# Keynote

## Spring Framework Roadmap (Juergen Hoeller)

* Oracle JDK 🡪 OpenJDK
  + JDK 7, 8, 11, 17 haben LTS
  + Consequence: Update every half year or buy LTS for one of the above
  + Deprecations happen in small releases (e.g. 9 🡪 10)
* Spring Framework 5
  + Java 8 baseline
  + benötigt Java 8 (für Java 6/7 muss man Spring 4.x verwenden)
  + Functional api design
  + Reactive
  + Kotlin capable
* JDK Upgrade Considerations
  + Stay on JVM classpath (don’t use module system)
  + Stay at Java 8 bytecode level (tools often reject unknown bytecode versions)
  + Build against JDK 8, run against JDK 11 🡪 make JDK 11 a deployment options

## Spring Boot 2.0 (Madhura Bhave, Brian Clozel)

* Uses semver
* /actuator/status 🡪 /actuator/heath
* Spring WebFlux support (reactive programming)
* Metrics
  + Micrometer
* Property Binding
  + Can bind to Set<>
  + Uniform format for reading properties
* Rework Security AutoConf
  + Actuator Security detects custom security config

## Functions and Platforms (Dave Syer)

## Serverless

* Dynamic resource utilization (“scale to zero”)
* Billing per message
* Focus on business logic
* Easy integration with platform services

### What is missing?

* Architectural cohesion
* Observability
* On premise and BYO services
* Vendor neutrality

### DevOps platform

* Apps (PAS – Pivotal Application Service)
* Functions (PFS – Pivotal Function Services)
* Containers (PKS – Pivotal Container Service)
* Tied together by e.g. K8s

# Implementing DDD with the Spring Ecosystem (Michael Plöd)

<https://github.com/mploed/ddd-with-spring>

<https://github.com/ddd-by-examples>

<https://github.com/mploed>

<https://speakerdeck.com/mploed>

<https://leanpub.com/ddd-by-example>

“A good developer is like a werewolf: afraid of silver bullets.”

* **D**on’t **R**epeat **Y**ourself (inside a bounded context)
* Bounded context: customer means different things dependent on context

## Building blocks

* Entities are core business objects (not data objects)
* Value object is sth. like “80€”
* Customer is entity or value object – dependent on context
* DDD Entity ≠ JPA Entity
* Aggregates
  + Aggregates group Entities
  + Prefer small aggregates
  + Root entity is the lead in terms of access to the object graph and lifecycle
  + Reference aggregates by identity
  + On TX per aggregate
  + Keep your aggregates Spring free (use POJO)
  + Place aggregates into separate packages (🡪 information hiding)
  + Unit test with ArchUnit (tests Architecture Violation), e.g. “classes annotated with @Aggregate should have \*Aggregate in its name”
  + Place Spring Data Repository into the same package as Entities, but in another module

## Architecture

* Hexagonal
  + For …
* CRUD
  + For …
* Query Driven
  + For bounded contexts that perform complex queries

## Event driven

* Broker
  + AMQP
  + Kafka
  + Spring Cloud Stream
* HTTP Feeds
  + RSS
  + …
* …

# Spring Framework 5 – Feature Highlights & Hidden Gems (Juergen Hoeller)

*Remark: Spring is not an annotation based framework – annotation awareness sits on top of the chore architecture.*

## Baseline upgrades

JDK 8, Bean Validation 1.1 JPA 2.1, JUnit 5, and others

## Component Style

* Composable Annotations (@MyService, defined from several Spring Annotations)

### Functional Style API

* Programmatic bean registration and endpoint registration
  + Provide “instance suppliers” (describe how to create a bean)
* No need for annotations or scanning, even avoiding reflection
* Non-null API design with explicit Nullable declarations

### First-class support for Kotlin

* …

## Reactive Architectures

* Subscriber is in charge of control flow
* Publisher only produces data if subscriber needs it
* Spring MVC on Servlets: Blocking I/O
  + Imperative logic, simple to write and debug
  + JDBC, JPA, blocking deps
* Spring WebFlux on Spring Web API (Reactor, Reactive Streams): Non-blocking I/O
  + …

### WebFlux

* Support for Netty (most powerful) – only in conjunction with WebFlux!
* Mono<> (one or none), Flux<> (multiple / Stream)

## Hidden Gems

* Guarantee non-nullability for input/output unless otherwise declared (e.g. with @Nullable)
* Automatic detection of SLF4J / Logback, Log4J2, fallback to java.util.logging (no need to exclude logging)

