How Amazon.com Migrated Inventory Management Systems to Amazon Aurora with PostgreSQL compatibility

Brent Bigonger
Senior Database Engineer
Amazon Fulfillment Technologies (AFT)

aws

Agenda

Amazon Fulfillment Technologies (AFT)

Technical challenges

AFT’s Oracle to Aurora PostgreSQL migration

Amazon Fulfillment Technologies (AFT)

Amazon Fulfillment Technologies

Amazon Fulfillment Technologies

Amazon Fulfillment Technologies

Amazon Fulfillment Technologies

Technical Challenges

Complexity and Scale

About 350 Oracle source databases

Some of the oldest Amazon databases

- 20+ years of optimization for Oracle

Minimal downtime requirements

Dozens of services

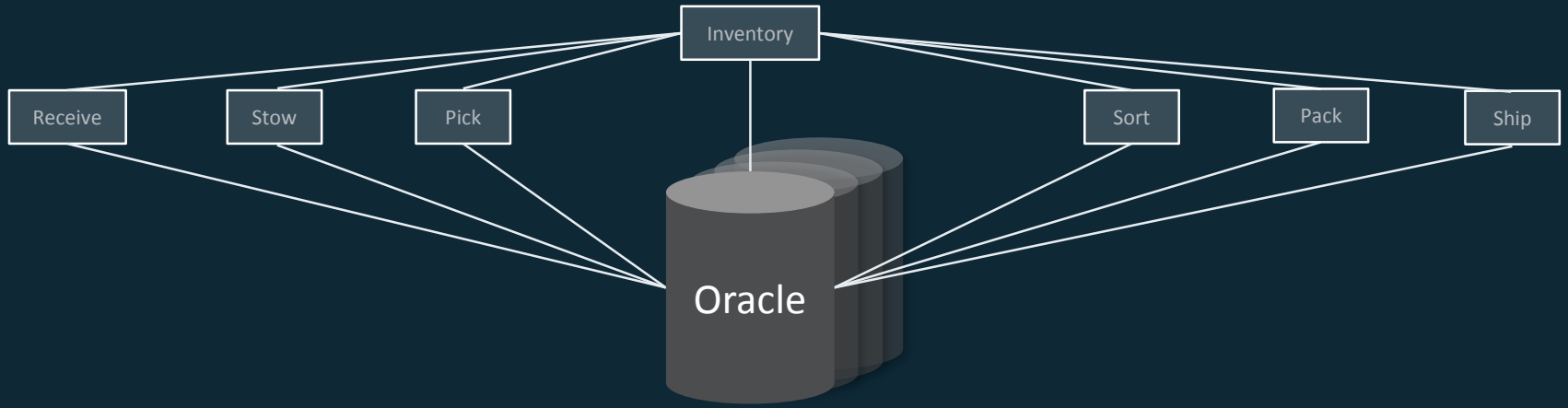
Complex and some unknown dependencies

Stringent database call latency requirements

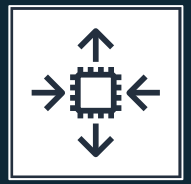
aws

Amazon Fulfillment Technologies

- Previous architecture



Challenges with Oracle and on-premise databases



Scalability

- Difficult to scale
- Required custom hardware

Availability

- Hardware failures compromised availability
- Longer time to recover

Hardware management

- Hardware forecasting, ordering and provisioning
- High operational burden to maintain

