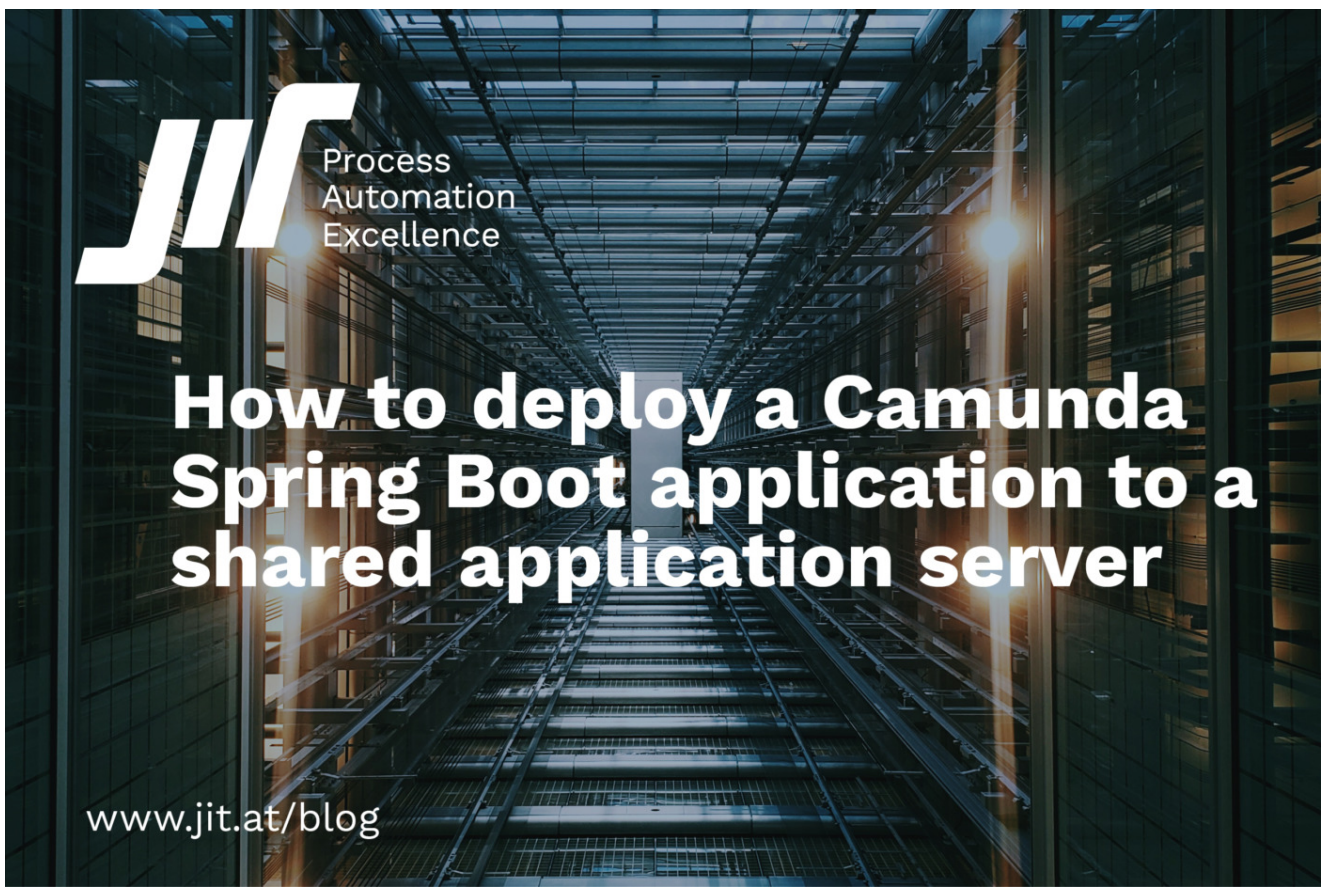
Blog



02.12.2021. | **Author:** Maximilian Kamenicky

How to deploy a Camunda Spring Boot application to a shared application server

Context

Do you still remember our Blog: How to deploy a Camunda Spring Boot application to an external application server?

