Pivotal

Functional web applications with Kotlin and Spring 5

Sébastien Deleuze
@sdeleuze

Step 1 Switch from Java to Kotlin
Step 2 Upgrade to Spring 5
Step 3 Migrate to WebFlux
Step 4 WebFlux/Bean Kotlin DSLs

Step 1 Switch from Java to Kotlin

Step 2 Upgrade to Spring 5

Step 3 Migrate to WebFlux

Step 4 WebFlux/Bean Kotlin DSLs

Today

Spring Boot 1

Spring MVC

8

Today

Spring Boot 1

Spring MVC

Tomorrow Step 1

Spring Boot 1

Spring MVC

Kotlin 1.1

Kotlin

- ➔ Created by **JetBrains**
- ➔ Elegant and pragmatic language
- ➔ Concise code
- ➔ Simple and easy to learn
- ➔ Embrace both functional and object oriented programming
- ➔ Very good Java interoperability

How does it compare with …

Keep the good parts of Java and toss the bad ones. Kotlin is as statically typed as Java, and more suitable for functional programming than Java 8.

Kotlin “stole” some good ideas to Groovy, but is statically typed by design. It also produces cleaner and smaller bytecode.

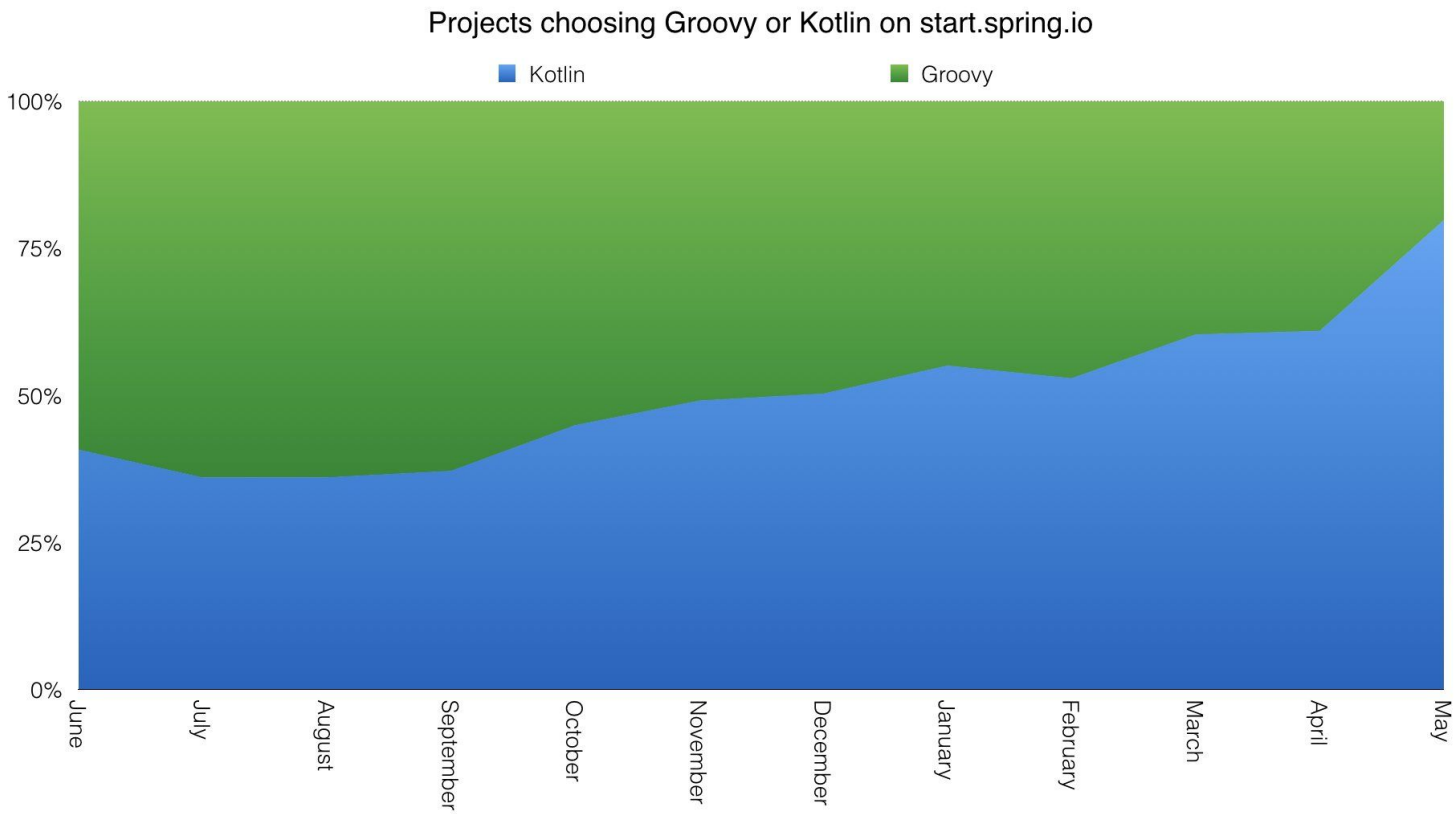
Kotlin is a little bit less powerful than Scala, but much simpler, more pragmatic while still elegant.

Kotlin and Swift are very close and both great languages.

Increasing adoption on GitHub

Pivotal

Increasing adoption on start.spring.io



Pivotal

Google now officially supports Kotlin on Android!

Pivotal

Why should you switch to Kotlin?

This section is freely inspired from Magnus Vinther “Why you should totally switch to Kotlin” blog post available at https://goo.gl/Gwxqw3

Familiar Syntax

```
class Foo {

  val b: String = "b"      // val means unmodifiable
  var i: Int = 0           // var means modifiable

  fun sum(x: Int, y: Int): Int {
     return x + y
  }

  fun maxOf(a: Float, b: Float) = if (a > b) a else b

}
```

Type Inference

```
val a = "abc"                          // type inferred to String
val b = 4                              // type inferred to Int

val c: Double = 0.7                    // type declared explicitly
val d: List<String> = ArrayList()      // type declared explicitly
```

String interpolation

```
val x = 4
val y = 7
print("sum of $x and $y is ${x + y}")  // sum of 4 and 7 is 11
```

Smart Casts

```
if (obj is String) {
  print(obj.toUpperCase())      // obj is now known to be a String
}
```

Intuitive Equals

```
val john1 = Person("John")
val john2 = Person("John")

john1 == john2    // true  (structural equality)
john1 === john2   // false (referential equality)
```

Default parameters

```
fun build(title: String, width: Int = 800, height: Int = 600) {
  ...
}

build("foo")              // width = 800, height = 600
build("foo", 400)         // width = 400, height = 600
build("foo", 400, 450)    // width = 400, height = 450
```

Named parameters

```
fun build(title: String, width: Int = 800, height: Int = 600) {
  ...
}

build("foo", 800, 300)                    // equivalent
build(title = "foo", height = 300)        // equivalent
build(height = 300, title = "foo")        // equivalent
```

The when expression

```
when (x) {
  1 -> print("x is 1")
  2 -> print("x is 2")
  3, 4 -> print("x is 3 or 4")
  in 5..10 -> print("x is 5, 6, 7, 8, 9, or 10")
  else -> print("x is out of range")
}
```

Properties

```
class Frame {
  var width: Int = 800
  var height: Int = 600

  val pixels: Int
     get() = width * height
}
```

