How (and why) Datadog is moving from cloud-managed to self-managed Postgres

Nick Canzoneri

pgconf.nyc

2025-09-30

Agenda

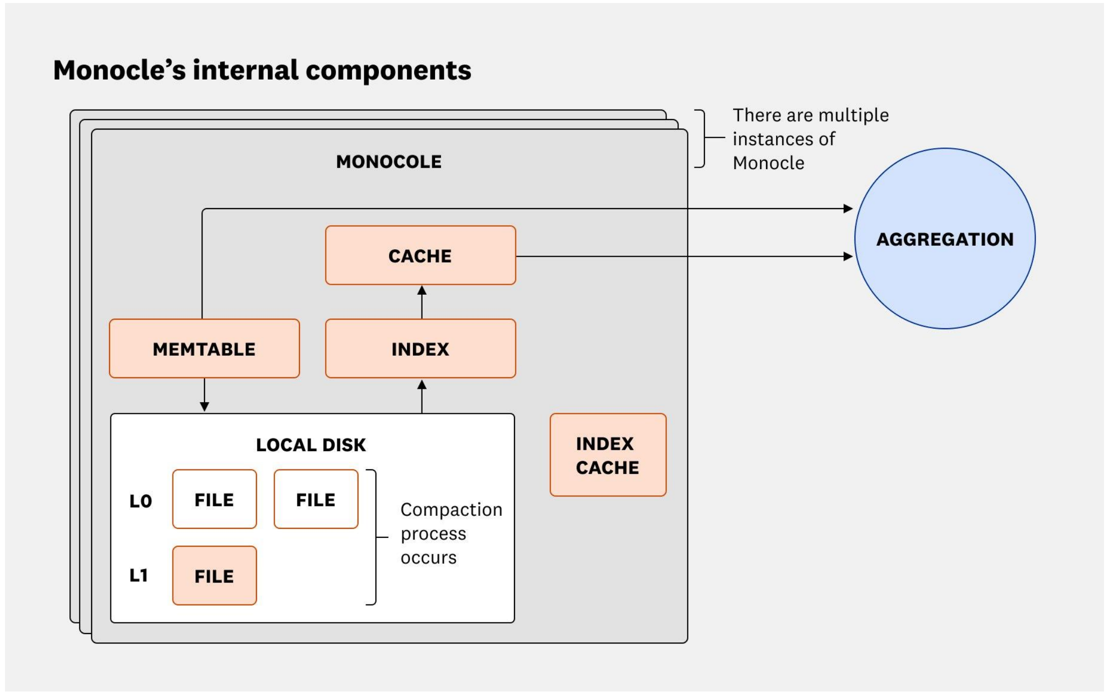
How Datadog uses Postgres

- What kind of data does Datadog store? Lots.
  - A small percentage of bytes are in Postgres, but those bytes are still important!

How Datadog uses Postgres

- Metrics timeseries data - not in Postgres

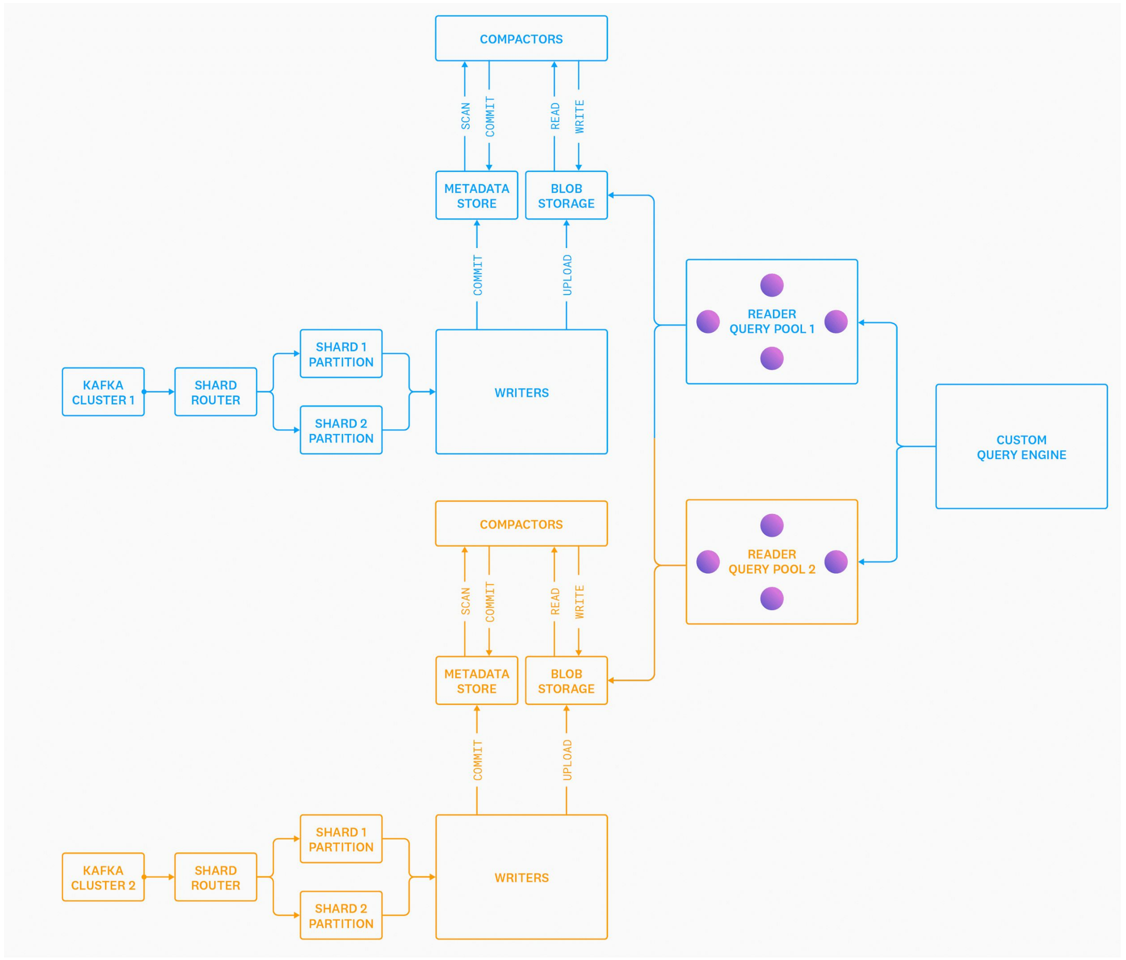
Introducing Monocle: a high-performance timeseries storage engine


Monocle's internal components
There are multiple instances of Monocle
MONOCOLE
AGGREGATION
CACHE
MEMTABLE
INDEX
LOCAL DISK
INDEX CACHE
L0
FILE
FILE
L1
FILE
Compaction process occurs

How Datadog uses Postgres

- Logs / events storage - not in Postgres

Introducing Husky, Datadog's third-generation event store



How Datadog uses Postgres

- Specialized tools for specialized jobs

real-time timeseries database purpose-built in Rust for high throughput and low latency

How Datadog uses Postgres

- Specialized tools for specialized jobs

real-time timeseries database purpose-built in Rust for high throughput and low latency

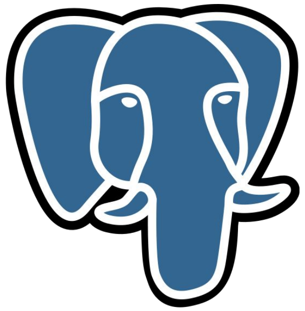
columnar store optimized for streaming ingestion and hybrid analytical and search queries

How Datadog uses Postgres

- Specialized tools for specialized jobs

real-time timeseries database purpose-built in Rust for high throughput and low latency

columnar store optimized for streaming ingestion and hybrid analytical and search queries



I have some data to store in a table…

How Datadog uses Postgres

I have some data to store in a table...

users

dashboards

monitors

vectors

Temporal backend

incidents

hosts

notebooks

internal control planes

How Datadog has hosted Postgres

US1

- Chef-configured EC2 instances for everything, including Postgres (RDS for Postgres didn’t exist!)

How Datadog has hosted Postgres

US1

- Chef-configured EC2 instances for everything, including Postgres

EU1, US3, US5

- Using new-to-Datadog cloud providers GCP and Azure
- *Not* using Chef to configure VMs, using Kubernetes
- Use the cloud-managed services in order to run Postgres, GCP CloudSQL and Azure Single Server

How Datadog has hosted Postgres

US1

- Chef-configured EC2 instances for ~~everything, including~~ Postgres
- Using Kubernetes for almost all workloads

EU1, US3, US5

- Using new-to-Datadog cloud providers GCP and Azure
- *Not* using Chef to configure VMs, using Kubernetes
- Use the cloud-managed services in order to run Postgres, GCP CloudSQL and Azure Single Server

How Datadog has hosted Postgres

US1

- ~~Chef configured EC2 instances for everything, including Postgres~~
- Using Kubernetes for almost all workloads
- Moved to RDS in 2022

EU1, US3, US5

- Using new-to-Datadog cloud providers GCP and Azure
- *Not* using Chef to configure VMs, using Kubernetes
- Use the cloud-managed services in order to run Postgres, GCP CloudSQL and Azure Single Server

How Datadog has hosted Postgres

- Making use of AWS RDS, GCP CloudSQL, and Azure Single Server all at the same time
    - All these managed services means we didn’t need any database team, right?

How Datadog has hosted Postgres

- Making use of AWS RDS, GCP CloudSQL, and Azure Single Server all at the same time
    - All these managed services means we didn't need any database team, right?

WRONG

Pros and cons of cloud-managed

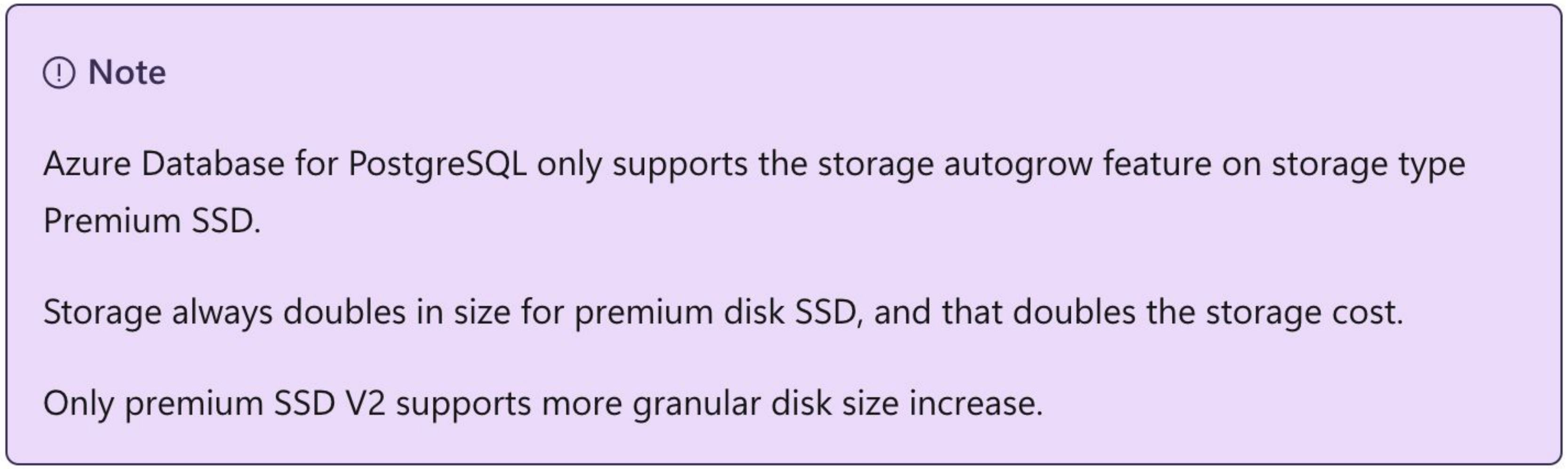
- Making use of AWS RDS, GCP CloudSQL, and Azure Single Server all at the same time

CON - learning ALL of the clouds - example: Storage Autoscaling

Pros and cons of cloud-managed

- Making use of AWS RDS, GCP CloudSQL, and Azure Single Server all at the same time

CON - learning ALL of the clouds - example: Storage Autoscaling



Note

Azure Database for PostgreSQL only supports the storage autogrow feature on storage type Premium SSD.

