

Event driven architecture & hyper-growth @Lunar

EventSourcing Live 2021

By Thomas Bøgh Fangel & Brian Nielsen

Story

"You can't predict the future, but you can plan for it"
- Lunar Tech

- A. Why event source in Lunar?
- B. Who are Thomas and Brian ?
- C. What is Lunar today ?
- D. In what context do we use Event sourcing in Lunar and what has it brought us?
- E. Take aways



\$ whois



Brian Nielsen

Lead Architect at Lunar

 [@Briannielsen76](#)

Married to Marianne & together we have 2 boys Christian and Gustav

- 1 Years Experience in Lunar
- 6 Years of Fintec experience as Lead Architect in Mobilepay (5.5 mio MAU)
- 17 Years of experience in the Banking domain across Engineering
- Classically trained engineer in: PL/1, Java and C#
- 7 years experience with DDD and event driven architecture
-



Thomas Bøgh Fangel

Senior Software Engineer

 [@tbfangel](#)

Married to Anne, father of 3, owner of a dog and a cat

- 5 years of experience in Lunar
- 15+ years experience in building distributed software systems
-
-
-
-

VISION

YOUR LIFE IN ONE FINANCIAL SUPER APP

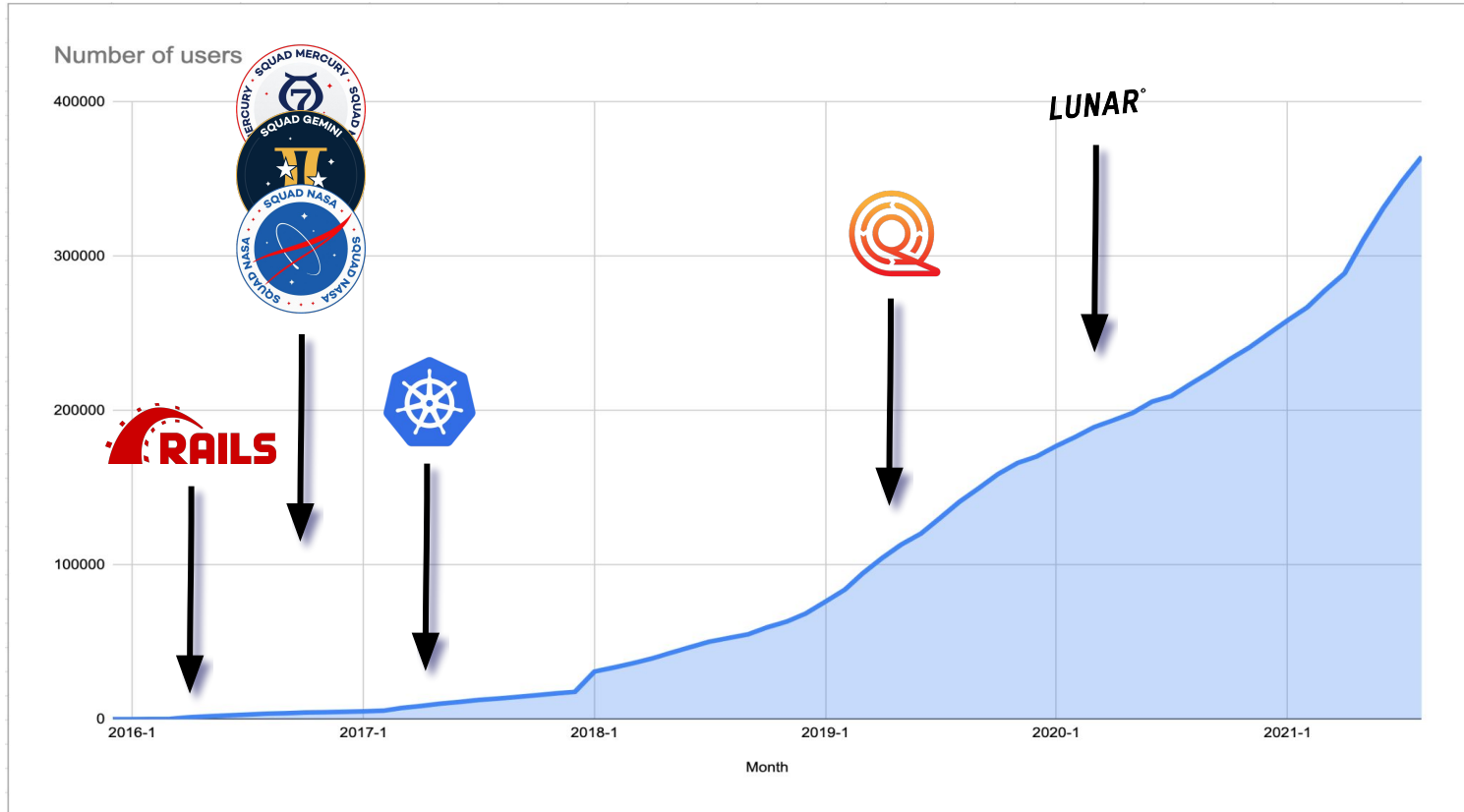
Make the most of your money when you save up, spend smarter, grow your business, invest your money and engage in personalised banking that matches your lifestyle.

It takes a bank unlike anyone else. A bank that morphs banking and entertainment for esports fans. Or empower you to clean plastic from the ocean every time you swipe your card.

Lunar is this bank, and we are building the first financial super app to shape the future of banking in the Nordics. We redefine the most profitable banking market in world.

