"Bootiful" Applications with Spring Boot

Dave Syer - Stéphane Nicoll

Pivotal

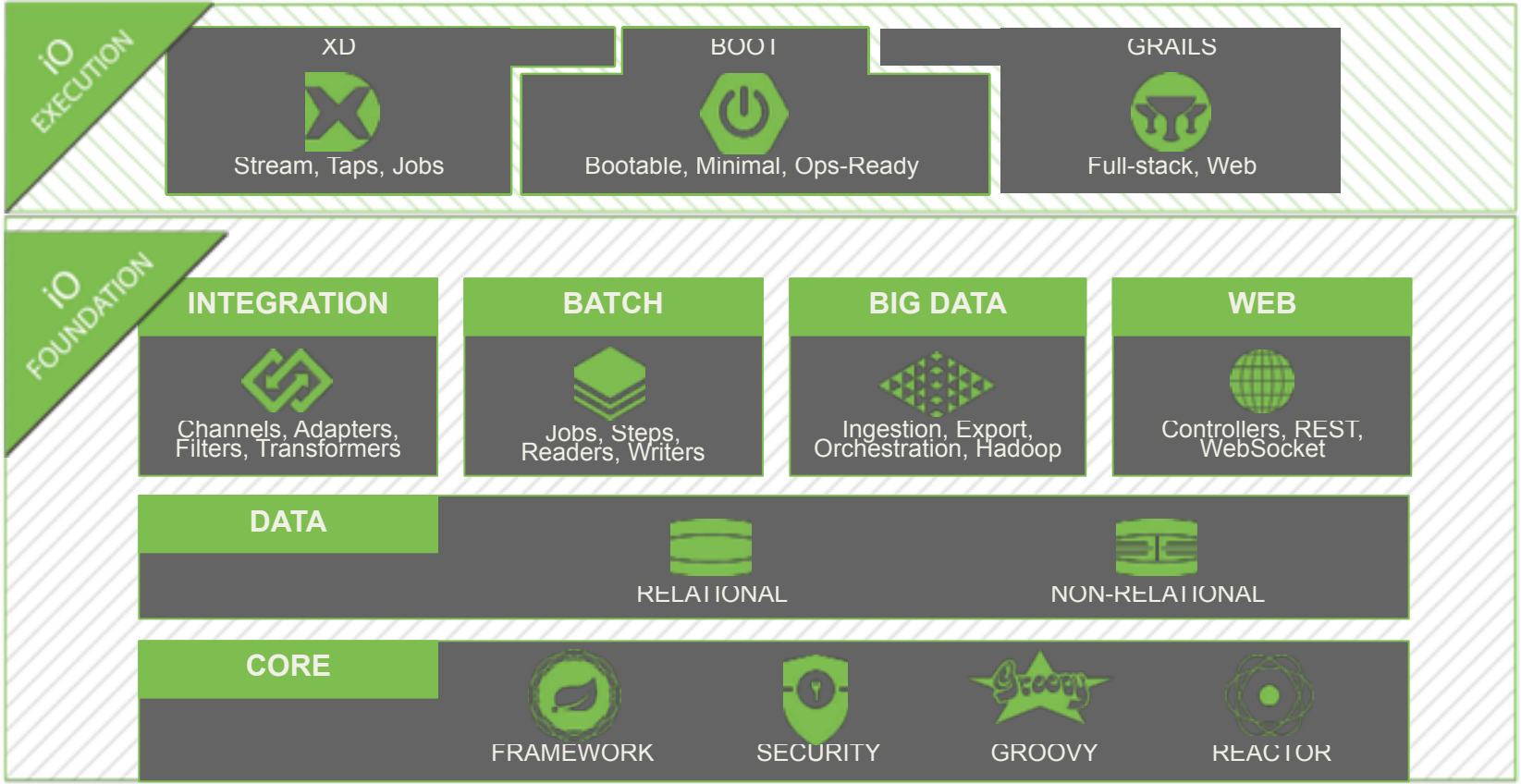
“Bootiful” Applications with Spring Boot

