Migrating PostgreSQL and MySQL Databases to VMware Data Services Manager

Best Practices Document

Table of contents

Executive Summary

VMware Data Services Manager™ (DSM) offers a comprehensive solution for modern database and data services management within VMware Cloud Foundation environments. It functions as a powerful database-as-a-service (DBaaS) toolkit, enabling organizations to achieve on-demand provisioning and automated management of PostgreSQL, MySQL and other databases, all within a highly virtualized infrastructure.

This document specifically addresses best practices for a "brownfield migration", which involves migrating existing PostgreSQL and MySQL databases to new databases that are managed by VMware Data Services Manager. Furthermore, the document outlines a step-by-step approach for executing the migration, ensuring that the desired business requirements for performance, availability, and scalability are fully met in the new DSM-managed environment.

Business Benefits

Migration to VMware Data Services Manager will bring the following business benefits:

- **Faster provisioning for rich data services**: Data services templates that are pre-packaged and certified by VMware and partners enabling organizations to quickly spin up desired data platforms.
- **Automated operations:** Common routine tasks such as lifecycle management, resource handling, scaling, data protection and availability can all be automated with Data Services Manager.
- **Extensive monitoring:** Gain visibility into database and underlying resources, including metrics such as the utilization of CPU, memory, network, local, and cloud storage, as well as curated alarms and notifications for improved visibility into the health of both the database and the underlying infrastructure.
- **Ready business for Private AI:** Enable enterprise to customize models and run generative AI and RAG workloads with vector database capability to enable fast querying of data and enhanced outputs of different LLMs.
- **Seamless integration with the private cloud platform:** Empower IT organizations with enterprise-hardened data services that are deeply integrated with VMware Cloud Foundation.

Target Audience

This document assumes a knowledge and understanding of VMware Cloud Foundation, VMware Data Services Manager, PostgreSQL databases and MySQL databases. The document is designed for architectural and management staff, infrastructure and database administrators to plan and migrate their existing database infrastructure to VMware Data Services Manager. The goal is to help model the business requirements, and take advantage of management simplicities and operational efficiencies achieved with various components of VMware Cloud Foundation platform.

Technology Overview

VMware Cloud Foundation

VMware Cloud Foundation™ is strategically positioned to empower organizations in their pursuit of IT modernization, by addressing three pivotal areas: infrastructure modernization, providing a cloud experience for developers, and enhancing security and resilience. This comprehensive approach ensures that enterprises are well-equipped to face the digital challenges of today and tomorrow. Empowering agility within your organization to drive innovation.

VMware Cloud Foundation is built on trusted VMware technologies for compute, storage, networking, and cloud management. These technologies are the building blocks of a full software-defined data center, integrating all these capabilities into a single platform that is easy to consume.

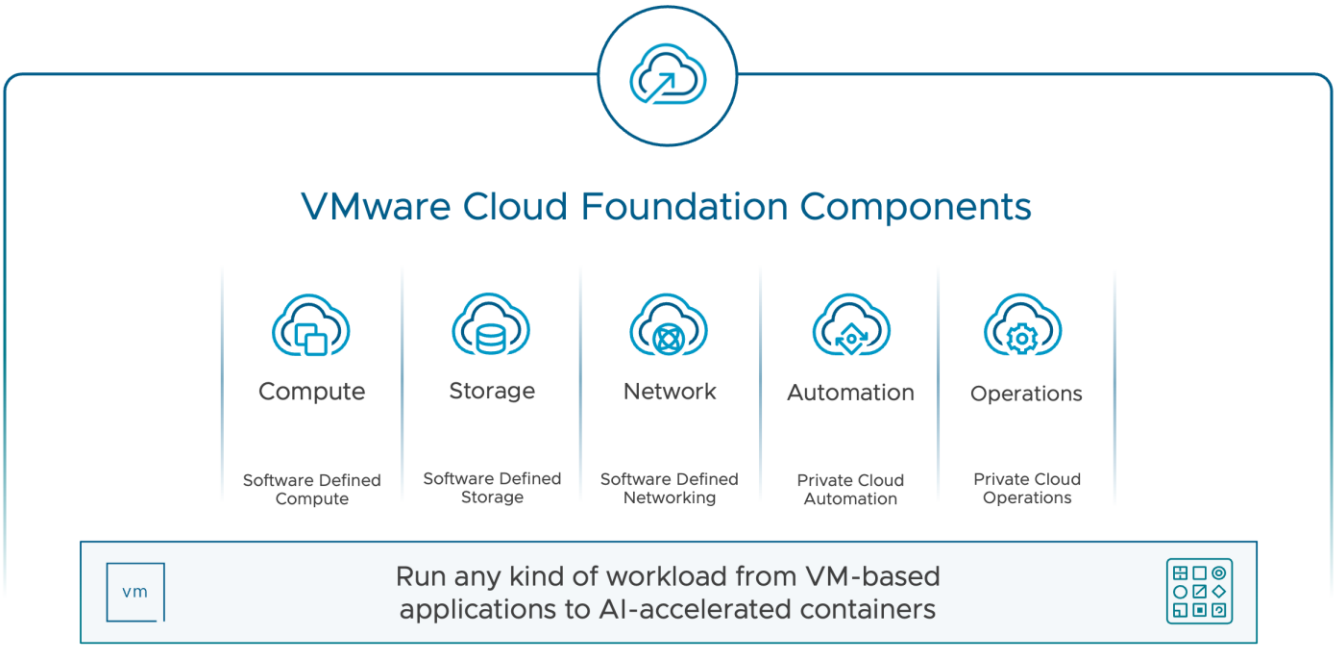
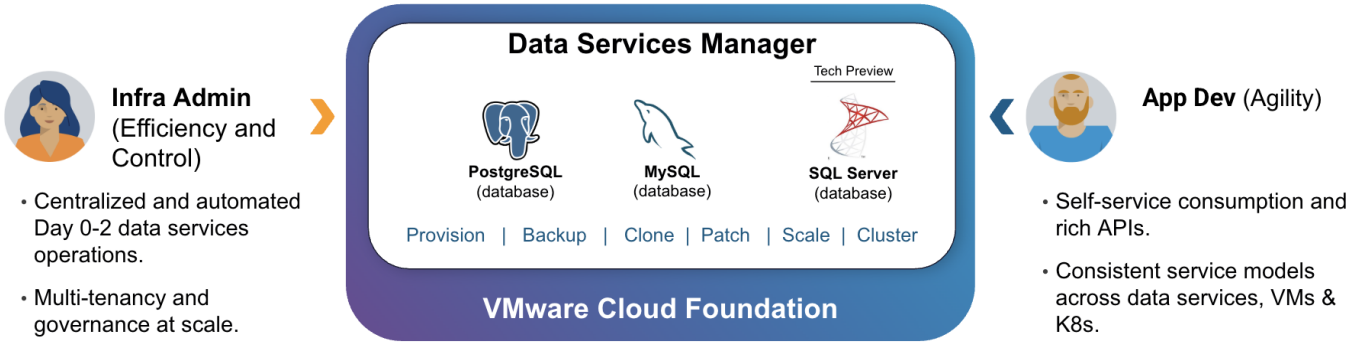


Figure 1 VMware Cloud Foundation Components

VMware Data Services Manager

VMware Data Services Manager is an advanced service for VMware Cloud Foundation, which allows a user to provision and manage at scale PostgreSQL, MySQL and Microsoft SQL Server (Tech Preview) relational databases. For Generative-AI applications, customers can also use Data Services Manager to deploy a pgvector-enabled PostgreSQL database, providing intelligence about storing and querying vector embeddings for Retrieval Augmented Generation.



PostgreSQL Migration Methodology

To migrate externally managed PostgreSQL databases and clusters to Data Services Manager, you can use the following methodology:

- Migration with PostgreSQL logical replication - replicating data objects and their changes, based upon their replication identity with a publish and subscribe model. This methodology can help minimize the business downtime during the migration process
- Migration with *pg_dump, pg_dumpall and pg_restore* utilities – migrate databases using traditional backup and restore. This methodology is for general purpose migration but requires certain business downtime during the migration process.
- Migrating legacy PostgreSQL versions into a supported version by using a tool such as Bucardo. This methodology can be considered when the source PostgreSQL is with older versions that do not natively support logical replication and the user wants to minimize the business downtime.

MySQL Migration Methodology

To migrate externally managed MySQL databases and clusters to Data Services Manager, you can use the following methodology:

- Migration with MySQL replication - migrate external managed MySQL databases to Data Services Manager using MySQL asynchronous replication.
- Migration with *mysqldump* utilities – migrate databases using traditional backup and restore. This methodology is for general purpose migration but requires certain business downtime during the migration process.

Before finalizing a migration methodology, it is recommended to try out the chosen methodology in a test-dev environment similar to the production setup prior to using it for production database migration.

Prepare Source database

Before you start migration, prepare the source PostgreSQL and MySQL database with the following considerations:

- Prepare a list of source databases under migration. Choose the appropriate migration methodology based on the database version, database size, and business SLAs etc.
- Compute resource consideration – collect the source database sizing in terms of CPU and memory and create VM class in Data Services Manager accordingly.
- Network consideration – verify the network of source databases is reachable to the target data services network configured in the Data Services Manager.
- Storage consideration – collect the source PostgreSQL or MySQL database storage sizing and performance requirements. If you have storage HA solutions implemented in the source cluster, make sure you have the same or equivalent solution in the target Data Services Manager environment. For example, if you run the target Data Services Manager on vSAN Express Storage Architecture (ESA), it provides FTT=1 by default with RAID-5 and FTT=2 with RAID-6 in storage policy settings.
- Database configuration – collect the configuration parameters on the source PostgreSQL or MySQL databases and you may apply those configurations on the target PostgreSQL or MySQL in Data Services Manager with the advanced database options. Refer to the supported PostgreSQL and MySQL parameters in Database Advanced Settings for databases managed by Data Services Manager.
- PostgreSQL extensions - collect the list of extensions on the source database and make sure all of them are included in the supported PostgreSQL extension of Data Services Manager.

