Modern Enterprise Java in 2018

Spring Framework 5 & Spring Boot 2.0

Oliver Gierke olivergierke ogierke@pivotal.io

Sample Code

https://github.com/olivergierke/spring-five-functional-reactive

Spring Framework 5

Infrastructure

Infrastructure

- **JavaSE 8 baseline**
  - JDK 9 compatibility through automatic modules and JDK 9 base CI builds
- **JavaEE 7 baseline**
  - Servlet API 3.1
  - JPA 2.1 (OpenJPA support dropped)
  - JMS 2.0
  - Bean Validation 1.1
- **Support for selected JavaEE 8 APIs**
  - Servlet 4
  - Bean Validation 2.0
  - JSON Binding API

JDK 9 Support

- **Many general JVM improvements**
  - Compact Strings
  - G1 by default
  - TLS protocol stack
- **Jigsaw**
  - Stable automatic module names
  - We currently recommend to still run in classpath mode

HTTP/2 Support

- **Mostly via Servlet 4.0**
  - HTTP/2 support in Servlet containers
  - `PushBuilder` API

JUnit 5

- **Test context framework support**
  - for both JUnit 4 and JUnit 5
- **Dependency injection capabilities inspired by Spring**
  - Constructor injection
  - Parameter injection á la Spring MVC
- **Meta annotation support inspired by Spring**

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(…)
public class MyTest { … }
```

JUnit 5

- **Test context framework support**
  - for both JUnit 4 and JUnit 5
- **Dependency injection capabilities inspired by Spring**
  - Constructor injection
  - Parameter injection á la Spring MVC
- **Meta annotation support inspired by Spring**

```
@SpringJUnitConfig(…)
public class MyTest { … }
```

Functional container extensions

```
GenericApplicationContext ctx = new GenericApplicationContext();
```

Functional container extensions

```
GenericApplicationContext ctx = new GenericApplicationContext();

// Default constructor via reflection
ctx.registerBean(First.class);
```

Functional container extensions

```
GenericApplicationContext ctx = new GenericApplicationContext();

// Default constructor via reflection
ctx.registerBean(First.class);

// Explicit constructor via Supplier
ctx.registerBean(Second.class,
  () -> new Second(ctx.getBean(First.class)));
```

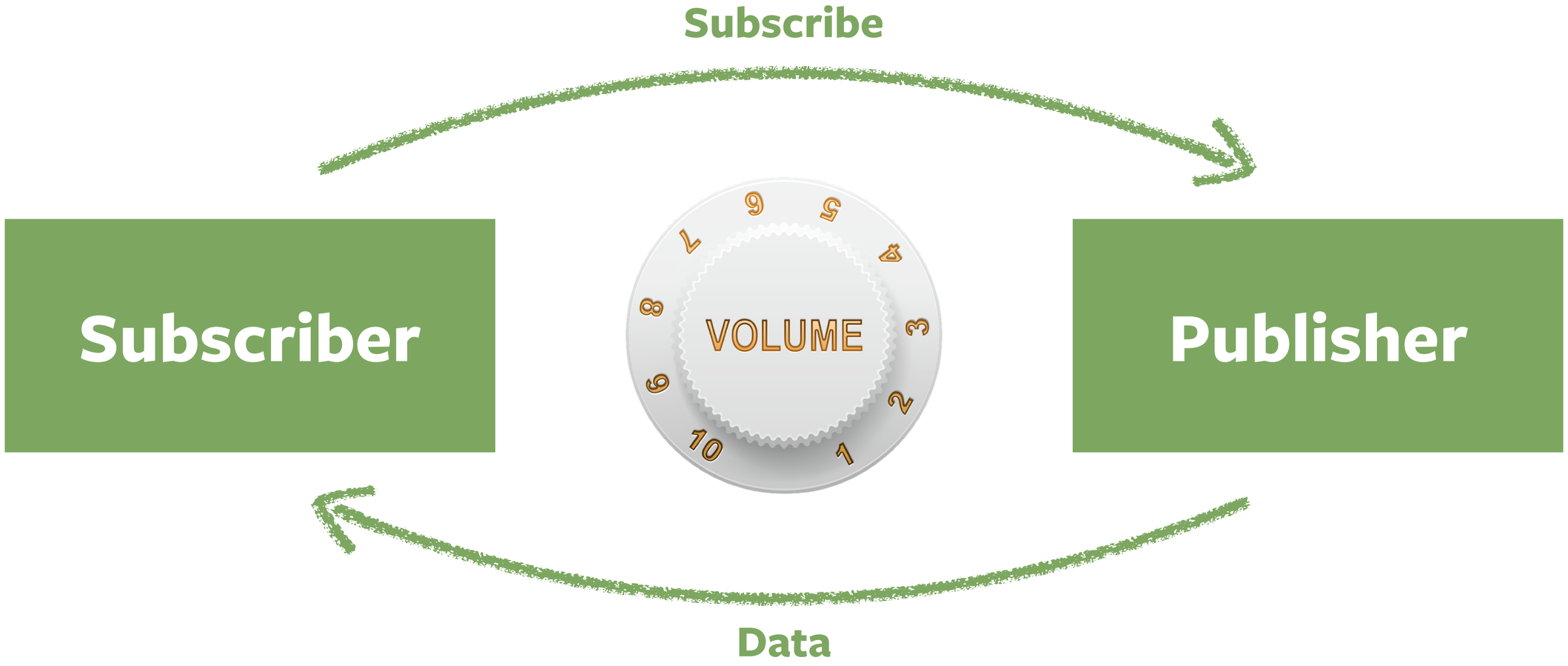
Functional container extensions