aws

Database characteristics

Relational

Referential integrity with strong consistency, transactions, and hardened scale

Complex query support via SQL

Amazon Aurora

Amazon RDS

Key-value

Low-latency, key-based queries with high throughput and fast data ingestion

Simple query methods with filters

Amazon DynamoDB

Document

Indexing and storing of documents with support for query on any property

Simple query with filters, projections and aggregates

Amazon DocumentDB

In-memory

Microsecond latency, key-based queries, specialized data structures

Simple query methods with filters

Amazon ElastiCache for Redis & Memcached

Graph

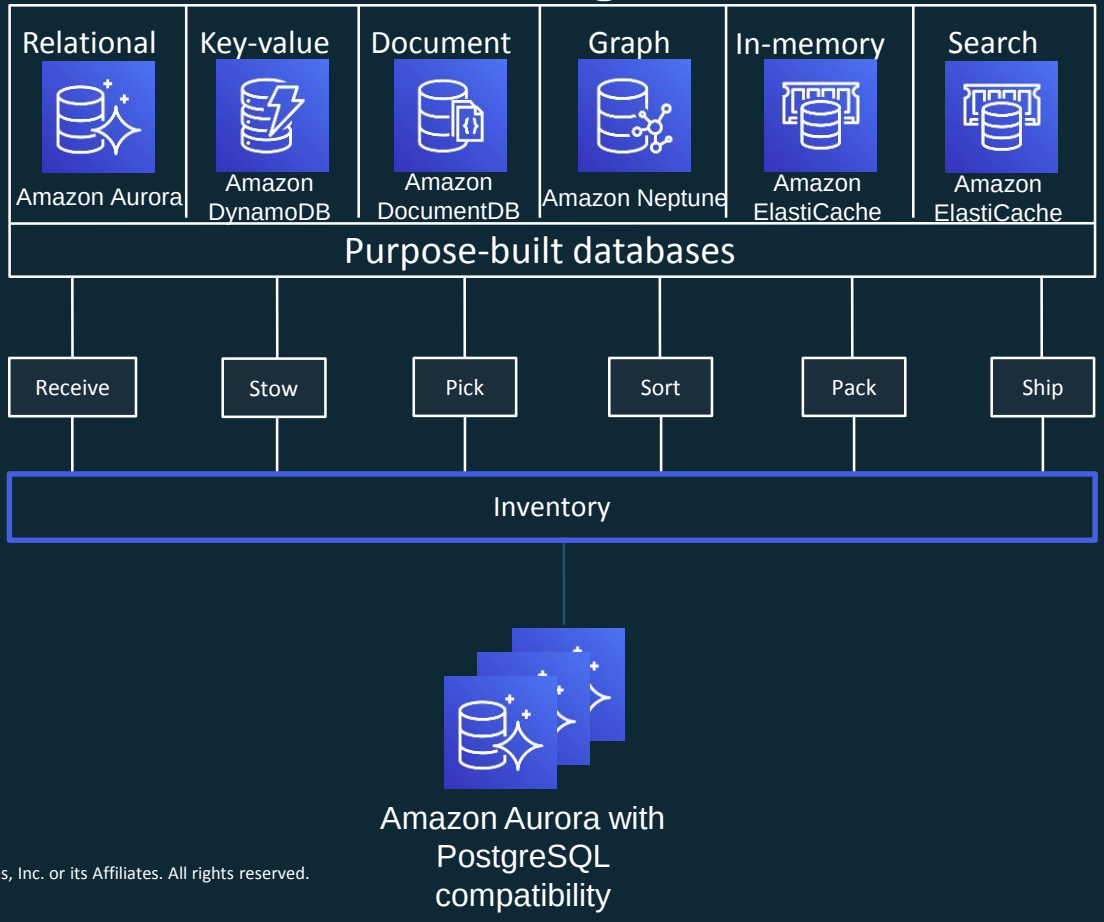
Creating and navigating relations between data easily and quickly

Easily express queries in terms of relations

Amazon Neptune

aws

Amazon Fulfillment Technologies



Why we chose Aurora PostgreSQL

Amazon Aurora with PostgreSQL compatibility

High performance

High availability

Feature/SQL parity with Oracle

Scalability

- Vertical
- Horizontal (Up to 15 Reader Instances)

Managed service

- Database snapshots (backups)
- Full production non-prod databases

aws

AFT Oracle to Aurora PostgreSQL migration

Migration

Preparation

Migration

Post-migration

Design patterns

Separate AWS prod/non-prod AWS accounts

Architect applications to leverage Readers

- Provides horizontal read scaling

Utilize database snapshot/clones

- Initial performance benchmarks, functional testing, load testing, etc.

Security

- Encryption in-transit and at-rest

Plan for automation (AWS APIs/CLIs)

Multiple non-prod cutovers/cut-backs

Oracle & PostgreSQL considerations

Timestamps/timezones differences

- Oracle - **server's** timezone
  - `SYSDATE()`
- PostgresSQL - **client's** timezone
  - `clock_timestamp()`

aws

Oracle & PostgreSQL considerations (cont.)

Partitioning differences

- No global uniqueness (PKs/UKs)
- PostgreSQL 9.6 – Inheritance Partitioning
- PostgreSQL 10 – Declarative Partitioning
- PostgreSQL 11 – Lots of exciting additions

aws

Oracle & PostgreSQL considerations (cont.)

