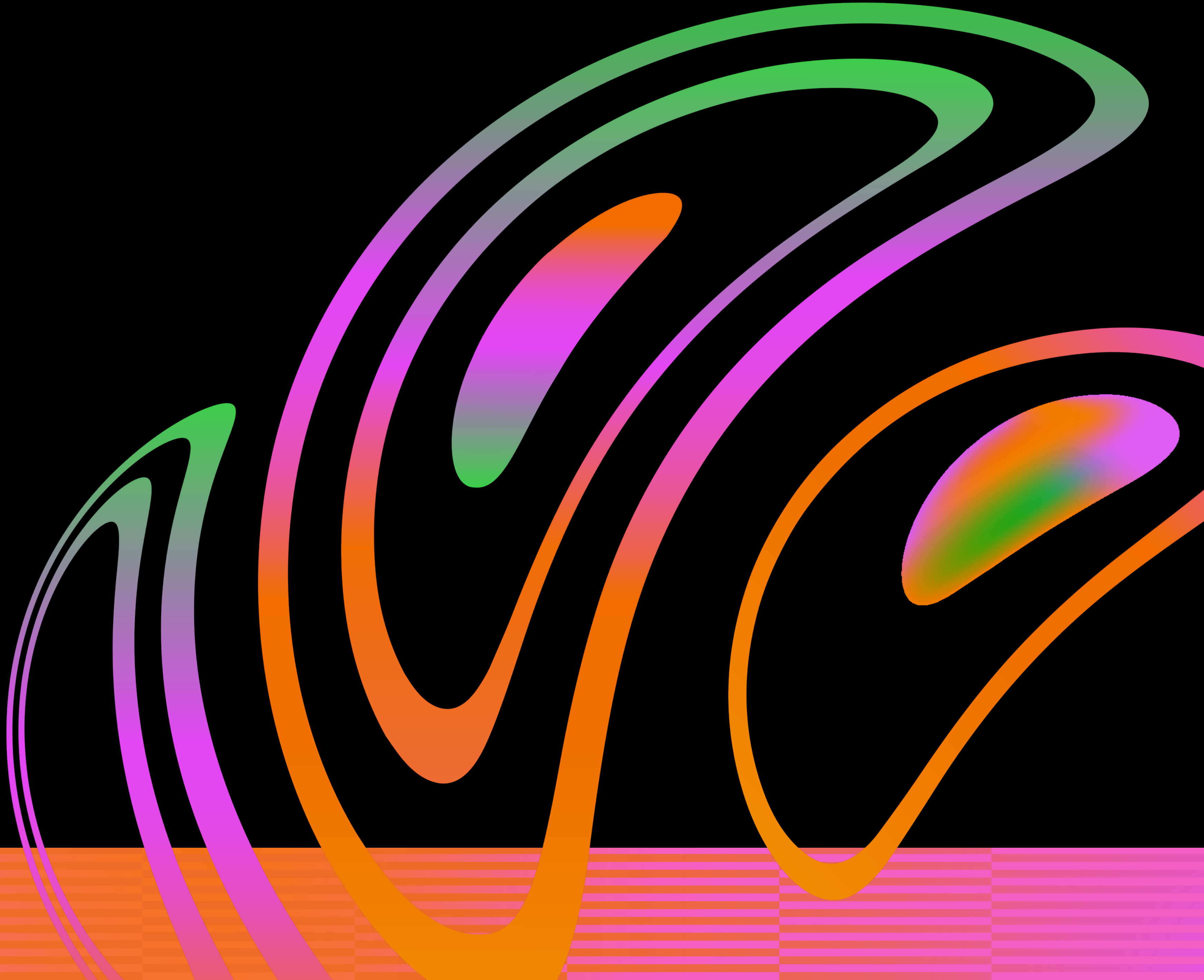
AWS re:Invent



DAT210

Transforming Hilton's reservation system with Amazon Aurora

Eddie Maier
Director of Cloud Platforms
Hilton

Erick Dame
Sr. Solutions Architect
AWS

AWS re:Invent

Agenda

Hilton overview

The journey of Hilton's reservation system to AWS

Testing the design

Architecting for zero downtime

Future plans

Hilton

Hilton is a leading global hospitality company with 18 brands spanning the lodging sector

Our brands are comprised of:

- More than 6,200 properties
- More than 983,000 rooms
- In 118 countries and territories

WORLDWIDE SUPPLY

6,215
Properties*

*Seven independent. Data as of June 30, 2020

The Journey of Hilton's Reservation System to AWS

AWS re:Invent

An iterative journey

A THREE-YEAR JOURNEY IN THE MAKING

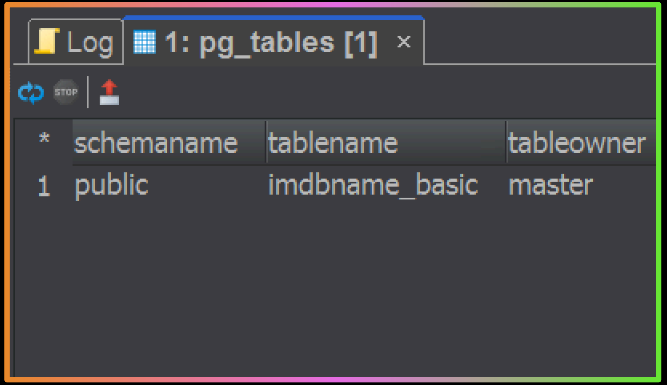
An agile approach to platform and application architectures

- Chose Amazon Aurora as the underlying database solution
- Started with Availability Engines and Aurora MySQL
- Database platforms changed to Amazon Aurora with PostgreSQL compatibility by the end of the migration
- Optimization efforts occurred across each iteration