```
GenericApplicationContext ctx = new GenericApplicationContext();

// Default constructor via reflection
ctx.registerBean(First.class);

// Explicit constructor via Supplier
ctx.registerBean(Second.class,
  () -> new Second(ctx.getBean(First.class)));

// Explicit constructor plus BeanDefinition customization
ctx.registerBean(Third.class,
  () -> new Third(ctx.getBean(First.class)),
  bd -> bd.setLazyInit(true));
```

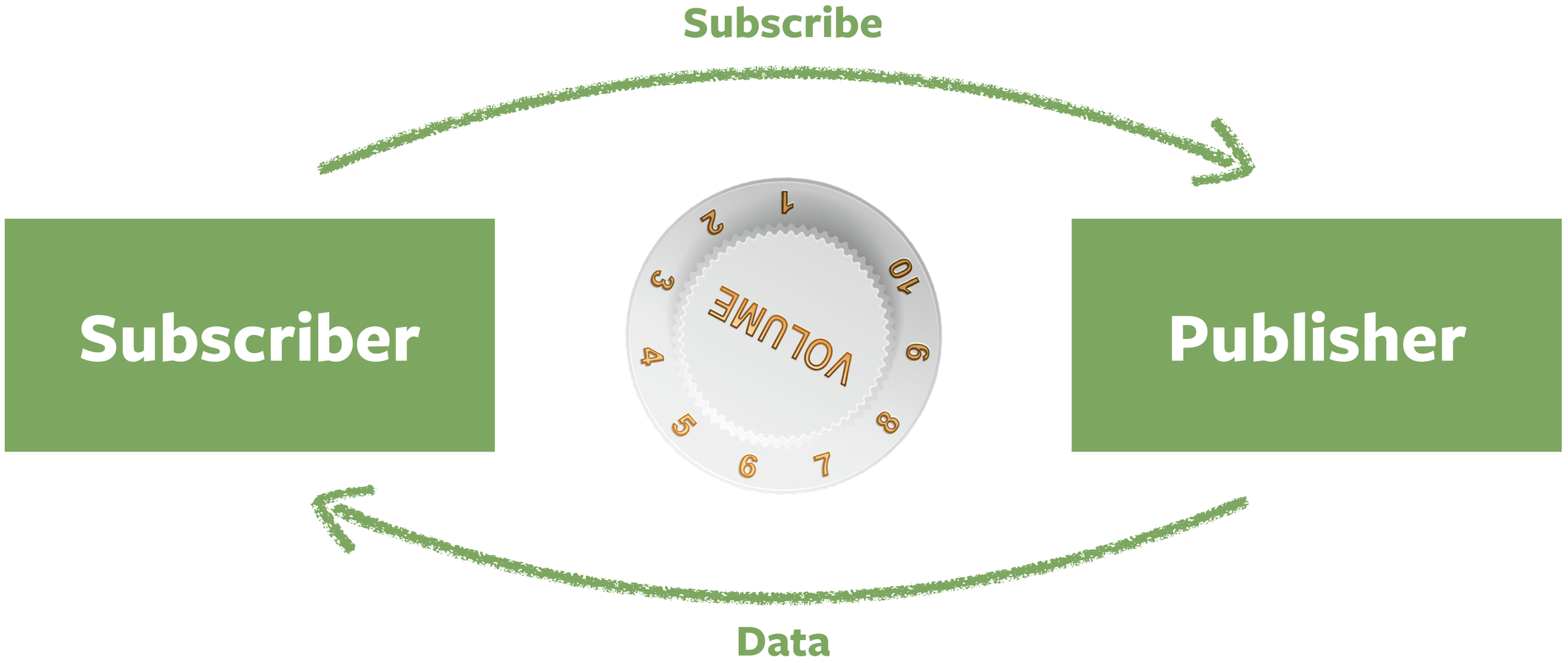
