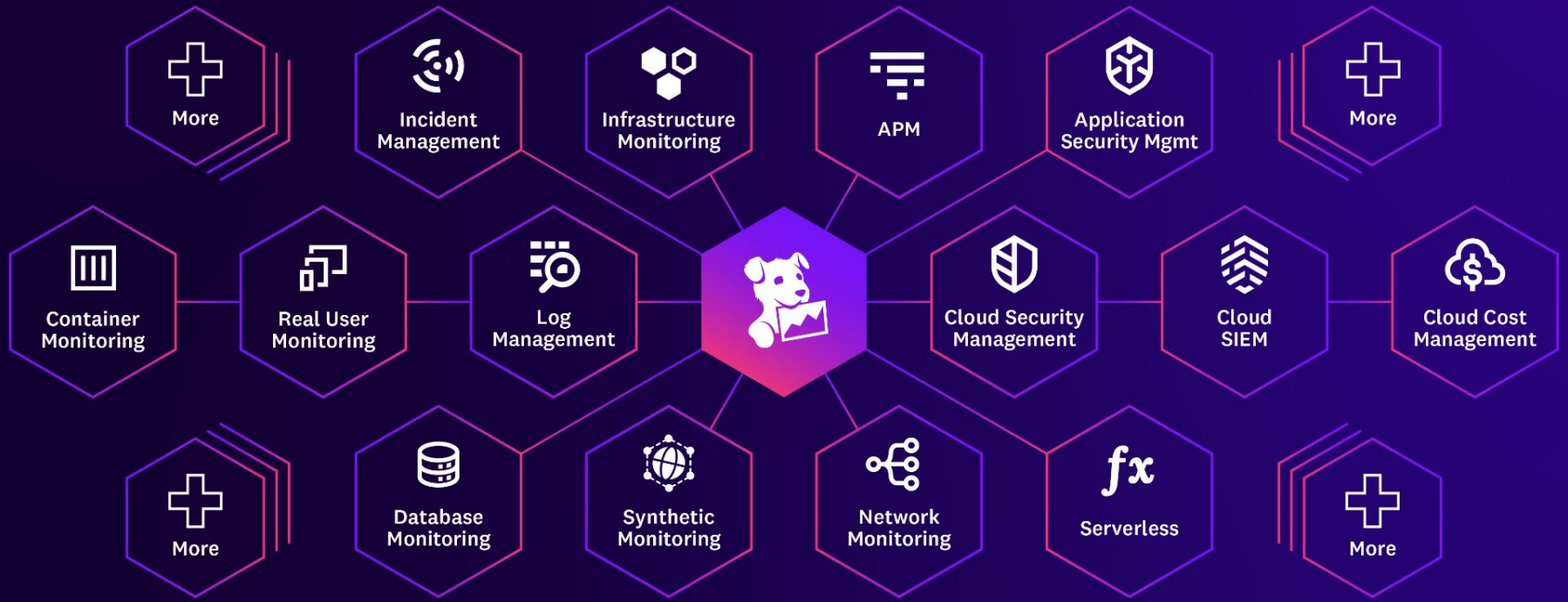
Breaking Down Monoliths: Automating PostgreSQL Cluster Isolation at Scale

