The Promise of Micro Services with Fast Data...
About Me

Creds
About Me

Creds

→ VP Engineering
→ Senior Software Engineer
→ Principal Systems Engineer
→ Building Distributed Systems 10+ years
→ Book: Reactive Application Design (Manning)
→ Twitter: @ironfish
→ Github: ironfish
About Us

Reactive Approach
About Us

Reactive Manifesto

Reactive: “Readily responsive to stimulus.”

- Reactive Manifesto
- Published on September 16 2014. (v2.0)
- +Jonas Bonér, Dave Farley, Roland Kuhn, and Martin Thompson
- 20K + Signatures

1 Merriam Webster Dictionary
About Us
About Us

Lightbend Reactive Platform
About Us
About Us

Lightbend Fast Data Platform
About Us
Outline

The Talk
Outline

The Talk

→ Micro Services
→ Promise Theory
→ Event Sourcing
→ Crud
→ Fast Data
→ Feedback Control
Micro Services

What Are They?
Micro Services

Good Definition

Supports *multiple autonomous teams* organized to scale the development where the teams can *develop, deploy and manage* their services independently.
Micro Services

Better Definition

A system of autonomous collaborative distributed services.
Micro Services

Key Concepts

- **Apply** Promise Theory
- **Keep your services** small, hence the name **micro**
- **Cluster your micro service for resilience**
- **Consistency and load times are determined by your SLA**
- **Use polyglot persistence**
Micro Services

Key Concepts

- Inside data: the present
- Outside Data: the past
- Between Services: the future
- Own your data

“Regardless of storage medium, our view of state is always of the past!”
Micro Services

A MONOLITH DISGUISED AS A MICROSERVICE...

A MICROSERVICE OWNS ITS DATA!

...IS STILL A MONOLITH
Promise Theory

What Is It?
Promise Theory

Good Definition

A promise is a declaration of intent whose purpose is to increase the recipient’s certainty about a claim of past, present or future behavior.

— Mark Burgess²

²An Approach to Understanding Policy Based on Autonomy and Voluntary Cooperation
Promise Theory

Key Concepts

→ Think in promises not obligations
→ “Autonomy makes information local, leading to greater certainty and stability”³
→ An Autonomous Service can only promise its own behavior
→ Obligations diverge into unpredictable outcomes from definite beginnings ⇒ decreased certainty
→ Promises converge towards a definite outcome from unpredictable beginnings ⇒ increased certainty

³ In Search for Certainty by Mark Burgess
Event Sourcing

What Is It?
Event Sourcing

Good Definition

“Event Sourcing ensures that all changes to application state are stored as a sequence of events. Not just can we query these events, we can also use the event log to reconstruct past states, and as a foundation to automatically adjust the state to cope with retroactive changes.”

— Martin Fowler⁴

⁴ Event Sourcing – Martin Fowler
Event Sourcing

Key Concepts

- The storage system becomes an *additive only* architecture
- Append-only architectures *distribute*
- Far *fewer* locks to deal with
- When using an Event Store there is only *one key*
- Criteria is *tracked* from inception as an event stream
- You can answer questions *form the origin of the system*
- You can answer questions *not asked yet*
- Natural *audit log*
# Event Sourcing

<table>
<thead>
<tr>
<th>Date</th>
<th>Comment</th>
<th>Change</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/1/2012</td>
<td>Deposit from 3300</td>
<td>+10000.00</td>
<td>10000.00</td>
</tr>
<tr>
<td>1/3/2012</td>
<td>Check 1</td>
<td>-4000.00</td>
<td>6000.00</td>
</tr>
<tr>
<td>1/4/2012</td>
<td>ATM withdraw</td>
<td>-3.00</td>
<td>5997.00</td>
</tr>
<tr>
<td>1/11/2012</td>
<td>Check 2</td>
<td>-5.00</td>
<td>5992.00</td>
</tr>
<tr>
<td>1/12/2012</td>
<td>Deposit from 3301</td>
<td>+2000.00</td>
<td>7992.00</td>
</tr>
</tbody>
</table>
Crud

We All Know This!
Crud

Good Definition

In computer programming, create, read, update, and delete (as an acronym CRUD) are the four basic functions of persistent storage.

— Wikipedia

---

5 Create, Read, Update, Delete – Wikipedia
Crud

What I Mean Is ...

The relational current state model.
**Crud**

**Key Concepts**

- Represent *change between two points*
- Commonly referred to as *deltas*
- In static state models deltas are *implicit*
- They are left to *frameworks* such as an ORM
- ORMs save state, calculate diffs, update backing model
- As a result much of the *intent or behavior* is lost
- Auditing is almost always *explicit*
Crud

Order 123
Total $47

Item 1 @ $30

Shipping Information
Fast Data

What Is It?
Fast Data

Good Definition

“Where data is processed as it arrives, leading to so-called fast data systems that ingest and process continuous, potentially infinite data streams.”

