AARHUS UNIVERSITET

Microservices and DevOps

Scalable Microservices

Event Sourcing

Henrik Bærbak Christensen

AARHUS UNIVERSITET

Motivation

- Many (most?) large IT systems we use, we use to **store** information
  - Permanently or transiently

- Data Integrity:
  - **Data integrity** refers to maintaining and assuring the accuracy and consistency of data over its entire life-cycle, and is a critical aspect to the design, implementation and usage of any system which stores, processes, or retrieves data. [Wikipedia]

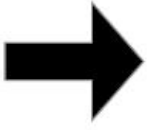
Antipattern

- **Dropbox**
- In our housing cooperative (andelsboligforening) we use dropbox to store our resumes from meetings
  - Board meetings, common meetings, general assembly, …
- Benefits
  - All have instant access to full history
- Liabilities
  - While I was writing these slides, Lars began deleting old resumes on his home computer
    - Thanks for the ‘restore’ facility!

Another example

- A regular RDB
  - Too many courses teach students to store *state*
- *Events* are handled by *state changes*
  - Event: Sally moves to New York
  - Update in the Sally tuple

Person	Location
Sally	Philadelphia
Bob	Chicago



Person	Location
Sally	New York
Bob	Chicago

- Liability in this approach?

Figures from Marz's GOTO 2012 slides

Nathan Marz

- Definition of Data System

> A system that manages the **storage** and **querying** of data with a lifetime measured in **years** encompassing every **version** of the application to ever exist, every **hardware failure**, and every **human mistake** ever made

AARHUS UNIVERSITET

Human fault-tolerance

- Bugs will be deployed to production over the lifetime of a data system
- Operational mistakes will be made
- Humans are part of the overall system, just like your hard disks, CPUs, memory, and software
- Must design for human error like you'd design for any other fault

- Nygard: ”Oh-no-second”

Good or bad

- The bad

The worst consequence is data loss or data corruption

- The good

As long as an error doesn't lose or corrupt good data, you can fix what went wrong

CRUD

Mutability

- The U and D in CRUD
- A mutable system updates the current state of the world
- Mutable systems inherently lack human fault-tolerance
- Easy to corrupt or lose data

Immutability

- An immutable system captures a historical record of events
- Each **event** happens at **a particular time** and is **always true**

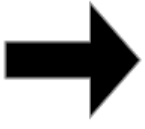
AARHUS UNIVERSITET

‘Event Sourcing’ Pattern

- **Capture events as events**
  - **State is computed from the history of events**

> Event Sourcing ensures that all changes to application state are stored as a sequence of events. Not just can we query these events, we can also use the event log to reconstruct past states, and as a foundation to automatically adjust the state to cope with retroactive changes.

Person	Location	Time
Sally	Philadelphia	1318358351
Bob	Chicago	1327928370

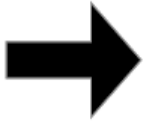


Person	Location	Time
Sally	Philadelphia	1318358351
Bob	Chicago	1327928370
Sally	New York	1338469380

Exercise

- Consider how to repair

Person	Location	Time
Sally	Philadelphia	1318358351
Bob	Chicago	1327928370



Person	Location	Time
Sally	Philadelphia	1318358351
Bob	Chicago	1327928370
Sally	New York	1338469380

  - Human error: I typed Aarhus instead of New York
  - Program error: Alg. chopped last character of location
  - Hw error: out-of-disk during tuple insert

AARHUS UNIVERSITET

Examples

- Telemedicine – New Danish standard format
  - PHMR: *Clinical* ***document***
    - Stores all information about a given **event**
      - Like a single measurement of blood pressure
    - Error – insert new event of type ‘correction’ in store

- Karibu in EcoSense
  - Stores the raw measurement events
    - In a next-to-impossible-to-query format
    - But raw data integrity is guaranties

AARHUS UNIVERSITET

Discussion

- *This is simply too slow!*
  - To draw a blood pressure graph for Nancy you have to fetch, de-XML and combine 30 huge documents!
- Yep. But how to solve this?
- *This consumes much more disk space!*
- Yep. *There is no free lunch…*

AARHUS UNIVERSITET

Outlook

- Marz: *Lambda-architectures*
  - *Run **batch** processing for historic events to produce fast query databases*
  - *Combine with **live** processing of current events (like last 24 hours) to augment the above query database*

AARHUS UNIVERSITET

Summary

- To ensure *data integrity* (reliable data in face of all types of errors) of a *data system* (one that stores data over large time spans), you should

- Store *events* and not *state*
  - Overwritten state is lost state
  - Events can be corrected – by new events
  - Old events are not lost
- Performance penalty can be mitigated by deriving state and store it separately…