Mainz, 25. April 2018

JAX 2018

Spring Boot 2: Hot topics

Michael Plöd, @bitboss

Michael Simons, @rotnroll666

INNOQ

@bitboss

Michael Plöd
Principal Consultant
at INNOQ Deutschland GmbH

- **Interessiert an Software Architektur, aktuell mit Fokus auf Domain-driven Design**
- **Spring Anwender seit 2004**
- **Twitter: @bitboss**

@rotnroll666

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

Spring Boot Versionierungsschema

- Kein Semantic-Versioning
- `{major}.{minor}.{micro}.{release_type}`
  - major: Signifikante neue Funktionen, nicht unbedingt abwärtskompatibel
  - minor: Neue Funktionen, idR abwärtskompatibel
  - micro: Bugfixes, Verbesserungen

4 Jahre Spring Boot

- 1.0 am 1. April 2014
  - Schneller Start für Entwicklung mit Spring
  - kein Code- oder Konfigurationsgenerator
  - Extern konfigurierbar
  - Einheitliches Komponenten-Modell
  - Eingebetteter Container

4 Jahre Spring Boot

- 1.1 am 10. Juni 2014
  - Metrics- und Health-Endpunkte
  - Datenbankmigrationen
  - Neue Starter
  - Custom Banner!!

4 Jahre Spring Boot

- 1.2 am 11. Dezember 2014
  - Servlet 3.1
  - @SpringBootApplication
  - JTA-Transaktionen

4 Jahre Spring Boot

- 1.3 am 16. November 2015
  - DevTools
  - Eigenständig ausführbare JARs
  - Größere Umbauten der Actuator Endpoints für Java 8

4 Jahre Spring Boot

- 1.4 am 28. Juli 2016
  - Besseres Debugging (Failure-Analyse)
  - Hibernate 5
  - Test-Slices
  - banner.jpg

4 Jahre Spring Boot

- 1.5 am 30. Januar 2017
  - Nativer Support für Kafka
  - Actuator-Erweiterungen für Cloud-Foundry

4 Jahre Spring Boot

- 2.0 am 1. März 2018

Themen

- Neue Grundlagen: Java 8, Spring 5 und aktualisierte Abhängigkeiten
- Fokus auf reaktive Anwendungsentwicklung
- Actuator 2.0
- Spring Security 5
- Spring Data
- Todos

First things first

DAS Enterprise Feature schlechthin!

Animated banner

Animated banner

Java 8, Spring 5 und aktualisierte Abhängigkeiten

Java 8

- Type-Inferenz
- Lambdas
- Default-Methods für diverse Konfigurations-Interfaces,
  Adapter idR deprecated

```
import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;

@Configuration
public class WebConfig extends WebMvcConfigurerAdapter {
    @Override
    public void configureContentNegotiation(ContentNegotiationConfigurer configurer) {
        super.configureContentNegotiation(configurer);
        configurer.favorParameter(true);
    }
}
```

Java 8

- Type-Inferenz
- Lambdas
- Default-Methods für diverse Konfigurations-Interfaces,
  Adapter idR deprecated

```
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class WebConfig implements WebMvcConfigurer {
    @Override
    public void configureContentNegotiation(ContentNegotiationConfigurer configurer) {
        configurer.favorParameter(true);
    }
}
```

Spring 5

- Nullsafe-Apis (@Nullable etc.)
- Indizierter Component-Scan (über META-INF/spring.components)
- spring-jcl
- Massives Baseline-Upgrade
  - Tomcat 8.5+, Jetty 9.4,
  - Hibernate 5+
- Depracted Klassen und Methoden wurden entfernt
- Framework-Support wurde entfernt
  - u.a. Portlets, Velocity, Guava Cache

Spring 5

- HTTP/2
  - Nativ mit Tomcat 9.x, Jetty 9.4 und Undertow 1.4+
  - Ja(va|karta) EE Servlet 4.0 mit Spring 5.1
- HTTP/2 Server Push in Spring MVC mit `PushBuilder`

