Amazon’s Database Migration Service, a magical wand for moving from closed source solutions

Dimitri Vanoverbeke
Solution Engineer
@ Percona
Who Am I

• I am your innocent Percona Solution Engineer.
• Living in Ghent, Belgium
• Works together with Sales, consulting, any departement.
• @dim0 on twitter (personal)
What is the exact point?

- Migrating towards a new infrastructure can be cumbersome
- A “silver bullet” to migrate Oracle, Microsoft SQL, Postgres to MySQL or another solution?
  - Easy migrations not requiring large downtime
  - Easy schema migrations using the Amazon Schema migration tool.
  - Performs automated feasibility assessment
- Supports a multitude of source database environments and a multitude of target databases.
What is a typical migration process

- Feasibility study
  - Can we migrate to required technology?
  - What are the limitations to the future technology?
    - What problems could we envision moving to the new technology.
  - What is really technology specific?
    - Stored procedures
    - Specific plugins
- Proof of Concept setup
  - Migration of the data
  - Setup infrastructure
- Benchmarking testing
- Application testing
- Migration planning
- Migration
- …
So what tools does Amazon provide

- Schema migration tool (Amazon SCT)
  - Windows, Ubuntu, MacOS package
  - Java tool
- Amazon Database migration services
  - AWS hosted environment providing replication to a multitude of technologies
- AWS DBaaS ecospace
  - Amazon RDS
  - Amazon RDS Aurora
- Amazon EC2 with your own choice of database
Schema Migration Tool interesting features

• Schema assessment report (per schema)
  • Verifies the schema for any problematic behavior:
  • Assesses the amount of manual actions for moving the schema to the new environment.
  • Verifies functions and checks if it can be migrated to the new environment.
• Creates separation in the following categories:
  • < 1 hour, < 4 hours, > 4 hours
  • A list of manual actions (or better proposed manual actions)
• Converts schema’s and provides you with the table create statements.
• Can verify application code for embedded SQL statements.
Schema migration Tool

- Two components
  - Java tool (eek)
  - Database driver
    - Multiple connectors supported (needs to be Java connector)
- Supports a lot of Technologies, even data warehousing solutions for migration to Redshift.
- Typically used for large Databases
Highlevel Best Practices / Limitations

- JVM based
  - Options for Performance
    - Balance speed with Memory consumption
    - Modify the amount of Memory
    - Run it on Linux (Mac OS X implementation seems broken)
- Use it to convert schema’s or copy database schema’s into the new infrastructure.
- You can still do it manual if it’s the same engine environment (using xtrabackup or mysqldump)
So what is Amazon Database migration services

- Replication instance
- Sources
- Targets
- Events and notifications
Replication instance

• Replication instance that replicates the activities in the transaction log of the source database, transforms and pushes the changes to the database backend.
• Multi AZ option (to make sure the replication environment is not a SPOF)
Replication instance
Sources

- On premise sources
- EC2 instances
- Amazon RDS instances
- Versions:
  - Oracle 10.2 and later
  - Microsoft SQL server
  - PostgreSQL
  - MongoDB 2.6>
  - Amazon RDS (Aurora, MySQL, …)
  - MySQL 5.5 - 5.7
  - MariaDB
Targets

• On premise
• AWS EC2 instance
• Amazon RDS instance
• Versions:
  • Oracle > 10g
  • Microsoft SQL server
  • MySQL 5.5 - 5.7
  • Postgres
  • Amazon RDS MySQL, Oracle, Aurora, MariaDB
Tasks

- Tasks are the actual workers for the migration.
  - Define the schema's or tables to migrate
  - It’s like a trigger for the actual tasks done on the replication instance level.
- Migration tasks
  - Full load of data to target database
  - Perform cached changes during load
  - Migrate + Migrate changes on the source database
  - Only migrate changes
  - Table mapping tasks (Filter, transform)
Events and notifications

Based on tasks
Can be determined events on the replication instance:
BEGIN_SCALE_STORAGE – The storage for the replication instance is being increased.
REPLICATION_INSTANCE_FAILURE – The replication instance has gone into storage failure.
So how do I start?

• Create a replication server. (there’s a button)
• Create source and target endpoints that have connection information about your data stores.
• Create one or more tasks to migrate data between the source and target data stores.
Create a replication instance

Replication instance created successfully.

