

Wildfly booteable

Un servidor embebido basado en wildfly que permita desplegar nuestra aplicación lo mas cercano al modo de despliegue en el servidor de RSI

Aplicacion + servidor en un jar booteable

“

Como última alternativa a evaluar en el tema de conexiones, datasource, transaccionalidad y que además tenga en cuenta la auditoria y que nos facilite configurar un proyecto backend tenemos la opción de un plugin para Maven llamado wildfly jar maven plugin, en esencia lo que nos permite este plugin es generar una instalación temporal de un servidor de aplicaciones wildfly sobre el cual se despliega el war de nuestro proyecto, este servidor es similar al que se llega a obtener si lo instalamos directamente en nuestra maquina o como esta en el servidor, sin embargo tiene algunas limitantes como : No tener un perfil de administrador, en caso de apagar el servidor se genera de nuevo la instalación temporal y la ejecución de la configuración, lo que se haga en tiempo de ejecución con la cli se pierde para el siguiente arranque.

El jar booteable que genera en la carpeta target es un archivo ejecutable desde java, podemos desde nuestro ide generar la configuración o con la terminal ejecutar el comando:

```
java -jar target/myapp-bootable.jar
```

Al ejecutar dicho comando, se instala el servidor temporal, se agregan las configuraciones y por ultimo se despliega el war de nuestra aplicación para que el servidor quede activo.

“

Por defecto no trae ninguna configuración de datasource como si esta configurado actualmente en el servidor de aplicaciones de RSI, y para ello hay mínimo dos opciones viables, la primera es usarlo simplemente como servidor aunque no habría mucha diferencia a manejar el que trae spring boot por defecto que es tomcat, pues de igual forma es necesario manejar las transacciones a nivel del código para que funcione auditoria y tengamos un proyecto estable, o la segunda es generando una configuración del servidor wildfly al mismo tiempo en el que este se crea.

Configuración a nivel de pom.xml

Para el ejemplo vamos a mirar la configuración del pom para dicho plugin.

```
1  <plugin>
2    <groupId>org.wildfly.plugins</groupId>
3    <artifactId>wildfly-jar-maven-plugin</artifactId>
4    <version>7.0.0.Final</version>
5    <configuration>
6      <cli-sessions>
7        <cli-session>
8          <script-files>
9            <script>datasource.cli</script>
10         </script-files>
11         <resolve-expressions>true</resolve-expressions>
12       </cli-session>
13     </cli-sessions>
14     <feature-packs>
15       <feature-pack>
16         <location>wildfly@maven(org.jboss.universe:community-universe)#20.0.0</location>
17       </feature-pack>
18       <feature-pack>
19         <groupId>org.wildfly</groupId>
20         <artifactId>wildfly-datasources-galleon-pack</artifactId>
21         <version>1.2.2.Final</version>
22       </feature-pack>
23     </feature-packs>
24     <layers>
25       <layer>jaxrs-server</layer>
26       <layer>oracle-driver</layer>
27     </layers>
28   </configuration>
29   <executions>
30     <execution>
31       <goals>
32         <goal>package</goal>
33       </goals>
34     </execution>
35   </executions>
36 </plugin>
```

Partimos del groupId y el artifactId que identifican nuestra dependencia así como también su versión, ahora es necesario especificar la configuración que queremos tenga nuestro servidor embebido.

Configuración datasource

Lo primero es mediante una configuración por medio de cli ejecutar un script que configura el datasource y garantiza la transaccionalidad y el correcto funcionamiento de auditoría, dicho script está en la raíz del proyecto y su contenido es el siguiente:

```

1  #Script para crear la configuración de un datasource en tiempo de ejecucion de
2
3  data-source add --name=myGreatDS --jndi-name=java:jboss/datasources/myGreatDS
4  --connection-url=jdbc:oracle:thin:@10.0.30.10:1521:uisdb01 --user-name=develop
5  --check-valid-connection-sql="SELECT 1 FROM DUAL" \
6  --background-validation=true \
7  --background-validation-millis=60000 \
8  --flush-strategy=IdleConnections \
9  --min-pool-size=4 --max-pool-size=16 --pool-prefill=true \
10 --idle-timeout-minutes=1 \

```

Como wilfly tiene una cli para manejar el servidor pues nos aprovechamos de ella y definimos la configuración del datasource que nos interesa, esto garantiza que al crearse el servidor temporal también se crea la configuración que necesita nuestro proyecto.