In that blog, my colleague Gerardo Manzano Garcia explained the steps necessary to run a spring boot application on a dedicated application server i.e. an external Tomcat. In other words, run the spring boot application as a deployment inside an application server.

More Blog articles

04.04.2022.

Maler oder Macher sein

Author: Mirko Pawlak

Digitale Transformation ohne Grenzen. Ich bin Prozessmacher. Ich hocke tief drinnen im Maschinenraum und schau mir an, was das Unternehmen einzigartig macht. Ich tu das gern, denn jeder Prozess ist lebendig und einzigartig. Es gibt einen Anfang, dann passiert etwas und am Ende ist es vo Wie geht man am bes t seinen Prozessen um? […]

configuration.

Why would you need it?

As mentioned in the other blog, using a spring boot application means that the configuration of the infrastructure is taken away from the developer to make development easier and faster. If we use the approach mentioned in the previous blog, we are able to deploy the application to a production server, but lose the benefit of starting a local server conveniently out of our IDE.

Problem

If we simply adjust the scope of our camunda-bpm-spring-boot-starter-webapp-ee and camunda-bpm-spring-boot-starter-rest dependencies, the local spring boot application will miss the necessary dependencies during startup.

Solution

To overcome this, we simplified the previous solution and added profiles loading the appropriate dependencies for each use case. Let us start from scratch again:

To recreate the example please create a new camunda spring boot application using the camunda maven archetype. (Maven Project Templates (Archetypes) | docs.camunda.org)

After your IDE has finished creating the maven project we are free to do all the modifications we need.

Adjust project's maven configuration in pom.xml

- Add <packaging>war</packaging>, below the version tag, or change it from jar to war if it exists

```
<artifactId>camunda-spring-boot-example</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>war</packaging>
```

- Add 2 maven profiles

```
<profiles>
 <profile>
  <id>local-dev</id>
 </profile>

 <profile>
  <id>default</id>
</profile> 9</profiles>
```

Ways to integrate Kafka with Databases

Author: Denis Savenko

Kafka is a great instrument. It is widely used in event-driven architectures, and it does the job perfectly fine. However, this is not the only use-case for it. Kafka is also very well suitable for data replication and ingestion use-cases, as well as for building modern ETL or ELT pipelines. The most noteworthy reasons for [...]

→ **read more**

10.09.2021.

BPMN Error Handling Mechanism – Dealing with many errors in a process

Author: JIT

Errors are inherently part of software development. They can happen because of several reasons. One common definition of a software error is a mismatch between the program and its specification. In other words, we can say, a program has an error when the program does not do what the end-user expects. In this blog post, [...]

→ **read more**

```
<profile>
 <id>local-dev</id>
 <dependencies>
  <dependency>
   <groupId>org.camunda.bpm.springboot</groupId>
   <artifactId>camunda-bpm-spring-boot-starter-webapp-
  ee</artifactId>
  </dependency>
  <dependency>
  <groupId>org.camunda.bpm.springboot</groupId>
  <artifactId>camunda-bpm-spring-boot-starter-
rest</artifactId>
  </dependency>
 </dependencies>
</profile>
```

- Copy the same dependencies to the default profile, but modify the scope to “**provided**”

```
<profile>
 <id>default</id>
 <dependencies>
  <dependency>
   <groupId>org.camunda.bpm.springboot</groupId>
   <artifactId>camunda-bpm-spring-boot-starter-webapp-
    ee</artifactId>
   <scope>provided</scope>
  </dependency>
  <dependency>
   <groupId>org.camunda.bpm.springboot</groupId>
   <artifactId>camunda-bpm-spring-boot-starter-
rest</artifactId>
   <scope>provided</scope>
  </dependency>
 </dependencies>
</profile>
```

