Spring Boot Introduction

Spring in a Nutshell

- Very popular framework for building Java applications
- Provides a large number of helper classes and annotations

The Problem

- Building a traditional Spring application is really HARD!!!

Q: Which JAR dependencies do I need?

Q: How do I set up configuration (xml or Java)?

Q: How do I install the server? (Tomcat, JBoss etc…)

And that's JUST the basics for getting started

Spring Boot Solution

- Make it easier to get started with Spring development
- Minimize the amount of manual configuration
  - Perform auto-configuration based on props files and JAR classpath
- Help to resolve dependency conflicts (Maven or Gradle)
- Provide an embedded HTTP server so you can get started quickly
  - Tomcat, Jetty, Undertow, …

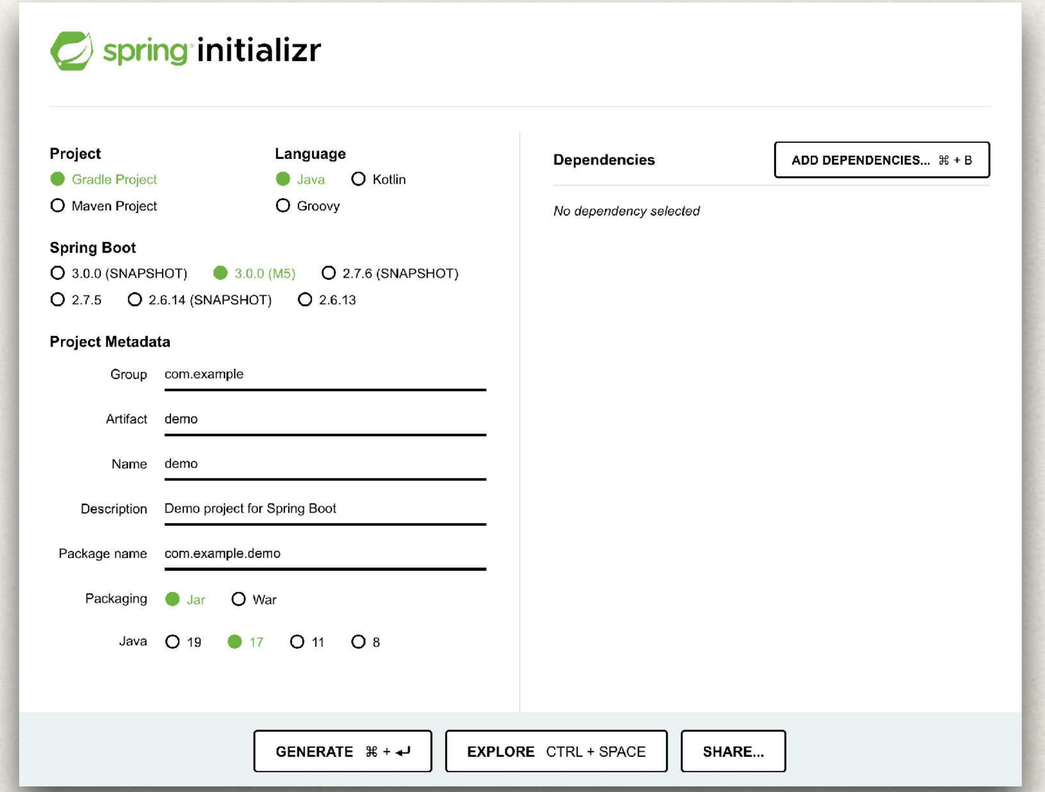
Spring Boot and Spring

- Spring Boot uses Spring behind the scenes
- Spring Boot simply makes it easier to use Spring

Spring Initializr

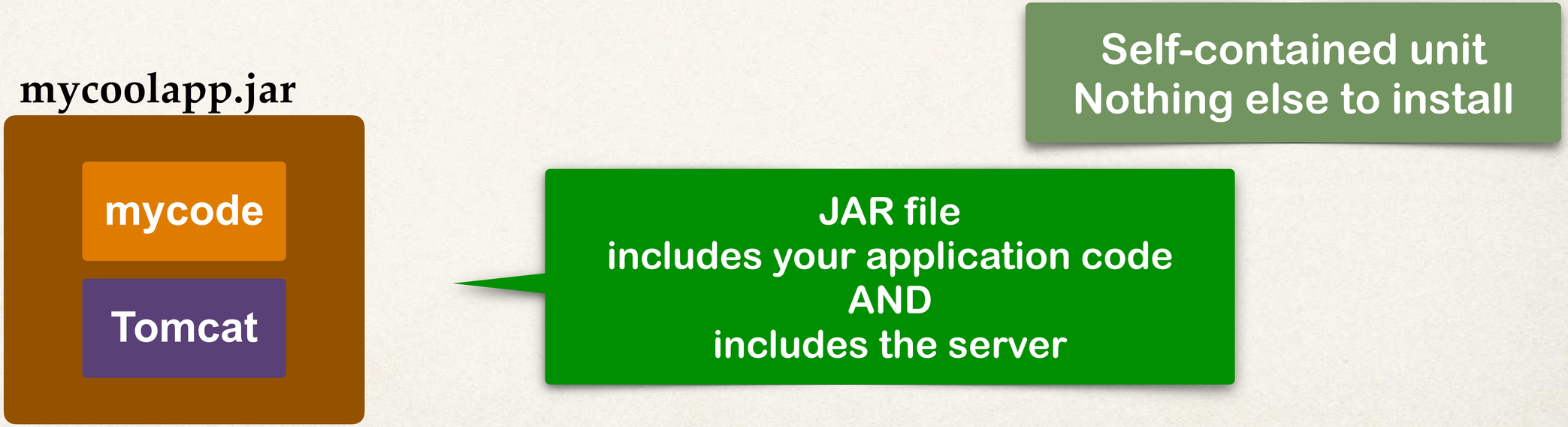
- Quickly create a starter Spring Boot project
- Select your dependencies
- Creates a Maven/Gradle project
- Import the project into your IDE
  - Eclipse, IntelliJ, NetBeans etc …

http://start.spring.io



Spring Boot Embedded Server

- Provide an embedded HTTP server so you can get started quickly
  - Tomcat, Jetty, Undertow, …
- No need to install a server separately



Running Spring Boot Apps

- Spring Boot apps can be run standalone (includes embedded server)
- Run the Spring Boot app from the IDE or command-line

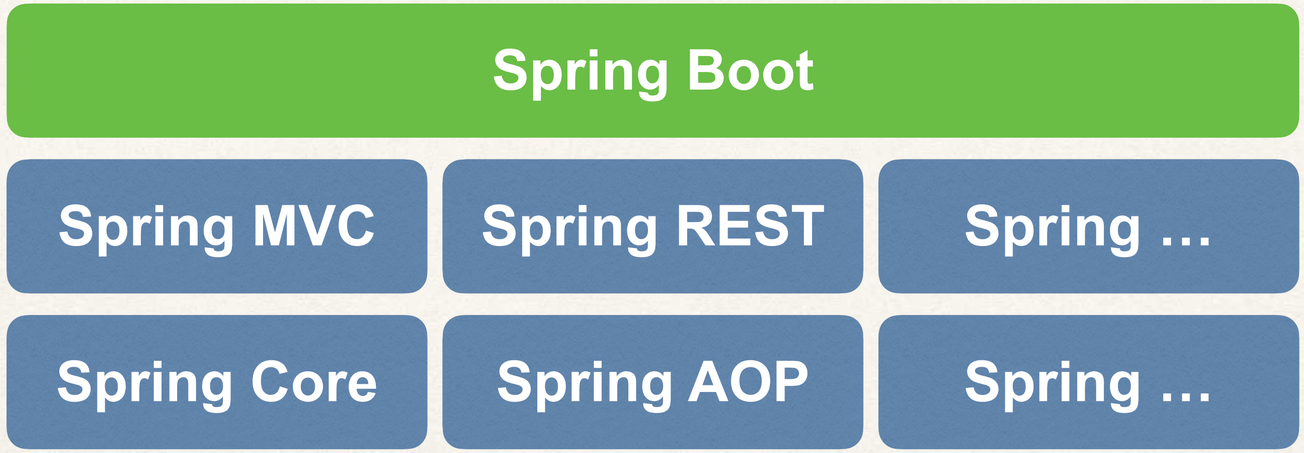
Deploying Spring Boot Apps

- Spring Boot apps can also be deployed in the traditional way
- Deploy **Web Application Archive (WAR) file** to an external server:
  - Tomcat, JBoss, WebSphere etc …

Spring Boot FAQ #1

Q: Does Spring Boot replace Spring MVC, Spring REST etc …?

- No. Instead, Spring Boot actually uses those technologies



Spring Boot FAQ #2

Q: Does Spring Boot run code faster than regular Spring code?

- No.
- Behind the scenes, Spring Boot uses same code of Spring Framework
- Remember, Spring Boot is about making it easier to get started
  - Minimizing configuration etc …

Spring Boot FAQ #3

Q: Do I need a special IDE for Spring Boot?

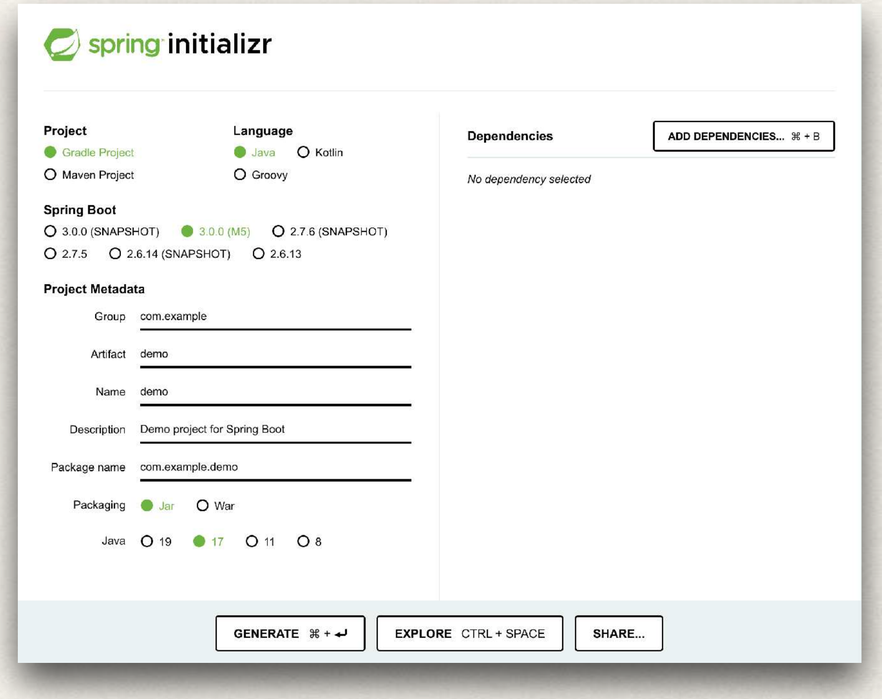
- No.
- You can use any IDE for Spring Boot apps … even use plain text editor
- The Spring team provides free *Spring Tool Suite (STS)* [IDE plugins]
- Some IDEs provide fancy Spring tooling support
- Not a requirement. Feel free to use the IDE that works best for you 

