

# Choosing an AWS database service



# Choosing an AWS database service: AWS Decision Guide

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

# Table of Contents

**Decision guide** ..... **i**

Introduction ..... 1

Understand ..... 2

Consider ..... 5

Choose ..... 8

Use ..... 11

Explore ..... 18

**Document history** ..... **20**

# Choosing an AWS database service

## Taking the first step

Purpose	Help determine which AWS database or databases are the best fit for your organization.
Last updated	December 22, 2024
Covered services	<ul style="list-style-type: none"><li><a href="#">Amazon Aurora</a></li><li><a href="#">Amazon DocumentDB (with MongoDB compatibility)</a></li><li><a href="#">Amazon DynamoDB</a></li><li><a href="#">Amazon ElastiCache</a></li><li><a href="#">Amazon Keyspaces (for Apache Cassandra)</a></li><li><a href="#">Amazon MemoryDB</a></li><li><a href="#">Amazon Neptune</a></li><li><a href="#">Amazon Relational Database Service (Amazon RDS)</a></li><li><a href="#">Amazon Timestream</a></li></ul>

## Introduction

AWS offers a growing number of database options (15+) with diverse data models to support a variety of workloads. These include relational, key-value, document, in-memory, graph, time series, vector, and wide-column.

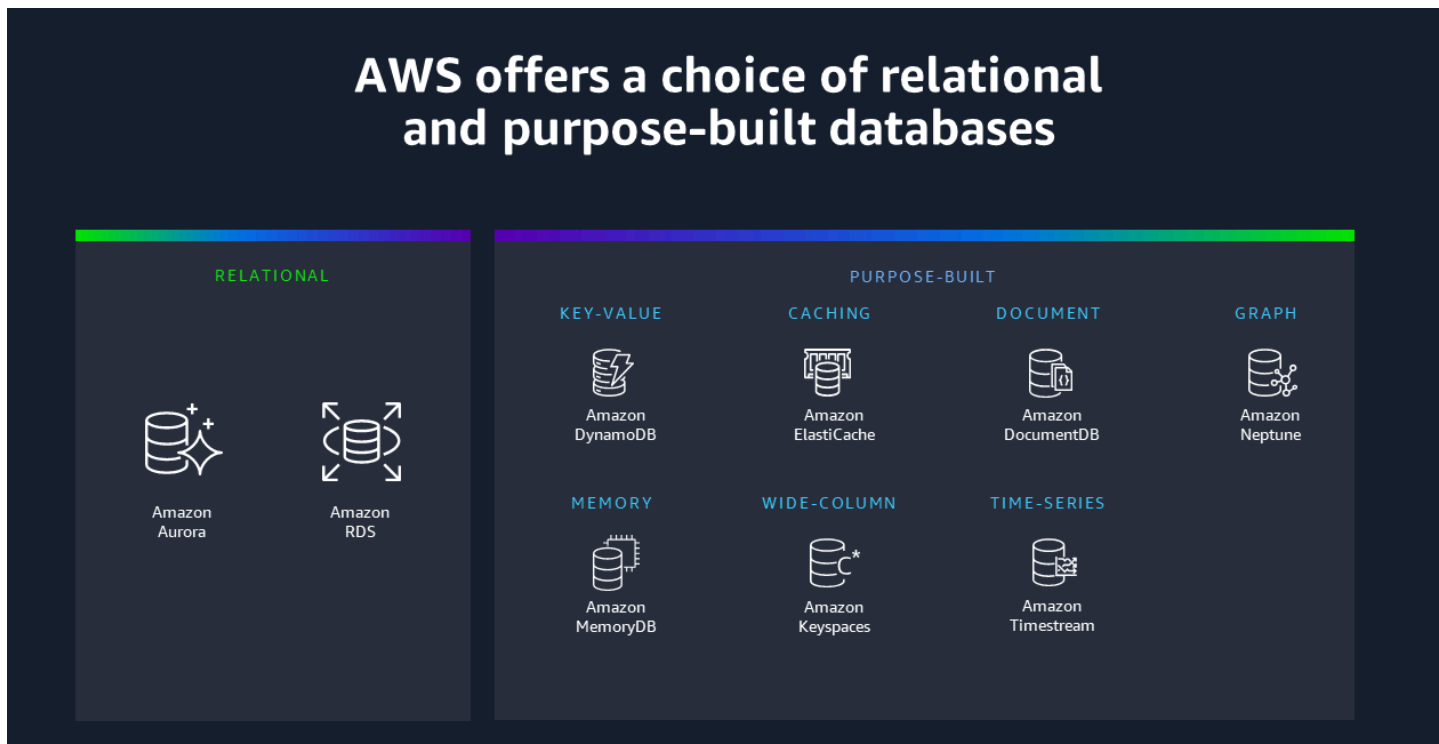
Choosing the right database, or multiple databases, requires you to make a series of decisions based on your organizational needs. This decision guide will help you ask the right questions, provide a clear path for implementation, and help you migrate from your existing database.

