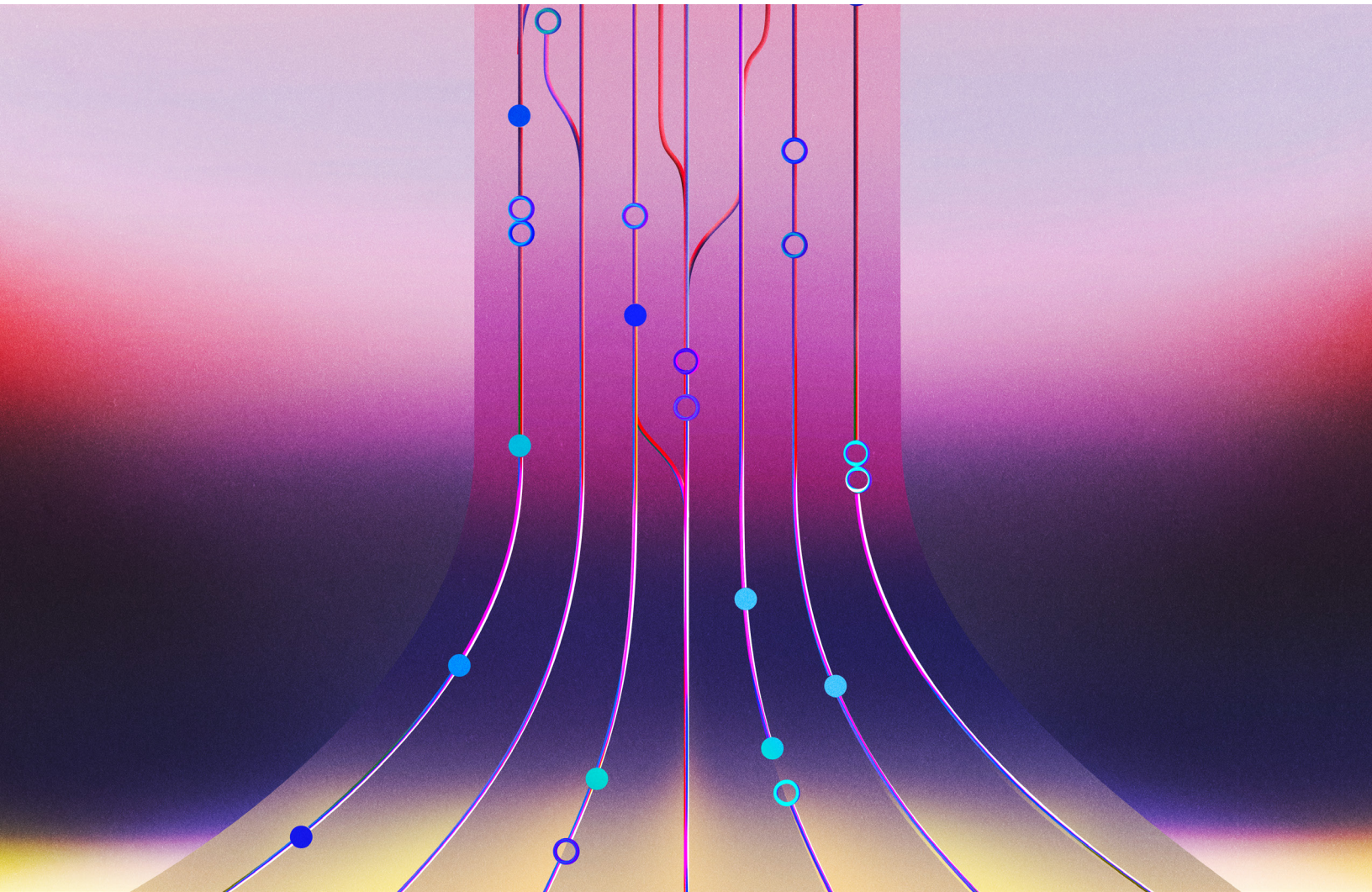
aws

Build data-driven applications using AWS databases



Contents

Contents

Key characteristics of data-driven applications

As organizations look to increase the pace of innovation and build new customer experiences, modernizing how you build and operate data-driven applications is key. From applications that are integral to the daily lives of customers to enterprise software that controls back office operations, applications are the driving force behind every successful organization.

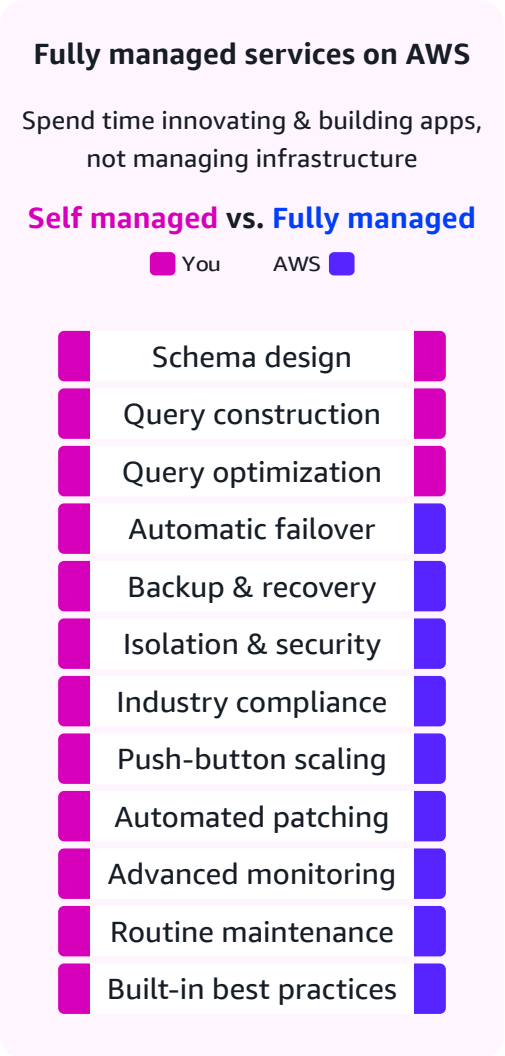
Data-driven applications require technologies that enable organizations to innovate faster and improve performance, security, and reliability while lowering their total cost of ownership. These technologies include:

- **Cloud infrastructure:** Offload undifferentiated management tasks, access continuous innovation, scale efficiently, and reduce costs with pay-as-you-go pricing to enable faster time to value and unlock competitive advantages
- **Security, identity, and compliance:** Build with the highest standards using encryption, automated threat detection, and real-time compliance monitoring for safe, authorized access, and secure infrastructure management
- **Low-code and no-code technologies:** Use drag-and-drop interfaces, pre-built components, and tools to empower teams to compose, assemble, and integrate applications without deep coding expertise
- **Microservices-based architecture:** Enable faster development with loosely coupled services to support incremental releases, team autonomy, deeper functionality, and faster time to market
- **DevOps automation:** Accelerate software delivery with practices like continuous integration and deployment to reduce production deployment times from months to hours while providing high-quality software
- **Multi-Region operations with local performance:** Scale automatically to millions of users and petabytes of data to provide near real-time response, dynamic scaling, and automated failure recovery for always-available applications
- **Embedded machine learning and artificial intelligence (AI):** Deliver insights through features like personalized recommendations and enhanced security to improve customer engagement without requiring machine learning expertise

Applications are the largest producers and consumers of data, and databases power applications. As such, we will discuss how AWS Databases deliver on the aforementioned technologies as a foundational component in the data-driven application technology stack. Selecting the right database(s) for your data infrastructure is critical as the architecture decisions made today are an investment for the next decade. The right database choices can create new and engaging customer experiences, process transactions faster, and spur innovation. This paper explains how AWS databases provide a high-performance, secure, and reliable foundation to power generative AI solutions and data-driven applications at any scale.

01 Making best-fit database choices

Making best-fit database choices



With fully managed databases, your operational burden is significantly reduced. Daily database management tasks can take up an inordinate amount of time spent on undifferentiated heavy lifting. AWS managed databases handle all the fundamental database instance operations—like provisioning, high availability and durability, patching, upgrades, setup, configuration, automated backups, and failover. Also, AWS continuously monitors your clusters to keep your workloads up and running with self-healing storage and automated scaling, so that developers and database operators can focus on higher value tasks, like new features, schema design, query optimization, and access control.

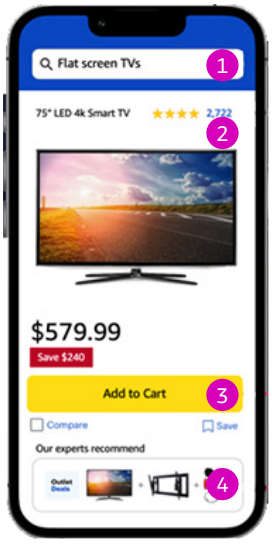
Advantages of a microservices architecture

Today's applications use a microservices-based architecture as it provides many advantages—such as increased agility, scalability, and flexibility—focused on faster innovation and reduced dependencies. Microservices teams work autonomously and are empowered to make their own technology choices, including the choice of database. For this approach to prevail, each team of microservices developers and DevOps professionals needs a rich set of database choices.