Prepare Target database in Data Services Manager

You must prepare the target Data Services Manager before migrating the source PostgreSQL or MySQL databases.

Installation and Deployment

You deploy Data Services Manager as a plug-in with an .ova file available in the Broadcom Software download page. Refer to Deploying the VMware Data Services Manager plug-in in vSphere Client documentation for detailed steps.

Configure Infrastructure Policy

The infrastructure Policy allows VI admins to define which vSphere infrastructure resources should be used for the placement of data services provisioned by Data Services Manager. It includes the following components.

- Compute resources – a vSphere cluster or resource pool for provisioning the data services.
- Network port groups – networks that a data services virtual machine can be attached to.
- IP Pools – a collection of available IP addresses used for dynamic IP assignment for data services. Based on the number of virtual machines and deployment type (single instance or cluster), assign enough IP addresses for the IP Pools.
- VM classes – specify the vCPU and Memory for a data services virtual machine. Based on the source PostgreSQL instance sizing, create corresponding VM classes in Data Services Manager.
- Storage Policies – specify the datastore placement of a data services virtual machine. For vSAN ESA as an example for target storage, an ESA default storage policy with RAID 5 or RAID 6 is recommended.

Configure object storage

Data Services Manager requires an S3-compatible object storage solution for the following buckets.

- A bucket for storing support bundles
- A bucket for Provider backups
- A bucket for database backups

It is required to configure the desired object storage with network encryption (TLS). Follow the Data Services Manager Certificate Requirements when generating a certificate for the object storage configured above. Use IP address for the Subject Alternative Name (SAN) and do not leave blank.

Provision PostgreSQL databases in Data Services Manager

Data Services Manager supports both single server and cluster deployment for PostgreSQL databases. You may select the desired deployment type based on your business SLAs.

Data Services Manager supports customized PostgreSQL parameters in Database Options. The default settings of parameters such as *shared_buffers* and *max_connections* for the target database provisioned by Data Services Manager can be overridden with ease according to the source database settings for different use cases. For more details, refer to PostgreSQL on Data Services Manager.

Based on the source PostgreSQL version, it is recommended to provision the same or newer version of PostgreSQL on the Data Services Manager deployment. Check the VMware Data Services Manager release notes for more details about supported PostgreSQL database versions.

PostgreSQL – Migration with Logical Replication

Logical Replication uses a publish and subscribe model, copying and replicating data objects and their changes based on their replication identity. It allows fine-grained control over data replication between different PostgreSQL databases. Logical Replication replicates individual tables and their changes including INSERTs, UPDATEs, DELETEs. The subscribers pull data from the publications and apply the data in the same order as the publisher so that transactional consistency is guaranteed. Logical Replication supports the following migration use cases.

- The source and target PostgreSQL with different major versions
- The source and target PostgreSQL with different platforms (Linux, Windows, etc.)

Migration Prerequisites

The logical replication replicates the Data Modification Language (DML) changes like INSERT, UPDATE, DELETE and TRUNCATE from the source PostgreSQL database to the target Data Services Manager PostgreSQL database. Before you migrate with logical replication, validate the following checklist

- Verify that you have the necessary CLI tools installed in your source environment including the *pg_dump* and *psql* utilities.
- Verify that you have configured the *pg_hba.conf* file to allow connection between the source environment and the target database environment provisioned by Data Services Manager.
- Verify that you have granted the corresponding replication privilege to the user account that will be used for the logical replication.
- Obtain a list of all the source databases needed to migrate. Logical replication is configured at the database level. For multiple databases, you need to configure logical replication for each of them.
- Understand the restrictions of logical replication before you start the migration process. Refer to Logical Replication Restrictions for more details.

Migration Steps

➢ Step 1 – Configure the source PostgreSQL database for logical replication. On the source PostgreSQL database and its replicas, set **wal_level** to **logical**.

Example:

```
sourcedb1=# show wal_level;
 wal_level
-----------
 replica
(1 row)

sourcedb1=# ALTER SYSTEM SET wal_level = logical;
ALTER SYSTEM
```

Restart PostgreSQL service and check **wal_level** and make sure it's changed to **logical**.

Example:

```
sourcedb1=# show wal_level;
 wal_level
-----------
 logical
(1 row)
```

➢ Step 2 – Extract Schema from source database. To perform this step, you must have access to the objects in the database.

Example:

```
pg_dump "host=source-cluster-host user=source-db-user password=source-db-passsword dbname=source-db-name sslmode=prefer"
--schema-only --clean --if-exists --no-owner -no-privilege --no-subscriptions --no-publications > /tmp/output.log
```

In this example, we populated the sample source database named "sourcedb1" using *pgbench* utility with scale factor set to 10000. The sample database size was about 146GB.

```
~$ pgbench -i -s 10000 sourcedb1
dropping old tables...
NOTICE:  table "pgbench_accounts" d0 oes not exist, skipping
NOTICE:  table "pgbench_branches" does not exist, skipping
NOTICE:  table "pgbench_history" does not exist, skipping
NOTICE:  table "pgbench_tellers" does not exist, skipping
creating tables...
generating data (client-side)...
1000000000 of 1000000000 tuples (100%) done (elapsed 941.42 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done in 1514.07 s (drop tables 0.00 s, create tables 0.00 s, client-side generate 946.15 s, vacuum 1.14 s, primary keys 566.78 s).
```

The sample output.log file may look like this:

```
~$ more /tmp/output.log
--
-- PostgreSQL database dump
--

-- Dumped from database version 16.8 (Ubuntu 16.8-0ubuntu0.24.04.1)
-- Dumped by pg_dump version 16.8 (Ubuntu 16.8-0ubuntu0.24.04.1)

SET statement_timeout = 0;
SET lock_timeout = 0;
SET idle_in_transaction_session_timeout = 0;
SET client_encoding = 'UTF8';
SET standard_conforming_strings = on;
SELECT pg_catalog.set_config('search_path', '', false);
SET check_function_bodies = false;
SET xmloption = content;
SET client_min_messages = warning;
SET row_security = off;

ALTER TABLE IF EXISTS ONLY public.pgbench_tellers DROP CONSTRAINT IF EXISTS pgbench_tellers_pkey;
ALTER TABLE IF EXISTS ONLY public.pgbench_branches DROP CONSTRAINT IF EXISTS pgbench_branches_pkey;
ALTER TABLE IF EXISTS ONLY public.pgbench_accounts DROP CONSTRAINT IF EXISTS pgbench_accounts_pkey;
DROP TABLE IF EXISTS public.pgbench_tellers;
DROP TABLE IF EXISTS public.pgbench_history;
DROP TABLE IF EXISTS public.pgbench_branches;
DROP TABLE IF EXISTS public.pgbench_accounts;
SET default_tablespace = '';
```

➢ Step 3 – Create the schema objects in the target database

Example:

```
psql connection_string_to_target_dsm_database < /tmp/output.log
```

Sample output:

```
~$ psql postgresql://pgadmin:Password@10.211.198.233:5432/dsmpg1 < /tmp/output.log
SET
SET
SET
SET
SET
 set_config
------------

(1 row)

SET
SET
SET
SET
ALTER TABLE
ALTER TABLE
ALTER TABLE
DROP TABLE
DROP TABLE
DROP TABLE
DROP TABLE
SET
SET
CREATE TABLE
ALTER TABLE
CREATE TABLE
ALTER TABLE
CREATE TABLE
ALTER TABLE
CREATE TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
```

➢ Step 4 – Create publication on the source database

Example:

```
create publication onboard_to_dsm for all tables;
```

Sample output:

```
sourcedb1=# create publication onboard_to_dsm for all tables;
CREATE PUBLICATION
```

➢ Step 5 – Create subscription on the target database (DSM-managed)

```
CREATE SUBSCRIPTION onboard_to_db_name CONNECTION 'dbname=source-db host=source-db-host port=port user=source-db-user password=source-db-password sslmode=prefer' PUBLICATION onboard_to_dsm WITH(create_slot = true, slot_name = onboard_to_dbname, enabled=true);
```

Sample output:

```
dsmpg1=# CREATE SUBSCRIPTION onboard_to_dsm_target CONNECTION 'dbname=sourcedb1 host=10.211.198.200 port=5432 user=postgres password=Password sslmode=prefer' PUBLICATION onboard_to_dsm WITH(create_slot=true, slot_name=slot1, enabled=true);
NOTICE:  created replication slot "slot1" on publisher
CREATE SUBSCRIPTION
```

➢ Step 6 – Monitor the logical replication status

Use the **pg_subscription_rel** catalog to view the logical replication status

```
select srsubid, class.relname, srsubstate from pg_subscription_rel join pg_class class on oid = srrelid;
```

Check the **srsubstate.** The sample output below with all "**r**" indicates the replication has been completed.

- i = initialize
- d = data is being copied
- f = finished table copy
- s = synchronized
- r = ready (normal replication)

```
dsmpg1=# select srsubid, class.relname, srsubstate from pg_subscription_rel join pg_class class on oid = srrelid;
srsubid |     relname      | srsubstate
---------+------------------+------------
   16457 | pgbench_accounts | r
   16457 | pgbench_branches | r
   16457 | pgbench_history  | r
   16457 | pgbench_tellers  | r
(4 rows)
```

You may also check the log sequence number (LSN) to make sure the logical replication is completed. On the source PostgreSQL database.

```
sourcedb1=# select pg_current_wal_lsn();
pg_current_wal_lsn
--------------------
25/3AF7E5E0
(1 row)
```

On the target Data Services Manager, check if the LSN matches the source PostgreSQL database.

```
dsmpg1=# select received_lsn, latest_end_lsn from pg_catalog.pg_stat_subscription;
received_lsn | latest_end_lsn
--------------+---------------
25/3AF7E5E0  | 25/3AF7E5E0
(1 row)
```

- Step 7 – Confirm the necessary data is replicated on the target DSM database
- Step 8 – Switch your application to start using the PostgreSQL database provisioned by Data Services Manager and drop the publication and subscription

  On the target DSM database:

```
dsmpg1=# drop subscription if exists onboard_to_dsm_target;
NOTICE:  dropped replication slot "slot1" on publisher
DROP SUBSCRIPTION
```

  On the source database:

```
sourcedb1=# drop publication if exists onboard_to_dsm;
DROP PUBLICATION
```

Tested Configuration

Once you have completed all the migration steps, you may configure your applications to start using the PostgreSQL databases provisioned by VMware Data Services Manager.

