# Planning a PoC with Amazon Aurora

**Steve Abraham**
Principal Data Architect
Amazon Web Services

# Agenda

- Introduction to Amazon Aurora
- When to do a proof of concept (PoC)
- PoC database environment setup
- Database metrics and statistics collection
- Best practices for running the PoC
- Types of PoC testing
- PoC test design template
- Q&A

# Introduction to Amazon Aurora

# What is Amazon Aurora?

Amazon Aurora

**Speed** and **availability** of high-end commercial databases

**Simplicity** and **cost-effectiveness** of open source databases

Drop-in **compatibility** with MySQL and PostgreSQL

Simple **pay as you go** pricing

# Understanding the value of Amazon Aurora

## Features

- Aurora Global Database
- Fast database cloning
- Amazon Aurora Serverless
- Database Backtrack (Aurora MySQL)
- Parallel Query (Aurora MySQL)
- Aurora multi-master (Aurora MySQL)
- Cluster cache management (Aurora PostgreSQL)
- Query plan management (Aurora PostgreSQL)

## Differentiators

- Storage architecture
- High availability, failover process
- Disaster recovery, backup and restore
- Database endpoint management
- Logging and audit
- Security model

# When to do a Proof of Concept (PoC)

# When should you do a proof of concept (PoC)?

## When?

- Replatforming an application (and its databases)
- Modernizing an application (and its databases)
- Migrating to AWS

## Why?

- Evaluate fit for purpose
- Ensure operational requirements are met
- Gauge level of effort needed to reach the goal

# Structuring your PoC

## What are your objectives?

- Application compatibility
- Functional use cases
- Throughput and latency needs
- Operational efficiency
- Security and compliance
- Overcome scaling limitations

▶ Quantify and use as a baseline!

# Identify workload characteristics

## How are you measuring your objectives?

- Run with production-representative workload
- Right mix of OLTP versus OLAP versus HTAP
- Appropriate concurrency level
- Appropriate data set size
- Number (and churn) of client connections
- Appropriate sizing

▶ Build appropriate monitoring instrumentation

# Define and rank your success criteria

## How do you know it was successful?

- Align on your requirements and be specific
- Define set of success criteria against these requirements
- Distinguish between “need” and “nice to have”
- Monitor metrics that relate to the requirements
- Use scoring model to quantify

## From → To

- “Reporting 2× slower”
- “High availability exceeds our needs”
- “Kind of meets our commit latency”
- And so on…

| Criteria | Criticality<br>(1, low; 2, medium; 3, high) | Score<br>(0, fail; 1, sometimes; 2, met; 3, exceeded) |
|---|---|---|
| Reporting perf. | 2 | 0 |
| High availability | 3 | 3 |
| Commit latency | 3 | 1 |

Pre-testing

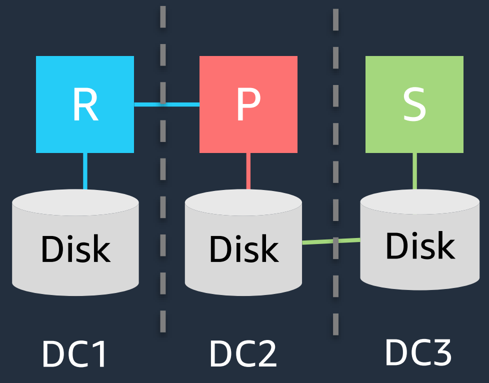
Post-testing

# PoC database environment setup

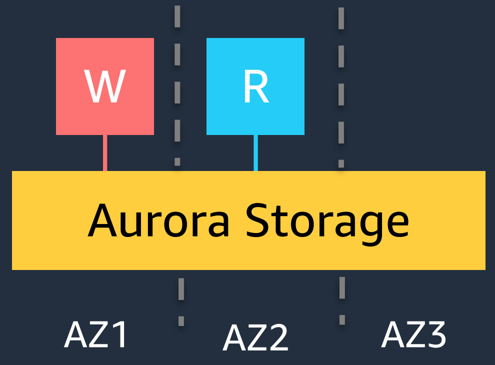
# PoC database environment setup

- Setup DB environment with equivalent topologies for high availability, durability, and H/W configurations
- Use read replica instances to distribute read and write workload in the Aurora cluster, if your application supports read and write splits
- Keep your PoC infrastructure separate from your current database and application infrastructure
- Try to automate the deployment environment using AWS CloudFormation or Terraform