Spring Boot Initializr Demo

Spring Initializr

- Quickly create a starter Spring project
- Select your dependencies
- Creates a Maven/Gradle project
- Import the project into your IDE
  - Eclipse, IntelliJ, NetBeans etc …

http://start.spring.io



Quick Word on Maven

- When building your Java project, you may need additional JAR files
  - For example: Spring, Hibernate, Commons Logging, JSON etc…
- One approach is to download the JAR files from each project web site
- Manually add the JAR files to your build path / classpath

Maven Solution

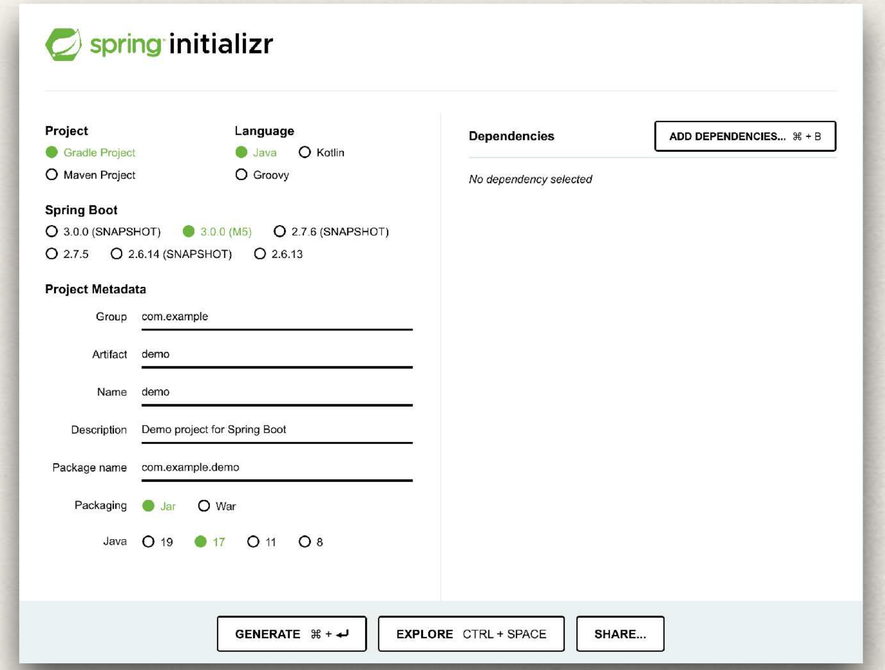
- Tell Maven the projects you are working with (dependencies)
  - Spring, Hibernate etc ….
- Maven will go out and download the JAR files for those projects for you
- And Maven will make those JAR files available during compile/run
- Think of Maven as your friendly helper / personal shopper :-)

Development Process

Step-By-Step

1. Configure our project at Spring Initializr website

http://start.spring.io



2. Download the zip file

3. Unzip the file

4. Import the project into our IDE

Spring Boot - Create REST Controller

REST Controller

- Let's create a very simple REST Controller

Create REST Controller

Set up rest controller

```
@RestController
public class FunRestController {

  // expose "/" that returns "Hello World"

  @GetMapping("/")
  public String sayHello() {
    return "Hello World!";
  }

}
```

Handle HTTP GET requests

Spring Framework Overview

Spring Website - Official

www.spring.io

Why Spring?

Simplify Java Enterprise Development

Goals of Spring

- Lightweight development with Java POJOs (Plain-Old-Java-Objects)
- Dependency injection to promote loose coupling
- Declarative programming with Aspect-Oriented-Programming (AOP)
- Minimize boilerplate Java code

Maven Crash Course

Spring Boot and Maven

- When you generate projects using Spring Initializr: start.spring.io
  - It can generate a Maven project for you

- In this section, we will learn the basics of Maven
  - Viewing dependencies in the Maven pom.xml file
  - Spring Boot Starters for Maven

What is Maven?

- Maven is a Project Management tool

- Most popular use of Maven is for build management and dependencies

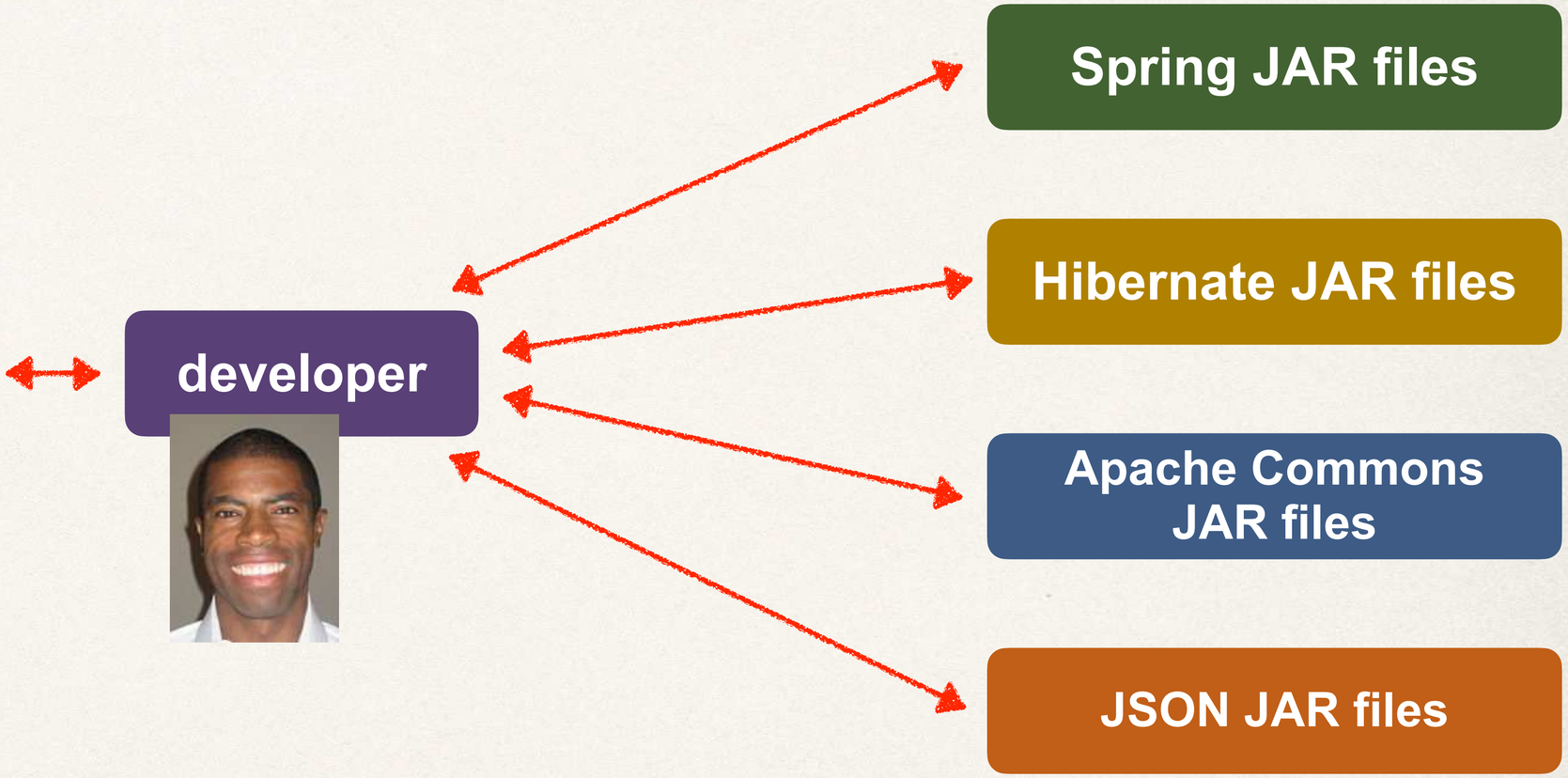
What Problems Does Maven Solve?

- When building your Java project, you may need additional JAR files
  - For example: Spring, Hibernate, Commons Logging, JSON etc…
- One approach is to download the JAR files from each project web site
- Manually add the JAR files to your build path / classpath

My Project without Maven

My Super Cool App

Spring
Hibernate
Commons Logging
JSON
…



Maven Solution

- Tell Maven the projects you are working with (dependencies)
  - Spring, Hibernate etc ….
- Maven will go out and download the JAR files for those projects for you
- And Maven will make those JAR files available during compile/run
- Think of Maven as your friendly helper / personal shopper :-)

My Project with Maven

Maven Central Repository (remote)

My Super Cool App

Spring
Hibernate
Commons Logging
JSON
...

developer

Maven

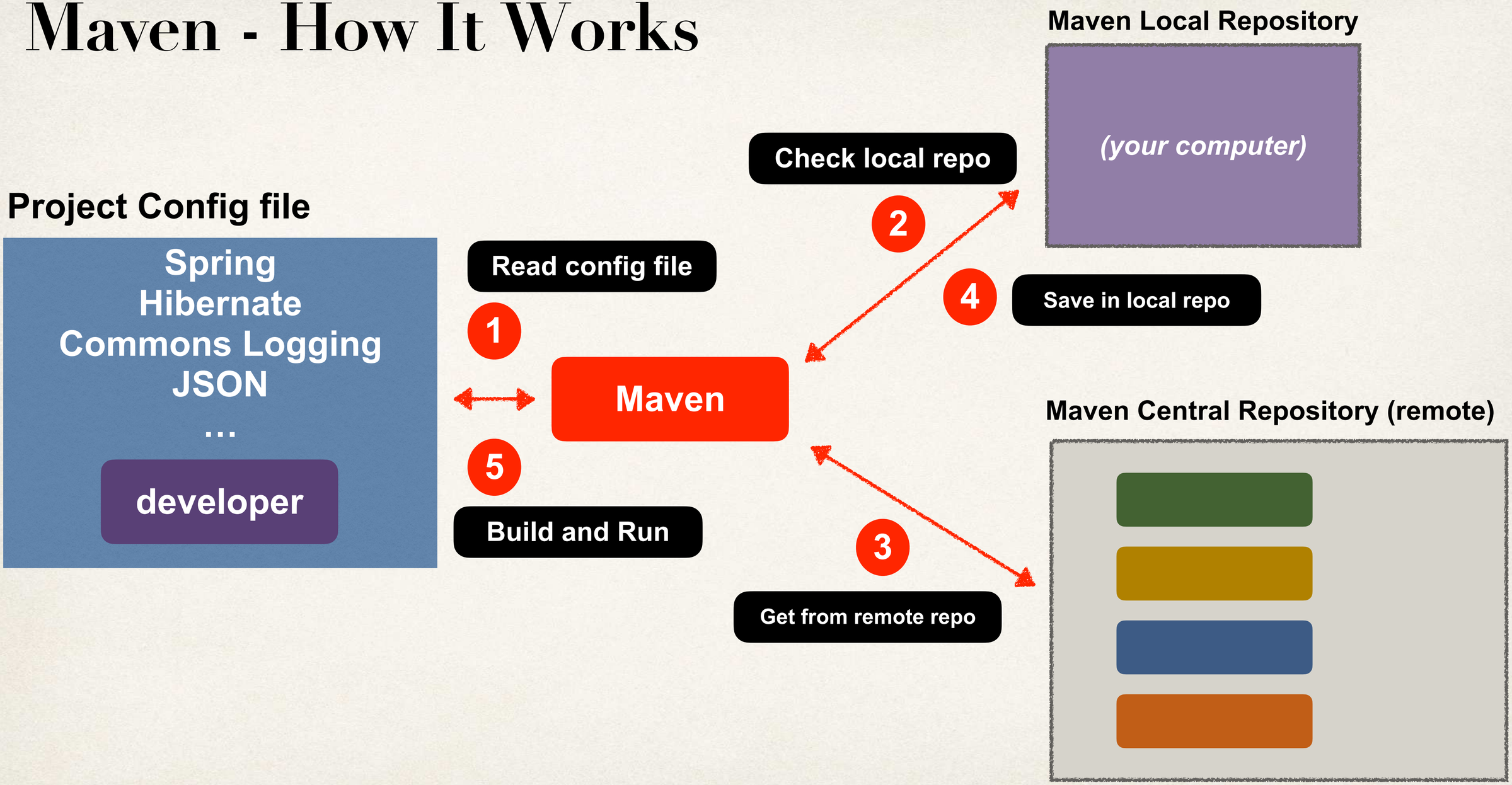
Spring JAR files

Hibernate JAR files

Apache Commons JAR files

JSON JAR files

Maven - How It Works



Handling JAR Dependencies

- When Maven retrieves a project dependency
  - It will also download supporting dependencies
  - For example: Spring depends on commons-logging …
- Maven will handle this for us automagically