Fabiana Scala
Senior Software Engineer

Nacho Mas
Software Engineer II

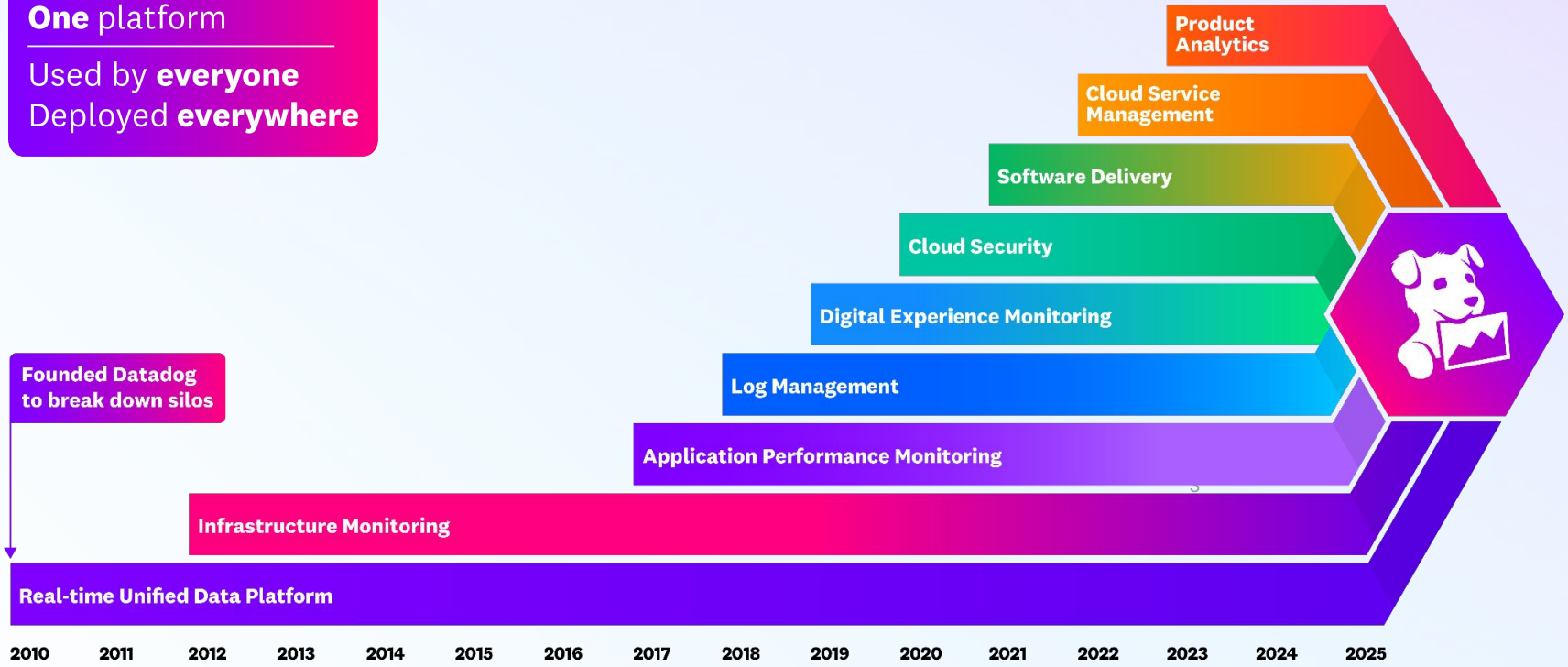
Datadog Features



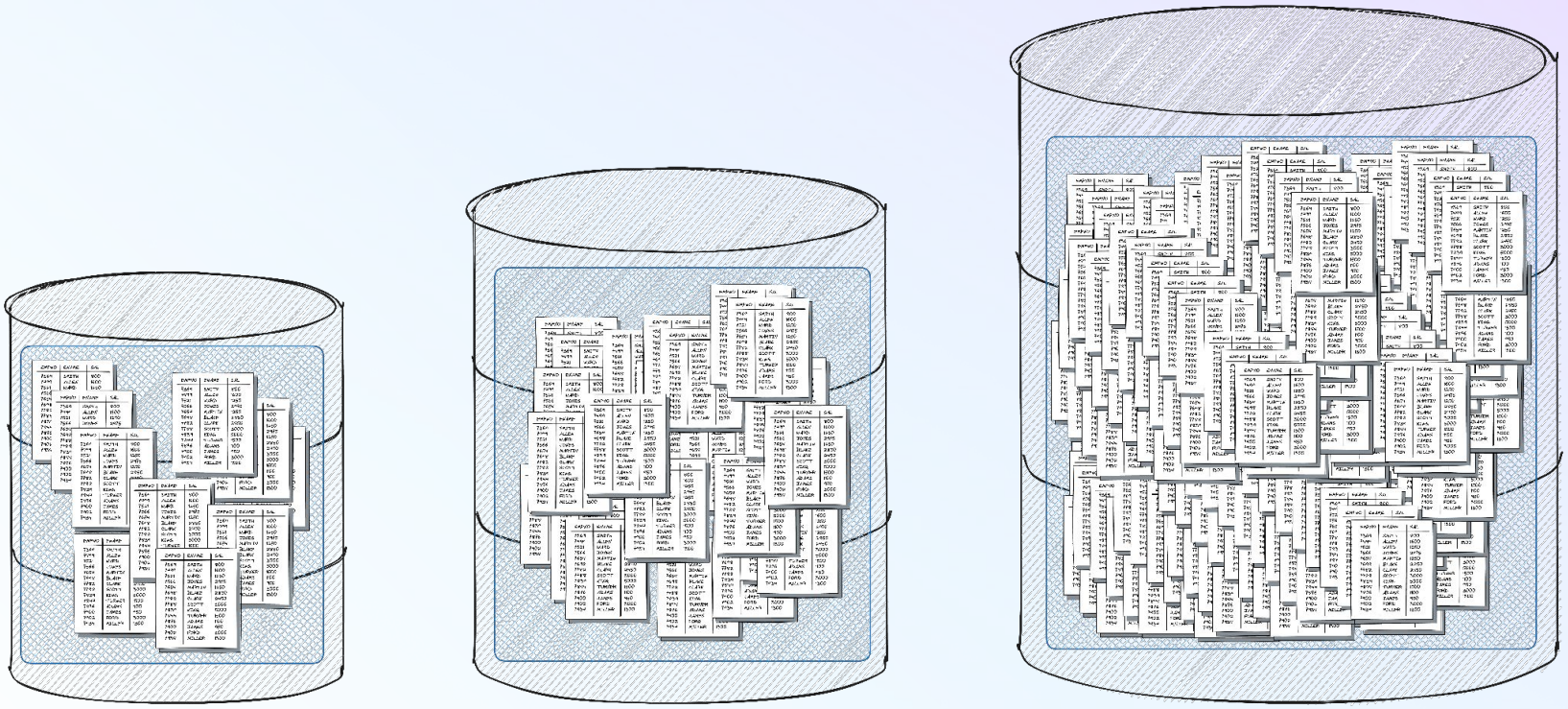
Our Portfolio's Growth

One product
One platform

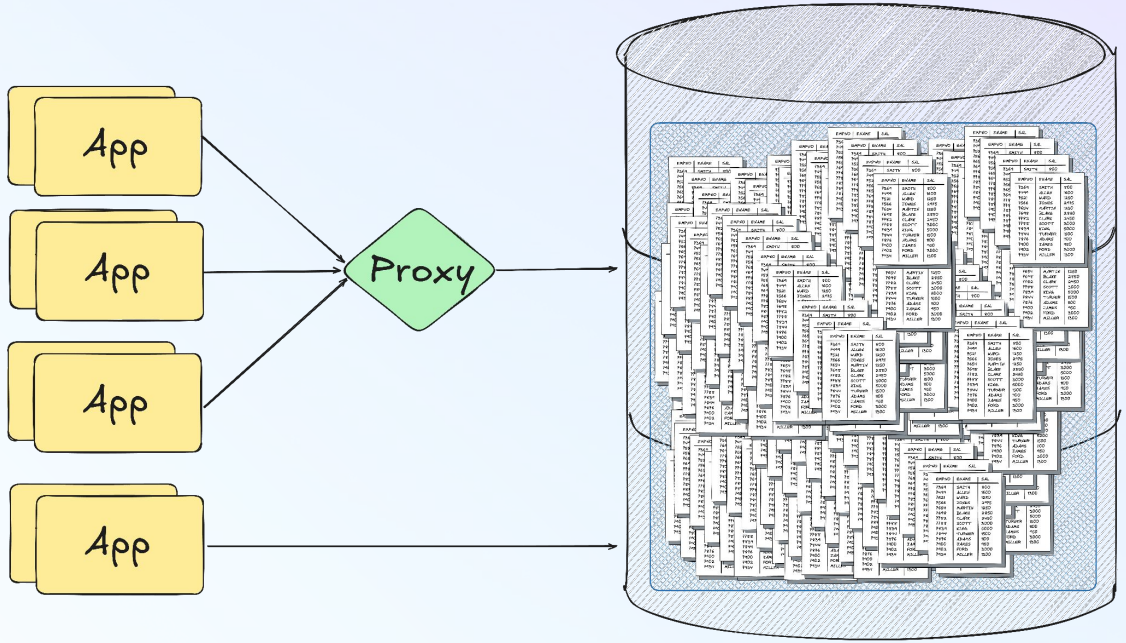
Used by **everyone**
Deployed **everywhere**



Our Postgres' Growth



Applications



Problems

Shared Fate: Huge blast radius

Scalability: Hitting limits in the database & connection pooler

Development: Interconnected development process & slow innovation

Security: Sharing the same database user & open permissions

Solutions

Platform Team

+

Postgres Proxy

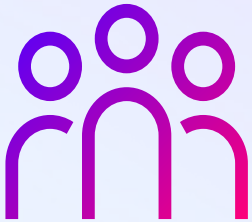
+

A Robot

Solution: Platform Team

Fully managed Postgres-as-a-Service

- Designed to store application data
- Modern features
- Leading security
- Excellent DevX
- Excellent production experience



Solution: Postgres Proxy

Connection pooling

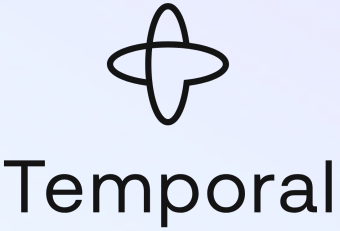
Passwordless authn

Great observability

Feature playground

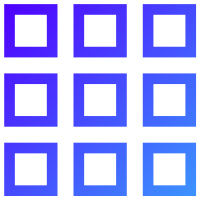
Solution: A Robot

- The robot is built using a set of **Temporal** Workflows and Activities
  - Reliable execution
  - Composable code
- Automates creating and tweaking a cluster
- Replacement for all manual steps



Why?

