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.

> 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 (Java 11) installed with `JAVA_HOME` and `GRAALVM_HOME` configured appropriately
- Apache Maven 3.6.3

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 Redis client extension (quarkus-redis-client),
- 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),
- the Reactive client for PostgreSQL extension (quarkus-reactive-pg-client),
- the Reactive client for MySQL extension (quarkus-reactive-mysql-client),
- the Reactive client for DB2 extension (quarkus-reactive-db2-client).
- the Cassandra client extensions (cassandra-quarkus-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-quickstart-1.0-SNAPSHOT-runner
-rwxrwxr-x. 1 gandrian gandrian 46M Jun 11 13:01 target/rest-client-quickstart-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-quickstart-1.0-SNAPSHOT-runner
-rwxrwxr-x. 1 gandrian gandrian 35M Jun 11 13:06 target/rest-
client-quickstart-1.0-SNAPSHOT-runner
```

Yes, it is now **35 MB** whereas it used to be **46 MB**. SSL comes with a 11 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

⚠ This behavior is new to GraalVM 19.3+.

When creating a native binary, GraalVM embraces the principle of "immutable security" for the root certificates. This essentially means that the root certificates are fixed at image build time, based on the certificate configuration used at that build time (which for Quarkus means when you perform a build having `quarkus.package.type=native` set). This avoids shipping a `cacerts` file or requiring a system property be set in order to set up root certificates that are provided by the OS where the binary runs.

As a consequence, system properties such as `javax.net.ssl.trustStore` do not have an effect at run time, so when the defaults need to be changed, these system properties must be provided at image build time. The easiest way to do so is by setting `quarkus.native.additional-build-args`. For example:

```
quarkus.native.additional-build-args=-J-
Djavax.net.ssl.trustStore=/tmp/mycerts,-J-
Djavax.net.ssl.trustStorePassword=changeit
```

will ensure that the certificates of `/tmp/mycerts` are baked into the native binary and used **in addition** to the default cacerts.

The file containing the custom TrustStore does **not** have to be present at runtime as its content has been baked into the native binary.

Working with containers

No special action needs to be taken when running the native binary in a container. If the native binary was properly built with the custom TrustStore as described in the previous section, it will work properly in container as well.

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.

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