**Spring Boot Features**

• **Core Features:** SpringApplication | External Configuration | Profiles | Logging

• **Web Applications:** MVC | Embedded Containers

• **Working with data:** SQL | NO-SQL

• **Messaging:** Overview | JMS

• **Testing:** Overview | Boot Applications | Utils

• **Extending:** Auto-configuration | @Conditions

**Moving to Production**

• **Management endpoints:** Overview | Customization

• **Connection options:** HTTP | JMX

• **Monitoring:** Metrics | Auditing | Tracing | Process

**System Requirements**

Java 8 or 9 and Spring Framework 5.0.5.BUILDSNAPSHOT

Maven 3.2+ and Gradle 4.

**Servlet Containers**

**Name Servlet Version**

Tomcat 8.5 3.1

Jetty 9.4 3.1

Undertow 1.4 3.1

You can also deploy Spring Boot applications to any Servlet 3.0+ compatible container

**Inside POM:**

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

**<parent>**

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

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

**<version>**2.0.1.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>**

**NOTE: same way we can add- Repository and Plugin Repositories as well.**

**Spring Boot Supports:** Maven, Gradle and Ant

**Developing Your First Spring Boot Application**

1. Spring.io site
2. Spring Boot starter in STS
3. CLI – specifically for Groovy only

Sample POM:

<?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>**

**<parent>**

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

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

**<version>**2.0.1.BUILD-SNAPSHOT**</version>**

**</parent>**

*<!-- Additional lines to be added here... -->*

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

**<repositories>**

**<repository>**

**<id>**spring-snapshots**</id>**

**<url>**[https://repo.spring.io/snapshot**</url**](https://repo.spring.io/snapshot%3c/url)**>**

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

**</repository>**

**<repository>**

**<id>**spring-milestones**</id>**

**<url>**https://repo.spring.io/milestone**</url>**

**</repository>**

**</repositories>**

**<pluginRepositories>**

**<pluginRepository>**

**<id>**spring-snapshots**</id>**

**<url>**https://repo.spring.io/snapshot**</url>**

**</pluginRepository>**

**<pluginRepository>**

**<id>**spring-milestones**</id>**

**<url>**https://repo.spring.io/milestone**</url>**

**</pluginRepository>**

**</pluginRepositories>**

**</project>**

we can look at what we currently have by running the following command:

1. **$ mvn dependency:tree**

**Writing the Code**

**import** org.springframework.boot.\*;

**import** org.springframework.boot.autoconfigure.\*;

**import** org.springframework.web.bind.annotation.\*;

@RestController

@EnableAutoConfiguration

**public class** Example {

@RequestMapping("/")

String home() {

**return *"Hello World!"***;

}

**public static void** main(String[] args) **throws** Exception {

SpringApplication.run(Example.**class**, args);

}

}

**The @RestController and @RequestMapping Annotations**

@RestController :render the resulting string directly back to the caller.

@RequestMapping :“routing” information.

**@EnableAutoConfiguration Annotation**

guess” how you want to configure Spring

**Running the Example**

1. **mvn spring-boot:run**

(run is a maven goal)

1. **java -jar target/mywebserviceapp-0.0.1-SNAPSHOT.jar**

remote debugging support enabled:

$ java -Xdebug -Xrunjdwp:server=y,transport=dt\_socket,address=8000,suspend=n \

-jar target/myapplication-0.0.1-SNAPSHOT.jar

**Creating an Executable Jar**

add the spring-boot-maven-plugin to our pom.xml

**<build>**

**<plugins>**

**<plugin>**

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

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

**</plugin>**

**</plugins>**

**</build>**

**Using Spring Boot**

Bills of Materials (spring-boot-dependencies), that can be used with both Maven and Gradle.

Sensible resource filtering for application.properties and application.yml including

profile-specific files (for example, application-dev.properties and application-dev.yml)

**override individual dependencies:**

**<properties>**

**<spring-data-releasetrain.version>**Fowler-SR2**</spring-data-releasetrain.version>**

**</properties>**

**Starters:** spring-boot-starter-\*

**e.g.:** spring-boot-starter-data-jpa

**3 vs 1**

@Configuration

@EnableAutoConfiguration

@ComponentScan

**Or,** @SpringBootApplication

@Configuration

@EnableAutoConfiguration

@ComponentScan

**public class** Application {

**public static void** main(String[] args) {

SpringApplication.run(Application.**class**, args);

}

}

Or,

@SpringBootApplication

**public class** Application {

**public static void** main(String[] args) {

SpringApplication.run(Application.**class**, args);

}

}

**Disabling Specific Auto-configuration Classes**

@Configuration

@EnableAutoConfiguration(exclude={DataSourceAutoConfiguration.class})

**public class** MyConfiguration {

}

**NOTE: If a bean has one constructor, you can omit the @Autowired, as shown in the following example:**

@Service

**public class** DatabaseAccountService **implements** AccountService {

**private final** RiskAssessor riskAssessor;

@Autowired

**public** DatabaseAccountService(RiskAssessor riskAssessor) {

**this**.riskAssessor = riskAssessor;

}

*// ...*

}

Or,

@Service

**public class** DatabaseAccountService **implements** AccountService {

**private final** RiskAssessor riskAssessor;

**public** DatabaseAccountService(RiskAssessor riskAssessor) {

**this**.riskAssessor = riskAssessor;

}

*// ...*

}

**Developer Tools:** spring-boot-devtools

LiveReload.

Disabling Restart: spring.devtools.restart.enabled property.

**public static void** main(String[] args) {

System.setProperty(***"spring.devtools.restart.enabled"***, ***"false"***);

SpringApplication.run(MyApp.**class**, args);

}

**Note:** If you do not want to start the LiveReload server when your application runs, you can set the

spring.devtools.livereload.enabled property to false.

**Spring Boot features**

**Startup Failure:** FailureAnalyzers

**enable the debug:** $ java -jar myproject-0.0.1-SNAPSHOT.jar –debug (advanced, cover it later)

**Customizing the Banner**

**Application Events and Listeners**

listeners to be registered automatically 🡪 **META-INF/spring.factories**

org.springframework.context.ApplicationListener=com.example.project.MyListener

* ApplicationStartingEvent
* ApplicationEnvironmentPreparedEvent
* ApplicationPreparedEvent
* ApplicationStartedEvent
* ApplicationReadyEvent
* ApplicationFailedEvent

**CommandLineRunner:**

run some specific code once the SpringApplication has started, called just before SpringApplication.run(…) completes.

**Shutdown Hook:**

**Custom Error Pages:**

src/

+- main/

+- java/

| + <source code>

+- resources/

+- public/

+- error/

| +- 404.html

+- <other public assets>