Dave Syer, Stéphane Nicoll

@david_syer @snicoll

[dsyer,snicoll]@pivotal.io

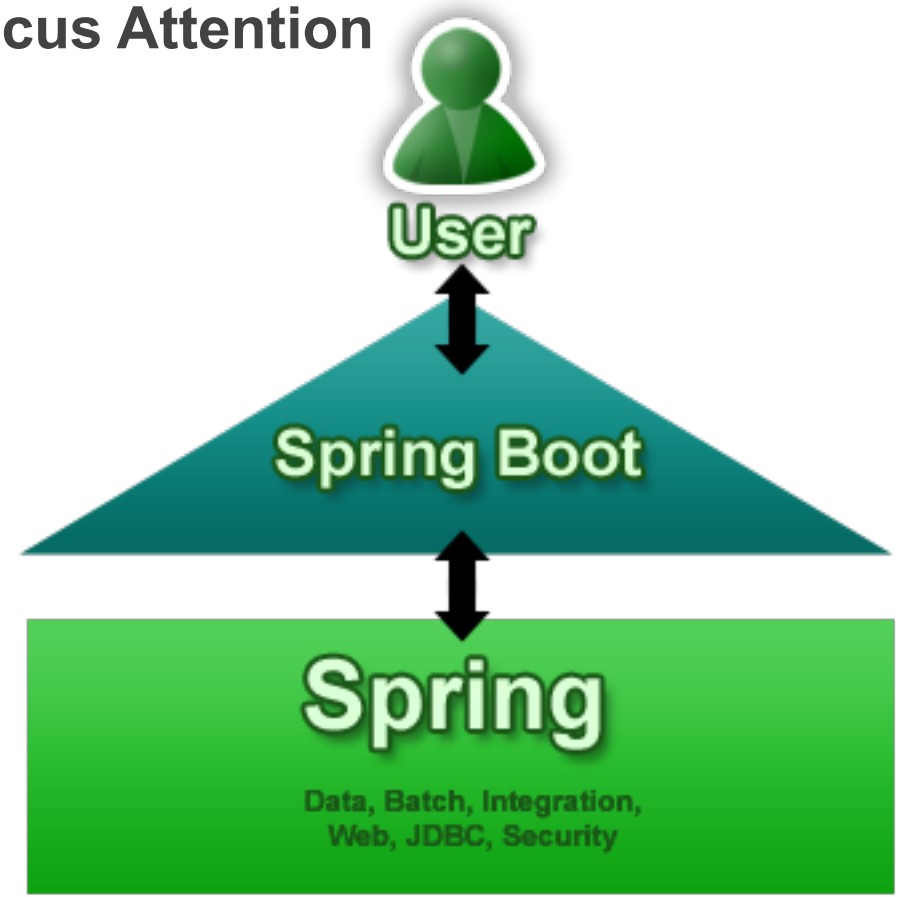
Spring IO platform



https://spring.io - Boot in production > 18 months!

github.com/spring-io/sagan

Spring Boot: Focus Attention



Introduction to Spring Boot

- Single point of focus (as opposed to large collection of `spring-*` projects)
- A tool for getting started very quickly with Spring
- Common non-functional requirements for a "real" application
- Exposes a lot of useful features by default
- Gets out of the way quickly if you want to change defaults
- An opportunity for Spring to be opinionated

> **Spring Boot lets you pair-program with the Spring team.**
>
> *Josh Long, @starbuxman*

ಠ_ಠ Spring Boot is NOT

- A prototyping tool
- Only for embedded container apps
- Sub-par Spring experience
- For Spring beginners only

Installation

- **Requirements:**
  - Java (>=1.6) + (for Java projects)
  - Maven 3.2+ or Gradle 1.12+
- **Spring Tool Suite has some nice features for Java projects**
- **Download: https://start.spring.io/spring.zip**
- **Unzip the distro (approx. 10MB), and find `bin/` directory**

```
$ spring –help
...
```

- **(You can also install Spring Boot CLI with gvm, brew or MacPorts)**

Getting Started *Really* Quickly

```
@RestController
class Example {

    @RequestMapping("/")
    String home() {
        'Hello world!'
    }
}
```

```
$ spring run app.groovy
```

... application is running at http://localhost:8080

What Just Happened?

```
import org.springframework.web.bind.annotation.RestController
// other imports ...

@RestController
class Example {

    @RequestMapping("/")
    public String hello() {
        return "Hello World!";
    }

}
```

