

Event Sourcing at Yammer

Michał Rutkowski (mrutkowski@yammer-inc.com)

Dmitry Stratiychuk (dstratiychuk@yammer-inc.com)

Philipp Fehre (pfehre@yammer-inc.com)

What is this talk about?

- Challenges Yammer faced
- *Why event sourcing?*
- How we've rolled it out
- What we've learned
- The future

What is this talk about?

- Challenges Yammer faced
- *Why event sourcing?*
- How we've rolled it out
- What we've learned
- The future

What is this talk about?

- Challenges Yammer faced
- *Why event sourcing?*
- How we've rolled it out
- What we've learned
- The future

What is this talk about?

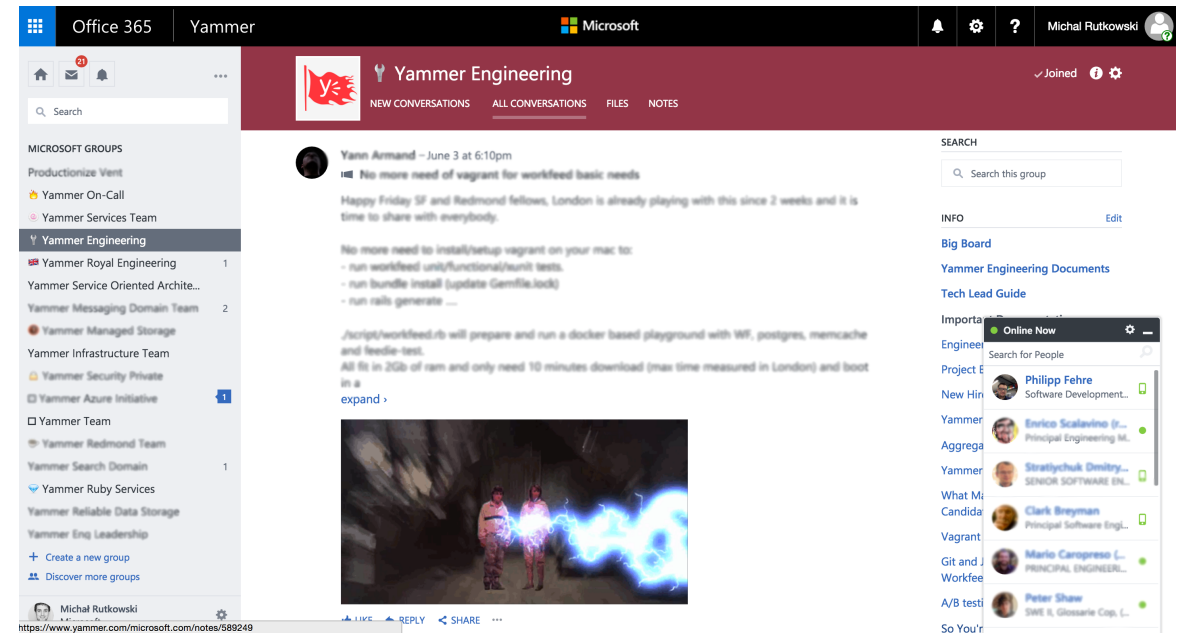
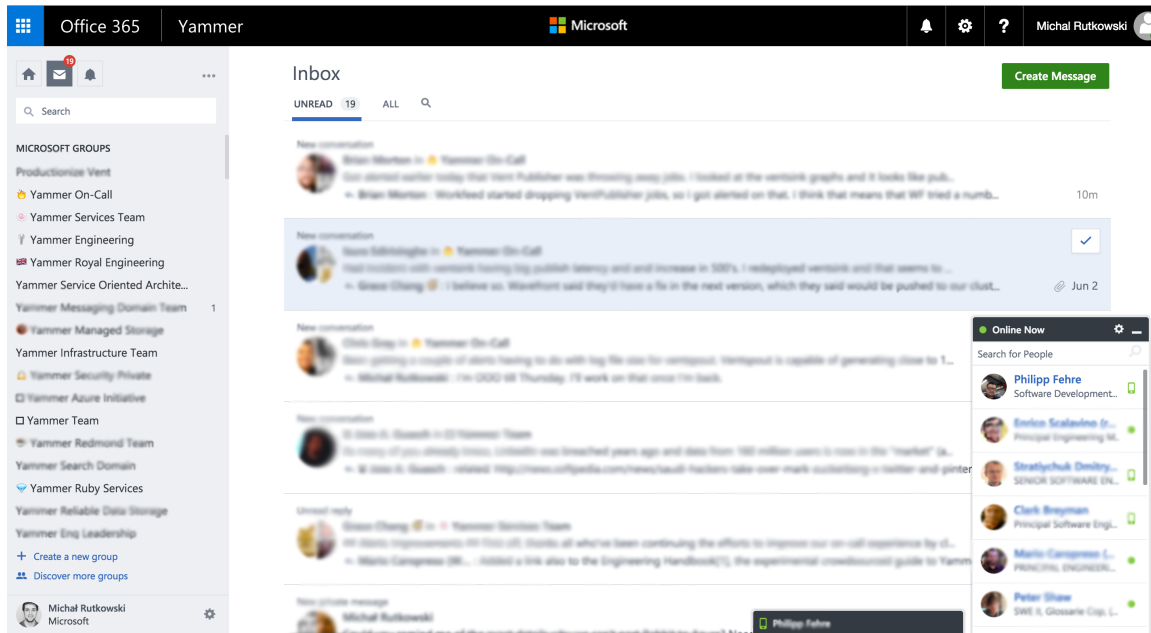
- Challenges Yammer faced
- *Why event sourcing?*
- How we've rolled it out
- **What we've learned**
- The future

What is this talk about?

- Challenges Yammer faced
- *Why event sourcing?*
- How we've rolled it out
- What we've learned
- The future

What is Yammer?

An Enterprise Social Network facilitating better and faster communication within an organization.



What is important?

- Our customers:
 - SLA
 - Performance
- **Velocity:** Ability to build and A/B test features fast

What is important?

- Our customers:
 - SLA
 - Performance
- **Velocity:** Ability to build and A/B test features fast

The Team

We want to help rest of the engineering team in:

- building quality features fast,
- while meeting our SLA commitments

To that end we:

- drive discussion and adoption of architectural patterns
- adopt and if necessary build tooling to facilitate that

The Team

We want to help rest of the engineering team in:

- building quality features fast,
- while meeting our SLA commitments

To that end we:

- drive discussion and adoption of architectural patterns
- adopt and if necessary build tooling to facilitate that

Things we've been working on

- Better tooling and process for release management:
 - Continuous delivery
 - Load testing
- Best practices for service design and development:
 - Testing for failure
 - Ensuring QoS
- Inter-service integration patterns

Things we've been working on

- Better tooling and process for release management:
 - Continuous delivery
 - Load testing
- Best practices for service design and development:
 - Testing for failure
 - Ensuring QoS
- Inter-service integration patterns

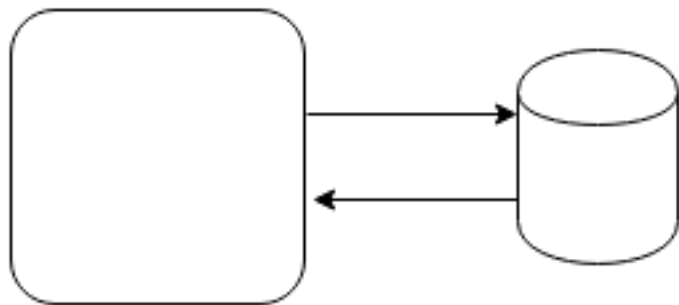
Things we've been working on

- Better tooling and process for release management:
 - Continuous delivery
 - Load testing
- Best practices for service design and development:
 - Testing for failure
 - Ensuring QoS
- Inter-service integration patterns