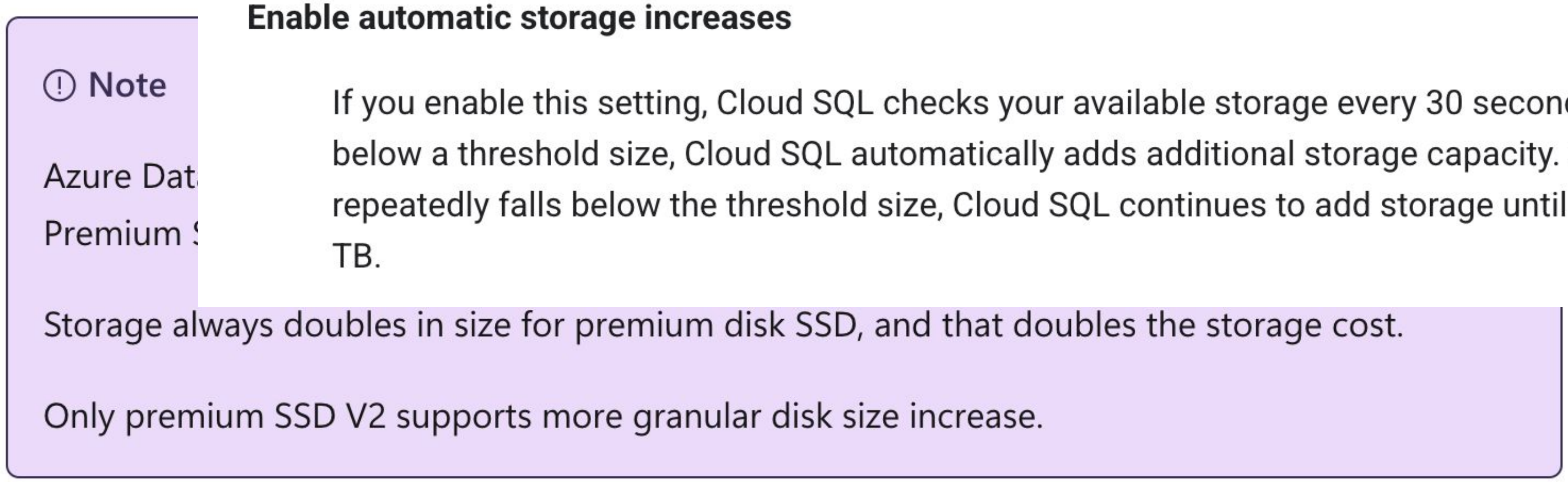
Storage always doubles in size for premium disk SSD, and that doubles the storage cost.

Only premium SSD V2 supports more granular disk size increase.

Pros and cons of cloud-managed

- Making use of AWS RDS, GCP CloudSQL, and Azure Single Server all at the same time

CON - learning ALL of the clouds - example: Storage Autoscaling

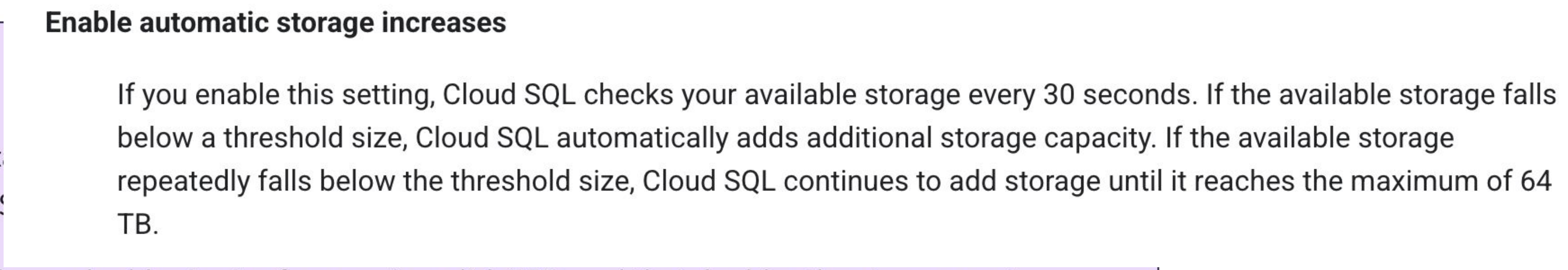

ⓘ Note

Azure Dat

Premium

Storage always doubles in size for premium disk SSD, and that doubles the storage cost.

Only premium SSD V2 supports more granular disk size increase.

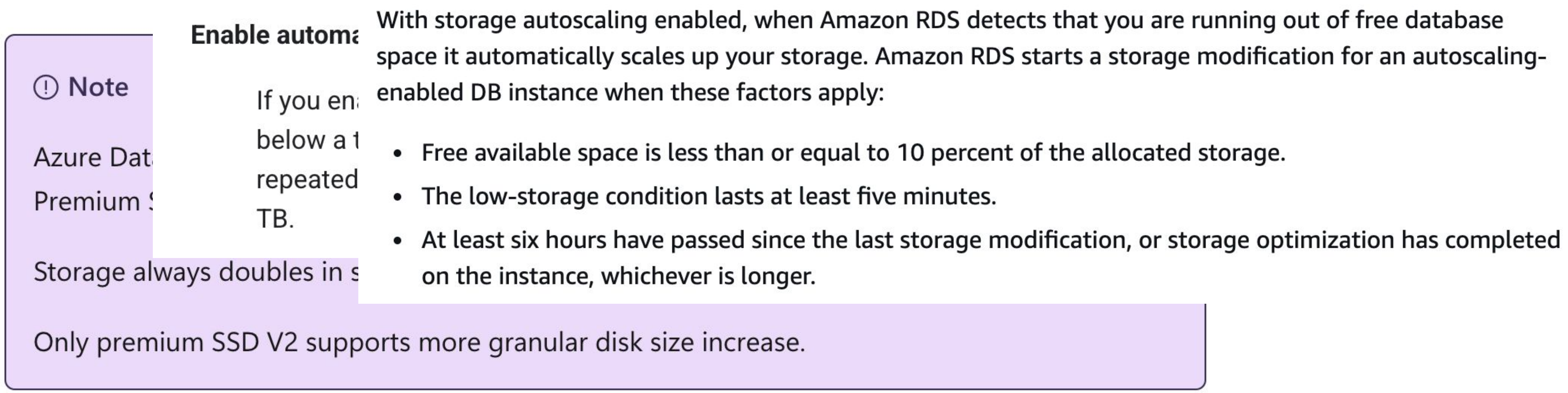

Enable automatic storage increases

If you enable this setting, Cloud SQL checks your available storage every 30 seconds. If the available storage falls below a threshold size, Cloud SQL automatically adds additional storage capacity. If the available storage repeatedly falls below the threshold size, Cloud SQL continues to add storage until it reaches the maximum of 64 TB.

Pros and cons of cloud-managed

- Making use of AWS RDS, GCP CloudSQL, and Azure Single Server all at the same time

CON - learning ALL of the clouds - example: Storage Autoscaling



ⓘ Note

Azure Dat

Premium

Storage always doubles in s

Only premium SSD V2 supports more granular disk size increase.

Enable automa

If you ena

below a t

repeated

TB.

With storage autoscaling enabled, when Amazon RDS detects that you are running out of free database space it automatically scales up your storage. Amazon RDS starts a storage modification for an autoscaling-enabled DB instance when these factors apply:

- Free available space is less than or equal to 10 percent of the allocated storage.
- The low-storage condition lasts at least five minutes.
- At least six hours have passed since the last storage modification, or storage optimization has completed on the instance, whichever is longer.

Pros and cons of cloud-managed

- Making use of AWS RDS, GCP CloudSQL, and Azure Single Server all at the same time

CON - Cloud provider APIs and concepts all vary greatly - Automation needs to be written 3 times

Pros and cons of cloud-managed

- Making use of AWS RDS, GCP CloudSQL, and Azure Single Server all at the same time

CON - When you *need* consistency, like support for new extensions like pgvector, across all the providers

Pros and cons of cloud-managed

- Making use of AWS RDS, GCP CloudSQL, and Azure Single Server all at the same time

PRO - the patching, maintenance, and backups are generally excellent

Pros and cons of cloud-managed

- Making use of AWS RDS, GCP CloudSQL, and Azure Single Server all at the same time

PRO - Availability/reliability is *generally* very good

Pros and cons of cloud-managed

- Making use of AWS RDS, GCP CloudSQL, and Azure Single Server all at the same time

NEUTRAL - Support - generally responsive and helpful, but most of the time only needed to contact when something wrong *inside* the black box

Pros and cons of cloud-managed

- Making use of AWS RDS, GCP CloudSQL, and Azure Single Server all at the same time

You get the foibles of **ALL** the providers, but the only get to take advantage of the benefits if all of the providers implement it

How Datadog is hosting Postgres

- Multi cloud-managed databases wasn't sustainable to support our growth
- We can run it ourselves, on Kubernetes!

How Datadog is hosting Postgres

Why Postgres on Kubernetes?

- Control of our own destiny
- Uniform stack - database works the same everywhere (even if we miss out on the best)

Reasons *not* to run Postgres on Kubernetes?