Extension Functions

```
// User defined extensions
fun String.format(): String {
  return this.replace(' ', '_')
}
val formatted = str.format()

// Kotlin standard library is provided with built-in JDK extensions
str.removeSuffix(".txt")
str.capitalize()
str.substringAfterLast("/")
str.replaceAfter(":", "classified")
```

Null Safety

```
var a: String = "abc"
a = null                    // compile error

var b: String? = "xyz"
b = null                    // no problem

val x = b.length            // compile error: b might be null
val y = b?.length           // type of y is nullable Int

val name = ship?.captain?.name ?: "unknown" // Chainable safe calls
```

Better Lambdas

```
val sum = { x: Int, y: Int -> x + y }   // type: (Int, Int) -> Int
val res = sum(4,7)                      // res == 11

numbers.filter({ x -> x.isPrime() })    // equivalent
numbers.filter { x -> x.isPrime() }     // equivalent
numbers.filter { it.isPrime() }         // equivalent

// Allow to write concise functional code
persons
     .filter { it.age >= 18 }
     .sortedBy { it.name }
     .map { it.email }
     .forEach { print(it) }
```

And much more

- ➔ Data classes
- ➔ Type aliases
- ➔ Co-routines
- ➔ Reified type parameters
- ➔ Underscore for unused parameters
- ➔ etc.

https://start.spring.io

SPRING INITIALIZR bootstrap your application now

Generate a Maven Project with Java and Spring Boot 1.5.4

Java
Kotlin
Groovy

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

Generate Project alt + ⏎

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

Pivotal

Spring MVC controller written in Kotlin

Classes and functions are public by default

Constructor injection without @Autowired if single constructor

```
@RestController
class UserController(val repo: UserRepository) {
  @GetMapping("/user/{id}")
  fun findOne(@PathVariable id: String) = repo.findOne(id)

  @GetMapping("/user")
  fun findAll() = repo.findAll()

  @PostMapping("/user")
  fun save(@RequestBody user: User) = repo.save(user)
}

interface UserRepository {
  fun findOne(id: String): User
  fun findAll(): List<User>
  fun save(user: User)
}
```

Static typing + type inference

kotlin-spring Gradle and Maven plugin

Automatically open Spring annotated classes and methods

Without kotlin-spring plugin

```
@SpringBootApplication
open class Application {

  @Bean
  open fun foo() = ...

  @Bean
  open fun bar() = ...

}
```

With kotlin-spring plugin

```
@SpringBootApplication
class Application {

  @Bean
  fun foo() = ...

  @Bean
  fun bar() = ...

}
```

kotlin-noarg Gradle and Maven plugin

Create a synthetic constructor with no argument, useful with JPA, Spring Data ...

```
noArg {
    annotation("org.springframework.data.mongodb.core.mapping.Document")
}
```

```
@Document
data class User(
     @Id val login: String,
     val firstname: String,
     val lastname: String,
     val email: String,
     val company: String? = null,
     val description: Map<Language, String> = emptyMap(),
     val logoUrl: String? = null,
     val role: Role = Role.ATTENDEE)
```


Pivotal

Gradle build files written in Kotlin

```
version = "1.0.0-SNAPSHOT"

repositories {
    mavenCentral()
    maven { setUrl("https://repo.spring.io/milestone") }
    maven { setUrl("https://repo.spring.io/snapshot") }
}

tasks.withType<KotlinCompile> {
    kotlinOptions {
        jvmTarget = "1.8"
    }
}

node {
    version = "6.9.4"
    de
    defaultTasks(vararg p0: String!)                                  Unit
    dependencies(p0: Closure<(raw) Any!>!)                            Unit
    defaultTasks (from getDefaultTasks… (Mutable)List<String!>!
    dependencies (from getDependencies())     DependencyHandler!
    depth (from getDepth())                                           Int
    description (from getDescription()/setDescription(… String!
    delete(vararg p0: Any!)                                           Boolean
    delete(p0: Action<in DeleteSpec!>!)                               WorkResult!
    delete {...} (p0: (DeleteSpec!.() -> Unit)!)                      WorkResult!
    depthCompare(p0: Project!)                                        Int
Dot, space and some other keys will also close this lookup and be inserted into editor >>
    compileOnly("org.springframework:spring-context-indexer")
    compile("org.springframework.boot:spring-boot-starter-data-mongodb-reactive")
    testCompile("org.springframework.boot:spring-boot-starter-test")
    runtime("de.flapdoodle.embed:de.flapdoodle.embed.mongo")
    compile("com.samskivert:jmustache:1.13")
    compile("com.atlassian.commonmark:commonmark:0.9.0")
    compile("com.atlassian.commonmark:commonmark-ext-autolink:0.9.0")
```

Pivotal

Step 1 Switch from Java to Kotlin

Step 2 Upgrade to Spring 5

Step 3 Migrate to WebFlux

Step 4 WebFlux/Bean Kotlin DSLs

Step 1

Spring Boot 1

Spring MVC

Kotlin 1.1

Step 2

Spring Boot 2

Spring MVC

Kotlin 1.1

Spring ❤️ Kotlin

And officially supports it

Introducing Kotlin support in Spring Framework 5.0

À l'origine en anglais

Introducing Kotlin support in Spring Framework 5.0

Following the Kotlin support on start.spring.io we introduced a few months ago, we have continued to work to ensure that Spring and Kotlin play well together. One of the key strengths of Kotlin is...

spring.io

RETWEETS 214 J'AIME 231

15:06 - 4 janv. 2017

Pivotal

Leveraging Kotlin nullable information

To determine `@RequestParam` required attribute, also works for `@Autowired`

```
// "GET /foo" and "GET /foo?bar=baz" are allowed
@GetMapping("/foo")
fun foo(@RequestParam bar: String?) = ...
```

```
// "GET /foo?bar=baz" is allowed and "GET /foo" will return an error
@GetMapping("/foo")
fun foo(@RequestParam bar: String) = ...
```

Spring provides Kotlin specific API via extensions

- Spring Framework 5
  - ApplicationContext
  - Spring MVC
  - Spring WebFlux
  - RestTemplate
  - JDBC
- Spring Boot 2
- Spring Data Kay release
- Reactor 3.1

Extension example : reified type parameters

Goodbye type erasure, we are not going to miss you at all!

```
// Java
List<User> users = mongoTemplate.findAll(User.class);
```

```
// Spring Data will provide this kind of Kotlin extension, see DATAMONGO-1689
inline fun <reified T : Any> MongoOperations.findAll(): List<T> =
    findAll(T::class.java)
```

```
// So in Kotlin we just have to write
val users: List<User> = mongoTemplate.findAll()
// Or
val users = mongoTemplate.findAll<User>()
```

Null safety of Spring APIs

Leveraging JSR 305 meta-annotations for generic tooling support

```java
package org.springframework.lang;

@Target({METHOD,PARAMETER})
@Retention(RUNTIME)
@Documented
@Nonnull(when = MAYBE)
@TypeQualifierNickname
public @interface Nullable {
}
```

```java
package org.springframework.lang;

@Target(PACKAGE)
@Retention(RUNTIME)
@Documented
@Nonnull
@TypeQualifierDefault({METHOD,PARAMETER})
public @interface NonNullApi {
}
```

Null safety of Spring APIs

See SPR-15540

```java
// package-info.java
@NonNullApi
package org.springframework.cache;

import org.springframework.lang.NonNullApi;
```

```java
// Cache.java
import org.springframework.lang.Nullable;

public interface Cache {

    @Nullable
    <T> T get(Object key, @Nullable Class<T> type);

}
```

