AWS re:Invent

GAM304

Migrating the live Pokémon database to Aurora PostgreSQL

Chris Finch
Sr. SA Game Tech Evangelist
Amazon Web Services

Jeff Webb
Development Manager
The Pokémon Company International

David Williams
Sr. DevOps Engineer
The Pokémon Company International

AWS re:Invent

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws

Agenda

- Introduction
- A brief history
- The challenge
- The solution
- The results



Introduction

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Introduction

The Pokémon Company International (TPCi)

- Subsidiary of The Pokémon Company (TPC)
- Manages Pokémon property outside of Asia



Brand management

Localization

Trading card game

Marketing

Licensing

PokemonCenter.com

And, of course, engineering

A brief history

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Before Pokémon GO (pre-2016)

Pokemon.com was the focus of the small tech team and included:

Pokemon.com	Organized Play	Pokémon Trainer Club
Marketing and user engagement Pokémon TV	Trading card game league/ tournament management	User registration User profile management Authentication

Millions of accounts

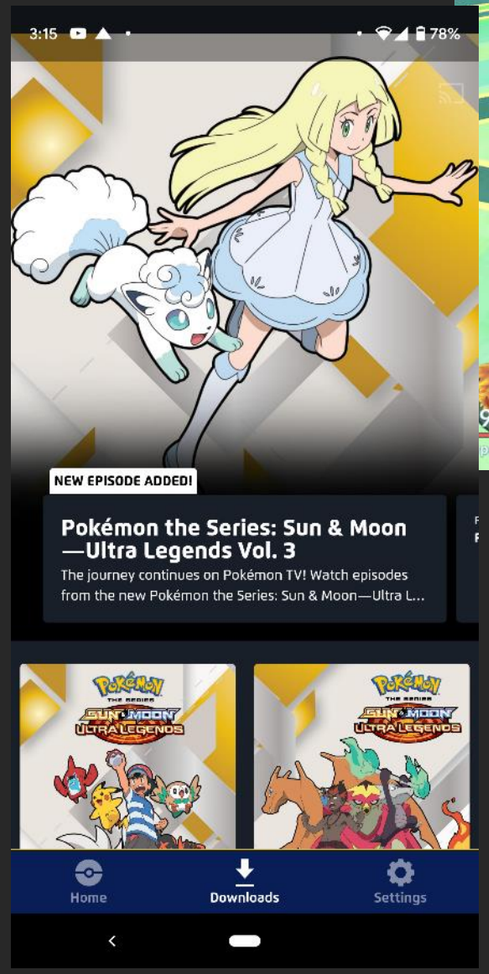
Used by Pokemon.com and a few smaller digital products

Pokémon Trainer Club Service (PTCS)

- Purpose: User registration and login system
  - COPPA
  - GDPR
- Size: Into the hundreds of millions
- Usage: Millions of logins a day

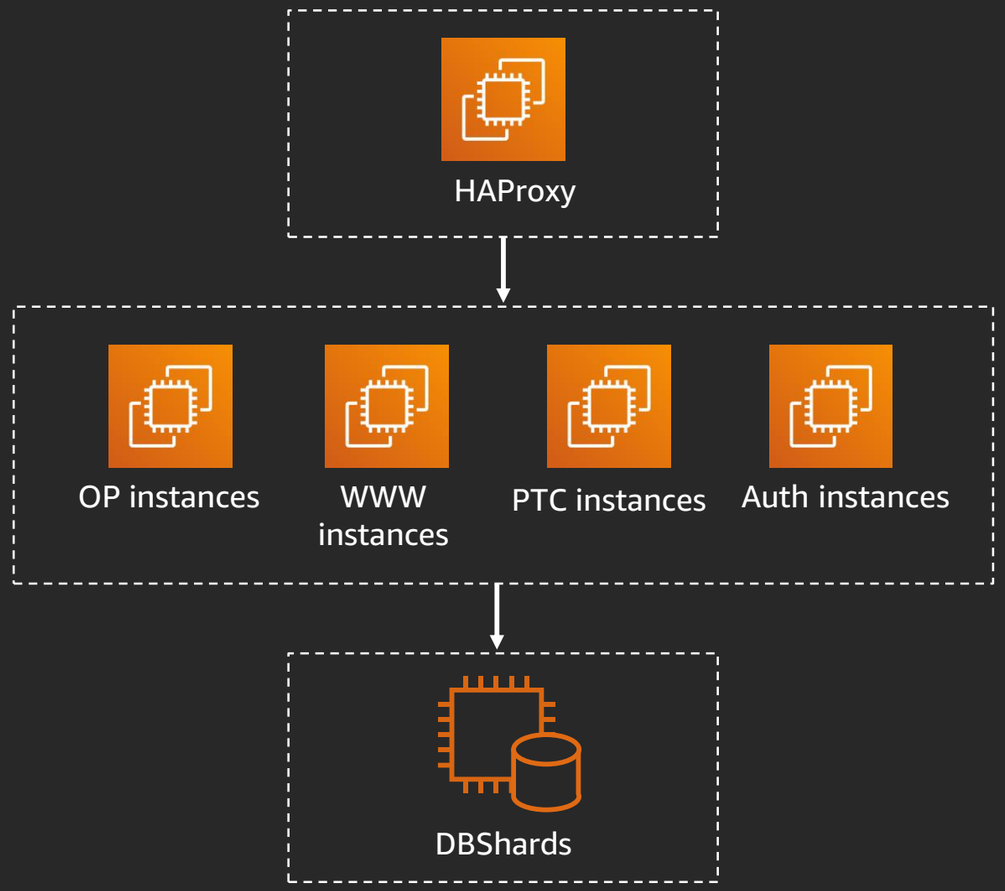




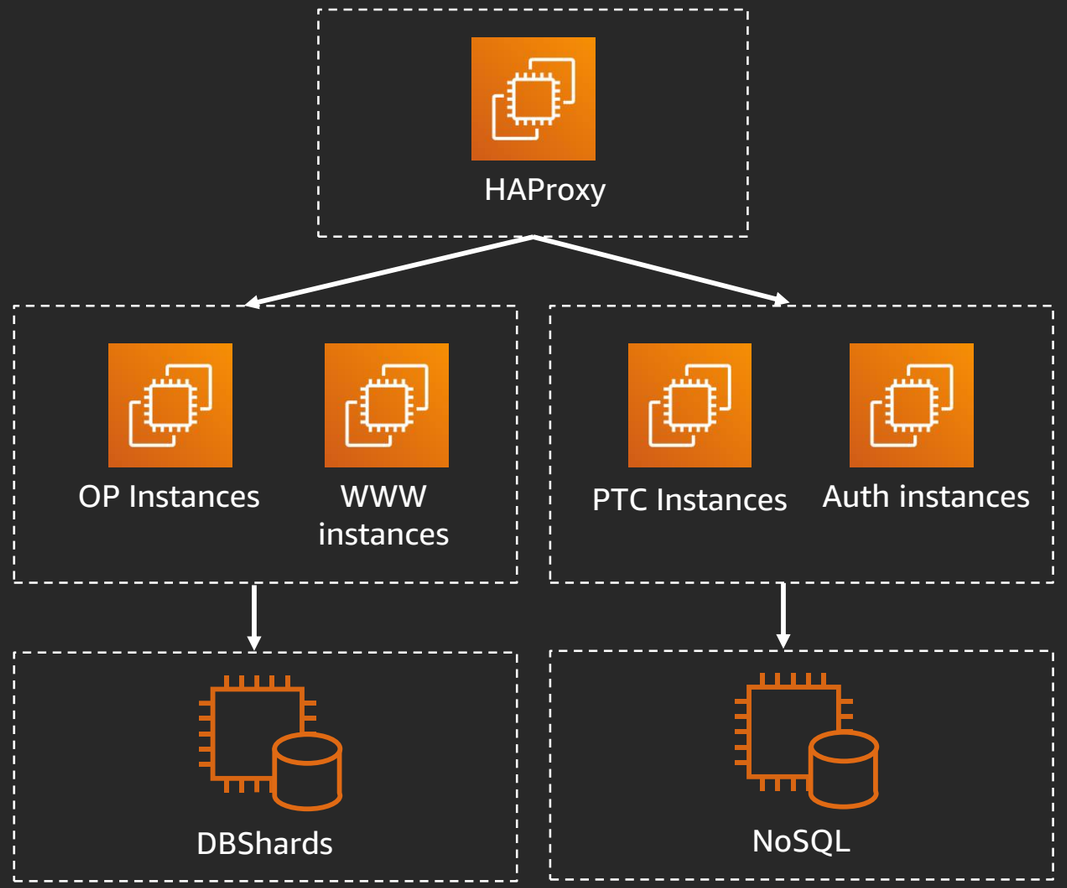


Preparing for Pokémon GO

- Lift & shift from co-lo to AWS Spring 2016



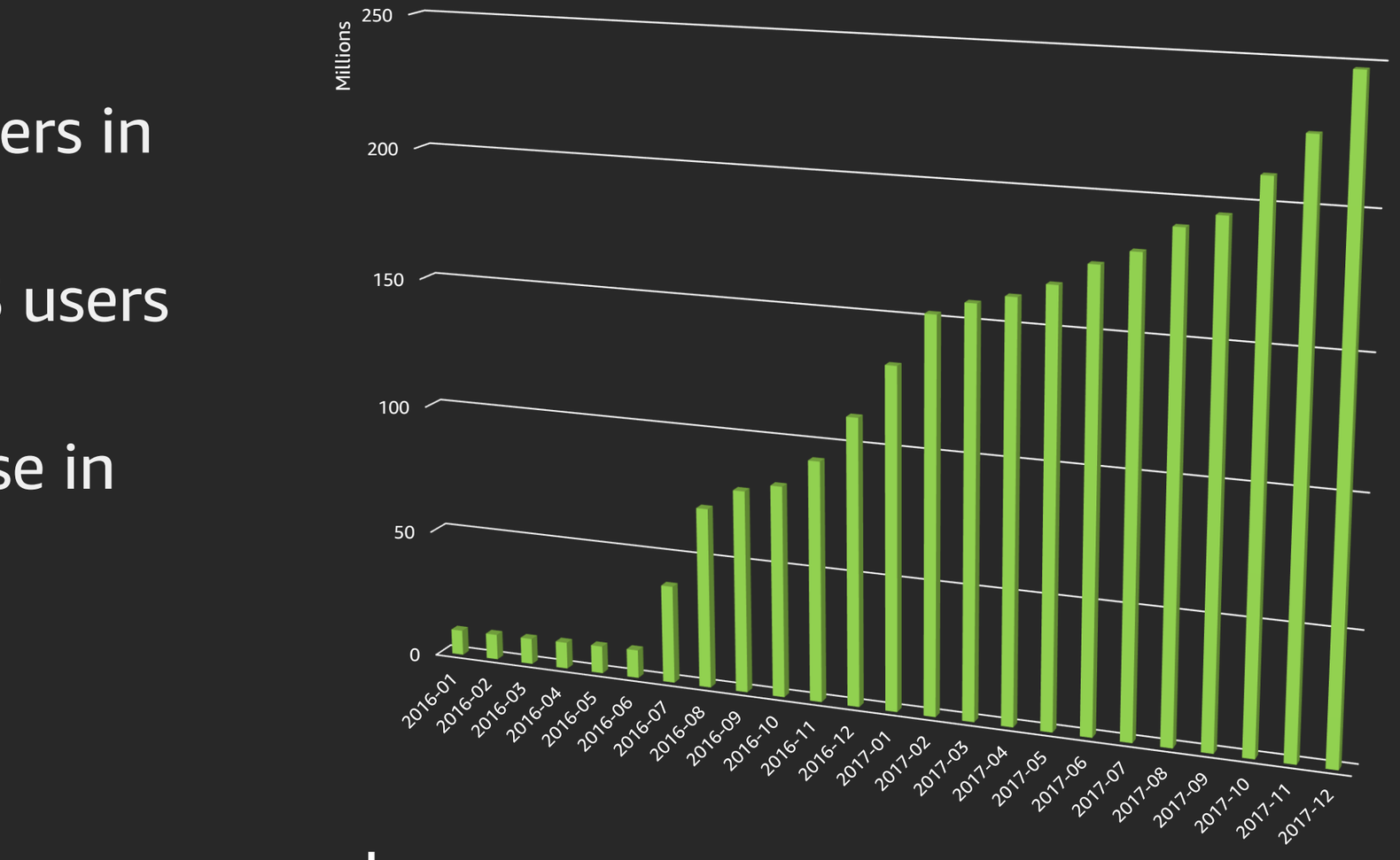
- Split PTC data out to NoSQL DB in preparation for GO



