In28
Minutes

Spring Boot in 10(ish) Steps

Getting Started with Spring Boot

- **WHY** Spring Boot?
  - You can build web apps & REST API WITHOUT Spring Boot
  - What is the need for Spring Boot?
- **WHAT** are the goals of Spring Boot?
- **HOW** does Spring Boot work?
- **COMPARE** Spring Boot vs Spring MVC vs Spring

Getting Started with Spring Boot - Approach

- **1:** Understand the world before Spring Boot (10000 Feet)
- **2:** Create a Spring Boot Project
- **3:** Build a simple REST API using Spring Boot
- **4:** Understand the MAGIC of Spring Boot
  - Spring Initializr
  - Starter Projects
  - Auto Configuration
  - Developer Tools
  - Actuator
  - ...

World Before Spring Boot!

- Setting up Spring Projects **before Spring Boot was NOT easy**!
- We needed to configure a **lot of things** before we have a **production-ready** application

World Before Spring Boot - 1 - Dependency Management

```xml
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>6.2.2.RELEASE</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.13.3</version>
</dependency>
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
</dependency>
```

- Manage frameworks and versions
  - **REST API** - Spring framework, Spring MVC framework, JSON binding framework, ..
  - **Unit Tests** - Spring Test, Mockito, JUnit, ...

World Before Spring Boot - 2 - web.xml

```
<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>
```

- **Example**: Configure **DispatcherServlet** for Spring MVC

World Before Spring Boot - 3 - Spring Configuration

```
<context:component-scan base-package="com.in28minutes" />

<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>
```

- Define your **Spring Configuration**
  - Component Scan
  - View Resolver
  - ....

World Before Spring Boot - 4 - NFRs

```xml
<plugin>
    <groupId>org.apache.tomcat.maven</groupId>
    <artifactId>tomcat7-maven-plugin</artifactId>
    <version>2.2</version>
    <configuration>
        <path>/</path>
        <contextReloadable>true</contextReloadable>
    </configuration>
</plugin>

<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
</dependency>
```

- Logging
- Error Handling
- Monitoring

World Before Spring Boot!

- Setting up Spring Projects **before Spring Boot was NOT easy**!
  - 1: Dependency Management (**pom.xml**)
  - 2: Define Web App Configuration (**web.xml**)
  - 3: Manage Spring Beans (**context.xml**)
  - 4: Implement Non Functional Requirements (NFRs)
- AND repeat this for every new project!
- Typically takes a **few days** to setup for each project (and countless hours to maintain)

Understanding Power of Spring Boot

```
// http://localhost:8080/courses
[
  {
    "id": 1,
    "name": "Learn AWS",
    "author": "in28minutes"
  }
]
```

- **1:** Create a Spring Boot Project
- **2:** Build a simple REST API using Spring Boot

What's the Most Important Goal of Spring Boot?

- Help you build **PRODUCTION-READY** apps **QUICKLY**
  - Build **QUICKLY**
    - Spring Initializr
    - Spring Boot Starter Projects
    - Spring Boot Auto Configuration
    - Spring Boot DevTools
  - Be **PRODUCTION-READY**
    - Logging
    - Different Configuration for Different Environments
      - Profiles, ConfigurationProperties
    - Monitoring (Spring Boot Actuator)
    - ...

Spring Boot

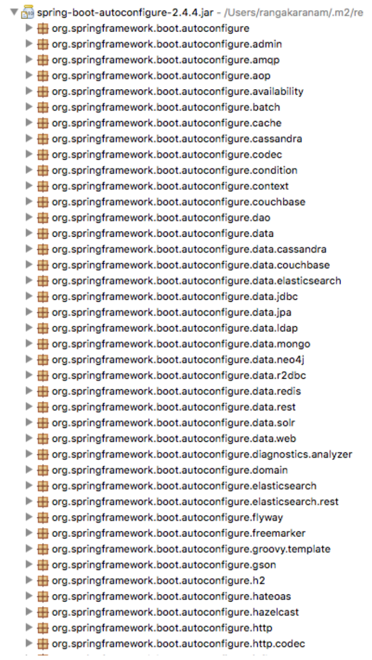
BUILD QUICKLY

Exploring Spring Boot Starter Projects

- I need a lot of frameworks to build application features:
  - **Build a REST API:** I need Spring, Spring MVC, Tomcat, JSON conversion...
  - **Write Unit Tests:** I need Spring Test, JUnit, Mockito, ...
- How can I group them and make it easy to build applications?
  - **Starters:** Convenient **dependency descriptors** for diff. features
- **Spring Boot** provides variety of starter projects:
  - **Web Application & REST API** - Spring Boot Starter Web (spring-webmvc, spring-web, spring-boot-starter-tomcat, spring-boot-starter-json)
  - **Unit Tests** - Spring Boot Starter Test
  - **Talk to database using JPA** - Spring Boot Starter Data JPA
  - **Talk to database using JDBC** - Spring Boot Starter JDBC
  - **Secure your web application or REST API** - Spring Boot Starter Security
- (REMEMBER) **Starters:** Define all application dependencies

