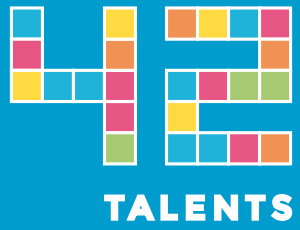
Demystifying Spring Boot Magic

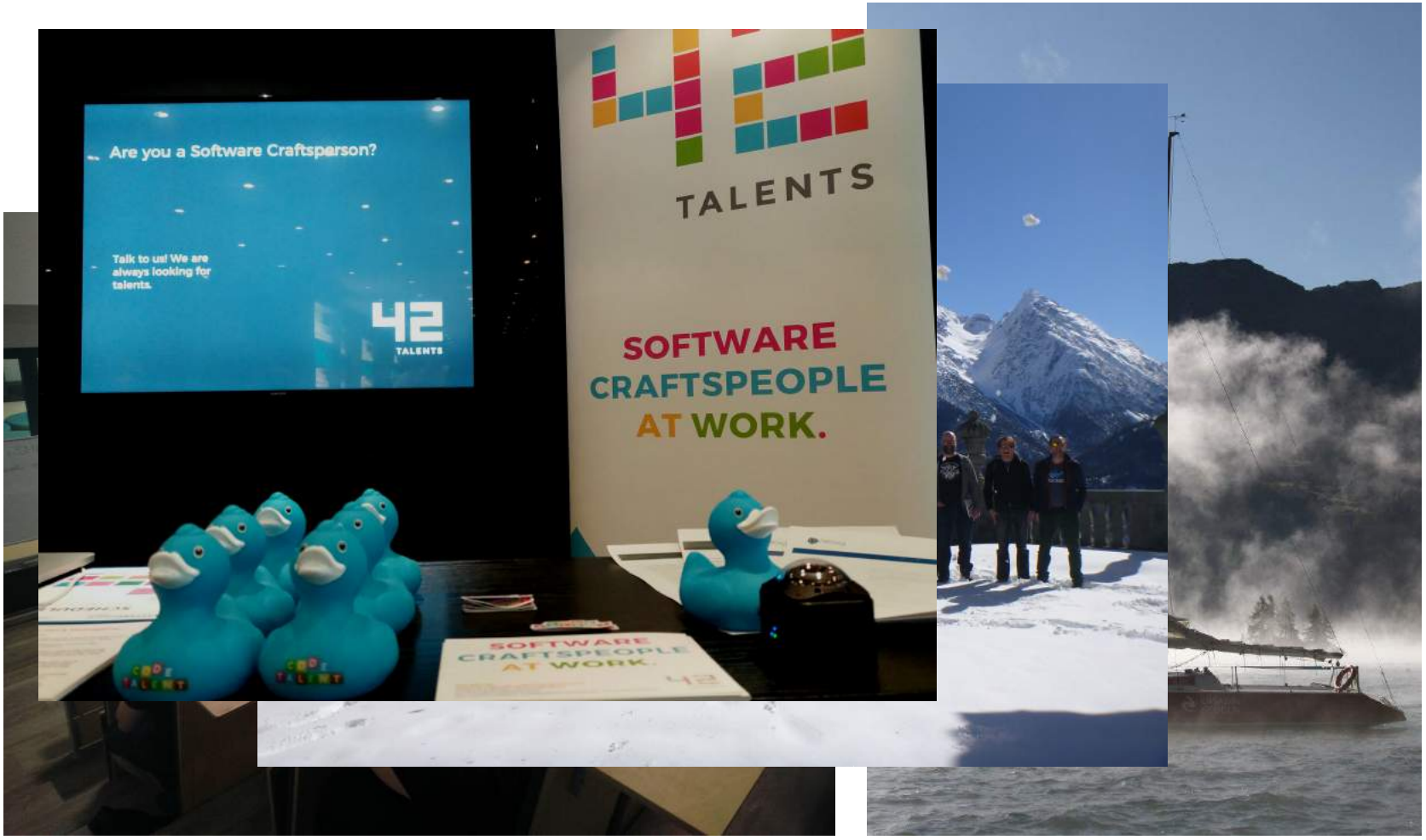
K100 on slido.co

Patrick Baumgartner | @patbaumgartner
DevDays Vilnius 2018
Socialiniu mokslu kolegija / 24.05.2018



Who am I?

Patrick Baumgartner



Spring Boot And Magic?

Some famous tweets...

Magic or Boilerplate?

Magic

- Things happen at runtime that I cannot immediately explain just by reading the code.
- The difference between “not magic” and “magic” is often just very thin line - A measure could be: “How big is the surprise?”.
- Violating the “Single Responsibility Principle” or “Command / Query Separation” looks like magic!

Boilerplate

- Boilerplate code is code that you have to write over and over.
- You will write it in every application you code. Sometimes you write it even multiple times in the same application.
- It’s the boring code, that nobody likes to write.

Magic or Boilerplate?

Magic or Boilerplate?

- Magic and boilerplate are both subjective.
- Magic and boilerplate are not necessarily bad.
- Boilerplate is not the opposite of magic!

Too Much of Both...

Too Much Magic

“Boilerplate is mostly a problem when writing code. Magic when reading & debugging. We read & debug more than we write.”

David Tanzer - @dtanzer

Magic becomes Knowledge

"... is to much magic until you read the docs, several books and the source code, after that, the magic becomes knowledge."

u/ramses79

Spring Boot

Spring Boot

- Takes an opinionated view of building production-ready Spring applications.
- Spring Boot favors convention over configuration and is designed to get you up and running as quickly as possible.
- Spring Boot helps developers to “boot” up Spring applications faster.

Auto Configuration

Auto Configuration

- Or convention over configuration.
  - Too much magic going on.

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

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

@ComponentScan

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

- @Component
- @Service
- @Repository
- @Controller -> @RestController
- @Configuration
- …

@ComponentScan

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

- com.patbaumgartner.app
  - com.patbaumgartner.app.one
  - com.patbaumgartner.app.two
  - com.patbaumgartner.app.three

- com.patbaumgartner.stuff 

META-INF/spring.factories

```
# Auto Configure
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
com.patbaumgartner.autoconfigure.HelloAutoConfiguration
```

```
@Configuration
@ConditionalOnClass(HelloService.class)
public class HelloAutoConfiguration {

    @Bean
    @ConditionalOnMissingBean
    public HelloService helloService() {
        return new ConsoleHelloService();
    }
}
```

Externalized configuration

Externalized configuration

- This feature that tends to be overlooked by newcomers and it can be very powerful.
- It is very flexible and well documented. For instance, you can choose to either use
  - `ENVIRONMENT_VARIABLES`
  - `--command-line-arguments`
  - `-Dsystem.properties`
  - Spring Cloud Config Server

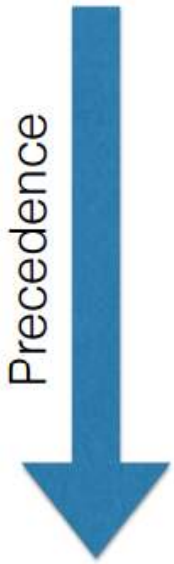
```
@Value("${cassandra.password}")
private String password;
```

