Reactive Microsystems

The Evolution of Microservices at Scale

Jonas Bonér
@jboner
Microservices Are Hyped
So You Want To Strangle The Almighty Monolith And Move Towards Microservices?
DO NOT SETTLE FOR Microliths
Microlith

Single instance microservice, communicating over blocking protocols
Microlith

Single-instance microservice, communicating over blocking protocols

Not Resilient
Microlith

Single instance microservice, communicating over blocking protocols

Not Resilient

Not Scalable
“Event-Driven Architecture (EDA) is a design paradigm in which a software component executes in response to receiving one or more event notifications.”

- GARTNER
The Nature of Events
Events represent FACTS OF INFORMATION

- FACTS ARE IMMUTABLE
- FACTS ACCRUE - KNOWLEDGE CAN ONLY GROW
The Nature of Events

- **Events** represent **FACTS OF INFORMATION**
  - **FACTS ARE IMMUTABLE**
  - **FACTS ACCRUE - KNOWLEDGE CAN ONLY GROW**
- **Events/Facts** CAN BE DISREGARDED/IGNOURED
The Nature of Events

- Events represent FACTS OF INFORMATION
  - FACTS ARE IMMUTABLE
  - FACTS ACCRUE - KNOWLEDGE CAN ONLY GROW
- Events/Facts CAN BE DISREGARDED/IGNORED
- Events/Facts CAN NOT BE RETRACTED (once accepted)
The Nature of Events

- **Events** represent **FACTS OF INFORMATION**
  - **FACTS ARE IMMUTABLE**
  - **FACTS ACCRUE - KNOWLEDGE CAN ONLY GROW**

- **Events/Facts** **CAN BE DISREGARDED/IGNORED**

- **Events/Facts** **CAN NOT BE RETRACTED** (once accepted)

- **Events/Facts** **CAN NOT BE DELETED** (once accepted)
  - Might be needed for **LEGAL OR MORAL REASONS**
The Nature of Events

- **Events** represent **FACTS OF INFORMATION**
  - **FACTS ARE IMMUTABLE**
  - **FACTS ACCRUE - KNOWLEDGE CAN ONLY GROW**
- **Events/Facts** **CAN BE DISREGARDED/IGNORED**
- **Events/Facts** **CAN NOT BE RETRACTED (once accepted)**
- **Events/Facts** **CAN NOT BE DELETED (once accepted)**
  - Might be needed for **LEGAL OR MORAL REASONS**
- **Events/Facts** (new) **CAN INVALIDATE existing Facts**
Event Driven Services

1. RECEIVE and REACT (or not) TO FACTS, that are coming its way
2. PUBLISH NEW FACTS (as events) to the rest of the world
3. INVERT THE CONTROL FLOW to minimize coupling and increase autonomy
Practice
Reactive Design
Go Async
Never Block

- Efficient use of resources
- Minimizes contention
Always Apply Back Pressure

A FAST system Should NOT OVERLOAD
A SLOW system
Always Apply Back Pressure

A FAST system Should NOT OVERLOAD

A SLOW system
Microservices
Come In Systems
Event Stream
USE THE Event Stream AS THE INTEGRATION FABRIC
Event Driven Services
Event Driven Services
Event Driven Services

COMMAND

[Diagram showing a person with an arrow pointing to a command]

[Blank space for additional text or steps]
Event Driven Services
Event Driven Services

Command

human figure ➤ command ➤ event processing
Event Driven Services

Event Stream

Command
Event Driven Services

Event Stream

Event

Event

Event

Command
Event Driven Services

Event Stream

Command
Event Driven Services

Event Stream
Event Driven Services

Event Stream

Command

Database
Event Driven Services

Event Stream

Event

Event

Event

Command

Mailbox ➔ Refresh
Event Driven Services

Event Stream

Eventual Consistency

Command
Eventual Consistency
No one wants eventual consistency. It's a necessary evil.
Information Has Latency
Information Is Always From the Past
Welcome To The Wild Ocean Of Non Determinism Distributed Systems
We Need To Model Uncertainty

“In a system which cannot count on distributed transactions, the management of uncertainty must be implemented in the business logic.”

- PAT HELLAND
Exploit Reality
WE NEED TO

Exploit Reality

IN OUR DESIGN
Events Can Lead To Greater Certainty
Events Can Help Us Craft Autonomous Islands Of Determinism
Mutable State Needs To Be Contained And Non Observable
Publish Facts To Outside World
Microservices Come As Systems
Each microservice needs to be designed as a distributed system.

