Hinter den Kulissen: Die Magie von Spring Boot

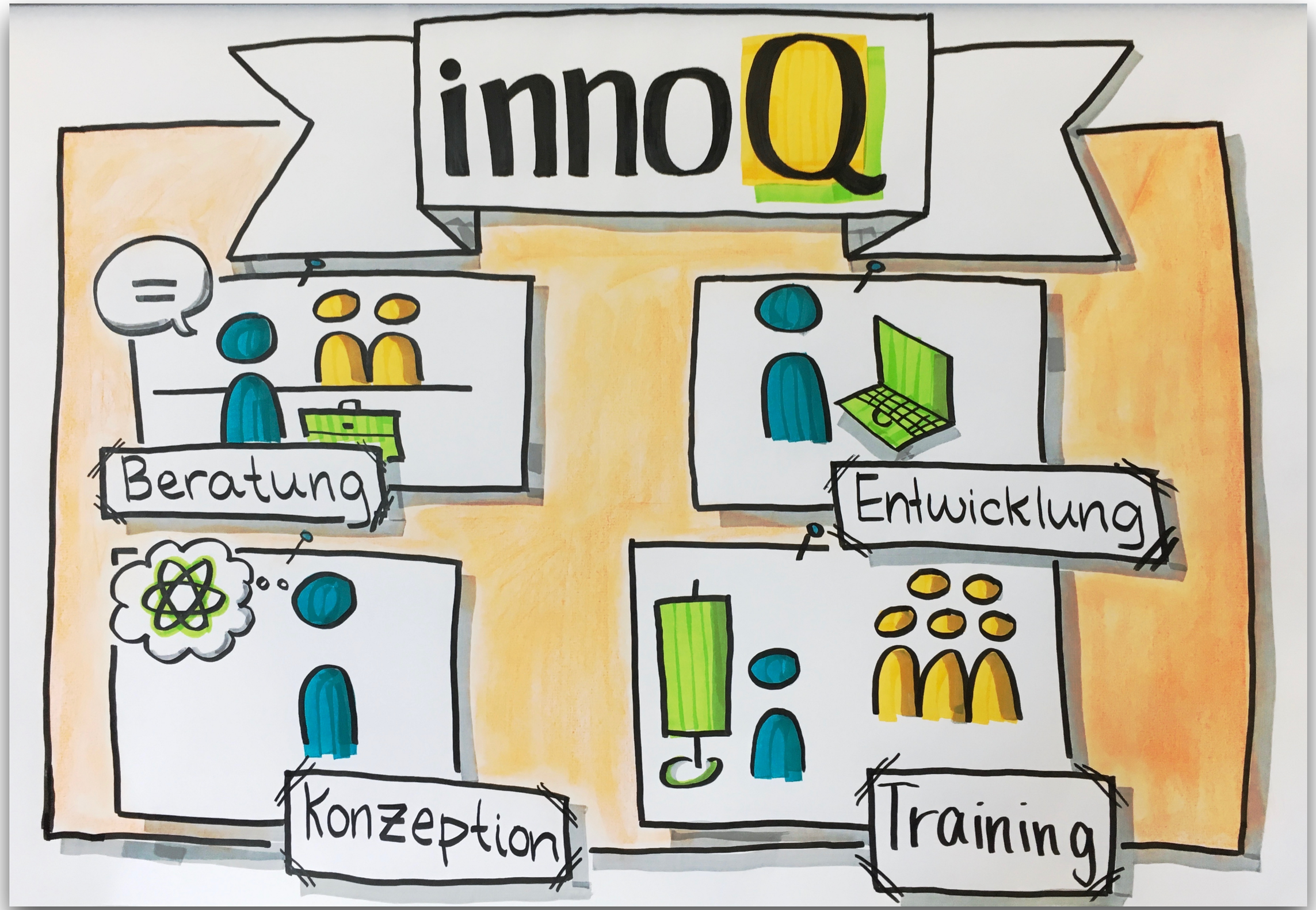
Java User Group Darmstadt

19. Oktober 2017

Michael Simons, @rotnroll666

Über mich

- Senior Consultant bei innoQ
- Erstes Spring Projekt 2009 (Spring 3)
- Erstes Spring Boot Projekt Anfang 2014
- Blog zu Java, Spring und Softwarearchitektur unter info.michael-simons.eu
- Schreibt gerne ->
- Regt sich auf Twitter als @rotnroll666 über alles mögliche auf