Reliability



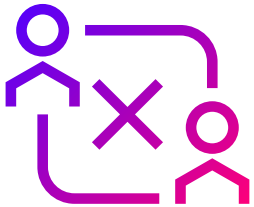
Consistency

Speed

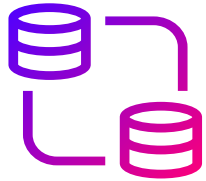
Breaking It Down

How to Solve it

Logical Separation

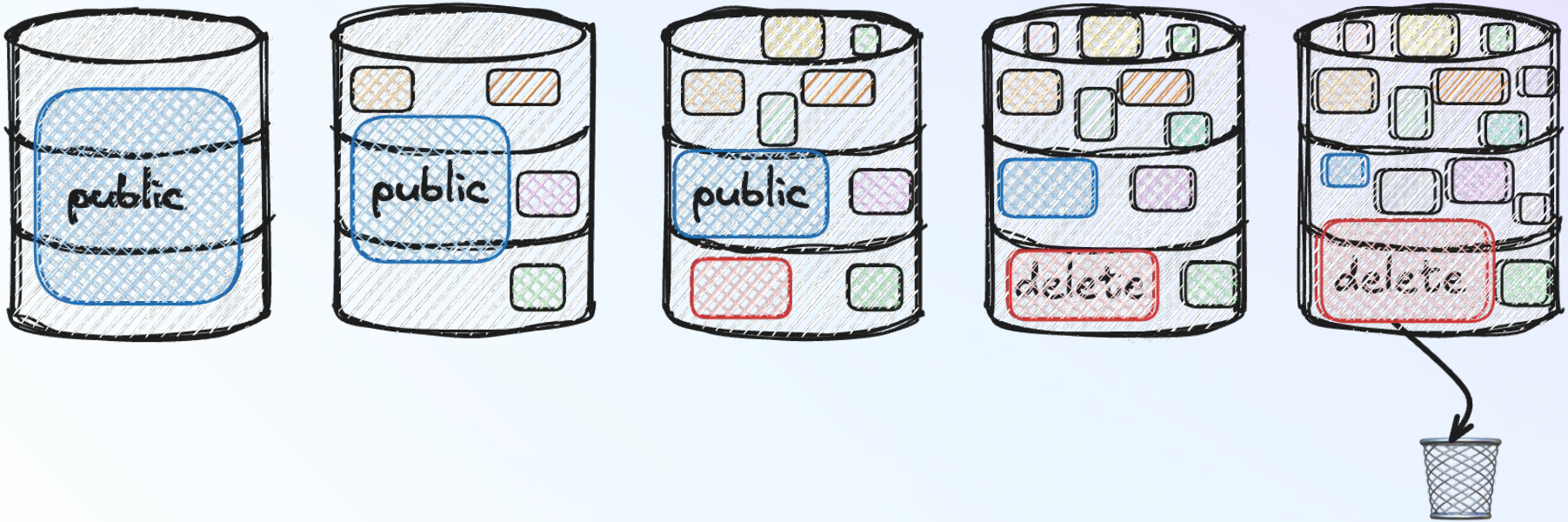


Access Reduction



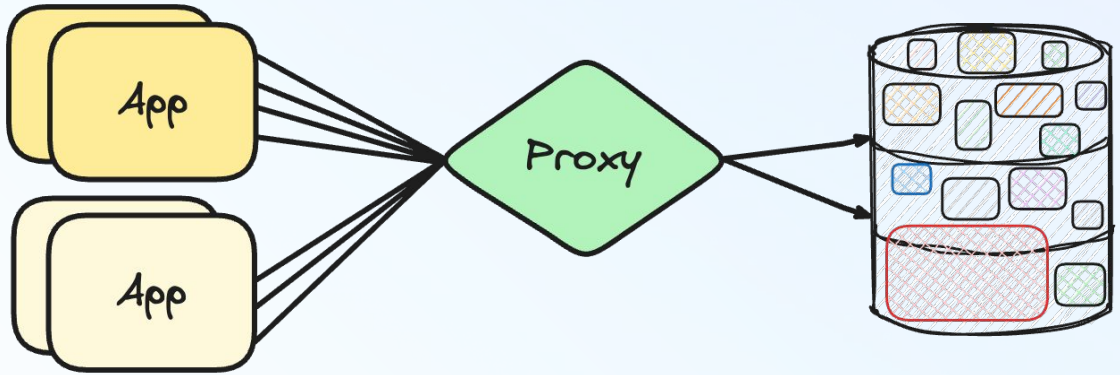
Physical Separation

Logical Separation & Deprecation



Onboarding to the Proxy

- Self-service multi-cloud automation
- At their own pace
- With dedicated users
- Leveraging a platform-owned abstraction