Building and Running

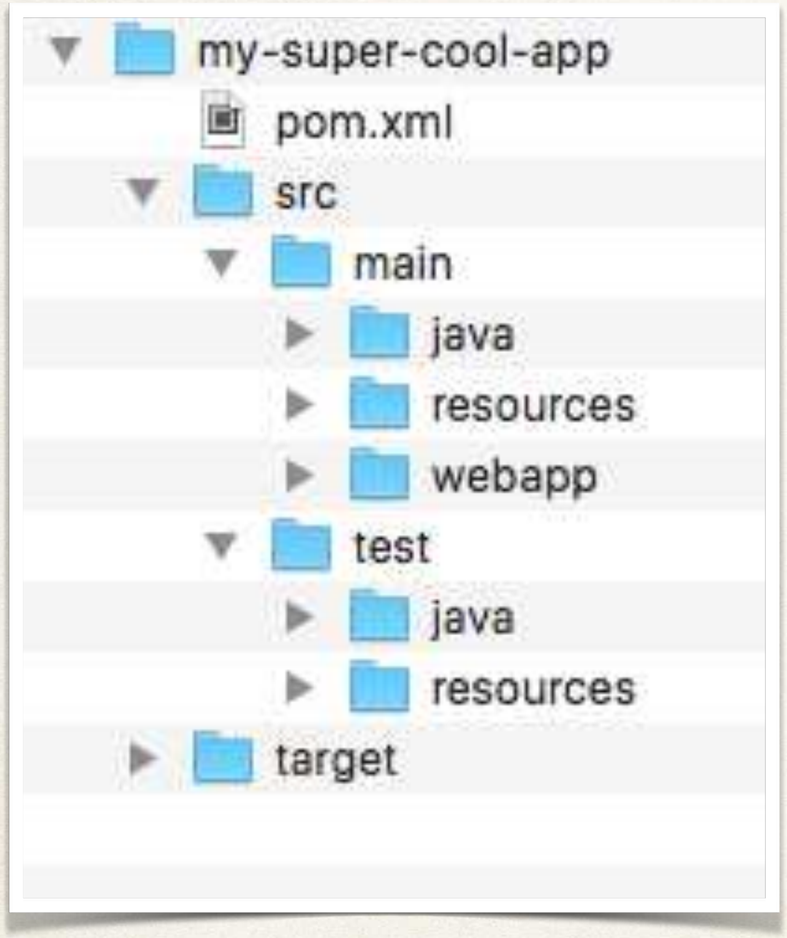
- When you build and run your app …
- Maven will handle class / build path for you
- Based on config file, Maven will add JAR files accordingly

Standard Directory Structure

- Normally when you join a new project
  - Each development team dreams up their own directory structure
  - Not ideal for new comers and not standardized

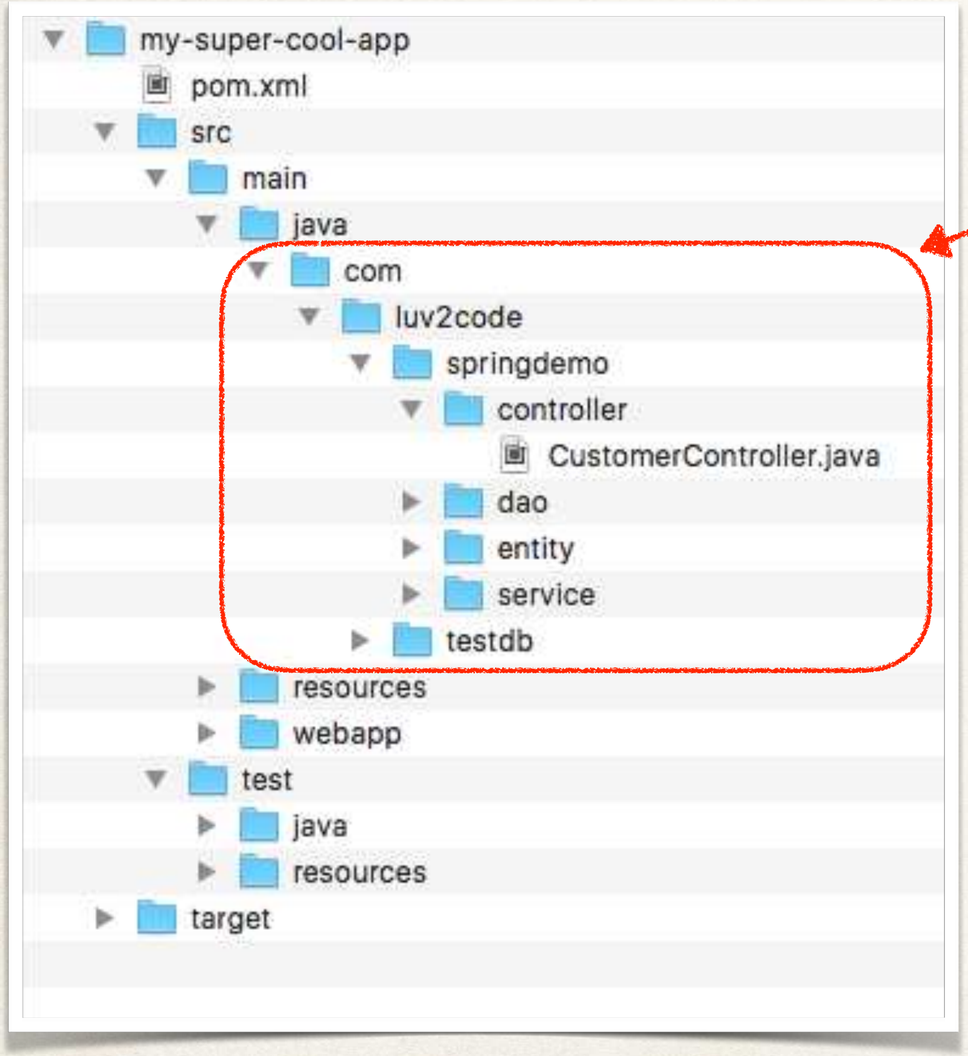
- Maven solves this problem by providing a standard directory structure

Standard Directory Structure



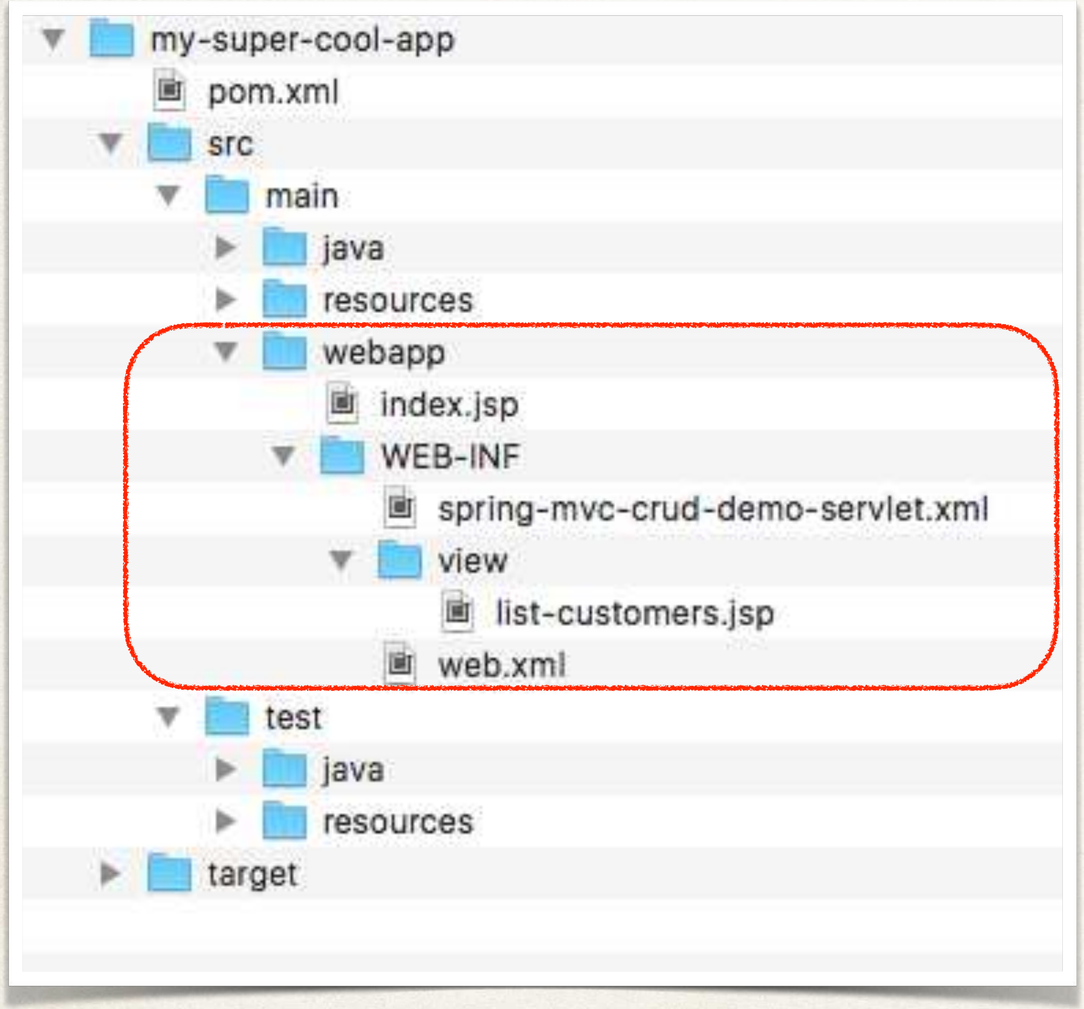
Directory	Description
src/main/java	Your Java source code
src/main/resources	Properties / config files used by your app
src/main/webapp	JSP files and web config files other web assets (images, css, js, etc)
src/test	Unit testing code and properties
target	Destination directory for compiled code. Automatically created by Maven

Standard Directory Structure



Place your Java source code here

Standard Directory Structure



Place your Web assets here

Standard Directory Structure Benefits

- For new developers joining a project
  - They can easily find code, properties files, unit tests, web files etc …

Standard Directory Structure Benefits

- Most major IDEs have built-in support for Maven
  - Eclipse, IntelliJ, NetBeans etc
  - IDEs can easily read/import Maven projects
- Maven projects are portable
  - Developers can easily share projects between IDEs
  - No need to fight about which IDE is the best LOL!

