Microsoft

Oracle to Azure Database for PostgreSQL Migration Cookbook

Prepared by

Paula Berenguel

pabereng@microsoft.com

Data Migration Jumpstart Engineering Architect

Microsoft Enterprise Services

Dec-2018

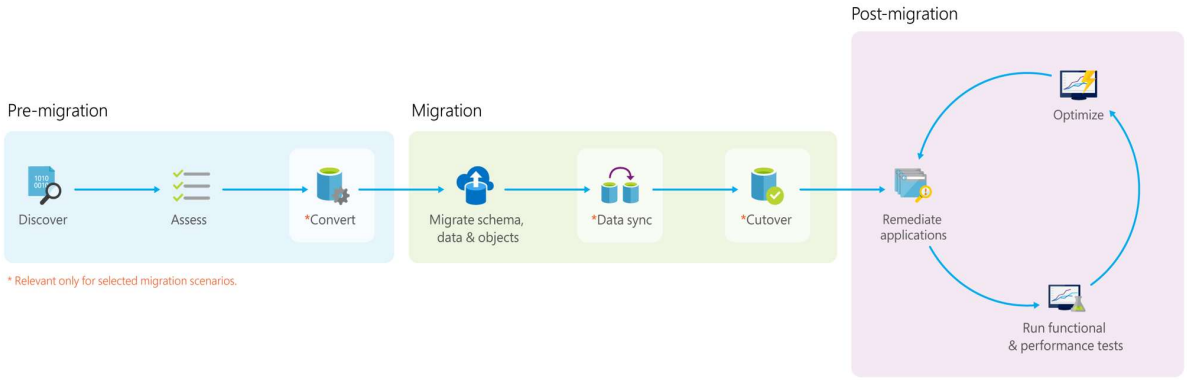
Table of Contents

Introduction

This document purpose is to provide Architects, Consultants, DBAs and related roles with a guide for quick migrating workloads from Oracle to Azure Database for PostgreSQL using ora2pg tool.

1 High Level Migration Approach

Replacing an Oracle database backend with an Azure Database for PostgreSQL backend is a simple exercise that can be executed using the following steps:



Discover: Customers discover all the databases they own, instances hosting them, source database type and version, other database metadata, etc.

Assess: In this phase, the goal is to identify which Azure database target a customer can migrate their database to and how much work is involved.

1. Target and SKU recommendation – Identifies which Azure database target and SKU is right for a database.
2. Readiness assessment – Compares the features used on the source vs the target and give an overview of work involved in migrating a database to Azure.
3. Performance assessment – Compares the query correctness and performance of a database on premise and in Azure.

Convert: Convert the schema from the source database type to target database type. This phase is valid only for heterogenous migrations.

Example: Oracle to SQL Server migrations.

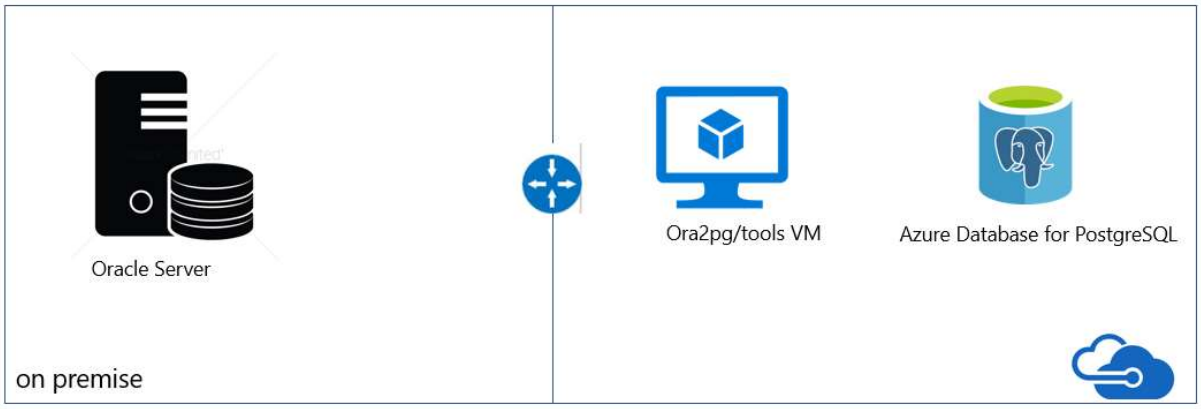
Migrate: Migrating a database involves migrating schema, data and objects usually in that order. There are 2 types of database migration methods:

1. Offline migrations – In this method, customers plan a downtime for the database, mark it read-only, perform the migration and switch over their applications to the new database.