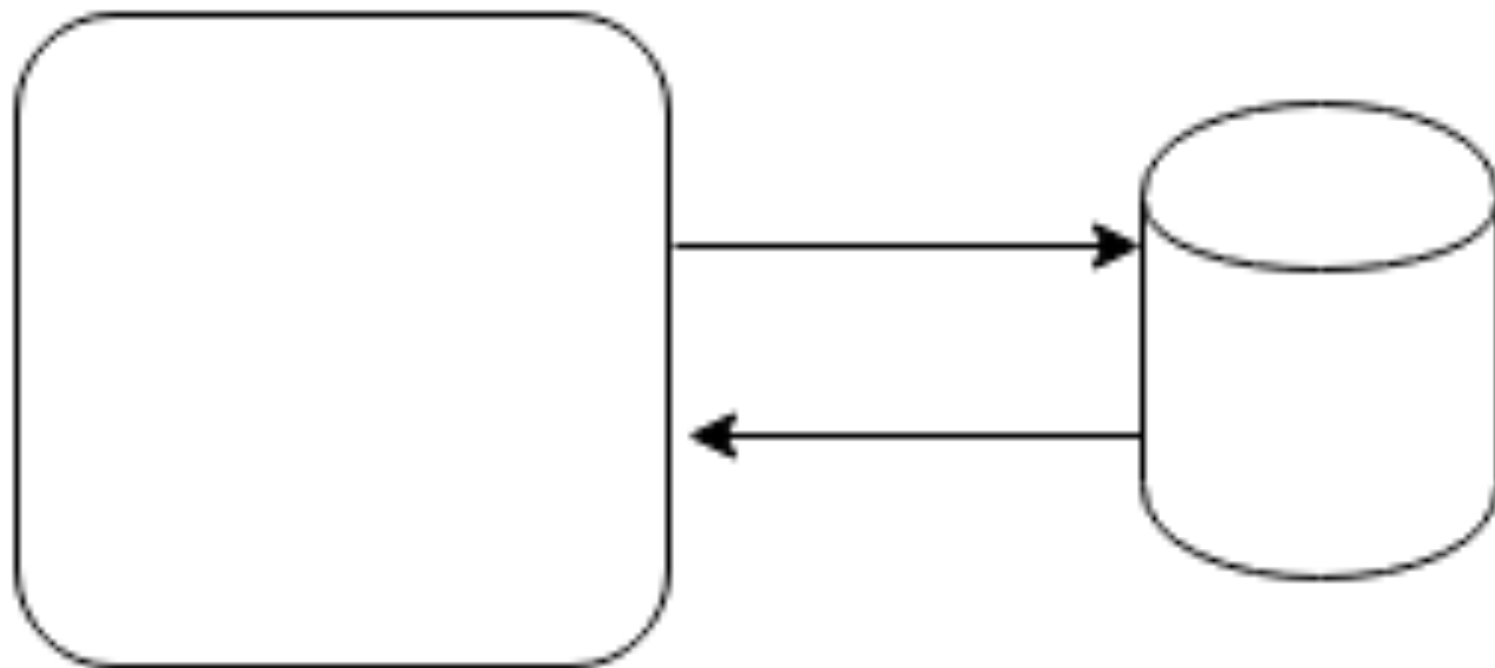
Things we've been working on

- Better tooling and process for release management:
 - Continuous delivery
 - Load testing
- Best practices for service design and development:
 - Testing for failure
 - Ensuring QoS
- **Inter-service integration patterns**