Advantages of Maven

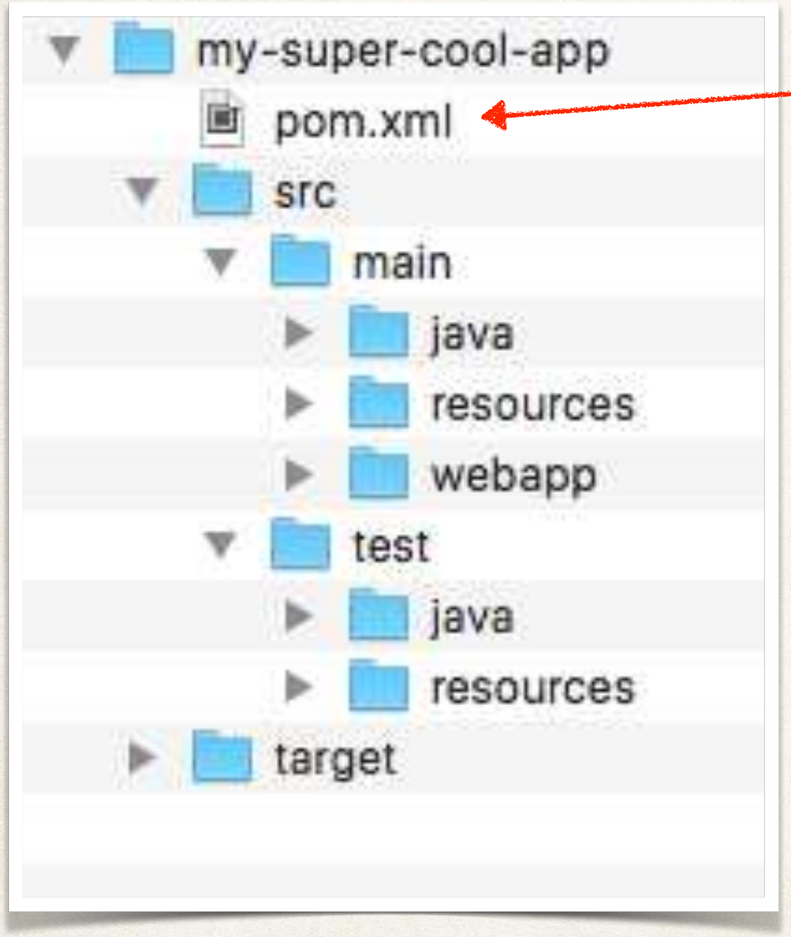
- Dependency Management
  - Maven will find JAR files for you
  - No more missing JARs
- Building and Running your Project
  - No more build path / classpath issues
- Standard directory structure

Maven Key Concepts

Maven Key Concepts

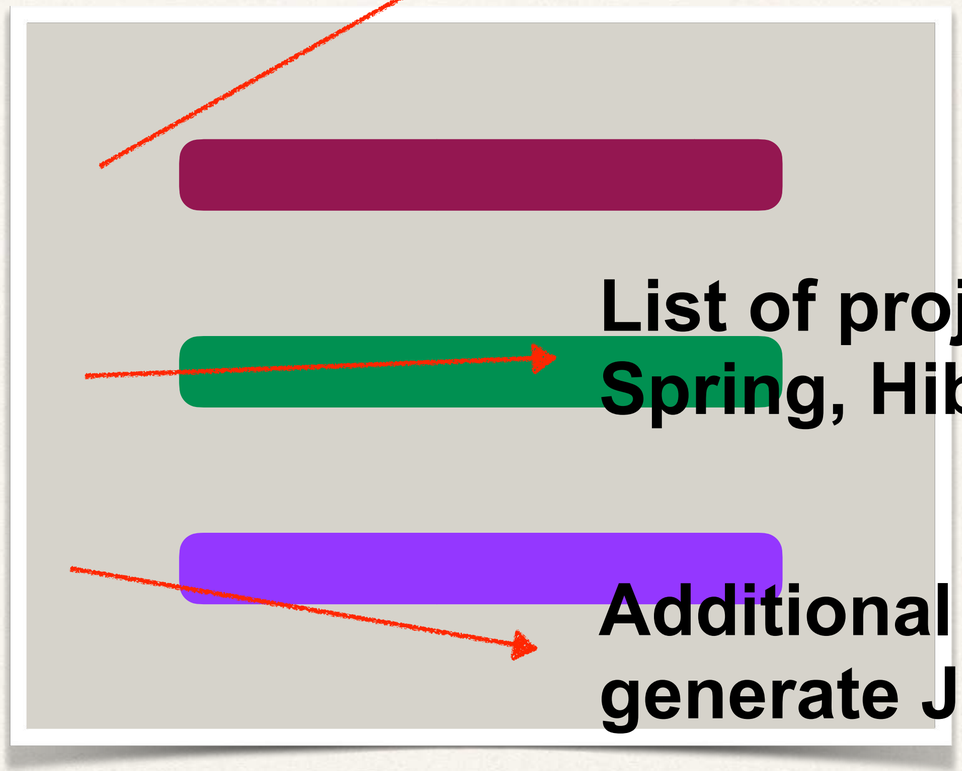
- POM File - pom.xml
- Project Coordinates

POM File - pom.xml



- Project Object Model file: POM file
- Configuration file for your project
  - Basically your “shopping list” for Maven :-)
- Located in the root of your Maven project

POM File Structure



Project name, version etc
Output file type: JAR, WAR, …

List of projects we depend on
Spring, Hibernate, etc…

Additional custom tasks to run:
generate JUnit test reports etc…

Simple POM File

```
<project ...>

  <modelVersion>4.0.0</modelVersion>

  <groupId>com.luv2code</groupId>
  <artifactId>mycoolapp</artifactId>
  <version>1.0.FINAL</version>
  <packaging>jar</packaging>

  <name>mycoolapp</name>

  <dependencies>
    <dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter</artifactId>
      <version>5.9.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>

    <!-- add plugins for customization -->

</project>
```

Project name, version etc
Output file type: JAR, WAR, …

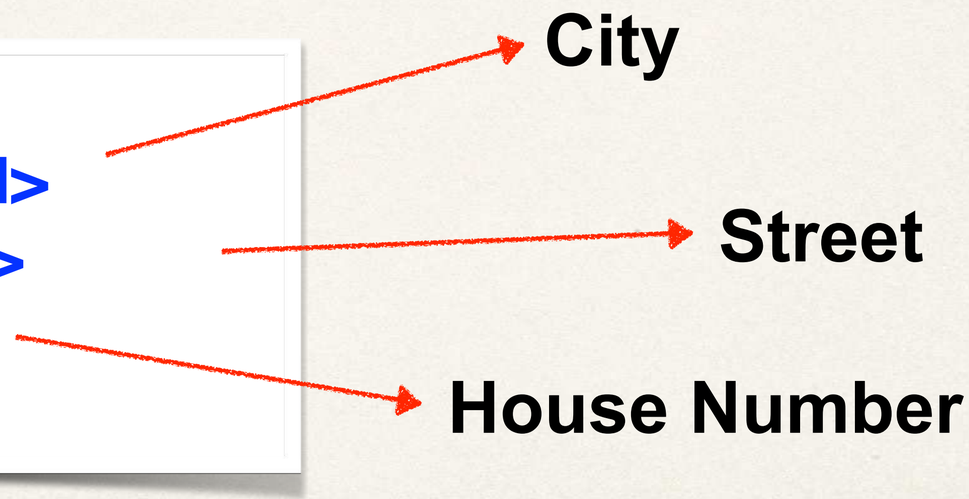
List of projects we depend on
Spring, Hibernate, etc…

Additional custom tasks to run:
generate JUnit test reports etc…

Project Coordinates

- Project Coordinates uniquely identify a project
  - Similar to GPS coordinates for your house: latitude / longitude
  - Precise information for finding your house (city, street, house #)

```
<groupId>com.luv2code</groupId>
<artifactId>mycoolapp</artifactId>
<version>1.0.FINAL</version>
```



Project Coordinates - Elements

Name	Description
Group ID	Name of company, group, or organization. Convention is to use reverse domain name: **com.luv2code**
Artifact ID	Name for this project: **mycoolapp**
Version	A specific release version like: **1.0, 1.6, 2.0** … If project is under active development then: **1.0-SNAPSHOT**

```
<groupId>com.luv2code</groupId>
<artifactId>mycoolapp</artifactId>
<version>1.0.FINAL</version>
```

Example of Project Coordinates

```
<groupId>com.luv2code</groupId>
<artifactId>mycoolapp</artifactId>
<version>1.0.RELEASE</version>
```

```
<groupId>org.springframework</groupId>
<artifactId>spring-context</artifactId>
<version>6.0.0</version>
```

```
<groupId>org.hibernate.orm</groupId>
<artifactId>hibernate-core</artifactId>
<version>6.1.4.Final</version>
```

Adding Dependencies

```
<project ...>
 ...
 <dependencies>

      <dependency>
         <groupId>org.springframework</groupId>
         <artifactId>spring-context</artifactId>
         <version>6.0.0</version>
      </dependency>

      <dependency>
         <groupId>org.hibernate.orm</groupId>
         <artifactId>hibernate-core</artifactId>
         <version>6.1.4.Final</version>
      </dependency>
      ...

 </dependencies>

</project>
```

Dependency Coordinates

- To add given dependency project, we need
  - **Group ID**, **Artifact ID**
  - **Version** is optional …
    - Best practice is to include the version (repeatable builds)

- May see this referred to as: **GAV**
  - **G**roup ID, **A**rtifact ID and **V**ersion

How to Find Dependency Coordinates

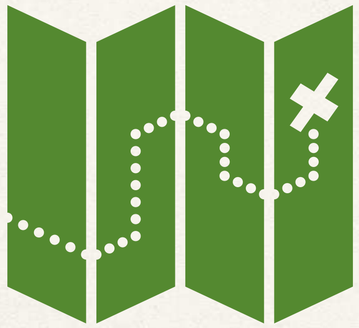
- Option 1: Visit the project page (spring.io, hibernate.org etc)

- Option 2: Visit http://search.maven.org (easiest approach)

