AWS re:Invent

DAT342

Lessons from migrating Oracle databases to Amazon Aurora

Abhinav Singh
Database Engineer – DMS
Amazon Web Services

Bijoy Nair
Software Development Manager – DMS
Amazon Web Services

AWS re:Invent

aws

Agenda

Introduction to AWS Database Migration Service (AWS DMS) and AWS Schema Conversion Tool

AWS DMS and AWS SCT product highlights

Amazon use case – Migrating Oracle databases to Amazon Aurora (PostgreSQL/MySQL)

Q&A

What to expect from this session

- AWS resources with regards to migration & replication
- How Amazon approached move-off Oracle initiative
- Top migration challenges faced by Amazon.com

What not to expect

- Schema conversion issues from Oracle
- How stuff works in PostgreSQL/MySQL
- Oracle vs. PostgreSQL vs. MySQL feature comparison

Related breakouts

DAT214 - Automation framework to migrate relational databases

DAT343 - Lessons from migrating SQL Server databases to Amazon Aurora

DAT345 - Assessing and categorizing your database migrations

DAT360 - Analytical use cases with AWS DMS

DAT362-R1- Dive deep into AWS SCT and AWS DMS (Jobvite use case)

DAT363 - Migrating your data warehouses to Amazon Redshift

DAT366 - Running Oracle on Amazon RDS and migrating to Aurora PostgreSQL

DAT367 - Running SQL Server on Amazon RDS and migrating to Aurora PostgreSQL

DAT377 - Getting started with AWS DMS and AWS SCT

Introduction to AWS DMS and AWS SCT

AWS re:Invent

aws

AWS migration tooling

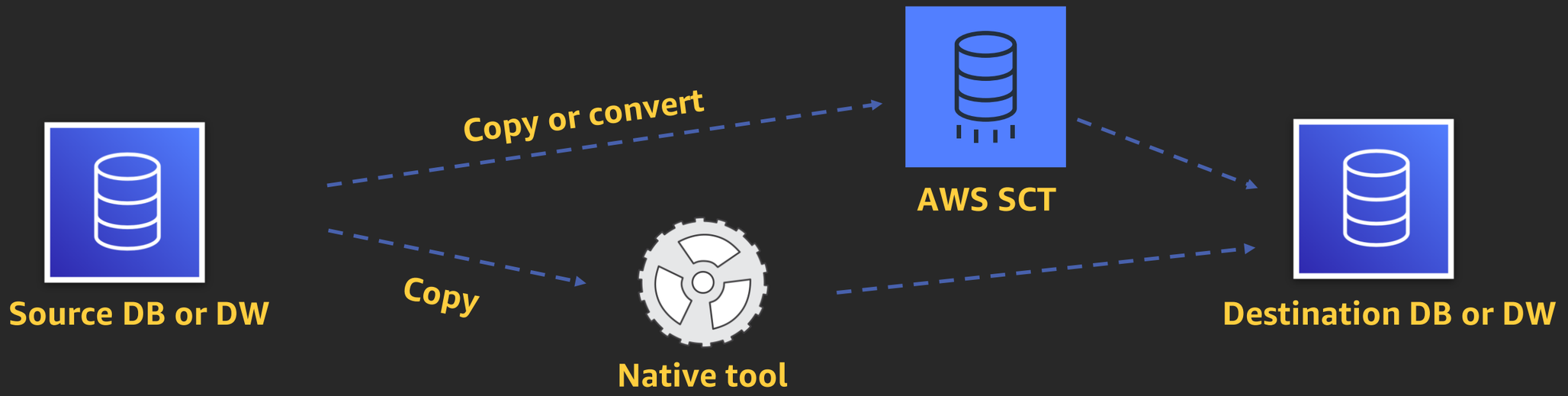
Our goal: Enable customers the freedom to choose the best data platform for their needs #DBFreedom

AWS Schema Conversion Tool (AWS SCT) converts your commercial database and data warehouse schemas to open-source engines or AWS native services, such as Amazon Aurora and Amazon Redshift