Pokémon GO launched (July 2016)

And everything changed

- 10x growth of PTCS users in six months
- 100% increase in PTCS users by end of 2017
- Additional 50% increase in users by end of 2018



Service and DB performance was good

The challenge

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Service stability issues: 2017/2018

Service and DB availability was not good

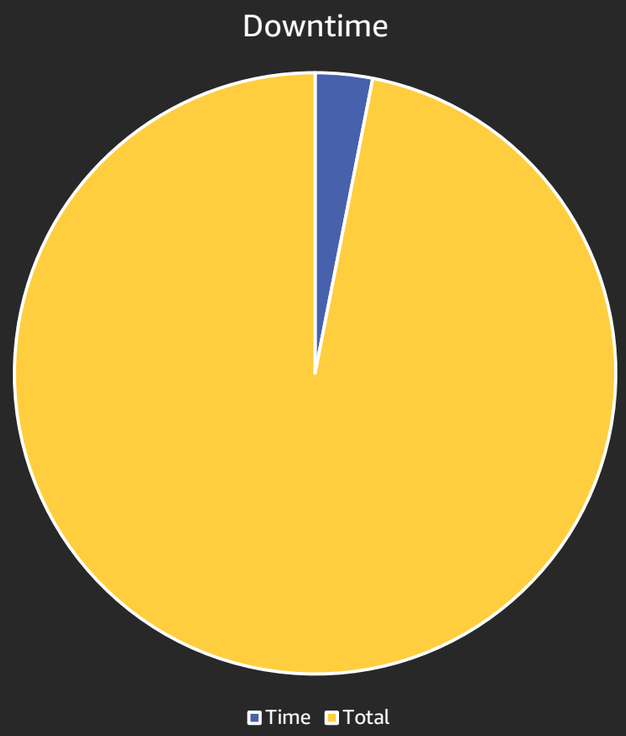
- Downtime: 137 hours (down or degraded) in six-month period

Licensing and infrastructure costs increasing

- 300 nodes required to support

Engineering time

- Full-time support from 1-2 resources



Business drivers for change

Instability of the DB platform was impacting customer and visitor experience

Future project success required achieving new goals

1. Stabilize our services & infrastructure for reduced downtime & customer impact
2. Reduce operational overhead of managing the DB platform
3. Reduce costs associated with our DB platform

Infrastructure drivers for change

Oversized EC2 instances

Duplicated infrastructure

- Maintain multiple redundant copies of data and indexes

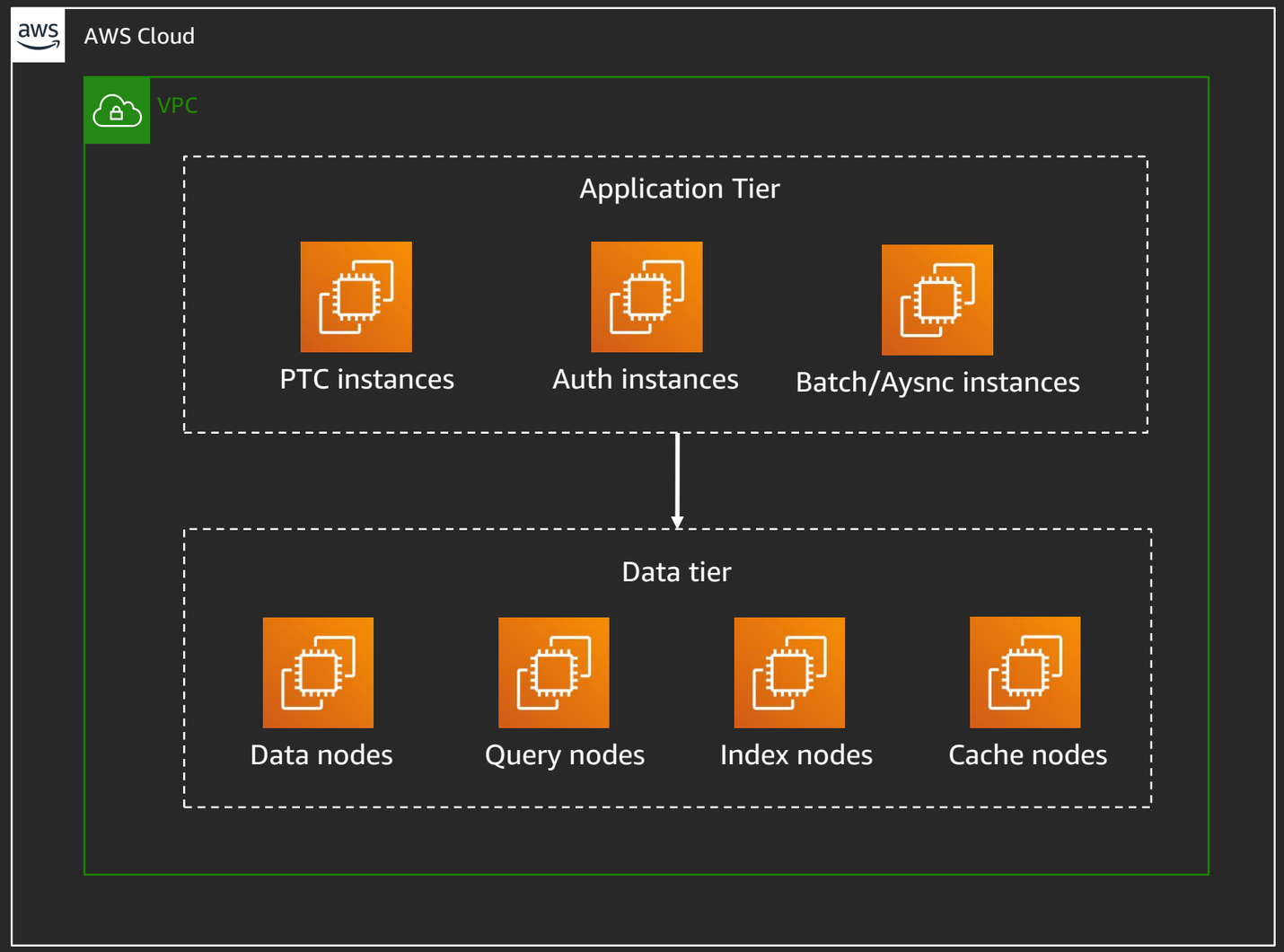
Operational overhead

- Routine activities no longer routine
- Unexpected behaviors: Amazon EC2 restarts, capacity, etc.
- DB backups
- Patching/upgrading became unmanageable



Data tier architecture

- Hundreds of DB instances across several roles
- All deployed with Amazon EC2 Auto Scaling groups
- One datastore for all data



The solution

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Design goals

Leverage managed services

- Optimize resource utilization for our core business

Use appropriate datastore for data

- Event data
- User data
- Configuration/TTL data

High availability, stability, performance

Reduce cost & right-size infrastructure

- Only want to use and pay for what we need



Choosing Amazon Aurora PostgreSQL

	Pros	Cons
Amazon DynamoDB	Tier 1 AWS service Multi-region Highly scalable Easier lift from JSON?	Encryption (at the time) Internal expertise
Aurora MySQL	Internal expertise Encryption Feature rich	Nascent tech Multi-region JSON to relational ETL
Aurora PostgreSQL	Internal expertise Encryption DB internals Native support for JSON	Feature lag Multi-region



Data-driven approach

Acceptance criteria

- Authentication: 2k/sec
- User signups: 60+/sec
- Admin bulk queries

Type	Name	# requests	# fails	Median (ms)	Average (ms)	Min (ms)	Max (ms)	Content Size	# reqs/sec
GET	1a_post_login_page	4	0	950	1360	895	2559	0	1.5
GET	1b_login	5	0	740	749	607	883	99	1
POST	2_authorize_callback	4	0	700	790	626	975	0	1
POST	3_get_access_token	3	0	800	850	718	1036	108	0
POST	4_login	5	0	740	774	713	880	208	1
INSERT	insert_insert	0	23	0	0	0	0	0	0
UPDATE	update_insert	28	0	0	0	0.0033578872680664062	0.00949501991271972	0	5
	Total	**49**	**23**	**0**	**383**	**713**	**2559**	**38**	**9.5**