Sample Current Environment



Sample Aurora PoC Environment



# Suggested database metrics and statistics for collection

# Apart from AWS CloudWatch metrics, you can configure additional database and OS statistics collection

- Enable Enhanced Monitoring, with monitoring frequency set to at least a 5 second interval
- Enable Amazon RDS Performance Insights
- Enable the pg_stat_statements and auto_explain extensions (Aurora PostgreSQL)
- Enable Query Plan management (Aurora PostgreSQL)
- Enable slow log query and performance_schema on the database (Aurora MySQL)

# Additional database metrics and statistics collection for Aurora MySQL

```
Select * from information_schema.processlist WHERE command <> "sleep"\G

Show Engine Innodb Status\G

Show global status;

Select * from information_schema.innodb_trx\G

Select r.trx_id waiting_trx_id, r.trx_mysql_thread_id waiting_thread,
r.trx_query waiting_query, b.trx_id blocking_trx_id, b.trx_mysql_thread_id
blocking_thread, b.trx_query blocking_query FROM
information_schema.innodb_lock_waits w INNER JOIN
information_schema.innodb_trx b ON b.trx_id = w.blocking_trx_id INNER JOIN
information_schema.innodb_trx r ON r.trx_id = w.requesting_trx_id\G
```

# Additional database metrics and statistics collection for Aurora PostgreSQL

List of pg_catalog objects

- pg_stat_statements
- pg_stat_activity
- pg_stat_*
- pg_stat_io
- pg_locks

# Best practices for running the PoC

# Best practices for running the PoC

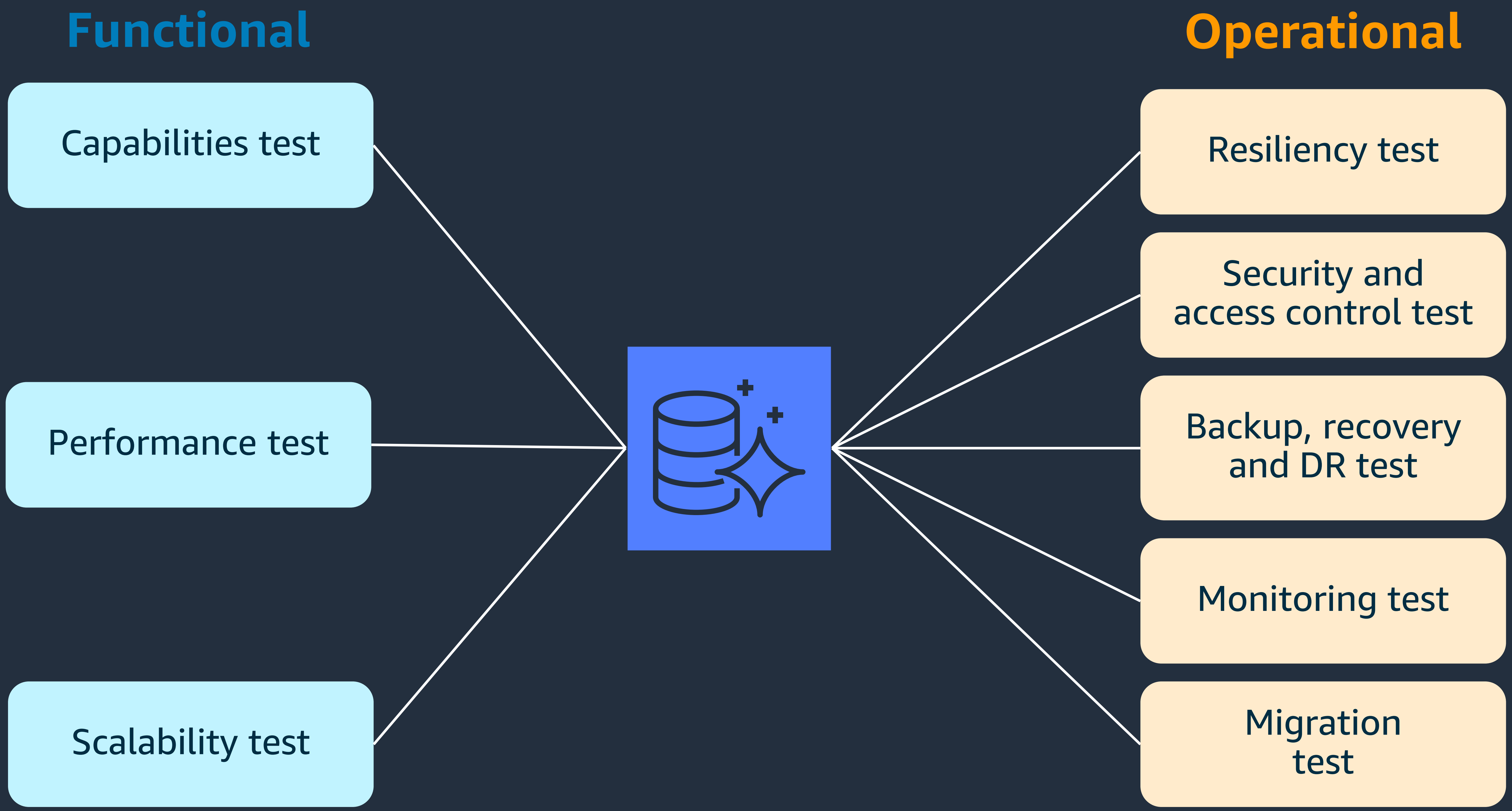
- Create custom parameter groups, only customize parameter values when needed
- Enable the additional logging related parameters as needed for example: log_lock_waits=1 (Aurora PostgreSQL)
- Run test using a warm database or application caches
- Simulate the production architecture while running tests using multiple app instances, primary and read replica database instances, as needed
- Run the tests for a period of the time that is representative of your production workload pattern

# Best practices for running the PoC (cont’d)

- Run multiple test run to make sure the results are consistent