While relational databases are still essential—in fact, they are still growing—a relational-only approach no longer works for today's database workloads. With the rapid growth of data—in volume, velocity, variety, complexity, and interconnections—database requirements have changed. Many new applications that have social, mobile, Internet of Things (IoT), and global access requirements are unable to scale using a central relational database alone. This has led to the use of purpose-built databases designed to handle specific data models.

Many data-driven applications consist of a mix of workloads—each of which has its own unique database requirements that needs both relational and purpose-built databases. This is why easy access to a portfolio of purpose-built databases is now a critical success factor for data-driven applications.

Making best-fit database choices



1. **Search:** Index-optimized store
2. **Customer reviews:** Key-value database
3. **Shopping cart:** Relational database
4. **Recommendations:** Graph database

As an example, an e-commerce shopping application uses an index-optimized store like Amazon OpenSearch Service to help users quickly find relevant information. Then, it uses a key-value database like Amazon DynamoDB for showing customer feedback that uses a five-star rating system. The purchase button uses a relational database, such as Amazon Aurora, to ensure transactional integrity for both inventory and financial accounting. Then, it uses a graph database like Amazon Neptune to power personalization algorithms, such as recommendations on what additional items the end customer might want to buy based on their past purchases. Data-driven applications are built using an array of different technologies, all simultaneously within the same application, to provide the performance and scale that the end users are seeking.

AWS offers 15+ database engines, each built to uniquely address specific customer needs. In addition to relational databases, the AWS database portfolio includes a full range of purpose-built databases including key-value, document, graph, in-memory, search, wide-column, and time-series.

Broadest and deepest set of relational and purpose-built databases

Relational	Purpose-Built			
Amazon Aurora	**Key-Value** Amazon DynamoDB	**Caching** Amazon ElastiCache	**Document** Amazon DocumentDB	**Graph** Amazon Neptune
Amazon RDS	**Memory** Amazon MemoryDB	**Wide-Column** Amazon Keyspaces	**Time-Series** Amazon Timestream	

Making best-fit database choices

Automated capacity changes with serverless databases

Many data-driven applications have workloads with variable demands, which makes changing database capacity a frequent occurrence. Operating at the highest efficiency levels becomes a major challenge when capacity requirements can vary based on time of day, time of year, promotional events, and other factors, resulting in a complex multi-dimensional decision matrix of when to scale each capacity dimension.

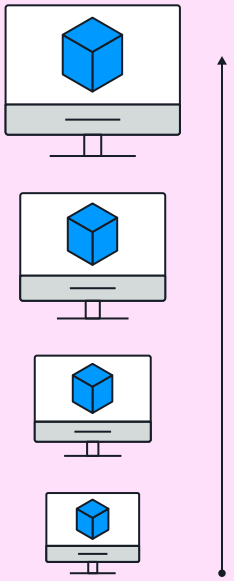
If the capacity is set too low, then databases can't keep up and end users feel application performance issues. If the capacity is set to meet peak demands, then the excess capacity is wasted during non-peak time intervals, incurring unnecessary expense. Manual scaling burdens IT operations teams with a series of detailed tasks to minimize system outages, taking up valuable time and distracting from other higher value activities such as schema design and application development. In most cases, manual scaling is not seamless and it involves some business disruption.

Over the years, our customers have adopted serverless architectures for a range of use cases, such as event processing, internet-scale applications, multi-tenant software-as-a-service (SaaS) applications, new applications with unknown capacity requirements, and more. With AWS serverless databases, your database automatically scales to match the workload demand 24/7.

With AWS Databases, serverless is implemented as automated vertical scaling or as an endpoint with zero infrastructure management. The available options for automated vertical scaling include Amazon Aurora, Amazon Neptune, and Amazon Timestream for LiveAnalytics. In contrast, single endpoint implementation for serverless provides automated vertical and horizontal scaling for both compute and storage, and is available for Amazon Aurora DSQL, Amazon DynamoDB, Amazon ElastiCache, and Amazon Keyspaces. Regardless of implementation, billing is based on usage and results in up to 90 percent cost savings when compared to provisioning for peak capacity. Under the hood, AWS uses multiple techniques for automatically changing the capacity of serverless databases.

Making best-fit database choices

Vertical Scaling
Increase size of instance
(RAM, CPU, etc.)



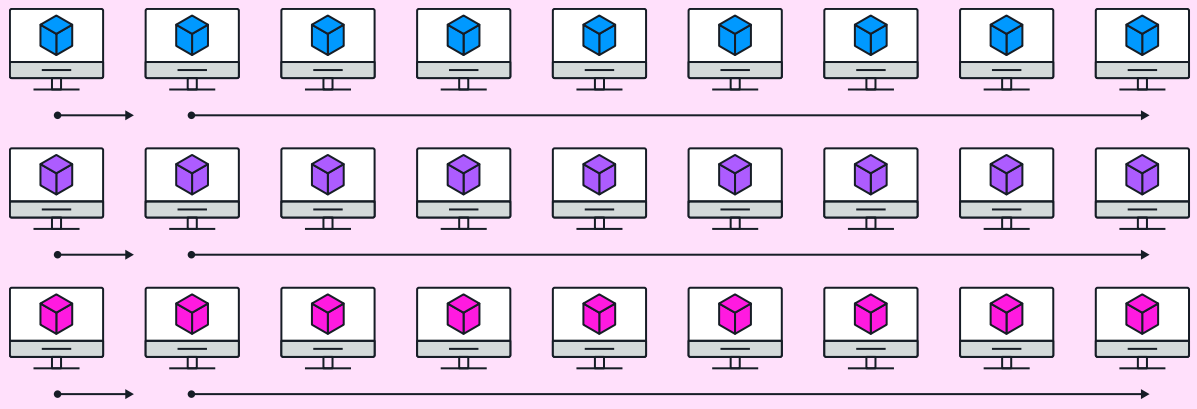
Vertical scaling

With vertical scaling, you get intelligent, automated capacity management. The virtual machine resources underpinning your database instance dynamically scale up or down. Memory, central processing unit (CPU), and network capacity grow when scaling up, and reduce when scaling down. This occurs in real-time while the database is operating, without disrupting the workload. These databases continuously monitor resource utilization, scaling seamlessly even with thousands of active connections, making it ideal for applications with variable workloads. This ensures optimal performance during peak times and cost-efficiency during idle periods.

Horizontal scaling

Horizontal scaling involves the distribution of the database workload across multiple infrastructure resources to improve performance and capacity. Common techniques include data partitioning, adding nodes to increase cluster capacity, and sharding data across nodes, each of which is discussed below.

Horizontal Scaling
Add more instances



Case study: BMW

BMW adopted Amazon Aurora Serverless v2 to handle unpredictable demand and reduce database costs. Previously, workloads were run at maximum capacity to manage spikes, resulting in resource waste. With Aurora Serverless v2, BMW now scales automatically to meet demand, minimizing costs and operational effort.

"We noticed a disproportionate increase in demand for scalability," says Marc Fiedler, product owner for BMW Messages. "By taking a serverless approach on AWS, we've achieved flexible scaling and cost reductions."

Migrating to Aurora PostgreSQL-Compatible Edition allowed BMW to reassign 12 operations team members to focus on product innovation. Automated scaling and zero-downtime patching have eliminated disruptions and downtime, ensuring seamless customer interactions.

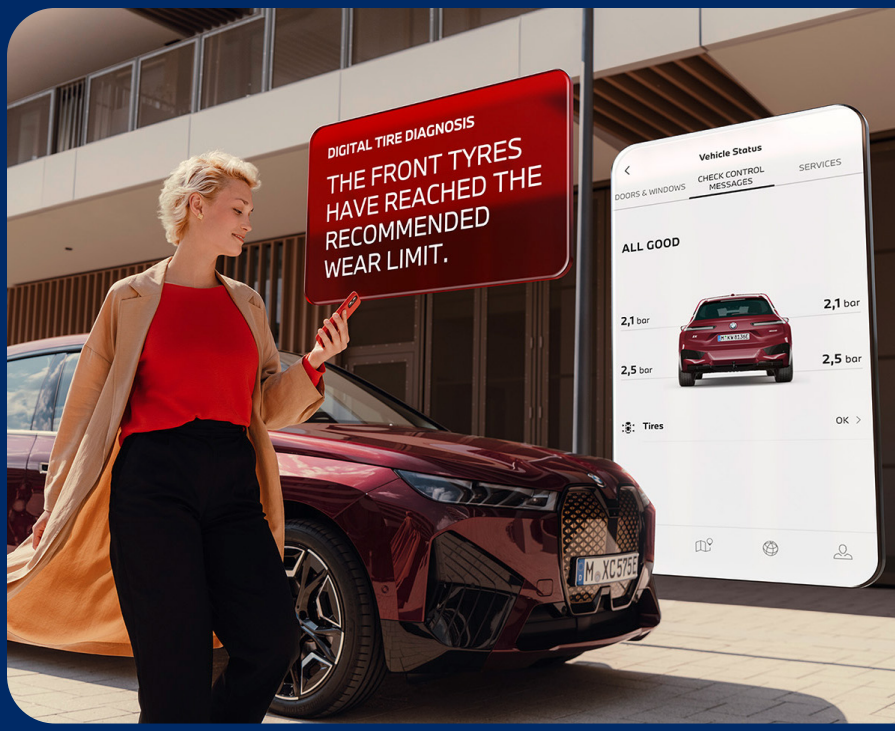
BMW also unified its global architecture of 1,300 microservices, which process over 12 billion daily requests, by standardizing on Aurora Serverless v2. In just six months, BMW rearchitected systems across its hubs to enhance scalability and connectivity.