Testing

- User generation: 200m
- Performance test suite: Burst, soak testing

Iterate

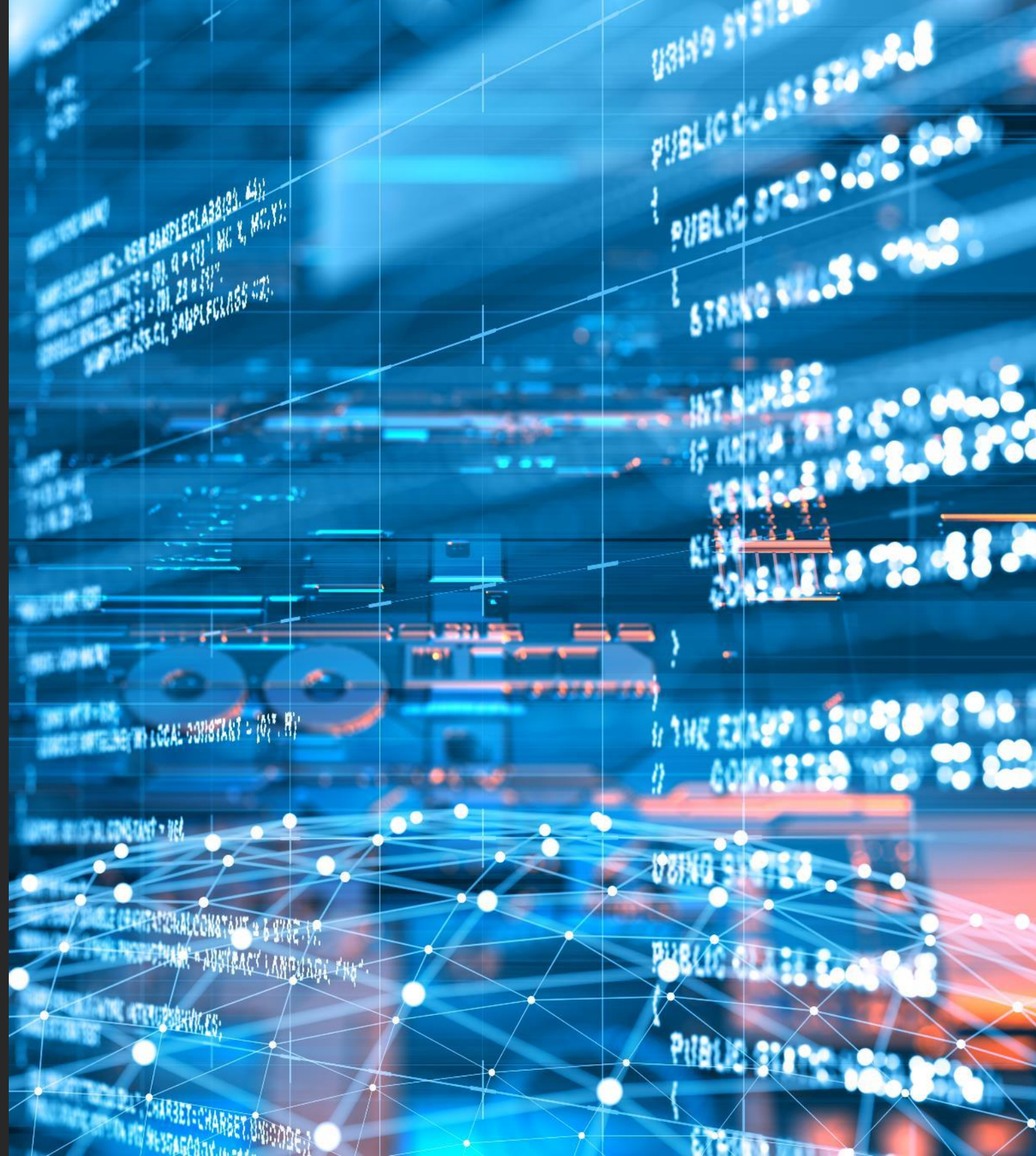
- Rework schema
- Rerun tests

The migration plan

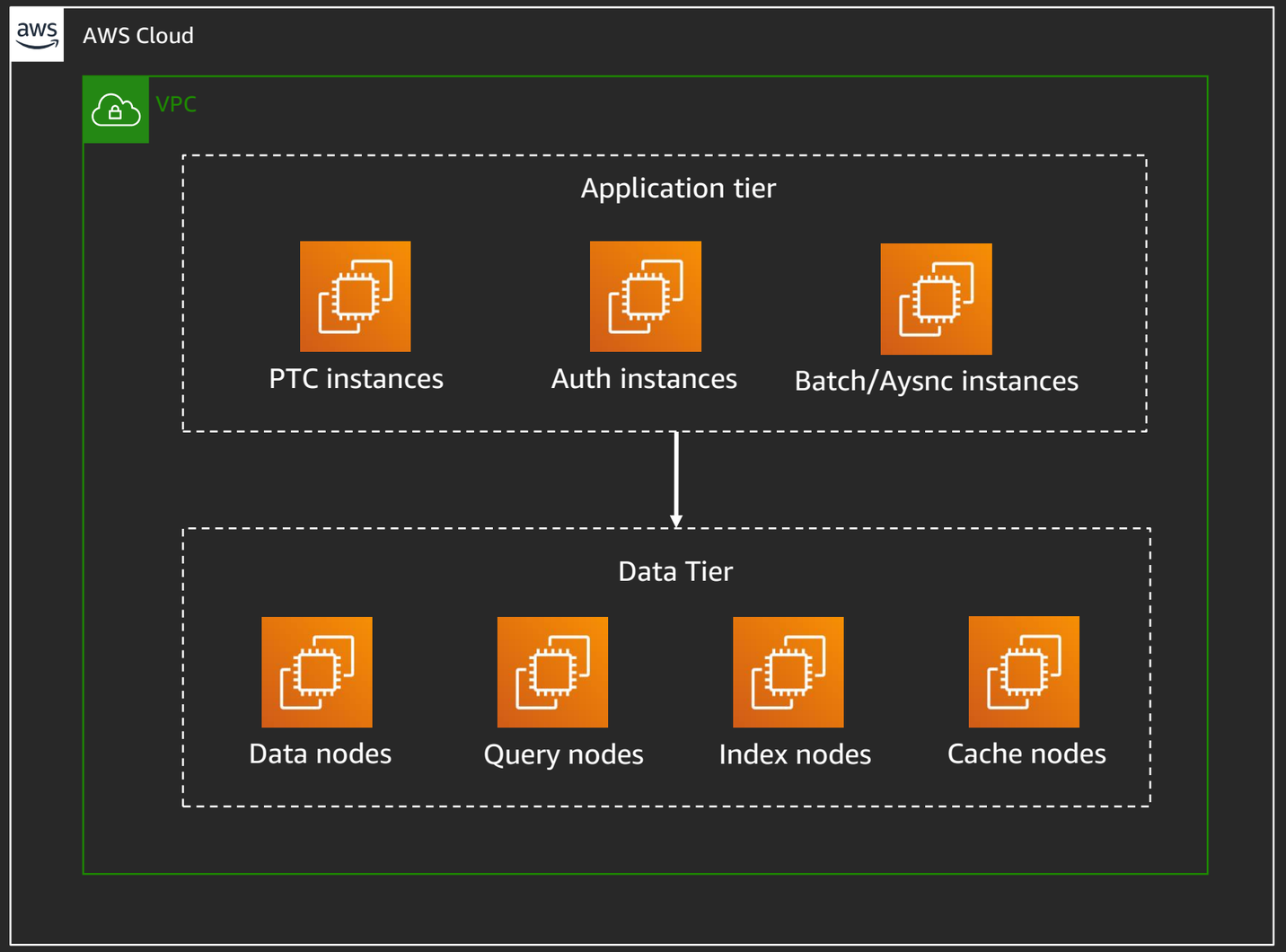
An iterative approach

- Improve cache stability
- Migrate out TTL and configuration data
- Stream event data

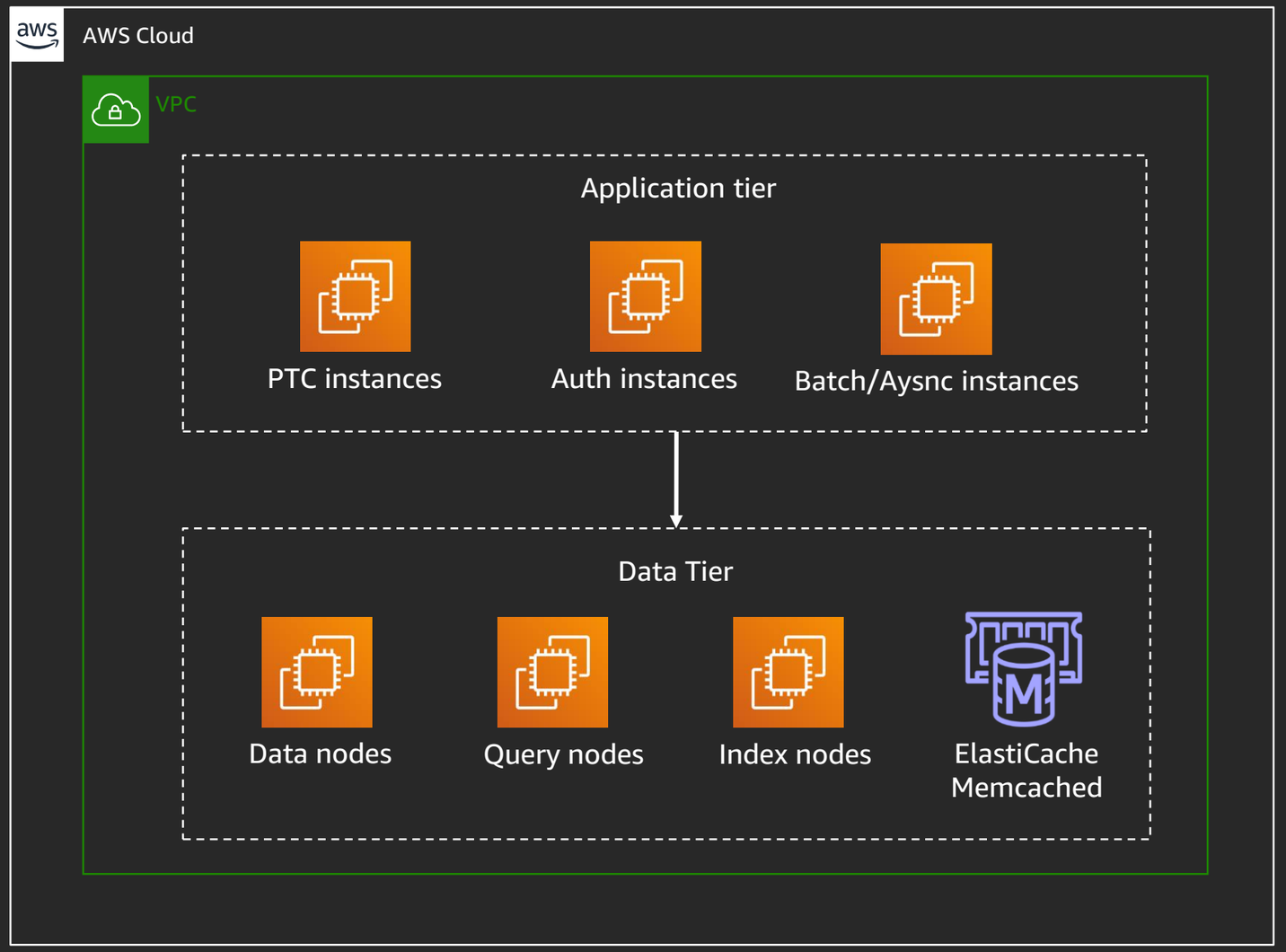
Each phase should independently deliver value