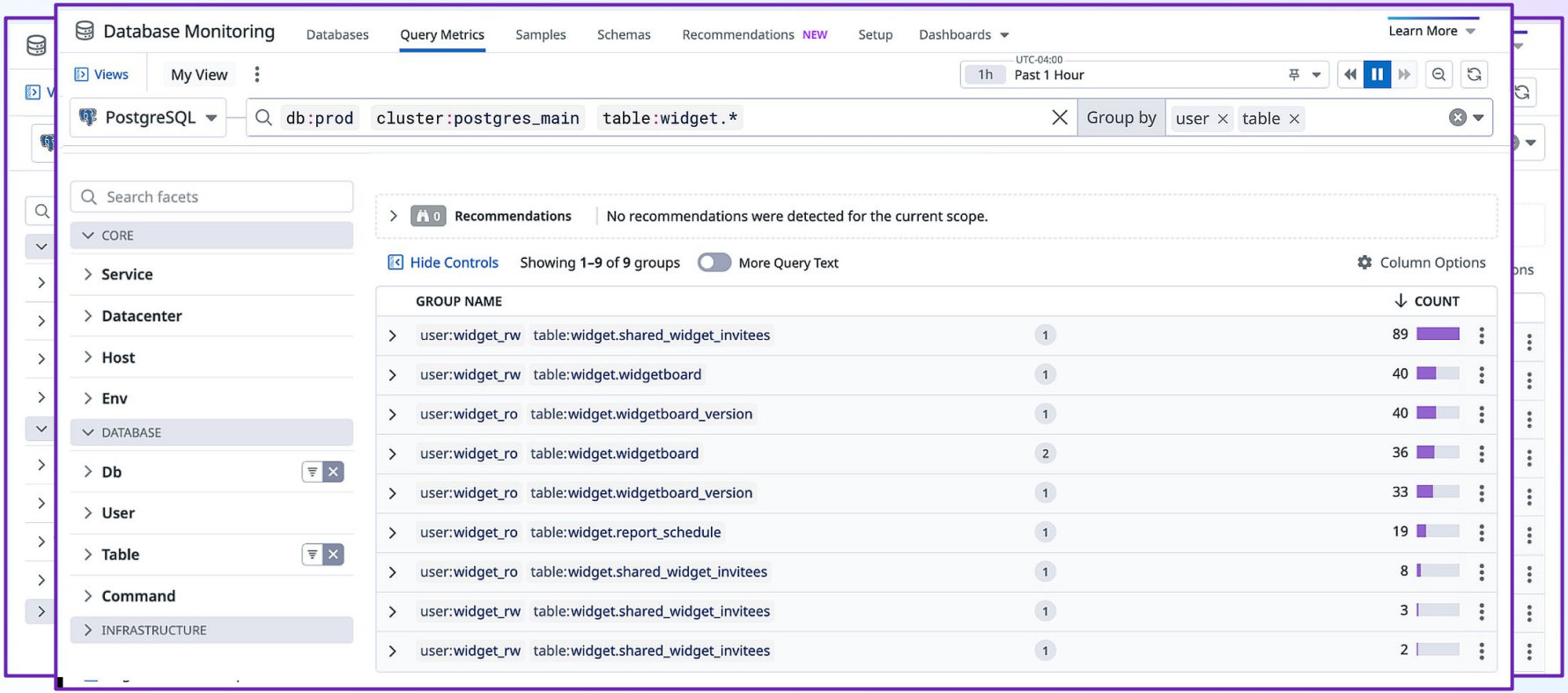
Access Reduction

Review access

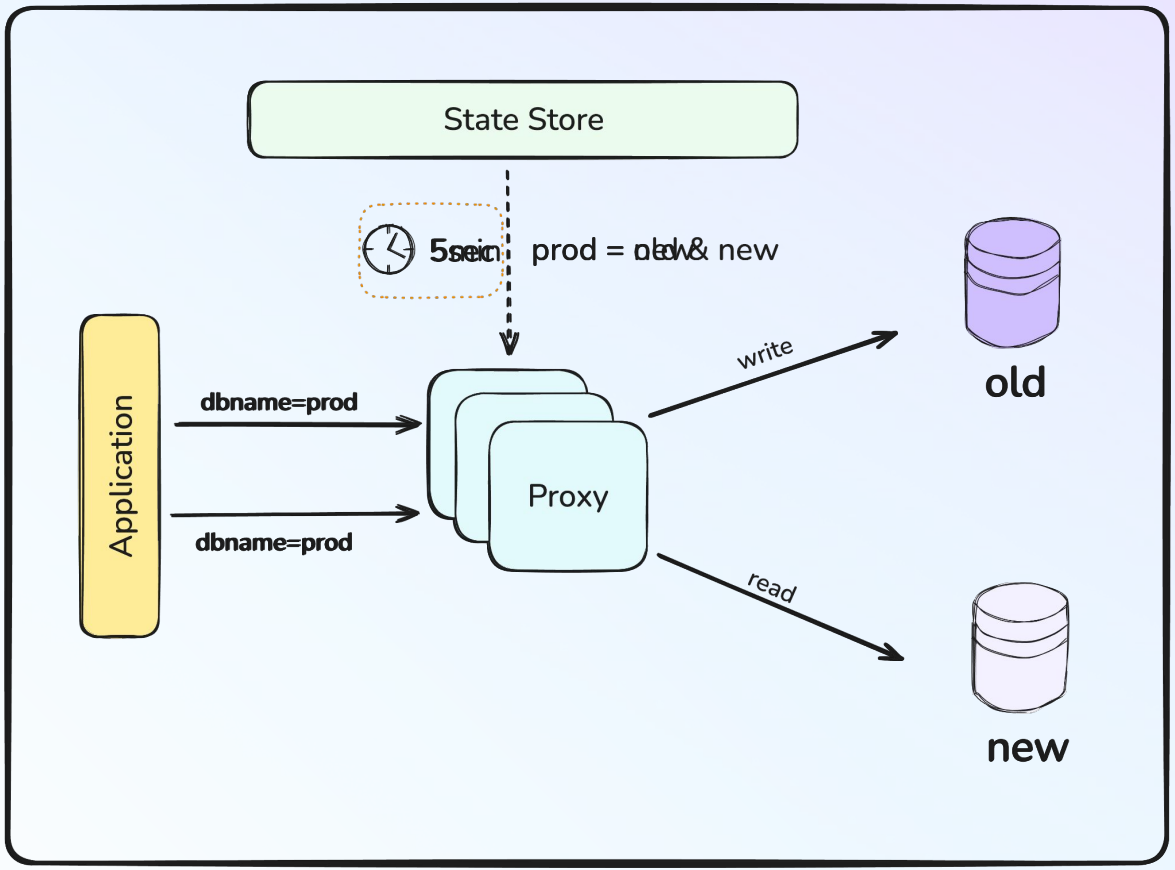
Decrease access

Restrict access

Database Monitoring

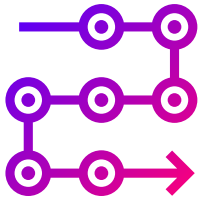


Physical Separation

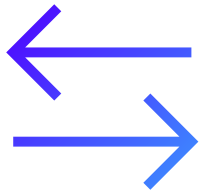


Automate all the things

How?



Temporal



Saga pattern

Checkpoints

No but really, how?

