


[ANDROID](#) [CORE JAVA](#) [DESKTOP JAVA](#) [ENTERPRISE JAVA](#) [JAVA BASICS](#) [JVM LANGUAGES](#) [SOFTWARE DEVELOPMENT](#) [DEVOPS](#)
[Home](#) » [DevOps](#) » [Docker](#) » Docker Hello World Example

ABOUT HARIHARAN NARAYANAN



Hari graduated from the School of Computer and Information Sciences in the University of Hyderabad. Over his career he has been involved in many complex projects in mobile applications, enterprise applications, distributed applications, micro-services, and other platforms and frameworks. He works as a consultant and is mainly involved with projects based on Java, C++ and Big Data technologies.



Docker Hello World Example

Posted by: Hariharan Narayanan | in Docker | November 4th, 2016 | 8 Comments | 4213 Views

In this example we will explore different ways of running basic "Hello, World" containers in Docker.

1. Introduction

Creating a Docker Hello World container and getting it to work verifies that you have the Docker infrastructure set up properly in your development system. We will go about this in 4 different ways:

1. Run Docker hello world image provided by Docker
2. Get a "Hello, world" printed from another basic Docker image
3. Write a Simple "Hello, World" Program in Java and Run it Within a Docker Container
4. Execute a "Hello, World" Program in Java Using Gradle and Docker

Want to master Docker?

Subscribe to our newsletter and download Docker Containerization Cookbook [right now!](#)

In order to help you master Docker, we have compiled a kick-ass guide with all the basic concepts of the Docker container system! Besides studying them online you may download the eBook in PDF format!

Email address:

[Sign up](#)

Table Of Contents

1. Introduction
2. Run Docker Hello World image provided by Docker
3. Get a "Hello, world" Printed from Another Basic Docker Image
4. Write a Simple "Hello, World" Program in Java and Run it Within a Docker Container
 - 4.1. Create HelloWorld.java
 - 4.2. Create a Dockerfile
 - 4.3. Create Docker Hello World Image and Run it
5. Execute a "Hello, World" Program in Java Using Gradle and Docker
 - 5.1. Initialize a New Java Project and Create HelloWorld.java

NEWSLETTER

167,740 insiders are already e weekly updates and complimentary whitepapers!

Join them now to gain [EX ACCESS](#) to the latest news in the J as well as insights about Android, S Groovy and other related technologi

Email address:

☒ Receive Java & Developer job al your Area

[Sign up](#)

JOIN US



With **1,240,6** unique visitors **500** authors placed among related sites a Constantly bei lookout for pa encourage you So If you have unique and interesting content then you check out our **JCG** partners program. You be a **guest writer** for Java Code Geel your writing skills!

- 5.2. Create a Dockerfile
- 5.3. Include Gradle-Docker Plugin
- 5.4. Configure the Docker Tasks in Gradle Build File
- 5.5. Create Docker Container and Run it
- 6. Summary
- 7. Download

You should have Docker installed already to follow the examples. Please go to the Docker home page to install the Docker engine before proceeding further, if needed. On Linux, please do ensure that your user name is added into the "docker" group while installing so that you need not invoke sudo to run the Docker client commands. You should also have Java and Gradle installed to try examples 3 and 4. Install your favorite J2SE distribution to get Java and install Gradle from gradle.org.

So let us get started!

2. Run Docker Hello World image provided by Docker

This is the simplest possible way to verify that you have Docker installed correctly. Just run the hello-world Docker image in a terminal.

```
1 | $ docker run hello-world
```

This command will download the

```
hello-world
```

Docker image from the Dockerhub, if not present already, and run it. As a result, you should see the below output if it went well.

```
File Edit View Search Terminal Help
sh-4.2$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world

c04b14da8d14: Pull complete
Digest: sha256:0256e8a36e2070f7bf2d0b0763dbabdd67798512411de4cdc f9431a1feb60fd9
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker Hub account:
https://hub.docker.com

For more examples and ideas, visit:
https://docs.docker.com/engine/userguide/

sh-4.2$
```