[This video explores the broad range of AWS database services outlined at AWS re:Invent 2024..](#)

# Understand

Databases are essential backend systems that efficiently store, manage, and retrieve data, ensuring integrity, scalability, and performance for applications of all types and sizes.

This decision guide is designed to help you understand the range of choices that are available, establish the criteria for making your database choice, and provide you with detailed information on the unique properties of each database. Then you can learn more about the capabilities that each database offers.

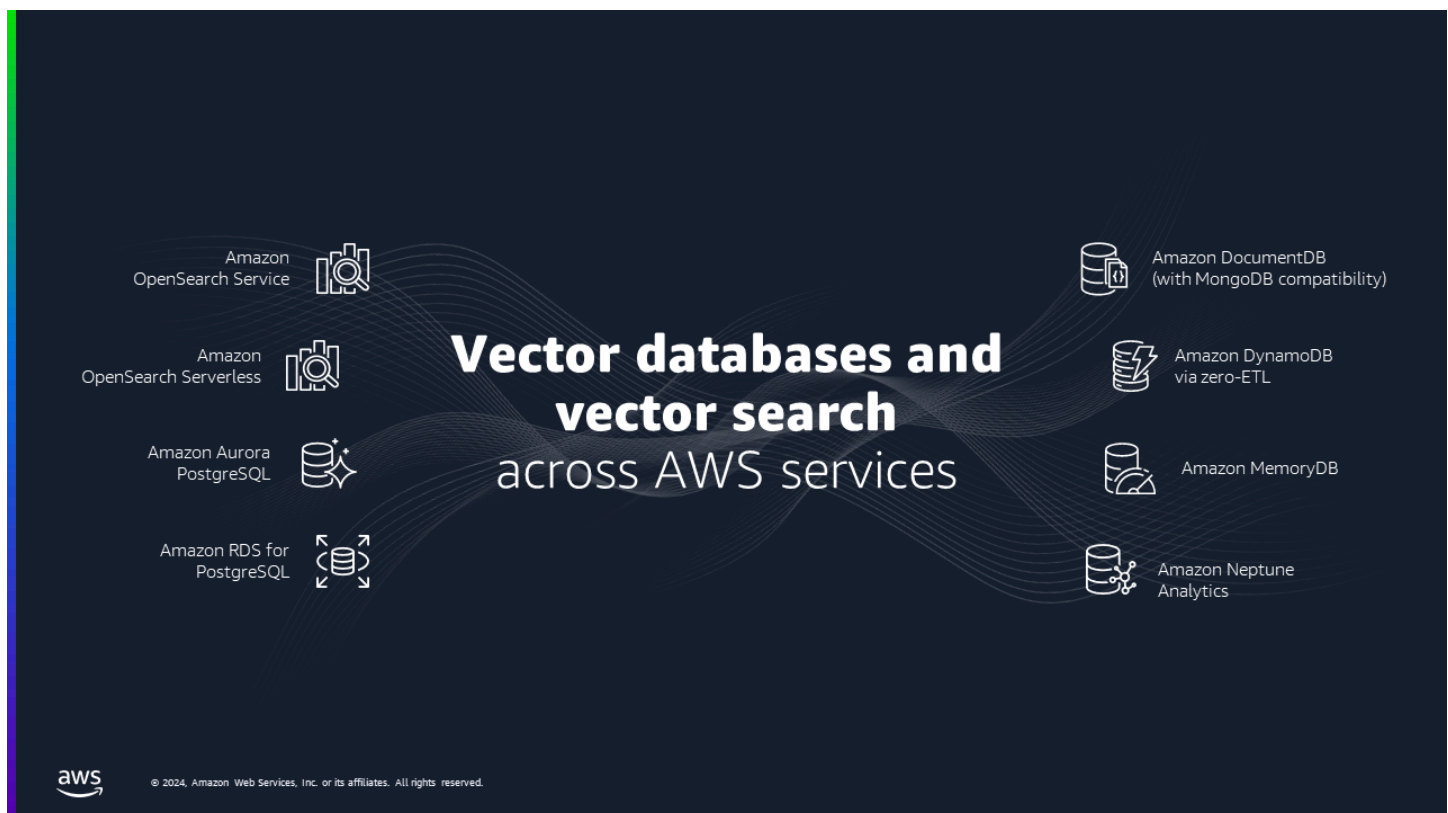


## What are the properties of applications that people build with AWS databases?

- **Internet-scale applications:** These applications can handle 100 million+ requests per second over hundreds of terabytes of data. They automatically scale vertically and horizontally to provide flexibility for your workloads.
- **Real-time applications:** Real-time applications such as caching, session stores, gaming leaderboards, ride hailing, ad targeting, and real-time analytics need microsecond latency and high throughput to support a trillion+ requests per second.
- **Enterprise applications:** Enterprise applications manage core business processes (such as sales, billing, customer service, and human resources) and line-of-business processes (such as

a reservation system at a hotel chain or a risk-management system at an insurance company). These applications need databases that are fast, scalable, secure, available, and reliable.

- **Vector databases and vector search for use with generative AI applications:** Whatever database service you use will likely contain a wealth of domain-specific data (such as financial records, health records, genomic data, and supply chain information). This data can provide you with a unique and valuable perspective on your business and the broader industry that you work within. For generative AI usage, the domain-specific data you plan to use for semantic context must be encoded as a set of elements, each expressed internally as a “vector”. This contextually relevant