- Run tests for Aurora and current database engine environments with the same data set, application load and application version. If possible, use application payload size and concurrency similar to production to simulate a mirror workload
- Pay attention to table bloat and autovaccum processes based on the workload pattern (Aurora PostgreSQL)

# Types of PoC testing

# Types of PoC testing



Functional

- Capabilities test
- Performance test
- Scalability test

Operational

- Resiliency test
- Security and access control test
- Backup, recovery and DR test
- Monitoring test
- Migration test

# Functional – Capabilities test

Test entire application functionality

Importance

- The most critical part of PoC testing
- Core main application functionality and capabilities should not diminish
- Identify opportunity to application improvements, if applicable

Best practices

- Modular testing in case of large or complex applications
- Full application testing including all use case scenarios and time based processes (day/week/monthly/quarter end)

# Functional – Performance test

Validate the application performance SLAs with normal and peak application load.

## Importance

- Validate the database performance against all application load profiles
- Ensure consistent performance within accepted SLAs
- Application load profiling

## Best practices

- Use production equivalent H/W infrastructure including SSL, load balancer and firewalls
- Run PoC with production equivalent load, application payload and concurrency
- Consider Aurora Cluster Cache Management(Aurora PostgreSQL), Parallel Query (Aurora MySQL) based on application use case

# Functional – Scalability test

Test and define database scalability strategy.

Importance

- Measure application performance by scaling Aurora DB instances to meet changing application workloads

Best practices

- Use production environment architecture to test scalability
- Use auto-scaling configuration for Aurora read replicas to manage sudden increases in connection or workload, if applicable
- Define threshold limits and performance monitoring to trigger scaling event
- Discover the DB instance type workload throughput

# Operational – Resiliency test

Test the application resiliency with different kind of database failures.

