Liberica NIK
Containerizing native images

Liberica NIK
Revision 1.0
October 17, 2023

bellsoft

BellSoft software contains open source software. Additional information about third party code is available at https://bell-sw.com/third_party_licenses. You can also get more information on how to get a copy of source code by contacting info@bell-sw.com.

The use of any software product referenced in this document is governed by the applicable license agreement, which is not modified in any way by the terms of this notice.

Contents

1. Introduction

Native Image technology is gaining traction among developers whose primary goal is to accelerate startup time of applications. This document will guide you through containerizing native images for further deployment in the Cloud. We will use Liberica Native Image Kit (NIK) as a native-image compiler and Alpaquita Stream as a base image.

2. Create an application

Create a folder for your demo project. Then go to the project folder and build a simple Java application in the console:

```
cat >./Demo.java <<EOL
public class Demo {
    public static void main(String[] args) {
        System.out.println("Hello from Native Image!");
    }
}
EOL
```

3. Pull a Docker image

BellSoft provides a variety of images with Liberica NIK hosted on Docker Hub. For instance, if you want Liberica NIK 21 for Java 11 and musl libc, pull the following image:

```
docker container run --rm -it bellsoft/liberica-native-image-kit-container:jdk-11-nik-21.3.3-stream-musl
```

You can skip this step, but pulling an image is recommended if you don't want to do that every time you repeat the build process.

4. Write a Dockerfile

We are going to build a native image straight in a container, which is useful when the development and deployment architectures are different. It is also helpful if you want to build a musl-based image, which takes up less memory than a glibc-based one.

We need to write a Dockerfile to generate a Docker image container. Put the following file into the application folder:

```
FROM bellsoft/liberica-native-image-kit-container:jdk-11-nik-21.3.3-stream-musl
WORKDIR /home/myapp
COPY Demo.java /home/myapp/
RUN javac Demo.java
RUN native-image Demo
FROM bellsoft/alpaquita-linux-base:stream-musl-230404
WORKDIR /home/myapp
COPY --from=0 /home/myapp/demo .
CMD [“./demo”]
```

The code above does the following:

1. Specifies the base image for Native Image generation;
2. Points to the directory where the image will execute inside Docker;
3. Copies the program to the directory;
4. Runs the javac compiler to create the bytecode of our app;
5. Runs the native-image tool to build a native image;
6. Creates another image with Alpaquita Linux base image (the native image doesn’t need a JVM to execute);
7. Specifies the executable directory;
8. Copies the app into the new image;
9. Runs the program inside the container.

5. Build a native image container

We recommend closing the browser, IDE, and other programs consuming a lot of memory before building a container. To generate a native image and containerize it, run

```
docker build .
```

Check that the image was created with the following command:

```
docker images
REPOSITORY     TAG            IMAGE ID       CREATED              SIZE
<none>         <none>         2921e3483bb2   21 seconds ago       18.4MB
```

Tag the newly created image:

```
docker tag 2921e3483bb2 nik-example
```

Now you can run the image with the following command:

```
docker run -it -rm 2921e3483bb2
Hello from Native Image!
```

6. Considerations

We used a simple program that did not require any manual configuration. But dynamic Java features (Reflection, JNI, Serialization, etc.) are not supported by GraalVM, so you have to make the native-image tool aware of them.

Liberica NIK

Containerizing native images

bellsoft