2. Online migrations – Also known as minimal downtime, this method requires a short or no downtime for migrations. This method involves doing a front load of data to the target, enabling data sync so that data continues to be replicated from source to target until customer is ready to cut-over.

Validate: After the migration is complete, it is important to ensure that target database has the same schema, data and objects as source database. Also, it is important to validate if applications using the database behaves the same way on both source and target databases.

2 Typical ora2pg Migration Architecture



After provisioning the VM and Azure Database for PostgreSQL, two configurations are needed for enabling connectivity between them: "Allow Azure Services" and "Enforce SSL Connection", depicted as follow:

"Connection Security" blade -> Allow access to Azure Services -> ON
"Connection Security" blade -> SSL Settings -> Enforce SSL Connection -> DISABLED

3 Steps to Migrate from Oracle to Azure Database for PostgreSQL

3.1 Pre-Migration: Discovery

Microsoft Oracle pre-assessment scripts run against the Oracle database. The Pre-assessment script are a set of queries that hits the Oracle metadata and provides the following:

- Inventory of the database: counts of objects by schema, type and status
- Gives a rough estimate of Raw Data in each schema – this is based on statistics
- Provides sizing of tables in each schema
- Provides how many lines of code per Package, Function, Procedure, etc.

Scripts available here.

3.2 Pre-Migration: Assessment

After understanding the inventory of the Oracle database and having an idea of the database size and what the challenges are, the next step is to run the Assessment.

Estimating the cost of a migration process from Oracle to PostgreSQL is not easy. To obtain a good assessment of this migration cost, Ora2Pg will inspect all database objects, all functions and stored procedures to detect if there's still some objects and PL/SQL code that cannot be automatically converted by Ora2Pg.

Ora2Pg has a content analysis mode that inspect the Oracle database to generate a text report on what the Oracle database contains and what cannot be exported.

To activate the "analysis and report" mode, use the export de type SHOW_REPORT like in the following command:

```
ora2pg -t SHOW_REPORT
```

Once the database can be analysed, Ora2Pg, by his ability to convert SQL and PL/SQL code from Oracle syntax to PostgreSQL, can go further by estimating the code difficulties and estimate the time necessary to operate a full database migration.

To estimate the migration cost in man-days, Ora2Pg allow you to use a configuration directive called ESTIMATE_COST that you can also enabled at command line:

```
ora2pg -t SHOW_REPORT --estimate_cost
```

The default migration unit represent around five minutes for a PostgreSQL expert. If this is your first migration you can get it higher with the configuration directive COST_UNIT_VALUE or the --cost_unit_value command line option:

The last line of the report shows the total estimated migration code in man-days following the number of migration units estimated for each object. This migration unit represent around five minutes for a PostgreSQL expert. If this is your first migration you can get it higher with the configuration directive COST_UNIT_VALUE or the --cost_unit_value command line option. Find below some variations of assessment a) tables assessment; b) columns assessment c) schema assessment using default cost_unit (5 min) d) schema assessment using 10 min as cost unit

```
ora2pg -t SHOW_TABLE -c c:\ora2pg\ora2pg_hr.conf > c:\ts303\hr_migration\reports\tables.txt

ora2pg -t SHOW_COLUMN -c c:\ora2pg\ora2pg_hr.conf > c:\ts303\hr_migration\reports\columns.txt

ora2pg -t SHOW_REPORT -c c:\ora2pg\ora2pg_hr.conf --dump_as_html --estimate_cost > c:\ts303\hr
_migration\reports\report.html

ora2pg -t SHOW_REPORT -c c:\ora2pg\ora2pg_hr.conf –-cost_unit_value 10 --dump_as_html --estima
te_cost > c:\ts303\hr_migration\reports\report2.html
```

The output of the schema assessment is illustrated as below:

Migration level: B-5

```
  Migration levels:

       A - Migration that might be run automatically

       B - Migration with code rewrite and a human-days cost up to 5 days

       C - Migration with code rewrite and a human-days cost above 5 days

    Technical levels:

       1 = trivial: no stored functions and no triggers

       2 = easy: no stored functions but with triggers, no manual rewriting

       3 = simple: stored functions and/or triggers, no manual rewriting

       4 = manual: no stored functions but with triggers or views with code rewriting

       5 = difficult: stored functions and/or triggers with code rewriting
```

The assessment consists in a letter A or B to specify whether the migration needs manual rewriting or not, and a number from 1 to 5 to indicate the technical difficulty level. You have an additional option --human_days_limit to specify the number of human-days limit where the migration level should be set to C to indicate that it need a huge amount of work and a full project management with migration support. Default is 10 human-days. You can use the configuration directive HUMAN_DAYS_LIMIT to change this default value permanently.