Read the full story →

"We've moved away from physical server limitations and now scale worry-free in the cloud with Aurora Serverless.

With AWS, we're building a serverless future to optimize costs and reliability."

Marc Fiedler
Product Owner for BMW Messages,
BMW Group



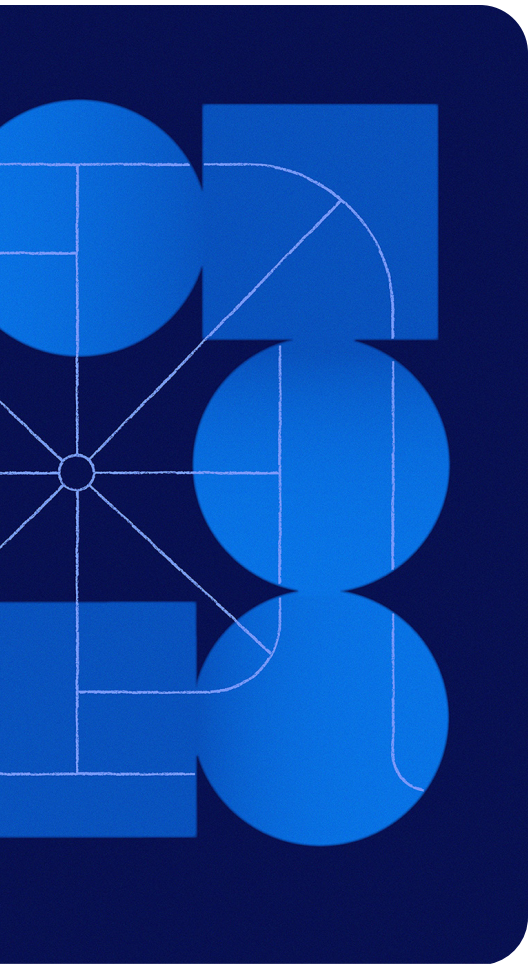
Making best-fit database choices

Data partitioning

Data partitioning breaks down database tables into subsets (partitions) of data based on values in one or more table columns, and can even allocate each subset to a different storage device on the same machine. This reduces contention for data on a single storage device and for other reasons such as portability, reducing size of index strictures, and more. Partition management occurs automatically in the background and is transparent to your applications. Unlike sharding (see discussion below), partitioning does not require partitions to reside on different database nodes/instances.

For example, Amazon Aurora DSQL offers built-in support for various approaches to partitioning. With range partitioning, a table is partitioned into “ranges” defined by a key column or set of columns. With list partitioning, the table is partitioned by explicitly listing which key value(s) appear in each partition. Hash partitioning applies a hash function to a key. The result of the hash function determines which partition the data will be stored in.

Making best-fit database choices



Adding nodes to a cluster

Database capacity and performance can be increased by scaling out a database cluster with additional nodes. There are multiple techniques for distributing data and workload across the nodes. You can segment the data across the nodes—a practice known as sharding, covered in the next section. Alternatively, you can replicate the entire data set across multiple nodes.

Only AWS instance-based databases support horizontal scaling by creating read replicas for accelerating read performance. AWS Databases like Aurora DSQL, DynamoDB, ElastiCache, and Keyspaces are fully distributed systems, scaling both compute and storage horizontally based on demand.

DynamoDB maintains multiple copies of your data in a single Region, and applications read and write to a Regional endpoint. With DynamoDB global tables, data is further replicated to other Regions, and applications in multiple Regions can read and write to their respective Regional endpoints. DynamoDB can be configured for either eventual or strong consistency.

Aurora DSQL is a fully serverless distributed SQL database with automated scaling whether your application is running in a single or multiple Regions. Aurora DSQL is ideal for new applications, offering a serverless endpoint that eliminates the need for infrastructure management. Under the hood, Aurora DSQL maintains multiple copies of your data set within a Region and across multiple Regions depending on the cluster deployment (see Meeting users where they are with global deployments).

Case study: DraftKings

DraftKings offers online sports betting services, daily fantasy sports contests, and iGaming solutions. During events like the Super Bowl, DraftKings experiences massive traffic spikes, as users update bets and check balances simultaneously. To handle these surges with low latency and high reliability, DraftKings chose Amazon Aurora, a high-performance, scalable database service with MySQL compatibility. DraftKings uses Aurora MySQL-Compatible Edition to power its financial ledger, which tracks balances and processes transactions for over 3.1 million monthly users. Aurora's ability to provision read replicas rapidly and its 18x input/output performance improvement compared to traditional databases allowed DraftKings to scale efficiently during high-demand periods. For example, during the 2024 Super Bowl, Aurora maintained throughput and latency metrics, handling traffic peaks 50 percent higher than the season opener. Aurora read replicas distributed read traffic, ensuring fast, seamless user interactions. Key features like database cloning and I/O optimization supported rapid testing, efficient reads/writes, and high availability.

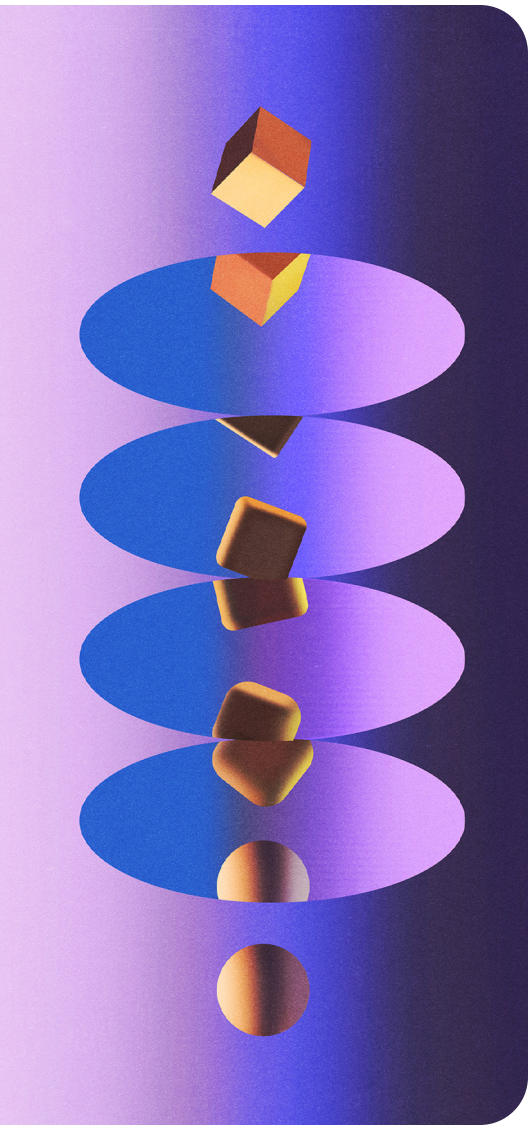
AWS also offers programs and services, ranging from AWS Professional Services that taps into the deep expertise of tenured professionals for migration assistance to Database Migration Accelerator (DMA), where for a fixed fee, a team of AWS professionals handles the conversion of both the database and application for you. Database Freedom provides expert advice and migration assistance to qualified customers. Additionally, AWS DMS Partners have knowledge, expertise, and experience with migrations.

Read the full story →

Today, Aurora enables DraftKings to process 1 million operations per minute with read latency under 1 millisecond and write latency averaging 6 milliseconds.

As DraftKings expands to new markets, it continues to rely on Aurora's performance and scalability to deliver an exceptional user experience during peak events.

Making best-fit database choices



Adding nodes to a cluster with sharding

Sharding takes the concept of partitioning, discussed earlier, and adds an additional step by mapping each partition to a separate node in a cluster of compute instances. Performance is optimized by hosting each partition in a physically separated database node in a cluster. The capacity of each instance is dedicated to processing the subset of data hosted by the node. All database shards usually have the same type of hardware, database engine, and data structure to generate a similar level of performance. However, they have no knowledge of each other, similar to a shared-nothing architecture.