Importance

- Test the application behavior during database failures
- Determine how failures will remediate and possible impact on the application
- Identify the process for auto remediation
- Identify and define alert notifications

Best practices

- Use Amazon Aurora fault injection queries
- Conduct a Aurora writer DB instance failure test by manual failover
- Review client side DNS endpoint caching configuration
- Application based use case consider proxy solution (RDS Proxy)

# Operational – Security & access control test

Test the application security and access control requirements.

Importance

- Validate all the security compliance requirement like data encryption in transit and/or at rest, as well as access control

Best practices

- Grant least privilege access control
- Validate and configure separate security policies between production and non-production environments
- Access control via IAM to manage AWS database infrastructure access and change management
- Password rotation and database object access management tests
- Configure Auditing (DAS) to meet security requirements, if needed

# Operational – Backup, recovery & DR test

Test the database backup and recovery including DR test procedures, expected application downtime and notification alerts.

## Importance

- Understand the database backup and recovery procedure.
- Configure the backup retention period and set the SLAs accordingly
- Define and test the database DR procedure based on business requirement

## Best practices

- Set up the automated backup retention period according to SLAs for PiTR
- Define and test logical database/table level backup and restore procedure based on application requirements
- Define database DR design according to your RTO and RPO SLAs (i.e., use Aurora Global Database, snapshot copy etc.)
- Full dry run of DR database failure and failback to primary AWS region test

# Operational – Monitoring test

Validate critical database monitoring required and modify monitoring based on the Aurora database, if needed.

Importance

- Understand the Aurora cluster features, CloudWatch metrics, and RDS Event notification monitoring to manage database more effectively
- Database availability monitoring test and notification

Best practices

- Set up CloudWatch alarms for key performance indicators
- Run application with full functionality and validate the monitoring threshold limits and alert notifications
- Configure RDS Performance Insights and Enhanced Monitoring for additional Aurora database and OS related key metrics
- Run manual database failure process to review monitoring alert notifications
- Define and validate key alert notifications for Aurora cost components like backup, I/O and storage usages

# Operational – Migration test

Test the entire database migration procedure.

Importance

- Familiarize yourself with the full migration procedure and estimated the migration time
- Create the run book for migration procedure

Best practices

- Use the production database copy to test migration process on production type database architecture to avoid any unexpected errors
- Identify and validate the database consistency and integrity during migration testing
- Repeat documented procedures to ensure consistent results
- Create and test rollback database migration plan, if needed

# Sample PoC design doc

# Sample PoC design doc template

| Test Type | Test Name | Test Description | Current Measurements | PoC Success Criteria | Test Results | Test Start Time | Test End Time | Criticality Level (1-> Low, 2->Medium, 3->High | Overall Score (0->Fail, 1->Sometimes, 2->Met, 3->Exceeded |
|---|---|---|---|---|---|---|---|---|---|
| Functional | Capabilities | | | | | | | | |
| Functional | Performance | | | | | | | | |
| Functional | Scalability | | | | | | | | |
| Operational | Resiliency | | | | | | | | |
| Operational | Security & Access control | | | | | | | | |
| Operational | Backup, Recovery & DR | | | | | | | | |
| Operational | Monitoring | | | | | | | | |
| Operational | DB Migration | | | | | | | | |

# Sample PoC design doc

| Test Type | Test Name | Test Description | Current Measurements | PoC Success Criteria | Test Results | Test Start Time | Test End Time | Criticality Level (1-> Low, 2->Medium, 3->High | Overall Score (0->Fail, 1->Sometimes, 2->Met, 3->Exceeded |
|---|---|---|---|---|---|---|---|---|---|
| Functional | Capabilities | All modules<br>ETL processing<br>Other app integration | Pass<br>Pass<br>Pass | Pass<br>Pass<br>Pass | | | | 3<br>1<br>2 | |
| Functional | Performance | Production load | App Users: 1000<br>QPS:3000<br>Read Latency: 4ms<br>Write Latency:6ms | App Users: 1000<br>QPS:3000<br>Read Latency: 4ms<br>Write Latency:6ms | | | | 3<br>3 | |
| Functional | Scalability | prod load<br>150% prod load | 1W+2RR<br>1W+3RR | 1W+2RR<br>1W+3RR | | | | 3<br>2 | |
| Operational | Resiliency | DB failover | 5 minutes | 1 minute | | | | 3 | |
| Operational | Security & Access control | IAM roles<br>PCI requirements | Current setup<br>Current setup | Pass<br>Pass | | | | 3<br>3 | |