MySQL

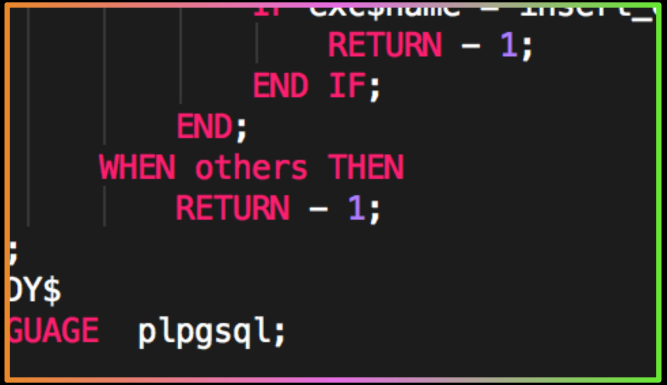
PostgreSQL

Understand basic database engine differences



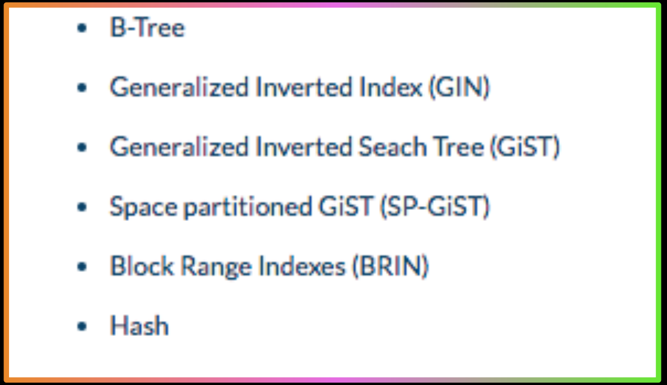
*	schemaname	tablename	tableowner
1	public	imdbname_basic	master

PostgreSQL is a lowercase data dictionary



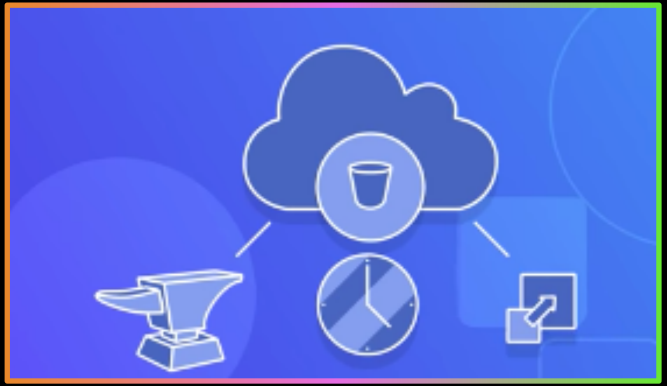
```
            IF exc$name = insert_
                RETURN - 1;
            END IF;
        END;
    WHEN others THEN
        RETURN - 1;
;
DY$
GUAGE  plpgsql;
```

Use “exception handlers” when needed, not by default

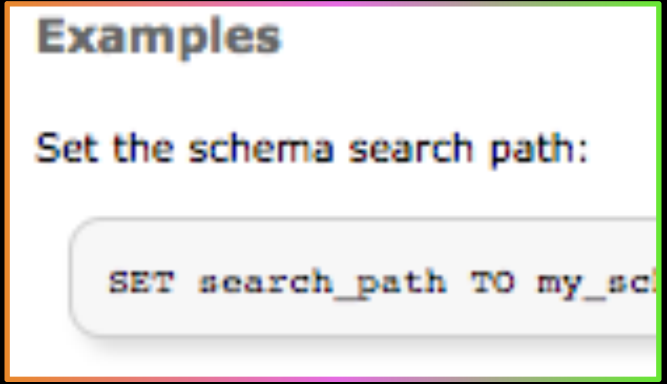


- B-Tree
- Generalized Inverted Index (GIN)
- Generalized Inverted Seach Tree (GiST)
- Space partitioned GiST (SP-GiST)
- Block Range Indexes (BRIN)
- Hash

PostgreSQL has six different index types



search_path replaces PUBLIC SYNONYM

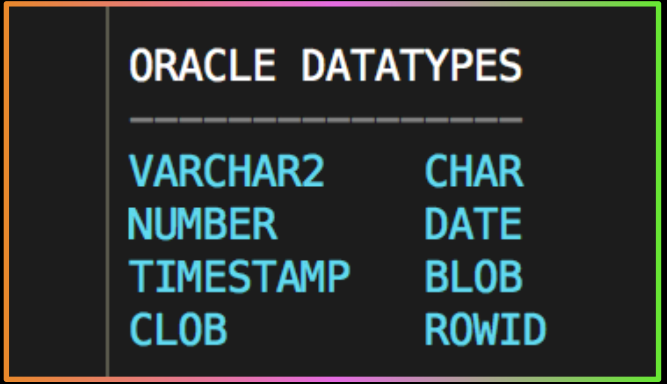


Examples

Set the schema search path:

```
SET search_path TO my_sc
```

Store your BLOBs in Amazon S3 instead of the database



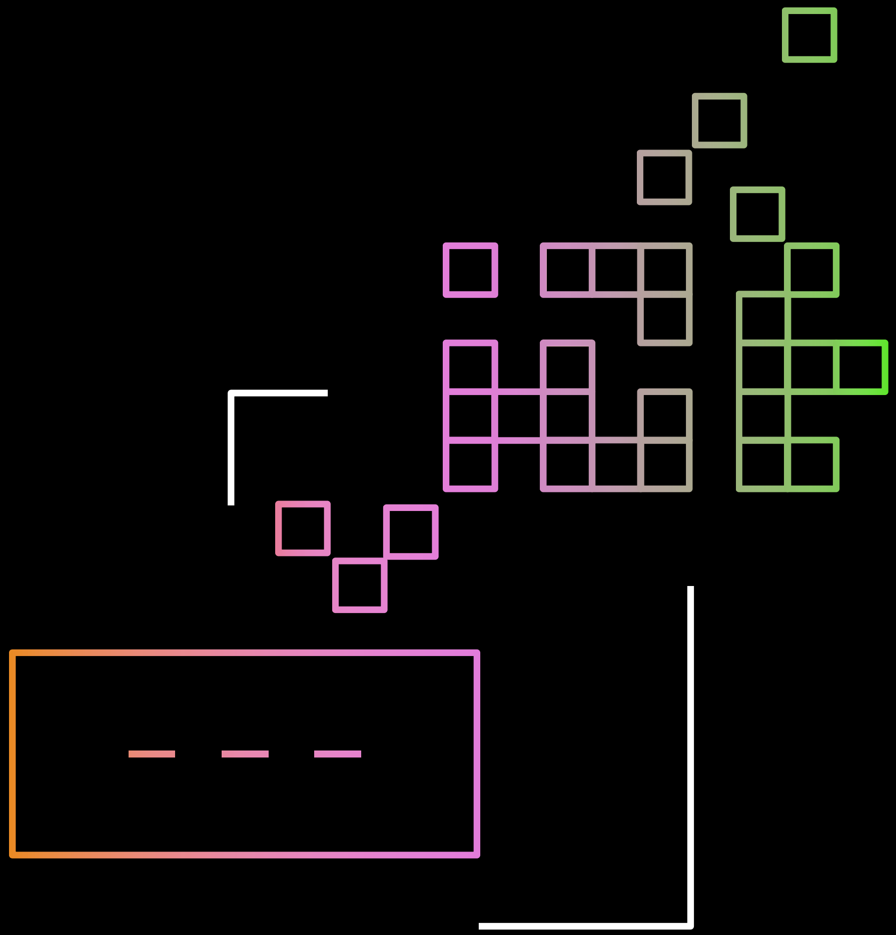
```
ORACLE DATATYPES
----------------
VARCHAR2     CHAR
NUMBER       DATE
TIMESTAMP    BLOB
CLOB         ROWID
```