@ConfigurationProperties

- Property binding with prefix on Spring Beans

```
@ConfigurationProperties(prefix = "tenants")
public class RoutingDataSourceTenantProperties {

private List<String> tenants;

    public List<String> getTenants() {
        return tenants;
    }

}
```

Property Precedence



Config Server	key1=**kiwi** key2=**flamingo** key3=**dolphin**
Command line arguments	key1=apple
System Properties	key2=dragon
System Environment	key1=banana
classpath: application.yml	key1=coconut key2=elephant
classpath: bootstrap.yml	key3=shark

E.g.: Command-line parameters

```
// Maven
// Spring Boot 1.x:
$ mvn spring-boot:run -Drun.arguments=--server.port=8085

// Spring Boot 2.x:
$ mvn spring-boot:run -Dspring-boot.run.arguments=--server.port=8085


// Gradle
// bootRun task in build.gradle file

bootRun {
    if (project.hasProperty('args')) {
        args project.args.split(',')
    }
}

// CLI
$ ./gradlew bootRun -Pargs=--spring.main.banner-mode=off,--server.port=8181


// Shell
$ java -jar command-line.jar --server.port=9090
```

E.g.: Environment – docker-compose

```
version: '3'

networks:
  app-integration:

services:

  app-service-account:
    image: docker.patbaumgartner.com/patbaumgartner/app-service-account:5.0.0-SNAPSHOT
    container_name: app-service-account
    restart: unless-stopped
    networks:
      - app-integration
    ports:
      - "9010"
    depends_on:
      - app-service-config-node1
    environment:
      DEBUG: 'true'
      SPRING_PROFILES_ACTIVE: integration
      SPRING_ZIPKIN_BASEURL: http://app-service-zipkin:9411/
      SPRING_CLOUD_CONFIG_URI: http://admin:admin@app-service-config-node1:8888
      EUREKA_CLIENT_SERVICE_URL_defaultZone: http://app-service-eureka-node1:8761/eureka/
```

Profiles

@Profile

- Used to limit the availability of `@Component` or `@Configuration`
- With one or more parameters
  - `@Profile("prod")` or `@Profile({"dev", "prod"})`
- Negation possible or even hierarchies
  - `@Profile("!dev")`

```
@Profile("!dev")
@Configuration
public class PrdConfiguration{…}
```

```
@Service
@Profile("dev")
public class DevEmployeeService
    implements EmployeeService {…}
```

Selecting Profiles

- Command Line

```
$ java -Dspring.profiles.active=dev -jar command-line.jar
```

- Maven Plugin

```
$ mvn spring-boot:run -Dspring-boot.run.profiles=dev,staging
$ SPRING_PROFILES_ACTIVE=production mvn spring-boot:run
```

- Property File (`application.properties` or `application.yaml`)

```
spring.profiles.active=dev,staging
```

- Programmatically setting profile

```
public static void main(String[] args) {
    SpringApplication app =
        new SpringApplication(DemoApplication.class);
    app.setAdditionalProfiles("dev","staging");
}
```

Profile Specific Configurations

- `application-{profile}.properties`
- Loaded the from the same location as `application.properties`
- Will override default `application.properties`
- `application-staging.properties`

```
db.driver=oracle.jdbc.driver.OracleDriver
db.username=<username>
db.password=<password>
db.tableprefix=
```

- `application-dev.properties`

```
db.url=jdbc:hsqldb:file:configurations
db.driver=org.hsqldb.jdbcDriver
db.username=sa
db.password=
db.tableprefix=
```

Property Files

- Loading and overriding in a defined order.

```
application-dev.properties
application-prod.properties
application-default.properties
application.properties
```

```
application-dev.yml
application-prod.yml
application-default.yml
application.yml
```

YAML and Profiles

- Uses SnakeYAML to parse
- A YAML file might contain more than one profile
- Default profile not named

```yaml
# default profile
server:
 port:9000
---
spring:
 profiles:dev
server:
 port:8000
---
spring:
 profiles:staging
server:
 port:8080
```

Creating a Custom Spring Boot Starter

Creating a Custom Starter

Creating a Custom Starter

- We need the following components:
  - An auto-configure class for our library along with a properties class for custom configuration.
  - A starter *`pom`* to bring in the dependencies of the library and the autoconfigure project.

The Autoconfigure Module

The Autoconfigure Module

- Custom module should be named like
  - `{module}-spring-boot-autoconfigure`

- Contains
  - Java classes
    - `{Module}Properties` which will enable setting custom properties through `application.properties`
    - `{Module}AutoConfiguration` which will create the spring beans
  - `spring.factories` *in META-INF*
    - *Pointing to the* `{Module}AutoConfiguration`
  - `spring-boot-configuration-processor`
    - *Generating IDE support for auto completion*

@ConfigurationProperties

- Used to make properties from `application.properties` accessible in the code

```
@Data
@ConfigurationProperties(prefix = "javafaker")
public class JavaFakerProperties {

    /**
    * Locale in the format of de-CH, higher precendence over language.
    */
    private String locale;

    /**
    * Language in the 2 character format like de.
    */
    private String language;
}
```

```
javafaker.language=fr
javafaker.locale=de-CH
```

`application.properties` from the business application

META-INF/spring.factories

- While startup, Spring Boot looks for all files named `spring.factories` in the classpath to trigger auto configuration
- File is located in the `META-INF` directory
- E.g. from the `javafaker-boot-autoconfigure` project

```
# Auto Configure
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
com.patbaumgartner.javafaker.autoconfigure.JavaFakerAutoConfiguration
```

AutoConfiguration

- Configuration with `@Conditional`
  - OnClassCondition / OnMissingClassCondition
  - OnBeanCondition / OnMissingBeanCondition
  - OnPropertyCondition
  - OnResourceCondition
  - OnExpressionCondition
  - OnJavaCondition
  - OnJndiCondition
  - OnWebApplicationCondition
  - OnCloudPlatformCondition
  - …

AutoConfiguration

```java
@Configuration
@ConditionalOnClass(Faker.class)
@EnableConfigurationProperties(JavaFakerProperties.class)
public class JavaFakerAutoConfiguration {

    @Autowired
    private JavaFakerProperties fakerProperties;

    @Bean
    @ConditionalOnMissingBean
    public Faker faker() {
        if (!StringUtils.isEmpty(fakerProperties.getLocale())) {
            return Faker.instance(new Locale(fakerProperties.getLocale()));
        }

        if (!StringUtils.isEmpty(fakerProperties.getLanguage())) {
            return Faker.instance(new Locale(fakerProperties.getLanguage()));
        }

        return Faker.instance();
    }
}
```

spring-boot-configuration-processor

- Maven dependency generates `META-INF/spring-configuration-metadata.json`

```
{
"hints": [],
"groups": [
      {
      "sourceType": "com.patbaumgartner.javafaker.autoconfigure.JavaFakerProperties",
      "name": "javafaker",
      "type": "com.patbaumgartner.javafaker.autoconfigure.JavaFakerProperties"
      }
],
"properties": [
      {
      "sourceType": "com.patbaumgartner.javafaker.autoconfigure.JavaFakerProperties",
      "name": "javafaker.language",
      "description": "language in the 2 character format like de.",
      "type": "java.lang.String"
      },
      {
      "sourceType": "com.patbaumgartner.javafaker.autoconfigure.JavaFakerProperties",
      "name": "javafaker.locale",
      "description": "Locale in the format of de-CH, higher precedence over language.
      "type": "java.lang.String"
      }
]}
```

