

The background is a solid teal color. In the center, there is a circle composed of many small, light-yellow dots. Four paper airplanes, drawn in a simple line-art style with teal outlines and hatching for shading, are positioned around the central dotted circle, appearing to fly outwards. The text 'Spring 5 - Spring-Boot 2' is written in a bold, white, sans-serif font, centered over the dotted circle and slightly tilted upwards to the right.

Spring 5 - Spring-Boot 2

Cabinet de conseil IT et Agilité



Qui sommes nous ?



David Wursteisen

Consultant au sein de Soat



Patrick Allain

Consultant au sein de Soat

Petite page de pub

Le blog Soat :

<http://blog.soat.fr/>

Si vous êtes sur Paris, retrouvez nos TechLabs sur Meetup :

<https://www.meetup.com/fr-FR/TechLabs-by-SOAT/>

-

Rappels & Contexte

Spring 4 - Spring Boot 1.X.X

- Spring 4 :

- **Compatibilité Java 6, 7, 8**
- **Première release en le 12 décembre 2013**
- **API Servlet 2.5 +**

- Spring Boot 2 :

- **Java 7 & 8 (avec un support de Java 6)**
- **Container de Servlet :**
 - Tomcat 7, 8
 - Jetty 8, 9
 - Undertow 1.3
 - ... N'importe quel container de servlet supportant l'API Servlet 3.0 +

Spring 5 : Guidelines

- Java 8
- Compatibilité Java 9
- Dernières versions :
 - Servlet 3.1 & Servletv 4.0
 - JPA 2.1,
 - JMS 2.0
 - Support de JUnit 5
- Intégration de Kotlin
- Programmation Réactive :
 - Reactor
 - WebFlux
 - RouterFunction
- HTTP/2

—

Reactor

Les Reactives Streams

Une API qui contient 4 classes :

- `org.reactivestreams.Publisher<T>`
- `org.reactivestreams.Processor<T, R>`
- `org.reactivestreams.Subscriber<T>`
- `org.reactivestreams.Subscription`

Dans Java 9 : `java.util.concurrent.Flow`

Les Reactives Streams

Comment fait-on de l'asynchrone en Java ?

Quels sont les avantages du code asynchrone ?

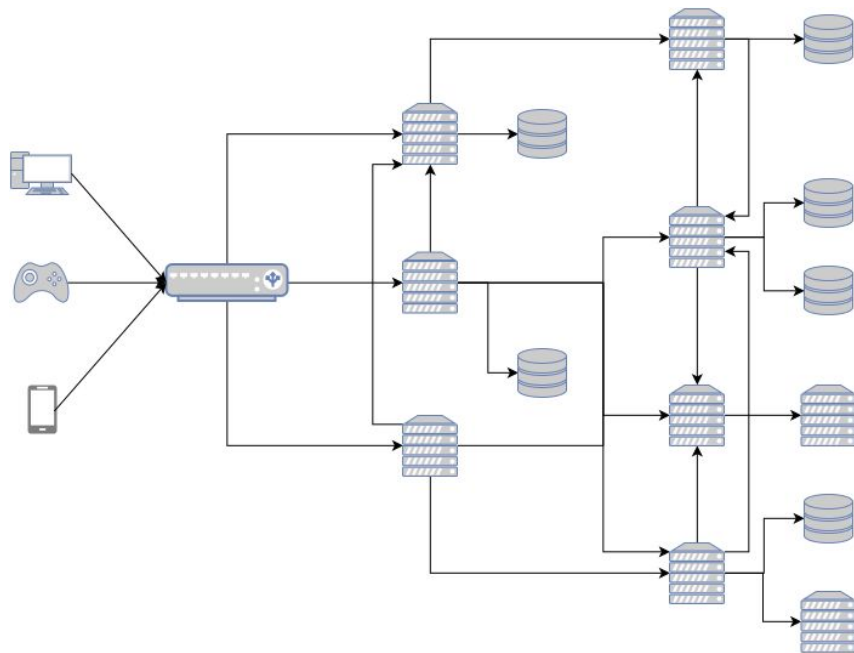
Les Reactive Streams

Une application, ça peut ressembler à :



Les Reactive Streams

De plus en plus, ça à tendance à ressembler à :



Les Reactive Streams

C'est quoi la nouveauté ?