- Cost (without re-architecting databases) is **not** cheaper
- Increased complexity

How Datadog is hosting Postgres

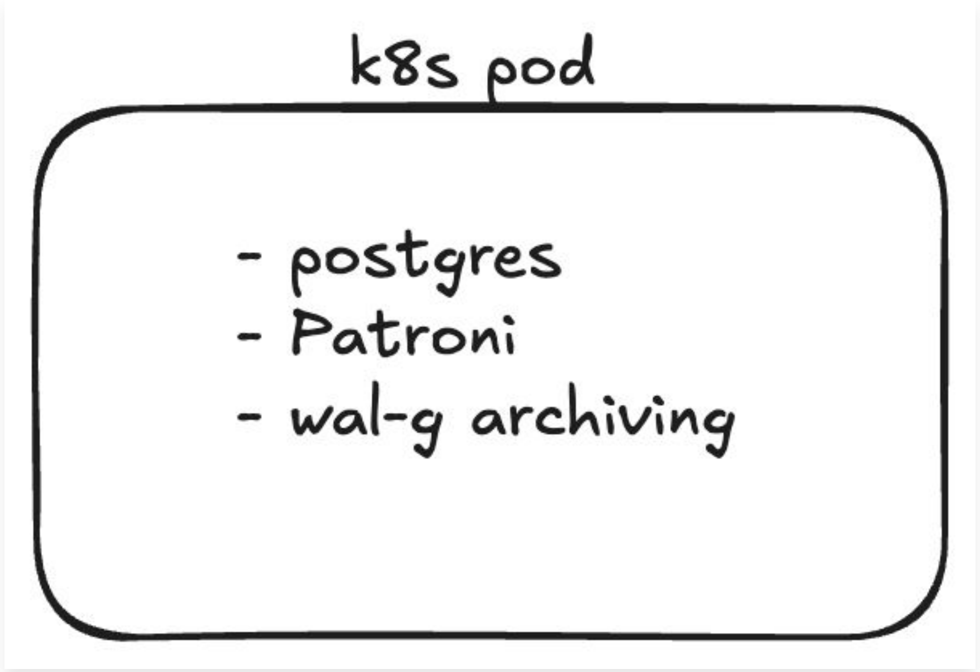
What does our Postgres on Kubernetes set up look like?

Open source components:

- Postgres
- Patroni
- Wal-g
- Just a helm chart, not an operator
    - Higher level automation is achieved through Temporal workflows

How Datadog is hosting Postgres

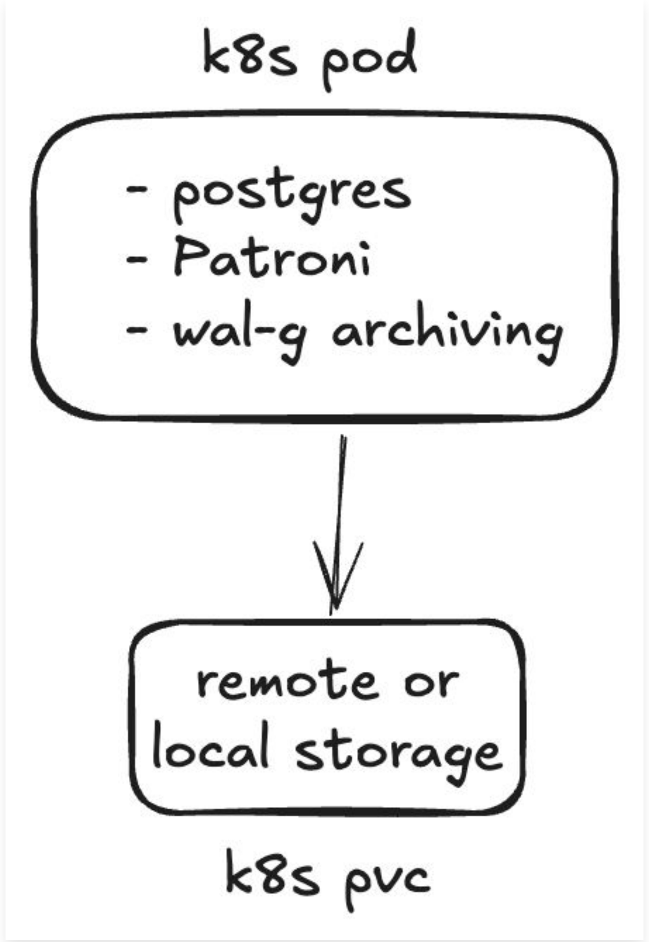
Start with a Kubernetes pod



How Datadog is hosting Postgres

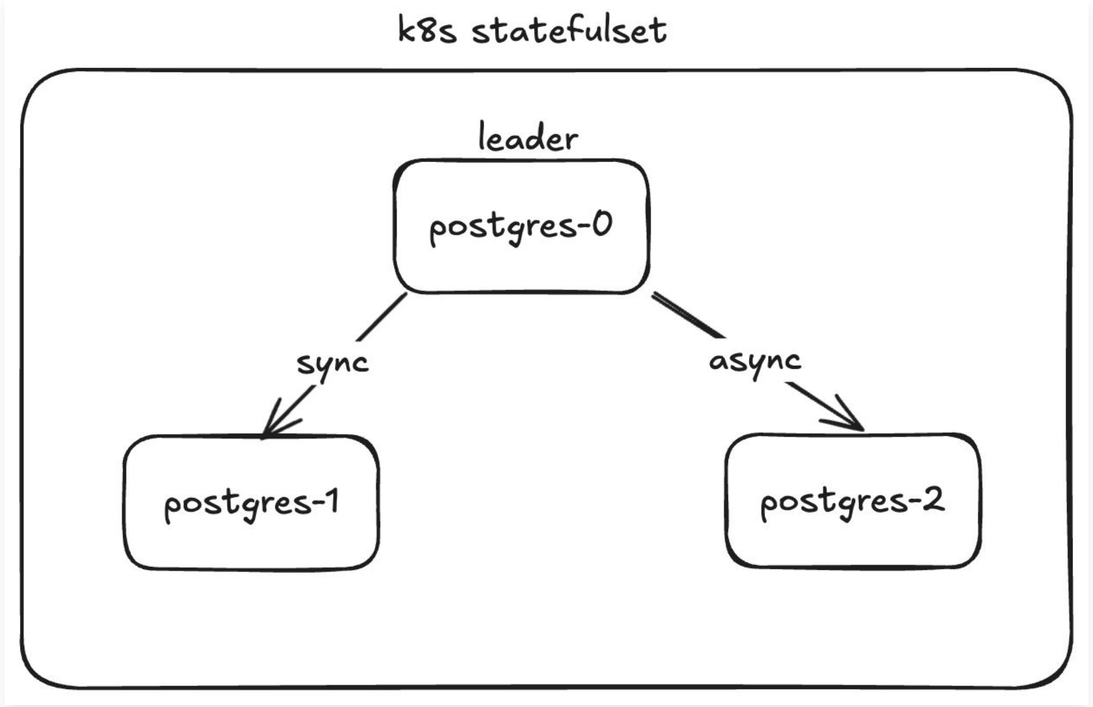
Every pod gets a PVC

- Default to remote storage
  - Scale compute independently
  - Easy resizing
  - attach/detach when replacing hosts
- Local NVME instances when needed
  - High I/O use cases
  - Cheaper at the extremes