It can be difficult to break up a relational data model into slices of data to be spread across multiple virtual machines. In traditional implementations, the user would implement the data mapping and routing logic themselves, which can be complex to operate and error-prone. With the large number of tables typical of relational databases, extreme care must be taken to ensure that each shard has the same key ranges that are needed for joining tables locally.

However, with Amazon Aurora PostgreSQL Limitless Database, you have fully managed sharding and can operate sharded data sets without the overhead of manual shard management and query routing. Aurora PostgreSQL Limitless Database offers automated horizontal scaling for millions of write transactions commits per second, and automatically rebalances the data in a cluster when shards are added or removed. It takes advantage of built-in data mapping and routing logic to send requests to the appropriate shards, removing the operational complexity with sharding while the application is running.

Since each shard handles an independent subset of data, the overall capacity of a cluster is a function of the number of shards. Adding capacity is simply a matter of adding shards. Sharding increases the overall capacity of a cluster into petabytes of data in a single database. It is effective in online transaction processing (OLTP) environments as long as the transactions are relatively small and can be bounded on data ranges that can be located on the same shard. Each shard only handles a subset of the write requests, increasing the overall write requests a cluster can handle.

Case study: Careem

Careem serves over 50 million customers across 10 countries. As its everything app services expanded to include ride-hailing, food delivery, and money transfers, Careem faced challenges with its monolithic database infrastructure, leading to high latency and frequent downtime. To address these issues, Careem adopted Amazon Web Services (AWS) to transition to a microservices architecture.

The company migrated its MySQL-based system to Amazon DynamoDB, a serverless NoSQL database offering single-digit millisecond performance, to handle driver location data. Careem also optimized costs, lowering expenses from thousands to $1,600 per month by adopting a schema-free design and improved table management.

Additionally, Careem uses Amazon Relational Database Service (RDS) for other workloads, enabling efficient scaling and high availability across its microservices. The transition improved app reliability and deployment flexibility, allowing Careem to expand its services while enhancing performance.

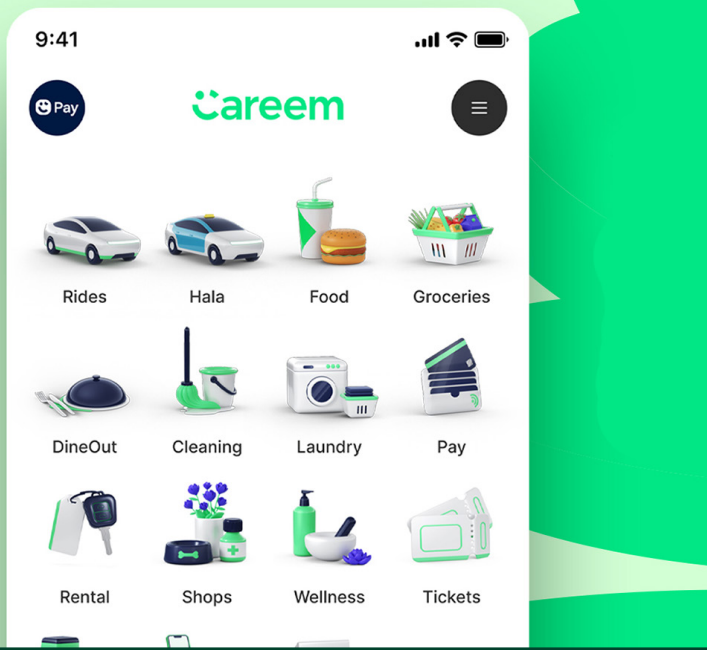
With support from AWS, Careem continues to innovate, with security tools like Amazon GuardDuty and complying with regional data regulations. “Using AWS enables us to have access to innovative technology solutions,” says Naorus Abdulghani, VP of Engineering at Careem.

Read the full story →

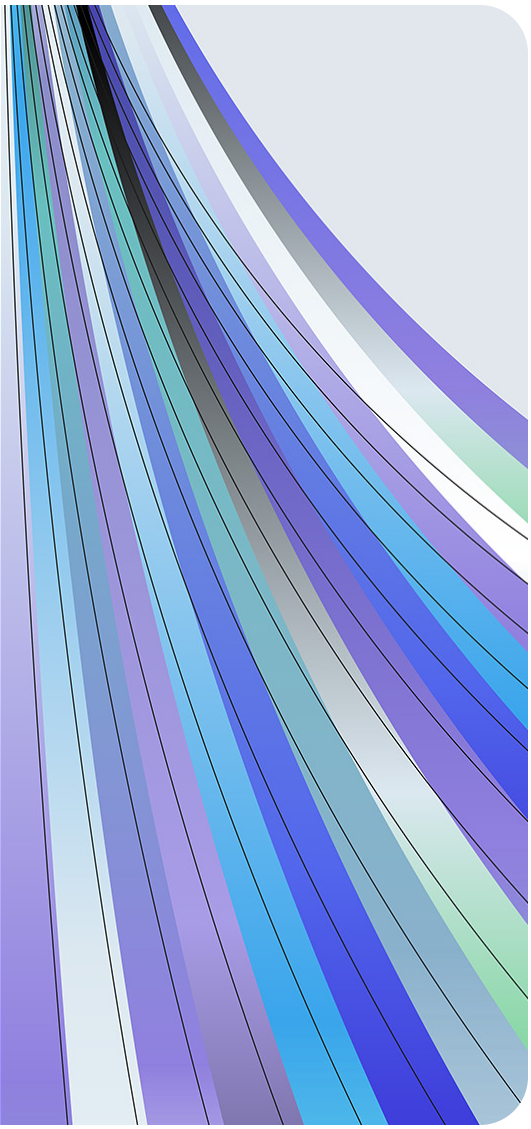
“We’re getting locations from millions of drivers every few seconds.

We needed a scalable solution for so much data, and Amazon DynamoDB was just about perfect.”

Khurram Naseem
Senior Director of Engineering, Careem
(a wholly owned subsidiary of Uber)



Making best-fit database choices



In-memory benefits of low-latency, high-throughput data access

In-memory caches and databases provide a good solution for use cases that require microsecond data access. Many applications use a cache to improve both the performance of the application and the database. Moreover, using a cache with a database is considered a best practice for microservices-based architecture to offset network latency. AWS offers two options for in-memory data access: Amazon ElastiCache (Valkey-, Memcached-, and Redis OSS-compatible) and Amazon MemoryDB (Valkey- and Redis OSS-compatible). Valkey is a drop-in replacement for Redis OSS (Redis removed BSD 3-Clause License in March 2024) stewarded by the Linux Foundation.

For read performance, both ElastiCache and MemoryDB provide microsecond latency and throughput improvements over disk storage. ElastiCache can process one million requests per second per node and 500 million requests per second for a cluster. The difference between ElastiCache and MemoryDB has to do with durability. ElastiCache is a cache, otherwise known as a non-durable or semi-durable data store. MemoryDB is a fully durable, in-memory database. Like other caches, ElastiCache does write to disk but the copy of the data on disk may lag. As such, it's susceptible to data loss in the event of a cache failure. The high availability feature in ElastiCache mitigates this exposure considerably. If the primary instance of ElastiCache fails, a replica node that becomes the new primary has most of the data from the failed primary. Only the data that was not yet replicated from the failed primary is lost.

If any amount of data loss is not acceptable, then MemoryDB is a better fit. MemoryDB is an in-memory database with in-memory performance. When data is written to MemoryDB, it is synchronously written to a durable Multi-AZ transaction log before MemoryDB acknowledges the write. These synchronous writes are slower than the write performance of ElastiCache, which writes in microseconds, whereas MemoryDB can take a few milliseconds.