# Sample PoC design doc (cont’d)

| Test Type | Test Name | Test Description | Current Measurements | PoC Success Criteria | Test Results | Test Start Time | Test End Time | Criticality Level (1-> Low, 2->Medium, 3->High | Overall Score (0->Fail, 1->Sometimes, 2->Met, 3->Exceeded |
|---|---|---|---|---|---|---|---|---|---|
| Operational | Backup, Recovery & DR | Daily backup-14 days retention<br>DR failure | Pass<br>20 minutes | Pass<br>10 minutes | | | | 3<br>2 | |
| Operational | Monitoring | DB, critical processes & Infrastructure monitoring | Pass | Pass | | | | 2 | |
| Operational | DB Migration | 2 successful tests<br>Migration downtime 10 minutes | Pass<br>10 minutes | Pass<br>10 minutes | | | | 3<br>2 | |

# Sample PoC design doc

| Test Type | Test Name | Test Description | Current Measurements | PoC Success Criteria | Test Results | Test Start Time | Test End Time | Criticality Level (1->Low, 2->Medium, 3->High | Overall Score (0->Fail, 1->Sometimes, 2->Met, 3->Exceeded |
|---|---|---|---|---|---|---|---|---|---|
| **Functional** | Capabilities | All modules<br>ETL processing<br>Other app integration | Pass<br>Pass<br>Pass | Pass<br>Pass<br>Pass | Pass<br>Fail<br>Pass | | | 3<br>1<br>2 | 3<br>0<br>2 |
| **Functional** | Performance | Production load | App Users: 1000<br>QPS:3000<br>Read Latency: 4ms<br>Write Latency:6ms | App Users: 1000<br>QPS:3000<br>Read Latency: 4ms<br>Write Latency:6ms | App Users: 1000<br>QPS:4000<br>Read Latency: 3ms<br>Write Latency:3ms | | | 3<br>3 | 3<br>3 |
| **Functional** | Scalability | prod load<br>150% prod load | 1W+2RR<br>1W+3RR | 1W+2RR<br>1W+3RR | 1W+1RR<br>1W+2RR | | | 3<br>2 | 3<br>3 |
| **Operational** | Resiliency | DB failover | 5 minutes | 1 minute | 30 seconds | | | 3 | 3 |
| **Operational** | Security & Access control | IAM roles<br>PCI requirements | Current setup<br>Current setup | Pass<br>Pass | Pass<br>Pass | | | 3<br>3 | 2<br>2 |

# Sample PoC design doc (cont'd)

| Test Type | Test Name | Test Description | Current Measurements | PoC Success Criteria | Test Results | Test Start Time | Test End Time | Criticality Level (1-> Low, 2->Medium, 3->High | Overall Score (0->Fail, 1->Sometimes, 2->Met, 3->Exceeded |
|---|---|---|---|---|---|---|---|---|---|
| Operational | Backup, Recovery & DR | Daily backup- 14 days retention | Pass | Pass | Pass | | | 3 | 2 |
| | | DR failure | 20 minutes | 10 minutes | 2 minutes | | | 2 | 3 |
| Operational | Monitoring | DB, critical processes & Infrastructure monitoring | Pass | Pass | Pass | | | 2 | 2 |
| Operational | DB Migration | 2 successful tests | Pass | Pass | Pass | | | 3 | 2 |
| | | Migration downtime 10 minutes | 10 minutes | 10 minutes | 5 minutes | | | 2 | 3 |

