# HEROKU POSTGRES

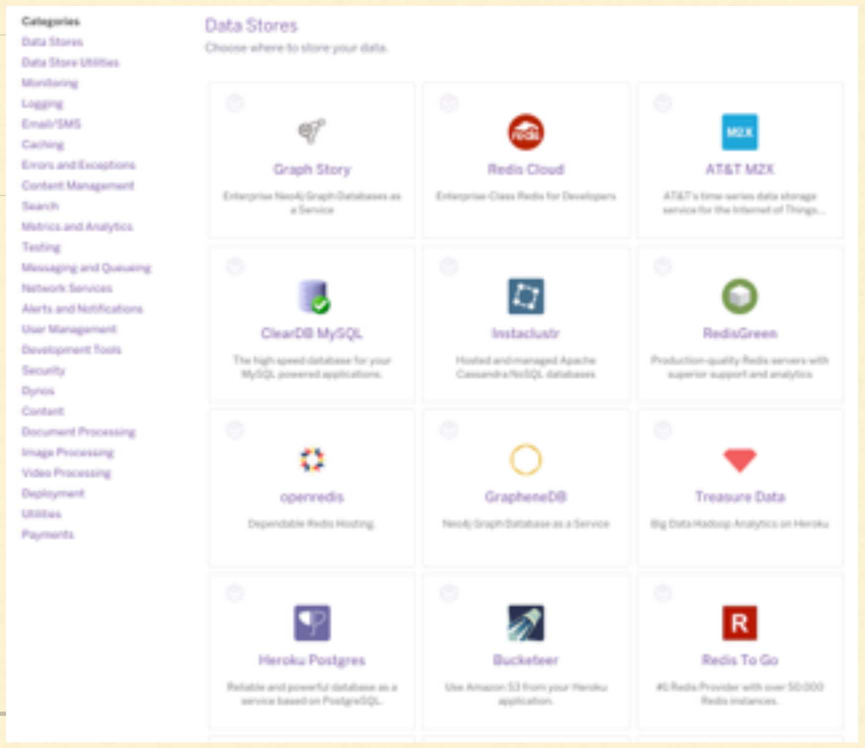
architecture of a cloud database service

## HEROKU

## DEPLOY&RUN CODE

mainly web apps
routing and scaling

## ADD-ONS



Categories
- Data Stores
- Data Store Utilities
- Monitoring
- Logging
- Email/SMS
- Caching
- Errors and Exceptions
- Content Management
- Search
- Metrics and Analytics
- Testing
- Messaging and Queueing
- Network Services
- Alerts and Notifications
- User Management
- Development Tools
- Security
- Dynos
- Content
- Document Processing
- Image Processing
- Video Processing
- Deployment
- Utilities
- Payments

Data Stores

Choose where to store your data.

**Graph Story**
Enterprise Neo4j Graph Databases as a Service

**Redis Cloud**
Enterprise-Class Redis for Developers

**AT&T M2X**
AT&T's time series data storage service for the Internet of Things...

**ClearDB MySQL**
The high speed database for your MySQL powered applications.

**Instaclustr**
Hosted and managed Apache Cassandra NoSQL databases

**RedisGreen**
Production-quality Redis servers with superior support and analytics

**openredis**
Dependable Redis Hosting.

**GrapheneDB**
Neo4j Graph Database as a Service

**Treasure Data**
Big Data Hadoop Analytics on Heroku

**Heroku Postgres**
Reliable and powerful database as a service based on PostgreSQL.

**Bucketeer**
Use Amazon S3 from your Heroku application.

**Redis To Go**
#1 Redis Provider with over 50,000 Redis instances.

heroku postgres is an add-on
internal customer of the add-ons service to test new features, help discover requirements

## ORIGINS

## SINATRA & SIMPLE-DB

implement addons api
no database service at the time, so used aws simpledb, kv store

## HOSTED ON HEROKU ITSELF

not only an addon, but all
[tk ororobous]

## SELF-HOSTED POSTGRES & SEQUEL

will talk later on moving from a KV store to postgres with hstore

## RESOURCE & SERVER

server => aws instance
resource => customer point of view
this model has decomposed into many more pieces over time

## WEB ADMIN INTERFACE

has gone through several iterations, but started just as simple http page that refreshed.
very valuable to have
there are many services at Heroku that don’t, and I’m glad we had this

