Best Practices *for* Database Platform Engineers

Chelsea Dole
Database Engineer, Citadel

DBA

SRE

SysAdmin

Infra Engineer

“Database Platform Engineer”

DBRE

Cloud Engineer

DevOps

Database Architect

Why bother to talk about this?

- ❌ Pitch to start leveraging this title on job postings
- ❌ “DBAs are dead!”
- ✅ Discussion of changes in industry need
- ✅ Personal career experience

- Variety of “database-ish” titles
- Variety of organizations:
  - Employee count
  - Startup funding round
  - Database team size
  - Data criticality & volume

Chelsea Dole

What do I do in this role?

1) Efficiently manage many databases as a cohesive fleet via automation: “cattle”, not pets

2) Leverage specialized database knowledge to build internal “managed database platforms” for engineers to use

What do I do in this role?

"Building RDS // CloudSQL // Azure SQL DB

for

<CURR_EMPLOYER>"

Database Admin (DBA)

- Deep DB & SQL mastery
- SysAdmin mastery
- Weaker coding skills
- Org size: all

Database Platform Engineer

- Deep DB & SQL mastery
- Weaker SysAdmin
- Strong coding skills
- Org size: medium → large

Where are Database Platform Engineers?

- Lower ROI on "platform building" for small companies

- Overkill for large companies without heavy data challenges

1.

Medium to large SaaS companies

Where are Database Platform Engineers?

“Cattle vs pets”

- Provisioning frequency
- Microservice architecture

2.

Organizations with many databases

Where are Database Platform Engineers?

- Startups: correlation
- Cloud → increased ROI of automation
  - CLIs
  - Infra-as-code

3.

Organizations which heavily leverage the cloud

Database Admin (DBA)

- 10,000-person consulting firm with 5 huge, high-traffic databases
- Credit union of any size running on-prem

Database Platform Engineer

- 500-person SaaS company with an IOT product and many databases
- 70-person AI startup running on AWS or Azure

Database Platform Engineer

Best Practices

(in my opinion)

1. Own the database provisioning process

- Control & understand the “playground” you provide
- Hardware & software
- Establish consistency

}

Consistency: groundwork for automation

1. Own the database provisioning process

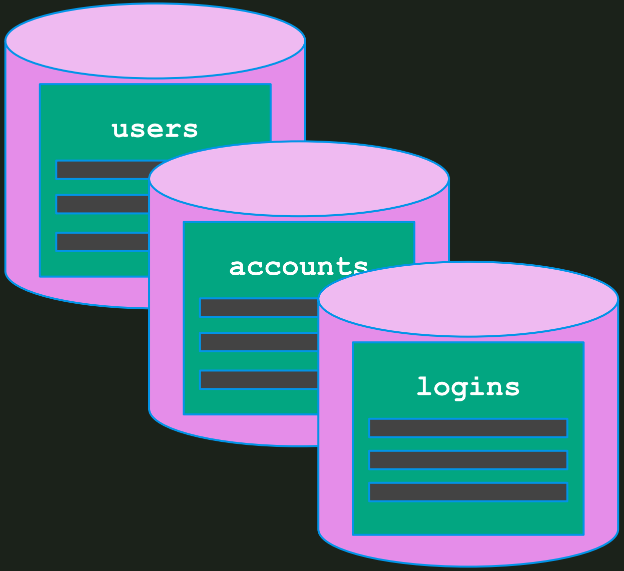
Consistency in what?

- Naming conventions
- Limits on databases, schemas
- Standard permissions
- Secrets storage
- Server/database relationships

}

Consistency:
groundwork
for automation

1. Own the database provisioning process



1) Form submission

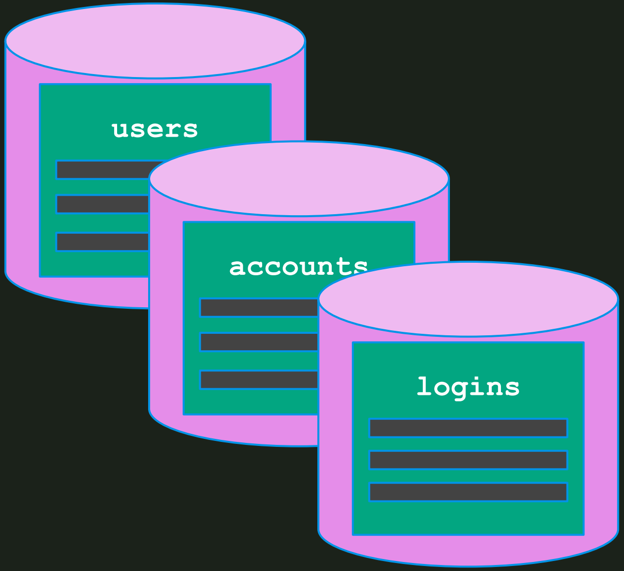
2) Worker queue
    a) Hardware**
    b) Software
    c) Secrets
    d) Observability