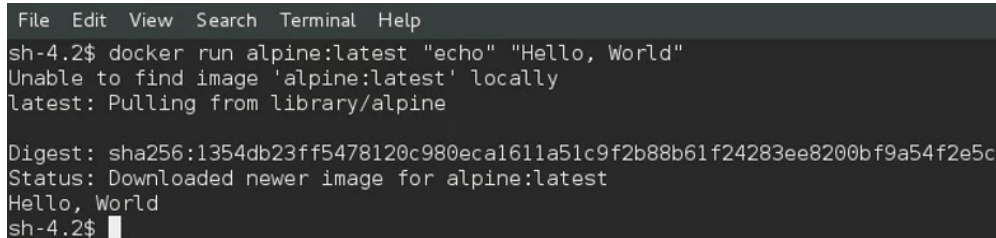
Run Docker Hello-World Image

3. Get a "Hello, world" Printed from Another Basic Docker Image

This too is simple. Docker provides a few baseimages that can be used directly to print a "Hello, World". This can be done as shown below and it verifies that you have a successful installation of Docker.

```
1 | $ docker run alpine:latest "echo" "Hello, World"
```

This command downloads the Alpine baseimage the first time and creates a Docker container. It then runs the container and executes the echo command. The echo command echoes the "Hello, World" string. As a result, you should see the output as below.

A terminal window with a dark background and light text. The menu bar at the top includes 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The terminal shows the command 'sh-4.2\$ docker run alpine:latest "echo" "Hello, World"' and its output: 'Unable to find image 'alpine:latest' locally', 'latest: Pulling from library/alpine', 'Digest: sha256:1354db23ff5478120c980eca1611a51c9f2b88b61f24283ee8200bf9a54f2e5c', 'Status: Downloaded newer image for alpine:latest', and 'Hello, World'. The prompt 'sh-4.2\$' is followed by a cursor.

```
File Edit View Search Terminal Help
sh-4.2$ docker run alpine:latest "echo" "Hello, World"
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
Digest: sha256:1354db23ff5478120c980eca1611a51c9f2b88b61f24283ee8200bf9a54f2e5c
Status: Downloaded newer image for alpine:latest
Hello, World
sh-4.2$
```

Use the Alpine Docker Image to Print Hello World

4. Write a Simple “Hello, World” Program in Java and Run it Within a Docker Container

Let us take it up a notch now. Let us write a simple hello world program in Java and execute it within a Docker container.

4.1. Create HelloWorld.java

First of all, let us create a simple Java program that prints “Hello, World”. Open up your favorite text editor and enter the following code:

```
1 public class HelloWorld {
2     public static void main(String[] args) {
3         System.out.println("Hello, World");
4     }
5 }
```

This is a standard HelloWorld program in Java very closely resembling the one provided in the Official Java tutorial. Save the file as

HelloWorld.java

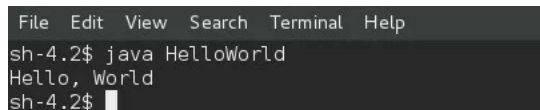
. Now, compile this file using the Java compiler.

```
1 $ javac HelloWorld.java
```

This should create the class file

HelloWorld.class.

Normally, we would use the java command to execute HelloWorld as below

A terminal window with a dark background and light text. The menu bar at the top includes 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The terminal shows the command 'sh-4.2\$ java HelloWorld' and its output: 'Hello, World'. The prompt 'sh-4.2\$' is followed by a cursor.

```
File Edit View Search Terminal Help
sh-4.2$ java HelloWorld
Hello, World
sh-4.2$
```

Run HelloWorld.main() in Your Native OS

But we want to run this from within a Docker container. To accomplish that we need to create a Docker image. Let us do that now.