What Just Happened?

```
import org.springframework.web.bind.annotation.RestController
// other imports ...

@Grab("org.springframework.boot:spring-boot-web-starter:1.2.1.RELEASE")
@RestController
class Example {

    @RequestMapping("/")
    public String hello() {
        return "Hello World!";
    }

}
```

What Just Happened?

```
import org.springframework.web.bind.annotation.RestController
// other imports ...

@Grab("org.springframework.boot:spring-boot-web-starter:1.2.1.RELEASE")
@EnableAutoConfiguration
@RestController
class Example {

    @RequestMapping("/")
    public String hello() {
        return "Hello World!";
    }
}
```

What Just Happened?

```
import org.springframework.web.bind.annotation.RestController
// other imports ...

@Grab("org.springframework.boot:spring-boot-web-starter:1.2.1.RELEASE")
@EnableAutoConfiguration
@RestController
class Example {

    @RequestMapping("/")
    public String hello() {
        return "Hello World!";
    }

    public static void main(String[] args) {
        SpringApplication.run(Example.class, args);
    }
}
```

Getting Started in Java

- Create a skeleton project on https://start.spring.io
- Choose the web option and download the project
- Add a simple controller alongside the app

```
@RestController
public class HomeController {

    @RequestMapping("/")
    public String home() {
        return "Hello World!";
    }
}
```

Starter POMs

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

- Standard Maven POMs
- Define dependencies that we recommend
- Parent optional
- Available for web, batch, integration, data, amqp, aop, jdbc, ...
- e.g. data-jpa = hibernate + spring-data-jpa + JSR 303

SpringApplication

```
SpringApplication app = new SpringApplication(MyApplication.class);
app.setShowBanner(false);
app.run(args);
```

- **Gets a running Spring** `ApplicationContext`
- **Uses** `EmbeddedWebApplicationContext` **for web apps**
- **Can be a single line**
  - `SpringApplication.run(MyApplication.class, args)`
- **Or customized via** `SpringApplicationBuilder`

SpringApplicationBuilder

- **Flexible builder style with fluent API for building `SpringApplication` with more complex requirements.**

```
new SpringApplicationBuilder(ParentConfiguration.class)
    .profiles("adminServer", "single")
    .child(AdminServerApplication.class)
    .run(args);
```

@EnableAutoConfiguration

```
@Configuration
@ComponentScan
@EnableAutoConfiguration
public class MyApplication {
}
```

```
@SpringBootApplication
public class MyApplication {
}
```

- **Attempts to auto-configure your application**
- **Backs off as you define your own beans**
- **Regular `@Configuration` classes**
- **Usually with `@ConditionalOnClass` and `@ConditionalOnMissingBean`**

Testing with Spring Test (and MVC)

- `spring-boot-starter-test` **provides useful test dependencies**
  - `spring-test`, `Mockito`, `Hamcrest` and `JUnit`
- `@SpringApplicationConfiguration`
  - Alternative to the standard spring-test `@ContextConfiguration`
  - Does not start the full context by default
- `@WebIntegrationTest`
  - Requires a web application context
  - Can add additional properties to the environment

Packaging For Production

- **Maven plugin (using spring-boot-starter-parent):**

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

```
$ mvn package
```

- **Gradle plugin:**

```
apply plugin: 'spring-boot'
```

```
$ gradle build
```

Packaging For Production

```
$ java -jar yourapp.jar
```

- Easy to understand structure
- No unpacking or start scripts required
- Typical REST app ~10Mb
- Cloud Foundry friendly (works & fast to upload)

Not a Web Application?

- `CommandLineRunner` **is a hook to run application-specific code after the context is created**

```
@Component
public class Startup implements CommandLineRunner {

    @Override
    public void run(String... args) throws Exception {
        System.out.println("Hello World");
    }

}
```

Environment and Profiles

- **Every `ApplicationContext` has an `Environment`**
- **Spring `Environment` available since 3.1**
- **Abstraction for key/value pairs from multiple sources**
- **Used to manage `@Profile` switching**
- **Always available: System properties and OS ENV vars**

Command Line Arguments

- `SpringApplication` **adds command line arguments to the Spring** `Environment` **so you can inject them into beans:**

```
@Value("${name}")
private String name;
```

```
$ java -jar yourapp.jar --name=BootDragon
```