The Starter Module

pom.xml

```
<project>
    <groupId>com.patbaumgartner</groupId>
    <artifactId>javafaker-spring-boot-starter</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
            <version>${spring-boot.version}</version>
        </dependency>
        <dependency>
            <groupId>com.patbaumgartner</groupId>
            <artifactId>javafaker-spring-boot-autoconfigure</artifactId>
            <version>${project.version}</version>
            </dependency>
        <dependency>
            <groupId>com.github.javafaker</groupId>
            <artifactId>javafaker</artifactId>
            <version>${java-faker.version}</version>
        </dependency>
    </dependencies>
</project>
```

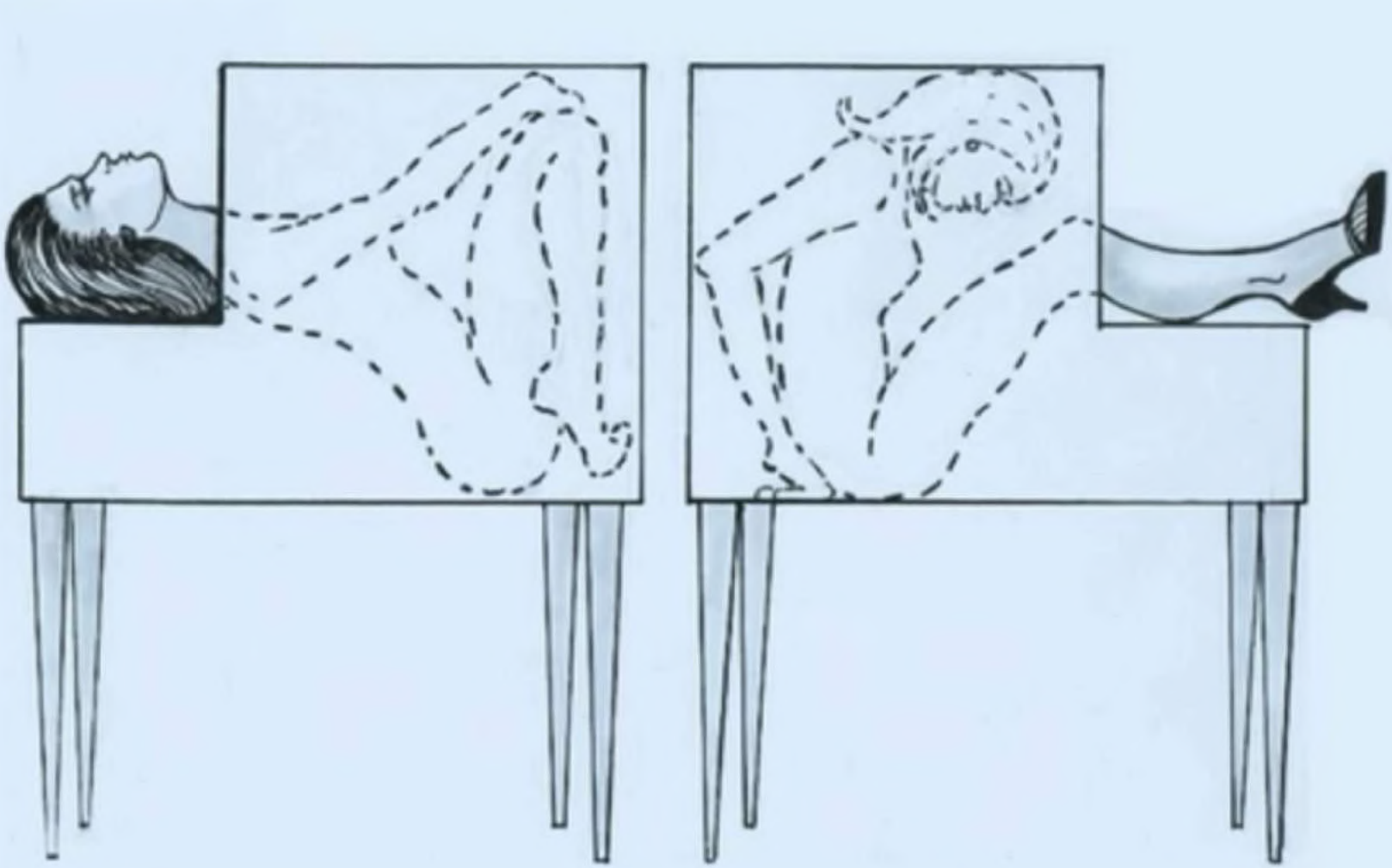
Using the starter

Using the starter

```
<dependency>
    <groupId>com.patbaumgartner</groupId>
    <artifactId>javafaker-spring-boot-starter</artifactId>
    <version>0.0.1-SNAPSHOT</version>
</dependency>
```