4.2. Create a Dockerfile

To create a new Docker image we need to create a Dockerfile. A Dockerfile defines a docker image. Let us create this now. Create a new file named

```
dockerfile
```

and enter the following in it and save it.

```
1 FROM alpine:latest
2 ADD HelloWorld.class HelloWorld.class
3 RUN apk --update add openjdk8-jre
4 ENTRYPOINT ["java", "-Djava.security.egd=file:/dev/./urandom", "HelloWorld"]
```

The complete Dockerfile syntax can be found from Docker docs but here is briefly what we have done. We extend our image from the Alpine baseimage. We next added

```
HelloWorld.class
```

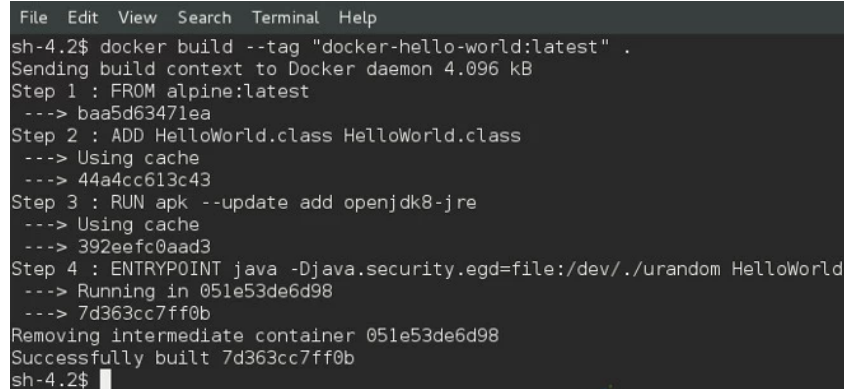
into the image with the same name. Later we installed a JRE environment using OpenJDK. Finally, we gave the command to execute when this image is run – that is to run our HelloWorld in the JVM.

4.3. Create Docker Hello World Image and Run it

Now, build an image from this Dockerfile by executing the below command.

```
1 | $ docker build --tag "docker-hello-world:latest" .
```

As a result of this you should see the following output



```
File Edit View Search Terminal Help
sh-4.2$ docker build --tag "docker-hello-world:latest" .
Sending build context to Docker daemon 4.096 kB
Step 1 : FROM alpine:latest
--> baa5d63471ea
Step 2 : ADD HelloWorld.class HelloWorld.class
--> Using cache
--> 44a4cc613c43
Step 3 : RUN apk --update add openjdk8-jre
--> Using cache
--> 392eefc0aad3
Step 4 : ENTRYPOINT java -Djava.security.egd=file:/dev/./urandom HelloWorld
--> Running in 051e53de6d98
--> 7d363cc7ff0b
Removing intermediate container 051e53de6d98
Successfully built 7d363cc7ff0b
sh-4.2$
```

Build Docker Image Called docker-hello-world

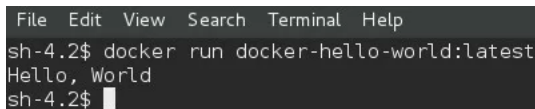
Finally, run the Docker image to see the

```
"Hello, world"
```

printed.

```
1 | $ docker run docker-hello-world:latest
```

As a result of this you should see the below output



```
File Edit View Search Terminal Help
sh-4.2$ docker run docker-hello-world:latest
Hello, world
sh-4.2$
```

Run the Docker Image docker-hello-world

That's it. This verifies that you can create your own custom Docker images too and run them.

5. Execute a “Hello, World” Program in Java Using Gradle and Docker

Let us take it up another notch now. Real world programs are more complex than the ones shown above. So let us create another hello world Java program and run it in a Docker container using a few best-practices of the Java ecosystem. Furthermore, let us set up a Java project using Gradle, add the Docker plugin for Gradle into it, create a Docker image and run it using Gradle.