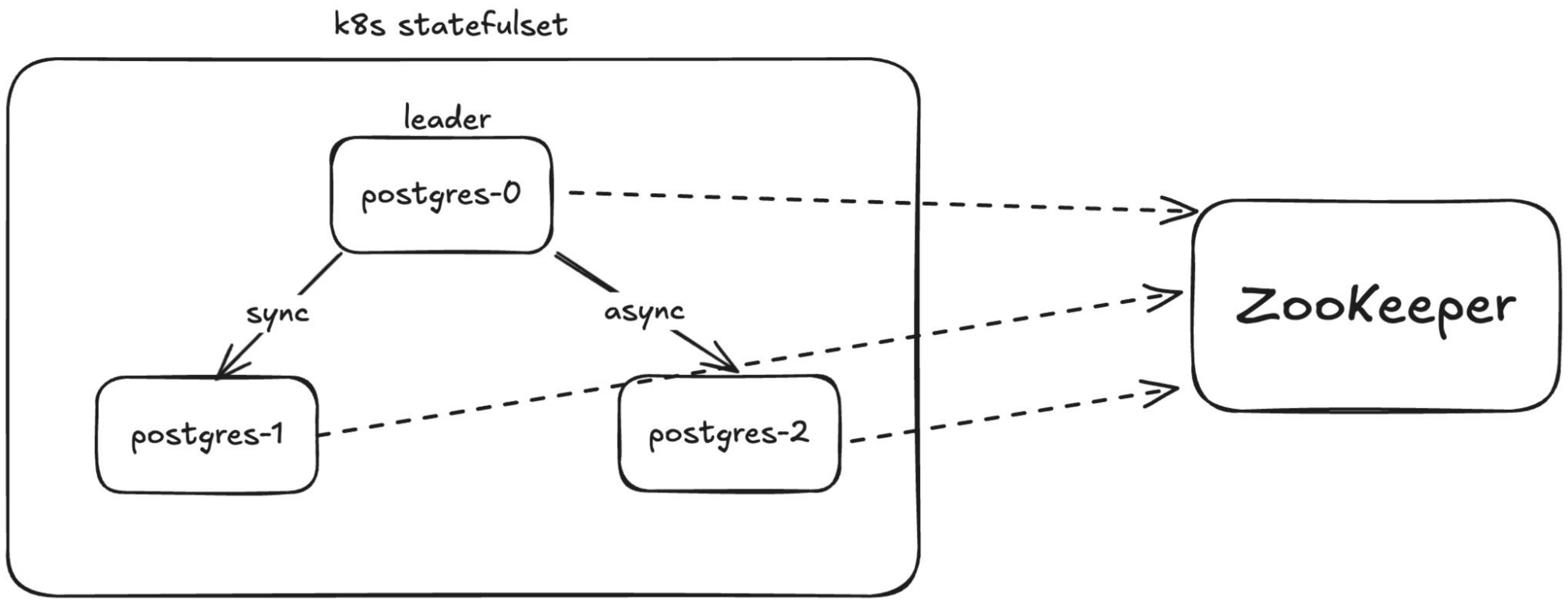
How Datadog is hosting Postgres

Need high-availability, so we add 2 more pods



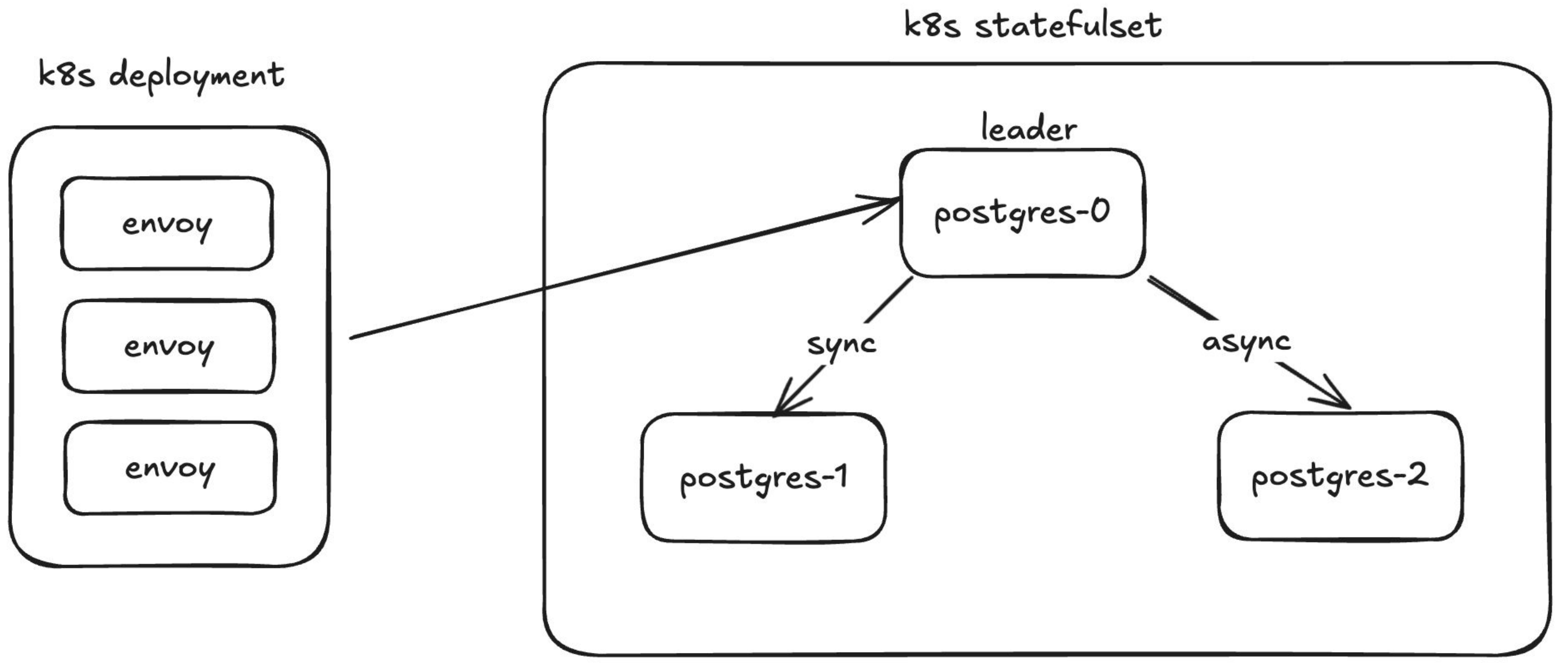
How Datadog is hosting Postgres

Patroni uses ZooKeeper for leader election and configuration



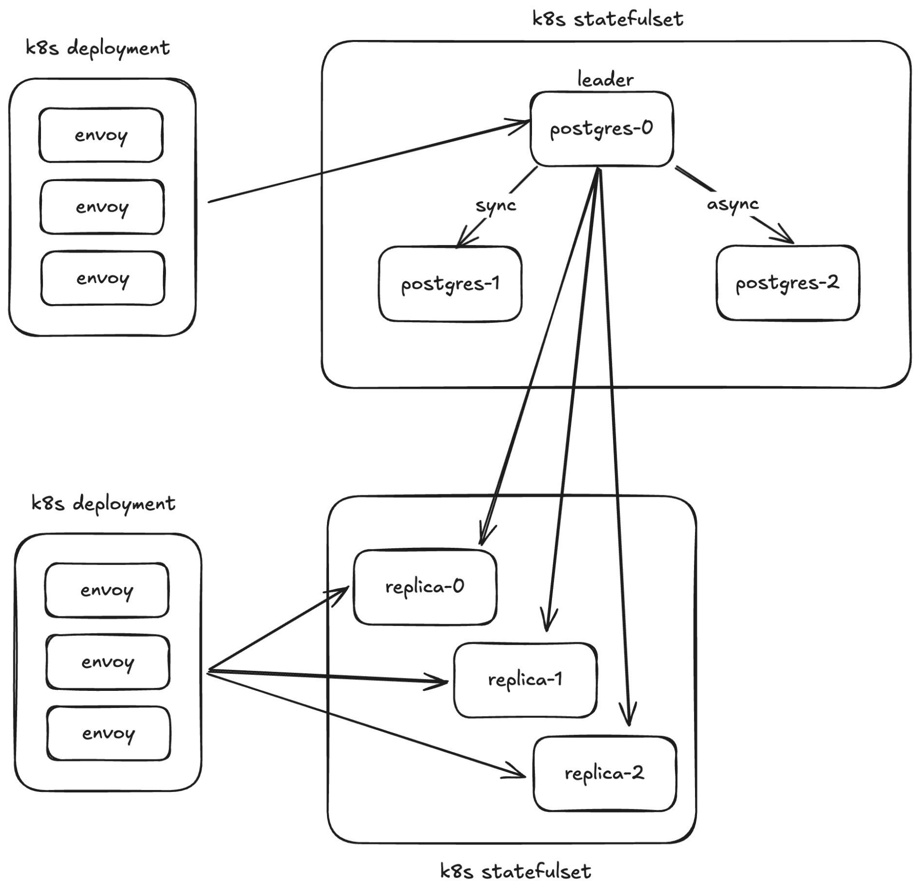
How Datadog is hosting Postgres

A simple envoy proxy deployment will redirect traffic to the current Postgres leader using Patroni apis



How Datadog is hosting Postgres

Optionally, we can also set up dedicated read replica pools for high traffic clusters



Migrating a datacenter off of cloud-managed

We can run Postgres on Kubernetes, but we have a large existing footprint of databases in each datacenter

How do we prioritize what to migrate to Postgres on Kubernetes?

Migrating a datacenter off of cloud-managed

`:azurechaos:`

Migrating a datacenter off of cloud-managed

`:azurechaos:`

(there is no :awschaos: or :gcpchaos:)

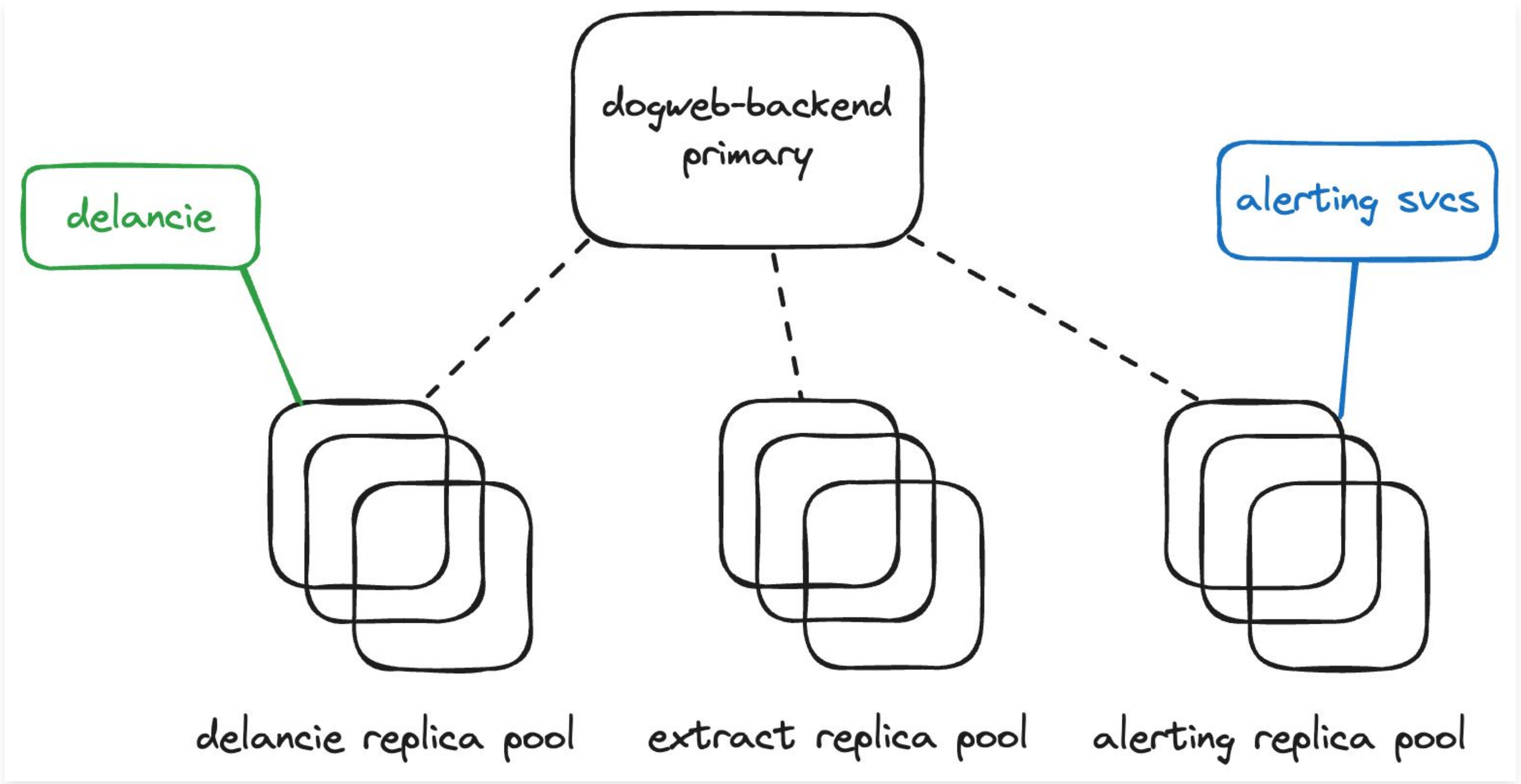
Migrating a datacenter off of cloud-managed

Postgres limitations for Azure Single Server (which is now “retired” 💀)

- Severe limitations on scaling and reliability due to a limit of 5 replicas

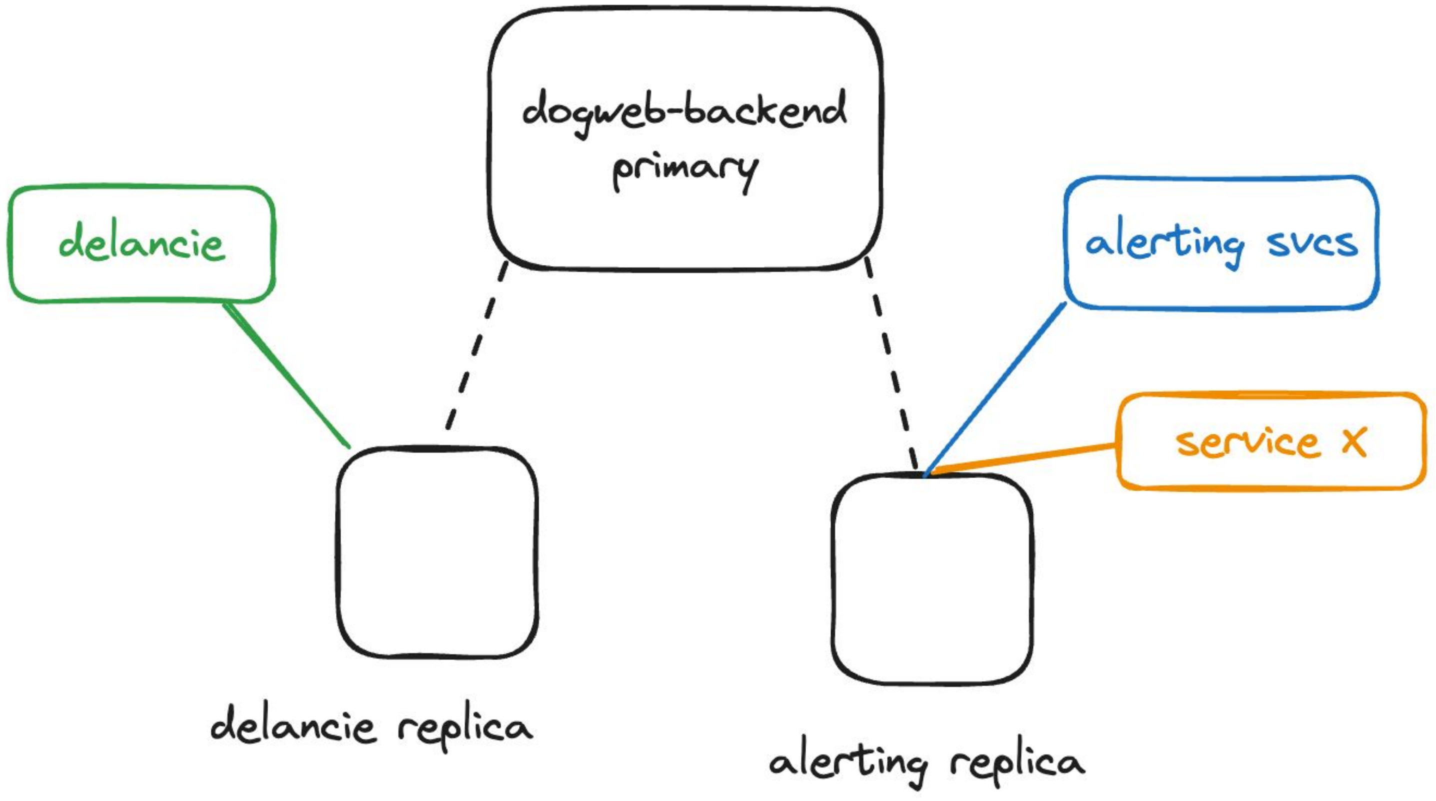
Migrating a datacenter off of cloud-managed