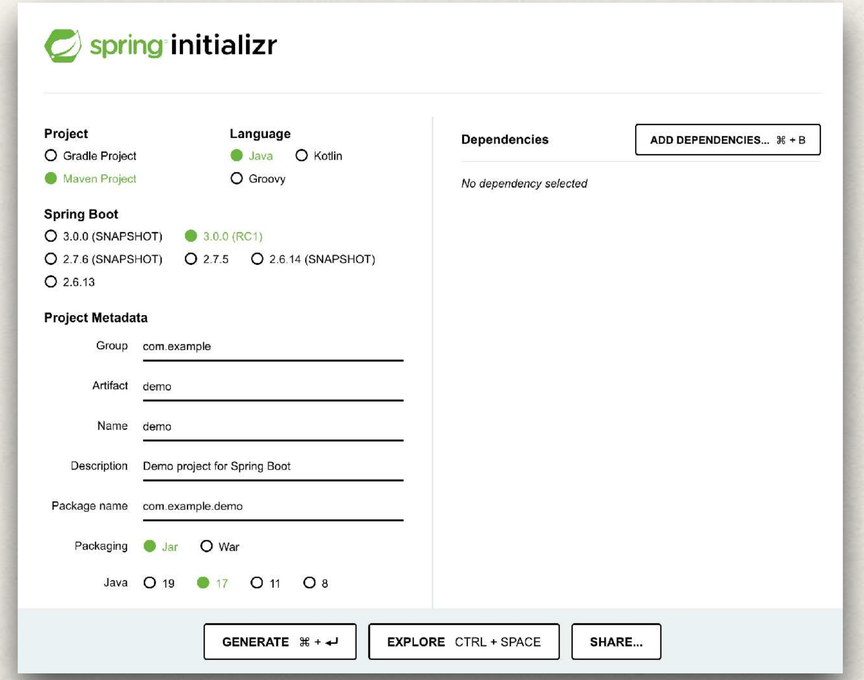
Spring Boot Project Structure

Spring Initializr

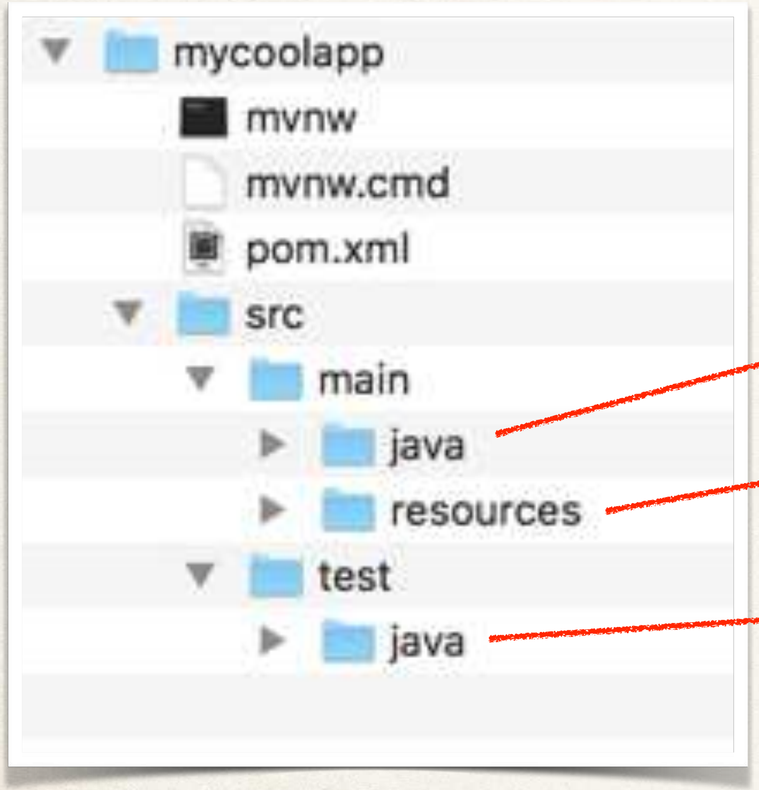
- Spring Initializr created a Maven project for us
- Let's explore the project structure



http://start.spring.io



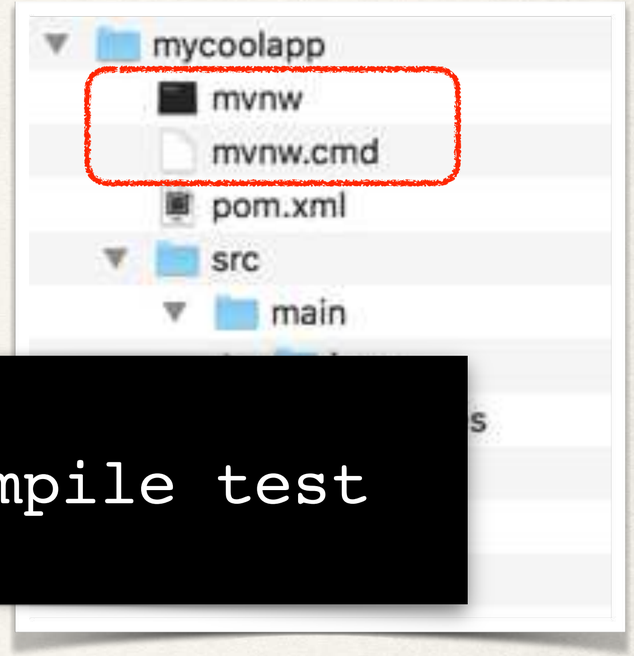
Maven Standard Directory Structure



Directory	Description
src/main/java	Your Java source code
src/main/resources	Properties / config files used by your app
src/test/java	Unit testing source code

Maven Wrapper files

- **mvnw** allows you to run a Maven project
  - No need to have Maven installed or present on your path
  - If correct version of Maven is NOT found on your computer
    - **Automatically downloads** correct ve
      and runs Maven
- Two files are provided
  - **mvnw.cmd** for MS Windows
  - **mvnw.sh** for Linux/Mac



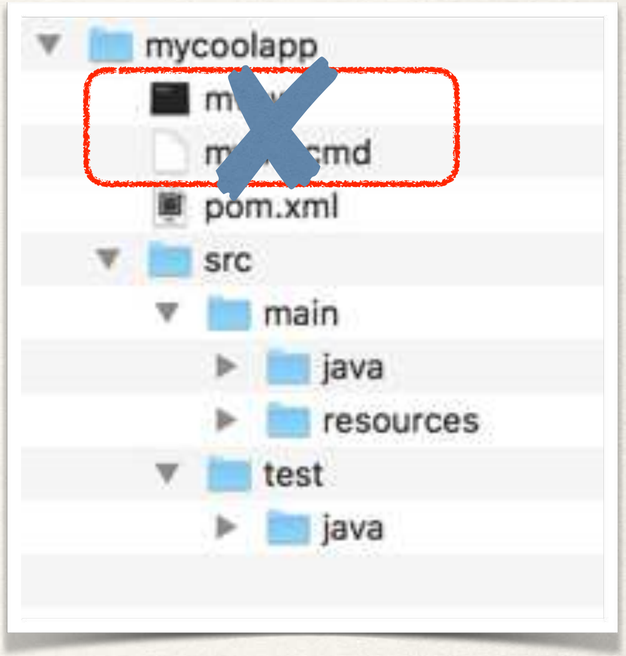

```
> mvnw clean compile test
```

```
$ ./mvnw clean compile test
```

Maven Wrapper files

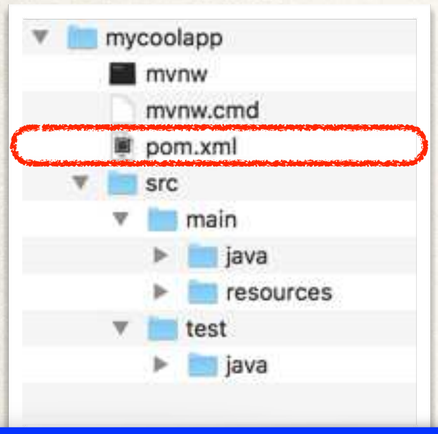
- If you already have Maven installed previously
  - Then you can ignore/delete the **`mvnw`** files
- Just use Maven as you normally would

```
$ mvn clean compile test
```



Maven POM file

- **`pom.xml`** includes info that you entered




```
<groupId>com.luv2code.springboot.demo</groupId>
<a
<v
<p
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    framework.b
    oot-starte

  </dependencies>
```

Spring Boot Starters

A collection of Maven dependencies (Compatible versions)

Save's the developer from having to list all of the individual dependencies

Also, makes sure you have compatible versions

spring-web
spring-webmvc
hibernate-validator
tomcat
json
...

Maven POM file

- Spring Boot Maven plugin


```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

To package executable jar or war archive

Can also easily run the app

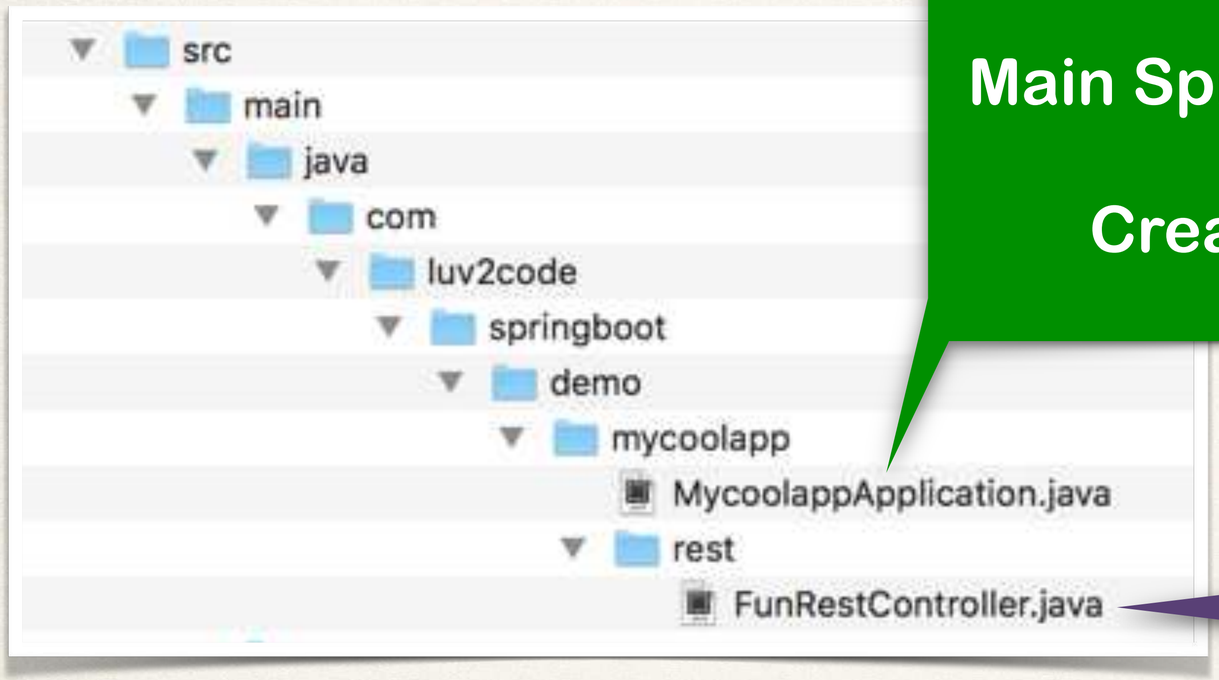
Can also just use:

mvn package
mvn spring-boot:run