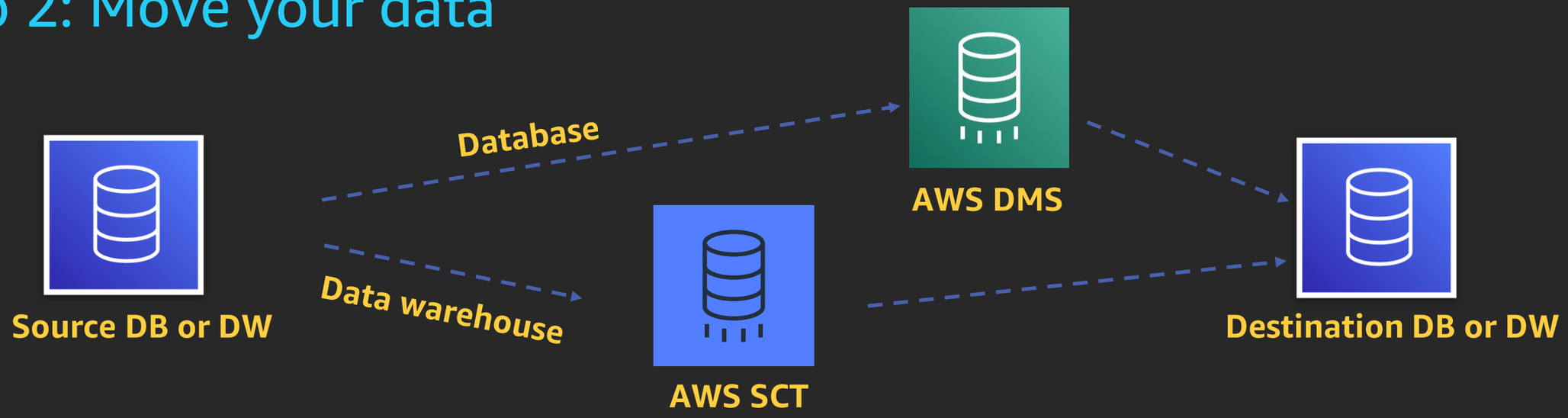
AWS Database Migration Service (AWS DMS) easily and securely migrates and/or replicate your databases *and* data warehouses to AWS

Database migration process

Step 1: Convert or copy your schema



Step 2: Move your data



Migration and replication resources

AWS Schema Conversion Tool

The AWS Schema Conversion Tool helps automate many database schema and code conversion tasks when migrating from source to target database engines

Features

Create assessment reports for homogeneous/heterogeneous migrations

Convert database schema

Convert data warehouse schema

Convert embedded application code

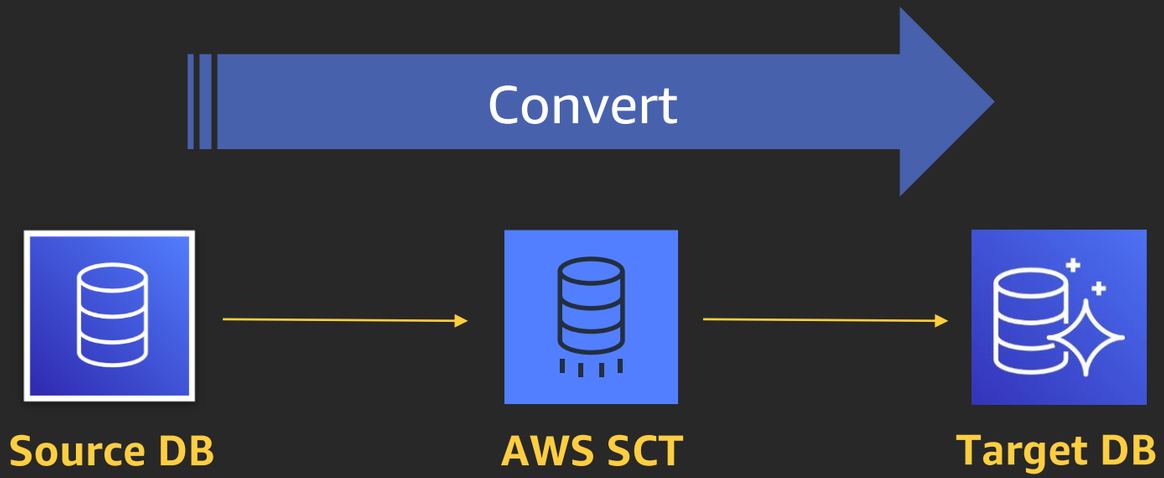
Code browser that highlights places where manual edits are required

Secure connections to your databases with SSL

Service substitutions/ETL modernization to AWS Glue

Migrate data to data warehouses using AWS SCT data extractors

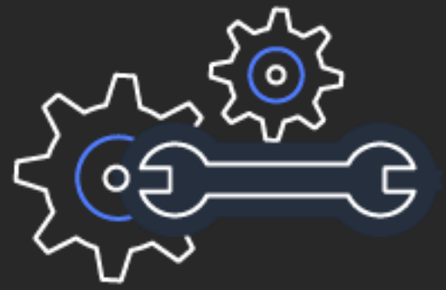
Optimize schemas in Amazon Redshift



AWS SCT product highlights

Assess

Plan



Convert schema and code

Optimize

Migrate data warehouses

Database migration assessment

Database migration assessment report

Source database:
DMS_SAMPLE.tharunk@oracle.c435kv7tmkj2.us-west-2.rds.amazonaws.com:8193:TEST
Oracle Database 12c Enterprise Edition 12.1.0.2.0 (64bit Production), Enterprise edition

