



Migration from RDS for Oracle to RDS/Aurora PostgreSQL

Krishna Sarabu

Sr Database Specialist SA

Joseph DiCaro

Sr Specialist SA

Agenda

- Why migrate
- Modernize and migrate
- Migration options
- Top 10 best practices
- How AWS can help

Why migrate from RDS Oracle to RDS/Aurora PostgreSQL?



Reduced
Cost

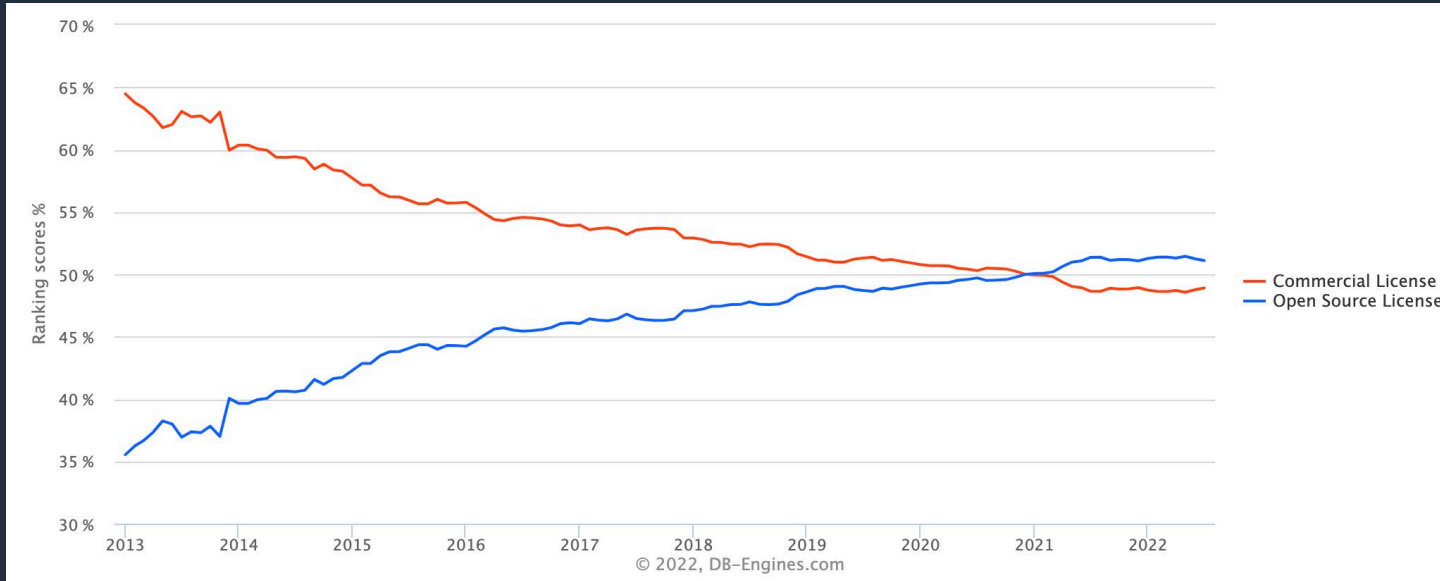


Increased
Agility

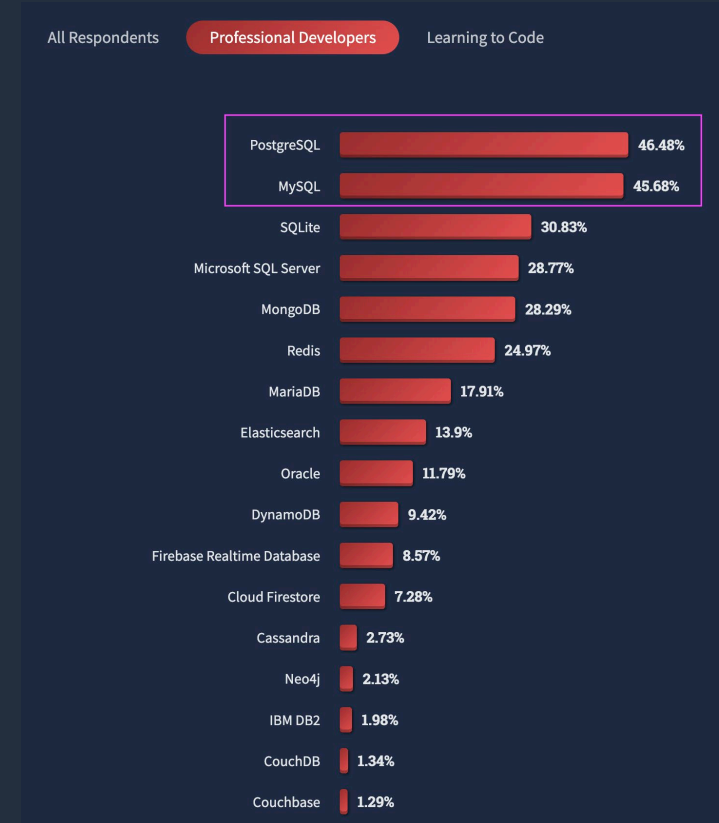


Compatibility

Existing state of commercial databases



Source: DB-Engines.com



Source: Stack Overflow Developer Survey, 2022

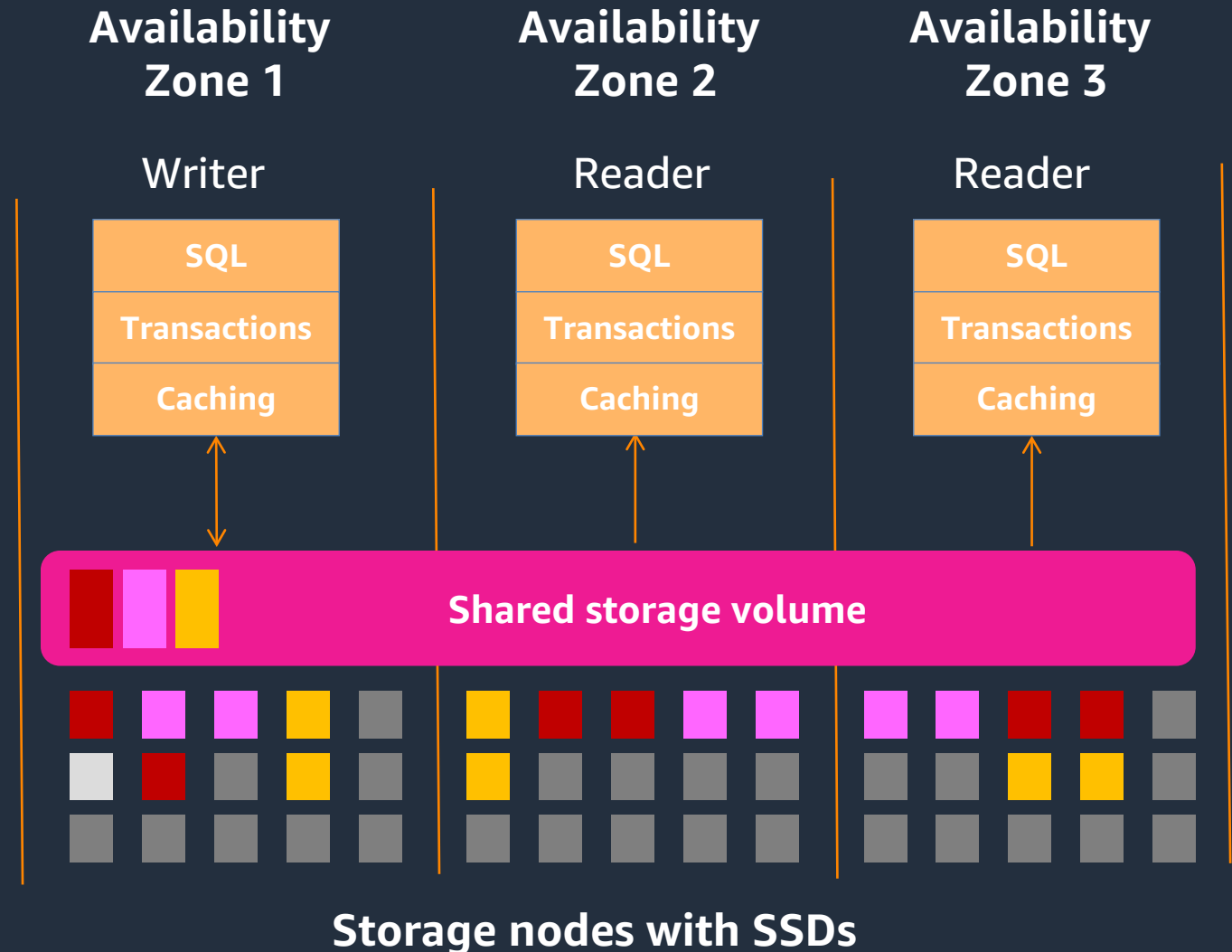
