AWS Prescriptive Guidance

Migrating Microsoft SQL Server databases to the AWS Cloud

AWS Prescriptive Guidance: Migrating Microsoft SQL Server databases to the AWS Cloud

Table of Contents

Migrating Microsoft SQL Server databases to the AWS Cloud

Sagar Patel, Senior Database Specialty Architect, AWS Professional Services

October 2020 (last update (p. 52): March 2021)

Amazon Web Services (AWS) provides a comprehensive set of services and tools for deploying Microsoft SQL Server databases on the reliable and secure AWS Cloud infrastructure. Benefits of running SQL Server on AWS include cost savings, scalability, high availability and disaster recovery, better performance, and ease of management. For more information, see Learn why AWS is the best cloud to run Microsoft Windows Server and SQL Server workloads on the AWS Compute blog.

This guide describes the options available for migrating SQL Server databases from on premises to the AWS Cloud, to Amazon Relational Database Service (Amazon RDS), Amazon Elastic Compute Cloud (Amazon EC2), or VMware Cloud on AWS. It dives into the best practices and recommendations for using these migration options. It also provides information about how to set up a high availability and disaster recovery solution between an on-premises SQL Server environment and AWS, using native SQL Server features like log shipping, replication, and Always On availability groups.

This guide is for program or project managers, product owners, database administrators, database engineers, and operations or infrastructure managers who are planning to migrate their on-premises SQL Server databases to AWS.

Overview

Before you migrate your SQL Server databases to AWS, you should understand and evaluate your migration strategy by using the framework discussed in Migration strategy for relational databases.

The first step is to perform an analysis of your application and SQL Server database workloads by understanding the complexity, compatibility, and cost of migration. Here are some of the top points you should consider when you plan to migrate:

- **Database size** – Check the current size and overall capacity growth of your database. For example, if you're planning to migrate your SQL Server database to Amazon RDS, you can create Amazon RDS DB instances with up to 16 TiB of storage. You can request more storage by opening a support ticket with AWS Support. For the latest information, see Amazon RDS DB instance storage in the Amazon RDS documentation.
- **IOPS** – Determine the IOPS and throughput of your databases. If you're planning to migrate to Amazon RDS, consider the I/O performance of Amazon RDS DB instances.
- **Dependencies** – Check current database dependencies. If your database is dependent on other databases, you can either migrate them together or create dependencies after you migrate your main database.

  Inventory all SQL Server dependencies. Find out which web servers (for example, reporting servers or business intelligence servers) interface with SQL Server. When it's time to migrate, this information helps you determine what will be impacted and how you can minimize the impact.
- **Compliance** – Review your current architecture and auditing or compliance needs, to make sure you can satisfy these requirements after moving to Amazon RDS or Amazon EC2.

- **HA/DR** – Do you need high availability (HA) and automated failover capabilities? If you are running a production workload, high availability and disaster recovery (DR) are recommended best practices.

  Understand your HA/DR requirements to determine whether you need a multi-Region architecture. If so, migrate your SQL Server database to Amazon EC2. Amazon RDS doesn't support a multi-Region configuration.
- **Version support** – Check the version and edition of your SQL Server software if you're planning to move to Amazon RDS for SQL Server (see currently supported versions).
- **Network connectivity** – Check the network connectivity between your on-premises environment and AWS, to make sure that it provides enough bandwidth for fast transfers of data between on premises and AWS.
- **Migration downtime** – Determine the amount of downtime available for migration so you can plan your migration approach and decide whether you want to use online or offline migration.
- **RTO, RPO, SLA requirements** – Identify your recovery time objective (RTO), recovery point objective (RPO), and service-level agreement (SLA) requirements for your existing database workloads.
- **Licensing** – Understand your licensing options. You can choose license-included options on Amazon EC2 and Amazon RDS, or choose to bring your own license (BYOL) on Amazon EC2.
- **Feature support** – Identify the database features and functionality that your application uses, whether it was developed in-house or it's commercial-off-the-shelf (COTS) software. This information can help you determine whether you can reduce your licensing costs by switching from SQL Server Enterprise edition to Standard edition. However, review Standard edition resource limitations before you switch. For example, Standard edition supports only 128 GB of RAM.

  Does your workload fit within the features and capabilities offered by Amazon RDS for SQL Server? For more information, see SQL Server features on Amazon RDS. If you need features that aren't supported, migrating to Amazon EC2 is an option.

SQL Server database migration strategies

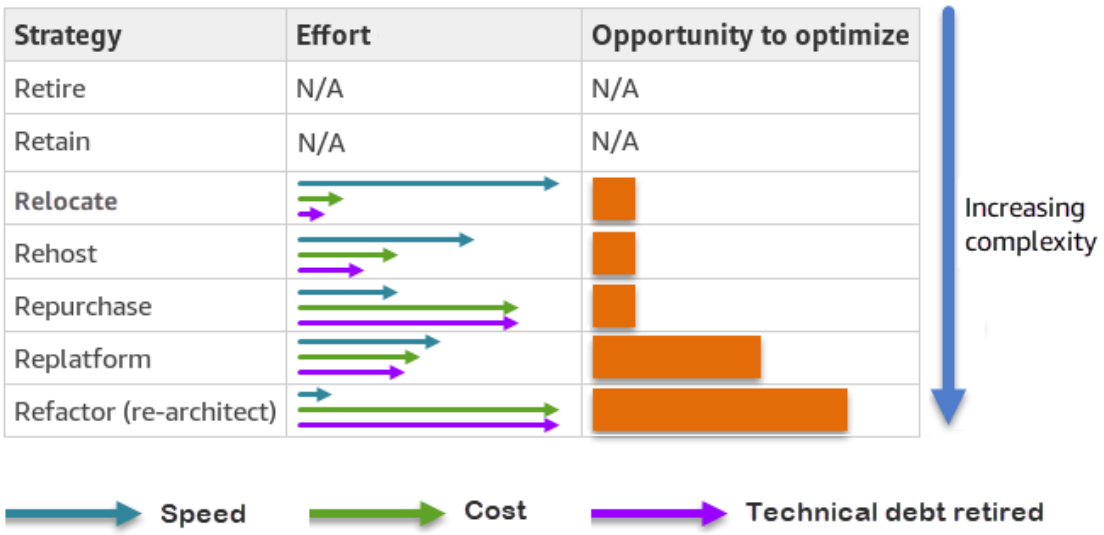
At a high level, there are two options for migrating a SQL Server database from on premises to the AWS Cloud: either stay on SQL Server (*homogenous migration (p. 16)*) or move off SQL Server (*heterogenous migration (p. 30)*). In a homogenous migration, you don't change the database engine. That is, your target database is also a SQL Server database. In a heterogenous migration, you switch your SQL Server databases either to an open-source database engine such as MySQL, PostgreSQL, or MariaDB, or to an AWS Cloud-native database such as Amazon Aurora, Amazon DynamoDB, or Amazon Redshift.

There are three common strategies for migrating your SQL Server databases to AWS: rehost, replatform, and re-architect (refactor). These are part of the 7 Rs of application migration strategies and described in the following table.

Strategy	Type	When to choose	Example
Rehost	Homogeneous	You want to migrate your SQL Server database as is, with or without changing the operating system, database software, or configuration.	SQL Server to Amazon EC2 (Browse rehost patterns)
Replatform	Homogeneous	You want to reduce the time you spend managing database instances by using a fully managed database offering.	SQL Server to Amazon RDS for SQL Server (Browse Replatform patterns)
Re-architect (refactor)	Heterogeneous	You want to restructure, rewrite, and re-architect your database and application to take advantage of open-source and cloud-native database features.	SQL Server to Amazon Aurora PostgreSQL, MySQL, or MariaDB Browse Re-architect patterns)

Choosing the right migration strategy

Choosing the correct strategy depends on your business requirements, resource constraints, migration timeframe, and cost considerations. The following diagram shows the effort and complexity involved in migrations, including all seven strategies.



Refactoring your SQL Server database and migrating to an open-source or AWS Cloud-native database such as Aurora PostgreSQL or MySQL can help you modernize and optimize your database. By moving to an open-source database, you can avoid expensive licenses (resulting in lower costs), vendor lock-in periods, and audits. However, depending on the complexity of your workload, refactoring your SQL Server database can be a complicated, time-consuming, and resource-intensive effort. AWS can help you assess the complexity of your workload and recommend migration strategies and tools. For more information, see the AWS WQF (p. 32) section later in this guide.

To reduce complexity, instead of migrating your database in a single step, you might consider a phased approach. In the first phase, you can focus on core database functionality. In the next phase, you can integrate additional AWS services into your cloud environment, to reduce costs, and to optimize performance, productivity, and compliance. For example, if your goal is to replace your on-premises SQL Server database with Aurora MySQL, you might consider rehosting your database on Amazon EC2 or replatforming your database on Amazon RDS for SQL Server in the first phase, and then refactor to Aurora MySQL in a subsequent phase. This approach helps reduce costs, resources, and risks during the migration phase and focuses on optimization and modernization in the second phase.

Online and offline migration

You can use two methods to migrate your SQL Server database from an on-premises or another cloud environment to the AWS Cloud, based on your migration timeline and how much downtime you can allow: offline migration or online migration.

- **Offline migration:** This method is used when your application can afford a planned downtime. In offline migration, the source database is offline during the migration period. While the source database is offline, it is migrated over to the target database on AWS. After the migration is complete, validation and verification checks are performed to ensure data consistency with the source database. When the database passes all validation checks, you perform a cutover to AWS by connecting your application to the target database on AWS.
- **Online migration:** This method is used when your application requires near zero to minimal downtime. In online migration, the source database is migrated in multiple steps to AWS. In the initial steps, the data in the source database is copied to the target database while the source database is still running. In subsequent steps, all changes from the source database are propagated to the target database. When the source and target databases are in sync, they are ready for cutover. During the

cutover, the application switches its connections over to the target database on AWS, leaving no connections to the source database. You can use AWS Database Migration Service (AWS DMS) or tools available from AWS Marketplace (such as Attunity) to synchronize the source and target databases.

SQL Server database migration methods

There are various methods to migrate your SQL Server databases to AWS. You can choose these methods based on your assessment and requirements. This section describes some of the most common methods, which are summarized in the following table. Detailed discussions of some of these methods are included in the sections on Amazon EC2 and Amazon RDS later in this guide.

Migration method	Target	Features and limitations	More information
Native backup and restore	Amazon EC2 Amazon RDS	• Can be applied to one or many databases at one time • Requires downtime • Supports all database sizes	Native SQL Server backup/ restore (p. 9) section
Log shipping	Amazon EC2 Amazon RDS	• Applied per database • Can be delayed	Log shipping (p. 10) section
Database mirroring	Amazon EC2	• Applied per database • Can be synchronous or asynchronous, based on the SQL Server edition • Secondary database isn't readable; it acts as a standby • Supports both automatic and manual failover	Database mirroring (p. 11) section
Always On availability groups	Amazon EC2	• Applied to a set of user databases • Can be synchronous or asynchronous • Secondary database is readable (SQL Server Enterprise edition only) • Supports both automatic and manual failover • Failover can be initiated for multiple databases at a time, at the database group level	Always On availability groups (p. 11) section

Migration method	Target	Features and limitations	More information
Basic Always On availability groups	Amazon EC2	• Supported in SQL Server Standard edition • Applied to a single user database per availability group • Can be synchronous or asynchronous • Supports both automatic and manual failover • Failover can be initiated at the availability group level • Can be used as a hybrid environment between on premises and AWS	Not covered in this guide (see Basic Always On availability groups for a single database in the Microsoft documentation)
Distributed availability groups	Amazon EC2	• Can be used for multi-Region SQL Server deployments • Can fail over to a later version of SQL Server • Doesn't require Windows Server Failover Clustering (WSFC) to be extended to the target AWS environment • Can be used between Windows-based (source) and Linux-based (target) SQL Server databases • Can be used as a hybrid SQL Server deployment between on premises and AWS	Distributed availability groups (p. 12) section

Migration method	Target	Features and limitations	More information
Transactional replication	Amazon EC2 Amazon RDS	• Supports migration of a set of objects (tables, view, stored procedures) • Supports asynchronous replication with near real-time data • Subscriber database is readable • Requires close monitoring of SQL Server replication jobs that perform the replication	Transactional replication (p. 13) section
AWS Snowball Edge	Amazon EC2 Amazon RDS	• Supports very large databases (up to 80 TB) • Uses Amazon Simple Storage Service (Amazon S3) for storing and restoring data	Snowball Edge (p. 14) section
CloudEndure Migration	Amazon EC2	• Highly automated lift-and-shift solution • Agent-based, block-level replication	CloudEndure Migration (p. 14) section
AWS DMS	Amazon EC2 Amazon RDS Amazon Aurora	• Supports full load and CDC • Supports all database sizes	AWS DMS (p. 32) section
Bulk copy program (bcp)	Amazon EC2	• Supports small databases • Requires downtime • Schema is pre-created at the destination • Used for moving data, but not metadata	Not covered in this guide (see Importing and exporting SQL Server data using other methods, *Bulk copy* section in the Amazon RDS documentation)
Detach and attach	Amazon EC2	• No backup needed • Requires downtime • Involves stopping, detaching, copying files, and attaching to Amazon EC2	Not covered in this guide (see Database Detach and Attach in the Microsoft documentation)

Migration method	Target	Features and limitations	More information
Import/export	Amazon EC2	• Supports small databases • Requires downtime • Schema is pre-created at the destination • Used for moving data, but not metadata	Not covered in this guide (see Importing and exporting SQL Server data using other methods in the Amazon RDS documentation)

Native SQL Server backup/restore

Amazon RDS supports native backup and restore operations for Microsoft SQL Server databases using full and differential backup (.bak) files. It also supports differential restore and log restore options on an Amazon RDS for SQL Server DB instance or Amazon EC2 SQL Server instance, to minimize downtime for your application.

> **Note**
> You can perform full, differential, and log restore operations on Amazon RDS for SQL Server. However, you can perform only full and differential backup (not log backup) at this time.

