AWS Prescriptive Guidance

Migration strategy for relational databases

AWS Prescriptive Guidance: Migration strategy for relational databases

Table of Contents

Migration strategy for relational databases

Yaser Raja, Senior Consultant, AWS Professional Services

December 2019 (last update (p. 21): November 2020)

In your enterprise portfolio, you are likely to have multiple types of databases. When you migrate to Amazon Web Services (AWS), you can choose to do a "lift and shift" of your databases (rehost) or modernize your applications by switching to AWS managed database services (replatform).

If you choose to rehost your database, AWS provides a number of services and tools that can help you securely move, store and analyze your data. If you choose to switch to an AWS managed database service, AWS offers a multitude of options so you never have to trade off functionality, performance, or scale. For more information about the AWS family of databases, see Databases on AWS on the AWS website.

This document focuses on strategies for migrating relational databases to the AWS Cloud, for IT and business executives, program or project managers, product owners, and operations/infrastructure managers who are planning to migrate their on-premises databases to AWS.

Overview

The best database migration strategy enables you to take full advantage of the AWS Cloud. This involves migrating your applications to use purpose-built, cloud-native databases. You shouldn't limit yourself to the same old-guard database that you have been using on premises. Instead, consider modernizing your applications and choose the databases that best suit your applications' workflow requirements.

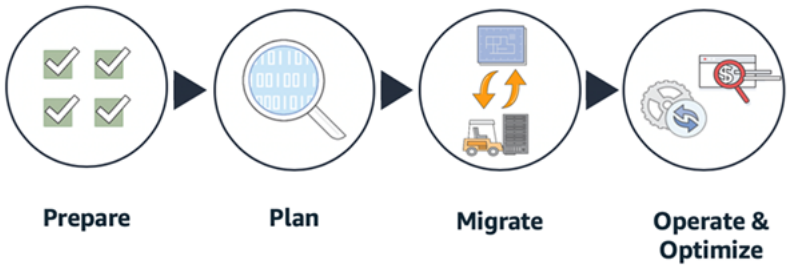
Many enterprises have adopted this approach. For example, Airbnb needed to quickly process and analyze 50 GB of data daily. They needed a key-value database to store user search history for quick lookups that enabled personalized search, an in-memory data store to store session state for faster (sub-millisecond) site rendering, and a relational database as their primary transactional database. They chose Amazon DynamoDB as their key-value database, Amazon ElastiCache as their in-memory store, and Amazon Relational Database Service (Amazon RDS) for their transactional database. For more information about how Airbnb is using AWS database services, see the Airbnb case study.

Database migration strategy is tied closely to your organization's overarching cloud strategy. For example, if you choose to first transition your applications and then transform them, you might decide to lift and shift your database first. When you are fully in the AWS Cloud, you start working to modernize your application. This strategy can help you exit out of your current data centers quickly, and then focus on modernization.

Your database migration is tightly coupled with your application migration. All database migration strategies involve some level of changes to the applications that use those databases. These changes range from pointing to the new location of the database in the AWS Cloud to a total rewrite of the application, if it can't be changed because the source code isn't available, or it's a closed-source, third-party application.

Phases of migration

When you've identified a database for migration, you go through the phases of preparation, planning, migration, and optimization of the database.



The following sections discuss each phase in detail:

Phase 1: Prepare

The first phase of the database migration process is preparation. During preparation, you identify the interdependencies between your applications and databases. You also analyze the database workloads to determine the migration categories: from simple rehost (homogeneous) migration to re-architect (heterogeneous) migration. Without completing this phase, you risk running into delayed migration timelines.

These tasks are discussed in the following sections:

Identify dependencies

You start by identifying application and database dependencies, by asking questions such as the following:

- Is this database directly accessed by any other application?

  If so, you should determine how migrating the database affects that application. If you're rehosting the database, you need to make sure that the application can still access the database with acceptable performance.
- Does the application directly access any other database?

  If so, determine the migration plan for the other database. If it's also migrating, you need to update the application accordingly. If it isn't migrating, you need to make sure that the application can continue to connect to it with acceptable latency.
- Is the database using database links to fetch data from other databases?

  As in the previous point, determine the migration plan for the other database and handle the links accordingly.
- Is the application dependent on any on-premises software?

  If so, you should determine the migration plan for that software. If it's migrating, you need to update your application accordingly. If it isn't, make sure that the application can continue to connect to the software and the latency is acceptable.
- Are there any hardware dependencies?