Null safety of Spring APIs

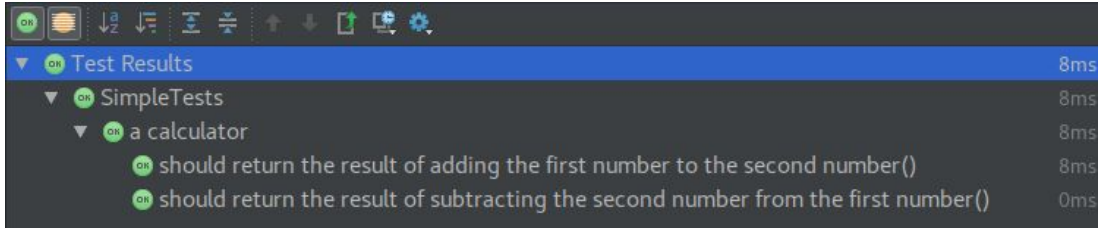
Useful for both Java and Kotlin developers

- ➔ Comprehensive null-safety of Spring API in Kotlin (KT-10942)
- ➔ Warnings in Java IDE like IDEA (2017.1.4+) or Eclipse
- ➔ SonarSource already plan to leverage it
- ➔ Other Spring projects may provide null-safe API as well
- ➔ Other Java libraries may follow the same path ...

Specification like tests with Kotlin and JUnit 5

```kotlin
class SimpleTests {

  @Nested
  @DisplayName("a calculator")
  inner class Calculator {
     val calculator = SampleCalculator()

     @Test
     fun `should return the result of adding the first number to the second number`() {
        val sum = calculator.sum(2, 4)
        assertEquals(6, sum)
     }

     @Test
     fun `should return the result of subtracting the second number from the first number`() {
        val subtract = calculator.subtract(4, 2)
        assertEquals(2, subtract)
     }
  }
}
```

Kotlin support expressive function names between backticks



JUnit 5 will improve Kotlin support

junit-team / junit5

Watch 149 | Star 876 | Fork 217

Code | Issues 105 | Pull requests 8 | Projects 0 | Wiki | Pulse | Graphs

Allow @BeforeAll and @AfterAll methods to be non-static #419

New issue

Open mfulton26 opened this issue on 22 Jul 2016 · 14 comments

mfulton26 commented on 22 Jul 2016 • edited

Overview

#88 explains why `@BeforeAll` and `@AfterAll` methods currently must be `static`; however, the question arises again and again.

Proposal

Reintroduce support for `@TestInstance` as was present in the JUnit 5 Prototype, thereby allowing developers and third party extension authors to make use of the feature as well. In other words, we do not want to limit use of this feature to JUnit Jupiter itself.

Open Issues

Special caution must be taken with regard to `@Nested` test classes in terms of the lifecycle of the test

Assignees

No one assigned

Labels

enhancement

Jupiter

programming model

team discussion

Projects

None yet

Milestone

5.0 M5

Pivotal

Kotlin type-safe templates

See https://github.com/sdeleuze/kotlin-script-templating

- ➔ Available via Spring MVC & WebFlux JSR-223 support
- ➔ Regular Kotlin code, no new dialect to learn
- ➔ Extensible, refactoring and auto-complete support
- ➔ Need to cache compiled scripts for good performances (work in progress)

```
import io.spring.demo.*
"""
${include("header")}
<h1>${i18n("title")}</h1>
<ul>
   ${users.joinToLine{ "<li>${i18n("user")} ${it.name}</li>" }}
</ul>
${include("footer")}
"""
```