# Making microservices micro with Istio Service Mesh & Kubernetes (Ray Tsang)

<http://istio.io>

K8s: Control plane vs. data plane

## Istio

* Complete framework for connection, securing, managing and monitoring services
* Secure and monitor traffic for microservices and legacy services

## Control Plane (Istio Pilot)

* Service to service comm
* Routing rules
* Retries
* Circuit breaker
* Performance monitor
* Tracing

Ribbon, Hystrix, Metrics, Tracer going from Microservice to Mesh (providing one proxy per service)

* Aspect oriented programming for services

**Not ready for production yet.**

# REST beyond the obvious – API design for ever evolving systems (Oliver Gierke)

<https://speakerdeck.com/olivergierke/rest-beyond-the-obvious-api-design-for-ever-evolving-systems>

## Stefan Tilkov: Microservice Patterns & Antipatterns – Anemic Service

Levels from top to bottom (coupling in lower levels is worse):

* Presentation
* Process Flow: Useful and specific
* Domain Logic: Spring Data REST; reusable, but low level
* Data: CRUD via HTTP / JDBC in disguise

**Ironically, the more generic your API, the more coupling you get.**

<https://en.wikipedia.org/wiki/Connascence>

Instead of asking “How do I version my API?” ask “How can I build my system, so that changes do not break the system?”

It is as important to know what not to document as what to document. I.e. do not document business logic – if you have to, remove the need of it in the API.

* HATEOS to the rescue
* Removes need of business logic in clients
* Increases the amount of protocol knowledge, decreases the coupling

# What's New in Spring Boot 2.0 (Madhura Bhave)

## Spring Framework 5

Reactive support

Servlet Stack: Servlet Container, Servlet API, Spring MVC

Reactive Stack: Netty, Servlet 3.1, Reactive HTTP layer, Spring WebFlux

## Micrometer support

Auto-config for metrics instrumentation lib­­­­­­

## Configuration

* Duration
* Support for Set<>
* Bugfixes for List<>
* Exposition through /actuator/env/<variable>
  + Default format: alphanum, lower-case, hyphens (a1-b)
* Support for arrays
  + my\_var\_1 🡪 my\_var[1]
  + my\_var\_12 🡪 my\_var[1][2]
* Migration from 1.x 🡪 2.x: org.springframework.boot:spring-boot-properties-migrator

## Actuator endpoints

* Support for custom actuators got easier: one endpoint for Mvc and Jmx
* New Annotations for easier access to actuator (write/read)

## Security

* Simpler security auto-conf
* Auto-conf backs-off easily
* Adding custom security is easy
* No ordering issues with WebSecurityConfigurerAdapter
* RequestMatchers for static resources and actuators
* Auto-conf for reactive security
* Support for OAuth 2.0 (through Spring Security 5.0)
  + <https://github.com/spring-projects/spring-security-oauth2-boot>

# Dynamic Configuration Management in Microservice Architecture with Spring Cloud (Bartłomiej Słota)

<https://github.com/bslota/cloud-config>

<https://github.com/bslota/config-repository>

* Externalization
* Deployment bundle
* Environment independent
* Scalability
* Security (Production Dbs)
* Traceability

Config Server + Git + message-bus to the rescue

### Config Server

1. pom.xml: org.springframework.cloud:spring-cloud-config-server
2. @EnableConfigServer class application {}
3. application.yml

spring.cloud.config.server.git

uri: http://example.com

clone-one-start: true

search-paths: config/{application}

Notes

* config is identified by application, profile, label
* Config Server endpoints: /{application}-{profiles}
* Always scale config-server for resilience!

## Client

pom.xml: org.springframework.cloud:spring-cloud-starter-config

application.yml:

spring.cloud-config

label: “springio”

uri: localhost:8888 (default strategy 🡪 no eureka)

discovery: true (eureka-strategy)

service-id: config-service (eureka-strategy)

## Provisioning

* Spring Cloud Stream abstracts Message Broker. RabbitMQ: lightweight; Kafka: mighty.
* For automatic provisioning of config changes use a message broker.
* pom.xml (Server): spring-cloud-config-monitor
* Fully automation: Web-hook on github calling config-server update

**Drawbacks**: continuous integration (extra place for configurations!)

# Testing every level of your spring microservices application (Jeroen Sterken / Kristof Van Sever)

<https://github.com/faros/bdd-cucumber.git>

<https://github.com/faros/spring-cloud-contract.git>

## JUnit 5

### Components

* JUnit Platform
* JUnit Jupiter
* JUnit Vintage

### Features

* Multiple assertions (multiple args)
* Lambda support
* @DisplayName at beginning of test
* @...Source