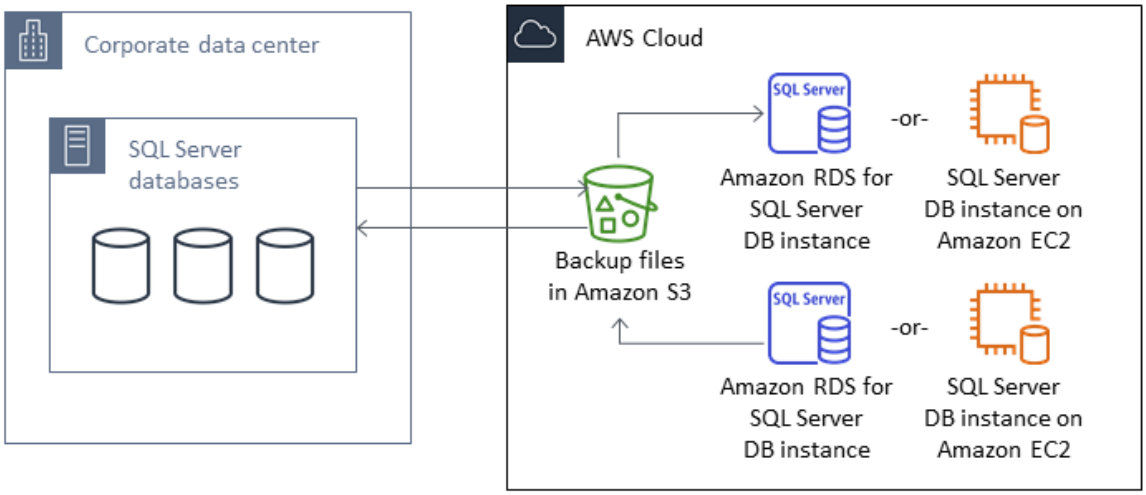
Using native .bak files is the simplest way to back up and restore SQL Server databases. You can use this method to migrate databases to or from Amazon RDS. You can back up and restore single databases instead of entire DB instances. You can also move databases between Amazon RDS for SQL Server DB instances.

When you use Amazon RDS, you can store and transfer backup files in Amazon Simple Storage Service (Amazon S3), for an added layer of protection for disaster recovery. For example:

- You can create a full backup of your database from your local server, copy it to an S3 bucket, and then restore it onto an existing Amazon RDS SQL Server DB instance.
- You can take backups from an Amazon RDS for SQL Server DB instance, store them in Amazon S3, and then restore them wherever you want.
- You can implement Amazon S3 Lifecycle configuration rules to archive or delete long-term backups.

Amazon RDS for SQL Server supports restoring SQL Server native backups onto SQL Server DB instances that have read replicas configured. This means that you don't have to remove the read replica before restoring the native backup file onto your Amazon RDS for SQL Server DB instance.

The following diagram shows the native SQL Server backup/restore process. You can use this process to back up and restore SQL Server databases to Amazon EC2 as well.



To set up native backup/restore using Amazon S3, see the Amazon RDS documentation.

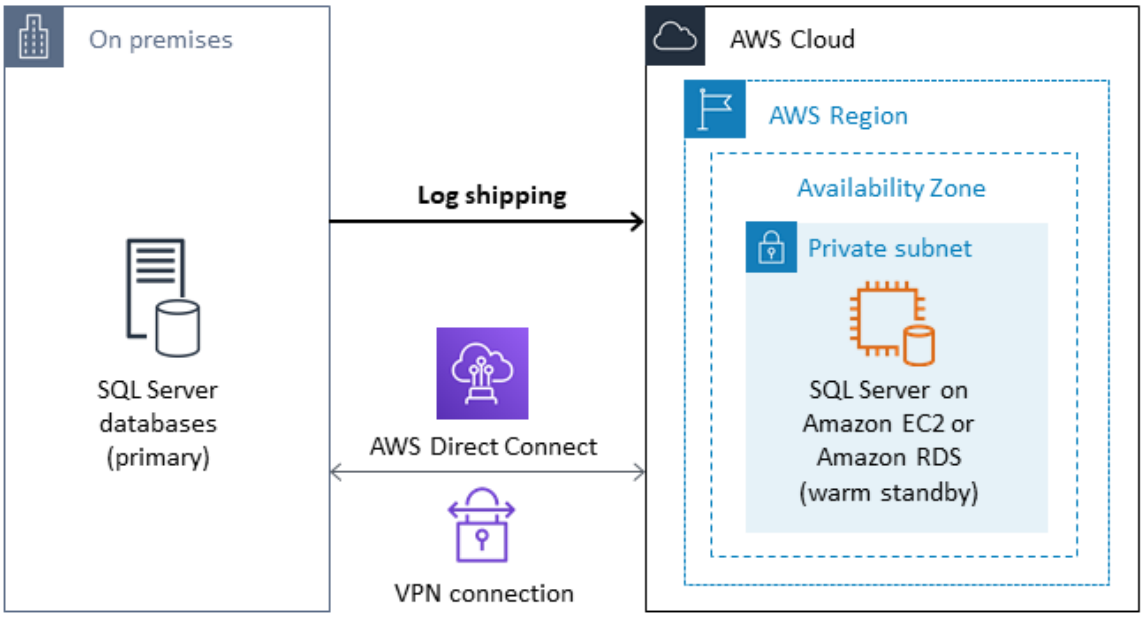
For limitations when using SQL Server native backup and restore, see Limitations and recommendations in the Amazon RDS documentation.

Log shipping

You can use log shipping to send transaction log backups from your primary, on-premises SQL Server database to one or more secondary (warm standby) SQL Server databases that are deployed on EC2 instances or Amazon RDS for SQL Server DB instances in the AWS Cloud. To set up log shipping on Amazon RDS for SQL Server, you have to use your own custom scripts.

In this scenario, you configure a warm standby SQL Server database on an EC2 instance or Amazon RDS for SQL Server DB instance, and send transaction log backups asynchronously between your on-premises database and the warm standby server in the AWS Cloud. The transaction log backups are then applied to the warm standby database. When all the logs have been applied, you can perform a manual failover and cut over to the cloud.

This option supports all versions and editions of SQL Server. After you have migrated the database to the AWS Cloud, you can add a secondary replica by using an Always On availability group for high availability and resiliency purposes.



For more information about using this method to achieve high availability, data protection, and disaster recovery for your SQL Server databases on Amazon EC2, see Log shipping (p. 23) in the *Amazon EC2 for SQL Server* section.

Database mirroring

You can use database mirroring to set up a hybrid cloud environment for your SQL Server databases. This option requires SQL Server Enterprise edition. In this scenario, your principal SQL Server database runs on premises, and you create a warm standby in the cloud. You replicate your data asynchronously, and perform a manual failover when you're ready for cutover. After you have migrated the database to the AWS Cloud, you can add a secondary replica by using an Always On availability group for high availability and resiliency purposes.

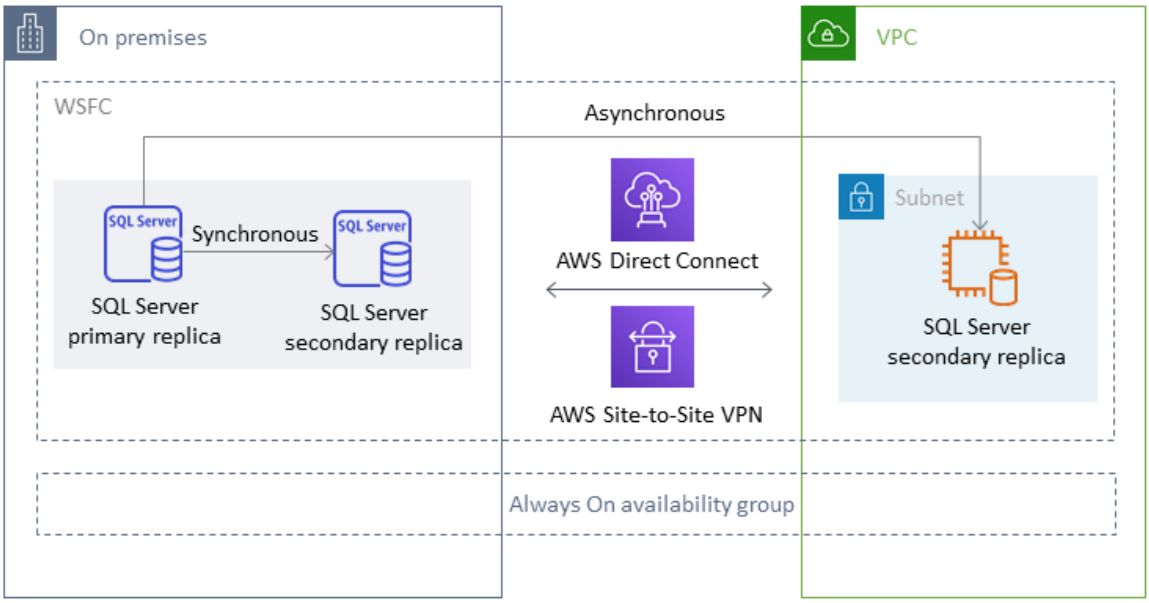
For more information about using this method to achieve high availability, data protection, and disaster recovery for your SQL Server databases on Amazon EC2, see Database mirroring (p. 24) in the *Amazon EC2 for SQL Server* section.

Always On availability groups

SQL Server Always On availability groups is an advanced, enterprise-level feature to provide high availability and disaster recovery solutions. This feature is available if you are using SQL Server 2012 and later versions. You can also use an Always On availability group to migrate your on-premises SQL Server databases to Amazon EC2 on AWS. This approach enables you to migrate your databases either with downtime or with minimal downtime.

If you have an existing on-premises deployment of SQL Server Always On availability groups, your primary replica and secondary replica will be synchronously replicating data within the availability group. So, to migrate your database to AWS Cloud, you can extend your Windows Server Failover Clustering (WSFC) cluster to the cloud. This can be temporary, just for migration purposes. You then create a secondary replica in the AWS Cloud and use asynchronous replication, as shown in the following

diagram. After the secondary replica is synchronized with the primary on-premises database, you can perform a manual failover whenever you are ready for cutover.



For more information about using this method to achieve high availability, data protection, and disaster recovery for your SQL Server databases on Amazon EC2, see Always On availability groups (p. 25) in the *Amazon EC2 for SQL Server* section.

Distributed availability groups

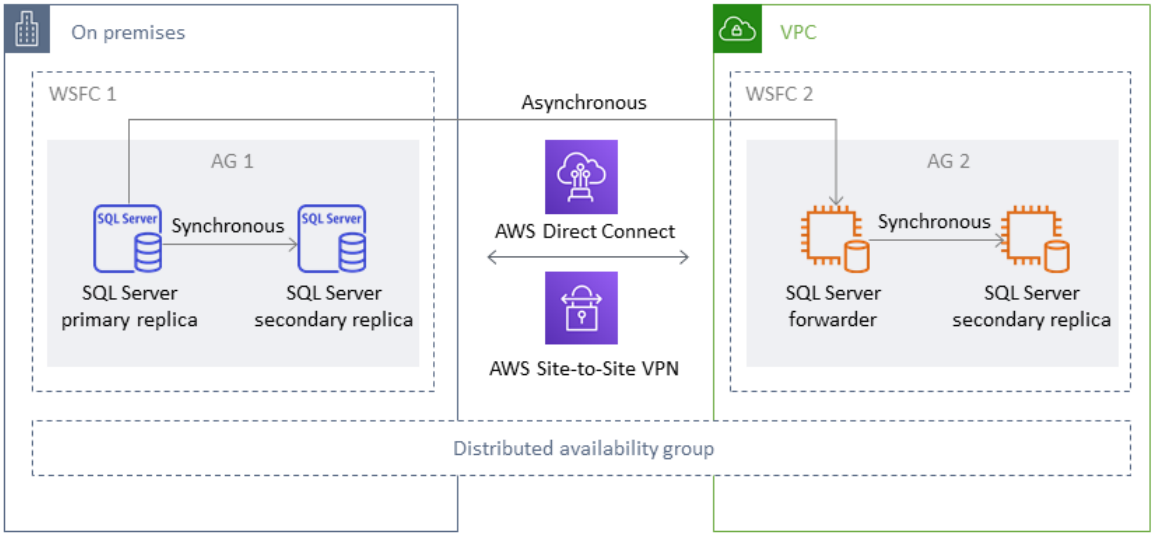
A distributed availability group spans two separate availability groups. You can think of it as an availability group of availability groups. The underlying availability groups are configured on two different WSFC clusters. The availability groups that participate in a distributed availability group do not need to share the same location. They can be physical or virtual, on premises or in the public cloud. The availability groups in a distributed availability group don't have to run the same version of SQL Server. The target DB instance can run a later version of SQL Server than the source DB instance.

A distributed availability group architecture gives you a flexible way to rehost a mission-critical SQL Server instance or database on AWS. It provides a hybrid solution for lifting and shifting (or lifting and transforming) your critical SQL Server databases on AWS.

Using a distributed availability group architecture is more efficient than extending existing on-premises WFSC clusters to AWS. Data is transferred only from the on-premises primary to one of the AWS replicas (the *forwarder*). The forwarder is responsible for sending data to other secondary read replicas in AWS.

In the following diagram, the first WSFC cluster (WSFC 1) is hosted on premises and has an on-premises availability group (AG 1). The second WSFC cluster (WSFC 2) is hosted on AWS and has an AWS availability group (AG 2). AWS Direct Connect is used as a dedicated network connection between the on-premises environment and AWS. The on-premises availability group (AG 1) has two replicas (*nodes*). The data transfer between the nodes is synchronous, with automatic failover. Similarly, the AWS availability group (AG 2) also has two replicas, and the data transfer between them is synchronous with automatic failover. The distributed availability group keeps the databases in sync in an asynchronous manner. Data is transferred from the SQL Server primary replica in AG 1 (which is on premises) to the primary replica (the forwarder) in AG 2 (which is on AWS). The forwarder is responsible for sending data to other read replicas on AWS and keeping them updated. After the on-premises and AWS databases are synchronized,

you can perform a manual failover of the distributed availability group to AWS. The AWS database becomes the primary database for read/write access from applications.



Note
At any given point of time, there is only one database that is available for write operations. You can use the remaining secondary replicas for read operations. To scale out your read workloads, you can add more read replicas in multiple Availability Zones on AWS.

For more information about distributed availability groups, see the Microsoft SQL Server documentation and How to architect a hybrid Microsoft SQL Server solution using distributed availability groups on the AWS Database blog.