  If so, come up with a plan to address those.
- Are there any strict bandwidth or networking requirements?

  If so, choose the AWS services that can help you meet these requirements.
- Does the application use any special database engine options or features?

  If you're migrating to a different database engine, you need to update the application accordingly.

If the answers to these questions are complex, a better option is to decouple the database from the application by using microservices. This way, an application can get data by calling the microservice instead of directly connecting to the database.

Qualify workloads

To determine the best migration strategy for your database, it's important to understand the current database workload. You need to analyze your database to determine which features you are currently using and what's involved in migrating to another cloud-native database engine such as Amazon Aurora PostgreSQL.

AWS provides a workload qualification tool called AWS Workload Qualification Framework (AWS WQF). This tool can help identify the complexity of your Oracle and Microsoft SQL Server database migration by analyzing database schemas and code objects, application code, dependencies, performance characteristics, and similar inputs. WQF provides recommendations on the target database engine. It also estimates the type of work involved and the level of effort required.

WQF evaluates your migration workload and places it in one of five workload categories, summarized in the following table.

	Category 1	ODBC/JBDC workloads	< 50 manual changes, easy to refactor
	Category 2	Light, proprietary feature workloads	< 200 manual changes, medium complexity
	Category 3	Heavy, proprietary feature workloads	> 200 manual changes, high complexity
	Category 4	Engine-specific workloads	Not recommended for refactoring
	Category 5	COTS or other non-portable workloads	Not recommended for refactoring

- **Category 1:** Workloads that use Open Database Connectivity (ODBC) or Java Database Connectivity (JDBC) instead of proprietary drivers to connect to the database. This category typically has simple stored procedures that are used for access controls. The conversion requires fewer than 50 manual changes.
- **Category 2:** Workloads with light use of proprietary features and that don't use advanced SQL language features. This type of workload requires fewer than 200 manual changes.
- **Category 3:** Workloads with heavy use of proprietary features. Workloads in this category are completely driven by advanced stored procedure logic or proprietary features. This type of workload requires more than 200 manual changes that involve database-resident code and features.
- **Category 4:** Engine-specific workloads. Workloads in this category use frameworks that can work only with a specific commercial database engine. For example, these frameworks might include Oracle Forms, Oracle Reports, Oracle Application Development Framework (ADF), Oracle Application Express (APEX), or applications that use .NET ActiveRecord extensively.
- **Category 5:** Nonportable, unacceptable risk, or "lift and shift" workloads. Workloads in this category might be implemented on database engines that have no cloud-based equivalent. In some cases, you might not have the source code for these programs.

This categorization can help you determine the migration path for your application, as we'll discuss in the section Phase 2: Plan (p. 5).

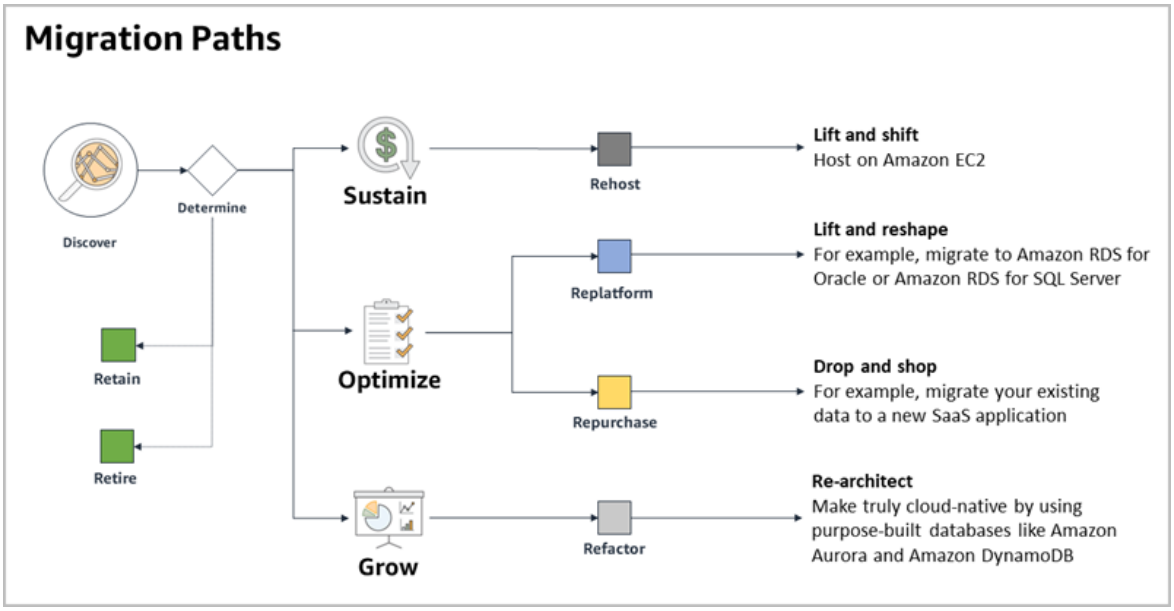
AWS doesn't currently provide AWS WQF for downloading. If you need help assessing a migration to AWS with AWS WQF, we recommend opening a support ticket. AWS will engage with you directly to help make the process work for you.