## VIDEO GAME DEVELOPMENT

pvh’s past in video game led to this model
works surprisingly well

## OBSERVE / TICK

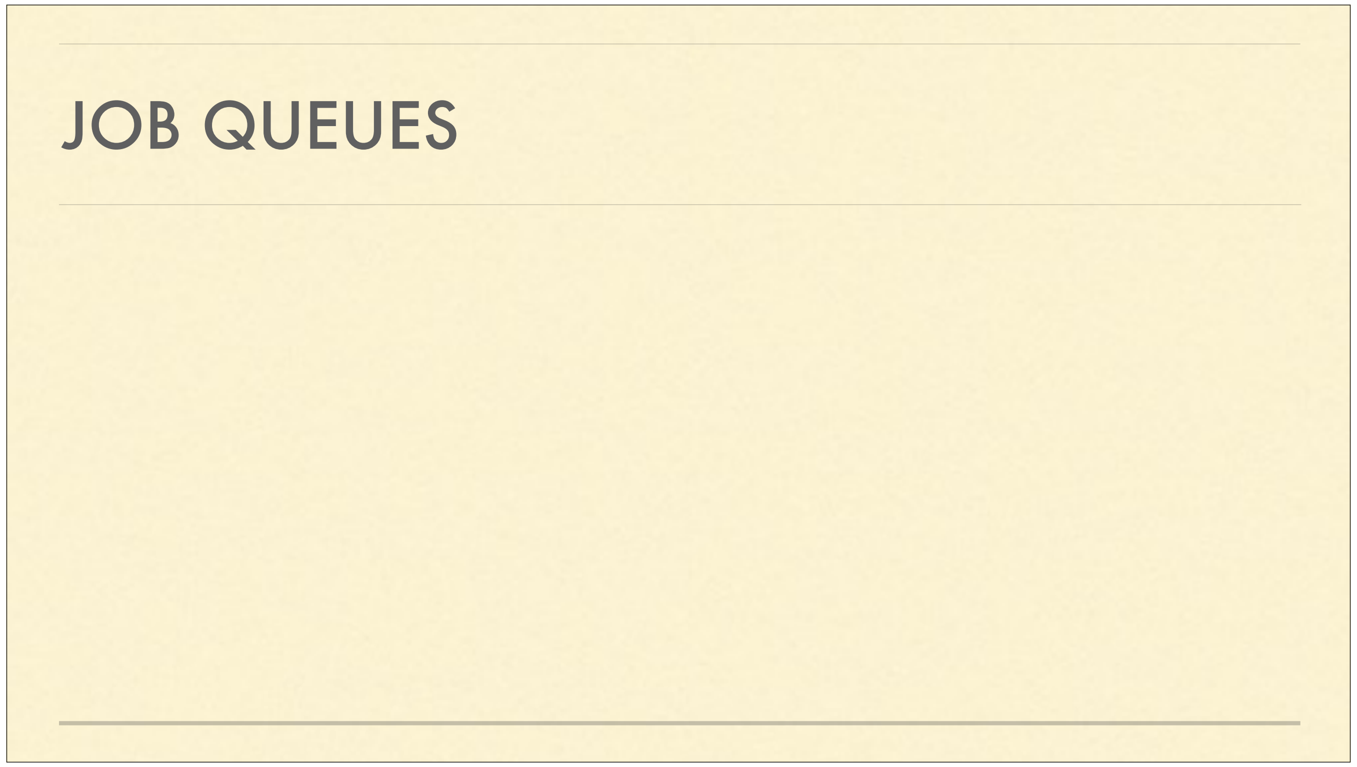
observe environment, then take action on that information, repeat

## STATE MACHINES

[TK diagram] [TK examples]

## ALWAYS CONVERGING

most of the state machines are always trying to get to a good state
example: jan 14 aws outage, need to reassociate all EIPs, can detach all and rely on states to fix everything