data typically comes from your internal databases, data lakes, or unstructured data or document stores—the data stores that host your domain-specific data or knowledge. These data stores are generically called knowledge bases. [Retrieval Augmented Generation \(RAG\)](#) is the process for retrieving facts from these knowledge bases to ground [large language models \(LLMs\)](#) with up-to-date, accurate, and insightful data. As outlined in the following diagram, [AWS has added vector capabilities to AWS database and search services](#) so you can store vector datasets where your data is, simplify your application architecture, and use tried, tested, and familiar tools. A vector database, or vector data store, simply means a database with vector capabilities. Such a database can also provide additional enhancements to the ways you use your data with generative AI.



**Note**

This guide focuses on databases that are suitable for online transaction processing (OLTP) applications. If you need to store and analyze massive amounts of data quickly and efficiently (a requirement that is typically met by an OLAP application), AWS offers [Amazon Redshift](#). Amazon Redshift is a fully managed, cloud-based data warehousing service that is designed to handle large-scale analytics workloads.

There are two high-level categories of AWS OLTP databases—relational and non-relational.

- The AWS relational database family includes eight popular engines for Amazon Aurora and Amazon RDS. The Amazon Aurora engines include Amazon Aurora with PostgreSQL-Compatible Edition or Amazon Aurora MySQL-Compatible Edition. The Amazon RDS engines include PostgreSQL, MySQL, MariaDB, SQL Server, Oracle, and Db2.
- The non-relational database options are designed for specific data models. These include key-value, document, caching, in-memory, graph, time series, and wide-column data models.

We explore all of these in detail in the [Choose](#) section of this guide.

**Database migration**

Before deciding which database service you want to use, you should consider your business objective, database selection, and how you're going to migrate your existing databases.