Transactional replication

Transactional replication is a SQL Server technology that is used to replicate changes between two databases. These changes can include database objects like tables (primary key is required), stored procedures, views, and so on, as well as data. The replication process involves a *publisher* (the primary database that publishes data), a *subscriber* (a secondary database that receives replicated data), and a *distributor* (a server that stores metadata and transactions for transactional replication). You can use transactional replication for SQL Server on Amazon EC2 and Amazon RDS for SQL Server DB instances.

Transactional replication creates a snapshot of the objects and data in your on-premises (publication) database and sends it to the subscriber database. After the snapshot is applied to the subscriber, all subsequent data changes and schema modifications made at the publisher are sent over to the subscriber as they occur. The data changes are then continuously applied to the subscriber in the same order as they occurred at the publisher.

After synchronization is complete, you perform validation on the target SQL Server DB instance. When the two databases are in sync, you stop stop the activity on the on-premises database, ensure that replication has completed, and then perform the cutover to the target SQL Server DB instance. You can then stop the push subscription, delete it, and start using Amazon RDS for SQL Server.

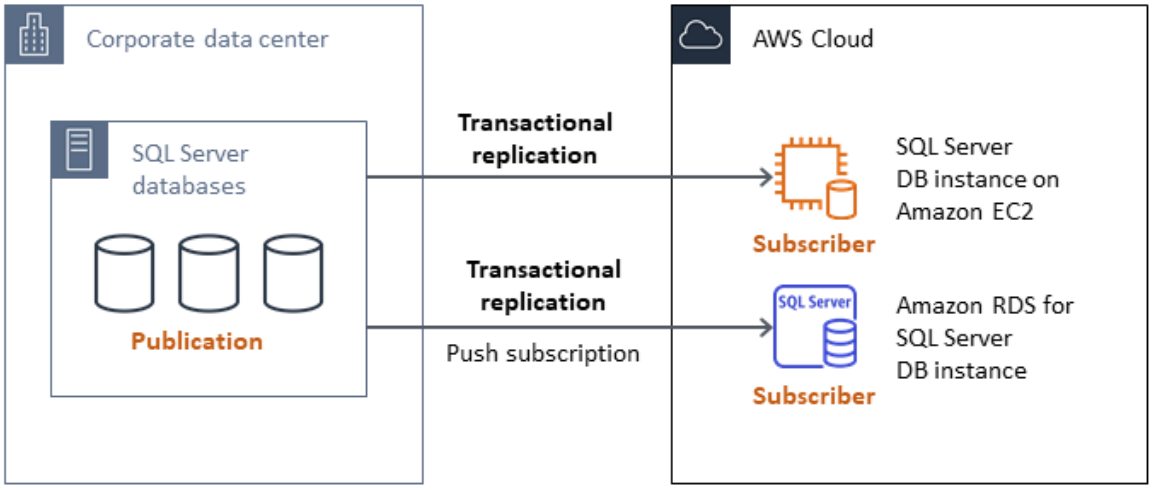
Subscriber databases can also be used as read-only databases. The distributor, which records synchronization jobs, is recommended to be on a separate server. If your target database is on Amazon RDS for SQL Server, you can set up a push subscription to propagate changes to the subscriber.

We recommend that you use transactional replication when you want to:

- Perform a one-time migration of your data to Amazon RDS or Amazon EC2.
- Migrate schema-level or table-level objects to AWS.
- Migrate a portion of a database to AWS.
- Migrate with minimal downtime using existing SQL Server replication strategies by adding additional subscribers.

If you're planning to use transactional replication for one-time migration of your data to Amazon RDS for SQL Server, we recommend that you set up a Single-AZ configuration for the replication. After the replication process is complete, you can convert your environment into a Multi-AZ architecture for high availability.

The following diagram shows the transactional replication process for databases on Amazon RDS and Amazon EC2.



For more information about transactional replication, see the Microsoft SQL Server documentation and the post How to migrate to Amazon RDS for SQL Server using transactional replication on the AWS Database blog.

AWS Snowball Edge

You can use AWS Snowball Edge to migrate very large databases (up to 80 TB in size). Snowball has a 10 Gb Ethernet port that you plug into your on-premises server and place all database backups or data on the Snowball device. After the data is copied to Snowball, you send the appliance to AWS for placement in your designated S3 bucket. You can then download the backups from Amazon S3 and restore them on SQL Server on an EC2 instance, or run the `rds_restore_database` stored procedure to restore the database to Amazon RDS. You can also use AWS Snowcone for databases up to 8 TB in size. For more information, see the AWS Snowball Edge documentation and Importing and exporting SQL Server databases, *Restoring a database* section, in the Amazon RDS documentation.

CloudEndure Migration

CloudEndure Migration is an agent-based solution that migrates your application or SQL Server database servers to AWS by using rehosting (lift-and-shift). It supports self-service, rapid, reliable migrations with

minimal business disruption. Using CloudEndure Migration, you can migrate SQL Server database servers that are running Windows Server version 2003, 2008, 2012, 2016, or 2019.

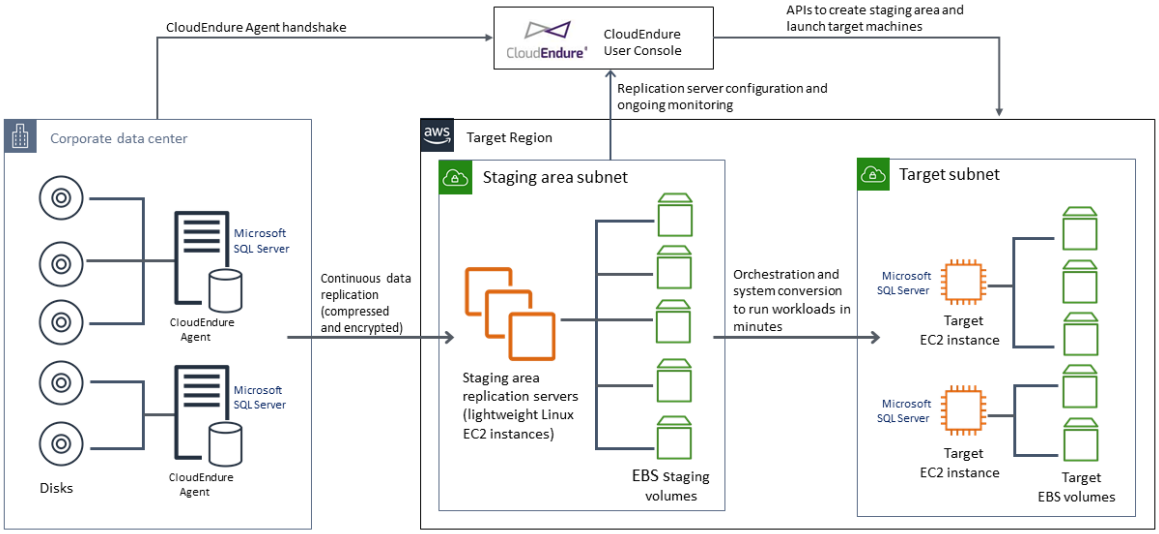
To migrate using CloudEndure, you first install the CloudEndure Agent on your source SQL Server machine, which connects to a self-service, web-based CloudEndure User Console that issues an API call to the selected target AWS staging area.

The staging area consists of both lightweight EC2 instances that act as replication servers and low-cost staging Amazon Elastic Block Store (Amazon EBS) volumes. The replication server receives data from the CloudEndure Agent that's running on the source machine and writes this data to the staging EBS volumes.

After all the source disks have been replicated to the staging area, the Agent continues to track and replicate any changes on the source disk to the target staging area asynchronously, using block-level data replication.

You configure your target machine in the CloudEndure User Console. When the target machines are ready to be launched, CloudEndure automatically converts them to boot and run natively on AWS. After this process finishes, the machines are ready and available for use.

The following diagram shows the migration process.



For more information, see the CloudEndure Migration website on AWS and the CloudEndure Migration pattern for SQL Server.

Homogeneous database migration for SQL Server

AWS offers you the ability to run SQL Server databases in a cloud environment. For developers and database administrators, running SQL Server database in the AWS Cloud is very similar to running SQL Server database in a data center. This section describes options for migrating your SQL Server database from an on-premises environment or a data center to the AWS Cloud.

AWS offers four options for running SQL Server on AWS, as described in the following table.

Option	Highlights	More information
SQL Server on Amazon RDS	Managed service, provides easy provisioning and licensing, cost-effective, easy to set up, manage, and maintain.	Amazon RDS for SQL Server (p. 16) section
SQL Server on Amazon RDS for VMware	AWS-managed relational databases in your on-premises, virtualized VMware vSphere environment.	Amazon RDS on VMware (p. 21) section
SQL Server on Amazon EC2	Self-managed, provides full control and flexibility.	Amazon EC2 for SQL Server (p. 22) section
SQL Server on VMware Cloud on AWS	Set up, scale, and operate your SQL Server workloads on VMware Cloud on AWS and integrate with AWS Directory Service, Active Directory Connector, and Amazon S3.	VMware Cloud on AWS for SQL Server (p. 28) section

Your application requirements, database features, functionality, growth capacity, and overall architecture complexity will determine which option to choose. If you are migrating multiple SQL Server databases to AWS, some of them might be a great fit for Amazon RDS, whereas others might be better suited to run directly on Amazon EC2. You might have databases that are running on SQL Server Enterprise edition but are a good fit for SQL Server Standard edition. You might also want to modernize your SQL Server database running on Windows to run on a Linux operating system to save on cost and licenses. Many AWS customers run multiple SQL Server database workloads across Amazon RDS, Amazon EC2, and VMware Cloud on AWS.

Amazon RDS for SQL Server

Amazon RDS for SQL Server is a managed database service that simplifies the provisioning and management of SQL Server on AWS. Amazon RDS makes it easy to set up, operate, and scale SQL Server deployments in the cloud. With Amazon RDS, you can deploy multiple versions of SQL Server

(2012, 2014, 2016, 2017, and 2019) and editions (including Express, Web, Standard and Enterprise) in minutes, with cost-efficient and resizable compute capacity. You can provision Amazon RDS for SQL Server DB instances with either General Purpose SSD or Provisioned IOPS SSD storage. (For details, see Amazon RDS Storage Types in the AWS documentation.) Provisioned IOPS SSD is designed to deliver fast, predictable, and consistent I/O performance, and is optimized for I/O-intensive, transactional (OLTP) database workloads.

Amazon RDS frees you to focus on application development, because it manages time-consuming database administration tasks, including provisioning, backups, software patching, monitoring, and hardware scaling. Amazon RDS for SQL Server also offers Multi-AZ deployments and read replicas (for SQL Server Enterprise edition) to provide high availability, performance, scalability, and reliability for production workloads.

For more information about migrating from SQL Server to Amazon RDS, see the replatform patterns on the AWS Prescriptive Guidance website.

When to choose Amazon RDS

Amazon RDS for SQL Server is a migration option when:

- You want to focus on your business and applications, and you want AWS to take care of undifferentiated heavy lifting tasks such as the provisioning of the database, management of backup and recovery tasks, management of security patches, minor SQL Server version upgrades, and storage management.
- You need a highly available database solution, and you want to take advantage of the push-button, synchronous Multi-AZ replication offered by Amazon RDS, without having to manually set up and maintain database mirroring, failover clusters, or Always On availability groups.
- You want to pay for the SQL Server license as part of the instance cost on an hourly basis instead of making a large, upfront investment.
- Your database size and IOPS needs are supported by Amazon RDS for SQL Server. See Amazon RDS DB Instance Storage in the AWS documentation for the current maximum limits.
- You don't want to manage backups or point-in-time recoveries of your database.
- You want to focus on high-level tasks, such as performance tuning and schema optimization, instead of the daily administration of the database.
- You want to scale the instance type up or down based on your workload patterns without being concerned about licensing complexities.

After assessing your database and project requirements, if you decide to migrate to Amazon RDS for SQL Server, see the details provided in the following sections, and review the migration best practices (p. 39) we discuss later in this guide.

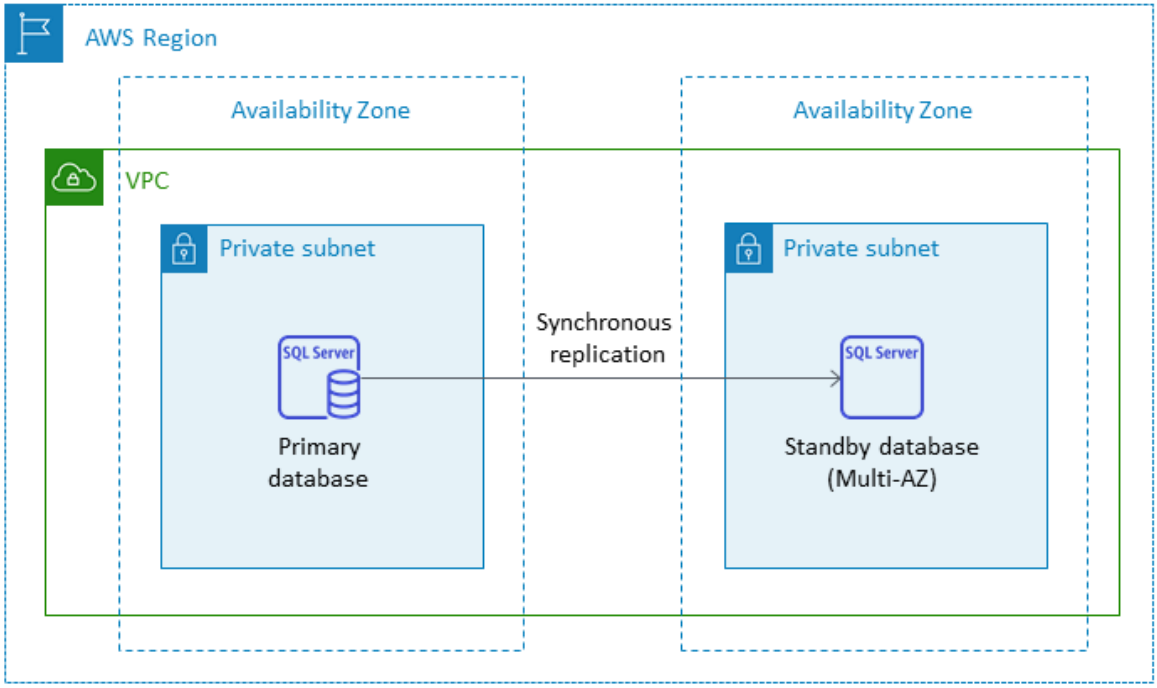
For the currently supported features, versions, and options, see Amazon RDS for SQL Server features on the AWS website and Microsoft SQL Server on Amazon RDS in the AWS documentation.

