Getting Started Building Docker Images with Gradle

1. Introduction

In this Gradle Guide, You will learn how to build a simple Docker image using the Gradle Docker Plugin.

2. Getting Started

Let's get started on building a Docker image!

2.1. What You'll Build

You will be using the necessary classes defined in the Gradle Docker Plugin to create a `Dockerfile`, build the image, create the container, then start the container.

2.2. What You'll Need

- A text editor or IDE such as IntelliJ IDEA
- A Java Development Kit (JDK), version 11+
- The latest version of Docker
- The latest Gradle distribution
- The Docker Java library

This plugin requires Gradle >= 7.4.0.

2.3. Initiate the Project

Follow the Gradle Guide, Building Java Applications with Libraries Sample to initiate your project.

2.4. Java Application Plugin

You will be using the Java Application Plugin for this example application.

2.5. Update the `build.gradle` File

Now that you have a directory structure in place, let's build upon the `build.gradle` file that was generated by `gradle init` command.

First, let's import the required classes:

```
import com.bmuschko.gradle.docker.tasks.DockerInfo
import com.bmuschko.gradle.docker.tasks.DockerVersion

import com.bmuschko.gradle.docker.tasks.container.DockerCreateContainer
import com.bmuschko.gradle.docker.tasks.container.DockerExecContainer
import com.bmuschko.gradle.docker.tasks.container.DockerStartContainer
import com.bmuschko.gradle.docker.tasks.container.DockerStopContainer

import com.bmuschko.gradle.docker.tasks.image.Dockerfile
import com.bmuschko.gradle.docker.tasks.image.DockerBuildImage
import com.bmuschko.gradle.docker.tasks.image.DockerListImages
```

Let's update the `plugins` block with the required plugins:

```
plugins {
    id 'java'
    id 'application'
    id 'java-gradle-plugin'
    id 'com.bmuschko.docker-java-application' version '9.1.0'
    id 'com.bmuschko.docker-remote-api' version '9.1.0'
}
```

Let's update our `dependencies` block with the required dependencies:

```
dependencies {
implementation group: 'com.bmuschko', name: 'gradle-docker-plugin', version: '6.7.0'
implementation group: 'com.bmuschko', name: 'asciidoctorj-tabbed-code-extension',
version: '0.3'
}
```

2.6. Java Application Plugin Classes

The Java classes are instantiated in the `build.gradle` file, in other words, you won't be using the `new` keyword. Instead,

The `Docker.tmpl` file is a template that is used for the plugin to create the official `Dockerfile` in the `build` directory upon success of the build.

`# template for generated Dockerfile'

```
gradle clean startMyAppContainer --warning-mode all
```

The `--warning-mode` flag is used for listing deprecated Gradle features that may be incompatible with Gradle 8.0 scheduled for release on { date }.

The generated `Dockerfile` may be found in the `/build/docker` directory. It contains:

```
FROM openjdk:11.0.15-jre-slim
LABEL maintainer="Michael Redlich \"mike@redlich.net\""
WORKDIR /app
COPY libs libs/
COPY classes classes/
ENTRYPOINT ["java", "-Xms256m", "-Xmx2048m", "-cp",
"/app/resources:/app/classes:/app/libs/*", "org.gradle.MainApp"]
EXPOSE 9090 5701
RUN ls -la
ENV JAVA_OPTS="-XX:+UnlockExperimentalVMOptions -XX:+UseCGroupMemoryLimitForHeap"
# template for generated Dockerfile
```

3. Tasks

A Gradle Task represents a single atomic piece of work for a Gradle build, such as compiling classes or generating JavaDocs. Tasks are comprised by series of actions as defined by implementations of the `Action` interface. Tasks are allowed to depend on other tasks.

You can create your own custom tasks by extending the `DefaultTask` class which implements the `Task` interface.

In the Getting Started section, you imported all the required classes for this example application

Directly instantiating these classes is **not** supported. You can only instatiate them in the Gradle API or DSL, such as the `build.gradle` file.

3.1. Defined Tasks

These are the defined task names for the example application:

- `createMyAppDockerfile` generates a working `Dockerfile` file based on the template, `Dockefile.tmpl`.
- `buildMyAppImage` builds the Docker image from the generated `Dockerfile`.
  - depends on `createMyAppDockerfile`
- `createMyAppContainer` creates the Docker container.
  - depends on `buildMyAppImage`
- `startMyAppContainer` starts the Docker container.
  - depends on `createMyAppContainer`
- `stopMyAppContainer`
- `executeMyAppContainer`
- `getMyDockerInfo`
- `getMyDockerVersion`

- `getMyDockerImageList`
- `getMyDockerOperation`

You can individually add all of these to your `build.gradle` file. We will discuss each of these and their corresponding classes.