Exploring Spring Boot Auto Configuration

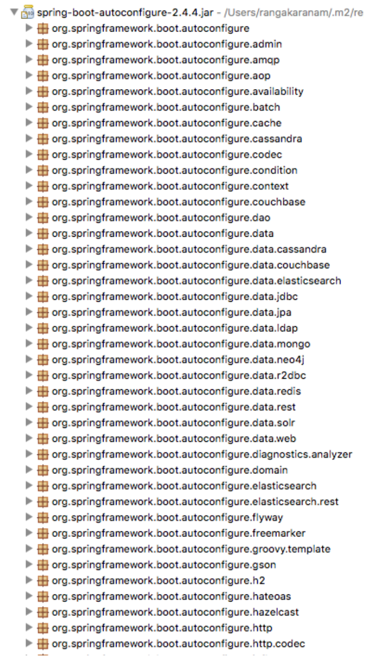
- I need **lot of configuration** to build Spring app:
  - Component Scan, DispatcherServlet, Data Sources, JSON Conversion, ...
- How can I simplify this?
  - **Auto Configuration: Automated configuration** for your app
    - **Decided** based on:
      - Which frameworks are in the Class Path?
      - What is the existing configuration (Annotations etc)?
- **Example**: Spring Boot Starter Web
  - Dispatcher Servlet (`DispatcherServletAutoConfiguration`)
  - Embedded Servlet Container - Tomcat is the default (`EmbeddedWebServerFactoryCustomizerAutoConfiguration`)
  - Default Error Pages (`ErrorMvcAutoConfiguration`)
  - Bean<->JSON (`JacksonHttpMessageConvertersConfiguration`)



```
spring-boot-autoconfigure-2.4.4.jar - /Users/rangakaranam/.m2/re
  org.springframework.boot.autoconfigure
  org.springframework.boot.autoconfigure.admin
  org.springframework.boot.autoconfigure.amqp
  org.springframework.boot.autoconfigure.aop
  org.springframework.boot.autoconfigure.availability
  org.springframework.boot.autoconfigure.batch
  org.springframework.boot.autoconfigure.cache
  org.springframework.boot.autoconfigure.cassandra
  org.springframework.boot.autoconfigure.codec
  org.springframework.boot.autoconfigure.condition
  org.springframework.boot.autoconfigure.context
  org.springframework.boot.autoconfigure.couchbase
  org.springframework.boot.autoconfigure.dao
  org.springframework.boot.autoconfigure.data
  org.springframework.boot.autoconfigure.data.cassandra
  org.springframework.boot.autoconfigure.data.couchbase
  org.springframework.boot.autoconfigure.data.elasticsearch
  org.springframework.boot.autoconfigure.data.jdbc
  org.springframework.boot.autoconfigure.data.jpa
  org.springframework.boot.autoconfigure.data.ldap
  org.springframework.boot.autoconfigure.data.mongo
  org.springframework.boot.autoconfigure.data.neo4j
  org.springframework.boot.autoconfigure.data.r2dbc
  org.springframework.boot.autoconfigure.data.redis
  org.springframework.boot.autoconfigure.data.rest
  org.springframework.boot.autoconfigure.data.solr
  org.springframework.boot.autoconfigure.data.web
  org.springframework.boot.autoconfigure.diagnostics.analyzer
  org.springframework.boot.autoconfigure.domain
  org.springframework.boot.autoconfigure.elasticsearch
  org.springframework.boot.autoconfigure.elasticsearch.rest
  org.springframework.boot.autoconfigure.flyway
  org.springframework.boot.autoconfigure.freemarker
  org.springframework.boot.autoconfigure.groovy.template
  org.springframework.boot.autoconfigure.gson
  org.springframework.boot.autoconfigure.h2
  org.springframework.boot.autoconfigure.hateoas
  org.springframework.boot.autoconfigure.hazelcast
  org.springframework.boot.autoconfigure.http
  org.springframework.boot.autoconfigure.http.codec
```

