Choosing a migration tool for rehosting databases

AWS Prescriptive Guidance

AWS Prescriptive Guidance: Choosing a migration tool for rehosting databases

Table of Contents

Choosing a migration tool for rehosting databases

Mike Kuznetsov and Harpreet Virk, Amazon Web Services

October 2025 (document history)

When you plan to migrate your large workloads to AWS, we recommend that you follow AWS guidance and split your migration process into three stages: assess, mobilize, and migrate. The migration journey involves many factors, including scope ("what?"), strategy ("why?"), and timeline ("when?"), as discussed in AWS large-migration strategy and best practices. Each workload you choose to migrate might follow a different migration strategy, as defined in seven common strategies (7 Rs). Most workloads follow the rehost scenario for lift-and-shift migrations. After you select a strategy, you can address the question "how?", which focuses on at least three aspects of migration (people, technology, and processes).

This guide is for anyone who is planning to migrate their on-premises workloads to the AWS Cloud, including IT and business executives, program and project managers, product owners, and operations and infrastructure managers.

This guide helps you determine your high-level migration approach but doesn't cover detailed implementation strategies for specific database engines or applications. For these detailed recommendations, see the Resources section, which points to various workload-specific and database engine-specific guides.

The focus of this guide is on the rehost migration path, which involves moving an application to the cloud without making any changes to take advantage of cloud capabilities. For example, migrating your on-premises Microsoft SQL Server database to SQL Server on an Amazon Elastic Compute Cloud (Amazon EC2) instance in the AWS Cloud is a rehosting strategy. More specifically, the guide discusses tools best suited for lift-and-shift migrations of workloads that have databases included in their scope, and factors to consider when selecting a particular service for the migration. It answers questions such as the following:

- Which service best supports database migrations?
- Could the service used for non-database servers be used for database servers as well, or should database servers be treated differently?
- What if my rehost migration turns into a mixed approach of rehosting and replatforming?

In this guide:

Migration scenarios

Migration scenarios can be differentiated based on scale (how many servers or applications are involved), migration time (how fast it must be done), and length of downtime allowed during cutover. These considerations affect the choice of AWS services to best support the migration.

- A large-scale migration (such as a full data center exit) includes multiple business units, and many applications and their databases. These typically have a tight time frame or a fixed deadline (such as completion before the closure of the data center) but less strict downtime limitations, which allow for longer cutover windows.
- A large but less complex migration of a business unit or an enterprise group might also involve multiple applications. However, these might have a more flexible timeline and sometimes stricter downtime requirements (shorter cutover windows).
- The migration of a single, large, business-critical application usually includes a highly intensive database server or a cluster, such as an online transaction processing (OLTP) database that handles a large number of transactions. These migrations typically require minimal or near zero downtime during the cutover window.

The migration requirements for these use cases range from moving at the fastest possible pace, which is generally associated with the most downtime, to maximum granularity and the closest attention to details, which generally support the least possible downtime during the migration. Most migrations represent some combination of these factors.

Migration waves, servers, and databases

Migration projects start with the following list of open questions, which are common across the three scenarios described in the previous section:

- How can we build migration waves?
- How can we group servers inside each wave?
- Should database servers be migrated before application servers or with them?
- Which tool should we use for database migrations?

However, to address these questions, we need to clarify some definitions first. This section of the guide focuses on the term *database* and what it means in the context of migration waves. This definition is important, because the understanding of that term could change the overall migration approach for particular migration waves or even change migration waves by shifting servers between different waves.

Does the term *database* describe a server that runs the DBMS software or the logical database entry point? Does it refer to one database out of several on the same server or a cluster of servers? Depending on the context, a database could refer to either option. Database administrators (DBAs) usually consider logical databases, not physical servers. However, in the context of migration, especially large-scale, lift-and-shift migrations, a database usually corresponds to a physical server or a cluster of servers.