Aktualisierte Abhängigkeiten mit Spring Boot 2

- Hibernate 5.2
- Flyway 5
- Spring Data Kay
- Spring Security 5
- Thymeleaf 3

Fokus auf reaktive Anwendungsentwicklung

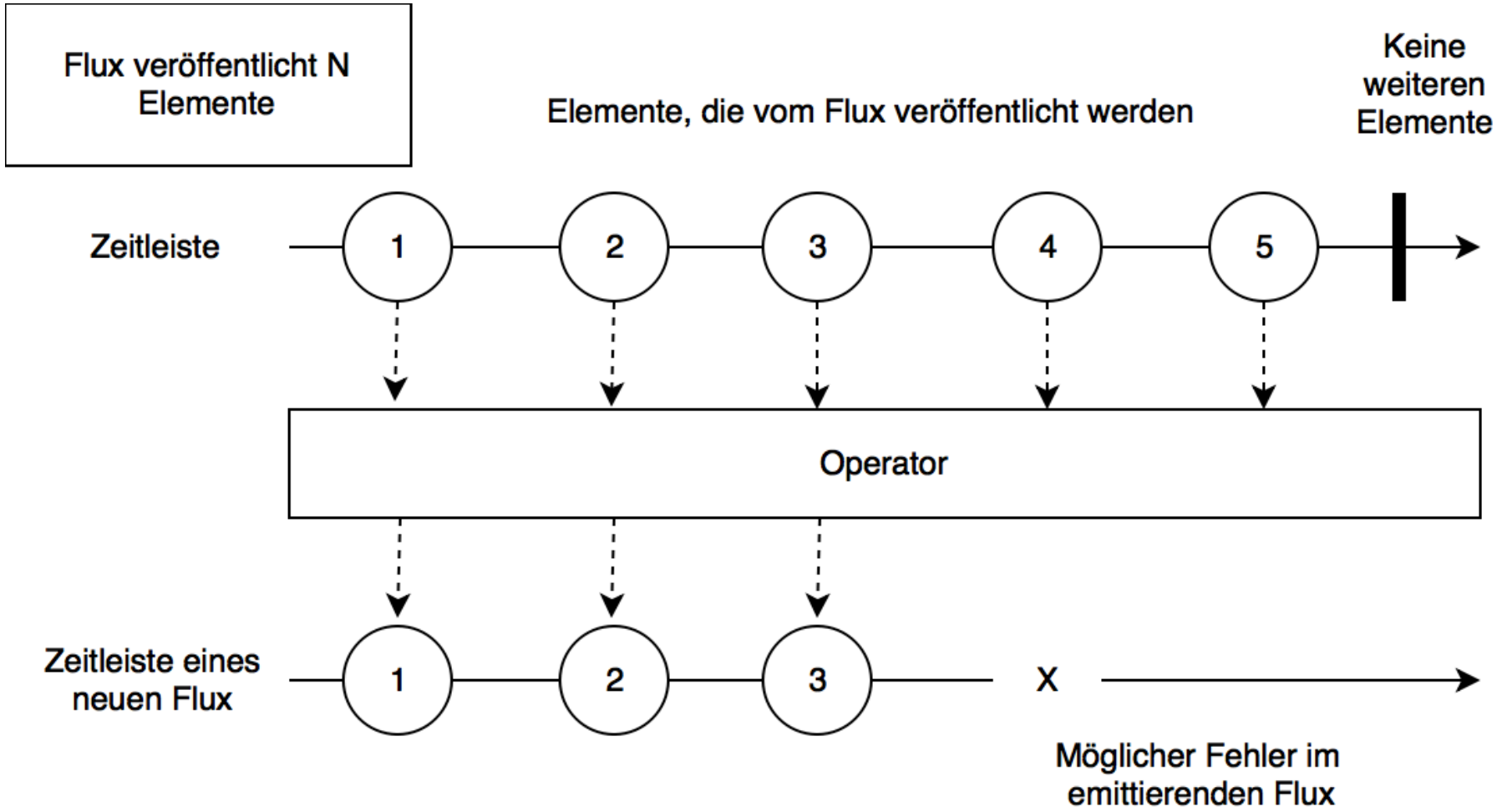
Die große Reactive Story

- Wichtige Neuerung in Spring 5
- Parallel zum bestehenden Ansatz
- Kern: robustere und ressourcenschonendere Anwendungen
- Reactive Manifest:
  „elastisch, robust, jederzeit antwortbereit und nachrichtenorientiert.“