The best database migration strategy helps you to take full advantage of the AWS Cloud. This might involve migrating your applications to use purpose-built cloud databases. You might just want the benefit of using a fully managed version of your existing database, such as RDS for PostgreSQL or RDS for MySQL.

Alternatively, you might want to migrate from your commercially licensed databases, such as Oracle or SQL Server, to Amazon Aurora. Consider modernizing your applications and choosing the databases that best suit your applications' workflow requirements. [Amazon Aurora DSQL](#), now in preview, is designed to be the fastest serverless distributed SQL database for always available applications with virtually unlimited scale, highest availability, and zero infrastructure management.

If you choose to first transition your applications and then transform them, you might decide to re-platform. This process makes no changes to the application that you use, but lets you take

advantage of a fully managed service in the cloud. When your databases are fully in the AWS Cloud, you can start working to modernize your application. This strategy can help you exit your current on-premises environment quickly, and then focus on modernization.

You can use [AWS Database Migration Service](#) to move data to Amazon Aurora. For resources to help with your migration strategy, see the [Explore](#) section.

## Consider

You're considering hosting a database on AWS. This might be to support a greenfield/pilot project as a first step in your cloud migration journey, or you might want to migrate an existing workload with as little disruption as possible. Or perhaps you might want to port your workload to managed AWS services, or even refactor it to be fully cloud focused.

Of course, the first major consideration when choosing your database is your business objective. What is the strategic direction that is driving your organization to change? Consider whether you want to rehost an existing workload, or refactor to a new platform so that you don't have to commit to commercial licenses.

Whatever your goal is, considering the right criteria can make your database decision easier. Here's a summary of the key criteria to consider.

### Migration strategy

You can choose a rehosting strategy to deploy to the cloud faster, with fewer data migration problems. Install your database engine software on Amazon Elastic Compute Cloud (Amazon EC2), migrate your data, and manage your database in a similar way to how you manage on premises. While rehosting is a fast path to the cloud, you're still left with the operational tasks such as upgrades, patches, backups, capacity planning and management, maintaining performance, and availability targets.

Alternatively, you can choose a re-platform strategy where you migrate your on-premises relational database to a fully managed Amazon RDS instance.

You might consider that this an opportunity to refactor your workload to be cloud focused. For example, you could use Amazon Aurora or purpose-built NoSQL databases such as Amazon DynamoDB, Amazon Neptune, or Amazon DocumentDB (with MongoDB compatibility).

Finally, AWS offers serverless databases, which can scale to an application's demands with a pay-for-use pricing model and built-in high availability. With serverless databases, you can



increase agility and optimize costs. In addition to removing the need to provision, patch, or manage servers, many AWS serverless databases provide maintenance options that reduce downtime.

AWS serverless offerings include Amazon Aurora Serverless, Amazon DynamoDB, Amazon ElastiCache, Amazon Keyspaces (for Apache Cassandra), Amazon Timestream for LiveAnalytics, and Amazon Neptune Serverless.

## Characteristics of your data

The core of any database choice includes the characteristics of the data that you need to store, retrieve, analyze, and work with. This includes:

- Your data model. For example, is it relational, structured, semi-structured, time series, vector, or using a highly connected dataset?
- Data access. How do you need to access your data?
- The extent to which you need real-time data.
- Whether there is a particular data record size you have in mind.

## Operational considerations

Your primary operational considerations are where your data is going to be located and how it will be managed. The two key choices that you need to make are:

- **Will your database be self-hosted or fully managed?:** The core question here is where is your team going to provide the most value to the business? If your database is self-hosted, you'll be responsible for the day-to-day maintenance, monitoring, and patching of the database.

Choosing a fully managed AWS database simplifies your work by removing undifferentiated database management tasks. This option allows your team to focus on delivering value by improving schema design, query construction, and query optimization. Your team can also develop applications that align with your business objectives.