Modernize and migrate to Amazon RDS/Aurora PostgreSQL

Amazon RDS PostgreSQL

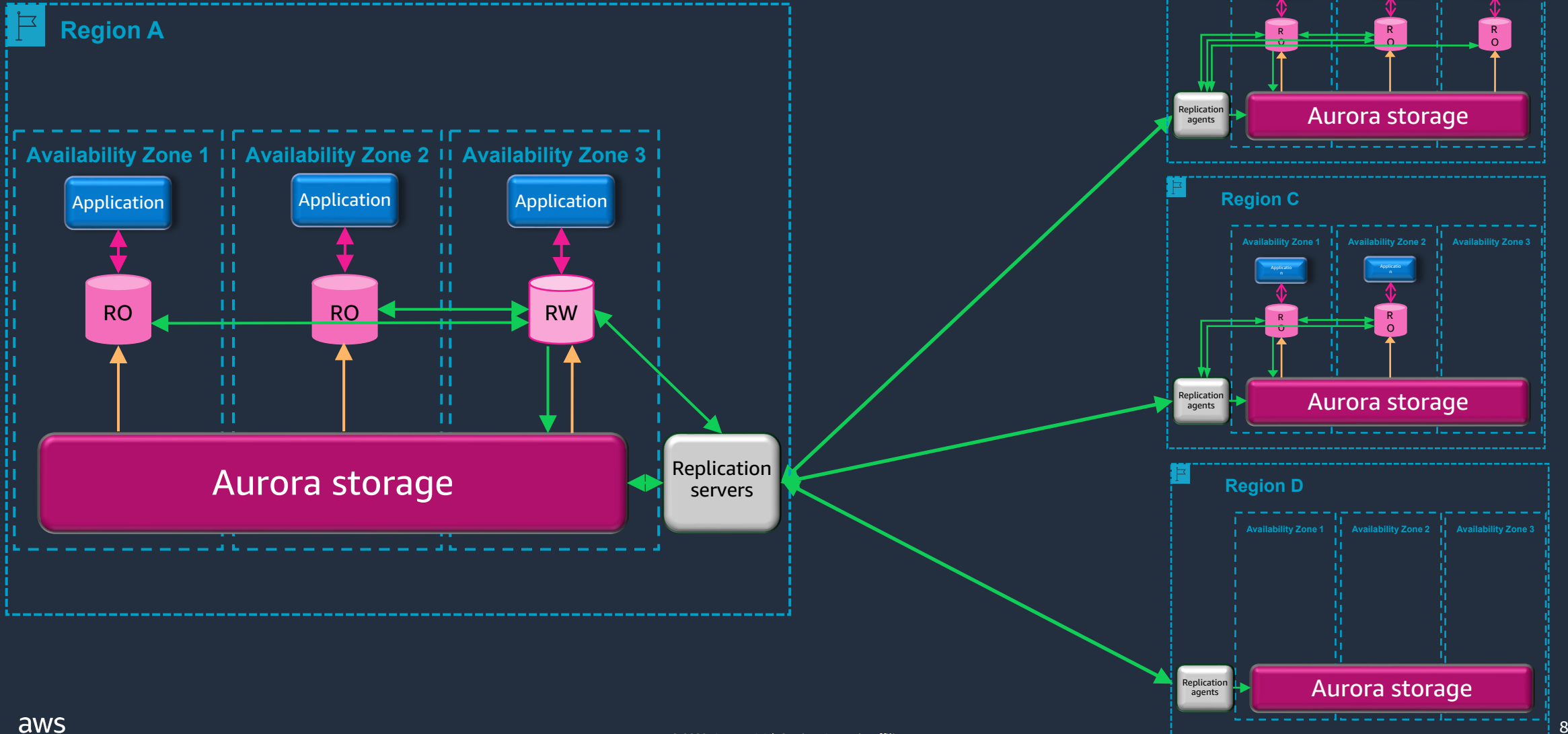
- Fully managed
- Scale compute and storage with ease
- Highly Secure
 - KMS (Encrypt at rest, Encrypt in transit)
 - IAM
 - Kerberos
- High Availability and Durability
- Read Replicas for read intensive workload horizontal scaling
- Amazon RDS supports PostgreSQL 14, 13, 12, 11, 10 engine versions
- PostgreSQL is more feature rich for developers than Oracle
 - Stored procedure languages
 - Foreign data wrappers
 - Data types
 - Spatial
 - Extensions

