Sample steps to build Spring Boot Application: ---

System Requirements: -

Spring Boot 2.0.3.RELEASE requires [Java 8 or 9](https://www.java.com/) and [Spring Framework 5.0.7.RELEASE](https://docs.spring.io/spring/docs/5.0.7.RELEASE/spring-framework-reference/) or above. Explicit build support is provided for Maven 3.2+ and Gradle 4.

Spring boot provides Embeded tomcat/Jetty servers.

Spring Boot supports the following embedded servlet containers:

| Name | Servlet Version |
| --- | --- |
| Tomcat 8.5 | 3.1 |
| Jetty 9.4 | 3.1 |

Maven Installation

Spring Boot is compatible with Apache Maven 3.2 or above.

Spring based applications have a lot of configuration.

When we use Spring MVC, we need to configure component scan, dispatcher servlet, a view resolver, web jars (for delivering static content) among other things which requires lot to XML configuration and crearting beans for that.

<bean

class="org.springframework.web. servlet.view.InternalResourceViewResolver">

<property name="prefix">

<value>/WEB-INF/views/</value>

</property>

<property name="suffix">

<value>.jsp</value>

</property>

</bean>

<mvc:resources mapping="/webjars/\*\*" location="/webjars/"/>

Below code snippet shows typical configuration of a dispatcher servlet in a web application.

<servlet>

<servlet-name>dispatcher</servlet-name>

<servlet-class>

org.springframework.web.servlet.DispatcherServlet

</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/todo-servlet.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>dispatcher</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

When we use Hibernate/JPA, we would need to configure a datasource, an entity manager factory, a transaction manager among a host of other things.

<bean id="dataSource" class="com. mchange. v2. c3p0.ComboPooledDataSource"

destroy-method="close">

<property name="driverClass" value="${db.driver}" />

<property name="jdbcUrl" value="${db.url}" />

<property name="user" value="${db.username}" />

<property name="password" value="${db.password}" />

</bean>

<jdbc:initialize-database data-source="dataSource">

<jdbc:script location="classpath:config/schema.sql" />

<jdbc:script location="classpath:config/data.sql" />

</jdbc:initialize-database>

<bean

class="org.springframework.orm.jpa. LocalContainerEntityManagerFactoryBean"

id="entityManagerFactory"

<property name="persistenceUnitName" value="hsql\_pu" />

<property name="dataSource" ref="dataSource" />

</bean>

<bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">

<property name="entityManagerFactory" ref="entityManagerFactory" />

<property name="dataSource" ref="dataSource" />

</bean>

<tx:annotation-driven transaction-manager="transactionManager"/>

Instead of all these XML configuration to follow,spring boot provide Auto-configuration where some beans can be Auto-Configured automatically.and the only thing we have to follow is to add the respective dependencies in Pom.xml file like below:--

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>4.2.2. RELEASE</version>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

<version>2.5.3</version>

</dependency>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-validator</artifactId>

<version>5.0.2. Final</version>

</dependency>

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.17</version></dependency>

Creating Pom.xml file and add dependencies of spring boot and Maven plugins into it like below

All Spring Boot projects typically use spring-boot-starter-parent as the parent in the pom.xml like below: -

<parent>

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

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

<version>2.0.0. RELEASE</version>

</parent>

Spring Boot provides a wide range of starter projects. Spring Initializr suppports all of them and more. Among the varied range of starter projects and options supported are:

spring-boot-starter-web-services: Build applications exposing SOAP web services

spring-boot-starter-web: Build Web applications and RESTful applications

spring-boot-starter-test: Write great unit and integration tests

spring-boot-starter-jdbc: Traditional JDBC applications

spring-boot-starter-security: Authentication and authorization using Spring Security

spring-boot-starter-data-jpa: Spring Data JPA with Hibernate

spring-boot-starter-cache: Enabling Spring Framework’s caching support

spring-boot-starter-data-rest: Expose simple REST services using Spring Data REST

We have several build tools available in the market to build an application: -

Apache Ant

Apache Maven

Gradle

Out of which Apache Maven (using XML) has the ability to download dependencies over the network and focused mostly on dependency management

Apache Ant: --Gradle uses XML. Major drawback was XML as the format to write build scripts. XML, being hierarchical in nature, is not a good fit for procedural programming approach Ant uses. Another problem with Ant is that its XML tends to become unmanageably big when used with all but very small projects.

As dependency management over the network became a must, Ant adopted [Apache Ivy](http://ant.apache.org/ivy/) which became popular as Apache Maven.

Gradle: -- Gradle does not use XML. Instead, it had its own DSL based on Groovy (one of JVM languages). As a result, Gradle build scripts tend to be much shorter and clearer than those written for Ant or Maven. The amount of boilerplate code is much smaller with Gradle since its DSL is designed to solve a specific problem.

Advantages of springboot:--

1. Simplified & version conflict free dependency management through the starter POMs.
2. We can quickly setup and run standalone, web applications and micro services at very less time.
3. You can just assemble the jar artifact which comes with an embedded Tomact, Jetty or Undertow application server and you are ready to go.
4. Spring Boot provides HTTP endpoints to access application internals like detailed metrics, application inner working, health status, etc.
5. No XML based configurations at all. Very much simplified properties. The beans are initialized, configured and wired automatically.
6. The Spring Initializer provides a project generator to make you productive with the certain technology stack from the beginning.
7. You can create a skeleton project with web, data access (relational and NoSQL datastores), cloud, or messaging support.

To instantiate the beans automatically we will use the below annotation

@EnableAutoConfiguration:- It tells Spring Boot to start adding beans based on classpath settings, other beans, and various property settings.

Launching an spring boot application: -

The **main ()** method uses Spring Boot’s **SpringApplication.run()** method to launch an application

There wasn’t a single line of XML? No **web.xml** file either. This web application is 100% pure Java and you didn’t have to deal with configuring any plumbing or infrastructure.