High availability

Amazon RDS provides high availability and failover support for databases that are deployed with the Multi-AZ option. When you provision your database with the Multi-AZ option, Amazon RDS automatically provisions and maintains a synchronous standby instance in a different Availability Zone. The primary database synchronously replicates the data to the standby instance. If problems occur, Amazon RDS automatically repairs the unhealthy instance and re-establishes synchronization. In case of infrastructure failure or Availability Zone disruption, Amazon RDS performs an automatic failover to the standby instance. Failover occurs only if the standby and primary databases are fully synchronized. Because the endpoint remains the same for the primary and standby instances, you can resume database operations

as soon as the failover is complete, without performing a manual intervention. The failover time depends on the time it takes to complete the recovery process. Large transactions increase the failover time.

The following diagram illustrates the Amazon RDS for SQL Server Multi-AZ deployment option.



When you set up SQL Server in a Multi-AZ configuration, Amazon RDS automatically configures standby database instance using database mirroring or Always On availability groups, based on the version of SQL Server that you deploy.

Amazon RDS supports Multi-AZ with Always On availability groups for the following SQL Server versions and editions:

- SQL Server 2019 Enterprise edition 15.00.4043.16 or later
- SQL Server 2017 Enterprise edition 14.00.3049.1 or later
- SQL Server 2016 Enterprise edition 13.00.5216.0 or later

Amazon RDS supports Multi-AZ with database mirroring for the following SQL Server versions and editions, except for the versions of Enterprise edition noted previously:

- SQL Server 2017 Standard and Enterprise editions
- SQL Server 2016 Standard and Enterprise editions
- SQL Server 2014 Standard and Enterprise editions
- SQL Server 2012 Standard and Enterprise editions

In Multi-AZ deployments, operations such as instance scaling or system upgrades such as operating system (OS) patching are applied first on the standby instance, before the automatic failover of the primary instance, for enhanced availability.

Due to failover optimization of SQL Server, certain workloads can generate greater I/O load on the standby instance than they do on the primary instance, particularly in database mirroring deployments.

This functionality can result in higher IOPS on the standby instance. We recommend that you consider the maximum IOPS needs of both the primary and standby instances when you provision the storage type and IOPS of your Amazon RDS for SQL Server DB instance. You can also specify `MultiSubnetFailover=True`, if your client driver supports it, to significantly reduce the failover time.

Limitations

- The Multi-AZ option isn't available for SQL Server Express and Web editions. It's available only for SQL Server Standard and Enterprise editions.
- You can't configure the standby DB instance to accept database read activity.
- Cross-Region Multi-AZ isn't supported.
- In Amazon RDS you can issue a stop command to a standalone DB instance and keep the instance in a stopped state to avoid incurring compute charges. You can't stop an Amazon RDS for SQL Server DB instance in a Multi-AZ configuration. Instead, you can terminate the instance, take a final snapshot before termination, and recreate a new Amazon RDS instance from the snapshot when you need it. Or, you can remove the Multi-AZ configuration first and then stop the instance. After seven days, your stopped instance will restart so that any pending maintenance can be applied.

For additional limitations, see Microsoft SQL Server Multi-AZ deployment notes and recommendations in the Amazon RDS documentation.

Read replicas

Read replicas provide scalability and load balancing. A SQL Server read replica is a physical copy of an Amazon RDS for SQL Server DB instance that is used for read-only purposes. Amazon RDS helps reduces the load on the primary DB instance by offloading read-only workloads to the read replica DB instance. Updates made to your primary DB instance are asynchronously copied to the read replica instance.

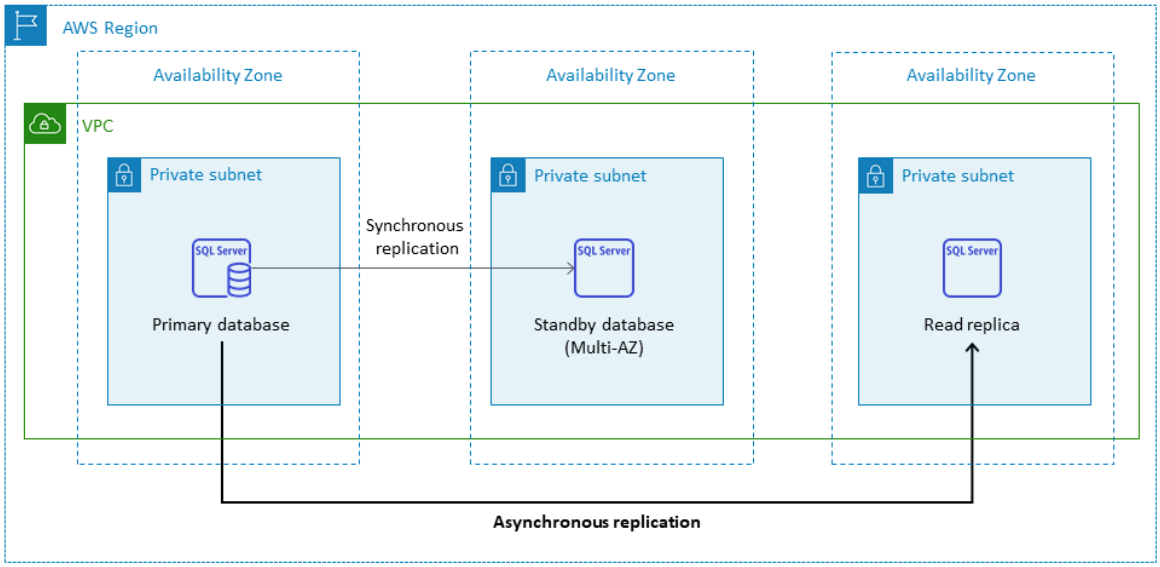
When you request a read replica, Amazon RDS takes a snapshot of the source DB instance, and this snapshot becomes the read replica. There is no outage while creating and deleting a read replica. Amazon RDS for SQL Server upgrades the primary database immediately after upgrading the read replicas, regardless of the maintenance window. Every read replica comes with a separate endpoint that you use to connect to the read replica database.

Amazon RDS for SQL Server makes it easy to create read replicas by configuring Always On availability groups, and maintaining secure network connections between a primary DB instance and its read replicas.

You can set up a read replica in the same AWS Region as your primary database. Amazon RDS for SQL Server doesn't support cross-Region read replicas. You can create up to five read replicas for one source DB instance.

> **Note**
> Read replicas are available only with the following SQL Server versions and editions:
>
> - SQL Server 2017 Enterprise edition 14.00.3049.1 or later
> - SQL Server 2016 Enterprise edition 13.00.5216.0 or later
>
> SQL Server versions and editions that support database mirroring for Multi-AZ environments do not offer read replicas.

The following diagram illustrates an Amazon RDS for SQL Server DB instance in a Multi-AZ environment with a read replica in another Availability Zone within the same AWS Region. Not all AWS Regions offer more than two Availability Zones, so you should check the Region you're planning to use before adopting this strategy.



A SQL Server read replica doesn't allow write operations. However, you can promote the read replica to make it writable. After you promote it, you cannot revert it back to a read replica. It will become a single, standalone DB instance that has no relationships with its original primary database instance. The data in the promoted read replica will match the data in the source DB instance up to the point when the request was made to promote it. The SQL Server DB engine version of the source DB instance and all of its read replicas will be the same.

For efficient replication, we recommend the following:

- Set up each read replica with the same compute and storage resources as the source DB instance.
- You must enable automatic backups on the source DB instance by setting the backup retention period to a value other than 0 (zero).
- The source DB instance must be deployed in a Multi-AZ environment with Always On availability groups.

For SQL Server version support, editions, and limitations, see Read replica limitations with SQL Server in the Amazon RDS documentation.

For more information about using read replicas, see Working with read replicas and Working with SQL Server read replicas for Amazon RDS in the AWS documentation. For more information about data transfer pricing, see Amazon RDS pricing.

Disaster recovery

With Amazon RDS for SQL Server you can create a reliable, cross-Region disaster recovery (DR) strategy. The main reasons for creating a DR solution are business continuity and compliance:

- An effective DR strategy helps you keep your systems up and running with minimal or no interruptions during a catastrophic event. A reliable and effective cross-Region DR strategy keeps your business in operation even if an entire Region goes offline.
- A cross-Region DR solution helps you meet auditing and compliance requirements.

Recovery point objective (RPO), recovery time objective (RTO), and cost are three key metrics to consider when developing your DR strategy. For other options for providing cross-Region replicas, see AWS

Marketplace. For more information about these approaches, see Cross-Region disaster recovery of Amazon RDS for SQL Server on the AWS Database blog.

Amazon RDS on VMware

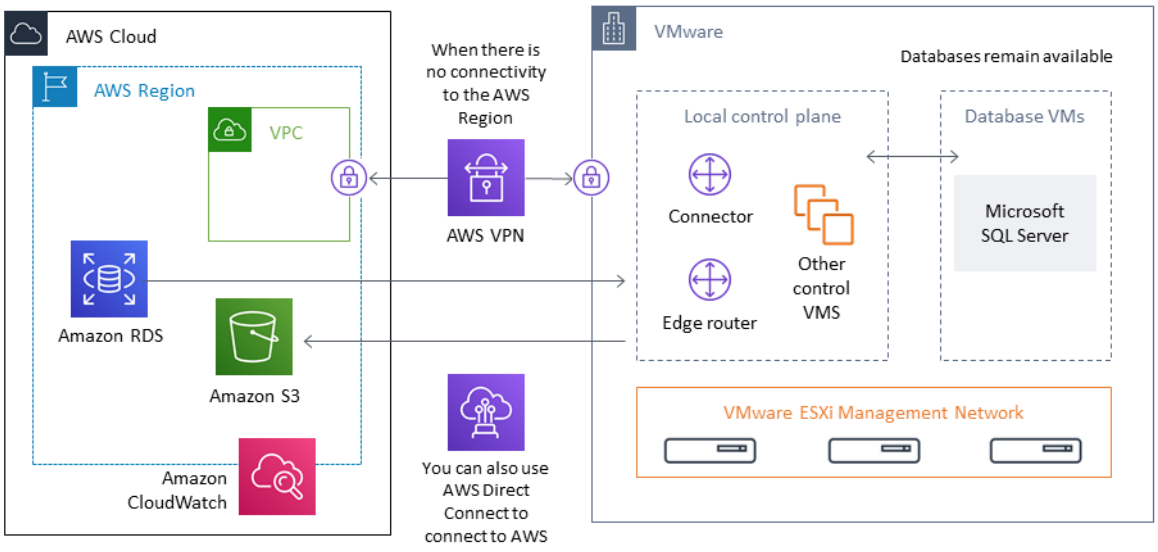
Amazon RDS on VMware lets you deploy managed Amazon RDS for SQL Server databases in your on-premises, virtualized VMware vSphere environment. You can provision new databases in minutes, take automatic snapshot backups, and restore them in your on-premises VMWare vSphere environment.

Amazon RDS provides cost-efficient and resizable capacity while automating time-consuming administration tasks like database setup, patching, and snapshot backups, and frees you to focus on your applications. Amazon RDS on VMware brings these benefits to your on-premises deployments, and makes it easy to set up, operate, and scale databases in VMware vSphere private data centers.

Amazon RDS on VMware uses the RDS connector to create a virtual private network (VPN) tunnel during the onboarding process. The VPN connection enables communications between your vSphere cluster and your AWS Region. You can manage your SQL Server databases by using the Amazon RDS console on vSphere, and issue API commands as you would in the AWS Cloud. You can view your databases in the cloud and on premises from a single pane.

In the following diagram, the Amazon RDS control plane resides in an AWS Region. From there, you can manage your databases on vSphere using the Amazon RDS console, run API commands in the same way you would manage your database in the cloud, and view your on-premises and cloud databases from a single pane. Amazon RDS on VMware also provides access to other AWS services such as Amazon CloudWatch and Amazon S3.

On the vSphere side, Amazon RDS deploys a local control plane and a data plane. The local control plane is responsible for communicating with the control plane in the AWS Region, and for managing the DB instances on premises. The SQL Server DB instances run from the VMs in the data plane. AWS and vSphere are connected through a secure VPN that's managed by Amazon RDS. The local control plane and the VPN are installed as part of the Amazon RDS on VMware onboarding process.



Amazon RDS on VMware currently supports SQL Server 2016 SP2 Enterprise edition. You have to bring your own media and your own Microsoft SQL Server (on-premises) license to create SQL Server databases that are managed by Amazon RDS on VMware.

For more information about Amazon RDS on VMware, see Amazon RDS on VMware on the AWS website and the Amazon RDS on VMware documentation.

When to choose Amazon RDS on VMware

Amazon RDS on VMware is a migration option for your SQL Server database when:

- You want your data to reside in your on-premises data center.
- You have applications that require low latency.

High availability

Amazon RDS on VMware uses VMware vSphere High Availability to provide availability and failover protection solutions. VMware vSphere HA offers availability protection by using local, on-premises monitoring that detects unhealthy database instances and automatically recovers them by using the same storage volume. For more information about high availability, see the VMware vSphere documentation.

Amazon EC2 for SQL Server

Amazon EC2 supports a self-managed SQL Server database. That is, it gives you full control over the setup of the infrastructure and the database environment. Running the database on Amazon EC2 is very similar to running the database on your own server. You have full control of the database and operating system-level access, so you can use your choice of tools to manage the operating system, database software, patches, data replication, backup, and restoration. This migration option requires you to set up, configure, manage, and tune all the components, including EC2 instances, storage volumes, scalability, networking, and security, based on AWS architecture best practices. You are responsible for data replication and recovery across your instances in the same or different AWS Regions.

For more information about migrating from SQL Server to Amazon EC2, see the rehost patterns on the AWS Prescriptive Guidance website.