- **You can also configure many aspects of Spring Boot itself:**

```
$ java -jar yourapp.jar --server.port=9000
```

Externalizing Configuration to Properties

- **Just put `application.properties` in one of the following locations:**
  - A `/config` sub-directory of the current directory
  - The current directory
  - A classpath `/config` package
  - The root classpath
- **Properties can be overridden**
  - command line arg > file > classpath
  - locations higher in the list override lower items

```
server.port=9000
name=BootDragon
```

Using YAML

- **Just include** `snake-yaml.jar`
  - Already available if you're using the starters
- **Write an** `application.yml` **file**

```
name: BootDragon
server:
  port: 9000
```

Binding Configuration To Beans

- MyProperties.java

```
@ConfigurationProperties(prefix="mine")
public class MyProperties {
    private Resource location;
    private boolean skip = true;
    // ... getters and setters
}
```

- application.properties

```
mine.location=classpath:mine.xml
mine.skip=false
```

Data Binding to `@ConfigurationProperties`

- **Spring `DataBinder` does type coercion and conversion where possible**
- **Custom `ConversionService` additionally discovered by bean name (same as `ApplicationContext`)**
- **Ditto for validation**
  - `configurationPropertiesValidator` bean if present
  - JSR303 if present
  - `ignoreUnkownFields=true` (default)
  - `ignoreInvalidFields=false` (default)
- **Uses a `RelaxedDataBinder` which accepts common variants of property names (e.g. CAPITALIZED, camelCased or with_underscores)**

Configuration Meta-data

- **Annotation processor**

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-configuration-processor</artifactId>
    <optional>true</optional>
</dependency>
```

- **Generates a meta-data file while compiling your project**
  - Javadoc on fields are translated to descriptions
  - Default values are detected (to some extend)
  - Additional meta-data can be provided for corner cases
    - `META-INF/additional-spring-configuration-metadata.json`

Customizing Configuration Location

- **Set**
  - `spring.config.name` - default **`application`**, can be comma-separated list
  - `spring.config.location` - a `Resource` path
    - Ends with / to define a directory
    - Otherwise overrides name

```
$ java -jar app.jar --spring.config.name=production
$ java -jar app.jar --spring.config.location=classpath:/cfg/
$ java -jar app.jar --spring.config.location=classpath:/cfg.yml
```

Spring Profiles

- **Activate external configuration with a Spring profile**
  - file name convention e.g. `application-development.properties`
  - or nested documents in YAML:

```
server:
    address: 192.168.1.100
---
spring:
    profiles: development
server:
    address: 127.0.0.1
---
spring:
    profiles: production
server:
    address: 192.168.1.120
```

Spring Profiles

- Set the default spring profile(s) in external configuration

```
spring.profiles.active=default, postgresql
```

```
$ java -jar yourapp.jar –spring.profiles.active=production
```

- Add some profile(s) to the active profiles rather than replacing them

```
spring.profiles.include=another
```

Adding some Autoconfigured Behavior

- **Extend the demo and see what we can get by just modifying the class path**
  - Create a simple domain object
  - Expose the repository as a REST endpoint

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-rest</artifactId>
</dependency>
```

Logging

- **Spring Boot provides default configuration files for 4 logging frameworks: Logback, Log4j, Log4j2 and `java.util.Logging`**
- **Starters (and Samples) use Logback with colour output**
- **Default log level set to INFO**
  - Debug output can be easily enabled using the `--debug` option
- **Log to console by default**
  - `logging.file` and `logging.path` to enable file logging
- **Logging levels can be customised through configuration**

```
logging.level.org.acme=TRACE
```

Add static resources

- **Easiest: use** `classpath:/static/**`
- **Many alternatives:**
  - `classpath:/public/**`
  - `classpath:/resources/**`
  - `classpath:/META-INF/resources/**`
- **Normal servlet context `/` (root of WAR file, see later)**
  - i.e. `src/main/webapp`
  - `static/**`
  - `public/**`
  - set documentRoot in `EmbeddedServletContextFactory`

Web template engines

- **Spring Boot includes auto-configuration support for *Thymeleaf*, *Groovy*, *FreeMarker, Velocity* and *Mustache***
- **By default, templates will be picked up automatically from `classpath:/templates`**
- **Common configuration, e.g. for Thymeleaf**
  - `spring.thymeleaf.prefix` (location of templates)
  - `spring.thymeleaf.cache` (set to `false` to live reload templates)
- Extend and override, just add beans:
  - `thymeleafViewResolver`
  - `SpringTemplateEngine`

