# **Publishing Spring Boot Docker Images to Elastic Container Registry using Maven Plugin**

**Overview**

The purpose of this guide is to show how to simplify and accelerate the process of building, tagging and pushing SpringBoot artifacts (i.e. jar or war files) as a docker images to AWS Elastic Container Registry.

We will be building a simple SpringBoot application to demonstrate the end to end process of building, tagging and pushing an image to ECR.

**Prerequisites**

Please make sure the following software’s are installed in your computer to follow along.

* JDK 1.8 or above
* Maven3.0 or above <https://maven.apache.org/install.html>
* Docker for Windows <https://docs.docker.com/docker-for-windows/install/>
* ECR Repo <https://console.aws.amazon.com/ecr/create-repository?region=ap-southeast-1>
* Install and Configure AWS CLI
  + <https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-install.html>
  + <https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-configure.html>

**Build Starter/Demo App**

Generate a spring boot maven project using the Spring Initializer. You can either use the API as shown below or the web UI (https://start.spring.io) to generate maven projects.

$ curl https://start.spring.io/starter.zip -d dependencies=web -o starter.zip

Unzip the generated project.

$ unzip starter.zip

$ ls

HELP.md starter.zip mvnw mvnw.cmd pom.xml src

After creating the project add a simple spring controller as shown below to display the Name, OS architecture and Java version of the container.

Open a new file called “**src/main/java/hello/HelloController.java**” and copy the below content and save it.

package com.example.demo;

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

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

import java.net.InetAddress;

import java.net.UnknownHostException;

@RestController

public class HelloController {

@RequestMapping("/")

public String index() {

InetAddress ip = null;

String hostname;

try {

            ip = InetAddress.getLocalHost();

        } catch (UnknownHostException e) {

            e.printStackTrace();

        }

hostname = ip.getHostName();

return "Container Info :"+ hostname + " OS Architecture : " +System.getProperty("os.arch")+ " Java Version : " + System.getProperty("java.version");

}

}

**Configure your pom.xml file to use the Docker Maven Plugin**

Open the pom.xml file add the following properties and plugins required to build and push the docker image to Elastic container registry.

 <properties>

   <java.version>1.8</java.version>

   <docker.image.prefix>ECR-URL</docker.image.prefix>

<docker.image.appid>IMAGE-NAME</docker.image.appid>

 </properties>

**Note:**

Please specify the ECR to which you intend to push the docker image.

Below a screenshot shows how to find your ECR-URL and IMAGE-NAME properties.



Navigate to the plugins section (i.e. <plugins> ….. </plugins>) and add the dockerfile-maven-plugin

            <plugin>

              <groupId>com.spotify</groupId>

              <artifactId>dockerfile-maven-plugin</artifactId>

              <version>1.4.10</version>

              <executions>

                <execution>

                  <id>default</id>

                            <goals>

                                <goal>build</goal>

                                <goal>push</goal>

                            </goals>

                </execution>

              </executions>

              <configuration>

                 <repository>${docker.image.prefix}/${docker.image.appid}</repository>

<serverId>${docker.image.prefix}</serverId>

<registryUrl>https://${docker.image.prefix}</registryUrl>

                 <tag>${project.version}</tag>

                 <buildArgs>

                        <JAR\_FILE>${project.build.finalName}.jar</JAR\_FILE>

                    </buildArgs>

              </configuration>

            </plugin>

**Create a Dockerfile**

Create a file called “**Dockerfile**” in project home directory as show below. The definition in this file will be used by the maven plugin to pull the base image, add application jar file and build the docker image.

FROM openjdk:8-jdk-alpine

VOLUME /tmp

ADD target/\*-0.0.1-SNAPSHOT.jar app.jar

ENV JAVA\_OPTS=""

ENTRYPOINT [ "sh", "-c", "java $JAVA\_OPTS -Djava.security.egd=file:/dev/./urandom -jar /app.jar" ]

**Retrieve Credential for Elastic Container Registry**

Navigate to your terminal window and run below command to login and retrieve the credentials required to push the container images to the AWS Elastic container Registry.

**Note:**

Please make sure your configured to connect to AWS using awsci commands.

<https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-configure.html>

$ $(aws ecr get-login --no-include-email --region ap-southeast-1)

WARNING! Using --password via the CLI is insecure. Use --password-stdin.

Login Succeeded

Now let’s run the maven command to build, tag and push the container images to AWS Elastic container Registry.

$ mvn package dockerfile:build dockerfile:push

[INFO] Scanning for projects...