Understanding the Glue - @SpringBootApplication

- Questions:
  - Who is launching the Spring Context?
  - Who is triggering the component scan?
  - Who is enabling auto configuration?
- Answer: **@SpringBootApplication**
  - 1: **@SpringBootConfiguration**: Indicates that a class provides Spring Boot application @Configuration.
  - 2: **@EnableAutoConfiguration**: Enable auto-configuration of the Spring Application Context,
  - 3: **@ComponentScan**: Enable component scan (for current package, by default)



```
spring-boot-autoconfigure-2.4.4.jar - /Users/rangakaranam/.m2/re
  org.springframework.boot.autoconfigure
  org.springframework.boot.autoconfigure.admin
  org.springframework.boot.autoconfigure.amqp
  org.springframework.boot.autoconfigure.aop
  org.springframework.boot.autoconfigure.availability
  org.springframework.boot.autoconfigure.batch
  org.springframework.boot.autoconfigure.cache
  org.springframework.boot.autoconfigure.cassandra
  org.springframework.boot.autoconfigure.codec
  org.springframework.boot.autoconfigure.condition
  org.springframework.boot.autoconfigure.context
  org.springframework.boot.autoconfigure.couchbase
  org.springframework.boot.autoconfigure.dao
  org.springframework.boot.autoconfigure.data
  org.springframework.boot.autoconfigure.data.cassandra
  org.springframework.boot.autoconfigure.data.couchbase
  org.springframework.boot.autoconfigure.data.elasticsearch
  org.springframework.boot.autoconfigure.data.jdbc
  org.springframework.boot.autoconfigure.data.jpa
  org.springframework.boot.autoconfigure.data.ldap
  org.springframework.boot.autoconfigure.data.mongo
  org.springframework.boot.autoconfigure.data.neo4j
  org.springframework.boot.autoconfigure.data.r2dbc
  org.springframework.boot.autoconfigure.data.redis
  org.springframework.boot.autoconfigure.data.rest
  org.springframework.boot.autoconfigure.data.solr
  org.springframework.boot.autoconfigure.data.web
  org.springframework.boot.autoconfigure.diagnostics.analyzer
  org.springframework.boot.autoconfigure.domain
  org.springframework.boot.autoconfigure.elasticsearch
  org.springframework.boot.autoconfigure.elasticsearch.rest
  org.springframework.boot.autoconfigure.flyway
  org.springframework.boot.autoconfigure.freemarker
  org.springframework.boot.autoconfigure.groovy.template
  org.springframework.boot.autoconfigure.gson
  org.springframework.boot.autoconfigure.h2
  org.springframework.boot.autoconfigure.hateoas
  org.springframework.boot.autoconfigure.hazelcast
  org.springframework.boot.autoconfigure.http
  org.springframework.boot.autoconfigure.http.codec
```

Build Faster with Spring Boot DevTools

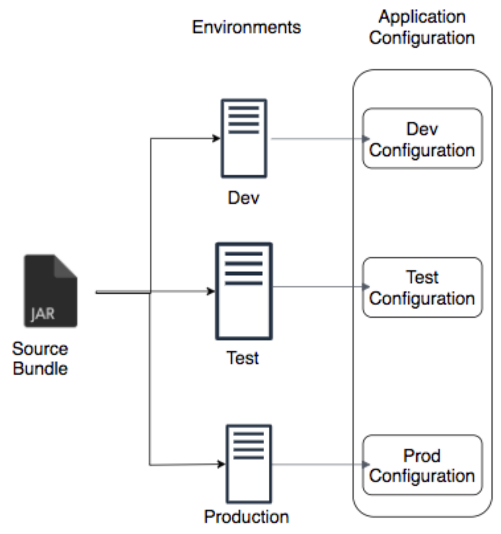
- Increase developer productivity
- Why do you need to restart the server **manually** for every code change?
- **Remember**: For pom.xml dependency changes, you will need to restart server **manually**

