Leipzig, 12. April 2018

JUG Saxony

Hinter den Kulissen: Die Magie von Spring Boot

Michael Simons

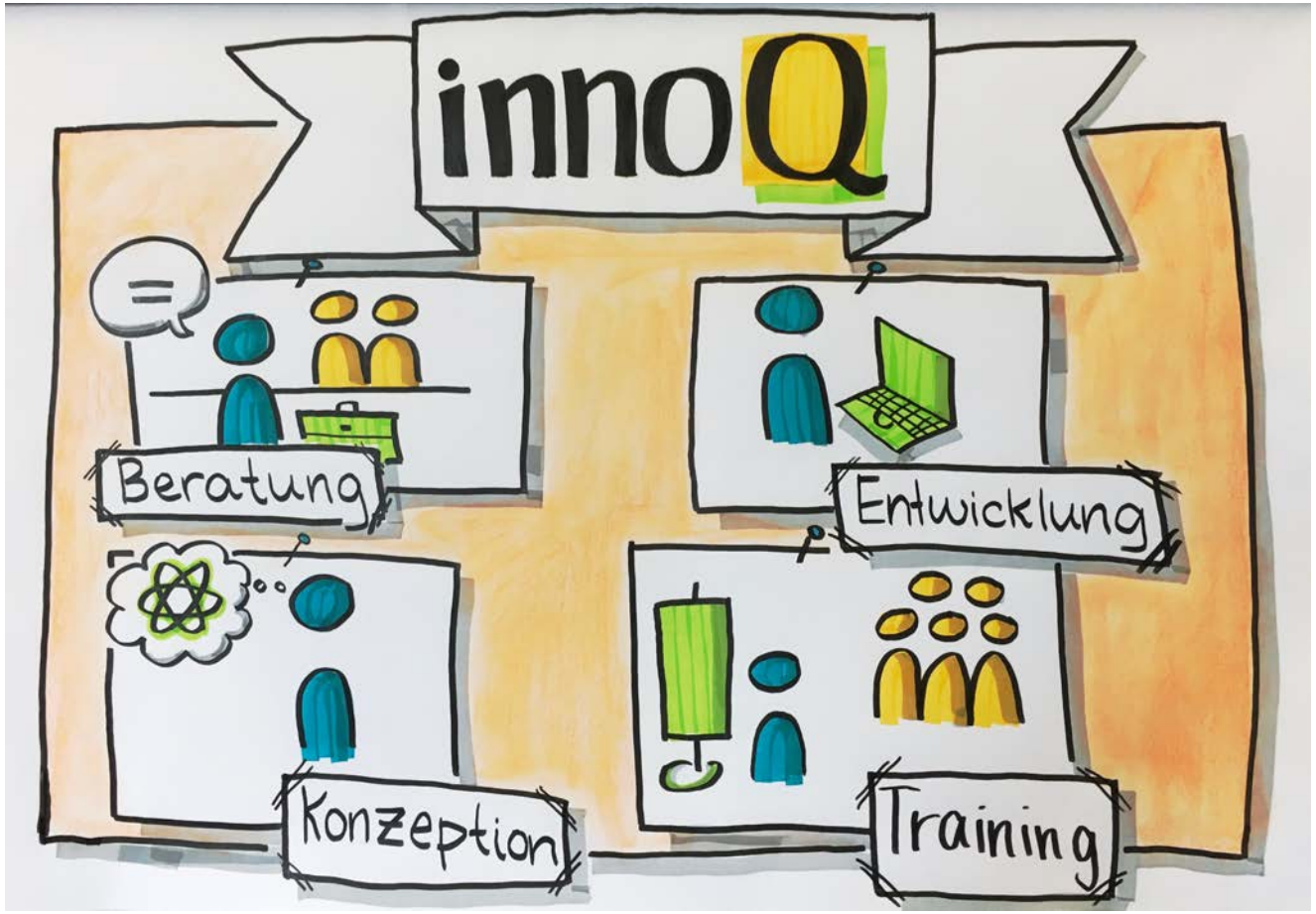
@rotnroll666

INNOQ

JUG saxony

INNOQ

- **Beratung**
- **Konzeption**
- **Entwicklung**
- **Training**



Über mich

Michael Simons
Senior Consultant
at INNOQ Deutschland GmbH

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

```
            <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-
er-factory-ref="entityManagerFactory"/>
x:annotation-driven transaction-manager="transactionManager"/>
an id="exampleBean" class="examples.ExampleBean">
<!-- setter injection using the nested <ref/> element -->
property name="beanOne">
  <ref bean="anotherExampleBean"/>
```

