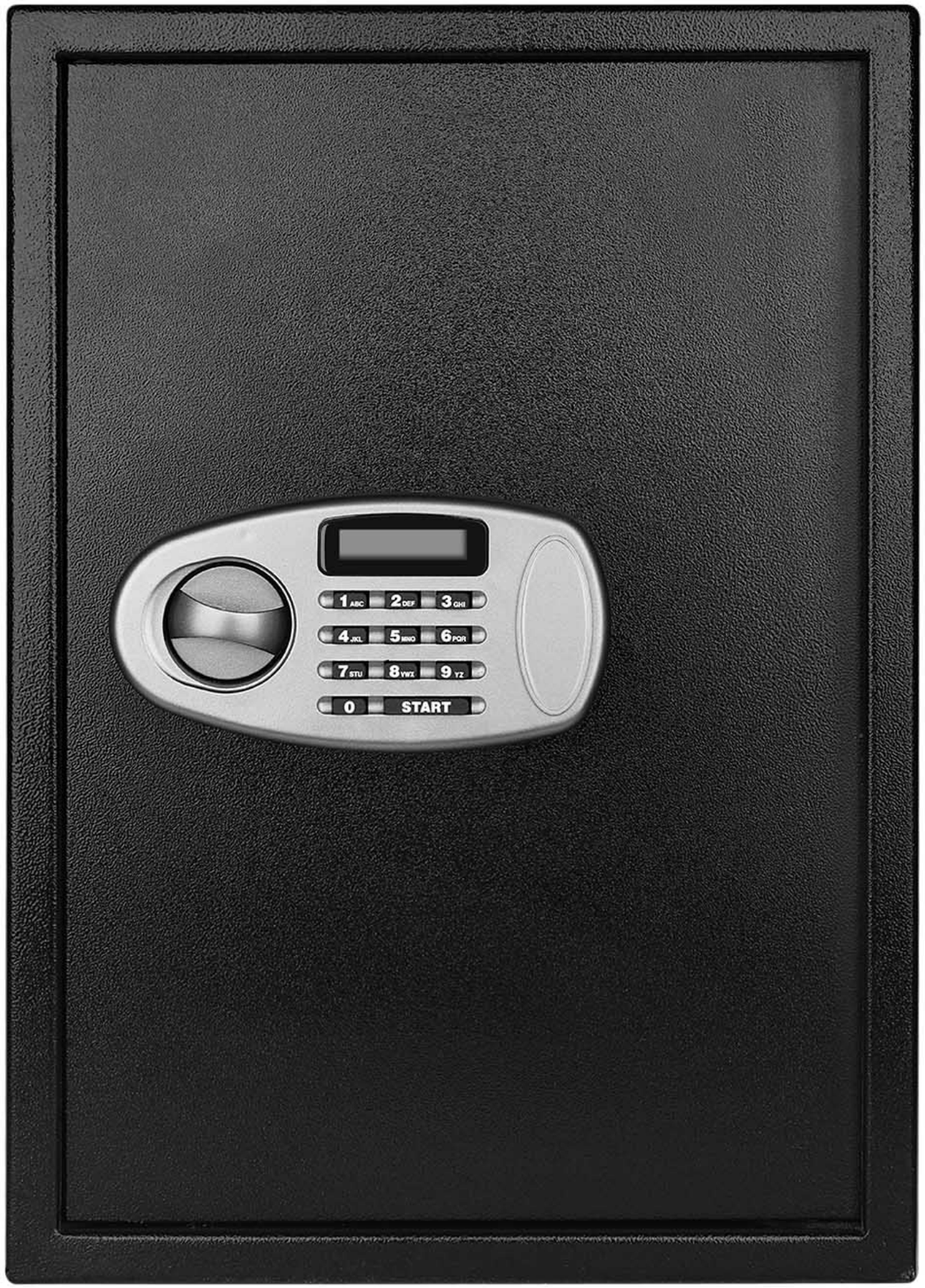


Hosted PostgreSQL:
An Objective Look

Christophe Pettus
PostgreSQL Experts, Inc.

FOSDEM PGDay 2020

Christophe Pettus

CEO, PostgreSQL Experts, Inc.

christophe.pettus@pgexperts.com

thebuild.com

twitter @xof

"It's more of a comment..."