This feature has been developed to help deciding which database could be migrated first and what is the team that need be mobilized.

3.3 Pre-Migration: Convert

In this step of the migration, the conversion or translation of the Oracle Code + DDLS to PostgreSQL occurs. The ora2pg tool exports the Oracle objects in a PostgreSQL format automatically. For those objects generated, some won't compile in the PostgreSQL database without manual changes.

The process of understanding which elements need manual intervention consists in compiling the files generated by ora2pg against the PostgreSQL database, checking the log and making the necessary changes until all the schema structure is compatible with PostgreSQL syntax.

a) First, the creation of migration template is recommended. This come out of the box with ora2pg. The two options --project_base and --init_project when used indicate to ora2pg that he has to create a project template with a work tree, a configuration file and a script to export all objects from the Oracle database. For further information on that, consult ora2pg documentation.
The command is executed as following:

```
ora2pg --project_base /app/migration/ --init_project test_project
```

```
ora2pg --project_base /app/migration/ --init_project test_project
        Creating project test_project.
        /app/migration/test_project/
                schema/
                        dblinks/
                        directories/
                        functions/
                        grants/
                        mviews/
                        packages/
                        partitions/
                        procedures/
                        sequences/
                        synonyms/
                        tables/
                        tablespaces/
                        triggers/
                        types/
```

```
                views/
        sources/
                functions/
                mviews/
                packages/
                partitions/
                procedures/
                triggers/
                types/
                views/
        data/
        config/
        reports/

Generating generic configuration file
Creating script export_schema.sh to automate all exports.
Creating script import_all.sh to automate all imports.
```

The sources/ directory contains the Oracle code, the schema/ will contains the code ported to PostgreSQL. The reports/ directory contains the html reports with the migration cost assessment.

After the project structure is created, a generic config file is created. Define the Oracle database connection as well as the relevant config parameters in the config. Refer to ora2pg documentation to understand what can be configured in the config file and how.

b) Exporting the Oracle objects as PostgreSQL objects is done by running the file export_schema.sh

```
cd /app/migration/mig_project
./export_schema.sh
```

Manual run:

```
SET namespace="/app/migration/mig_project"

ora2pg    -t    DBLINK    -p    -o    dblink.sql    -b    %namespace%/schema/dblinks    -c
%namespace%/config/ora2pg.conf
ora2pg   -t   DIRECTORY   -p   -o   directory.sql   -b   %namespace%/schema/directories   -c
%namespace%/config/ora2pg.conf
ora2pg   -p   -t   FUNCTION   -o   functions2.sql   -b   %namespace%/schema/functions   -c
%namespace%/config/ora2pg.conf
ora2pg -t GRANT -o grants.sql -b %namespace%/schema/grants -c %namespace%/config/ora2pg.conf
ora2pg -t MVIEW -o mview.sql -b %namespace%/schema/mviews -c %namespace%/config/ora2pg.conf
```

```
ora2pg -p -t PACKAGE -o packages.sql -b %namespace%/schema/packages -c
%namespace%/config/ora2pg.conf
ora2pg -p -t PARTITION -o partitions.sql -b %namespace%/schema/partitions -c
%namespace%/config/ora2pg.conf
ora2pg -p -t PROCEDURE -o procs.sql -b %namespace%/schema/procedures -c
%namespace%/config/ora2pg.conf
ora2pg -t SEQUENCE -o sequences.sql -b %namespace%/schema/sequences -c
%namespace%/config/ora2pg.conf
ora2pg -p -t SYNONYM -o synonym.sql -b %namespace%/schema/synonyms -c
%namespace%/config/ora2pg.conf
ora2pg -t TABLE -o table.sql -b %namespace%/schema/tables -c %namespace%/config/ora2pg.conf
ora2pg -t TABLESPACE -o tablespaces.sql -b %namespace%/schema/tablespaces -c
%namespace%/config/ora2pg.conf
ora2pg -p -t TRIGGER -o triggers.sql -b %namespace%/schema/triggers -c
%namespace%/config/ora2pg.conf
ora2pg -p -t TYPE -o types.sql -b %namespace%/schema/types -c %namespace%/config/ora2pg.conf
ora2pg -p -t VIEW -o views.sql -b %namespace%/schema/views -c %namespace%/config/ora2pg.conf
```