# Q&A

Steve Abraham
abrsteve@amazon.com

# Thank you!

# Enrollment Broker Application Migration

## Pilot - Oracle to Aurora PostgreSQL Migration

Team Virtusa

2/8/2021

# About Virtusa

Virtusa is a Premier Consulting Partner since 2017. Partnership programs include:

- AWS DB Freedom Partner
- AWS ISV Workload Migration (Pega)
- AWS Well Architected Program
- AWS Public Sector Partner
- AWS Public Sector Solution Provider
- AWS Solution Provider Program
- APN Immersion Days

Current list of competencies:

- AWS Migration
- AWS DevOps
- AWS Life Sciences
- AWS Financial Services
- AWS Data & Analytics
- AWS SaaS
- AWS Digital Workplace

# Enrollment Broker application needed to be scalable and flexible, and reduce overall efforts and costs

- Our client primarily supports Government-sponsored programs and enrolls people in programs as per their eligibility
- The current Enrollment Broker applications are running on-premise using a legacy technology stack
    - Java/iBatis for real-time processing
    - Kettle for batch processing
    - The database is on Oracle
- Any increase in daily enrollments slowed down batch processing, requiring after-office hours support to monitor and ensure no batch failures
- The customer was interested in modernizing the application on the cloud to reduce operating costs and increase system performance
- The client partnered with Virtusa to do the first set of migration and validate the technical approach

# Migrating the Enrollment Broker application to AWS

- For the pilot, migrated one instance of the Enrollment Broker application to AWS infrastructure
  - Re-architected the database from on-premise Oracle to Aurora PostgreSQL
  - Re-hosted application to Amazon EC2s
- Built a regex-based utility to reduce manual efforts in remediating Postgres compatibility issues for ~650 iBatis mapper and Kettle ETL scripts
- Defined the deployment process for performing repetitive migrations in higher environments
- Built a framework for schema reconciliation and data validations
- Supported performance testing to compare application performance metrics in Oracle and new Aurora PostgreSQL