Phase 2: Plan

In this phase, you use the information gathered during the preparation phase and come up with the migration strategy. A critical aspect of migration planning is rationalizing the information you collected against the 6 Rs of migration: rehost, replatform, refactor/re-architect, repurchase, retire, and retain.

Choosing your migration strategy depends on your business drivers for cloud adoption, as well as time considerations, business and financial constraints, and resource requirements. If you want to sustain your current workload in the cloud, choose rehosting. However, if you want to optimize and scale your workloads, consider one of the other options.

Here's an overview of the 6 Rs of database migration. These are illustrated in the following diagram and discussed in detail in the AWS migration whitepaper.



- **Rehost** (lift and shift) – Move an application to the cloud without making any changes. For example, migrate your on-premises Oracle database to Oracle on an Amazon Elastic Compute Cloud (Amazon EC2) instance in the AWS Cloud.
- **Replatform** (lift and reshape) – Move an application to the cloud, and introduce some level of optimization to take advantage of cloud capabilities. For example, migrate your on-premises Oracle database to Amazon RDS for Oracle in the AWS Cloud.
- **Refactor** (re-architect) – Move an application and modify its architecture by taking full advantage of cloud-native features to improve agility, performance, and scalability. For example, migrate your on-premises Oracle database to Aurora PostgreSQL. This strategy can also include rewriting your application to use the purpose-built databases that AWS offers for different workflows. Or, you can choose to modernize your monolithic application by breaking it down into smaller microservices that access their own database schemas.
- **Repurchase** (drop and shop) – Change to a different product, typically by moving from a traditional application to a software as a service (SaaS) product, and migrate data from your on-premises application to the new product. For example, migrate your customer data from your on-premises customer relationship management (CRM) system to Salesforce.com.
- **Retire** – Decommission or remove applications that are no longer needed in your source environment.

- **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.

Choose a migration strategy

In the majority of the database migrations, you can choose to rehost, replatform, or refactor. Any of these strategies can work for you. The guiding principle should be how you can get the maximum benefit out of your migration. Choosing to refactor your application and migrate to a cloud-native database such as Aurora can enable you to enhance your database application. However, depending on your workload complexity, refactoring a database can be time-consuming and resource-intensive.

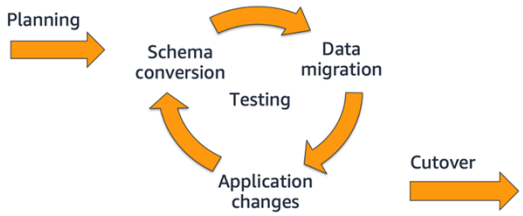
The WQF categorization helps you decide when you should consider a particular migration strategy. A higher WQF category means that the migration effort required is significant; therefore, you might want to choose another option, such as rehost or replatform, to complete the migration within an acceptable timeframe. The following table shows the suggested strategies based on the WQF category.

Category	Workload complexity	Workload	Migration strategy
1		ODBC/JBDC workloads	Candidate for refactor
2		Light, proprietary feature workloads	Candidate for refactor
3		Heavy, proprietary feature workloads	Candidate for refactor or replatform
4		Engine-specific workloads	Candidate for replatform or rehost
5		Non-portable, high-risk, or lift-and-shift workloads	Candidate for replatform or rehost

The rehost and replatform options are suitable when the complexity involved in refactoring is high. In these scenarios, based on your modernization needs, you might consider refactoring your database after you have completed the migration to the AWS Cloud.