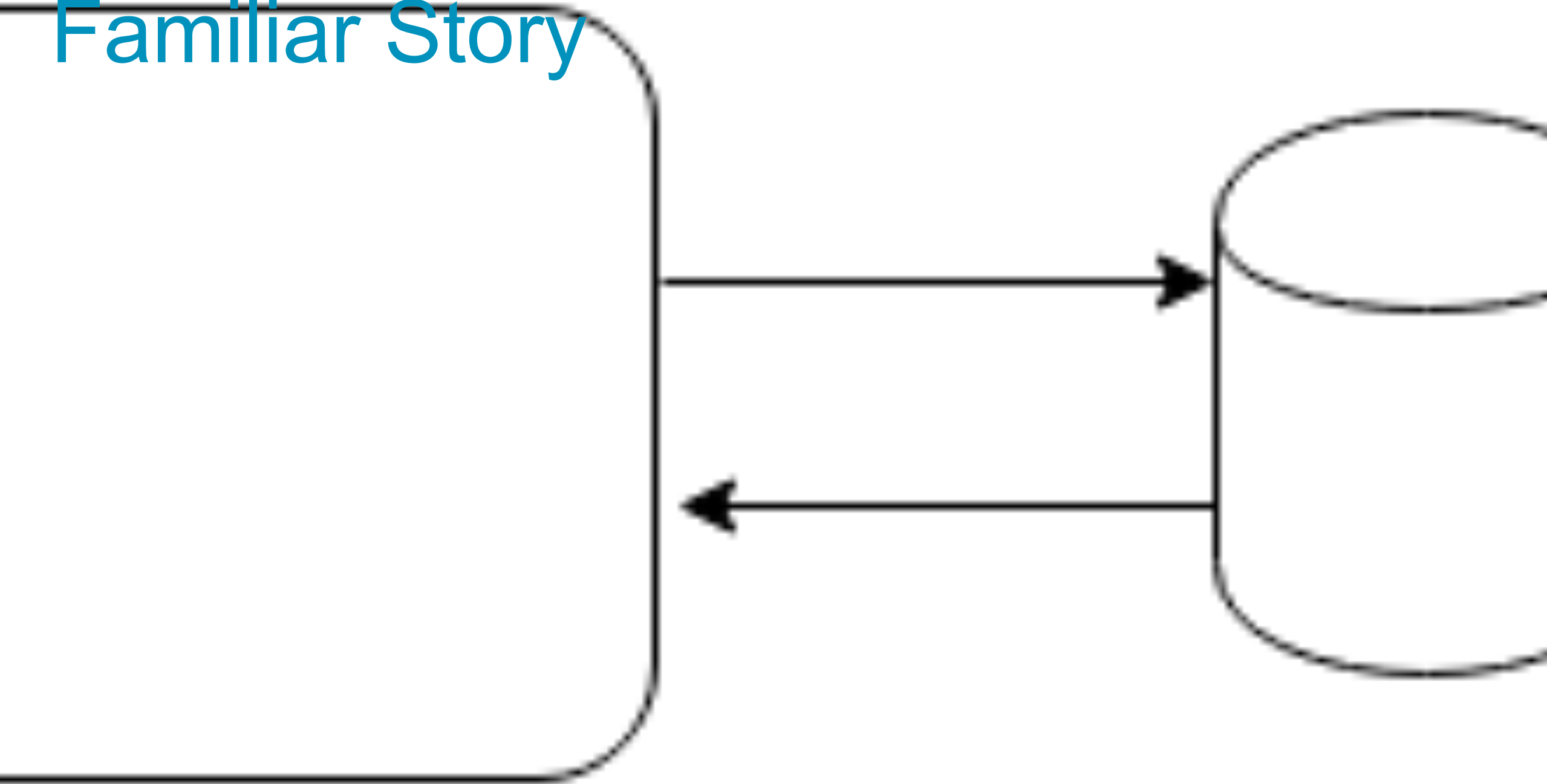
Familiar Story



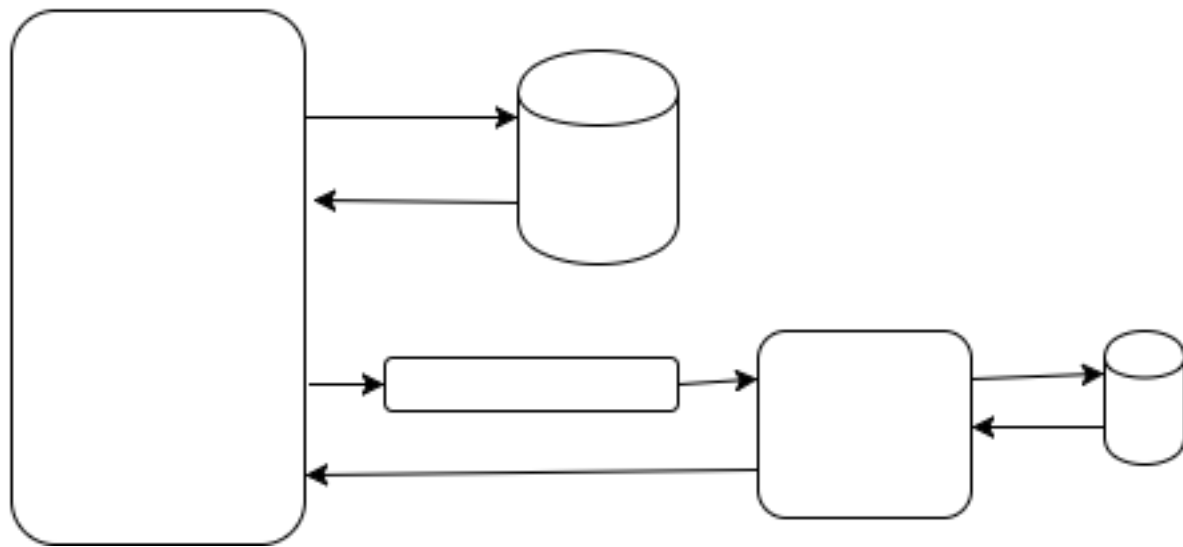
Familiar Story



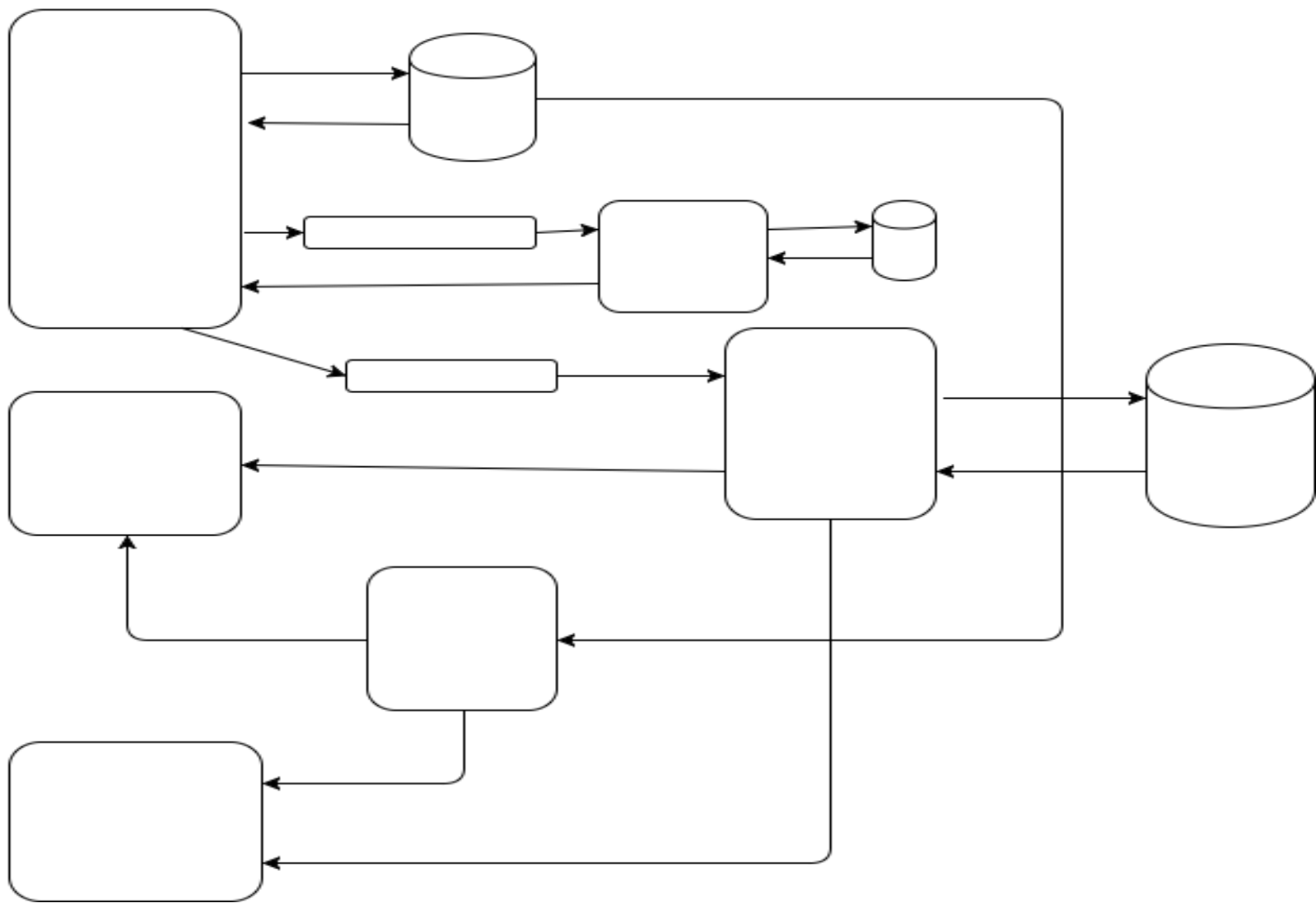
Familiar Story



Familiar Story



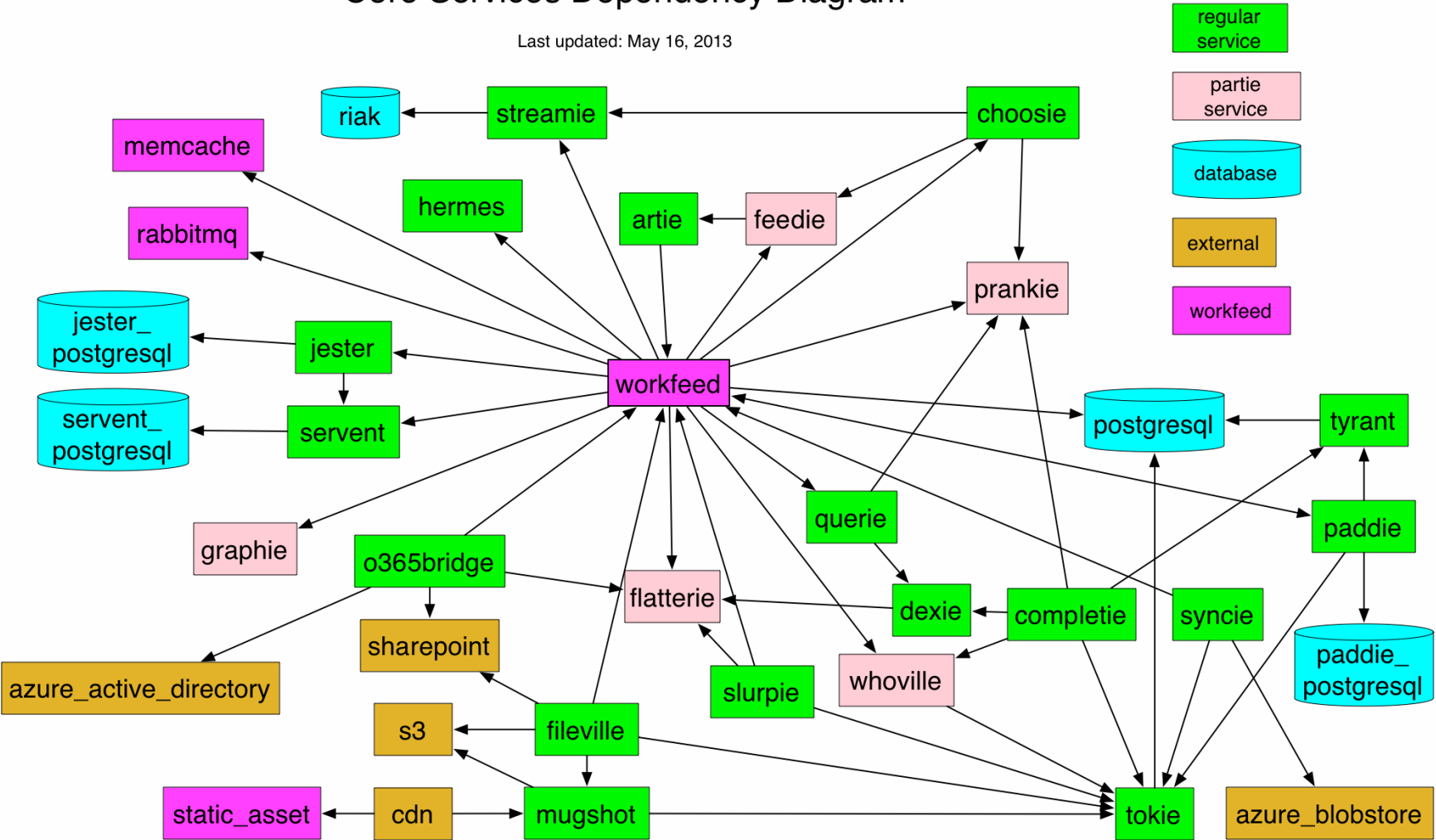
Familiar Story



Familiar Story

Core Services Dependency Diagram

Last updated: May 16, 2013



Familiar Story

That was mid-2013

Familiar Story

No one even tries to draw this diagram **now!**

Issues we've faced

Feature development become too slow:

- Too many inter-service dependencies
- Overly chatty services
- Too many inter-team, cross-time-zone dependencies

Issues we've faced

Feature development become too slow:

- Too many inter-service dependencies
- Overly chatty services
- Too many inter-team, cross-time-zone dependencies

Issues we've faced

Feature development become too slow:

- Too many inter-service dependencies
- Overly chatty services
- Too many inter-team, cross-time-zone dependencies

Issues we've faced

Meeting the SLA become much harder:

- Too many external dependencies on the read/write path
- Shared DB
- Un-expected, transitive dependencies
- Cascading failures (despite circuit breaking)
- Very easy to make a breaking code change

Issues we've faced

Meeting the SLA become much harder:

- Too many external dependencies on the read/write path
- Shared DB
- Un-expected, transitive dependencies
- Cascading failures (despite circuit breaking)
- Very easy to make a breaking code change

Issues we've faced

Meeting the SLA become much harder:

- Too many external dependencies on the read/write path
- Shared DB
- Un-expected, transitive dependencies
- Cascading failures (despite circuit breaking)
- Very easy to make a breaking code change

Issues we've faced

Meeting the SLA become much harder:

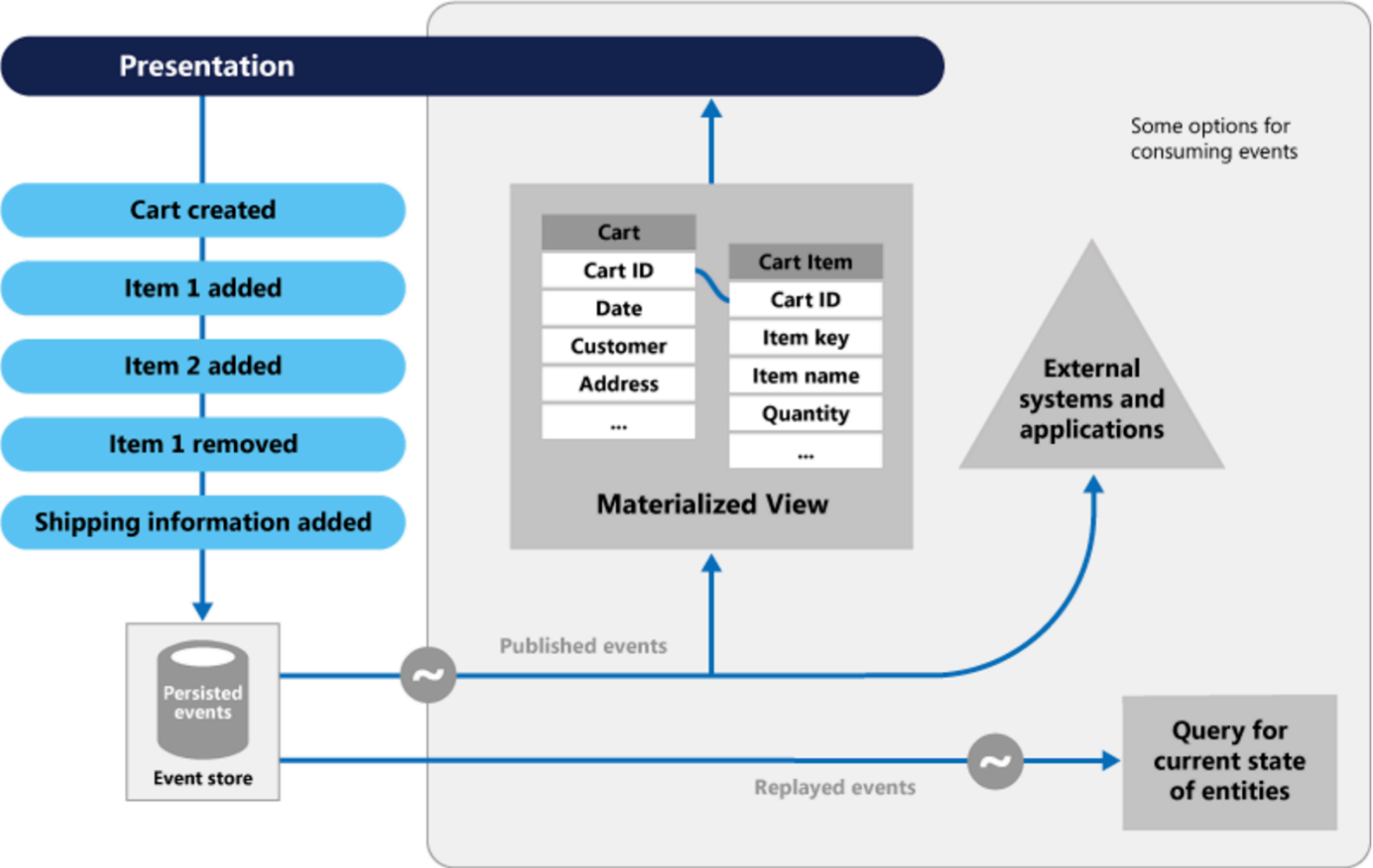
- Too many external dependencies on the read/write path
- Shared DB
- Un-expected, transitive dependencies
- Cascading failures (despite circuit breaking)
- Very easy to make a breaking code change

Issues we've faced

Meeting the SLA become much harder:

- Too many external dependencies on the read/write path
- Shared DB
- Un-expected, transitive dependencies
- Cascading failures (despite circuit breaking)
- Very easy to make a breaking code change

Event Sourcing



Event Sourcing

- One Data Owner Service
 - Publishes Events
 - Persists in DB
- View Services
 - Consume Events
 - Materialized Views (local DB)

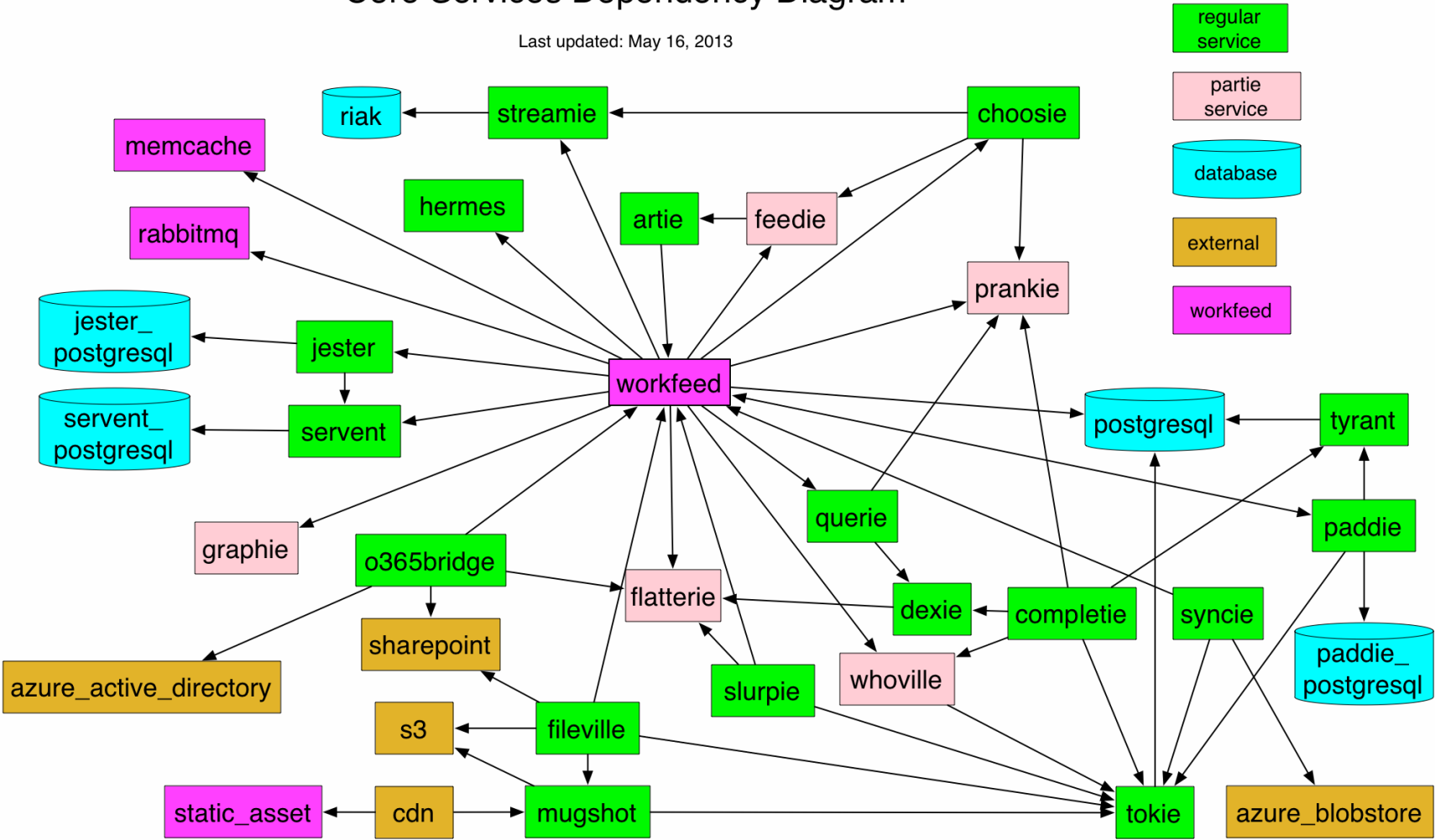
Event Sourcing

- One Data Owner Service
 - Publishes Events
 - Persists in DB
- View Services
 - Consume Events
 - Materialized Views (local DB)