- Add the dependency camunda-engine to both profiles, again providing it for the default profile. This tells the application that the process engine is provided and does not need to be bundled with the application:

```
<profiles>
 <profile>
  <id>local-dev</id>
  <dependencies>
   <dependency...>
   <dependency...>
   <dependency>
    <groupId>org.camunda.bpm</groupId>
    <artifactId>camunda-engine</artifactId>
    </dependency>
  </dependencies>
 </profile>

 <profile>
```

```
   <dependency>
    <groupId>org.camunda.bpm</groupId>
    <artifactId>camunda-engine</artifactId>
    <scope>provided</scope>
   </dependency>
 </dependencies>
 </profile>
</profiles>
```

- In the section build, add the following plugin to create a war artefact. This plugin will take care of the war packaging.

```
<plugin>
 <artifactId>maven-war-plugin</artifactId>
 <version>3.3.1</version>
 <configuration>
  <failOnMissingWebXml>false</failOnMissingWebXml>
 </configuration>
</plugin>
```

- In the section build, delete (if present) the following dependency, that does the packaging for Spring Boot, so that, there are no conflicts between this dependency and the one in step 6

```
<plugin>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-maven-plugin</artifactId>
 <version>${springBoot.version}</version>
 <configuration>
  <layout>ZIP</layout>
 </configuration>
 <executions>
  <execution>
   <goals>
    <goal>repackage</goal>
   </goals>
  </execution>
 </executions>
</plugin>
```

- After this step, we are already able to build the dependencies depending on the maven profile we set in the build command. Your pom.xml should look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>

 <groupId>at.jit</groupId>
 <artifactId>spring-boot-tomcat-demo</artifactId>
 <version>1.0-SNAPSHOT</version>
 <packaging>war</packaging>

 <name>Camunda Spring Boot Application</name>
 <description>Spring Boot Application using [Camunda]
(http://docs.camunda.org). [The project has been generated
by the Maven archetype 'camunda-archetype-spring-boot-
7.16.0']</description>

 <properties>
```

```
 <camunda.version>7.16.0-ee</camunda.version>
 Make sure you also switch to the ee webapp dependency
 and EE repository below
 -->
 <springBoot.version>2.5.4</springBoot.version>

 <maven.compiler.source>1.8</maven.compiler.source>
 <maven.compiler.target>1.8</maven.compiler.target>
 <version.java>1.8</version.java>

 <project.build.sourceEncoding>UTF-
 </project.build.sourceEncoding>
 <failOnMissingWebXml>false</failOnMissingWebXml>
</properties>

<dependencyManagement>
 <dependencies>
  <dependency>
   <groupId>org.camunda.bpm</groupId>
   <artifactId>camunda-bom</artifactId>
   <version>${camunda.version}</version>
   <scope>import</scope>
   <type>pom</type>
  </dependency>
  <dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-dependencies</artifactId>
   <version>${springBoot.version}</version>
   <type>pom</type>
   <scope>import</scope>
  </dependency>
 </dependencies>
</dependencyManagement>

<dependencies>
 <dependency>
  <groupId>org.camunda.bpm.springboot</groupId>
  <artifactId>camunda-bpm-spring-boot-starter-
test</artifactId>
   <scope>test</scope>
  </dependency>

  <dependency>
   <groupId>com.h2database</groupId>
   <artifactId>h2</artifactId>
  </dependency>

  <!-- required to use H2 as a file based database
(otherwise it's in-memory) -->
  <dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-jdbc</artifactId>
  </dependency>

  <!-- Required to use Spin dataformat support -->
  <dependency>
   <groupId>org.camunda.spin</groupId>
   <artifactId>camunda-spin-dataformat-all</artifactId>
```

```
 </dependency>

 <!-- Used to generate test coverage reports, see
https://github.com/camunda/camunda-bpm-process-test-
coverage -->
  <dependency>
   <groupId>org.camunda.bpm.extension</groupId>
   <artifactId>camunda-bpm-process-test-
coverage</artifactId>
   <version>0.4.0</version>
   <scope>test</scope>
  </dependency>

  <dependency>
   <groupId>org.camunda.bpm.extension</groupId>
   <artifactId>camunda-bpm-assert-scenario</artifactId>
   <version>1.0.0</version>
   <scope>test</scope>
  </dependency>