Phase 3: Migrate

After you complete migration planning and identify a migration strategy, the actual migration takes place. In this phase, the target database is designed, the source data is migrated to the target, and the data is validated.



This is an iterative process that includes multiple cycles of conversion, migration, and testing. After the functional and performance testing is complete, you can cut over to the new database.

The migration phase consists of the following key steps, which are discussed in the following sections:

- Converting the schema (p. 7)
- Migrating the data (p. 8)
- Updating the application (p. 9)
- Testing the migration (p. 9)
- Cutting over to the new database (p. 10)

Convert the schema

One of the key tasks during the database migration is to migrate your schema from the source database engine to the target database engine. If you rehost or replatform, your database engine won't change. This is referred to as a *homogeneous database migration*, and you can use your native database tools to migrate the schema.

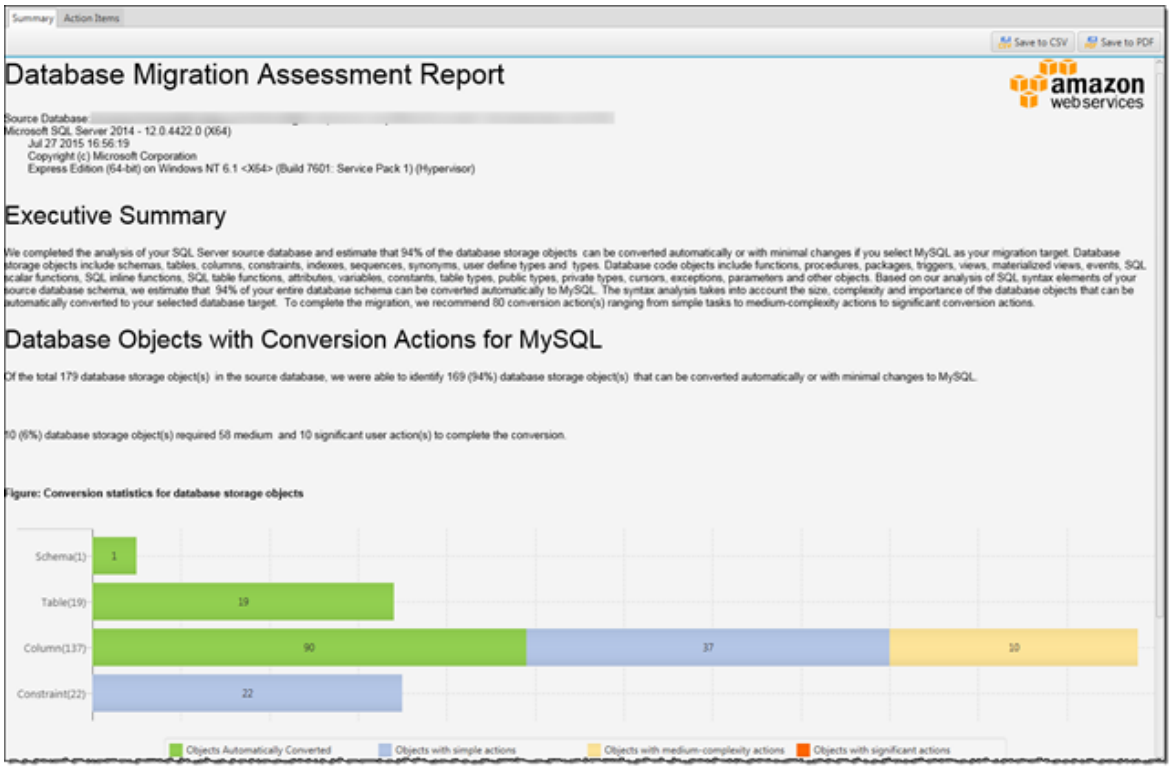
However, if you are rearchitecting your application, schema conversion might require more effort. In this case, you will be doing a *heterogeneous database migration*, where your source and target database engines will be different. Your current database schema may be using packages and features that cannot be directly converted to the target database engine. Some features might be available under a different name. Therefore, converting the schema requires a good understanding of your source and target database engines. This task can be challenging, depending on the complexity of your current schema.

AWS provides two resources to help you with schema conversion: AWS Schema Conversion Tool (AWS SCT) and migration playbooks.

AWS SCT

AWS SCT is a free tool that can help you convert your existing database from one engine to another. AWS SCT supports a number of source databases, including Oracle, Microsoft SQL Server, MySQL, Sybase, and IBM Db2 LUW. You can choose from target databases such as Aurora MySQL and Aurora PostgreSQL.