02 Maintaining operational stability

Maintaining operational stability

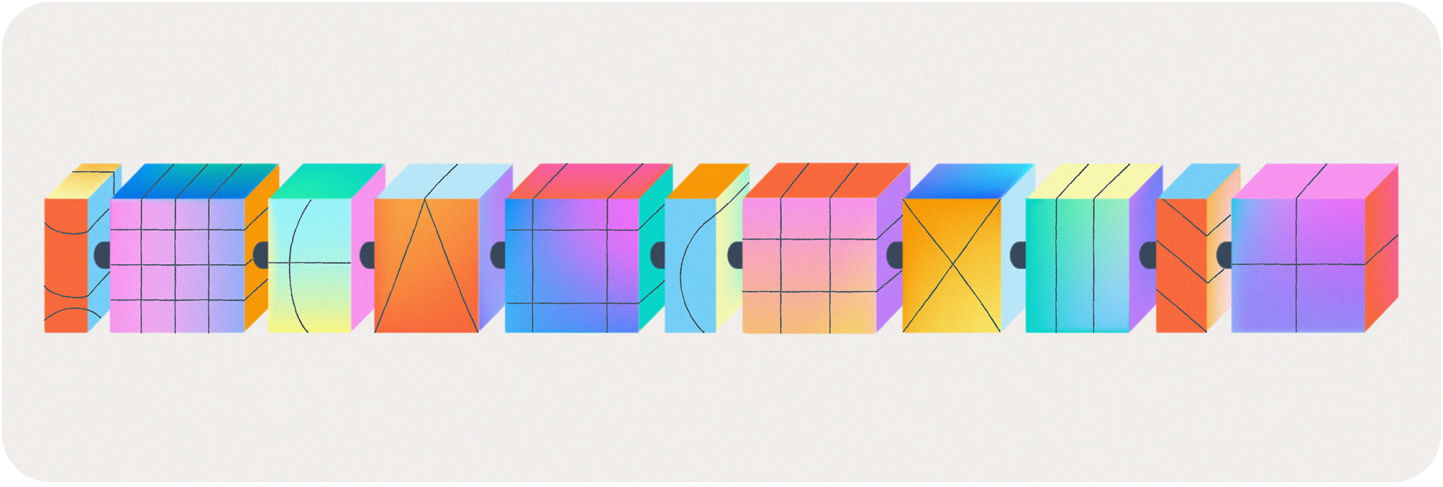
Security and compliance

AWS maintains the highest standards in security, governance, and compliance. Data in your databases can be encrypted at rest and in transit. You use AWS Key Management Service (AWS KMS) to create and control keys used to encrypt or digitally sign your data. AWS databases also support secure connections via Transport Layer Security (TLS) protocol.

AWS Identity Services help you securely manage identities, resources, and permissions at scale.

Amazon GuardDuty is an advanced threat detection service that uses machine learning to continuously monitor your AWS accounts and workloads for malicious activity. It provides detailed security findings, enabling enhanced visibility and effective remediation. With GuardDuty, you can quickly identify potential security misconfigurations, detect threats, and address unexpected behaviors, allowing you to respond swiftly to unauthorized or malicious activities in your environment.

AWS complies with all major regulations, including PCI DSS, HIPAA/HITECH, FedRAMP, GDPR, FIPS 140-2, and NIST 800-171.



Maintaining operational stability

High availability

Modern organizations regard high availability of their IT infrastructure as a critical requirement. Increasing levels of digital transformations create a mission-critical dependence on the availability of IT. AWS Databases are protected by fault isolation boundaries that limit the blast radius of a failure to a limited number of components. Database clusters are deployed across multiple Availability Zones (AZs). An AZ is a logical collection of data centers in a single Region designed to be fault tolerant and highly available. AZs are isolated from each other. Data redundancy is maintained by storing copies of your data in multiple AZs.

Implementations vary, but all AWS databases are designed for high availability and resilience. Amazon Aurora and Amazon DynamoDB serve as good examples. Amazon Aurora supports Multi-AZ DB cluster deployments with readers deployed in different AZs. Aurora makes your data durable across three AZs, but only charges for one copy. It automatically fails over to a reader in the event of a writer database instance failure, or an entire AZ failure. DynamoDB automatically partitions, stores, and synchronously replicates data across three AZs within a region in a multi-active configuration that does not require failover in the event of a writer or AZ failure. Each database in the AWS database portfolio provides similar mechanisms for high availability.

Aurora DSQL, DynamoDB, Keyspaces, and MemoryDB provide up to 99.999% availability. RDS, ElastiCache, DocumentDB, and Timestream offer up to 99.99% availability.

Maintaining operational stability

Blue/green deployments

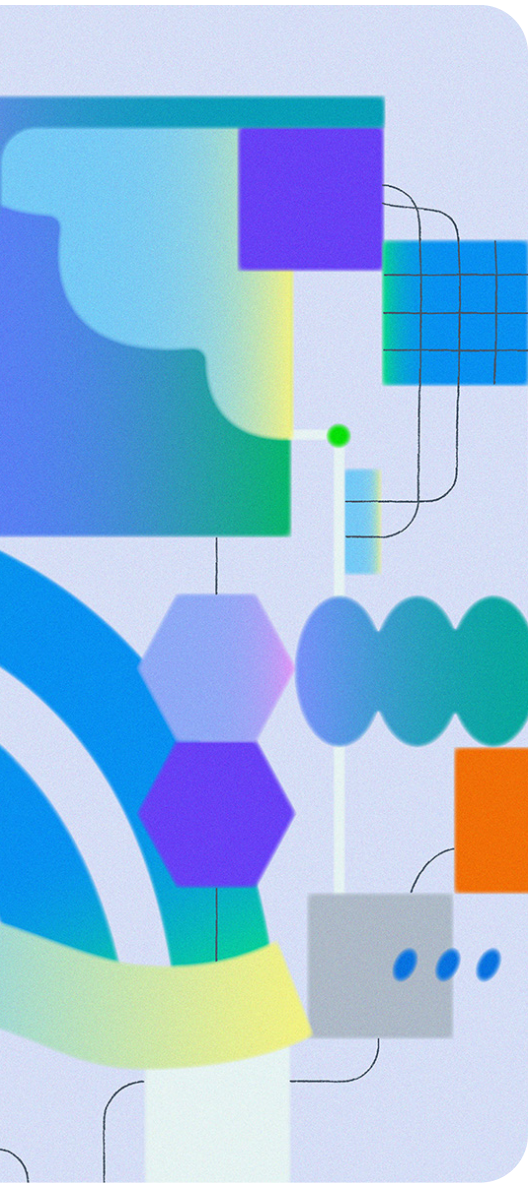
Some changes, like major version upgrades and schema changes, are under your control and subject to your timing requirements. These changes need to provide mechanisms for testing version upgrades and ensuring their quality. One such mechanism is blue/green deployments. This approach can be applied to databases as part of making changes to production. In the current atmosphere of 24/7 operations, downtime for major version upgrades, schema changes, or data loss due to failed attempts at updates is not acceptable.

Amazon RDS Blue/Green Deployments—available for both Amazon RDS and Amazon Aurora—provide a simpler, safer, faster, and secure way to make these changes. In this DevOps technique, the production environment is the blue environment and the staging environment is the green environment. Typically, organizations test new versions of software in a green environment under a production load, before putting it in production. But this requires advanced operational knowledge, careful planning, and time. With Amazon RDS Blue/Green Deployments, AWS provides a fully managed staging environment. When an upgrade is deemed to be ready, the switchover can occur in less than a minute with zero data loss.

In addition, Amazon RDS Multi-AZ deployments with two readable standbys now support minor version upgrades and system maintenance updates with typically less than one second of downtime when using Amazon RDS Proxy, which can also be used with Amazon Aurora. This capability allows you to take advantage of the most recent performance improvements, bug fixes, and any new security fixes or patches from the latest minor versions of PostgreSQL and MySQL with minimal interruption to your application.

03 Meeting users where they are with multi-Region deployments

Meeting users where they are with multi-Region deployments



Users expect local response time from their databases, rather than being subjected to network latency. Multi-Region data distribution can locate data near users, so that they experience local response time. It also provides an alternate source of data in the event application processing is interrupted in one Region.

There are three approaches to multi-Region data distribution, with each one presenting a different set of benefits: (1) maintaining local copies of data for faster read response time, (2) using active-active data replication for improving the response time of both reads and writes, and (3) using active-active data replication with strong consistency for always-available applications.

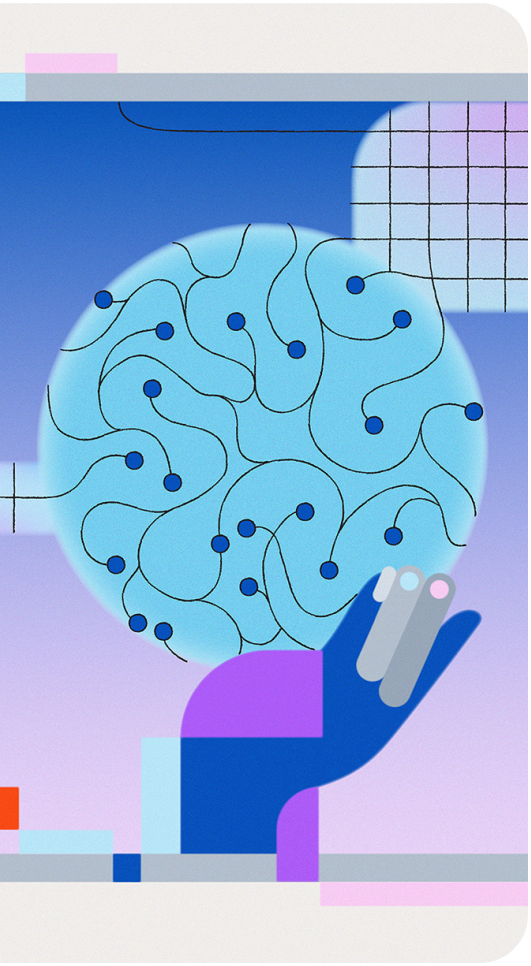
Multi-Region read replicas

Beyond Multi-AZ read replicas, the AWS operational databases that support multi-Region distributed read replicas are Amazon RDS, Amazon Aurora Global Database, Amazon DocumentDB, and Amazon Neptune. Multi-Region read replicas eliminate the need for a long network hop for retrieving data. Changes to data in the primary region are continuously and asynchronously replicated to secondary regions.