The migration of a database must be aligned and associated with the application it works with, because the logical database is always a part of the application and its dependencies. However, the physical location of that logical database could vary. For example, it could be located on:

- A standalone physical server, without any other databases present.
- A standalone physical server colocated with other logical databases.
- A cluster of physical servers, either as a single logical database or as part of a larger set of databases that serve other applications.

Forming the dependency mapping for migration waves is complicated if there's no clear one-to-one dependency between the application and the database servers, especially when multiple logical databases from different applications are colocated on the same physical server. Such

database systems require a replatforming approach that involves a decomposition or consolidation of databases, which complicates large-scale lift-and-shift migration efforts.

This is where database migration tools (such as native tools from the database engine's vendors or third parties) can help. These tools work on a logical database level instead of the block-level replication approach of AWS Transform MGN, and can move the data or databases on a logical level between physical servers and locations. These native database migration tools can help you consolidate logical databases on a single physical server. They can also do the opposite: They can decompose logical databases between various physical database servers, to align them with different applications and spread them across different migration waves.

Migration with AWS Application Migration Service

Most large lift-and-shift migrations to AWS use AWS Transform MGN. This service works on a physical level by moving data that is stored on any directly attached block storage device (such as a hard drive or SAN drive) to the corresponding Amazon Elastic Block Store (Amazon EBS) storage device on AWS. It essentially implements a traditional backup/restore procedure (by cloning the hard drives), but also provides a recovery point objective (RPO) of near seconds and a recovery time objective (RTO) of minutes by achieving continuous data protection (CDP) synchronization mode between the source systems and the target storage devices in the staging area.

This block-level replication method doesn't support network-attached storage (NAS), shared drives such as Network File System (NFS) shares, or Common Internet File System (CIFS) / Server Message Block (SMB) shares. It supports only block-level storage that's directly attached to the migrated system at the time of migration. (For more information, see the MGN FAQ on SAN/NAS support.) This limits the applicability of replication through MGN for most clustered systems, because the majority of clusters rely on shared storage of various implementations. For more information, see the *Advantages* and *Disadvantages* sections on this page.

The block-level replication method requires you to install an AWS Replication Agent at the operating system (OS) level, and that agent supports only x86 platforms that are based on the Windows or Linux operating system (see Operating systems supported by MGN). Non-x86 platforms are out of scope for this migration method. These include ARM, RISC/CISC systems, PowerPC variations, IBM systems such as pSeries, iSeries, zSeries, and their respective operating systems such as AIX, HP-UX, Solaris, Linux for PowerPC, zLinux on mainframes, and other non-x86 architectures.

The AWS Replication Agent works at the level of a virtual file system device driver* in the OS stack, capturing any data blocks to be written to the underlying block-level devices (including drives, hard drives, and directly attached SAN devices), as explained on the MGN FAQ page under these questions:

- What does the AWS Replication Agent do?
- Does the AWS Replication Agent cache any data to disk?
- What is there to note regarding SAN/NAS support?

* See the definitions of file system, virtual file system, and device driver in Wikipedia.

Advantages

For lift-and-shift migrations of any scale, MGN has many benefits:

- The replication is easy to set up (it doesn't require a steep learning curve).
- It's fast to scale, by rolling out agents on the source machines.
- You can use the Cloud Migration Factory to automate most manual tasks, manage multiple machines, and orchestrate migration waves.
- It supports a wide range of x86 operating system architectures.
- It supports any type of source environment (on-premises data center, any other cloud, or another AWS Region).
- It's application-agnostic, so it supports any application running on the source servers.
- No license or purchase is required. AWS offers pay-as-you-go pricing, so you pay for the service only as long as you use it, without long-term contracts or complex licensing. MGN provides a free 90-day period per server, which is enough for most migrations. For details, see AWS Transform MGN pricing on the AWS website.