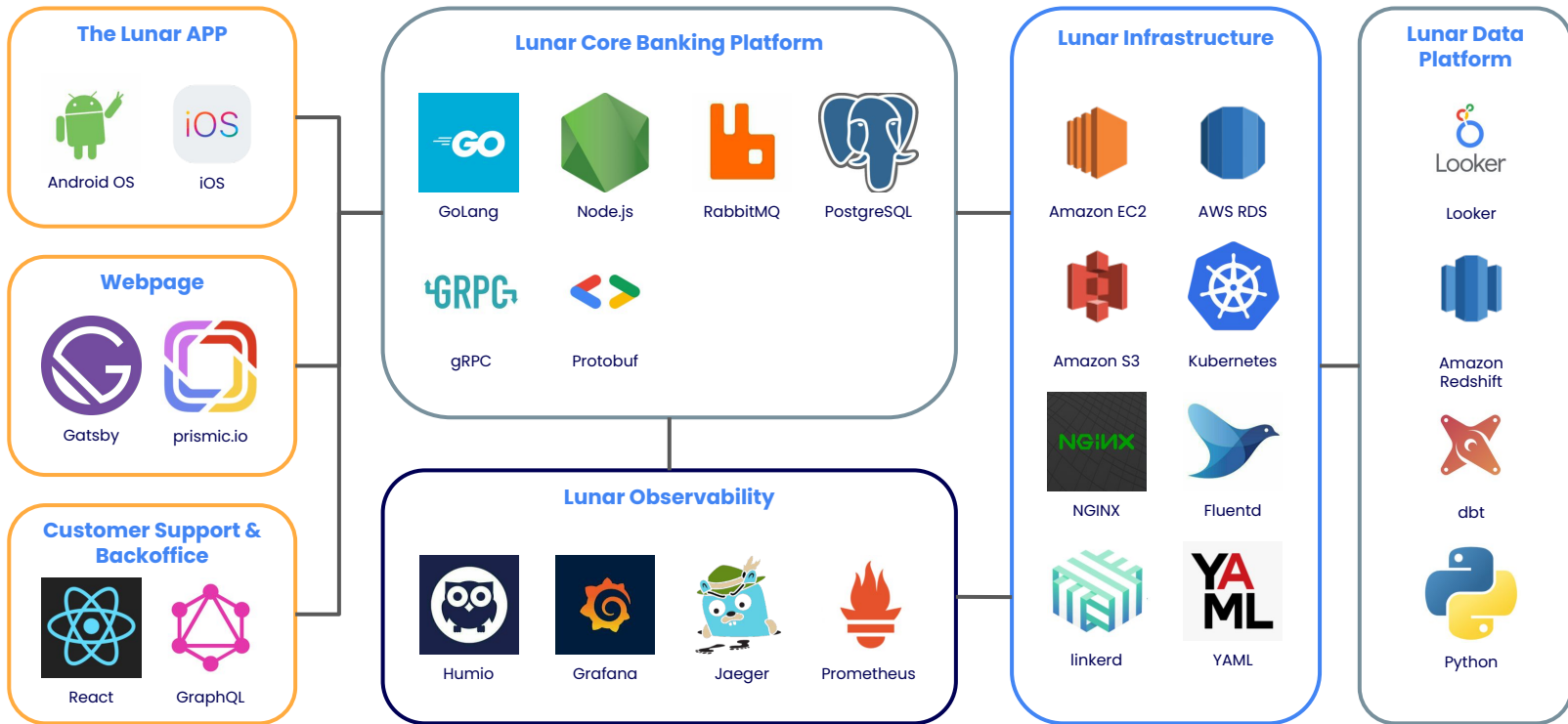


From 0 to 400k users...



Cloud Native, AWS, Kubernetes, Go, Event Driven, Event Sourced, Domain Driven

Welcome to – The Lunar Tech Stack



Lunar organisation

growth over years



17 Squads & 200+ microservices 480+Repos



Observation:

Increasing user base -> more out layer cases

More Countries / Currencies

More complex products: Credit / Loan

More Channels: App / Web

Geographically distributed squads

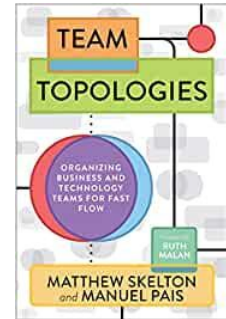
Principle suggestion:

Simplify with “Single” responsibility :

Domain and subdomain segregation

Set user focus





(Squads may need to span more domains from the get go)




Multi talented group of people with specific business and domain focus
'Squads on a mission'

Squad Artemis

4 fundamental topologies

-  Stream-aligned team
-  Enabling team
-  Complicated Subsystem team
-  Platform team

 Team Topologies

PROCESS

Week

Quarte

SUCCESS

application

to approval)

Automated backoffice flow

Multiple hubs to collaborate on a shared vision

Organized around hubs – domain & Squads (inverse Conway)



Lending
Credit engine



Banking Operations
Reporting
Data



Consumer
Banking Operations
Data
+
Invest
Business
Cloud and platform



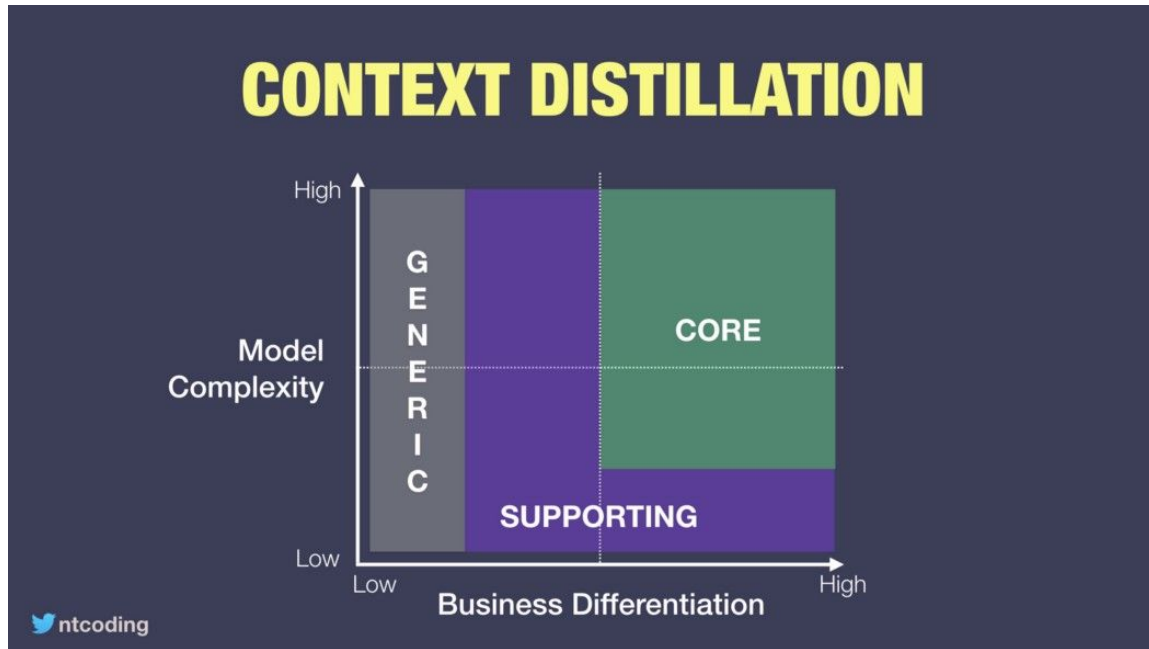
Domain design

Bounded context classification

“Time and resources are limited. How we spend our time and apply our resources when developing software systems is possibly the most fundamental and difficult challenge. Of all the things we could be doing, what should we do and how much quality and rigour should we invest?” --