Default PostgreSQL collation != Oracle

DMS data validation uses ‘C’ collation

COLLATE 'en_US':

```
ttest1=> SELECT * FROM bb_sort
ORDER BY val;
 id |  val
----+--------
  2 | three
  3 | t-wo
  1 | ONE
```

COLLATE 'C':

```
ttest1=> SELECT * FROM bb_sort
ORDER BY val;
 id |  val
----+--------
  1 | ONE
  3 | t-wo
  2 | three
```


aws

Dependency analysis

Capture all query and DML activities

- Shared objects – tables, views, sequences
- Cross-database Materialized Views
- Data Warehouse ETL feeds

How?

- Query historical views (DBA_HIST_%, v$active_session_history)
- Periodic sampling active v$ views (vsql, vsqlarea)
- Login monitoring (logon trigger)

aws

Migration

Preparation
Migration
Post-migration

Database launch automation

Enter database in fleet-wide metadata

Create Reserved Instance

Create database

Onboard Data Warehouse ETL feeds

Amazon Aurora with PostgreSQL compatibility

AWS Data Migration Service (DMS) kick-off

Application schema creation

Create monitoring

Create scheduled jobs

Integrate Amazon CloudWatch metrics

aws

Migration Requirements

Performance

- No operational impact on FC operations
- Short downtime
- Aurora PostgreSQL same or better performance from Oracle

