



Cloudy database management - no problems at all?

Kaarel Moppel, Cognite

pgDay Nordic 2023

Meet the presenter

- Full-time “fighting” with databases for the last 16 years
- 12 years with Postgres, with 5 of that as a consultant
 - Thus I've gotten a pretty unique insight into architectural / tooling decisions of dozens of companies and have a pretty good understanding of what works reasonably well and what not
- Principal DBRE at [Cognite](#) where we build an Industrial DataOps [product](#) with all the modern buzzwords covered (AI, Digital twins, Knowledge graphs, ...)
 - Currently we have around 300 primary managed instances of Postgres + hundreds of other non-relational instances (FoundationDB, Kafka, Elastic, ...)

Agenda

- **Managed databases overview, pros and cons**
- **Typical DB-related DevOps tooling**
- **Newer trends**

Managed Databases high-level overview



So...welcome to the managed Cloud!

- No way around managed databases for scale-ups
 - They do solve many problems for us to focus on other things
 - Very difficult to hire DB engineers

Current vacancies

LOCATION	PRODUCT	TEAM
Design System Lead		Oslo
Database Reliability Engineer - ElasticSearch		Remote - Oslo
Database Reliability Engineer - Kafka		Remote - Oslo
Database Reliability Engineer (DBRE) - Go		Remote - Oslo

Managed DB infrastructure PROS

- Well, don't have to think about DB infrastructure anymore - it becomes a guaranteed service with a SLA!
- Some things become really hard to screw up
 - The initial provisioning part - a one-liner on the CLI basically
 - High-Availability / automatic failover is another “checkbox”
 - Backups
 - Most essential tuning flags set to match the SKU. Workload agnostic though (*)
- Can scale easily if to throw \$\$ at the problem
 - Not talking only about standard SKU++ here - there also some quite high-performance wire-compatible derivatives available (the likes of Azure Cosmos DB, AlloyDB, Aurora)
- Don't need to know any engine internals (initially), can just jump to SQL / business logic!

* AlloyDB brings something to the table here

Managed DB infrastructure CONS

- \$\$ - easily 2-3x the cost of plain VMs with database self-installed (*)
 - Even more if to add HA (*)
 - Some obnoxious cloud providers don't even lay out the price for you in the UI
 - Extra sad fact of the matter - developers generally have no good understanding of future performance needs (especially if using some ORM-s etc, so that quick benchmarking is hard) and thus commonly just over-provision
 - On some clouds / products there are some ways to reduce costs a bit though (VM suspend, auto-scaling based on load) if load is not constant
- Very slow or “just” slow disk access for low / medium tier SKUs
 - IOPS tied to both SKU size and volume size in most cases
 - Some clouds provide guaranteed IOPS for \$++ as a remedy
- Debugging is super hard, if it should come down to that (and it eventually will*)
 - Some clouds are clearly better than others when it comes to contacting the “helpdesk”

DB domain specifics - well SOLVED themes

- Operational safety - no real superuser
- Point-in-Time and normal Disaster Recovery (*)
- AD / IAM authentication and authorization integration
 - Both for control and SQL plane
 - VPC only SQL access option
- Auto-updates to minor versions
- Alerting integration (on some clouds)
- Auditing for most important instance lifecycle actions

DB domain specifics - partially or NOT SOLVED themes

- Lacking metrics - very basic / generic insight on DB internals on most clouds
 - Need to roll your own custom layer still for larger organizations
 - Sometimes just to solve the access problem i.e. to make metrics public
- Provider side config tuning at very basic level
 - Need engine-specific knowledge
 - A lot of settings even can't be set for no good reason (*)
- Basic perf problem detection / solving (Top X slowest SQL mostly)
 - Need to have engine knowledge for background processes and general workload
- Major version upgrades require manual attention
 - And they take a lot more downtime than necessary, compared to "on-prem"
- Access security management could be better
 - Only few clouds support client certificates
 - Looking at HBA rules leaves you guessing most of the time
 - Brute-force protection not activated on most clouds, even if world open 🤖

DB domain specifics - partially or NOT SOLVED themes

- For Postgres - very limited selection of “extensions” are allowed
 - From “super-extensions” PostGIS is the only one universally supported
 - TimescaleDB not available everywhere and lagging in versions
- Instance lifecycle events tracking suffers often from “selective memory”
- Restrictive server log search and long term archival options (*)
 - Logging options (log_line_prefix) not tunable at all on most clouds
- In addition to operator configuration / tuning mistakes, providers also take “interesting” defaults sometimes (huge max_connections, synchronous_commit=off)
- Obviously no 100% guarantees also against provider side mishaps / downtimes. Some problems we’ve faced during last 1.5yrs:
 - Collations FU on minor update
 - Subnet IP caching / depletion
 - Storage auto-scale fragmentation penalty
 - Too active API saturation / throttling

Another “problem” - Wow, so many clouds...