  <!-- java util logging => slf4j -->
  <dependency>
   <groupId>org.slf4j</groupId>
   <artifactId>jul-to-slf4j</artifactId>
   <scope>test</scope>
  </dependency>

  <!-- Add your own dependencies here, if in compile
scope, they are added to the jar -->

  <!-- JAXB -->
  <dependency>
   <groupId>javax.xml.bind</groupId>
   <artifactId>jaxb-api</artifactId>
   <scope>runtime</scope>
  </dependency>

  </dependencies>


  <!-- Add the necessary profiles here. local-dev includes
the camunda dependencies, default expects them to be
provided -->
  <profiles>
   <profile>
    <id>local-dev</id>
    <dependencies>
     <dependency>
      <groupId>org.camunda.bpm.springboot</groupId>
      <artifactId>camunda-bpm-spring-boot-starter-webapp-
      ee</artifactId>
     </dependency>
     <dependency>
      <groupId>org.camunda.bpm.springboot</groupId>
      <artifactId>camunda-bpm-spring-boot-starter-
      rest</artifactId>
     </dependency>
     <dependency>
      <groupId>org.camunda.bpm</groupId>
```

```
<profile>
 <id>default</id>
 <dependencies>
  <dependency>
   <groupId>org.camunda.bpm.springboot</groupId>
   <artifactId>camunda-bpm-spring-boot-starter-webapp-
   ee</artifactId>
   <scope>provided</scope>
  </dependency>
  <dependency>
   <groupId>org.camunda.bpm.springboot</groupId>
   <artifactId>camunda-bpm-spring-boot-starter-
   rest</artifactId>
   <scope>provided</scope>
  </dependency>
    <dependency>
     <groupId>org.camunda.bpm</groupId>
     <artifactId>camunda-engine</artifactId>
     <scope>provided</scope>
    </dependency>
   </dependencies>
  </profile>
 </profiles>

 <repositories>
  <repository>
   <id>camunda-bpm-nexus</id>
   <name>Camunda Maven Repository</name>
   <url>https://app.camunda.com/nexus/content/groups/
   public</url>
  </repository>
  <repository>
   <id>camunda-bpm-nexus-ee</id>
   <name>Camunda Enterprise Maven Repository</name>

<url>https://app.camunda.com/nexus/content/repositories/
   camunda-bpm-ee</url>
  </repository>
</repositories>

<build>
 <finalName>${project.artifactId}</finalName>
 <plugins>
  <plugin>
   <artifactId>maven-war-plugin</artifactId>
   <version>3.3.1</version>
   <configuration>
    <failOnMissingWebXml>false</failOnMissingWebXml>
   </configuration>
  </plugin>
  <plugin>
   <groupId>org.codehaus.mojo</groupId>
   <artifactId>exec-maven-plugin</artifactId>
   <version>1.6.0</version>
   <configuration>
    <mainClass>at.jit.CamundaApplication</mainClass>
   </configuration>
```

Change the application code configuration

- Enable your application as a servlet by extending your main class with SpringBootServletInitializer and overriding the configuration method in the main class. This tells the application from where to pick the configured process and beans:

```
@SpringBootApplication
@EnableProcessApplication("spring-boot-tomcat-demo")
public class CamundaApplication extends
SpringBootServletInitializer {
 private static final Class <CamundaApplication>
 CAMUNDA_APPLICATION_CLASS = CamundaApplication.class;

 public static void main(String... args) {
  SpringApplication.run(CamundaApplication.class, args);
 }

 @Override
 protected SpringApplicationBuilder
 configure(SpringApplicationBuilder application){
  return application.sources(CAMUNDA_APPLICATION_CLASS);
 }
}
```

- Create a class that is used to configure and bootstrap the application context. Put the class in the same folder as the class with the main method. The purpose of this class is, to tell the application that it is deployed on an external application server, or so-called, shared process engine, and to tell the application to load our beans that are evaluated as expressions in our services tasks.