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

Stand heute

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

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)
}
```

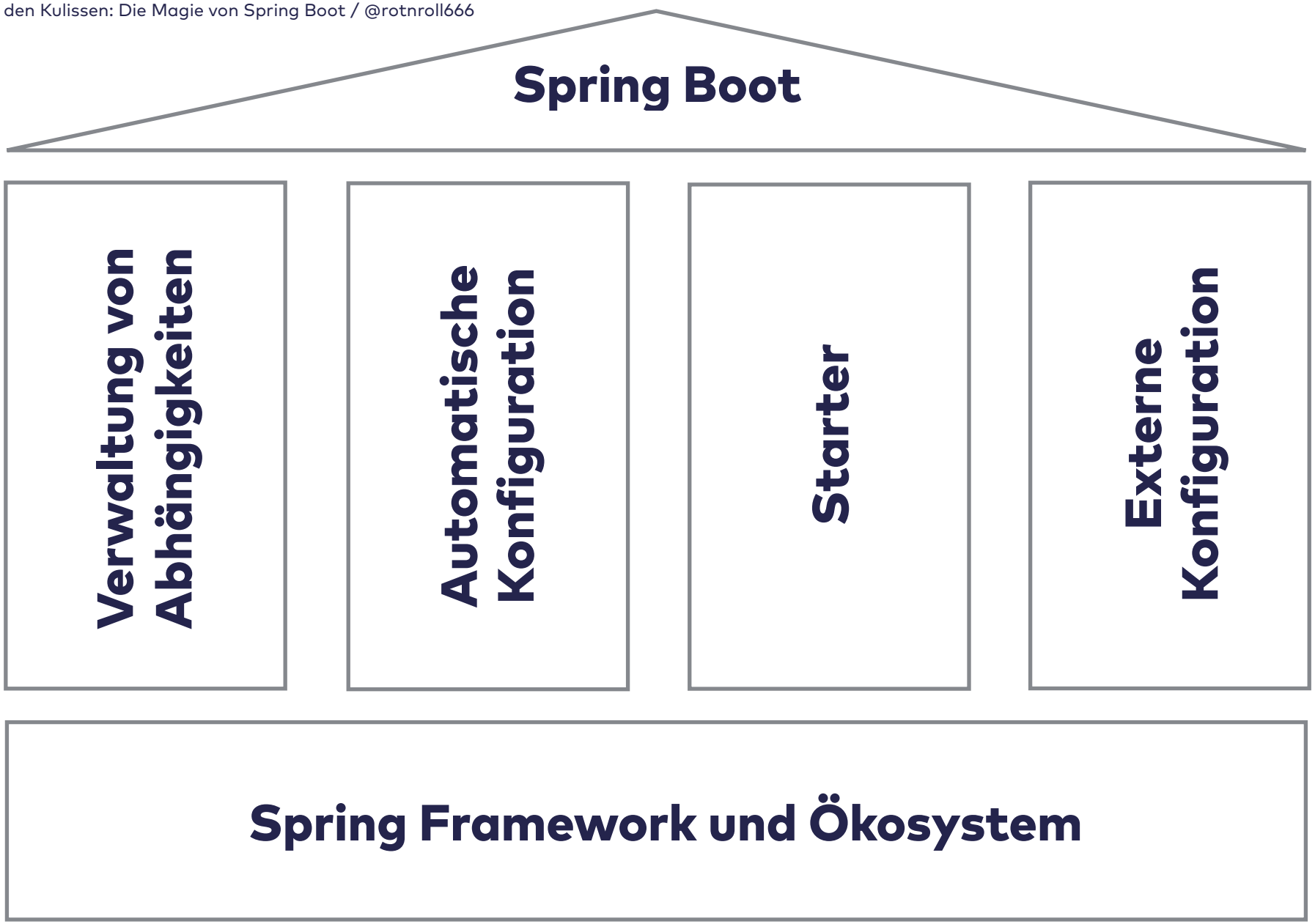
„Das ist mir zu viel Magie"

Was genau ist Spring Boot?

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

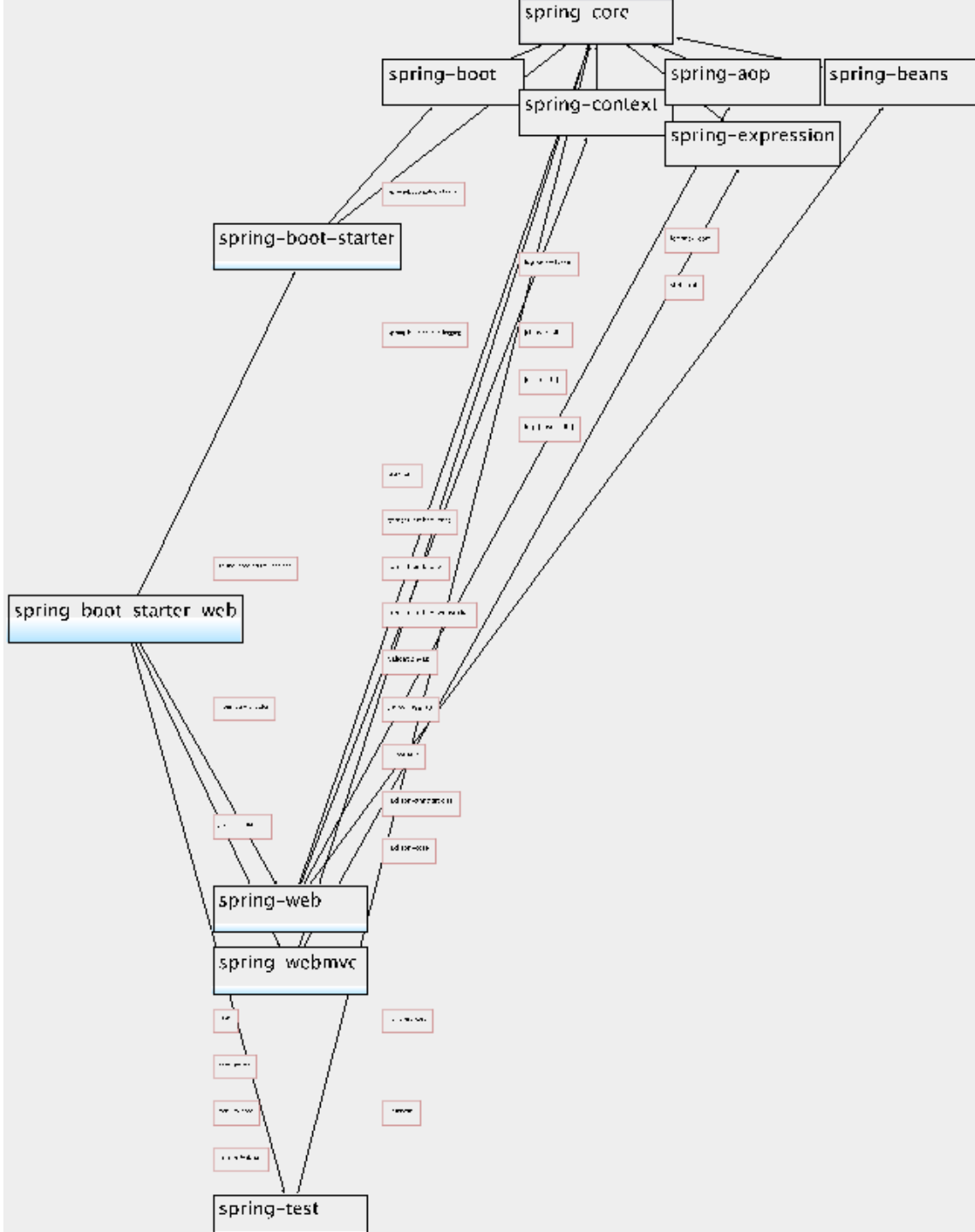
Spring Boot: Ziele

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


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

Verwaltung von Abhängigkeiten

Beispiel: Spring Web MVC



start.spring.io

SPRING INITIALIZR bootstrap your application now

Generate a Maven Project with Kotlin and Spring Boot 2.0.0

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

Reactive Web

Generate Project ⌘ + ⏎

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

Was bekommt man für Spring WebFlux?