## JOB QUEUES

simple model each observable thing having a queue

## QUEUE CLASSIC

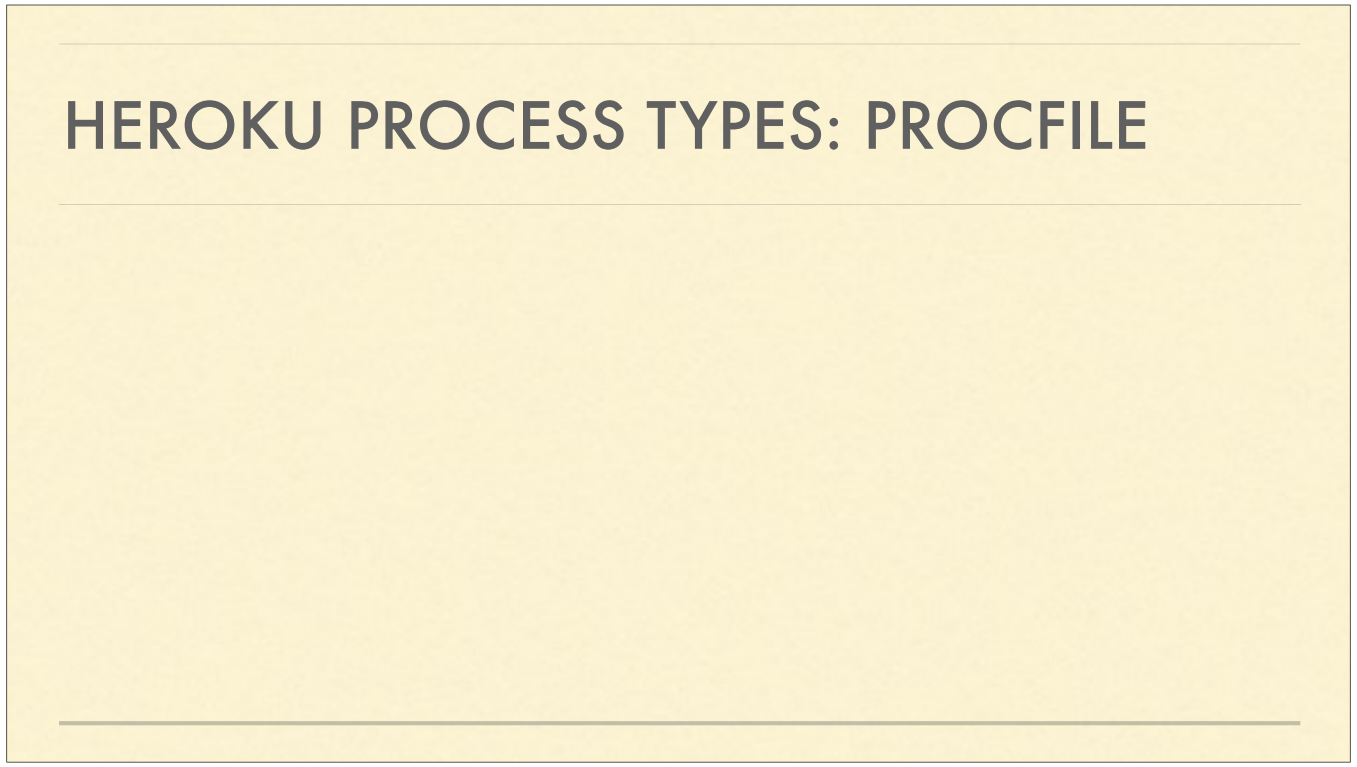
postgres queue. interesting concept of picking randomly from the head of the queue to remove contention.
in postgres nice because enqueuing a job can be done transactionally. we don’t but other teams do
[TK link]

## SIDEKIQ AND REDIS

eventually moved for performance reasons

## QUEUE PROBLEMS

first relied on jobs reenqueuing themselves, but jobs can fall out
then moved to something that examined and put in missing jobs
now have workers that fill the queue



## HEROKU PROCESS TYPES: PROCFILE

with heroku you can have one codebase with multiple different entry points
typically web for requests and worker for background jobs. maybe clock for periodic
[TK procfile example]

## ORTHOGONAL SCALING

```
$ wc -l Procfile
      27 Procfile
```

“if you have to have a trick, do it a lot”
some queues are busy, some are slow, we can dedicate as many workers as needed for each

## HSTORE

nice to be in the home of hstore
let us move from KV store to postgres

## SEMI-SCHEMA

most of our tables still have an attrs_unparsed column [TK show]
allows for quick iteration, exploration of ideas

## PROMOTE TO REAL COLUMNS

over time. don’t do this as much as probably should

## OBSERVATIONS

especially for the observations, hstore is perfect.
as we think of new things to monitor, we can just start. drop old checks. without migrations.
we only need recent data.
[TK importance sharding]

