**Spring Boot**

------------------------

**Reference**

https://docs.spring.io/spring-boot/docs/current/reference/html/getting-started-first-application.html

**Points to remember**

1. **Uber Jars** - Java does not provide any standard way to load nested jar files (i.e. jar files that are themselves contained within a jar). This can be problematic if you are looking to distribute a self-contained application.

To solve this problem, many developers use “uber” jars. An uber jar simply packages all classes, from all jars, into a single archive. The problem with this approach is that it becomes hard to see which libraries you are actually using in your application. It can also be problematic if the same filename is used (but with different content) in multiple jars.

Some General definition from (http://stackoverflow.com/questions/11947037/what-is-an-uber-jar)

Über is the German word for above or over, as in a line from a previous national anthem: Deutschland, Deutschland, über alles (Germany, Germany above all else).

Hence, in this context, an uber-jar is an "over-jar", one level up from a simple "jar", defined as one that contains both your package *and* all its dependencies in one single JAR file. The name can be thought to come from the same stable as ultrageek, superman, hyperspace, and metadata, which all have similar meanings of "beyond the normal".

The advantage is that you can distribute your uber-jar and not care at all whether or not dependencies are installed at the destination, as your uber-jar actually *has* no dependencies.

All the dependencies of your own stuff within the uber-jar are *also* within that uber-jar.

1. To terminate the executed JAR

* Currently, I followed to kill the port used by that jar by using ,

C:\Users\pkalwar> netstat -o -n -a | findstr 0.0:8080.

TCP 0.0.0.0:8080 0.0.0.0:0 LISTENING 2068

C:\Users\pkalwar>taskkill /F /PID 2068

SUCCESS: The process with PID 2068 has been terminated.

1. If you look in the target directory you should see **myproject-0.0.1-SNAPSHOT.jar**. The file should be around 10 Mb in size. If you want to peek inside, you can use jar tvf:

$ jar tvf target/myproject-0.0.1-SNAPSHOT.jar

You should also see a much smaller file named **myproject-0.0.1-SNAPSHOT.jar.original** in the target directory. This is the original jar file that Maven created before it was repackaged by Spring Boot.

To run that application, use

java -jar <jar Name>

**Introduction to Spring Boot**

**Pre Requisite**

* Basic understanding of Spring MVC
* Understanding of any build tool like Maven/ Gradle

**System Requirements**

* Java 7 & above [ Recommended Java 8]
* Spring Framework 4.15 & above
* For build support " Maven 3.2 + " & "Gradle 1.12+"
* ***Servlet Containers***

Tomcat 8/7 => Servlet 3.1/3.0 => Java 7/6

Jetty 9/8 => Servlet 3.1/3.0 => Java 7/6

* Set environment variable for Java/Maven or Gradle etc.

***Steps before start with your first Spring Boot Application :***

* Install Java and set the same in your environment variable
* Configure Maven
  + Download the Maven binary zip or install the same on your local
  + Set the environment variable for same as "M2\_HOME"
  + Below is the typical ***pom.xml.***

<?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>com.example</groupId>

<artifactId>myproject</artifactId>

<version>0.0.1-SNAPSHOT</version>

*<!-- Inherit defaults from Spring Boot -->*

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.4.0.BUILD-SNAPSHOT</version>

</parent>

*<!-- Add typical dependencies for a web application -->*

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

</dependencies>

*<!-- Package as an executable jar -->*

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

*<!-- Add Spring repositories -->*

*<!-- (you don't need this if you are using a .RELEASE version) -->*

<repositories>

<repository>

<id>spring-snapshots</id>

<url>http://repo.spring.io/snapshot</url>

<snapshots><enabled>true</enabled></snapshots>

</repository>

<repository>

<id>spring-milestones</id>

<url>http://repo.spring.io/milestone</url>

</repository>

</repositories>

<pluginRepositories>

<pluginRepository>

<id>spring-snapshots</id>

<url>http://repo.spring.io/snapshot</url>

</pluginRepository>

<pluginRepository>

<id>spring-milestones</id>

<url>http://repo.spring.io/milestone</url>

</pluginRepository>

</pluginRepositories>

</project>

* Configure Gradle
  + Download the Gradle binary zip or install the same on your local
  + Set the environment variable for same as "GRADLE\_HOME"
  + Below is the typical ***build.gradle.***

Gradle users can directly import “starter POMs” in their dependencies section. Unlike Maven, there is no “super parent” to import to share some configuration.

apply plugin: 'java'

repositories {

maven { url "http://repo.spring.io/snapshot" }

maven { url "http://repo.spring.io/milestone" }

}

dependencies {

compile("org.springframework.boot:spring-boot-starter-web:1.4.0.BUILD-SNAPSHOT")

}

The [***spring-boot-gradle-plugin***](http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#build-tool-plugins-gradle-plugin) is also available and provides tasks to create executable jars and run projects from source. It also provides [*dependency management*](http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#build-tool-plugins-gradle-dependency-management) that, among other capabilities, allows you to omit the version number for any dependencies that are managed by Spring Boot:

buildscript {

repositories {

maven { url "http://repo.spring.io/snapshot" }

maven { url "http://repo.spring.io/milestone" }

}

dependencies {

classpath("org.springframework.boot:spring-boot-gradle-plugin:1.4.0.BUILD-SNAPSHOT")

}

}

apply plugin: 'java'

apply plugin: 'spring-boot'

repositories {

maven { url "http://repo.spring.io/snapshot" }

maven { url "http://repo.spring.io/milestone" }

}

dependencies {

compile("org.springframework.boot:spring-boot-starter-web")

testCompile("org.springframework.boot:spring-boot-starter-test")

}

***Structuring your code***

Some best practice that help.

* **Don’t use default package structure** - Because, It can cause particular problems for Spring Boot applications that use @ComponentScan, @EntityScan or @SpringBootApplication annotations, since every class from every jar, will be read. Always follow ***"com.example.projectname"***
* **Locating your main application class by annotating "***@****EnableAutoConfiguration*" -**

***@EnableAutoConfiguration*** - Place it on your main class, implicitly defines your base "search package" for certain items.

e.g. For JPA application @EnableAutoConfiguration will be used to search @Entity items.

***@SpringBootApplication -***  Alternative of above annotation is this one.

***@ComponentScan -***  This is used to scan / automatically picks up all the components defines in a project