Step 1 Switch from Java to Kotlin

Step 2 Upgrade to Spring 5

Step 3 Migrate to WebFlux

Step 4 WebFlux/Bean Kotlin DSLs

Step 2

Spring Boot 2

Spring MVC

Kotlin 1.1

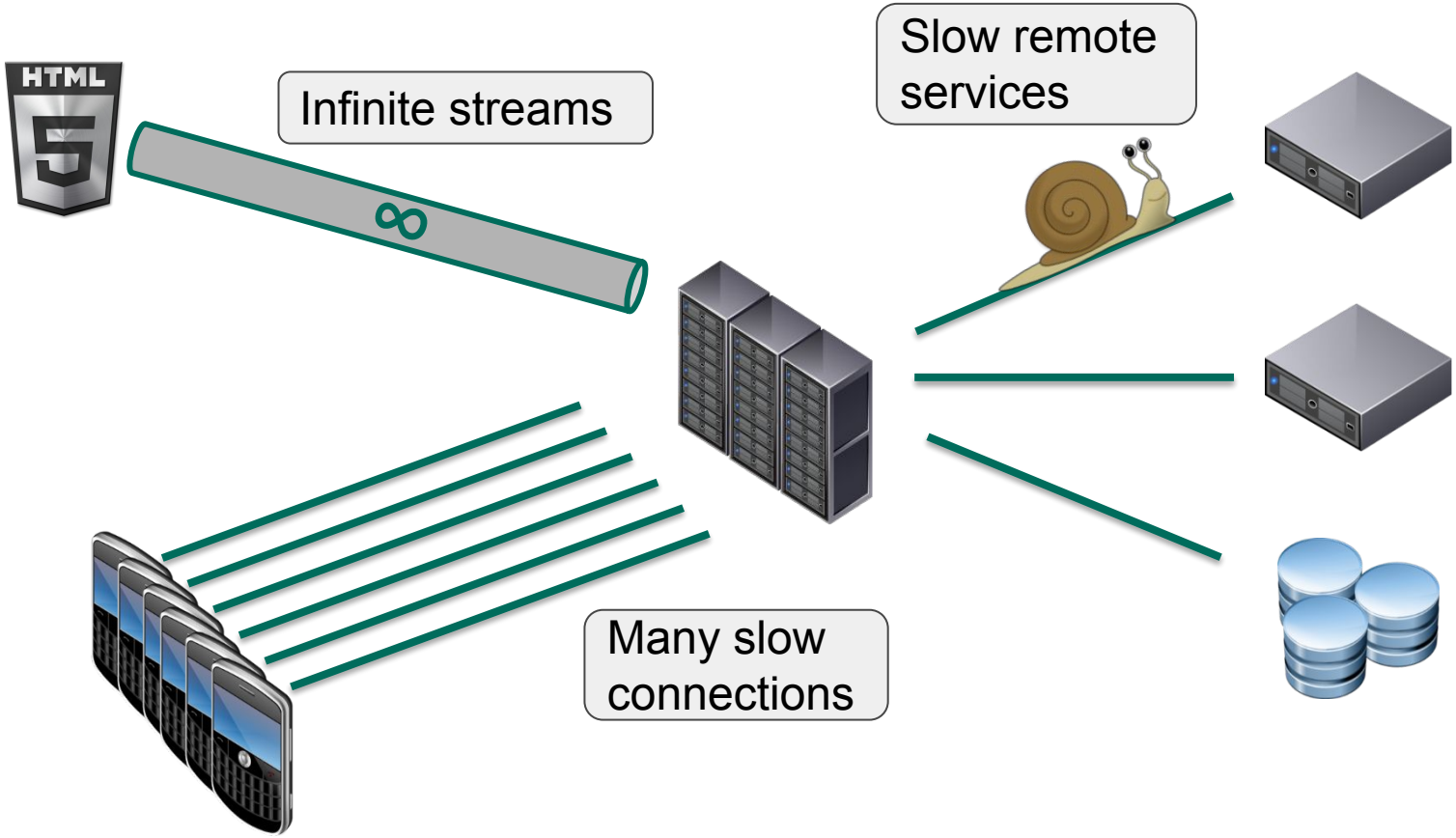
Step 3

Spring Boot 2

Spring WebFlux

Kotlin 1.1

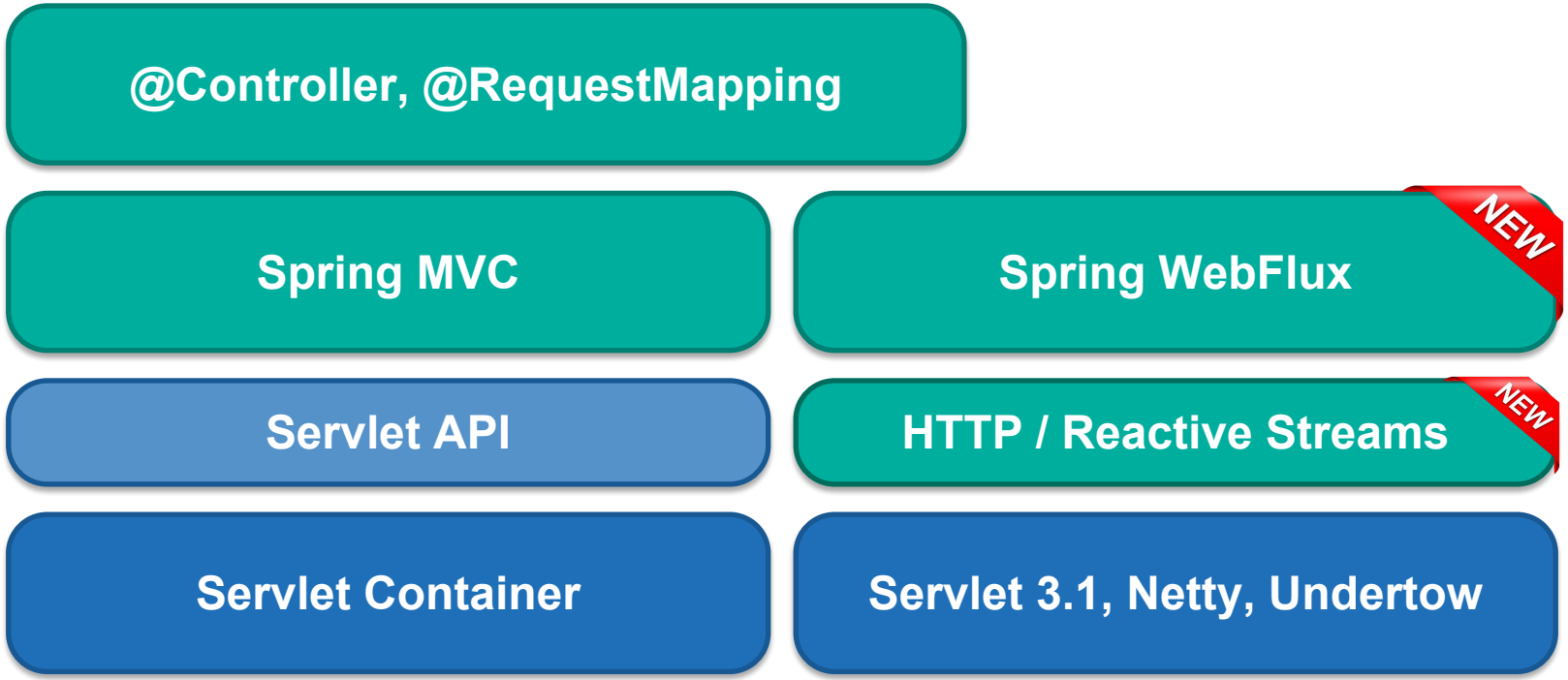
What problems are we trying to solve by going Reactive?


HTML
5
Infinite streams
∞
Slow remote services
Many slow connections

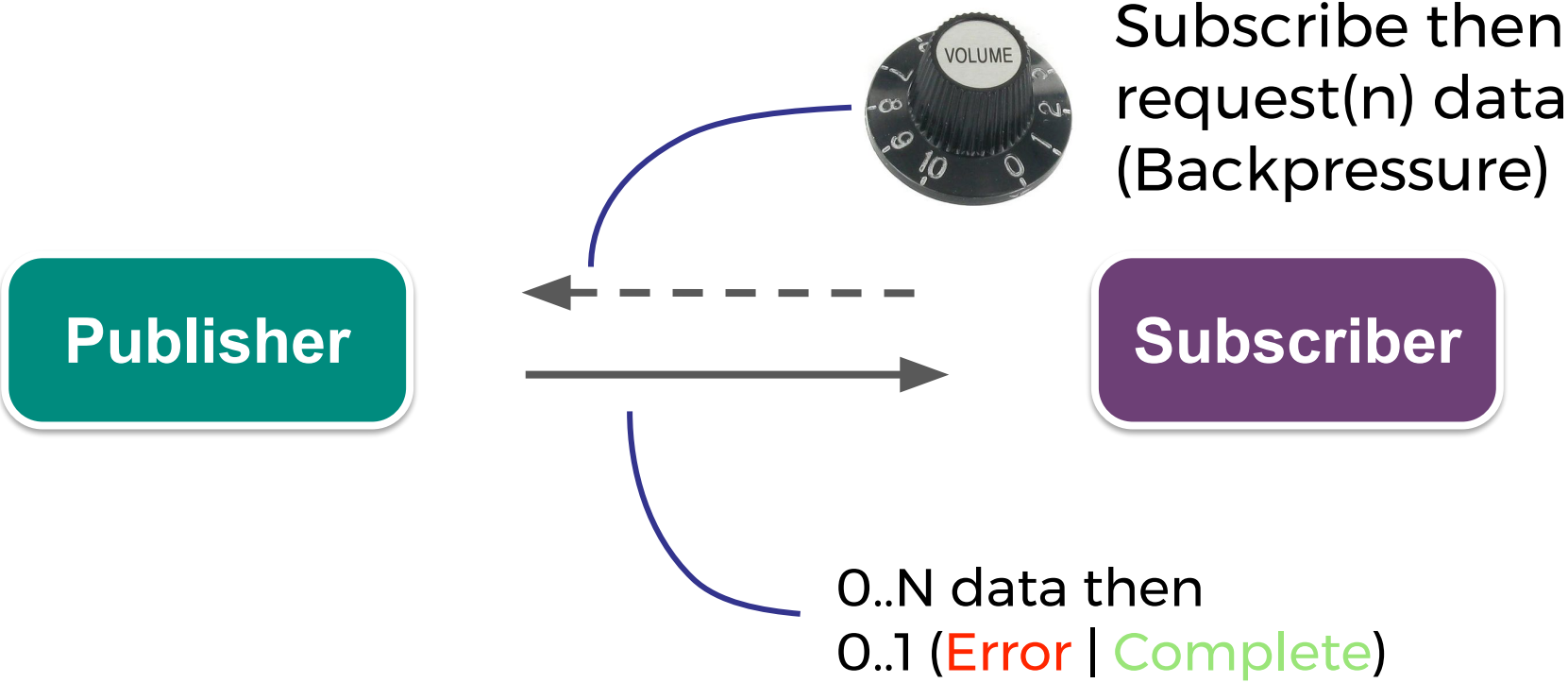
Pivotal

Going Reactive

More for scalability and stability than for speed


@Controller, @RequestMapping
Spring MVC
Spring WebFlux
NEW
Servlet API
HTTP / Reactive Streams
NEW
Servlet Container
Servlet 3.1, Netty, Undertow