Amazon Aurora leverages a scale-out, distributed architecture

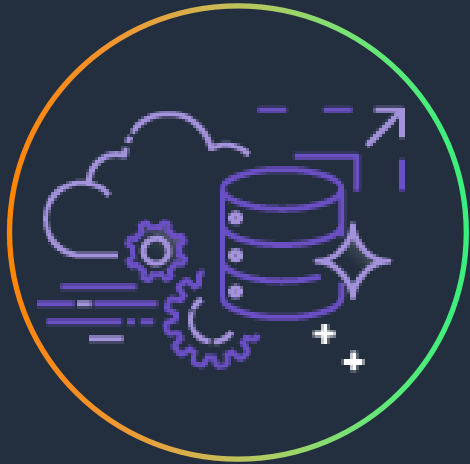
- Purpose-built log-structured distributed storage system designed for databases
- Storage volume is striped across hundreds of storage nodes distributed over 3 different availability zones
- Six copies of data, two copies in each availability zone to protect against AZ+1 failures
- Writer and readers (up to 15) all point to the same storage



Amazon Aurora global database



Amazon Aurora Serverless v2



On-demand and autoscaling configuration

Automatically scales capacity based on application needs

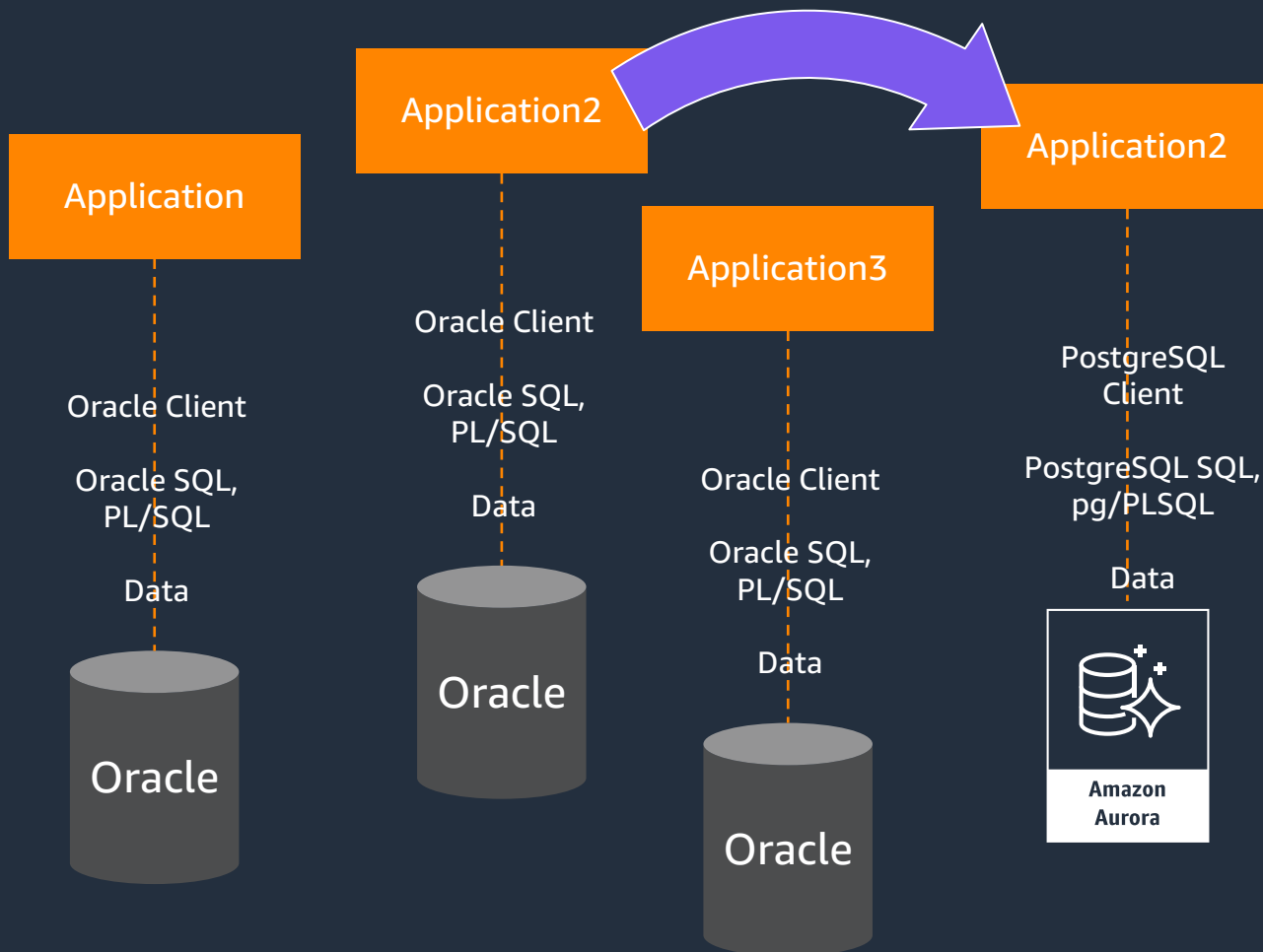
Simple **pay-per-use** pricing per second