Nick Tune

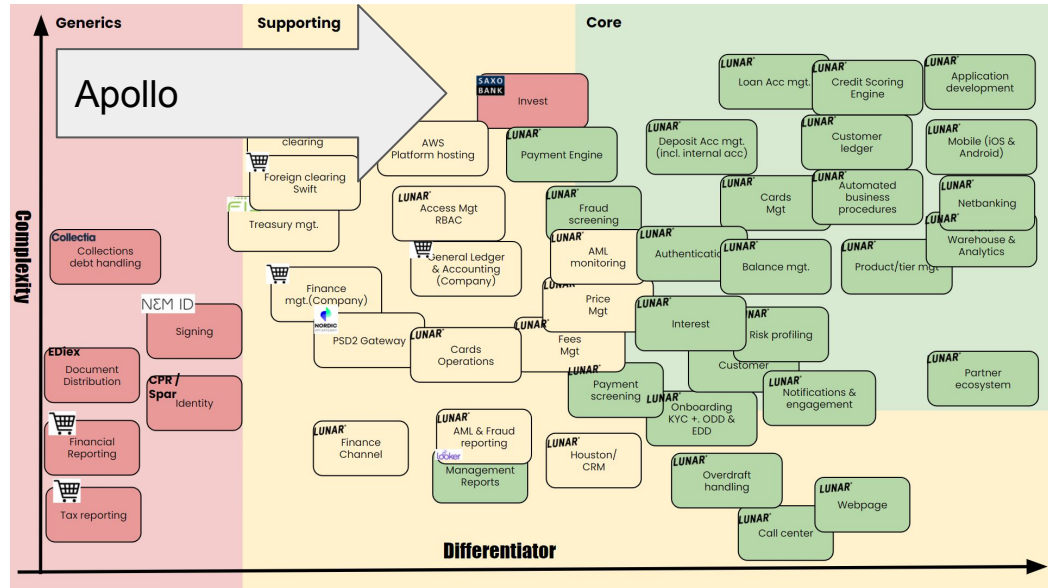


Domain design

Domain distillation for the entire Lunar domain back in 2020

We want to outsource Classical Banking components like
Clearing
Treasury
SWIFT
Financial reporting

And we actually value user journeys for our invest products high

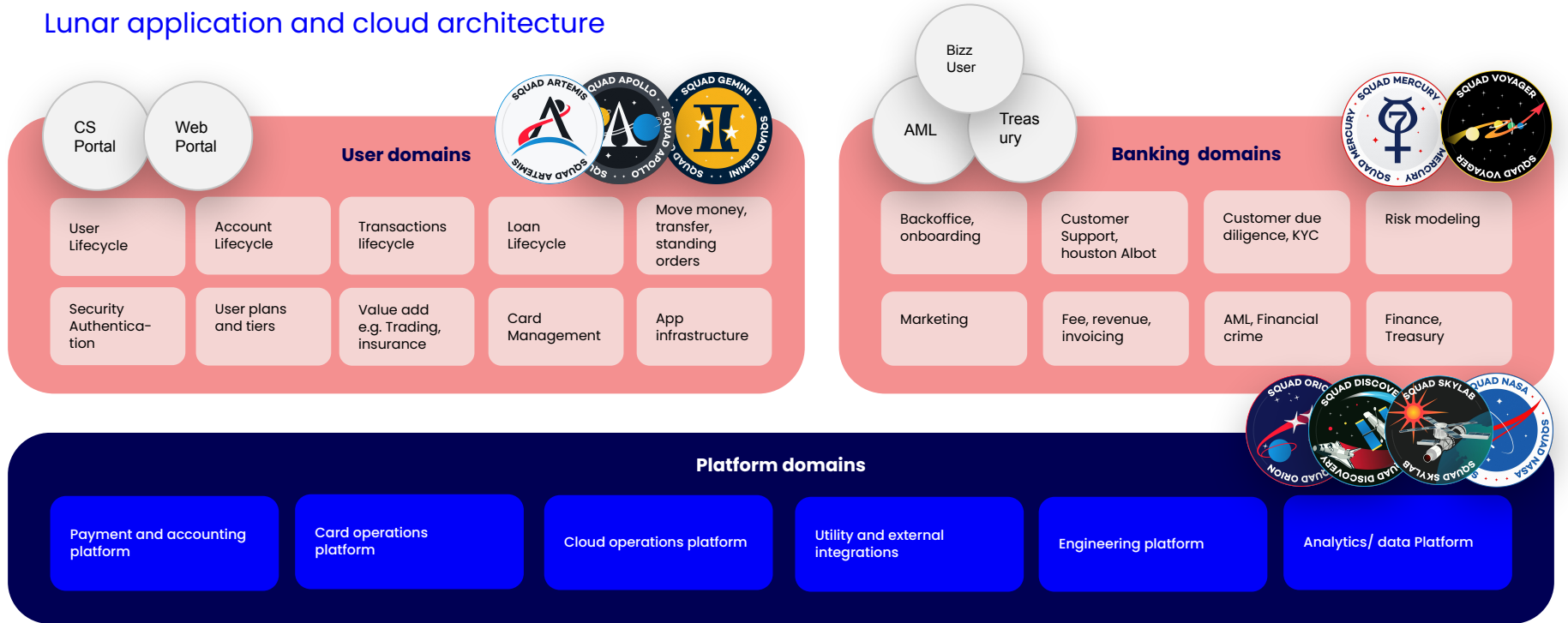


Yes Banking is complex hence the map is very busy - sorry



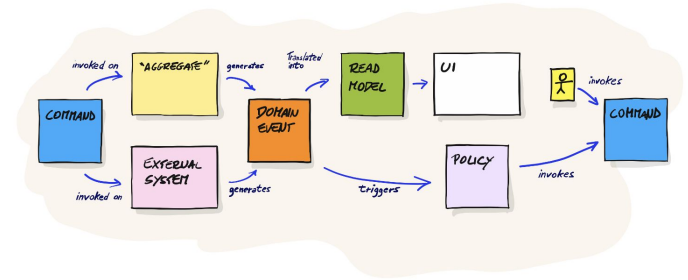
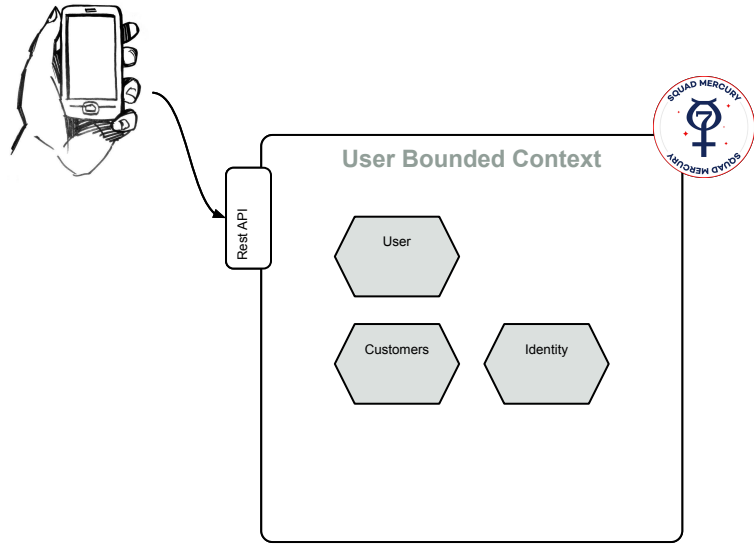
Designed to decouple, simplify and promote efficiency and scalability