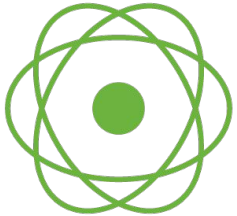
Reactive Streams



Pivotal

Reactive Streams based APIs

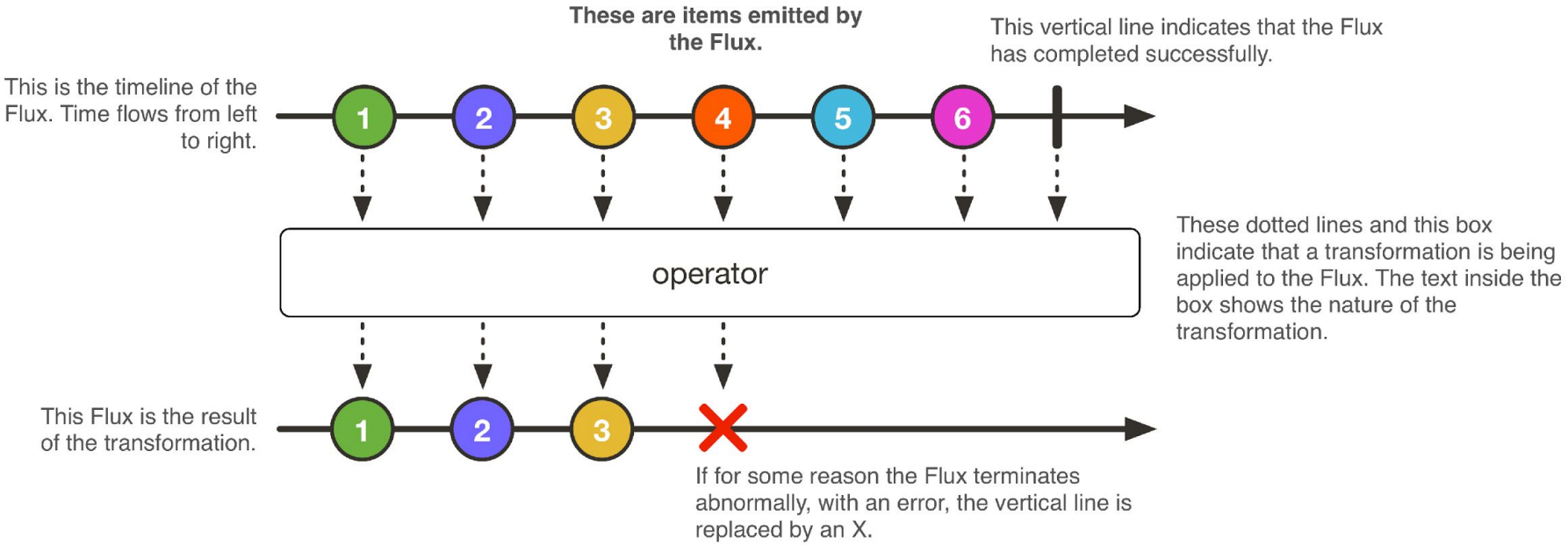
RxJava



Reactor

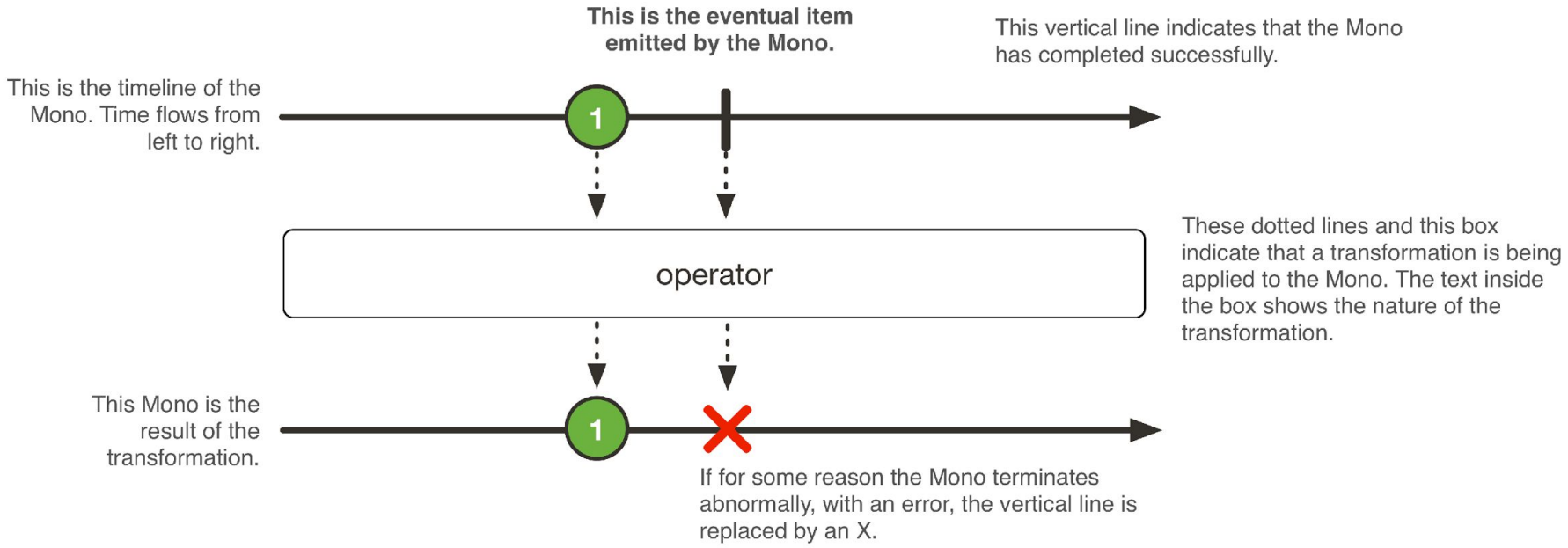
Akka Streams

Flux<T> is a Publisher<T> for 0..n elements

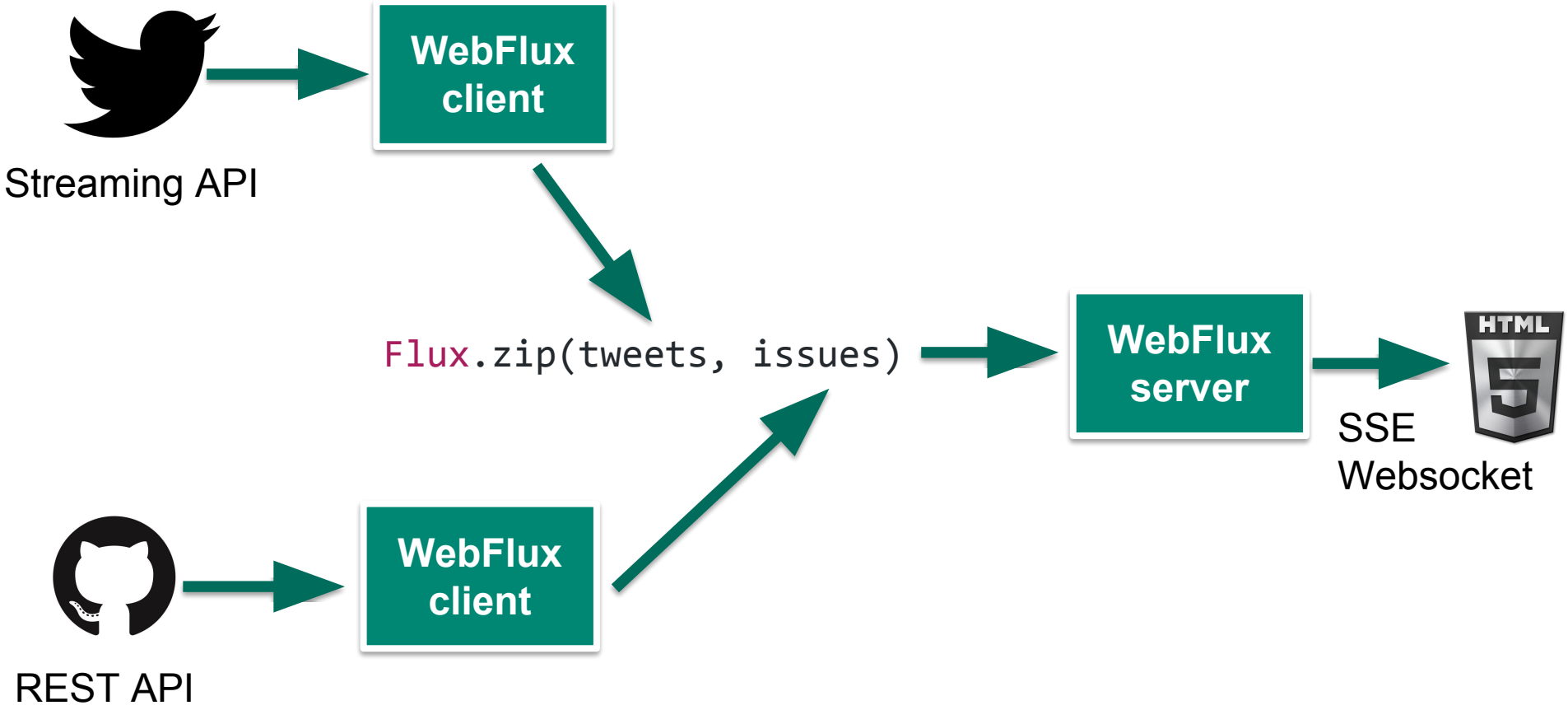


Pivotal

Mono<T> is a Publisher<T> for 0..1 element



Pivotal


Streaming API
WebFlux client
Flux.zip(tweets, issues)
WebFlux server
SSE
Websocket
HTML
REST API
WebFlux client

Reactive APIs are functional

```
fun fetchWeather(city: String): Mono<Weather>
```

```
val location = "Lyon, France"

mainService.fetchWeather(location)
     .timeout(Duration.ofSeconds(2))
     .doOnError { logger.error(it.getMessage()) }
     .onErrorResume { backupService.fetchWeather(location) }
     .map { "Weather in ${it.getLocation()} is ${it.getDescription()}" }
     .subscribe { logger.info(it) }
```

```
val location = "Lyon, France"

mainService.fetchWeather(location)
     .timeout(Duration.ofSeconds(2))
     .doOnError { logger.error(it.getMessage()) }
     .onErrorResume { backupService.fetchWeather(location) }
     .map { "Weather in ${it.getLocation()} is ${it.getDescription()}" }
     .subscribe { logger.info(it) }
```

times out and emits an error after 2 sec

```
val location = "Lyon, France"

mainService.fetchWeather(location)
     .timeout(Duration.ofSeconds(2))
     .doOnError { logger.error(it.getMessage()) }
     .onErrorResume { backupService.fetchWeather(location) }
     .map { "Weather in ${it.getLocation()} is ${it.getDescription()}" }
     .subscribe { logger.info(it) }
```

logs a message in case of errors

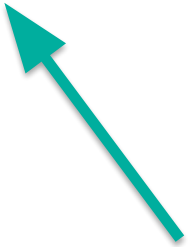
```
val location = "Lyon, France"