How we use replica pools



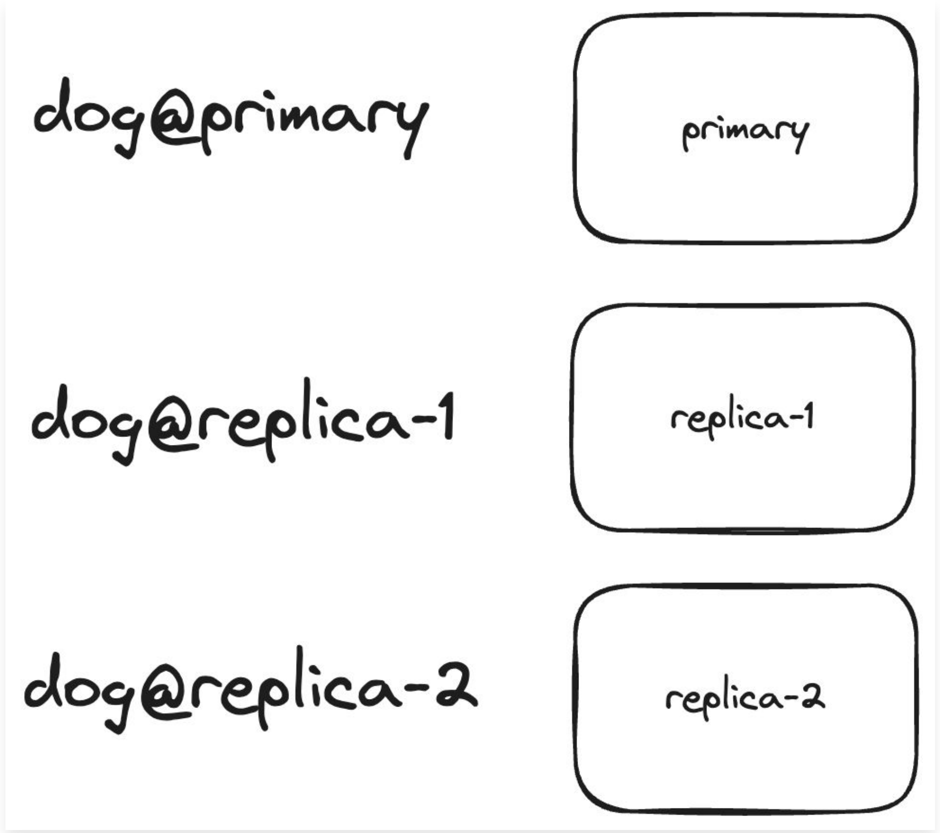
Migrating a datacenter off of cloud-managed

5 replica limit in Azure Single Server

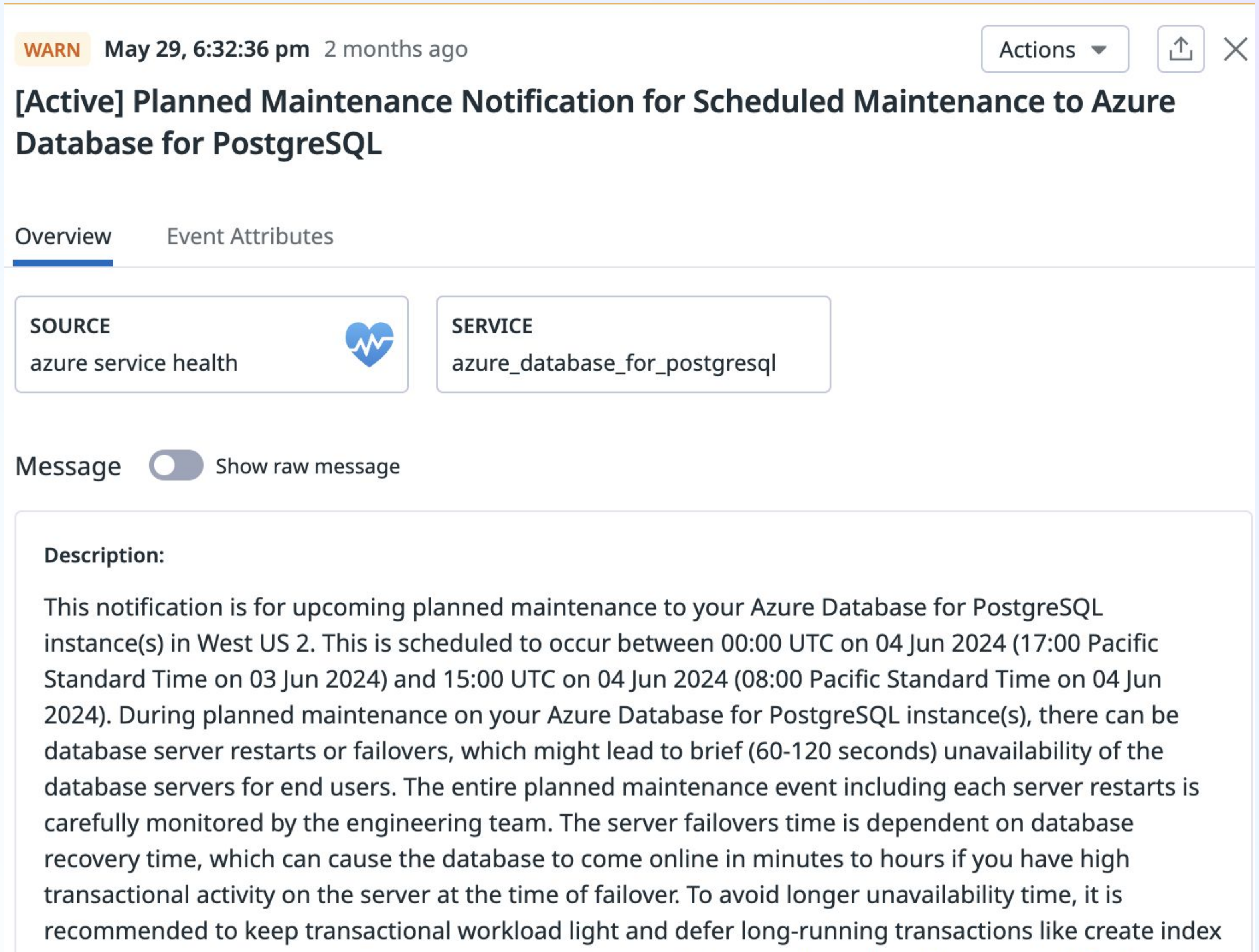


Migrating a datacenter off of cloud-managed

Host in username to connect - Very hard to do traffic shifting



Migrating a datacenter off of cloud-managed



May 29, 6:32:36 pm 2 months ago

[Active] Planned Maintenance Notification for Scheduled Maintenance to Azure Database for PostgreSQL

SOURCE
azure service health

SERVICE
azure_database_for_postgresql

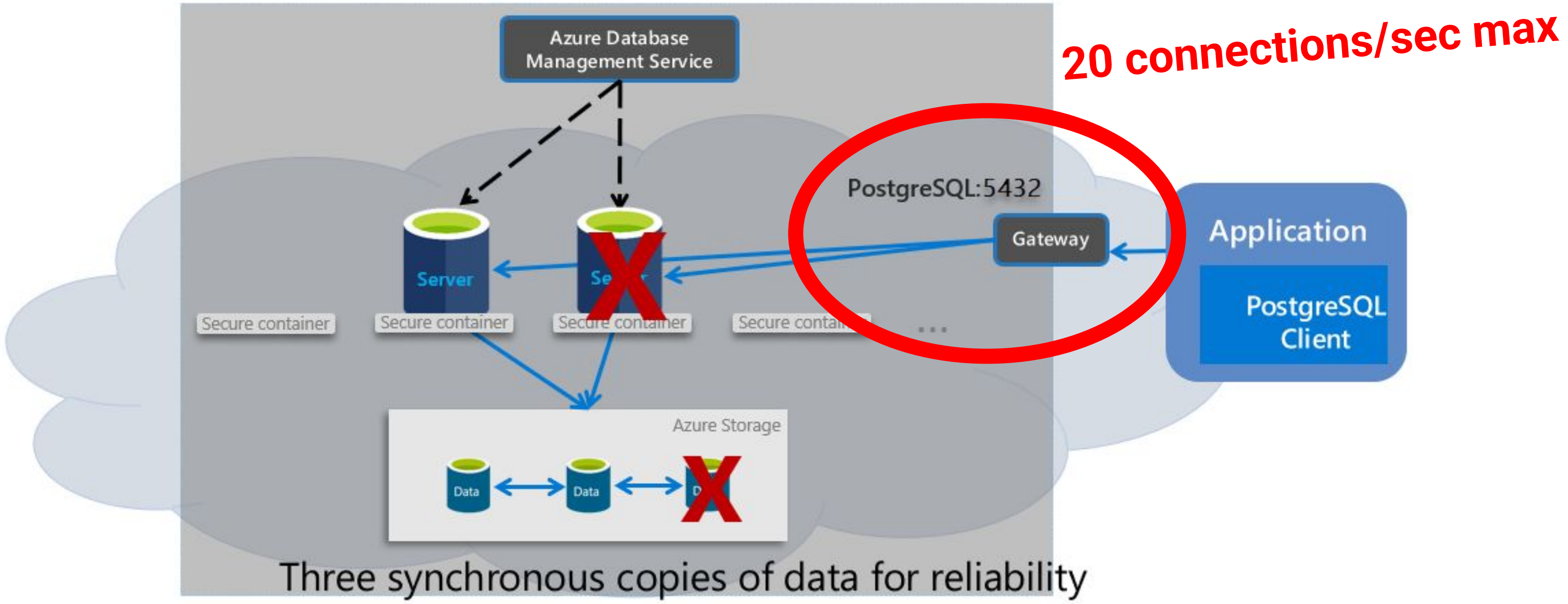
Message

Description:

This notification is for upcoming planned maintenance to your Azure Database for PostgreSQL instance(s) in West US 2. This is scheduled to occur between 00:00 UTC on 04 Jun 2024 (17:00 Pacific Standard Time on 03 Jun 2024) and 15:00 UTC on 04 Jun 2024 (08:00 Pacific Standard Time on 04 Jun 2024). During planned maintenance on your Azure Database for PostgreSQL instance(s), there can be database server restarts or failovers, which might lead to brief (60-120 seconds) unavailability of the database servers for end users. The entire planned maintenance event including each server restarts is carefully monitored by the engineering team. The server failovers time is dependent on database recovery time, which can cause the database to come online in minutes to hours if you have high transactional activity on the server at the time of failover. To avoid longer unavailability time, it is recommended to keep transactional workload light and defer long-running transactions like create index

Migrating a datacenter off of cloud-managed

A Gateway with strict limits that we have no control over



Three synchronous copies of data for reliability

Migrating a datacenter off of cloud-managed

Single point of failure DB replicas

+

Mass restart events

+

Strict connection limits

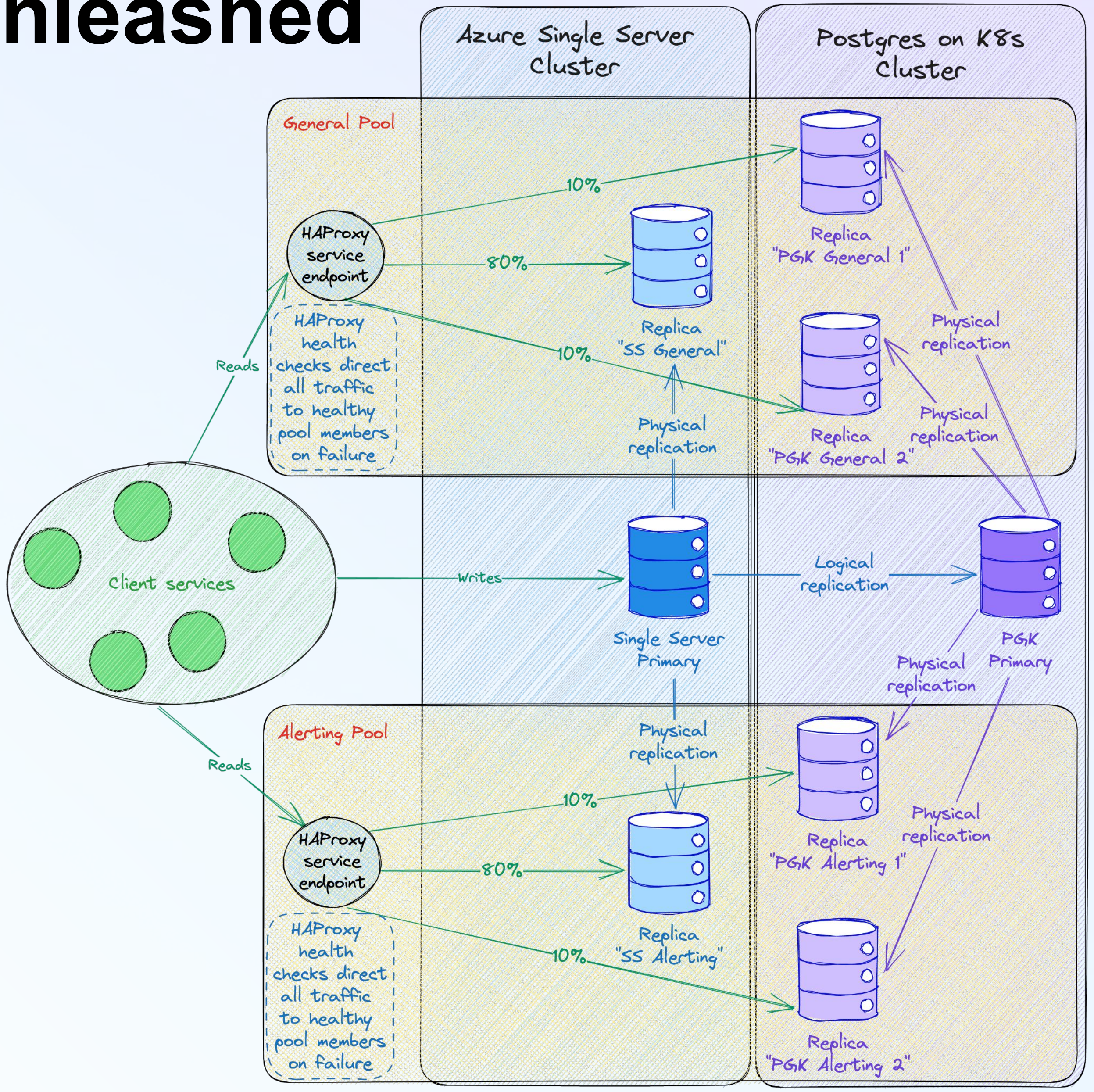
=

Lots of incidents :(

Replicas Unleashed

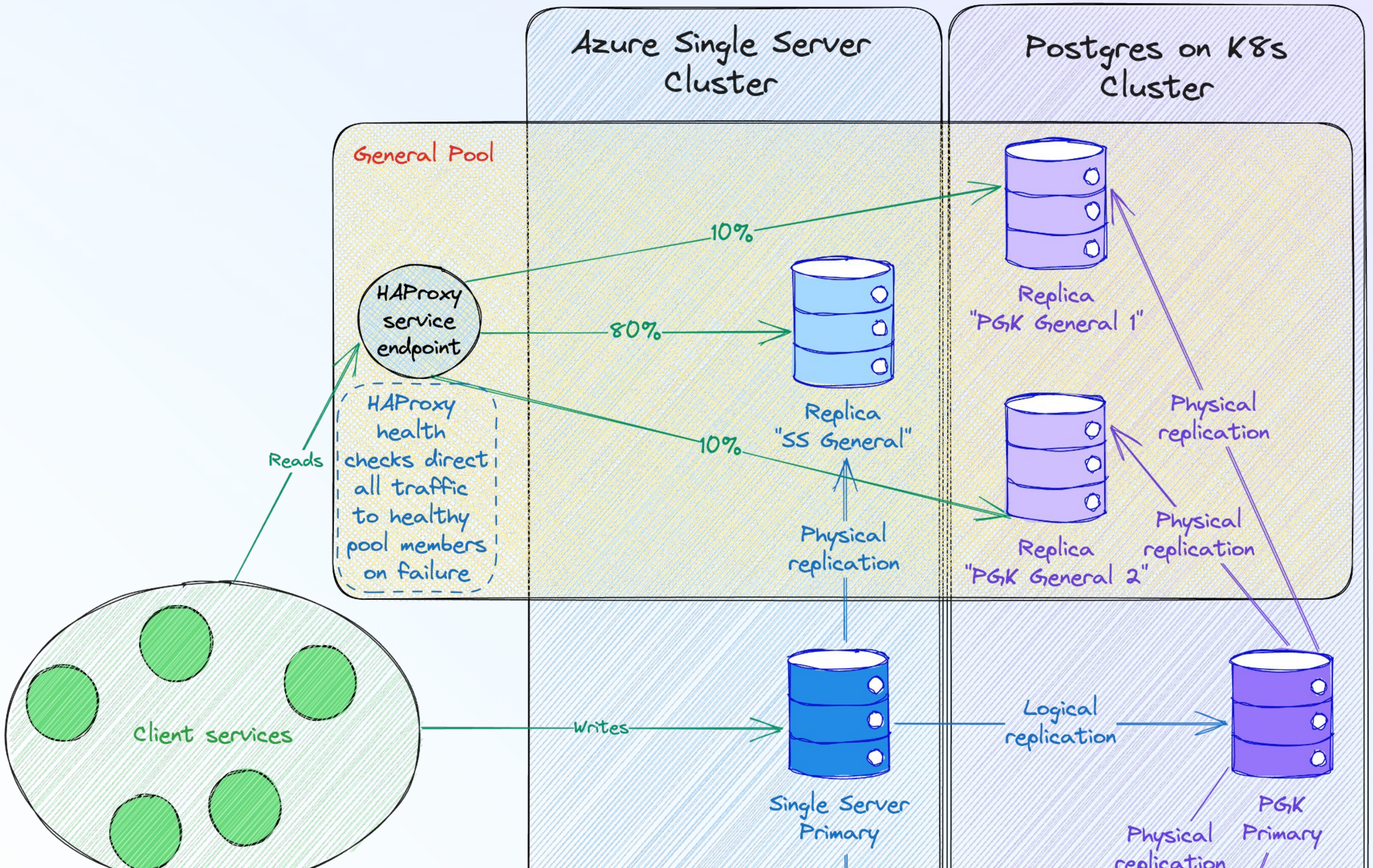
How do we begin to solve the problem?