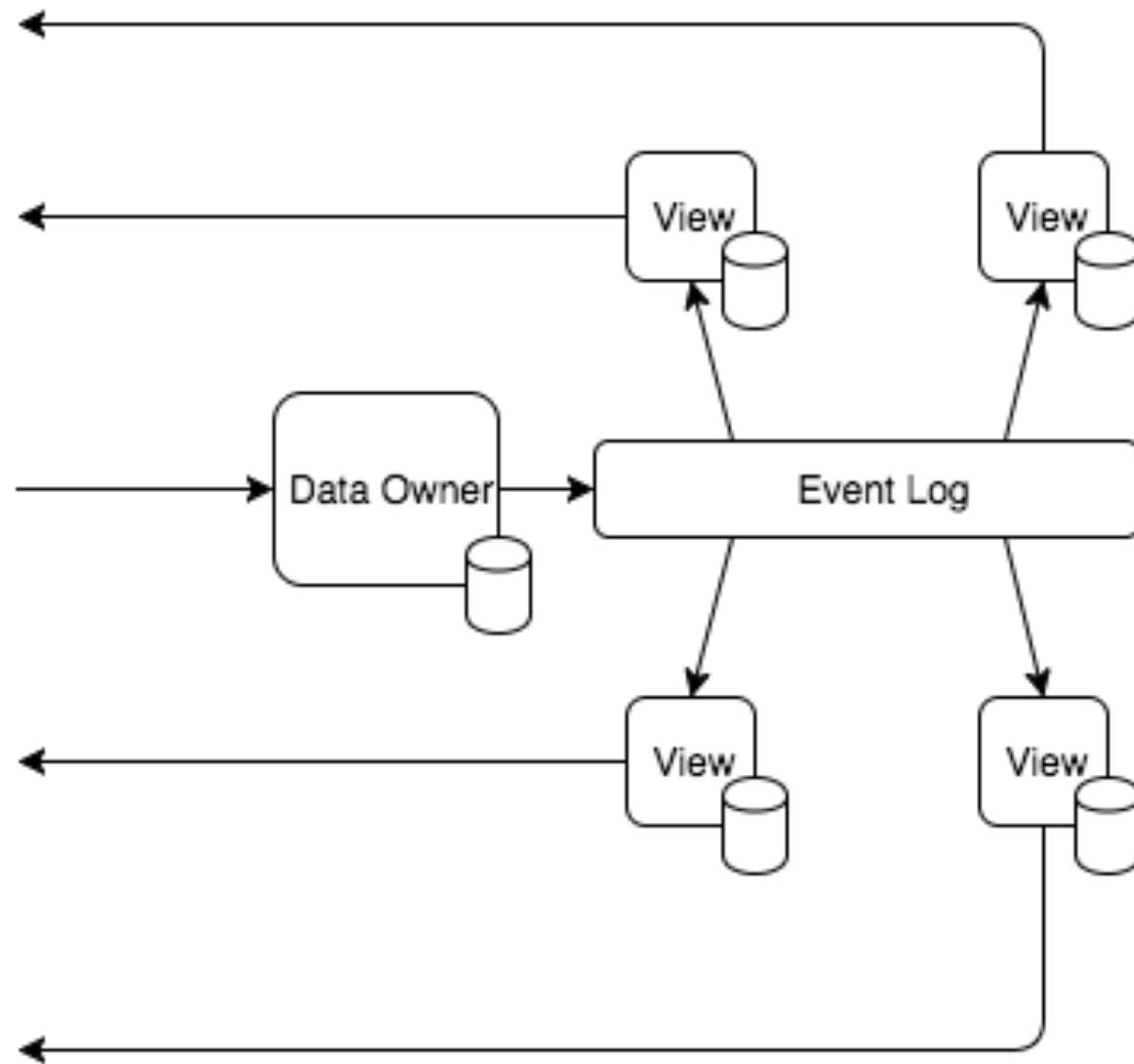
Distributed Monolith

Core Services Dependency Diagram

Last updated: May 16, 2013



Event Sourcing



Event Sourcing

- Less runtime dependencies (SLA, performance)
- Less chattiness (performance, velocity)
- Loose Coupling (velocity, SLA)
- Events / Not Commands – (velocity vs. 1-1 coupling)
- Cheap to setup/backfill new service (velocity)

Event Sourcing

- Less runtime dependencies (SLA, performance)
- Less chattiness (performance, velocity)
- Loose Coupling (velocity, SLA)
- Events / Not Commands – (velocity vs. 1-1 coupling)
- Cheap to setup/backfill new service (velocity)

Event Sourcing

- Less runtime dependencies (SLA, performance)
- Less chattiness (performance, velocity)
- Loose Coupling (velocity, SLA)
- Events / Not Commands – (velocity vs. 1-1 coupling)
- Cheap to setup/backfill new service (velocity)

Event Sourcing

- Less runtime dependencies (SLA, performance)
- Less chattiness (performance, velocity)
- Loose Coupling (velocity, SLA)
- **Events / Not Commands – (velocity vs. 1-1 coupling)**
- Cheap to setup/backfill new service (velocity)

Event Sourcing

- Less runtime dependencies (SLA, performance)
- Less chattiness (performance, velocity)
- Loose Coupling (velocity, SLA)
- Events / Not Commands – (velocity vs. 1-1 coupling)
- Cheap to setup/backfill new service (velocity)

Challenges

- We can't make it happen overnight
- There are a lot of risks:
 - Can this pattern deliver?
 - How long will it take to learn?
 - What stack to use?
 - Cost of tech onboarding?

Challenges

- We can't make it happen overnight
- There are a lot of risks:
 - Can this pattern deliver?
 - How long will it take to learn?
 - What stack to use?
 - Cost of tech onboarding?

Two challenges

- Validate Event Sourcing
- Choose and on-board appropriate tech stack

Ideally we can decouple the two, to:

- Validate early
- Deliver value early
- Invest in tech once idea validated

Two challenges

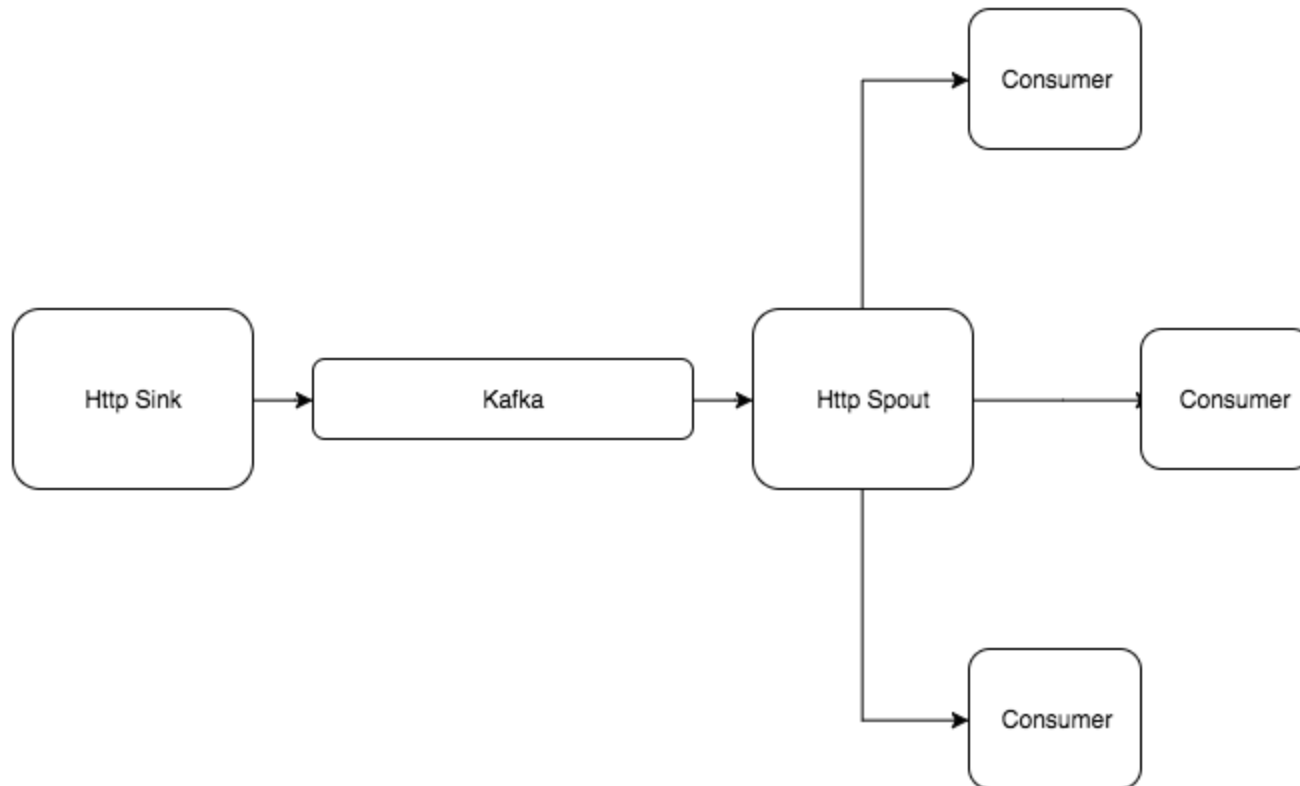
- Validate Event Sourcing
- Choose and on-board appropriate tech stack

Ideally we can decouple the two, to:

- Validate early
- Deliver value early
- Invest in tech once idea validated

Our approach

Leverage familiar legacy to minimize tech risk.



Our approach

- Not the greatest design:
 - centralization
 - http proxies, why?
- Built of familiar components we are already operating
- Minimizes tech risk, letting us focus on validation
- Short term, centralization allows for faster iteration

Our approach

- Not the greatest design:
 - centralization
 - http proxies, why?
- Built of familiar components we are already operating
- Minimizes tech risk, letting us focus on validation
- Short term, centralization allows for faster iteration

Our approach

- Not the greatest design:
 - centralization
 - http proxies, why?
- Built of familiar components we are already operating
- Minimizes tech risk, letting us focus on validation
- Short term, centralization allows for faster iteration

Our approach

- Not the greatest design:
 - centralization
 - http proxies, why?
- Built of familiar components we are already operating
- Minimizes tech risk, letting us focus on validation
- Short term, centralization allows for faster iteration

What did we build?

- We've established an API and semantics
- Tooling for consumer/event monitoring/management
- End-to-end test suite (focus on failure handling)
- Automated load and throughput tests

Will help us in the future, when we want to iterate on the tech stack.

What did we build?

- We've established an API and semantics
- Tooling for consumer/event monitoring/management
- End-to-end test suite (focus on failure handling)
- Automated load and throughput tests

Will help us in the future, when we want to iterate on the tech stack.

What did we build?

- We've established an API and semantics
- Tooling for consumer/event monitoring/management
- End-to-end test suite (focus on failure handling)
- Automated load and throughput tests

Will help us in the future, when we want to iterate on the tech stack.

What did we build?

- We've established an API and semantics
- Tooling for consumer/event monitoring/management
- End-to-end test suite (focus on failure handling)
- Automated load and throughput tests

Will help us in the future, when we want to iterate on the tech stack.

What did we build?

- We've established an API and semantics
- Tooling for consumer/event monitoring/management
- End-to-end test suite (focus on failure handling)
- Automated load and throughput test









Will help us in the future, when we want to iterate on the tech stack.

Easy to generate dashboards



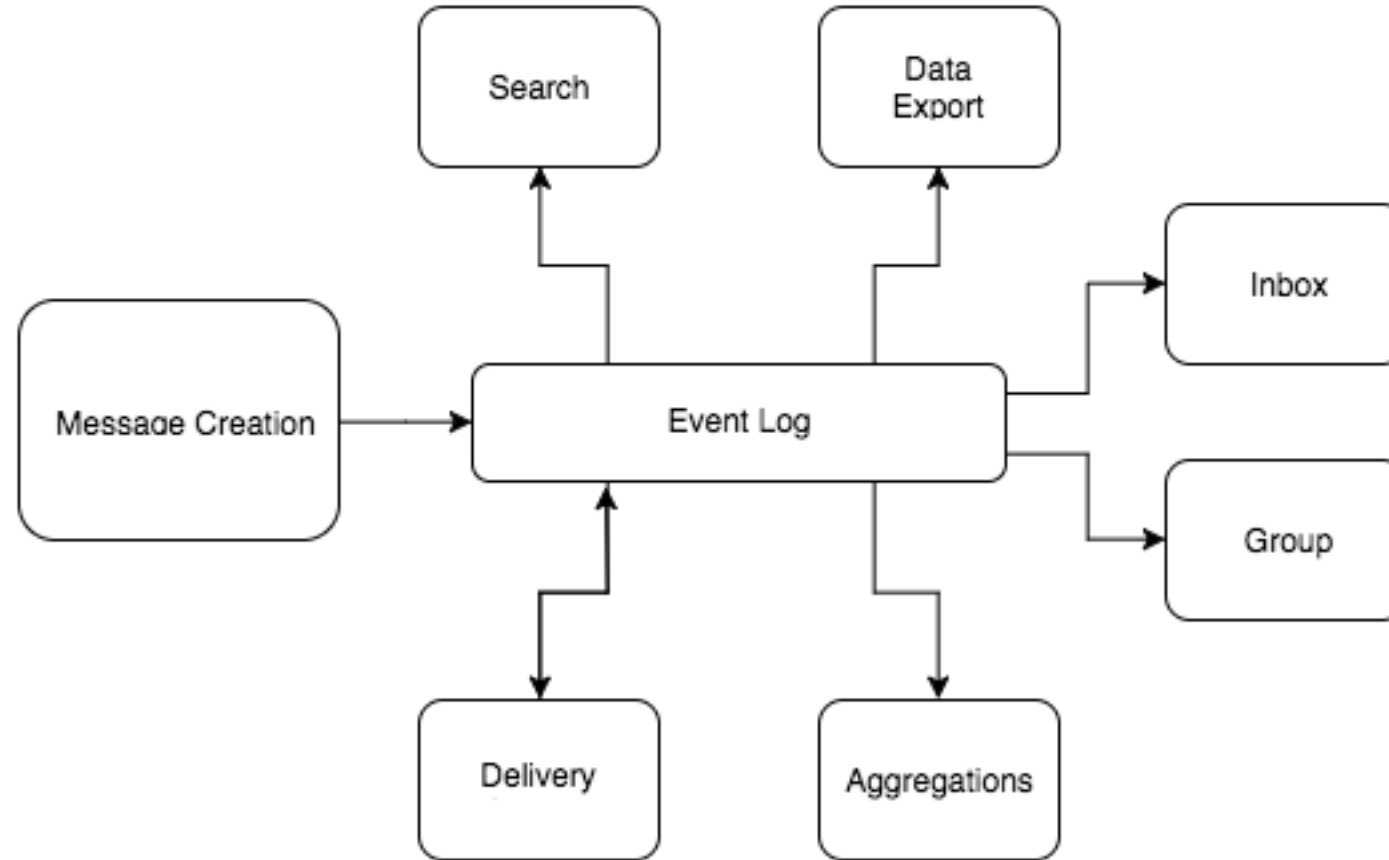
Troubleshooting bad events

All Failing **Stuck**

Consumer	Topic	Partition	Host	Read	Lag	Latency	Uncommitted	Retries	Actions
pulse_ogo_create	message	174	ventspout-1.az2.dm2	43107	0	6 hours	11	21741	   
stored_feeds_delivery_thread_and_group_delivered	message	60	ventspout-1.az1.dm2	53668	0	a few seconds	1	11	   