[INFO]

[INFO] --------------------------< com.example:demo >--------------------------

[INFO] Building demo 0.0.1-SNAPSHOT

[INFO] --------------------------------[ jar ]---------------------------------

<##########################OUTPUT TRUNCATED#################################>

[INFO]

[INFO]

[INFO] Image will be built as **765500136580.dkr.ecr.ap-southeast-1.amazonaws.com/maven-demo:0.0.1-SNAPSHOT**

[INFO]

[INFO] Step 1/5 : FROM openjdk:8-jdk-alpine

<##########################OUTPUT TRUNCATED#################################>

[INFO] Successfully built d95e7f3c7f84

[INFO] Successfully tagged 765500136580.dkr.ecr.ap-southeast-1.amazonaws.com/maven-demo:0.0.1-SNAPSHOT

[INFO]

[INFO] Detected build of image with id d95e7f3c7f84

[INFO] Building jar: /Users/vramasub/Documents/docker-maven-package/target/demo-0.0.1-SNAPSHOT-docker-info.jar

[INFO] Successfully built **765500136580.dkr.ecr.ap-southeast-1.amazonaws.com/maven-demo:0.0.1-SNAPSHOT**

[INFO]

[INFO] --- dockerfile-maven-plugin:1.4.10:push (default-cli) @ demo ---

[INFO] **The push refers to repository [765500136580.dkr.ecr.ap-southeast-1.amazonaws.com/maven-demo**]

[INFO] Image d40ec5ca02da: Preparing

<##########################OUTPUT TRUNCATED#################################>

[INFO] Image d40ec5ca02da: Pushing

[INFO] Image d40ec5ca02da: Pushed

[INFO] 0.0.1-SNAPSHOT: digest: sha256:8d0000de69c15b28c9b09f77f2cd3c079abd3662d3991f324984ac98d0abe4e8 size: 1159