Luego de la configuración del datasource tenemos un feature-pack que nos permite definir características del servidor que vamos a instalar, la primera de ella es la versión de wildfly que vamos a manejar, en este caso la 20.0.1.Final y la segunda nos define un conjunto de configuraciones para el datasource sobre el cual estamos interesados, las funcionalidades galleon son aquellas configuraciones que posee nuestro servidor y en este caso nos interesa contar con la funcionalidad para datasource entonces la declaramos en la dependencia para que sea tomada en cuenta.

Por último, tenemos las layers donde especificamos el driver de Oracle que necesita nuestro datasource para poder conectarse y administrar las conexiones a base de datos y también tenemos la jaxrs-server la cual es otra extensión que hace parte de datasources-web-server y soporta JPA entre otros.

la parte de exclusiones donde podemos del paquete general excluir lo que no nos interesa, funcionalidades que no utilizaremos y de esta manera liberamos peso y se produce una ejecución más rápida de la aplicación.

De esta manera al ejecutar la aplicación obtenemos lo siguiente:

```

C:\Users\totto\.jdk\corretto-11.0.13-1\bin\java.exe -Dfile.encoding=windows-1252 -jar C:\Users\totto\Desktop\ap-backend\target\admonpersonal-bootable.jar
11:27:28,525 INFO [org.wildfly.jar] (main) WFLYJAR0007: Installed server and application in C:\Users\totto\AppData\Local\Temp\wildfly-bootable-server527456476365233382, took 2517ms

```

Lo primero que hace el plugin es generar el servidor en la carpeta temporal Y **luego** procede a generar la configuración, entonces si accedemos al archivo standalone.xml que contiene la configuración del servidor podemos ver tanto la configuración del datasource como del driver requerido.

```

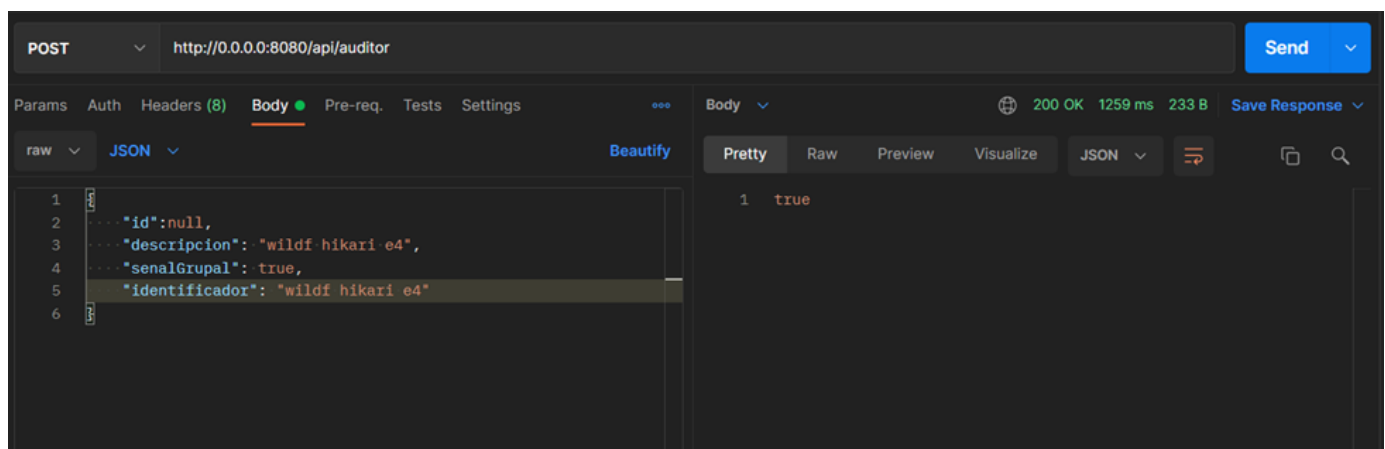
1  <datasource jndi-name="java:jboss/datasources/myGreatDS" pool-name="myGreatDS"

```

```
2      <connection-url>jdbc:oracle:thin:@10.0.30.10:1521:uisdb01</connection-url>
3      <driver>oracle</driver>
4      <pool>
5          <min-pool-size>4</min-pool-size>
6          <max-pool-size>16</max-pool-size>
7          <prefill>true</prefill>
8          <flush-strategy>AllIdleConnections</flush-strategy>
9      </pool>
10     <security>
11         <user-name>develop</user-name>
12         <password>d3v_3lop$</password>
13     </security>
14     <validation>
15         <valid-connection-checker class-name="org.jboss.jca.adapters.jdbc.extensions
16         <check-valid-connection-sql>SELECT 1 FROM DUAL</check-valid-connection-sql>
17         <validate-on-match>true</validate-on-match>
18     </validation>
19     <timeout>
20         <idle-timeout-minutes>1</idle-timeout-minutes>
21     </timeout>
22     <statement>
23         <track-statements>TRUE</track-statements>
24     </statement>
25 </datasource>
```