A MICROSYSTEM
Event Driven Microservices
Powered By
Reactive Systems Fabric

Ladder of abstraction
Event Driven Microservices
Powered By
Reactive Systems Fabric
Event Driven Microservices
Powered By
Reactive Systems Fabric

Ladder of abstraction
Event Driven Microservices
Powered By
Reactive Systems Fabric

Ladder of abstraction
Event Driven Microservices
Powered By
Reactive Systems Fabric

Event-driven Microservices (Pub/Sub, Point-to-Point, Streaming)

Network Boundary

Ladder of abstraction
Event Driven Microservices
Powered By
Reactive Systems Fabric

Event-driven Microservices (Pub/Sub, Point-to-Point, Streaming)

Ladder of abstraction

NETWORK BOUNDARY

EVENT 1
Event Driven Microservices
Powered By
Reactive Systems Fabric

Event-driven Microservices (Pub/Sub, Point-to-Point, Streaming)

Local Event Loop

Network Boundary

EVENT 1

Ladder of abstraction
Event Driven Microservices
Powered By
Reactive Systems Fabric

Event-driven Microservices (Pub/Sub, Point-to-Point, Streaming)

Local Event Loop

Distribution Fabric

Network Boundary

Event 1

Ladder of abstraction
Event Driven Microservices
Powered By
Reactive Systems Fabric

Event-driven Microservices (Pub/Sub, Point-to-Point, Streaming)

Local Event Loop

Distribution Fabric

Network Boundary

Message Passing

Distribution Fabric

Ladder of abstraction
Event Driven Microservices
Powered By
Reactive Systems Fabric

Event-driven Microservices (Pub/Sub, Point-to-Point, Streaming)

Local Event Loop

Distribution Fabric

MESSAGE PASSING

Local Event Loop

Distribution Fabric

NETWORK BOUNDARY

EVENT 1

Ladder of abstraction
Event Driven Microservices
Powered By
Reactive Systems Fabric

Event-driven Microservices (Pub/Sub, Point-to-Point, Streaming)

Local Event Loop

Distribution Fabric

MESSAGE PASSING

Local Event Loop

Distribution Fabric

EVENT 1

NETWORK BOUNDARY

Ladder of abstraction
Event Driven Microservices
Powered By
Reactive Systems Fabric

- Event-driven Microservices (Pub/Sub, Point-to-Point, Streaming)
- Local Event Loop
- Distribution Fabric
- Network Boundary
- Event 1
- Local Event Loop
- Distribution Fabric
- Message Passing
Event Driven Microservices
Powered By
Reactive Systems Fabric

Event-driven Microservices (Pub/Sub, Point-to-Point, Streaming)

Local Event Loop

Network Boundary

EVENT 1

EVENT 1

Local Event Loop

Ladder of abstraction

Reactive Systems is doing the heavy lifting

Distribution Fabric

MESSAGE PASSING

Distribution Fabric
Event Driven Microservices
Powered By
Reactive Systems Fabric

Event-driven Microservices (Pub/Sub, Point-to-Point, Streaming)

Local Event Loop

Network Boundary

EVENT 1

REACTIVE PROGRAMMING MAINTAINS A SIMPLE PROGRAMMING MODEL

Event-driven Microservices (Pub/Sub, Point-to-Point, Streaming)

Local Event Loop

Ladder of abstraction

REACTIVE SYSTEMS IS DOING THE HEAVY LIFTING

Distribution Fabric

MESSAGE PASSING

Distribution Fabric

Reactive Programming maintains a simple programming model.

Reactive Systems is doing the heavy lifting.
Events First
Domain Driven
Design
“When you start modeling events, it forces you to think about the behavior of the system. As opposed to thinking about the structure of the system.”

- GREG YOUNG
DON'T FOCUS ON THE THINGS

The Nouns
The Domain Objects

FOCUS ON WHAT HAPPENS

The Verbs
The Events
Mine the Facts
Event Storming
Event Driven Design
Event Driven Design

INTENTS
- Communication
- Conversations
- Expectations
- Contracts
- Control Transfer
Event Driven Design

**INTENTS**
- Communication
- Conversations
- Expectations
- Contracts
- Control Transfer

**FACTS**
- State
- History
- Causality
- Notifications
- State Transfer
Event Driven Design

**INTENTS**
- Communication
- Expectations
- Contracts
- Control Transfer

**FACTS**
- State
- History
- Causality
- Notifications
- State Transfer

※ Commands
Event Driven Design

**INTENTS**
- Communication
- Expectations
- Contracts
- Control Transfer

**FACTS**
- State
- Causality
- Notifications
- State Transfer

**Commands**

**Events**
Event Driven Design
Event Driven Design

**COMMANDS**

- Object form of METHOD/ACTION REQUEST
- IMPERATIVE: CreateOrder, ShipProduct
Event Driven Design

