MIGRATION PLAYBOOK · DATABASES

How to move MySQL to a managed cloud database

Move a MySQL database from your own server or VM to Amazon RDS for MySQL or Azure Database for MySQL Flexible Server. Copy it with a dump, keep it in sync with binary log replication, and cut over in minutes instead of a night.

Updated 4 Nov 2026 5 phases 12 detailed steps 5 official sources

START HERE

How do you move MySQL to a managed cloud database?

SHORT ANSWER

This moves a MySQL database you run yourself into a managed MySQL service in AWS or Azure. You take a consistent copy of the database, load it into the new service, then let the new service follow changes on the old server through replication (it reads the old server's binary log, the record of every change).

- **Recommended route:** mysqldump (or mydumper for large databases) to load the copy, then binary log replication from the old server until cutover
- **Order:** Check the source can replicate before you build anything, load the copy, then let replication run for a few days so cutover is just a short pause.
- **5 phases:** Assess, Prepare source and target, Copy and replicate, Cut over and Finish.
- **You're done when:** The application runs on the managed database with no errors.

At cutover you stop the application, wait for replication to catch up, and point the application at the new database.

RECOMMENDED ROUTE

mysqldump (or mydumper for large databases) to load the copy, then binary log replication from the old server until cutover

Both AWS and Microsoft document this pattern for minimal downtime: the bulk copy happens while the application runs, and only the final catch-up needs a pause.

▼ **Other options and when to use them**

- **Dump and import only**
  : A small database where an outage of an hour or two is acceptable.
- **AWS Database Migration Service or Azure Database Migration Service**
  : You want a guided, repeatable service to run the copy and the change capture.

TOOLS FOR THIS PLAN

Are you ready?
22-question readiness check

Plan your dates
Dated countdown you can add to your calendar

New to the jargon?
Plain-English migration glossary

ORDER TO MIGRATE IN, AND WHY

Check the source can replicate before you build anything, load the copy, then let replication run for a few days so cutover is just a short pause.

1. **Inventory: MySQL version, size, storage engines, tables without primary keys, users and scheduled events**
   Replication needs InnoDB and works best with a primary key on every table; users and routines must be recreated by hand.
2. **Binary logging on the source and a replication user**
   Replication reads the binary log, and the replica signs in with an account that has REPLICATION CLIENT and REPLICATION SLAVE.
3. **Target database created with backups and Multi-AZ left off**
   AWS advises enabling Multi-AZ, backup retention and read replicas only after the import, to speed up loading.
4. **Consistent dump loaded into the target**
   A dump taken with --single-transaction and --source-data records the binary log position replication starts from.
5. **Replication running and caught up**
   Once lag is zero, the target holds every change and cutover only waits for the last few seconds.

6 **Cutover, then backups and high availability turned on**

The application moves while the old server is still intact as a fallback.

SETTINGS CHECKLIST: DECIDE THESE BEFORE YOU COPY

Built-in and third-party tools treat these differently. Tick each one off once you have decided.

Download this checklist (Excel)

- [ ] **Method by size and downtime**

  Dump and import, dump plus replication, or a migration service.

  ▼ How each tool handles it

Built-in method	AWS lists: a command-line copy for small databases with some downtime; replication from an external source for any size with minimal downtime; AWS DMS for minimal downtime and ongoing replication. Microsoft notes mysqldump is not fast for large databases and suggests mydumper and myloader, which run on Linux.
MyDumper and MyLoader	Open-source parallel dump and load; Microsoft says it can speed up migration by a factor of 10 or more.

  Tip: Test the dump and load time on a copy first; it decides whether you need replication at all.

  AWS: Importing data into an Amazon RDS for MySQL DB instance

- [ ] **Source requirements**

  Whether the source can replicate as it is.

  ▼ How each tool handles it

Built-in method	Azure data-in replication: source MySQL 5.7 or later, ideally the same major version as the target, InnoDB engine, a primary key on each table, and port 3306 open both ways. AWS: binary logging on (log-bin) with a unique server-id.

  Tip: Convert MyISAM tables to InnoDB and add primary keys on the source before you start, during a quiet period.

  Microsoft Learn: Data-in replication in Azure Database for MySQL Flexible Server

- [ ] **Users, routines and events**

  How accounts and stored code get to the new server.