Replicas Unleashed



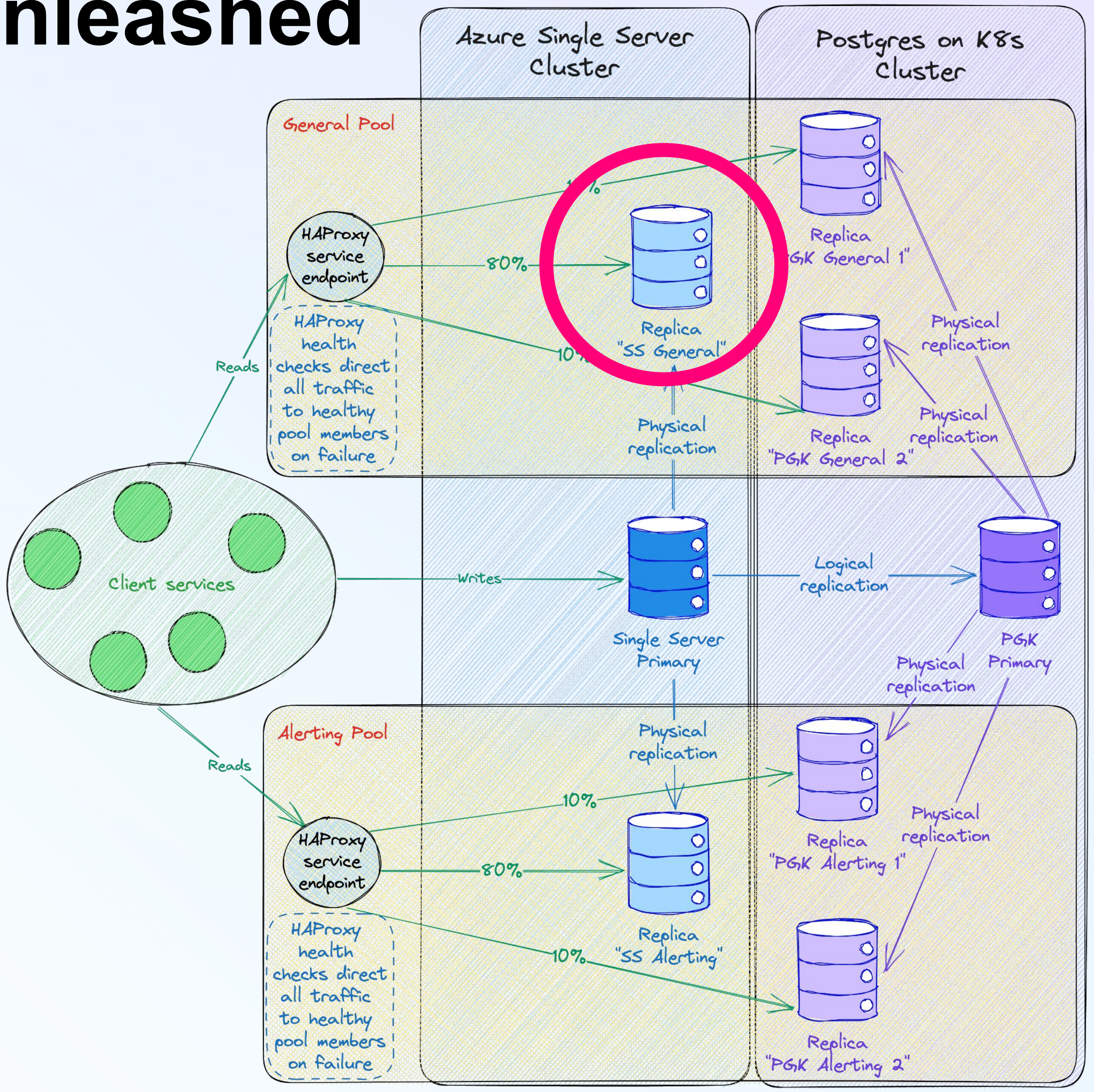
DATADOG

Replicas Unleashed



Replicas Unleashed

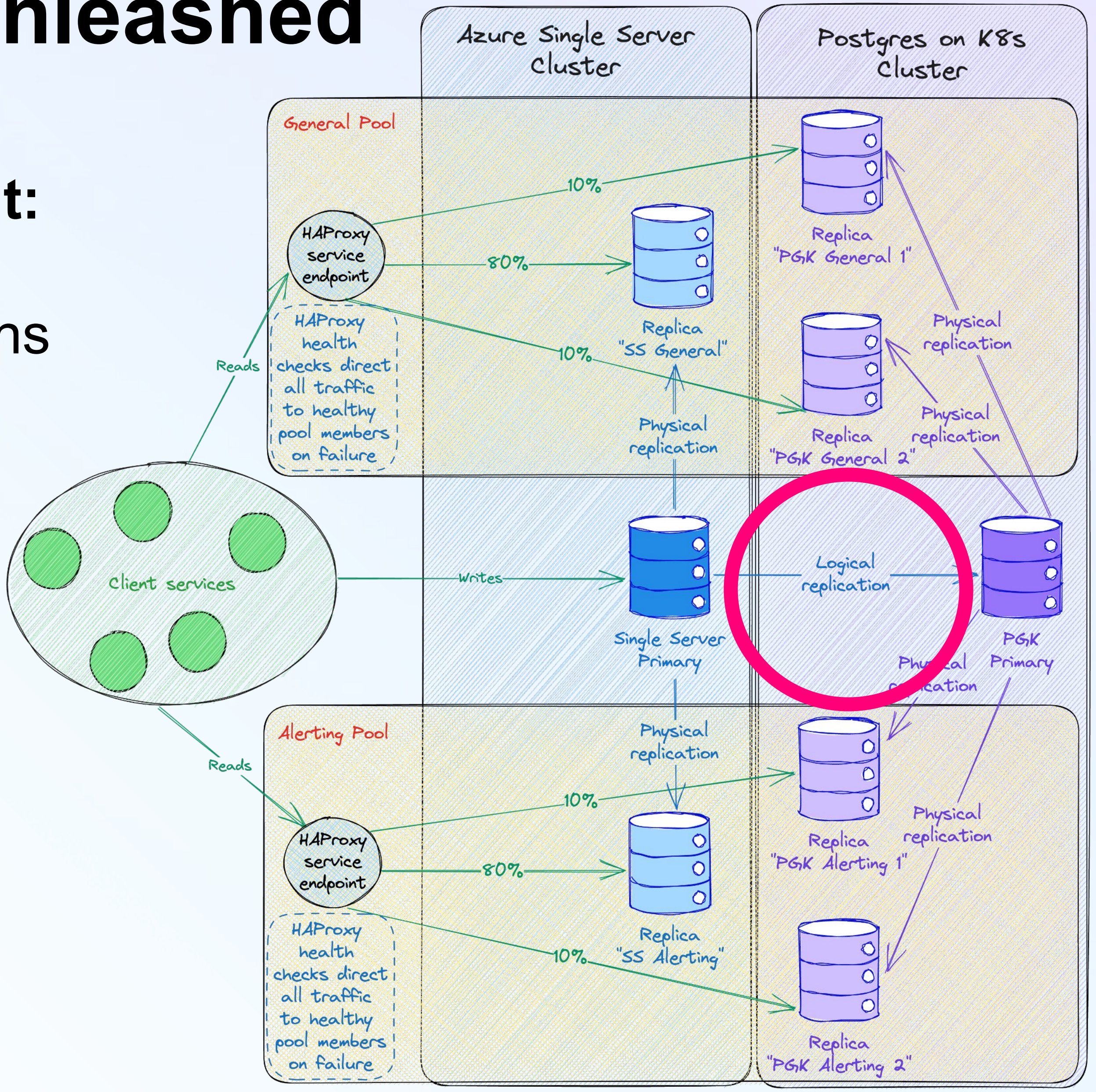
Benefits



Replicas Unleashed

New failure point:

- Migrations
- Large write txns



A recipe for zero* downtime cutovers

A recipe for zero* downtime cutovers

***: This can mean very different things to different people…**

- Zero connections broken? That's good, but tight statement timeouts don't tolerate long pauses
- Zero errors return to users? That's great! Application code needs to handle that everywhere
- **Zero SLOs broken** - keep impact within "typical" maintenance operations, like restarts, <30sec. 100% uptime is not a standard I want to be oncall for…

A recipe for zero* downtime cutovers

What you need?

- Ahead of time: A proxy/piece of infra in front of your database that you control and can update quickly to shift traffic
    - Have used pgbouncer, haproxy, DNS record, envoy, our custom pg-proxy

A recipe for zero* downtime cutovers

What you need?

- Ahead of time: Logical replication
  - Tables need replica identities
  - Ideally, no DDL operations
  - For large databases with small write volume, can start from a backup and use `pg_replication_slot_advance` from when the backup was taken
  - Doesn't trigger autoanalyze!

A recipe for zero* downtime cutovers

What you need?

- Cutover: block traffic to the source
  - pgbouncer pause
  - `ALTER ROLE NOLOGIN` + kill connections

A recipe for zero* downtime cutovers

What you need?

- Cutover: Making sure data is synced
  - Check flush_lsn from pg_replication_slots
  - Use a “tracer” table
  - Need to sync sequence values! Some dbs have 10s of thousands…

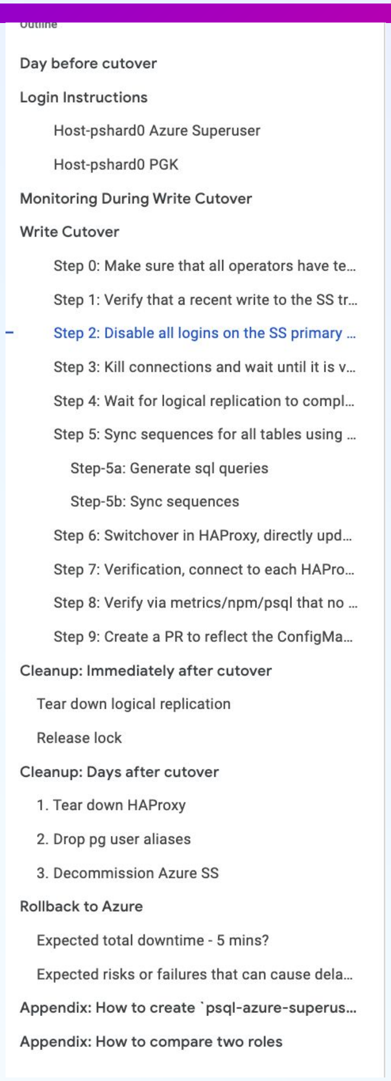
    -
      ```
      SELECT format(E'SELECT format(E\\'SELECT pg_catalog.setval(\\\\\\'%1\$I.%2\$I\\\\\\',
      %%s, %%L);\\', last_value, is_called) FROM %1\$I.%2\$I;', relnamespace::regnamespace,
      relname) FROM pg_class WHERE relkind = 'S' AND relnamespace NOT IN
      ('information_schema'::regnamespace, 'pg_catalog'::regnamespace,
      'pg_toast'::regnamespace) ORDER BY relnamespace::regnamespace::text, relname
      ```

A recipe for zero* downtime cutovers

What you need?

- Cutover: Direct traffic to the target and see queries succeed!
  - Update configuration for whatever proxy/DNS you’re using
  - Use percentage based rollouts to slowly ramp up read traffic
  - Write traffic can be scary due to the “big bang” aspect