We measured the migration result of the following test scenario. Both source and target PostgreSQL databases are configured with 2 vCPU and 8GB memory. Note the estimated migration duration is subject to change due to the hardware configuration, network bandwidth, resource allocation and other factors and should be only used for reference purposes.

- In the first scenario, we applied no workload during the migration process. This is a recommended way to migrate with PostgreSQL Logical Replication unless your production workload can not stop due to business requirements. The estimated time for 146GB data migration is about 50 minutes.
- In the second scenario, we applied workload on the source database after the first scenario was completed. The Logical Replication will continue to replicate the transaction workloads on the source database to the target database in Data Services Manager. The estimated migration duration is 50 minutes plus additional 15 minutes to complete the new transactions from the source to target.
- In the third scenario, we applied transaction workload during the migration process at the source side. As both Logical Replication and transaction workload will consume the resource at source and target side at the same time, the estimated total migration duration is 73 minutes. Note this is not a recommended way to migrate your databases to Data Services Manager, but only a simulation of a use case where the source workload can not stop due to certain business reasons.

Logical Replication migration result		
Scenario	Description	Estimated Migration Duration (Normalized)
Migration with no workload on the source database	Migrate sample source PostgreSQL with about 146GB data	Up to 10 time slots
Workload applied on the source database after migration completed	Migrate sample source PostgreSQL with about 146GB data, then apply pgbench workload with 100000 transactions	Up to 10 + 3 time slots

Workload applied on the source database during migration	Migrate sample source PostgreSQL with about 146GB data, and apply pgbench workload with 100000 transactions during the migration process	Up to 15 time slots

PostgreSQL – Migration with Backup and Restore Methodology

The backup and restore methodology refers to using *pg_dump, pg_restore* and *pg_dumpall* utilities to export the desired source databases and metadata, to the target databases and objects managed by Data Services Manager. The *pg_dump* utility dumps in plain-text files containing the SQL commands required to reconstruct the database to the state it was in at the time it was saved. The restore can be completed by feeding the dump files to *psql* utility or *pg_restore* utility to rebuild the database at the target side.

Here is the difference between *pg_dump* and *pg_dumpall*.

- *pg_dump* only dumps a single database.
- *pg_dump* supports flexible output files with a custom format which is compressed by default.
- *pg_dumpall* backup the entire database cluster including all databases.
- *pg_dumpall* also backs up the global objects that are common to a cluster, such as database roles and tablespaces.

Migration Prerequisites

The *pg_dump* command requires a staging location for the output files, in a format that is flexible to restore the target database into a Data Services Manager deployed PostgreSQL database. Read the following prerequisites before migrating with the backup and restore methodology

- Verify that you have the necessary CLI tools installed in your source environment including the *pg_dump*, *pg_restore*, *psql* and *pg_dumpall* utilities.
- Obtain a list of databases under migration. The *pg_dump* command can export with custom output with compression enabled by default, but can only migrate one database at a time, while *pg_dumpall* is more useful to migrate multiple databases or the entire database cluster within a batch.
- Use *pg_dumpall* with the "*–schema-only*" option to migrate global database objects including roles, tablespaces etc.
- *pg_dumpall* needs to connect to the database with **superuser privileges** to read tables from all databases on the source side and execute the saved script on the target side. For more details, read pg_dumpall requirements.

Migration Steps (pg_dump to migrate single database)

➢ Step 1 – On the source database, use **pg_dump** to create the dump files. You may specify "-Fc" to allow custom format for the dump file.

```
pg_dump -h source-host -p port-number -U user-name -Fc -b -v -f dump-file-pwd -d  database-name
```

Example:

```
~$ pg_dump -h localhost -p 5432 -U postgres -Fc -b -v -f dumpsourcedb.sql -d sourcedb1
pg_dump: last built-in OID is 16383
pg_dump: reading extensions
pg_dump: identifying extension members
pg_dump: reading schemas
...
```

```
…
pg_dump: dumping contents of table "public.pgbench_accounts"
pg_dump: dumping contents of table "public.pgbench_branches"
pg_dump: dumping contents of table "public.pgbench_history"
pg_dump: dumping contents of table "public.pgbench_tellers"
```

➢ Step 2 – Restore the dump files to the target database in Data Services Manager

```
pg_restore -v -h target-host -U user-name -d database-name -j number-of-jobs dump-file-pwd
```

Example:

```
~$ pg_restore -v -h 10.211.198.233 -U pgadmin -d dsmpg1 -j 2 dumpsourcedb.sql
pg_restore: connecting to database for restore
Password:
pg_restore: processing item 3416 ENCODING ENCODING
pg_restore: processing item 3417 STDSTRINGS STDSTRINGS
pg_restore: processing item 3418 SEARCHPATH SEARCHPATH
pg_restore: processing item 3419 DATABASE sourcedb1
pg_restore: processing item 217 TABLE pgbench_accounts
pg_restore: creating TABLE "public.pgbench_accounts"
…

…
pg_restore: finished item 3412 TABLE DATA pgbench_accounts
pg_restore: launching item 3263 CONSTRAINT pgbench_accounts pgbench_accounts_pkey
pg_restore: creating CONSTRAINT "public.pgbench_accounts pgbench_accounts_pkey"
pg_restore: finished item 3263 CONSTRAINT pgbench_accounts pgbench_accounts_pkey
pg_restore: finished main parallel loop
```

➢ Step 3 – On the source database, export the global objects using *pg_dumpall*

```
pg_dumpall -U user-name -h source-host -f dump-global-pwd --no-role-passwords -g
```

Example:

```
~$ pg_dumpall -U postgres -h localhost -f dumpglobal.sql --no-role-passwords -g
```

➢ Step 4 – Import the global object into the database provisioned by Data Services Manager

```
psql -h target-host -U user-name -f dump-global-pwd
```

Example:

```
~$ psql -h 10.211.198.233 -U pgadmin -f dumpglobal.sql dsmpg1
Password for user pgadmin:
SET
SET
SET
ALTER ROLE
ALTER ROLE
ALTER ROLE
ALTER ROLE
```

Migration Steps (pg_dumpall to migrate multiple databases at the source)

- Step 1 – Get a list of databases on the source side to migrate to Data Services Manager.

  In this example, we populated the two sample source databases named “sourcedb1” and “sourcedb2” using the *pgbench* utility with scale factor set to 5000. Each sample database is 73GB.

```
postgres-# \l+
                                                    List of databases
  Name    | Owner   | Encoding | ... |  Size   | Tablespace |            Description
----------+----------+----------+ ... +---------+------------+----------------------------------------
bucardo   | postgres | UTF8     | ... | 8636 kB | pg_default |
postgres  | postgres | UTF8     | ... | 7524 kB | pg_default | default administrative connection database
sourcedb1 | postgres | UTF8     | ... | 73 GB   | pg_default |
sourcedb2 | postgres | UTF8     | ... | 73 GB   | pg_default |
template0 | postgres | UTF8     | ... | 7345 kB | pg_default | unmodifiable empty database
template1 | postgres | UTF8     | ... | 7564 kB | pg_default | default template for new databases

(6 rows)
```

- Step 2 – Use *pg_dumpall* on the source host to create the dump files of multiple databases under migration

```
pg_dumpall -U user-name -f dump-file-pwd --exclude-database=PATTERN
```

Example:

```
~$ pg_dumpall -U postgres -f dumpall.sql --exclude-database=bucardo --exclude-
database=postgres --exclude-database=template0 --exclude-database=template1
```

- Step 3 – Reload the multiple databases into a Data Services Manager-managed database at the target side

```
psql -h target-host -U user-name -p port-number -f dump-file-pwd target-connection-
database
```

Example:

The target database instance has been created by Data Services Manager. In this example, it is called “dsmpg1”. The command recreates the two databases from the source side (named “sourcedb1” and “sourcedb2”) on the target side.

```
psql -h 10.211.198.233 -U pgadmin -p 5432 -f dumpall.sql dsmpg1

…

CREATE DATABASE
ALTER DATABASE
psql (16.9 (Ubuntu 16.9-0ubuntu0.24.04.1), server 16.8 (VMware Postgres 16.8.1))
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, compression: off)
You are now connected to database "sourcedb1" as user "pgadmin".
```

```
…

CREATE DATABASE
ALTER DATABASE
psql (16.9 (Ubuntu 16.9-0ubuntu0.24.04.1), server 16.8 (VMware Postgres 16.8.1))
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, compression: off)
You are now connected to database "sourcedb2" as user "pgadmin".
SET
SET
SET
…
```

➢ Step 4 – Verify the multiple databases in the target database instance provisioned by Data Services Manager. You can see the multiple databases “sourcedb1” and “sourcedb2” have been successfully migrated to the target database instance.

```
dsmpg1=# \l+
                                    List of databases
  Name    |         Owner         | ... |  Size   | Tablespace |            Description
----------+-----------------------+ ... +---------+------------+---------------------------------------------
dsmpg1    | pgautofailover_replicator | ... | 7684 kB | pg_default |
postgres  | postgres              | ... | 7724 kB | pg_default | default administrative connection database
sourcedb1 | postgres              | ... | 73 GB   | pg_default |
sourcedb2 | postgres              | ... | 73 GB   | pg_default |
template0 | postgres              | ... | 7505 kB | pg_default | unmodifiable empty database
template1 | postgres              | ... | 7772 kB | pg_default | default template for new databases

(6 rows)
```

Tested Configuration

Once the migration steps are completed, configure your applications to start using the PostgreSQL databases provisioned by VMware Data Services Manager.