Executive summary

We completed the analysis of your Oracle source database and estimate that 98% of the database storage objects and 96% of database code objects can be converted automatically or with minimal changes if you select Amazon RDS for PostgreSQL as your migration target. Database storage objects include schemas, tables, table constraints, indexes, types, collection types, sequences, synonyms, view-constraints, clusters and database links. Database code objects include triggers, views, materialized views, materialized view logs, procedures, functions, packages, package constants, package cursors, package exceptions, package variables, package functions, package procedures, package types, package collection types, scheduler-jobs, scheduler-programs and scheduler-schedules. Based on the source code syntax analysis, we estimate 99.8% (based on # lines of code) of your code can be converted to Amazon RDS for PostgreSQL automatically. To complete the migration, we recommend 6 conversion action(s) ranging from simple tasks to medium-complexity actions to significant conversion actions.

Migration guidance for database objects that could not be converted automatically can be found here

Database objects with conversion actions for Amazon RDS for PostgreSQL

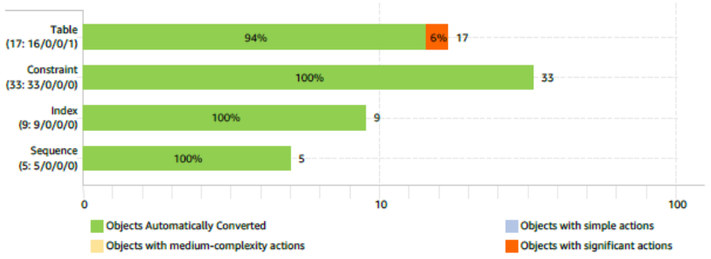
Of the total 64 database storage object(s) and 23 database code object(s) in the source database, we identifed 63 (98%) database storage object(s) and 22 (96%) database code object(s) that can be converted to Amazon RDS for PostgreSQL automatically or with minimal changes.

The target database version is less than PostgreSQL 11.1 (11.1). The converted code might not work properly.

1 (2%) database storage object(s) require 1 significant user action(s) to complete the conversion.

1 (4%) database code object(s) require 1 medium user action(s) to complete the conversion.

Figure: Conversion statistics for database storage objects



Connect AWS SCT to source and target databases

Run assessment report

Read executive summary

Follow detailed instructions

Database migration assessment report

Source database:
DMS_SAMPLE.tharunk@oracle.c435kv7tmkj2.us-west-2.rds.amazonaws.com:8193:TEST
Oracle Database 12c Enterprise Edition 12.1.0.2.0 (64bit Production), Enterprise edition

Package Procedure Changes

Not all package procedures can be converted automatically. Youll need to address these issues manually.

Issue 5584: Converted functions depends on the time zone settings

Recommended action: Review the transformed code, and set time zone manually if necessary.

Issue code: 5584 | Number of occurrences: 2 | Estimated complexity: Simple

Documentation references: http://www.postgresql.org/docs/9.6/static/functions-datetime.html

Schemas.DMS_SAMPLE.Packages.TICKETMANAGEMENT.Private procedures.TRANSFERTICKET: 1463:1469
Schemas.DMS_SAMPLE.Packages.TICKETMANAGEMENT.Public procedures.SELLTICKETS: 1772:1778

Package Function Changes

Not all package functions can be converted automatically. Youll need to address these issues manually.

Issue 5644: Unable automatically convert assign operation of array or global nested table,

Recommended action: Perform a manual conversion.

Issue code: 5644 | Number of occurrences: 1 | Estimated complexity: Medium

Schemas.DMS_SAMPLE.Packages.TICKETMANAGEMENT.Private functions.GET_OPEN_EVENTS: 270:297

Procedure Changes

Not all procedures can be converted automatically. Youll need to address these issues manually.

Issue 5103: Unable to convert hints

Recommended action: Use PostgreSQL methods for performance tuning.