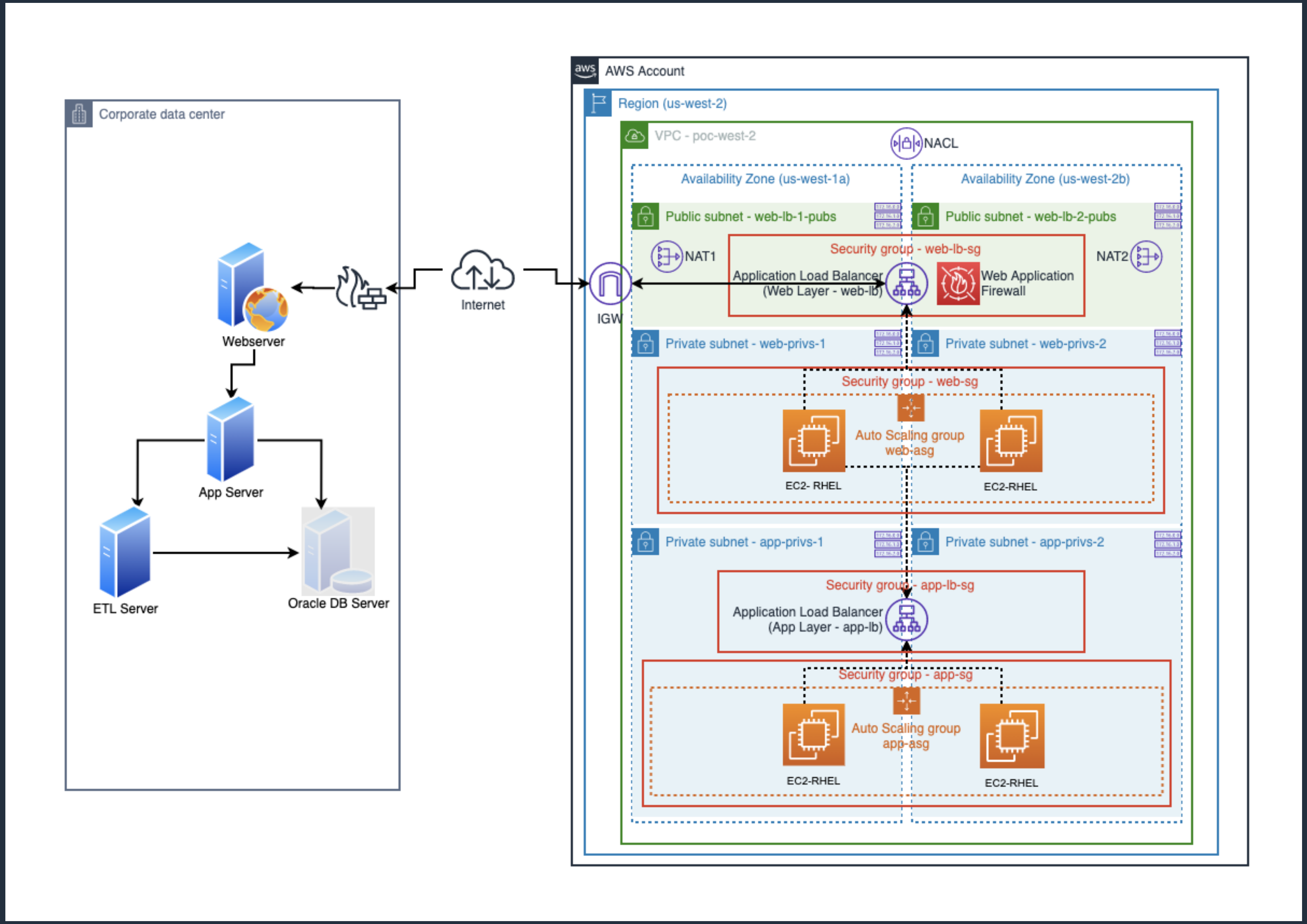
# Application migration

1. Application migration required development primarily for:

- Infrastructure provisioning
- EC2s with autoscaling groups created to run web, application, and ETL servers

2. Application Configuration

- Java application with iBatis ORM layer and Kettle ETL jobs remediated to run against Aurora PostgreSQL database
- Virtusa built a regex-based utility to reduce manual efforts in remediating iBatis and ETL code, which can be leveraged for all further migrations as part of DC exit