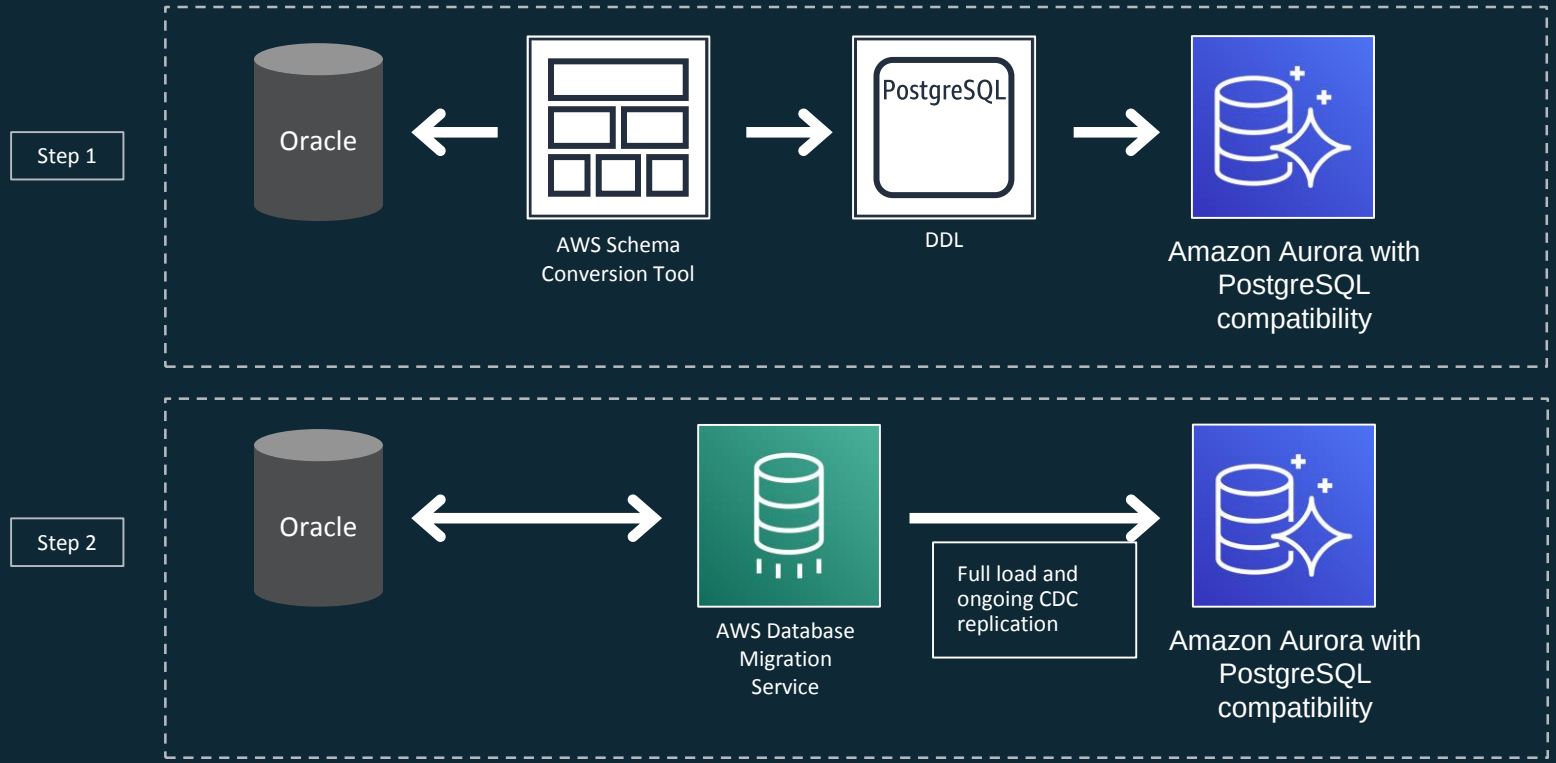
Migration management

- Full data load and ongoing replication
- Continuous data validation
- Replication monitoring and alarming

Automation

- Automated database provisioning and build
- Automated data migration
- Full cutover automation for services and database
- Support multiple concurrent cutovers