♥ from the SOA team

Projects that benefited immediately



Problems we've faced

- What to publish?
 - Whole pieces of data (potentially unbounded size)
 - IDs, but requires:
 - Immutable versioned data
 - Uniform Resource Identifiers (REST done well)
- There will be multiple publishers
 - Consumers need to deal with it gracefully

Problems we've faced

- What to publish?
 - Whole pieces of data (potentially unbounded size)
 - IDs, but requires:
 - Immutable versioned data
 - Uniform Resource Identifiers (REST done well)
- There will be multiple publishers
 - Consumers need to deal with it gracefully

Adoption Challenges

- It is a big paradigm shift, it takes time for knowledge to propagate through an organization
- We are not experts either, we are still learning
- But good news, even if imperfect, it already had big impact on how we work

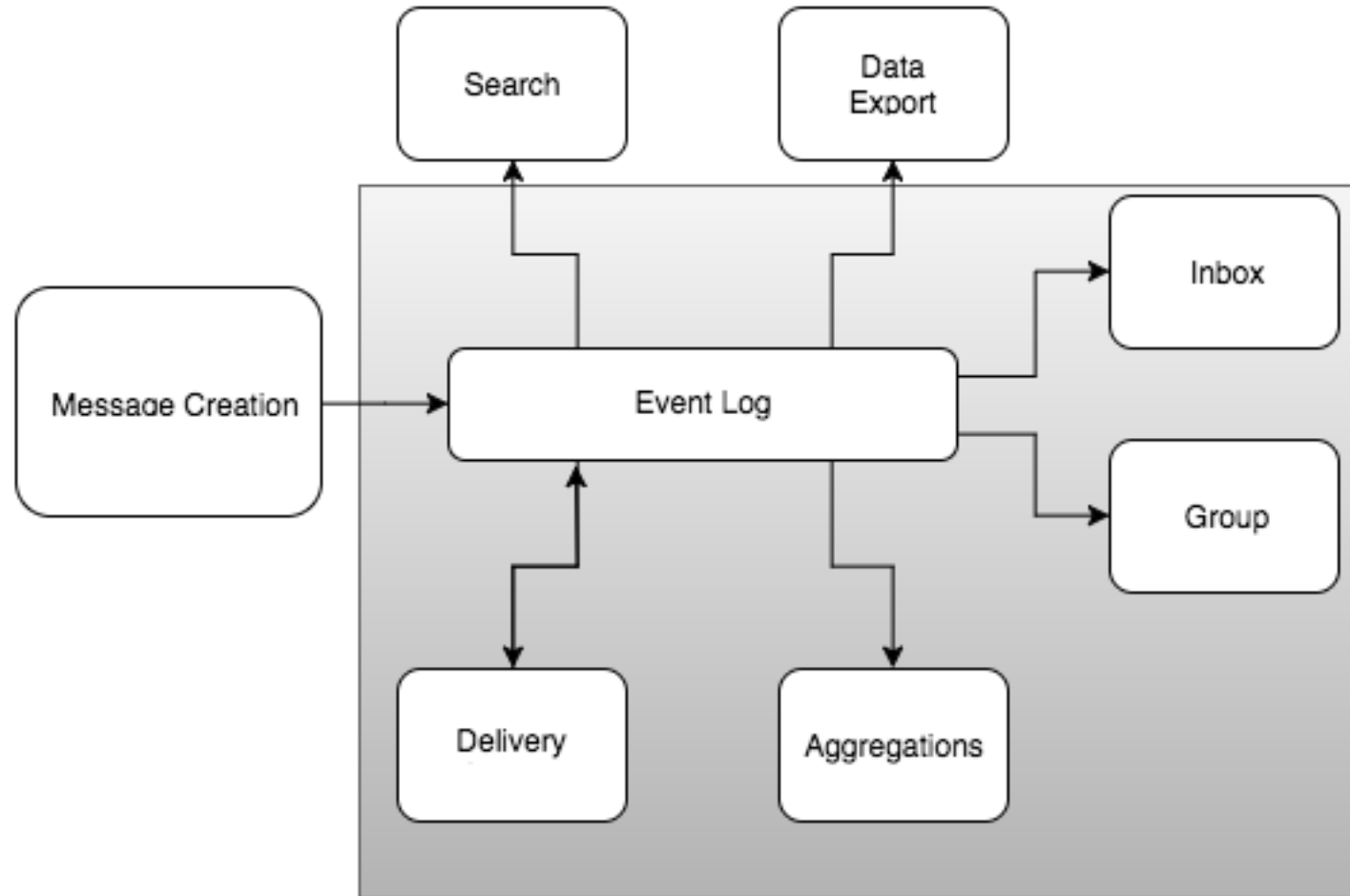
Adoption Challenges

- It is a big paradigm shift, it take time for knowledge to propagate through an organization
- We are not experts either, we are still learning
- But good news, even if imperfect, it already had big impact on how we work

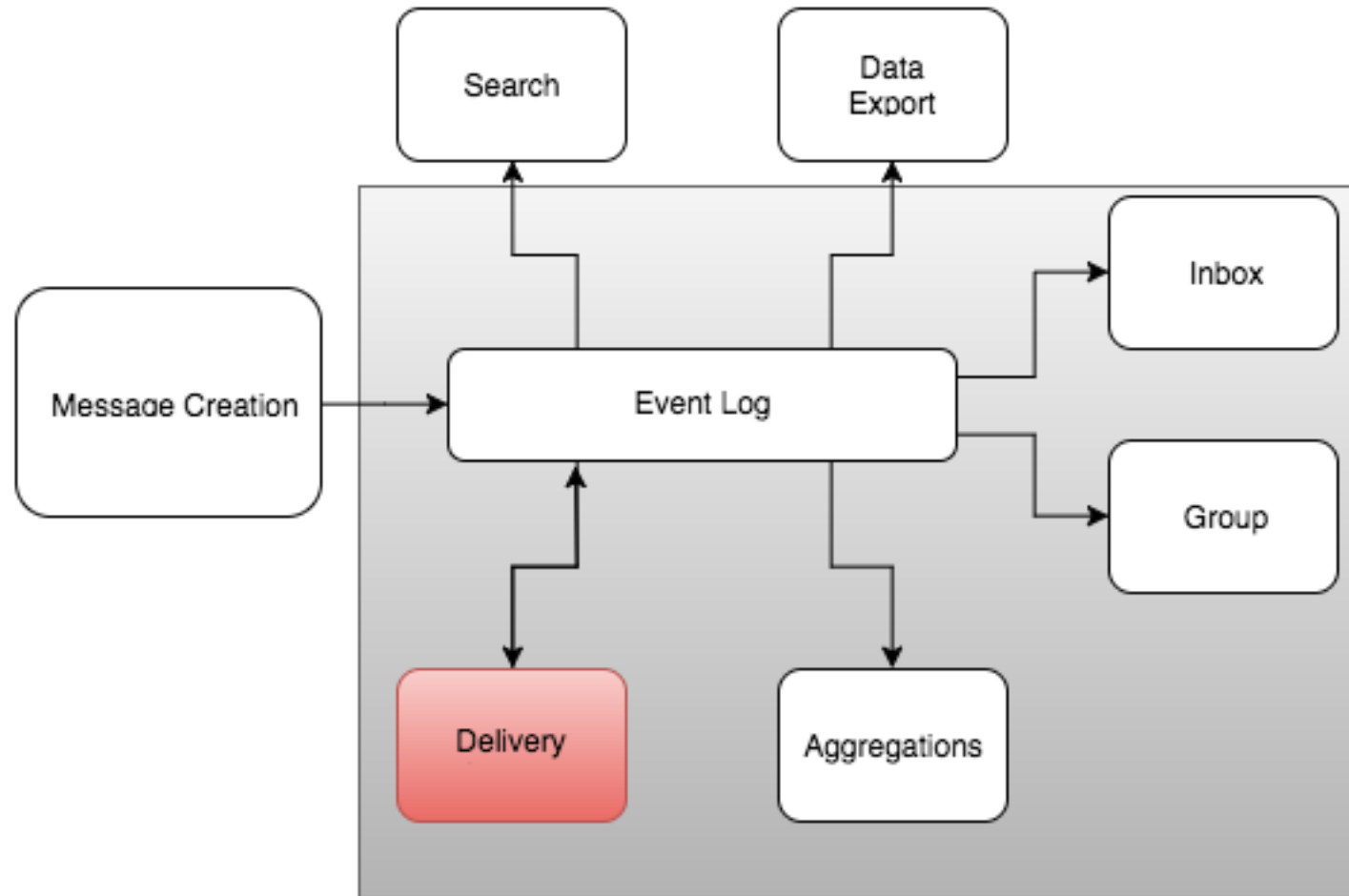
Adoption Challenges

- It is a big paradigm shift, it take time for knowledge to propagate through an organization
- We are not experts either, we are still learning
- But good news, even if imperfect, it already had big impact on how we work