- De nouvelles interactions
- De nouveaux devices

De nouvelles problématiques !

Les Reactive Streams

Aujourd'hui, quand on code en Java, on fait souvent :

```
public EntityResponse<Something> something() {  
    Something1 s1 = api1.getSomething();  
    Something2 s2 = api2.getSomething(s1.getValue());  
    Something3 s3 = db.getSomething(s1.getValue());  
    return EntityResponse.ok(new Something(  
        s1, s2, s3  
    ));  
}
```

Les Reactive Streams

Quand on veut être asynchrone, on utilise alors :

```
public EntityResponse<Something> something() {  
    Future<Something> async = api1.somethingAsync((s1) -> {  
        Something2 s2 = api2.getSomething(s1.getValue());  
        Something3 s3 = api2.getSomething(s1.getValue());  
        return new Something(s1, s2, s3);  
    })  
    return EntityResponse.ok(async.get());  
}
```

Mais ... Est-on vraiment asynchrone ?

Que risque de devenir ce code ?

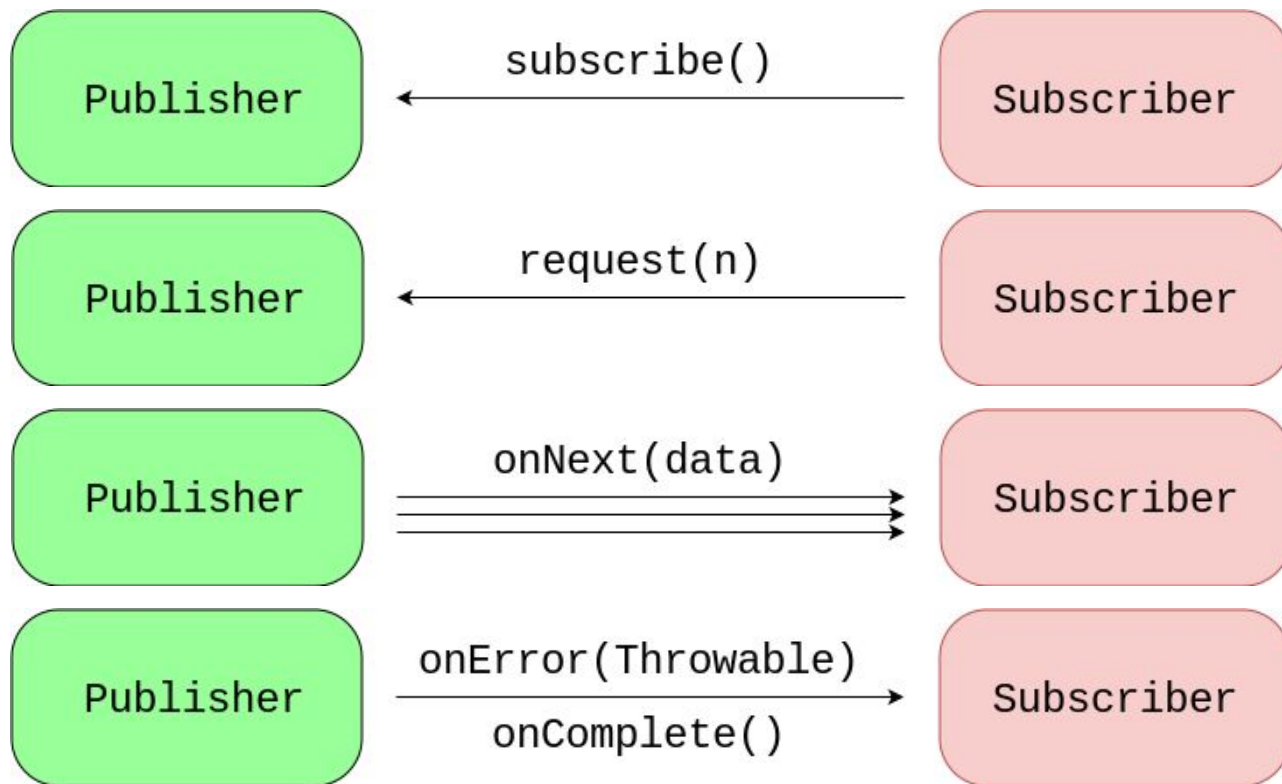
Les Reactive Streams

Le callback hell :

```
public EntityResponse<Something> something() {
    Future<Something> async = api1.something1Async((s1) -> {
        return db.something2Async(s1.getValue(), (s2) -> {
            return api2.something3Async(s1.getValue(), s2.getValue(), (s3)->
{
            return new Something(s1, s2, s3);
        });
    })
})
return EntityResponse.ok(async.get());
}
```

Et niveau HTTP ?