Preparation

Revoke access

Cancel connections

Block operations

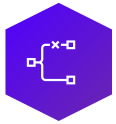
Reverse replication

Verify freshness

Tear down replication

Set up *reverse* replication

Remapping



Point old to new

Monitor

Persist change

Execution

No migration plan survives first contact with the database

The issues

The work is mysterious and important

Manually maintained DBs

Different cloud providers

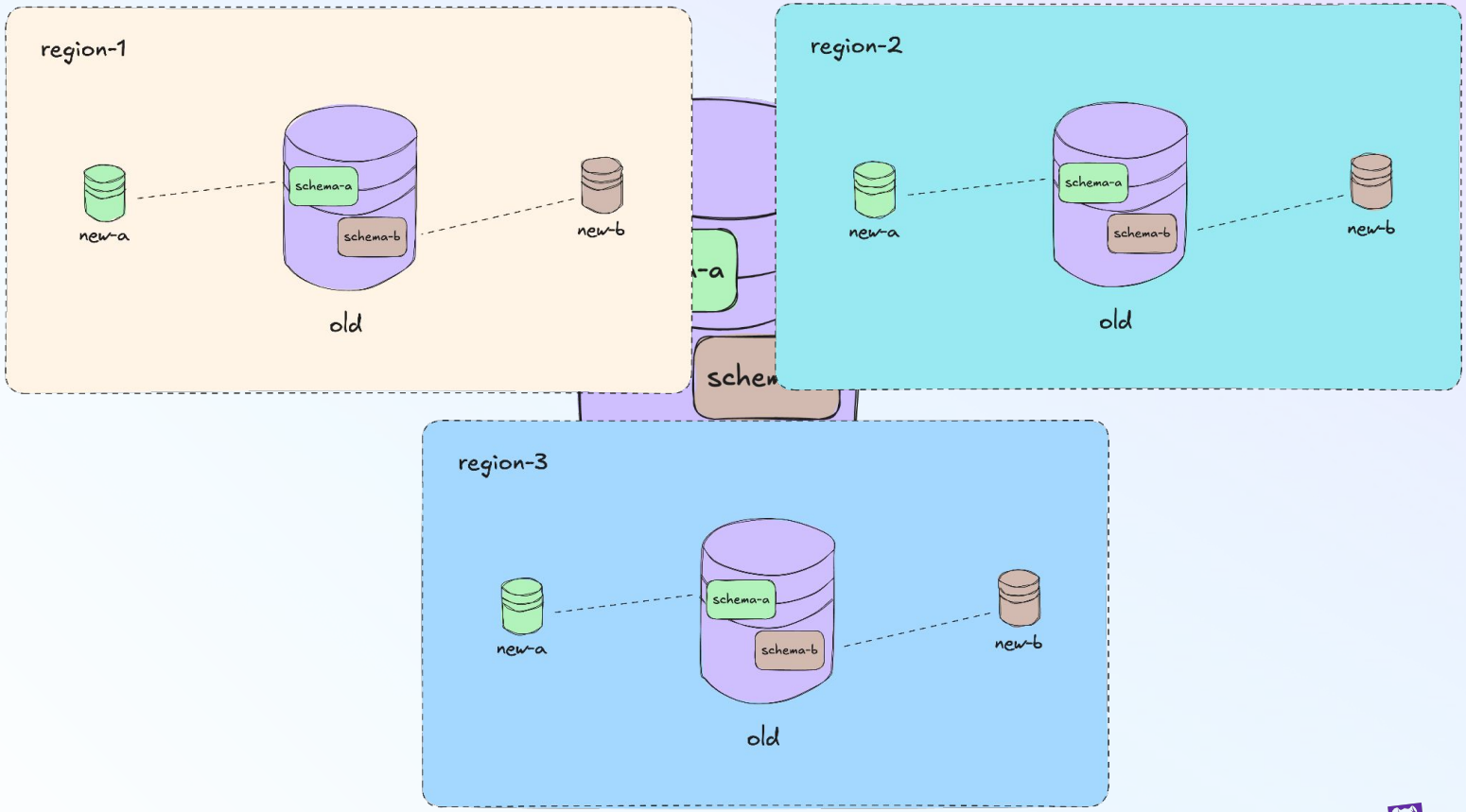
Different Postgres versions

Postgres rough edges

The issues

Manually maintained DBs

Regions galore



Manually maintained DBs

```
postgres=> SELECT current_role;
 current_role
--------------
 owner
(1 row)

postgres=> \dn
       List of schemas
    Name    |       Owner
------------+-------------------
 new_schema | owner
 public     | pg_database_owner
(2 rows)

postgres=> \dt public.application_table
                List of relations
 Schema |       Name        | Type  |    Owner
--------+-------------------+-------+-------------
 public | application_table | table | application
(1 row)
```

```
postgres=> ALTE                                              EMA new_schema;
ERROR:  must be
```

The issues

Different Postgres versions

Different Postgres versions

```
SELECT 1
    FROM pg_subscription s
    JOIN pg_subscription_rel sr ON s.oid = sr.srsubid
    JOIN pg_class c ON sr.srrelid = c.oid
    JOIN pg_namespace n ON c.relnamespace = n.oid
    WHERE s.subname = $1
        AND n.nspname = $2
        AND c.relname = $3
        AND sr.srsubstate = 'r'
```

Fig.1: checking whether the subscription is ready, ca. 2024 (colorized)

```
ERROR:  permission denied for table pg_subscription