[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESS

[INFO] ------------------------------------------------------------------------

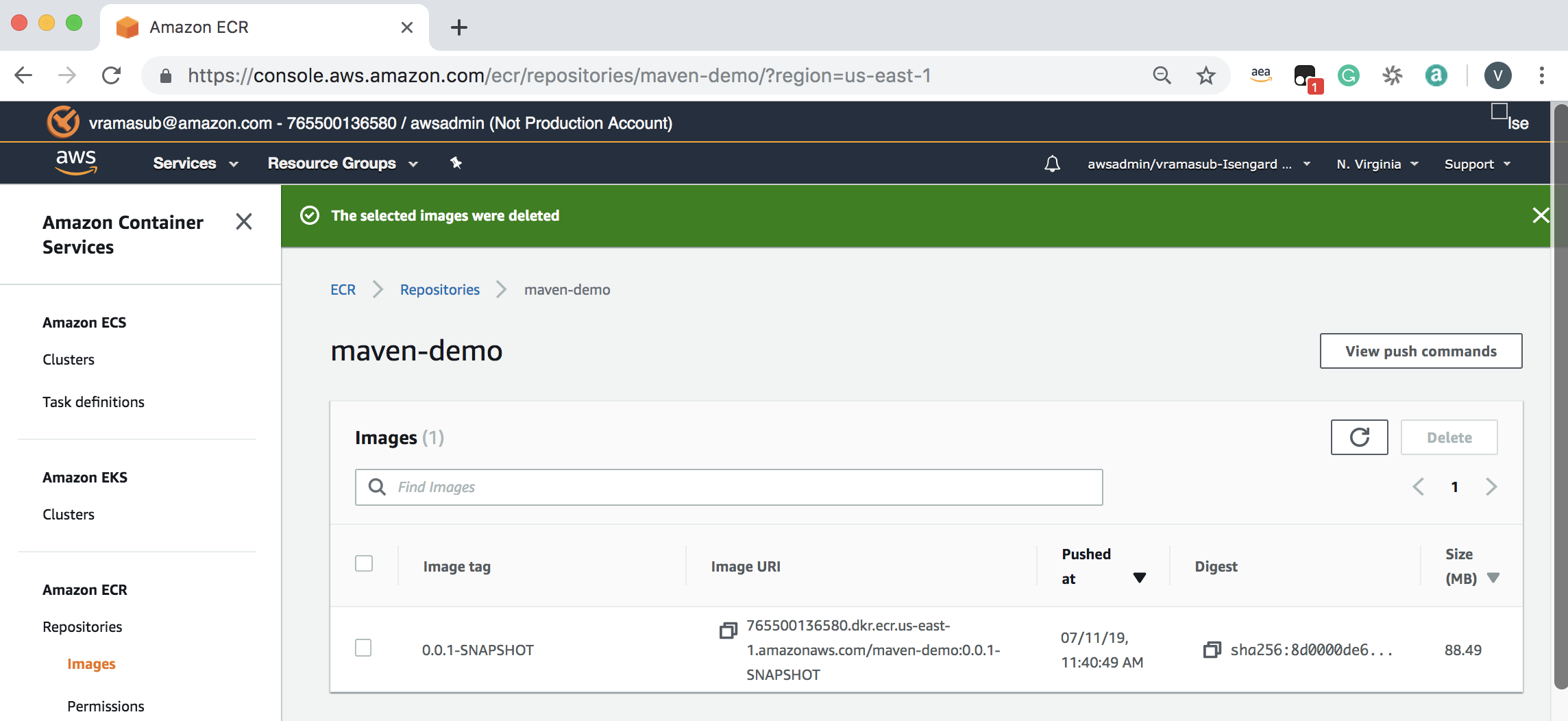
[INFO] Total time: 41.421 s

[INFO] Finished at: 2019-07-11T11:40:48-04:00

[INFO] -----------------------------------------------------------------------

**Validate push to AWS Elastic Container Registry**

Log into the AWS console and navigate to the ECR page to make sure the container image has been built and pushed by maven plugin.



**Validate Application Build**

Run the docker image in our terminal to verify if the Spring Boot application starts up and runs as expected.

$ docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

765500136580.dkr.ecr.ap-southeast-1.amazonaws.com/maven-demo 0.0.1-SNAPSHOT d95e7f3c7f84 12 minutes ago 122MB

$ docker run -itd -p 8080:8080 d95e7f3c7f84

798f2959902dcaed83801c86c7ac35d4fff4d99bcec95b0791b5b7285563efc6

$ docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

798f2959902d d95e7f3c7f84 "sh -c 'java $JAVA\_O…" 39 seconds ago Up 38 seconds 0.0.0.0:8080->8080/tcp stupefied\_rosalind

The above out shows that the container started up successful, and exposed for access on port 8080.

Let’s run the validate if it’s the application

$ curl localhost:8080

Container Info : 798f2959902d OS Architecture : amd64 Java Version : 1.8.0\_20138f9d3529ecd

There you go, the Spring Boot application started up successfully and it’s displays the name, OS architecture and Java version of the container.

Real time application logs can be accessed by running the “docker logs” command as show below.

$ docker logs 798f2959902d

. \_\_\_\_ \_ \_\_ \_ \_

/\\ / \_\_\_'\_ \_\_ \_ \_(\_)\_ \_\_ \_\_ \_ \ \ \ \

( ( )\\_\_\_ | '\_ | '\_| | '\_ \/ \_` | \ \ \ \

\\/ \_\_\_)| |\_)| | | | | || (\_| | ) ) ) )

' |\_\_\_\_| .\_\_|\_| |\_|\_| |\_\\_\_, | / / / /

=========|\_|==============|\_\_\_/=/\_/\_/\_/

:: Spring Boot :: (v2.1.6.RELEASE)

2019-07-11 15:54:49.173 INFO 1 --- [ main] com.example.demo.DemoApplication : Starting DemoApplication v0.0.1-SNAPSHOT on 798f2959902d with PID 1 (/app.jar started by root in /)

2019-07-11 15:54:49.184 INFO 1 --- [ main] com.example.demo.DemoApplication : No active profile set, falling back to default profiles: default

2019-07-11 15:54:52.001 INFO 1 --- [ main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)

2019-07-11 15:54:52.036 INFO 1 --- [ main] o.apache.catalina.core.StandardService : Starting service [Tomcat]

2019-07-11 15:54:52.036 INFO 1 --- [ main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.21]

2019-07-11 15:54:52.371 INFO 1 --- [ main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext

2019-07-11 15:54:52.371 INFO 1 --- [ main] o.s.web.context.ContextLoader : Root WebApplicationContext: initialization completed in 2989 ms

2019-07-11 15:54:52.764 INFO 1 --- [ main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'

2019-07-11 15:54:53.227 INFO 1 --- [ main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''

2019-07-11 15:54:53.243 INFO 1 --- [ main] com.example.demo.DemoApplication : Started DemoApplication in 4.842 seconds (JVM running for 5.453)

**Summary**

Docker Maven plugin is very useful utility that simplifies the error prone process of manually building and pushing images to centralized registries such as Elastic Container Registry (ECR). Additionally, this plugin helps developers quickly spin up images of Spring Boot application and test them locally before committing it to centralized registries.

In the next part of this series, I will demonstrate how to use this process to build images using AWS code pipeline and deploy it to Elastic container Service (ECS) and Elastic Kubernetes as a Service (EKS).