```
<parent>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-parent</artifactId>
   <version>2.0.0.RELEASE</version>
   <relativePath/>
</parent>
```

Was bekommt man für Spring WebFlux?

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

Starter

Starter

- Bündeln Abhängigkeiten eines konkreten Aspekts
- 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

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

Automatische Konfiguration

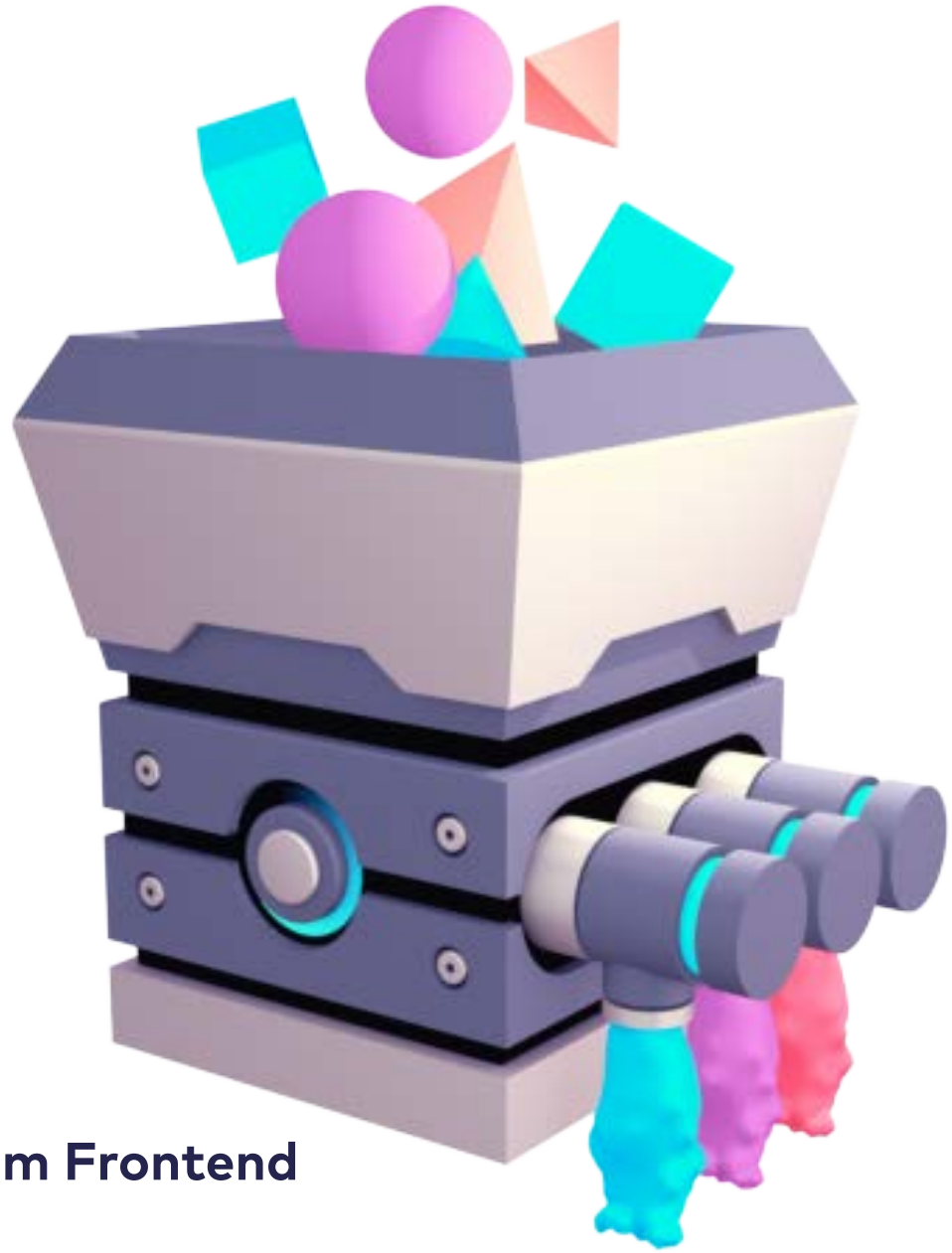
Eine Art Magie?

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

Demo: faucet-pipeline

faucet-pipeline

„faucet makes managing web assets a breeze: Whether pre-processing CSS or compiling modern JavaScript, all you need is a few simple lines of configuration to take advantage of the front-end community's established tooling. It doubles as a framework-independent asset pipeline, fingerprinting files to take advantage of HTTP caching."



Eine schlanke Abstraktion über aktuelle Werkzeuge im Frontend

JavaScript, SCSS-Kompilierung und Fingerprint