Issue code: 5103 | Number of occurrences: 2 | Estimated complexity: Simple

Documentation references: http://www.postgresql.org/docs/9.6/static/geqo.html

Schemas.DMS_SAMPLE.Procedures.GENERATESEATS: 2041:2089
Schemas.DMS_SAMPLE.Procedures.GENERATE_TICKETS: 290:1184

Supported source and targets

Relational

NoSQL

Data lake

Data warehouse

* Supported via AWS SCT data extractors

Sources

Oracle, MySQL, PostgreSQL, SQL Server, MariaDB, Amazon Aurora, SAP ASE, Db2 LUW, SQL Azure

MongoDB, Cassandra

Amazon S3, AWS Snowball

Oracle, SQL Server, Netezza, Greenplum, Teradata, Vertica

Targets

Oracle, MySQL, PostgreSQL, SQL Server, MariaDB, Amazon Aurora, SAP ASE

Amazon DynamoDB, Amazon DocumentDB (with MongoDB compatibility)

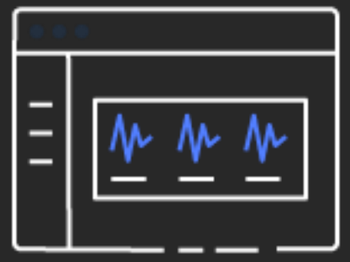
Amazon Elasticsearch Service (Amazon ES), Amazon Kinesis Data Streams, Amazon Simple Storage Service (Amazon S3)

Amazon Redshift

AWS DMS product highlights

Secure

Assess



Validate

AWS Snowball integration

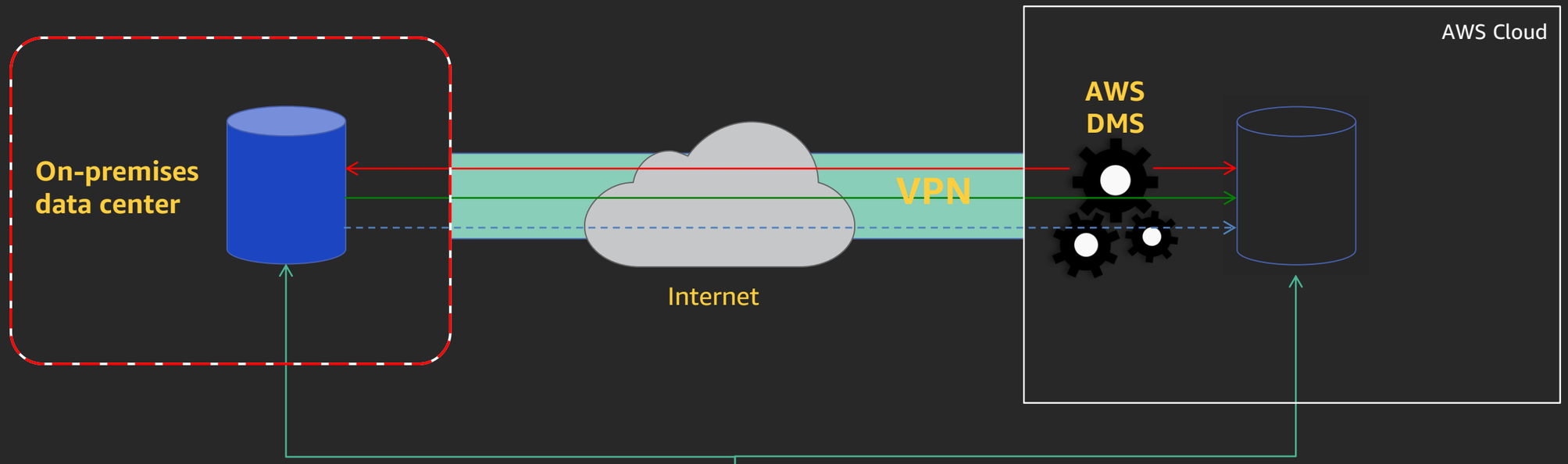
Monitor

Stream data

Low cost

Multiple options

The migration process



- Start a replication instance
- Connect to source and target databases
- Select tables, schemas, or databases

- Let AWS DMS load data, and keep them in sync
- Switch applications over to the target once in sync at your convenience

