**Case2 :::: Migration of 1TB of data. Major considerations for time and cost. Initially migration within the same database. Later we should consideration for different databases. Use PODMAN container for provisioning the databases**

1. Without any cloud service or say using local resources

Resources used PostgreSQL and PODMAN install these on the local system.

Migration within the same database::

Firstly backup the source database

“pg\_dump -U postgres mydatabase > backup.sql”

Now restore the snapshot to new database instance on a different server or VM

“psql -U postgres -d mydatabase -f backup.sql”

Chunk Data for migration

Estimate the number of chunks :: For a 1TB (1024GB) dataset, deciding on 10GB chunks will result in approximately 102 chunks

need to choose a column for chunking, typically an id or a timestamp column that has an even distribution of data. Calculate row ranges based on your chosen column

“SELECT COUNT(\*) FROM mytable;”

Suppose the table has 1,000,000 rows, we can calculate rows per chunk

“total\_rows=1000000

chunk\_size=$(( total\_rows / 102 )) ”  
\*\*\* Approximately 9803 rows per chunk

Exporting data in chunks::: Using a loop to automate the chunk export process

start=0

chunk\_size=9803

for (( i=0; i<102; i++ ))

do

end=$(( start + chunk\_size ))

pg\_dump -U postgres -t mytable --data-only --column-inserts --file=chunk\_$i.sql --where="id BETWEEN $start AND $end"

start=$(( end + 1 ))

done”

We can also automate this process using bash script ::

#!/bin/bash

DB\_USER="postgres"

DB\_NAME="mydatabase"

TABLE\_NAME="mytable"

TOTAL\_ROWS=$(psql -U $DB\_USER -d $DB\_NAME -t -c "SELECT COUNT(\*) FROM $TABLE\_NAME;")

CHUNK\_SIZE=$(( TOTAL\_ROWS / 102 ))

start=0

for (( i=0; i<102; i++ ))

do

end=$(( start + CHUNK\_SIZE ))

pg\_dump -U $DB\_USER -t $TABLE\_NAME --data-only --column-inserts --file=chunk\_$i.sql --where="id BETWEEN $start AND $end"

start=$(( end + 1 ))

done

Some additional tools to monitor the operations

Monitor Network Bandwidth and Latency

Tools ::: nload -- Real-time network traffic and bandwidth usage monitor  
 **iperf --** Network bandwidth measurement tool  
 **ping --** Measures round-trip time for messages sent from the originating host to a destination computer

Commands : : sudo apt-get install nload

nload

# On the server

iperf -s

# On the client

iperf -c <server\_ip>  
   
 ping <destination\_ip>

Monitor Database Load ::   
Tools ::: pg\_stat\_activity -- Provides information about the current activity of the server  
 **pg\_stat\_statements --**  Provides information about the most frequently executed queries

Commands :: SELECT \* FROM pg\_stat\_activity;

SELECT \* FROM pg\_stat\_statements;

Implement Parallel Processing :::

Tools ::: **GNU Parallel --**  Executes jobs in parallel using one or more computers  
 **Background Processes --** Run shell commands in the background

Commands ::: sudo apt-get install parallel

seq 1 10 | parallel -j 4 echo {}

mysqldump -u root -p --where="id BETWEEN 0 AND 9803" mydatabase mytable > chunk\_0.sql &

mysqldump -u root -p --where="id BETWEEN 9804 AND 19606" mydatabase mytable > chunk\_1.sql &

1. **Using AWS cloud solution**
2. Set Up AWS  
   Set up IAM roles and policies for managing database access
3. Install PODMAN on your local machine
4. Setting up database instance   
   provisioning RDS Instances  
   Note: Provision instances in the same VPC to minimize latency and costs
5. Initial Migration Within the Same Database  
   Backup the Source Database  
   Create a Snapshot  
   Restore the Snapshot to a New Instance

Migration to Different Databases:  
Set Up PODMAN Containers  
Create a PODMAN Network::   
 “ podman network create db-network “  
  