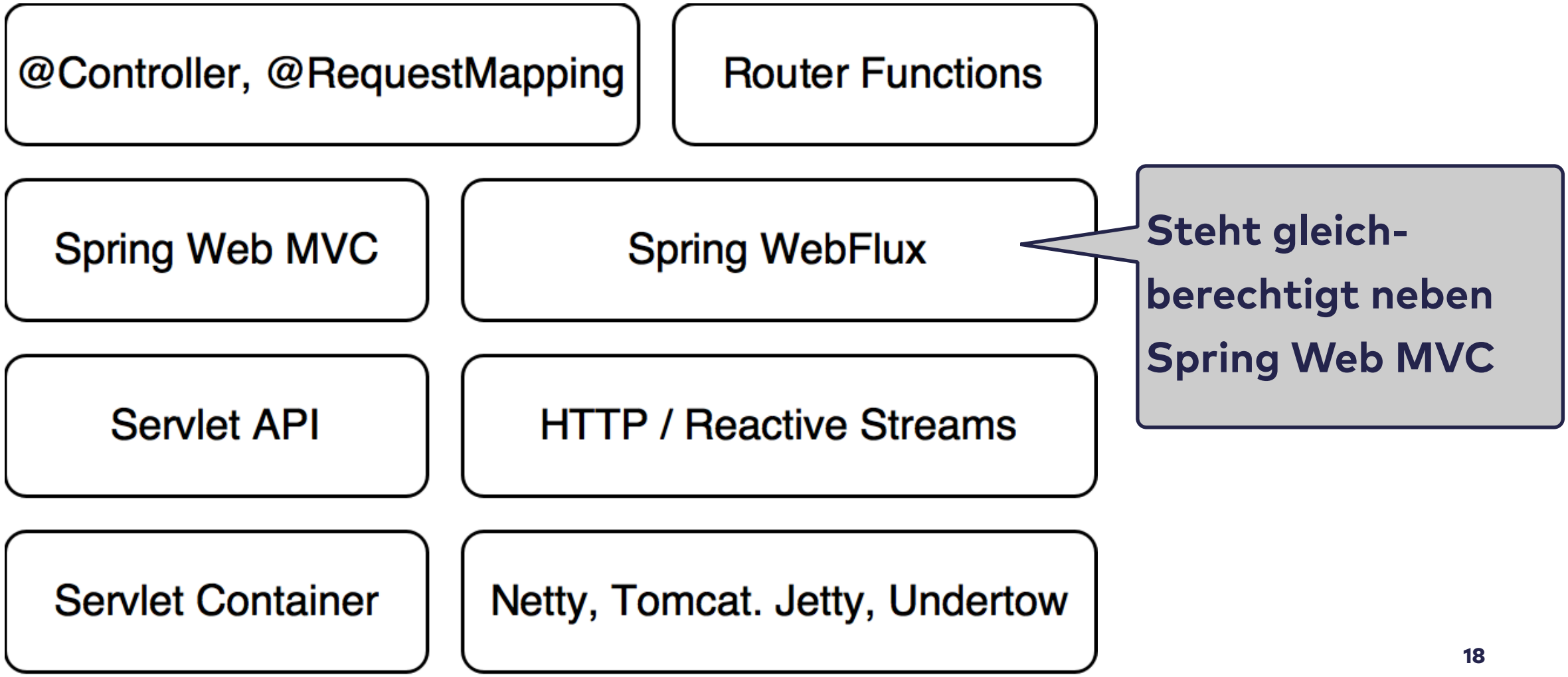
Exkurs: Mono und Flux

- Implementierungen eines Publishers der Flow-API
- Mono: Publiziert 0 oder 1 Element
- Flux: Publiziert n Elemente