Next version scales instantly to support demanding applications

Worry-free database capacity management

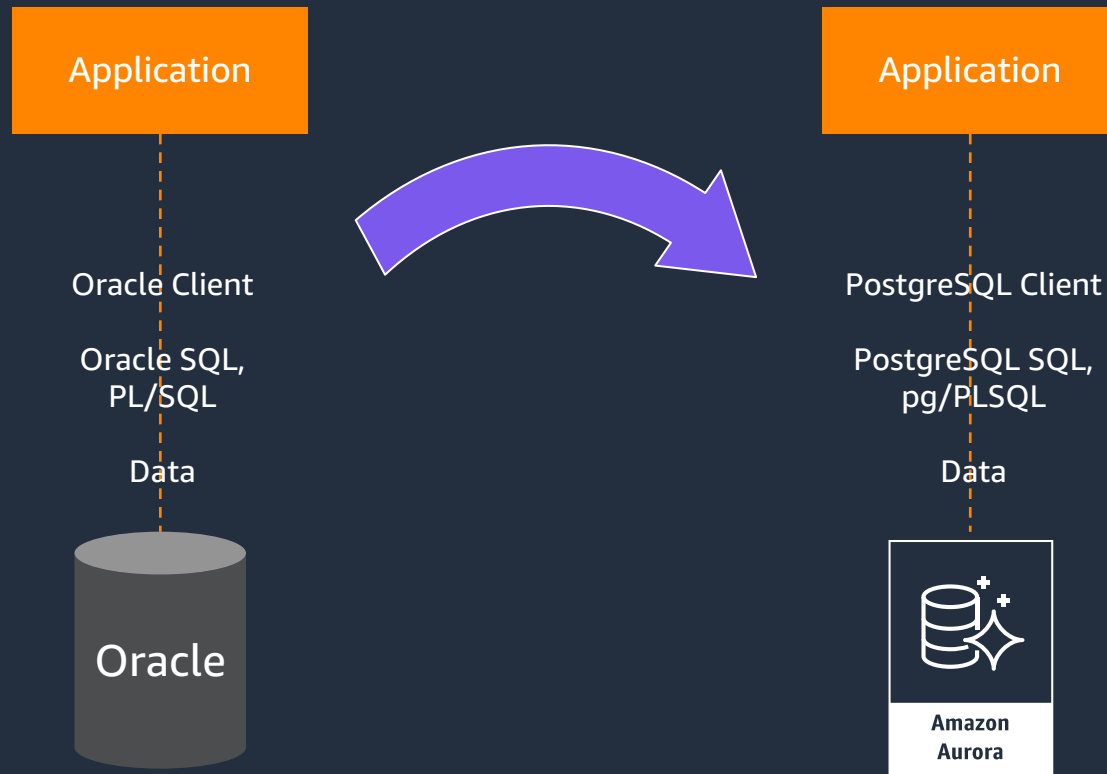
Migrating from RDS Oracle to Amazon RDS/Aurora PostgreSQL

Migrating RDS Oracle to Amazon RDS/Aurora PostgreSQL



1. Assess Application+Database pairs for migration complexity and classification
2. Switch Database engine from Oracle to PostgreSQL
3. Evolve from Oracle RDS to Amazon RDS/Aurora
4. Convert database schema (tables, datatypes, etc.) from Oracle to PostgreSQL
5. Convert database code-objects (functions, triggers, etc.) from PL/SQL to pg/PLSQL
6. Modify application SQL from Oracle SQL to PostgreSQL (ANSI) SQL
7. Migrate data from Oracle to PostgreSQL
8. Replace Oracle client with PostgreSQL client
9. Test application
10. Cut over production

Migrating RDS Oracle to Amazon RDS/Aurora PostgreSQL



1. Assess Application+Database pairs for migration complexity and classification
2. Switch database engine from Oracle to PostgreSQL
3. Evolve from customer-managed to managed database service
4. Convert database schema (tables, datatypes, etc.) from Oracle to PostgreSQL
5. Convert database code-objects (functions, triggers, etc.) from PL/SQL to pg/PLSQL
6. Modify application SQL from Oracle SQL to PostgreSQL (ANSI) SQL
7. Migrate data from Oracle to PostgreSQL
8. Replace Oracle client with PostgreSQL client
9. Test application
10. Cut over production