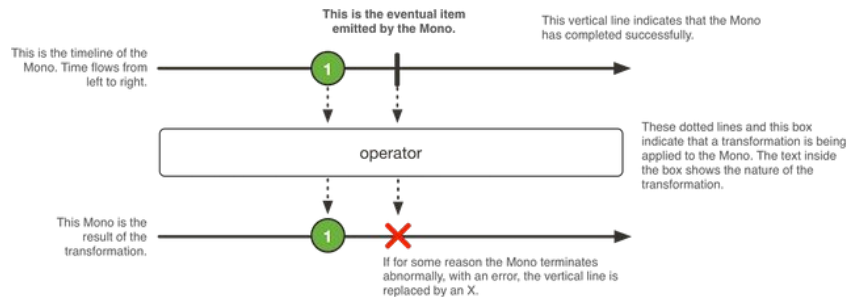
Les Reactive Streams



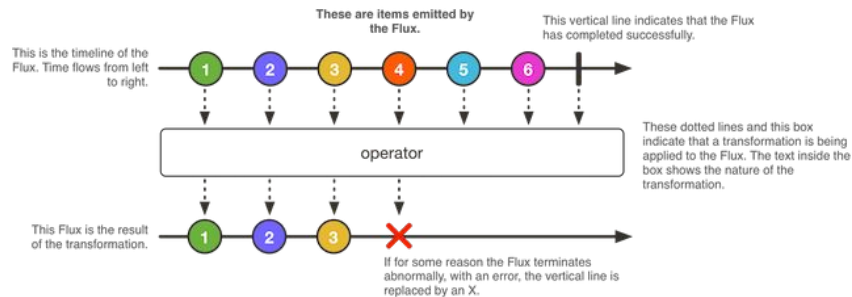
Les Reactive Streams

Reactor est une implémentation de l'API des Reactive Streams.

Mono<T>



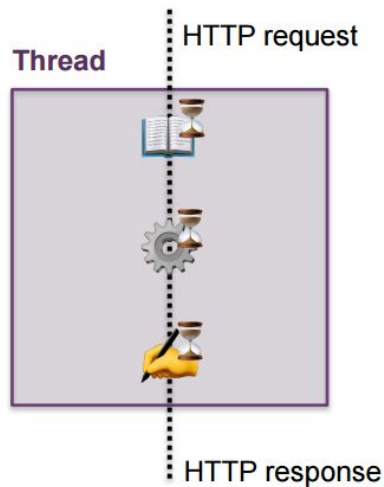
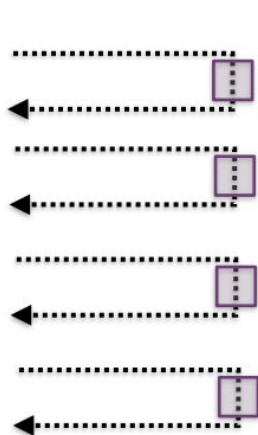
Mono<T>