"It's more of a comment..."


EXIT

What we'll talk about.

- Heroku Postgres ("Heroku").
- Amazon RDS for PostgreSQL ("RDS").
- Azure Database for PostgreSQL ("Azure").
- Google Cloud SQL for PostgreSQL ("Google").

What we won’t.

- Amazon Redshift.
- Azure Database for PostgreSQL Hyperscale (Citus).
- Amazon Aurora PostgreSQL.

What (else) we won't.

- Pricing.
  - Far too many variables.
  - As a **very rough** guideline, these services cost around 30% more than an equivalent "bare" instance.
- GUI quality.
  - Except my subjective impression.
- Comparative support quality.
  - Too much noise in the data.

What they are.

What they are.

What they are.

What they are.

Common to All…

- Provide a database service using the standard PostgreSQL protocol.
- Run the community version of PostgreSQL (with very minor patches, if any).
- Run in a sealed environment (no shell access to the instance, no PostgreSQL superuser access, no extensions with system access).
    - Built on a locked-down Linux box and NAS storage.
    - All controls are through a web GUI, command-line interface, and an API.
- Handle basic database backups and high-availability for you.

General Limitations.

- Cannot install your own extensions.
  - Such as: pg_partman.
- No true PostgreSQL superuser account.
- Tend to lag behind community PostgreSQL by 1-2 minor versions.
- New major versions can take an extended period to be released.
- Highly shared infrastructure completely out of your control.
  - Can be over-provisioned and have mysterious outages and slowdowns.

As Gilbert and Sullivan Said…

CAPT. The NAS mount is never degraded!

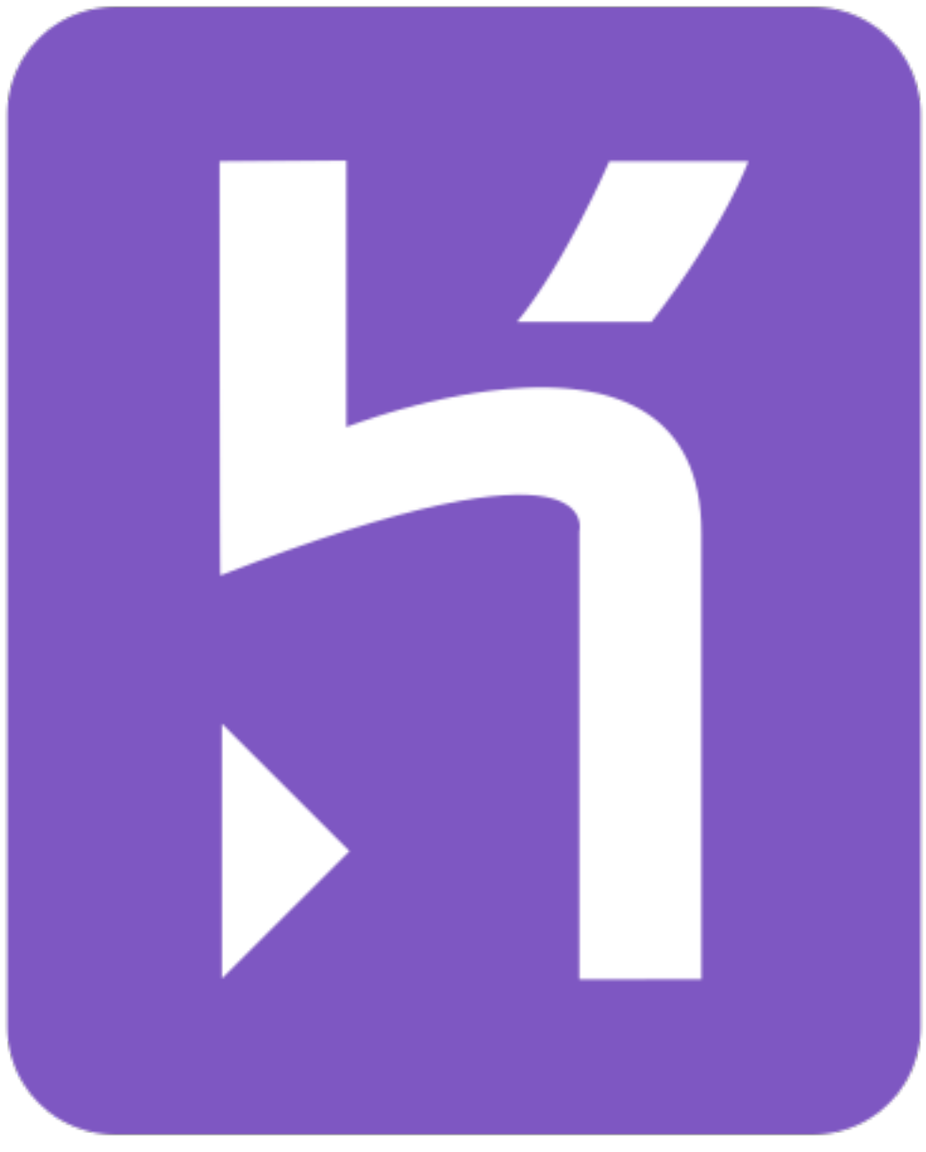
ALL. What, never?

