SERVERLESS DATABASE MIGRATION WITH LIQUIBASE

Garvesh Gupta1st, Vikash Kumar 2nd
MCA-Storage and Cloud Technology, Assistant Professor
Master of Computer Applications
Jain (Deemed to be University), Bangalore, India

Abstract: A dramatic surge of interest in database migration has been noticed in the recent years. There has been an increase in adoption of cloud-based databases. Increasing number of companies are seeing the value in moving from traditional on-premise data stores to off-premise cloud services. This paper shows how we can migrate our existing database to serverless database or from one version to another with minimum or no downtime and store the schema changes in the log files using Liquibase version control tool. The system gives both the forward and backward mechanism of database change and high availability through replication.

IndexTerms **– database migration, cloud, version control, serverless, Liquibase**

I. INTRODUCTION

Cloud computing has emerged as the dominant enterprise computing paradigm. Cloud computing simply means using computational resources as a service through networks. Serverless computing has introduced cost-efficiencies, reduced overheads, and rapid increase in an application's speed and scalability. Enterprises are moving their data and database assets to the cloud. After upgrading existing servers to an entirely new system, the next step is to carry out a database migration in a stable environment ensuring low downtime, high security and minimal human intervention. Migrating databases to the cloud is one of the biggest IT challenges in decades.[1]

Database migration refers to the transferring of database from one platform to another or management of incremental, reversible changes to relational database schemas. Database migration is a complex process and it is very difficult without schema control tool. If the target database does not support some of the features, changes may need to be implemented by middleware software. Liquibase is the software tool which simplify this process in case of schema mismatch in target database. It is a database version tracker for managing the state changes of the database during development. Furthermore, a seamless migration prevents business downtime. The conventional migration poses significant challenges to an enterprise such as exceedingly long durations, disruption in business. This can result in non-alignment of IT strategy with corporate strategy. [2]

II. PROBLEMS OCCURRED IN DATABASE MIGRATION

- Datatypes incompatibility issue**.**
- Extended or unexpected downtime.
- Data corruption, missing data or data loss.
- We can't trace who made a database change and when, or what version of database schema we are running in an environment.
- Application performance issues.
- Heavy maintenance, operational and performance costs.
- Technical compatibility issues.

III. EXISTING WORK

The traditional model consists of a migration script which is formed by taking backup of source database schema which is then sent to target database where it is restored also there are various tools like MySQL Workbench that supports database migration but the drawback is there are chances of data loss.

Few disadvantages of existing system are:

- No record of schema changes.
- Cost to maintain servers as they run continuously even if they are not in use.
- There is always a probability of losing data.

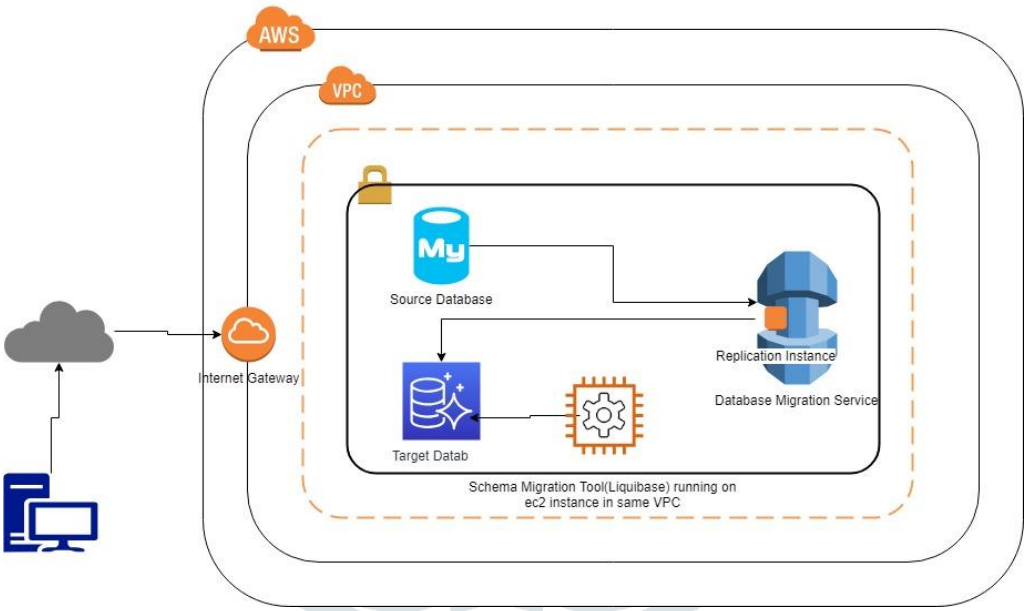
- The servers may go down in case of heavy load.