- Which to take?
 - Well, generally the one which you're already using...
 - But can see up to 50% perf differences on comparable SKUs
- What if you're going multi-cloud?
 - Many simple things become problematic now...there are always differences not only in API but also concepts / naming
 - Given you're on the same Cloud + Region, could also use meta-providers!
 - Like aiven.io e.g.
 - Get nicer API / UX usually
- Postgres or Postgres-compatible? Or Serverless?
 - [Aurora](#)
 - [Neon](#)
 - [Bit.io](#)
 - ...

DevOps and Cloud DBs



DevOps - DB-related tooling needs

Handling the most essential DB lifecycle events like initial provisioning

- Lots of choice
- A few no-brainers like Terraform and Ansible

Larger shops though will often discover that there are lot more DB-specific tasks to handle / automate in a repeatable manner:

- Tools for evolving the DB schema in a VC way
 - Flyway, Liquibase, ...
- DB clones for cheap feature testing / development
- Cloning prod data to staging with anonymization / obfuscation
- Unified metrics / logging analysing experience
- Regular failover, DR and chaos-testing
- Automated data lineage tracking
- Postgres specific maintenance (bloat reduction, reindexing)
- ...

DevOps tooling - Terraform

The most common tool for essential DB-lifecycle management

PROS

- A very popular choice for Cloud in general
 - Thus a relatively safe bet also for DB management
- Good development / rollout tooling support
- Top tier DB modules maintained by cloud provider directly or use some code generation from API specs
 - i.e. relatively short delays to support new cloud features / APIs
 - Can just look at the Cloud provider API documentation for available attribute values etc

DevOps tooling - Terraform

CONS

- Lacking imperative actions and full Postgres lifecycle support
- Copy-Paste templates can / will become a problem for hundreds of instances
 - Some unnecessary stuff will start to make rounds
- Some danger of “versioning hell” - hard to evolve root modules without cryptic errors / manual state-file modifications for already running stuff
 - Results in outdated settings for most already provisioned instances
 - Note that it's even debatable if it's a good idea to “backfill” e.g. apply new tuning settings
- Not a state machine per se, configuration drift can occur on manual overrides
 - Due to incident handling mostly - needs discipline from DevOps / Incident teams!

So in short - take existing tooling as "something", but be open to mix & match, and prepared for going "manual" for certain actions still

Terraform at Cognite

How it's used at Cognite in DB space

- "Root" module and a sample "intermediary" module maintained by DB engineers, project "intermediary" and rollout modules by teams
 - Root - best practice tuning, backup, HA settings + some company specifics, e.g. store logins in a certain space, monitoring, schema default privileges
 - Intermediary modules - project specific changes + naming / tag changes
 - Rollout modules - mostly a clone stamp from "intermediary" for some env X
- Devs copy-paste and replace, create a PR, get initial "plan" errors if any
 - Relying on [Atlantis](#) here
- Gets peer / DBRE reviewed depending on project, devs roll out
 - Some transient errors are possible still, re-plan / re-apply first

Things that can't be automated / DevOps-ed away

Sad fact of life - the bigger you scale, the less an average dev knows / sees / cares about databases. Some remedies:

- Documentation for devs - just the right amount, and not too tribal
- Take as many "sane default" decisions as possible as a technology expert
- Knowledge sharing. Some topics can be "looped" eternally basically as new devs onboard
 - Effective indexing
 - Partitioning
 - Testing your queries properly
 - Sufficient(*) test data generation + EXPLAIN ANALYZE
 - A "Postgres guild" with monthly presentations to tackle that at Cognite
- "One size fits" alerting thresholds from DB experts generally don't work well
 - must be easily configurable but team-driven and owned