## DURABILITY & REPLICATION

## WAL-E

github.com/wal-e/wal-e

written by Dan Farina
archive command, takes WAL and uploads to s3
<Explain WAL>
takes base-backups, handles downloading+restoring wal also
critically important: things fail all the time, aws is hostile

## DISASTER RECOVERY

## TIMELINES

became a first-class model with server and resource
progression of data over time

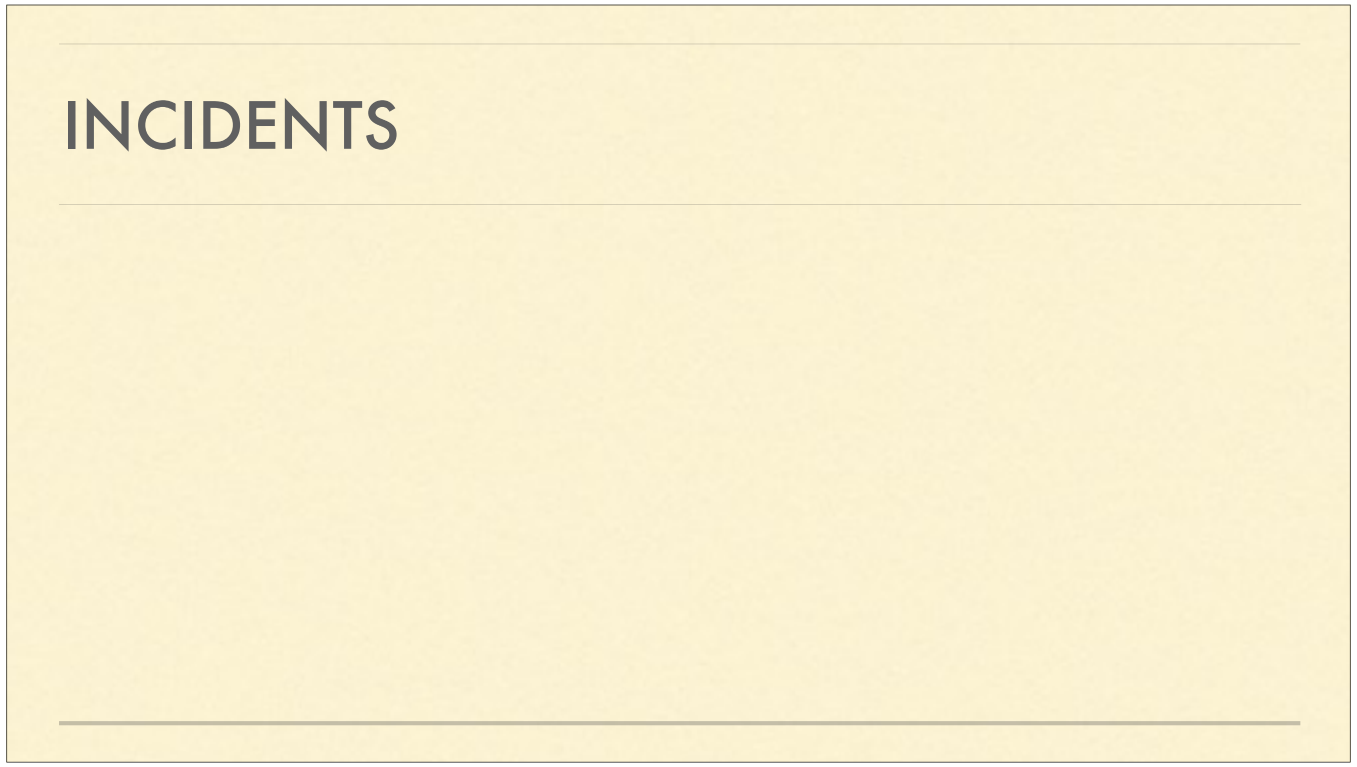
[TK diagrams]