Disadvantages

When you use block-level replication tools such as MGN, you might encounter the following corner cases and factors to consider:

- MGN doesn't support mounted shared file systems or shared storage devices such as NAS, including CIFS/SMB and NFS file systems. For more information about replication methods for NAS or shared file systems, see MGN replication agent to migrate NFS Share (AWS re:Post article) and Migrate shared file systems in an AWS large migration (AWS Prescriptive Guidance pattern).
- MGN doesn't support most clustered file server or database cluster configurations with shared storage because of limitations in how that shared storage is represented to the OS level through the device driver layers. For example, Microsoft SQL Server clusters with the Storage Space Direct (S2D) option are not supported. However, you still can use MGN to replicate other types of clustered systems with shared block storage, such as shared storage in Failover Cluster Instance (FCI) configurations in Windows Server Failover Cluster (WSFC), as described in the blog post Migrating your Microsoft Windows clusters to AWS using CloudEndure Migration, storage exposed from external SAN arrays (iSCSI connections), and some Microsoft SQL Server Always On availability group (AAG) clusters in specific corner cases. In general, MGN supports replication

of a block-level device from a server where the storage device is fully present on the server during the migration. (The AWS Replication Agent must be installed on the server, and the device must be visible to the agent for replication.) However, all of these scenarios require very specific procedures and result in longer cutover windows.

- Systems that have a high rate of data changes (such as OLTP databases) might have higher performance or network requirements, which complicate migration efforts.
- Network bandwidth must be sufficient for the amount of data to be transferred within the planned migration and cutover time window. This is because MGN doesn't provide an offline shipping option, unlike AWS Snowball.
- Migration requires a small downtime window, within an RTO of minutes. MGN uses EBS snapshots as part of the migration process, and new EBS disks that are created from such snapshots initially have worse performance. With some database read patterns, this might increase the effective cutover window until the migrated database is at full performance.

Best-fit scenarios

MGN covers almost any lift-and-shift migration fully, with few disadvantages, as discussed in the previous two sections. This tool can handle most of the corner cases, such as database clusters, as long as the longer cutover window required for these systems satisfies your migration requirements.

The following sections cover two scenarios in more depth:

- Large-scale migration with multiple migration waves
- Single application migration that requires minimal downtime

Large-scale migration with multiple migration waves

An example of a large-scale migration is a data center exit that is characterized by size and speed requirements. It also usually involves a specific time constraint, such as the end of the lease contract for the data center. In this case, the scale dictates the approach. The migration waves are determined and grouped by applications, including databases, and each wave has a planned preparation, migration, and final cutover phase with specific downtime requirements.

Inside each migration wave, the database servers are best moved at scale by using MGN block-level replication, except for a few exceptions for the following special cases that might require a separate migration approach:

- Most clustered database systems
- Database systems where the rate of change exceeds available network throughput (which might result in a replication lag)

For each special case, consider the tradeoff between your downtime requirements and the level of effort involved in using another migration tool. In some cases, it's much easier to use the same migration approach for all systems instead of using separate tools and creating different processes for specific systems. If MGN doesn't support the downtime requirements for a specific system, we recommend that you use one of the methods described in the Migration with native database tools section instead.

Single application migration

The single application scenario represents a granular approach for migrating a single business-critical application that requires minimum or near-zero downtime during migration, or a few such applications. The scope of the migration might vary, depending on business criticality and downtime requirements, in contrast to the speed and scale requirements of the previous (large-scale migration) scenario. If the application allows for downtime within the RTO of MGN, it should be handled in the same way as any large-scale migration described earlier.

However, in cases where the cutover time must be as close to the minimum as possible—for example, when the migrated database has read patterns that require a long time to operate at full performance and the cutover windows have to remain small—you should consider a more detailed and granular approach. In general, that approach involves additional steps and requirements, which require more effort and time. One of the steps is to separate the database migration from the application server migration and to use the database migration tools described in the next section to keep the source and the target databases in sync. You can use various methods to achieve synchronization. Each method has advantages and disadvantages, and involves different tradeoffs between the amount of downtime and the complexity of synchronization. Nevertheless, keeping the database in sync between the source and target environments enables you to unlink the database layer from the application layer. Assuming that application servers don't store persistent data locally but pass everything to the database layer, synchronization also removes downtime restrictions from the application layer.