```
$ ./mvnw package

$ ./mvnw spring-boot:run
```

Java Source Code



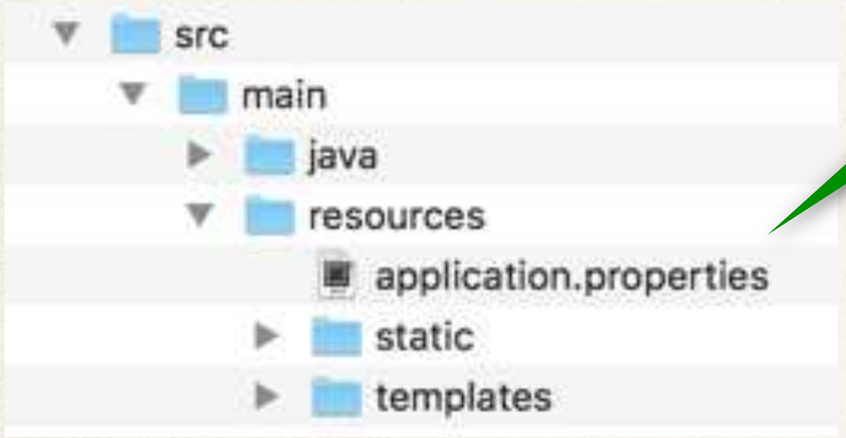
Main Spring Boot application class

Created by Spring Initializr

RestController that we created in an earlier video

Application Properties

- By default, Spring Boot will load properties from: **`application.properties`**



Created by Spring Initializr

Empty at the beginning

Can add Spring Boot properties

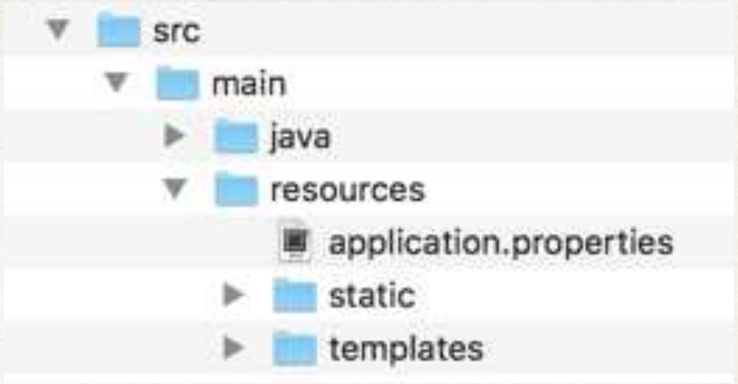
```
server.port=8585
```

Also add your own custom properties

```
coach.name=Mickey Mouse
```

Application Properties

- Read data from: **application.properties**




```
# configure server port
server.port=8484

# configure my props
coach.name=Mickey Mouse
team.name=The Mouse Crew
```

```
@RestController
public class FunRestController {

  @Value("${coach.name}")
  private String coachName;

  @Value("${team.name}")
  private String teamName;

  ...
}
```

Spring Boot Starters

The Problem

- Building a Spring application is really HARD!!!

FAQ: Which Maven dependencies do I need?

The Problem

```
<!-- Spring support -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>6.0.0-RC1</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
    <version>6.0.0-RC1</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version>6.0.0-RC1</version>
</dependency>
```

Spring version

```
<!-- Spring Security -->
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-web</artifactId>
    <version>6.0.0-RC1</version>
</dependency>

<!-- Hibernate ORM -->
<dependency>
    <groupId>org.hibernate.orm</groupId>
    <artifactId>hibernate-core</artifactId>
    .1.4.Final</version>


 Validator -->

g.hibernate.validator</groupId>
hibernate-validator</artifactId>
    <version>7.0.5.Final</version>
</dependency>
```

Which versions are compatible?

Very error-prone

Easy to make a simple mistake

Why Is It So Hard?

- It would be great if there was a simple list of Maven dependencies
- Collected as a group of dependencies … one-stop shop
- So I don't have to search for each dependency

There should be an easier solution

The Solution - Spring Boot Starters

- **Spring Boot Starters**
- A curated list of Maven dependencies
- A collection of dependencies grouped together
- Tested and verified by the Spring Development team
- Makes it much easier for the developer to get started with Spring
- Reduces the amount of Maven configuration

Spring Boot Starters

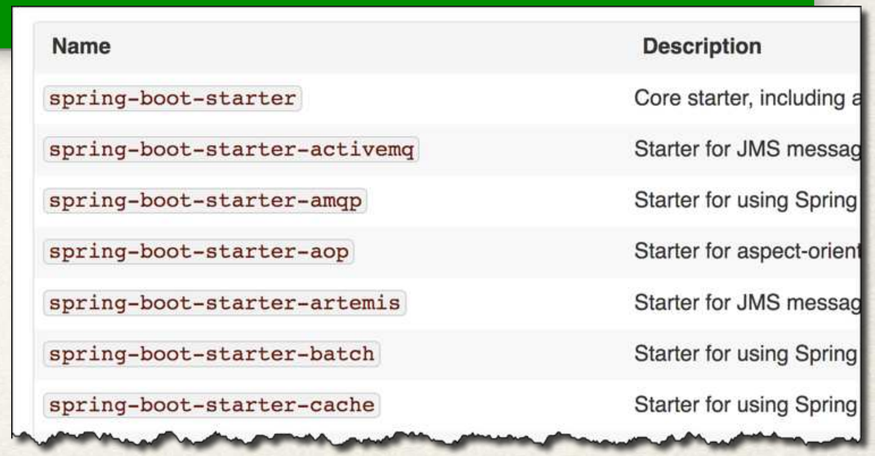
- There are 30+ Spring Boot Starters from the Spring Development team

Name	Description
`spring-boot-starter-web`	**Building web apps, includes validation, REST. Uses Tomcat as default embedded server**
`spring-boot-starter-security`	**Adding Spring Security support**
`spring-boot-starter-data-jpa`	**Spring database support with JPA and Hibernate**
...	

Spring Boot Starters

Full list

www.luv2code.com/spring-boot-starters



Name	Description
spring-boot-starter	Core starter, including a
spring-boot-starter-activemq	Starter for JMS messag
spring-boot-starter-amqp	Starter for using Spring
spring-boot-starter-aop	Starter for aspect-orient
spring-boot-starter-artemis	Starter for JMS messag
spring-boot-starter-batch	Starter for using Spring
spring-boot-starter-cache	Starter for using Spring

Spring Boot Starter Parent

Spring Boot Starter Parent

- Spring Boot provides a "Starter Parent"
- This is a special starter that provides Maven defaults

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>3.0.0-RC1</version>
  <relativePath/> <!-- lookup parent from repository -->
</parent>
```

Included in pom.xml when using Spring Initializr

Spring Boot Starter Parent

- Maven defaults defined in the Starter Parent
  - Default compiler level
  - UTF-8 source encoding
  - *Others …*

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>3.0.0-RC1</version>
  <relativePath/> <!-- lookup parent from repository -->
</parent>
```

Spring Boot Starter Parent

- To override a default, set as a property

```
<properties>
  <java.version>17</java.version>
</properties>
```

Specify your Java version

Spring Boot Starter Parent

- For the **`spring-boot-starter-*`** dependencies, no need to list version

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>3.0.0-RC1</version>
  <relativePath/> <!-- lookup parent from repository -->
</parent>
```

Specify version of Spring Boot

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
```

Inherit version from Starter Parent

No need to list individual versions
Great for maintenance!

Spring Boot Starter Parent

- Default configuration of Spring Boot plugin

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

```
> mvn spring-boot:run
```

Benefits of the Spring Boot Starter Parent

- Default Maven configuration: Java version, UTF-encoding etc
- Dependency management
  - Use version on parent only
  - **`spring-boot-starter-*`** dependencies inherit version from parent
- Default configuration of Spring Boot plugin

Spring Boot Dev Tools

The Problem

- When running Spring Boot applications
  - If you make changes to your source code
  - Then you have to manually restart your application :-(

Solution: Spring Boot Dev Tools

- **`spring-boot-devtools`** to the rescue!
- Automatically restarts your application when code is updated
- Simply add the dependency to your POM file
- No need to write additional code :-)
- For IntelliJ, need to set additional configurations ... will discuss shortly

Spring Boot Dev Tools

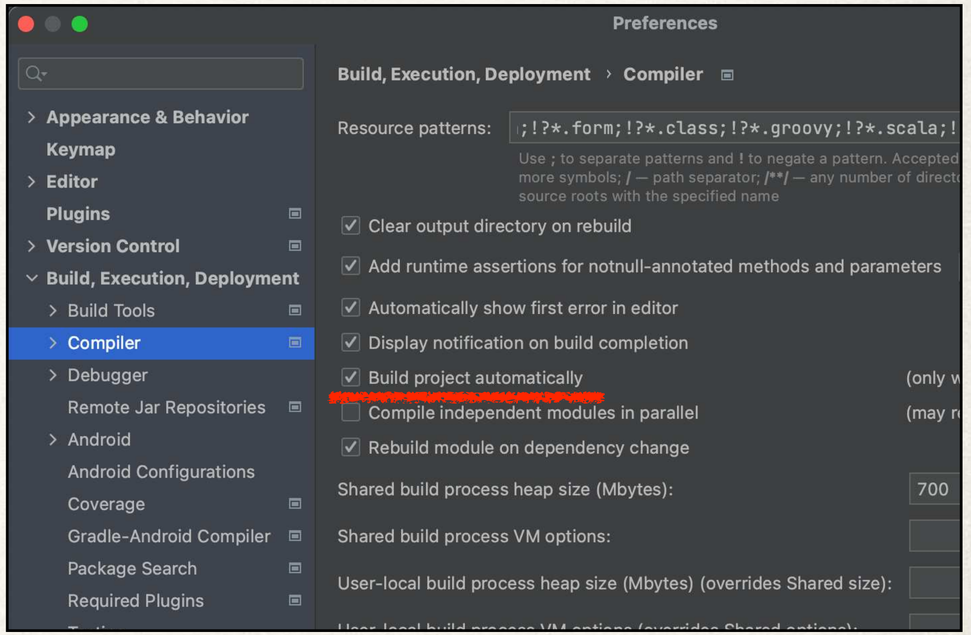
- Adding the dependency to your POM file

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
</dependency>
```

Automatically restarts your application when code is updated

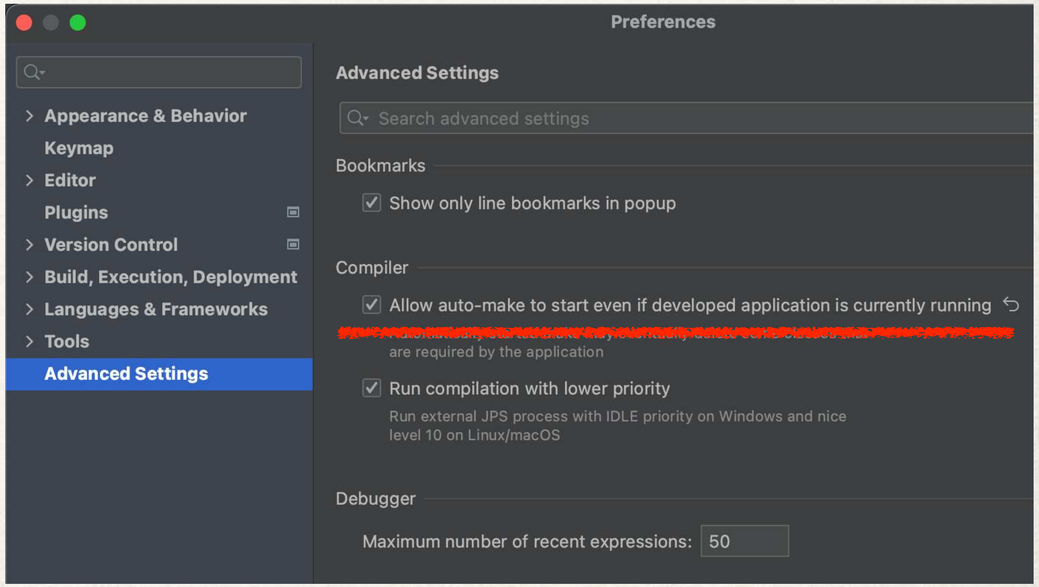
IntelliJ Community Edition - DevTools

- IntelliJ Community Edition does not support DevTools by default
- Select: **Preferences > Build, Execution, Deployment > Compiler**
  - Check box: **Build project automatically**



IntelliJ Community Edition

- Additional setting
- Select: **Preferences > Advanced Settings**
  - Check box: **Allow auto-make to ...**



Development Process

1. Apply IntelliJ configurations

2. Edit **`pom.xml`** and add **`spring-boot-devtools`**

3. Add new REST endpoint to our app

4. Verify the app is automatically reloaded

Spring Boot DevTools Documentation

For more details

www.luv2code.com/devtools-docs

Spring Boot Actuator

Problem

- How can I monitor and manage my application?
- How can I check the application health?
- How can I access application metrics?

