Yet another event sourcing library

Robert Pofuk
/about-me

- Raiffeisen Bank International AG
  - Software Developer/DevOps/Architect at
  - Clojure developer
    - EventSourcing, CQRS, REST, and CI/CD pipeline, integrating testing, business analysis
- Co-Founder of AlloraIT

- https://github.com/alpha-prosoft/edd-core
- https://github.com/alpha-prosoft/edd-core-web

- https://github.com/rpofuk
- https://github.com/raiffeisenbankinternational
Agenda

- History
- Serverless and AWS lambda
- CQRS
- Event Sourcing
- Architecture
- Why open source?
- Conclusion
History

- Started project in 2019
  - Wanted to use Serverless
  - Wanted to utilize managed infrastructure as much as possible
- We also wanted to keep business logic vendor independent
  - We want to be able to move (It is also regulatory requirements to be movable)
- Simple API
  - No discussion about path, query parameters, headers….
- Data must be open
  - No hidden binary stored messages in queues, or custom formats
Serverless and AWS lambda

- Startup problem
  - Each lambda instance is new VM
  - JAVA takes time to start (Clojure much more)
- Most of libs did not compile to GraalVM
  - Created AWS SDK
    - Whatever AWS does in SDK in the end is just POST request
    - Very high inconsistency in signing
  - Forked hikaricp
  - Lots of exceptions for LogBack
CQRS

- Must see video
  - https://www.youtube.com/watch?v=qDNPQo9UmJA
- CQRS stands for Command and Query Responsibility Segregation
- **Our implementation:**
  - Commands
    - POST /commands
    - Commands-queue
    - Commands s3 bucket
  - Query
    - GET /query
  - Frontend client implementation is simple ~300 lines of code
    - Frontend mocking
    - Simple infrastructure
Event Sourcing

- Event Sourcing is a pattern for storing data as events in an append-only log
  - Store events that happened
  - Requires lots of resources for storage and processing
- Lots of critical systems requires proving current state
  - Plenty of audit log implementations
  - In event sourcing this comes naturally
- **Our implementation:**
  - PostgreSQL
    - Just simple table with metadata and one JSONB field
  - We collect all events per aggregate to be sequential (Optimistic Locking)
    - Unique constraint on event_sequence and aggregate_id
  - One command updates one aggregate
Why Open Source?

● As internal-only library:
  ○ Internal implementation was not meant as library
    ■ Very specific requirements
  ○ No open sourcing process existed before in company
  ○ Not alternative implementations
    ■ Only Postgres (event-store) and OpenSearch (view-store)

● As open source project:
  ○ Hobby projects
    ■ Using it for side projects
    ■ Implemented DynamoDB support
    ■ Alternative implementations help form proper abstractions and improvements
  ○ Contribute back all improvements to internal implementation
    ■ Lots of tests and quality improvements (i.e. integration tests and compatibility tests)
Conclusion

- Was very positive experience to deliver this solution to production
  - On daily basis it pays off that we store everything (request_log, command_store)
- Super easy to recover data
  - Unprocessed messages (DLQ) because of business error, or data quality is super easy
- Business is happy
  - System is eventually consistent but we can also eventually check if it actually is