Embracing Clojure

A journey into Clojure adoption

Luca Grulla

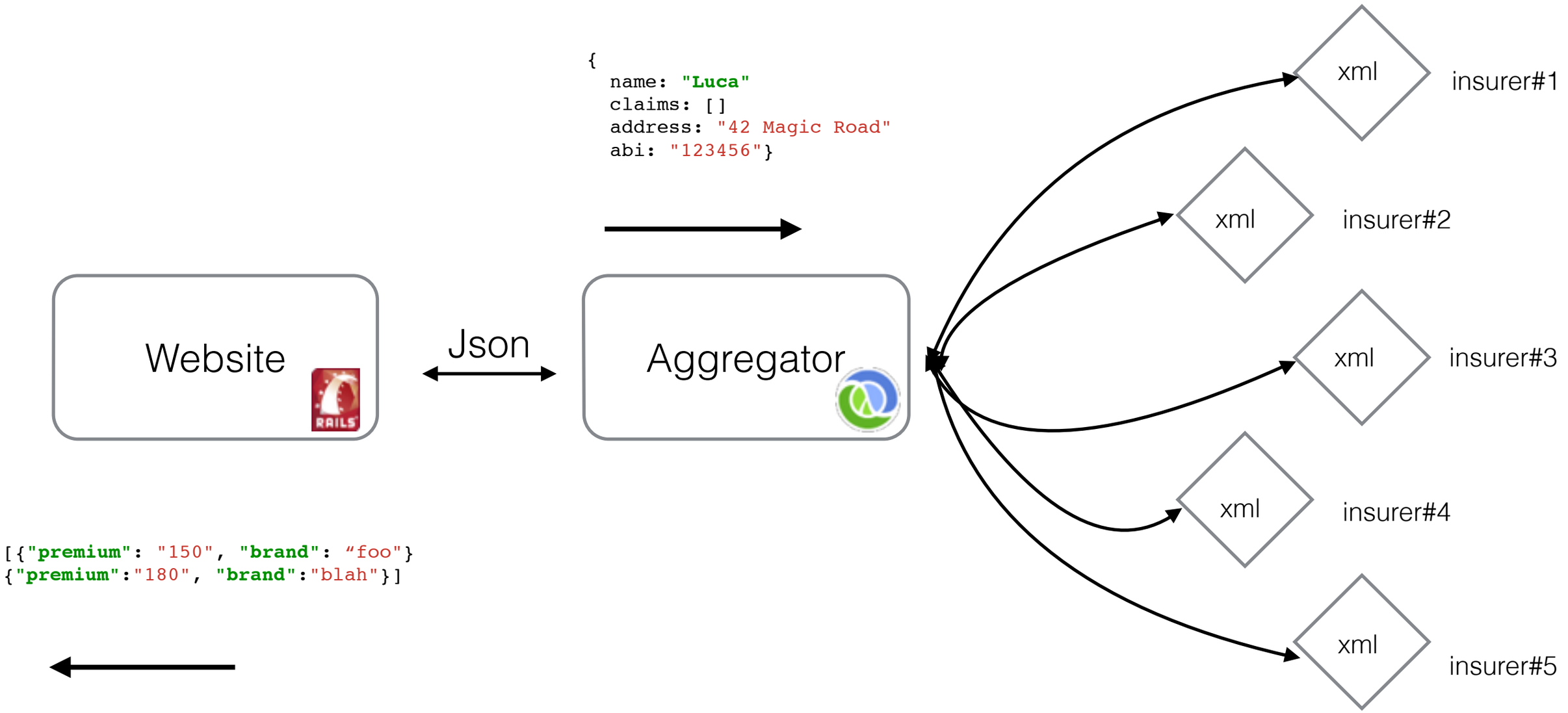
LambdaCon,

2015-3-28 Bologna, Italy

Back in 2012 we had…

- A new product idea
- Lean team: 2 developers
  - both with 10+ years of experience
  - both polyglot
  - no Functional Programming experience

Product Architecture



Why Clojure?

- The problem we had to solve was data transformation: FP was a pretty good match
- Mature stack(JVM, repos, Java libraries)
- Language still under heavy development
- Strong Clojure support within the organization.

~$ lein new aggregator

Challenge #1: Emacs

Steep learning curve

Challenge #2: read LISP

```clojure
(def my-vector [{:val 8} {:val 33} {:val 42} {:val 13}])

(* (reduce + (map :val (filter(fn[y] (< (:val y) 20)) my-vector)))
2) ;;42
```

Epiphany #1: threading macro for local pipelines

(->> x & forms)

```
(* (reduce + (map :val (filter(fn[y] (< (:val y) 20)) my-vector)))
2) ;;42
```

```
(->>
    my-vector
    (filter (fn[y] (< (:val y) 20)))
    (map :val)
    (reduce +)
    (* 2)) ;;42
```

Challenge #3: idiomatic Clojure

```
(def people [{:name "Luca" :nationality “Italian"}
             {:name "Tim" :nationality “Canadian"}
             {:name "Davie" :nationality “British"}
             {:name "David" :nationality “Belgium"}
             {:name "Mike" :nationality "British"}])

(filter not-italian? people)
(defn not-italian?[p] (not= "Italian" (:nationality p)))

(filter (fn[p] (not= "Italian" (:nationality p))) people)
(filter (fn[p] (not= "British" (:nationality p))) people)
```

Epiphany #2: function composition

(partial f arg1)

```
(defn not?[nationality p] (not= nationality (:nationality p)))

(filter (partial not? "Italian") people)
```

Epiphany #3: functions are first class citizens

Everything is geared toward functions creation and composition

Epiphany #4: parallelism made easy

```
(def request-func [partner-1-func partner-2-func partner-3-func])

(pmap #(apply % []) request-func) ;;hardware bound(num of processors +2)

(map #(future (apply % [])) request-func) ;;collection of futures
```

clojure.core.async

tl;dr

- If you are doing some sort of data transformation you should look at Clojure

- Clojure code is terse and expressive

- Mature ecosystem: JVM, repos, libraries

- Personal growth: people are challenged by the new paradigm, solid CS concepts took back into the game

- Business impact: functional composition + parallelism will give you a competitive advantage

Questions?

@lucagrulla

www.lucagrulla.it

we're hiring!! http://www.uswitch.com/careers/