AWS SCT provides a graphical user interface that directly connects to the source and target databases to fetch the current schema objects. When connected, you can generate a database migration assessment report to get a high-level summary of the conversion effort and action items. The following screen illustration shows a sample database migration assessment report.



With AWS SCT you can convert the schema and deploy it into the target database directly, or you can get SQL files for the converted schema. For more information, see Using the AWS Schema Conversion Tool User Interface in the AWS documentation.

Migration playbooks

Although AWS SCT converts many of your source objects, some aspects of conversion require manual intervention and adjustments. To help with this task, AWS provides migration playbooks that detail incompatibilities and similarities between two databases. For more information about these playbooks, see AWS Database Migration Service resources on the AWS website.

Migrate the data

When the schema migration is complete, you can move your data from the source database to the target database. Depending on your application availability requirements, you can run a simple extraction job that performs a one-time copy of the source data into the new database. Or, you can use a tool that copies the current data and continues to replicate all changes until you are ready to cut over to the new database. For rehost and replatform migrations, we recommend that you use native database-specific tools to migrate your data.

Tools that can help you with the data transfer include AWS Database Migration Service (AWS DMS) and offline migration tools. These are described in the following sections.

AWS DMS

After you use AWS SCT to convert your schema objects from the source database engine to the target engine, you can use AWS DMS to migrate the data. With AWS DMS you can keep the source database

up and running while the data is being replicated. You can perform a one-time copy of your data or copy with continuous replication. When the source and target databases are in sync, you can take your database offline and move your operations to the target database. AWS DMS can be used for homogeneous database migrations (for example, from an on-premises Oracle database to an Amazon RDS for Oracle database) as well as heterogeneous migrations (for example, from an on-premises Oracle database to an Amazon RDS for PostgreSQL database). For more information about working with AWS DMS, see the AWS DMS documentation.

Offline migration options

You can use other options in addition to AWS DMS to extract your data from the source database and load it to the target database. These options are mostly suitable when application downtime is allowed during the data migration activity. Examples of these methods include:

- A comma-separate values (CSV) extract from the source database loaded to the target database
- For Oracle source databases, the **ora2pg** utility to copy the data to PostgreSQL
- Custom extract, transform, load (ETL) jobs to copy the data from source to target

Update the application

A database migration is hardly ever a database-only migration. You have to look at the application that's using the database to make sure that it works as expected with the new database. The changes are minimal if you are simply rehosting or replatforming the same database engine, but can be more significant if you decide to move to a new database engine.

If your application relies on an object-relational mapping (ORM) to interact with the database, it won't require as many changes when you migrate to a new database engine. However, if your application has custom database interactions or dynamically built SQL queries, the changes can be sizable. There might be differences in the query formats that need to be corrected to make sure that the application works as expected.

For example, in Oracle, concatenating a string with `NULL` returns the original string. However, in PostgreSQL, concatenating a string with `NULL` returns `NULL`. Another example is how `NULL` and empty strings are treated. In PostgreSQL, `NULL` and empty strings are two different things, whereas databases like Oracle treat them in the same way. In Oracle, if you insert a row with the column value set to `NULL` or empty string, you can fetch both types of values by using the `where` clause: `where <mycolumn> is NULL`. In PostgreSQL, this `where` clause will return only one row where the column value is actually NULL; it won't return the row that has an empty string value. For more information about these differences, see the migration playbooks listed on the AWS Database Migration Service resources webpage.

Test the migration

Functional and performance testing is an essential part of database migrations. Detailed functional testing will make sure that your application is working with the new database without any issues. You should invest time to develop unit tests to test out the application workflows.

Performance testing makes sure that your database response times are within an acceptable time range. You can identify bottlenecks, optimize, and repeat the performance test. You repeat the cycle as required to get the desired performance results.

Testing can be manual or automated. We recommend that you use an automated framework for testing. During migration, you will need to run the test multiple times, so having an automated testing framework helps speed up the bug fixing and optimization cycles.

This testing can reveal issues that were missed during development phases. For example, any incorrectly converted queries will fail or return incorrect results, causing the functional testing to fail. Performance testing can reveal issues such as missing indexes causing slow query response time. They can also reveal performance issues that require database engine tuning, depending on the workload, or modifying the query.