```
let targetBaseDir = "./target/classes/static"

module.exports = {
    js: [{
        source: "./src/main/assets/javascripts/application.js",
        target: targetBaseDir + "/javascripts/application.js"
    }],
    sass: [{
        source: "./src/main/assets/stylesheets/application.scss",
        target: targetBaseDir + "/stylesheets/application.css"
    }],
    manifest: {
        target: "./target/classes/manifest.json",
        webRoot: targetBaseDir
    }
};
```

faucet-Manifest

```
{
    "application.css": "/stylesheets/application-70d5f3dc18d122548efadcedfc0874f0.css",
    "application.js": "/javascripts/application-749a4217bb580c4537e5667c61f7c93c.js",
    "faucet-logo.png": "/images/faucet-logo-86ca3ec17bfc9579a4f985caa5aaf347.png"
}
```

Idee: Springs Static-Resource-Chain nutzen

- Frameworkfeature für WebMVC und WebFlux
- Fingerprinting und statische Versionen Out-Of-The-Box
- Pluggable-System für ResourceResolver
- Es wird benötigt
  - Ein Eintrag in der ResourceChain
  - Ein ResourceResolver
  - Falls nicht vorhanden: ein URL-Transformer

Notwendige Bedingungen

```
public class FaucetPipelineAutoConfiguration {

    Manifest faucetManifest() {
        return new Manifest();
    }
}
```

Notwendige Bedingungen

- **Auto-Configuration ist immer explizite `@Configuration`**

```
@Configuration
```

```
public class FaucetPipelineAutoConfiguration {
   @Bean
   Manifest faucetManifest() {
        return new Manifest();
    }
}
```

Notwendige Bedingungen

- **Auto-Configuration ist immer explizite `@Configuration`**
- **Spring Resource-Chain muss aktiv sein**

```
@Configuration
@ConditionalOnEnabledResourceChain
```

```
public class FaucetPipelineAutoConfiguration {
   @Bean
   Manifest faucetManifest() {
        return new Manifest();
    }
}
```

Notwendige Bedingungen

- Auto-Configuration ist immer explizite `@Configuration`
- Spring Resource-Chain muss aktiv sein
- Das Manifest muss vorliegen

```
@Configuration
@ConditionalOnEnabledResourceChain
@ConditionalOnResource(resources =
  "${faucet-pipeline.manifest:" +
  "classpath:/manifest.json}")

public class FaucetPipelineAutoConfiguration {
   @Bean
   Manifest faucetManifest() {
        return new Manifest();
    }
}
```

Notwendige Bedingungen

- Auto-Configuration ist immer explizite `@Configuration`
- Spring Resource-Chain muss aktiv sein
- Das Manifest muss vorliegen
- Es muss ein Mechanismus zum Parsen vorhanden sein

