Spring Boot & Actuators

Agenda

- **Problems with Development**
- **What is Spring Boot?**
- **How do we use it?**
- **Live coding – Build a spring boot app**
- **Live coding – Add actuators for monitoring**
- **Deployment as a service**
- **Live coding - Hook up to the admin console**

Problems With Development

In Java, and in general.

spring

What problems are we trying to solve?

- **Developers keep re-solving the same problems.**
  - The point of coding is to build your business value-added services; be they your trading algorithms, your music service, etc.
  - Any time spent doing anything else is (usually) wasted.
- **Development with Java is mostly boiler-plate.**
  - Web-applications are very common and require lots of configuration/setup.
  - Packaging, deployment, and monitoring take time (every time!). This is a waste to your productivity; they are tangential to your business logic.
  - There are too many custom variations of essentially the same configuration, project layout, and deployment (makes standardization/training hard).

spring

What is Spring Boot?

An introduction to the framework

According to the project itself:

> Spring Boot makes it easy to create standalone, production-grade Spring based applications that you can just run.
>
> *Spring.io*

According to the project itself:

> We take an opinionated view of the platform and third-party libraries so that you can get started with minimum fuss.
>
> *Spring.io*

spring

Key Features

- **Get Spring web-services running with just couple lines of code (literally).**
- **Embed Tomcat or Undertow directly into them, providing a runnable JAR.**
- **Automatically configure Spring wherever possible (you can actually run without any configuration out of the box).**
- **Make your WAR function as an init.d service.**
- **Add standard metrics, health checks, and externalized configuration.**
- **Integrate with a huge number of things out of the box; for example:**
  - Consul for discoverability/distributed configuration.
  - Admin console for managing spring apps.

How Do We Use It?

A brief introduction.

spring

Bare Application Code

You can actually start up a Spring Boot web-app with just this code (and a few maven dependencies):

```
1  import org.springframework.boot.SpringApplication;
2  import org.springframework.boot.autoconfigure.SpringBootApplication;
3
4  @SpringBootApplication
5  public class MyApp {
6      public static void main(String[] args) {
7          SpringApplication.run(MyApp.class, args);
8      }
9  }
```

It will properly launch a web server and pick up any configuration files you provide. But that's all it does as we have no endpoints!

Adding an Endpoint

To make our service useful, we need to add a controller and endpoint.

```
1  @Controller
2  public class HomeController {
3      @Value(value = "${custom.message}") private String message;
4
5      @RequestMapping("/message")
6      @ResponseBody
7      public String index() {
8          return message;
9      }
10 }
```

Spring Boot will automatically component scan our classes (locate them and wire them together) when it finds annotations like @Controller.

Maven Dependencies

To make the code we just saw work, we need just 2 things from maven.

1. Derive from Spring Boot starter parent POM.
2. Include the spring-boot starter web dependency.

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.5.3.RELEASE</version>
</parent>

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

Configuration File

Since we are trying to read a ${custom.message} property, we also need a configuration file.

But by default we don't need one, it would just run on port 8080 out of the box; this is just specific to our code.

- **Add a "resources" folder under src/main**
- **Add an application.properties file to it.**
- **Add custom.message=Hello Spring Days!!!**
- **You can also override the server defaults here:**
  - server.port=[number]
  - server.contextpath=[/path]

What Does @SpringBootApplication Do?

It is syntactic sugar for multiple other powerful annotations commonly used together in Spring. From the docs:

> Many Spring Boot developers always have their classes annotated with @Configuration, @EnableAutoConfiguration, and @ComponentScan…Spring Boot provides a convenient @SpringBootApplication alternative.
>
> *Spring.io*

What do @SpringBootApplication's sub-annotations do?

