Evolution of Information Systems

How Information Systems come to be, evolve and what can be done about it (other than rest and rpc) ...

In the beginning, there was Monolith

- Most systems start as a single team, single code base, single deployable artifact
- This tends to change (pacing varies)
  - eg. fast, slow or steady change does happen
- Entropy and big ball of mud

The Fragments

What usually becomes of these systems?

- Things get out of hand
- Remedies(soas appear) -> different apps and services
- Entropy and big ball of mud **Again but with network included**
- Misconceived notion that modularizing software over the network will somehow improve its sustainability

The Fragments (continued)

- Systems become far more about people than they are about software
  - Organization, Software, and Data
- Ever growing issues with team sizes, ownership and business requirements
- Conways law kicks in hard
- What worked on smaller scale doesn't work any more

The Leftovers

The undying legacy

- The central / core db source of truth (aka God Services)
- Management interfaces eg. Backoffice / Resources which enable the enterprise / platform
- Everything else tying into it with ever increasing web of dependencies (runtime and compile time)

The Slowdown

Issues that arise with this kind of approach

- New teams forming requiring for ever more autonomy (the larger the company the more autonomy is needed)
- Total independence is impossible
- Balance must be struck
- Balancing between Organization, Software and Data as they evolve becomes the key (they are interdependent)

Microproblems

Why microservices don't (usually) work

- Works well if separations are clear
- They are mostly not
- Requirements mostly cross cut multiple domains / services
- A lot of synchorinization is necessarry

Microproblems (continued)

- Data sits at the heart of the issue (access to shared datasets but remain loosely coupled)
- Service interfaces provide tight points of coupling (contract, funciton and data) aggregation etc ...
- Systems are temporaly coupled (issues we had past weeks come to mind)

Issues at Hand

God service symptom at TS: **Masterdata API**
Other examples from experience ...

Commonly used (enabler) resources (points of coupling):

- Settings (all different kinds and varieties)
- Products and Professions
- Insurer information and Customer information
- Broker / Account information and Agency Numbers
- etc ...

Issues at Hand (continued)

- Many times its not clear who owns what piece of data
- Unclear who should expose something or where
- Core not going any time soon all backoffice is in there - reluctance to upgrade php bcs it's considered legacy
- Hard to build overviews / query models eg. consultation (aggregation)
- Many abstraction layers + network to expose simple things
- Temporal coupling - cascading failures

Issues at Hand (continued)

- Coordination with multiple teams (to expose data) and enable business flows
- Need more flexible way of evolving organization, software and data with a clear direction
- Moving data out of the core mostly implies connecting to gv24 which is really subotimal or building a totally new service which is an investment
- Integration with Data

Intermezzo

Diagrams and stuff

An Alternative

- Event Driven Architecture (Streams and Messages)
- Often refered to as DB Inside-Out (Coined by Martin K.)
- Make "data on the outside" a first class citizen
- CDC
- Expose resources as streams of data (ie. changes - data and even schema) - great way to handle legacy systems
- Systems become temporaly decoupled

An Alternative (continued)

- Provides a strategy that enables organization, software and data in a natural way
- Introduce new systems in a plug and play fashion, get rid of them even easier
- Build query models and query services in a straightforward fashion
- Services become secondary concerns (most of the time)

How

- CDC as intermediary / migration step for resources (can be permanent)
- Start simple (low scale eg. dev + prod)
- Start with a couple / few streams introduce more incrementally
- MQ and/or Streams for Business Events but Streams would be real enablers

Implication

> Instead of what new service we need to build, who will build it, what it's dependencies and contracts are, what architecture - those become secondary and quesiton becomes, what streams of messages are affected or added and how those messages (events) are defined - consumers and producers are free to change, merge and separate

> Centralize the source of truth distribute the freedom to act and change

Existing use case candidates examples

- Profession finder
- Product recommendation
- Starting Consultation / Inquiry

Potential use cases

- AS Notifications
- Aggregated daily sumarry / tender emails
- Document generation
- ...
- put differently -> ability to implement / enable a wide variety of use cases without large-scale refactoring and minimal amount of new code

Questions / Discussion

The Other Thing