- **Do you need a serverless or provisioned database?:** To start with, review these links to [Amazon DynamoDB](#), [Amazon Keyspaces \(for Apache Cassandra\)](#), [Amazon Timestream for LiveAnalytics](#), [Amazon ElastiCache](#), [Amazon Neptune](#), and [Amazon Aurora](#); documentation on how to think about provisioned throughput capacity and scaling. Additionally, this guidance for [Amazon Aurora Serverless v2](#) explains why it is suitable for highly variable workloads

(meaning, for example, that your database usage might be heavy for a short period of time, followed by long periods of light activity or no activity at all).

## Resiliency, performance, and security

It's important to ensure that your choice of database provides the resiliency, performance, and levels of security that you need.

- **Resiliency** - Database resiliency is key for any business. To achieve resiliency, you have to pay attention to a number of key factors, including capabilities for backup and restore, replication, failover, and point-in-time recovery (PITR).
- **Performance** - Consider whether your database will need to support a high concurrency of transactions (10,000 or more), and whether it needs to be deployed in multiple geographic regions. If your workload requires extremely high read performance with a response time measured in microseconds (rather than single-digit milliseconds), you might want to consider using in- memory caching solutions such as [Amazon ElastiCache](#) alongside your database, or a fully durable, persistent in-memory database such as [Amazon MemoryDB](#).
- **Security** - Security is a [shared responsibility](#) between AWS and you. The AWS shared responsibility model describes this as security of the cloud that AWS manages, and security in the cloud that the customer manages. Specific security considerations include data protection at all levels of your data, authentication, compliance, data security, storage of sensitive data, and support for auditing requirements.

## Vector database and vector search considerations

When choosing an AWS service with vector database or vector search capabilities, start by thinking about how familiar your team is with the service you are exploring. When developer teams are already familiar with a particular database engine, using the same database engine for vector search helps them make better use of existing knowledge and develop faster. Instead of learning a new skillset, developers can use their current skills, tools, frameworks, and processes to include a new feature of an existing database engine. Here's how that may apply to your situation:

- Your team of database engineers may already manage a set of 100 relational databases hosted on [Amazon Aurora PostgreSQL](#). If they want to support a new database with vector search requirement for their applications, they should first start with evaluating the pgvector extension on their existing Amazon Aurora PostgreSQL databases. Meanwhile, if your team

prefers using the community versions of PostgreSQL, [Amazon RDS for PostgreSQL](#) also supports the [pgvector](#) extension.

- Similarly, if your team is working with graph data, consider using [Amazon Neptune Analytics](#), which seamlessly integrates with your existing AWS infrastructure and provides useful graph querying and visualization features. It is ideal for [GraphRAG](#) use cases, or in analyzing large amounts of graph data to get insights and find trends.
- For teams that work with popular open source data stores Valkey and Redis OSS that need a highly scalable, in-memory database for real-time applications, consider using [Amazon MemoryDB](#). It provides a familiar interface, allowing the team to use their existing Valkey and Redis OSS knowledge and client libraries while benefiting from the fully managed, durable, and scalable capabilities of Amazon MemoryDB. Vector search for Amazon MemoryDB extends the functionality of Amazon MemoryDB. It can be used in conjunction with existing Amazon MemoryDB functionality. Applications that do not use vector search are unaffected by its presence. Vector search is available in all Regions that Amazon MemoryDB is available. Vector search for Amazon MemoryDB is ideal for use cases where peak performance and scale are the most important selection criteria. You can use your existing Amazon MemoryDB data, or a Valkey or Redis OSS API, to build machine learning and generative AI use cases. This includes retrieval-augmented generation, anomaly detection, document retrieval, and real-time recommendations.
- If your current tech stack lacks vector search support, you can take advantage of serverless offerings to help fill the gap in your vector search needs. For example, [OpenSearch Serverless](#) lets you [quickly create](#) an experience on the [Amazon Bedrock](#) console without having to create or manage a cluster. If your data is stored in [Amazon DynamoDB](#), OpenSearch Serverless can be an excellent choice for vector search using zero-ETL integration.

Additional criteria to consider include ease of implementation, scalability, and performance. They are discussed in-depth in this blog: [Key considerations when choosing a database for your generative AI applications](#).