Marble Diagramm Flux



Spring WebFlux



Spring WebFlux

- **Neuer Starter:**

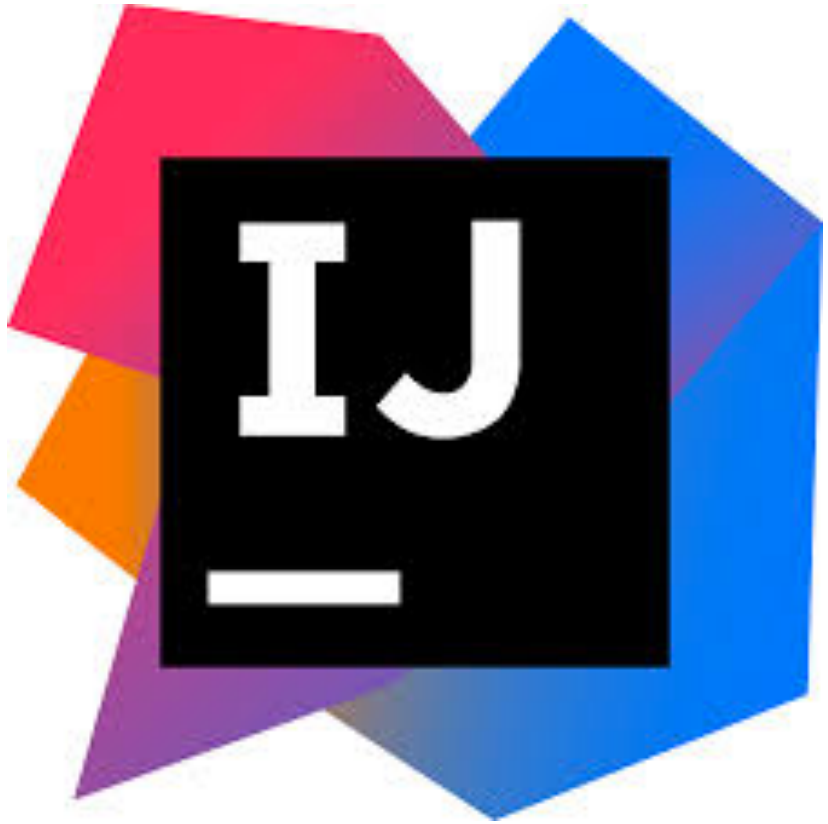
```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webflux</artifactId>
</dependency>
```

- **Setzt automatisch @EnableWebFlux**
- **Lauffähig auf allen Servlet 3.1 Containern, die Non-Blocking IO API unterstützen, aber auch auf Netty oder Undertow**
- **Basis-Technologie: Project Reactor, alternativ kann RxJava konfiguriert werden.**

Spring WebFlux mit klassischen Annotationen

```
@RestController
public class DemoRestController {
    @GetMapping("/helloworld")
    public Mono<String> getGreeting(
        @RequestParam(defaultValue = "World") String name
    ) {
        return Mono.just("Hello")
            .flatMap(s -> Mono
                .just(s + ", " + name + "!\n")
            );
    }
}
```

Kotlin-Extensions & Reactive Anwendungen ohne Annotationen



Ich wechsle mal eben in die IDE

Actuator 2

Technologieunabhängig

Gleichermaßen und gleichartig verfügbar für

- Spring Web MVC
- Spring Web Flux
- Jersey / JAX RS
- JMX

Eigene Endpunkte

Kennzeichnung über @Endpoint

```
@Endpoint(id = "custom")
public class CustomEndpoint {

    private final AtomicInteger cnt =
            new AtomicInteger();

    @ReadOperation
    public String someReadOperation() {
        return "Current value " + cnt.get();
    }

    @WriteOperation
    public String someWriteOperation() {
        return "New value " +
          cnt.incrementAndGet();
    }
}
```