— Dean Wampler⁶

Fast Data
Fast Data

Key Concepts

→ (Near) **Real-time boundaries are defined by the SLA**
→ **Streamed data has synthetic temporal boundaries**
→ **Streams can potentially be infinite**
→ **Steaming models should implement back-pressure**
→ **Reality is based on snapshots**
→ **Use operations like GROUP BY and JOIN**
Feedback Control

What Is It?
Feedback Control

Good Definition

Feedback works by constantly comparing the actual behavior of a system to its desired behavior. If the actual behavior differs from the desired one, a corrective action is applied to counteract the deviation and drive the system back to its target.

— Feedback Control for Computer Systems

7 Feedback Control for Computer Systems
Feedback Control

Key Concepts

→ Feedback control requires little knowledge about the system
→ One needs the direction in which to nudge the system
→ Feedback systems are self-correcting in the presence of external disturbances
→ One must be careful not overcorrect a disturbance
→ Control actions replace a deviation with another
Feedback Control

Key Concepts

- **Laws of behavior** in computer systems are much less constrained than physical assemblies
- Favor **anomaly detection** over **threshold violations**
- Use **smoothing algorithms**
- **Train your model** with **historical data**
Feedback Control

Diagram showing the flow of a feedback control system:
- Disturbance
- Input
- System
- Measuring Element
- Value
- Error
- Set Point
- Controller
- Effector
- Command
- Feedback
ReCap

Let's Review
ReCap

Outline

- Micro Services
- Promise Theory
- Event Sourcing
- Crud
- Fast Data
- Feedback Control
Building Systems
From Theory To Reality
Building Systems

The Big Question

Now that we have a good sense of the components, how do we decide what goes where and how to use it?
Building Systems

Micro Services Recap

→ Apply Promise Theory
→ Own your data
→ Keep your services small, hence the name micro
→ Cluster your micro service for resilience
→ Consistency and load times are determined by your SLA
→ Use polyglot persistence
Building Systems

Start With Micro Services
Micro Service Owns Its Own Data
Building Systems

Event Sourcing vs Crud

→ If you care about *history* use event sourcing
→ If you care about *auditing* use event sourcing
→ If you need to *shard* your data use event sourcing
→ If you need *strong consistency* use crud
→ If you only care about *current state* use crud
Building Systems

Implement Event Sourcing, CQRS and CRUD
Micro Service Owns Its Own Data
Building Systems

Fast Data

→ Remember Fast Data is based on streaming
→ Streamed data has synthetic temporal boundaries
→ Fast Data should be orthogonal to our micro services
→ Think of Fast Data as a data backplane
→ Fast Data isn’t a historical persistent store
Building Systems
Add Fast Data (an analogy is in order)
Think of your Fast Data pipe as a river with many communities pulling from it.
Building Systems

Add Fast Data
Micro Service Owns Its Own Data
Building Systems

Feedback Control

→ Feedback control can be the secret sauce in a system
→ Feedback control can be orthogonal to your micro services
→ Feedback control can and should be within a micro service
→ Remember to be careful not overcorrect a disturbance
→ Remember favor anomaly detection over threshold violations
→ Remember use smoothing algorithms
→ Remember train your model with historical data
Building Systems

Add Feedback Control (within)
import akka.actor.OneForOneStrategy
import akka.actor.SupervisorStrategy._
import scala.concurrent.duration._

override val supervisorStrategy =
    OneForOneStrategy(maxNrOfRetries = 10, withinTimeRange = 1 minute) {
    case _ : ArithmeticException => Resume
    case _ : NullPointerException => Restart
    case _ : IllegalArgumentException => Stop
    case _ : Exception => Escalate
    case _ =>
}
Building Systems

Add Feedback Control (orthogonal)
Building Systems

Almost There

→ We are *almost* there
→ *One thing is missing* though
Building Systems

Add Insight
Building Systems

Insight (possible solution)

- **Instrumentation**: Local aggregation of events and metrics
- **Collection**: Metrics registry for local aggregation
- **Collection**: report to CollectD
- **Aggregation**: Distributed Apache Kafka cluster
Building Systems

Insight (possible solution)

- **Storage**: Elasticsearch storage for events
- **Storage**: Cassandra TSDB cluster for metrics
- **Analytics**: Spark for analytics
- **Notifications**: Riemann notifications
- **Visualization**: Grafana for metrics and events
- **Visualization**: Kibana data exploration and discovery
The End