CAPT. No, never!

ALL. What, never?

CAPT. Hardly ever!

Heroku.



Heroku.

- The oldest of the bunch.
- Now a part of Salesforce.
- Built on top of Amazon Web Services.
- Unique architecture.
  - Database-oriented rather than instance-oriented.
  - Very… distinctive database names like `phajlfadsehreaq`.
- Technically an add-on product under the general Heroku grid computing offering.

Heroku: How Much?

- Database sizes up to 3TB.
- Largest “instance” is 488 GB of RAM.
- Heroku’s offerings are “plans” rather than instances.
- Your individual database may be hosted on the same PostgreSQL server as other customer’s.
    - Although unlikely at higher plans.
- Execution units available not published.

Heroku: Interface and Controls.

- Makes very heavy use of the CLI for tasks.
- Many operations can’t be done or are awkward using the GUI.
- Good role and delegation system.
- *IMHO, GUI is confusing and hard to navigate for most database tasks, made up for by a very powerful CLI.*

Heroku: Configuration.

Heroku: Configuration.

Nope.

Heroku: Configuration.

- Hope you like their settings!
- Almost no ability to configure PostgreSQL.
  - Even non-intrusive settings like log format.
- OK, you can configure three: log_lock_waits, log_min_duration_statement, log_statement (on some plans).
- Their default settings are, however, generally reasonable.

Heroku: Access Control.

- No exposure of pg_hba.conf.
- For network-level access, firewall-based (whitelisted IP ranges).
- Wraps the PostgreSQL role system with a “credential“ architecture.
  - Slightly annoying if you are used to PostgreSQL roles.
  - Very handy if you aren’t familiar with the role system and just want to grant blocks of permissions.
- pgbouncer can be configured as a front-end pooler.

Heroku: Monitoring.

- Largely relies on outside services for graphs and database monitoring.
  - Specifically, Librato.
- A pretty good suite of query analysis tools (based around pg_stat_statements and pg_stat_activity data).
- A strange obsession with cache hit ratio…
  - … which is kind of a problem on a shared instance.

Heroku: Backups.

- Scheduled and on-demand base backups.
- WAL archiving for PITR.
  - Uses WAL-E!

Heroku: Upgrades.

- Upgrades use pg_upgrade and the CLI.
- Nicely designed and orchestrated for minimum downtime.
- Given the locked-down environment, unlikely for anything to go wrong.

Heroku: HA and Replicas.

- Only on higher plans.
- Built around streaming replication.
- Promotes and swaps in the secondary for you.
- New endpoint is automatically propagated within Heroku, but not to outside apps.
- ~~Followers~~ replicas can be spun up as read secondaries.

Heroku: Logging.

- Fixed-format logging. Hope you like it!
- Uses the CLI to download and tail logs.
- Very unfriendly with tools like pgbadger.
- Does allow additional log information with database and system-level statistics.

Heroku: Quirks and Goodies.

- A very locked-down environment.
- Too locked-down to be very quirky!
- No logical replication in or out.
- “Dataclips”: Shareable, parameterized queries with cached results.

Amazon RDS for PostgreSQL.