When to choose Amazon EC2

Amazon EC2 is a good migration option for your SQL Server database when:

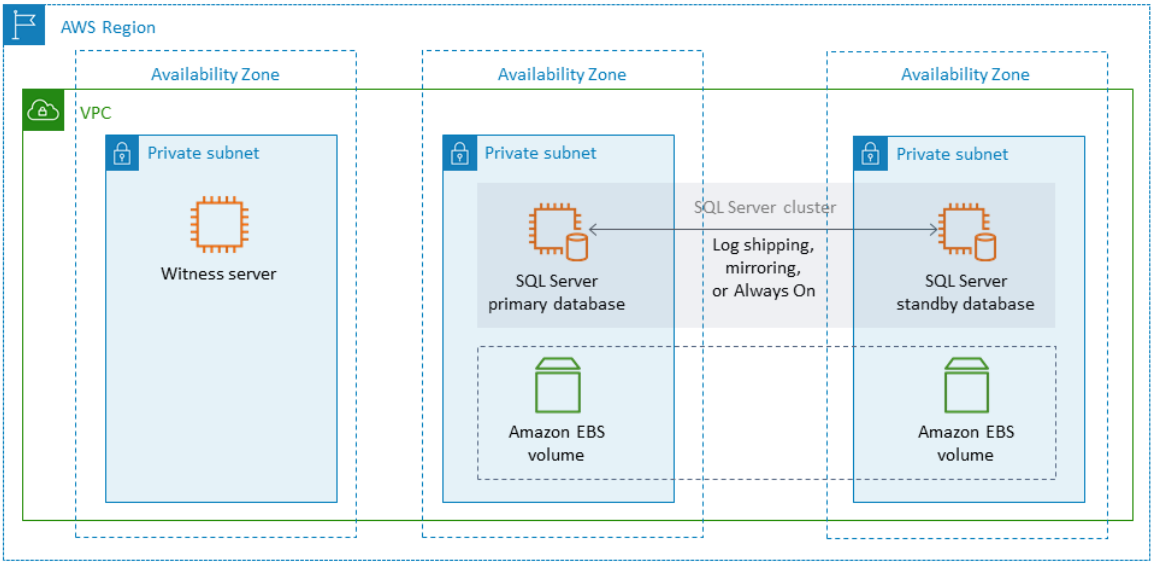
- You need full control over the database and access to its underlying operating system, database installation, and configuration.
- You want to administer your database, including backups and recovery, patching the operating system and the database, tuning the operating system and database parameters, managing security, and configuring high availability or replication.
- You want to use features and options that aren't currently supported by Amazon RDS. For details, see Features not supported and features with limited support in the Amazon RDS documentation.
- You need a specific SQL Server version that isn't supported by Amazon RDS. For a list of supported versions and editions, see SQL Server versions on Amazon RDS in the Amazon RDS documentation.
- Your database size and performance needs exceed the current Amazon RDS for SQL Server offerings. For details, see Amazon RDS DB instance storage in the Amazon RDS documentation.
- You want to avoid automatic software patches that might not be compliant with your applications.
- You want to bring your own license instead of using the Amazon RDS for SQL Server license-included model.

- You want to achieve higher IOPS and storage capacity than the current limits. For details, see Amazon RDS DB instance storage in the Amazon RDS documentation.

High availability

You can use any SQL Server-supported replication technology with your SQL Server database on Amazon EC2 to achieve high availability, data protection, and disaster recovery. Some of the common solutions are log shipping, database mirroring, Always On availability groups, and Always On Failover Cluster Instances.

The following diagram shows how you can use SQL Server on Amazon EC2 across multiple Availability Zones within a single AWS Region. The primary database is a read-write database, and the secondary database is configured with log shipping, database mirroring, or Always On availability groups for high availability. All the transaction data from the primary database is transferred and can be applied to the secondary database asynchronously for log shipping, and asynchronously for Always On availability groups and mirroring.



Log shipping

Log shipping lets you automatically send transaction log backups from a primary database instance to one or more secondary databases (also known as *warm standby*) on separate DB instances. Log shipping uses SQL Server Agent jobs to automate the process of backing up, copying, and applying the transaction log backups. Although log shipping is typically considered a disaster recovery feature, it can also provide high availability by allowing secondary DB instances to be promoted if the primary DB instance fails. If your RTO and RPO are flexible, or your databases aren't considered highly mission-critical, consider using log shipping to provide better availability for your SQL Server databases.

Log shipping increases the availability of databases by providing access to secondary databases to use as read-only copies of the primary database when needed. You can configure a lag delay (a longer delay time) during which you can recover accidentally changed data on the primary database before these changes are shipped to the secondary database.

We recommend running the primary and secondary DB instances in separate Availability Zones, and deploying a monitor instance to track all the details of log shipping. Backup, copy, restore, and failure events for a log shipping group are available from the monitor instance. A log shipping configuration

doesn't automatically fail over from the primary server to the secondary server. However, any of the secondary databases can be brought online manually if the primary database becomes unavailable.

Log shipping is often used as a disaster recovery solution but also can be used as a high availability solution, depending on your application requirements. Use log shipping when:

- You have flexible RTO and RPO requirements. Log shipping provides an RPO of minutes, and an RTO of minutes to hours.
- You do not need an automatic failover to the secondary database.
- You want to read from the secondary database, but you don't require readability during a restore operation.

For more information about log shipping, see the Microsoft SQL Server documentation.

Database mirroring

Database mirroring takes a database that's on an EC2 instance and provides a complete or almost complete read-only copy (mirror) of it on a separate DB instance. Amazon RDS uses database mirroring to provide Multi-AZ support for Amazon RDS for SQL Server. This feature increases the availability and protection of databases, and provides a mechanism to keep databases available during upgrades.

> **Note**
> According to the Microsoft documentation, database mirroring will be removed in a future version of SQL Server. You should plan to use Always On availability groups instead.

In database mirroring, SQL servers can take one of three roles:

- The principal server, which hosts the primary read/write version of the database.
- The mirror server, which hosts a copy of the principal database.
- An optional witness server. This server is available only in high-safety mode. It monitors the state of the database mirror and automates the failover from the primary database to the mirror database.

A mirroring session is established between the principal and mirror servers. During mirroring, all database changes that are performed in the principal database are also performed on the mirror database. Database mirroring can be either a synchronous or an asynchronous operation. This is determined by two mirroring operating modes: high-safety mode and high-performance mode.

- **High-safety mode:** This mode uses synchronous operations. In this mode, the database mirroring session synchronizes the insert, update, and delete operations from the principal database to the mirror database as quickly as possible. As soon as the database is synchronized, the transaction is committed in both the principal and the mirror databases. We recommend that you use this operating mode when the mirror databases are in the same or different Availability Zones, but hosted within the same AWS Region.
- **High-performance mode:** This mode uses asynchronous operations. In this mode, the database mirroring session synchronizes the insert, update, and delete operations from the principal database to the mirror database, but there can be a lag between the time the principal database commits transactions and the time the mirror database commits transactions. We recommend that you use this mode when the mirror databases are in different AWS Regions.

Use database mirroring when:

- You have strict RTO and RPO requirements, and cannot have delays between the primary and secondary databases. Database mirroring provides an RPO of zero seconds (with synchronous commit) and an RTO of seconds to minutes.

- You do not have a requirement to read from the secondary database.
- You want to perform automatic failover when you have a witness server configured in synchronization mode.
- You cannot use Always On availability groups, which is the preferred option.

Limitations:

- Only one-to-one failover is supported. You cannot have multiple database destinations sync with the primary database.

For more information about mirroring, see the Microsoft SQL Server documentation.

Always On availability groups

SQL Server Always On availability groups provide high availability and disaster recovery solutions for SQL Server databases. An availability group consists of a set of user databases that fail over together. It includes a single set of primary read/write databases and multiple (one to eight) sets of related, secondary databases. You can make the secondary databases available to the application tier as read-only copies of the primary databases (SQL Server Enterprise edition only), to provide a scale-out architecture for read workloads. You can also use the secondary databases for backup operations.

SQL Server Always On availability groups support both synchronous and asynchronous commit modes. In synchronous mode, the primary replica commits database transactions after the changes are committed or written to the log of the secondary replica. Using this mode, you can perform planned manual failover and automatic failover if the replicas are in sync. You can use synchronous commit mode between SQL Server instances within the same environment (for example, if all instances are on-premises or all instances are in AWS).

In asynchronous commit mode, the primary replica commits database transactions without waiting for the secondary replica. You can use asynchronous commit mode between SQL Server instances that are in different environments (for example, if you have instances on premises and in AWS).

You can use Always On availability groups for high availability or disaster recovery. Use this method when:

- You have strict RTO and RPO requirements. Always On availability groups provide an RPO of seconds, and an RTO of seconds to minutes.
- You want to manage and fail over a group of databases. Always On availability groups support 0-4 secondary replicas in synchronous commit mode for SQL Server 2019.
- You want to use automatic failover in synchronous commit mode, and you don't need a witness server.
- You want to read from the secondary database.
- You want to synchronize multiple database destinations with your primary database.

Starting with SQL Server 2016 SP1, SQL Server Standard edition provides basic high availability for a single, non-readable secondary database and listener per availability group. It also supports a maximum of two nodes per availability group.

Always On Failover Cluster Instances

SQL Server Always On Failover Cluster Instances (FCIs) use Windows Server Failover Clustering (WSFC) to provide high availability at the server instance level. An FCI is a single instance of SQL Server that is installed across WSFC nodes to provide high availability for the entire installation of SQL Server. If the underlying node experiences hardware, operating system, application, or service failures, everything

inside the SQL Server instance is moved to another WSFC node. This includes system databases, SQL Server logins, SQL Server Agent jobs, and certificates.

An FCI is generally preferable over an Always on availability group when:

- You're using SQL Server Standard edition instead of Enterprise edition.
- You have a large number of small databases per instance.
- You're constantly modifying instance-level objects such as SQL Server Agent jobs, logins, and so on.
- You want to optimize costs (both Amazon EC2 infrastructure costs and SQL Server license costs) for running the SQL Server instances. When you use an FCI, you remove the dependency of the SQL Server host EC2 instance on Amazon EBS throughput. As a result, you can right-size (that is, without the need to over-provision) the SQL Server EC2 instance and reduce your Amazon EC2 and SQL Server license costs, without affecting I/O performance.
- You want to optimize database performance on the SQL Server instances. Using an FCI provides about 30 percent better performance for typical workloads, because data replication is performed at the block level instead of the database level.

FCIs require some form of shared storage—disks on a storage area network (SAN), file shares on Server Message Blocks (SMBs), or locally attached storage with Storage Spaces Direct (S2D), SIOS Datakeeper, or Amazon FSx—to provide resiliency and high availability.

To deploy an FCI on AWS, you can use Amazon FSx for Windows File Server, which provides fully managed shared file storage. This service automatically replicates the storage synchronously across two Availability Zones to provide high availability. Using Amazon FSx for Windows File Server for file storage helps simplify and optimize your SQL Server high availability deployments on Amazon EC2. For more information, see the next section (p. 26).

Amazon FSx for Windows File Server

Amazon FSx for Windows File Server provides fully managed, highly reliable, and scalable file storage that is accessible by using the Server Message Block (SMB) protocol. It is built on Windows Server and delivers a wide range of administrative features such as user quotas, end-user file restore, and Microsoft Active Directory (AD) integration. It offers Single-AZ and Multi-AZ deployment options, fully managed backups, and encryption of data at rest and in transit. You can optimize cost and performance for your workloads with solid-state drives (SSD) and hard disk drives (HDD) storage options, and you can scale storage and change the throughput performance of your file system at any time. Amazon FSx file storage is accessible from Windows, Linux compute instances running on AWS, and on premises.

Amazon FSx makes it easier to deploy shared Windows storage for high availability SQL Server deployments through its support for continuously available (CA) file shares and smaller file systems. This option is suitable for these use cases:

- As shared storage used by SQL Server nodes in a WSFC instance.
- As an SMB file share witness that can be used with any SQL Server cluster with WSFC.

Amazon FSx provides fast performance with baseline throughput up to 2 GB/second per file system, hundreds of thousands of IOPS, and consistent sub-millisecond latencies.

To provide the right performance for your SQL instances, you can choose a throughput level that is independent of your file system size. Higher levels of throughput capacity also come with higher levels of IOPS that the file server can serve to the SQL Server instances accessing it.

The storage capacity determines not only how much data you can store, but also how many IOPS you can perform on the storage. Each gigabyte of storage provides 3 IOPS. You can provision each file system to be up to 64 TB in size.

For information about configuring and using Amazon FSx to reduce the complexity and costs of your SQL Server high availability deployments, see Simplify your Microsoft SQL Server high availability deployments using Amazon FSx for Windows File Server on the AWS Storage blog. To learn more about creating a new CA share, see the Amazon FSx for Windows File Server documentation.

Disaster recovery

Many organizations implement high availability for their SQL Server databases, but that isn't sufficient for organizations that require true IT resilience. We recommend that you implement a disaster recovery solution to avoid data loss and downtime of mission-critical databases. Adopting a multi-Region disaster recovery architecture for your SQL Server deployments help you:

- Achieve business continuity
- Improve latency for your geographically distributed customer base
- Satisfy your auditing and regulatory requirements

Options for disaster recovery include log shipping (p. 23), Always On availability groups (p. 25), Amazon EBS snapshots that are stored in Amazon S3 and replicated across AWS Regions, Always On Failover Cluster Instances (FCIs) (p. 25) combined with Always On availability groups, distributed availability groups, and CloudEndure Disaster Recovery. The following sections describe distributed availability groups and CloudEndure Disaster Recovery.

Distributed availability groups

An architecture with distributed availability groups is an optimal approach for multi-Region SQL Server deployment. A distributed availability group is a special type of availability group that spans two separate availability groups. You can think of it as an availability group of availability groups. The underlying availability groups are configured on two different WSFC clusters.

Distributed availability groups are loosely coupled, which means that they don't require a single WSFC cluster and they're maintained by SQL Server. Because the WSFC clusters are maintained individually and transmissions are primarily asynchronous between two availability groups, it's easier to configure disaster recovery at another site. The primary replicas in each availability group synchronize their own secondary replicas.