## FOLLOWERS

on the same timeline
read-only

## FORKS

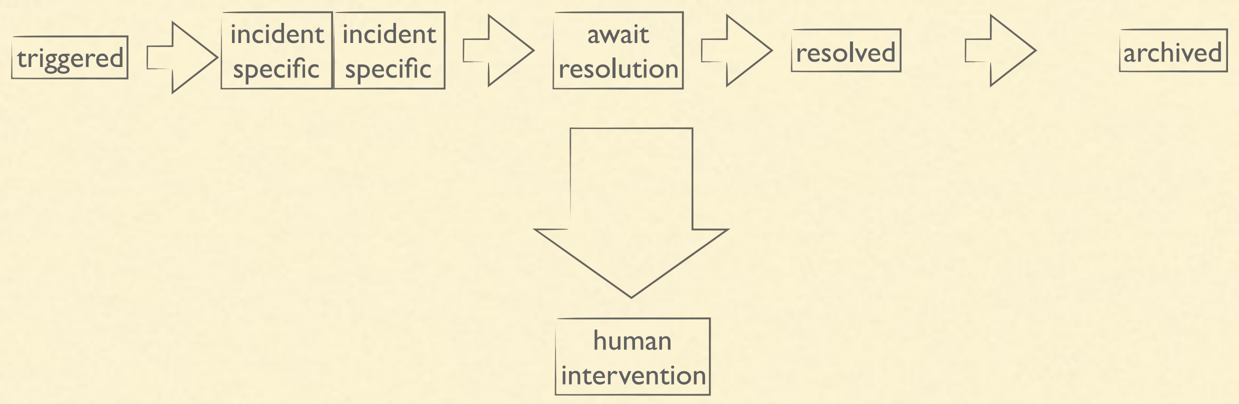
have same history, but create their own timeline
read+write
can also be point-in-time-recovery



## INCIDENTS

servers down restart, server broken need to replay from wal,
need to add another disk, replication lagged, etc

## OWN STATE MACHINE



often takes a while to resolve, need to wait on things to get provisioned, wal to download.
so each incident has its own lifecycle
again “if you’re going to use a trick use it a lot”

## HANDLES CUSTOMER COMMUNICATION

data is very sensitive, customers need to know how things are going.

## CAN TRIGGER PAGES

but hopefully does not
3 cases: not yet automated, stays too long in one state without resolution, runtime exception

## EXAMPLE INCIDENT

## CONTINUES TO GROW

started out just for simple problems, but has become the basis for a lot of automation
now long lived incidents for future action, like when we get notified that an instance will die, need to schedule changeover and notify customer, then wait until time.