2. Don't make it *too* easy to provision databases

- Frictionless ability to provision hardware: $$$

- Microservice architecture doesn't port well to databases
  - 1 overburdened main DB, vs
  - Too many small overprovisioned DBs

2. Don't make it *too* easy to provision databases



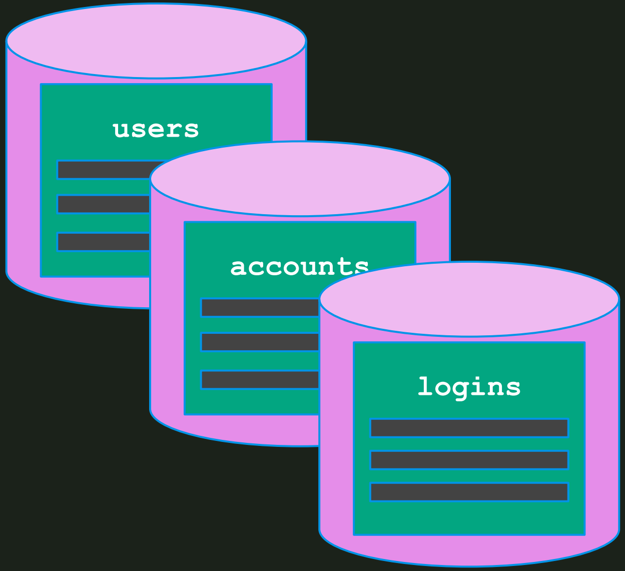
1) Form submission
2) Provisioning approval
3) Worker queue
    a) Hardware
    b) Software
    c) Secrets
    d) Observability

3. Maintain health beyond provisioning

- Archive/delete deprecated tables
- Delete unused indexes
- Maintain database metadata accuracy
- Right-sizing servers

vs 🪵

3. Maintain health beyond provisioning

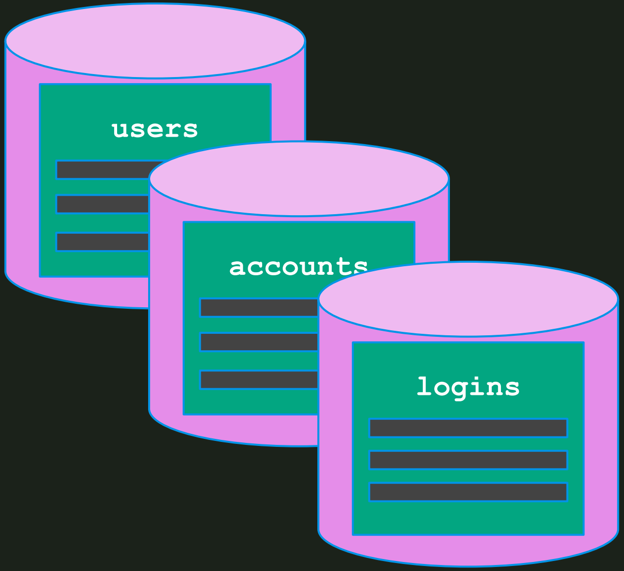


vs

- Cost visibility dashboards
- Underutilized resources
- Etc

(or else… what?)