```
@Configuration
@ComponentScan(basePackages = "at.jit")
@Profile("default")
public class CamundaApplicationContext{

 @Bean
 public ProcessEngineService processEngineService() {
  return BpmPlatform.getProcessEngineService();
 }

 @Bean(destroyMethod = "")
 public ProcessEngine processEngine(){
  return BpmPlatform.getDefaultProcessEngine();
 }

 @Bean
 public SpringProcessApplication processApplication()
 {
  return new SpringProcessApplication();
 }
```

```
processEngine) {
  return processEngine.getRepositoryService();
 }

 @Bean
 public RuntimeService runtimeService(ProcessEngine
processEngine) {
  return processEngine.getRuntimeService();
 }

 @Bean
 public TaskService taskService(ProcessEngine
processEngine) {
  return processEngine.getTaskService();
 }

 @Bean
 public HistoryService historyService(ProcessEngine
processEngine) {
  return processEngine.getHistoryService();
 }

 @Bean
 public ManagementService managementService(ProcessEngine
processEngine) {
  return processEngine.getManagementService();
 }
}
```

Adapt tests

As the spring boot tests already created by the archetype expect a working spring boot configuration to be executed, we have to annotate these tests to use the local-dev profile. This can be done as follows:

```
@ActiveProfiles(profiles = {"local-dev"})
public class ProcessScenarioTest {
```

- Please note that by default creation of the archetype, there are two spring boot tests that need to be annotated: ProcessScenarioTest.java & ProcessUnitTest.java

Create a process.xml file

Inside resources/META-INF create a file named processes.xml if it doesn't already exist. This file contains the deployment metadata for the application. The content of the file can be taken from here, [The processes.xml Deployment Descriptor | docs.camunda.org]

```
<process-application
xmlns="http://www.camunda.org/schema/1.0/ProcessApplicatio
n" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

```
    <property
name="isScanForProcessDefinitions">true</property>
  </properties>
 </process-archive>
</process-application>
```

- Without this file, the configuration class created before, cannot bind everything together. Failing to do so, will give us the false impression that everything is ok. However, problems like processes that do not start, or suddenly stop for no reason, and similar ones, are very likely to appear.

- If you need some local development configuration, it is recommended to create an application-local-dev.yaml in your resources folder as well.

```
 spring.datasource:
  url: jdbc:h2:./camunda-
db;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE
 #shareable h2 database: jdbc:h2:./camunda-
db;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE;AUTO_SERVER=TR
UE
  username: sa
  password: sa
 spring.h2.console.enabled: true
 camunda.bpm:
  admin-user:
   id: demo
   password: demo
   firstName: Demo
   lastName: Demo
  filter:
   create: All Tasks
 # default-serialization-format: application/json
server.port: 8080
```

location where the war file has been built. In IntelliJ that location can be seen in the console:

```
[INFO] Building war: …\target\CamundaApplication.war
[INFO] ————————————————————————
[INFO] BUILD SUCCESS
```

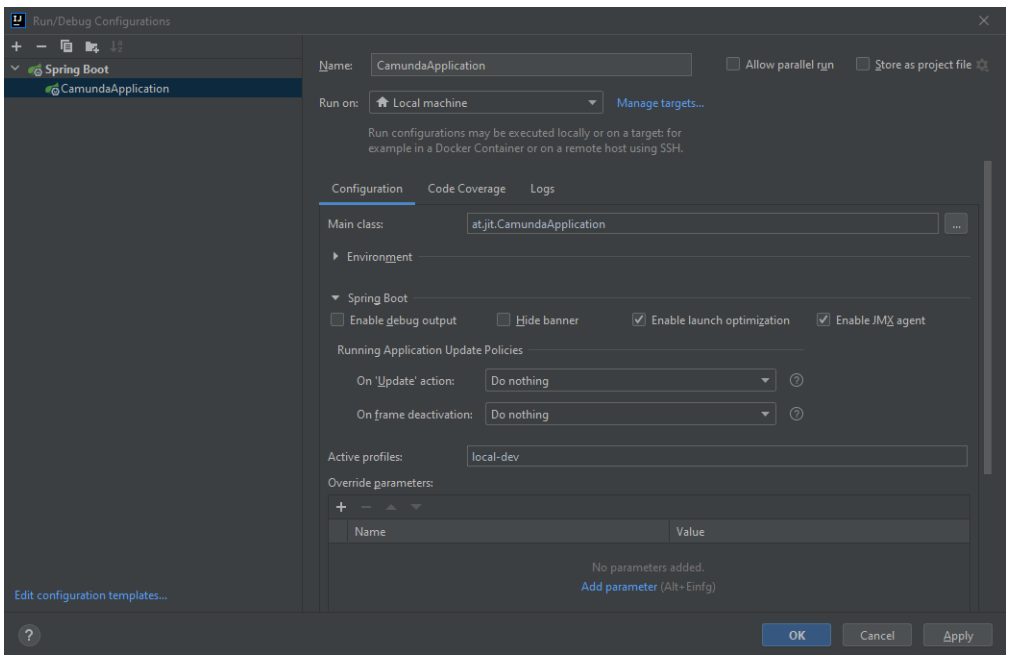
- Put the war file in the webapps folder of the application server, and check that Tomcat loads two things, spring and the BPMN processes.
- Optionally, wrap all these steps inside a Maven profile, for building the project comfortably.