Lunar application and cloud architecture



Scale up and Scale out

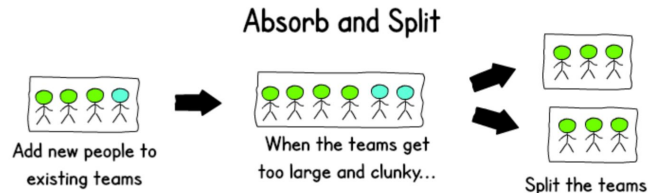
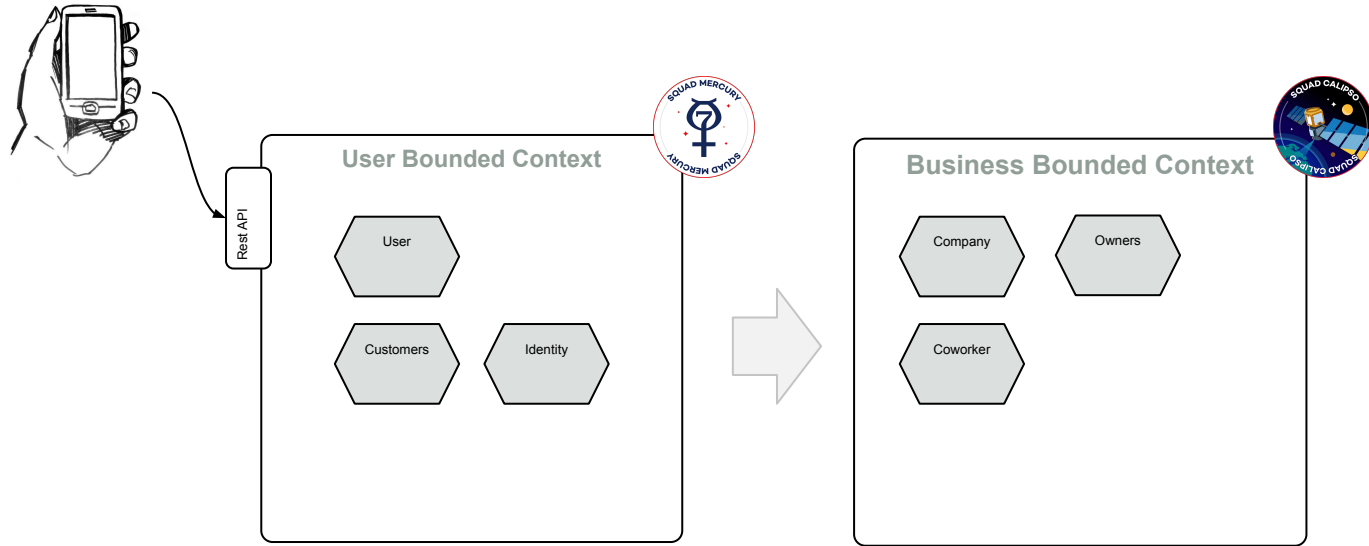
Focus on boundaries & Language



- 1) Build up vocabulary
- 2) Focus on internal and external language to identify boundaries
- 3) Focus on your integration events / API's to help other squads and increase fast flow of knowledge

Scale up and Scale out

Focus on boundaries & Language



learnings

ES was born in Lunar with a primary focus on Audit and because engineers found it fun. In all lecture managing an account is a classic example of eventsourcing.

Idempotency is important because when we have not transactional boundrie across domains
We have experienced that unique identifiers like transactionID was not unique

question Is the "account" aggregate correctly designed ?

We experienced that as our customers made many payment transactions - generating the aggregate state in mem became a problem. Snapshotting to the rescue

Issue Side effects was not consistently executed:

ex . Readmodels for seperate queries must be consistently updated together with the domain aggregate: "updating balance when receiving a payment command"

Guaranteed event handling Observer pattern to the rescue.

System architecture

Challenge of autonomous squads & microservices

Org



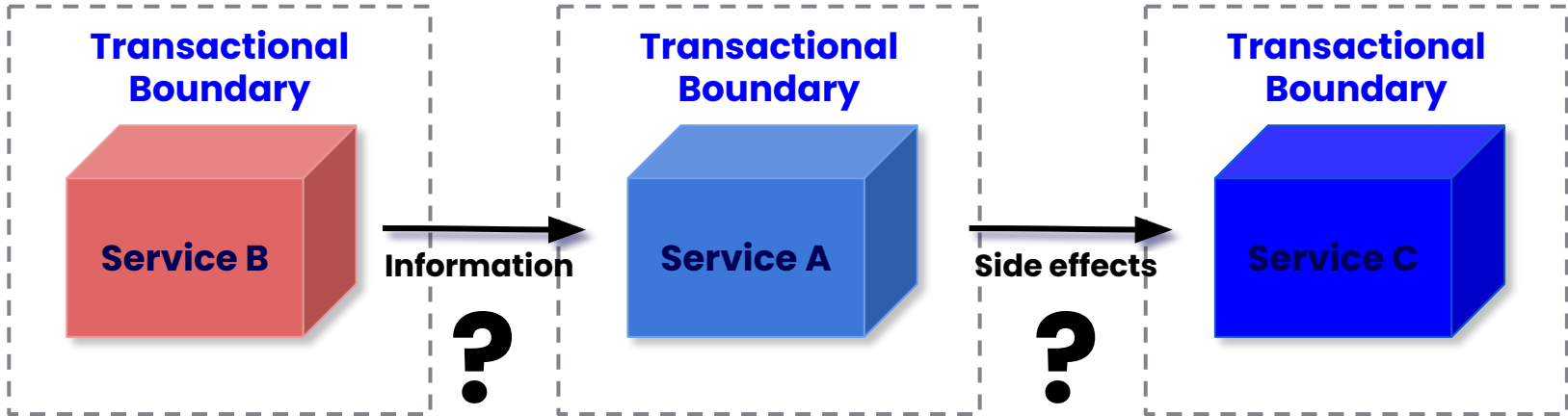
Independent,
but share
Information
consistently