PostgreSQL has 64 datatypes

Planning for the migration

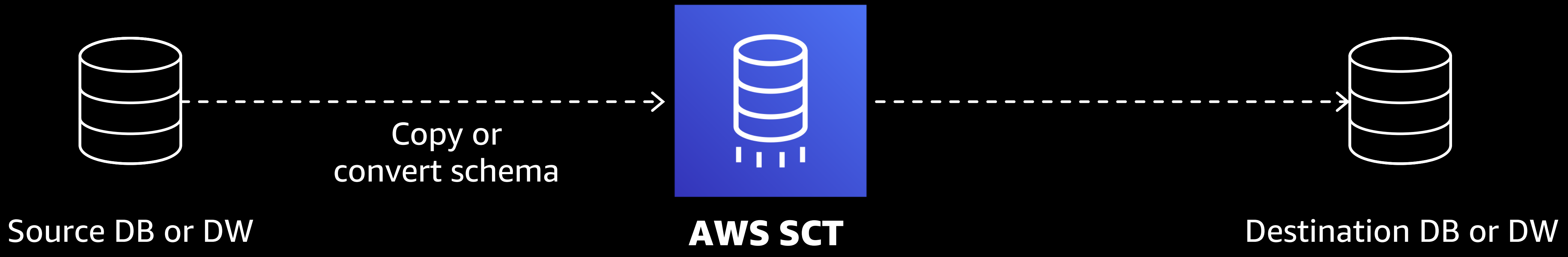
Application and platform architects drafted a target scope

- Create reference architecture for new environment
- Incorporate core reservation functions
- Preserve target transaction rates from datacenters in new AWS environment
- Leverage certain tools for data replication between on-premise and cloud environments
- Ensure architectures so that current recovery point objective (RPO) and recovery time objective (RTO) would be met
- Engage AWS support, IEM process: https://aws.amazon.com/premiumsupport/programs/iem/