```

Different Postgres versions

```
postgres=> SELECT oid FROM pg_subscription;
ERROR:  permission denied for table pg_subscription
postgres=> SELECT version();
                                          version
-------------------------------------------------------------------------------------------
 PostgreSQL 13.15 on aarch64-unknown-linux-gnu, compiled by gcc (GCC) 7.3.1 20180712 (Red H
(1 row)
```

```
postgres=> SELECT oid FROM pg_subscription;
    oid
-----------
 127907059
(1 row)

postgres=> SELECT version();
                                          version
-------------------------------------------------------------------------------------------
 PostgreSQL 15.10 on x86_64-pc-linux-gnu, compiled by Debian clang version 12.0.1, 64-bit
(1 row)
```

The issues

Different cloud providers

Different cloud providers

```
postgres=> SET ROLE cloudsqlsuperuser;
SET
postgres=> SELECT oid FROM pg_subscription;
ERROR:  permission denied for table pg_subscription
```

```
postgres=> SET ROLE rds_superuser;
SET
postgres=> SELECT oid FROM pg_subscription;
    oid
-----------
 127907059
(1 row)
```

The issues

Postgres rough edges

Postgres rough edges

The issues

Manually maintained DBs

Different cloud providers

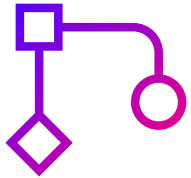
Entropy

Different Postgres versions

Postgres rough edges

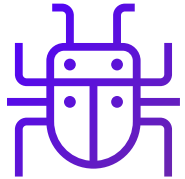
Lessons learned

Lessons learned

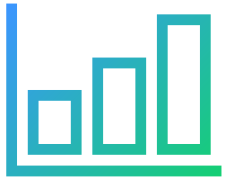


Automate early

Collaborate with customers



Test thoroughly



Plan for the long term

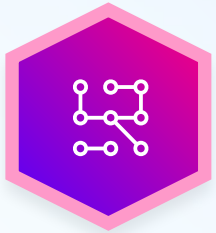
Thank you

Got feedback for us? Scan the QR code below!