Migration use case – Amazon

Where we started…

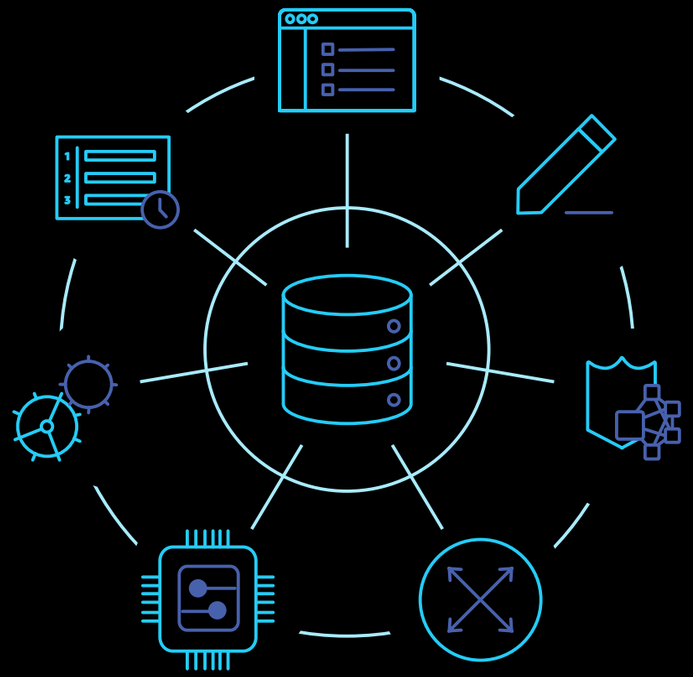
Amazon's Oracle footprint in 2017

- 7,406 Oracle database instances
- 20,000 CPU cores
- 100s of PBs of data
- >15% annual data growth since 2014
- 10s of Amazon data centers globally
- Thousands of sq. ft. of real estate
- Millions of $USD in operation costs
- Many careers based on Oracle

Where we started…

Challenges faced with Oracle Databases

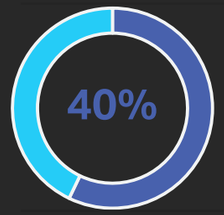
- Complex and costly to scale for growing service throughput
- Expensive and punitive Oracle licenses
- Complicated, expensive, and error-prone database administration
- 100s of hours/month in administrative tasks
- Complicated and inefficient hardware provisioning
- 400 hours/year forecasting and planning database hardware capacity



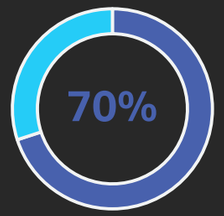
Overall benefits for Amazon.com, Inc.

Simplified cost allocation

New career paths for Oracle database engineers



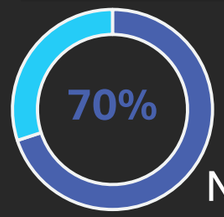
Reduction in latency, with 2x the number of transactions



Reduction in infrastructure costs

Eliminated capex infrastructure costs

Transformed organization with cost model where each team now manages its own compute and storage infrastructure based on predicted usage and growth



Reduction in database administration overhead

Now focus on tasks like query optimization and performance monitoring

Migration planning – Amazon

Amazon database freedom journey

Project Rollingstone

1. Eliminate Amazon's dependency on relational databases for critical services
2. Eliminate use of Oracle

Amazon DynamoDB

Other considerations:

- Stringent timeline
- Ensure no performance degradation of service
- Ensure BAU tasks are not impacted

Amazon Aurora

Amazon database freedom journey

1. Current application and database assessment

2. For relational migrations, MySQL vs. PostgreSQL:
    a) Application/database use case fit
    b) Migration complexity

3. Aspects are not considered as part of this evaluation:
    a) Operational or nonfunctional considerations
    b) Performance
    c) Native integration with AWS services

4. Majorly migration happened across three major databases
    a) Amazon DynamoDB – Nonrelational
    b) MySQL (Amazon RDS/MySQL)
    c) PostgreSQL (Amazon RDS / Amazon Aurora)

Amazon DynamoDB

Amazon Aurora

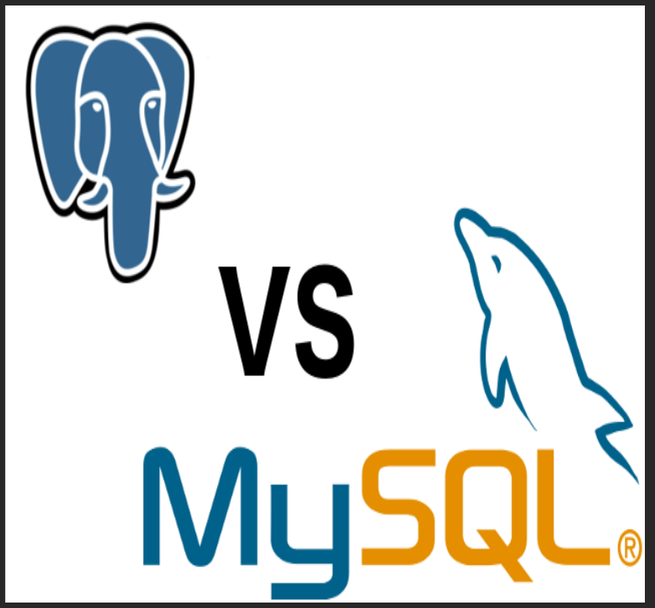
Decision Matrix : MySQL vs. PostgreSQL

Decision evaluation approach:

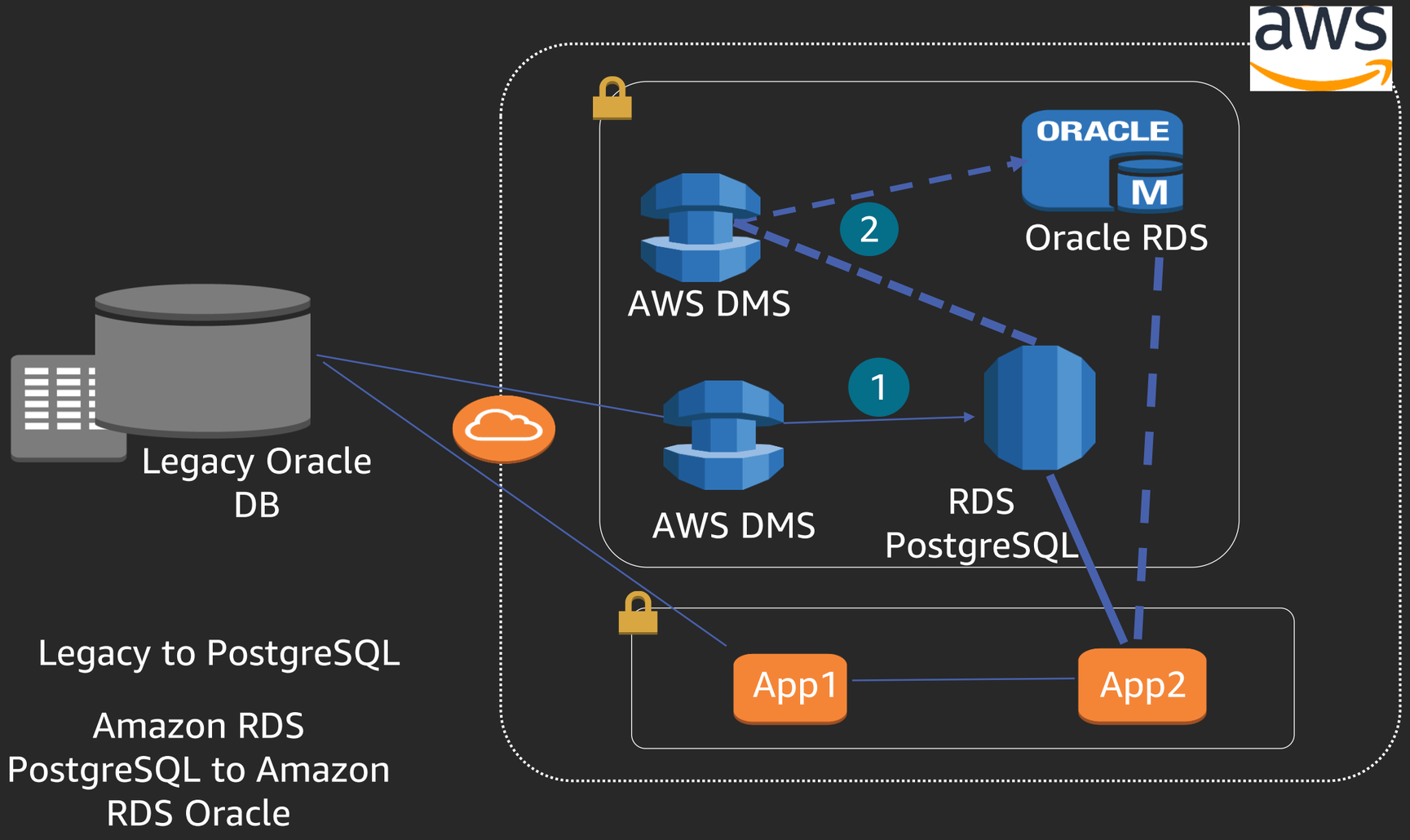
1. Determine team's appropriate expertise in which database

2. Use AWS Schema Conversion Tool for assessment of the differences from Oracle to MySQL/PostgreSQL

3. In case the above steps don't help, deep dive into database features
    - ACID compliance with multi-version concurrency control
    - OLTP use-case
    - OLAP use-case
    - Support for advanced data types such as JSON and XML
    - Partitioning
    - Built in scheduler



Migration architecture



1 Legacy to PostgreSQL

2 Amazon RDS PostgreSQL to Amazon RDS Oracle

Top 5 migration challenges – Amazon

Top AWS DMS challenges/learnings

- ORA-01555 (Snapshot too old) error with large table migrations
- Performance issues with Large Objects (LOBs)
- Migration task performance issues with high TPS source systems
- Datatype conversion issues
- 4 Byte character set issue
- Transaction isolation

ORA-01555 (Snapshot too old)

- Oracle’s default read consistency mechanism (using undo segments) which ensures that all rows from a select are intact from the time it was executed
- AWS DMS runs a “select” on the source for a table when the full load phase starts

ORA-01555: snapshot too old: rollback segment number %n with name "%segname" too small

What to do in Oracle?