Cut over

The database cutover strategy is usually tightly coupled with the downtime requirements for the application. Strategies that you can use for the database cutover include offline migration, flash-cut migration, active/active database configuration, and incremental migration. These are discussed in the following sections.

Offline migration

If you can take your application offline for an extended period during write operations, you can use AWS DMS full-load task settings or one of the offline migration options for your data migration. The read traffic can continue while this migration is in progress, but the write traffic must be stopped. Because all the data needs to be copied from the source database, source database resources such as I/O and CPU are utilized.

At a high level, offline migration involves these steps:

1. Complete the schema conversion.
2. Start downtime for write traffic.
3. Migrate the data using one of the offline migration options.
4. Verify your data.
5. Point your application to the new database.
6. End the application downtime.

Flash-cut migration

In flash-cut migration, the main objective is to keep the downtime to a minimum. This strategy relies on continuous data replication (CDC) from the source database to the target database. All read/write traffic will continue on the current database while the data is being migrated. Because all the data needs to be copied from the source database, source server resources such as I/O and CPU are utilized. You should test to make sure that this data migration activity doesn't impact your application performance SLAs.

At a high level, flash-cut migration involves these steps:

1. Complete the schema conversion.
2. Set up AWS DMS in continuous data replication mode.
3. When the source and target databases are in sync, verify the data.
4. Start the application downtime.
5. Roll out the new version of the application, which points to the new database.
6. End the application downtime.

Active/active database configuration

Active/active database configuration involves setting up a mechanism to keep the source and target databases in sync while both databases are being used for write traffic. This strategy involves more work

than offline or flash-cut migration, but it also provides more flexibility during migration. For example, in addition to experiencing minimal downtime during migration, you can move your production traffic to the new database in small, controlled batches instead of performing a one-time cutover. You can either perform dual write operations so that changes are made to both databases, or use a bi-directional replication tool like HVR to keep the databases in sync. This strategy has a higher complexity in terms of setup and maintenance, so more testing is required to avoid data consistency issues.

At a high level, active/active database configuration involves these steps:

1. Complete the schema conversion.
2. Copy the existing data from the source database to the target database, and then keep the two databases in sync by using a bi-directional replication tool or dual writes from the application.
3. When the source and target databases are in sync, verify the data.
4. Start moving a subset of your traffic to the new database.
5. Keep moving the traffic until all your database traffic has been moved to the new database.

Incremental migration

In incremental migration, you migrate your application in smaller parts instead of performing a one-time, full cutover. This cutover strategy could have many variations, based on your current application architecture or the refactoring you're willing to do in the application.

You can use a design pattern to add new independent microservices to replace parts of an existing, monolithic legacy application. These independent microservices have their own tables that are not shared or accessed by any other part of the application. You migrate these microservices to the new database one by one, using any of the other cutover strategies. The migrated microservices start using the new database for read/write traffic while all other parts of the application continue to use the old database. When all microservices have been migrated, you decommission your legacy application. This strategy breaks up the migration into smaller, manageable pieces and can, therefore, reduce the risks that are associated with one big migration.

Follow best practices on AWS

In addition to the migration activities discussed in the previous sections, you should invest time to make sure that you are following the best practices to host your database in the AWS Cloud. See the AWS documentation for best practices for working with relational databases on AWS.

Phase 4: Operate and optimize

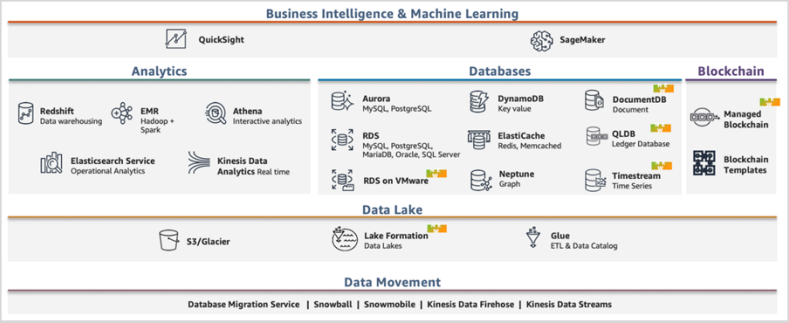
When your database is in AWS, you have to operate it in the cloud. You need to make sure that you are following the best practices for areas such as monitoring, alerting, backups, and high availability. The operation overhead of rehosted databases is higher than the databases that have been replatformed or refactored to use a managed AWS database service:

- A rehosted database runs on an EC2 instance. You're responsible for all database management tasks such as setting up backups, high availability, and disaster recovery solutions.
- If you replatform or refactor your database on Amazon RDS, these database management tasks require only a few clicks to set up. This means that the database administrator will spend less time managing a database in Amazon RDS, compared with managing a rehosted database on an EC2 instance. Amazon RDS also provides a performance monitoring tool called Amazon RDS Performance Insights, which enables even non-experts to detect performance problems by using an easy-to-understand dashboard that visualizes database load.