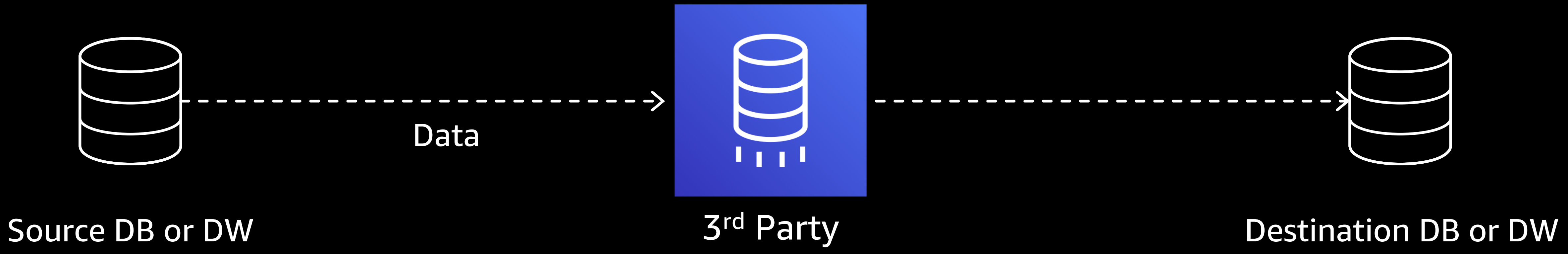


AWS Database Migration Process

STEP 1: Convert or copy your schema



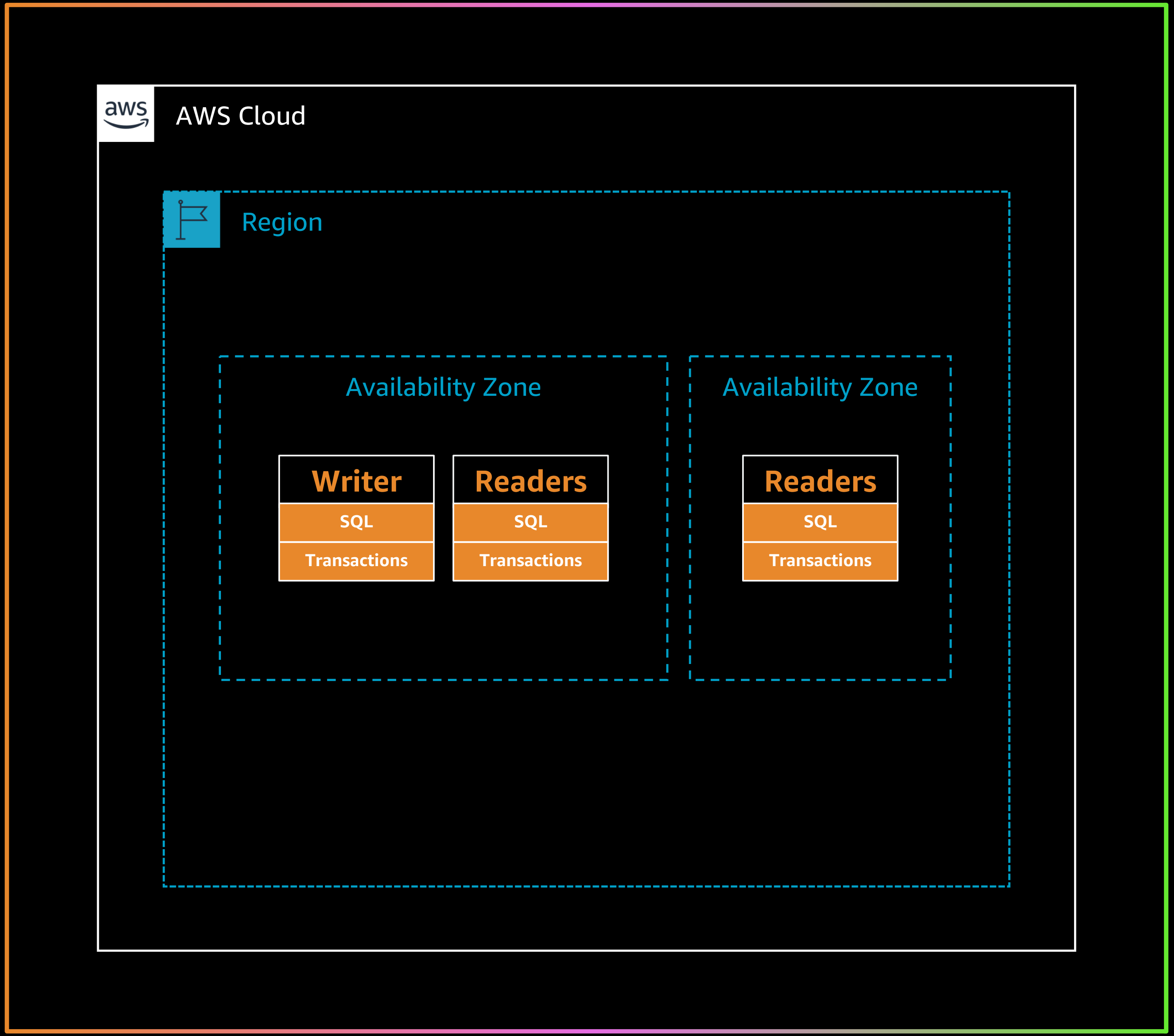
STEP 2: Move your data



Testing the design

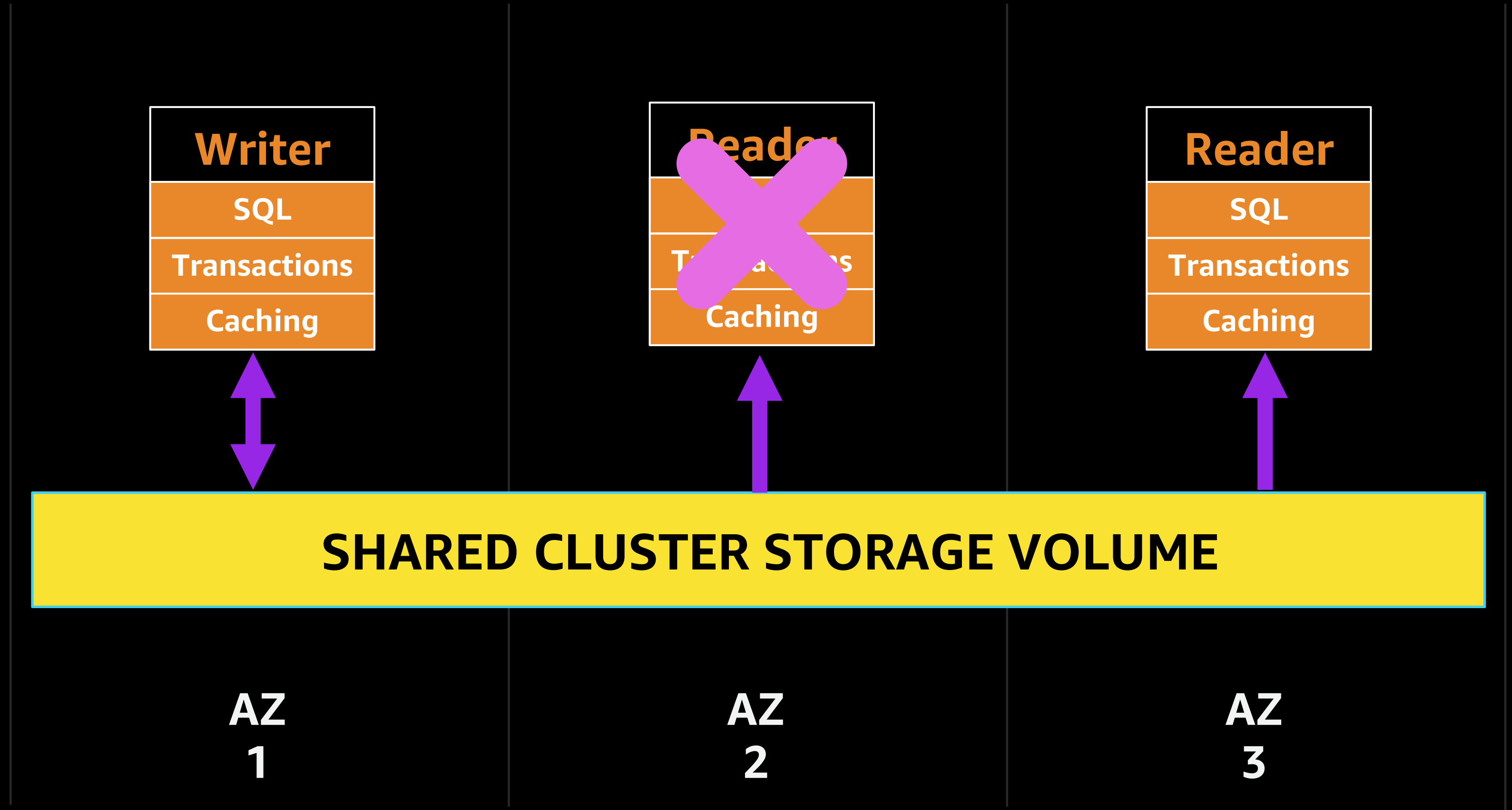
Testing for success

- Rigorous application testing and tuning to get optimal query response time
- Results determine target infrastructure needed for the release
- Infrastructure testing and drills part of the new environment
- Lessons learned