Start the application locally

- Change the maven profile to local-dev
- Using maven, perform a “clean and install”. The build should finish successfully.

```
[INFO] BUILD SUCCESS
```

- Create a spring boot run configuration and set the active spring profile there as well



The server should start up successfully.

```
2021-11-30 17:19:30.947 INFO 22532 --- [ main]
org.camunda.bpm.application : ENGINE-07021
```

```
  spring-boot-tomcat-demo[version: 1, id: spring-boot-
tomcat-demo:1:26]
  spring-boot-tomcat-demo[version: 2, id: spring-boot-
tomcat-demo:2:103]
  spring-boot-tomcat-demo[version: 3, id: spring-boot-
tomcat-demo:3:203]
Deployment does not provide any case definitions.
2021-11-30 17:19:30.951 INFO 22532 --- [ main]
org.camunda.bpm.container : ENGINE-08050 Process
application spring-boot-tomcat-demo successfully deployed
2021-11-30 17:19:30.954 INFO 22532 --- [ main]
at.jit.CamundaApplication : Started CamundaApplication in
6.078 seconds (JVM running for 7.086)
2021-11-30 17:19:30.957 INFO 22532 --- [ main]
org.camunda.bpm.engine.jobexecutor : ENGINE-14014 Starting
up the
JobExecutor[org.camunda.bpm.engine.spring.components.jobex
ecutor.SpringJobExecutor].
2021-11-30 17:19:30.958 INFO 22532 --- [ingJobExecutor]]
org.camunda.bpm.engine.jobexecutor : ENGINE-14018
JobExecutor[org.camunda.bpm.engine.spring.components.jobex
ecutor.SpringJobExecutor] starting to acquire jobs
```

Summary

If you compare the old blog to this one, there are only a few slight changes in the approach of how to do it. The goal was to make it more convenient to use and to allow developers to still access all the features regardless of the environment.

Links to resources to understand more

Using Shared Process Engine | docs.camunda.org
Maven – Introduction to build profiles
Process applications | docs.camunda.org
How to Deploy a Spring Boot Application on Tomcat as a WAR Package [Intermediate Spring Boot]

More about JIT and our services: JIT Services

Written by Maximilian Kamenicky
Developer at JIT IT-Dienstleistungs Gesmbh

Photo by Ian Battaglia on Unsplash

SOLUTIONS

About Solutions
By Roles

WHY JIT

SERVICES

Licences
Services

TRAININGS

About JIT Trainings
BPMN Modelling Training
Camunda Developer Trainings

ABOUT US

Our History
Values
Certified Experts
Career
News
Blog
Contact us

PARTNER PRODUCTS

Process Mining by Celonis
Workflow Automation by Camunda
Robotic Process Automation by IBM AA

JIT PRODUCTS

Camunda Visualizer for Celonis
BPMN Diff
Microservices
Database Development

CLIENTS

"J-IT" IT-DIENSTLEISTUNGS GESMBH Vorgartenstraße 206B, A-1020 Vienna office@jit.at