WebFlux

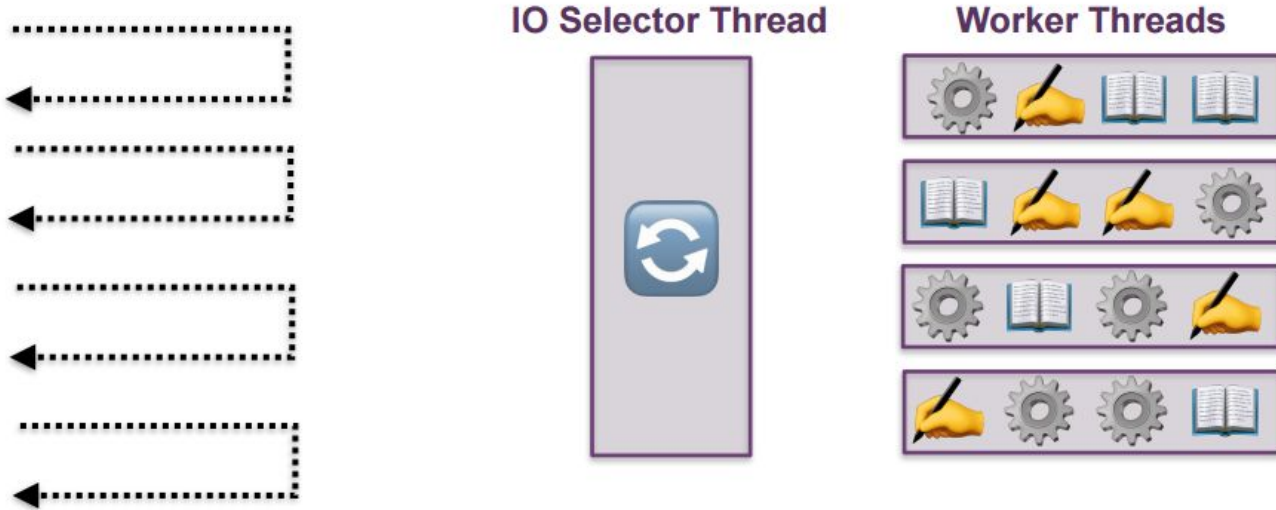
WebFlux

- Le Thread Pool HTTP de Tomcat ...
 - Par défaut, on commence à 10 (minSpareThreads)
 - Ensuite, on augmente jusqu'à 200 (maxThreads)
 - Puis on met en queue 100 connexion (acceptCount)



WebFlux

Avec les Reactive Streams, on change de paradigme



RouterFunction

RouterFunction

- Alternative à @Controller + @(Request/Get/Post/)Mapping
- Définition des routes de notre application
- Centralisée dans un seul @Bean !!

RouterFunction

Exemple :

```
@Bean
public RouterFunction routes() {
    return RouterFunctions.route(
        RequestPredicates.GET("/"),
        request -> ServerResponse.ok().syncBody("Hello World")
    )
}
```

Autres améliorations !!...

Kotlin

Intégration du support de Kotlin via l'utilisation de méthode d'extensions.

```
@Service
class MyService {

    @Autowired
    lateinit var ctx: ApplicationContext

    // ...

    public void doSomething() {
        val bean: MyCustomBean = ctx.getBean()
    }
}
```

+ d'info: <http://blog.soat.fr/2017/12/kotlin-un-langage-taille-pour-le-backend/>

Spring Data

Et Spring Data aussi est devenu réactif :

- Redis
- MongoDB
- Cassandra
- Couchbase

```
public class UserRepository
    extends ReactiveCrudRepository<User> {

    Mono<User> findByEmail(String email);

}
```

Security

De nombreuses simplifications :

- Des builders pour simplifier l'écriture de configuration
- De nouvelles propriétés :
 - `security.sessions`
 - `security.basic`
 - `security.enable-csrf`
 - `security.headers`
 - `security.ignored`
 - `security.require-ssl`
- Actuator configuration

Actuator

- Metrics n'existe plus :(...
- Mais ... Il devient plus simple d'écrire ses propres métriques :
 - `@Autowired MeterRegistry`
 - `Metrics.counter("feature");`
- Et de les exporter !

Thymeleaf

- On passe sur la V3 de Thymeleaf
- Intégration des Reactive Streams dans le moteur :
 - **ReactiveDataDriverContextVariable**
 - **SpringWebReactiveLinkBuilder**
 - **ThymeleafReactiveView**

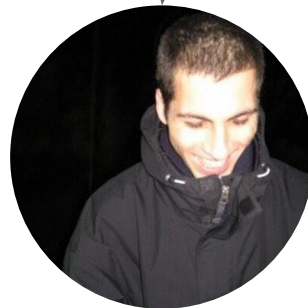
Let's code !

- Copier **toute** la clé USB
 - mvnw install -f workaround.xml ← **A FAIRE UNE FOIS EN PREMIER**
 - mvnw install
 - ouvrir dans votre IDE
-
- Ouvrir le projet dans votre IDE puis suivre les tests dans les packages "step1", "step2", ...
 - **Besoin d'aide ? Appelez-nous !**

David



Patrick



Contact

<http://www.soat.fr>

<http://blog.soat.fr>

David Wursteisen : david.wursteisen@soat.fr

Patrick Allain : patrick.allain@soat.fr