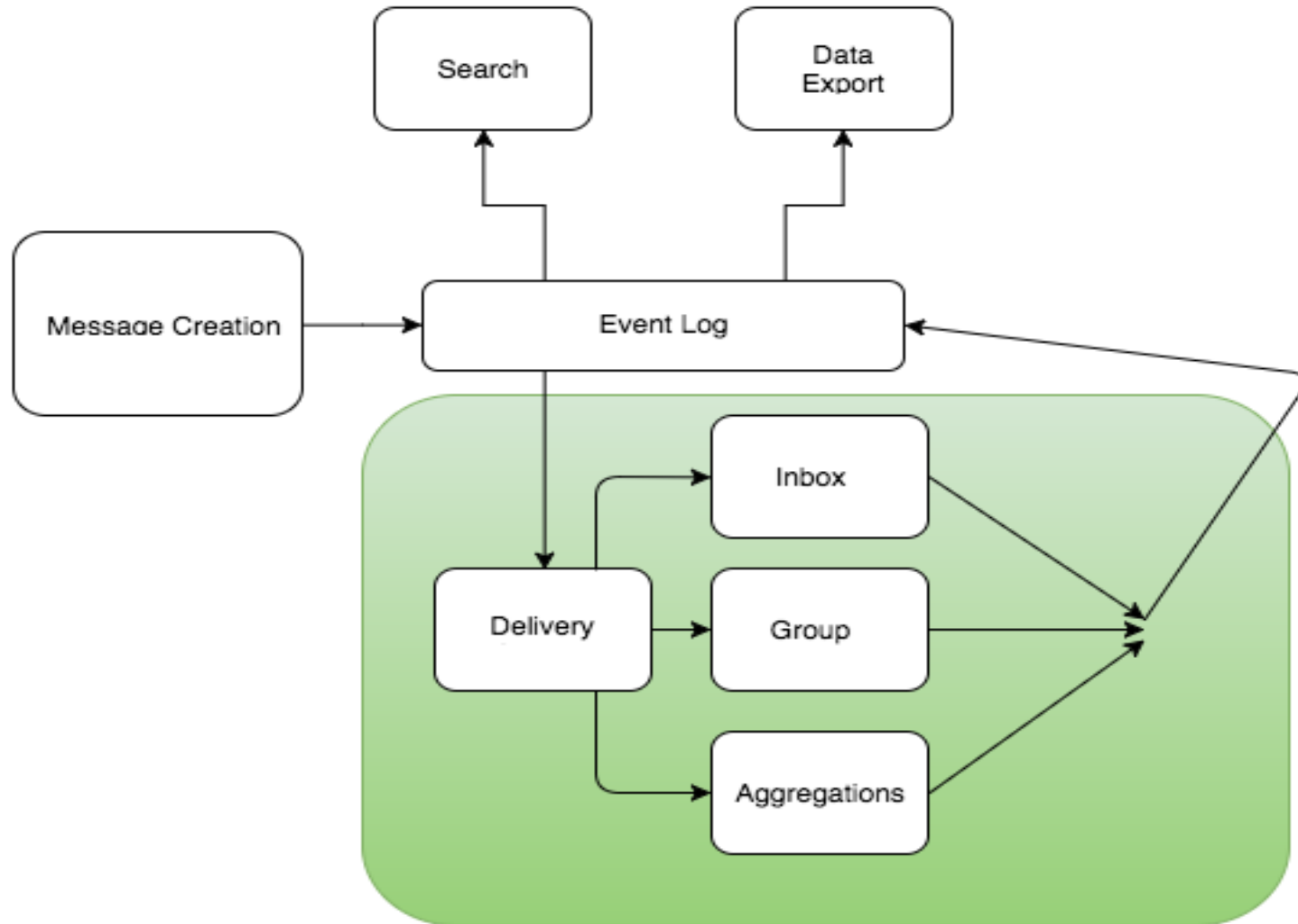
What we've discovered - Workflows



What we've discovered - Workflows



What we've discovered - Workflows



Workflows

- Transformation logic outside of view service boundaries
- Stream Processing / CQRS
- We can express them, but too much plumbing
- We need a higher level of abstraction

Workflows

- Transformation logic outside of view service boundaries
- **Stream Processing / CQRS**
- We can express them, but too much plumbing
- We need a higher level of abstraction

Workflows

- Transformation logic outside of view service boundaries
- Stream Processing / CQRS
- We can express them, but too much plumbing
- We need a higher level of abstraction

Workflows

- Transformation logic outside of view service boundaries
- Stream Processing / CQRS
- We can express them, but too much plumbing
- We need a higher level of abstraction

What is the Future?

- Move to a fully managed solution
- Provide Rx bindings
- Remove centralization/http proxy components
- Find a solution for Workflows

Migration To Event Hubs

Azure Event Hubs

- Event Log offering from Azure (Kafka, Kinesis)
 - Automated failover within a region
 - No provisioning concerns, simply purchase TUs
- AMQP 1.0 protocol
- Successfully used internally and externally
- Our Metrics pipeline already uses it

Migration To Event Hubs

Azure Event Hubs

- Event Log offering from Azure (Kafka, Kinesis)
 - Automated failover within a region
 - No provisioning concerns, simply purchase TUs
- AMQP 1.0 protocol
- Successfully used internally and externally
- Our Metrics pipeline already uses it

Migration To Event Hubs

Azure Event Hubs

- Event Log offering from Azure (Kafka, Kinesis)
 - Automated failover within a region
 - No provisioning concerns, simply purchase TUs
- AMQP 1.0 protocol
- **Successfully used internally and externally**
- Our Metrics pipeline already uses it

Migration To Event Hubs

Azure Event Hubs

- Event Log offering from Azure (Kafka, Kinesis)
 - Automated failover within a region
 - No provisioning concerns, simply purchase TUs
- AMQP 1.0 protocol
- Successfully used internally and externally
- Our Metrics pipeline already uses it

RxJava SDK

- Using Azure SDK
 - backed by ProtonJ
 - Offset tracking
 - Multi-host Consumer with Failover
- Raising the level of abstraction:
 - Data Stream in Event Hubs available as an Observable

RxJava SDK

- Using Azure SDK
 - backed by ProtonJ
 - Offset tracking
 - Multi-host Consumer with Failover
- Raising the level of abstraction:
 - Data Stream in Event Hubs available as an Observable

Workflows

- Azure Service Fabric Reliable Actors
- High Level PAAS offering from Azure
- Based on Project Orleans
- Successfully used by HALO

Workflows

- Azure Service Fabric Reliable Actors
- High Level PAAS offering from Azure
- Based on Project Orleans
- Successfully used by HALO

Workflows

- Azure Service Fabric Reliable Actors
- High Level PAAS offering from Azure
- Based on Project Orleans
- Successfully used by HALO

Workflows

- Azure Service Fabric Reliable Actors
- High Level PAAS offering from Azure
- Based on Project Orleans
- **Successfully used by HALO**

Summary

- Successfully used Event Sourcing to solve our SLA/Velocity problems caused by a (distributed) monolith
- It addressed both architectural and org aspects
- We did so in an iterative fashion focusing
 - Reducing risk
 - Delivering early
- This is ongoing work

Summary

- Successfully used Event Sourcing to solve our SLA/Velocity problems caused by a (distributed) monolith
- It addressed both architectural and org aspects
- We did so in an iterative fashion focusing
 - Reducing risk
 - Delivering early
- This is ongoing work

Summary

- Successfully used Event Sourcing to solve our SLA/Velocity problems caused by a (distributed) monolith
- It addressed both architectural and org aspects
- We did so in an iterative fashion focusing
 - Reducing risk
 - Delivering early
- This is ongoing work

Summary

- Successfully used Event Sourcing to solve our SLA/Velocity problems caused by a (distributed) monolith
- It addressed both architectural and org aspects
- We did so in an iterative fashion focusing
 - Reducing risk
 - Delivering early
- This is ongoing work

Thank you!

Any Questions?