Data migration



Migration methodology

Not a “zero” downtime migration

Primarily a lift-and-shift

Start with manual migrations

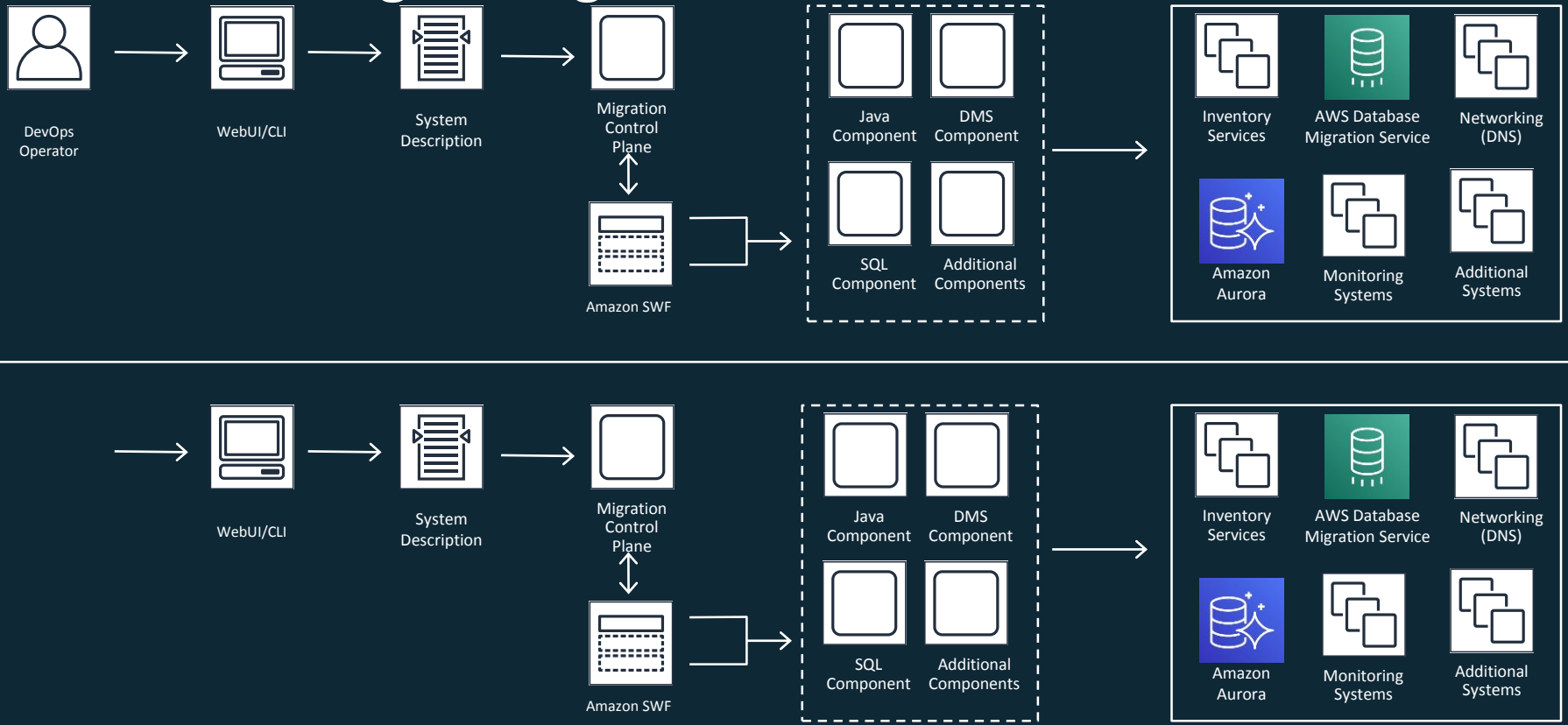
- Some pilot FCs

Some automation

Migrate largest database

Full migration automation

Workflow Engine migration automation



Migration

Preparation
Migration
Post-migration

Database Migration Service (DMS) lessons learned

All source tables should have a PK/UK

- Primary Key required for LOBs
- Primary Key or Unique Index required for data validation

Identify tables with LOBs

- Limited LOB Mode, Full LOB Mode & Inline LOB Mode

Large replication instances (R4 class)

Monitor replication instances

Consider one heartbeat table per DMS task

Enable any PostgreSQL triggers after stopping DMS

- DMS validation

aws

Migration lessons learned

Evaluate data lifecycle retention

- Ex: Date range partitioned tables

Better telemetry by instrumenting SQL with comments.

- `"SELECT /* SVC-API-FooClass.java-UUID-v1 */ ..."`

Sequences

- Oracle global sequence caching.
- PostgreSQL sequence caching occurs per connection.

Use SSL (verify-full)

Inserting NULL

- In Oracle, "" (empty string) and null are treated the same.
- In PostgreSQL, "" (empty string) is stored as an empty string.

aws

Migration lessons learned

Exception blocks (e.g. in triggers)

- `EXCEPTION` blocks consume Transaction IDs (XIDs)
  - Even when exception not fired
- `RAISE` notices or error doesn't

Idle transactions

- Prevents dead tuple and multi-transaction cleanup
- Consider '`idle_in_transaction_session_timeout`' (PostgreSQL 9.6+)
  - `FATAL:  terminating connection due to idle-in-transaction timeout`

No Global Temporary tables

- Session level frequent create/drop

aws

Migration lessons learned

Auto Vacuum - critical consideration of PostgreSQL

CloudWatch alarm for MaximumUsedTransactionIDs

- 2.1 billion un-vacuumed Transaction IDs

Monitor

- pg_stat_all_tables
  - Key columns n_live_tup, n_dead_tup, autovacuum_date
  - Absolute: n_dead_tup
  - Ratio: n_dead_tup/n_live_tup