## Choose

Now that you know the criteria for evaluating your database options, you're ready to choose which AWS database services might be a good fit for your organization.

This table highlights the type of data that each database is optimized to handle. Use it to help determine the database that is the best fit for your use case.

<b>Data model</b>	<b>When would you use it?</b>	<b>What is it optimized for?</b>	<b>Related database engines or services</b>
<b>Relational</b>	Use when you're migrating or modernizing an on-premises relational workload, or if your workload has less predictable query patterns.	Optimized for structured data that is stored in tables, rows, and columns. Relational databases support complex queries through joins.	<a href="#">Amazon Aurora</a> <a href="#">Amazon RDS</a>
<b>Key-value</b>	Use for workloads such as session stores or shopping carts. Key-value databases can scale to large amounts of data and extremely high throughput of requests, while servicing millions of simultaneous users through distributed processing and storage.	Optimized to provide a serverless, NoSQL, fully managed database with single-digit millisecond performance at any scale.	<a href="#">Amazon DynamoDB</a>
<b>In-memory</b>	Use Amazon ElastiCache when you need a caching layer to improve read performance. Use Amazon MemoryDB when you need full data persistence, but still need sub-millisecond read latencies.	Optimized to support microsecond reads and sub-millisecond writes. MemoryDB supports microsecond reads and single-digit millisecond writes. ElastiCache is an ephemeral cache, while MemoryDB	<a href="#">Amazon ElastiCache</a> <a href="#">Amazon MemoryDB</a>

Data model	When would you use it?	What is it optimized for?	Related database engines or services
<b>Document</b>	Use when you want to store JSON-like documents with rich querying abilities across the fields of the documents.	is an in-memory database.  Optimized for storing semi-structured data as documents with multilayered attributes.	<a href="#">Amazon DocumentDB (with MongoDB compatibility)</a>
<b>Wide-column</b>	Use when you need to migrate your on-premises Cassandra workloads, or when you need to process data at high speeds for applications that require single-digit millisecond latency.	Optimized for workloads that require heavy reads/writes and high throughput, coupled with low latency and linear scalability.	<a href="#">Amazon Keyspaces (for Apache Cassandra)</a>
<b>Graph</b>	Use when you have to model complex networks of objects, such as social networks, fraud detection, and recommendation engine use cases.	Optimized for traversing and evaluating large numbers of relationships, and identifying patterns with minimal latency.	<a href="#">Amazon Neptune</a>

Data model	When would you use it?	What is it optimized for?	Related database engines or services
Time series	Use when you have a large amount of time series data, potentially from a number of sources, such as Internet of Things (IoT) data, application metrics, and asset tracking.	Optimized for storing and querying data that is associated with timestamps and trend lines.	<a href="#">Amazon Timestream</a>

## Use

This section helps you learn more about the database service or services that you've chosen, and how to get started with them.

The database you've chosen might not satisfy all of your requirements perfectly, so it's important to consider your needs and workload requirements carefully.

Prioritize based on the considerations covered in this guide, your own specific “must have” requirements, and the requirements for which you have some flexibility. This will help you make effective trade-offs and lead to the best possible outcome for your needs.

Also consider that, usually, you can cover your application requirements with a mix of best-fit databases. By building a solution with multiple database types, you can use the strengths that each type provides.

For example, in an ecommerce use case, you might use Amazon DocumentDB (for product catalogs and user profiles) for the flexibility that is provided by semi-structured data—but then combine it with the low, predictable latency provided by DynamoDB (for when your users are browsing your product catalog). You might also add Aurora into the mix for inventory and order processing, where a relational data model and transaction support are needed.

To help you learn more about each of the available AWS database services, we have provided a pathway to explore how each of the services work. The following section provides links to in-depth documentation, hands-on tutorials, and resources to help you get started.

## Amazon Aurora



### Getting started with Amazon Aurora

This guide includes tutorials and covers more advanced Aurora concepts and procedures, such as the different kinds of endpoints and how to scale Aurora clusters up and down.

