Go Reactive

Blue Print for Future Applications

Dr. Roland Kuhn
Akka Tech Lead
@rolandkuhn
The Four Reactive Traits

http://reactivemanifesto.org/
Starting Point:
The User
The User → browser
Responsiveness

always available
interactive
(near) real-time
Bounded Latency
Bounded Latency

• fan-out in parallel and aggregate
Bounded Latency

- fan-out in parallel and aggregate
Bounded Latency

- fan-out in parallel and aggregate
Bounded Latency

- fan-out in parallel and aggregate
Bounded Latency

- fan-out in parallel and aggregate
- use circuit breakers for graceful degradation
Bounded Latency

- fan-out in parallel and aggregate
- use circuit breakers for graceful degradation

![Diagram showing the process of bounded latency with nodes labeled 'do work' and 'fail fast'.]
Bounded Latency

- fan-out in parallel and aggregate
- use circuit breakers for graceful degradation
- use bounded queues, measure flow rates
Bounded Latency

- fan-out in parallel and aggregate
- use circuit breakers for graceful degradation
- use bounded queues, measure flow rates

Use Bounded Queues:

\[
\text{Latency} = \text{QueueLength} \cdot \text{ProcessingTime}
\]

(for reasonably stable average processing time)
Bounded Latency

- fan-out in parallel and aggregate
- use circuit breakers for graceful degradation
- use bounded queues, measure flow rates

**Use Bounded Queues:**

\[ \text{Latency} = \text{QueueLength} \times \text{ProcessingTime} \]

(for reasonably stable average processing time)
Resilience

Responsive in the Face of Failure
Handle Failure

• software will fail
• hardware will fail
• humans will fail
• system still needs to respond $\Rightarrow$ resilience
Distribute!
Asynchronous Failure

- parallel fan-out & distribution
  ➟ asynchronous execution
- compartmentalization & isolation

Asynchronous Failure
Asynchronous Failure

- parallel fan-out & distribution ➞ asynchronous execution
- compartmentalization & isolation
- no response? ➞ timeout events
- someone else’s exception? ➞ supervision
Asynchronous Failure

- parallel fan-out & distribution
  ➟ asynchronous execution
- compartmentalization & isolation
- no response? ➟ timeout events
- someone else's exception? ➟ supervision

Request ➞ Failure ➞ Response
Asynchronous Failure

- parallel fan-out & distribution
  ➟ asynchronous execution
- compartmentalization & isolation
- no response?
  ➟ timeout events
- someone else's exception?
  ➟ supervision

Asynchronous Failure

**Request** ➟ **Response**

.failure**

(event-driven)
Asynchronous Failure

- parallel fan-out & distribution ➞ asynchronous execution
- compartmentalization & isolation
- no response? ➞ timeout events
- someone else’s exception? ➞ supervision
- location transparency ➞ seamless resilience
Scalability

Responsive in the Face of Changing Load
Handle Load
Handle Load
Handle Load

- partition incoming work for distribution
- share nothing
- scale capacity up and down on demand
- supervise and adapt
- **location transparency**
  ➞ seamless scalability
Handle Load

- partition incoming work for distribution
- share nothing
- scale capacity up and down on demand
- supervise and adapt
- **location transparency**
  ⇾ seamless scalability
... this has some interesting consequences!
Consequences

• distribution & scalability
  ➞ loss of strong consistency
• CAP theorem? — not as relevant as you think
• eventual consistency
  ➞ gossip, heartbeats, dissemination of change

Pat Helland: Life beyond Distributed Transactions
Peter Bailis: Probabilistically Bounded Staleness (http://pbs.cs.berkeley.edu)
Consequences

- distribution & scalability
  ➞ loss of strong consistency
- CAP theorem? — not as relevant as you think
- eventual consistency
  ➞ gossip, heartbeats, dissemination of change

Pat Helland: Life beyond Distributed Transactions
Peter Bailis: Probabilistically Bounded Staleness (http://pbs.cs.berkeley.edu)
Corollary

- Reactive needs to be applied all the way down
- Polyglot deployments demand collaboration
  ➟ for example http://reactive-streams.org/
But what about us, the developers?
Step 1: Take a Leap of Faith

- thread-based models have made us defensive
  - “don’t let go of your thread!”
  - “asynchrony is suspicious”
  - “better return strict value, even if that needs blocking”
Step 1: Take a Leap of Faith

• thread-based models have made us defensive
  • “don’t let go of your thread!”
  • “asynchrony is suspicious”
  • “better return strict value, even if that needs blocking”

• it is okay to write a method that returns a Future!
Step 2: Rethink the Architecture

• break out of the synchronous blocking prison
• focus on communication & protocols
• asynchronous program flow
  ➟ no step-through debugging
  ➟ tracing and monitoring
• loose coupling
Step 3: Profit!

• clean business logic, separate from failure handling
• distributable units of work
• effortless parallelization
• less assumptions ➞ lower maintenance cost
Step 3: Profit!

• clean business logic, separate from failure handling
• distributable units of work
• effortless parallelization
• less assumptions ➞ lower maintenance cost
• independent agents ➞ fun to work with!
The Four Reactive Traits

responsive
scalable
event-driven
resilient

http://reactivemanifesto.org/