Solution: Spring Boot Actuator

- Exposes endpoints to monitor and manage your application
- You easily get DevOps functionality out-of-the-box
- Simply add the dependency to your POM file
- REST endpoints are automatically added to your application

No need to write additional code!

You get new REST endpoints for FREE!

Spring Boot Actuator

- Adding the dependency to your POM file

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

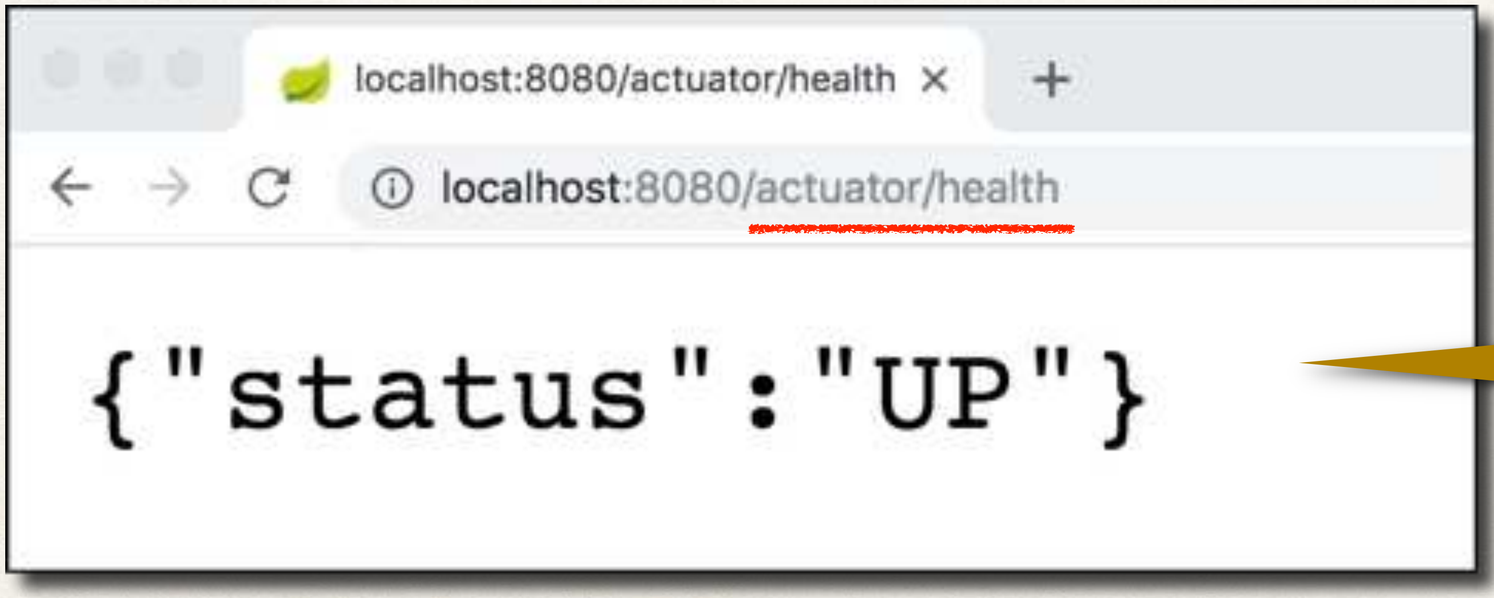
Spring Boot Actuator

- Automatically exposes endpoints for metrics out-of-the-box
- Endpoints are prefixed with: **`/actuator`**

Name	Description
`/health`	**Health information about your application**
...	

Health Endpoint

- **`/health`** checks the status of your application
- Normally used by monitoring apps to see if your app is up or down



Health status is customizable based on your own business logic

Exposing Endpoints

- By default, only **`/health`** is exposed
- The **`/info`** endpoint can provide information about your application
- To expose **`/info`**

File: src/main/resources/application.properties

```
management.endpoints.web.exposure.include=health,info
management.info.env.enabled=true
```

Info Endpoint

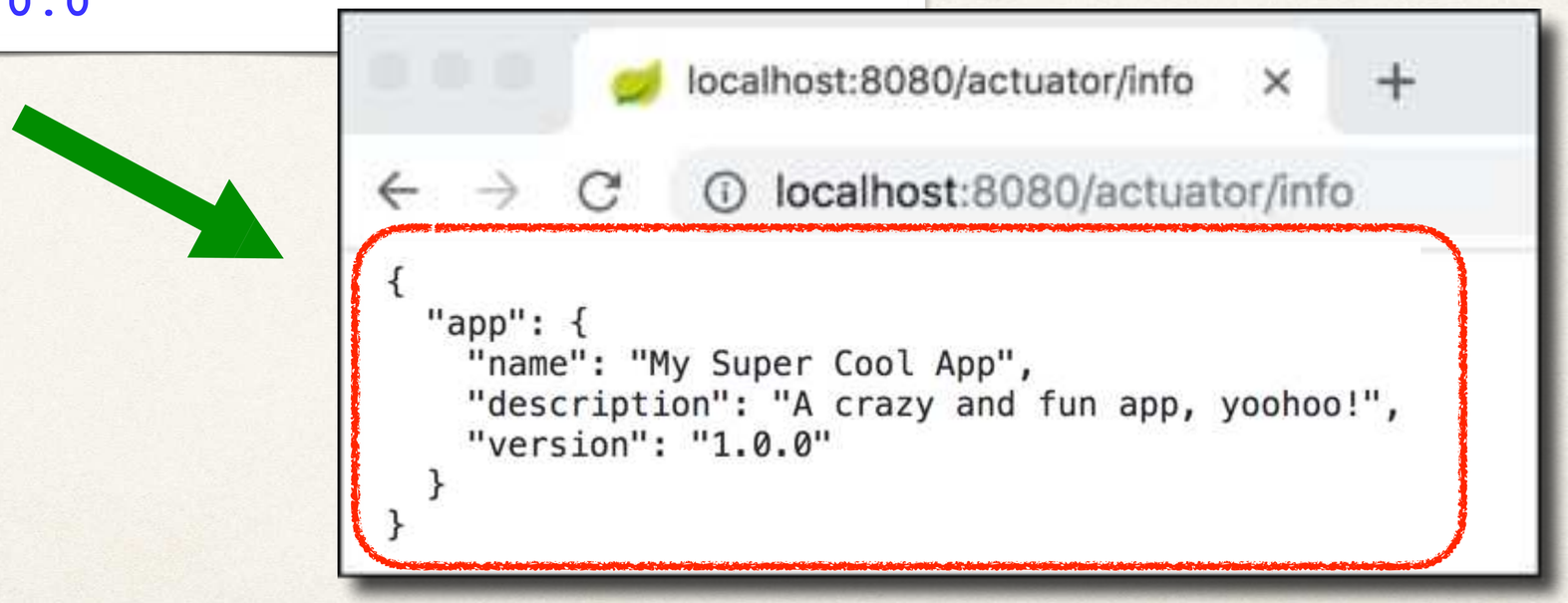
- `/info` gives information about your application
- Default is empty

Info Endpoint

- Update **`application.properties`** with your app info

File: src/main/resources/application.properties

```
info.app.name=My Super Cool App
info.app.description=A crazy and fun app, yoohoo!
info.app.version=1.0.0
```



Spring Boot Actuator Endpoints

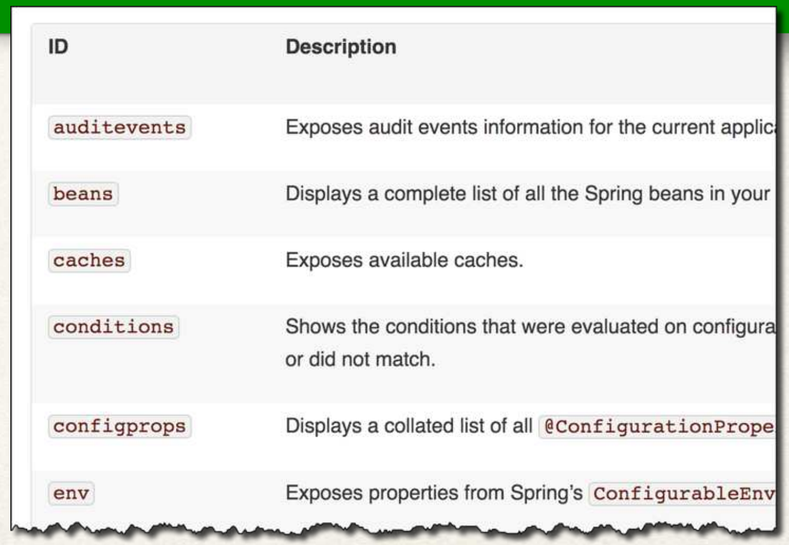
- There are 10+ Spring Boot Actuator endpoints

Name	Description
`/auditevents`	**Audit events for your application**
`/beans`	**List of all beans registered in the Spring application context**
`/mappings`	**List of all @RequestMapping paths**
...	