von @iamjoyclark

```
        <property name="fallbackToSystemLocale" value="false"/>
    </bean>
    <!-- Configure the JPA Adapter -->
    <bean id="jpaDialect" class="org.springframework.orm.jpa.vendor.HibernateJpaDialect"/>
    <bean id="jpaVendorAdapter" class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
      <property name="database" value="MYSQL"/>
      <property name="databasePlatform" value="org.hibernate.dialect.MySQLDialect"/>
      <property name="generateDdl" value="false"/>
      <property name="showSql" value="false"/>
    </bean>
    <!-- Configure the local Entity Manager Factory -->
    <bean id="entityManagerFactory"
  class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
      <property name="persistenceUnitName" value="theEntities"/>
      <property name="dataSource" ref="theDataSource"/>
      <property name="jpaDialect" ref="jpaDialect"/>
      <property name="jpaVendorAdapter" ref="jpaVendorAdapter"/>
      <property name="loadTimeWeaver">
        <bean class="org.springframework.instrument.classloading.InstrumentationLoadTimeWeaver"/>
      </property>
    </bean>
    <!-- Enable Spring JPA Transactions -->
    <bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager" p:entity-
manager-factory-ref="entityManagerFactory"/>
    <tx:annotation-driven transaction-manager="transactionManager"/>
    <bean id="exampleBean" class="examples.ExampleBean">
      <!-- setter injection using the nested <ref/> element -->
      <property name="beanOne">
        <ref bean="anotherExampleBean"/>
      </property>
      <!-- setter injection using the neater 'ref' attribute -->
      <property name="beanTwo" ref="yetAnotherBean"/>
      <property name="integerProperty" value="1"/>
    </bean>
    <bean id="anotherExampleBean" class="examples.AnotherBean"/>
    <bean id="yetAnotherBean" class="examples.YetAnotherBean"/>
</beans>
```

A long time ago, in a framework far,
far away

Stand heute

- XML-Konfiguration
- Komponenten-Scanning
- Explizite Java-Konfiguration
- Funktionale Bean-Registrierung (Spring 5)

