Building Java Applications with Build Tools and Plugins



Esteban Herrera Author | Developer | Consultant

@eh3rrera eherrera.net

Overview



Maven and Gradle plugins

- Fabric8's Docker Maven Plugin
- Palantir's Docker Gradle Plugin

Overview



Layered images

- Spring Boot
- Google Jib

Fabric8 Docker Maven Plugin

Goals

Goal	Description	Default Phase
docker:build	Builds images	install
docker:start and docker:run	Create and start containers	pre-integration-test
docker:stop	Stops and destroy containers	post-integration-test
docker:push	Pushes images to a registry	deploy
docker:remove	Removes images from local docker host	post-integration-test

Plugin Configuration

pom.xml

```
<plugin>
   <groupId>io.fabric8</groupId>
   <artifactId>fabric8-maven-plugin</artifactId>
   <configuration>
        <dockerHost>https://localhost:1234</dockerHost>
        <outputDirectory>target/fabric8-maven-plugin/outputDirectory>
        <verbose>true</verbose>
        <images>
           <image>
               <name>my-image</name>
               <alias>app</alias>
               <build>
                    <from>java:11
               </build>
               <run>
                    <ports>
                        <port>9000:8080</port>
                    </ports>
               </run>
           </image>
        </images>
   </configuration>
</plugin>
```

Build Configuration

pom.xml

```
<build>
    <from>openjdk:15
   <labels>
        <my-label>foo</my-label>
   </labels>
   <workdir>/app</workdir>
    <ports>
        <port>8080</port>
    </ports>
    <volumes>
        <volume>/my-volume</volume>
   </volumes>
    <shell>
        <exec>
            <arg>/bin/sh</arg>
            <arg>-c</arg>
        </exec>
    </shell>
   <runCmds>
        <run>groupadd -r my-group</run>
        <run>useradd -r -g my-group my-user</run>
   </runCmds>
    <entryPoint>
        <exec>
            <arg>java</arg>
            <arg>-jar</arg>
            <arg>app.jar</arg>
        </exec>
    </entryPoint>
</build>
```

pom.xml

Build Configuration

```
<build>
    <dockerFile>myDockerfile</dockerFile>
         <contextDir>${project.basedir}/docker</contextDir>
         </build>
```

Run Configuration

pom.xml

```
<run>
   <ports>
       <port>9000:8080</port>
   </ports>
   <labels>
       <environment>development
   </labels>
   <volumes>
       <bind>
           <volume>/host_dir:/container_dir
       </bind>
   </volumes>
   <restartPolicy>
       <name>always</name>
   </restartPolicy>
   <cmd>java -jar /maven/docker-demo.jar</cmd>
</run>
```

Demo



Using the plugin with a JAR application

Palantir Docker Gradle Plugin

Palantir Plugins

com.palantir.docker com.palantir.docker-compose com.palantir.docker-run

Plugin Tasks

Plugin	Task	Description	
docker			
	docker	Builds Docker image	
	dockerClean	Cleans Docker build directory	
	dockerPrepare	Prepares Docker build directory	
	dockerPush	Pushes named Docker image to registry	
docker-run			
	dockerRun	Runs the container	
	dockerRunStatus	Checks the run status of the container	
	dockerStop	Stops the container if it's running	

Docker Configuration

build.gradle

```
plugins {
    id 'com.palantir.docker' version '<version>'
docker {
    name 'my-image'
    files 'file1.txt', 'file2.txt'
    dockerfile file('Dockerfile')
    tag 'my-tag'
    buildArgs( [BUILD_VERSION: 'version'] )
    labels( ['key': 'value'] )
    pull true
    noCache true
```

DockerRun Configuration

build.gradle

```
plugins {
    id 'com.palantir.dockerRun' version '<version>'
dockerRun {
    name 'my-container'
    image 'busybox'
    volumes 'hostvolume': '/containervolume'
    ports '7080:5000'
    daemonize true
    env 'MYVAR1': 'MYVALUE1', 'MYVAR2': 'MYVALUE2'
    command 'sleep', '100'
    arguments '--hostname=custom', '-P'
    clean true
```