Active-active multi-Region databases with eventual consistency

Active-active means that you can read and write to any Regional endpoint at the same time. An active-active, multi-Region database with eventual consistency means each local copy of data is eventually consistent with the Region where the data was initially written. This technique is used to scale writes across Regions. Unlike sharding, this architecture maintains active-active data replication of the entire dataset, and replication usually occurs within one second.

Meeting users where they are with multi-Region deployments



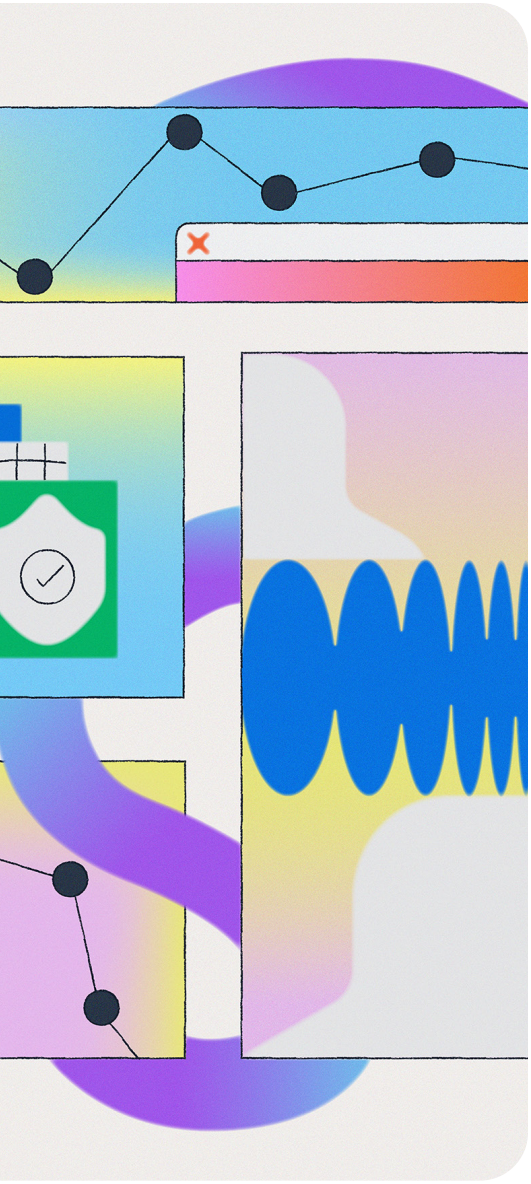
Amazon DynamoDB global tables is a multi-active, multi-Region database—every Region's replica is active, so you can write to it and read from it at the same time. The local instance of an application communicates with the local replica, resulting in single-digit-millisecond latency. The data that is written to a local instance is asynchronously replicated across all Regions selected for a global table (the data is not sharded). It is easy to convert a table to a global table and add or remove regions from a global table. Data is secure as it is encrypted in transit and at rest. This approach opens up the possibility of update conflicts, which are resolved by using the "last writer wins" technique.

Amazon MemoryDB Multi-Region is an active-active, multi-Region database suited for customers who need microsecond reads and up to 99.999% availability. It uses asynchronous replication across multiple Regions, preserving in-memory performance.

Amazon Keyspaces also has an active-active multi-Region database ideal for customers who need an Apache Cassandra–compatible database service with up to 99.999% availability.

Since the data in each Region is eventually consistent with the data in the Region where it was initially written, an application that is unable to reach its local endpoint can access the data from an alternate Region. An eventually consistent approach opens up the possibility of data loss because of 'in flight' changes that may not have been successfully replicated before the failover. If your workload cannot tolerate any data loss at all, you need a database that offers strong consistency across Regions.

Meeting users where they are with multi-Region deployments



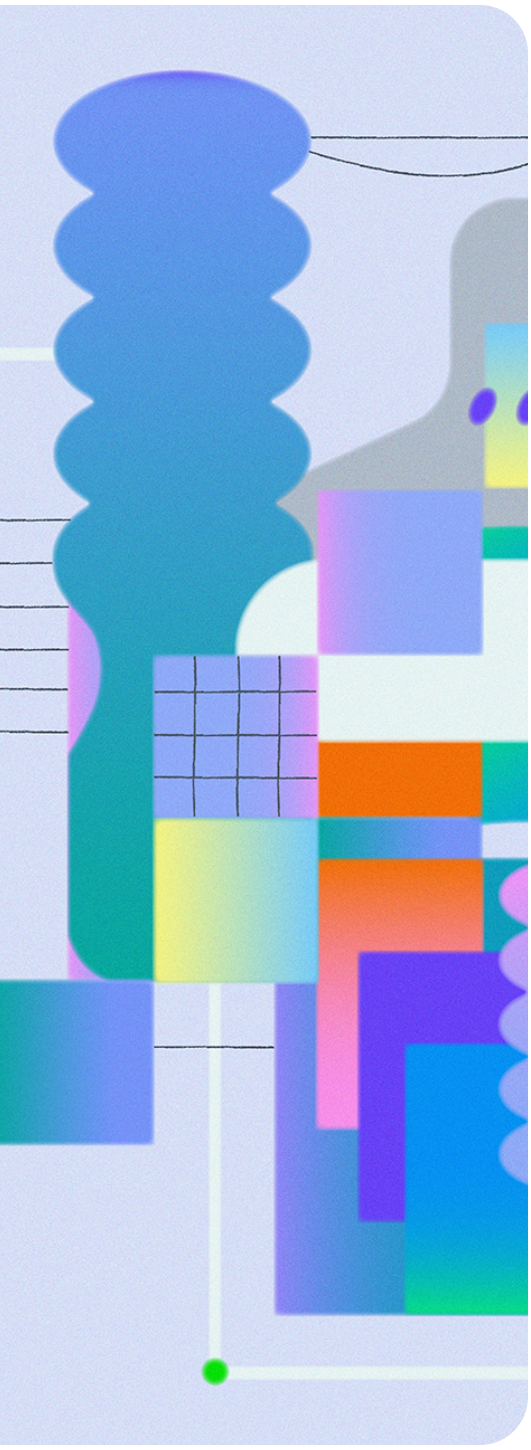
Active-active multi-Region databases with strong consistency

Active-active multi-Region databases with strong consistency guarantee the application is always reading the most up-to-date and consistent view of the data regardless of the Region they are located in. All transactions written in one Region are reflected in other Regions with strong consistency, which eliminates issues caused by data conflicts or divergence across Regions. Amazon Aurora DSQL and Amazon DynamoDB global tables are both examples of databases that offer strong consistency across Regions.

Amazon Aurora DSQL is a serverless distributed SQL database with the fastest reads and writes among distributed SQL databases, virtually unlimited scale, zero infrastructure management, and highest availability for always-available applications. Its high writes throughput is critical for a broad range of write heavy applications like internet-scale ecommerce business applications that need to accurately process peak order transactions. For reads, DSQL is designed to allow fast, strongly consistent reads with no latency penalty from data synchronization across Regions.

Aurora DSQL is ideal for customers who are building multi-Region distributed applications that can support millions of end users worldwide with effortless scaling. Data-driven applications, including SaaS applications, will benefit from the serverless, distributed architecture as it efficiently scales with microservices and serverless design patterns. Aurora DSQL automatically scales to meet any workload demand without database sharding or instance upgrades. It eliminates scaling bottlenecks while maintaining performance, offering virtually unlimited horizontal scaling. You also have flexibility to scale reads and writes independently.

Meeting users where they are with multi-Region deployments



With Aurora DSQL, there is no need to provision, patch, or manage database instances, and all updates and security patching happen with no downtime and zero impact to performance. It is designed for 99.99% single-Region and 99.999% multi-Region availability with no single point of failure and automated failure recovery. Applications are always-available as Aurora DSQL allows applications to read and write to the same DSQL cluster from any regional endpoint with strong data consistency in and across multiple Regions. There is no need to build custom application logic and manual database failover procedures, including managing the complex task of ensuring data consistency and data recovery during failover operations. There is no need to account for missing data during failure recovery due to replication lag or inconsistent database recovery logs. If you need to cut over your applications to an alternate Region, they will always access the most recent data.

