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.DockerOperation

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'
implementation(buildsrclibs.asciidoctor.jvm.plugin)
runtimeOnly(buildsrclibs.asciidoctorj.tabbed.code.extension)
}
```

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 build, such as compiling classes or generating JavaDocs. Tasks are allowed to depend on other tasks.

You can create your own 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 tasks 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`

We will individually define each of these and their corresponding classes.

```
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'))
}
```

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

`gradle clean startMyAppContainer`

4. 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`.

5. Resources

- JavaDocs: Gradle Docker Plugin
- User Guide: Gradle Docker Plugin User Guide and Examples
- Gradle Plugin Portal

- Documentation: Using Gradle Plugins

6. 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

7. Next Steps

7.1. User Guide and Examples

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