Eigene Endpunkte

Instanziierung mittels Java-Config und...

```
@ManagementContextConfiguration
public class CustomEndpointConfig {

    @Bean
    public CustomEndpoint customEndpoint() {
        return new CustomEndpoint();
    }
}
```

spring.factories

```
org.springframework.boot.actuate.autoconfigure.web.ManagementContextConfiguration = \
  de.springbootbuch.actuators.CustomEndpointConfig
```

Neuer Basispfad und Namen

- Alle Endpunkte befinden sich nun unterhalb von `/actuator` konfigurierbar über `management.endpoints.web.base-path`
- Umbenennungen
  - /trace -> /httptrace
  - /autoconfig -> /conditions
- Entfernt
  - /actuator (als dezidierter Endpunkt)
  - /docs

Neues JSON-Format

Downstream-Services müssen angepasst werden für

- /actuator/env
- /actuator/flyway
- /actuator/httptrace
- /actuator/mappings
- /actuator/metrics (Neues Format für micrometer.io!!)
- /actuator/liquibase

Spring Security 5

Neue Funktionalitäten

- OAuth 2.0 Login ist Kernfunktion von Spring Security 5
- Vollständige Kompatibilität zum Reactive Stack
  - Reactor-Context übernimmt Rolle des Thread-Locals
- Modernisierte Passwortspeicherung

Modernisierte Passwortspeicherung

- Entfernung von `org.springframework.security.authentication.encoding.PasswordEncoder`
  - `ShaPasswordEncoder`
  - `Md5PasswordEncoder`
- Neu `org.springframework.security.crypto.password.*`

Modernisierte Passwortspeicherung

`DelegatingPasswordEncoder`

- Hashing neuer Passwörter
- Validierung von Passwörtern mit unterschiedlichen Hashformaten
- Hash-Format soll in Zukunft einfacher änderbar sein
- Neues Format
  `{id}encodedPassword`
  - Validierung mit Encoder für entsprechende ID
  - Default-Encoder möglich für Passwörter ohne ID
  - Hashing gemäß konfigurierter ID


Magic be gone...

Weniger Autokonfiguration ist mehr

- Feingranulare Autokonfiguration entfällt
- Anpassung über eigene Bean vom Typ `WebSecurityConfigurationAdapter`
  - Achtung: Sobald vorhanden, übernimmt Boot keine Konfiguration mehr

Nicht mehr über Properties konfigurierbar

- **security.basic.authorize-mode**
- **security.basic.enabled**
- **security.basic.path**
- **security.basic.realm**
- **security.enable-csrf**
- **security.headers.cache**
- **security.headers.content-security-policy**
- **security.headers.content-security-policy-mode**
- **security.headers.content-type**
- **security.headers.frame**
- **security.headers.hsts**
- **security.headers.xss**
- **security.ignored**
- **security.require-ssl**
- **security.sessions**

Also quasi nichts mehr ;)

Standardverhalten entspricht „plain“ Spring Security 5

- Schutz aller Endpunkte per Default
- Content-Negotiation
  - Form-Login für Browser-ähnliche Requests
  - Basic-Auth für REST Requests
- Integration mit Spring Data

Spring Data

Die Neuerungen im Überblick

- Upgrade auf Spring Data Kay:
  - Überarbeitung der Repository-API
  - Eigenständige Kotlin-Unterstützung für Datenklassen, die immutable sind
  - Deklaration für Nullability-Contraints
  - Reactive Support, für Stores mit non-blocking APIs
  - Datastore spezifische Aktualisierungen