**COMMANDS**
- Object form of METHOD/ACTION REQUEST
- IMPERATIVE: CreateOrder, ShipProduct

**REACTIONS**
- Represents SIDE-EFFECTS
Event Driven Design

**COMMANDS**

- Object form of METHOD/ACTION REQUEST
- IMPERATIVE: CreateOrder, ShipProduct

**REACTIONS**

- Represents SIDE-EFFECTS

**EVENTS**

- Represents something that HAS HAPPENED
- PAST-TENSE: OrderCreated, ProductShipped
COMMANDS  vs  EVENTS
COMMANDS vs EVENTS

1. All about intent

1. Intentless
COMMANDS vs EVENTS

1. All about intent
2. Directed

1. Intentless
2. Anonymous
COMMANDS vs EVENTS

1. All about intent
2. Directed
3. Single addressable destination

1. Intentless
2. Anonymous
3. Just happens – for others (0–N) to observe
COMMANDS vs EVENTS

1. All about intent
2. Directed
3. Single addressable destination
4. Models personal communication

1. Intentless
2. Anonymous
3. Just happens – for others (0–N) to observe
4. Models broadcast (speakers corner)
COMMANDS vs EVENTS

1. All about intent
2. Directed
3. Single addressable destination
4. Models personal communication
5. Distributed focus

1. Intentless
2. Anonymous
3. Just happens – for others (0–N) to observe
4. Models broadcast (speakers corner)
5. Local focus
COMMANDS vs EVENTS

1. All about intent
2. Directed
3. Single addressable destination
4. Models personal communication
5. Distributed focus
6. Command & Control

1. Intentless
2. Anonymous
3. Just happens – for others (0–N) to observe
4. Models broadcast (speakers corner)
5. Local focus
6. Autonomy
Let the Events Define the Bounded Context
ARE WE DONE NOW?
Perhaps. We have come a long way. But events can also be used for:

➡ PERSISTENCE
➡ MANAGING TIME
Event Based Persistence
“Update-in-place strikes systems designers as a cardinal sin: it violates traditional accounting practices that have been observed for hundreds of years.”

- JIM GRAY

The Transaction Concept, Jim Gray (1981)
Event Logging

The Bedrock

<table>
<thead>
<tr>
<th>Item</th>
<th>Quantity</th>
<th>Unit</th>
<th>Price</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item 1</td>
<td>10</td>
<td>Unit</td>
<td>5.00</td>
<td>50.00</td>
</tr>
<tr>
<td>Item 2</td>
<td>15</td>
<td>Unit</td>
<td>3.50</td>
<td>52.50</td>
</tr>
<tr>
<td>Item 3</td>
<td>5</td>
<td>Unit</td>
<td>2.50</td>
<td>12.50</td>
</tr>
</tbody>
</table>

Notes:
- Item 1: Made of silver.
- Item 2: Imported from Germany.
- Item 3: Special order.
“The truth is the log. The database is a cache of a subset of the log.”

- PAT HELLAND

Immutability Changes Everything, Pat Helland (2015)
The Log
A Database Of the Past
Not Just the Present
Event Sourcing
A Cure For the Cardinal Sin
Event Sourced Services
Event Sourced Services
Event Sourced Services

1) Receive and verify Command ("ApprovePayment")
Event Source Services

HAPPY PATH

1) Receive and verify Command ("ApprovePayment")
2) Create new Event ("PaymentApproved")
Event Sourced Services

1) Receive and verify Command ("ApprovePayment")
2) Create new Event ("PaymentApproved")
3) Append Event to Event Log

Happy Path
Event Sourced Services

HAPPY PATH

1) Receive and verify Command ("ApprovePayment")
2) Create new Event ("PaymentApproved")
3) Append Event to Event Log
4) Update internal component state
Event Sourced Services

1) Receive and verify Command (“ApprovePayment”)
2) Create new Event (“PaymentApproved”)
3) Append Event to Event Log
4) Update internal component state
5) Run side-effects (approve the payment)
Event Sourced Services

**HAPPY PATH**

1. Receive and verify Command ("ApprovePayment")
2. Create new Event ("PaymentApproved")
3. Append Event to Event Log
4. Update internal component state
5. Run side-effects (approve the payment)

---

**SAD PATH - RECOVERING FROM FAILURE**
Event Sourced Services

**Happy Path**
1) Receive and verify Command ("ApprovePayment")
2) Create new Event ("PaymentApproved")
3) Append Event to Event Log
4) Update internal component state
5) Run side-effects (approve the payment)