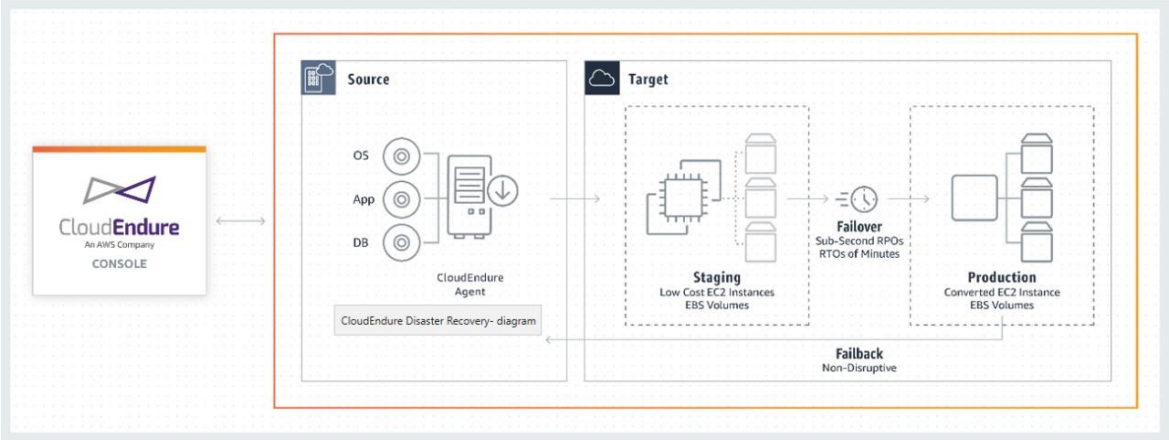
Distributed availability groups support only manual failover at this time. To ensure that no data is lost, stop all transactions on the global primary databases (that is, on the databases of the primary availability group). Then set the distributed availability group to synchronous commit.

CloudEndure Disaster Recovery

CloudEndure Disaster Recovery is an agent-based solution that replicates entire virtual machines, including the operating system, all installed applications, and all databases, into a staging area located in your target AWS Region. The staging area contains low-cost resources that are automatically provisioned and managed by CloudEndure Disaster Recovery.

CloudEndure Disaster Recovery maintains a continuous asynchronous replication of the disks into the staging area. This significantly reduces the cost of provisioning duplicate resources. Because the staging area doesn't run a live version of your SQL Server workloads, you don't have to pay for duplicate OS licenses or high-performance compute resources. Instead, you pay for low-cost compute and storage.

In an event of a disaster, CloudEndure Disaster Recovery triggers an automated conversion process. It recovers each source machine that is running SQL Server databases on a single EC2 instance, achieving sub-second recovery point objectives (RPOs) and recovery time objectives (RTOs) of minutes. CloudEndure Disaster Recovery uses ACID-compliant, crash consistent replication to provide consistent data. AWS makes CloudEndure available at no additional cost for migration projects.



For more information, see the CloudEndure Disaster Recovery for Microsoft SQL Server data sheet.

VMware Cloud on AWS for SQL Server

VMware Cloud on AWS is an integrated cloud offering jointly developed by AWS and VMware. SQL Server easily integrates with VMware Cloud on AWS. This migration option makes it possible for you to build on your existing investment in virtualization.

You can access VMware Cloud on AWS on an hourly, on-demand basis or in subscription form. It includes the same core VMware technologies that you run in your data centers, including vSphere Hypervisor (ESXi), Virtual SAN (vSAN), and the NSX network virtualization platform, and is designed to provide an efficient, seamless experience for managing your SQL Server databases. You can scale the storage, compute and memory of your SQL Server databases on VMware Cloud on AWS within minutes.

VMware Cloud on AWS runs directly on the physical hardware, but takes advantage of network and hardware features that were designed to support the AWS security-first infrastructure model. This means that the VMware virtualization stack runs on AWS infrastructure without having to use nested virtualization.

VMware Cloud on AWS makes it is easy to set up, scale, and operate your SQL Server databases workloads on AWS. It provides high availability solutions, integrates with on-premises Active Directory, and provides access to AWS services like AWS Directory Service for Microsoft Active Directory and AD Connector, Amazon Route 53, Amazon CloudWatch, and Amazon S3. You can store your backups in Amazon S3, and modernize and simplify your disaster recovery process.

When to choose VMware Cloud on AWS

VMware Cloud on AWS is an option for your SQL Server database when:

- Your SQL Server databases are already running in an on-premises data center in a vSphere virtualized environment.
- You have a large number of databases and you need fast migration (for example, only a few hours) to the cloud for one of the following reasons, without requiring any additional work from the migration team:
  - Data center extension. You need on-demand capacity to run virtualized desktops, to publish applications, or to provide a development/testing environment.
  - Disaster recovery. You want to set up a new disaster recovery system or replace your existing system.
  - Cloud migration. You want to migrate your entire data center to the cloud, or refresh your infrastructure.

If your SQL Server database requires more than 80K IOPS, you can use vSAN.

For more information, see In the Works – VMware Cloud on AWS on the AWS News blog, and Deploy Microsoft SQL Server on VMware Cloud on AWS on the AWS website.

Heterogeneous database migration for SQL Server

Because of the innovations and improvements in open-source databases and cloud computing platforms like AWS, many organizations are moving from proprietary (online transaction processing or OLTP) database engines such as SQL Server to open-source engines. SQL Server databases are mission-critical systems for any organization, but being locked into a particular vendor is a risky and costly situation. Low operating cost and no licensing fees are compelling reasons to consider switching the underlying database technology to open-source or AWS Cloud-native databases.

Other reasons for migrating off SQL Server are vendor lock-in periods, licensing audits, expensive licensing, and cost. For this reason, many organizations choose to migrate their SQL Server databases to either open-source databases (such as PostgreSQL, MySQL, or MariaDB) or AWS Cloud-native databases (such as Amazon Aurora or Amazon DynamoDB) when they migrate to AWS.

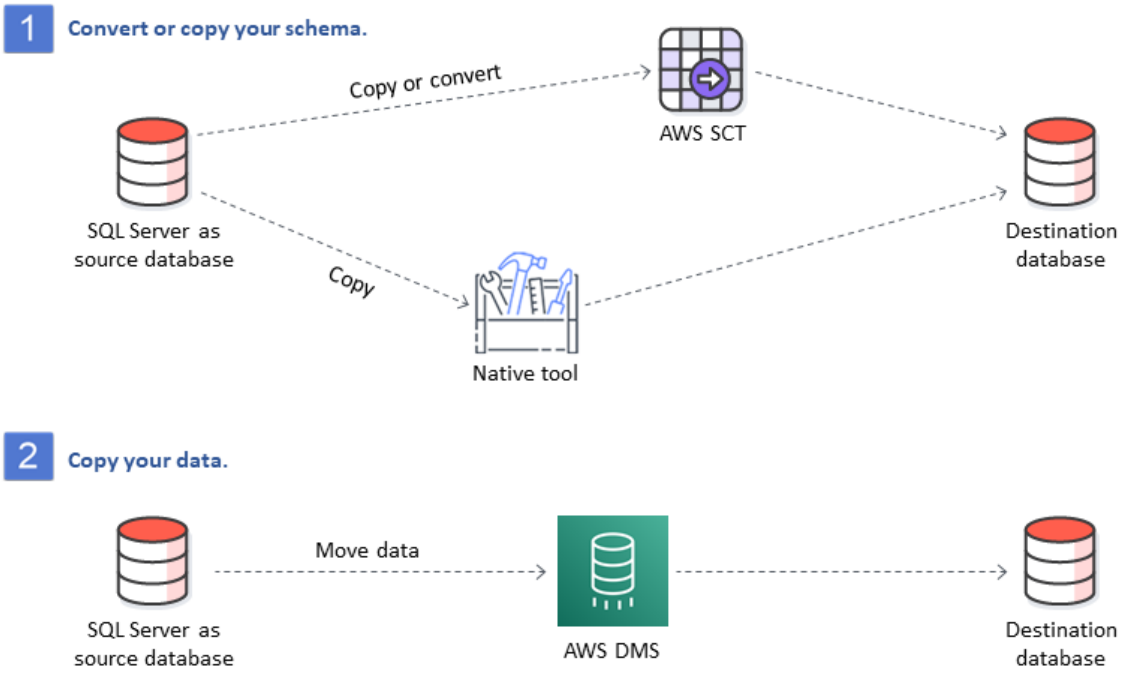
You can also migrate your SQL Server data warehouse database to Amazon Redshift, which is a fast, fully managed cloud data warehouse. Amazon Redshift is integrated with your data lake, offers up to three times faster performance than any other data warehouse, and costs up to 75 percent less than any other cloud data warehouse. For more information, see the pattern Migrate an on-premises Microsoft SQL Server database to Amazon Redshift using AWS DMS on the AWS Prescriptive Guidance website.

To migrate to an open-source or AWS-native database, choose the right database depending on the type of data you have, the access model, scalability, application practicalities, and complexity. Migrating from SQL Server to PostgreSQL and to other open-source databases has often been difficult and time-consuming, and requires careful assessment, planning, and testing.

This process becomes easier with services like AWS Database Migration Service (AWS DMS) and AWS Schema Conversion Tool (AWS SCT), which help you migrate your commercial database to an open-source database on AWS with minimal downtime.

In heterogeneous database migrations, the source and target databases engines are different, as in SQL Server to Aurora, or SQL Server to MariaDB migrations. The schema structure, data types, and database code in the source and target databases can be quite different, so the schema and code must be transformed before the data migration starts. For this reason, heterogeneous migration is a two-step process:

- Step 1. Convert the source schema and code to match that of the target database. You can use AWS SCT for this conversion.
- Step 2. Migrate data from the source database to the target database. You can use AWS DMS for this process.



AWS DMS handles the major data type conversions automatically during migration. The source database can be located in your own premises outside AWS, it can be a database that's running on an EC2 instance, or it can be an Amazon RDS database (see Sources for Data Migration in the AWS DMS documentation). The target can be a database in Amazon EC2, Amazon RDS, or Aurora. For information about using MySQL as a target database, see Migrating a SQL Server Database to a MySQL-Compatible Database Engine on the AWS Database blog.

For more information about refactoring your SQL Server database on AWS, see the re-architect patterns on the AWS Prescriptive Guidance website.

Tools for heterogeneous database migrations

The following chart provides a list of tools that you can use to migrate from SQL Server to another database engine.

Migration tool	Target database support	Used for
AWS WQF (p. 32)	Amazon RDS for MySQL Amazon RDS for PostgreSQL Amazon Aurora MySQL Amazon Aurora PostgreSQL	Pre-migration analysis
AWS SCT (p. 32)	Amazon RDS for MySQL Amazon RDS for PostgreSQL Amazon Aurora MySQL	Schema conversion

Migration tool	Target database support	Used for
	Amazon Aurora PostgreSQL	
AWS DMS (p. 32)	Amazon RDS for MySQL Amazon RDS for PostgreSQL Amazon Aurora MySQL Amazon Aurora PostgreSQL	Data migration

The following subsections provide more information about each tool.

AWS WQF

AWS Workload Qualification Framework (AWS WQF) is used to analyze enterprise databases, code, third-party dependencies, and frameworks for migrating to the AWS Cloud. It assesses and rates the workload for the entire migration, including database and application modifications. WQF can recommend strategies and tools that you can use for your migration, and provide actionable feedback. It can also identify actions for completing a migration to Amazon RDS or Amazon Aurora.

You can use WQF during the planning phase of your migration process to determine what you need to do to migrate your data and applications. WQF reports on the following:

- Workload assessment, based on complexity, size, and technology used
- Recommendations on migration strategies
- Recommendations on migration tools
- Feedback on what exactly to do
- Assessment of the effort required, based on team size and member roles

AWS doesn't currently provide the AWS WQF for downloading. If you need help assessing a migration to AWS with AWS WQF, we recommend opening a support ticket. AWS will engage with you directly to help make the process work for you.

AWS SCT

AWS Schema Conversion Tool (AWS SCT) converts your existing commercial database schemas to an open-source engine or to an AWS Cloud-native database. AWS SCT makes heterogeneous database migrations predictable by automatically converting the source database schema and a majority of the database code objects, including views, stored procedures, and functions, to a format that's compatible with the target database.

When you convert your database schema from one engine to another, you also need to update the SQL code in your applications to interact with the new database engine instead of the old one. AWS SCT also converts the SQL code in C++, C#, Java or other application code. Any objects that can't be automatically converted are clearly marked for manual conversion. AWS SCT can also scan your application source code for embedded SQL statements and convert them as part of a database schema conversion project. For more information, see Using Microsoft SQL Server as a source for AWS SCT in the AWS documentation.

AWS DMS

AWS Database Migration Service (AWS DMS) migrates your data rapidly and securely to AWS. During migration, the source database remains fully operational, minimizing application downtime. AWS DMS

supports homogenous migrations such as migrating data from one SQL Server database to another. It also supports heterogenous migrations between different database platforms, such as migrating your SQL Server database to an open-source database or to an AWS cloud-native database. AWS DMS manages the complexities of the migration process, including automatically replicating data changes that occur in the source database to the target database. After the database migration is complete, the target database remains synchronized with the source database for as long as you choose, and you can switch over to the target database at a convenient time. For more information, see Using a Microsoft SQL Server database as a source for AWS DMS in the AWS documentation.

Hybrid migration scenarios for SQL Server

You can also run SQL Server workloads in a hybrid environment that includes AWS. For example, you might already be running SQL Server in your on-premises or co-located data center but want to use the AWS Cloud to enhance your architecture to provide a high availability or disaster recovery solution. You can also use hybrid solutions to store long-term SQL Server backups on AWS, to roll back your migration, in case of issues, or to run a secondary replica using SQL Server Always On availability groups in the AWS Cloud. SQL Server has several replication technologies that offer high availability and disaster recovery solutions.

Backing up your SQL Server databases to the AWS Cloud

Amazon Simple Storage Service (Amazon S3) enables you to take advantage of the flexibility and pricing of cloud storage. It gives you the ability to back up your SQL Server databases to a secure, highly available, highly durable, reliable storage system. You can securely store your SQL Server backups in Amazon S3. You can also use Amazon S3 Lifecycle policies to store your backups for the long term. Amazon S3 allows you to store large amounts of data at a very low cost. You can use AWS DataSync to transfer backup files to Amazon S3.