mainService.fetchWeather(location)
     .timeout(Duration.ofSeconds(2))
     .doOnError { logger.error(it.getMessage()) }
     .onErrorResume { backupService.fetchWeather(location) }
     .map { "Weather in ${it.getLocation()} is ${it.getDescription()}" }
     .subscribe { logger.info(it) }
```

switches to a different service in case of error

```
val location = "Lyon, France"

mainService.fetchWeather(location)
     .timeout(Duration.ofSeconds(2))
     .doOnError { logger.error(it.getMessage()) }
     .onErrorResume { backupService.fetchWeather(location) }
     .map { "Weather in ${it.getLocation()} is ${it.getDescription()}" }
     .subscribe { logger.info(it) }
```

transforms a weather instance into a String message

```
val location = "Lyon, France"

mainService.fetchWeather(location)
     .timeout(Duration.ofSeconds(2))
     .doOnError { logger.error(it.getMessage()) }
     .onErrorResume { backupService.fetchWeather(location) }
     .map { "Weather in ${it.getLocation()} is ${it.getDescription()}" }
     .subscribe { logger.info(it) }
```



triggers the processing of the chain

Going Reactive

Imply moving from imperative to functional programing

Spring WebFlux annotation-based

```
@RestController
class ReactiveUserController(val repository: ReactiveUserRepository) {

  @GetMapping("/user/{id}")
  fun findOne(@PathVariable id: String) = repository.findOne(id)

  @GetMapping("/user")
  fun findAll() = repository.findAll()

  @PostMapping("/user")
  fun save(@RequestBody user: Mono<User>) = repository.save(user)
}

interface ReactiveUserRepository {
  fun findOne(id: String): Mono<User>
  fun findAll(): Flux<User>
  fun save(user: Mono<User>): Mono<Void>
}
```