Spring Boot

PRODUCTION-READY

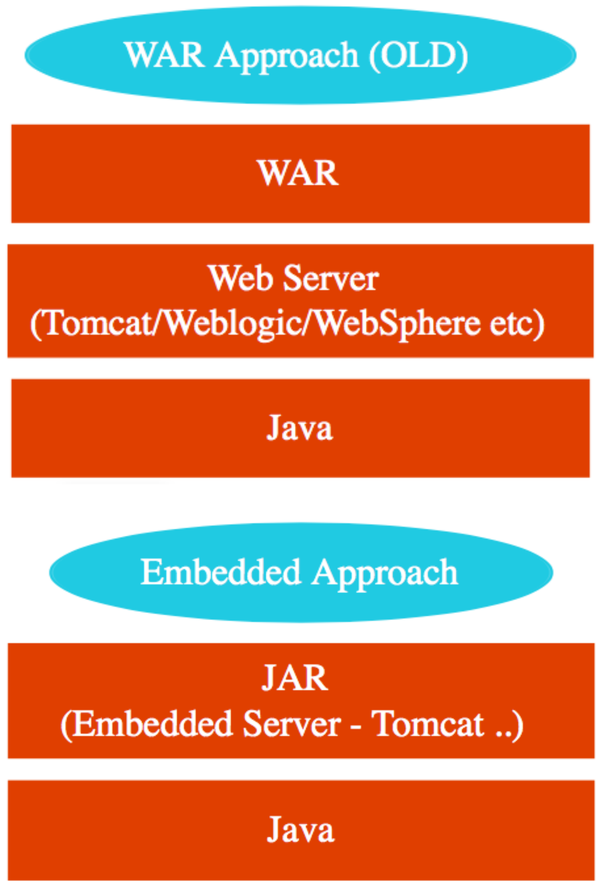
Managing App. Configuration using Profiles

- Applications have different environments: **Dev, QA, Stage, Prod**, ...
- Different environments need **different configuration**:
  - Different Databases
  - Different Web Services
- How can you provide different configuration for different environments?
  - **Profiles**: Environment specific configuration
- How can you define externalized configuration for your application?
  - **ConfigurationProperites**: Define externalized configuration



Simplify Deployment with Spring Boot Embedded Servers

- How do you deploy your application?
  - Step 1 : Install Java
  - Step 2 : Install Web/Application Server
    - Tomcat/WebSphere/WebLogic etc
  - Step 3 : Deploy the application WAR (Web ARchive)
    - This is the OLD **WAR** Approach
    - Complex to setup!
- **Embedded Server** - Simpler alternative
  - Step 1 : Install Java
  - Step 2 : Run **JAR** file
  - **Make JAR not WAR** (Credit: Josh Long!)
  - Embedded Server **Examples**:
    - spring-boot-starter-tomcat
    - spring-boot-starter-jetty
    - spring-boot-starter-undertow



WAR Approach (OLD)
WAR
Web Server (Tomcat/Weblogic/WebSphere etc)
Java
Embedded Approach
JAR (Embedded Server - Tomcat ..)
Java

Monitor Applications using Spring Boot Actuator

- Monitor and manage your application in your production
- Provides a number of endpoints:
  - **beans** - Complete list of Spring beans in your app
  - **health** - Application health information
  - **metrics** - Application metrics
  - **mappings** - Details around Request Mappings

Understanding Spring Boot vs Spring MVC vs Spring

- **Spring Boot vs Spring MVC vs Spring**: What's in it?
  - **Spring Framework**: Dependency Injection
    - @Component, @Autowired, Component Scan etc..
    - Just Dependency Injection is NOT sufficient (You need other frameworks to build apps)
      - **Spring Modules and Spring Projects**: Extend Spring Eco System
        - Provide good integration with other frameworks (Hibernate/JPA, JUnit & Mockito for Unit Testing)
  - **Spring MVC** (Spring Module): Simplify building web apps and REST API
    - Building web applications with Struts was very complex
    - @Controller, @RestController, @RequestMapping("/courses")
  - **Spring Boot** (Spring Project): Build **PRODUCTION-READY** apps **QUICKLY**
    - **Starter Projects** - Make it easy to build variety of applications
    - **Auto configuration** - Eliminate configuration to setup Spring, Spring MVC and other frameworks!
    - Enable non functional requirements (NFRs):
      - **Actuator**: Enables Advanced Monitoring of applications
      - **Embedded Server**: No need for separate application servers!
      - Logging and Error Handling
      - Profiles and ConfigurationProperties

Spring Boot - Review

- **Goal**: 10,000 Feet overview of Spring Boot
  - Help you understand the terminology!
    - Starter Projects
    - Auto Configuration
    - Actuator
    - DevTools
- **Advantages**: Get started quickly with production ready features!