AWS Tools to help migrate to RDS/Aurora PostgreSQL



AWS Schema
Conversion Tool

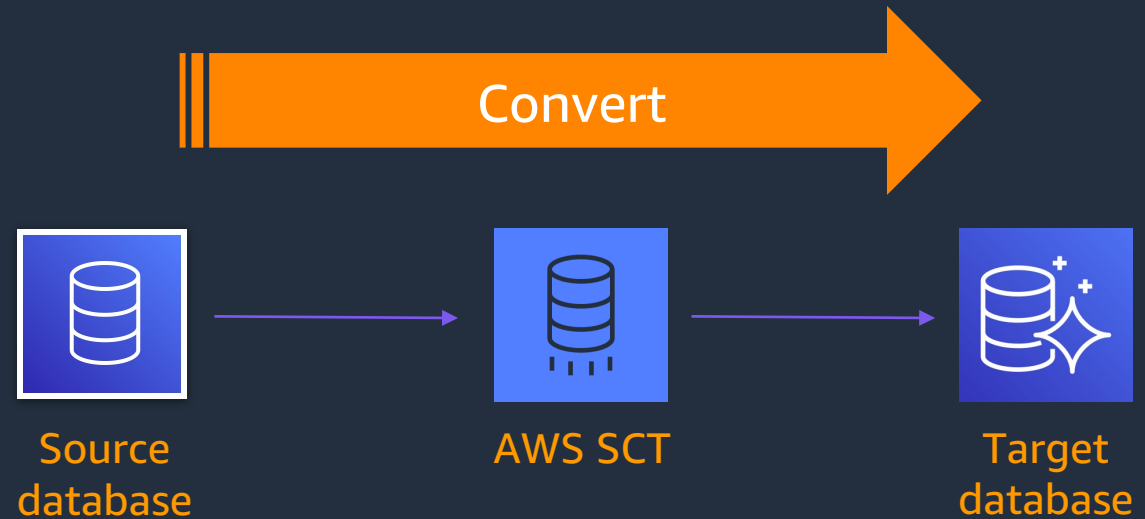


AWS Database
Migration
Service

1. Assess Application+Database pairs for migration complexity and classification
2. Switch database engine from Oracle to PostgreSQL
3. Evolve from commercial database engine to Amazon RDS/Aurora PostgreSQL
4. Convert database schema (tables, datatypes, etc.) from Oracle to PostgreSQL
5. Convert database code-objects (functions, triggers, etc.) from PL/SQL to pg/PLSQL
6. Modify application SQL from Oracle SQL to PostgreSQL (ANSI) SQL
7. Migrate data from Oracle to RDS/Aurora PostgreSQL
8. Replace Oracle Client with PostgreSQL Client
9. Test Application
10. Cut over production

AWS Schema Conversion Tool (SCT)

SCT makes heterogeneous database migrations predictable by automatically converting the source database schema and a majority of the database code objects, including views, stored procedures, and functions, to a format compatible with the target database

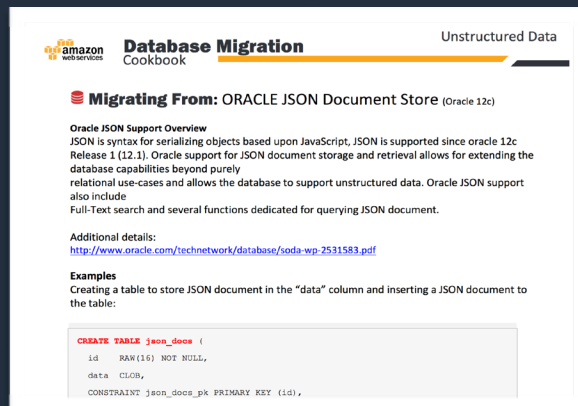


Features

- Database Migration Assessment report for choosing the right target engine
- Automatic conversion for eligible database objects and code
- Convert embedded application code
- Secure connections to your databases with SSL
- Code browser to highlight places where manual edits are required

AWS Schema Conversion Tool (SCT) – best practices

- Use SCT multi-server assessment feature
- Save assessment report as a CSV
- Do not treat Target database like the Source
- Consult the Oracle to Aurora PostgreSQL Migration Playbook



amazon Database Migration Cookbook Unstructured Data

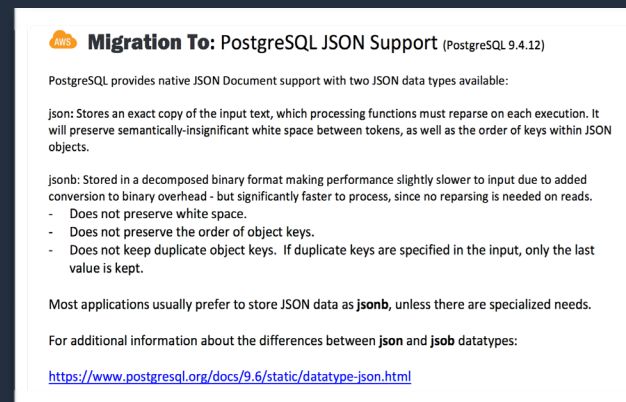
Migrating From: ORACLE JSON Document Store (Oracle 12c)

Oracle JSON Support Overview
JSON is syntax for serializing objects based upon JavaScript, JSON is supported since Oracle 12c Release 1 (12.1). Oracle support for JSON document storage and retrieval allows for extending the database capabilities beyond purely relational use-cases and allows the database to support unstructured data. Oracle JSON support also includes Full-Text search and several functions dedicated for querying JSON document.

Additional details:
<http://www.oracle.com/technetwork/database/soda-wp-2531583.pdf>

Examples
Creating a table to store JSON document in the "data" column and inserting a JSON document to the table:

```
CREATE TABLE json_docs (  
  id RAW(16) NOT NULL,  
  data CLOB,  
  CONSTRAINT json_docs_pk PRIMARY KEY (id),
```



Migration To: PostgreSQL JSON Support (PostgreSQL 9.4.12)

PostgreSQL provides native JSON Document support with two JSON data types available:

json: Stores an exact copy of the input text, which processing functions must reparse on each execution. It will preserve semantically-insignificant white space between tokens, as well as the order of keys within JSON objects.

jsonb: Stored in a decomposed binary format making performance slightly slower to input due to added conversion to binary overhead - but significantly faster to process, since no reparsing is needed on reads.

- Does not preserve white space.
- Does not preserve the order of object keys.
- Does not keep duplicate object keys. If duplicate keys are specified in the input, only the last value is kept.

Most applications usually prefer to store JSON data as **jsonb**, unless there are specialized needs.