Step 1 Switch from Java to Kotlin

Step 2 Upgrade to Spring 5

Step 3 Migrate to WebFlux

Step 4 WebFlux/Bean Kotlin DSLs

Step 3

Spring Boot 2

Spring WebFlux

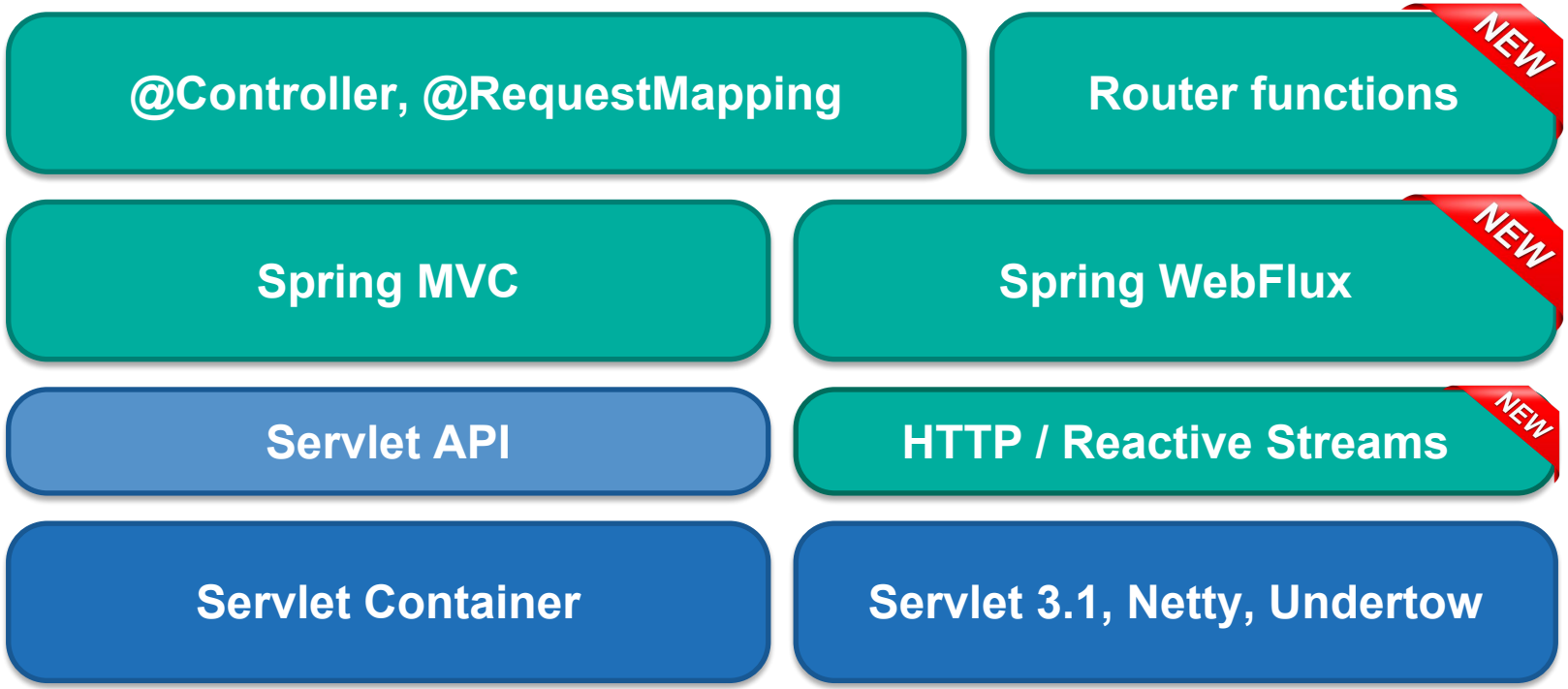
Kotlin 1.1

Step 4

Spring Boot 2

Spring WebFlux Functional

Kotlin 1.1


@Controller, @RequestMapping
Router functions
NEW
Spring MVC
Spring WebFlux
NEW
Servlet API
HTTP / Reactive Streams
NEW
Servlet Container
Servlet 3.1, Netty, Undertow

```java
// Annotation-based Java
@RequestMapping("/quotes/feed", produces = TEXT_EVENT_STREAM_VALUE)
public Flux<Quote> fetchQuotesStream() { ... }
```

```java
// Annotation-based Java
@RequestMapping("/quotes/feed", produces = TEXT_EVENT_STREAM_VALUE)
public Flux<Quote> fetchQuotesStream() { ... }
```

```java
// Functional Java without static imports
RouterFunctions.route(
     RequestPredicates.path("/quotes/feed")
           .and(RequestPredicates.accept(MediaType.TEXT_EVENT_STREAM)),
     { ... }
)
```

```java
// Annotation-based Java
@RequestMapping("/quotes/feed", produces = TEXT_EVENT_STREAM_VALUE)
public Flux<Quote> fetchQuotesStream() { ... }
```

```java
// Functional Java with static imports
route(
    path("/quotes/feed").and(accept(TEXT_EVENT_STREAM)),
    { ... }
)
```

```java
// Annotation-based Java
@RequestMapping("/quotes/feed", produces = TEXT_EVENT_STREAM_VALUE)
public Flux<Quote> fetchQuotesStream() { ... }
```


```kotlin
// Functional Kotlin
router {
  "/quotes/feed" and accept(TEXT_EVENT_STREAM) { ... }
}
```

```kotlin
@Configuration
class ApplicationRoutes(val userHandler: UserHandler,
                        val blogHandler: BlogHandler,
                        val shopRepository: ShopRepository) {

  @Bean
  fun appRouter() = router {
    GET("/users", userHandler::fetchAll)
    GET("/users/{id}", userHandler::fetch)
  }

  @Bean
  fun nestedRouter() = router {
      ...
  }

  @Bean
  fun dynamicRouter() = router {
      ...
  }
}
```

```kotlin
@Bean
fun nestedRouter() = router {
    ("/blog" and accept(TEXT_HTML)).nest {
        GET("/", blogHandler::findAllView)
        GET("/{slug}", blogHandler::findOneView)
    }
    ("/api/blog" and accept(APPLICATION_JSON)).nest {
        GET("/", blogHandler::findAll)
        GET("/{id}", blogHandler::findOne)
        POST("/", blogHandler::create)
    }
}
```

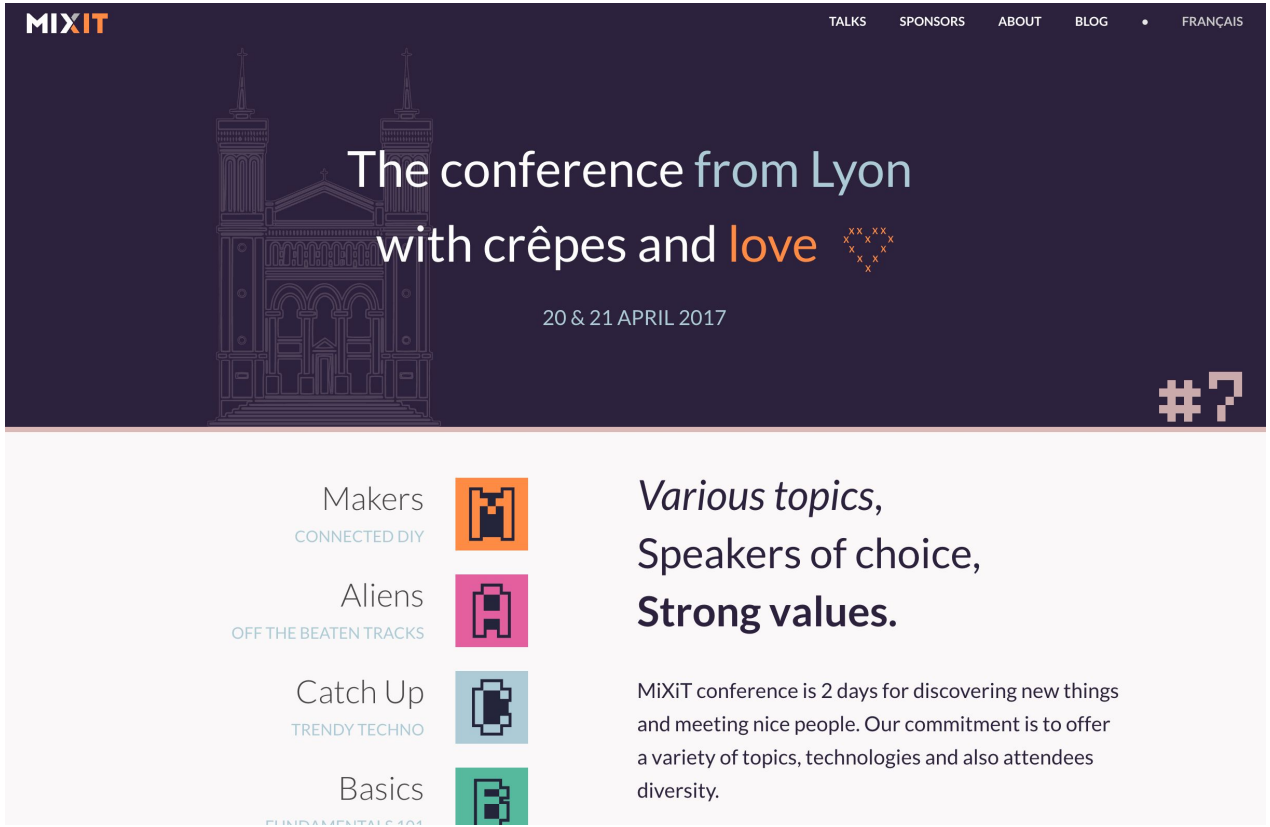
```kotlin
@Bean
fun dynamicRouter() = router {
  shopRepository.findAll()
        .toIterable()
        .forEach { shop ->
           GET("/${shop.id}") {
              req ->
              shopHandler.homepage(shop, req)
           }
        }
}
```

```
@Component
class EventHandler(val repository: EventRepository) {

    fun findOne(req: ServerRequest) =
        ok().json().body(repository.findOne(req.pathVariable("id")))

    fun findAll(req: ServerRequest) =
        ok().json().body(repository.findAll())

}
```

```
@Component
class NewsHandler {

    fun newsView(req: ServerRequest) = ok().render("news")

    fun newsSse(req: ServerRequest) = ok()
            .contentType(TEXT_EVENT_STREAM)
            .body(Flux.interval(ofMillis(100)).map { "Hello $it!" })

}
```