IV. PROPOSED WORK

Our proposed model is formed on improving database migration by minimizing the downtime, handling the data loss and maintaining the logs of database changes.

Few functionalities of the proposed system are:

- Charge you only for what you use.
- Require zero server maintenance.
- Provide continuous scaling.
- Transfer and conversion of the schema.
- Keeps track of what has run and who has done the changes.
- Automatically rollback the changes
- Movement of the actual data
- Support built-in high availability and fault tolerance.
- Reduce overall cost of the business.



V. Database Migration Tools

Migration tools are very well suited for the task of migrating data from one database to another. Liquibase is one such tool for tracking, managing and applying database changes which manages different revisions of an applications database schema – or more specifically, the changes required to migrate the database schema from one version to another. Liquibase also supports migrating your data across schema changes, in both forward and backward direction. The implementation of this database migration tool also allows us to achieve continuous delivery, which is vital for increasing speed, improving reliance, boosting productivity and comparing differences between schema. Liquibase supports JSON, XML, YAML, and SQL script. The tool will translate these definitions into the correct SQL for the concerned database. The changelog file is a document of every change made to the database and it's backed by the DATABASECHANGELOG table.

A. WORKING OF LIQUIBASE

In Liquibase a change is called a changeSet. Every changeSet tag is uniquely identified by the combination of the "id" tag, the "author" tag and the changelog file classpath. Liquibase executes the database Changelogfile where changeSet files are listed. It reads the changeSets in the order as the changes are mentioned in the changelog. Liquibase checks the databaseChangeLog table to see if the combination of id/author/filepath has been run. Liquibase will insert a new row with the id/author/filepath along with an md5 checksum of the changeSet in the databaseChangeLog table. To make any change in the database, always create a new changeSet. If an old executing file is changed, then the tool will fail to run and throws error. Liquibase offers a rollback feature, this gives us the control to roll back certain changes. You can tag our current state of

changes with a unique id execute your new changes and then call a rollback to the created tag. Changes are rolled back in the reverse order that they were run. Liquibase only tracks changes that have been applied to the database. When you roll back a change, it is no longer in the log.[3]

B. CHANGELOG FILE

The following code shows our simple database changelog file written in XML. It creates tables in the database. The noticeable thing is how change log uses a predefined XSD schema and how we are defining attributes for the changeset (id, author etc.). The actual change is contained within the <changeset></changeset> tags. The database for which this changeset is made is specified at runtime.

```
<?xml version="1.0" encoding="UTF-8"?>
<databaseChangeLog
xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog
http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-3.1.xsd">
  <changeSet author="A****n" id="6986">
    <sqlFile path="create.sql"/>
  </changeSet>
</databaseChangeLog>
```

C. RUNNING LIQUIBASE

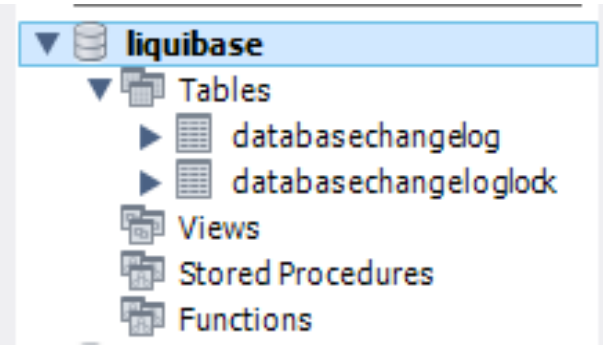
Liquibase is a command line tool, so we need to run it from the command prompt (Windows) or the terminal window (Linux / Mac OS X). The Liquibase command syntax is shown below:

liquibase <options> <command><command parameters>

There are a number of options and a number of commands available for Liquibase, which can be referred to on liquibase's website. We will run the update command which will apply the changeset against a target database. We have a MySQL database called "liquibase" in our test instance. Also, we created a server login called "root". This is the account Liquibase use to connect to the database. When the Liquibase runs for the first time these two tables are automatically generated by Liquibase.

```
C:\Program Files\liquibase-3.5.5-bin\project>liquibase --
driver=com.mysql.jdbc.Driver --changeLogFile=dbchangelog.xml --
url="jdbc:mysql://localhost:3306/liquibasedb?autoReconnect=true&amp
" --username=root --password=root@123 update
```

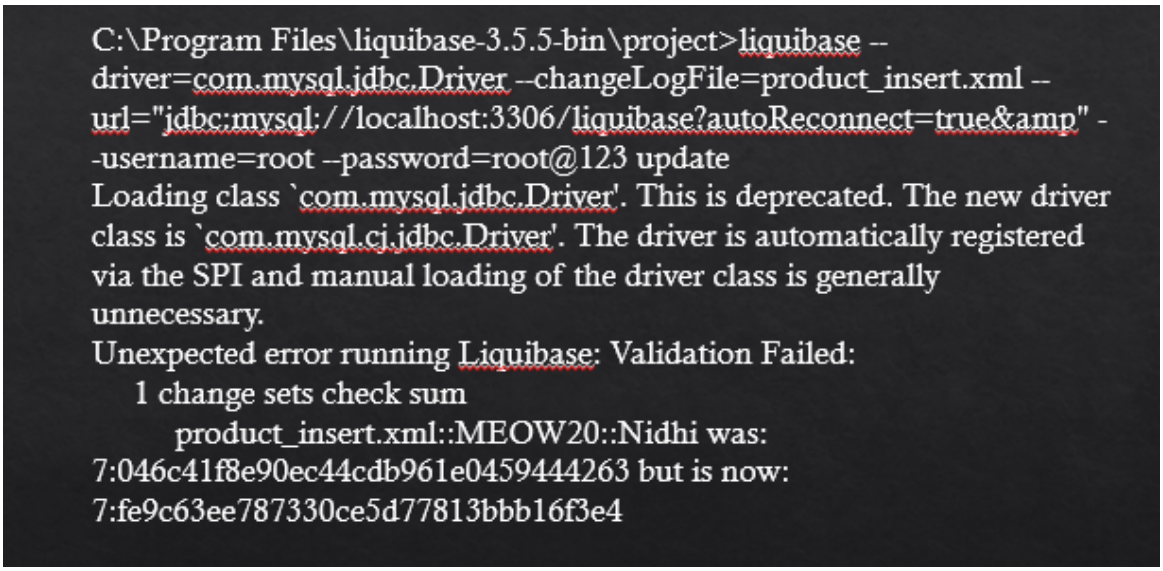
Liquibase commands and their parameters are case sensitive. The first part of the command or option starts in lower case and the latter parts are in uppercase. After execution of command we can check the database. There will be two tables: DATABASECHANGELOG and DATABASECHANGELOGLOCK table. These two tables are automatically generated by Liquibase the first time it runs and is updated in every subsequent run.



As we can see, the combination of ID, AUTHOR and FILENAME fields would uniquely identify each changeset also it shows the MD5 checksum of the changeset.

	ID	AUTHOR	FILENAME	DATEEXECUTED	ORDEREXECUTED	EXECTYPE	MD5SUM
▶	6986	Anirban	create.xml	2019-03-22 20:17:02	1	EXECUTED	7:9d3d7dd7b52859662f38a9bcbd4f6bd5

Liquibase compares the changeset's id, author and the changelog file name with what's stored in the DATABASECHANGELOG table and skips over the changeset and does not try to recreate the table. It checks the MD5 checksum if there is an entry in the table stored in the table against the MD5 checksum computed from the file. Liquibase knows the changeset has already been applied if the MD5 is same, so it does not run it again. Liquibase throws an error if the MD5 checksums are different and exits because it has no way of knowing what changes have been applied to the object.