Decoupled,
but cause
changes with
guarantees

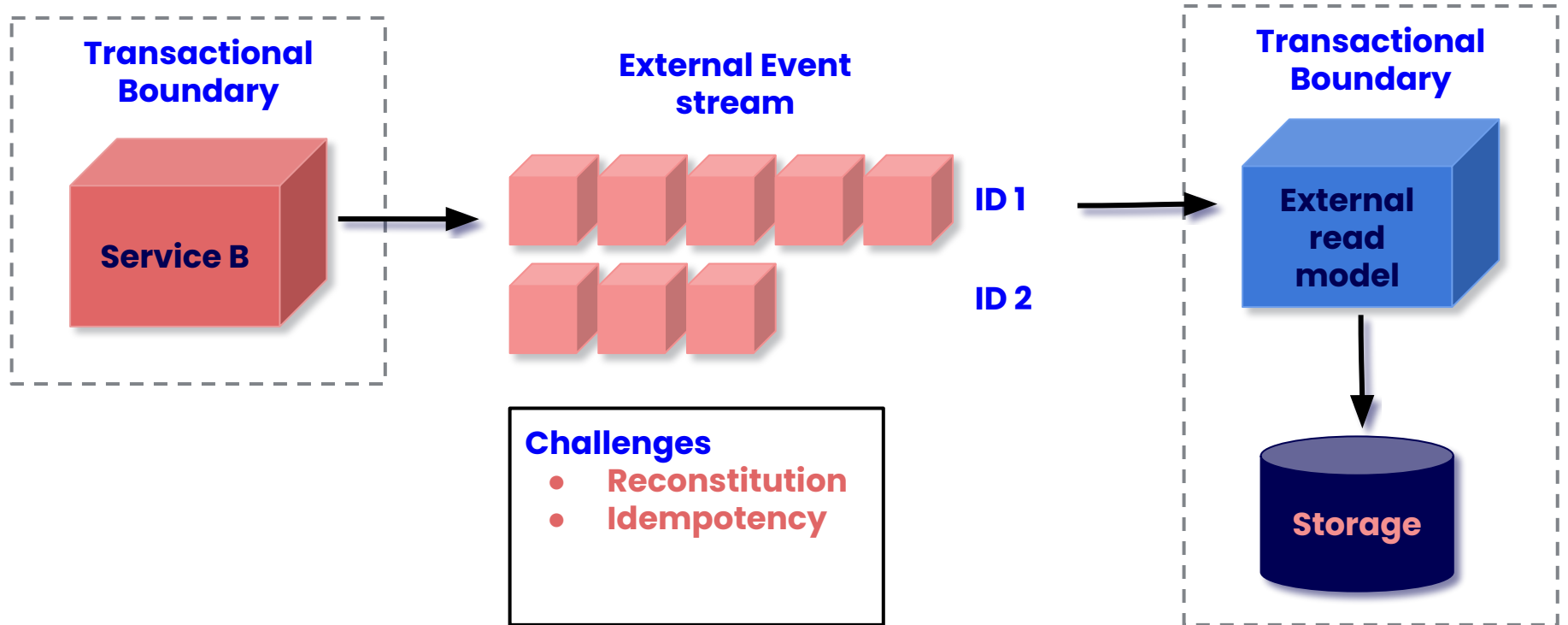


Tech



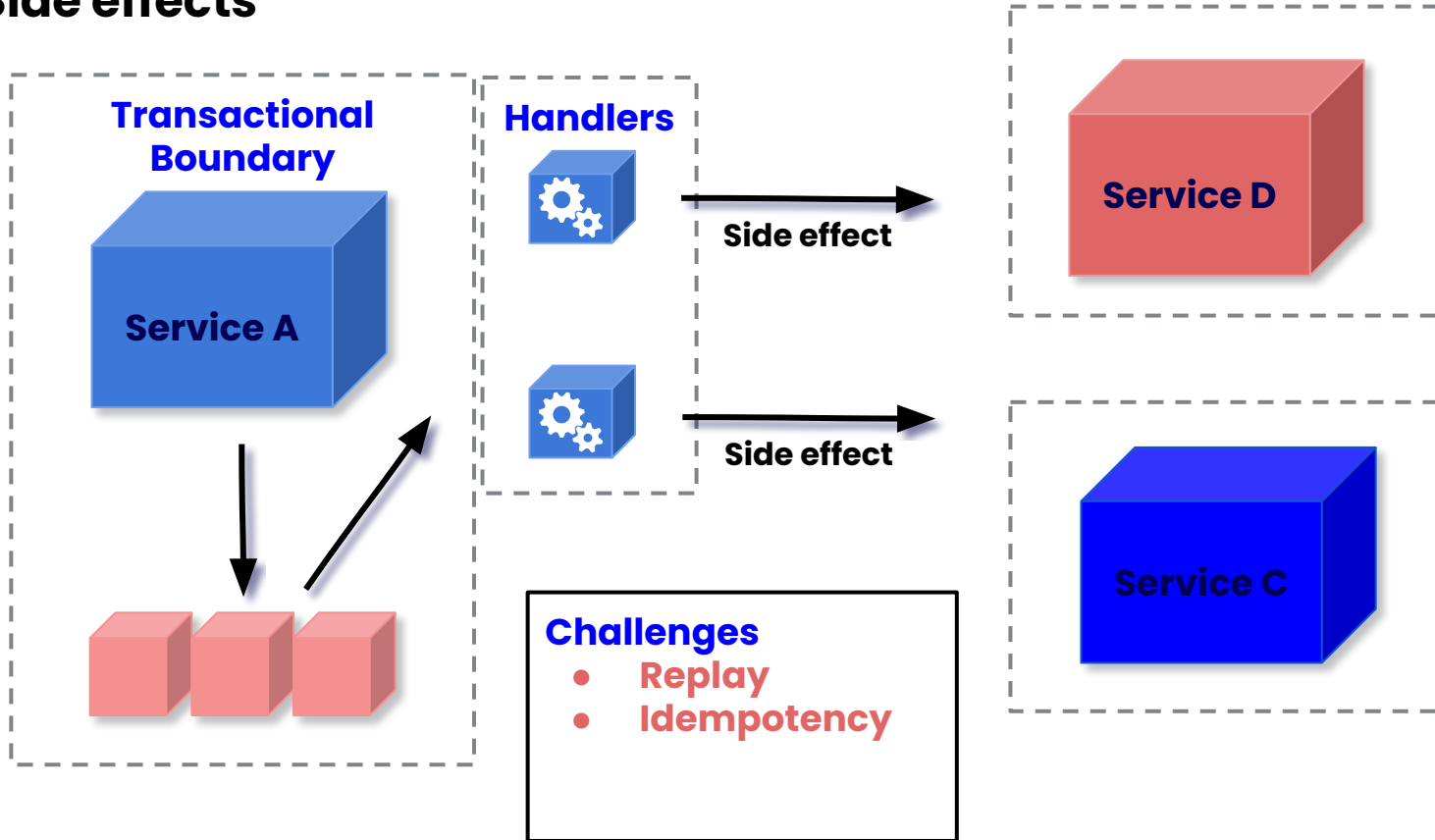
System architecture

Event Streams on the Outside



System architecture

Side effects



