Apache Tomcat, your webapp and the Graal

Using Graal native images with Apache Tomcat ™

Rémy Maucherat

Software engineer at Red Hat

Working on Red Hat JBoss Web Server

Apache Tomcat committer since 2000

ASF member

Overview

The Graal project

Project developed by Oracle

Multi language JVM

Allows compiling to self contained native executables

https://www.graalvm.org/

The Damsel of the Sanct Grael by Dante Gabriel Rossetti (1874) (public domain)

Fitting Tomcat inside a native image

Tomcat standalone multiple JARs would be too difficult to compile

Webapps cannot load classes from /WEB-INF/lib and /WEB-INF/classes

No problem for most configuration files access, logging, static files of webapps

Multiple webapps, multiple hosts, TLS enabled configs are all possible in theory

Best is to use the tomcat-maven Maven based fat JAR packaging

Preparing webapps

Use webapp packaging Ant script from tomcat-maven to:

- Precompile JSPs
- Add tomcat-web.xml to webapp for JSP Servlets declaration
- Copy all Servlet sources and properties files to src/main

Add dependencies to tomcat-maven pom.xml as needed

Use Maven to build the fat JAR with Tomcat and all webapp classes

Process

So … How does it work ?

Demo based on the Tomcat examples webapp

Includes Servlets, JSP, EL, websockets, tomcat-native OpenSSL

Step by step demo along the way

At the end, have a native image with the examples

Tomcat documentation page with the full instructions

1
Using tomcat-maven

What’s there

Dockerfile: Container packaging

DockerfileGraal: Container packaging with a native image

graal-webapp.ant.xml: Ant script for webapp packaging

pom.xml: Uses Maven shade plugin to package Tomcat, webapp classes and dependencies

tomcat.yaml: Container deployment plan with port declarations

tomcat-*.json: Base Tomcat Graal native-image descriptors

Similar to Tomcat standalone

bin folder: Default tomcat-native location

conf folder: Regular Tomcat configuration files and resources, server.xml, web.xml, certificates, logging config, etc

src/main: Additional Java sources and resources that can be compiled in the JAR

webapps: Your webapps

Work: Tomcat work folder

2
Examples webapp packaging

Adding the examples webapp

Add JSTL as dependencies in the main pom

Precompile JSPs and add JSPs and Servlets sources from /WEB-INF/classes to the Maven src folder

Build the shaded tomcat-maven JAR

All Tomcat and example webapp classes and dependencies are now in a shaded JAR

3
Graal reflection setup

Descriptors for the webapps

Start substrate VM with the tracing agent

Access Servlets, JSPs, websockets and generally any path that would use dynamic class loading …

… Or force load on startup for Servlets and JSPs

Manual configuration for missing items

Add resource bundles manually

Interfaces may need to be manually added as well

Anything that fails to get picked up by tracing

4
Native image creation

Run native-image

Many errors may occur at this step

Incompatible libraries is a possibility

Code may need to be fixed

Go back to step 3 when it fails

5
Testing

Run the executable

Even more errors may occur at this step

It might crash too ...

Go back to step 3 when it fails

In a container

6
Building container image

Container image changes

Base image does not need Java, so no openjdk:8-jre-alpine

Busybox works better than Alpine

Needs to use the native image to replace the shaded JAR

No Jolokia or Prometheus for monitoring

Problems

TLS support is unavailable as static native images cannot use it

- SunEC needed for JSSE TLS is a dynamic library
- tomcat-native has dependencies on APR and OpenSSL
- Graal static linking only includes core static libraries

Everything needs to be built for the target platform

Incomplete static linking capability in Graal: https://github.com/oracle/graal/issues/827

Packaging

Use the same base image as the host OS to avoid TLS issues

Add OpenSSL (if needed), APR, SunEC () and tomcat-native libraries to the container image

Use the Graal Dockerfile to package the native image, configuration and webapps

Test using Docker first …

7
Deploying to cloud and testing

Using a Kubernetes distribution

Start local OpenShift cluster

Deploy the image to the local OpenShift

Test again

Problems

No session replication due to missing Java serialization

Java serialization support: https://github.com/oracle/graal/issues/460

Conclusion

Impressive benefits

More than 10x faster startup

Very low initial memory use

On demand services based on Tomcat are possible

Works well as a small container image

Still looks like a “normal” Tomcat

However

Lots of setup work and testing required

Tracing agent needs additional capabilities and improved ease of use

Many libraries may be Graal native image incompatible

Some useful features are removed or harder to use, such as JMX, TLS or serialization

Not beneficial for large high load servers