Error handling

- `/error` **handles all errors in a sensible way**
  - Registered as global *error page* in the servlet container
  - Add a view that resolve to ‘error’ to customize the representation
- **Default representation**
  - Whitelabel error page for browser if none is found
  - Standardized JSON format for machine clients
- **Customize or extend `ErrorAttributes`**
- **Create dedicated error pages via** `EmbeddedServletContainerCustomizer`

Adding some Autoconfigured Behavior

- **Secure the web application**
  - Application endpoints secured via `security.basic.enabled=true` (on by default)
- **See how you can ask Boot to back off**
  - Configure a custom AuthenticationManager

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

Currently Available Autoconfigured Behaviour

- Embedded servlet container (Tomcat, Jetty or Undertow)
- DataSource (Tomcat, Hikari, Commons DBCP)
- SQL and NoSQL stores: Spring Data JPA, MongoDB and Redis
- Messaging: JMS (HornetQ, ActiveMQ), AMQP (Rabbit)
- Thymeleaf, Groovy templates, Freemarker, Mustache and Velocity
- Batch processing - Spring Integration
- Cloud connectors
- Rest repositories
- Spring Security

Currently Available Autoconfigured Behaviour

- Data grid: Spring Data Gemfire, Solr and Elasticsearch
- Websocket
- Web services
- Mobile & Social (Facebook, Twitter and LinkedIn)
- Reactor for events and async processing
- Jersey
- JTA
- Email, CRaSH, AOP (AspectJ)
- Actuator features (Security, Audit, Metrics, Trace)

The Actuator

- **Adds common non-functional features to your application and exposes endpoints to interact with them (REST, JMX)**
  - Secure endpoints: `/env`, `/metrics`, `/trace`, `/dump`, `/shutdown`, `/beans`, `/autoconfig`, `/configprops`, `/mappings`
  - `/info`
  - `/health`
  - Audit

If embedded in a web app or web service can use the same port or a different one (`management.port`) and/or a different network interface (`management.address`) and/or context path (`management.context-path`).

Add a remote SSH server

- Add spring-boot-starter-remote-shell to class path
- Application exposed to SSH on port 2000 by default

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

Building a WAR

We like launchable JARs, but you can still use WAR format if you prefer. Spring Boot Tools take care of repackaging a WAR to make it executable. If you want a WAR to be deployable (in a "normal" container), then you need to use `SpringBootServletInitializer` instead of or as well as SpringApplication.

```
public class ServletInitializer
        extends SpringBootServletInitializer {

    @Override
    protected SpringApplicationBuilder configure(
            SpringApplicationBuilder application) {

        return application.sources(MyApplication.class);
    }
}
```

Customizing the Servlet Container

- **Some common features exposed with external configuration, e.g.** `server.port` **(see** `ServerProperties` **bean)**
  - Also container-specific properties, i.e. `server.tomcat.*`
- **Add bean(s) of type** `EmbeddedServletContainerCustomizer`
  - all instances get a callback to the container
- **Add bean of type** `EmbeddedServletContainerFactory` **(replacing auto-configured one)**

Customizing @EnableAutoConfiguration

- **Disable specific feature**
  - `@EnableAutoConfiguration(exclude={WebMvcAutoConfiguration.class})`
  - `@SpringBootApplication(exclude={WebMvcAutoConfiguration.class})`
- **Write your own…**
  - Create your own `@Configuration` class
  - Add the FQN of your configuration class in `META-INF/spring.factories`
  - All entries from classpath merged and added to context

Customizing the CLI

- **Uses standard Java `META-INF/services` scanning**
- `CompilerAutoConfiguration`**: add dependencies and imports based on matches in the code**
- `CommandFactory`**: add additional commands**
  - **name and description**
  - **usage help**
  - **actual execution based on command line arguments**

Links

- Documentation: http://projects.spring.io/spring-boot
- Source: https://github.com/spring-projects/spring-boot
- Blog: http://spring.io/blog
- Twitter: @SpringBoot, @david_syer, @snicoll
- Email: dsyer@pivotal.io, snicoll@pivotal.io