"RDS."

- The one to beat.
- Introduced (for PostgreSQL) in 2013.
- Popularized the "PostgreSQL as a general DBaaS" concept.
- Built on top of standard EC2 instances using EBS storage.
  - No local storage; everything is NAS.
- By far the market leader, which means we know more bad stuff about it than the others. This is not really fair to RDS.

RDS: How Much?

- Database sizes up to 16TB.
- db.r5.24xlarge instance is 96 execution units, 768 GB main memory.
- All storage is on an EBS mount.
  - Up to 80,000 IOPS maximum performance.

RDS: Interface and Controls.

- Very comprehensive API and matching set of tools.
  - Lots of automation support (Terraform, Ansible, etc.).
- The GUI allows pretty much all of the common operations without too much fuss.
- *IMHO, GUI is way too 2001: lots of clicks and page reloads to do basic operations.*

RDS: Configuration.

- Near complete configurability through parameter groups.
- Very weird and quirky interface: need to understand what underlying units PostgreSQL uses.
  - work_mem in 8KB, booleans as 0/1, etc.
- Parameter groups can be shared between instances… very handy!
- Can calculate parameter values using expressions based on instance configuration.
  - Community PostgreSQL should totally have this.
- Parameter groups are not moved forward on upgrades, and units can change… be careful!

RDS: Access Control.

- pg_hba.conf? What's that?
- 100% based around AWS security groups.
- No role-based access control to the instance.
- Instances can have a public IP, a private IP, or both.

RDS: Monitoring.

- Lots and lots of graphs which are probably correct most of the time.
- All of the major monitoring services can monitor RDS as well.
- Performance Insights is a very handy graphical wrapper digesting pg_stat_activity and pg_stat_statements output.
- You also get a web interface around `top`. So there's that.

RDS: Backups.

- Scheduled and on-demand base backups.
- Internal tooling that highly resembles WAL-E for backups.
- Can do PITR with 5 minute granularity.

RDS: Upgrades.

- Upgrades use pg_upgrade.
- “Push-button” from the GUI, either scheduled or immediate.
- Upgrades can fail, especially with databases that have been brought forward from earlier versions.
- You sometimes need the CLI to get the actual failure reason out of a file on the instance.

RDS: High Availability and Replicas.

- HA is built around a “shadow“ replica in a different AZ.
    - Not streaming replication; some kind of exciting DRBD-like replication between EBS mounts.
- You have to pay for it, but it doesn’t take query traffic.
- Failover is DNS based; same DNS name now points to the new primary on failover.
- Can spin up replicas from the GUI/CLI/API, and promote them to primaries.
    - Can be in a different region than the primary.

RDS: Logging.

- There are logs.
- You can use the API to download them. It's very slow.
- You can carefully navigate to one, find it, click a radio button, click another button, open it, and then right click to download it.
- Log format, rotation, retention are not configurable. Hope that event you're diagnosing hasn't aged out!
- Can turn on CSV logging, but then you get both stderr and CSV.
- Logs always go to the database volume; you can choke it with too-high logging.
- This is not RDS' strong point.

RDS: Quirks and Goodies.

- The richest set of extensions and PostgreSQL core features.
- Logging is a mess.
- Parameter group UI is actively user-hostile.
  - Real-life large company sites have been brought down by it.
  - RDS often forces an instance restart for parameter changes that do not technically require it.
- RDS databases tend to run high in CPU.
- Strange things only seen on RDS.
  - LWLock pileups.

Azure Database for PostgreSQL.

“Azure.”

- Microsoft has joined the party.
- Introduced (for PostgreSQL) in 2017.
- Runs in the general Azure compute cloud environment.

Azure: How Much?

- Database sizes up to 16TB.
- Up to 64 execution units, 5GB main memory per execution unit.
- I/O to 20,000 IOPS.
- Connections are limited depending on instance size.
  - But the connection limits are probably fine.
- Retention period of backups is up to 35 days.

Azure: Interface and Controls.

- Comprehensive API.
  - Terraform and Ansible support basic, but usable.
- The GUI is modern and generally well-laid-out.
- *IMHO, you do need to navigate around a lot more to different services than with RDS to complete provisioning.*

Azure: Configuration.