aws

Migration lessons learned

Review *autovacuum* parameters

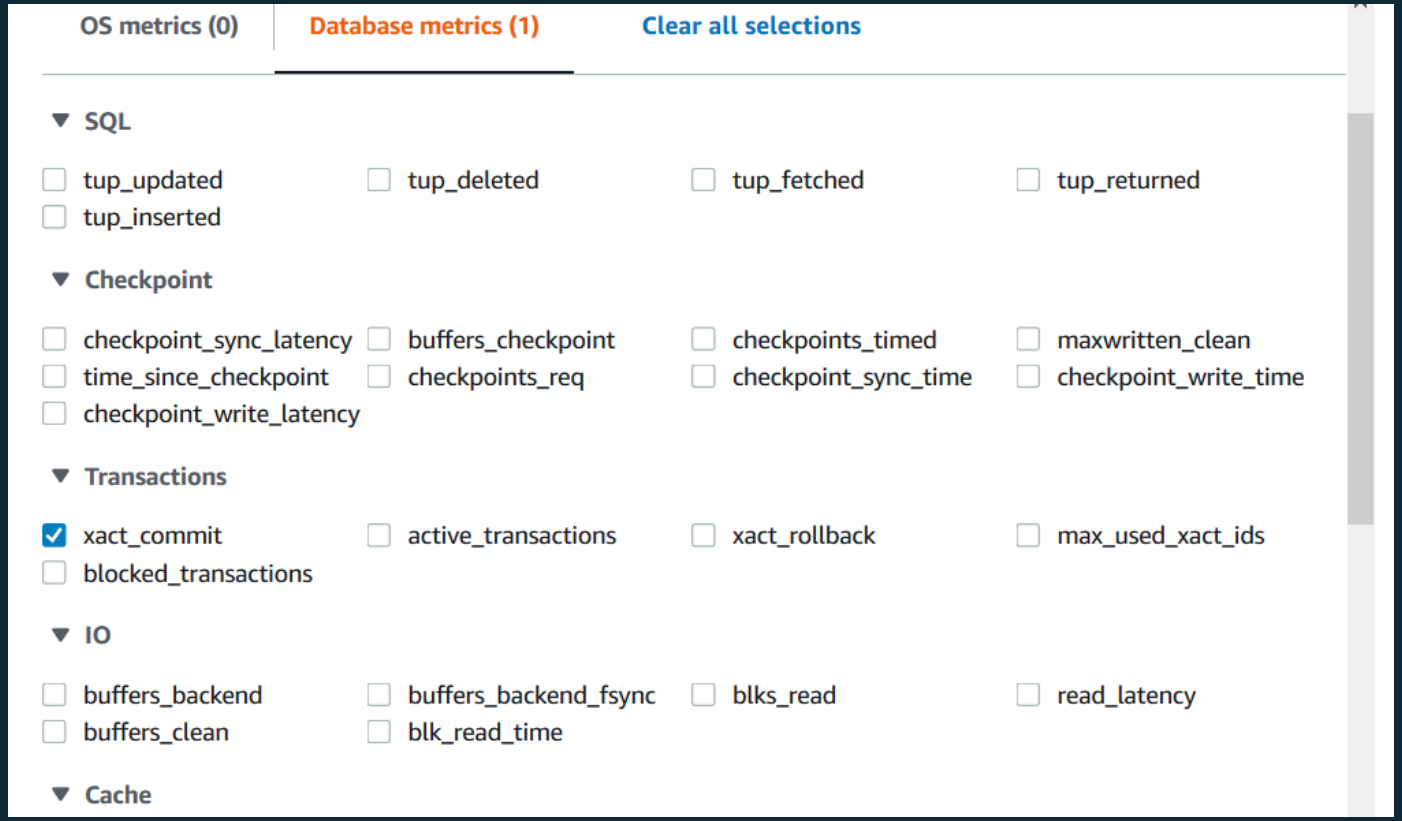
- Defaults can’t accommodate all situations
  - autovacuum_vacuum_scale_factor
  - autovacuum_vacuum_threshold
  - autovacuum_analyze_scale_factor
  - autovacuum_analyze_threshold

Type of Table

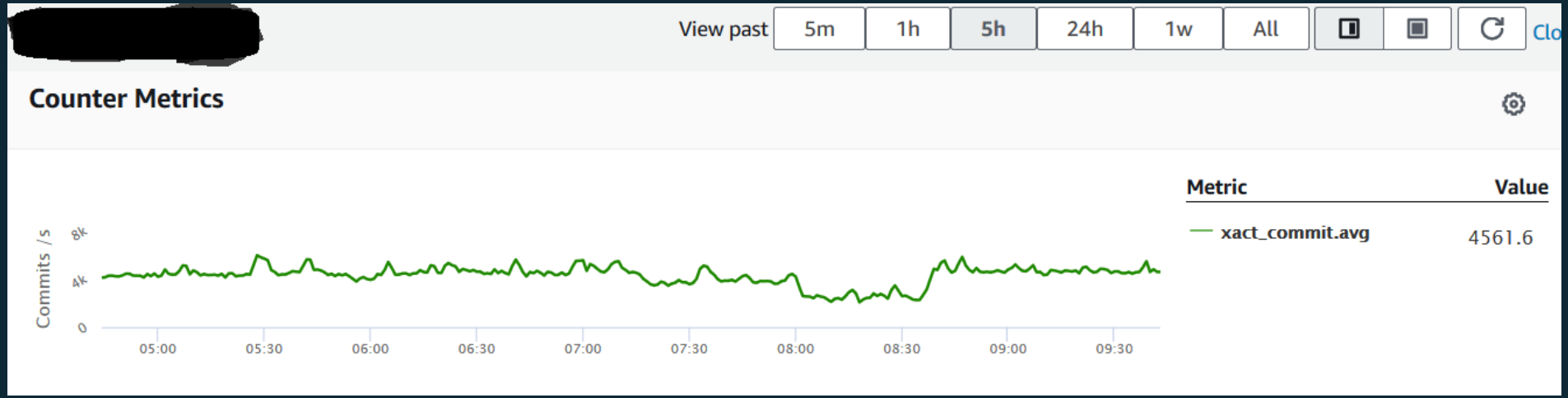
- fillfactor (table default 100)
- INSERT only
- INSERT + UPDATE/DELETE