For additional information about the differences between **json** and **jsonb** datatypes:
<https://www.postgresql.org/docs/9.6/static/datatype-json.html>

SCT resources

- Migration documentation [AWS Database Migration Service Documentation](#)
- Check the [AWS Database Blog](#) for additional topics
- You can narrow down with the [Amazon Aurora](#) and [RDS For PostgreSQL](#) tags

AWS Database Migration Service (DMS)

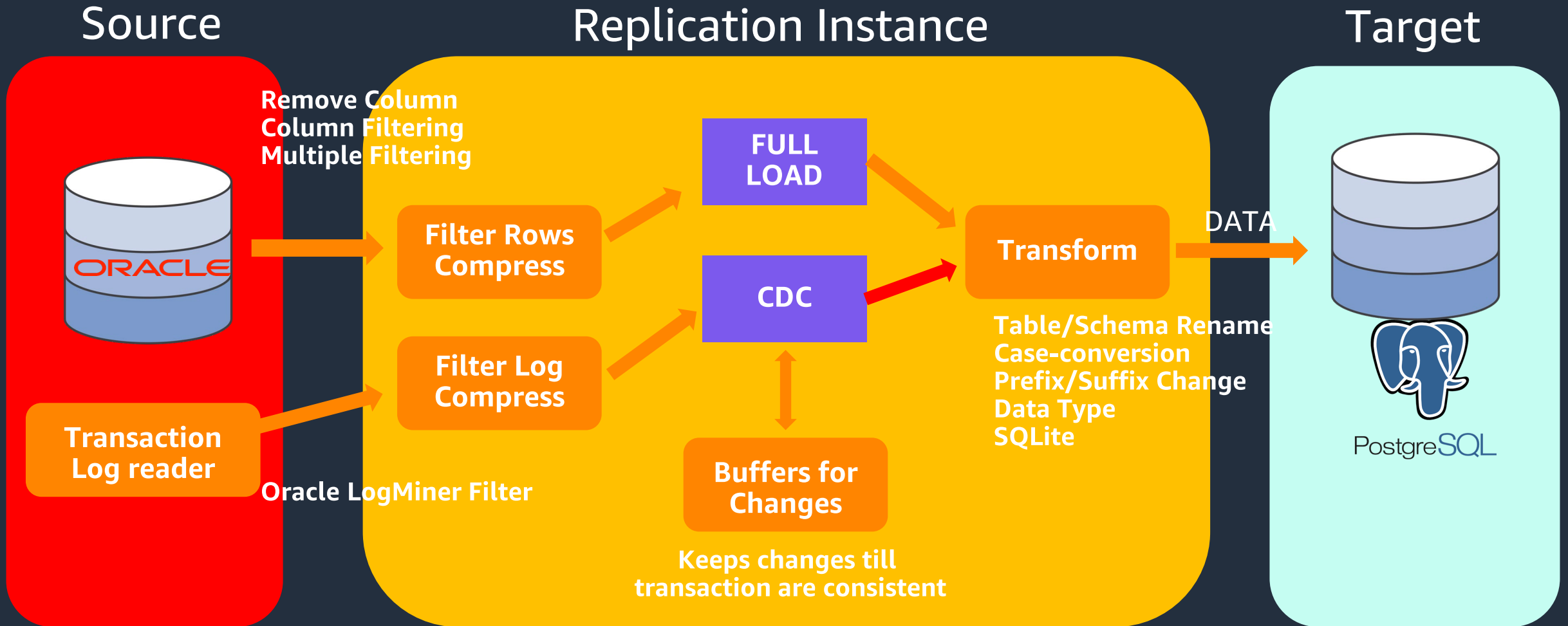
- Start your first migration in *10 minutes or less*
- Keep your *apps running* during the migration
- *Replicate* from within, to, or from AWS
- Move data to the same or *different database engine*

Sources*
Oracle
SQL Server
Azure SQL Server
PostgreSQL
MySQL
SAP ASE
MongoDB
Amazon S3
IBM Db2 (LUW)
Amazon DocumentDB

Targets*
Oracle
SQL Server
PostgreSQL
MySQL
SAP ASE
Amazon Redshift
Amazon S3
Amazon DynamoDB
Amazon Kinesis
Amazon OpenSearch
Amazon DocumentDB
Amazon Neptune
Apache Kafka
Redis

Consult DMS Documentation for latest DMS sources and targets

AWS Database Migration Service (DMS)



- ✓ Start a replication instance
- ✓ Connect the source and target endpoints
- ✓ DMS FULL LOAD the data from Source to Target
- ✓ DMS Change Data Capture to replicate ongoing changes
- ✓ At steady state – take an outage, validate & redirect connection

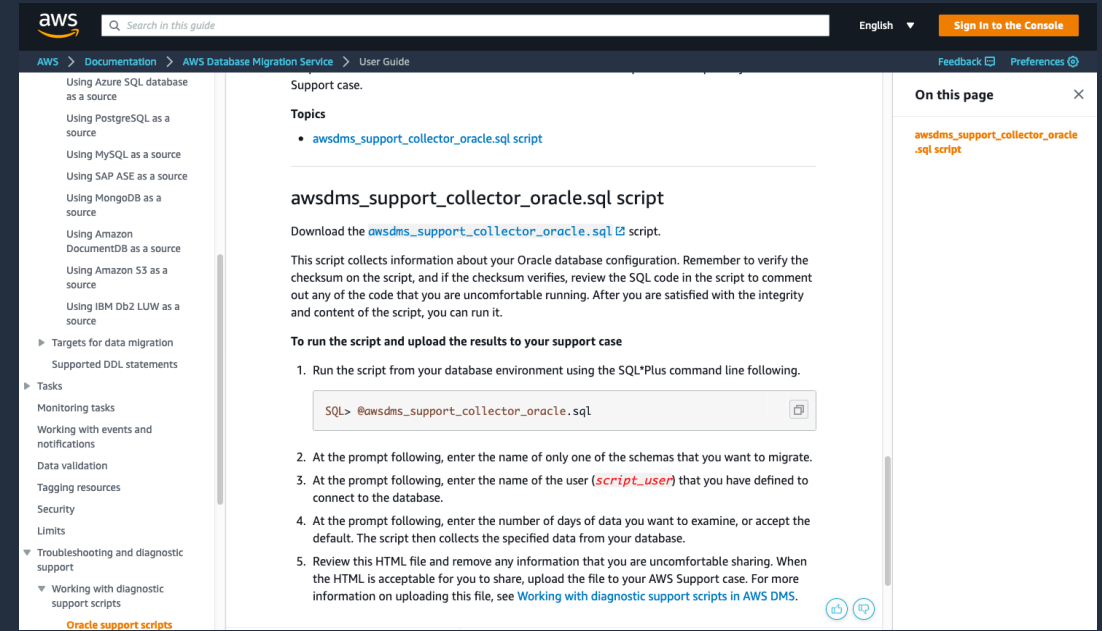
DMS best practices – Oracle as a Source

Start with the Oracle as a Source chapter in the DMS documentation. It is important. It is updated regularly.