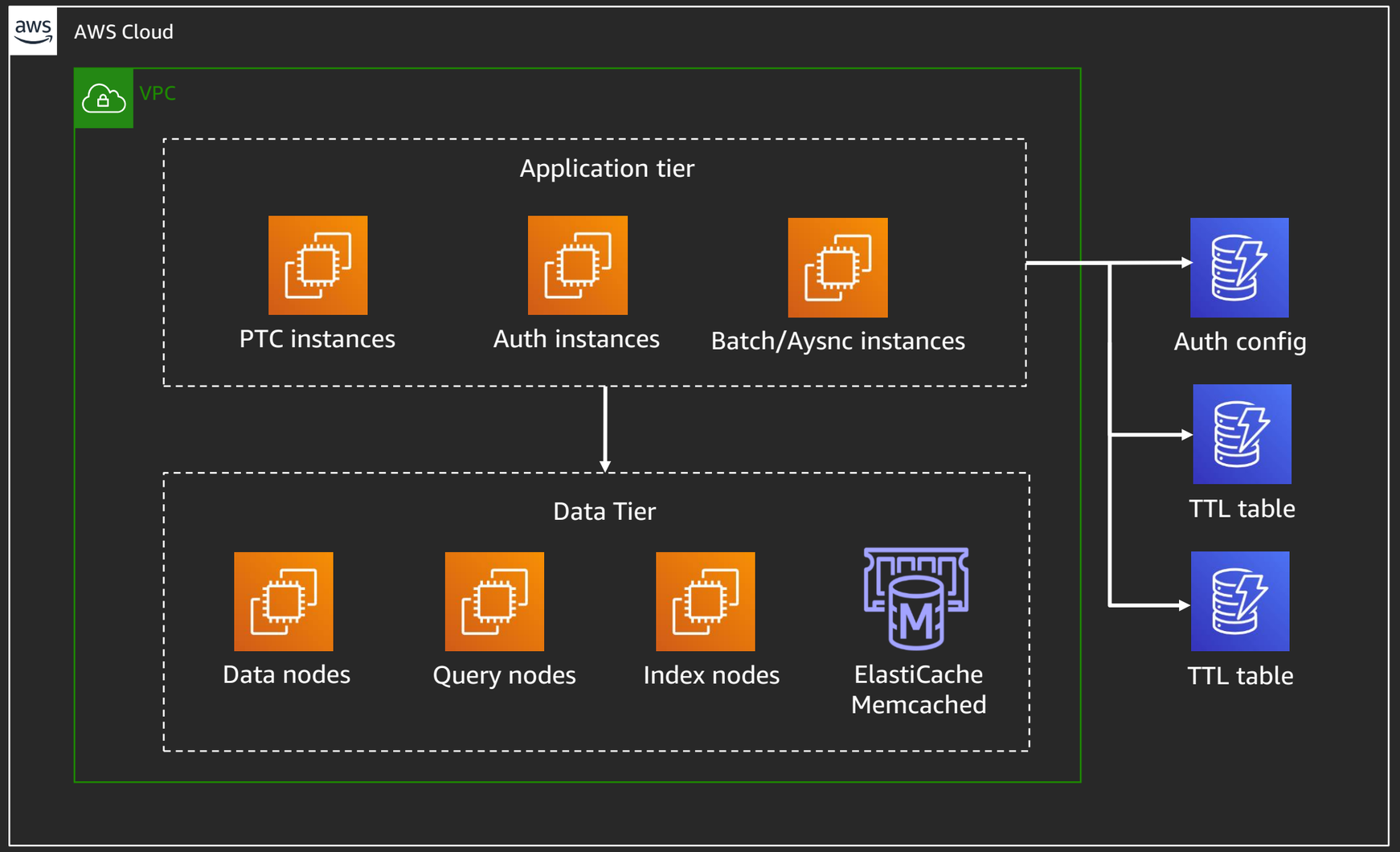
Migration phases



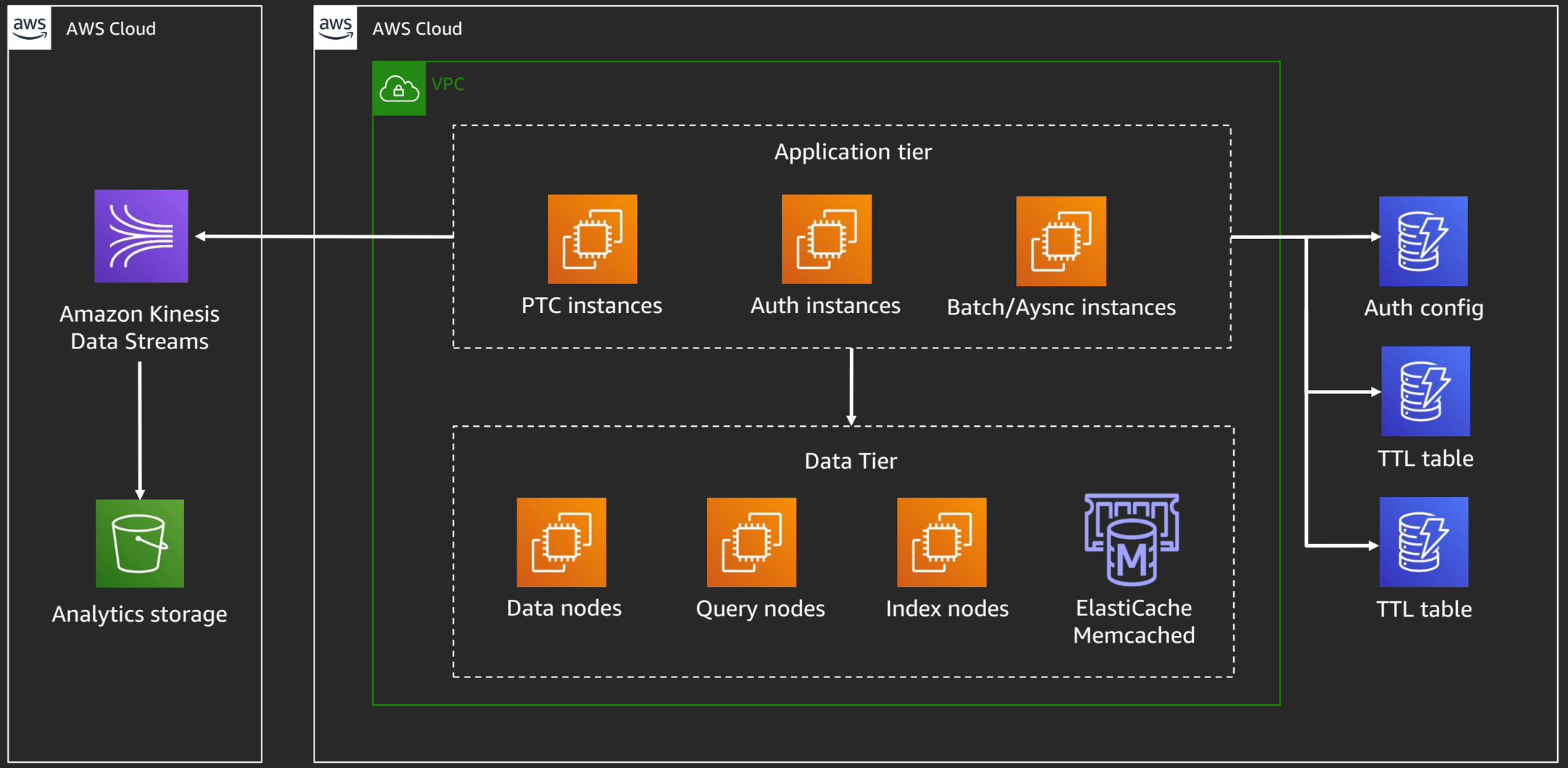
Migration phases



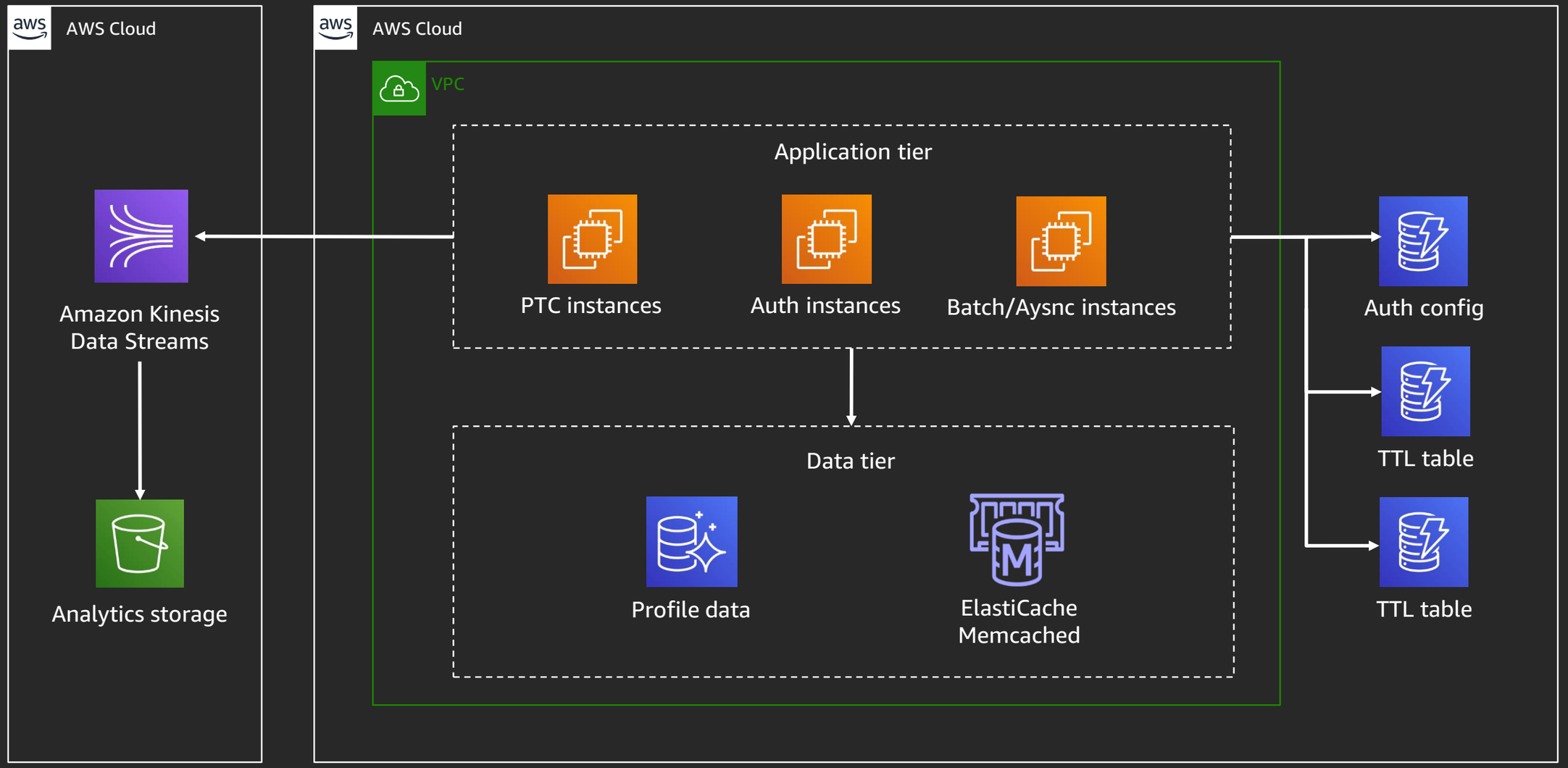
Migration phases



Migration phases



Migration phases



Planning for Aurora PostgreSQL

AWS Professional Services to bridge the knowledge gap

- Validate our schema design and provide feedback
- Advice on how to tune DB parameter groups
- Tools for planning monitoring and tuning