Performance Insights

Performance Insights – Counter Metrics

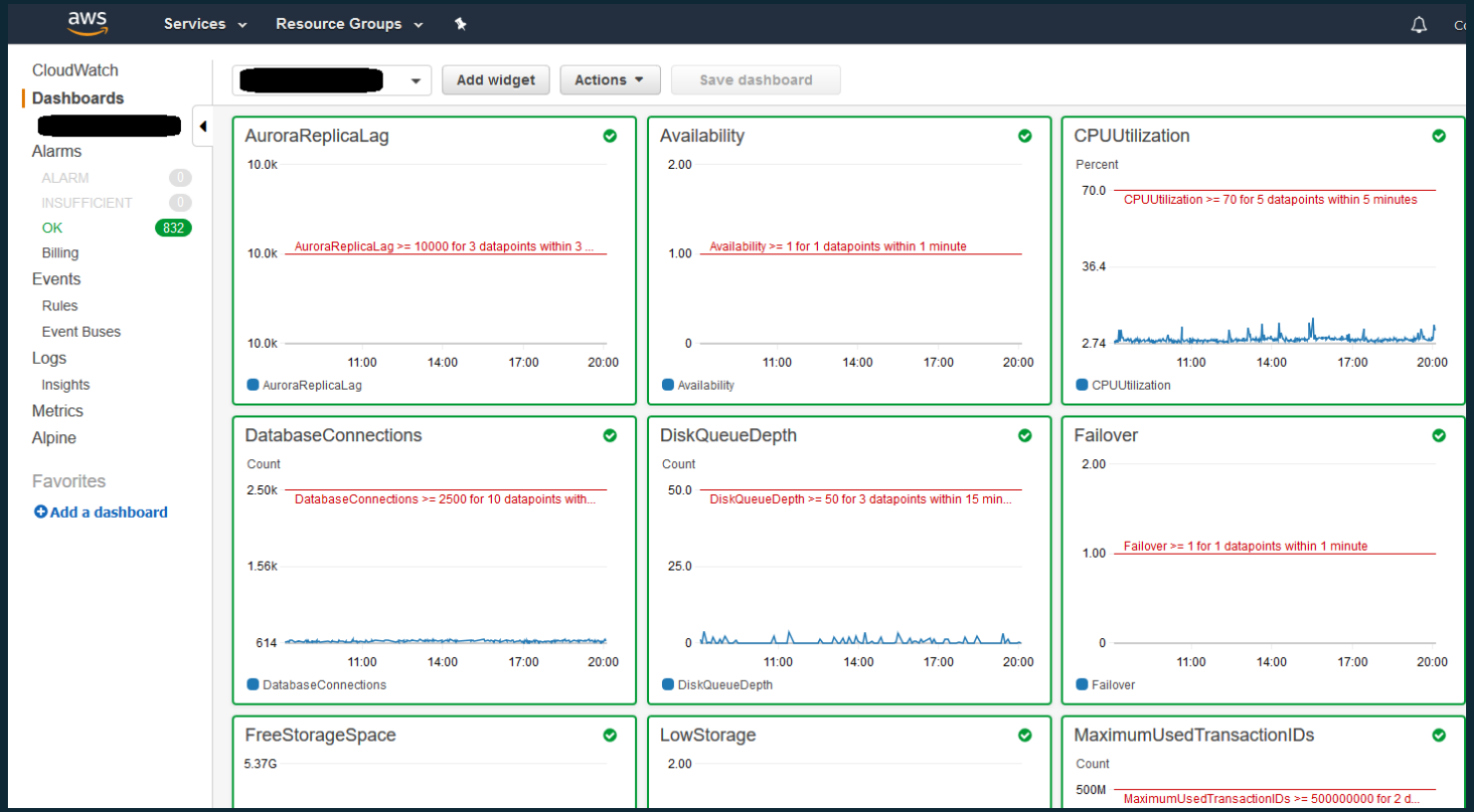


Performance Insights

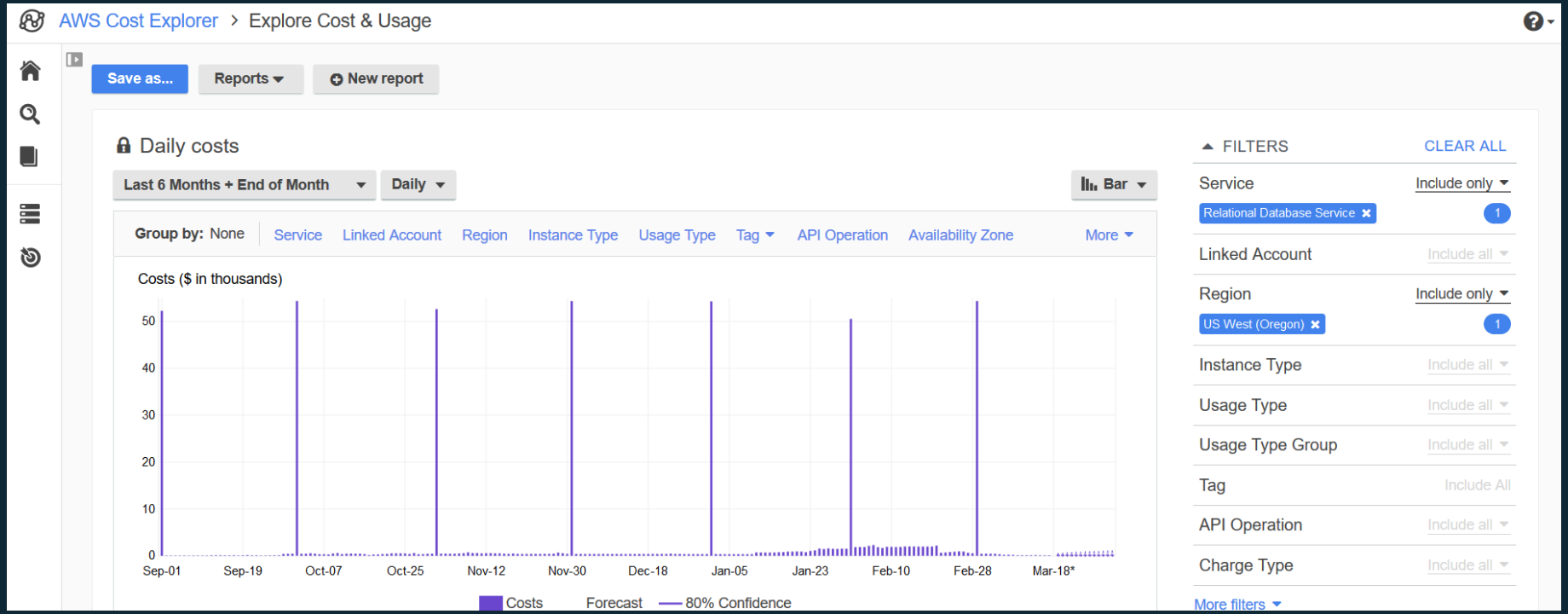


aws

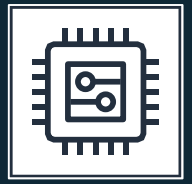
CloudWatch Dashboards and Alarms



AWS Cost Explorer

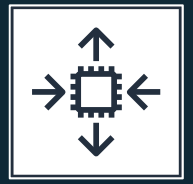


Aurora PostgreSQL Benefits for AFT



Performance

- Scales to largest AFT workloads
- Supports our strict query latency requirements
- No impact snapshot backups



Scalability

- Now scaling up/down takes minutes, not hours
- Seamless horizontal read scaling

Availability

- Faster failovers
- Back to high availability in minutes after hardware failure

Hardware management

- Provisioning our hardware in minutes, not months
- Better use of DBA team resources

Cloud-based automation

- AWS API/CLI enabled management
- CloudWatch monitoring
- Build test databases from snapshots

Cost

- No more Oracle license costs

aws

Thank you!

Brent Bigonger

Amazon.com