```
@Configuration
@ConditionalOnEnabledResourceChain
@ConditionalOnResource(resources =
  "${faucet-pipeline.manifest:" +
  "classpath:/manifest.json}")
@ConditionalOnClass(ObjectMapper.class)
public class FaucetPipelineAutoConfiguration {
   @Bean
   Manifest faucetManifest() {
        return new Manifest();
    }
}
```

Sieht das nicht wie normale Konfiguration aus? 🤔

Sieht das nicht wie normale Konfiguration aus? 🤔

```
@SpringBootApplication
public class DemoWebmvcApplication {
}
```

Sieht das nicht wie normale Konfiguration aus? 🤔

```
@SpringBootApplication
public class DemoWebmvcApplication {
}
```

Sieht das nicht wie normale Konfiguration aus? 🤔

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan
public @interface SpringBootApplication {
}
```

Sieht das nicht wie normale Konfiguration aus? 🤔

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan
public @interface SpringBootApplication {
}
```

Sieht das nicht wie normale Konfiguration aus? 🤔

```
org.springframework.boot.autoconfigure.EnableAutoConfiguration = \
  org.faucet_pipeline.spring.autoconfigure.FaucetPipelineAutoConfiguration
```

Sieht das nicht wie normale Konfiguration aus? 🤔

```
org.springframework.boot.autoconfigure.EnableAutoConfiguration = \
  org.faucet_pipeline.spring.autoconfigure.FaucetPipelineAutoConfiguration
```

`META-INF/spring.factories` 🧐

Hinreichende Bedingungen

```
@Configuration
@ConditionalOnEnabledResourceChain
@ConditionalOnResource(resources =
  "${faucet-pipeline.manifest:" +
  "classpath:/manifest.json}")
@ConditionalOnClass(ObjectMapper.class)
```

```
public class FaucetPipelineAutoConfiguration {
    @Bean
    Manifest faucetManifest() {
        return new Manifest();
    }
}
```

Hinreichende Bedingungen

- **Support für WebMVC und WebFlux**

```
@Configuration
@ConditionalOnEnabledResourceChain
@ConditionalOnResource(resources =
  "${faucet-pipeline.manifest:" +
  "classpath:/manifest.json}")
@ConditionalOnClass(ObjectMapper.class)
@Import({
    PipelineForWebMvcConfiguration.class,
    PipelineForWebFluxConfiguration.class
  })
public class FaucetPipelineAutoConfiguration {
    @Bean
    Manifest faucetManifest() {
        return new Manifest();
    }
}
```

Hinreichende Bedingungen

- **Support für WebMVC und WebFlux**

```
@Configuration


class PipelineForWebFluxConfiguration {

    @Bean

    ResourceUrlProvider resourceUrlProvider() {
        return new ResourceUrlProvider();
    }
}
```

Hinreichende Bedingungen

- **Support für WebMVC und WebFlux**
- **WebFlux bitte nur bei entsprechendem Typ**

```
@Configuration
@ConditionalOnWebApplication(type = REACTIVE)


class PipelineForWebFluxConfiguration {

    @Bean

    ResourceUrlProvider resourceUrlProvider() {
        return new ResourceUrlProvider();
    }
}
```

Hinreichende Bedingungen

- **Support für WebMVC und WebFlux**
- **WebFlux bitte nur bei entsprechendem Typ**
- **In der richtigen Reihenfolge**

```
@Configuration
@ConditionalOnWebApplication(type = REACTIVE)
@AutoConfigureBefore(
  WebFluxAutoConfiguration.class)
class PipelineForWebFluxConfiguration {

    @Bean

    ResourceUrlProvider resourceUrlProvider() {
        return new ResourceUrlProvider();
    }
}
```

Hinreichende Bedingungen

- Support für WebMVC und WebFlux
- WebFlux bitte nur bei entsprechendem Typ
- In der richtigen Reihenfolge
- „Behave nicely“

```
@Configuration
@ConditionalOnWebApplication(type = REACTIVE)
@AutoConfigureBefore(
  WebFluxAutoConfiguration.class)
class PipelineForWebFluxConfiguration {