Parameters

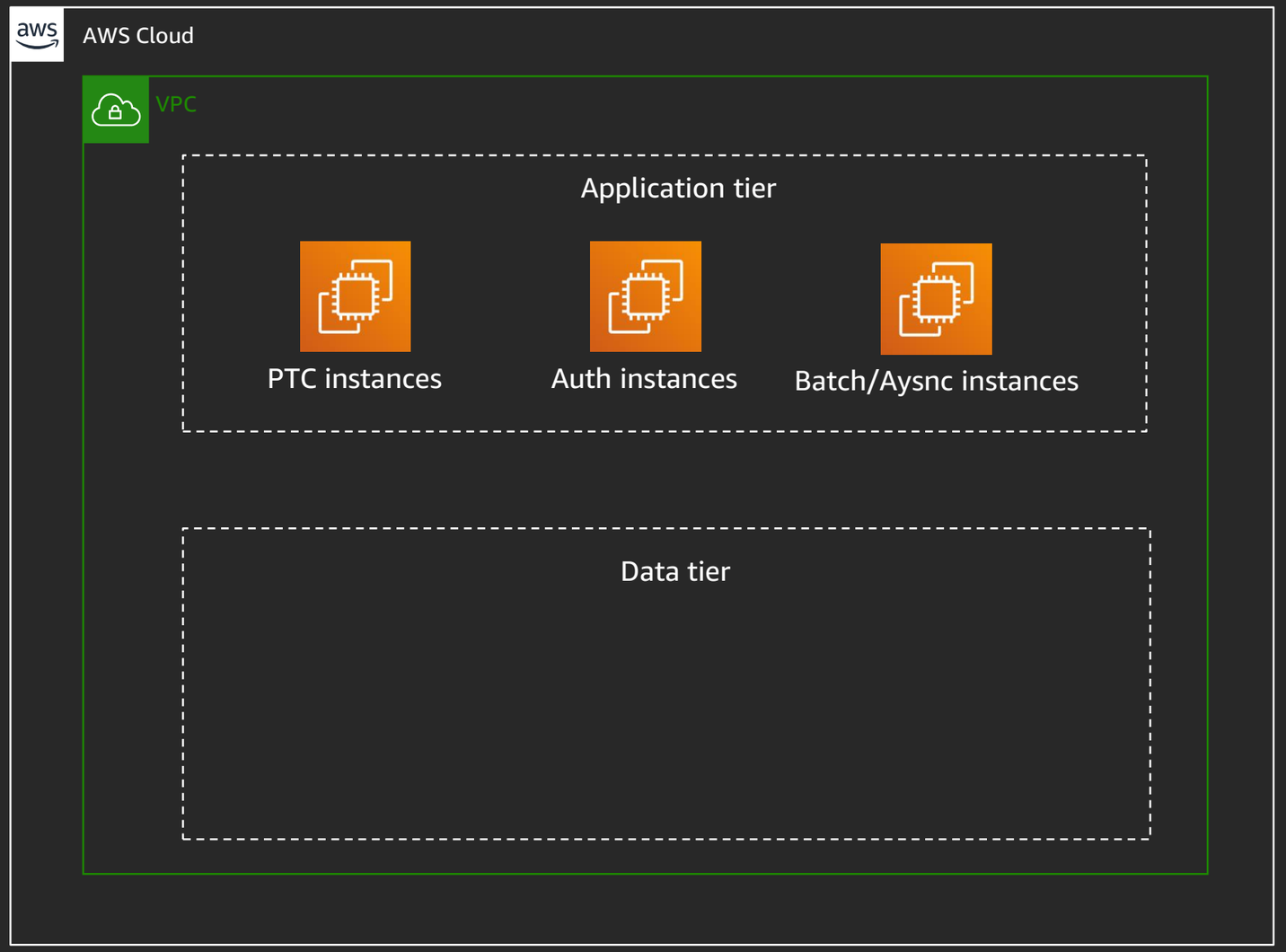
Edit parameters

autovacuum_

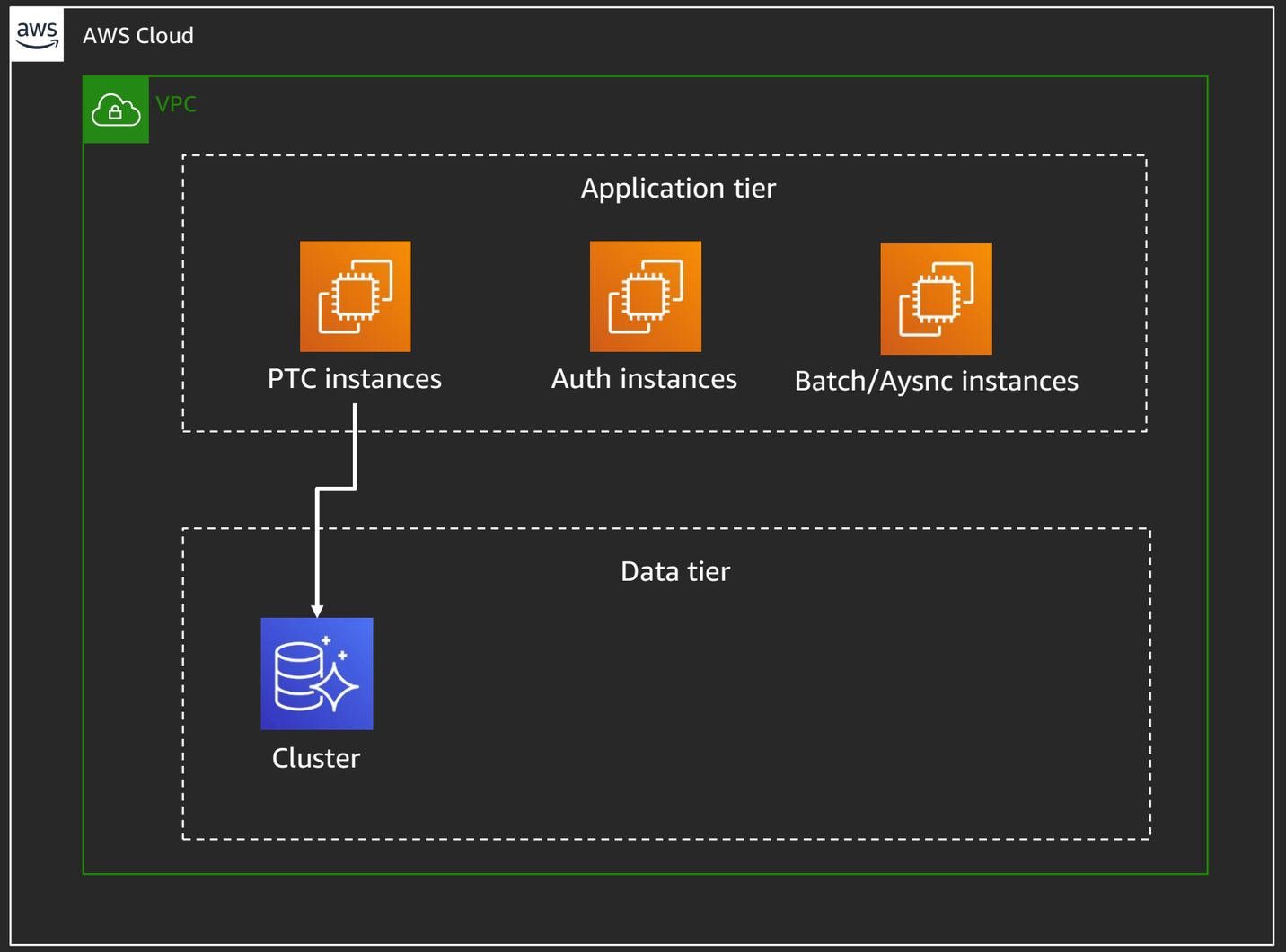
1

Name	Values	Allowed values	Modifiable	Source	Apply type	Data type	Description
autovacuum_analyze_threshold		0-2147483647	true	engine-default	dynamic	integer	Minimum number of tuple inserts, updates or deletes prior to analyze.
autovacuum_freeze_max_age		100000000-750000000	true	engine-default	static	integer	Age at which to autovacuum a table to prevent transaction ID wraparound.
autovacuum_multixact_freeze_max_age		10000000-2000000000	true	engine-default	static	integer	Multixact age at which to autovacuum a table to prevent multixact wraparound.
autovacuum_vacuum_cost_delay	5	-1-100	true	engine-default	dynamic	integer	(ms) Vacuum cost delay in milliseconds, for autovacuum.
autovacuum_vacuum_cost_limit		-1-10000	true	engine-default	dynamic	integer	Vacuum cost amount available before napping, for autovacuum.
autovacuum_vacuum_threshold		0-2147483647	true	engine-default	dynamic	integer	Minimum number of tuple updates or deletes prior to vacuum.
autovacuum_work_mem		-1-2147483647	true	engine-default	dynamic	integer	(kB) Sets the maximum memory to be used by each autovacuum worker process.
log_autovacuum_min_duration		-1-2147483647	true	engine-default	dynamic	integer	(ms) Sets the minimum execution time above which autovacuum actions will be logged.