[Explore the guide](#)



### Create a highly available database

Configure an Amazon Aurora cluster to create a highly available database. This database consists of compute nodes that are replicated across multiple Availability Zones to provide increased read scalability and failover protection.

[Get started with the tutorial](#)



### Use Amazon Aurora global databases

Get started using Aurora global databases . This guide outlines the supported engines and AWS Region availability for Aurora global databases with Aurora MySQL and Aurora PostgreSQL.

[Explore the guide](#)

## Amazon RDS



### Getting started with Amazon RDS



Create and connect to a DB instance using Amazon RDS. You learn to create a DB instance that uses DB2, MariaDB, MySQL, Microsoft SQL Server, Oracle, or PostgreSQL.

[Explore the guide](#)

### Create and connect to a PostgreSQL database

Create an environment to run your PostgreSQL database (we call this environment a DB instance), connect to the database, and delete the DB instance.

[Get started with the tutorial](#)



### Create a web server and an Amazon RDS DB instance

Learn how to install an Apache web server with PHP and create a MySQL database. The web server runs on an Amazon EC2 instance using Amazon Linux. The MySQL database is a MySQL DB instance.

[Get started using the tutorial](#)

## Amazon DocumentDB



### Getting started with Amazon DocumentDB

We help you get started using Amazon DocumentDB in just seven steps. This guide uses AWS Cloud9 to connect and query your cluster using the MongoDB shell directly from the AWS Management Console.



### Setting up a document database with Amazon DocumentDB

This tutorial helps you to get started connecting to your Amazon DocumentDB cluster from your AWS Cloud9 environment



[Explore the guide](#)

with a MongoDB shell, and then run a few queries.

[Get started with the tutorial](#)

### Best practices for working with Amazon DocumentDB

Learn best practices for working with Amazon DocumentDB, along with the basic operational guidelines when working with it.

[Explore the guide](#)

### Migrate from MongoDB to Amazon DocumentDB

Learn how to migrate an existing self-managed MongoDB database to a fully managed database on Amazon DocumentDB.

[Get started with the tutorial](#)

### Assessing MongoDB compatibility

Use the Amazon DocumentDB compatibility tool to help you assess the compatibility of a MongoDB application by using the application's source code or MongoDB server profile logs.

[Use the tool](#)

## Amazon DynamoDB



### What is Amazon DynamoDB?



This guide explains how you can use this fully managed NoSQL database service to offload the administrative burdens of operating and scaling a distributed database (including hardware provisioning, setup and configuration, replication, software patching, and cluster scaling).

[Explore the guide](#)



### Create and query a NoSQL table with Amazon DynamoDB

Use these hands-on tutorials to get started with Amazon DynamoDB and the AWS SDKs. You can run the code examples on either the downloadable version of DynamoDB or the DynamoDB web service.

[Get started with the tutorials](#)

### Getting started with DynamoDB and the AWS SDKs

We help you get started with Amazon DynamoDB and the AWS SDKs. This guide includes hands-on tutorials that show you how to run code examples in DynamoDB.

[Explore the guide](#)



### Create an Amazon DynamoDB table

We show you how to create a DynamoDB table and use the table to store and retrieve data. This tutorial uses an online bookstore application as a guiding example.

[Get started with the tutorial](#)

## Amazon ElastiCache



### Documentation for Amazon ElastiCache

Explore the full set of Amazon ElastiCache documentation, including user guides as well as specific AWS CLI and API references.

[Explore the guide](#)



### Getting started with Amazon ElastiCache

Learn how to create, grant access to, connect to, and delete a Valkey (cluster mode disabled) cluster using the Amazon ElastiCache console.

[Explore the guide](#)

## Amazon MemoryDB



### Getting started with Amazon MemoryDB

We guide you through the steps to create, grant access to, connect to, and delete a MemoryDB cluster using the MemoryDB console.

[Use the guide](#)

### Get started with Amazon MemoryDB for Valkey

Get an overview of MemoryDB for Valkey, its benefits, and how you can upgrade your MemoryDB for Redis OSS database to MemoryDB for Valkey database.