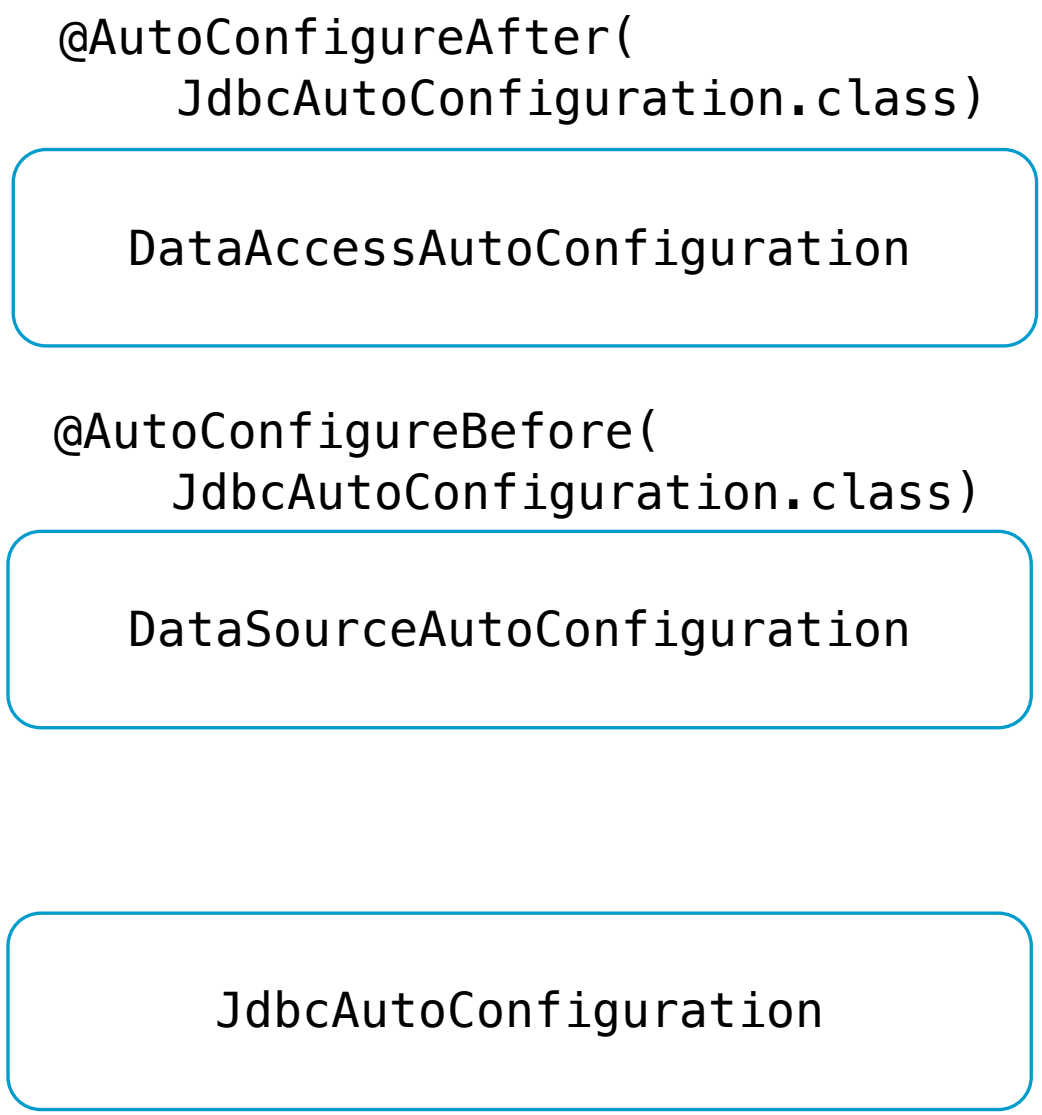
```
javafaker.locale=de-CH
```

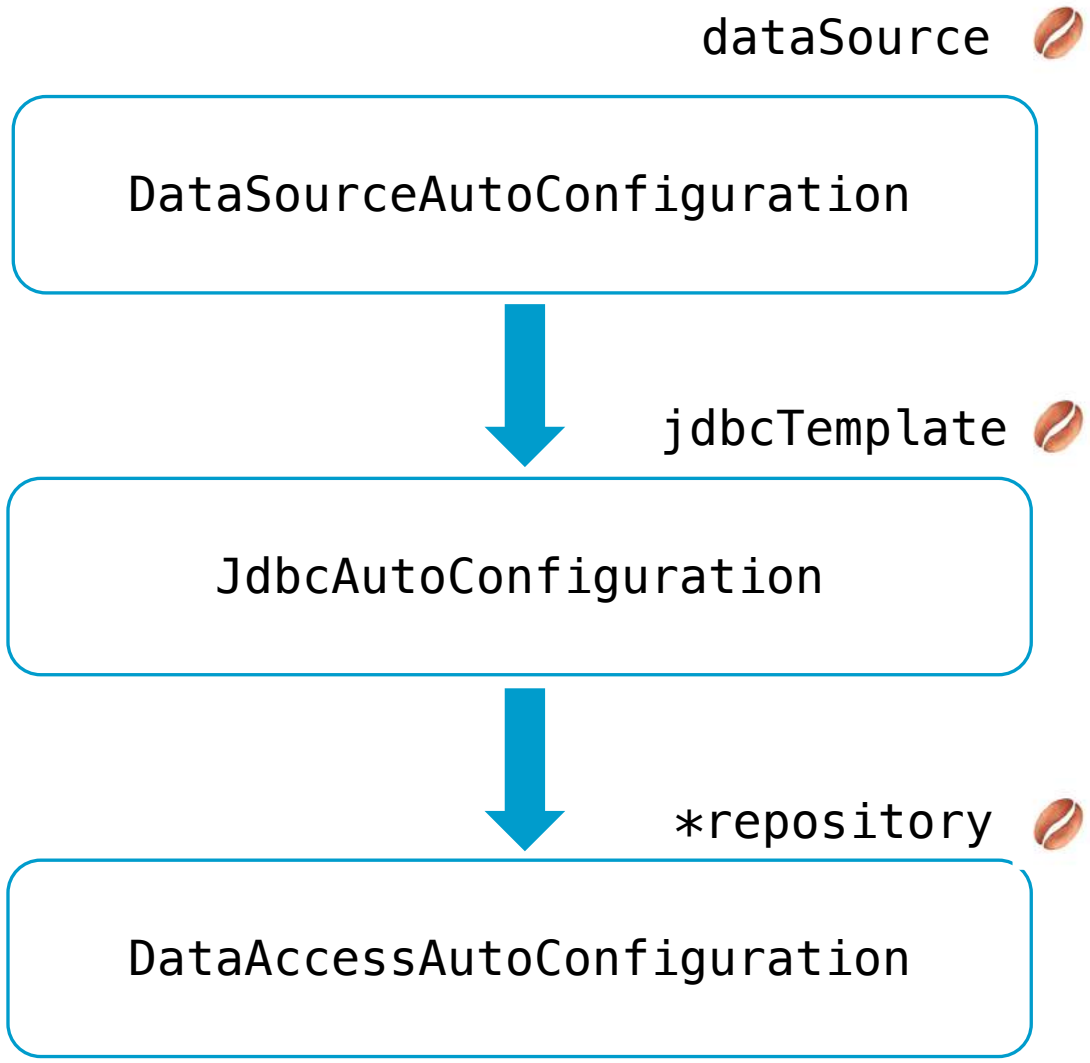
```
@Slf4j
@SpringBootApplication
public class JavafakerApplication implements CommandLineRunner {

    @Autowired
    private Faker faker;

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



Tools

Ordering Configuration Classes


@AutoConfigureAfter(
JdbcAutoConfiguration.class)
DataAccessAutoConfiguration
@AutoConfigureBefore(
JdbcAutoConfiguration.class)
DataSourceAutoConfiguration
JdbcAutoConfiguration


dataSource
DataSourceAutoConfiguration
jdbcTemplate
JdbcAutoConfiguration
*repository
DataAccessAutoConfiguration

FailureAnalyzer

```
at org.springframework.beans.factory.support.DefaultListableBeanFactory.doResolveDepe...
at org.springframework.beans.factory.support.DefaultListableBeanFactory.resolveDepend...
at org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor$...
... 226 common frames omitted
Caused by: org.springframework.beans.factory.BeanCreationException: Could not autowire
fi...
at org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor$...
at org.springframework.beans.factory.annotation.InjectionMetadata.inject(InjectionMet...
at org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor....
... 246 common frames omitted
Caused by: org.springframework.beans.factory.BeanCurrentlyInCreationException: Error
crea...
at org.springframework.beans.factory.support.DefaultSingletonBeanRegistry.beforeSingl...
at org.springframework.beans.factory.support.DefaultSingletonBeanRegistry.getSingleto...
at org.springframework.beans.factory.support.AbstractBeanFactory.doGetBean(AbstractBe...
at org.springframework.beans.factory.support.AbstractBeanFactory.getBean(AbstractBean...
at org.springframework.beans.factory.support.DefaultListableBeanFactory.findAutowireC...
at org.springframework.beans.factory.support.DefaultListableBeanFactory.doResolveDepe...
at org.springframework.beans.factory.support.DefaultListableBeanFactory.resolveDepend...
at org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor$...
... 248 common frames omitted
```

FailureAnalyzer

- Special handling of Exceptions during the start-up of the application. E.g. `NoSuchBeanDefinition, Bind, BeanCurrentlyInCreation, PortInUse, DataSourceBeanCreation`, etc.
- Write your own `CustomFailureAnalyzer`