- Über 550 Changes in 13 Modulen

Überarbeitung der Repository-API

- Neue Methoden-Namen
  - findOne - findById
  - Iterable bei save / delete mit All am Ende
  - ID Typ muss nicht mehr serialisierbar sein

```
interface CrudRepository<T, ID>
          extends Repository<T, ID>
{
    S save(S entity);

    Iterable<S> saveAll(Iterable<S> entities)

    Optional<T> findById(ID id);

    boolean existsById(ID id);

    Iterable<T> findAllById(Iterable<ID> ids);

    void deleteById(ID id);

    void delete(T entity);

    void deleteAll(Iterable<? extends T> entities);
}
```

Komponierbare Repositories

- Repositories können mit Spring Data Kay aus aus unterschiedlichen Fragmenten zusammengestellt werden
- Die Einschränkung auf eine Basis-Implementierung und eine zusätzliche Implementierung wurde aufgehoben

Komponierbare Repositories

```
public interface CustomerRepository extends
    CrudRepository<CustomerEntity, Integer>,
    PersonFragment,
    CustomerBulkLoader {
}

public interface CustomerBulkLoader {
    void loadCustomers(String fileName);
}

public class CustomerBulkLoaderImpl implements CustomerBulkLoader {
    @Override
    public void loadCustomers(String fileName) {
        System.out.println("Loading customers from " + fileName);
    }
}
```

Reactive Support für Stores mit non-blocking APIs

- Auch Spring Data folgt der großen „Reactive Story“ von Spring 5
- Aber: nicht alle Datastores / Technologien haben non-blocking APIs (z.B. JPA / JDBC)
- Aktuell unterstützt Spring Data Kay folgende Technologien im Hinblick auf reactive:
  - Redis
  - MongoDB
  - Couchbase
  - Cassandra

Reactive Support am Beispiel von MongoDB

- **Neuer Starter:**

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-mongodb-reactive</artifactId>
</dependency>
```

- **Repository:**

```
public interface FilmRepository extends
    ReactiveMongoRepository<Film, String>
{
    @Tailable
    Flux<Film> streamAllBy();
}
```

Unterstützung für infinite Streams (wurde von @InfiniteStream umbenannt)

Reactive Support am Beispiel von MongoDB

- Verwendung von Reactive Repositories

```
@RestController
public class FilmRestController {
    private final FilmRepository filmRepository;

    public FilmRestController(FilmRepository filmRepository) {
        this.filmRepository = filmRepository;
    }