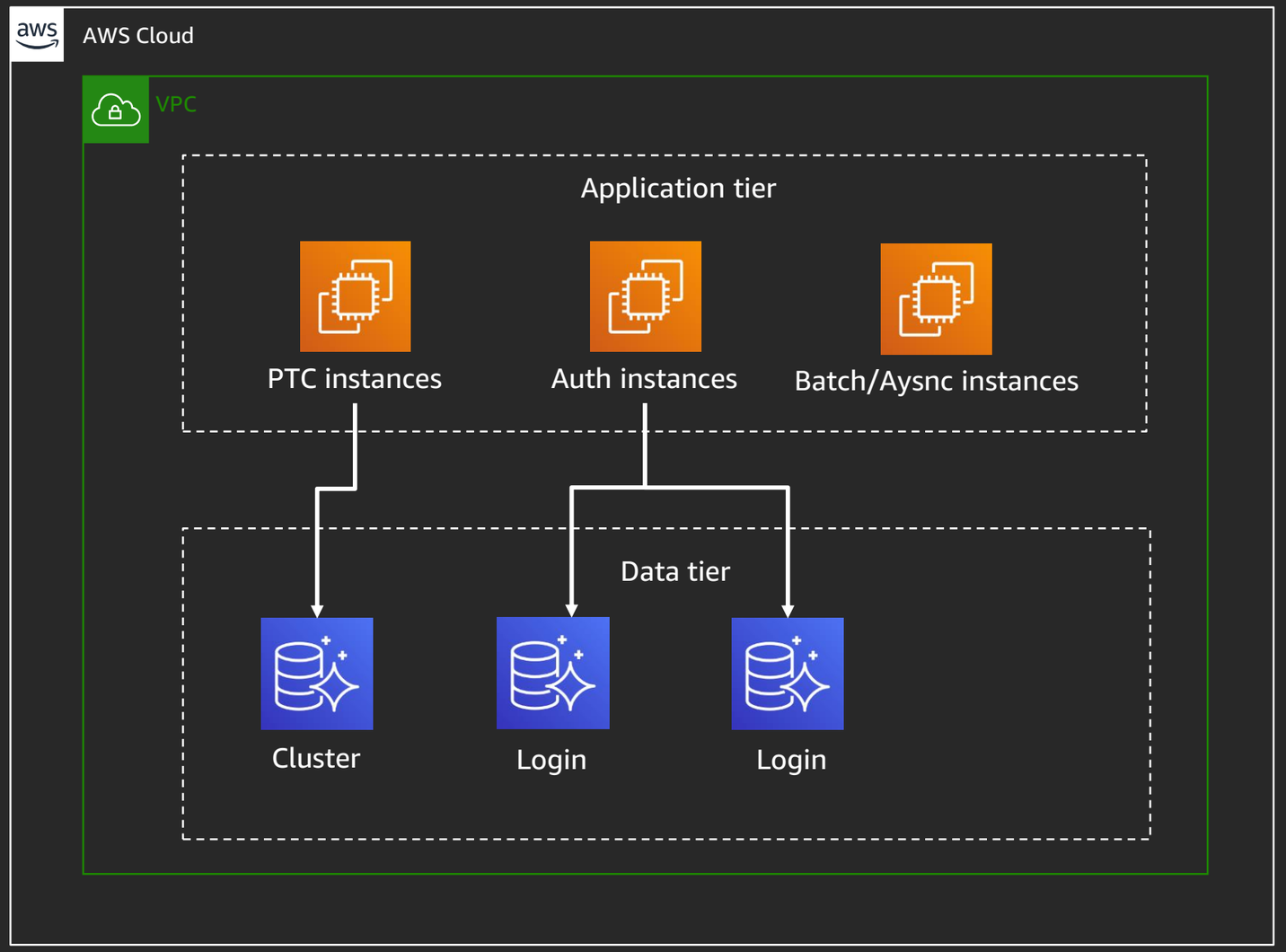
Aurora PostgreSQL cluster design



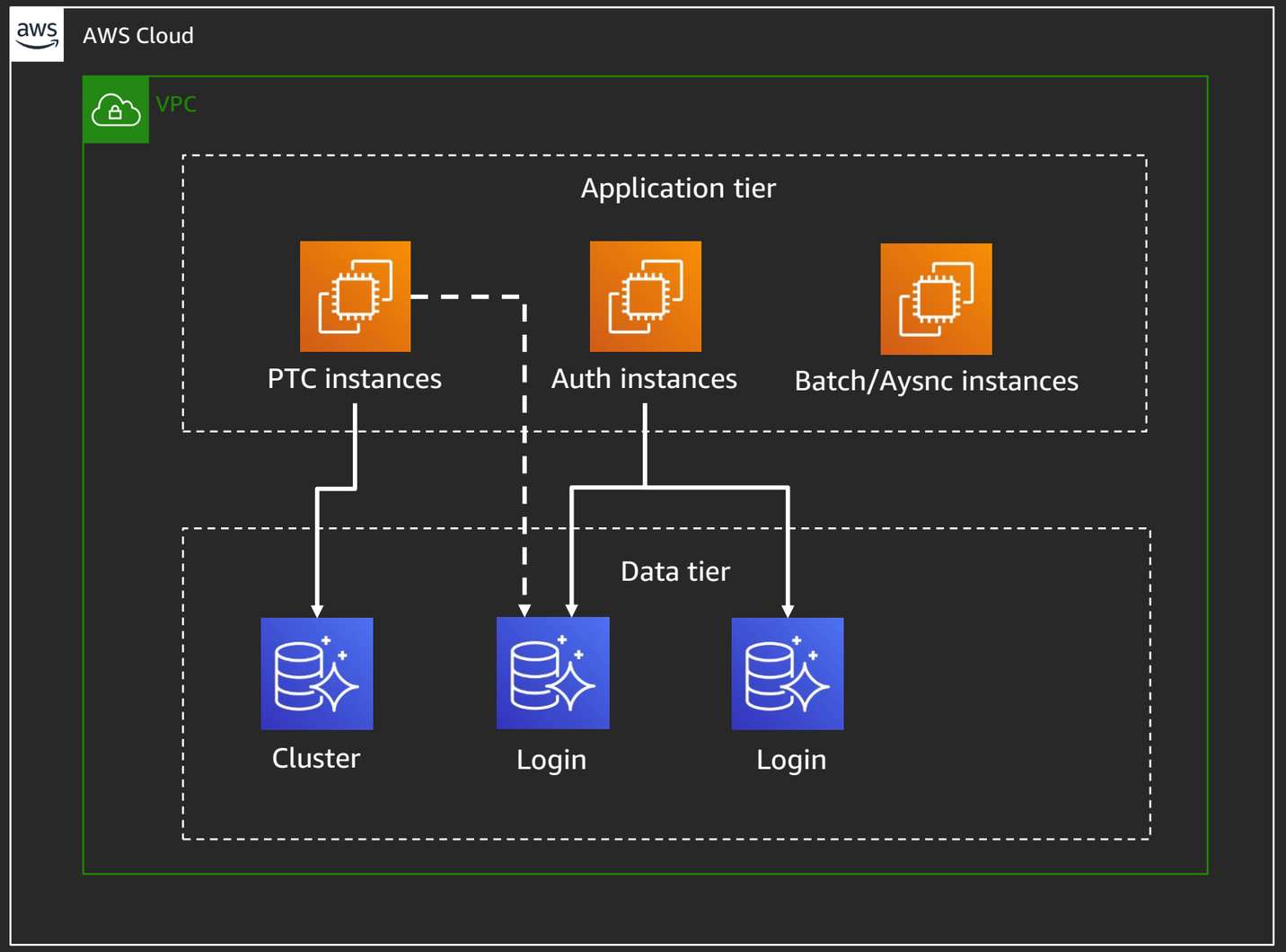
Aurora PostgreSQL cluster design



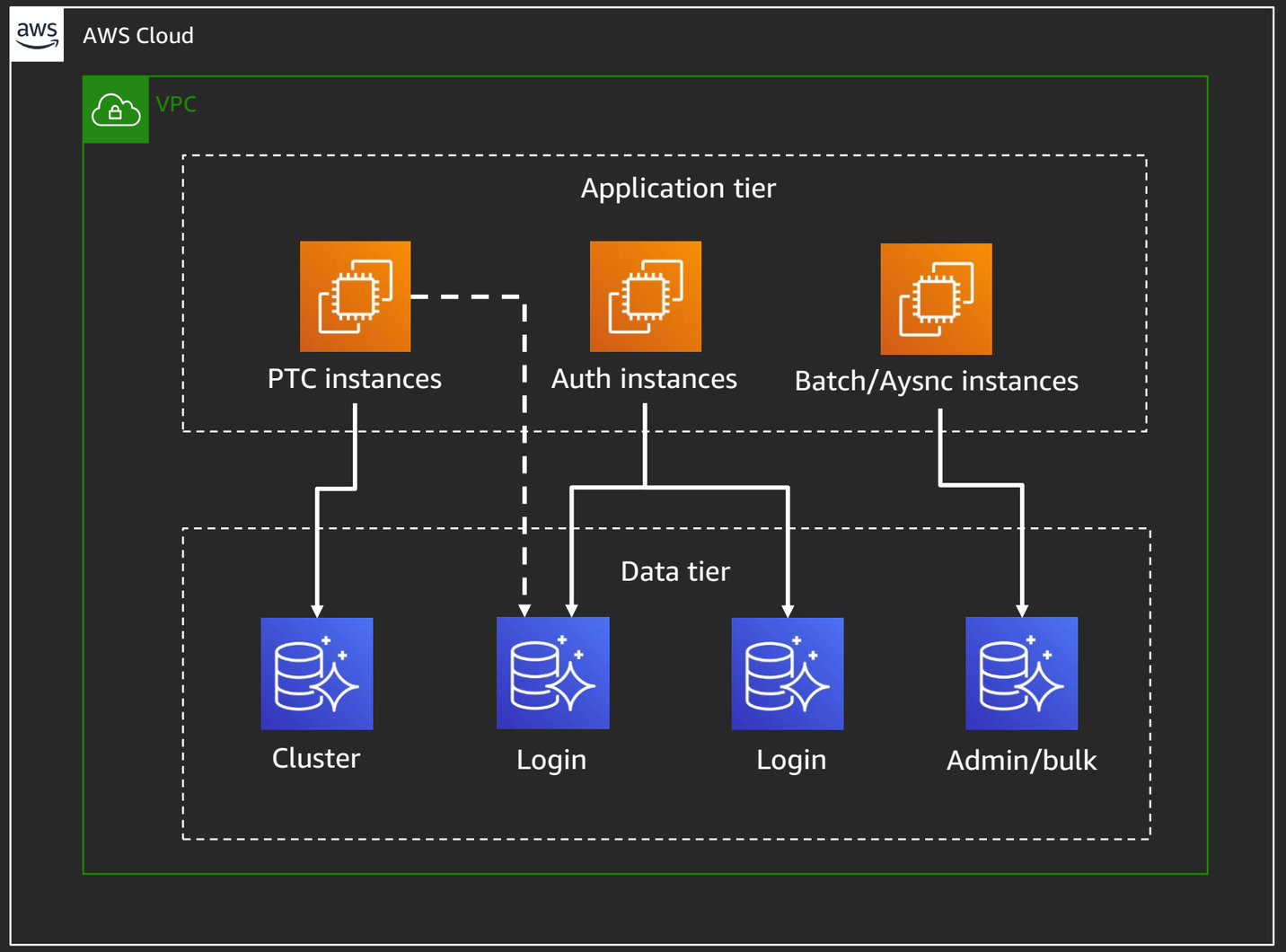
Aurora PostgreSQL cluster design



Aurora PostgreSQL cluster design



Aurora PostgreSQL cluster design



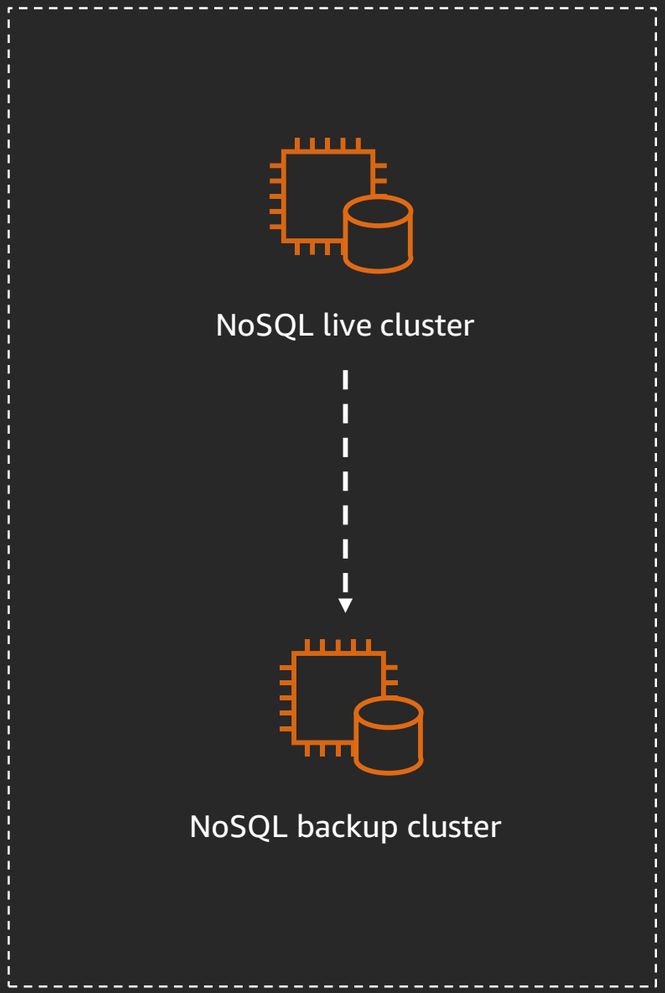
The migration: Extract-Transform-Load (ETL)

Transform & load

- ”Pretty easy”
- Abandon users that had never activated
- Minor data changes, nothing structural

Extract

- Leverage NoSQL cluster architecture
- Map-reduce to find users not extracted
- Extraction process marks users as such in backup cluster
- Any user changes in production would overwrite changes in backup cluster



ETL – Test and Iterate

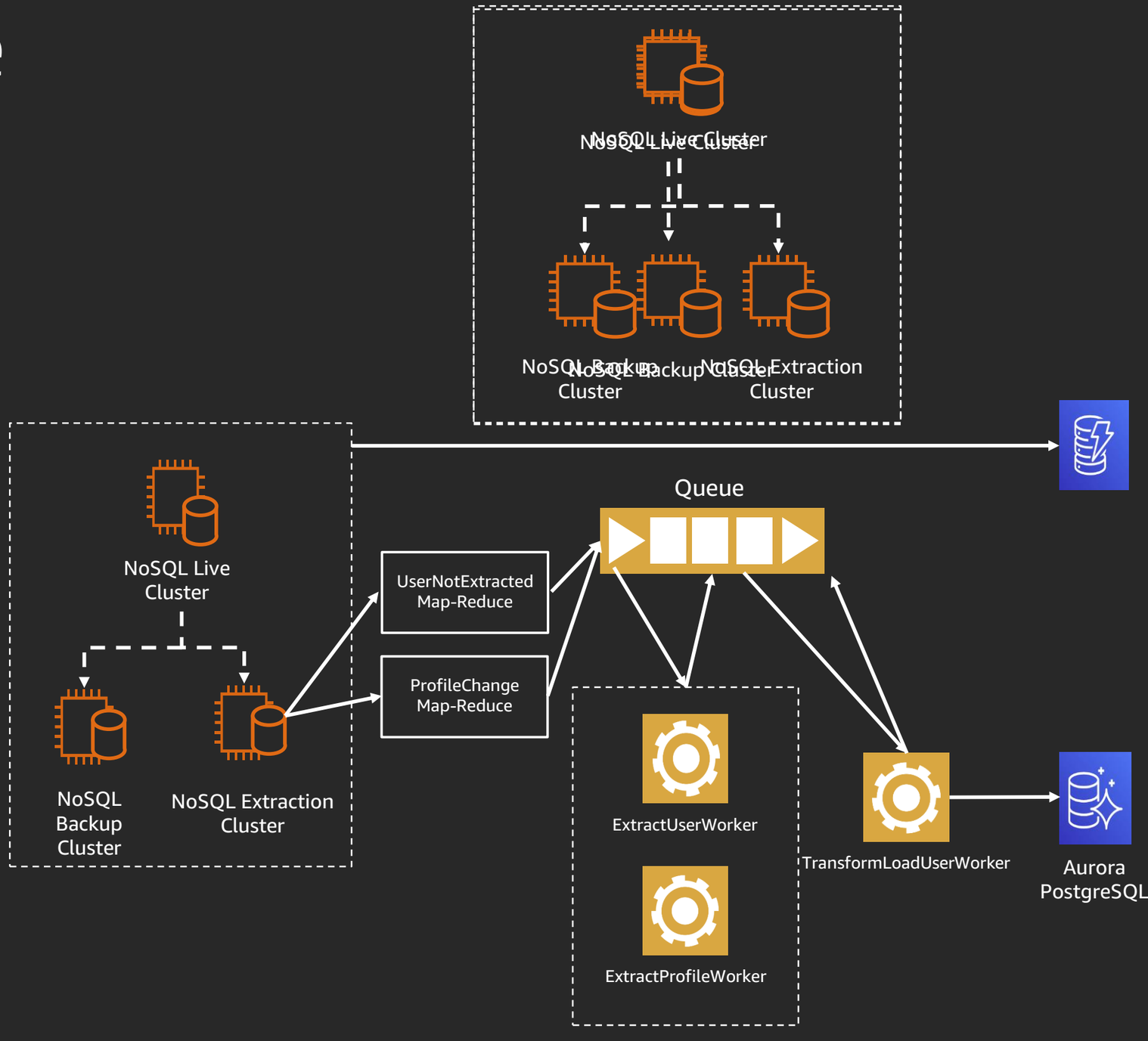
Test

- 11m user multi-cluster test setup
- Dozens of test runs
- Test cases – inactive users, user updates