Decoupling the database and application layers enables you to migrate application servers by using MGN as in the previous scenario. Target application servers can be started in the target environment while source systems are still working, processing the data, and storing it in the database layer. Because the database layer is already in sync with the target, cutover time is minimal—it only involves switching networks and redirecting customer requests from the old source system to the target environment. You can use multiple methods to do this, following the guidance for blue/green deployments, typically by switching DNS endpoints, using weighted DNS zones, using AWS Global Accelerator to reduce Time to Live (TTL) DNS propagation delays, and similar methods.

Migration with native database tools and AWS DMS

Many DBAs are familiar with a wide range of tools that handle database migration and replication. These tools are typically offered by database engine vendors and third-party companies, and they work on the logical level of the specific database engine, unlike the fully application-agnostic, block-level replication approach offered by MGN.

Here's a list of these tools, ranging from the simplest to more complex approaches:

- **Full backup/restore** is a familiar, well-known, and easy to use process for IT staff. The method depends on the type of the database engine. The process usually moves multiple logical databases that are colocated on the same database server, and can also be used to restore the databases into a managed service such as Amazon Relational Database Service (Amazon RDS). Backup/restore is the simplest method but requires a much longer cutover window compared with the other options, due to the size of the backups and the time it would require to create, copy, and restore them on the target database. For more information about this approach, see Native SQL Server backup/restore and Oracle RMAN on the AWS Prescriptive Guidance website.
- **Logical backup or export** is another method that takes a copy of a full or partial logical database. This native database engine tool lets you decompose a large database server to migrate selected databases that are associated with a particular application. It provides more control than full backup/restore on what to migrate, and also supports Amazon RDS as a target. However, this option also requires a longer cutover window for the same reasons as the previous method.
- **Native database high availability (HA) tools** include the Always On or distributed availability group clusters in Microsoft SQL Server and Oracle's Data Guard replications. This approach requires a major effort to set up across extended, cross-site HA clusters, and might cause some performance degradation because of the longer latency to achieve fully synchronous active-active deployments. However, this method provides the closest to near-zero downtime during the cutover.
- **Change data capture (CDC)** replication is supported by AWS Database Migration Service (AWS DMS) and native database replication tools such as Oracle GoldenGate, Qlik, and Talend. You can use these tools to copy a partial or complete database with the advantage of near-zero downtime, because they keep the target database in sync with the source database. You can also use this method with AWS Schema Conversion Tool (AWS SCT) and AWS DMS for heterogeneous migrations, to migrate and modernize your database at the same time.

- If network throughput is a bottleneck during your database migration, you can use AWS DMS in conjunction with AWS Snowball to migrate and modernize very large databases. For more information, see the blog post Enable large-scale database migrations with AWS DMS and AWS Snowball.

Advantages

Using database tools for migration has the following advantages over block-level replication methods:

- Some tools provide migration with minimal downtime. These include AWS DMS and native tools that support native HA clusters or CDC replication.
- You're able to use tools that are familiar to most DBAs to migrate your clustered databases.
- You can modernize the database as part of the migration workflow, and move into managed database services such as Amazon RDS or Amazon Aurora.
- You can take advantage of consolidation and decomposition (or partial database migrations), when moving from a monolithic infrastructure to microservices, splitting up a large database server or a cluster, or merging smaller databases into a larger instance or into an AWS service.

Disadvantages

Most of the benefits discussed in the previous section are outside a typical lift-and-shift migration scenario and fall under the replatform approach. Moreover, native database migration methods have some disadvantages in large-scale migrations, such as:

- Preparation – You have to pre-provision and configure the target infrastructure, database servers, and clusters fully before you can use any of the native database methods.
- Complexity – Some methods, such as full or logical backup/restore, have to be combined with another replication method to detect all the changes since the initial backup was created.
- Scalability – There's no simple automation framework available to roll these methods out to other database clusters and servers when you migrate at scale.