Next level of DevOps



Next level of DevOps = no “Ops” at all, just “Dev” :)

- Meaning “~~Turtles~~ K8s all the way down”
 - Abstracting away the infrastructure
 - Well, for developers at least...
- Apps getting deployed in a fully declarative, platform-agnostic way, including the databases!
- That model doesn't match well though with managed databases, because they're still external to K8s.
- Need to choose:
 - Install / run DBs directly in K8s in a semi-managed way
 - Or intermediate somehow between K8s and the Cloud

Semi-managed Postgres on K8s

- Should be only used for some specific needs(*) or for those who want more control over all tuning settings or need a true “superuser”
 - Doesn't guarantee any automatic performance improvement, especially on managed K8s
 - Good for saving some \$\$ though
- Need to choose an operator - for Postgres:
 - <https://github.com/CrunchyData/postgres-operator> 3K (Github stars)
 - <https://github.com/zalando/postgres-operator> 2.9K
 - <https://github.com/cloudnative-pg/cloudnative-pg> <1K
 - <https://github.com/ongres/stackgres> <1K
 - <https://github.com/percona/percona-postgresql-operator> <1K
 - ...
- NB! You're now basically on your own with non-standard events!
 - Consider getting also some support contract for (possible) problems



K8s intermediated managed Postgres

- Need to choose some “operator of external resources” layer
 - We experimented with [Crossplane](#), but didn't work out for us
 - There are a few others like [movetokube/postgres-operator](#)
 - All clouds will start to look the same:

```
---
apiVersion: database.example.org/v1alpha1
kind: PostgreSQLInstance
metadata:
  name: my-db
  namespace: default
spec:
  parameters:
    storageGB: 20
  compositionSelector:
    matchLabels:
      provider: gcp
  writeConnectionSecretToRef:
    name: db-conn
```

Another alternative - build your own IDP!

- Internal Developer Platforms (IDP) are all the rage nowadays
 - Meaning a “self-service”, API or some configuration language (K8s CRD-s) that already knows about your environment and aims to simplify **developer friction basically to zero**
 - Not possible to cover 100% of requirements / apps of course, aim for Pareto / golden path
- That's what we at Cognite actually see as a long-term solution and are building one
 - Not a particularly easy or cheap undertaking though - be warned :)
 - A K8s Custom Resource looks something like that then:

```
apiVersion: infra.cognite.ai/v1alpha1
kind: CdfService
metadata:
  name: kaarel-test-db1
  namespace: app-1
  labels:
    team: devtooling
spec:
  postgresdb:
    enabled: true
```

Wrap-up



Key takeaways

- Managed databases are not a silver bullet by far - solves just a part of the equation
- Large organizations will hit some limitations of managed services sooner or later - test (performance) thoroughly and have a backup plan in place
 - For example test how to migrate off your chosen cloud provider or product - some providers make it harder than necessary
- Most popular DevOps tools “per design” don’t cover the whole lifecycle
 - Forget the idea of a "single tool to rule em' all" as you get bigger
 - Need to combine a few and be prepared to go "manual" for certain actions
 - A custom control-plane / IDP can be a better solution
- Need to have a few competent people onboard still - we’re not as far yet as one might think with managed DB offerings and DevOps tooling
 - As databases really are inherently special with their “annoying” lifecycle and service criticality

THANK
YOU

```
sh/IsString', m
import {createAuthen
ClientOptions {appld: string
i<T>(api: T | undefined): T {if
export default class BaseCognite
unction'; import {isObject from 'ad
sicalHttpClient'; import {CDFH
ns?: OnTokens?; OnToken
anager; try to create key
logout?: LogoutOpti
pRequestOptions; st
from package.js
nAuthenticate?; On
do not re-login wi
ngedIn: boolean
_SDK_HEADER
Url, isUsingSSL
UP!; export inte
n api; export f
adataMap; priv
tApi'; import {
in!; import { M
string; export const
thApi? or loginWith
logout; return this.log
LoginAPI from './api/lo
entiateFunction, OnAut
appld: string; leUrl?: str
| undefined): T {if
export default class BaseCogniteClient
unction'; import {isObject from 'ad
sicalHttpClient'; import {CDFH
lidator, RetryValidator,} from
POPUP?; onTokens?: OnTok
Cognite client instan
Api: LoginAPI
```