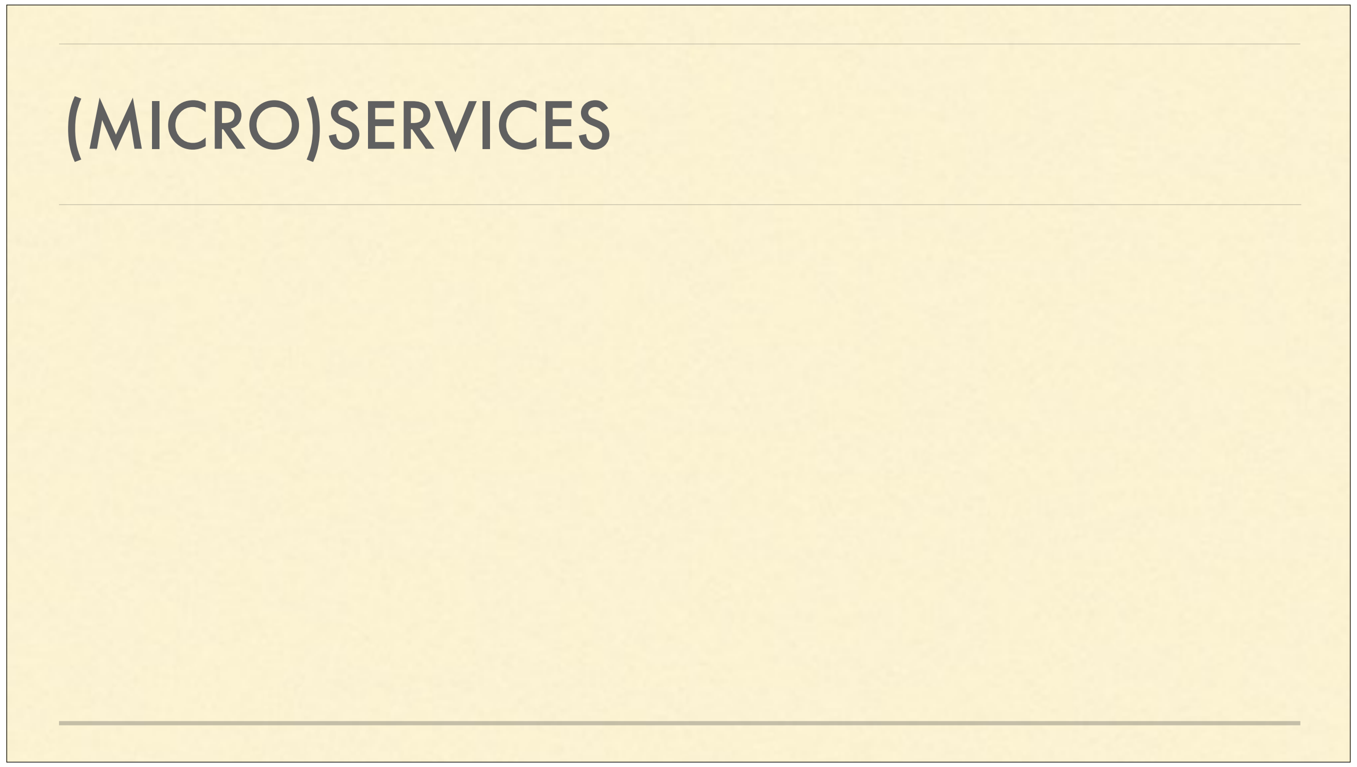
## LOGGING

## LOGPLEX

very old, central component of heroku.
[TK diagram]
all user code if prints to STDOUT ends up in logplex
monitoring and postgres logs go to logplex

## LOGS AS EVENT STREAMS

[TK] logfmt example



## (MICRO)SERVICES

eventually as things grew, just scaling out
important we don’t (often) reach for services first
breaks up code.

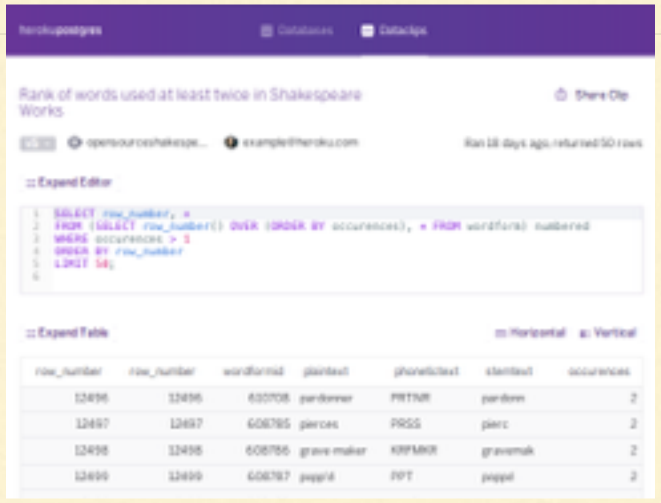
## ASIDE: CONWAY'S LAW

“organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations”
people often think this is bad, but if it’s unavoidable might as well not fight it.

## LOGICAL BACKUPS

github.com/heroku/transferatu

## DATACLIPS



Rank of words used at least twice in Shakespeare Works

Ran 18 days ago, returned 50 rows

```
SELECT row_number, *
FROM (SELECT row_number() OVER (ORDER BY occurences), * FROM wordform) numbered
WHERE occurences > 1
ORDER BY row_number
LIMIT 50;
```

| row_number | row_number | wordformid | plaintext | phonetictext | stemtext | occurences |
|---|---|---|---|---|---|---|
| 12496 | 12496 | 610708 | pardonner | PRTNR | pardonn | 2 |
| 12497 | 12497 | 608785 | pierces | PRSS | pierc | 2 |
| 12498 | 12498 | 608786 | grave-maker | KRFMKR | gravemak | 2 |
| 12499 | 12499 | 608787 | pepp'd | PPT | peppd | 2 |

## FREE DATABASE SERVICE

is a heroku app that splits up “real” databases a bunch for free customers
separate app

## MONITORING

what was just in the main app, got pushed into it’s own app that only does monitoring

## ADMIN UI

now that things were spread out between so many services, one view to see into everything

## DOWNSIDES OF MICROSERVICES

can no longer do simple joins to find things out
each app might be simpler, but the complexity is pushed into the channels