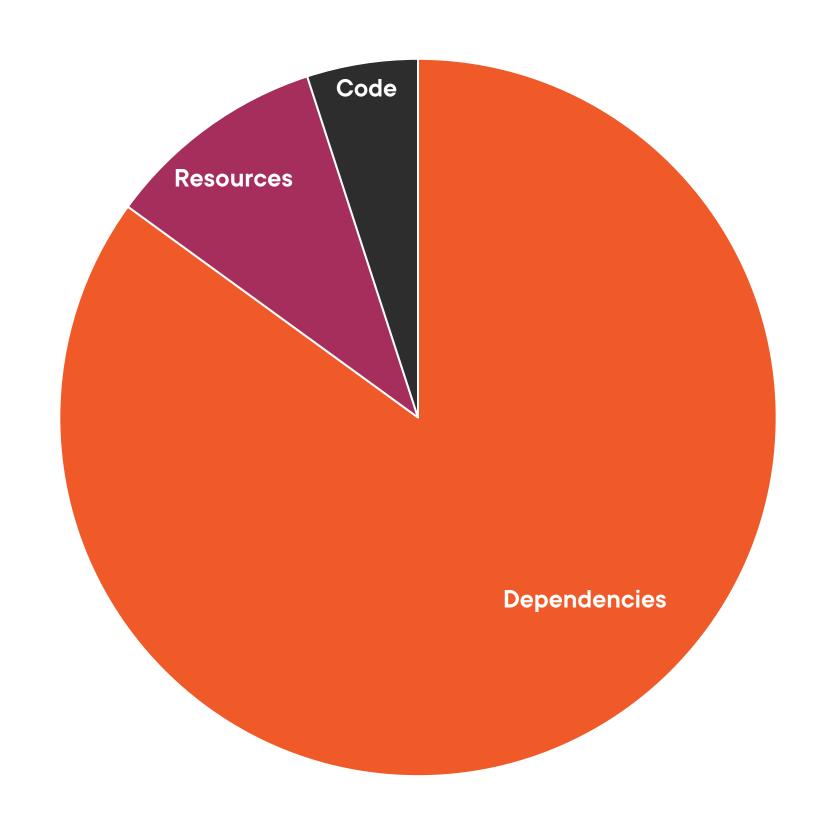
Demo



Using the plugin with a WAR application

Layered Deployment with Spring Boot

Application's Parts Size



Layers

Application dependencies

Other resources and classes

Application code

Most likely to change

Spring Boot JAR Structure

BOOT-INF

- classes
- lib

META-INF

- MANIFEST.MF

org

- springframework
 - boot
 - loader

Sample Dockerfile for a Layered Spring Boot

```
FROM openjdk:slim-buster
WORKDIR /my-app
COPY lib lib
```

COPY classes classes

COPY META-INF META-INF

```
ENTRYPOINT ["java","-cp","classes:lib/*","com.demo.Application"]
```

Spring Boot 2.3 and Above



mvn spring-boot:build-image

gradle bootBuildImage

Generate an Image with Buildpacks

Sample Plugin Configuration

pom.xml

```
<plugin>
   <groupId>org.springframework.boot
   <artifactId>spring-boot-maven-plugin</artifactId>
   <configuration>
       <image>
          <name>${project.artifactId}</name>
          <publish>true<publish>
      </image>
      <docker>
          <publishRegistry>
              <username>user</username>
              <password>password>
              <url>https://docker.example.com/</url>
          </publishRegistry>
      </docker>
   </configuration>
</plugin>
```

build.gradle

```
bootBuildImage {
   imageName = "${project.name}"
   publish = true
   docker {
      publishRegistry {
         username = "user"
         password = "passw"
         url = "https://docker.example.com/"
      }
   }
}
```

layers.idx

```
- "dependencies":
  - "BOOT-INF/lib/"
- "spring-boot-loader":
  - "org/"
- "snapshot-dependencies":
- "application":
  - "BOOT-INF/classes/"
  - "BOOT-INF/classpath.idx"
  - "BOOT-INF/layers.idx"
  - "META-INF/"
```