▼ How each tool handles it

Built-in method	Azure: the mysql system database and changes to accounts and permissions are not replicated. AWS: create stored procedures, triggers, functions and events manually on RDS, and exclude them from the dump with --routines=0 --triggers=0 --events=0.

Tip: Script users, grants, routines and events from the source and apply them to the target before cutover.

AWS: Importing data to an Amazon RDS for MySQL database with reduced downtime

☐ **Network path**

How the managed database reaches the old server for replication.

▼ How each tool handles it

Built-in method	AWS: the RDS instance must be publicly accessible to set up replication with the source in this procedure. Azure: data-in replication is not supported with a private endpoint only (public access disabled).

Tip: Allow only the managed database's address to reach port 3306 on the source, and turn public access off again after cutover where you can.

Microsoft Learn: Data-in replication in Azure Database for MySQL Flexible Server

☐ **MySQL 8 invisible primary keys**

How to handle tables without primary keys on Azure.

▼ How each tool handles it

Built-in method	Azure: generated invisible primary keys are on by default for MySQL 8.0 and can cause errors such as "Multiple primary key defined" during data-in replication.

Tip: Add real primary keys on the source, or follow Microsoft's fix for the affected tables, before starting replication.

Microsoft Learn: Data-in replication in Azure Database for MySQL Flexible Server

THE PLAN, PHASE BY PHASE

1 Assess — Week 1

Record MySQL version, database

sizes and storage engines.

Find tables without primary keys and non-InnoDB tables.

List users, grants, routines, triggers and events.

> **Watch out:** Tables without primary keys replicate slowly or fail on Azure with invisible primary keys.
>
> **WAYS AROUND IT**
>
> - Add primary keys on the source before replication.

Show the detailed steps (2) →

2 Prepare source and target

Week 1 to 2

Turn on binary logging with a unique server-id on

the source.

Create a replication user with REPLICATION CLIENT and REPLICATION SLAVE.

Create the managed database with Multi-AZ and backups off until after the load.

> **Watch out:** Turning on binary logging needs a MySQL restart on the source.
>
> **WAYS AROUND IT**
>
> - Do it in a short maintenance window ahead of the migration.

Show the detailed steps (3) →

3 Copy and replicate Week 2

Take a consistent dump that records

the binary log position.

Load it into the managed database.

Start replication from that position and watch the lag fall to zero.

> **Watch out:** Binary logs purged on the source before the replica reads them break replication.
>
> **WAYS AROUND IT**
>
> - Keep binary logs on the source for longer than the copy and load take.

Show the detailed steps (4) →

4 Cut over Planned window

Stop the application and confirm replication lag is zero.

Stop replication and point the application at the new endpoint.

Start the application and test.

> **Watch out:** Writes made on the old server after cutover are not copied.
>
> **WAYS AROUND IT**
>
> - Make the old database read-only before switching.

Turn this phase into dated tasks with the cutover planner

Show the detailed steps (3) →

5 Finish

After cutover

Turn on automated backups and Multi-AZ or high availability.

Remove the replication configuration and

close the source firewall rule.

Keep the old server read-only for the fallback period, then retire it.

> ⚠ **Watch out:** Leaving public access on longer than needed exposes the database.
>
> **WAYS AROUND IT**
>
> - Turn off public access once replication has been removed, where your setup allows.

Show the detailed steps (3) →

✓ You are finished when

- The application runs on the managed database with no errors.
- Row counts and checksums match for key tables.
- Users, routines and scheduled events work on the new server.
- Backups and high availability are on, replication is removed, and the old server is read-only.

CLICK BY CLICK

What does each step involve?

Open a part when you reach it in the plan. Each step says where to go, what to do and what you should see.

About this migration

What it covers, limits at a glance

A managed MySQL service runs the server, patches it and takes backups for you. Amazon RDS for MySQL and Azure Database for MySQL Flexible Server both accept data from an external MySQL server in the same way: load a copy, then replicate changes from the old server using its binary log until you switch the application over.

The method you pick depends on size and how much downtime the application can take. For a WordPress site on the same database, also see WordPress migration. For SQL Server, see databases to Amazon RDS.