3.2. Create Dockerfile

This task instantiates the `Dockerfile` class to generate a standard `Dockerfile` based on a template file, `Dockerfile.tmpl`. You will need to manually create this file and add the following contents:

```
# template for generated Dockerfile
```

This task contains a few calls to the `instruction()` method that specify the various Docker commands. The `environmentalVariable()` method accepts the environmental variable, `JAVA_OPTS` and the two values. The `instructionsFromTemplate()` method specifies the `Dockerfile.tmpl` file.

```
task createMyAppDockerfile(type: Dockerfile) {
instruction('FROM openjdk:11.0.15-jre-slim')
instruction('LABEL maintainer=\"Michael Redlich\"')
instruction('WORKDIR /app2')
instruction('ENTRYPOINT ["java", "-Xms256m", "-Xmx2048m", "-cp",
"/app2/resources:/muDockerApp/classes:/app2/libs/*", "org.gradle.MainApp"]')
instruction('EXPOSE 9090 5701')
instruction('RUN ls -la')
environmentVariable('JAVA_OPTS', '-XX:+UnlockExperimentalVMOptions
-XX:+UseCGroupMemoryLimitForHeap')
instructionsFromTemplate(file('Dockerfile.tmpl'))
}
```

3.3. Build the Image

The `buildMyAppImage` task instantiates the `DockerBuildImage` class to build the Docker image.

```
task buildMyAppImage(type: DockerBuildImage) {
    dependsOn(createMyAppDockerfile)
    inputDir.set(file('build/docker'))
    images.add('test/app2:latest')
    }
```

3.4. Create the Docker Container

The `createMyAppContainer` task instantiates the `DockerCreateContainer` class.

```
task createMyAppContainer(type: DockerCreateContainer) {
    dependsOn(buildMyAppImage)
    targetImageId(buildMyAppImage.getImageId())
    }
```

3.5. Start the Docker Container

The `startMyAppContainer` task instantiates the `DockerStartContainer` class.

```
task startMyAppContainer(type: DockerStartContainer) {
    dependsOn(createMyAppContainer)
    targetContainerId(createMyAppContainer.getContainerId())
    }
```

3.6. Stop the Docker Container

The `task instantiates the` class.

```
task stopMyAppContainer(type: DockerStopContainer) {
    targetContainerId(createMyAppContainer.getContainerId())
    }
```

3.7. Execute the Container

The `executeMyAppContainer` task instantiates the `DockerExecContainer` class.

```
task executeMyAppContainer(type: DockerExecContainer) {
    targetContainerId(createMyAppContainer.getContainerId())
    }
```

3.8. Obtain the Docker Information

The `getMyDockerInfo` task instantiates the `DockerInfo` class.

```
task getMyDockerInfo(type: DockerInfo) {
    }
```

3.9. Obtain the Docker Version

The `getMyDockerVersion` task instantiates the `DockerVersion` class.

```
task getMyDockerVersion(type: DockerVersion) {
    }
```

3.10. Obtain the List of Docker Images

The `getMyDockerImageList` task instantiates the `DockerListImages` class.

```
task getMyDockerImageList(type: DockerListImages) {
    }
```

4. Build and Execute the Application

You can launch parts of the application with the `gradle` command and the various tasks.

Since we have a defined `application` block, we can execute the `main()` method defined in the MainApp:

```
gradle clean run
```

Let's create and build the image and start the container. The `startMyAppContainer` task depends on the tasks that require execution. Therefore, execute:

```
gradle clean startMyAppContainer
```

```
> Task :buildMyAppImage
Building image using context '/usr/local/apps/gradle-apps/getting-started-building-
docker-images-with-gradle/build/docker'.
Using images 'test/app2:latest'.
Step 1/7 : FROM openjdk:11.0.15-jre-slim
 ---> 699c24828c34
Step 2/7 : LABEL maintainer="Michael Redlich"
 ---> Using cache
 ---> c7f43ff98289
Step 3/7 : WORKDIR /app2
 ---> Using cache
 ---> 7eb1b1aaa358
Step 4/7 : ENTRYPOINT ["java", "-Xms256m", "-Xmx2048m", "-cp",
"/app2/resources:/muDockerApp/classes:/app2/libs/*", "org.gradle.MainApp"]
 ---> Using cache
 ---> 7a6087ee375c
Step 5/7 : EXPOSE 9090 5701
 ---> Using cache
 ---> c2752b3bb2be
Step 6/7 : RUN ls -la
 ---> Using cache
 ---> 8b5c28802e00
Step 7/7 : ENV JAVA_OPTS="-XX:+UnlockExperimentalVMOptions
-XX:+UseCGroupMemoryLimitForHeap"
 ---> Using cache
 ---> c2f34a8c1df6
Successfully built c2f34a8c1df6
Successfully tagged test/app2:latest
Created image with ID 'c2f34a8c1df6'.

> Task :createMyAppContainer
Created container with ID
'b7fb995f1d59698a927333471b446f051ee803ac5a3c174395645bbfbc3b58e8'.

> Task :startMyAppContainer
Starting container with ID
'b7fb995f1d59698a927333471b446f051ee803ac5a3c174395645bbfbc3b58e8'.

BUILD SUCCESSFUL in 4s
5 actionable tasks: 4 executed, 1 from cache
```

