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

# Overview

The purpose of this guide is to show how to simplify & 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** (**ECR**).

We will be building a simple SpringBoot application to demonstrate the end-to-end process of building, tagging and pushing a **Docker Image** to ECR.

# Prerequisites 🡪 ./install-prerequisites.sh

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

* **JDK 11** or above
* **Maven** 3.0 or above <https://maven.apache.org/install.html>
* **Docker Desktop** for Mac & Windows   
  <https://www.docker.com/products/docker-desktop>
* **ECR** Repository  
  <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 SpringBoot 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.

https://start.spring.io/#!type=maven-project&language=java&platformVersion=2.4.0.RELEASE&packaging=jar&jvmVersion=11&groupId=io.job4u&artifactId=starter&name=starter&description=Starter%20project%20for%20Spring%20Boot&packageName=io.job4u.starter

Click GENERATE >> then 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/version/VersionController.java” and copy the below content and save it.

package io.job4u.starter.version;

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 VersionController {

@RequestMapping("/springboot")

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.

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

 <properties>

    <java.version>11</java.version>

 </properties>

# 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: [openjdk:11-jre-slim](https://hub.docker.com/_/openjdk?tab=tags&page=1&ordering=last_updated&name=11-jre-slim) (~69 MB)

##############################

## Build stage

##############################

FROM maven:3.6-jdk-11-slim AS build

COPY src /home/app/src

COPY pom.xml /home/app

# COPY scripts /home/app

RUN mvn -f /home/app/pom.xml clean package

##############################

## Package stage

##############################

FROM openjdk:11-jre-slim

COPY --from=build /home/app/target/starter-0.0.1-SNAPSHOT.jar /app.jar

# VOLUME /tmp

# ARG JAR\_FILE

# ADD ./target/${JAR\_FILE} app.jar

RUN sh -c 'touch /app.jar'

ENV JAVA\_OPTS=""

EXPOSE 80

ENV ACTIVE\_PROFILE=dev

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

# ENTRYPOINT ["java","-jar","/usr/local/lib/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 [AWS-CLI commands](https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-configure.html).

export AWS\_ACCOUNT\_ID=$(aws sts get-caller-identity | jq -r '.Account' | tr -d '\n')

export AWS\_REGION=${AWS\_REGION:-"ap-southeast-1"}

aws ecr create-repository --repository-name springboot

aws ecr get-login-password --region ap-southeast-1 | docker login --username AWS --password-stdin ${AWS\_ACCOUNT\_ID}.dkr.ecr.${AWS\_REGION}.amazonaws.com

Login Succeeded

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

./deploy-docker-ecr.sh react

./deploy-docker-ecr.sh springboot

# 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

docker run -itd -p 8080:8080 d95e7f3c7f84

docker ps

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:

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

# 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).