And that's how Liquibase helps in maintaining a consistent database change chain. Every change has to be part of a different changeset: we cannot modify a changeset that has already been run.

D. DOCUMENTING DATABASE CHANGES

With dbDoc, Liquibase will generate a JavaDoc style html document from DATABASECHANGELOG table. This file will show when each database objects was created or changed, what changes were made to those components and who made those changes.[5]

```
liquibase --driver=com.mysql.jdbc.Driver --changeLogFile=db_doc.xml --
url="jdbc:mysql://target-db.cluster-[redacted]-
1.rds.amazonaws.com/liquibase?autoReconnect=true&amp" --
username=root –password=root#123 dbDoc "C:\Program
Files\liquibase-3.5.5-bin\project\document"
```

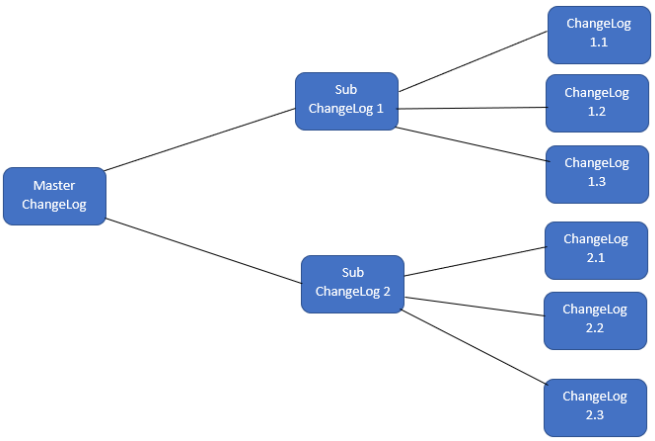
If we browse to the directory, we will see a number of files:

PC › DATA (D:) › Liquibase › dbDoc

Name	Date modified	Type	Size
authors	15/05/2016 8:35 PM	File folder	
changelogs	15/05/2016 8:36 PM	File folder	
columns	15/05/2016 8:35 PM	File folder	
pending	15/05/2016 8:35 PM	File folder	
recent	15/05/2016 8:35 PM	File folder	
tables	15/05/2016 8:35 PM	File folder	
authors.html	15/05/2016 8:35 PM	Chrome HTML Do...	1 KB
changelogs.html	15/05/2016 8:35 PM	Chrome HTML Do...	1 KB
currenttables.html	15/05/2016 8:35 PM	Chrome HTML Do...	1 KB
globalnav.html	15/05/2016 8:35 PM	Chrome HTML Do...	1 KB
index.html	15/05/2016 8:35 PM	Chrome HTML Do...	1 KB
overview-summary.html	15/05/2016 8:35 PM	Chrome HTML Do...	1 KB
stylesheet.css	15/05/2016 8:35 PM	CSS Document	2 KB

E. ORDERING CHANGELOG FILES

More changesets will be added to the changelog file throughout its life cycle. When we write our changelogs in XML, even simple changes can take multiple lines. Moreover, every time Liquibase runs the changelog, it has to work on every changeset from the beginning: something that will take longer and longer. Fortunately, we can have a single "master" changelog file which can include a number of "child" changelog files, which in turn can point to more changelog files and so on. This saves time and cost. The changelog files can have a relationship like this:



The following code block shows how we have refactored the original changelog file for our liquibase database as it can be seen, we have *included* reference to different XML files. Each changelog will run in order when liquibase starts running.

```
<?xml version="1.0" encoding="UTF-8"?>

<databaseChangeLog
xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog
http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-3.1.xsd">
 <preConditions onFail="HALT" onFailMessage="This changelog needs to run under the
liquibase user account">
  <runningAs username="root"/>
 </preConditions>
 <include file="create.xml"/>
 <include file="foreignkey.xml"/>
 <include file="department_insert.xml"/>
<include file="branch_insert.xml"/>
<include file="employee_insert.xml"/>
<include file="superior_insert.xml"/>
<include file="product_insert.xml"/>
<include file="cor_res_insert.xml"/>
<include file="account_insert.xml"/>
</databaseChangeLog>
```

To learn more about liquibase visit https://www.liquibase.org/index.html.