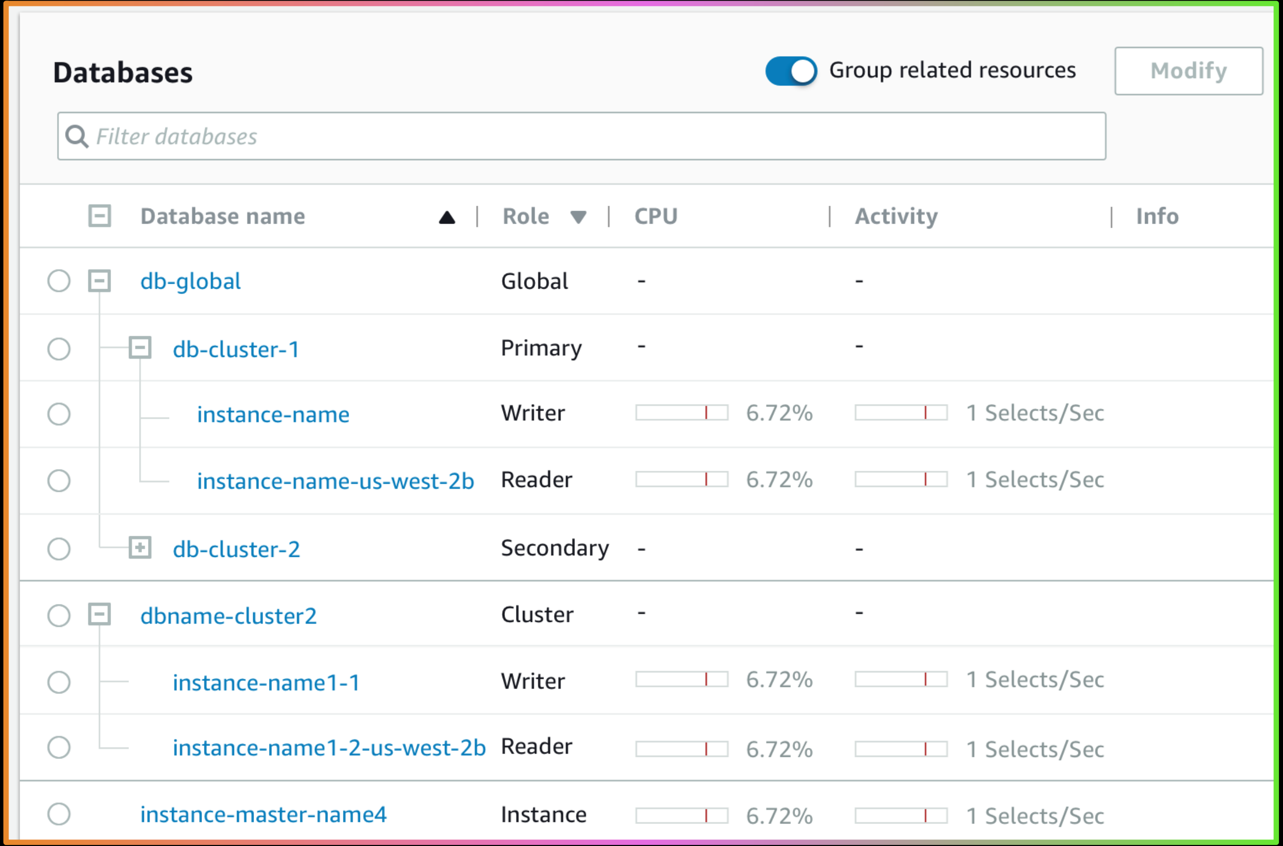
Testing for failures

```
SELECT aurora_inject_replica_failure(
   percentage_of_failure,
   quantity,
   'replica_name'
);
```



Additional fault injection queries can be found here:
https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/AuroraPostgreSQL.Managing.FaultInjectionQueries.html

Monitoring and performance insights

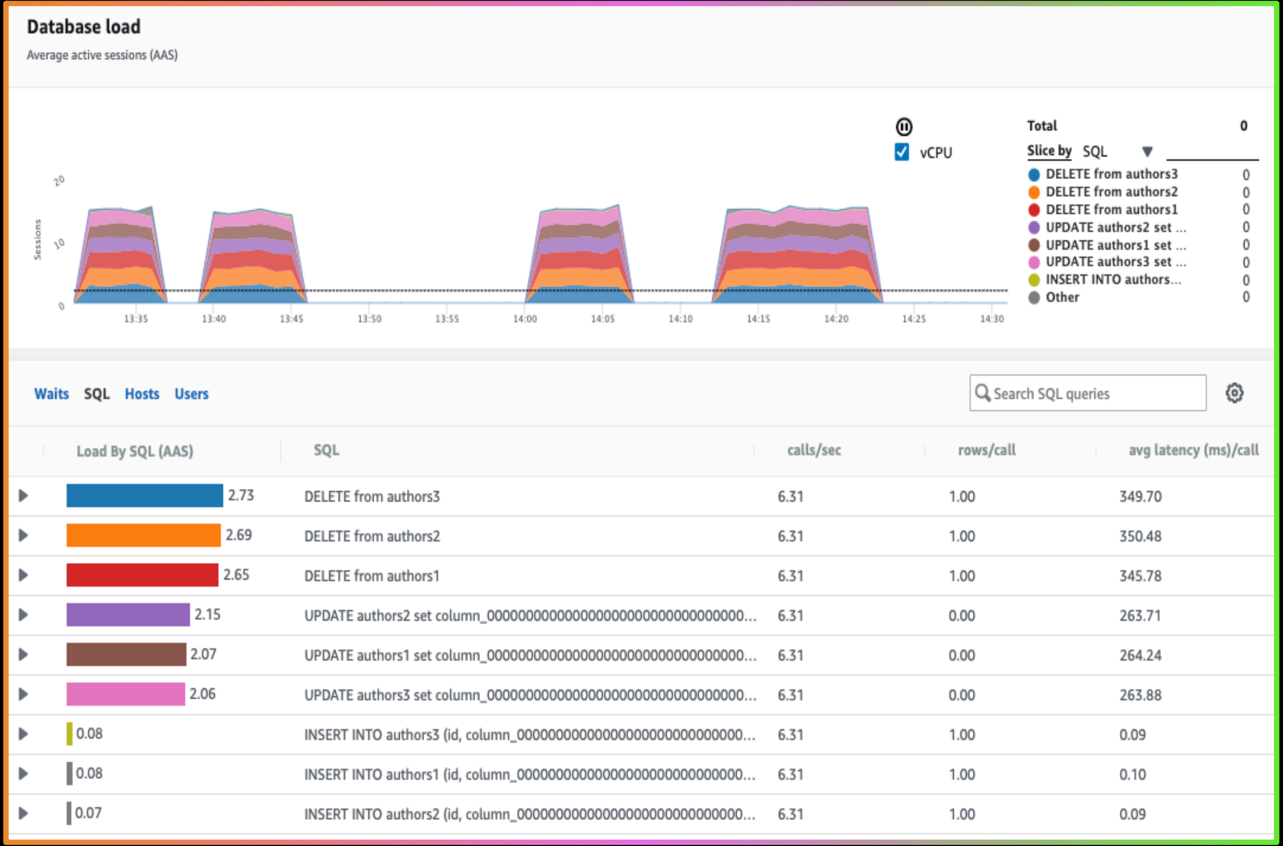


Databases

Group related resources | Modify

Filter databases

	Database name ▲	Role ▼	CPU	Activity	Info
○	⊟ db-global	Global	-	-	
○	⊟ db-cluster-1	Primary	-	-	
○	instance-name	Writer	6.72%	1 Selects/Sec	
○	instance-name-us-west-2b	Reader	6.72%	1 Selects/Sec	
○	⊞ db-cluster-2	Secondary	-	-	
○	⊟ dbname-cluster2	Cluster	-	-	
○	instance-name1-1	Writer	6.72%	1 Selects/Sec	
○	instance-name1-2-us-west-2b	Reader	6.72%	1 Selects/Sec	
○	instance-master-name4	Instance	6.72%	1 Selects/Sec	



