven via des plugins, sans installation et exécution de logiciel et avec des restitutions de tests réalisés via des fichier JSON, CSV, HTML, etc.
* Sonar est plus complexe et contraignant dans ce type d'environnement car il s'agit d'une application à part entière devant être démarrée dans son intégralité (y compris son interface Web écoutant sur un port alloué sur le serveur) et stockant ses résultats dans une base de données.

Dans ce modop on utilise **CodeBuild au lieu de Fargate** car le tarif au temps d'exécution de CodeBuild semble beaucoup plus avantageux.

Ici on utilise CodeBuild avec une **image Docker pré-packagée par AWS** afin de réduire le temps de provisioning de l'image (une douzaine de secondes au lieu d'une quarantaine, soit une réduction de plus de trente secondes au provisioning).

### Utilisation de Sonar:

3 approches sont possibles pour utiliser Sonar :

* **Persistance de Sonar sur un volume EBS** : CodeBuild ne peut pas attacher une volume EBS à un conteneur de build. De plus, même si la base de données et les résultats Sonar sont persistés, il faudrait remonter le volume sur un autre conteneur ou une instance EC2 pour accéder aux résultats : si un build/test est lancé à ce moment, le build échouera car le volume sera monté ailleurs. On ne peut pas consulter les résultats et exécuter des builds en même temps.
* **Persistance de Sonar sur un point de montage EFS** : déjà traité dans un autre modop. Permet d'éxécuter des builds et tests tout en consultant les résultats (en théorie) et d'utiliser l'interface très riche de Sonar. Problèmes : EFS est lent et le stockage d'une base de données sur un Filer est une mauvaise pratique. Même si NFS gère les accès concurrents aux ressources, l'accès à la base par des instances différentes peuvent corrompre la base.
* **Installation de Sonar à la volée et export des résultats** : pas de persistance de Sonar. Sonar est installé à chaque build sur l'image DockerBuild (moins de 5 secondes, donc plus rapide et avantageux que d'utiliser un EFS ou de construire une image Build customisée avec SOnar déjà installé (+30 secondes pour la phase de provisioning)). Une fois les tests exécutés par Sonar, ils sont récupéré via l'API Sonar puis exportés dans un bucket S3. La mise en forme des résultats est réalisé par Athena, puis les dashboard sont mis à jour dans QuickSight. Permet de customiser la restitution des résultats et de les agréger avec les résultats d'autres tests. Par contre on ne bénéficie pas de l'interface native de Sonar, très riche. A voir si les exports des résultats Sonar dans la version Entreprise permet d'avoir autant de richesse.

## 

## Fonctionnement de l'ensemble

### Déploiement de la toolchain

* **Pré-requis** :
  + Les tableaux de bord QuickSight ne sont pas automatisables (pas de déploiement via CloudFormation) ni même exportables/sauvegardables... Il faut forcément les créer à la main à chaque instanciation de la toolchain.
* Un seul template permet de créer la toolchain et les requêtes Atheana, mais celles-ci ne sont pas exécutées.
* Une fois le template déployé, les requêtes Athena doivent être exécutées une première fois manuellement afin de créer les tables et les vues des résultats utilisées par QuickSight. Après c'est automatique.

### Exécution de la toolchain

* Les sources sont récupérées via GitHub.
* L'image de build 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**, Sonar est récupéré, dézippé et 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 (Le résultat des tests OWASP est généré en JSON). 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**, les résultats des tests Sonar et Tests Unitaires sont récupérés à partir de l'API Sonar (au format JSON). Un timestamp est généré afin de l'inclure dans les noms des fichiers et en tant que paramètre 'buildtimestamp' dans les fichiers JSON des résultats des tests (utilisé par les requêtes Athena pour identifier les résultats du dernier build).
  + Chaque type de test a son propre répertoire dans le bucket S3 du projet : une commande les créé s'ils n'existent pas déjà, puis pousse les résultats des tests en JSON dans les répertoires S3.
  + 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.
* L'utilisateur peut ensuite visualiser ses résultats dans QuickSight via l'application mobile ou la console Web QuickSight après avoir rafraichit les données des datasets manuellement. Le fait de rafraichir les données permet à QuickSight d'interroger Athena de façon transparente. A son tour, Athena récupère l'ensemble des fichiers des buckets S3 configurés et met à jour ses tables ses vues pour les transmettre à QuickSight.