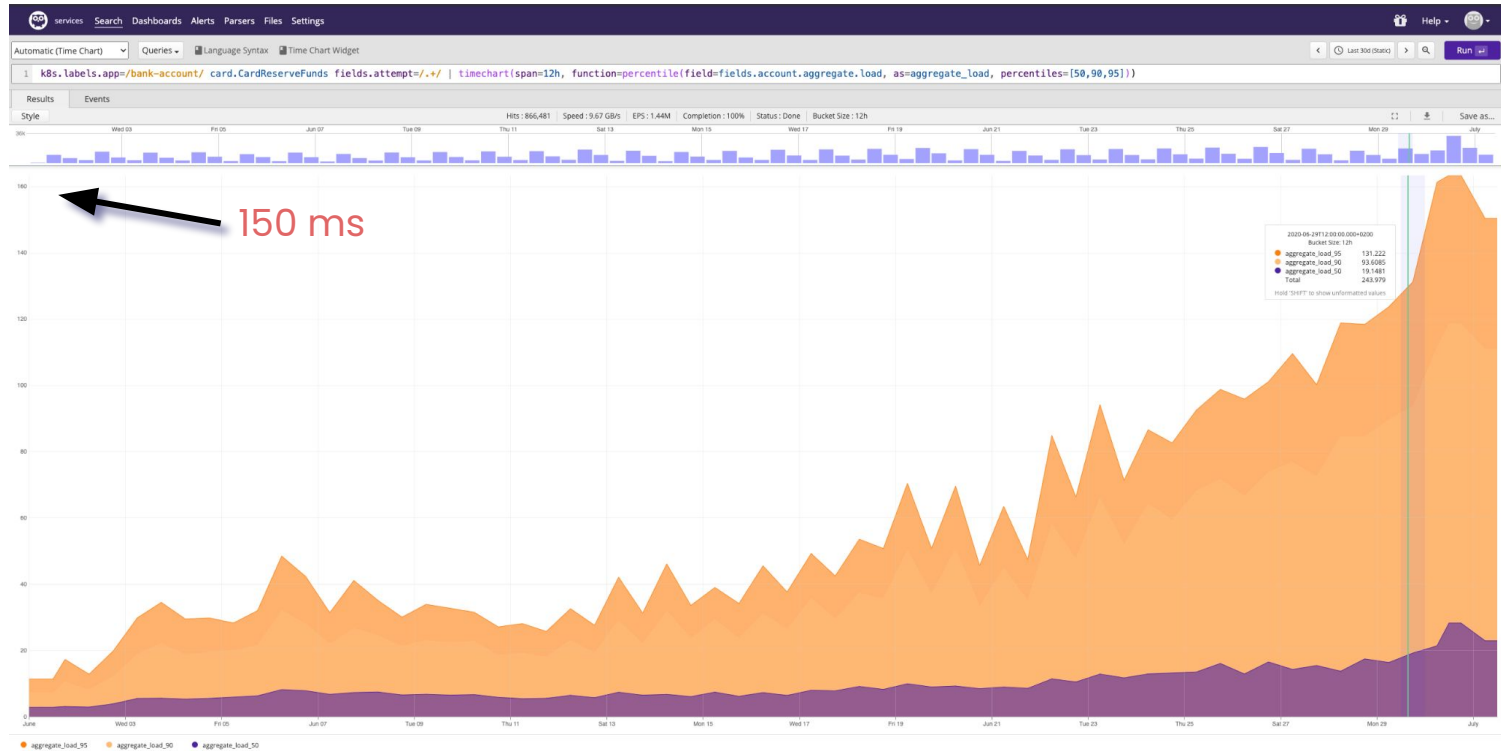
Event Sourcing Learnings & Challenges

Use cases

- Short running aggregates: payment sagas, process managers
 - lifecycle measured in days
- Long running aggregates: user, account
 - lifecycle measured in months, years