DATADOG

The issues

The work is mysterious and important



Manually maintained DBs

Postgres rough edges

Different Postgres versions

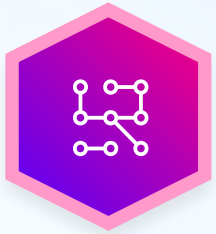
Different topologies

Different cloud providers

Entropy

The issues

The work is mysterious and important



Manually maintained DBs

Postgres rough edges

Different Postgres versions

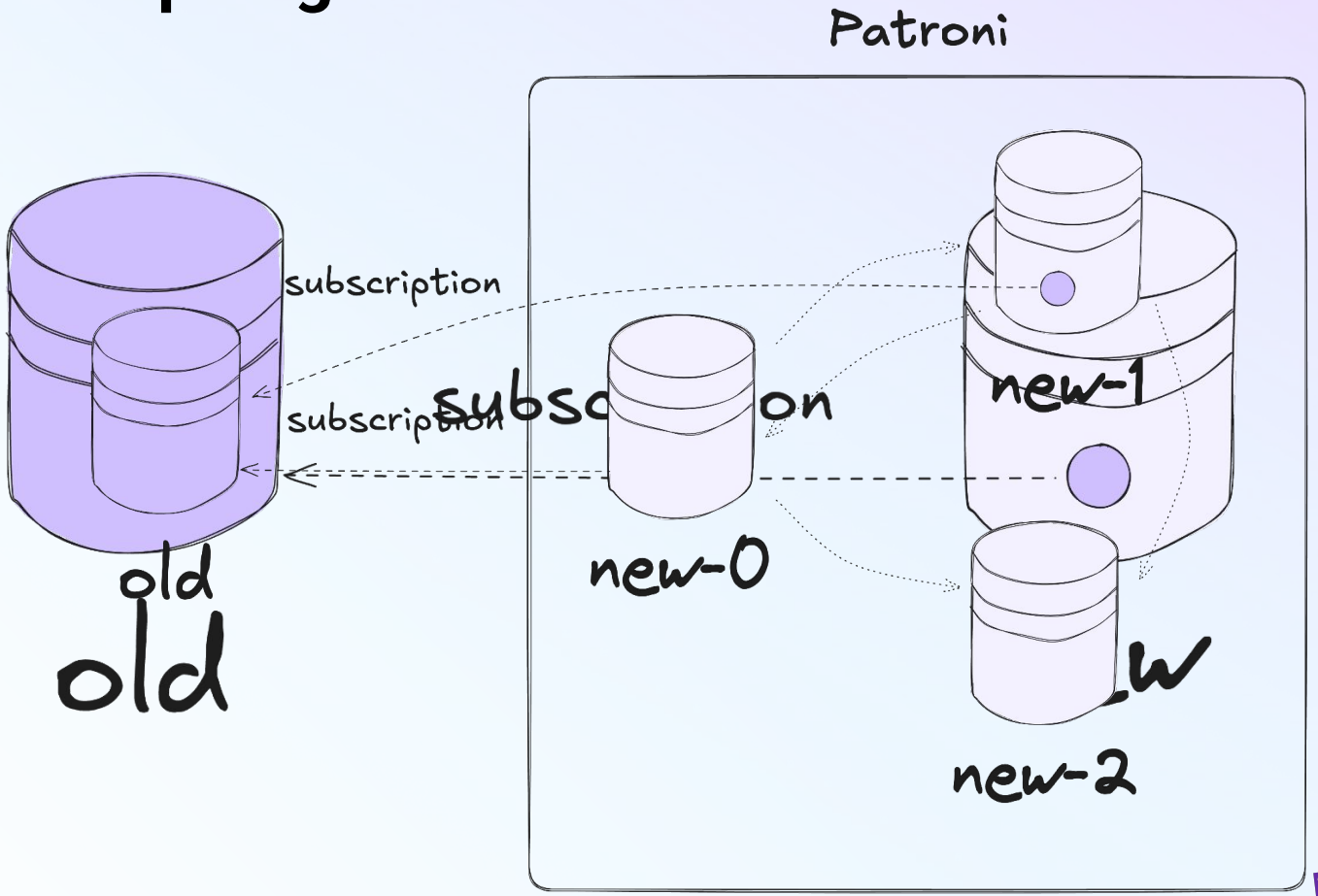
Different topologies

Different cloud providers

The issues

Different topologies

Different topologies



Different topologies

```
new=# SELECT pg_drop_replication_slot('replication_slot');
 pg_drop_replication_slot
__________________________


(1 row)
```

```
old=# CREATE SUBSCRIPTION reverse_subscription
  CONNECTION 'host=new port=5432 dbname=new'
  PUBLICATION reverse_publication WITH (slot_name = 'replication_slot');
ERROR:  could not create replication slot "replication_slot":
ERROR:  replication slot "replication_slot" already exists
```