5.1. Initialize a New Java Project and Create HelloWorld.java

First of all, Create a new folder called “docker-hello-world-example”, open a terminal and change into this folder. Next, use Gradle to initialize a new Java project.

```
1 | $ gradle init --type java-library
```

Once done you should see the following folder structure created. You can freely delete the files

```
Library.java
```

and

```
LibraryTest.java
```

. We will not be needing them.

```
File Edit View Search Terminal Help
sh-4.2$ tree
.
|-- build.gradle
|-- gradle
|   |-- wrapper
|   |   |-- gradle-wrapper.jar
|   |   |-- gradle-wrapper.properties
|-- gradlew
|-- gradlew.bat
|-- settings.gradle
|-- src
|   |-- main
|   |   |-- java
|   |   |   |-- Library.java
|   |-- test
|   |   |-- java
|   |   |   |-- LibraryTest.java
.
7 directories, 8 files
sh-4.2$
```

Folder structure for the Sample Java Project Created by Gradle

Now, create a new file

```
src/main/java/com/javacodegeeks/examples/HelloWorld.java
```

and enter the following in it. Save it once done.

```
1 | package com.javacodegeeks.examples;
2 |
3 | public class HelloWorld {
4 |     public static void main(String[] args) {
5 |         System.out.println("Hello, World");
6 |     }
7 | }
```

5.2. Create a Dockerfile

Next, create a Docker file in src/main/resources called “src/main/resources/Dockerfile” and enter the following. Save and close the file once done.

```
1 | FROM alpine:latest
2 | RUN apk --update add openjdk8-jre
3 | COPY docker-hello-world-example.jar app.jar
4 | ENTRYPOINT ["java", "-Djava.security.egd=file:/dev/./urandom", "-jar", "app.jar"]
```

Let us quickly go through what we did in the above Dockerfile. We derive our image from the Alpine Linux base image. Next, we installed J2SE from OpenJDK. Next, we copied the jar file generated by the build process as

```
app.jar
```

into the Docker image. Finally, we set the command to be run when the Docker container is run which is to execute the app.jar file.

5.3. Include Gradle-Docker Plugin

Next, make the following changes into the

```
build.gradle
```

file to update the Jar manifest.

```
1  mainClassName = 'com.javacodegeeks.examples.HelloWorld'
2
3  jar {
4      manifest {
5          attributes(
6              'Main-Class': mainClassName
7          )
8      }
9  }
```

We set the

```
mainClassName
```

to

```
com.javacodegeeks.examples.HelloWorld
```

and set the same attribute to be inserted into the manifest file of the jar that will be created.

Next, insert the following changes at the very top of

```
build.gradle
```

file to configure and add the Gradle plugins for Docker.

```
01  buildscript {
02      repositories {
03          maven {
04              url "https://plugins.gradle.org/m2/"
05          }
06      }
07      dependencies {
08          classpath "gradle.plugin.com.palantir.gradle.docker:gradle-docker:0.9.1"
09      }
10  }
11
12  plugins {
13      id 'com.palantir.docker' version '0.9.1'
14      id 'com.palantir.docker-run' version '0.9.1'
15  }
16
17  apply plugin: 'com.palantir.docker'
```

Here we basically added the Gradle plugins called

```
docker
```

and

```
docker-run
```

provided by Palantir. These plugins enable us to create Docker containers and run them using Gradle. We added the URL to tell where to find the gradle-docker plugin (

```
https://plugins.gradle.org/m2/
```

). We also added the plugins into the classpath as build dependencies.

5.4. Configure the Docker Tasks in Gradle Build File

Next, insert the following changes at the bottom of build.gradle file to configure the details of the Docker image.

```
1  docker {
2      name 'com.javacodegeeks.examples/docker-hello-world:1'
3      tags 'latest'
4      dockerfile 'src/main/docker/Dockerfile'
5      dependsOn tasks.jar
6  }
```

Here we configured the details of the docker image. We assigned the image name

```
com.javacodegeeks.examples/docker-hello-world
```

with version

```
1
```

. We assigned it the tag

```
latest
```

and gave the path where to find the Docker file. Finally, we made it depend on the Gradle task

```
jar
```

so that the output of the jar task will be fed to this task.