FFFFFFF
FFFFFFF
FFFFFF
FFFUU
UUUU
UUUU
UUUU
UUUU-

A new hope

```
package de.springbootbuch.helloworld;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {

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

A new hope

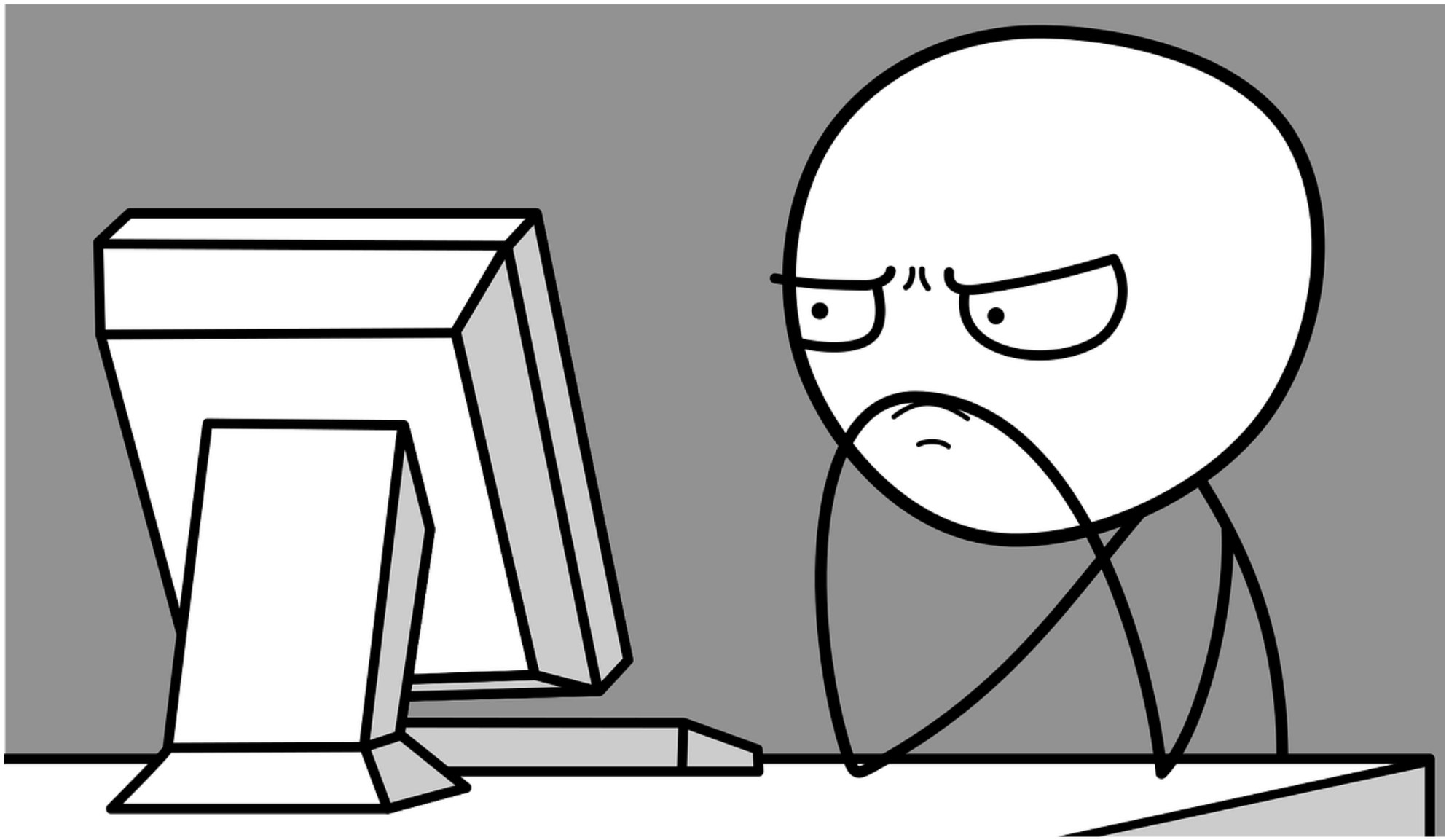
```
package de.springbootbuch.helloworld;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
class Application