Functional container extensions

Reactive Programming

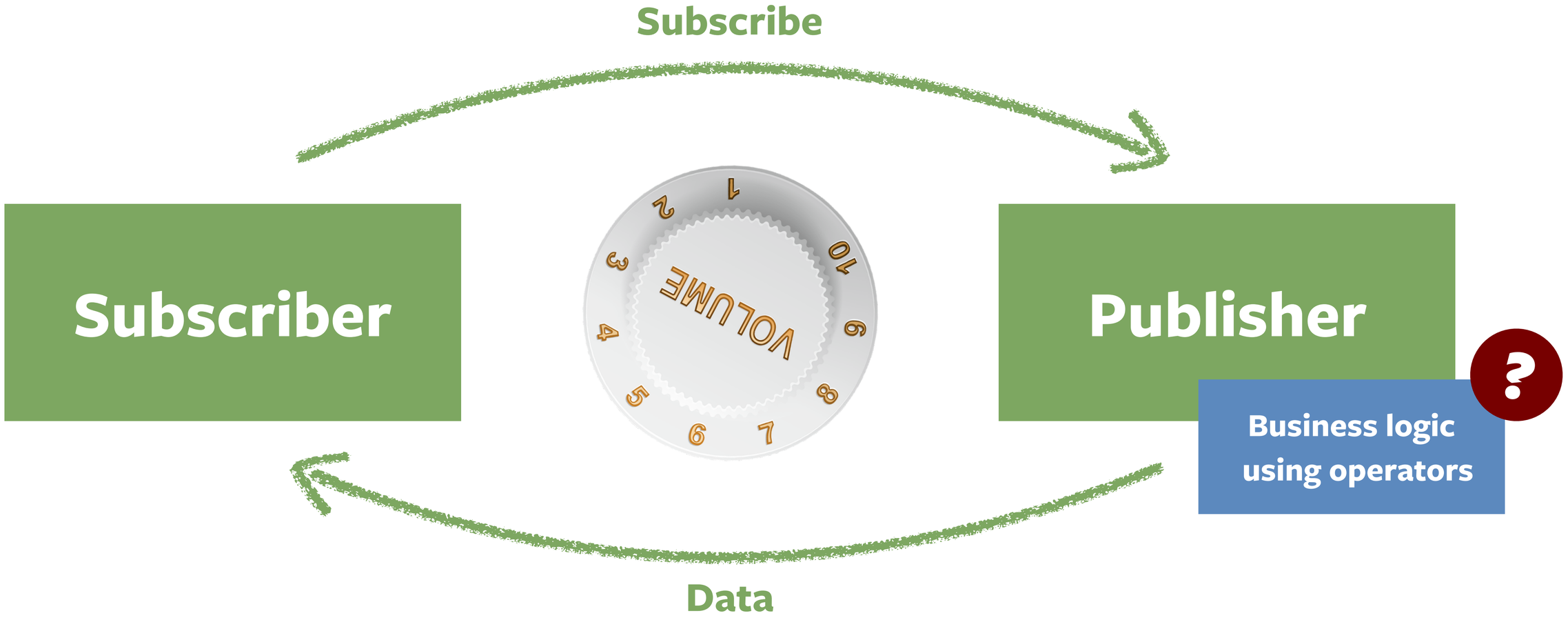
A quick detour



Reactive fundamentals



Reactive fundamentals



Reactive fundamentals

```
Mono.just("Hello")
   .map(word -> word.concat(" World!"))
   .subscribe(System.out::println);

Flux.just("Hello", "World")
   .flatMap(word -> Flux.fromArray(word.split("")))
   .subscribe(System.out::println);
```

Reactive programming 101