Finally, insert the following changes at the bottom of build.gradle file to configure the details of the Docker container.

```
1  dockerRun {
2      name 'docker-hello-world-container'
3      image 'com.javacodegeeks.examples/docker-hello-world:1'
4      command 'java', '-Djava.security.egd=file:/dev/./urandom', '-jar', 'app.jar'
5  }
```

Here we configured the details of the docker container. We assigned the container name "docker-hello-world-container". We assigned it the image from where to create the container and gave it a command to execute when running the container.

5.5. Create Docker Container and Run it

Now we are ready to build and run this. Use Gradle to build and run this program

```
1 | $ ./gradlew clean build docker dockerRun
```

You will notice that the code was built, jar was created, Docker image and containers were created, and the container was run too. However, there is no "Hello, World" actually printed on the screen here.

```
File Edit View Search Terminal Help
sh-4.2$ ./gradlew clean build docker dockerRun
Starting a Gradle Daemon (subsequent builds will be faster)
:clean
:compileJava
:processResources UP-TO-DATE
:classes
:jar
:startScripts
:distTar
:distZip
:dockerfileZip
:assemble
:compileTestJava UP-TO-DATE
:processTestResources UP-TO-DATE
:testClasses UP-TO-DATE
:test UP-TO-DATE
:check UP-TO-DATE
:build
:dockerClean UP-TO-DATE
:dockerPrepare
:docker
Sending build context to Docker daemon 245.8 kB
Step 1 : FROM alpine:latest
--> baa5d63471ea
Step 2 : RUN apk --update add openjdk8-jre
--> Using cache
--> 2b4b741d1723
Step 3 : COPY docker-hello-world-example.jar app.jar
--> da7f4534b8f0
Removing intermediate container 896d167763e9
Step 4 : ENTRYPOINT java -Djava.security.egd=file:/dev/./urandom -jar app.jar
--> Running in b981f0cb206a
--> c074699e538d
Removing intermediate container b981f0cb206a
Successfully built c074699e538d
:dockerRun
0a68b3942fcd6333f4bc13e013931402ea3f3e31d0214d58c28f0d589c1fd05b
:dockerRunStatus
Docker container 'docker-hello-world-container' is RUNNING.

BUILD SUCCESSFUL

Total time: 13.492 secs
sh-4.2$
```



Run the Gradle Tasks clean, build, docker, and dockerrun

The last line simply says that the Docker container 'docker-hello-world-container' is running. To see if it printed "Hello, World" successfully, we need to check the logs created by the Docker container. This can be checked by executing the following command.

```
1 | $ docker logs docker-hello-world-container
```

```
File Edit View Search Terminal Help
sh-4.2$ docker logs docker-hello-world-container
Hello, World
sh-4.2$
```

Check the Docker Container Logs to Check if Hello world was Printed

There it is!

6. Summary

In this example we learned how to create Docker Hello world type containers in 4 different ways:

1. First, we learned how to run the hello-world image provided by Docker. This printed Hello World as soon as it was run
2. Next, we learned how to run the Alpine image and print a Hello World using the echo shell command
3. Next, we learned how to get the basic Hello World program into a Docker image and run it to print Hello World. The source code for this can be downloaded from the links given below.
4. Finally, we learned how to set up a structured Java project using Gradle and generate Docker images and Containers using Gradle plugins. We used this structure to create a Hello World Java project and got it to run in a container. Later, we verified the Docker logs for the container to see that Hello World was printed when the container was run. The source code for this also can be downloaded from the links given below.

7. Download the Source Code

Download

The source code for example 3 can be downloaded here: [docker-hello-world.zip](#)

The source code for example 4 can be downloaded here: [docker-hello-world-example.zip](#)



(-1 rating, 1 votes) 8 Comments 4213 Views Tweet it!

Do you want to know how to develop your skillset to become a **Java Rockstar?**

Subscribe to our newsletter to start Rocking right now!