For NoSQL customers, Amazon DynamoDB global tables now offers multi-Region strong consistency, offering customers the flexibility to choose strongly consistent reads for their applications across multiple Regions. If your application processing is interrupted in one Region, there is no need for a database failover as global tables multi-active architecture allows customers to read and write to any replica table. Global tables also eliminate the difficult work of replicating data between Regions and resolving update conflicts for multi-Region workloads. Its multi-Region strong consistency synchronously writes data to at least two replica tables, ensuring customers' data is persisted across two Regions before it is read by the application, and customers can route their application traffic to a different Region and be assured that their application is reading the latest data. When customers choose multi-region strong consistency, they also obtain the highest availability, virtually unlimited scalability, and zero infrastructure management already available in DynamoDB global tables.

04 Performing analytics and search on operational data with zero-ETL

Performing analytics and search on operational data with zero-ETL

Databases that are optimized for operational workloads are usually not the optimal choice for use cases like analytics and search. Operational databases are optimized for processing transactions, updating records, and managing real-time business operations.

Performing analytics or search on operational data often requires the process of moving data from data stores optimized for operational workloads to those optimized for analytics or search. Traditionally, this task is managed through an extract, transform, and load (ETL) process, where data engineers build, test, and maintain pipelines.

The challenges with ETL

ETL presents considerable challenges. First, developers have to design an ETL pipeline architecture. They have to decide where to extract the data from—often it comes from multiple sources. Then, they have to write code to transform the data to remove duplicates, filter outliers, retrieve missing data, and identify corrupted data. And after all that, they have to load their transformed data to its new destination, which typically requires more custom coding. If something changes, like a change to a table name or a new field, then all the custom code must be updated and redeployed. ETL pipelines are complex, brittle, inflexible, and subject to scalability limits.

The time needed to create or modify data pipelines makes ETL unsuitable for near real-time applications, such as those detecting fraudulent transactions, optimizing online advertisements, or tracking supply chains. This creates significant barriers to achieving business objectives, such as exploring new opportunities or reducing risks. Also, the data movement lag associated with ETL carries a negative impact, particularly when insights gleaned from analytics of transactional data have relevance for only a limited time frame.

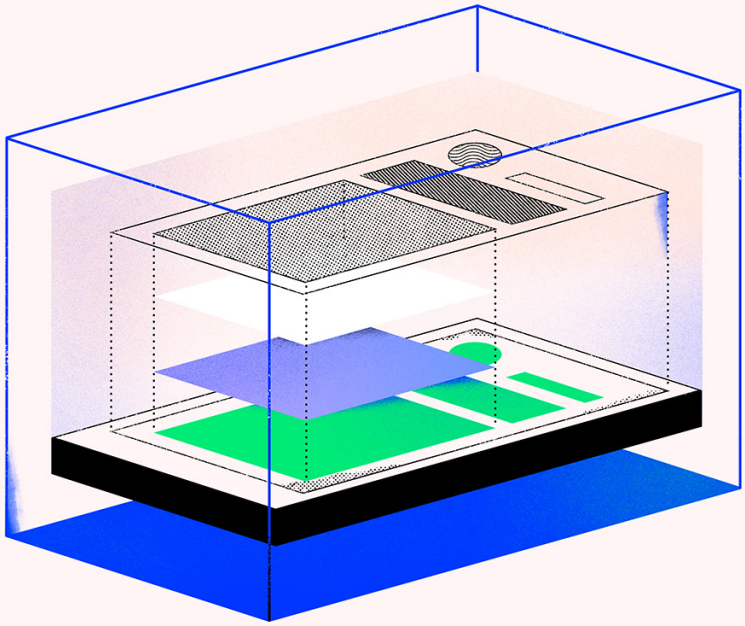
Performing analytics and search on operational data with zero-ETL

AWS zero-ETL integrations

Zero-ETL integrations are no-code integrations between data services so you can gain insights from your data faster. It lets you make your application data available for analytics, search, and AI use cases in near real time with just a few clicks.

Performing analytics and search on operational data with zero-ETL

Zero-ETL integrations open up near real-time analytics and search use cases on petabytes of operational data without the customer having to maintain data pipelines. You can use these zero-ETL features to consolidate data from multiple instances of the source database into a single Amazon Redshift data warehouse to derive holistic insights across several applications, while also consolidating your core analytics assets and gaining significant cost savings and operational efficiencies. Customers can access the core capabilities of Amazon Redshift, such as materialized views, data sharing, and federated access to multiple data stores and data lakes. Zero-ETL integration with Amazon Redshift enables customers to combine near real-time and core analytics to effectively derive time-sensitive insights that inform business decisions. Similarly, multiple instances of the source database can be consolidated into OpenSearch Service, providing a holistic search experience across several applications.



05 Unlocking vector search for generative AI applications

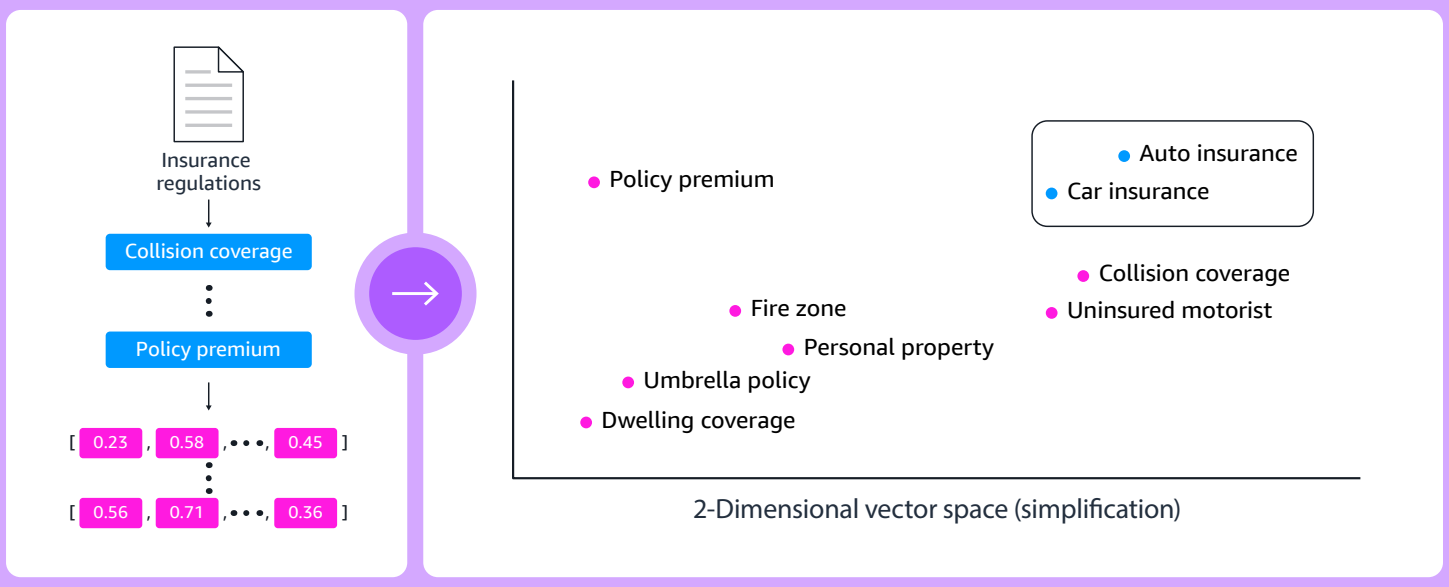
Unlocking vector search for generative AI applications

Generative AI holds the promise of ushering in a new wave of innovative applications, and these applications rely on foundation models (FMs) or large language models (LLMs). FMs are trained on vast datasets, such as all the content accessible on the internet, but they need to be augmented with domain-specific data to produce accurate and relevant results for that domain. FMs can return inaccurate responses because: 1) they have a knowledge cut-off date and can't accurately respond to questions referring to events more recent than the cut-off date and 2) they try to respond to a question requiring domain specific knowledge that wasn't part of the training data set. In both cases, prompt instructions that are not correctly optimized may force the model to provide a response even if the model lacks context to answer the question. Augmenting generative AI applications for domain-specific accuracy is crucial for organizations looking to build domain-specific generative AI applications.

Unlocking vector search for generative AI applications

Adding domain-specific data using databases

One approach to adding domain specificity to generative AI applications relies on encoding domain-specific data into n-dimensional vectors. In generative AI, a vector is a data type that provides a mathematical representation of its source content, such as text, images, video, audio, or other structured or unstructured data types. Vectors are generated using embedding models, which are specialized machine learning models that capture the semantic meaning of their source data. This transformation provides a way to compare the relationships between the data through “distance functions” that determine the similarity between two vectors. For example, an embedding model may generate vectors that show “auto insurance” is more similar in meaning to “car insurance” than it is to “collision coverage.”