- @Configuration is used to turn a class into a JavaConfig source so you can define beans in it, etc. The class used in SpringApplication.run() should be a configuration class (though XML configs are possible).
- @EnableAutoConfiguration attempts to guess and configure beans that you are likely to need based on our code and class-path. E.g. if Tomcat is present due to web dependencies in the POM, it will set it up and use it for you.
- @ComponentScan is used to tell Spring to automatically search for and wire up classes based on their annotations (e.g. make @Controllers register themselves, populate @Value variables, etc.)

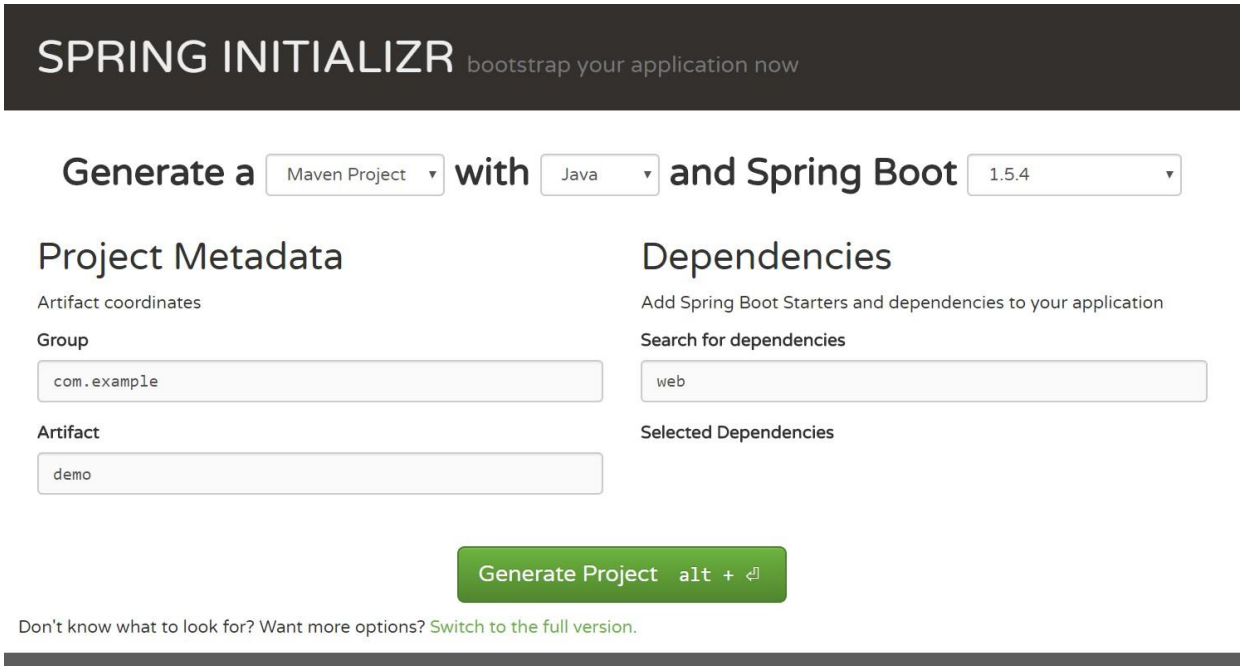
Live Coding Session Part #1

Let’s make the application!

spring

Spring Boot InitializR

Before we code this, just know that Spring Initializr can do it for you! I'm just doing it to show you how easy it is ☺.



spring

Live Coding Session Part #1

- **The code is hosted on GitHub; feel free to explore it.**
  - https://github.com/w00te/spring-days-spring-boot-example
- **Also, to keep things co-located, here's the admin console code for later on.**
  - https://github.com/w00te/spring-days-spring-boot-admin-console
- **What we'll achieve**
  - Create a new Java 1.8 Maven application using the quick-start archetype.
  - Add in our maven dependencies.
  - Add in our main class and controller.
  - Add a properties file.
  - Test our application on our own custom-defined port/context.