SMALL DATABASE, DOWNTIME FINE **mysqldump and import**	SHORT DOWNTIME **Dump and load, then binary log replication until cutover**	AWS REPLICATION **mysql.rds_set_external_source, then mysql.rds_start_replication**
AZURE REPLICATION **Data-in replication, from a MySQL 5.7 or later source**	NOT COPIED BY REPLICATION **The mysql system database, users and permissions**	LARGE DATABASES **Microsoft suggests mydumper and myloader (Linux)**

1 What do you need before you move MySQL to a managed cloud database?

Need	Details	Where to get it
Admin access to the source MySQL	To enable binary logging and create users.	Your server or hosting panel
AWS or Azure subscription	Rights to create RDS or Azure Database for MySQL resources.	AWS console or Azure portal
A machine near the target for dump and load	AWS uses an EC2 instance in the same region; Microsoft suggests Linux for mydumper.	EC2 or an Azure VM

Need	Details	Where to get it
Application owner and test plan	Someone to test the application against the new database.	Your business team
Maintenance windows	One to enable binary logging, one for cutover.	Change advisory process

2 Assess the source database

2 steps

1 Check version, size and engines

Where: mysql client connected to the source as an admin

Record the version and list each table's engine and size. Convert non-InnoDB tables before you start.

```
SELECT VERSION();
SELECT table_schema, table_name, engine, ROUND((data_length+index_length)/1024/1024) AS mb
FROM information_schema.tables WHERE table_schema NOT IN
('mysql','sys','information_schema','performance_schema');
```

✓ **You should see:** A list of tables with engine and size in MB.

2

Find tables without primary keys

Where: mysql client on the source

Tables without a primary key replicate slowly. Add a primary key to each one found.

```
SELECT t.table_schema, t.table_name FROM information_schema.tables t
LEFT JOIN information_schema.table_constraints c ON c.table_schema=t.table_schema AND
c.table_name=t.table_name AND c.constraint_type='PRIMARY KEY'
WHERE c.constraint_name IS NULL AND t.table_type='BASE TABLE' AND t.table_schema NOT IN
('mysql','sys','information_schema','performance_schema');
```

✓ **You should see:** No rows, once every table has a primary key.

3 Prepare the source and the target

3 steps

1

Turn on binary logging

Where: The source server's my.cnf (Linux) or my.ini (Windows), in the [mysqld] section

Add the two lines below, then restart MySQL in your maintenance window.

```
[mysqld]
log-bin=mysql-bin
server-id=1
```

✓ **You should see:** SHOW MASTER STATUS (or SHOW BINARY LOG STATUS on MySQL 8.4) returns a file name and position.

2

Create the replication user

Where: mysql client on the source

Create an account only the managed database can use for replication.

```
CREATE USER 'repl_user'@'%' IDENTIFIED BY 'a-long-random-password';
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'%';
```

✓ **You should see:** The user exists with the two privileges.

3

Create the managed database

Where: AWS console > RDS > Create database > MySQL, or Azure portal > Azure Database for MySQL flexible servers > Create

Choose the same major MySQL version as the source. On AWS, leave Multi-AZ, backup retention and read replicas off until after the import.

✓ **You should see:** The new database is available and you can connect with the admin user.

4 Copy the data and start replication

4 steps

1

Take a consistent dump

Where: A shell on a machine that can reach the source

Use `--single-transaction` so the dump is consistent without locking, and `--source-data=2` (MySQL 8.0.26 and later; `--master-data=2` before that) to record the binary log position in the file. Avoid schema changes during the dump. See also export and import a MySQL database.

```
mysqldump --databases appdb --single-transaction --order-by-primary --source-data=2 --routines=0
--triggers=0 --events=0 -h source.example.com -u admin -p | gzip > appdb.sql.gz
```

✓ **You should see:** A compressed dump whose header contains a CHANGE MASTER or CHANGE REPLICATION SOURCE line with the log file and position.

2

Load into the managed database

Where: A shell on a machine in the same region as the target

Load the dump, then create users, grants, routines, triggers and events from your scripts.

```
gunzip < appdb.sql.gz | mysql -h target-endpoint -u admin -p
```

✓ **You should see:** All tables exist on the target.