Source database connection details:
- Endpoint identifier: PostId:dev
- Source engine: mysql
- Server name: blah.oauth2.mysqloev-central1.rds.amazonaws.com
- Port: 3306
- SSL mode: none
- Username: root
- Password: [hidden]

Target database connection details:
- Endpoint identifier: devtest
- Target engine: postgres
- Server name: mohp.oauth2.mysqloev-central1.rds.amazonaws.com
- Port: 5432
- SSL mode: none
- Username: un
- Password: [hidden]
- Database name: informcost

Advanced:
- Test connection
  - Connection tested successfully
Create tasks

first replication task

Create task

A task can contain one or more table mappings which define what data is moved from the source to the target. If a table does not exist on the target, it can be created automatically.

Task name: ProdEndpoint TestEndpoint
Task description: migrate the data
Source endpoint: sourceendpoint
Target endpoint: targetendpoint
Replication instance: replication-instance-1
Migration type: Migrate existing data and replicate ongoing changes

Your source database is MySQL. Replicating ongoing changes requires the MySQL binary log to be enabled and set to rew.

Please ensure your binary logs are retained on the server for a sufficient amount of time (24 hours is usually enough). To set your binary log retention time on RDS instances, you can use the following command: `cat /etc/my.cnf | grep binary-log-time`. For non-RDS instances, you can use: `cat /etc/mysql/my.cnf | grep binary-log-time`.

Task Settings

- Target table preparation mode:
  - Do nothing
  - Drop tables on target
  - Transliterate
Create a tablehopping

Table mappings

- Guided
- JSON

Selection rules

where schema name is like 'inform' and table name is like '%%', include

- add selection rule

Transformation rules

For schema where schema name is like 'inform', add prefix 'meh'

- add transformation rule
Status information

Task name: prodendpoint-testendpoint
Status: Load complete, replication ongoing
Migration type: Full Load, Ongoing Replication
Replication Instance: replication-instance-1
Source endpoint: prodendpoint
Target endpoint: testendpoint
Mapping method:

```json
{ "rules": [{ "rule-type": "selection", "rule-id": "1", "rule-name": "1", "object-locator": { "schema-name": "inform", "table-name": "%" }, "rule-action": "include" }, { "rule-type": "transformation", "rule-id": "2", "rule-name": "2", "rule-target": { "schema": "Schema", "object-locator": { "schema-name": "inform", "rule-action": "add-prefix", "value": "meh" } } }]
```
How can Amazon database migration service help

- Feasibility study
  - Can we migrate to required technology?
  - What are the limitations to the future technology?
    - What problems could we envision moving to the new technology.
  - What is really technology specific?
    - Stored procedures
    - Specific plugins
- Proof of Concept setup
  - Migration of the data
  - Setup infrastructure
- Benchmarking testing
- Application testing
- Migration planning
- Migration
- …
Best Practices for replication instance

- Make sure you scale it enough, investigate how much caching is required for the data set
- Parallel import
  - Limit or increase the threads
- Having multiple tasks can put strain on the source database.
- Data migration services create full table scans on the source environment
- It does not migrate the schema!
- Activate task log
- Don’t do DDL’s on the source database (it will ignore most of them)
Best Practices for replication instance

- It does not migrate the schema!
- Activate task log
- Don’t do DDL’s on the source database (it will ignore most of them)
Best Practices for replication instance

- It doesn't create secondary indexes, non-primary key constraints.
- Foreign key constraints need to be disabled during the migration's "full load" and "cached change apply" phases.
- Avoid using Multi-AZ on the target during migration of the data.
- Perform the validation!!! EnableValidation = true
So how would it work?
What if your source is MySQL and target is RDS

MySQL → Replication Instance → RDS MySQL

OR JUST

MySQL → MySQL dump → RDS MySQL

MySQL → Replication → RDS MySQL
But also it can help you consolidating…
Stay aware

• Make sure you are aware what you are getting into.
• This tool is to facilitate migrations not make them a success
• Test your migration and document your findings
• Imply your development team, moving to another database engine might require training
• Understand that some parts are a blackbox
Conclusion

- It’s interesting to see that Amazon has invested so many resources in this tool.
- The database migration instances are a simple way to replicate to a new infrastructure.
- It apparently has a lot of traction.
- Try it if you are interested and test it if it works for your infrastructure. (https://github.com/aws-samples/aws-database-migration-samples)