https://docs.aws.amazon.com/dms/latest/userguide/CHAP_Source.Oracle.html

There is a very good support SQL*Plus script you can use as a pre-check:

https://docs.aws.amazon.com/dms/latest/userguide/CHAP_SupportScripts.Oracle.html



The screenshot shows the AWS documentation page for the `awsdms_support_collector_oracle.sql` script. The page is titled "Support case." and includes a search bar at the top. The left sidebar contains a navigation menu with various topics. The main content area is titled "awsdms_support_collector_oracle.sql script" and provides instructions on how to run the script and upload the results to a support case. The instructions are as follows:

1. Run the script from your database environment using the SQL*Plus command line following.

```
SQL> @awsdms_support_collector_oracle.sql
```
2. At the prompt following, enter the name of only one of the schemas that you want to migrate.
3. At the prompt following, enter the name of the user (`script_user`) that you have defined to connect to the database.
4. At the prompt following, enter the number of days of data you want to examine, or accept the default. The script then collects the specified data from your database.
5. Review this HTML file and remove any information that you are uncomfortable sharing. When the HTML is acceptable for you to share, upload the file to your AWS Support case. For more information on uploading this file, see [Working with diagnostic support scripts in AWS DMS](#).

DMS best practices – Handling Oracle LOBS

- What LOB columns do you have?
- What is the biggest LOB size for each LOB column?
- Do any of the tables with LOBs not have PKeys?
- Consider using [per table LOB settings](#) in DMS task

✓ 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 ');
```

✓ 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');
```

DMS best practices – Table Mappings JSON

- Understand the richness of the [Table Mappings](#) JSON
- You can specify filters
- You can adjust for the UPPERCASE (Oracle) vs lowercase (PostgreSQL) data dictionary differences
- You can use different settings (LOB, parallel) by table
- You can replicate big tables in parallel chunks
- You can order big tables to load first

DMS best practices – Plan time for setup and dry runs

- When migrating databases to the cloud, almost every customer's pre-cloud database configuration is unique
- Allow time in your schedule to work through any site-specific source configuration setup
- Allot time for dry runs to make sure you do not have surprises for the production cutover
- Reach out to AWS Support if you run into technical issues

DMS best practices – Understand the scope

Understand the scope of DMS. For instance,

- It does not replicate stored procedures
- It does not replicate sequence values
- It does not enable/disable foreign keys or triggers for you
- Do not migrate objects and data not being used
- Helpful [blog](#) showing a realistic workflow:

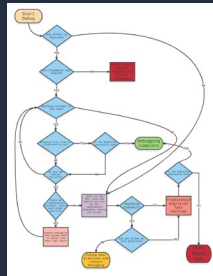
1. Create your schema in the target database.
2. Drop foreign keys and secondary indexes on the target database, and disable triggers.
3. Set up a DMS task to replicate your data – full load and change data capture (CDC).
4. Stop the task when the full load phase is complete, and recreate foreign keys and secondary indexes.
5. Enable the DMS task.
6. Migrate tools and software, and enable triggers.

DMS resources

Good [blog](#) for dealing with troubleshooting:

As we traverse the flow chart through all the decision boxes, we explore what the importance of each step is in this troubleshooting process. We talk about a few tips and tricks that we have picked up along the way while helping debug thousands of AWS DMS migrations. As stated earlier, migrations are complex and require some configuration tuning and testing based on a number of factors to be successful. As you determine the best possible configuration parameters for your migration using AWS DMS, here are a few factors to consider:

1. Infrastructural issues on the AWS DMS replication instance or source database or target database instances
2. Network issues between the source and replication instance or between the replication instance and the target
3. Data-related issues on the sources
4. AWS DMS limitations (you can find specific limitations for each of our [sources](#) and [targets](#))



[Oracle to Aurora PostgreSQL migration playbook](#)



Top 10 best practices when migrating from Oracle to PostgreSQL

1. Tablespaces

- In PostgreSQL, tablespaces are just directory locations
- Provides no real benefit unless the database spans multiple mount points

2. Index Types

PostgreSQL has more and different types of indexes than Oracle

- B-tree
- Hash
- GIN
- GiST
- SP-GiST
- BRIN

PostgreSQL can use indexes on LIKE queries

```
CREATE INDEX idx_users_lname
ON users USING gin (lname gin_trgm_ops);
```

```
EXPLAIN SELECT * FROM users WHERE lname LIKE '%ing%';

          QUERY PLAN
-----
Bitmap Heap Scan on users (cost=8.00..12.02 rows=1 width=654)
  Recheck Cond: ((lname)::text ~~ '%ing%')::text)
  -> Bitmap Index Scan on idx_users_lname
      (cost=0.00..8.00 rows=1 width=0)
      Index Cond: ((lname)::text ~~ '%ing%')::text)
```

3. Data Types

Oracle has a few main data types that are typically used

- VARCHAR2
- DATE
- NUMBER

And a couple of Large object types

- CLOB
- BLOB

PostgreSQL has 64 base types and can be extended for more

- Number 38 digits Vs Numeric 131,072 digits (and 16,383 decimals)
- Numeric Vs bigint performance

4. Numeric Vs bigint

```
CREATE TABLE t1 (c1 numeric);  
CREATE TABLE t2 (c1 numeric, c2 numeric);
```

```
test=> SELECT count(*)  
        FROM t1  
        INNER JOIN t2  
        ON (t1.c1 = t2.c1);
```

count

10000000

(1 row)

Time: 2757.392 ms (00:02.757)

```
CREATE TABLE t1 (c1 bigint);  
CREATE TABLE t2 (c1 bigint, c2 bigint);
```

```
test=> SELECT count(*)  
        FROM t1  
        INNER JOIN t2  
        ON (t1.c1 = t2.c1);
```

count

10000000

(1 row)

Time: 1977.685 ms (00:01.978)