- Configurable with a typical web interface.
- UI is friendly (on/off buttons, enum dropdowns, etc.).
- Still doesn't support PostgreSQL-style units ("8GB").
- Includes parameter descriptions, in a slightly glitchy display.
- Many parameters are surprisingly not changeable (shared_buffers, checkpoint_timeout, etc.).
    - Site suggest you do a fan vote on the support forum to get them supported.

Azure: Access Control.

- Combination of firewall and pg_hba.conf.
  - pg_hba.conf is confusingly called a “firewall” in the documentation.
- Until very recently, could only have a public IP (although with comprehensive firewalling).
- Private IP endpoints are in preview.
  - The setup and management of them is somewhat arcane.

Azure: Monitoring.

- A very complete set of graphs and alerts within the application.
- A proprietary query-analysis tool that seems reasonable enough.
- A “performance recommendations” tool that offers tuning suggestions (mostly trivial, but often useful and at least harmless).

Azure: Backups.

- Backups happen automatically without configuration.
- Internal tooling for backups.
  - Includes incremental backups, and snapshots for large volumes.
- Can do PITR with 5 minute granularity.

Azure: Upgrades.

- pg_dump.
- Really.

Azure: High Availability and Replicas.

- HA is done automatically and does not need to be configured.
  - On node failure, storage volume is attached to a new instance, and standard crash-recovery handles inconsistency.
- Failover is IP based; all traffic runs through a front-end gateway that routes to current node.
- Can spin up replicas from the GUI/CLI/API, and promote them to primaries.
  - Can be in a different location than the primary.

Azure: Logging.

- Slightly better than RDS' interface, which is not saying much.
- Log format and rotation are not configurable. Keeps up to seven days of logs.
- Logs are stored in instance storage, up to 1GB worth.
- Can feed logs into Azure's general logging infrastructure for more analysis and retention.

Azure: Quirks and Goodies.

- Provides HA without special charge or configuration. Thanks!
- A lot of detail and control, but this can mean a lot of "to create this, first create that, no first create this thing, then create that…" to do relatively simple tasks.
- Lots of restrictions on parameter settings.
  - Not sure about the fan-vote thing to get new ones adopted.
- "This is in preview" pops up a lot.
- No logical replication in or out.
- Without creating a private IP address, traffic runs over the public internet, not Azure's backbone (apparently).

Google Cloud SQL for PostgreSQL

“Google.”

- Not to be left behind…
- Introduced (for PostgreSQL) in 2019.
- Still very new.
- Part of the general Google Compute Cloud environment, which is pretty nice.

Google: How Much?

- Database sizes up to 30TB (!).
- Up to 64 execution units, 416GB main memory.
- I/O to 30,000 write IOPS, 100,000 read; automatic depending on storage type.
- You can pick a particular number of cores and amount of memory independently.

Google: Interface and Controls.

- Comprehensive API.
  - Terraform and Ansible support good.
- The GUI is modern and generally well-laid-out.
- *IMHO, the best of the web GUIs for the various services.*

Google: Configuration.

- First, for some reason Google calls them “flags” instead of “parameters.”
- Go to Edit, and then select a searchable drop down with all of the editable parameters in it. Yes, a drop down.
  - At least it is easy to see which ones you’ve overridden.
- The usual Mystery Units problem for numeric values.
- At least we get yes/no for booleans.
- Many parameters missing (shared_buffers) and some in “beta,” whatever that means for a parameter (work_mem? really?).

Google: Access Control.

- If the instance is not in a VPC, you get a public IP address automatically.
  - You have to whitelist public IPs.
- Otherwise, you have to create a separate public IP and assign it to the instance.
  - (Google firewalling is ***very*** strict, even within VPCs.)
- No pg_hba.conf; firewall is where it's at.

Google: Monitoring.

- Really just an OK set of monitoring tools.
- This is an area that badly needs work.

Google: Backups.

- Scheduled and manual backups.
- Backups appear to be disk-image snapshots, but…
- ***No PITR.***
  - YMMV, but this is a show-stopper for us.

Google: Upgrades.

- pg_dump.
- Only more complicated and fiddly.
- Really.

Google: High Availability and Replicas.