Database load

Average active sessions (AAS)

Total	0
Slice by SQL ▼	
● DELETE from authors3	0
● DELETE from authors2	0
● DELETE from authors1	0
● UPDATE authors2 set ...	0
● UPDATE authors1 set ...	0
● UPDATE authors3 set ...	0
● INSERT INTO authors...	0
● Other	0

Waits SQL Hosts Users

Search SQL queries

	Load By SQL (AAS)	SQL	calls/sec	rows/call	avg latency (ms)/call
▶	2.73	DELETE from authors3	6.31	1.00	349.70
▶	2.69	DELETE from authors2	6.31	1.00	350.48
▶	2.65	DELETE from authors1	6.31	1.00	345.78
▶	2.15	UPDATE authors2 set column_000000000000000000000000000000000...	6.31	0.00	263.71
▶	2.07	UPDATE authors1 set column_000000000000000000000000000000000...	6.31	0.00	264.24
▶	2.06	UPDATE authors3 set column_000000000000000000000000000000000...	6.31	0.00	263.88
▶	0.08	INSERT INTO authors3 (id, column_0000000000000000000000000000...	6.31	1.00	0.09
▶	0.08	INSERT INTO authors1 (id, column_0000000000000000000000000000...	6.31	1.00	0.10
▶	0.07	INSERT INTO authors2 (id, column_0000000000000000000000000000...	6.31	1.00	0.09

- Analyze and tune Database Performance
- Available through AWS Management Console and AWS API SDK
- Set up alarms for key issues

- Database load is determined by average active sessions (AAS)
- Categorized data by wait events, SQL, hosts, and users
- SQL statistics for queries NEW!

Achieving a zero downtime deployment

Architecting for zero downtime

MONITOR
Create monitors and notification channels across all tiers

RESILIENCY
Ensure enough instances are always available

CONFIGURATION
Tune settings for planned recovery scenarios

VALIDATE
Test successful and failure conditions repeatedly

SIMPLIFY
Automate and script as much as possible

Aurora backup and restore

AUTOMATED BACKUPS

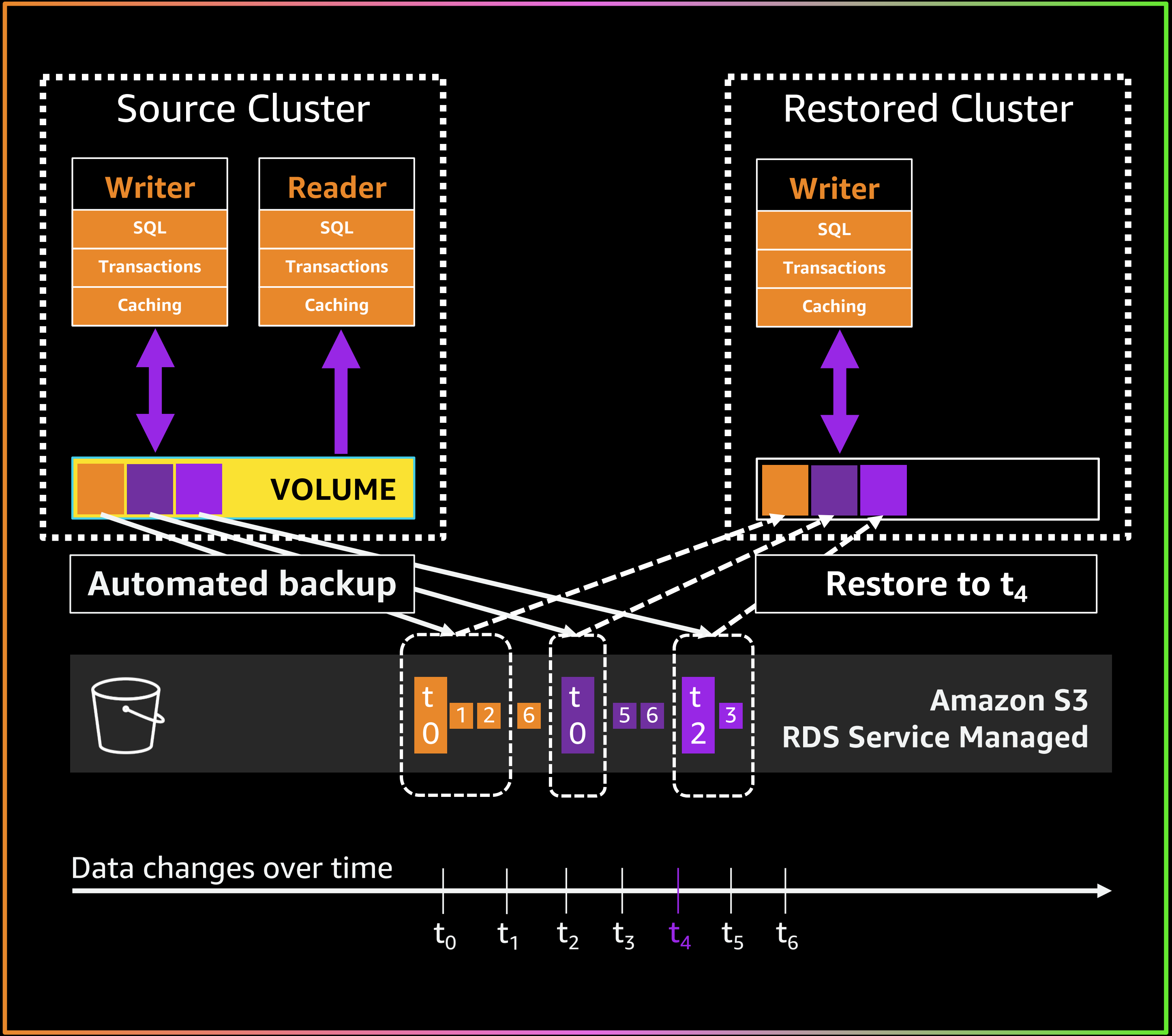
- Between 1–35 days retention
- Recover up to the last ~5 min point in time