Decision matrix

Although each migration is unique and has its own challenges, limitations, and multiple factors to consider, there are common criteria that you can use to identify the most appropriate migration strategy and service for your use case. Identifying and prioritizing these factors help you narrow down the choice. Use the following table as a decision tree: Start from the most important factor for your use case and choose the best tool for your migration.

> **Note**
>
> The following table provides high-level directional factors to consider; it doesn't include an exhaustive list of criteria for a migration project. The purpose is to provide a generalized comparison of two vastly different data migration methods: block-level replication (provided by MGN) compared with logical data-level replication (provided by a multitude of native database migration tools). These two methods are applicable in many migration scenarios and can sometimes be used together, but they also have unique advantages that the table highlights.

Criteria	AWS Transform MGN	Database tools (native tools or AWS DMS)
Architecture	Physical (block level)	Logical, database engine level
Scale	Large-scale migration	Granular; scale limitations
Speed vs. complexity	Fast exit scenario; reduced complexity	Slower, more complex approach; requires more planning and testing
Timeline	Supports aggressive timeline	Requires additional effort and time
Migration type	Lift and shift as is (one to one only)	Replatforming or modernization with decomposition and consolidation options (one to many, many to one)

Pre-provisioning	Not required; automatic migration	Database and infrastructure provisioning required
Downtime	Downtime required, within RTO of minutes	Near-zero downtime possible but very expensive (through sync/async extended clusters, CDC replication, and similar methods)
Data change rate	Might have networking or performance limits	More options available
Limitations	Doesn't support most clustered systems;* supports only x86 platforms**	Native database tools support clustered databases and non-x86 platforms; AWS DMS covers most database engines

***** The block-level replication method doesn't support network-attached storage (NAS), shared drives such as NFS shares, or CIFS/SMB shares. It supports only block-level storage that's directly attached to the migrated system at the time of migration. (For more information, see the MGN FAQ on SAN/NAS support.) This limits the applicability of replication through MGN for most clustered systems, because the majority of clusters rely on shared storage of various implementations. For more information, see *Advantages* and *Disadvantages* in the Migration with AWS Transform MGN section earlier in this guide.

****** The block-level replication method requires you to install an AWS Replication Agent at the OS level, and that agent supports only x86 platforms that are based on the Windows or Linux operating system (see Operating systems supported by MGN). Non-x86 platforms are out of scope for this migration method. These include ARM, RISC/CISC systems, PowerPC variations, IBM systems such as pSeries, iSeries, zSeries, and their respective operating systems such as AIX, HP-UX, Solaris, Linux for PowerPC, zLinux on mainframes, and other non-x86 architectures.

Resources

AWS services

- Migrate and modernize with AWS
- AWS Transform MGN documentation
- AWS DMS documentation

Migration strategy

- Strategy and best practices for AWS large migrations
- Mobilize your organization to accelerate large-scale migrations
- About the migration strategies (in *Guide for AWS large migrations*)

Database migration and native database tools

- Migration strategy for relational databases
- Migrating Microsoft SQL Server databases to the AWS Cloud
- Migrating Oracle databases to the AWS Cloud
- Database decomposition on AWS

Blog posts

- Accelerating your migration to AWS (AWS Architecture Blog)
- 6 strategies for migrating applications to the cloud (AWS Cloud Enterprise Strategy Blog)
- Database migration—What do you need to know before you start? (AWS Database Blog)

Document history

The following table describes significant changes to this guide. If you want to be notified about future updates, you can subscribe to an RSS feed.

Change	Description	Date
Updates	In the sections on AWS Transform MGN and the decision matrix, clarified the limitations of block-level replication.	October 14, 2025
Initial publication	—	June 22, 2022