1. Reschedule long-running queries when the system has less DML load
2. Increasing the size of your rollback segment (undo) size

Example – One of the internal AWS teams Timber migrated >=30 TB data store and had more than 2 tables of >=4 TB each

How do we handle ORA-01555 on the source for large table migrations in AWS DMS?

Use parallel-load AWS DMS feature:

1. This feature basically chunks the table based on partitions or column ranges and loads each chunk in parallel
2. Documentation
3. Start a single CDC task for the entire table from a custom CDC start time

Read from a fully synced active data guard standby or snapshot standby without impacting source:

1. Sync standby from source
2. Stop the MRP process and note the SCN at which the sync is stopped
3. Run full load from standby
4. Run CDC from primary/ADG instance using the SCN

Truncation and performance issues with Large Object (LOBs)

- **MANDATORY:** For AWS DMS to **replicate** LOBs, the table must have a primary key or a unique index because LOB replication is a two-step process

1. AWS DMS first migrates the entire row without LOBs
2. AWS DMS then using the PK/UK from the first step, queries for LOB data on source and then updates the target table with LOB data
3. Commit

PS: If you choose **FULL LOB MODE** in AWS DMS, then even the migration (full load) phase will be done in the above manner and not in bulk mode

Truncation and performance issues with Large Object (LOBs)

Error handling

1. For Oracle and PostgreSQL sources, use: **failTasksOnLobTruncation**
2. Have custom Amazon CloudWatch alarms on AWS DMS task logs to trap for LOB truncation messages
3. Use validation within AWS DMS

Performance

1. Use parallel-load feature in AWS DMS
2. For Oracle targets with LOBs greater than 10 MB, disable **directpathfullload** in AWS DMS
3. Use limited lob mode, if the LOB size is deterministic from the database and application side
4. Use inline lob mode, if the LOB sizes are not deterministic
5. Use per table lob mode if many tables needed to be part of the same task

Important things to note while migrating LOBs

1. Need to plan migrations for tables that have no PKs and contain LOBs. Here is a query to identify those tables:

   - `SELECT owner,table_name FROM dba_tables where owner=‘schema_name' and table_name NOT IN (SELECT table_name FROM dba_constraints WHERE constraint_type ='P' and owner='schema_name ') and table_name in (select DISTINCT table_name from dba_tab_cols where data_Type IN ('CLOB', 'LOB', 'BLOB') and owner ='schema_name ');`

2. Find the max LOB size using Oracle system tables:

```
select 'select (max(length(' || COLUMN_NAME || '))/(1024)) as "Size in KB" from '
|| owner || '.' || TABLE_NAME ||';' "maxlobsizeqry" from dba_tab_cols where
owner=‘schema_name' and data_type in ('CLOB','BLOB','LOB');
```

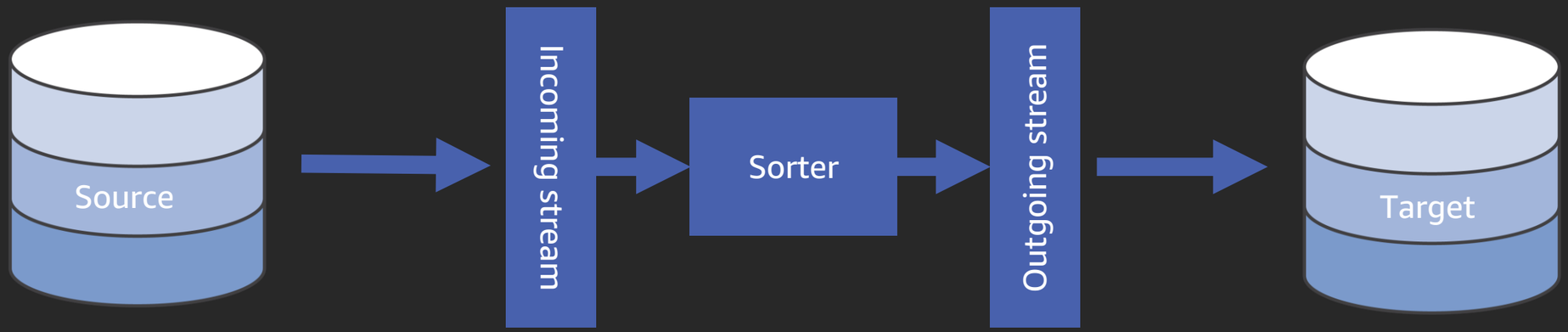
3. Attention to stream buffer settings

Migration task performance issues with high TPS source systems

Full load process for Oracle to PostgreSQL/MySQL Migration

1. Run a “Select *” on the source
2. Unload data into the replication instance memory and prepare to create comma-delimited CSV files
3. Use **copy/load data infile** command in PostgreSQL to copy data from the replication instance to the target instance

