Opportunities & Pitfalls of Event-Driven Utopia

aberndruecker
Why this talk
Towards the end of last year I attended a workshop with my colleagues in ThoughtWorks to discuss the nature of “event-driven” applications.

The biggest outcome of the summit was recognizing that when people talk about “events”, they actually mean some quite different things. So we spent a lot of time trying to tease out what some useful patterns might be.
Agenda

1. Events on the inside
   Service 1

2. Events inside out

3. Events on the outside
   Service 2
Agenda

1. Events on the inside

Service 1

2. Events inside out

3. Events on the outside

Service 2
Once upon a time...

**BBC architecture**
(box - arrow - box - arrow - cylinder)
Every architecture diagram you'll ever need
The great thing about this architecture
The problem

Application

RDMS

not cloud-scale
resiliency is expensive

Does not fit in Kubernetes:

RDMS
Immutability Changes Everything!
Append-only Log

| bank account created | transferred | -14.99$ paid by credit card |

Current Balance = 2,485.01 $

Persistent state

<table>
<thead>
<tr>
<th>Account #</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345</td>
<td>2,500$</td>
</tr>
</tbody>
</table>
**Persistent change**

<table>
<thead>
<tr>
<th>Event</th>
<th>Bank Account Created</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>2019/04/16 11:00</td>
</tr>
<tr>
<td>#</td>
<td>12345</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Event</th>
<th>Money Transfer Received</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>2019/04/16 11:00</td>
</tr>
<tr>
<td>#</td>
<td>12345</td>
</tr>
</tbody>
</table>

### Append-only Log

<table>
<thead>
<tr>
<th>bank account created</th>
<th>+2,500 $ transferred</th>
</tr>
</thead>
<tbody>
<tr>
<td>-14.99$ paid by credit card</td>
<td></td>
</tr>
</tbody>
</table>
An example from my world
Warning: Contains opinion!

Bernd Ruecker
Co-founder and Chief Technologist of Camunda

mail@berndruecker.io
@berndruecker
We build two different workflow engines. Why?

- **Camunda BPM**
  - Persistent State

- **Zeebe**
  - Persistent change
<table>
<thead>
<tr>
<th>Workflow Instance Id</th>
<th>Current Activity</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>RetrievePayment</td>
<td>running</td>
</tr>
</tbody>
</table>

1. **UPDATE**

<table>
<thead>
<tr>
<th>Workflow Instance Id</th>
<th>Current Activity</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>ShipGoods</td>
<td>running</td>
</tr>
</tbody>
</table>

2. **UPDATE**

<table>
<thead>
<tr>
<th>Workflow Instance Id</th>
<th>Current Activity</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>OrderDelivered</td>
<td>ended</td>
</tr>
</tbody>
</table>

3. **UPDATE**
1. Create workflow instance
   - Start event occurred
   - Activity activated
   - Lock created

2. Complete task
   - Activity completed

Workflow Engine

Append-only Log
Event Handling, Replication & Single Writer

1. **Send**
2. **Append command**
3. **Store & replicate command**
4. **Process**
5. **Respond**
6. **Append event**
7. **Store & replicate event**

**Single Writer (single thread)**

**Stream Processor**

**Broker (Leader)**

**Follower**

**RocksDB**
What we do different

1. Send
2. Append command
3. Store & replicate command
4. Process
5. Respond
6. Append event
7. Store & replicate event

- Delete records that are fully processed
- Persist & replicate internal state

1. Single Writer (single thread)
2. Stream Processor
3. Leader
4. Follower
Consistency
Availability
Partition
Zeebe is CP

1. send
2. append command
3. store & replicate command
4. process
5. respond
6. append event
7. store & replicate event
Horizontal scalability by partitioning

Every workflow instance is exactly handled by one partition
Queries and read models

Zeebe Broker

Streaming Exporter

elasticsearch
Recap 1 – Events on the inside

# Natural mechanism to build scalable services in distributed systems (with Outbox & co included)

But
# You have to think about reads, queries & eventual consistency
# Few industry experience available
Agenda

1. Events on the inside

2. Events inside out

3. Events on the outside
Event Store and Messaging

1. Create Customer

Customer Event Store
Merge Messaging and Event Store

1. Create Customer

Customer

Customer Event Store
Merge messaging and event store

1. Create Customer

Shared Event Store
Enter the world of Kafka...
Merge messaging and event store

1. Create Customer

Shared Event Store
Kafka as transport

1. Create Customer

Used as queue (but persistent!)

Customer
Kafka as transport

1. Create Customer

Used as queue (but persistent!)
Agenda

1. Events on the inside

Service 1

2. Events inside out

3. Events on the outside

Service 2
Once upon a time

Customer

Change Address

Billing
Event Notification

Customer

Address changed

Billing
Event Notification

Reverse direction of dependency

Customer
Change Address
Billing

Customer
Address changed
Billing

What's general
Which
Who
Change Address

Event Notification

Address Changed

```json
{  customerId: 42
}
```

Ask for details
Event-carried State Transfer

AddressChanged
{
  customerId: 42,
  oldAddress: ...
  newAddress: ...
}

AddressChanged
{
  customerId: 42,
  address: ...
}

CustomerChanged
{
  customerId: 42,
  status: A,
  address: ...
}

CustomerMoved
{
  ...
}
This decision is complex

Reverse direction of dependency

Billing

Customer

Change Address
Example

Change Address

Address

Submit

Incoming Email

From: bla@company.com
Date: 2019-04-23 09:05

To confirm your address change please click on this link:

http://company.com/confirm?id=82e97d49-166c-4862-9973-4db548e6225d
Example

Change Address

Address change requested

Address change confirmed

Notification

http://company.com/confirm?id=8e97d49-4852-9973-4b3f5d62b5d

direction of dependency

Customer

Submit

http://company.com/confirm?id=8e97d49-4852-9973-4b3f5d62b5d

http://company.com/confirm?id=8e97d49-4852-9973-4b3f5d62b5d

http://company.com/confirm?id=8e97d49-4852-9973-4b3f5d62b5d
Example

Customer

Address changed

Send mail 'Confirmation'