3

Start replication on Amazon RDS

Where: mysql client connected to the RDS instance

Use the log file and position from the dump header. On MySQL 8.0 and earlier the procedure is `mysql.rds_set_external_master`.

```
CALL mysql.rds_set_external_source ('source.example.com', 3306, 'repl_user', 'a-long-random-password', 'mysql-bin.000031', 107, 1);
CALL mysql.rds_start_replication;
SHOW REPLICA STATUS\G
```

✓ **You should see:** Replica IO and SQL threads are running, and Seconds_Behind_Source falls to 0.

4

Or start data-in replication on Azure

Where: mysql client connected to the Azure Database for MySQL flexible server

Microsoft sets the source with its `mysql.az_replication_change_master` procedure (include the CA certificate in `master_ssl_ca` if the source uses SSL), then starts it with `mysql.az_replication_start`. Follow Microsoft's data-in replication how-to for the exact parameters.

✓ **You should see:** Replication status shows running with lag falling to 0.

5 Cut over and finish

3 steps

1

Stop the application and let replication catch up

Where: Application servers, then the mysql client on the target

Stop the application or make the source read-only, then check lag is 0 on the target. Follow the cutover runsheet.

```
SET GLOBAL read_only = ON; -- on the source
SHOW REPLICA STATUS\G       -- on the target
```

✓ **You should see:** Lag is 0 and no new writes reach the source.

2

Stop replication and switch the application

Where: mysql client on the target, then the application's database settings

On RDS run `CALL mysql.rds_stop_replication;` then `CALL mysql.rds_reset_external_source;`. On Azure stop and remove data-in replication. Change the application's host to the managed database endpoint and start it.

✓ **You should see:** The application works against the new database.

3

Turn on protection

Where: AWS RDS console > Modify, or Azure portal > the server > High availability and Backup

Enable automated backups and Multi-AZ or zone-redundant high availability, and remove the source firewall rule for replication.

✓ **You should see:** Backups show as enabled and a first backup completes.

WORTH KNOWING

What can go wrong, and how do you avoid it?

The detail behind the plan: options compared, and what experienced engineers check first. Tap any item.

Compare all the ways to do this

3 options –

Dump, load, then binlog replication

MEDIUM EFFORT

Best for: Production databases that need a short cutover

+ Downtime limited to the final catch-up
+ Works on AWS and Azure
+ Old server stays intact
– Needs binary logging and a restart on the source
– Users and routines recreated by hand

Read more →

Dump and import only

LOW EFFORT

Best for: Small databases where an outage is acceptable

+ Simple
+ No replication setup
– Application down for the whole dump and load
– Slow for large databases

Read more →

AWS DMS or Azure Database Migration Service

MEDIUM EFFORT

Best for: Teams that want a managed, repeatable migration process

+ Guided setup
+ Repeatable runs
– Schema still prepared separately
– Another service to configure

Read more →

OUR ADVICE

For anything in production, load a dump and run binlog replication until cutover. Use a plain dump and import only for small databases where the outage is acceptable.

Replication keeps downtime short

SAVES TIME

AWS and Microsoft both describe loading a copy and then replicating from the binary log, so only the final catch-up needs the application stopped.

WHAT TO DO

Use replication for anything that cannot be down for the length of a full dump and load.

AWS: Importing data to an Amazon RDS for MySQL database with reduced downtime

Users and permissions are not replicated

REDUCES RISK

Microsoft notes the mysql system database and account changes are not replicated. AWS has you create routines, triggers and events manually on RDS.

WHAT TO DO

Script users, grants and stored code from the source, apply them to the target, and freeze account changes on the source until cutover.

Microsoft Learn: Data-in replication in Azure Database for MySQL Flexible Server

Binary logging adds load on the source

PERFORMANCE

Microsoft notes binlog replication has high CPU and extra storage requirements on the source.

WHAT TO DO

Check free disk and CPU on the source, and keep binary logs only as long as needed.

Microsoft Learn: Migrate MySQL on-premises to Azure Database for MySQL: migration methods

mysqldump is slow for big databases

SAVES TIME

Microsoft says mysqldump is not intended for fast migration of large amounts of data, and suggests mydumper and myloader, which can be 10 times faster.

WHAT TO DO

Time a trial dump and load; switch to mydumper and myloader on Linux if it is too slow.