    @Bean
    @ConditionalOnMissingBean
    ResourceUrlProvider resourceUrlProvider() {
        return new ResourceUrlProvider();
    }
}
```

Ist das überhaupt testbar?

- **Alle Bedingungen sind zielgenau testbar!**
- **Neu in Spring Boot 2: „Assertable-Contexts“**

```
public class FaucetPipelineAutoConfigurationTest {
  private ApplicationContextRunner contextRunner =
    new ApplicationContextRunner()
      .withConfiguration(AutoConfigurations.of(
        FaucetPipelineAutoConfiguration.class));

  @Test
  public void shouldProvideManifestAndProperties() {
    contextRunner
      .withPropertyValues(
        "spring.resources.chain.enabled = true",
        "faucet-pipeline.manifest = classpath:/m.json")
      .run(ctx -> assertThat(ctx)
          .hasSingleBean(Manifest.class)
          .hasSingleBean(FaucetPipelineProperties.class)
    );
  }
}
```

Ist das überhaupt testbar?

- **Alle Bedingungen sind zielgenau testbar!**
- **Neu in Spring Boot 2: „Assertable-Contexts“**

```
public class FaucetPipelineAutoConfigurationTest {
  private ApplicationContextRunner contextRunner =
    new ApplicationContextRunner()
      .withConfiguration(AutoConfigurations.of(
        FaucetPipelineAutoConfiguration.class));

  @Test
  public void shouldRequireManifest() {
    contextRunner
      .withPropertyValues(
        „spring.resources.chain.enabled = true")
      .run(ctx -> assertThat(ctx)
        .doesNotHaveBean(Manifest.class)
        .doesNotHaveBean(FaucetPipelineProperties.class)
      );
  }
}
```

Ist das überhaupt testbar?

- **Alle Bedingungen sind zielgenau testbar!**
- **Neu in Spring Boot 2: „Assertable-Contexts“**

```
public class FaucetPipelineAutoConfigurationTest {
  private ApplicationContextRunner contextRunner =
    new ApplicationContextRunner()
      .withConfiguration(AutoConfigurations.of(
        FaucetPipelineAutoConfiguration.class));

  @Test
  public void shouldRequireObjectMapperOnClasspath() {
    contextRunner
      .withClassLoader(
          new FilteredClassLoader(ObjectMapper.class))
      .withPropertyValues(REQUIRED_PROPERTIES)
      .run(ctx -> assertThat(ctx)
        .doesNotHaveBean(Manifest.class)
        .doesNotHaveBean(FaucetPipelineProperties.class)
      );
  }
}
```

Ist das überhaupt testbar?

- **Alle Bedingungen sind zielgenau testbar!**
- **Neu in Spring Boot 2: „Assertable-Contexts“**

```
public class FaucetPipelineAutoConfigurationTest {
  private ApplicationContextRunner contextRunner =
    new ApplicationContextRunner()
      .withConfiguration(AutoConfigurations.of(
        FaucetPipelineAutoConfiguration.class));

  @Test
  public void shouldNeedWebEnvironment() {
    contextRunner
      .withPropertyValues(REQUIRED_PROPERTIES)
      .run(ctx -> assertThat(ctx)
        .doesNotHaveBean(WEB_MVC_CONFIGURER_NAME)
        .doesNotHaveBean(WEB_FLUX_CONFIGURER_NAME)
      );
  }
}
```

Ist das überhaupt testbar?

- **Alle Bedingungen sind zielgenau testbar!**
- **Neu in Spring Boot 2: „Assertable-Contexts“**

```
public class PipelineForWebFluxConfigurationTest {
  private ReactiveWebApplicationContextRunner contextRunner
    = new ReactiveWebApplicationContextRunner()
        .withConfiguration(AutoConfigurations.of(
            WebFluxAutoConfiguration.class,
            FaucetPipelineAutoConfiguration.class));