Change data capture process



Transactional apply vs. Batch apply

Transactional apply

- AWS DMS does this by default
- All transactions are applied in the same order as it happened on the source
- Designed to work with foreign key constraints and triggers being active on the target after applying cached changes

Batch apply

- Read changes from the source engine and build a batch
  - Based on timeout
  - Based on batch size
- Create an intermediate net changes table in the target with all changes from a batch
- Apply a net changes algorithm to condense changes and apply separate DMLs in one transaction
- Foreign keys and triggers need to be disabled

EXAMPLE – Clicks team used batch apply for the higher TPS workload described earlier

Data type conversion issues

Two-step process:

1. Convert source data type to AWS DMS type data type
2. Convert AWS DMS type data type to target data type

Scenario – Multiple internal customers:

- Number in Oracle becomes a double in PostgreSQL despite it having 0 precision because of the way Oracle stores it. E.g., an ID column with entry 2045 becomes 2045.00 on the target
- Number to Boolean migration

Recommended workarounds:

1. For small tables, use a trigger on the target to convert this into an INT or BIGINT
2. For larger tables, stop full load, change column definition, resume task into CDC to get rid of extra unwanted precision

Solution:

AWS DMS data type transformations

4-byte UTF8 character set issues

- AWS DMS used to not support migrating 4-byte UTF8 characters
  - ➢ This limitation was overcome with the launch of AWS DMS version 2.x
- During migration/replication phase, AWS DMS will encode supplementary characters to UTF-16 which causes issues as encoding strings in surrogate range are invalid UTF8 chars and fails when written to the target

Workarounds:

- Run CSSCAN on Oracle or use the following query to find the problematic characters:

SELECT <Primary_Key_Column>||' '||<Problematic Column> FROM Table name WHERE REGEXP_LIKE(<Problematic Column>, UNISTR('[\D800-\DFFF]'));

- Replace invalid characters on the source manually and run the migration
- Identify invalid characters and replace them during the migration through an extra connection attribute

Transaction isolation

- Isolation level in Oracle is read-committed for every transaction
    - ➢ This was something that was not an issue with regards to migration to PostgreSQL, but an issue with MySQL

- MySQL InnoDB engine default isolation level set to **repeatable read.** This change also requires autocommit to be disabled

When using AWS DMS, in the initial migration (full load) phase with **parallelLoadThreads** parameter, rollbacks were happening, causing data mismatch between source and target

Workarounds:

- AWS DMS allows configuration of session-level parameters using extra connection attribute:

initstmt=SET AUTOCOMMIT=1

Solution

- This was permanently fixed with AWS DMS version 3.x

Before you begin

- Identify key “technical blockers within your organization” and unblock them

- Database skills are important. Build a core team involving DBAs and SDEs.

- Build a community to share knowledge across the organization

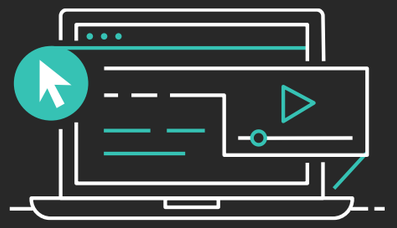
- Establish a process to provide feedback to AWS

"Difficulties are just things to overcome, after all. When things are easy, I hate it."

Sir Ernest Shackleton

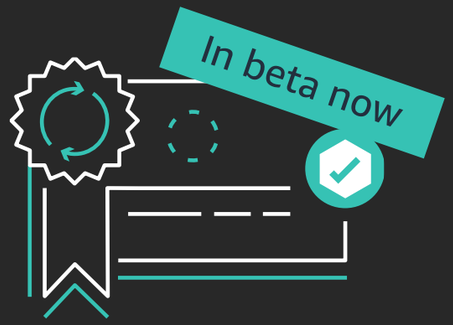
Learn databases with AWS Training and Certification

Resources created by the experts at AWS to help you build and validate database skills



25+ free digital training courses cover topics and services related to databases, including:

- Amazon Aurora
- Amazon Neptune
- Amazon DocumentDB
- Amazon DynamoDB
- Amazon ElastiCache
- Amazon Redshift
- Amazon RDS



Validate expertise with the new **AWS Certified Database - Specialty** beta exam

Visit aws.training

aws training and certification

Thank you!

AWS re:Invent

aws

Please complete the session survey in the mobile app.