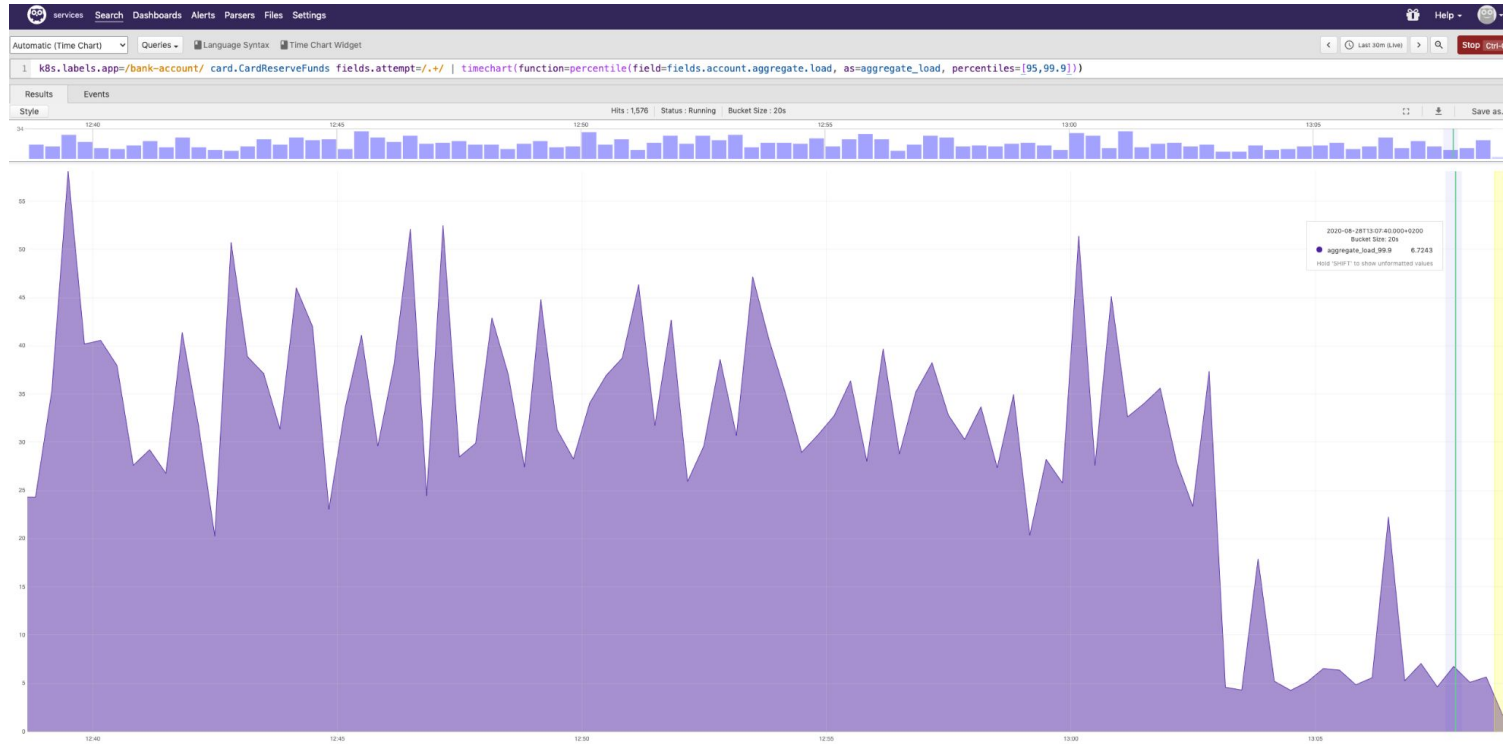
System architecture

Event Sourcing Learnings & Challenges



System architecture

Event Sourcing Learnings & Challenges

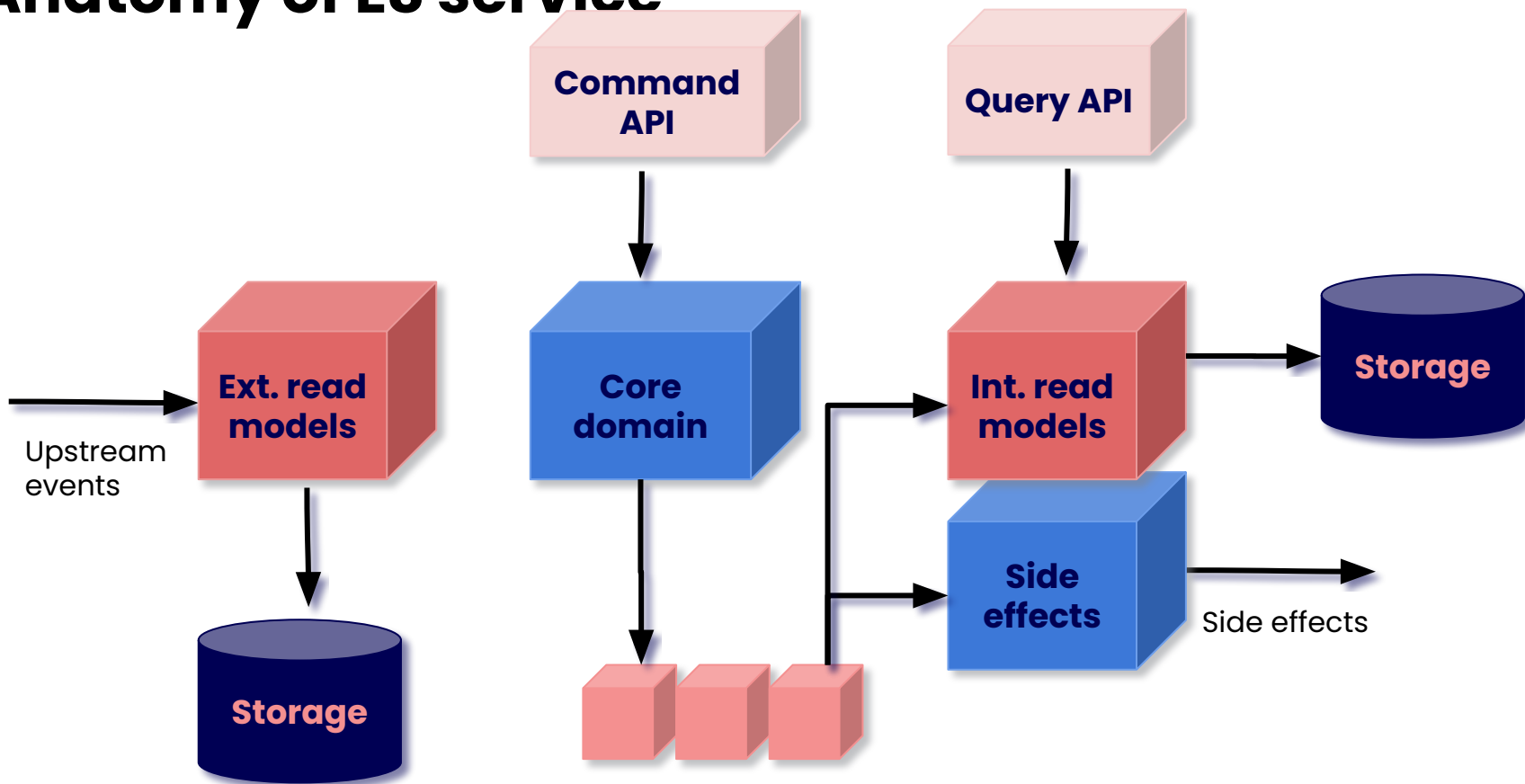


Event Sourcing Learnings & Challenges

Learnings

- Expect surprises
- Just-in-time-development is OK
- Prioritize observability

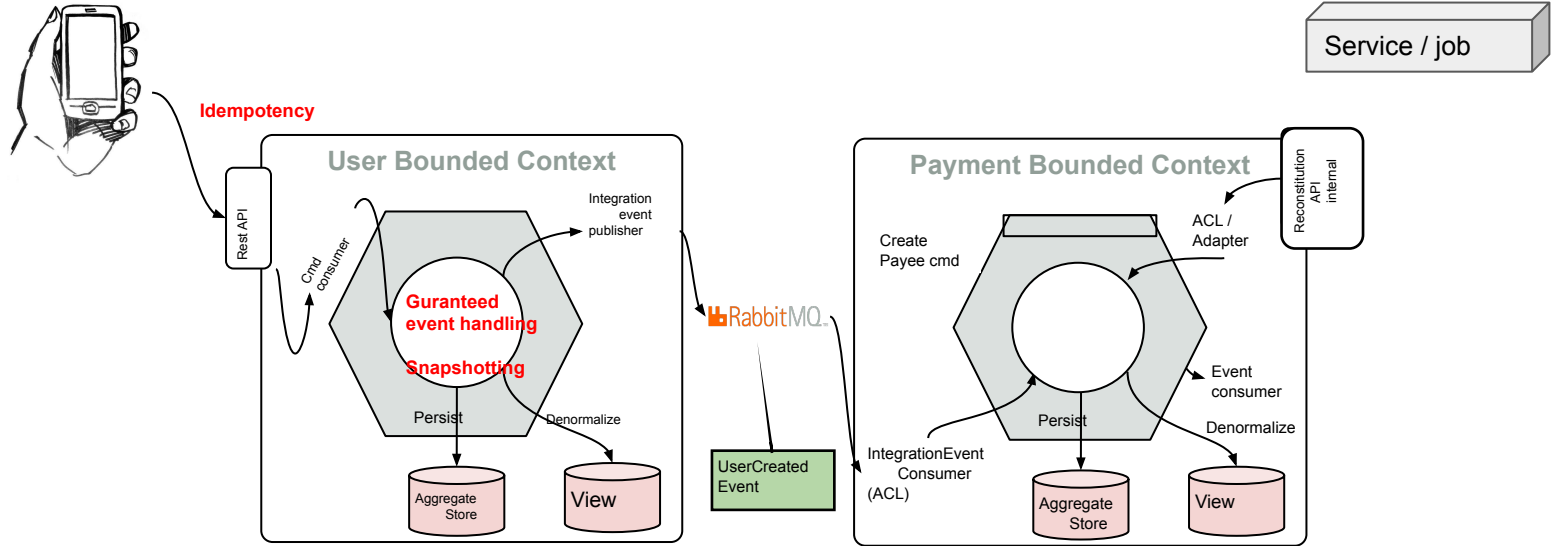
Anatomy of ES service



Sumup

Distributed and event driven Architecture considerations

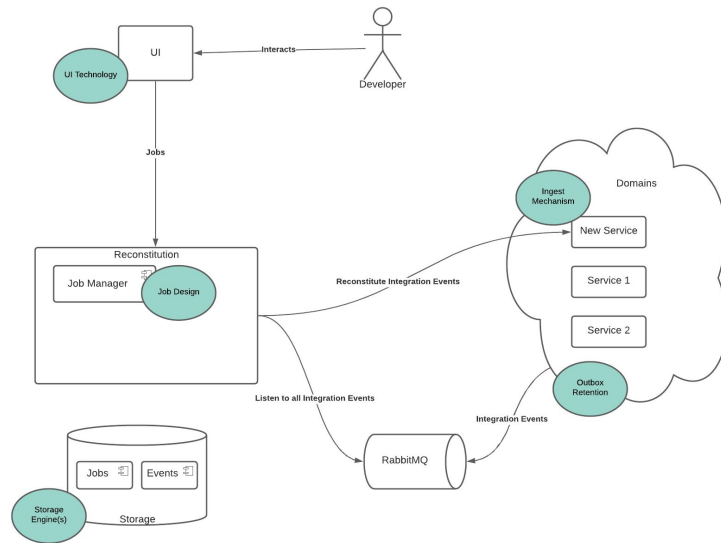
Reconciliation jobs = eventual consistency Across Domain



Sagas
Error Queues
Service discoverability



Guaranteed event delivery
Persist I.E
+ I.E Data



Make domain expansion easy Instrument through Backstage.io

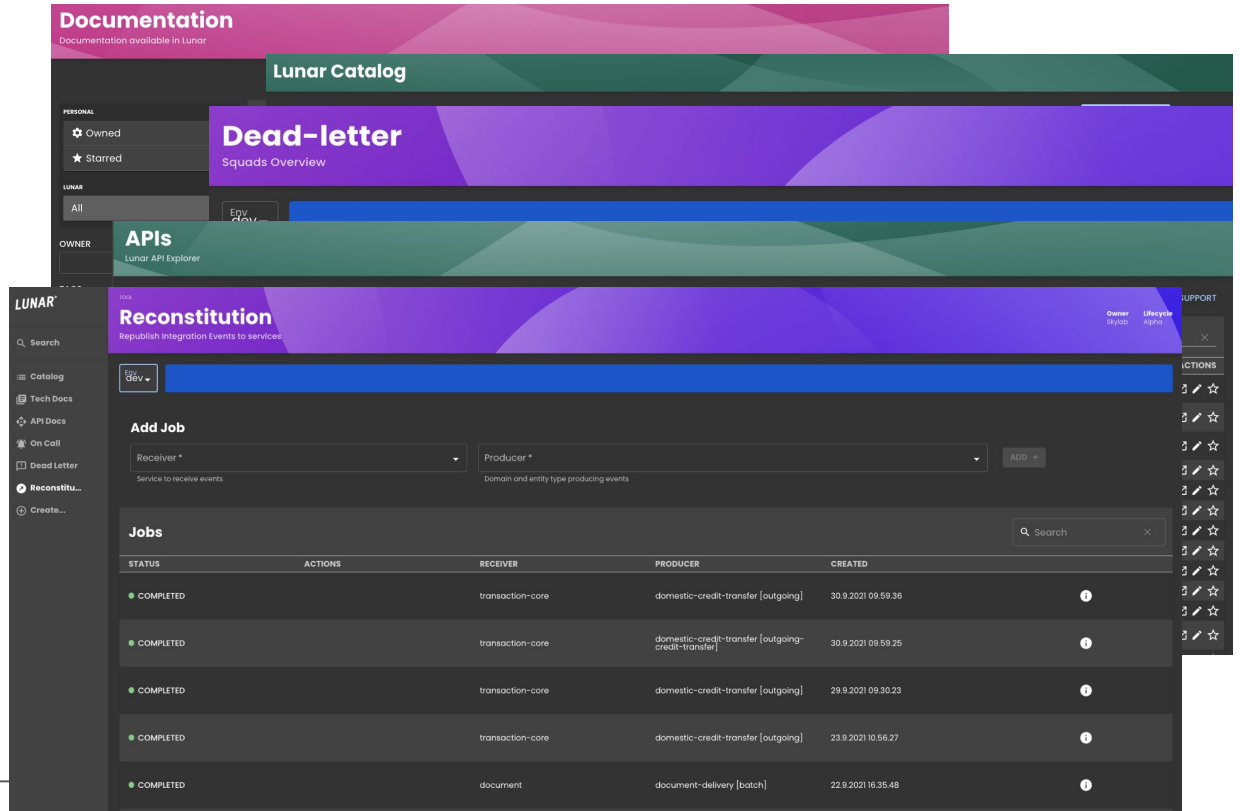
Best practise

Service Overview

Deadletter / error Queue

API Overview

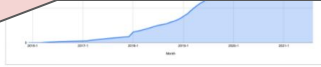
Replay / reconstitute



Event sourcing in Lunar

Key take away

“You can’t predict the future, but you can plan for it”
- Lunar Tech

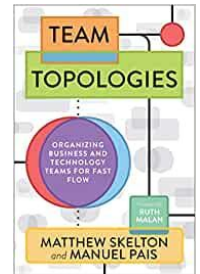


This is how we have implemented Eventsourcing in Lunar with Event streams.

Eventsourcing alone is one technique among several that fit well into an event driven and domain driven architecture

In Lunar vi have experienced rapid growth not only in Systems and Technology but just as well in our organisation and we plan for further growth

Along the way we have learned a lot and done mistakes and hard learnings - But we are certain the foundation of **event sourcing, event and domain driven is the correct path for us**



LUNAR[®]