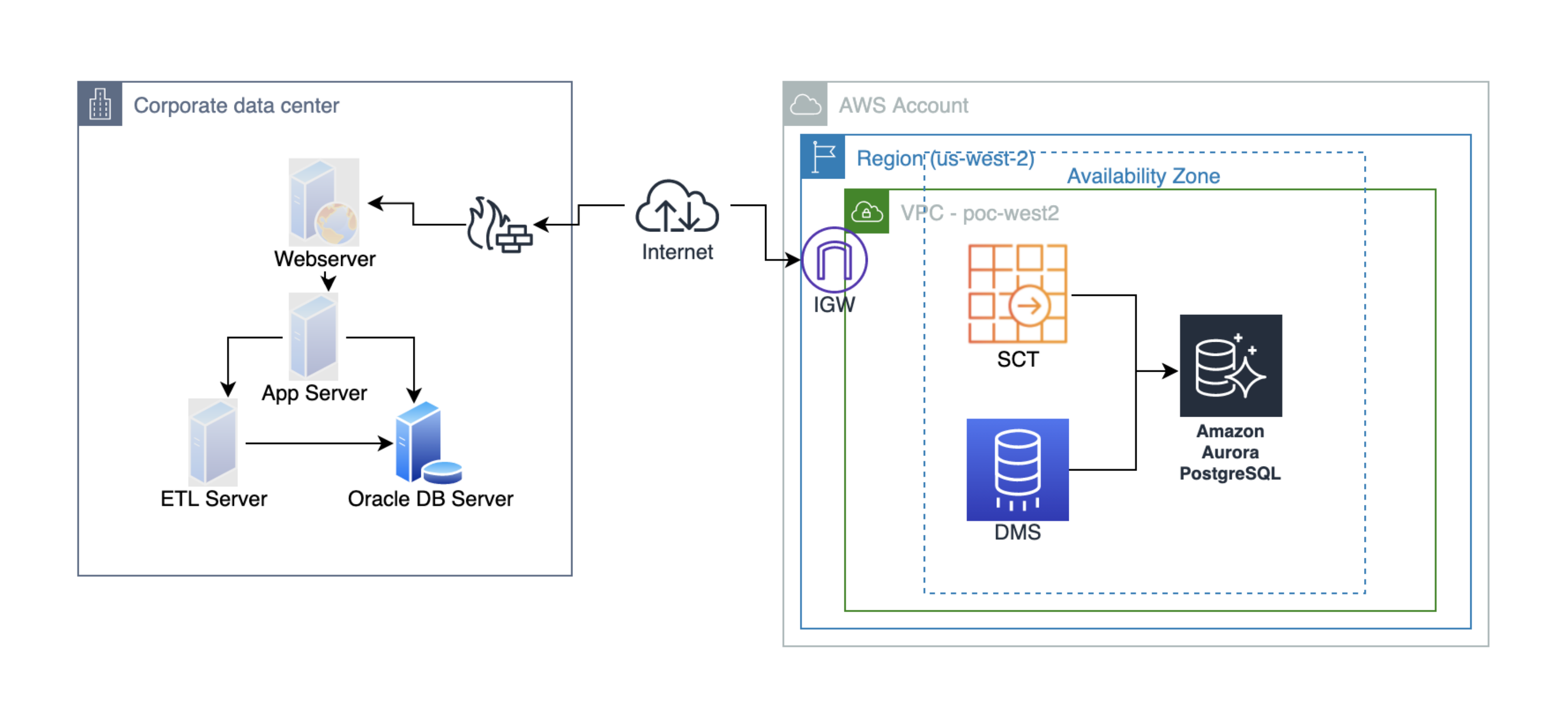
Corporate data center
Webserver
App Server
ETL Server
Oracle DB Server
Internet
IGW
AWS Account
Region (us-west-2)
VPC - poc-west-2
NACL
Availability Zone (us-west-1a)
Availability Zone (us-west-2b)
Public subnet - web-lb-1-pubs
Public subnet - web-lb-2-pubs
NAT1
NAT2
Security group - web-lb-sg
Application Load Balancer
(Web Layer - web-lb)
Web Application
Firewall
Private subnet - web-privs-1
Private subnet - web-privs-2
Security group - web-sg
Auto Scaling group
web-asg
EC2- RHEL
EC2-RHEL
Private subnet - app-privs-1
Private subnet - app-privs-2
Security group - app-lb-sg
Application Load Balancer
(App Layer - app-lb)
Security group - app-sg
Auto Scaling group
app-asg
EC2-RHEL
EC2-RHEL

# Database migration

AWS SCT and DMS are the primary migration tools used for database migration.

We formed three teams for executing the three phases of the DB migration:

1. Schema conversion
   - SCT converted 80% of the code objects automatically
   - The remaining 20% of the code having complex conversion issues was remediated manually
2. Database migration
   - DMS used to migrate the data from Oracle to Aurora PostgreSQL
3. Schema reconciliation and data validation post-migration



# Demonstrated a successful POC to efficiently modernize the Enrollment Broker application

- Batch processing performance improved greatly, and the run time is consistently inline with the support team’s estimates for any given load volume
- It is estimated that the newly built remediation utility helped reduce development efforts by 70%, and will be reused for all future migrations
- Virtusa’s approach led to the client working on a scalable factory model for migrating rest of the Enrollment Broker applications

# Visit Virtusa Web portal for more details on the case study

- A detailed case study has been shared in Virtusa's web portal. Link to the case study will be shared in the audience chat.
- Learn more about Virtusa and our solutions from the following:
    - https://www.virtusa.com/solutions
    - https://www.virtusa.com/partners/aws

# Thank you!