You can use AWS Storage Gateway to store your on-premises SQL Server backups and archive data on Amazon S3 or Amazon S3 Glacier. You can create cached storage volumes and mount them as Internet Small Computer System Interface (iSCSI) devices from your on-premises backup application servers. All data is securely transferred to AWS over SSL and stored in encrypted format in Amazon S3. Using gateway cached volumes saves the upfront cost of maintaining and scaling costly storage hardware on premises. If you want to keep your primary data or backups on premises, you can use gateway stored volumes to keep this data locally, and back up the data off-site to Amazon S3.

Extending high availability and disaster recovery solutions

You can extend your existing on-premises high availability practices and provide a disaster recovery solution in AWS by using the native log shipping feature in SQL Server. You can transfer your SQL Server transaction logs from your on-premises or co-located data centers to a SQL Server instance that is running on an EC2 instance or an Amazon RDS for SQL Server DB instance in a virtual private cloud (VPC). You can transmit this data securely over a dedicated network connection by using AWS Direct Connect, or transmit it over a secure VPN tunnel. The transaction log backups are sent to the EC2 instance, and they are applied to secondary database instances.

You can use the AWS Cloud to provide a higher level of high availability and disaster recovery by using SQL Server Always On availability groups between your on-premises data center and Amazon EC2. This can be done by extending your data center into a VPC on AWS by using a dedicated network connection like AWS Direct Connect, or by setting secure VPN tunnels between these two environments.

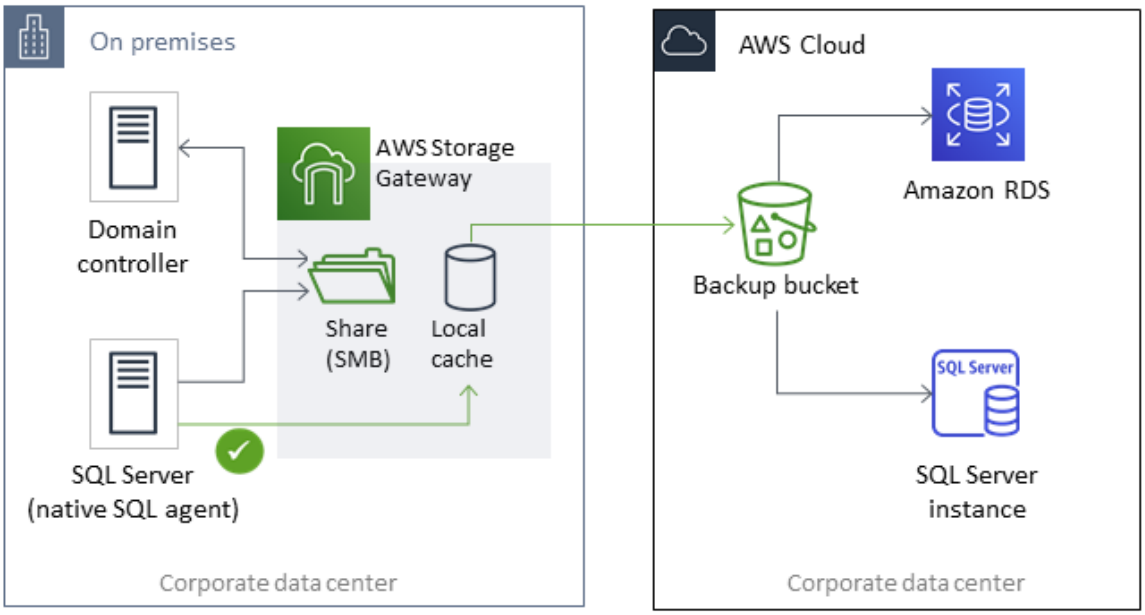
Here are a few things to consider when planning a hybrid implementation of SQL Server Always On availability groups:

- Establish secure, reliable, and consistent network connections between your on-premises environment and AWS through AWS Direct Connect or VPN.
- Create a VPC by using the Amazon Virtual Private Cloud (Amazon VPC) service. Use Amazon VPC route tables and security groups to enable the appropriate communications between the two environments.
- Extend Active Directory domains into the VPC by deploying domain controllers as EC2 instances, or by using AWS Directory Service for Microsoft Active Directory. You can also use AWS Managed Microsoft AD for Amazon RDS for SQL Server. For more information, see the Amazon RDS documentation.

AWS Storage Gateway

AWS Storage Gateway enables you to store and retrieve files by using a Server Message Block (SMB) share for Windows. You can join the storage gateway to your on-premises Active Directory domain. By having your SQL Server database and storage gateway in the same domain, you can take the backups directly to the SMB network share instead of storing them locally and then uploading them to the network share. The storage gateway is configured to use an S3 bucket, so all your backups will be available in the S3 bucket on AWS. You can restore your database by downloading the backup files to SQL Server on an EC2 instance, or restore the database directly to Amazon RDS.

The following diagram shows how to store and access backups by using AWS Storage Gateway and Amazon S3. For more information, see the AWS Storage Gateway documentation.



Using AWS DMS and AWS SCT

You can use AWS DMS in hybrid SQL Server environments, to migrate data from your on-premises database to the cloud, or the other way around. You can migrate your SQL Server database to MySQL or PostgreSQL by using AWS DMS with AWS SCT. For migration steps, see the AWS SCT documentation. Before you migrate your data, you can run a migration assessment report that flags any additional manual work that might be required.

You can also use AWS DMS for ongoing replication (change data capture or CDC). For more information, see Using ongoing replication (CDC) from a SQL Server source in the AWS DMS documentation.

Modernizing your SQL Server database

This section describes how to modernize your SQL Server workloads on AWS by switching from the Windows operating system to Linux. This change enables you to take advantage of open-source technologies and to save on Windows licensing costs without drastically altering your system architecture or retraining your users.

Migrating your SQL Server workloads from Windows to Linux

Starting with SQL Server 2017, SQL Server is available to run on Linux operating systems. Moving your SQL Server workloads to Linux provides both cost savings and performance improvements.

Almost all SQL Server functions, applications, statements, and scripts that you use on Microsoft Windows are supported on Linux as well. You can also use tools such as SQL Server Management Studio (SSMS), SQL Server Data Tools (SSDT), and PowerShell module (sqlps) to manage SQL Server on Linux from a Windows instance.

You can use one of these three options to migrate your SQL Server workloads to Linux:

- Native SQL Server backup and restore feature (see the Microsoft SQL Server documentation)
- Distributed availability groups (to change your operating system while you migrate to AWS)
- The AWS replatforming assistant, which is a PowerShell-based scripting tool

The AWS replatforming assistant helps you migrate from your existing SQL Server workloads from Windows to a Linux operating system. When you run the PowerShell script for the replatforming assistant on a source SQL Server database, the Windows instance backs up the database to an encrypted Amazon S3 storage bucket. It then restores the backup to new or existing SQL Server database on an EC2 Linux instance. You can replicate your database and test your applications while your source SQL Server database remains online. After testing, you can schedule application downtime and rerun the PowerShell backup script to perform your final cutover.

For more information about using the replatforming assistant, see Migrate your on-premises SQL Server Windows Workloads to Amazon EC2 Linux on the AWS Database blog, and the Amazon EC2 documentation.

High availability on Linux

SQL Server 2017 supports Always On availability groups between Windows and Linux to create read-scale workloads without high availability. Unfortunately, you cannot achieve high availability between Windows and Linux, because there is no clustered solution that can manage that cross-platform configuration.

To use high availability with Always On availability groups, consider using a Windows Server Failover Cluster (WSFC) or Pacemaker on Linux. This solution is suitable for a migration path from SQL Server on

Windows to Linux and vice versa, or for disaster recovery using manual failover. For more information about this scenario, see Deploying Always On availability groups between Amazon EC2 Windows and Amazon Linux 2 instances on the AWS Database blog.

Best practices for migrating to Amazon RDS for SQL Server

Based on the assessment of your database and your project requirements, if your goal is to migrate to Amazon RDS for SQL Server, follow the best practices in this section to provision your target database, perform the migration, and test, operate, and optimize your Amazon RDS for SQL Server database.

Important
Make sure that you have a rollback plan before you migrate your database.

Provisioning your target database

After you finish assessing, planning, and preparing your database migration strategy, follow these best practices when provisioning your Amazon RDS for SQL Server database:

- Right-size the Amazon RDS for SQL Server DB instance based on your requirements for CPU, memory, IOPS, and storage type. (If you're using SQL Server Standard edition, provision CPU and memory within the limitations of Standard edition.)
- Set the correct time zone and collation.
- Make sure to launch Amazon RDS in the correct virtual private cloud (VPC).
- Create the security groups with correct port and IP addresses.
- Provision your Amazon RDS database in a private subnet for security.
- If possible, provision the SQL Server instance with the latest version of SQL Server.
- Create a separate option group and parameter group for each Amazon RDS database.
- Collect and extract logins, users, and roles for migration.
- Review SQL Server Agent jobs for maintenance and applications that need to be migrated.

Backing up from your source database

There are many tools for migrating a SQL Server database to an Amazon RDS for SQL Server database. The most commonly used method is using SQL Server native backup and restore if your requirements allow downtime.

If you have limited downtime, you can use native SQL Server backup/restore with differential backup and log backup. Or you can use AWS DMS, which provides three options: full-load, full-load and CDC, or CDC only.

Transferring data dump files to AWS

- If you're using AWS Direct Connect, which provides high bandwidth connectivity between your on-premises environment and AWS, you can copy your SQL Server backups to Amazon S3 and set up Amazon S3 integration.
- If you don't have high bandwidth through AWS Direct Connect, use AWS Snowball to transfer large database backup files. You can also use AWS DMS to transfer the data when replication is required.

Restoring data to your target database

- If you're migrating a very large database, we recommend that you provision a bigger Amazon RDS instance type initially, for the duration of the migration, for faster data loads.
- Disable Multi-AZ. (This can be re-enabled after migration.)
- Disable backup retention. (This can be re-enabled after migration.)
- Restore the database by using the native SQL Server **restore** command.
- Create logins and users, and fix orphaned users, if required.
- Create SQL Server Agent jobs and review the schedule, as needed.

Post-migration steps

After the migration is complete, you can:

- Change the DB instance to the right-sized instance type.
- Enable Multi-AZ and backup retention.
- Make sure that all jobs are created on secondary nodes (for Multi-AZ configuration).
- Publish SQL Server error and agent logs to Amazon CloudWatch Logs, and use CloudWatch to view metrics and create alarms. For more information, see the Amazon RDS documentation.
- Enable enhanced monitoring to get metrics for your DB instance in real time.
- Set up Amazon Simple Notification Service (Amazon SNS) topics for alerts.

Testing the migration

We recommend the following tests to validate your application against your new Amazon RDS for SQL Server database:

- Perform functional testing.
- Compare the performance of SQL queries in your source and target databases, and tune the queries as needed. Some queries might perform more slowly in the target database, so we recommend that you capture the baselines of the SQL queries in the source database.

For additional validation during the proof-of-concept (POC) phase, we recommend the following supplemental tests:

- Run performance tests to ensure that they meet your business expectations.
- Test database failover, recovery, and restoration to make sure that you're meeting RPO and RTO requirements.
- List all critical jobs and reports, and run them on Amazon RDS to evaluate their performance against your service-level agreements (SLAs).

Operating and optimizing your Amazon RDS database

When your database is on AWS, make sure that you are following best practices in areas such as monitoring, alerting, backups, and high availability in the cloud. For example:

- Set up CloudWatch monitoring, and enable detailed monitoring.
- Use Amazon RDS Performance Insights and other third-party monitoring solutions like SentryOne or Foglight for SQL Server to monitor your database.
- Set up alerts by using SNS topics.
- Set up automatic backups by using AWS Backup or native SQL Server backups, and copy to Amazon S3.
- For high availability, set up the Amazon RDS Multi-AZ feature.
- If you need read-only databases, set up a read replica (p. 19) within the same or across AWS Regions according to your needs.

SQL Server database migration patterns

Use the following links to see the AWS Prescriptive Guidance patterns for migrating n SQL Server database to AWS:

- Rehost patterns (from SQL Server to Amazon EC2)
- Replatform patterns (from SQL Server to Amazon RDS for SQL Server)
- Re-architect patterns (from SQL Server to open-source and AWS Cloud-native databases)
- SQL Server patterns that use CloudEndure Migration

If you're looking for patterns that cover the use of a specific tool, type in the tool name in the search box or choose it from a filter. For example, you can use this query to see all SQL Server migration patterns that use AWS DMS.

Partners

Database migration can be a challenging project that requires expertise and tools. You can accelerate your migration and time to results through partnership. AWS Database Migration Service partners have the required expertise to help customers migrate to the cloud easily and securely. These partners have the expertise for both homogenous migrations such as SQL Server to SQL Server, and heterogeneous migrations between different database platforms, such as SQL Server to Amazon Aurora or Amazon RDS for MySQL.

Based on your requirements and preferences, you can use the partner to handle the complete migration or to help with only some aspects of the migration. In addition, you can use tools and solutions provided by AWS Partner Network (APN) Partners to help with the migration. For a complete catalog of migration tools and solutions, see APN Partner tools and solutions.

Additional resources

Blog posts

- Cross-Region disaster recovery of Amazon RDS for SQL Server
- Database Migration—What Do You Need to Know Before You Start?
- Deploying Always On availability groups between Amazon EC2 Windows and Amazon Linux 2 instances
- How to architect a hybrid Microsoft SQL Server solution using distributed availability groups
- How to migrate to Amazon RDS for SQL Server using transactional replication
- Introducing Ongoing Replication from Amazon RDS for SQL Server Using AWS Database Migration Service
- Learn why AWS is the best cloud to run Microsoft Windows Server and SQL Server workloads
- Migrate your on-premises SQL Server Windows Workloads to Amazon EC2 Linux
- Migrating a SQL Server Database to a MySQL-Compatible Database Engine
- Migrating your on-premises SQL Server Windows workloads to Amazon EC2 Linux
- Simplify your Microsoft SQL Server high availability deployments using Amazon FSx for Windows File Server
- Store SQL Server backups in Amazon S3 using AWS Storage Gateway

AWS documentation

- Amazon Aurora
- Amazon EC2
- Amazon RDS
- AWS DMS
- AWS SCT
- CloudEndure Migration
- SQL Server Licensing

Appendix: SQL Server database migration questionnaire

Use the questionnaire in this section as a starting point to gather information for the assessment and planning phases of your migration project. You can download this questionnaire in Microsoft Excel format and use it to record your information.