```
***************************
APPLICATION FAILED TO START
***************************
Description:
Embedded servlet container failed to start. Port 8080 was
already in use.
Action:
Identify and stop the process that's listening on port 8080 or
configure this application to listen on another port.
```

JARs

Fat JARs

- Fat jars aren't a new thing
  - Uber-JAR / One-JAR
  - Apache Maven Shade Plugin

- Spring Boot Maven Plugin

```
<plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
    <version>2.0.2.RELEASE</version>
    <executions>
        <execution>
            <goals>
                <goal>repackage</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

Exploded Fat JARs

- Fat JARs – An anti-pattern in a Docker world
- Fat layers instead of thin layers
- Take advantage of caching e.g. less network traffic

```
$ du -sh target/demo-0.0.1-SNAPSHOT*
14M  target/demo-0.0.1-SNAPSHOT.jar
4.0K target/demo-0.0.1-SNAPSHOT.jar.original
```

```
$ du -sh target/demo-0.0.1-SNAPSHOT* target/thin
12K  target/demo-0.0.1-SNAPSHOT.jar
4.0K target/demo-0.0.1-SNAPSHOT.jar.original
14M  target/thin
```

```
<plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot.experimental</groupId>
            <artifactId>spring-boot-thin-layout</artifactId>
            <version>${spring-boot-thin-layout.version}</version>
        </dependency>
    </dependencies>
</plugin>
```

```
<plugin>
    <groupId>org.springframework.boot.experimental</groupId>
    <artifactId>spring-boot-thin-maven-plugin</artifactId>
    <version>${spring-boot-thin-layout.version}</version>
    <executions>
        <execution>
            <id>resolve</id>
            <goals>
                <goal>resolve</goal>
            </goals>
            <inherited>false</inherited>
        </execution>
    </executions>
</plugin>
```

Actuator


SNAKE
OIL

Metrics from within your Application

- Adds several production grade services to your application.
- Sensors provide information from within our application.
- Almost no effort for developers ☺
- Uses `micrometer.io` which is the SLF4j for Metrics
- Very simple to add your own!

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

Actuator Endpoints

/actuator/{endpoint}

`auditevents`	Exposes audit events information for the current application.	Yes
`beans`	Displays a complete list of all the Spring beans in your application.	Yes
`conditions`	Shows the conditions that were evaluated on configuration and auto-configuration classes and the reasons why they did or did not match.	Yes
`configprops`	Displays a collated list of all @ConfigurationProperties.	Yes
`env`	Exposes properties from Spring's ConfigurableEnvironment.	Yes
`flyway`	Shows any Flyway database migrations that have been applied.	Yes
`health`	Shows application health information.	Yes
`httptrace`	Displays HTTP trace information (by default, the last 100 HTTP request-response exchanges).	Yes
`info`	Displays arbitrary application info.	Yes
`loggers`	Shows and modifies the configuration of loggers in the application.	Yes
`liquibase`	Shows any Liquibase database migrations that have been applied.	Yes
`metrics`	Shows 'metrics' information for the current application.	Yes
`mappings`	Displays a collated list of all @RequestMapping paths.	Yes
`scheduledtasks`	Displays the scheduled tasks in your application.	Yes
`sessions`	Allows retrieval and deletion of user sessions from a Spring Session-backed session store. Not available when using Spring Session's support for reactive web applications.	Yes
`shutdown`	Lets the application be gracefully shutdown.	No
`threaddump`	Performs a thread dump.	Yes

```java
@Endpoint(id = "loggers")
public class LoggersEndpoint {

    @ReadOperation
    public Map<String, Object> loggers() { ... }

    @ReadOperation
    public LoggerLevels loggerLevels(@Selector String name) {
       …
    }

    @WriteOperation
    public void configureLogLevel(@Selector String name,
       LogLevel configuredLevel) {
       …
    }

}
```

```
/actuator/loggers
/actuator/loggers/{name}
```

```
{
    "configuredLevel": "WARN"
}
```

Git Commit Id

git-commit-id-plugin

- Seeing what revision of your application you're running
- Spring Actuator, the exact git commit information
- Exposes it through the /info actuator endpoint

```
<plugin>
    <groupId>pl.project13.maven</groupId>
    <artifactId>git-commit-id-plugin</artifactId>
</plugin>
```

Feign

Feign from Spring Cloud

- Feign is an HTTP client from Netflix. Integrates closely with a discovery client like Eureka.
- It's like Spring-Data but for HTTP.
- Creates the client implementation with a proxy when using `@EnableFeignClients` on a configuration class.

```
@FeignClient(name = "bookStoreClient", url = "http://mybookstore.com")
public interface BookStoreClient {

    @GetMapping(path = "getBooks")
    public List<Book> getBooks();

    @PostMapping(path = "buyBook")
    @Headers("Content-Type: application/json")
    public Book buyBook(@RequestBody Book book);
}
```

Spring Boot Developer Tools

DevTools

- Provides property defaults. E.g. overrides caching
- Automatic restart / reloads when a files changes
- Activates and embedded Live Reload server
- Allows global settings `.spring-boot-devtools.properties` to `${user.home}`
- Remote update and debugging via HTTP

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

DevTools

Project naming

- When creating a *demo* project called `spring-boot`, dependencies can not be properly autowired.
  - Proxy does not contain interface
  - Only applies when DevTools added
- At first not explainable – Is this magic?!