SNAPSHOTS

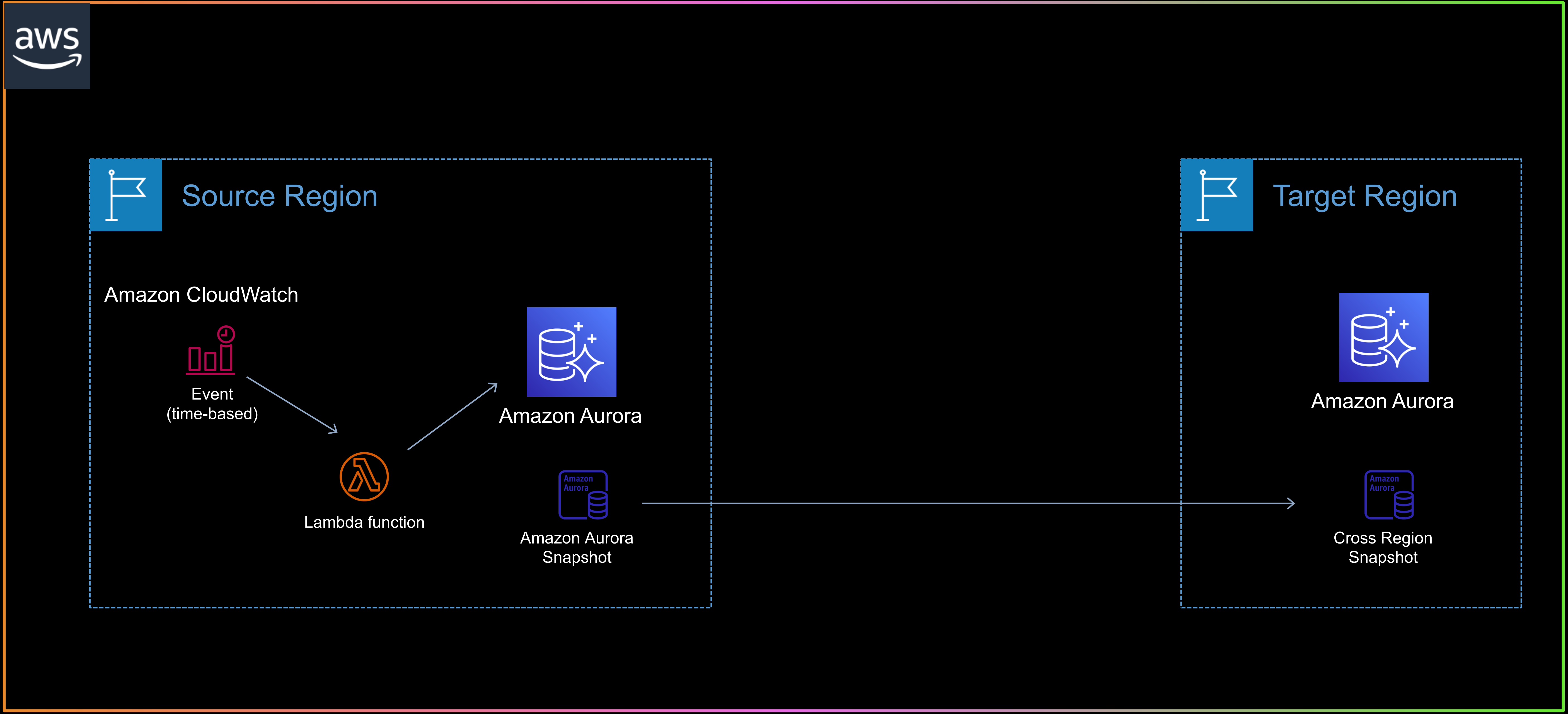
- Instantly create user snapshots
- No performance impact
- Copy snapshots to another region
- Share snapshots with other AWS accounts

RESTORE

- Time depends on cluster volume size
- Always creates a new DB cluster

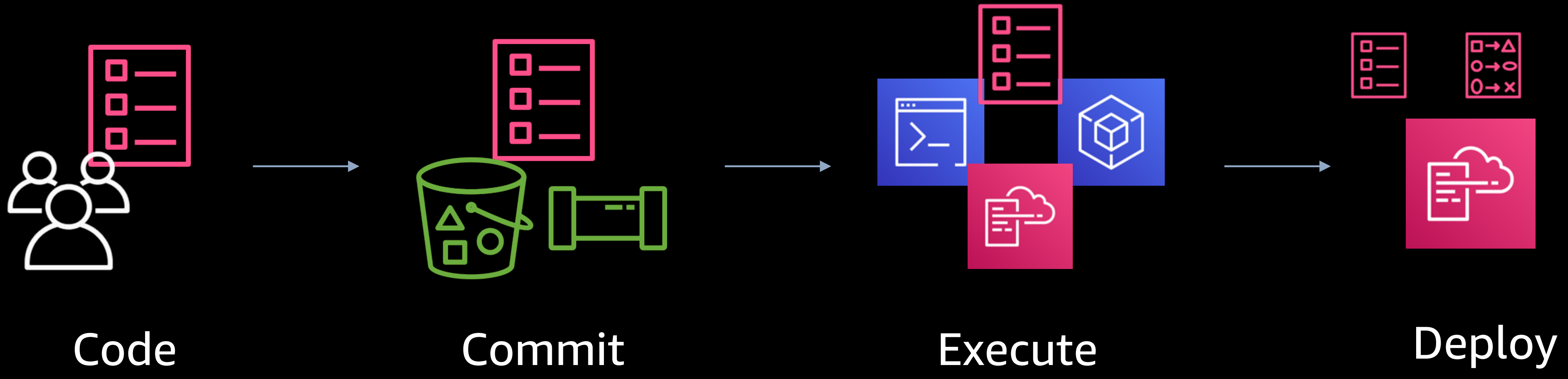


Cross Region Snapshot



Treat Infrastructure as Code, Even for Databases!

