Quarkus - Using SSL With Native Executables

We are quickly moving to an SSL-everywhere world so being able to use SSL is crucial.

In this guide, we will discuss how you can get your native executables to support SSL, as native executables don't support it out of the box.

> ℹ️ If you don't plan on using native executables, you can pass your way as in JDK mode, SSL is supported without further manipulations.

Prerequisites

To complete this guide, you need:

- less than 20 minutes
- an IDE
- GraalVM installed with `JAVA_HOME` and `GRAALVM_HOME` configured appropriately
- Apache Maven 3.5.3+

This guide is based on the REST client guide so you should get this Maven project first.

Clone the Git repository: `git clone https://github.com/quarkusio/quarkus-quickstarts.git`, or download an archive.

The project is located in the `rest-client-quickstart` directory.

Looks like it works out of the box?!?

If you open the application's configuration file (`src/main/resources/application.properties`), you can see the following line:

```
org.acme.restclient.CountriesService/mp-
rest/url=https://restcountries.eu/rest
```

which configures our REST client to connect to an SSL REST service.

Now let's build the application as a native executable and run the tests:

```
./mvnw clean install -Pnative
```

And we obtain the following result:

```
[INFO]
-------------------------------------------------------------------
-----
[INFO] BUILD SUCCESS
[INFO]
-------------------------------------------------------------------
-----
```

So, yes, it appears it works out of the box and this guide is pretty useless.

It's not. The magic happens when building the native executable:

```
[INFO] [io.quarkus.creator.phase.nativeimage.NativeImagePhase]
/opt/graalvm/bin/native-image -J
-Djava.util.logging.manager=org.jboss.logmanager.LogManager -J
-Dcom.sun.xml.internal.bind.v2.bytecode.ClassTailor.noOptimize=true
-H:InitialCollectionPolicy=com.oracle.svm.core.genscavenge.Collecti
onPolicy$BySpaceAndTime -jar rest-client-1.0-SNAPSHOT-runner.jar -J
-Djava.util.concurrent.ForkJoinPool.common.parallelism=1
-H:+PrintAnalysisCallTree -H:EnableURLProtocols=http,https --enable
-all-security-services -H:-SpawnIsolates -H:+JNI --no-server -H:
-UseServiceLoaderFeature -H:+StackTrace
```

The important elements are these 3 options:

```
-H:EnableURLProtocols=http,https --enable-all-security-services
-H:+JNI
```

They enable the native SSL support for your native executable.

As SSL is de facto the standard nowadays, we decided to enable its support automatically for some of our extensions:

- the Agroal connection pooling extension (`quarkus-agroal`),

- the Amazon DynamoDB extension (`quarkus-amazon-dynamodb`),

- the Hibernate Search Elasticsearch extension (`quarkus-hibernate-search-elasticsearch`),

- the Infinispan Client extension (`quarkus-infinispan-client`).

- the Jaeger extension (`quarkus-jaeger`),

- the JGit extension (`quarkus-jgit`),

- the Keycloak extension (`quarkus-keycloak`),

- the Kubernetes client extension (`quarkus-kubernetes-client`),

- the Mailer extension (`quarkus-mailer`),

- the MongoDB extension (`quarkus-mongodb-client`),

- the Neo4j extension (`quarkus-neo4j`),

- the OAuth2 extension (`quarkus-elytron-security-oauth2`),

- the REST client extension (`quarkus-rest-client`).

As long as you have one of those extensions in your project, the SSL support will be enabled by default.

Now, let's just check the size of our native executable as it will be useful later:

```
$ ls -lh target/rest-client-1.0-SNAPSHOT-runner
-rwxrwxr-x. 1 gsmet gsmet 34M Feb 22 15:27 target/rest-client-1.0-SNAPSHOT-runner
```

Let's disable SSL and see how it goes

Quarkus has an option to disable the SSL support entirely. Why? Because it comes at a certain cost. So if you are sure you don't need it, you can disable it entirely.

First, let's disable it without changing the REST service URL and see how it goes.

Open `src/main/resources/application.properties` and add the following line:

```
quarkus.ssl.native=false
```

And let's try to build again:

```
./mvnw clean install -Pnative
```

The native executable tests will fail with the following error:

```
Exception handling request to /country/name/greece:
com.oracle.svm.core.jdk.UnsupportedFeatureError: Accessing an URL
protocol that was not enabled. The URL protocol https is supported
but not enabled by default. It must be enabled by adding the
--enable-url-protocols=https option to the native-image command.
```

This error is the one you obtain when trying to use SSL while it was not explicitly enabled in your native executable.