'Confirmation' approved

Change Address

Notification

http://company.com/confirm?id=82e97d49-166c-4832-9933-4ab348e6225d
Challenge: Command vs. Event
It is NOT about communication protocols

It can be messaging, REST, whatever, ....
Manifold ways of transport

Apache Kafka®

RabbitMQ

webhooks

...
Manifold ways of transport
Fact, happened in the past, immutable

Intend, Want s.th. to happen, The intention itself is a fact
The Customer Needs To Be Sent A Message To Confirm Address Change Event
Examples

More general, does not need to know who is retrieving payments

More general, does not need to care about who is interested in address changes

Global service

Order Notification

Service that can handle notifications for orders autonomously
Distributed Monoliths
Define stable contract/API instead
Next challenge: Event chains
Event Chains

1. Registration requested
2. Credit check
3. Address check
4. Customer registered

Event Bus
Event Chains

How does customer registration work?
The danger is that it’s very easy to make nicely decoupled systems with event notification, without realizing that you’re losing sight of that larger-scale flow, and thus set yourself up for trouble in future years.

https://martinfowler.com/articles/201701-event-driven.html
The danger is that it’s very easy to make nicely decoupled systems with event notification, without realizing that you’re losing sight of that larger-scale flow, and thus set yourself up for trouble in future years.

https://martinfowler.com/articles/201701-event-driven.html
The danger is that it’s very easy to make nicely decoupled systems with event notification, without realizing that you’re losing sight of that larger-scale flow, and thus set yourself up for trouble in future years.

https://martinfowler.com/articles/201701-event-driven.html
Monitoring Workflows Across Microservices

Key Takeaways

- Peer-to-peer communication between components can lead to emergent behavior, which is challenging for developers, operators, and business analysts to understand.
- You need to make sure to have the overview of all the backwards-and-forwards communication that is going on in order to fulfill a business capability.
- Solutions that provide an overview range from distributed tracing, which typically misses the business perspective; data lakes, which require some effort to tune to what you need to know; process tracking, where you have to model a workflow for the tracking; process mining, which can discover the workflow, all the way through to orchestration, which comes with visibility built in.

https://www.infoq.com/articles/monitor-workflow-collaborating-microservices
Typical approaches

Distributed Tracing

Data Lake / Event Monitoring

Process Mining

@berndruecker
What we currently build with customers...

Camunda
Optimize

Registration requested → Credit checked → Address checked → Customer registered → Registration completed

Elastic
All great – until you have to move...
Keep it stable, just move that one stick down there to the top.

How hard can it be?
Changes required for an additional check

Registration requested → Credit checked → Event Bus → Address checked → Customer registered

Credit Check

Adress Check

Criminal Check
Changes required for an additional check

- Registration requested
- Credit checked
- Event Bus
- Address checked
- Criminal checked
- Customer registered
Alternative flow

"Credit checks got more expensive, do that only if all other checks succeed"
What we wanted vs. what we got

Photo by Lijian Zhang, under Creative Commons SA 2.0 License and Wikimedia Commons / CC BY-SA 4.0
Registration requested → Check credit → Check address → Address checked → Customer registered

Kafka

Orchestration
Changes...

- Registration requested
- Credit checked
- Check address
- Check crimes
- Crimes checked
- Customer registered
- Credit Check
- Address Check
- Criminal Check
Coupling myths
Comparison

See also https://www.infoworld.com/article/3391592/how-to-tame-event-driven-microservices.html
Your IT architecture

Your services or applications

Choreography

Orchestration

@berndruecker
Your services or applications

Your IT architecture

Choreography

Orchestration

Process Monitoring

Monolith

Chaos

@berndruecker
Balance choreography and orchestration

Your services or applications

Process Monitoring

Your IT architecture

Choreography

Orchestration

Monolith

Chaos
In my world...

Leverage Workflow Engine & BPMN within Service
Local Orchestration
Recap 2

# Commands vs. Events: Decide about the direction of dependencies

# Beware of event-chains and avoid losing sight

# Balance choreography and orchestration
Recap

1. Events on the inside
   - Persistent state vs persistent change
   - Event sourcing & Event Store
   - Consistency & CAP
   - Read Models & CQRS

2. Events inside out
   - Shared Event Store

3. Events on the outside

- Events as API
- Event vs Command
- Event chains & visibility
- Orchestration vs Choreography
Want to see code?

Sample application demonstrating an order fulfillment system decomposed into multiple independent components (e.g., microservices). Showing concrete implementation alternatives using e.g., Java, Spring Boot, Apache Kafka, Camunda, Zeebe, ...
Meet practitioners around orchestration & workflow
April 23-24

Capital One,
Cox Automotive,
Nokia Bell Labs,
Goldmann Sachs, ...
https://www.camundacon.com/
Nothing for the faint of heart...

Events on the inside

Events on the outside
Nothing for the faint of heart...

...but doable...

...and worth it
Thank you!
Contact: mail@berndruecker.io
@berndruecker

Slides: https://berndruecker.io

Blog: https://medium.com/berndruecker

Code: https://github.com/berndruecker

https://www.infoworld.com/article/3254777/application-development/
3-common-pitfalls-of-microservices-integrationand-how-to-avoid-them.html

https://www.infoq.com/articles/events-workflow-automation

https://thenewstack.io/5-workflow-automation-use-cases-you-might-not-have-considered/