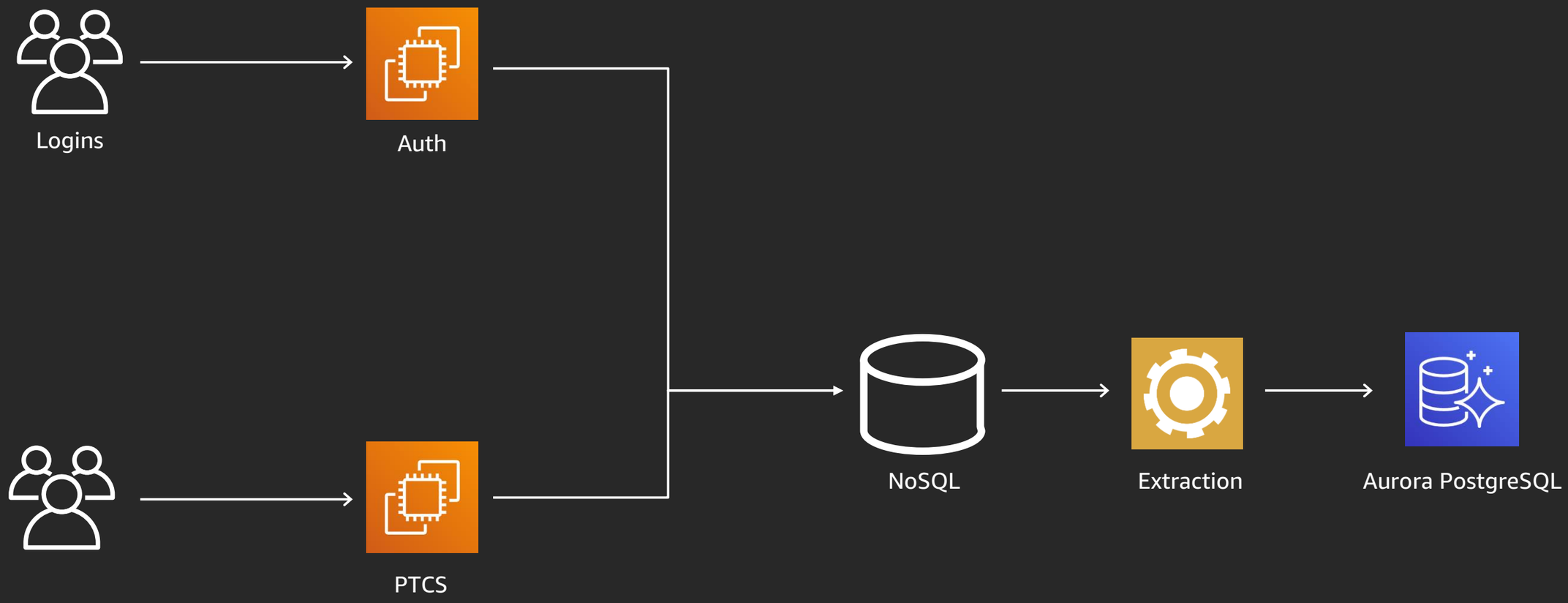
~2% of documents were not overwritten

And iterate

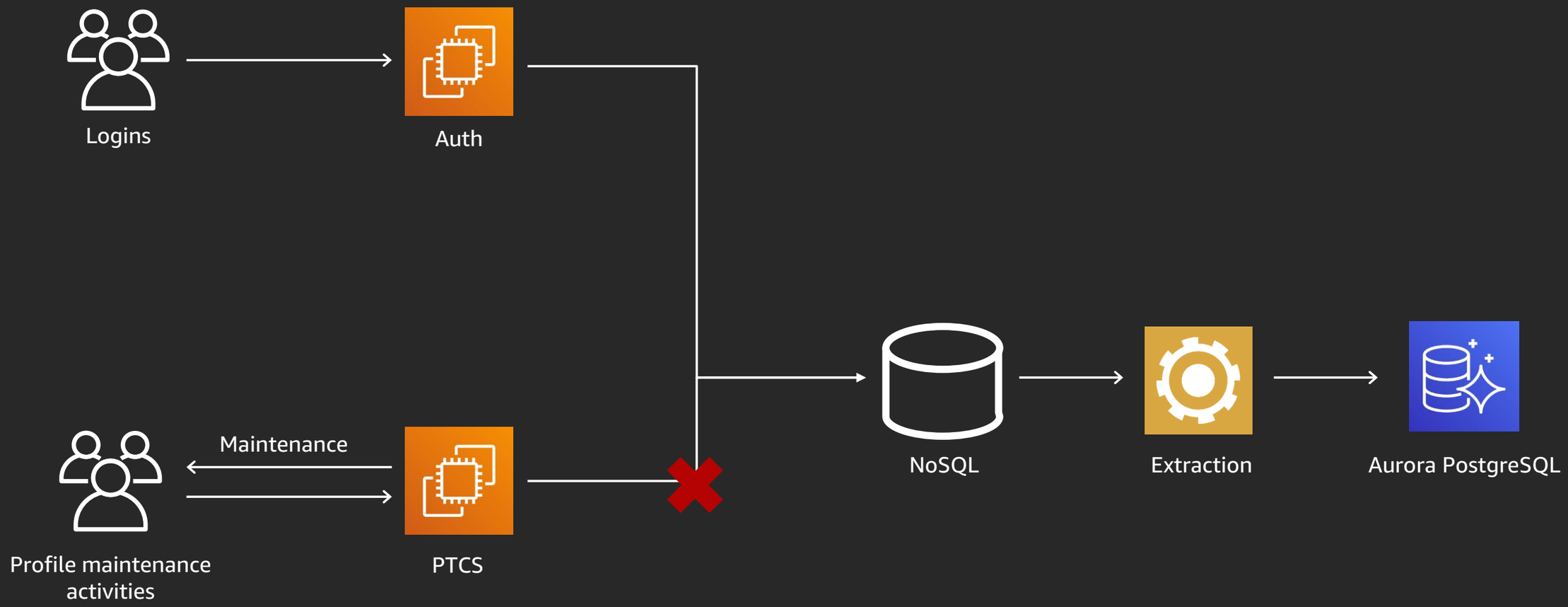
- User profile change documents
- Third cluster