4.1. Docker Version

You can `gradle getMyDockerVersion`

```
> Task :getMyDockerVersion
Retrieving Docker version.
Version          : 20.10.21
Git Commit       : 3056208
Go Version       : go1.18.7
Kernel Version   : 5.15.49-linuxkit
Architecture     : amd64
Operating System : linux

BUILD SUCCESSFUL in 1s
1 actionable task: 1 executed
```

4.2. List of Docker Images

`gradle getMyDockerImageList`

```
> Task :getMyDockerImageList
Repository Tags : test/app2:latest
Image ID        :
sha256:c2f34a8c1df67b413537f9d706c85724bec78cfcaa85374194b10fda76e2fbae
Created         : Sun Jan 08 10:41:03 EST 2023
Virtual Size    : 227459049
-----------------------------------------------
Repository Tags : test/app:latest
Image ID        :
sha256:13e86aa2705c759b869ecc3c4de2e3d3a0c44cd13e7d1fa573693e34ebcbefdf
Created         : Sun Jan 08 09:25:07 EST 2023
Virtual Size    : 227459049
-----------------------------------------------
Repository Tags : test/myapp:latest
Image ID        :
sha256:ec013200a08d606abdf922e4941d5d557ed7a1718ad0a7f83a32e236503a70f4
Created         : Sat Jan 07 20:14:53 EST 2023
Virtual Size    : 425724205
-----------------------------------------------
Repository Tags : mongo:latest
Image ID        :
sha256:0850fead9327a6d88722c27116309022d78e9daf526b407a88de09762c32e620
Created         : Thu Dec 08 21:37:35 EST 2022
Virtual Size    : 699901543
-----------------------------------------------
Repository Tags : cassandra:latest
Image ID        :
sha256:5b647422e184fb0fd8f6d5513541e85c06876bdaa68decc026abcb65c3fe4ec5
Created         : Fri Nov 04 19:47:02 EDT 2022
Virtual Size    : 353334938
-----------------------------------------------
Repository Tags : arangodb/arangodb:latest
Image ID        :
```

```
sha256:d81cf81aaa4b1b637874eab9b877e34dcdee97a00800aaece837fd3d32f7eb56
Created         : Thu Sep 29 10:22:06 EDT 2022
Virtual Size    : 439791606
-----------------------------------------------
Repository Tags : jakartaee-cafe:v1
Image ID        :
sha256:f03eac10057c875306561572ce9ce338aa14f4d73917689ec911e891d0a9ce4d
Created         : Tue Sep 13 13:38:25 EDT 2022
Virtual Size    : 484008628
-----------------------------------------------
Repository Tags : openjdk:11.0.15-jre-slim
Image ID        :
sha256:699c24828c341c27d15a4f62b5c8fa3c5c986bf52fa76228906c50634f430311
Created         : Mon Jul 11 22:00:45 EDT 2022
Virtual Size    : 227459049
-----------------------------------------------

BUILD SUCCESSFUL in 1s
1 actionable task: 1 executed
```

5. Plugins

There are three available plugins to create and build a Docker image:

- `DockerRemoteApiPlugin`
- `DockerJavaApplicationPlugin`
- `DockerSpringBootApplicationPlugin`

Details on use cases and corresponding details on all three of these plugins may be found in the Provided Plugins section of the Gradle Docker Plugin User Guide and Examples guide.

For our example application, you will be using the `DockerJavaApplicatioPlugin`.

6. Resources

- JavaDocs: Gradle Docker Plugin
- User Guide: Gradle Docker Plugin User Guide and Examples
- Gradle Plugin Portal
- Documentation: Using Gradle Plugins

7. Summary

That's it! You've now successfully configured and built a Java application project with Gradle. You've learned how to:

- Initialize a project that produces a Java application
- Create a modular software project by combining multiple subprojects
- Share build configuration logic between subprojects using convention plugins in buildSrc
- Run similar named tasks in all subprojects
- Run a task in a specific subproject
- Build, bundle and run the application

8. Next Steps

8.1. User Guide and Examples

You can learn more from Gradle Docker Plugin User Guide and Examples documentation.