Default Configuration for Layers

layers.xml

```
<layers xmlns=...>
 <application>
   <into layer="spring-boot-loader">
     <include>org/springframework/boot/loader/**</include>
    </into>
   <into layer="application" />
 </application>
  <dependencies>
   <into layer="application">
     <includeModuleDependencies />
   </into>
   <into layer="snapshot-dependencies">
     <include>*:*:*SNAPSHOT</include>
    </into>
   <into layer="dependencies" />
 </dependencies>
  <layerOrder>
   <layer>dependencies
   <layer>spring-boot-loader</layer>
   <layer>snapshot-dependencies
   <layer>application</layer>
  </layerOrder>
</layers>
```

build.gradle

```
bootJar {
  layered {
    application {
      intoLayer("spring-boot-loader") {
        include "org/springframework/boot/loader/**"
      intoLayer("application")
    dependencies {
      intoLayer("application") {
        includeProjectDependencies()
      intoLayer("snapshot-dependencies") {
        include "*:*:*SNAPSHOT"
      intoLayer("dependencies")
    layerOrder = ["dependencies", "spring-boot-loader",
                    "snapshot-dependencies", "application"]
```

Usage:

java -Djarmode=layertools -jar my-app.jar command

Layer tools JAR Mode

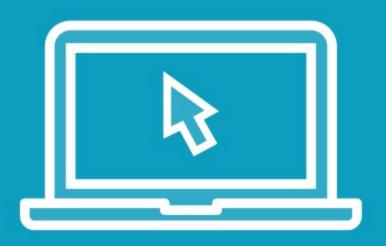
Available commands:

list List layers from the jar that can be extracted

extract Extracts layers from the jar for image creation

help Help about any command

Demo



Buidpacks and layer customization

Building Docker Images with Google Jib

Jib Features



Available as Maven and Gradle plugins

- No Dockerfile
- No Docker installation needed (in some cases)

Organizes your application into layers

Creates reproducible build images

- Builds images declaratively

mvn compile jib:build -Dimage=\$IMAGE_PATH

gradle jib --image=\$IMAGE_PATH

Build and Push the Image to a Container Registry

Requires authorization credentials for the registry

- Credential helpers
- CLI tools
- auth parameter in plugin's configuration
- Maven settings

mvn compile jib:dockerBuild

gradle jibDockerBuild

Build Image with Local Docker Installation

Sample Plugin Configuration

pom.xml

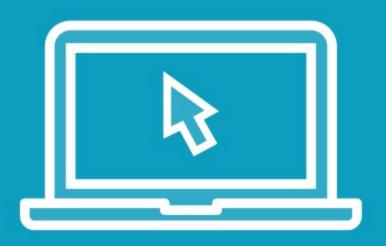
```
<plugin>
   <groupId>com.google.cloud.tools</groupId>
   <artifactId>jib-maven-plugin</artifactId>
   <version>3.0.0
   <configuration>
       <from>
          <image>openjdk:11</image>
      </from>
       <to>
          <image>my-image</image>
          <tags>
              <tag>my-tag</tag>
          </tags>
      </to>
       <container>
          <jvmFlags>
              <jvmFlag>-Xms256m</jvmFlag>
          </jvmFlags>
      </container>
   </configuration>
</plugin>
```

build.gradle

```
plugins {
    id 'com.google.cloud.tools.jib' version '3.0.0'
}

jib {
    from {
        image = 'openjdk:11'
    }
    to {
        image = 'my-image'
        tags = ['my-tag']
    }
    container {
        jvmFlags = ['-Xms512m']
    }
}
```

Demo



Create an image with Google Jib for a WAR application

Summary



Fabric8's Docker Maven plugin
Palantir's Docker Gradle plugin
Spring Boot
Google Jib

Summary



Which of all the options should I use?

- Adopt Docker through different stages
- All the options have drawbacks and benefits

Up Next:

Running Multi-Container Java Applications with Docker Compose