- HA is based on having a standby (not queryable) alternate node.
  - You pay for this node.
- Failover is done by switching the IP to the new primary.
- The shared disk is moved to the new primary.
- Replicas can be created from the GUI/API/CLI.

Google: Logging.

- Really, Google? Really?
- You can download the last couple hundred entries.
- Otherwise, hope you like Stackdriver!
  - At $0.50/GiB per month.
  - To be fair, if you are fully committed to GCP, you probably *do* like (or at least have come to terms with) Stackdriver.

Google: Quirks and Goodies.

- Instance config is flexible.
- Not supported:
  - Point in time recovery. This is **very bad**.
  - CSV import/export. This is just weird.
  - JIT. Really?
  - Logical replication.
- Product still feels rough.
  - Setting checkpoint_timeout too high causes backups to stop silently.
  - “This is beta” pops up a lot.

So, which one?

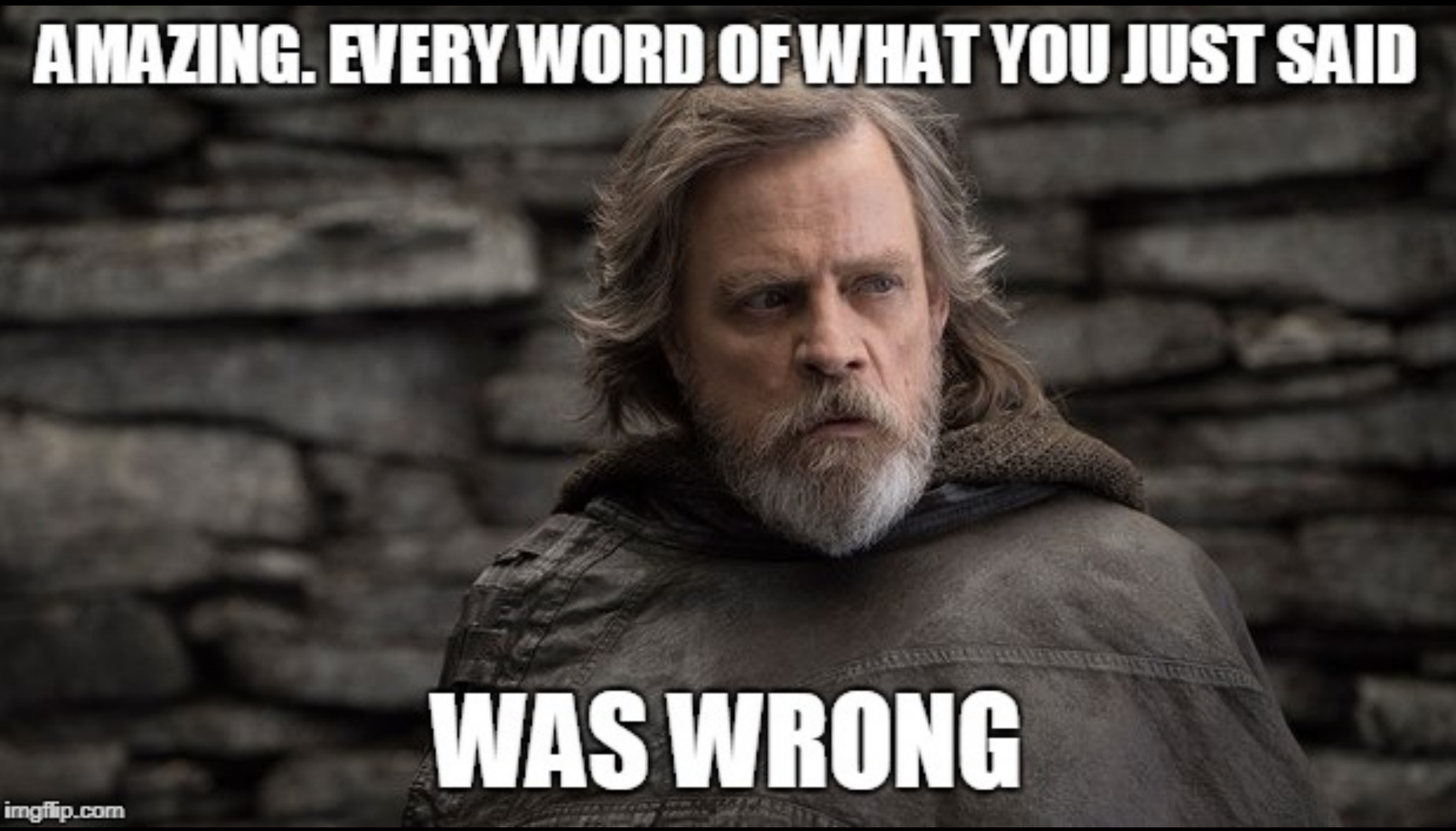
Well…

- Use the one your compute engines are in.
- If you are picking one purely on PostgreSQL functionality:
  - RDS is the most mature and "PostgreSQL-like."
  - Google still has rough edges, and the lack of PITR is daunting.
  - Azure is somewhere between them.
- They are all (especially Azure) evolving quickly.
- Of course, the big question is…

WHY?!

Why use a hosted solution?

- “You can scale out indefinitely.”
- “You never have to worry about backups.”
- “We take care of the database management for you.”
- “We provide 99.99999999999999999999…% uptime.”
- “Great Amazon Prime playlist. Pity if something ***happened*** to it.”


AMAZING. EVERY WORD OF WHAT YOU JUST SAID
WAS WRONG
imgflip.com

You still have to…

- … tune the database engine.
- … tune your queries.
- … set up, configure, and provide HA for pooling (except Heroku).
- … monitor and respond to resource issues.
- … process logs and look for errors, warnings, problematic queries.
- … design your schema.
- … confirm your backup and disaster recovery strategy.
- … do capacity planning.
- Hosted solutions handle 20% of the problem.
- You have to handle the other 80%.

The typical support experience.

“Our database has caught fire.”

“Hello, I am here to help you. I understand your database is on fire. Here is a link to an article about tuning autovacuum.”