[Read the blog](#)

### Integrating Amazon MemoryDB with Java-based AWS Lambda

We discuss some of the common use cases for the data store, Amazon MemoryDB, which is built to provide durability and faster reads and writes.

[Read the blog](#)

## Amazon Keyspaces



## Getting started with Amazon Keyspaces (for Apache Cassandra)

This guide is for those who are new to Apache Cassandra and Amazon Keyspaces (for Apache Cassandra). It walks you through installing all the programs and drivers that you need to successfully use Amazon Keyspaces.

[Explore the guide](#)

## Beginner course on using Amazon Keyspaces (for Apache Cassandra)

Learn the benefits, typical use cases, and technical concepts of Amazon Keyspaces. You can try the service through the sample code provided or the interactive tool in the AWS Management Console.

[Take the course \(requires sign-in\)](#)

## Amazon Neptune



### Getting started with Amazon Neptune

We help you get started using Amazon Neptune, a fully managed graph database service. This guide shows you how to create a Neptune database.

[Explore the guide](#)



### Using knowledge graphs to build GraphRAG applications with Amazon Bedrock and Amazon Neptune

Build GraphRAG applications using Amazon Bedrock and Amazon Neptune with [LlamaIndex](#) framework.

[Read the blog](#)



### Build a recommendation engine with Amazon Neptune

We show you how to build a friend recommendation engine for a multiplayer game application using Amazon Neptune.



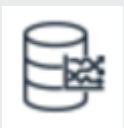
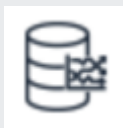


### Build a fraud detection service using Amazon Neptune

We walk you through the steps to create a Neptune database, design your data model, and use the database in your application.

<a href="#">Explore the guide</a>	<a href="#">Explore the guide</a>
-----------------------------------	-----------------------------------

Amazon Timestream

 <p><b>Getting started with Amazon Timestream</b></p> <p>We help you get started with Amazon Timestream. This guide provides instructions for setting up a fully functional sample application.</p> <p><a href="#">Explore the guide</a></p>	 <p><b>Best practices with Amazon Timestream</b></p> <p>We explore best practices, including the practices that relate to data modeling, security, configuration, data ingestion, queries, client applications, and supported integrations.</p> <p><a href="#">Explore the guide</a></p>
 <p><b>Accessing Amazon Timestream using AWS SDKs</b></p> <p>Learn how to access Amazon Timestream using the AWS SDKs in the language of your choice: Java, Go, Python, Node.js, or .NET.</p> <p><a href="#">Explore the guide</a></p>	 <p><b>Understanding time-series data and why it matters</b></p> <p>Explore the nature of time-series data, its presence across different types of industries and various use cases it enables.</p> <p><a href="#">Read the blog</a></p>

Explore

<b>Role</b>  <a href="#">Developers</a>	<b>Migration strategy</b>  <a href="#">Getting started with AWS Database Migration Service</a>
-----------------------------------------------	------------------------------------------------------------------------------------------------------

[Solution architects](#)[Professional development](#)[Startups](#)[Decision makers](#)[Using the AWS Schema Conversion Tool](#)[Selecting the right database and database migration plan for your workloads](#)**Architecture diagrams**

Explore reference architecture diagrams to help you develop, scale, and test your databases on AWS.

[Explore architecture diagrams](#)**Whitepapers**

Explore whitepapers to help you get started, learn best practices, and migrate your databases.

[Explore whitepapers](#)**AWS solutions**

Explore vetted solutions and architectural guidance for common use cases for databases.

[Explore solutions](#)

## Document history

The following table describes the important changes to this decision guide. For notifications about updates to this guide, you can subscribe to an RSS feed.

Change	Description	Date
<a href="#">Guide updated</a>	Updated to include vector database and vector search options, and provided additional links to new service-specific documentation.	December 22, 2024
<a href="#">Guide updated</a>	Updated Amazon RDS description to include Db2 support, and provided additional links to service-specific documentation.	May 13, 2024
<a href="#">Initial publication</a>	Guide first published.	September 11, 2023