The backup and restore methodology using *pg_dump* or *pg_dumpall* requires certain business downtime during the migration process, compared to migration using Logical Replication. Here’s the tested configuration running both source and target databases on vSAN and we measured the test result. Both source and target PostgreSQL databases are configured with 2 vCPU and 8GB memory. Note the estimated migration duration is subject to change due to the hardware configuration, network bandwidth, resource allocation and other factors and should be only used for reference purposes.

From the test result we can see the same compute/network/storage resource allocation, backup and restore methodology may take more time than migration using Logical Replication. However *pg_dump* and *pg_dumpall* can help migrate both database and global objects, and support database DDL migration (e.g. schema/tablespace creation etc.) to the target side, which can not be achieved by Logical Replication only.

Migration with backup and restore test result (pg_dump)		
Scenario	Description	Estimated Migration Duration (Normalized)
Backup using *pg_dump*	Backup sample source PostgreSQL with about 146GB data, with "-Fc" option (compression) enabled	Up to 3 time slots
Restore using *pg_restore*	Restore the sample database to target database in Data Services Manager	Up to 10 time slots
Backup and Restore global objects	Export and import global objects using *pg_dumpall*	Completed at once

Migration with backup and restore test result (pg_dumpall)		
Scenario	Description	Estimated Migration Duration (Normalized)
Backup using *pg_dumpall*	Backup two source PostgreSQL database with total 146GB data. Note this is with no compression for dump files	Up to 2.5 time slots
Restore databases in Data Services Manager	Restore the two sample databases to target database in Data Services Manager	Up to 10 time slots

PostgreSQL – Migration with Bucardo

Bucardo is an open-source asynchronous PostgreSQL replication system that allows for multi-source, multi-target database migrations. Bucardo is a Perl daemon that listens for notify requests and acts on them, by connecting to target databases and copying data back and forth. All the specific information that the daemon needs is stored in the main bucardo database, including a list of all the databases involved in the replication and how to reach them, all the tables that are to be replicated, and how each is to be replicated. The replication operation occurs at the table level which is achieved by database triggers.

We recommend using Bucardo for PostgreSQL migrations to Data Services Manager, if the source database is a legacy version (prior to 10.x) of PostgreSQL and there is a requirement for minimal downtime.

Here are the limitations of migration using Bucardo.

- Bucardo migrates PostgreSQL databases with the granularity of table, which does not support migrating a table without a primary key.
- Bucardo does not migrate the DDL operations. Use *pg_dump –schema-only* to migrate the schema first from the source to the target database.
- Bucardo requires a central database which can either be deployed at source PostgreSQL or on a PostgreSQL on a third location. You can not deploy the central database on the PostgreSQL provisioned by the Data Services Manager.
- The initial table migration from the source database to the target (Data Services Manager-managed) database can be achieved via onetimecopy option available in Bucardo. This setting overrides the default pushdelta mode to use fullcopy mode.

Here are the high-level steps to migrate a standalone PostgreSQL database to a Data Services Manager-managed PostgreSQL database using Bucardo. For more details refer to the Bucardo documentation.

1. Configure the Bucardo dependencies and requirements and install Bucardo.
2. Deploy the target database using Data Services Manager
3. Migrate the schema from the source database to the target database using *pg_dump*
4. Add the source and target PostgreSQL databases to Bucardo.
5. Add the tables from the source to replicate the target with Bucardo.
6. Create Bucardo sync to start the replication.
7. Logon to PostgreSQL in Data Services Manager and monitor the migration progress.

Here are the tested configuration of migration with Bucardo

Bucardo migration result		
Scenario	Description	Estimated Migration Duration (Normalized)
Migrate PostgreSQL database to DSM using Bucardo	Source PostgreSQL with about 146GB data. Both source database and target database VM is configured with 2vCPU and 8GB memory	Up to 10 time slots

MySQL – Migration with Asynchronous Replication

MySQL replication enables data from the source MySQL database server to be copied to one or more MySQL database servers as the replicas. Replication is asynchronous by default and supports both statement-based and row-based replication formats. Binary log replication enables continuous data synchronization between source and target databases. MySQL Replication supports the following migration use cases:

- The source and target MySQL with different minor versions (within the same major version)
- Migration with minimal downtime requirements
- Continuous data synchronization during migration window

Migration Prerequisites

The logical replication replicates the Data Modification Language (DML) changes like INSERT, UPDATE, DELETE and TRUNCATE from the source MySQL database to the target Data Services Manager MySQL database. Before you migrate with logical replication, validate the following checklist.

- Verify the source database is version 8.0.37 or later
- Verify the target Data Services Manager is version 9.0.1 or later
- Verify the network connection, ports and firewall rules are allowed between the source external database and target MySQL database managed by Data Services Manager.
- Verify the source MySQL database instance is resolvable by DNS or accessible through an IP address.
- Verify the target MySQL database replica managed by Data Services Manager is with the same major version of the source database. For example, 8.0.x to 8.0.y or 8.4.x to 8.4.y. If your source database version is lower than the target, upgrade first to the desired MySQL version supported by Data Services Manager before migration.
- Verify the active plugins that are installed on the source database, and make sure the same plugins are also enabled on the target MySQL database managed by Data Services Manager.
    - Review the source database configuration file, for example, /etc/my.cnf, for manually enabled plug-ins
    - To find plug-ins enabled through SQL statements, run a query on the source database to check for active entries in the information_schema.plugins table that are associated with a specific plug-in library.

```
mysql> SELECT plugin_name, plugin_status, plugin_library FROM
information_schema.plugins WHERE plugin_status = 'ACTIVE' AND plugin_library !=
'NULL';
```

    - Install the plug-ins from the query results above on the target. (validate_password is installed by default)
- Verify the tables under migration on the source database using InnoDB database engine that Data Services Manager supports. Tables using other engines on the source may not be usable after migration. The following query helps determine which database engine the source databases are using.

```
mysql> SELECT table_schema, table_name, engine FROM information_schema.tables WHERE
table_schema NOT IN ('mysql', 'information_schema', 'performance_schema');
```

- VMware Data Services Manager leverages the MySQL clone plug-in and GTID-based replication. Verify that the source database meets all MySQL clone plug-in and GTID replication requirements. For more details, visit the following documentation
    - Remote Cloning Prerequisites - MySQL Reference Manual

- MySQL Replication Documentation
- MySQL Replication Compatibility Documentation

Migration Steps

➢ Step 1 – On the source MySQL database, check the following MySQL parameters

- Enabling binary logging (log_bin=ON) and changing this setting requires a database restart.
- Configure row-based replication by setting the binlog_format configuration parameter to "ROW".
- Activate the Global Transaction Identifiers (GTIDs) and configure the server to enforce GTID consistency by turning on the "gtid_mode" and "enforce_gtid_consistency" configuration parameters.

Sample configuration in my.cnf file

```
# Enable binary logging
log-bin = ON
binlog-format = ROW

# Enable GTID
gtid-mode = ON
enforce-gtid-consistency = ON
```

Verify the value of current settings

```
mysql> SELECT @@GLOBAL.log_bin;
mysql> SELECT @@GLOBAL.binlog_format;
mysql> SELECT @@GLOBAL.enforce_gtid_consistency;
mtsql> SELECT @@GLOBAL.gtid_mode;
```

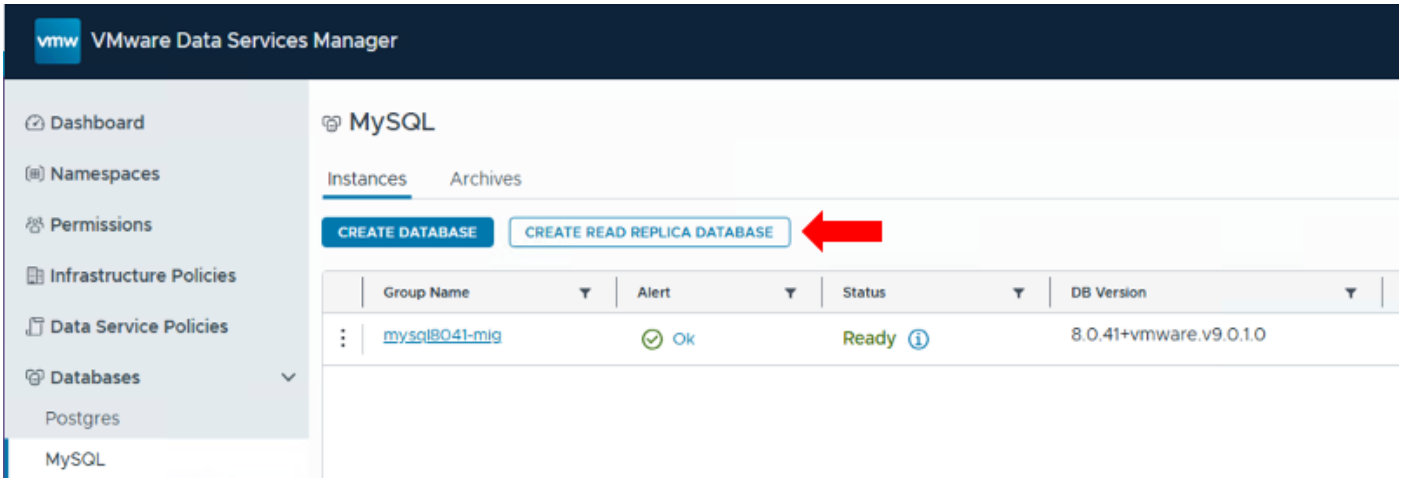
➢ Step 2 – Create a MySQL user on the source side and make sure it's granted with the following minimum privileges:

- BACKUP_ADMIN
- REPLICATION SLAVE
- SELECT
- INSERT

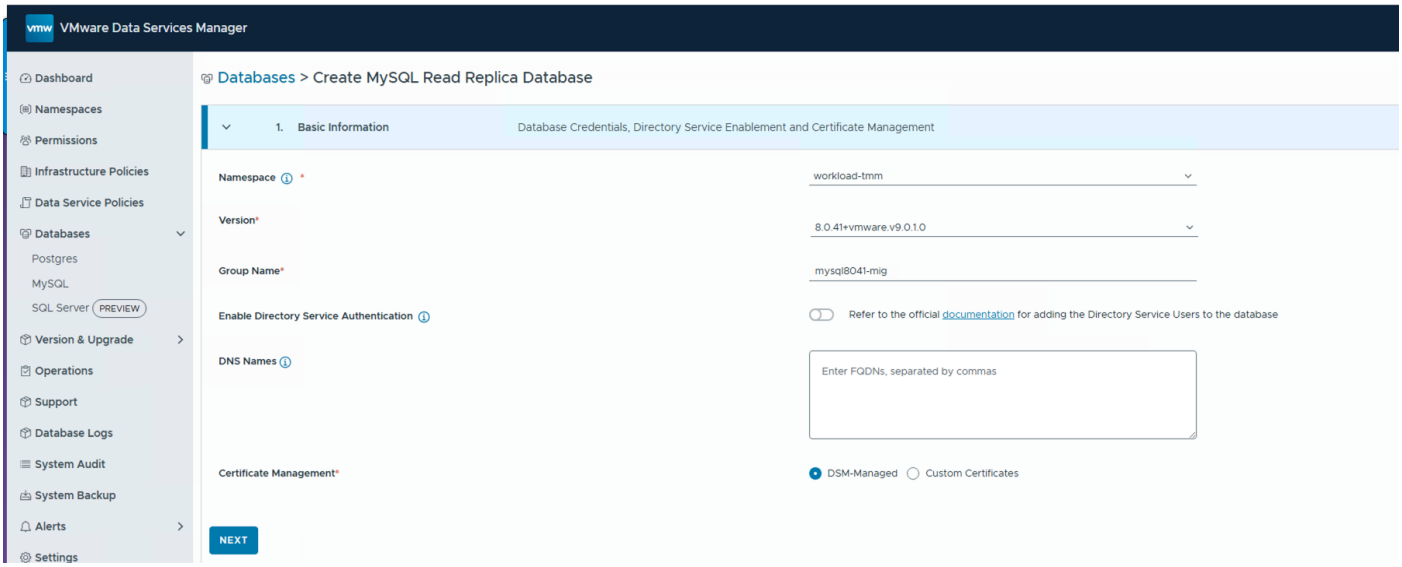
Sample code to create a replication user

```
CREATE USER IF NOT EXISTS 'replica-user'@'%' IDENTIFIED WITH caching_sha2_password BY
'strong_replica_password';
GRANT BACKUP_ADMIN, REPLICATION SLAVE, SELECT ON *.* TO 'replica-user'@'%';
GRANT INSERT ON mysql.plugin TO 'replica-user'@'%';
```

➢ Step 3 – Login to Data Services Manager and create a MySQL replica database

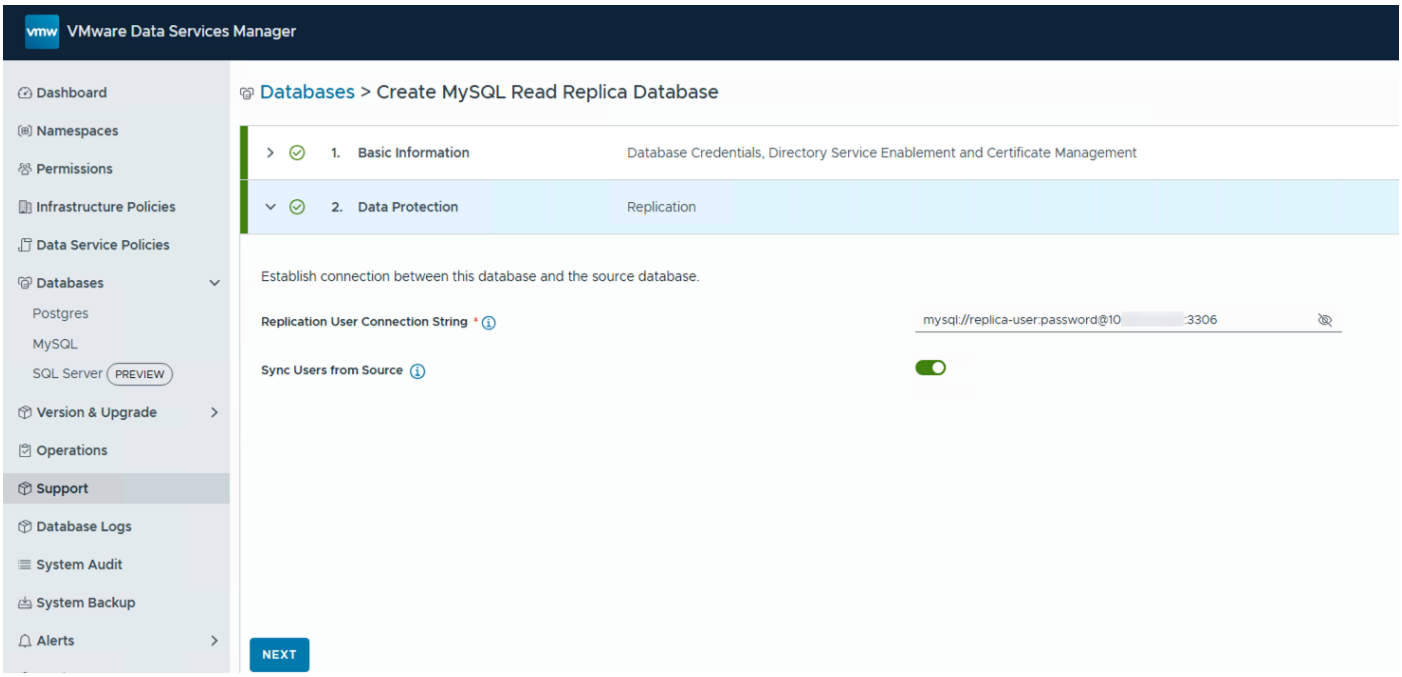


➢ Step 4 – Fill in the basic information of the MySQL replica database created by Data Services Manager

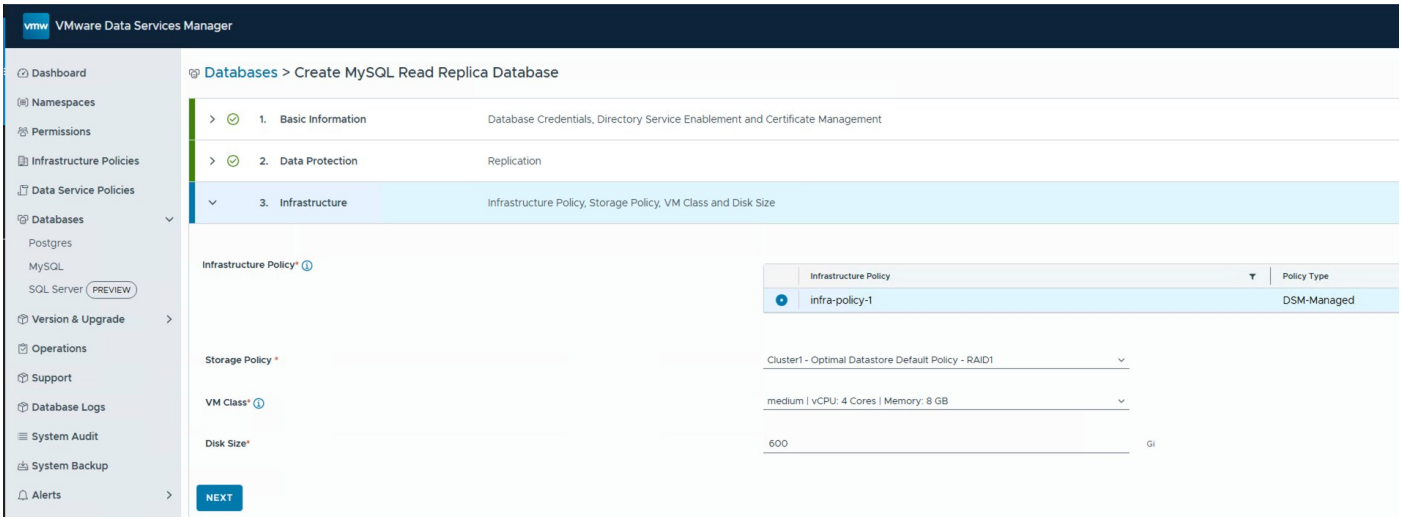


➢ Step 5 - Get the MySQL connection string for the source database with the following format.