DevTools

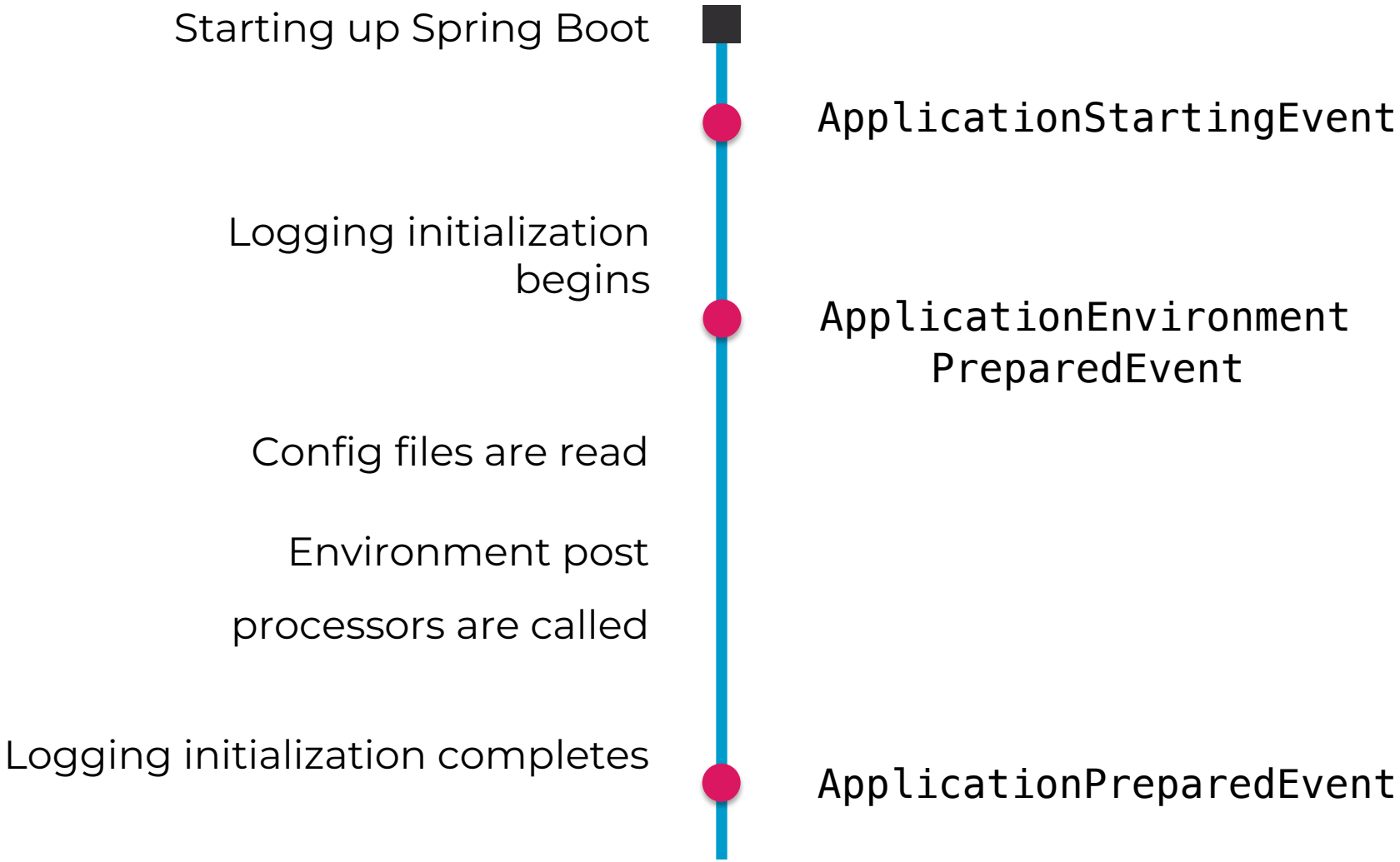
Project naming

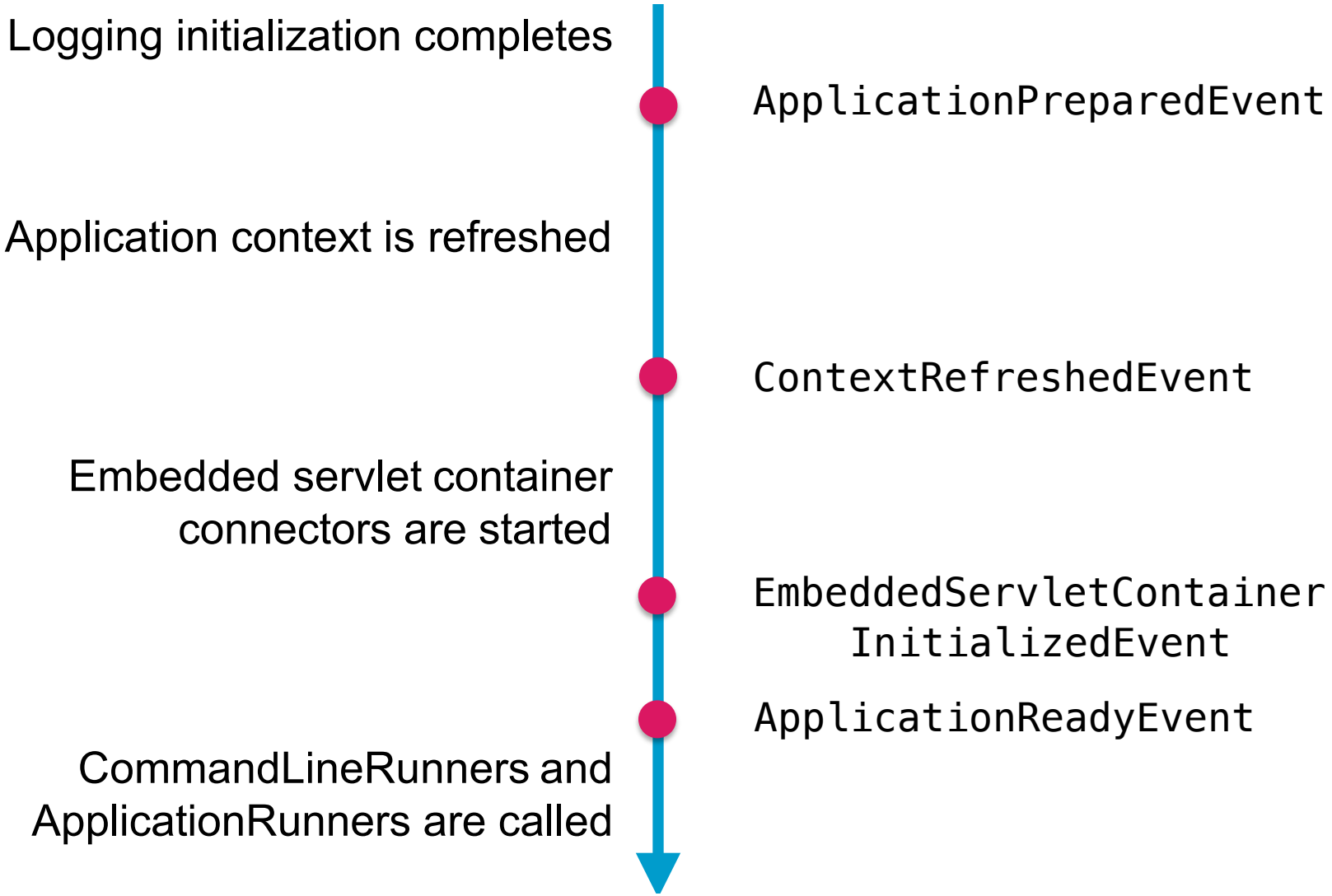
- The documentation states the following:

> When deciding if an entry on the classpath should trigger a restart when it changes, DevTools automatically ignores projects named `spring-boot`, `spring-boot-devtools`, `spring-boot-autoconfigure`, `spring-boot-actuator`, and `spring-boot-starter`.

https://docs.spring.io/spring-boot/docs/current/reference/html/using-boot-devtools.html

Spring Boot Events


Starting up Spring Boot
ApplicationStartingEvent
Logging initialization begins
ApplicationEnvironment PreparedEvent
Config files are read
Environment post processors are called
Logging initialization completes
ApplicationPreparedEvent


Logging initialization completes
ApplicationPreparedEvent
Application context is refreshed
ContextRefreshedEvent
Embedded servlet container connectors are started
EmbeddedServletContainer InitializedEvent
ApplicationReadyEvent
CommandLineRunners and ApplicationRunners are called

Community

Spring Developers on Twitter ☺

- Josh Long (@starbuxman)
- Madhura Bhave (@madhurabhave23)
- Phil Webb (@phillip_webb)
- Stéphane Nicoll (@snicoll)
- Andy Wilkinson (@ankinson)
- Dave Syer (@david_syer)

- … and many others

Some Tweets...