fun main(args: Array<String>) {
    SpringApplication.run(Application::class.java, *args)
}
```

„Das ist mir zu viel Magie“



„Das ist mir zu viel Magie“

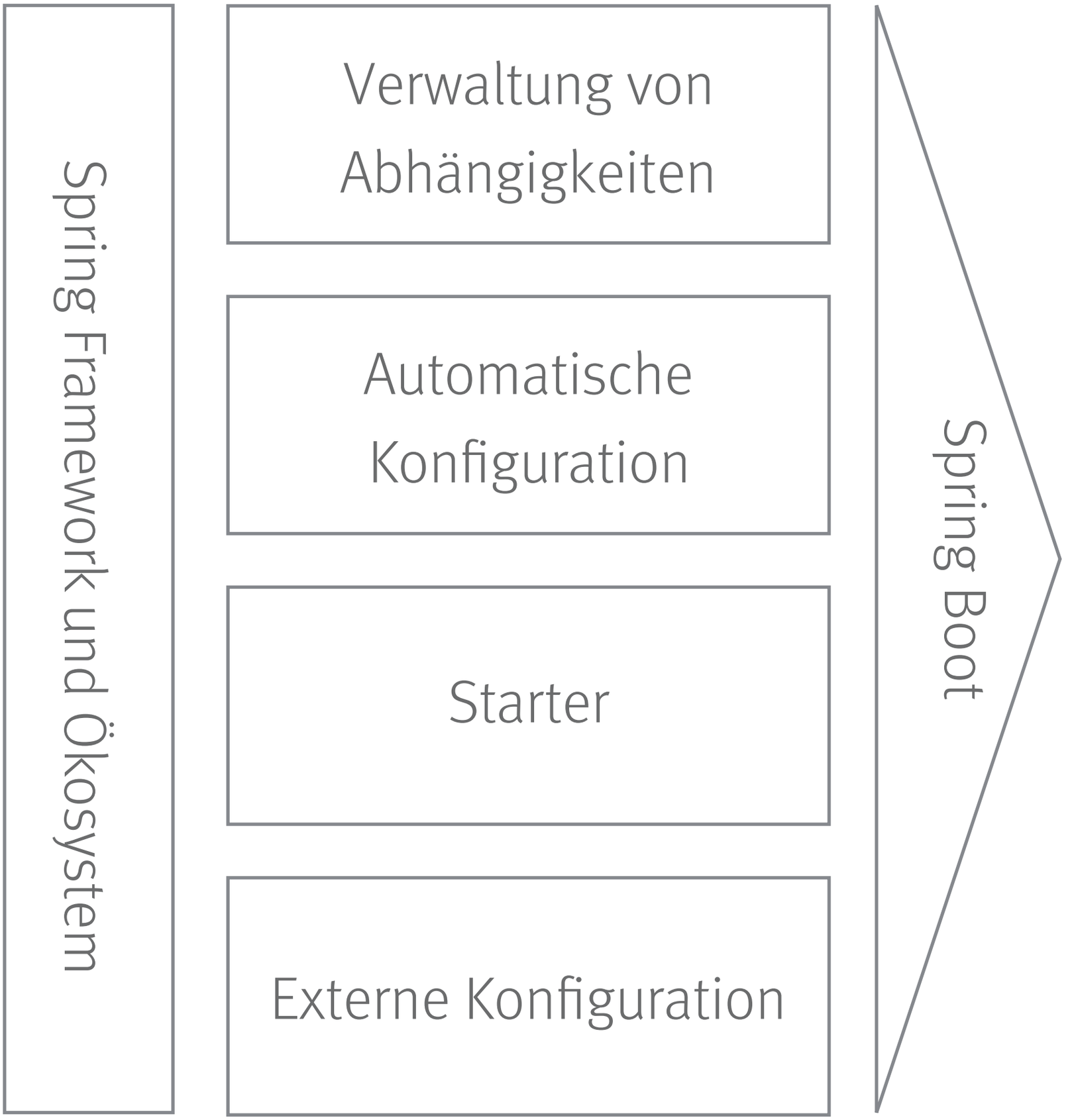
- Je nach deklarierten Abhängigkeiten passiert eine Menge, z.B. Konfiguration von
  - Spring Web MVC
  - WebFlux
  - Embedded Applicationcontainer
  - Datenbankverbindungen
  - und vieles mehr...

Was genau ist Spring Boot?

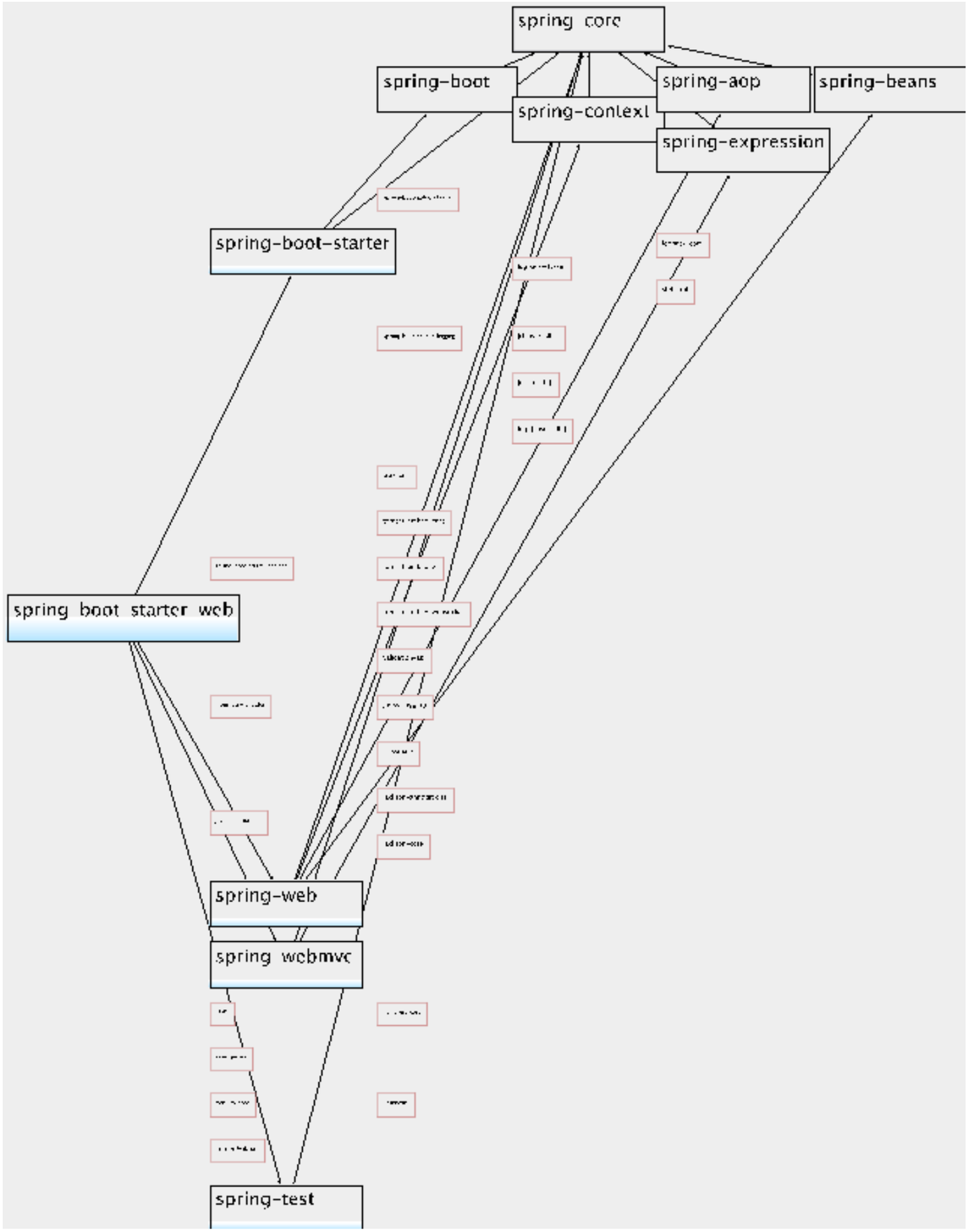
- Eine Sammlung von Libraries?
- Ein versteckter Application-Server?
- Ein neues Framework?
- Eine Runtime?
- Nicht per se ein Mikroservice-Framework!

Spring Boot: Ziele

- Schneller Start für Entwicklung mit Spring
- Sinnvolle Defaults
  - kein Code- oder Konfigurationsgenerator
  - nur solange wie nötig
- Extern konfigurierbar


Spring Framework und Ökosystem
Verwaltung von Abhängigkeiten
Automatische Konfiguration
Starter
Externe Konfiguration
Spring Boot

Verwaltung von Abhängigkeiten


spring core
spring-boot
spring-aop
spring-beans
spring-context
spring-expression
spring-boot-starter
spring boot starter web
spring-web
spring webmvc
spring-test

start.spring.io

SPRING INITIALIZR bootstrap your application now

Generate a Maven Project with Java and Spring Boot 1.5.7

Project Metadata

Artifact coordinates

Group

com.example

Artifact

demo

Dependencies

Add Spring Boot Starters and dependencies to your application

Search for dependencies

Web, Security, JPA, Actuator, Devtools...

Selected Dependencies

JPA ✕ Web ✕

Generate Project ⌘ + ⏎

Don't know what to look for? Want more options? Switch to the full version.

start.spring.io is powered by Spring Initializr and Pivotal Web Services

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.5.7.RELEASE</version>
    <relativePath/>
</parent>
```