Live Coding Session Part #2

Let’s add monitoring and deployment features!

spring

Adding Actuator

- Spring boot actuator provides production grade metrics, auditing, and monitoring features to your application.
- Just adding the actuator dependency makes it available on our 8080 port. If we add hateoas as well, it will make it restfully navigable under /actuator. No code is required ☺.
- We'll also disable actuator endpoint security to make this demo run smoother of course ☺. We can do this by adding these two properties:

```
1  security.basic.enabled=false
2  management.security.enabled=false
```

Actuator Features to Notice

- Live thread dumps
- Live stack traces
- Logger configurations
- Password-masked properties file auditing
- Heap dump trigger (can be disabled)
- Web-request traces
- Application metrics
- Spring bean analysis
- More! (and it’s extensible)

Application Deployment

Deploying as an init.d service on Linux

spring

Deploying as an Init.d Service (Part I)

- Spring Boot builds an uber-JAR by default. So, the JAR it builds has all dependencies required for your application to run.
- It does this with the spring-boot-maven-plugin.
- If you ask it to make the JAR executable as well, it will be runnable and will support Linux init.d services (management with “service <name> start/stop/restart/status, auto-start with OS, etc.).

```
<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
  <configuration>
    <executable>true</executable>
  </configuration>
</plugin>
```

Deploying as an Init.d Service (Part II)

- **Once you've made a JAR executable, you can view it in a text editor and you will notice that there is literally a bash script at the top of your JAR above the binary section; it's pretty interesting to see.**
- **External configuration:**
  - Put a <jar-name>.conf file next to the JAR (or a sym-link to one).
  - Define your environment variables in it (e.g. JVM size, logging and property file locations, etc).
  - It will automatically be picked up at run time.

Live Coding Session Part #3

Integrating with the Admin Console

spring

Admin Console Server

- Monitors your spring-boot applications and lets you interact with them.
- Is a spring-boot application itself; include parent POM and a couple dependencies, set ports in the same way. Run as a separate application. http://codecentric.github.io/spring-boot-admin/1.5.0/#getting-started
- Add a client-dependency to the apps you want to monitor (like the one we just built) and set a property, and it works great! ☺