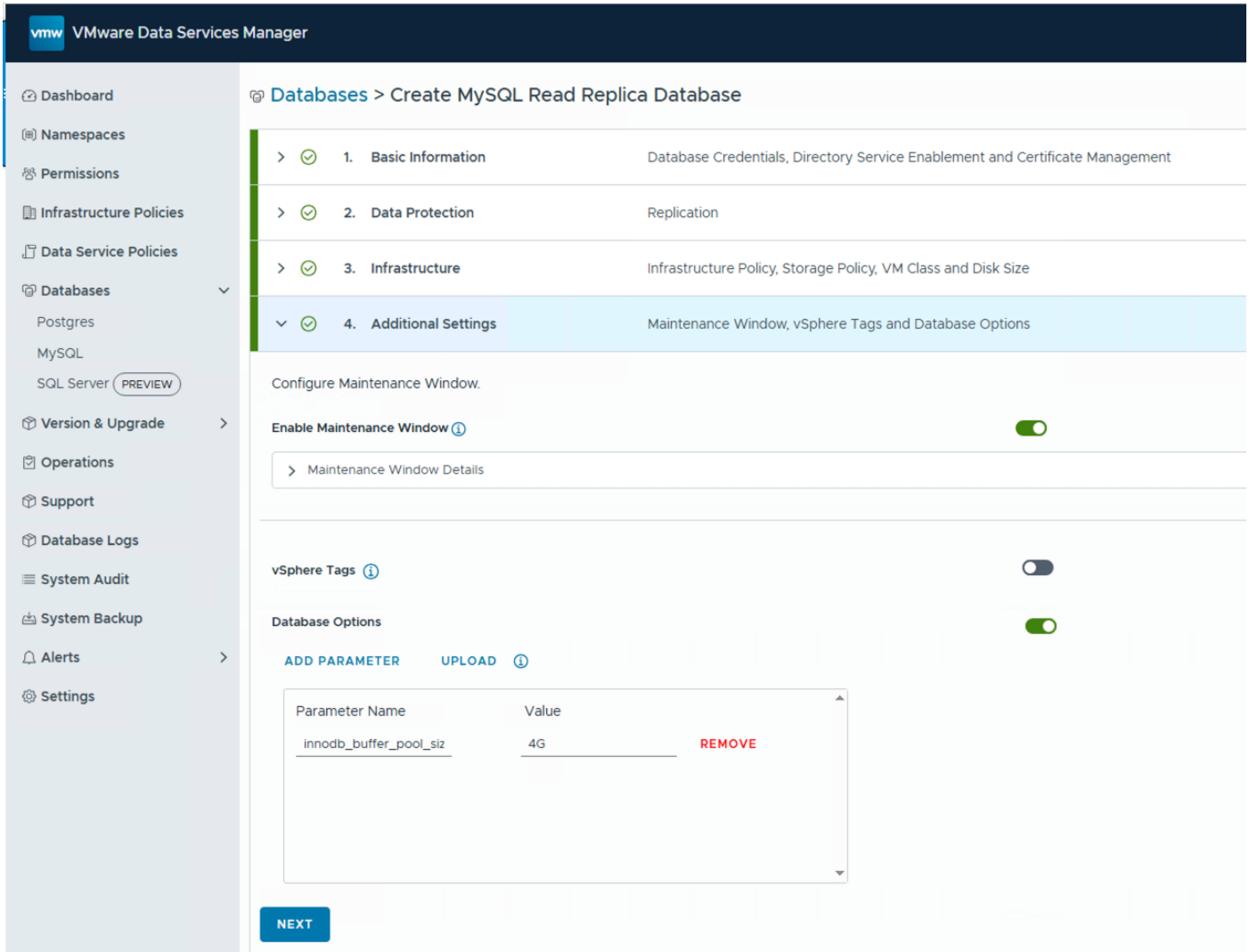
```
mysql://<user>:<password>@<external-mysql-ip>:<external-mysql-port>
```



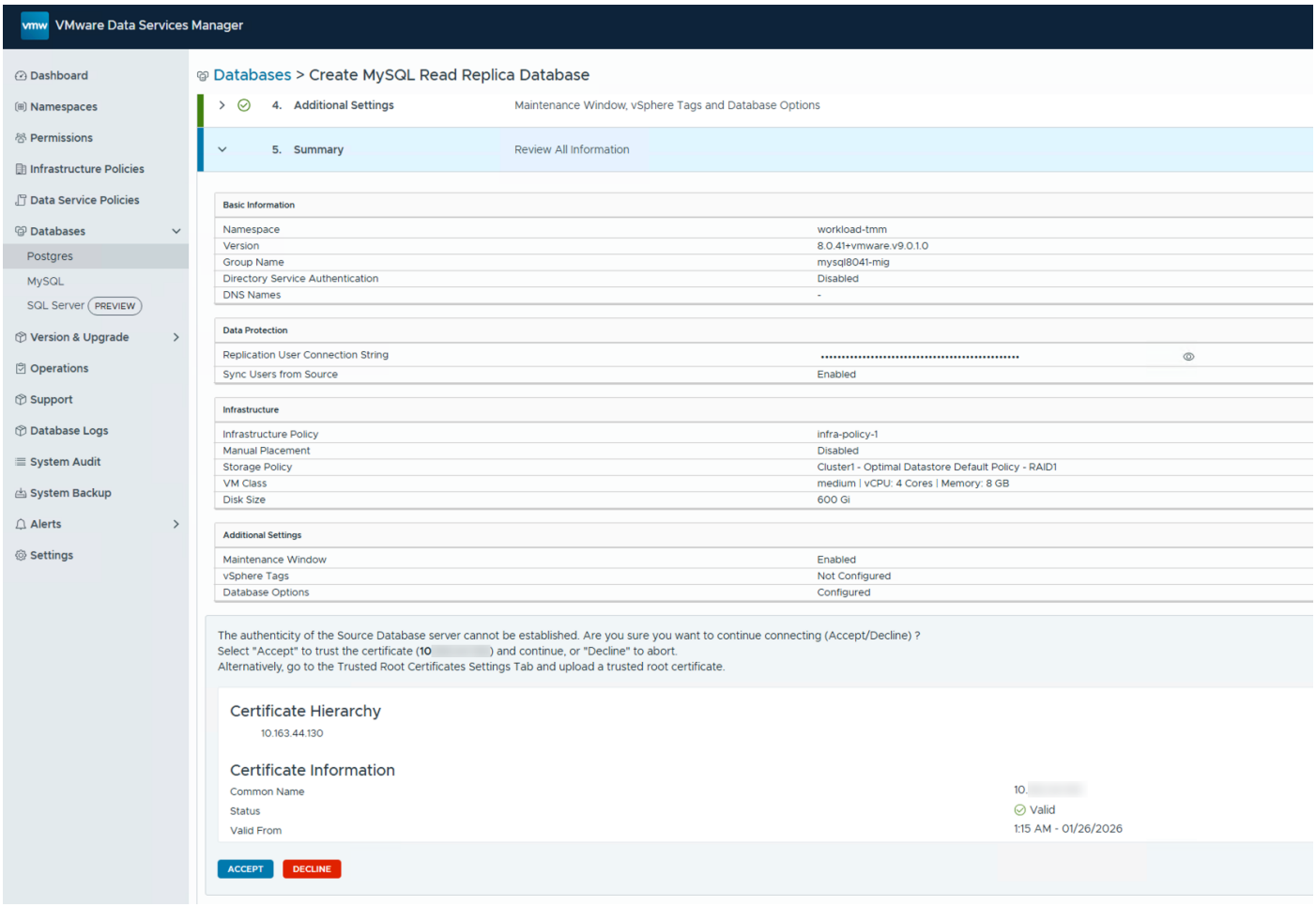
➢ Step 6 - Size the target MySQL database replica with the desired storage policy, VM class and data disk size.



➢ Step 7 - Configure the additional settings such as maintenance window and Innodb engine parameters in database options. Those settings can be also tweaked post deployment.

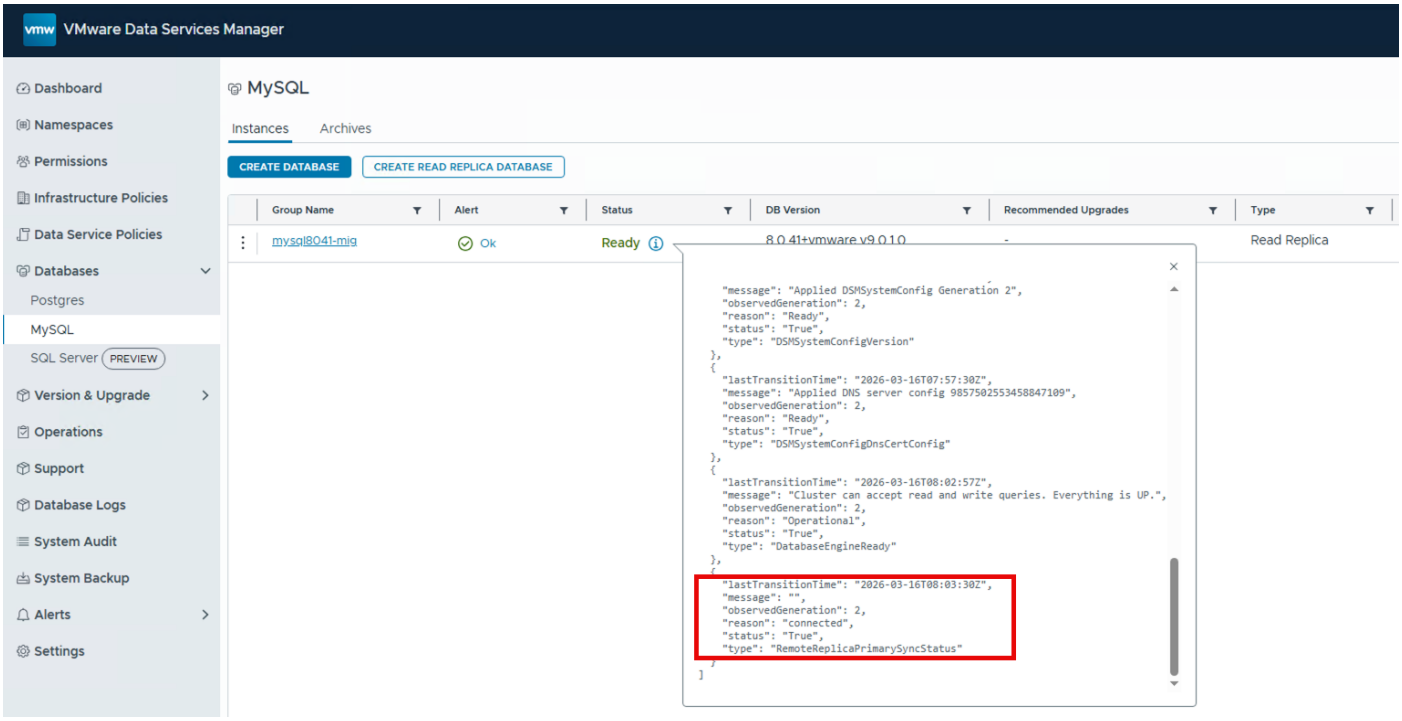


➢ Step 8 - Review the configuration details. Note the source database requires MySQL encrypted connection (SSL) configured so that a Read Replica in Data Services Manager can be created. For more details please refer to Configuring MySQL to Use Encrypted Connections documentation



Once click accept certificate button, Data Services Manager will start creating a MySQL read replica database of your source database for migration.