    @GetMapping("/api/films")
    public Flux<Film> getAll() {
        return filmRepository
            .findAll(Sort.by("title").ascending());
    }
}
```

Datastore Spezifische Upgrades

- MongoDB:
  - Fluent MongoOperations API
  - MongoDB Collation Support
- Redis:
  - Split der Client Konfiguration in environment- und client-spezifische Teile

Datastore Spezifische Upgrades

- Spring Data für Apache Cassandra:
  - Streamlining des Modul-Layouts
    (merge von spring-cql und spring-data-cassandra)
  - Cassandra Query & Update Objekte
  - Leichtgewichtige Transaktionen

- Elasticsearch:
  - Upgrade auf 5.4
  - Impact auf public APIs (replace von scan() durch scroll()

Todos

Harmonisierung von Konfigurationseigenschaften

- Verringerung der genutzten Präfixe
  - spring.*
  - management.*
  - server.*
    - server.servlet.*
  - logging.*

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

Duales System Servlet und Reactive

- Keine transitive Abhängigkeit mehr auf Spring MVC
  - Geben Sie explizit an, ob Sie MVC oder WebFlux nutzen
- Änderung der Package Strukturen
  - `org.springframework.boot.context.embedded -> org.springframework.boot.web.embbeded`

    Passen Sie entsprechend die Imports und Klassen an

Spring Data generell

- Harmonisierung von Methodennamen
  - insbesondere `findOne` -> `findById`
  - `Optional<T>` als Default-Rückgabewert
  - `save` und `delete` für Iterables sind nun `saveAll` beziehungsweise `deleteAll`

Relationale Datenbanken

- Spring Boot 2 bringt Flyway 5 mit https://flywaydb.org/documentation/releaseNotes#5.0.0
  - Vor dem Upgrade auf Spring Boot 2 auf Flyway 4 aktualisieren
  - Dann erst auf Boot 2 + Flyway 5
- HikariCP per Default

Nicht-relationale Datenbanken

- Lettuce ersetzt Jedis als Default-Redis-Treiber
- ElasticSearch nun mindest in 5.4+
    - Embedded Variante wird nicht mehr unterstützt

Security

- Konfiguration über Properties ist nicht mehr möglich
- Default-Verhalten entspricht nun Spring Security
- OAuth wird aktuell komplett überarbeitet
  - Spring Security Core
  - Spring Security OAuth
  - Spring Cloud Security
  - Spring Boot OAuth2 (Nur für Boot 1.5.x)
    Aktuelle FAQ zu den verschiedenen Modulen
    https://github.com/spring-projects/spring-security/wiki/OAuth-2.0-Features-Matrix
- Passwörter müssen idR aktiv migriert werden

Security (Passwörter)

- Default-Encoder in Boot 1 genutzt:
  - Ihre Passwörter sind als Plain-Text gespeichert. Hashen Sie diese!
  - Zur absoluten Not mit {noop} präfixen oder NoOpPasswordEncoder als Default-PasswordEncoder konfigurieren (Bitte nicht).
- Passwörter sind ohne Salt-Source gehashed
  - Präfix hinzufügen
  - Default-Encoder für Validierung konfigurieren

http://info.michael-simons.eu/2018/01/13/spring-security-5-new-password-storage-format/

Actuator

- Generell: Anpassung von Downstream-Diensten
  - Pfade
    (ID / Name nicht länger konfigurierbar)
  - Formate
  - Schnittstellen
    https://docs.spring.io/spring-boot/docs/current/actuator-api/html/
- Konfiguration
- Security

Actuator (Konfiguration)

Alt	Neu
`endpoints.<id>.*`	`management.endpoint.<id>.*`
`endpoints.cors.*`	`management.endpoints.web.cors.*`
`endpoints.jmx.*`	`management.endpoints.jmx.*`
`management.address`	`management.server.address`
`management.context-path`	`management.server.servlet.context-path`
`management.ssl.*`	`management.server.ssl.*`
`management.port`	`management.server.port`

Quelle: https://github.com/spring-projects/spring-boot/wiki/Spring-Boot-2.0-Migration-Guide#spring-boot-actuator

Actuator (Security)

- Keine spezielle Security-Konfiguration mehr
- Kein „sensitive“-Flag mehr
- Unterscheidung zwischen „enabled“ und „exposed“
- Alle enabled, nur „health“ und „info“ exposed
- Konfiguration technologiespezifisch, white oder blacklist
  - management.endpoints.(web|jmx).exposure.(exclude|include)

Konfiguration von Actuator-Security

```
public class ApplicationWebSecurityConfigurerAdapter
    extends WebSecurityConfigurerAdapter
{
    @Override
    protected void configure(final HttpSecurity http) throws Exception {
        http
            .httpBasic().and()
            .authorizeRequests()
                .requestMatchers(
                      to(
                            HealthEndpoint.class,
                            InfoEndpoint.class,
                            MetricsEndpoint.class
                      )
                )
                .permitAll()
                .requestMatchers(to(EnvironmentEndpoint.class))
                .authenticated();
    }
}
```

Testen

- Upgrade von Mockito 1.x auf 2.x
- JUnit 5 wird unterstützt

Ressourcen

- Slides: speakerdeck.com/michaelsimons
- Spring Boot Buch
  - Begonnen Januar 2017
  - Erscheint ~~Dezember 2017, Januar 2018, Februar 2018~~ ~~demnächst~~ endlich verfügbar 😅
  - @SpringBootBuch // springbootbuch.de

Bildquellen

- Heaven Shall Burn: Michael Plöd
- Geburtstagskuchen: https://www.flickr.com/photos/tillwe/4229605730
- Security: https://unsplash.com/photos/Nfw3-kdOt7o
- Todos: https://www.flickr.com/photos/29853404@N03/4579520419/