- Once you have a pattern, “stamp” it out in code
- For Amazon Aurora with PostgreSQL compatibility, define DB parameter groups in code
- Dynamic parameters apply immediately, while a static parameter will not change until there is a reboot



https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/USER_WorkingWithParamGroups.html#Aurora.Managing.ParameterGroups

The migration

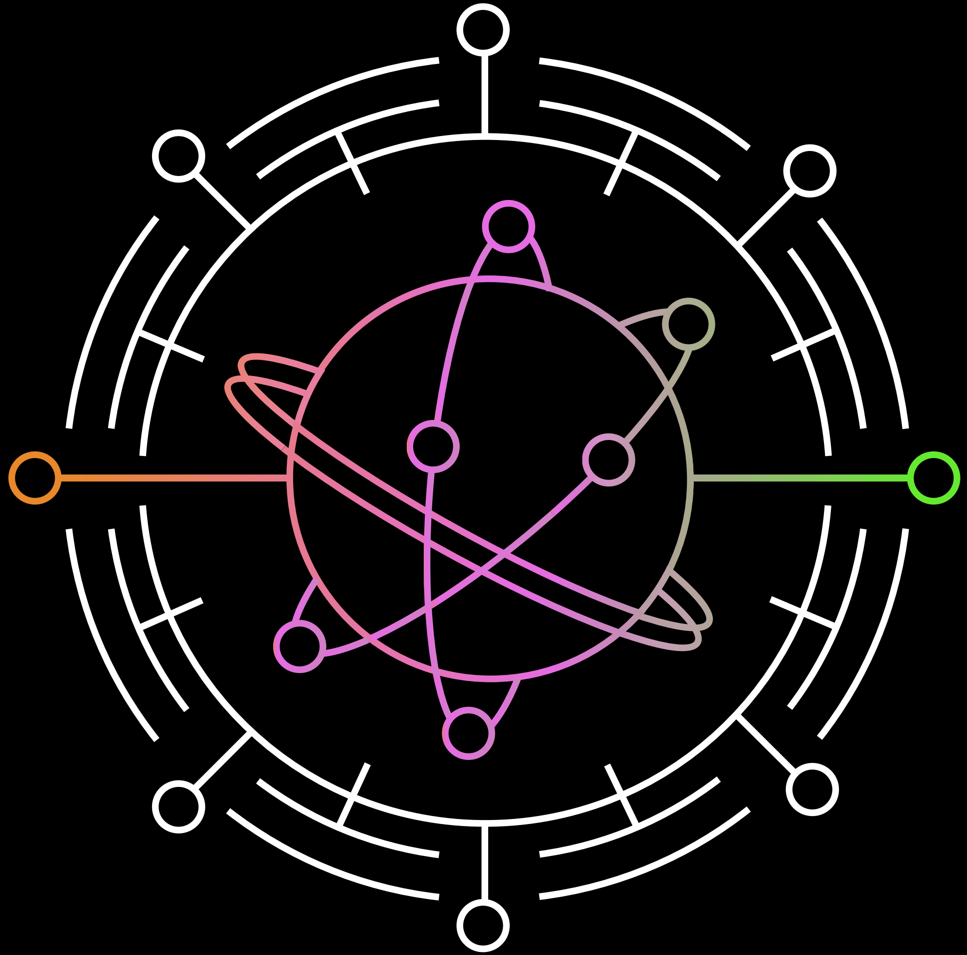
Transition event was in two phases over two weeks

Successfully migrated while completely remote

Zero customer impact across the world

Zero data loss

Additional items



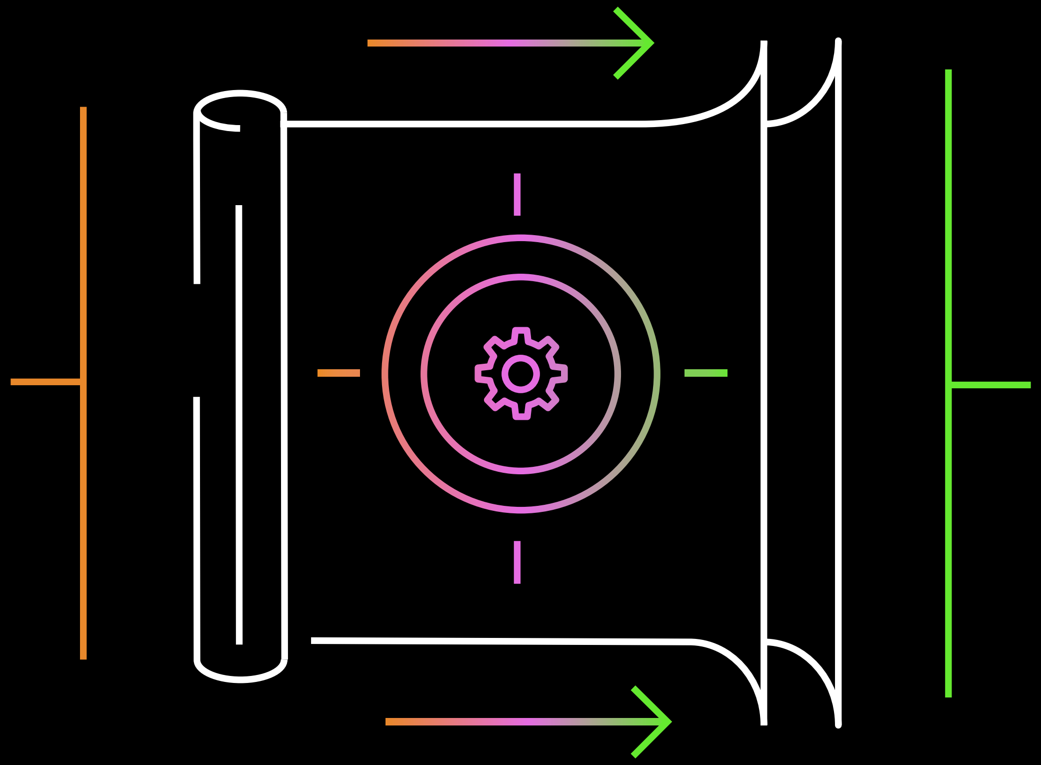
- Break out the parameter groups being used across clusters.
- Use Datadog to monitor long running transactions, if they are open longer than x amount of time, say 30 sec (this may need to be tuned).
- As a best practice it is good to set the "idle_in_transaction_session_timeout", in param group. By default, it is turned off in PostgreSQL 10. You can set the value so that it kills transactions open for x amount of time. In the PostgreSQL community, 10 seconds is commonly used.
- Monitor the Replica lag either in Cloudwatch or Datadog, this is a good metric to have on your dashboard and to see trending.
- On-going tuning - Go through often vacuumed tables and tweak vacuuming, tune vacuum configurations to handle tables with a high rate of change.

Future plans

AWS re:Invent

What's next?

- Eliminated several maintenance events with new architecture
- Migrate to newer versions of PostgreSQL
- Continue to optimize
- Begin to investigate AWS components like Amazon RDS Proxy and Amazon Aurora Global Database



Related sessions

DAT201

What's new in Amazon Aurora

DAT403

How Amazon Aurora helps you protect your data from mistakes

DAT404

Deep dive on Global Database for Amazon Aurora

AWS re:Invent

Thank you!

Eddie Maier

Director of Cloud Platforms
Hilton

Erick Dame

Sr. Solutions Architect
AWS

AWS re:Invent

Please complete the session survey

AWS re:Invent