More replies
Richard L Burton III @rburton · May 20
Replying to @phillip_webb
What about tweets?
1
Phil Webb @phillip_webb · May 20
I gave up the email to make time for the tweets!
1
2
Richard L Burton III @rburton · May 20
Now that's what I call balance ;)
2

Patrick Baumgartner @patbaumgartner · May 15

Why is my custom @springboot starter locally executed before HibernateJpaAutoConfiguration and in a docker container after? How can I fix this?

1 1

Stéphane Nicoll
@snicoll

Following

Replying to @patbaumgartner @springboot

Nothing particular if you haven't specified an auto-configuration order.

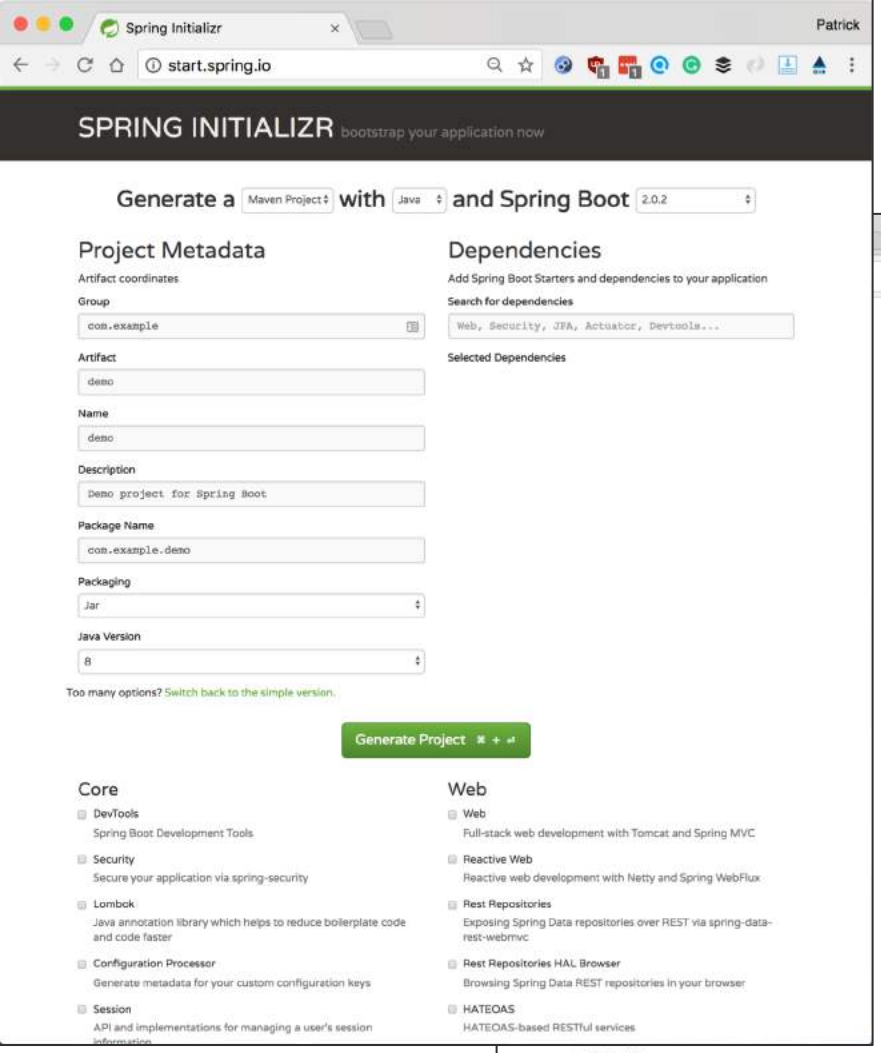
If you didn't, you should (see AutoConfigureBefore|After).

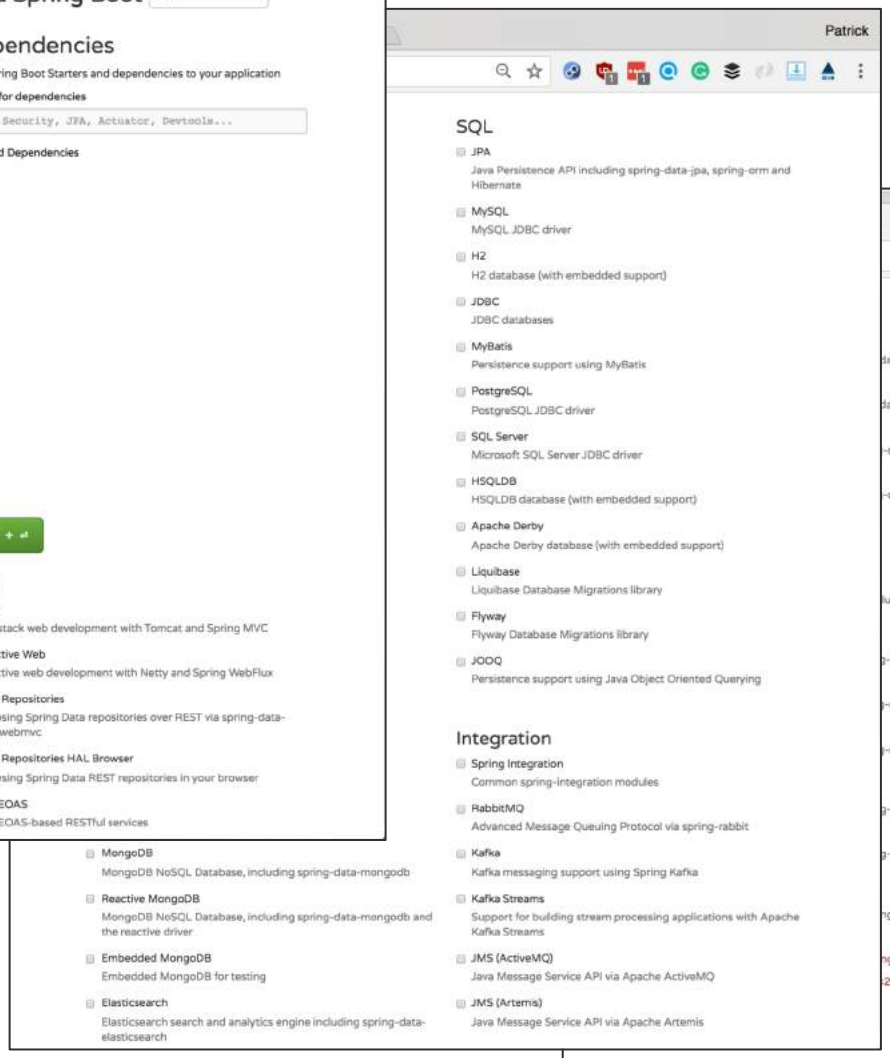
If you did, a sample we can run will be necessary