```
someMono
   .map(word -> word.concat("World"))
   .…

someFlux
   .flatMap(word -> Flux.fromArray(word.split("")))
   .…
```

Reactive programming 101

```
someMono // Created by the infrastructure
  .map(word -> word.concat("World"))
  .… // Handled by the infrastructure

someFlux // Created by the infrastructure
  .flatMap(word -> Flux.fromArray(word.split("")))
  .… // Handled by the infrastructure
```

Reactive programming 101

```
someMono // Created by the infrastructure
  .map(word -> word.concat("World"))
  .… // Handled by the infrastructure

someFlux // Created by the infrastructure
  .flatMap(word -> Flux.fromArray(word.split("")))
  .… // Handled by the infrastructure
```

Reactive programming 101

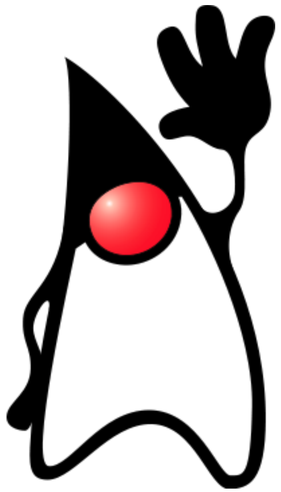
Reactive programming

- **Subscription based**
  - Push VS. pull model
  - Backpressure blurs the line
- **Two-phase execution**
  - First: to build up the pipeline. Second: once the data starts flowing
  - Similar to programming model of the Java 8 `Stream` API
- **Lazy execution**
  - Nothing happens until someone subscribes
- **Backpressure**
  - Subscriber signals capability to handle the amount of data

Reactive programming

- **Optimized for resource usage, scalability and stability**
  - Not necessarily faster
- **Optimized for „mean time to first result“**
- **Challenge: debugging and tests**
- **Needs non-blocking across the entire stack**
  - Usage of blocking APIs (e.g. JDBC) severely complicates the picture and limits efficiency

Spring WebFlux



Spring MVC

Servlet API

Servlet container

Spring WebFlux(.fn)

New!

Spring Web Reactive API

New!

Netty, Undertow, Servlet 3.1

Runtime stacks

```
@Controller
class ReactiveUserController {

  private final UserRepository users;

  // Constructor for Dependency Injection

  @GetMapping("/users")
  Flux<User> getUsers() {
    return this.repository.findAll();
  }
}
```

WebFlux Controller

```
@Controller
class ReactiveUserController {

  private final UserRepository users;

  // Constructor for Dependency Injection

  @GetMapping("/users")
  Flux<User> getUsers() {
    return this.repository.findAll();
  }
}
```

WebFlux Controller

```
@Controller
class ReactiveUserController {

  // … continued

  @PostMapping("/users")
  Mono<User> createUser(@RequestBody Mono<User> user) {
    return user.flatMap(this.repository::save);
  }

  @GetMapping("/users/{id}/")
  Mono<User> getUser(@PathVariable Long id) {
    return this.repository.findById(id);
  }
}
```

WebFlux Controller

```
@Controller
class ReactiveUserController {

  // … continued

  @PostMapping("/users")
  Mono<User> createUser(@RequestBody Mono<User> user) {
    return user.flatMap(this.repository::save);
  }

  @GetMapping("/users/{id}/")
  Mono<User> getUser(@PathVariable Long id) {
    return this.repository.findById(id);
  }
}
```

WebFlux Controller

Spring WebFlux.fn

```
@Component
class FunctionalUserController {

  private final UserRepository repository;

  // Constructor for Dependency Injection

  Mono<ServerResponse> getUser(ServerRequest request) {

    Mono<User> user = Mono.just(request.pathVariable("id"))
      .flatMap(this.repository::findById);

    return ServerResponse.ok().body(user, User.class);
  }
}
```

Functional controller

```
@Component
class FunctionalUserController {

  private final UserRepository repository;

  // Constructor for Dependency Injection

  Mono<ServerResponse> getUser(ServerRequest request) {

    Mono<User> user = Mono.just(request.pathVariable("id"))
      .flatMap(this.repository::findById);

    return ServerResponse.ok().body(user, User.class);
  }
}
```

Functional controller

```
@Component
class FunctionalUserController {

  // … continued

  Mono<ServerResponse> getUsers(ServerRequest request) {

    Flux<User> users = this.repository.findAll();
    return ServerResponse.ok().body(users, User.class);
  }
}
```

Functional controller