Now, let's change the REST service URL to **not** use SSL in `src/main/resources/application.properties`:

```
org.acme.restclient.CountriesService/mp-
rest/url=http://restcountries.eu/rest
```

And build again:

```
./mvnw clean install -Pnative
```

If you check carefully the native executable build options, you can see that the SSL related options are gone:

```
[INFO] [io.quarkus.creator.phase.nativeimage.NativeImagePhase]
/opt/graalvm/bin/native-image -J
-Djava.util.logging.manager=org.jboss.logmanager.LogManager -J
-Dcom.sun.xml.internal.bind.v2.bytecode.ClassTailor.noOptimize=true
-H:InitialCollectionPolicy=com.oracle.svm.core.genscavenge.Collecti
onPolicy$BySpaceAndTime -jar rest-client-1.0-SNAPSHOT-runner.jar -J
-Djava.util.concurrent.ForkJoinPool.common.parallelism=1
-H:+PrintAnalysisCallTree -H:EnableURLProtocols=http -H:
-SpawnIsolates -H:+JNI --no-server -H:-UseServiceLoaderFeature
-H:+StackTrace
```

And we end up with:

```
[INFO]
--------------------------------------------------------------------
-----
[INFO] BUILD SUCCESS
[INFO]
--------------------------------------------------------------------
-----
```

You remember we checked the size of the native executable with SSL enabled? Let's check again with SSL support entirely disabled:

```
$ ls -lh target/rest-client-1.0-SNAPSHOT-runner
-rwxrwxr-x. 1 gsmet gsmet 25M Feb 22 15:19 target/rest-client-1.0-
SNAPSHOT-runner
```

Yes, it is now **25 MB** whereas it used to be **34 MB**. SSL comes with a 9 MB overhead in native executable size.

And there's more to it.

Let's start again with a clean slate

Let's revert the changes we made to the configuration file and go back to SSL with the following command:

```
git checkout -- src/main/resources/application.properties
```

And let's build the native executable again:

```
./mvnw clean install -Pnative
```

The TrustStore path

You haven't noticed anything but, while building the image, Quarkus has automatically set `javax.net.ssl.trustStore` to point to the `cacerts` file bundled in the GraalVM distribution. This file contains the root certificates.

This is useful when running tests but, obviously, it is not portable as this path is hardcoded.

You can check that pretty easily:

- move your GraalVM directory to another place (let's call it `<new-graalvm-home>`)
- run the native executable `./target/rest-client-1.0-SNAPSHOT-runner`
- in a browser, go to `http://localhost:8080/country/name/greece`
- you will have an Internal Server Error
- in your terminal, you should have an exception: `java.security.InvalidAlgorithmParameterException: the trustAnchors parameter must be non-empty`
- hit `Ctrl+C` to stop the application

To make it work, you need to manually set `javax.net.ssl.trustStore` to point to the new GraalVM home:

```
./target/rest-client-1.0-SNAPSHOT-runner
-Djavax.net.ssl.trustStore=<new-graalvm
-home>/jre/lib/security/cacerts
```

Now, the application should work as expected:

- in a browser, go to `http://localhost:8080/country/name/greece`

- you should see a JSON output with some information about Greece

- hit `Ctrl+C` to stop the application

> The root certificates file of GraalVM might not be totally up to date. If you have issues with some certificates, your best bet is to include the `cacerts` file of a regular JDK instead.

> Don't forget to move your GraalVM directory back to where it was.

Working with containers

When working with containers, the idea is to bundle both the SunEC library and the certificates in the container and to point your binary to them using the system properties mentioned above.

You can for example modify your `Dockerfile.native` as follows to copy the required files to your final image:

Java 8

```
FROM quay.io/quarkus/ubi-quarkus-native-image:19.3.1-java8 as
nativebuilder
RUN mkdir -p /tmp/ssl-libs/lib \
  && cp /opt/graalvm/jre/lib/security/cacerts /tmp/ssl-libs \
  && cp /opt/graalvm/jre/lib/amd64/libsunec.so /tmp/ssl-libs/lib/

FROM registry.access.redhat.com/ubi8/ubi-minimal
WORKDIR /work/
COPY --from=nativebuilder /tmp/ssl-libs/ /work/
COPY target/*-runner /work/application
RUN chmod 775 /work /work/application
EXPOSE 8080
CMD ["./application", "-Dquarkus.http.host=0.0.0.0", "-
Djava.library.path=/work/lib", "-
Djavax.net.ssl.trustStore=/work/cacerts"]
```

Java 11

```
FROM quay.io/quarkus/ubi-quarkus-native-image:19.3.1-java11 as
nativebuilder
RUN mkdir -p /tmp/ssl-libs/lib \
  && cp /opt/graalvm/lib/security/cacerts /tmp/ssl-libs \
  && cp /opt/graalvm/lib/libsunec.so /tmp/ssl-libs/lib/

FROM registry.access.redhat.com/ubi8/ubi-minimal
WORKDIR /work/
COPY --from=nativebuilder /tmp/ssl-libs/ /work/
COPY target/*-runner /work/application
RUN chmod 775 /work /work/application
EXPOSE 8080
CMD ["./application", "-Dquarkus.http.host=0.0.0.0", "-
Djava.library.path=/work/lib", "-
Djavax.net.ssl.trustStore=/work/cacerts"]
```

Conclusion

We make building native executable easy and, even if the SSL support in GraalVM is still requiring some serious thinking, it should be mostly transparent when using Quarkus.

Hopefully, the situation will improve in the future: the native executables size overhead will be reduced and the SunCE library might not be needed anymore.

We track GraalVM progress on a regular basis so we will promptly integrate in Quarkus any improvement with respect to SSL support.