To get you started we give you our best selling eBooks for **FREE!**

1. JPA Mini Book
2. JVM Troubleshooting Guide
3. JUnit Tutorial for Unit Testing
4. Java Annotations Tutorial
5. Java Interview Questions
6. Spring Interview Questions
7. Android UI Design

and many more

Email address:

☒ Receive Java & Developer job alerts in your Area

Sign up

LIKE THIS ARTICLE? READ MORE FROM JAVA CODE GEEKS

8 Leave a Reply



Join the discussion...

8 0 1 ⚡ 🔥

6

✉ Subscribe ▾

▲ newest ▲ oldest ▲ most voted



Guest

Paul



To resolve "java.lang.ClassNotFoundException: org.gradle.api.internal.component.Usage", simply change the version of the gradle-docker plugin from 0.9.1 to 0.13.0.

+ 1 - [Reply](#)

🕒 1 year ago



Guest

Paul



To resolve "Cannot get the value of write-only property 'dockerfile' for object of type com.palantir.gradle.docker.DockerExtension.", simply change:

```
dockerfile 'src/main/docker/Dockerfile'
```

to

```
dockerfile project.file('src/main/docker/Dockerfile')
```

+ 0 - [Reply](#)

🕒 1 year ago



Guest

Paul



There is a path mismatch in this example, step 5.2 uses: "src/main/resources/Dockerfile", but the build.gradle file contains, "dockerfile 'src/main/docker/Dockerfile'" (See 5.4)

+ 0 - [Reply](#)

🕒 1 year ago



Guest

gowsalya



Very good brief and this post helped me alot. Say thank you I searching for your facts. Thanks for sharing with us! <https://www.besanttechnologies.com/training-courses/other-training-courses/devops-training-institute-in-chennai>

+ 0 - [Reply](#)

🕒 4 months ago



Guest

aruna ram



Excellent post, truly well post. It will be used for improve our self. Thank you for sharing this content...!

+ 0 - [Reply](#)

🕒 3 months ago



Guest

Harish



I am a beginner at java programming. Your blog is very useful for me.

+ 0 - [Reply](#)

🕒 3 months ago



Guest

amsaleka



Wow!! Really a nice Article. Thank you so much for your efforts. Definitely, it will be helpful for others. I would like to follow your blog. Share more like this. Thanks Again.
iot training in Chennai | Best iot Training Institute in Chennai

+ 0 - [Reply](#)

🕒 1 month ago



harman



how to use volumes in docker in java program so that changes made in java program reflected during running of container without building the dockerfile again and again?

KNOWLEDGE BASE

Courses
Minibooks
News
Resources
Tutorials

THE CODE GEEKS NETWORK

.NET Code Geeks
Java Code Geeks
System Code Geeks
Web Code Geeks

HALL OF FAME

Android Alert Dialog Example
Android OnClickListener Example
How to convert Character to String and a String to Character Array in Java
Java Inheritance example
Java write to File Example
java.io.FileNotFoundException – How to solve File Not Found Exception
java.lang.arrayindexoutofboundsexception – How to handle Array Index Out Of Bounds Exception
java.lang.NoClassDefFoundError – How to solve No Class Def Found Error
JSON Example With Jersey + Jackson
Spring JdbcTemplate Example

ABOUT JAVA CODE GEEKS

JCGs (Java Code Geeks) is an independent online community focused on creating the ultimate Java to Java developers resource center; targeted at the technical and non-technical team lead (senior developer), project manager and junior developer. JCGs serve the Java, SOA, Agile and Telecom communities with daily news, webinars, domain experts, articles, tutorials, reviews, announcements, code snippets and open source projects.

DISCLAIMER

All trademarks and registered trademarks appearing on Java Code Geeks are the property of their respective owners. Java is a trademark or registered trademark of Oracle Corporation in the United States and other countries. Examples Java Code Geeks is not connected to Oracle Corporation and is not sponsored by Oracle Corporation.