5. Dual table

- FROM clause is optional in PostgreSQL
- Do not mock a DUAL table
- Create it as a VIEW if necessary

```
test=# SELECT CURRENT_DATE;

 current_date 
-----
 2020-02-12
(1 row)
```

```
test=> CREATE VIEW dual AS SELECT 'X'::varchar AS dummy;
CREATE VIEW

test=> SELECT current_date FROM dual;
 current_date 
-----
 2020-02-12
(1 row)
```

6. Nulls

- PostgreSQL and Oracle handle nulls a bit differently
- Often seen with string concatenation
- Unique constraints

```
SQL> select null || ' ' || 'abc' from dual;  
  
NULL  
----  
abc
```

```
SQL> create table test1(col1 number(38), col2 number(38), constraint test1_uk1 unique (col1, col2));  
Table created.  
  
SQL> insert into test1 values(1, Null);  
1 row created.  
  
SQL> insert into test1 values(1, Null);  
insert into test1 values(1, Null)  
*  
ERROR at line 1:  
ORA-00001: unique constraint (SYS.TEST1_UK1) violated
```

```
postgres=> select null || ' ' || 'abc' ;  
?column?  
-----  
  
(1 row)  
  
postgres=> select coalesce(null, '') || ' ' || coalesce('abc', '');  
?column?  
-----  
  
abc  
(1 row)
```

```
postgres=> create table test1(col1 bigint, col2 bigint, constraint test1_uk1 unique (col1, col2));  
CREATE TABLE  
postgres=> insert into test1 values (1, Null);  
INSERT 0 1  
postgres=> insert into test1 values (1, Null);  
INSERT 0 1  
postgres=> insert into test1 values (1, Null);  
INSERT 0 1
```

7. Synonyms

Synonyms are used to not fully qualify cross schema objects

- Mostly a convenience feature
- In PostgreSQL, `search_path` can accomplish many of the same things and is less tedious to setup

```
test=# SET search_path = user1, user2, public;
SET

test=# SELECT get_int();
 get_int
-----
        1
(1 row)
```


8. Sequences

Sequences are cached per session in PostgreSQL

```
SQL> create sequence mytest_seq cache 100 ;  
Sequence created.
```

Session 1

```
SQL> select mytest_seq.nextval from dual;  
  
NEXTVAL  
-----  
1
```

Session 2

```
SQL> select mytest_seq.nextval from dual;  
  
NEXTVAL  
-----  
2
```

```
postgres=# create sequence mytest_seq cache 100 ;  
CREATE SEQUENCE
```

Session 1

```
postgres=# select nextval('mytest_seq');  
nextval  
-----  
1  
(1 row)
```

Session 2

```
postgres=# select nextval('mytest_seq');  
nextval  
-----  
101  
(1 row)
```

9. Exceptions

Many Oracle procedures use exceptions as part of standard practice

Most migration tools simply translate the code to PL/pgSQL

Not all Oracle exceptions are PostgreSQL exceptions

- Not found and too many rows are not PL/pgSQL exceptions
- Use STRICT to get Oracle like behavior

PostgreSQL uses sub transactions to handle exceptions

- Remove unnecessary exception blocks

```
CREATE FUNCTION get_first_name(p
  AS $$
DECLARE
  l_fname varchar;
BEGIN
  SELECT fname
  INTO STRICT l_fname
  FROM people
  WHERE lname = p_lname;

  RETURN l_fname;
EXCEPTION
  WHEN no_data_found THEN
```

10. Design guidelines

Declarative partitioning

- Use `pg_partman` & `pg_cron` for managing partitions automatically
- Keep Number of partitions to <100 per table.

Materialized views

- `FAST REFRESH` not available
- Use `pg_cron` for automatic refresh

Temporary tables

Reduce Foreign Key constraints

Sub transactions

- Consider turning off from ORM tools. For e.g. JDBC autosave -> never

`pg_prewarm`

- Use extension to cache blocks

Design guidelines - Contd

JDBC

- <https://jdbc.postgresql.org/>
- prepareThreshold
- hostRecheckSeconds
- setFetchSize
- executeBatch()
- autosave (default: never)

How AWS can help



AWS Database Freedom

Programs



Database Freedom **reduces the risk and cost** of migrations via technical workshops, POCs, Pilots, and trained Partners

Experts



Extend your talent with AWS Solutions Architects, Professional Services, System Integrators, Partner Services, and Training & Certification for your teams

Speed your migration by leveraging **proven practices and guidance**

Innovation



Innovative migration tools such as AWS Database Migration Service (**DMS**) and Schema Conversion Tool (**SCT**), with high automation to **reduce manual effort**

Amazon Database Migration Accelerator

Fixed-price, risk-mitigated way to convert legacy databases



AWS Conversion experts

Rely on AWS experts who have experience migrating countless workloads



Fixed, competitive price

Know exactly how much it will cost to migrate



Re-factored database & application

Leverage modern feature-rich databases



Speed

Reduce conversion time and time to value

AWS database migration partners



Resources

Database migration blogs

[Categorizing and Prioritizing a Large-Scale Move to an Open Source Database](#)

[How to Migrate Your Oracle Database to PostgreSQL](#)

[Configuring the AWS Schema Conversion Tool](#)

[Debugging Your AWS DMS Migrations: What to Do When Things Go Wrong \(Three Parts\)](#)

SCT & DMS additional resources

[Best practices for AWS SCT](#)

[Creating a multiserver assessment report for database migration](#)

[Saving the assessment report](#)

[AWS Database Migration Service Documentation](#)

[Using an Oracle database as a source for AWS DMS](#)

[Oracle diagnostic support scripts](#)

[Table and collection settings rules and operations](#)

[Oracle Database Migration Playbook for Oracle 19c](#)

[Oracle Database Migration Playbook for Oracle 11g/12c](#)

Database blogs

[AWS Database Blog](#)

[Amazon Aurora Blog](#)

[RDS PostgreSQL Blog](#)

Workshops

[Amazon RDS for PostgreSQL](#)

[Move to Aurora](#)



Thank you!

Krishna Sarabu

Sr Database
Specialist SA

Joseph DiCaro

Sr Specialist SA