“Hello, I am here to help you. I understand your database is on fire. Here is a link to an article about tuning autovacuum.”

did this help: yes/no

Over 50% of our clients are on a hosted solution.

WHY?!

Two Things.

The Two Things.

- Failover orchestration.
- Infrastructure-as-code support.
- These are not trivial!

Failover Orchestration.

- Getting all the moving pieces of proper failover working right is hard.
  - Detect and terminate the failed machine.
  - Pick the failover candidate.
  - Promote the candidate and reassign the endpoint.
  - Attach the secondaries.
  - Reprovision the failed instance.
  - Handle errors, split-brain, etc., etc.
- This is not simple on community PostgreSQL.

Infrastructure-as-Code

- Hosted database instances are a single resource.
- (Reasonably) easy to spin up and configure using Terraform, etc.
- PostgreSQL servers running on VMs are complex services.
- Requires lots of fiddly Ansible or the like to set up, configure, attach replicas, etc., etc.
- Infrastructure-as-Code is a highly desirable goal!

But you lose…

- Insight into instance performance.
- Flexibility in configuration (high-speed local disks, etc.).
- True `postgres` superuser (you don't miss it until it's gone).
- Extensions (pg_partman is a notable causality).
- Most PLs (PL/PythonU, etc.).
- Staying up-to-date on versions.

On community PostgreSQL…

- You can come close to a hosted solution.
- Use Patroni to manage your cluster.
- Use pgBackRest or Barman for backups.
- Use Terraform/Ansible for configuration management/distribution.
- Use whatever compute cloud you like, or even have a hybrid!
- Requires a non-trivial amount of setup and tooling.
- But this is non-recurring engineering, compared to the Hosted PostgreSQL tax.
- And, really, do we need another web GUI?

In conclusion…

- The hosted solutions solve important problems, but a very small range of them.
- **All the big problems are still up to you.**
- Hosted solutions are very handy for a quick-start database solution.
- But! Self-hosting is a completely viable solution; don’t assume that you must use a hosted solution to have a reliable database.

Thank you!

Questions?

Christophe Pettus

CEO, PostgreSQL Experts, Inc.

christophe.pettus@pgexperts.com

thebuild.com

twitter @xof

PGX

POSTGRESQL
EXPERTS, INC.

pgexperts.com