VI. SERVERLESS AURORA

Aurora is MySQL and PostgreSQL compatible database built for the cloud. It combines the performance and availability of high-end databases with the simplicity and cost-effectiveness of open source databases. It's fully-managed, amazingly fast (up to 5 times faster than MySQL), and can scale using shared-storage read replicas and multi availability zone deployments. This extra power comes with an added cost (23% more per hour), but it is well worth it.

Aurora Serverless provides an on demand, auto-scaling, high-availability relational database that only charges you when it's in use. After configuring the cluster all the maintenance, patching, backups, replication, and scaling are handled automatically for you. It can automatically pause when there is no load and resume automatically when requests come in which helps in significant cost savings, even if you maintained comparable capacity for 24 hours each day. The idea behind Aurora Serverless is to assume unpredictable workloads, only paying for occasional spikes in traffic would actually be significantly cheaper, if the application doesn't have extremely long periods of sustained load. [6]

To know how to create aurora serverless database visit https://aws.amazon.com/rds/aurora/serverless/

Connecting to Aurora Serverless Cluster

Any VPC resource that has access to the cluster (based on the chosen security groups) can connect to the cluster on port 3306. If you have an existing EC2 instance (with proper security group access and the CLI tools installed), you can connect from your terminal. If you want to connect to cluster from the local machine, either connect through a VPN, or set up an SSH tunnel.

Now that we have setup Aurora serverless database, it's time to migrate our on-premises/cloud database.

VII. DATABASE MIGRATION SERVICE

To work with AWS Database Migration Service, we setup and manage a replication instance on AWS. This instance is used to unload data from the source database and loads it into the destination database and can be used for a one-time migration followed by on-going replication to support a migration that involves minimal downtime. While migrating the database the source database remains fully functional hence minimizing downtime to applications that rely on the database. The service aims to minimize the span of database transfers, which can take months or years otherwise.

AWS Database Migration Service is simple to use. Once the migration has started, DMS manages all the complexities of the migration process including automatically replicating data changes that occur in the source database during the migration process. When the task gets started the database from on-premises/RDS MySQL will start migrating to AWS Aurora Serverless.

It is to be noted that one of the endpoints must always be in AWS, the other can be on-premises, running on an EC2 instance, or running on an RDS database instance.[8]

To know how to use database migration service visit https://aws.amazon.com/dms/.

VIII. CONCLUSION

Cloud environment is ever changing and scalable due to which unforeseen behavior can be seen leading to failures and faults. Serverless computing is a ground breaking development in the field of cloud computing that will help in achieving cost deduction for organizations. Due to the challenges IT leaders have historically avoided engaging in database migration projects. By proposing this model, we are trying to make our systems more fault tolerable, scalable, traceable, providing high availability and scalability and hence reducing the overall cost.

IX. FUTURE SCOPE

- We can use this database system when there are chances of sudden spikes in the database like sale on ecommerce platform.
- In future we can expect public accessibility outside the VPC so that we can access the database from on premises device.
- Compatibility with other databases, and
- Backup to s3 bucket also storing the logs in s3 bucket.

REFERENCES

[1] Lynn, T., Rosati, P., Lejeune, A., & Emeakaroha, V. (2017). A Preliminary Review of Enterprise Serverless Cloud Computing (Function-as-a-Service) Platforms.

[2]https://www.happiestminds.com/Insights/database-migration/

[3] Mahmoud Magdy (2017). How to Execute Database Migrations with Liquibase.

[4] Sadequl Hussain (2016) SQL Server Database Change Management with Liquibase.

[5] Sadequl Hussain (2016) SQL Server Database Change Management Use Cases with Liquibase - Part 3

[6] Jeremy Daly (2018) Aurora Serverless: The Good, the Bad and the Scalable.

[7] Jeff Barr (2016) AWS News Blog: AWS Database Migration Service.

[8] Pritam Agrawal (2019) Disaster Recovery (DR) using AWS Database Migration Services