No matter which migration option you choose, Amazon CloudWatch plays a very important role in collecting key metrics such as CPU, memory, and I/O utilization. It also provides the capability to set thresholds on metrics and to initiate actions when the given threshold is crossed. For example, you can create alarms on Aurora PostgreSQL cluster metrics, set notifications, and take actions to detect and shut down unused or underutilized reader instances. Setting real-time alarms on metrics and events enables you to minimize downtime and potential business impact.

In the operate and optimize phase, you can maximize the benefits derived from hosting applications on AWS. The optimizing activities can address cost, performance, security, or resiliency concerns for your application stack. For example, you can use automatic scaling features to add more read replicas during peak hours, and remove them during off-peak hours to lower costs. You can also use a number of AWS services that integrate seamlessly with Amazon RDS databases. For example, you can easily direct database engine logs to Amazon CloudWatch Logs for analysis.

Once you are in the AWS Cloud, you can start optimizing your application by taking advantage of a large number of services and features that you can spin up with few clicks. You can innovate faster, because you can focus your highly valuable IT resources on developing applications that differentiate your business and transform your users' experiences, instead of focusing on the undifferentiated heavy lifting of managing infrastructure and data centers. The following diagram shows some of the options provided by AWS services.



In addition, you have the ability to deploy globally in minutes. For example, with a few clicks you can create an Amazon Aurora Global Database that lets you easily scale database read operations across the world and place your applications close to your users.

Similarly, you can use integrations to get more value out of your data. For example, you can use machine learning (ML) capabilities in your Aurora database applications with a few simple steps.

Using AWS 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 Oracle to Oracle, and heterogeneous migrations between different database platforms, such as Oracle to Amazon Aurora or Microsoft SQL Server to 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.

Next steps

For more information about migrating your Oracle Database and SQL Server workloads, see the following guides on the AWS Prescriptive Guidance website:

- Migrating Oracle databases to the AWS Cloud
- Migrating SQL Server databases to the AWS Cloud

For step-by-step instructions for migrating specific relational databases, see the database migration patterns. You can use the filters on that page to view patterns by AWS service (for example, migrations to Aurora), by workload (for example, Oracle database migrations), by planned use (production or pilot), or by migration strategy (re-architect, rehost, relocate, or replatform).

Resources

- Migrating Oracle databases to the AWS Cloud
- Migrating SQL Server databases to the AWS Cloud
- AWS DMS documentation
- AWS SCT documentation
- Migration playbooks
- AWS database options
- General information about AWS managed database services:
  - Amazon RDS
  - Amazon Aurora
  - Amazon RDS for MySQL
  - Amazon RDS for Oracle
  - Amazon RDS for PostgreSQL
  - Amazon RDS for SQL Server
- Amazon RDS documentation

AWS Prescriptive Guidance glossary

6 Rs

Six 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:

- 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.
- 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 RDS for Oracle in the AWS Cloud.
- 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 Amazon Aurora PostgreSQL.
- Repurchase (drop and shop) – Switch to a different product, typically by moving from a traditional license to an SaaS model. Example: Migrate your customer relationship management (CRM) system to Salesforce.com.
- Retire – Decommission or remove applications that are no longer needed in your source environment.
- 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.

For details, see the AWS migration whitepaper.

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 Schema Conversion Tool (AWS SCT) can help 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 Readiness and Planning (MRP)	The process of gaining hands-on migration experience, including tools, processes, and best practices, to prepare your organization for cloud migration. This phase involves migrating 10 to 30 business applications to lay the foundation for migrating your systems at scale. It consists of a defined set of activities across eight workstreams. MRP is the second 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.

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 from the top navigation bar on this page.

update-history-change	update-history-description	update-history-date
Updated AWS WQF information (p. 21)	We updated the Qualify Workloads section with the latest information about AWS WQF.	November 5, 2020
— (p. 21)	Initial publication	December 15, 2019