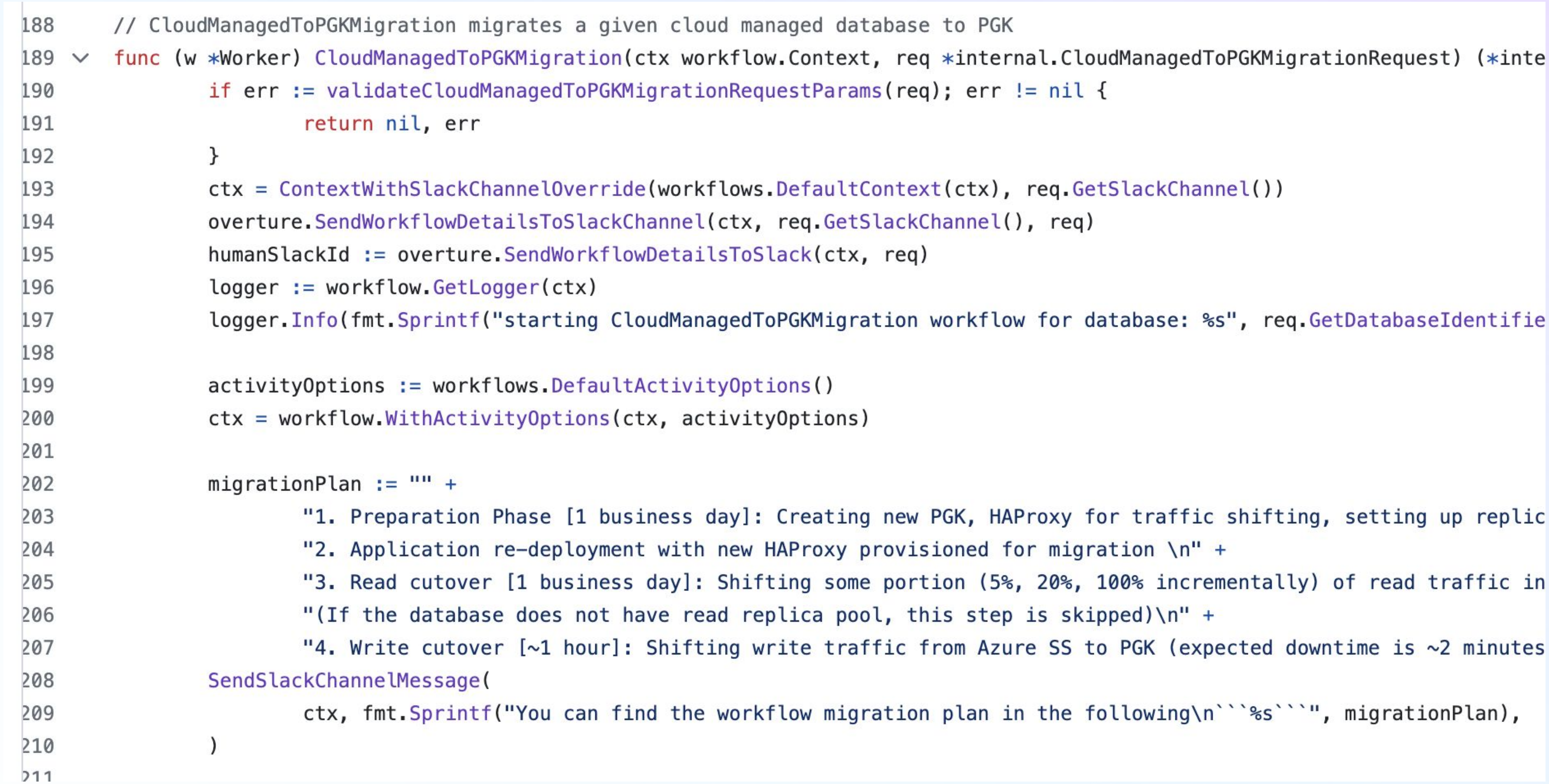
A recipe for zero* downtime cutovers



Outline

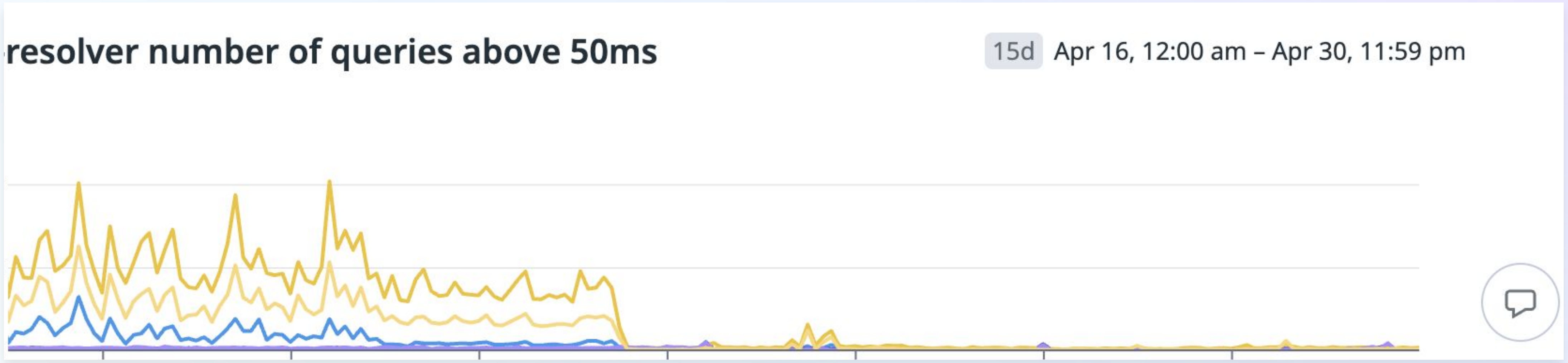
- Day before cutover
- Login Instructions
  - Host-pshard0 Azure Superuser
  - Host-pshard0 PGK
- Monitoring During Write Cutover
- Write Cutover
  - Step 0: Make sure that all operators have te...
  - Step 1: Verify that a recent write to the SS tr...
  - Step 2: Disable all logins on the SS primary ...
  - Step 3: Kill connections and wait until it is v...
  - Step 4: Wait for logical replication to compl...
  - Step 5: Sync sequences for all tables using ...
    - Step-5a: Generate sql queries
    - Step-5b: Sync sequences
  - Step 6: Switchover in HAProxy, directly upd...
  - Step 7: Verification, connect to each HAPro...
  - Step 8: Verify via metrics/npm/psql that no ...
  - Step 9: Create a PR to reflect the ConfigMa...
- Cleanup: Immediately after cutover
  - Tear down logical replication
  - Release lock
- Cleanup: Days after cutover
  - 1. Tear down HAProxy
  - 2. Drop pg user aliases
  - 3. Decommission Azure SS
- Rollback to Azure
  - Expected total downtime - 5 mins?
  - Expected risks or failures that can cause dela...
- Appendix: How to create `psql-azure-superus...
- Appendix: How to compare two roles

A recipe for zero* downtime cutoversPGK



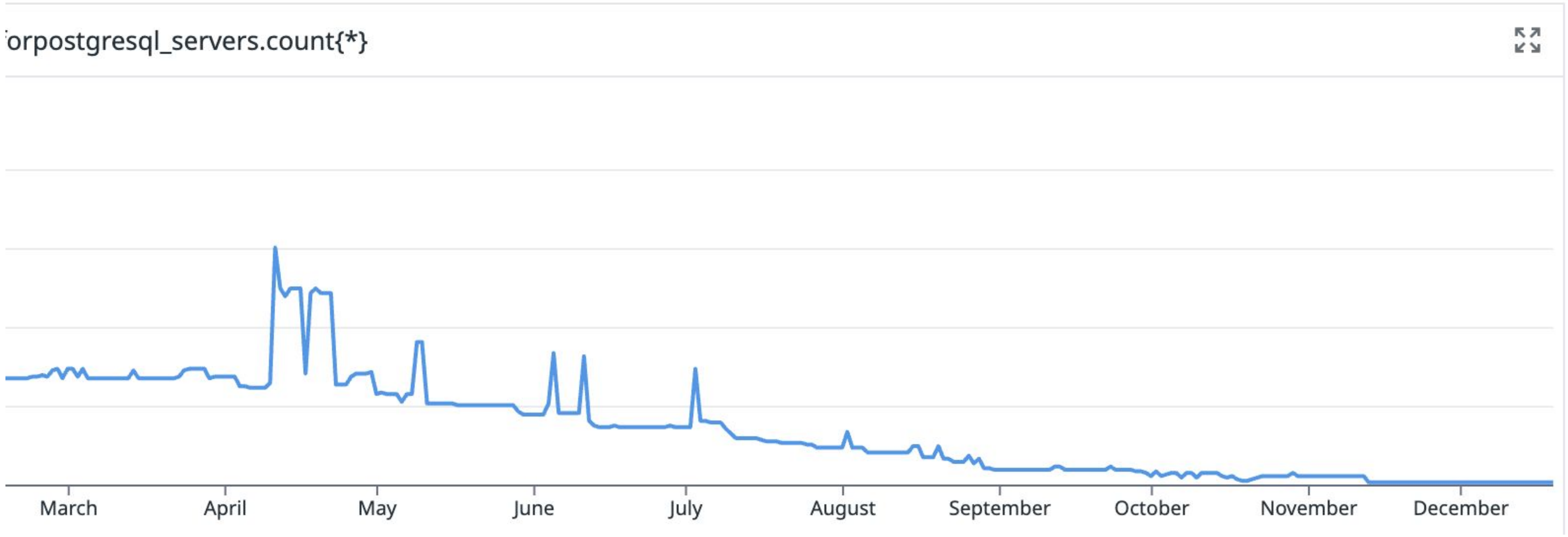
```go
// CloudManagedToPGKMigration migrates a given cloud managed database to PGK
func (w *Worker) CloudManagedToPGKMigration(ctx workflow.Context, req *internal.CloudManagedToPGKMigrationRequest) (*inte
	if err := validateCloudManagedToPGKMigrationRequestParams(req); err != nil {
		return nil, err
	}
	ctx = ContextWithSlackChannelOverride(workflows.DefaultContext(ctx), req.GetSlackChannel())
	overture.SendWorkflowDetailsToSlackChannel(ctx, req.GetSlackChannel(), req)
	humanSlackId := overture.SendWorkflowDetailsToSlack(ctx, req)
	logger := workflow.GetLogger(ctx)
	logger.Info(fmt.Sprintf("starting CloudManagedToPGKMigration workflow for database: %s", req.GetDatabaseIdentifie

	activityOptions := workflows.DefaultActivityOptions()
	ctx = workflow.WithActivityOptions(ctx, activityOptions)

	migrationPlan := "" +
		"1. Preparation Phase [1 business day]: Creating new PGK, HAProxy for traffic shifting, setting up replic
		"2. Application re-deployment with new HAProxy provisioned for migration \n" +
		"3. Read cutover [1 business day]: Shifting some portion (5%, 20%, 100% incrementally) of read traffic in
		"(If the database does not have read replica pool, this step is skipped)\n" +
		"4. Write cutover [~1 hour]: Shifting write traffic from Azure SS to PGK (expected downtime is ~2 minutes
	SendSlackChannelMessage(
		ctx, fmt.Sprintf("You can find the workflow migration plan in the following\n```%s```", migrationPlan),
	)
```

Effect of moving to self-managed



Effect of moving to self-managed

Azure Single Server instances



Pros and cons of self-managed

Pros:

- No artificial limits on disk space or replica counts
- Full control over extensions
- “Real” superuser
- `kill -15/-9` or `kubectl delete pod --force`

Cons:

- Validating version updates is a lot of work - Postgres, Patroni, wal-g, etc
- We’re on the hook for managing restarts/failovers
- k8s is a leaky abstraction sometimes - need host observability to understand perf problems

The platform

What are we doing with the control that we’ve gained?

The platform today

A custom proxy that matches Datadog’s style of internal service

- JWT authentication so that applications don’t worry about database passwords
- Metrics, logging, traces all in our Datadog standard
- Dynamic configuration updates for traffic shifting

The platform today

Supporting all database sizes

- Multitenant Postgres with many logical databases in the same pod
- Single tenant on Kubernetes nodes shared among other pods
- Single tenant on dedicated Kubernetes nodes
- Sharding an application across multiple Postgres clusters

The platform in the future

Big bets we're looking at (but aren't currently doing)

- Database devex - declarative infra
- "In database" sharding - Citus/pgdog/multigres/neki
- Storage extensions like orioledb
- Custom builds of Postgres for observability/features

Thanks!

Modern Role and Authentication Management Through a Custom Proxy

Date: 2025-09-30
Time: 16:00–16:50
Room: Forum B
Level: Intermediate

We’re hiring! For Postgres focused engineers and product managers and you'll see both on the careers site https://careers.datadoghq.com/