Spring WebFlux + Kotlin reference application

https://github.com/mixitconf/mixit



Pivotal

MiXiT project software design

- Reactive and non-blocking
- Idiomatic Kotlin code
- Functional routing DSL
- Immutable domain model
- Balance between functional and OOP + annotation style
- Constructor based and non-intrusive dependency injection
- Efficient development mode

Spring WebFlux

Your next microframework?

Pivotal

Step 3

Spring Boot 2

Spring WebFlux

Kotlin 1.1

Step 4
Microframework style

Spring WebFlux Functional

Kotlin 1.1

Choose the flavour you prefer!

Boot + WebFlux

- ➔ Automatic server configuration
- ➔ All Spring Boot goodness!
- ➔ Bean registration
  - ◆ Annotation-based
  - ◆ Optimized Classpath scanning
- ➔ MiXiT webapp startup:
  - ◆ 3 seconds
  - ◆ 20 Mbytes heap size after GC
  - ◆ Works with -Xmx32m

WebFlux standalone

- ➔ Manual server configuration
- ➔ Bean registration
  - ◆ Functional and lambda-based
  - ◆ No Cglib proxy
  - ◆ No need for kotlin-spring plugin
- ➔ MiXiT webapp startup:
  - ◆ 1.2 seconds
  - ◆ 10 Mbytes heap size after GC
  - ◆ Works with -Xmx32m

Functional bean registration DSL

- After XML and JavaConfig, a third major way to register your beans
- In a nutshell: lambda with Supplier act as a FactoryBean
- Very efficient, no reflection, no CGLIB proxies involved

```
beans {
    bean<Foo>()
    profile("profile") {
        bean { Bar(it.ref<Foo>()) }
        bean { Baz(it.env["baz.name"]) }
    }
}
```

See functional-bean-registration MiXiT branch

https://goo.gl/iGwzw3

```
beans {
  bean("messageSource") {
     ReloadableResourceBundleMessageSource().apply {
        setBasename("messages")
        setDefaultEncoding("UTF-8")
     }
  }
  bean {
     MustacheViewResolver().apply {
        setPrefix("classpath:/templates/")
        setSuffix(".mustache")
        setCompiler(Mustache.compiler().escapeHTML(false))
        setModelCustomizer({ model, exchange ->
           customizeModel(model, exchange, it.ref<MessageSource>()) })
     }
  }
  ...
```

What about frontend development?

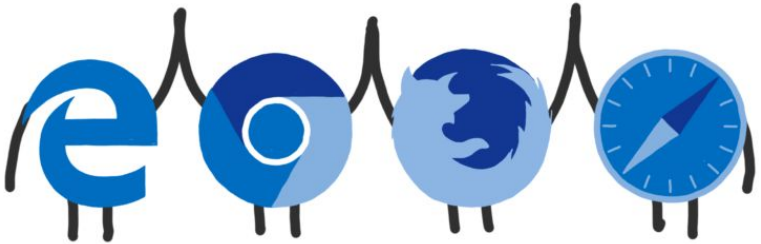
Kotlin 1.1 supports compiling to JavaScript

Could replace JavaScript / TypeScript for your frontend code

- Good JavaScript interop
- Could allow to:
  - use a single language for your webapp
  - share code between backend and frontend
- Before Kotlin 1.1.3, generates big JavaScript files
- Kotlin 1.1.4 will introduce a great Dead Code Elimination plugin!
  - Hello world = 65 Kb

WebAssembly

A unique opportunity to build an open and cross-platform native application ecosystem!



Read “An Abridged Cartoon Introduction To WebAssembly” by Lin Clark for more details
https://goo.gl/I0kQsC

WebAssembly compilation target instead of JavaScript?

Compiling non-JS based languages to JS is and will remains a hack

- ➔ WebAssembly is currently mostly C/C++ oriented
- ➔ But incoming features may change that:
  - ◆ DOM and Web API available directly from WASM
  - ◆ Built-in garbage collector
  - ◆ Exception Handling
- ➔ Kotlin could support WebAssembly via Kotlin Native (LLVM)
- ➔ You could leverage that to provide smaller and faster frontend code
- ➔ Fallback via asm.js

What can you expect?

- ➔ Spring + Kotlin guides
- ➔ Spring Framework 5 GA in September
- ➔ Spring Boot 2 GA in november
- ➔ Experiments on 2 important topics
  - ◆ Kotlin frontend (JS and maybe WebAssembly ...)
  - ◆ Coroutine based API for WebFlux (SPR-15413)

https://kotlin.link

Kotlin Conf
SUPER EARLY BIRD
REGISTER NOW

Fork me on GitHub

Type to Filter

LINKS

Official Links

JetBrains/kotlin 9069
Last update: May 18, 2017

Kotlin/KEEP 351
Last update: May 4, 2017
Kotlin Evolution and Enhancement Process

Home Page

Language Reference

Slack (6700+ users)

Public chat archive of Kotlin's Slack

Try Kotlin!

Resources

dbacinski/Design-Patterns-In-Kotlin 652
Last update: Feb 24, 2017
Design Patterns implemented in Kotlin.

Antonio Leiva - Android and any other monsters

Stackoverflow Documentation on Kotlin

Quora Kotlin

Trending Kotlin on Github

/r/Kotlin

Books and Courses

Kotlin in Action - Dmitry Jemerov, Svetlana Isakova

Kotlin for Android Developers - Antonio Leiva

Programming Kotlin - Stephen Samuel, Stefan Bocutiu

Kotlin for Java Developers
160-minute Android Course.

Kotlin Programming: Next Level Java Development
Learn coding in Kotlin from scratch!

Pivotal

Thanks!

- Slides available on https://goo.gl/qMA9Ho
- Follow me on @sdeleuze for fresh Spring + Kotlin news

Pivotal