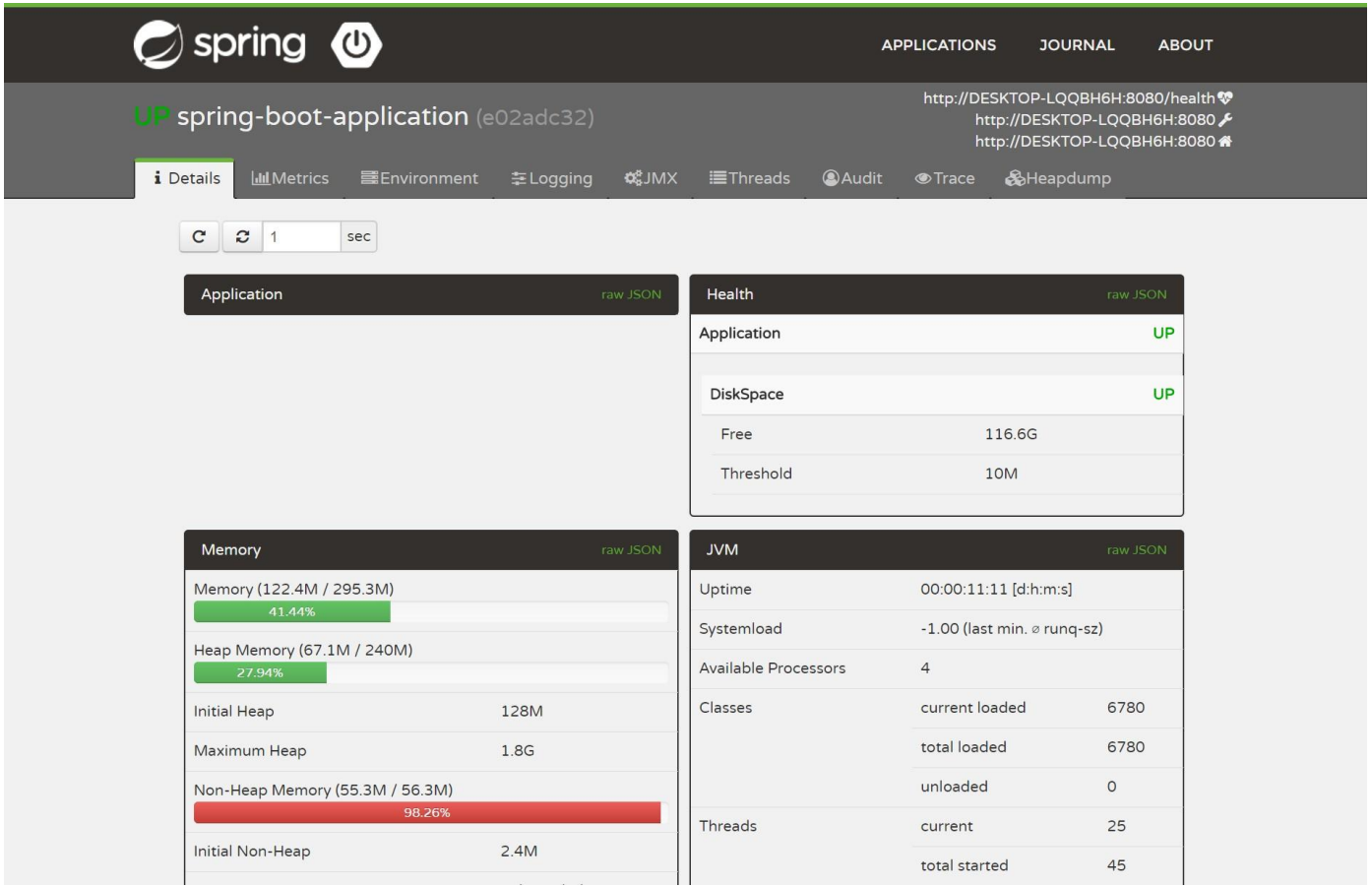
```xml
<dependency>
  <groupId>de.codecentric</groupId>
  <artifactId>spring-boot-admin-server-ui</artifactId>
  <version>1.5.0</version>
</dependency>
```

```
spring.boot.admin.url: http://localhost:8180
```


spring

Admin Console Server Features

- Dynamically change logging levels (e.g. add trace logging to debug a request temporarily and know how to fix it in dev).
- Perform JMX calls.
- View your application metrics.
- View web request and response pairs for your web-app (including headers).
- View the environment, stack traces, etc.
- Basically anything actuator exposes is beautifully represented here.
- Applications are automatically detected and you get a log of when they start up, shut down, etc.


spring
APPLICATIONS
JOURNAL
ABOUT
UP spring-boot-application (e02adc32)
http://DESKTOP-LQQBH6H:8080/health
http://DESKTOP-LQQBH6H:8080
http://DESKTOP-LQQBH6H:8080
Details
Metrics
Environment
Logging
JMX
Threads
Audit
Trace
Heapdump
1
sec
Application
raw JSON
Health
raw JSON
Application
UP
DiskSpace
UP
Free
116.6G
Threshold
10M
Memory
raw JSON
Memory (122.4M / 295.3M)
41.44%
Heap Memory (67.1M / 240M)
27.94%
Initial Heap
128M
Maximum Heap
1.8G
Non-Heap Memory (55.3M / 56.3M)
98.26%
Initial Non-Heap
2.4M
JVM
raw JSON
Uptime
00:00:11:11 [d:h:m:s]
Systemload
-1.00 (last min. ⌀ runq-sz)
Available Processors
4
Classes
current loaded
6780
total loaded
6780
unloaded
0
Threads
current
25
total started
45

spring