Provision PODMAN Containers for Databases (for mysql)  
 “ podman run --name mysql-container --network db-network -e MYSQL\_ROOT\_PASSWORD=root -d mysql “

Data Transfer Between Different Databases  
Export Data from Source Database  
 “ mysqldump -u root -p --all-databases > all-databases.sql “

Import Data to Target Database::   
 “ podman cp all-databases.sql mysql-container:/all-databases.sql

podman exec -it mysql-container mysql -u root -p < /all-databases.sql ”

Migrating 1TB of Data Using Chunking

1. Determine Chunk Size:: use same method as provided in step 1 i.e for local resouces
2. Export Data in Chunks(mysql)  
     
   mysqldump -u root -p --single-transaction --quick --lock-tables=false --where="id BETWEEN 1 AND 100000" mydatabase > chunk1.sql

mysqldump -u root -p --single-transaction --quick --lock-tables=false --where="id BETWEEN 100001 AND 200000" mydatabase > chunk2.sql

# Repeat for other chunks

Transfer and Import Data Chunks::

Create PODMAN Containers for the Target Database :::   
We're using MySQL as the target database. First, we'll create a PODMAN container running MySQL.  
Pull MySQL Image::   
 “ podman pull mysql:latest “  
  
Run MySQL Container ::   
 “ podman run --name mysql-container -e MYSQL\_ROOT\_PASSWORD=rootpassword -d mysql:latest ”  
  
Transfer Data Chunks to the Target Containers ::   
 We have already exported data chunks as SQL files (chunk\_0.sql, chunk\_1.sql, etc.), we'll now transfer these files into the PODMAN container

Copy Data Chunks to Container ::   
 podman cp chunk\_0.sql mysql-container:/chunk\_0.sql

podman cp chunk\_1.sql mysql-container:/chunk\_1.sql

Import Each Chunk into the Target Database ::   
We will now import the SQL data chunks into the MySQL database inside the PODMAN container

Access the MySQL Container ::   
 “ podman exec -it mysql-container bash ”  
  
Import Data Chunks :: Inside the container, run the following commands to import the data chunks  
  
 mysql -u root -prootpassword -e "CREATE DATABASE mydatabase;"

mysql -u root -prootpassword mydatabase < /chunk\_0.sql

mysql -u root -prootpassword mydatabase < /chunk\_1.sql

Optimize and Monitor the Migration ::   
 Compress SQL dumps to reduce transfer time ::   
 gzip chunk1.sql

gzip chunk2.sql

Use Parallel Processing to Speed Up Import/Export ::   
Using tools like GNU Parallel or custom scripts to run multiple export/import processes simultaneously

Export Data Chunks ::   
seq 1 10 | parallel -j 4 "mysqldump -u root -p'password' database table\_{} > chunk\_{}.sql"

Import Data Chunks ::   
seq 1 10 | parallel -j 4 "mysql -u root -p'password' new\_database < chunk\_{}.sql"

\*\* **seq 1 10** generates a sequence of numbers from 1 to 10

\*\* “**parallel -j 4”** runs 4 jobs simultaneously

Use AWS CloudWatch to Monitor RDS Performance

Check PODMAN Container Logs for Errors ::   
View Logs ::   
 podman logs mysql-container

Monitor in Real-Time ::   
 podman logs -f mysql-container

\*\* **podman logs mysql-container** shows the logs of the MySQL container  
\*\* **-f** option follows the logs in real-time

Verify Data Integrity by Running Checksums :::

Generate Checksum on Source Database :   
 mysqldump -u root -p'password' --skip-comments --quick database | md5sum  
  
Generate Checksum on Target Database :::   
 mysqldump -u root -p'password' --skip-comments --quick new\_database | md5sum  
  
\*\* **md5sum** generates a checksum of the database dump which can be compared between the source and target to verify integrity.

Flow ::

The flow is nearly same for both the processes in which DB can be change either SQL or PostgreSQL or RDS

Chunk 3

Chunk 2

Chunk 1

Local DB  
Target DB

Data Migration in Chunks

PODMAN  
PostgreSQL

Local DB  
Source DB

Initial migration