# 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:\serverlessFullToolchainJava »* 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 :

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

serverlessFullToolchainJava/**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>serverlessfulltoolchainjava</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>

<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>

serverlessFullToolchainJava/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;

}

}

serverlessFullToolchainJava/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));

}

}

serverlessFullToolchainJava/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);

}

}

serverlessFullToolchainJava/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/serverlessFullToolchainJava

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

Par exemple : serverlessFullToolchainJava

c) Copier le lien https du repository :

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

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

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

echo off

SET nom\_Projet\_Git= serverlessFullToolchainJava

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

SET Repertoire\_Local\_Projet=C:\serverlessFullToolchainJava

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.*

# Configuration d'Athena

Athena est utilisé pour récupérer les résultats des tests au format JSON déposés dans des buckets S3 par l'image de build de Codebuild. Une table est créée pour chaque fichier de résultat de test (test unitaires, Sonar et OWASP) et donc pour chaque fichier JSON généré. Les tables récupèrent le format des JSON et les convertit en tables SQL. Pour chaque table, une ou plusieurs vue est créé afin de parser et agencer les données souhaitées dans les dashboards afin qu'elles soient facilement intégrables dans QuickSight.

## Création des tables et des vues dans Athena

Athena n'est pas encore bien intégré dans Cloud Formation qui permet uniquement de créer des requêtes sans les exécuter.

Toutes les requêtes de création des tables et des vue sont créée via le template CloudFormation. Par contre elles doivent être exécutées manuellement afin de créer les tables et les vues (à faire une seule fois lors de la création de la toolchain).

Dans la console AWS Athena, cliquer sur l'onglet "Saved Queries" :

1. Sélectionner la requête de création de la table des tests unitaires (ayant pour préfixe le nom court du projet renseigné dans le template CloudFormation, puis le numéro 1).
2. Dans l'onglet "Query Editor" qui s'affiche automatiquement, cliquer sur le bouton "Run Query".
3. Répéter les étapes 1 et 2 pour chacune des requêtes créées par le template CloudFormation (préfixées par le nom court du projet) en suivant les numéros.

## Gestion des droits des buckets pour QuickSight

Par défaut, Athena a besoin du bucket *"aws-athena-query-results-*<*COMPTE-AWS*>*-*<*REGION*>*"* dans lequel il stocke tous les résultats de ses requêtes par défaut (modifiable). normalement ce bucket est créé automatiquement (cf settings), sinon il faut le créer manuellement (sans attribuer de permissions particulières sur le bucket : tout laisser par défaut).

Ce bucket n'est pas utilisé par QuickSight mais il est obligatoire pour Athena.

# Configuration de QuickSight

A la demande de l'utilisateur, QuickSight contacte Athena pour mettre ses données et afficher les dashboards du dernier build et l'historique des builds.

QuickSight n'est pas du tout intégré à CloudFormation et il n'est pas possible de sauvegarder/exporter des analyses ou des dashboards... **Tout est à faire manuellement !**

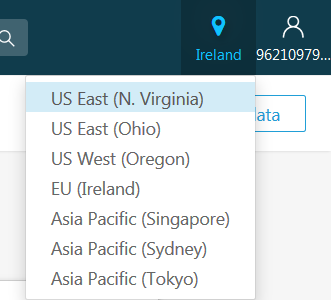
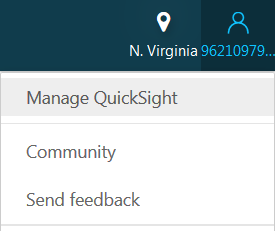
## Création d'un compte QuickSight

QuickSight requière la création d'un compte. Le nom de compte est à garder car il est requis pour se connecter via l'application QuickSight sur mobile. La création d'un compte est gratuite et permet d'utiliser librement QuickSight avec le compte qui l'a créé s'il n'est pas partagé avec d'autres utilisateurs et avec un max de 1Go de données stockées.

La gestion d'un compte QuickSight se fait uniquement via la localisation 'US East N. Virginia'.

## Gestion des droits des buckets S3 pour QuickSight

Dans la console de QuickSight, aller dans 'US East N. Virginia', puis cliquer sur - cliquer sur l'icone du compte et sur "Manage Quicksight"

Aller dans "Account settings" et cliquer sur le bouton "Manage QuickSight Permissions"

Cliquer sur le lien "Choose S3 Buckets" :

* Sélectionner le bucket du projet de la toolchain serverless à partir duquel Athena récupère les fichiers JSON pour créer ses tables (il s'agit du bucket 'root' spécifié dans le template CloudFormation).. Décocher les autres buckets.