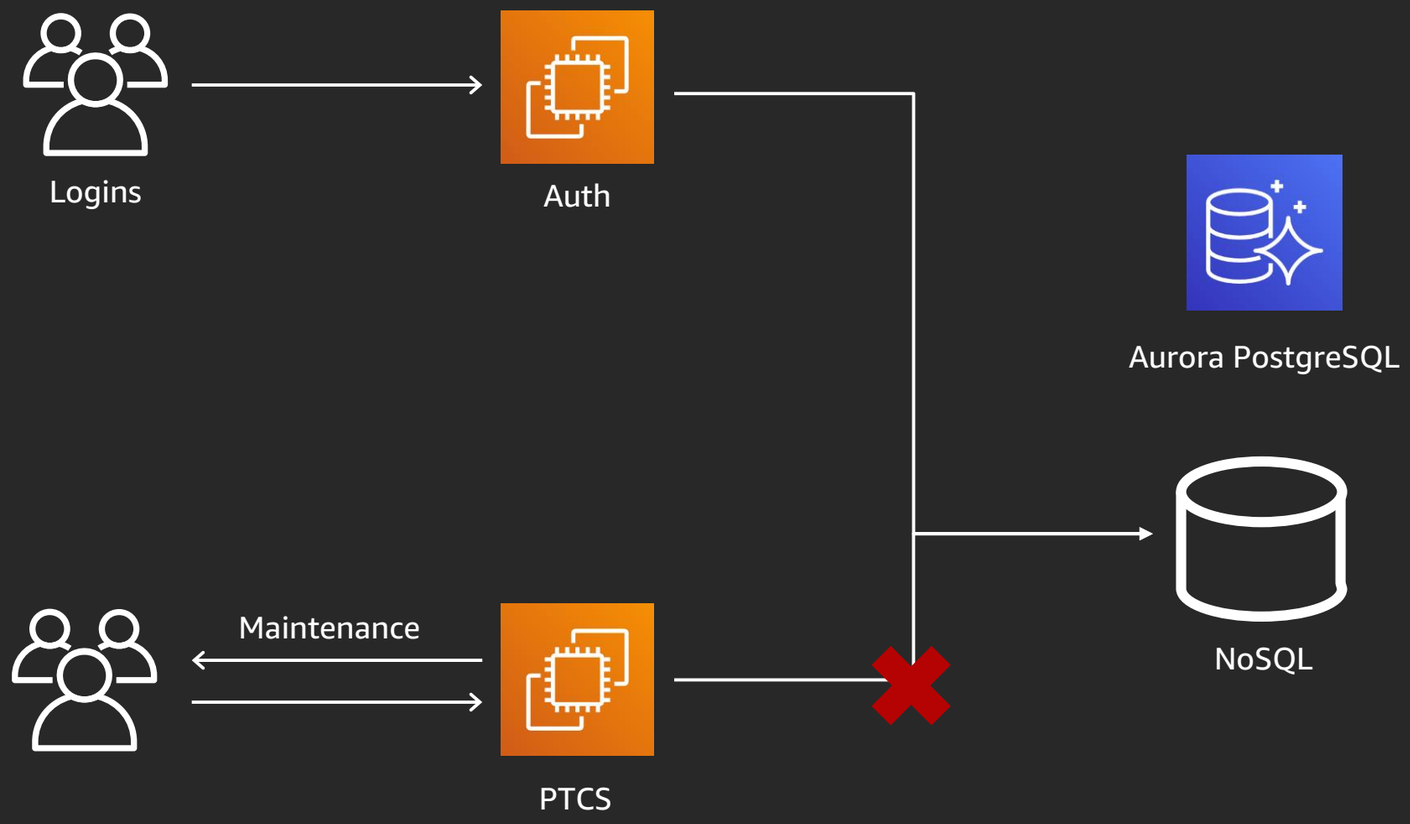
Migration Day



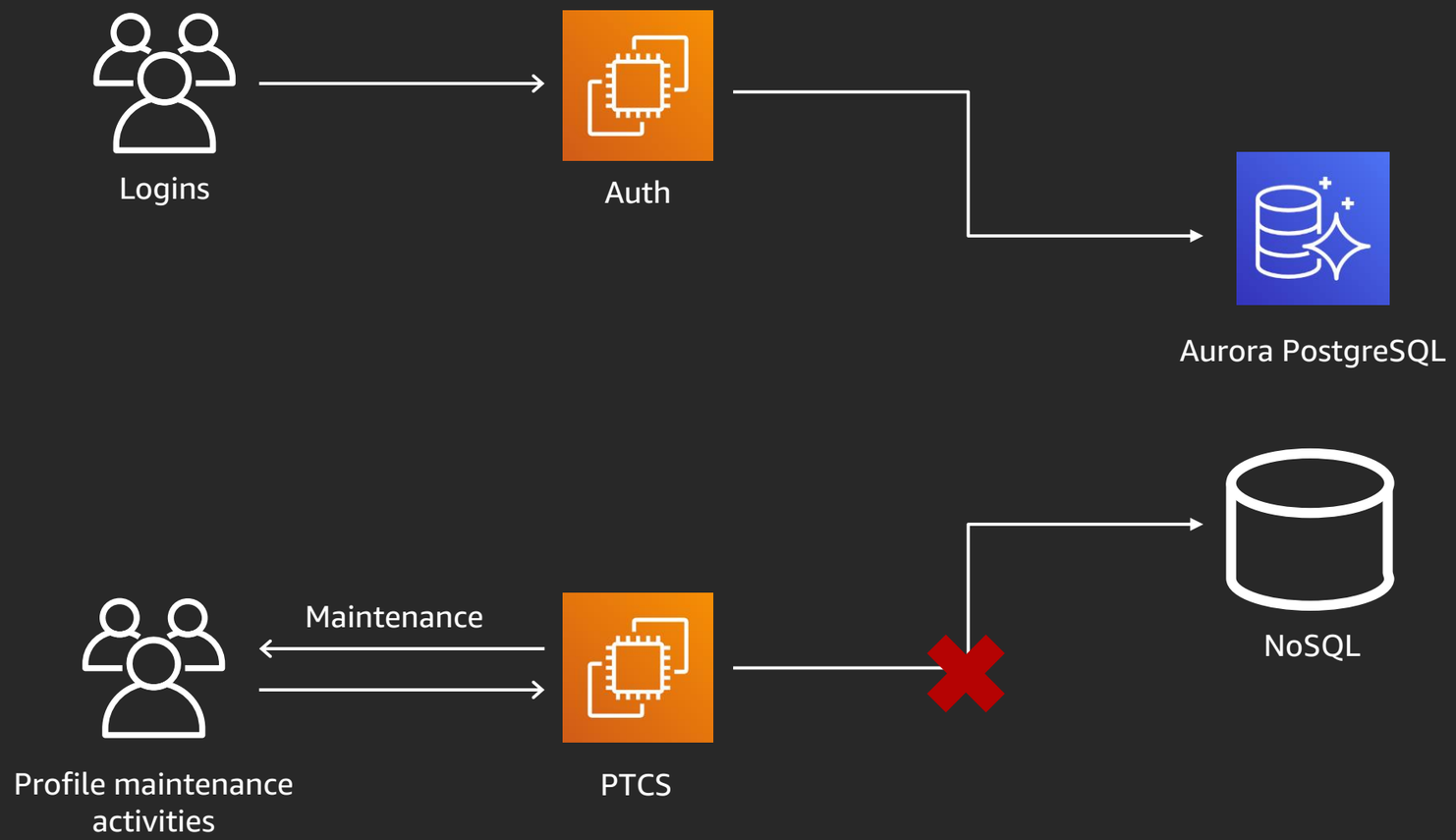
Migration Day



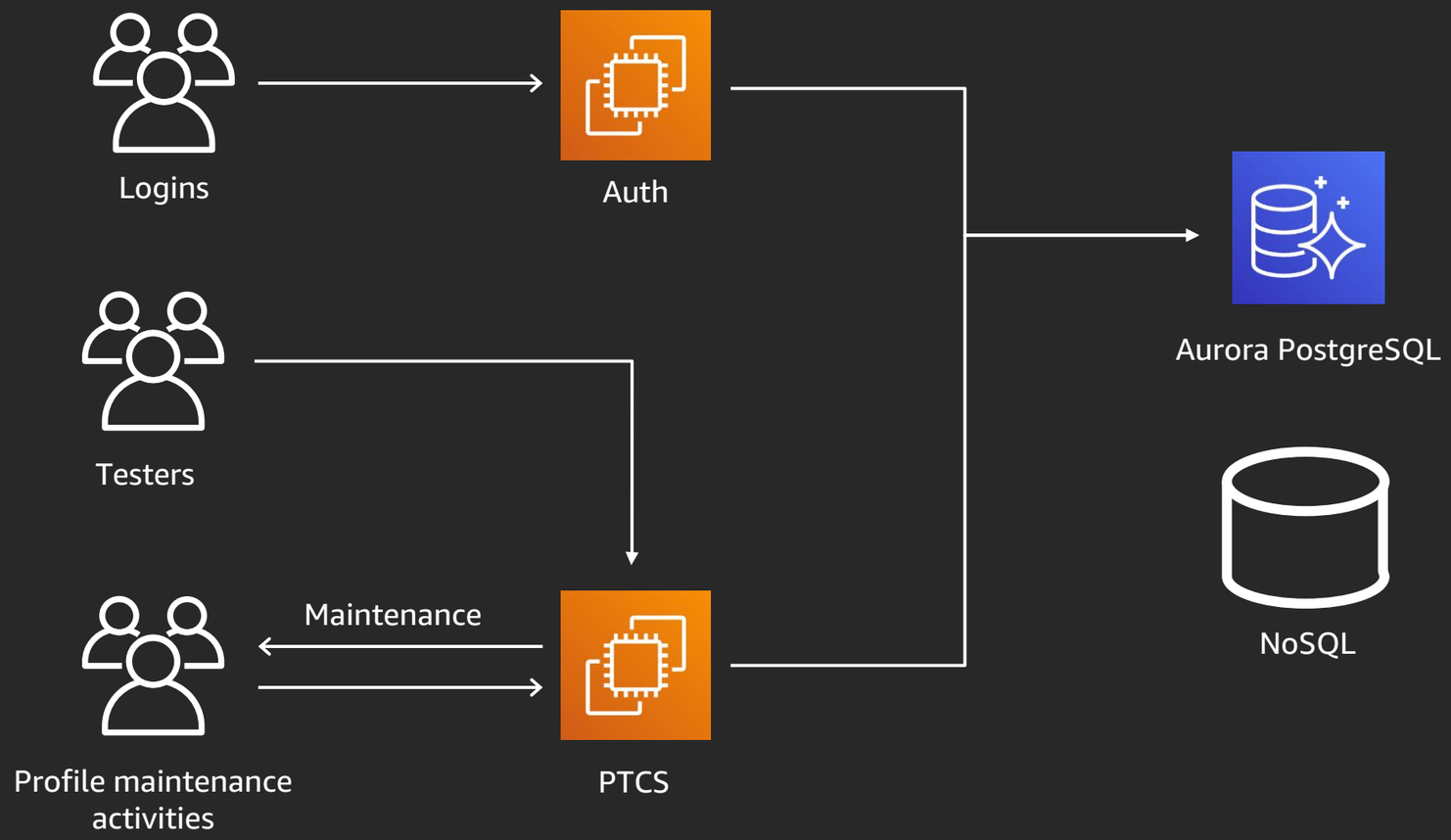
Migration Day



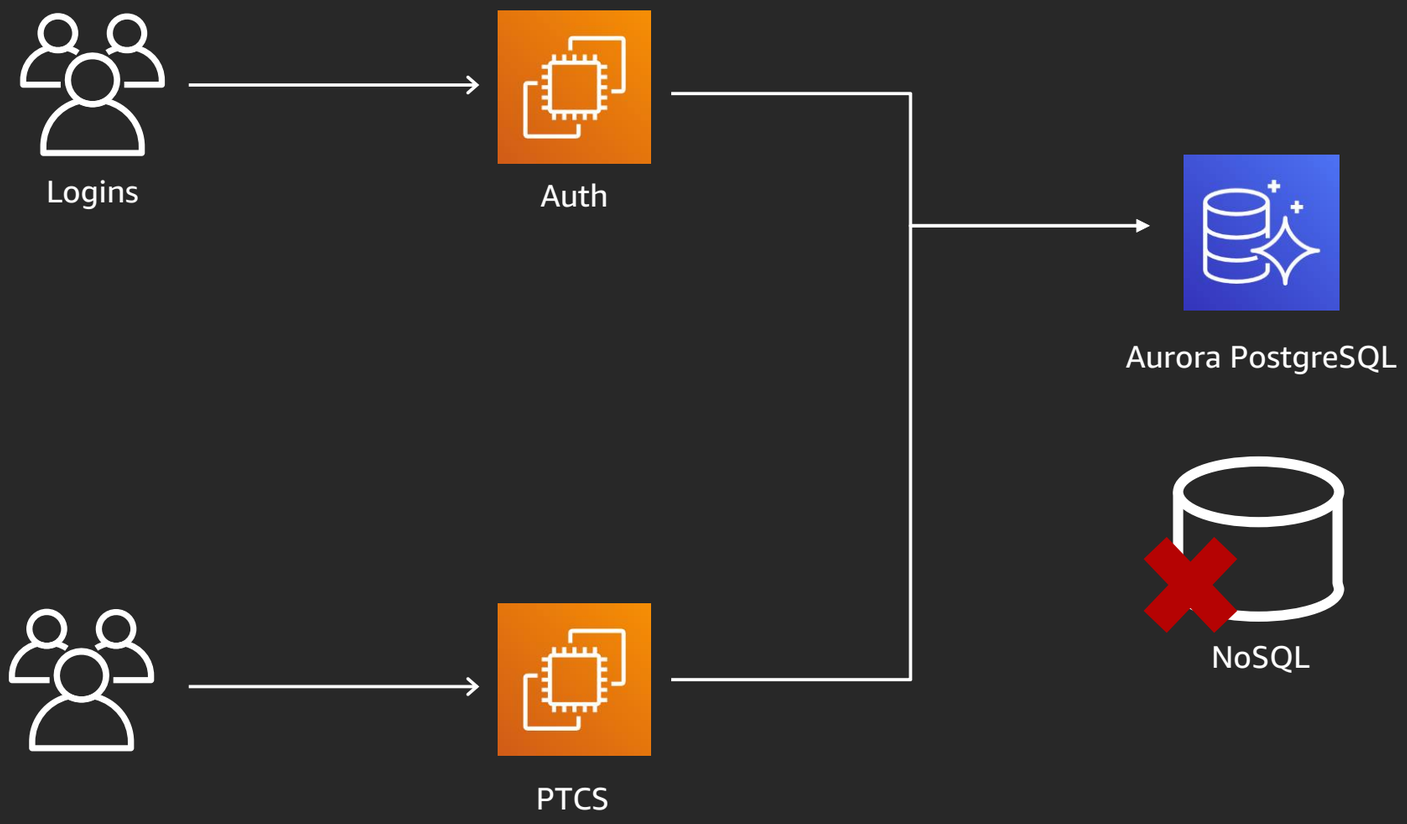
Migration Day



Migration Day



Migration Day



How did it go?

The good

- No authentication downtime
- Plan worked as expected

The bad

- Patches
- Some underperforming queries

The ugly

- Nothing

95% of users experienced no impact

Performance was good and consistent

The results

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Design goals revisited

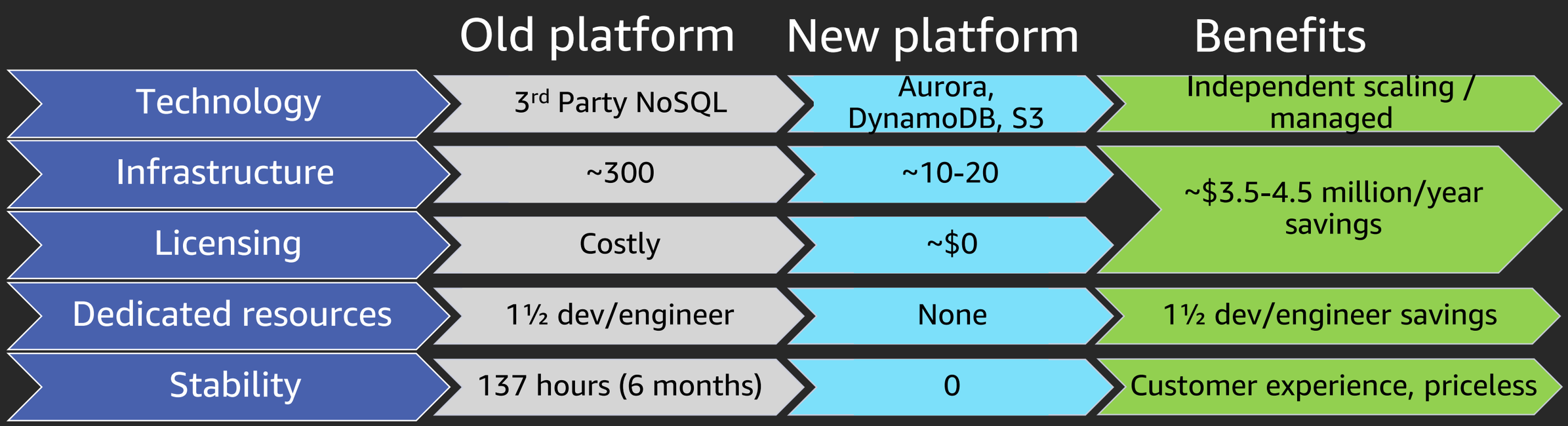
Leverage managed services ✓

Use the appropriate datastore for the data ✓

High availability, stability, and performance ✓

Reduce cost & right size infrastructure ✓

Overall value



Project retrospective

What went well

- An agile approach to the project & problems we encountered
- Leveraging the experts: AWS DB Professional Services
- Segmenting the data and how it was handled

Key learnings

- Deeper understanding of our data and services
- Prior solution was more expensive (money and people) than we realized



Tech org moving forward

Next phase

- Monitor & optimize the platform and our services
- Understand upcoming needs & changes to our data
- Scope our advanced analytics requirements

New architectural tenets

- Define and use common design patterns and components
- Simplify toolset, preferring managed services
- Documentation must exist, be well-defined
- Data standards & practices
- Use data to make decisions



amzn.to/GameTechAtreInvent

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Thank you!

AWS re:Invent

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws



Please complete the session survey in the mobile app.

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.