Getting Started Building Docker Images with Gradle

1. Introduction

This tutorial is ...

1.1. What You'll Build

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

1.2. 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.

1.3. User Guide and Examples

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

2. Getting Started

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

2.1. Define Appropriate Plugins for the Application

Let's incrementally build our `build.gradle` file by adding the `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'
}
```

2.2. 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'
```

2.3. Defined Tasks

These are the defined tasks for building and working with your Docker image.

2.3.1. Starting a Docker Image

- `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`

```
gradle clean startMyAppContainer
```

2.3.2. Docker Utilities

- `stopMyAppContainer`
- `executeMyAppContainer`
- `getMyDockerInfo`
- `getMyDockerVersion`
- `getMyDockerImageList`
- `getMyDockerOperation`

3. 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`.

4. Resources

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

5. 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

6. Next Steps