```
@SpringBootApplication
class ApplicationConfiguration {

  @Bean
  RouterFunction<?> routes(FunctionalUserController controller) {

    return RouterFunctions
      .route(GET("/users"), controller::getUsers)
      .andRoute(GET("/users/{id}"), controller::getUser);
  }
}
```

Router Functions

Kotlin Extensions

```
// In GenericApplicationContextExtension.kt in spring-context

inline fun <reified T : Any> GenericApplicationContext.registerBean(
    vararg customizers: BeanDefinitionCustomizer,
    crossinline function: (ApplicationContext) -> T) {

  registerBean(T::class.java, Supplier { function.invoke(this) }, *customizers)
}

// Allows …

context.registerBean { Foo() }
```

ApplicationContext extensions

```
val router = router {

  val users = …

  accept(TEXT_HTML).nest {
     "/" { ok().render("index") }
     "/sse" { ok().render("sse") }
     "/users" {
        ok().render("users", mapOf("users" to users.map { it.toDto() }))
     }
  }
  ("/api/users" and accept(APPLICATION_JSON)) {
     ok().body(users)
  }
}
```

Functional routing with Kotlin

```
val databaseContext = beans {

    bean<UserEventListener>()

    bean<UserRepository>()

    environment( { !activeProfiles.contains("cloud") } ) {
        bean {
            CommandLineRunner { initializeDatabase(ref()) }
        }
    }
}

fun initializeDatabase(userRepository: UserRepository) { // ... }
```

Bean definitions – Kotlin style

Miscellaneous

- **Component indexing for faster startup**
  - APT processor included in `spring-context-indexer`
  - Creates index in `META-INF/spring.components`
- **Nullability annotations in the entire codebase**
- **@Nullable to indicate optionality at injection points**
- **Data binding against immutable objects**
  - Via constructor argument resolution and support for Kotlin / Lombok
- **WebClient as reactive alternative to RestTemplate**

Spring Boot 2.0

Core themes

- **Upgrade to Java 8 and Spring Framework 5**
  - Includes upgrades to all ecosystem projects based on those versions
- **Infrastructure upgrades**
  - Jetty 9.4
  - Tomcat 8.5
  - Hibernate 5.2
  - Hikari connection pool (previously Tomcat)
- **Reactive web test support**
- **OAuth 2.0 support moved to Spring Security**
- **Tweaked defaults**

Core themes

- **Actuator refactorings**
  - Resources moved to `/application/...`
  - All secured by default
  - Micrometer support
- **Actuator customization API**
  - To abstract over Spring MVC, Spring WebFlux, JAX-RS and JMX

Pruning

- General deprecations present in 1.5
- CRaSH project, Spring Loaded

Related talks

10:45 - 11:45 — Going reactive with Spring Data

Jens Schauder (Spring Data team member), Raum: Partenkirchen

12:00 - 13:00 — The Beginner’s Guide to Spring Cloud

Spencer Gibb (Spring Cloud co-lead), Raum: Partenkirchen

Thank you!
Questions?

Oliver Gierke olivergierke ogierke@pivotal.io

Resources

Resources

Spring Framework 5 – Migration guide

https://github.com/spring-projects/spring-framework/wiki/What's-New-in-Spring-Framework-5.x

https://github.com/spring-projects/spring-framework/wiki/Upgrading-to-Spring-Framework-5.x

Spring Framework 5 – FAQ

https://github.com/spring-projects/spring-framework/wiki/Spring-Framework-5-FAQ

Resources

Blog series on Reactive Programming

https://spring.io/blog/2016/06/07/notes-on-reactive-programming-part-i-the-reactive-landscape

Reactor – Project homepage

https://projectreactor.io/

Resources

Spring Boot 2.0 release notes

https://github.com/spring-projects/spring-boot/wiki/Spring-Boot-2.0-Release-Notes

Spring Boot 1.5 -> 2.0 migration guide

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

Micrometer documentation

http://micrometer.io/docs

Documentation of Spring Boot metrics

https://docs.spring.io/spring-boot/docs/2.0.x/reference/htmlsingle/#production-ready-metrics

Spring Boot 2.0 Actuators (blog post)

https://spring.io/blog/2017/08/22/introducing-actuator-endpoints-in-spring-boot-2-0