database reliability engineering

what, why, how.

Charity Majors, Founder/CEO honeycomb.io
Laine Campbell, Sr. Dir, SRE, Fastly
O'Reilly Velocity Santa Clara, 2017

@mipsytipsy @lainevcampbell
we made a book! (2:50 early release signing)
yesterday's DBA

specialized siloed superhero master builder gatekeeper
today's DBRE

educator and mentor

platform builder

force multiplier

operations expert

t-shaped
database reliability engineering

develop reliable and resilient datastores and shared components for provisioning

Provide patterns and knowledge to support other team’s processes to facilitate their work

understand and teach data access and storage nuances to ensure all service level objectives can be met

anchor teams with expertise for troubleshooting, recovery and other tasks requiring depth, not breadth
why DBRE?

reliability is only a smidgen of the operational mandate

reliability isn’t always required in operations

the closer to the data, the more reliability is necessary

data requires paranoia, not chaos

- @jessietron - https://blog.codeship.com/growing-tech-stack-say-no/
Who’s your DBA? If you don’t know the answer, it’s probably you.

The ops engineer, taking data ownership

The DBA, continuing to evolve their skillset

Software engineers developing data-driven applications

The path to DBRE
paradigm shifts
Integration between engines is crucial. We vet, find edge cases, learn patterns. One engine is less likely to suit all cases. New and shiny brings risk. Relational is not the end of the line.

Polyglot persistence

The data must flow
virtualization and cloud

DBaaS availability reduces toil

the power of rapid provisioning

new storage paradigms

forces designing for resilience

commodity hardware for DBs

Virtualization and Cloud
infrastructure as code

infrastructure can be versioned
changes become deployments
requires coding chops
no special snowflakes
forces componentization
Continuous delivery

we must be teachers, not gatekeepers
testing and compliance become top priorities
we build guardrails and tools to replace reviews
we build patterns to replace reviews
Database engineers: O.G. devops shared goals, tools and processes
- Protect the data
- Self-service for scale
- Elimination of toil
- Databases are not special snowflakes
- Reduce the barriers between software and operations

dbre guiding principles
protect the data

software development lifecycle

• Durability and integrity baked into every part of the architecture and infrastructure.

• Standardization and automation with simplicity over expensive/complicated.

• Responsibility of data protection shared by cross-functional teams.
Self Service for Scale

- Self service for scale
- Metrics knowledge base and automated discovery/collection
- Backup/recovery utilities and APIs auto-deployed for new builds
- Reference architectures and configs for data stores deployment
- Security standards for data store deployments
- Safe deployment patterns and tests for database changesets
Dbs are not special snowflakes

- Keep it simple stupid
- Commoditization
- Databases are the last holdouts of paradigms a la Bill Baker
- Moving databases into the “cattle, not pets”

●
Reduce barriers between teams

- Data-land should feel familiar and intuitive,
- Use the same provisioning and config management processes,
- Use the same code review and deploy.

not alienating or „special”
choose boring tech if possible

testing operability

building gold configurations

benchmarking performance

testing consistency guarantees

datastore decision making

datastore decision making
If your company is very young, optimize for velocity and developer productivity. The more mature your company is, the more operational impact over the long term trumps all. Spend innovation tokens only on key differentiators.
Build an effective tiered backup solutions - the right backup for the right dataset

Integrate recovery into daily activities

Work with SWE to build data validation pipelines

Build recovery testing automation

Recoverability
If you are not regularly testing restores, you have Schrodinger’s backups.

Maybe u haz backups
data distribution

replication, partitioning and sharding
distribute broadly based on requirements
determine effective number of copies
understand availability strategies for each datastore

Evaluate limitations and consequences of failover (cold caches, data loss etc...)

practice and document failover in controlled circumstances

continued training of all staff on the process to get it "muscle memory"
scaling paths

Don’t screw your future self (that will impact the future (aka keep an eye for constraints))

Growth

Plan for and test up to 10x
• build guard rails

• don’t swoop in and do it all yourself
  • help them fix it
  • don’t get pissy when people mess up

• other
  • network isolation between prod & other
  • shared on call rotations
  • boring failovers

• migration heuristic analysis

• DDL migration patterns

• migration and fallback testing
  • unit tests
  • backups, restores, verified.

• don’t get pissy when people mess up, help them fix it
- automation
- rolling DDL
- scaling up or down
- detecting or killing bad queries
- failovers and traffic shifts
build for services

- graceful degradation
- observability
- instrumentation
build for people

- tools for debugging
- checklists, documentation
- tractable levels of graphs and alerting
Release management

- Develop patterns for migrations
- Heuristics to surface risky changes
- Tiered dataset sizes for testing
- Continued developer training and education
- Migration and fallback testing
- (Development, integration, full)
basically...

I just want it to work.

I want to not hate things with it.

Awesome can do it.
self-actualized databases

- your data is safe and recoverable
- resilient to common errors
- shares processes and tooling
- understandable, debuggable
- empowers you to achieve your mission.
- your data is safe and recoverable
Don’t be scared. Mind your damn backups. Everything else will probably be ok.

In conclusion...