**Sad Path - Recovering from Failure**
1) Rehydrate Events from Event Log
Event Sourced Services

**HAPPY PATH**

1) Receive and verify Command ("ApprovePayment")
2) Create new Event ("PaymentApproved")
3) Append Event to Event Log
4) Update internal component state
5) Run side-effects (approve the payment)

**SAD PATH - RECOVERING FROM FAILURE**

1) Rehydrate Events from Event Log
2) Update internal component state
Event Sourced Services

Happy Path

1) Receive and verify Command ("ApprovePayment")
2) Create new Event ("PaymentApproved")
3) Append Event to Event Log
4) Update internal component state
5) Run side-effects (approve the payment)

SAD Path - Recovering from Failure

1) Rehydrate Events from Event Log
2) Update internal component state

Memory Image
BENEFITS OF USING
Event Sourcing
BENEFITS OF USING

Event Sourcing

★ One single SOURCE OF TRUTH with ALL HISTORY
BENEFITS OF USING
Event Sourcing

- One single SOURCE OF TRUTH with ALL HISTORY
- Allows for MEMORY IMAGE - durable in-memory state
BENEFITS OF USING

Event Sourcing

✴ One single SOURCE OF TRUTH with ALL HISTORY
✴ Allows for MEMORY IMAGE - durable in-memory state
✴ Avoids the OBJECT-RELATIONAL IMPEDENCE MISMATCH
BENEFITS OF USING

Event Sourcing

✴ One single SOURCE OF TRUTH with ALL HISTORY
✴ Allows for MEMORY IMAGE - durable in-memory state
✴ Avoids the OBJECT-RELATIONAL IMPEDENCE MISMATCH
✴ Allows others to SUBSCRIBE TO STATE CHANGES
BENEFITS OF USING Event Sourcing

✴ One single SOURCE OF TRUTH with ALL HISTORY
✴ Allows for MEMORY IMAGE - durable in-memory state
✴ Avoids the OBJECT-RELATIONAL IMPEDANCE MISMATCH
✴ Allows others to SUBSCRIBE TO STATE CHANGES
✴ Mechanical SYMPATHY (Single Writer Principle etc.)
Disadvantages Of Using Event Sourcing

- **UNFAMILIAR** model
- **VERSIONING** of events
- **DELETION** of events (legal or moral reasons)
Events
Allow Us To Manage
Time
“Modeling events forces you to have a temporal focus on what’s going on in the system. Time becomes a crucial factor of the system.”

- GREG YOUNG
Event Sourcing Allows Us To Model Time

- Event is a **SNAPSHOT IN TIME**
- Event ID is an **INDEX FOR TIME**
- Event Log is our **FULL HISTORY**
- The **DATABASE OF OUR PAST**
- The **PATH TO OUR PRESENT**
Event Sourcing Allows For Time Travel
Event Sourcing Allows For Time Travel
Event Sourcing Allows For Time Travel

 Replay the log FOR HISTORIC DEBUGGING
 Replay the log FOR AUDITING & TRACEABILITY
 Replay the log ON FAILURE
 Replay the log FOR REPLICATION
We Can Even Fork the Past

...Or Join Two Distinct Pasts
Key Takeaways
Key Takeaways

1. Don’t build Microliths
Key Takeaways

1. Don’t build Microliths
2. Microservices come in (distributed) systems
Key Takeaways

1. Don’t build Microliths
2. Microservices come in (distributed) systems
3. Microservices come as (micro)systems
1. Don’t build Microliths
2. Microservices come in (distributed) systems
3. Microservices come as (micro)systems
4. Embrace the Reactive Principles
Key Takeaways

1. Don’t build Microliths
2. Microservices come in (distributed) systems
3. Microservices come as (micro)systems
4. Embrace the Reactive Principles
5. Embrace Events-first Domain Driven Design
Key Takeaways

1. Don’t build Microliths
2. Microservices come in (distributed) systems
3. Microservices come as (micro)systems
4. Embrace the Reactive Principles
5. Embrace Events-first Domain Driven Design
6. Embrace Event-driven Architecture
Key Takeaways

1. Don’t build Microliths
2. Microservices come in (distributed) systems
3. Microservices come as (micro)systems
4. Embrace the Reactive Principles
5. Embrace Events–first Domain Driven Design
6. Embrace Event–driven Architecture
7. Embrace Event–driven Persistence
Key Takeaways

1. Don’t build Microliths
2. Microservices come in (distributed) systems
3. Microservices come as (micro)systems
4. Embrace the Reactive Principles
5. Embrace Events-first Domain Driven Design
6. Embrace Event-driven Architecture
7. Embrace Event-driven Persistence
8. Profit!!!