Retourner dans EU-west et importer les données

*Note* : normalement QuickSight créé le bucket suivant automatiquement : *"aws-athena-query-results-*<*REGION*>*-*<*COMPTE-AWS*>*"* (**attention, region et compte aws sont inversés par rapport au bucket Athena**). Sinon le créer manuellement (sans droits particuliers) sinon QuickSight refusera de se connecter à Athena.

*Note* : QuickSight ne se sert pas du bucket *"aws-athena-query-results-*<*COMPTE-AWS*>*-*<*REGION*>*"* créé par Athena.

## Import des datasets depuis les vue d'Athena

Revenir dans la région dans laquelle le compte QuickSight a été créé (ex : Irelande) afin d'importer les Data Set qui serviront à la création des analyses puis des dashboards :

1. A l'écran d'accueil de QuickSight, cliquer sur le bouton "Manage Data" en haut à droite de l'écran.
2. Cliquer sur le bouton "New Data Set" en haut à gauche de l'écran.
3. Choisir "Athena" dans la liste des Data Sources, puis spécifier un nom et cliquer sur le bouton "Create Datasource".
4. Séléctionner la database spécifique du projet de toolchain serverless créée dans Athena, puis sélectionner l'une des vue créée dans Athena (pas les tables).
5. Cliquer sur le bouton "Select".
6. Laissez coché "Import to SPICE for quicker analytics" puis cilquer sur le bouton "Visualize".
7. QuickSight créé une analyse. Dans l'écran qui apparait, vérifier qui l'import de données est "Complete".
8. Revenir à l'écran d'acceuil de Quicksight.
9. Répéter l'opération pour toutes les vues créées dans Athena (à peu près une ou deux vues par résultat d'analyse, plus une vue d'historique).

## Création d'une analyse QuickSight

A l'écran d'accueil de QuickSight, sélectionner une analyse à modifier.

**TODO**

# 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 ...).*

serverlessToolchainJava/**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 ..."

- 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)"

- 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 "Tag de cette nouvelle version en 'latest'. C'est elle qui sera utilisee en cache pour le future de la future release de l'application"

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

- echo "Pushing the Docker image..."

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

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

- echo "Writing image definitions file... (utilise uniquement pour le deploiement dans FarGate".

- printf '[{"name":"%s","imageUri":"%s"}]' $IMAGE\_REPO\_NAME $AWS\_ACCOUNT\_ID.dkr.ecr.$AWS\_REGION.amazonaws.com/$IMAGE\_REPO\_NAME:$PACKAGE\_VERSION > imagedefinitions.json

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.

serverlessToolchainJava/**dockerfile** :

FROM java:8

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.

serverlessToolchainJava/**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.

serverlessToolchainJava/**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 . &

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 le template CloudFormation "Java-Fargate-Toolchain - Pipeline template.yml" qui déploiera 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/**Serverless-toolchain-Network-CodeBuild.yml** :

AWSTemplateFormatVersion: '2010-09-09'

Description: serverless Toolchain - DAN

Parameters:

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

VPCName:

Description: Nom du VPC a creer

Type: String

Default: 'serverlessToolchain\_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: 'serverlessToolchain\_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: 'serverlessToolchain\_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: 'serverlessToolchain\_PublicRouteTable\_CFN'

PrivateRouteTableName:

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

Type: String

Default: 'serverlessToolchain\_PrivateRouteTable\_CFN'

InternetGatewayName:

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

Type: String

Default: 'serverlessToolchain\_IGW\_CFN'

NATGatewayName:

Description: Nom de la NAT Gateway associee au VPC.

Type: String

Default: 'serverlessToolchain\_NAT\_CFN'

PrivateSGName:

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

Type: String

Default: 'serverlessToolchain\_CodeBuild-SG\_CFN'

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

ProjectName:

Description: Nom du projet utilise pour tagger les ressources creees

Type: String

Default: "serverlessToolchainJava"

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: "serverlesstoolchainjava"

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: "latest"

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: "serverlesstoolchainjava"

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: "serverlessToolchainJava"

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-toolchain-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.