## Cucumber

* Behaviour driven testing
* Uses BDD standards
* Given – When – Then – And/But

# Serverless Spring (David Syer)

<https://github.com/dsyer/spring-boot-micro-apps>

<https://github.com/dsyer/spring-cloud-function>

<https://github.com/dsyer/spring-boot-thin-launcher>

<https://github.com/spring-cloud/spring-cloud-function/>

## Providers

* AWS, Azure: native support for native Java support
* Google: no native Java support

## Functions

Can use java.util.functions or Spring Cloud Function.

### Spring Cloud Function

@SpringBootApplication

class Application {

@Bean

public void Function<String, String> uppercase() {

}

}

* @Beans of type Function, Consumer and Supplier, with Flux, Mono, Publisher)
* Adapters for several providers

## Startup

Can startup fast (functions only or micro / mini examples: < 1s) 🡪 **classes loaded is important**!

Sprint Boot 1.5 is a little faster than 2.0.

Warm startup is 10x faster than cold startup (class loader isolation).

On Amazon: AWS Lambda containers are slow – 6s

AWS let’s you only control the memory – coupled to this is the CPU usage 🡪 bad: not much memory needed, but CPU should not be throttled!

## Cost

Cost for running a lambda (a few seconds is enough!) goes slightly up when allowing more memory, but not much.

## Riff

<https://github.com/projectriff/riff>

<https://projectriff.io/>

# Just one more thing with Spring in IntelliJ IDEA (Yann Cébron / Stéphane Nicoll)

* Configure Facets unless using Spring Boot or Spring MVC
* Find Symbol > Bean Name or Endpoint starting with “/”
* Support for Spring with Kotlin is ongoing
* In-built support for spring-boot-properties-migrator
* **Devtools support for update of static resources**
* Run Dashboard

# Spring Cloud Stream: Developer Recipes, Tips and Tricks (Oleg Zhurakousky)

* Supports consumer groups, stateful partitions (otherwise only available with Kafka – supported within RabbitMQ through Spring Cloud Stream)
* Imperative and declarative programming model (declarative is planned to becomeq default)

# Micrometer: New insights into your Spring Boot application (Michael Simons)

* Like SLF5J but for metrics
* Dependency: org.springframework.boot:spring-boot-starter-actuator
* Expose metrics to activate it
* New: **real drill down**!
* Own metrics can be tagged
* Registries collect, store and export metrics
  + SimpleMeterRegistry
  + CompositeMeterRegistry
  + Metrics.globalRegistry (empty composite by default)

io.micrometer:micrometer-core

## Metrics

* System metrics (Classloader, JVM Memory/Garbage/Thread, Processor) – auto
* Application metrics (Spring MVC, Spring WebFlux, HTTP Client, Spring Integration, Spring MQ)
* Domain metrics (= KPIs) (own metrics)

## Supported monitoring systems

* Prometheus (poll via /actuator/prometheus)
* Graphite (push)
* InfluxDB (push)
* …

# Building and running Spring Cloud-based microservices on AWS ECS (Joris Kuipers)

* ECS runs on top of EC2 without extra tops (you only pay for the actual EC2)
* Can scale up num of containers on alarms like % CPU
* Scaling down is complicated: do not kill an EC2 with active containers
  + Use AWS lambda 🡪 but we don’t want to organize that our own
* Better alternative: AWS fargate (available in EU/Ireland since April 2018)
* AWS parameter store (key/value store)
  + Supports encryption, versioning
  + Access control through IAM
  + Hierarchical paths
  + No support from Spring Cloud yet 🡪 write own PropertySourceLocator or CompositePropertySource (copy from Consul support)
* Since March 2018, ECS support docker health checks 🡪 no ELB required
* Since March 2018, ECS has a service discovery

# Wrap Up

1. Kotlin – Eclipse and IntelliJ Support, can be mixed with Java, built-in Lombok, easily readable
2. Config Server + Git + message-bus
3. Spring Boot 2.0 – easy migration, mainly application.yml to be migrated
4. Metrics – updated in Boot 2.0, easy integration with Prometheus

# Spring I/O Nacharbeit

* Kotlin testen
* Spring Boot 2.0 & Reactive testen
* Neuen IntelliJ Idea HTTP Rest Client testen / verfolgen