3. Maintain health beyond provisioning



1) Form submission
2) Provisioning approval
3) Worker queue
    a) Hardware
    b) Software
    c) Secrets
    d) Observability
4) Maintain long-term health

4. Manage database metadata dynamically

- Team ownership
- Service discoverability
- Contact methods
- Etc

} Harden your systems to corporate reality & human fallibility

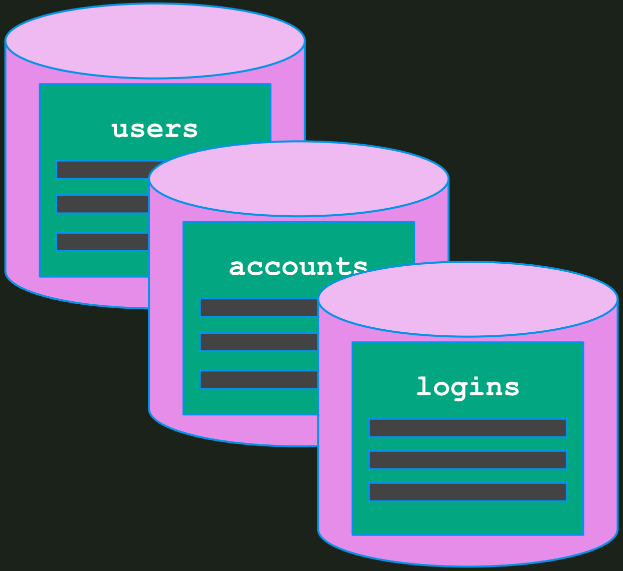
Risk: *introduction of new "single point of failure"*

4. Manage database metadata dynamically

Database metadata storage:

- Highly available
- Decoupled from other databases
- Document storage?

4. Manage database metadata dynamically



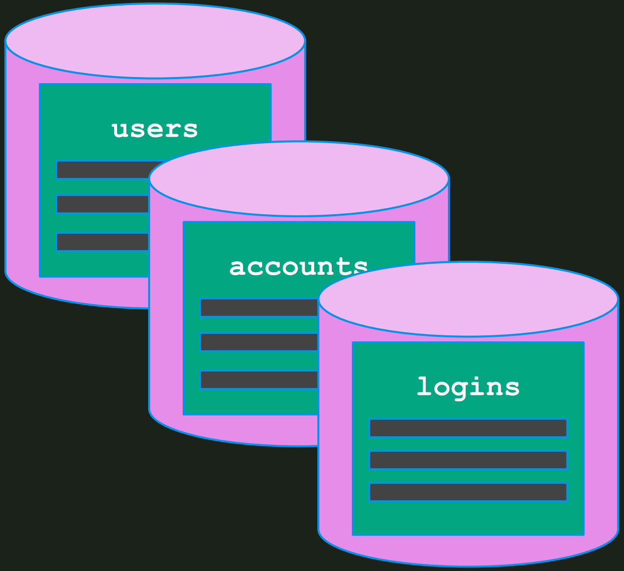
1) Form submission
2) Provisioning approval
3) Worker queue
    a) Hardware
    b) Software
    c) Secrets
    d) Observability
    e) Dynamic metadata store
4) Maintain long-term health

5. Build dev-owned tools, not "footguns"

- Package basic DBA tasks, but own the hard problems
  - ✅ Reindex, cancel PIDs, password rotation, advanced diagnostics...
  - ❌ Advanced logical replication, DR, unlimited config selection...
- Migration safety linter, DB CLI

$\}$ Allow engineers to learn, and leverage your expertise

5. Build dev-owned tools, not "footguns"



1) Form submission
2) Provisioning approval
3) Worker queue
    a) Hardware
    b) Software
    c) Secrets
    d) Observability
    e) Dynamic metadata store
4) Maintain long-term health++

6. Solve for fleet-wide change rollout

- OS upgrade
- Architecture changes
- Standard role/function
- SSL cert rotation
- Config change

$\}$

- Hardware/OS
- Postgres

7. Connect via static A Record/CNAME

10.22.34.01 → mydatabasecluster.company.com

- A Record, CNAME, Proxy
- Infra changes without app-side coordination
- “RDS for…”