Microsoft Learn: Migrate MySQL on-premises to Azure Database for MySQL: migration methods

Leave Multi-AZ and backups off during the load

SAVES TIME

AWS advises not enabling Multi-AZ, backup retention or read replicas until after the import.

WHAT TO DO

Turn them on straight after cutover, and confirm the first backup.

AWS: Importing data to an Amazon RDS for MySQL database with reduced downtime

FINISH SAFELY

How do you check it worked, or roll back?

Full validation checklist

5 checks

Run these before you declare the migration complete. Record the results in the data validation report.

Check	How to check	What good looks like
Row counts match	Run `SELECT COUNT(*)` on key tables on both servers just before cutover.	Counts are identical.
Table checksums match	Run `CHECKSUM TABLE appdb.orders;` on both servers for key tables.	Checksums are identical.
Users and stored code exist	Compare `SELECT user, host FROM mysql.user;` and the routines, triggers and events lists.	Every account and object from the source exists on the target.
Application works	The owner runs their test script against the new database.	All tests pass.

Check	How to check	What good looks like
Backups and high availability are on	Check the console for automated backups and Multi-AZ or high availability.	Both are enabled and a backup has completed.

Rollback plan

If it goes wrong

ROLL BACK IF

- The application fails against the new database and cannot be fixed in the window
- Data checks show missing or different rows
- Performance is unacceptable on the new service

POINT OF NO RETURN

Until the application writes to the new database, rollback is just pointing it back at the old server and making that writable again. After users write to the new database, going back means copying those changes back, so decide go or no-go before reopening the application.

1

Stop the application

Where: Application servers

Stop the application so nothing more is written to the new database.

✓ **You should see:** No connections to the new database.

2

Reopen the old server

Where: mysql client on the source

Turn read-only off and point the application back at the source host.

```
SET GLOBAL read_only = OFF;
```

✓ **You should see:** The application works against the old server.

3

Record and replay changes

Where: The new database

Export any rows written to the new database after cutover and apply them to the source.

✓ **You should see:** No transactions are lost.

Agree the triggers and who decides before the go / no-go meeting. Use the rollback plan template.

Common problems and fixes

5 errors

`Replication stops with a duplicate key or missing table error`

Cause: The dump and the replication start position do not match, or schema changes ran during the dump.

Fix: Reload from a fresh dump taken with --single-transaction and --source-data=2, and avoid schema changes during it.

`ERROR 1068 Multiple primary key defined (Azure)`

Cause: Generated invisible primary keys on MySQL 8.0 conflict with a table definition.

Fix: Follow Microsoft's fix for the table, or add real primary keys on the source.

`Replication cannot connect to the source`

Cause: Port 3306 is blocked, or the target is not publicly reachable for replication.

Fix: Open 3306 from the target's address only, and check the target's public access setting.

`Application cannot sign in after cutover`

Cause: Users and grants were not recreated on the target.

Fix: Create the users and grants from your scripts.

`Replication lag never reaches zero`

Cause: The source writes faster than the target applies, often due to tables without primary keys.

Fix: Add primary keys, size the target up during migration, or schedule cutover in a quiet period.

Search the full error library →

FAQ

Frequently asked questions

Can I move MySQL to RDS with almost no downtime?

Yes. Load a dump, then replicate from the source using mysql.rds_set_external_source and mysql.rds_start_replication until you cut over.

Does Azure Database for MySQL support replication from my server?

Yes, data-in replication, from MySQL 5.7 or later with InnoDB tables.

Are users and permissions copied? –

No. The mysql system database and account changes are not replicated, so recreate users and grants yourself.

Should I use mysqldump or mydumper? –

mysqldump is fine for small databases. Microsoft suggests mydumper and myloader for large ones.

When should I turn on Multi-AZ? –

After the import and cutover, as AWS advises.

SOURCES

Official sources

5 official pages this guide is based on –

Vendors change limits and features, so confirm details on the source before you rely on them. Last checked 4 Nov 2026.

- AWS: Importing data into an Amazon RDS for MySQL DB instance
- AWS: Importing data to an Amazon RDS for MySQL database with reduced downtime
- AWS: Configuring binary log file position replication with an external source instance
- Microsoft Learn: Migrate MySQL on-premises to Azure Database for MySQL: migrat...
- Microsoft Learn: Data-in replication in Azure Database for MySQL Flexible Server