Test de la configuración

Y **por último** probando auditoria vemos como todo está funcionando:



IPCIÓN	SENAL_GRUPAL	IDENTIFICADOR	VERSIONS_XID	VERSIONS_OPERATION	VERSIONS_STARTT
1 FORE EMBEBIDO	0	TBEF	00 00000000 00 03 00 12 00 05 22 D6	I	2022-03-16 09:46:4
2 ikari e2	1	wildf hikari e2 diferente	00 00000000 00 21 00 1A 00 00 EB E1	D	2022-03-16 10:13:5
3 ikari e2	1	wildf hikari e2 diferente	00 00000000 00 25 00 1C 00 03 97 44	I	2022-03-16 10:13:5
4 ikari e2	1	wildf hikari e2	00 00000000 00 25 00 1C 00 03 97 44	I	2022-03-16 10:13:5
5 ikari e3	1	wildf hikari e3 diferente	00 00000000 00 20 00 1A 00 01 39 7D	D	2022-03-16 11:02:3
6 ikari e3	1	wildf hikari e3 diferente	00 00000000 00 07 00 18 00 05 04 D8	I	2022-03-16 11:02:3
7 ikari e3	1	wildf hikari e3	00 00000000 00 07 00 18 00 05 04 D8	I	2022-03-16 11:02:3
8 ikari e4	1	wildf hikari e4 diferente	00 00000000 00 02 00 03 00 05 2E 40	D	2022-03-16 11:46:2
9 ikari e4	1	wildf hikari e4 diferente	00 00000000 00 23 00 0D 00 00 EB 7C	I	2022-03-16 11:46:2
10 ikari e4	1	wildf hikari e4	00 00000000 00 23 00 0D 00 00 EB 7C	I	2022-03-16 11:46:2

“

Ahora vamos a hacer una pequeña prueba de carga y ver los resultados que obtiene dicho servidor con la configuración establecida.

Test run on 16/03/2022 11:37:55 a. m.

**** Project and Scenario Comments, Operator ****

Results of period #1 (from 1 sec to 15 sec):

Completed Clicks: 1000 with 0 Errors (=0,00%)

Average Click Time for 1.000 Users: 1.996 ms

Successful clicks per Second: 70,16 (equals 252.567,29 Clicks per Hour)

Results of complete test

**** Results per URL for complete test ****

URL #1 (tes1): Average Click Time 1.996 ms, 1.000 Clicks, 0 Errors

Total Number of Clicks: 1.000 (0 Errors)

Average Click Time of all URLs: 1.996 ms

“

Vemos como con 1000 peticiones simultaneas se comporta bien y ahora la prueba final con 4000 para testear no solo el servidor sino también la configuración del datasource.

Test run on 16/03/2022 11:39:01 a. m.

**** Project and Scenario Comments, Operator ****

Results of period #1 (from 3 sec to 20 sec):

Completed Clicks: 1974 with 0 Errors (=0,00%)

Average Click Time for 4.000 Users: 2.202 ms

Successful clicks per Second: 117,75 (equals 423.915,31 Clicks per Hour)

Results of period #2 (from 20 sec to 34 sec):

Completed Clicks: 1170 with 0 Errors (=0,00%)

Average Click Time for 4.000 Users: 834 ms

Successful clicks per Second: 79,31 (equals 285.527,31 Clicks per Hour)

Results of period #3 (from 34 sec to 48 sec):

Completed Clicks: 856 with 0 Errors (=0,00%)

Average Click Time for 4.000 Users: 668 ms

Successful clicks per Second: 60,70 (equals 218.515,12 Clicks per Hour)

Results of complete test

**** Results per URL for complete test ****

URL #1 (tes1): Average Click Time 1.474 ms, 4.000 Clicks, 0 Errors

Total Number of Clicks: 4.000 (0 Errors)

Average Click Time of all URLs: 1.474 ms

“

Como se puede apreciar la configuración es buena, no se obtienen errores y se logra un tiempo bueno.

“

RECALCAMOS que el web stress no es una herramienta para medir el comportamiento, la métrica no es la ideal y no se debe usar como criterio de decisión sobre entornos de producción, pero para ver el funcionamiento nos vale para afirmar que la configuración responde bien bajo carga y bajo auditoria.