} Logical replication-based workflows

🌶️ slightly spicier 🌶️

Database Platform Engineer

Best Practices

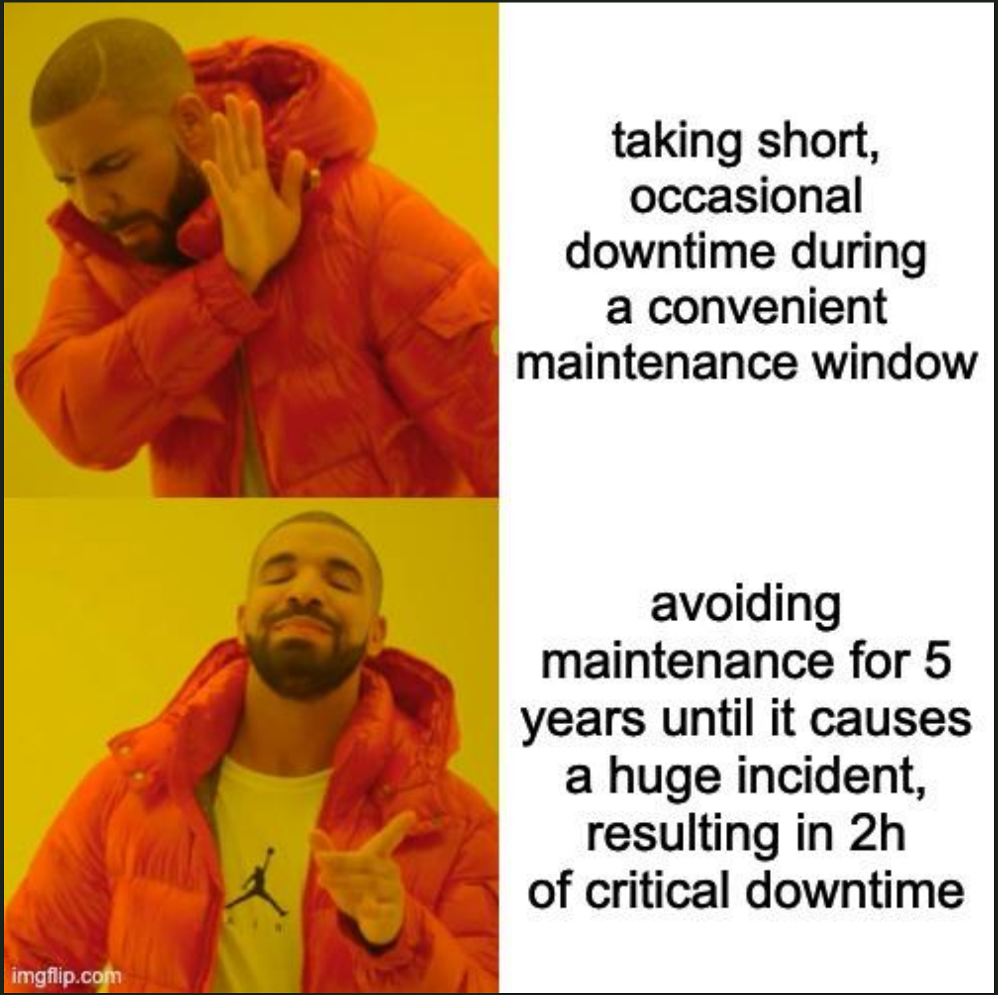
(still in my opinion)

Pop Quiz:

You ask a software engineer how much downtime their database can take.

What do they respond?

“None”


taking short, occasional downtime during a convenient maintenance window
avoiding maintenance for 5 years until it causes a huge incident, resulting in 2h of critical downtime
imgflip.com

8. Take planned downtime regularly

- Establish maintenance windows & expectations
- Enable engineers to schedule tasks
- Seek leadership buy-in

CYA:

- Measure server-level downtime
- Announce publicly

9. Prioritize observability > latency

- Default Postgres: latency > observability
- Advanced observability/logging “off” by default
  - More metrics/logs → more CPU/IO
- Modern disks

9. Prioritize observability > latency

- Microservices == app-level database sharding
- Choose when to turn off, not turn on

9. Prioritize observability > latency

- `pg_stat_statements`
- Basic logging
  - `log_connections` / `log_disconnections`
  - `log_lock_waits` (& `deadlock_timeout`)
  - `log_replication_commands`
- Auto-explain
  - `auto_explain.log_min_duration = '<>s'`
  - `auto_explain.log_analyze      = on`
  - `auto_explain.log_buffers      = on`

9. Prioritize observability > latency

Gotchas:

- `log_statement` (default = none)
- `log_destination = ‘jsonlog’`

and finally…

10. psql is still a "first class citizen"

- Incident management & debugging challenging issues
- Breakglass processes in case automation is broken

$\}$

Enable your team to spend time on interesting problems

Thank you! 👋

Chelsea Dole

chelseadole.com