➢ Step 9 - Monitor the read replica status. Once the migration is completed, you can see the “RemoteReplicaPrimarySyncStatus” becomes in “True” state for a successful replication.



You can also query with the following code for replica status. Here’s an sample output (partial)

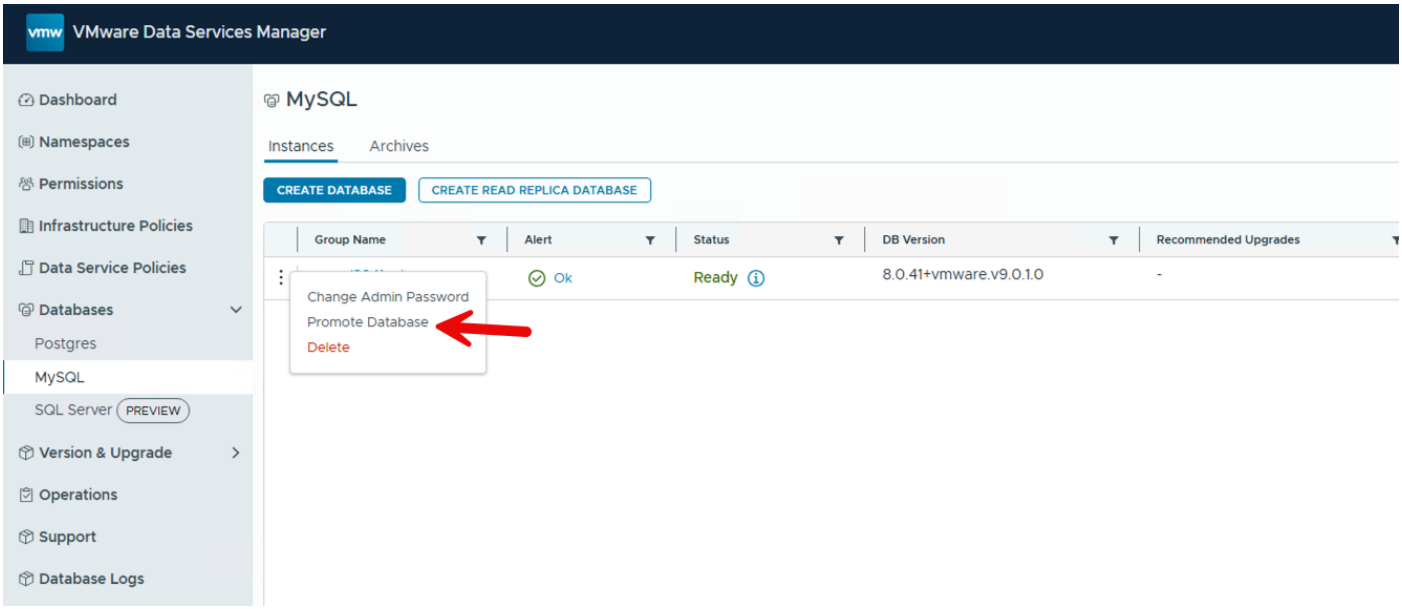
```
mysql> SHOW REPLICA STATUS\G;
*************************** 1. row ***************************
            Replica_IO_State: Waiting for source to send event
                 Source_Host: 10.xxx.xxx.xxx
                 Source_User: replica-user
                 Source_Port: 3306
               Connect_Retry: 10
             Source_Log_File: ON.000021
         Read_Source_Log_Pos: 197
              Relay_Log_File: relaylog-dsm@002dasync@002dreplication@002d0.000002
               Relay_Log_Pos: 359
       Relay_Source_Log_File: ON.000021
          Replica_IO_Running: Yes
         Replica_SQL_Running: Yes
             Replicate_Do_DB:
         Replicate_Ignore_DB: mysql_innodb_cluster_metadata
… …
… …
            Source_Info_File: mysql.slave_master_info
                   SQL_Delay: 0
         SQL_Remaining_Delay: NULL
```

```
   Replica_SQL_Running_State: Replica has read all relay log; waiting for more updates
         Source_Retry_Count: 6
                Source_Bind:
     Last_IO_Error_Timestamp:
    Last_SQL_Error_Timestamp:
              Source_SSL_Crl:
         Source_SSL_Crlpath:
         Retrieved_Gtid_Set:
          Executed_Gtid_Set: 9fe0aae5-ce87-11f0-8c0c-005056bc6be6:1-6129726
              Auto_Position: 1
       Replicate_Rewrite_DB:
               Channel_Name: dsm-async-replication-0
         Source_TLS_Version:
     Source_public_key_path:
      Get_Source_public_key: 0
          Network_Namespace:
1 row in set (0.00 sec)
```

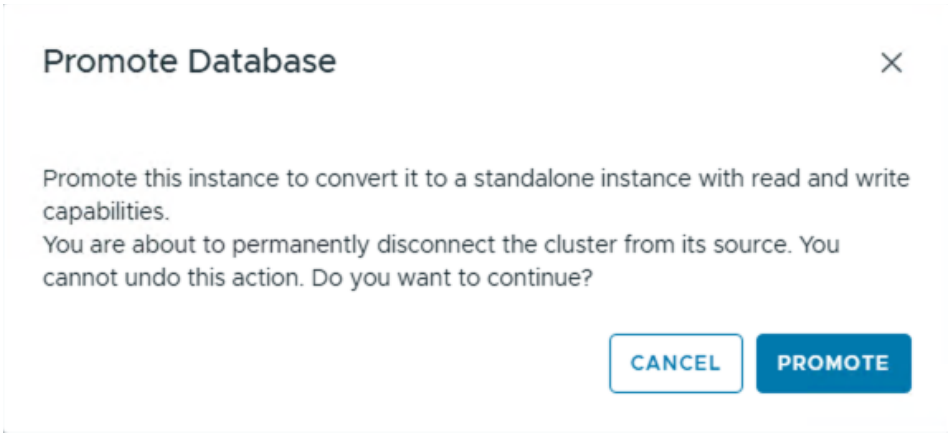
➢ Step 10 - Block all the workload on source and confirm the GTID sets match on both the source and target MySQL databases.

```
mysql> SELECT @@GLOBAL.gtid_executed;
+-----------------------------------------------+
| @@GLOBAL.gtid_executed                        |
+-----------------------------------------------+
| 9fe0aae5-ce87-11f0-8c0c-005056bc6be6:1-6129726 |
+-----------------------------------------------+
1 row in set (0.00 sec)
```

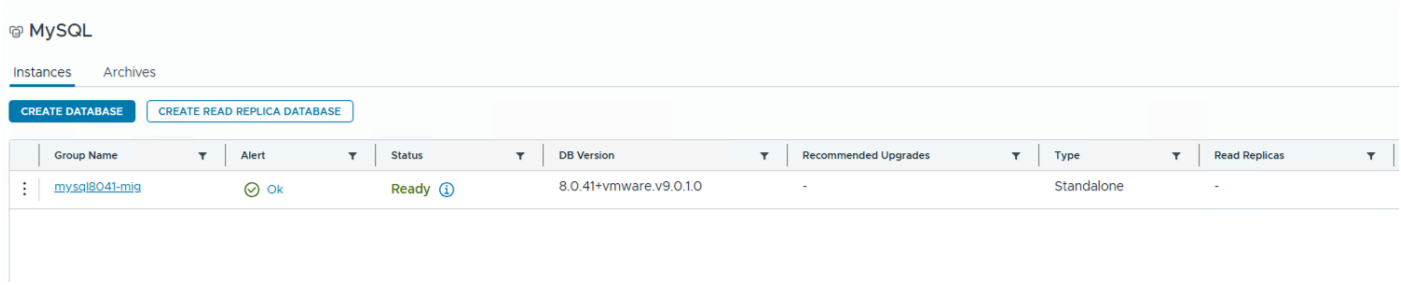
➢ Step 11 - Promote the replica database provisioned by the Data Services Manager and complete the migration.



Click “Promote” to convert the replica database to a standalone instance managed in Data Services Manager



Confirm the database is a “Standalone” database and migration is completed.



Tested Configuration

Once you have completed all the migration steps, you may configure your applications to start using the MySQL databases provisioned by VMware Data Services Manager.

We measured the migration result of the following test scenario. Both source and target MySQL databases are configured with 4 vCPU and 8GB memory. The migration duration is measured starting from the timestamp that a replica MySQL database was created by Data Services

Manager to the timestamp that shows “Ready” state from the console (screenshot in Step 9 above). Note the estimated migration duration is subject to change due to the hardware configuration, network bandwidth, resource allocation and other factors and should be only used for reference purposes.

- In the first scenario, we applied no workload during the migration process. This is a recommended way to migrate with MySQL Asynchronous Replication unless your production workload can not stop due to business requirements. The migration duration is normalized to 8 replication time slots.
- In the second scenario, we applied light OLTP workload on the source database during the migration, which is generated by sysbench tool with a single client thread and last for 60 seconds. The migration duration is slightly longer than the first scenario with 9 replication time slots.
- In the third scenario, we applied OLTP workload on the source database throughout the migration process with sysbench tool. The migration duration is again measured as 9 replication time slots. We also monitored replication activity after the read replica became “Ready” state. Due to the asynchronous nature of the read replica, it is estimated that the replication workflow will take longer even after the source workload was terminated.

