# Preface:

* Spring Applications, complexity increases with addition of more features

**Problems usually faced:**

* Configuration is getting repeated for complex applications.
* Need to add a lot of dependencies to the servlet XML.
* Configuring Spring applications tend to become challenging and error-prone, which includes several steps in set-up and configuration, build and deploy steps.



# Introduction to Spring Boot

Spring Boot allows you to build stand-alone, operating, production-grade Spring based Applications with ease.

The guiding principle of Boot is **convention over configuration.**

Let’s have a look at some of the important features in Boot:

* **starter modules** for simplifying dependency configuration
* **auto-configuration** whenever possible
* **advanced externalized configuration**
* embedded, built-in Tomcat, Jetty or Undertow
* stand-alone Spring applications
* production-ready features such as metrics, health checks, and externalized configuration
* no requirement for XML configuration

# Spring Boot Starters

Simply put, starters are dependency descriptors that reference a list of libraries.

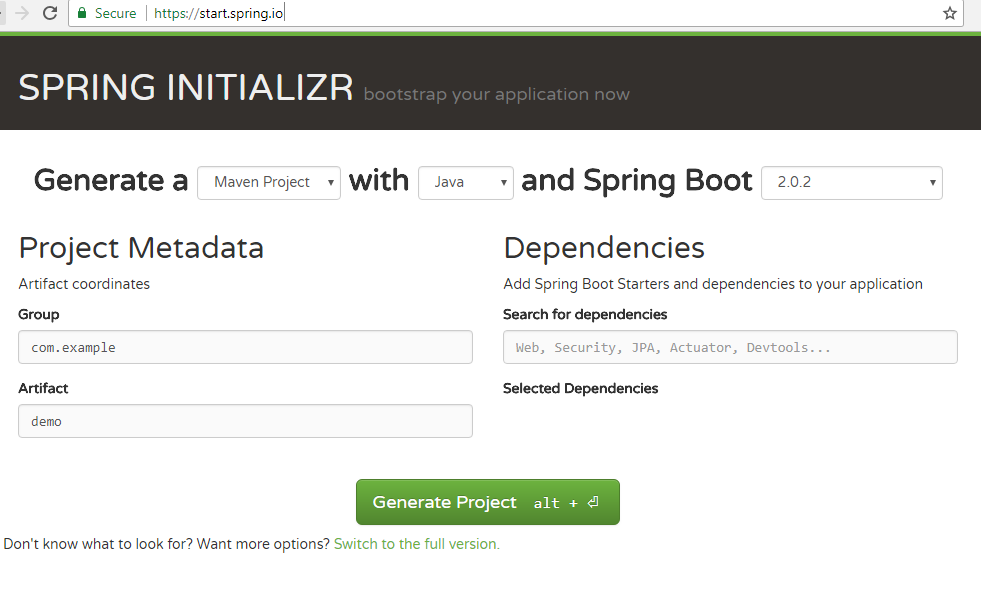
|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.0.2.RELEASE</version>  <relativePath />  </parent> |

* You only need to specify the dependency version once for the parent.
* It eliminates potential errors related to incompatible library versions.
* When you need to update the Boot version, you only need to change a single, central version, and everything else gets implicitly updated.

# **Spring Boot Initializr**

Use the Spring Boot Initializr page to download a pre-configured Spring Boot project, which you can then import into your IDE.

<https://start.spring.io/>



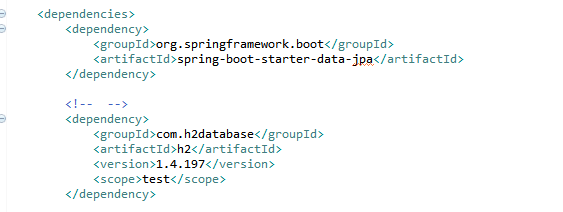
# **Spring Boot Auto-Configuration**

* Spring Boot aims to simplify this process by providing a **sensible default configuration, based on the dependencies on the classpath** and loaded automatically behind the scenes.
* It’s important to understand that, as soon as you define **your configuration beans**, then these **will take precedence over the auto-configured ones**.

**Example**:

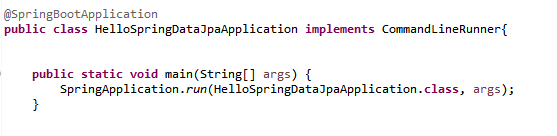
To work with Spring Data JPA, we also need to set up a database.

Luckily, Boot provides auto-configuration for three types of in-memory databases: H2, HSQL, and Apache Derby.



Auto-configures Hibernate as the default JPA provider.

# Entry Point: @SpringBootApplication



This is all we need to have a running Boot application.

* The shortcut **@SpringBootApplication** annotation is equivalent to using **@Configuration, @EnableAutoConfiguration, and @ComponentScan**
* And will pick up all config classes in or below the package where the class is defined.

# Advanced externalized configuration

Configure the behavior of an application via:

**External properties files | YAML files | Environment variables | Command-line arguments.**

These properties have standard names that will be automatically picked up by Boot and evaluated in a set order.