Spring Boot Actuator Endpoints

Full list

www.luv2code.com/actuator-endpoints



ID	Description
`auditevents`	Exposes audit events information for the current applic
`beans`	Displays a complete list of all the Spring beans in your
`caches`	Exposes available caches.
`conditions`	Shows the conditions that were evaluated on configura or did not match.
`configprops`	Displays a collated list of all `@ConfigurationPrope`
`env`	Exposes properties from Spring's `ConfigurableEnv`

Development Process

1. Edit **`pom.xml`** and add **`spring-boot-starter-acuator`**

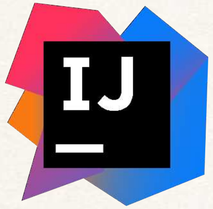
2. View actuator endpoints for: **`/health`**

3. Edit **`application.properties`** to customize **`/info`**

Spring Boot: Run from Command-Line

Running from the Command-Line

- During development we spend most of our time in the IDE



- However, we may want to run our Spring Boot app outside of the IDE

- One approach is running from the command-line

Running from the Command-Line

- When running from the command-line
  - No need to have IDE open/running



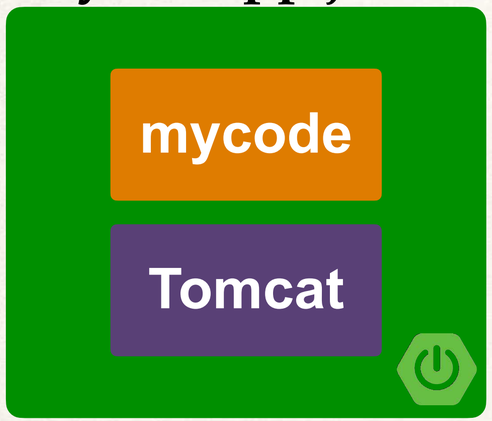
- Since we using Spring Boot, the server is embedded in our JAR file
  - No need to have separate server installed/running

- Spring Boot apps are self-contained

Running from the Command-Line

- Spring Boot apps are self-contained

mycoolapp.jar



Self-contained unit
Nothing else to install

JAR file
includes your application code
AND
includes the server

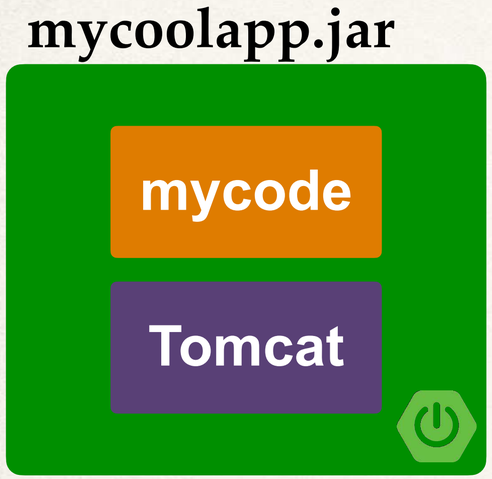
Running from the Command-Line

- Two options for running the app

- Option 1: Use **`java -jar`**

- Option 2: Use Spring Boot Maven plugin
  - **`mvnw spring-boot:run`**

Option 1: Use `java -jar`

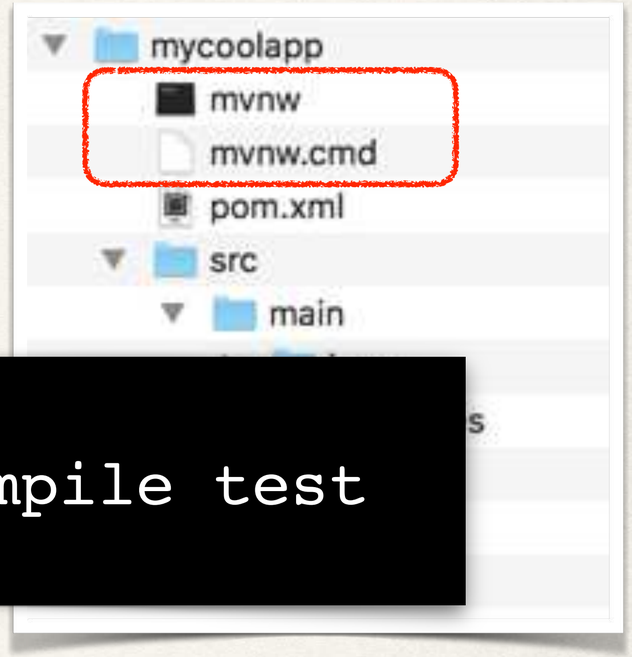



```
> java -jar mycoolapp.jar
```

Name of our JAR file

Option 2: Use Spring Boot Maven plugin

- **`mvnw`** allows you to run a Maven project
  - No need to have Maven installed or present on your path
  - If correct version of Maven is NOT found on your computer
    - **Automatically downloads** correct ve
      and runs Maven
- Two files are provided
  - **`mvnw.cmd`** for MS Windows
  - **`mvnw.sh`** for Linux/Mac



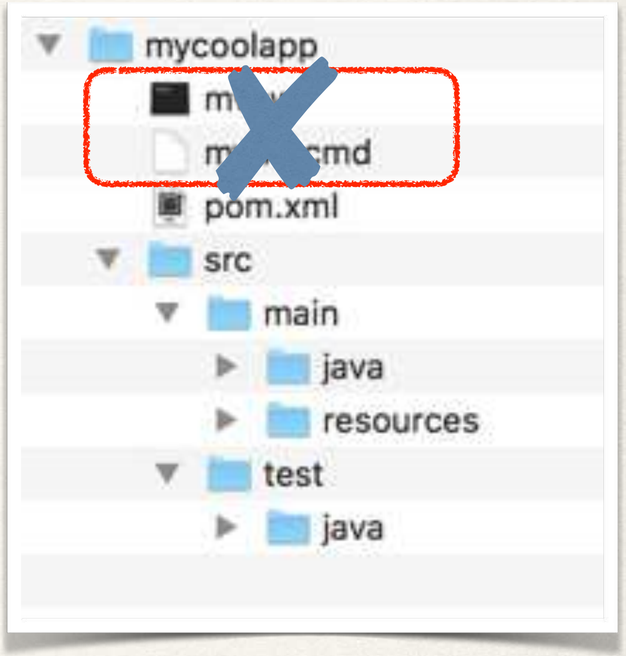

```
> mvnw clean compile test
```

```
$ ./mvnw clean compile test
```

Maven Wrapper files

- If you already have Maven installed previously
  - Then you can ignore/delete the **`mvnw`** files
- Just use Maven as you normally would

```
$ mvn clean compile test
```



Option 2: Use Spring Boot Maven plugin

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

To package executable jar or war archive

Can also easily run the app

```
$ ./mvnw package

$ ./mvnw spring-boot:run
```

Can also just use:

mvn package
mvn spring-boot:run

Development Process

1. Exit the IDE

2. Package the app using **`mvnw package`**

3. Run app using **`java -jar`**

4. Run app using Spring Boot Maven plugin, **`mvnw spring-boot:run`**

Spring Boot - Custom Application Properties

Problem

- You need for your app to be configurable … no hard-coding of values

- You need to read app configuration from a properties file

Solution: Application Properties file

- By default, Spring Boot reads information from a standard properties file
  - Located at: **`src/main/resources/application.properties`**
- You can define ANY custom properties in this file
- Your Spring Boot app can access properties using **`@Value`**

Standard Spring Boot file name

No additional coding or configuration required

Development Process

1. Define custom properties in **`application.properties`**

2. Inject properties into Spring Boot application using **`@Value`**

Step 1: Define custom application properties

File: src/main/resources/application.properties

```
#
# Define custom properties
#
coach.name=Mickey Mouse
team.name=The Mouse Club
```

You can give ANY custom property names

Step 2: Inject Properties into Spring Boot app

No additional coding or configuration required

```
@RestController
public class FunRestController {

  // inject properties for: coach.name and team.name

  @Value("${coach.name}")
  private String coachName;

  @Value("${team.name}")
  private String teamName;

  ...
}
```

File: src/main/resources/application.properties

```
#
# Define custom properties
#
coach.name=Mickey Mouse
team.name=The Mouse Club
```

Spring Boot Properties

Spring Boot Properties

- Spring Boot can be configured in the **`application.properties`** file
- Server port, context path, actuator, security etc …
- Spring Boot has 1,000+ properties … wowzers!

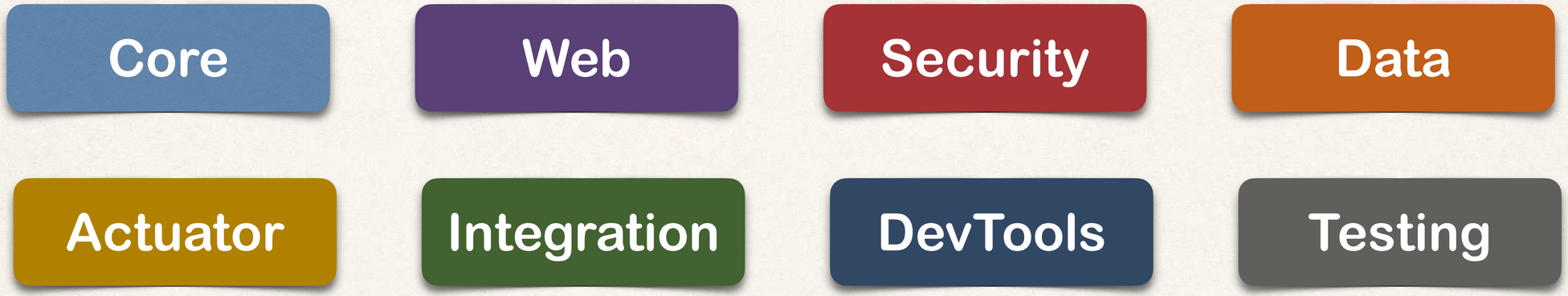
Spring Boot Properties

List of Common Properties

www.luv2code.com/spring-boot-props

Spring Boot Properties

- Don’t let the 1,000+ properties overwhelm you
- The properties are roughly grouped into the following categories



Spring Boot Properties

We'll review some of the properties …

Core Properties

Core

File: src/main/resources/application.properties

```
# Log levels severity mapping
logging.level.org.springframework=DEBUG
logging.level.org.hibernate=TRACE
logging.level.com.luv2code=INFO

# Log file name
logging.file.name=my-crazy-stuff.log
logging.file.path=c:/myapps/demo
...
```

Logging Levels

TRACE
DEBUG
INFO
WARN
ERROR
FATAL
OFF

Spring Boot Logging
www.luv2code.com/spring-boot-logging

Web Properties

Web

File: src/main/resources/application.properties

```
# HTTP server port
server.port=7070

# Context path of the application
server.servlet.context-path=/my-silly-app

# Default HTTP session time out
server.servlet.session.timeout=15m
...
```

http://localhost:7070/my-silly-app/fortune

15 minutes

Actuator Properties

Actuator

File: src/main/resources/application.properties

```
# Endpoints to include by name or wildcard
management.endpoints.web.exposure.include=*

# Endpoints to exclude by name or wildcard
management.endpoints.web.exposure.exclude=beans,mapping

# Base path for actuator endpoints
management.endpoints.web.base-path=/actuator
...
```

http://localhost:7070/actuator/health

Security Properties

Security

File: src/main/resources/application.properties

```
# Default user name
spring.security.user.name=admin

# Password for default user
spring.security.user.password=topsecret
...
```

Data Properties

Data

File: src/main/resources/application.properties

```
# JDBC URL of the database
spring.datasource.url=jdbc:mysql://localhost:3306/ecommerce

# Login username of the database
spring.datasource.username=scott

# Login password of the database
spring.datasource.password=tiger
...
```

More on this in later videos

Spring Boot Properties

List of Common Properties

www.luv2code.com/spring-boot-props