  @Test
  public void shouldProvideNeededBeans() {
    contextRunner
      .withPropertyValues(REQUIRED_PROPERTIES)
      .run(ctx -> assertThat(ctx)
        .hasBean("resourceUrlProvider")
        .hasBean("urlTransformingFilter")
        .hasBean(WEB_FLUX_CONFIGURER_NAME));
  }
}
```

Eigene Bedingungen

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

Keine Magie

- Spring Diagnostics
  - `--debug` Parameter
  - oder Spring Boot Actuator: `/actuator/conditions` <- Neu mit Spring Boot 2!

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

Externe und...

- 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)

interne Konfigurationsquellen

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

Zugriff mittels...

- Environment-Instanz
- @Value
- @ConditionalOnProperty
- @ConfigurationProperties
- Neu in Spring Boot 2: Binding-API

```
Binder.get(environment) // Retrieve a binder based on the
  // Bind all properties or just a subset
  .bind("a-prefix", FaucetPipelineProperties.class)
  // If it doesn't bind, use a default
  .orElseGet(FaucetPipelineProperties::new);
```

`@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
- Typsicher (Hinsichtlich Datentypen und „gebündelter“ Konfiguration)
- Validierbar
- Generierung von Metadaten (IDE-Support)
- Relaxed-binding

```
@Configuration
@ConditionalOnEnabledResourceChain
@ConditionalOnResource(resources =
  "${faucet-pipeline.manifest:" +
  "classpath:/manifest.json}"
)
@ConditionalOnClass(ObjectMapper.class)
@Import({
  PipelineForWebMvcConfiguration.class,
  PipelineForWebFluxConfiguration.class}
)
@EnableConfigurationProperties({
  ResourceProperties.class,
  FaucetPipelineProperties.class
})
public class FaucetPipelineAutoConfiguration {
    @Bean
    Manifest faucetManifest(FaucetPipelineProperties properties) {
        final ObjectMapper objectMapper = new ObjectMapper();
        objectMapper.enable(SerializationFeature.INDENT_OUTPUT);

        return new Manifest(objectMapper, prooperties.getManifest());
    }
}
```

```java
@Getter
@Setter
@ConfigurationProperties(prefix = "faucet-pipeline")
public class FaucetPipelineProperties {

    /**
     * Path or resource for Faucets manifest. Defaults to
     * <pre>manifest.json</pre>.
     */
    private Resource manifest = new ClassPathResource("manifest.json");

    private String[] pathPatterns = {"/**"};

    /**
     * Flag, wether the manifest should be cached or not. Set it to <code>false</code>
     * during development to use faucets watch task and get your assets reloaded.
     */
    private boolean cacheManifest = true;
}
```

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

target/classes/META-INF/spring-configuration-metadata.json

```json
{
  "hints": [],
  "groups": [
    {
      "sourceType": "org.faucet_pipeline.spring.autoconfigure.FaucetPipelineProperties",
      "name": "faucet-pipeline",
      "type": "org.faucet_pipeline.spring.autoconfigure.FaucetPipelineProperties"
    }
  ],
  "properties": [
    {
      "sourceType": "org.faucet_pipeline.spring.autoconfigure.FaucetPipelineProperties",
      "defaultValue": true,
      "name": "faucet-pipeline.cache-manifest",
      "description": "Flag, wether the manifest should be cached...",
      "type": "java.lang.Boolean"
    }
  ]
}
```

Fazit

- **Spring Boot ist keine Magie:**
  - **Infrastruktur ist gut dokumentiert**
    (Wer ist so verrückt und schreibt dazu noch ein Buch?!)
- **Starter sind sehr widerstandsfähige (resillient) Erweiterungen**
- **Persönliche Anwendungsfällen**
  - **Resource-Handler wie Faucet und Wro4J**
  - **Integration externer Dienste mit Spring-Security**
  - **Sehr spezielle Anpassungen des Entity-Managers**
- **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 über einen dedizierten Customizer?
- Falls es nicht passt, nehmt etwas anderes

Ressourcen

- Faucet-Pipeline und Spring-Boot-Starter
  faucet-pipeline.org
  github.com/faucet-pipeline/faucet-pipeline-spring-boot-starter
- Slides: speakerdeck.com/michaelsimons
- Spring Boot Buch
  - Begonnen Januar 2017
  - Erscheint ~~Dezember 2017, Januar 2018, Februar 2018~~ ~~demnächst~~ nächste Woche!!! 😅
  - @SpringBootBuch // springbootbuch.de