To extract the data, use the following command:

```
ora2pg -t COPY -o data.sql -b %namespace/data -c %namespace/config/ora2pg.conf
```

c) Lastly, compile all files against Azure Database for PostgreSQL server. It is possible now to choose to load the DDL files generated manually or use the second script import_all.sh to import those files interactively.

```
psql -f %namespace%\schema\sequences\sequence.sql -h server1-
server.postgres.database.azure.com -p 5432 -U username@server1-server -d database -l
%namespace%\ schema\sequences\create_sequences.log
psql -f %namespace%\schema\tables\table.sql -h server1-server.postgres.database.azure.com -
p 5432 -U username@server1-server -d database -l %namespace%\schema\tables\create_table.log
```

and etc..

Data import command:

```
psql -f %namespace%\data\table1.sql -h server1-server.postgres.database.azure.com -p 5432 -U
username@server1-server -d database -l %namespace%\data\table1.log

psql -f %namespace%\data\table2.sql -h server1-server.postgres.database.azure.com -p 5432 -U
username@server1-server -d database -l %namespace%\data\table2.log
```

During the compilation of files, check the logs and correct the necessary syntaxes that ora2pg was unable to convert out of the box.
Refer to this guide for support on working around issues (Link)

3.4 Migration: Schema & Data

After the fixes are in place, a stable build of the database is ready for deployment.

At this point, all that is required is to execute the *psql* import commands, pointing to the files containing the modified code in order to compile the database objects against the PostgreSQL database and import the data.

In this step, some level of parallelism on importing the data can be implemented.

3.5 Migration: Data Sync

As of Dec 2018, if near zero downtime migration is required, Attunity Replicate for Microsoft Migrations or Striim should be considered.

3.6 Migration: Cutover

As of Dec 2018, if near zero downtime migration is required, Attunity Replicate for Microsoft Migrations or Striim should be considered.

For "delta/incremental" migration using ora2pg: the technique consists in applying for each table a query that applies a filter (cut) by date or time, etc; and, after that finalize the migration applying a second query which will migrate the rest of the data (leftover).

In the source data table, migrate all the historical data first. An example of that is:

```
select * from table1 where filter_data < 01/01/2019
```

During the migration window, an example command is like the following:

```
select * from table1 where filter_data >= 01/01/2019
```

In this case it is recommended that the validation is enhanced by checking data parity on both sides, source and target.

3.7 Post-Migration: Perform Tests

In order to guarantee source and target are properly migrated the manual data validation scripts are run against the Oracle source and PostgreSQL target databases.

Ideally, if source and target databases have a networking path, ora2pg can should be used for data validation. Using the type of action called TEST allows to check that all objects from Oracle database have been created under PostgreSQL. The command run as following:

```
ora2pg -t TEST -c config/ora2pg.conf > migration_diff.txt
```

4 Migration Recommendations

- To improve the performance of the assessment or export operations in the Oracle server, collect statistics as following:

```
BEGIN
        DBMS_STATS.GATHER_SCHEMA_STATS
        DBMS_STATS.GATHER_DATABASE_STATS
        DBMS_STATS.GATHER_DICTIONARY_STATS
        END;
```

- Export data using COPY command instead of INSERT
- Avoid exporting tables with their FKs, constraints and indexes – it will make the data import into PostgreSQL slower
- Create materialized views using the “no data clause” and refresh it later
- If possible, implement unique indexes in materialized views, this will make the refresh faster with the syntax “REFRESH MATERIALIZED VIEW CONCURRENTLY”

5 References

https://docs.microsoft.com/en-gb/azure/postgresql/

https://ora2pg.darold.net/documentation.html

https://datamigration.microsoft.com/

https://www.postgresql.org/

https://www.striim.com/

https://www.attunity.com/

http://blog.dalibo.com/2016/08/19/Autonoumous_transactions_support_in_PostgreSQL.html