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

Starter

Starter

- Bündeln Abhängigkeiten
- Stellen widerstandsfähige, automatische Konfiguration zur Verfügung

Was genau „starten“?

- Security
- Datenbanken
- Template Engines
- Validation
- Service Discovery
- Vieles mehr: github.com/spring-projects/spring-boot/tree/master/spring-boot-project/spring-boot-starters

Architektur eines Starters

- Starter Modul
- Autokonfiguration
  - JavaConfig
  - spring.factories

Bildquelle:
https://de.wikipedia.org/wiki/Bogen_(Architektur)#/media/File:Pont_du_Gard_from_river.jpg

Automatische Konfiguration

Eine Art Magie?

- OnClassCondition / OnMissingClassCondition
- OnBeanCondition / OnMissingBeanCondition
- OnPropertyCondition
- OnResourceCondition
- OnExpressionCondition
- OnJavaCondition
- OnJndiCondition
- OnWebApplicationCondition

Demo: Single-User für Spring Boot Security 2

Zusammenfassung

- @SpringBootApplication schaltet automatische Konfiguration ein
- spring.factories für @Configuration Klassen nicht vergessen
- @AutoConfigureBefore
- @Bean

Auf die Umgebung reagieren

- @ConditionalOnClass
- @ConditionalOnBean / @ConditionalOnMissingBean
- @ConditionalOnProperty
- Und viele mehr...

Keine Magie

- Spring Diagnostics
- `--debug` Parameter
- oder Spring Boot Actuator:
  `/application/autoconfig`

Eigene Bedingungen

- Implementiere o.s.c.annotation.Condition
- Erweitere o.s.boot.autoconfigure.SpringBootCondition
- Verschachtelte Bedingungen mit
  - AllNestedConditions
  - AnyNestedCondition
  - NoneNestedCondition

```
static class OnNoBannerButFun
    extends AllNestedConditions {

    public OnNoBannerButFun() {
        super(ConfigurationPhase.REGISTER_BEAN);
    }

    @ConditionalOnProperty(
        name = "spring.main.banner-mode",
        havingValue = "off"
    )
    static class OnBannerTurnedOff {}

    @ConditionalOnProperty(
        "springbootbuch-banner.cache-name")
    static class OnCacheNameSpecified {}

    @ConditionalOnClass(ObjectMapper.class)
    @ConditionalOnBean(ObjectMapper.class)
    static class OnObjectMapperAvailable {}

    @ConditionalOnBean(CacheManager.class)
    static class OnCacheManagerAvailable {}
}
```

Externe Konfiguration

Mit externer Konfiguration...

- ...wird interne / automatische Konfiguration beeinflusst
- ...werden Profile ausgewählt
- ...wird Fachlichkeit konfiguriert
- ...wird das Verhalten eines Artefakts im Sinne der 12-factor-app nur aus der Umgebung beeinflusst

Konfigurationsquellen

Extern

- devtools (1)
- Parameter (Kommandozeile sowie Maven- und Gradle-Plugins) (3)
- Servletconfig- und Kontext (4)
- JNDI (5)
- System.getProperties() (6)
- Umgebungsvariablen (7)
- Property-Dateien für spezifische Profile außerhalb des Artefakts (8)
- Property-Dateien außerhalb des Artefakts (10)

Intern

- @TestPropertySource / @SpringBootTest (2)
- Property-Dateien für spezifische Profile innerhalb des Artefakts (9)
- Property-Dateien innerhalb des Artefakts (11)

Zugriff

- @Value
- @ConditionalOnProperty
- @ConfigurationProperties

@Value("${something}")

- Core-Container feature
- Ermöglicht Spring-Expression-Language-Ausdrücke (SpEL)
  - Defaults sowohl für Ausdrücke ("#{aBean.age ?: 21}“)
  - Als auch für Properties („${someValue:foobar}")
- Nachteile:
  - Kein „relaxed-Binding“
  - Keine Gruppierung, Gefahr von Duplikaten

@ConditionalOnProperty

- Spring-Boot feature
- Bitte nur im Kontext automatischer Konfiguration verwenden!

@ConfigurationProperties

- Spring-Boot feature
- Typischer (Hinsichtlich Datentypen und „gebündelter“ Konfiguration)
- Validierbar
- Generierung von Metadaten (IDE-Support)
- Relaxed-binding

Demo

Fazit

- Spring Boot ist keine Magie:
  - Infrastruktur ist gut dokumentiert
  - Starter sind sehr widerstandsfähige (resillient) Erweiterungen
  - **Weniger nebensächliche Komplexität!**

Ein Wort der Warnung

- Don’t fight it!
- „Hacks“ fallen euch i.d.R. auf die Füße
  - Gibt es eine Konfigurationsoption?
  - Ist es per eigener Bean konfigurierbar?
  - Oder einen dedizierten Customizer?
- Falls es nicht passt, nehmt etwas anderes

Ressourcen

- Der fertige Starter: github.com/michael-simons/configurable-single-user-spring-boot-starter
- Slides: speakerdeck.com/michaelsimons
- Spring Boot Buch: springbootbuch.de
- Twitter: @rotnroll666

Feedback oder Fragen?