Download questionnaire

General information

1. What is the name of your SQL Server instance?
2. What is the version of your SQL Server?
3. What is the edition of your SQL Server database: Standard, Developer, or Enterprise?
4. What is the database type (OLTP, DW, reporting, batch processing)?
5. How many databases do you have on the SQL Server instance?
6. What is the size of your database?
7. What is the database collation?
8. What is the time zone of the database?
9. What are the average and maximum I/O transactions per second (TPS)?
10. What is the IOPS (on average and maximum) for this database for read/write operations?
11. How many transaction logs do you generate per hour (with average and maximum size)?
12. Does the database have linked servers pointing to other databases?
13. What are the SLA requirements for your database?
14. What is the RTO and RPO requirements for your database?
15. How much database downtime can you allow for migration purposes?
16. Do you have any compliance, regulatory, or auditing requirements?
17. What tool do you use to monitor your SQL Server databases?

Infrastructure

1. What is the hostname of the database?
2. What is the operating system used for this database?
3. How many CPU cores does the server have?
4. What is the memory size on the server?
5. Is the database on a virtualized machine or a physical server?
6. Are you using local storage?
7. Do you use network-attached storage (NAS) or storage area network (SAN) storage types?
8. Do you have a cluster or single instances?

Database backups

1. How do you back up your database? How often?
2. What is your retention period for transaction logs and backups?
3. Where do you store your backup?

Database features

1. Do you use automatic tuning for your SQL Server instance?
2. Do you use parallel-indexed operations?
3. Do you use partitioned table parallelism features?
4. Do you use table and index partitioning?

Database security

1. Do you use dynamic data masking?
2. Do you use security features such as Transparent Database Encryption (TDE)?
3. Do you use server or database audits?
4. Do you use advanced compression?

Database high availability and disaster recovery

1. What are your high availability requirements?
2. Do you use transactional replication?
3. Do you use peer-to-peer transactional replication?
4. What type of high availability solutions (for example, failover clustering, Always On availability groups, database mirroring) do you use for your SQL Server environment?
5. Where are your primary and standby database regions?
6. What do you use as a disaster recovery solution (for example, log shipping, Always On availability groups, a SAN-based virtualized environment)?
7. Do you use a Domain Name System (DNS) alias for database connectivity?

AWS Prescriptive Guidance glossary

Migration terms (p. 47) | Modernization terms (p. 50)

Migration terms

The following are commonly used terms in migration-related strategies, guides, and patterns provided by AWS Prescriptive Guidance. To suggest entries, please use the **Provide feedback** link at the end of the glossary.

7 Rs

Seven common migration strategies for moving applications to the cloud. These strategies build upon the 5 Rs that Gartner identified in 2011 and consist of the following:

- Refactor/re-architect – Move an application and modify its architecture by taking full advantage of cloud-native features to improve agility, performance, and scalability. This typically involves porting the operating system and database. Example: Migrate your on-premises Oracle database to the Amazon Aurora PostgreSQL-Compatible Edition.
- Replatform (lift and reshape) – Move an application to the cloud, and introduce some level of optimization to take advantage of cloud capabilities. Example: Migrate your on-premises Oracle database to Amazon Relational Database Service (Amazon RDS) for Oracle in the AWS Cloud.
- Repurchase (drop and shop) – Switch to a different product, typically by moving from a traditional license to a SaaS model. Example: Migrate your customer relationship management (CRM) system to Salesforce.com.
- Rehost (lift and shift) – Move an application to the cloud without making any changes to take advantage of cloud capabilities. Example: Migrate your on-premises Oracle database to Oracle on an EC2 instance in the AWS Cloud.
- Relocate (hypervisor-level lift and shift) – Move infrastructure to the cloud without purchasing new hardware, rewriting applications, or modifying your existing operations. This migration scenario is specific to VMware Cloud on AWS, which supports virtual machine (VM) compatibility and workload portability between your on-premises environment and AWS. You can use the VMware Cloud Foundation technologies from your on-premises data centers when you migrate your infrastructure to VMware Cloud on AWS. Example: Relocate the hypervisor hosting your Oracle database to VMware Cloud on AWS.
- Retain (revisit) – Keep applications in your source environment. These might include applications that require major refactoring, and you want to postpone that work until a later time, and legacy applications that you want to retain, because there's no business justification for migrating them.

	• Retire – Decommission or remove applications that are no longer needed in your source environment.
application portfolio	A collection of detailed information about each application used by an organization, including the cost to build and maintain the application, and its business value. This information is key to the portfolio discovery and analysis process and helps identify and prioritize the applications to be migrated, modernized, and optimized.
artificial intelligence operations (AIOps)	The process of using machine learning techniques to solve operational problems, reduce operational incidents and human intervention, and increase service quality. For more information about how AIOps is used in the AWS migration strategy, see the operations integration guide.
AWS Cloud Adoption Framework (AWS CAF)	A framework of guidelines and best practices from AWS to help organizations develop an efficient and effective plan to move successfully to the cloud. AWS CAF organizes guidance into six focus areas called perspectives: business, people, governance, platform, security, and operations. The business, people, and governance perspectives focus on business skills and processes; the platform, security, and operations perspectives focus on technical skills and processes. For example, the people perspective targets stakeholders who handle human resources (HR), staffing functions, and people management. For this perspective, AWS CAF provides guidance for people development, training, and communications to help ready the organization for successful cloud adoption. For more information, see the AWS CAF website and the AWS CAF whitepaper.
AWS landing zone	A landing zone is a well-architected, multi-account AWS environment that is scalable and secure. This is a starting point from which your organizations can quickly launch and deploy workloads and applications with confidence in their security and infrastructure environment. For more information about landing zones, see Setting up a secure and scalable multi-account AWS environment.
AWS Workload Qualification Framework (AWS WQF)	A tool that evaluates database migration workloads, recommends migration strategies, and provides work estimates. AWS WQF is included with AWS Schema Conversion Tool (AWS SCT). It analyzes database schemas and code objects, application code, dependencies, and performance characteristics, and provides assessment reports.
business continuity planning (BCP)	A plan that addresses the potential impact of a disruptive event, such as a large-scale migration, on operations and enables a business to resume operations quickly.
Cloud Center of Excellence (CCoE)	A multi-disciplinary team that drives cloud adoption efforts across an organization, including developing cloud best practices, mobilizing resources, establishing migration timelines, and leading the organization through large-scale transformations. For more information, see the CCoE posts on the AWS Cloud Enterprise Strategy Blog.
cloud stages of adoption	The four phases that organizations typically go through when they migrate to the AWS Cloud: • Project – Running a few cloud-related projects for proof of concept and learning purposes • Foundation – Making foundational investments to scale your cloud adoption (e.g., creating a landing zone, defining a CCoE, establishing an operations model) • Migration – Migrating individual applications • Re-invention – Optimizing products and services, and innovating in the cloud

	These stages were defined by Stephen Orban in the blog post The Journey Toward Cloud-First & the Stages of Adoption on the AWS Cloud Enterprise Strategy blog. For information about how they relate to the AWS migration strategy, see the migration readiness guide.
configuration management database (CMDB)	A database that contains information about a company's hardware and software products, configurations, and inter-dependencies. You typically use data from a CMDB in the portfolio discovery and analysis stage of migration.
epic	In agile methodologies, functional categories that help organize and prioritize your work. Epics provide a high-level description of requirements and implementation tasks. For example, AWS CAF security epics include identity and access management, detective controls, infrastructure security, data protection, and incident response. For more information about epics in the AWS migration strategy, see the program execution guide.
heterogeneous database migration	Migrating your source database to a target database that uses a different database engine (for example, Oracle to Amazon Aurora). Heterogenous migration is typically part of a re-architecting effort, and converting the schema can be a complex task. AWS provides AWS SCT that helps with schema conversions.
homogeneous database migration	Migrating your source database to a target database that shares the same database engine (for example, Microsoft SQL Server to Amazon RDS for SQL Server). Homogeneous migration is typically part of a rehosting or replatforming effort. You can use native database utilities to migrate the schema.
IT information library (ITIL)	A set of best practices for delivering IT services and aligning these services with business requirements. ITIL provides the foundation for ITSM.
IT service management (ITSM)	Activities associated with designing, implementing, managing, and supporting IT services for an organization. For information about integrating cloud operations with ITSM tools, see the operations integration guide.
Migration Acceleration Program (MAP)	An AWS program that provides consulting support, training, and services to help organizations build a strong operational foundation for moving to the cloud, and to help offset the initial cost of migrations. MAP includes a migration methodology for executing legacy migrations in a methodical way and a set of tools to automate and accelerate common migration scenarios.
Migration Portfolio Assessment (MPA)	An online tool that provides information for validating the business case for migrating to the AWS Cloud. MPA provides detailed portfolio assessment (server right-sizing, pricing, TCO comparisons, migration cost analysis) as well as migration planning (application data analysis and data collection, application grouping, migration prioritization, and wave planning). The MPA tool (requires login) is available free of charge to all AWS consultants and APN Partner consultants.
Migration Readiness Assessment (MRA)	The process of gaining insights about an organization's cloud readiness status, identifying strengths and weaknesses, and building an action plan to close identified gaps, using the AWS CAF. For more information, see the migration readiness guide. MRA is the first phase of the AWS migration strategy.
migration at scale	The process of moving the majority of the application portfolio to the cloud in waves, with more applications moved at a faster rate in each wave. This phase uses the best practices and lessons learned from the earlier phases to implement a *migration factory* of teams, tools, and processes to streamline the migration of

	workloads through automation and agile delivery. This is the third phase of the AWS migration strategy.
migration factory	Cross-functional teams that streamline the migration of workloads through automated, agile approaches. Migration factory teams typically include operations, business analysts and owners, migration engineers, developers, and DevOps professionals working in sprints. Between 20 and 50 percent of an enterprise application portfolio consists of repeated patterns that can be optimized by a factory approach. For more information, see the discussion of migration factories and the CloudEndure Migration Factory guide in this content set.
operational-level agreement (OLA)	An agreement that clarifies what functional IT groups promise to deliver to each other, to support a service-level agreement (SLA).
operations integration (OI)	The process of modernizing operations in the cloud, which involves readiness planning, automation, and integration. For more information, see the operations integration guide.
organizational change management (OCM)	A framework for managing major, disruptive business transformations from a people, culture, and leadership perspective. OCM helps organizations prepare for, and transition to, new systems and strategies by accelerating change adoption, addressing transitional issues, and driving cultural and organizational changes. In the AWS migration strategy, this framework is called *people acceleration*, because of the speed of change required in cloud adoption projects. For more information, see the OCM guide.
playbook	A set of predefined steps that capture the work associated with migrations, such as delivering core operations functions in the cloud. A playbook can take the form of scripts, automated runbooks, or a summary of processes or steps required to operate your modernized environment.
responsible, accountable, consulted, informed (RACI) matrix	A matrix that defines and assigns roles and responsibilities in a project. For example, you can create a RACI to define security control ownership or to identify roles and responsibilities for specific tasks in a migration project.
runbook	A set of manual or automated procedures required to perform a specific task. These are typically built to streamline repetitive operations or procedures with high error rates.
service-level agreement (SLA)	An agreement that clarifies what an IT team promises to deliver to their customers, such as service uptime and performance.

Modernization terms

The following are commonly used terms in modernization-related strategies, guides, and patterns provided by AWS Prescriptive Guidance. To suggest entries, please use the **Provide feedback** link at the end of the glossary.

business capability	What a business does to generate value (for example, sales, customer service, or marketing). Microservices architectures and development decisions can be driven by business capabilites. For more information, see the Organized around business capabilities section of the Running containerized microservices on AWS whitepaper.
microservice	A small, independent service that communicates over well-defined APIs and is typically owned by small, self-contained teams. For example, an insurance system might include microservices that map to business capabilities, such as sales or marketing, or subdomains, such as purchasing, claims, or analytics. The benefits

	of microservices include agility, flexible scaling, easy deployment, reusable code, and resilience. For more information, see Integrating microservices by using AWS serverless services.
microservices architecture	An approach to building an application with independent components that run each application process as a microservice. These microservices communicate through a well-defined interface by using lightweight APIs. Each microservice in this architecture can be updated, deployed, and scaled to meet demand for specific functions of an application. For more information, see Implementing microservices on AWS.
modernization	Transforming an outdated (legacy or monolithic) application and its infrastructure into an agile, elastic, and highly available system in the cloud to reduce costs, gain efficiencies, and take advantage of innovations. For more information, see Strategy for modernizing applications in the AWS Cloud.
modernization readiness assessment	An evaluation that helps determine the modernization readiness of an organization's applications; identifies benefits, risks, and dependencies; and determines how well the organization can support the future state of those applications. The outcome of the assessment is a blueprint of the target architecture, a roadmap that details development phases and milestones for the modernization process, and an action plan for addressing identified gaps. For more information, see Evaluating modernization readiness for applications in the AWS Cloud.
monolithic applications (monoliths)	Applications that run as a single service with tightly coupled processes. Monolithic applications have several drawbacks. If one application feature experiences a spike in demand, the entire architecture must be scaled. Adding or improving a monolithic application's features also becomes more complex when the code base grows. To address these issues, you can use a microservices architecture. For more information, see Decomposing monoliths into microservices.
polyglot persistence	Independently choosing a microservice's data storage technology based on data access patterns and other requirements. If your microservices have the same data storage technology, they can encounter implementation challenges or experience poor performance. Microservices are more easily implemented and achieve better performance and scalability if they use the data store best adapted to their requirements. For more information, see Enabling data persistence in microservices.
split-and-seed model	A pattern for scaling and accelerating modernization projects. As new features and product releases are defined, the core team splits up to create new product teams. This helps scale your organization's capabilities and services, improves developer productivity, and supports rapid innovation. For more information, see Phased approach to modernizing applications in the AWS Cloud.
two-pizza team	A small DevOps team that you can feed with two pizzas. A two-pizza team size ensures the best possible opportunity for collaboration in software development. For more information, see the Two-pizza team section of the Introduction to DevOps on AWS whitepaper.

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.

update-history-change	update-history-description	update-history-date
Updated log shipping information (p. 52)	In the Log shipping section, clarified that log shipping on Amazon RDS for SQL Server requires custom scripts.	March 25, 2021
Initial publication (p. 52)	—	October 9, 2020