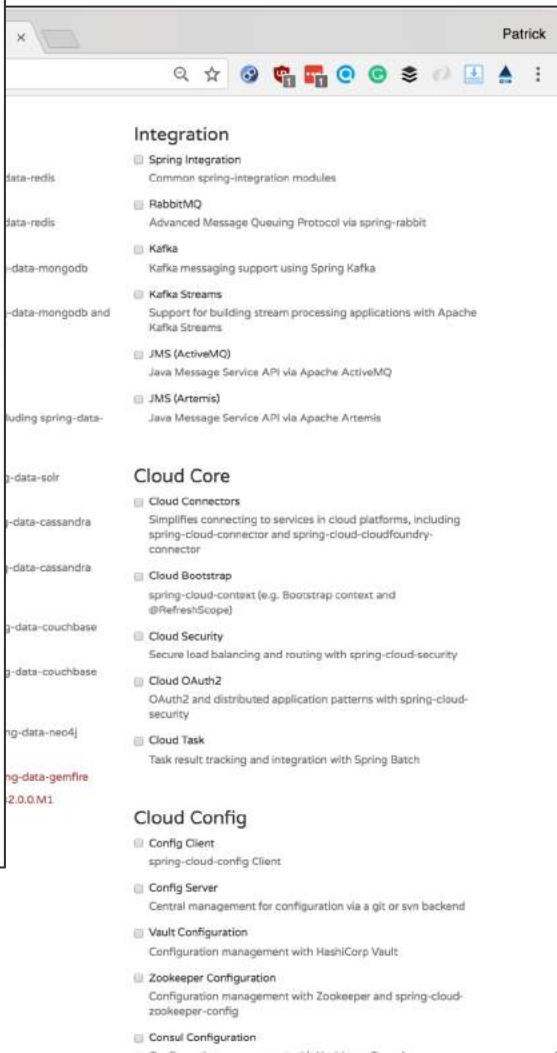
7:25 AM - 15 May 2018

4 Likes

1 4


Spring Initializr
Patrick
start.spring.io
SPRING INITIALIZR bootstrap your application now
Generate a Maven Project with Java and Spring Boot 2.0.2
Project Metadata
Artifact coordinates
Group
com.example
Artifact
demo
Name
demo
Description
Demo project for Spring Boot
Package Name
com.example.demo
Packaging
Jar
Java Version
8
Too many options? Switch back to the simple version.
Dependencies
Add Spring Boot Starters and dependencies to your application
Search for dependencies
Web, Security, JPA, Actuator, Devtools...
Selected Dependencies
Generate Project ⌘ + ⏎
Core
DevTools
Spring Boot Development Tools
Security
Secure your application via spring-security
Lombok
Java annotation library which helps to reduce boilerplate code and code faster
Configuration Processor
Generate metadata for your custom configuration keys
Session
API and implementations for managing a user's session
Web
Web
Full-stack web development with Tomcat and Spring MVC
Reactive Web
Reactive web development with Netty and Spring WebFlux
Rest Repositories
Exposing Spring Data repositories over REST via spring-data-rest-webmvc
Rest Repositories HAL Browser
Browsing Spring Data REST repositories in your browser
HATEOAS
HATEOAS-based RESTful services


Patrick
SQL
JPA
Java Persistence API including spring-data-jpa, spring-orm and Hibernate
MySQL
MySQL JDBC driver
H2
H2 database (with embedded support)
JDBC
JDBC databases
MyBatis
Persistence support using MyBatis
PostgreSQL
PostgreSQL JDBC driver
SQL Server
Microsoft SQL Server JDBC driver
HSQLDB
HSQLDB database (with embedded support)
Apache Derby
Apache Derby database (with embedded support)
Liquibase
Liquibase Database Migrations library
Flyway
Flyway Database Migrations library
JOOQ
Persistence support using Java Object Oriented Querying
Integration
Spring Integration
Common spring-integration modules
RabbitMQ
Advanced Message Queuing Protocol via spring-rabbit
Kafka
Kafka messaging support using Spring Kafka
Kafka Streams
Support for building stream processing applications with Apache Kafka Streams
JMS (ActiveMQ)
Java Message Service API via Apache ActiveMQ
JMS (Artemis)
Java Message Service API via Apache Artemis
MongoDB
MongoDB NoSQL Database, including spring-data-mongodb
Reactive MongoDB
MongoDB NoSQL Database, including spring-data-mongodb and the reactive driver
Embedded MongoDB
Embedded MongoDB for testing
Elasticsearch
Elasticsearch search and analytics engine including spring-data-elasticsearch


Patrick
Integration
Spring Integration
Common spring-integration modules
RabbitMQ
Advanced Message Queuing Protocol via spring-rabbit
Kafka
Kafka messaging support using Spring Kafka
Kafka Streams
Support for building stream processing applications with Apache Kafka Streams
JMS (ActiveMQ)
Java Message Service API via Apache ActiveMQ
JMS (Artemis)
Java Message Service API via Apache Artemis
Cloud Core
Cloud Connectors
Simplifies connecting to services in cloud platforms, including spring-cloud-connector and spring-cloud-cloudfoundry-connector
Cloud Bootstrap
spring-cloud-context (e.g. Bootstrap context and @RefreshScope)
Cloud Security
Secure load balancing and routing with spring-cloud-security
Cloud OAuth2
OAuth2 and distributed application patterns with spring-cloud-security
Cloud Task
Task result tracking and integration with Spring Batch
Cloud Config
Config Client
spring-cloud-config Client
Config Server
Central management for configuration via a git or svn backend
Vault Configuration
Configuration management with HashiCorp Vault
Zookeeper Configuration
Configuration management with Zookeeper and spring-cloud-zookeeper-config
Consul Configuration

Recap

Recap

AutoConfiguration is not magic!

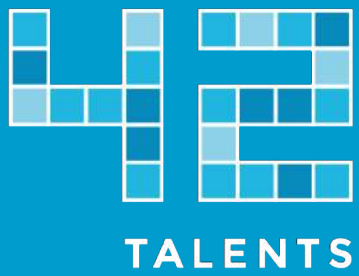
- `@SpringBootApplication` enables AutoConfiguration
- `SpringFactoriesLoader` removes the need for classpath scanning
- `@ConditionalOnClass/Bean` allows for flexibility and overriding
- Do not start your artifact name with `spring-boot-starter`!
- Autoconfigurations are executed after the regular beans
- Control the priority of your starters with `@AutoconfigureBefore/After`
- Check out the autoconfig in debug mode with `--debug` parameter
- Or include the actuator and browse to `/autoconfig` and `/beans`

Q & A

K100 on slido.co

Patrick Baumgartner, @patbaumgartner
patrick.baumgartner [at] 42talents [dot] com

http://www.42talents.com @42talents



Q & A

Meet me in the "Ask me anything corner" afterwards!

Patrick Baumgartner, @patbaumgartner
patrick.baumgartner [at] 42talents [dot] com

http://www.42talents.com @42talents