MySQL – Migration with Backup and Restore Methodology

The backup and restore methodology of MySQL refers to using *mysqldump, mysqlpump or* MySQL Shell dump/load utilities to export the desired source databases to the target databases in Data Services Manager. The *mysqldump* utility creates logical backups containing SQL statements to recreate database structure and data to the state it was in at the time it was saved.

Here is the difference of the available tools

- *mysqldump* is a single-threaded backup tool which supports all storage engines. It generates SQL script files and suitable for small to medium sized databases
- *mysqlpump* (deprecated as of MySQL 8.0.34, not recommended) is a multi-threaded backup tool with parallel processing. It can be faster than *mysqldump* for large database backups and supports database-level parallelism
- MySQL Shell dump/load tool (recommended) is highly optimized for dumping and loading databases. It supports chunking and built-in compression to generate backup sets. MySQL Shell utility is available in MySQL 8.0 and later.

Migration Prerequisites

The *mysqldump* and related utilities require a staging location for the backup files, in a format that is flexible to restore the target database into a Data Services Manager deployed PostgreSQL database. Read the following prerequisites before migrating with the backup and restore methodology

- Verify that you have the necessary CLI tools installed in your source environment including the mysqldump, mysqlpump, and optionally MySQL Shell utilities.
- Verify that you have configured firewall rules to allow connection between the source environment and the target Data Services Manager environment.
- Get a list of databases under migration and ensure sufficient disk space available for backup files.
- For large databases, consider using --single-transaction option to avoid locking tables during backup.

Migration Steps (mysqldump)

➢ Step 1 - On the source database, use mysqldump or mysqlpump to create the backup files. For more details please refer to mysqldump manual.

Example:

```
~$ mysqldump --set-gtid-purged=off \
--no-tablespaces \
--single-transaction \
--user=root --password=P@ssw0rd \
--host=localhost \
--port=3306 \
--databases sourcedb > /tmp/sourcedb_backup.sql
```

➢ Step 2 - Transfer the backup file to a location accessible by the target Data Services Manager environment

➢ Step 3 - Restore the backup to the target database in Data Services Manager

Example:

```
mysql -h 192.168.xxx.xxx -P 5432 -u mysql-admin -p < /tmp/sourcedb_backup.sql
```

Migration Steps (MySQL Shell)

➢ Step 1 - Connect to the source database and use MySQL shell to create a parallel dump. For example, the following code uses MySQL shell to create a backup using 8 threads and compressed using zstd. For more details please refer to MySQL Shell manual.

```
MySQL  localhost:33060+ ssl  JS > util.dumpSchemas(['sourcedb'], '/tmp/sourcedb_dump', {threads:
8, showProgress: true, compression: 'zstd'})
Acquiring global read lock
Global read lock acquired
Initializing - done
1 schemas will be dumped and within them 10 tables, 0 views.
Gathering information - done
All transactions have been started
Locking instance for backup
Global read lock has been released
Writing global DDL files
Running data dump using 8 threads.
NOTE: Progress information uses estimated values and may not be accurate.
Writing schema metadata - done
Writing DDL - done
Writing table metadata - done
Starting data dump
101% (500.00M rows / ~493.61M rows), 1.42M rows/s, 282.95 MB/s uncompressed, 126.43 MB/s
compressed
Dump duration: 00:05:57s
Total duration: 00:05:58s
Schemas dumped: 1
Tables dumped: 10
Uncompressed data size: 98.89 GB
Compressed data size: 44.22 GB
Compression ratio: 2.2
Rows written: 500000000
Bytes written: 44.22 GB
Average uncompressed throughput: 276.43 MB/s
```

You can see the dump file of a 100GB source database is about 44GB after being dumped using MySQL shell with zstd compression.

➢ Step 2 - Connect to the target database and use MySQL shell to restore the dump filecreate a parallel dump.

```
MySQL  10.163.xxx.xxx:33060+ ssl  JS > util.loadDump("/tmp/sourcedb_dump", {threads: 8,
showProgress: true, resetProgress: true});
Loading DDL and Data from '/mnt/minio/mysql/mysqlsh' using 8 threads.
Opening dump...
Target is MySQL 8.0.41-32. Dump was produced from MySQL 8.0.41
```

```
Scanning metadata - done
Checking for pre-existing objects...
Executing common preamble SQL
Executing DDL - done
Executing view DDL - done
Starting data load
4 thds loading | 100% (98.89 GB / 98.89 GB), 25.68 MB/s, 10 / 10 tables done
Recreating indexes - done
Executing common postamble SQL
1653 chunks (500.00M rows, 98.89 GB) for 10 tables in 1 schemas were loaded in xx hour xx min xx sec (avg throughput xxx.xx MB/s)
0 warnings were reported during the load.
```

Tested Configuration

Once you have completed the above migration steps, you may configure your applications to start using the MySQL databases provisioned by VMware Data Services Manager.

The backup and restore methodology using mysqldump and MySQL Shell requires business downtime during the migration process compared to migration using MySQL Replication. Here's the tested configuration running both source and target databases on vSAN and we measured the test result. Both source and target MySQL databases are configured with 4 vCPU and 16GB memory. The test database is populated by Sysbench with 50 million rows with up to 100GB size. Note the estimated migration duration is subject to change due to the hardware configuration, network bandwidth, resource allocation and other factors and should be only used for reference purposes.

Migration with backup and restore test result				
Scenario	Database size	Backup Time	Restore Time	Description
mysqldump	100GB	3 time slots (backup)	15 time slots (restore)	Single thread
MySQL Shell	100GB	1 time slot (backup)	5 time slots (restore)	8 threads

Best Practices and Key Considerations

- It is highly recommended to perform the entire migration steps with a test/dev database before migration of the actual production databases. This will help identify potential issues and resolve them beforehand.
- Migrate PostgreSQL databases with logical replication
    - Minimize the business downtime of the source PostgreSQL databases.
    - Replicate the subsequent changes during the migration period of your source databases.
    - Require source PostgreSQL database with version 10.0 or later.
- Migrate PostgreSQL databases with backup and restore methodology (*pg_dump* and *pg_restore*)
    - Simple to configure and can be considered for small-to-medium sized source PostgreSQL database environments (requires storage for backup output files).
    - Support to migrate PostgreSQL objects including view, triggers, stored procedures etc.
    - Support to migrate PostgreSQL global objects including database roles, tablespaces, and privilege grants for configuration parameters using *pg_dumpall* utility.
    - *pg_dump* migrates single database at one time with compression for dump files by default
    - *pg_dumpall* can migrate multiple databases or the entire database instance within a batch
    - Make sure the source database has no workload / transaction during the migration process (extended down time required). *pg_dump* does not capture ongoing changes.
- Migrate PostgreSQL databases with open-source tool - Bucardo - for legacy versions of PostgreSQL databases
    - Bucardo can support multi-source / multi-target migration.
    - Read the limitations of the Bucardo tool before migrating the databases.
    - If you have not used this tool previously, be sure to install it in a test-dev environment and familiarize yourself with its use and capabilities.
    - Bucardo is an open source tool maintained by the community. Check the Bucardo page for updates, support and for further open-source tool related help.
- Migrate MySQL databases with Asynchronous Replication
    - Minimize the business downtime of the source MySQL databases.
    - Replicate the subsequent changes during the migration period of your source databases
    - Require source MySQL database with version 8.0.37 or later. Upgrade before migration if the source database version is lower than requirement.
- Migrate MySQL databases with backup and restore (mysqldump and MySQL Shell)
    - Mysqldump is simple to configure and can be considered for small to medium sized migration scenarios.
    - Support to migrate MySQL objects including views, triggers, stored procedures, and events.
    - MySQL Shell is available only for MySQL 8.0 and later versions for better migration experience with parallel processing, compression options. This is a recommended tool to help migrate MySQL database to Data Services Manager using backup and restore methodology.
    - Make sure the source database has no workload / transaction during the migration process.
- Data Services Manager supports the following deployment options for PostgreSQL and MySQL database high availability. You may choose the deployment type for the target database based on your business SLAs.
    - Single server deployment - standalone option for databases provided by Data Services Manager.
    - Cluster deployment - You can deploy databases with primary and replica across single or multiple vSphere clusters. It provides automatic failover and resiliency to a single compute node failure.
- VM Class

  - Data Services Manager maintains a database virtual machine template with an optimized configuration so that the infrastructure administrators only need to focus on the sizing of the virtual machine class for the database.
  - Data Services Manager supports customized virtual machine classes for PostgreSQL and MySQL databases. It is recommended to start with the same sizing as of the source database configuration in terms of CPU cores and memory and you can easily scale-up or scale-down for different use cases.
- IP Pools
  - Assign sufficient IP addresses to each of the IP Pools for data services provisioned by Data Services Manager. Single server deployments require one IP address for each node, as well as two additional cluster-level IP addresses for the database and control plane access. Three-node cluster deployments require 5 IP addresses.
  - Maintenance operations such as database engine upgrades or changes to cluster topology require additional IP addresses. For planning purposes, you need to calculate the size of the pool based on a higher number of servers and clusters.
- Compute Resources
  - Make sure to enable vSphere HA and vSphere DRS for the cluster used to deploy data services by Data Services Manager. For more information, refer to How vSphere HA Works in vSphere Availability and Create a vSphere DRS Cluster in vSphere Resource Management.
- Storage Policies
  - For vSAN use FTT=1 or FTT=2 based on the protection level required for the data services. vSAN Express Storage Architecture provides RAID-5 for FTT=1 and RAID-6 for FTT=2 with optimal performance and capacity. For more information, refer to Using RAID 5 or RAID 6 Erasure Coding in vSAN Cluster
  - For external storage, create a host-based or tag-based storage policy for Data Services Manager. See Creating and Managing vSphere VM Storage Policies.

Item No: vmw-bc-wp-tech-temp-a4-word-2024 1/24