Unlocking vector search for generative AI applications

Retrieval Augmented Generation to enhance contextual relevance

Vector similarity search is often used when building applications that leverage retrieval-augmented generation, or RAG. RAG is a technique that lets you bring in more context to a foundation model (FM), such as information about a specific situation or facts that weren't available when the FM was trained. For example, when a generative AI app user asks a question, the question is routed to an embedding model, and the application uses that embedding to perform a vector similarity search to determine if it needs to supply additional context to the FM to answer the question.

There are two workflows in RAG: ingestion and agency. The ingestion workflow is where source data (such as, text, images, video) are transformed into vectors through an embedding model. As part of this transformation, it may be necessary to break up the data into smaller "chunks" due to the size of the "context windows" that a specific model can process. The generated vectors are then stored in a database for retrieval during the agency phase. During the agency phase a user submits a request that may require additional context. This requires running a similarity search over vectors, which can be handled through brute-force search (searching over every vector in a dataset) or through "approximate nearest neighbor" (ANN) search that searches over a subset of data to make a "best guess" at the most similar results. Examples of ANN algorithms used in databases include Hierarchical Navigable Small World (HNSW), and Inverted Files with Flat Compression (IVFFlat).

AWS databases that support vectors as a data type offer rapid lookup capabilities, along with core database features like scalability, availability, and security. The outcome of vector searches typically yields a ranked list of vectors with the highest similarity scores to the vector that represents the prompt, along with the original source data (the chunk).

Knowledge bases for Amazon Bedrock

For automated RAG, Knowledge Bases for Amazon Bedrock is a great option. It automates the RAG workflow, including both the ingestion workflow (fetching documents, chunking, creating embeddings, and storing them in a vector-enabled database) and the runtime orchestration (creating embeddings for the end-user's query, finding relevant chunks from the vector database, and passing them to an FM).

GraphRAG is a technique that uses a knowledge graph during the retrieval part of the RAG process. Based on Amazon Neptune, GraphRAG automatically generates graphs that link data across multiple sources, including unstructured data like text, images, video, and audio. This is quite remarkable as over 80 percent of organizations' data is unstructured data. During the retrieval process, GraphRAG will automatically traverse these graphs to provide more comprehensive, accurate, and explainable responses from LLMs—all with a single API call. You can select the unstructured data stored in Amazon S3 and Amazon Neptune to automatically create a graph (powered by Amazon Neptune) that can be used during retrieval as part of the RAG process. With Bedrock Knowledge Bases, interoperability between Amazon Bedrock, Amazon Neptune, and Amazon S3 is built in to provide you an automated workflow for GraphRAG.

The Knowledge Bases setup process involves key decisions around the specific FM and the integrated database to use for the knowledge base. A number of database options are available. For example, you can choose Aurora, OpenSearch Serverless, or Neptune within the Amazon Bedrock console, and Amazon Bedrock will automatically store your vectors in the database of your choice and pull vectors to augment queries with contextually relevant data in support of RAG. All three databases have a quick create setup making it easy to get started.

Unlocking vector search for generative AI applications

Storing vectors alongside your application data reduces the need for data processing pipelines or complex application logic to combine data across different systems, and reduces overhead and licensing costs.

Many use cases, including natural language processing, rely on databases with vector capabilities for their applications. These use cases necessitate both semantic understanding and precise data matching. Vector storage and similarity search are vital for addressing the limitations of LLMs related to relevance and accuracy in a particular data domain.
Vector search can also add more current information that is absent from LLMs, giving LLMs an external memory. LLMs are updated infrequently because of the high computational cost of retraining them on new data. Vector embeddings, on the other hand, can be updated more frequently as new data is added or existing data is updated. Subsequently, the updated vector embeddings can be discovered through vector similarity searches and can become part of the up-to-date context that is sent to LLMs along with the end-user's queries.

Storing vectors and operational data together

Vectors are a data type that is supported across many databases that customers currently use. This eliminates the need to migrate data to a specialized vector database while enabling you to take advantage of vector database capabilities within existing architectures. AWS databases with vector database capabilities allow the storage of vast amounts of vectors, perform vector searches without data movement, and implement RAG use cases. This integration allows for access to both your source data and vector embeddings (generated from your source data) from the same place. You will also benefit from the advantages of a fully managed database, including cost savings and the elimination of undifferentiated heavy lifting.

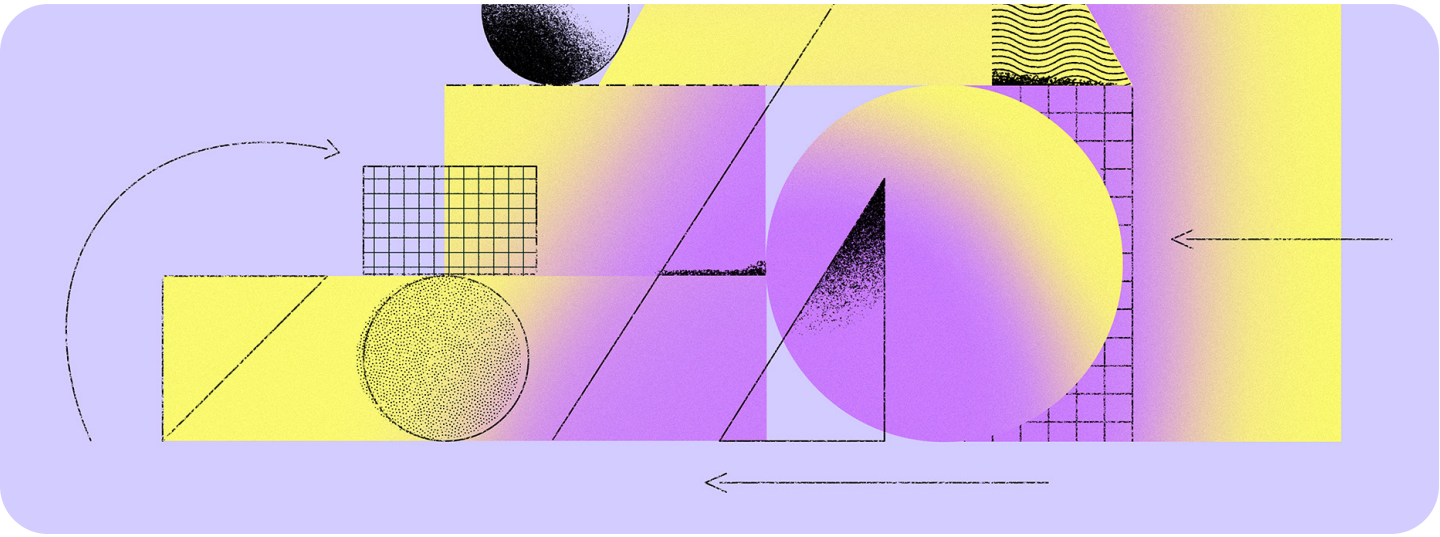
In contrast, standalone or specialized vector-enabled databases necessitate data migration. Furthermore, the need to integrate an additional database with your applications leads to significant changes in your existing applications. Opting for a separate specialized database creates challenges related to redundant data, data consistency, specialized skills, and additional licensing costs. On the other hand, vector database capabilities embedded in familiar operational databases significantly reduce operational burden.

Unlocking vector search for generative AI applications

AWS provides a range of databases with vector capabilities, including:

- Amazon Aurora PostgreSQL
- Amazon RDS for PostgreSQL
- Amazon DynamoDB (via zero-ETL integration with Amazon OpenSearch Service)
- Amazon Neptune
- Amazon MemoryDB
- Amazon DocumentDB
- Amazon OpenSearch Service and Vector Engine for OpenSearch Serverless

You can fully unlock the benefits of generative AI with your existing database, avoiding the operational complexity of learning and managing a new database. Plus, you gain access to database performance, scalability, availability, and security tailored to your application's requirements.



06 Bridging Kubernetes workloads to AWS databases

Bridging Kubernetes workloads to AWS databases

DevOps has become a popular approach with the promise of delivering faster software development and deployment. However, DevOps is not a one-size-fits-all approach; organizations have the flexibility to adopt the practices, tools, and technologies that best suit their needs. Among the technologies often embraced within DevOps is Kubernetes (K8s), a container orchestration system. But how do the worlds of K8s and fully managed database services on AWS come together? Fortunately, you can manage AWS databases as external AWS managed resources directly from K8s. The key cluster management actions are supported, like scaling up, down, in, or out, and scaling the number of read replicas; as well as creating other database resources such as snapshots, parameter groups, and subnet groups. Managing AWS databases in a K8s world relies on controllers that extend K8s. Controllers use the K8s APIs to control the lifecycle of custom resources that are not built into K8s, like databases and caches. By using a controller for an AWS database, the management of the database can be automated, much like a native K8s resource.

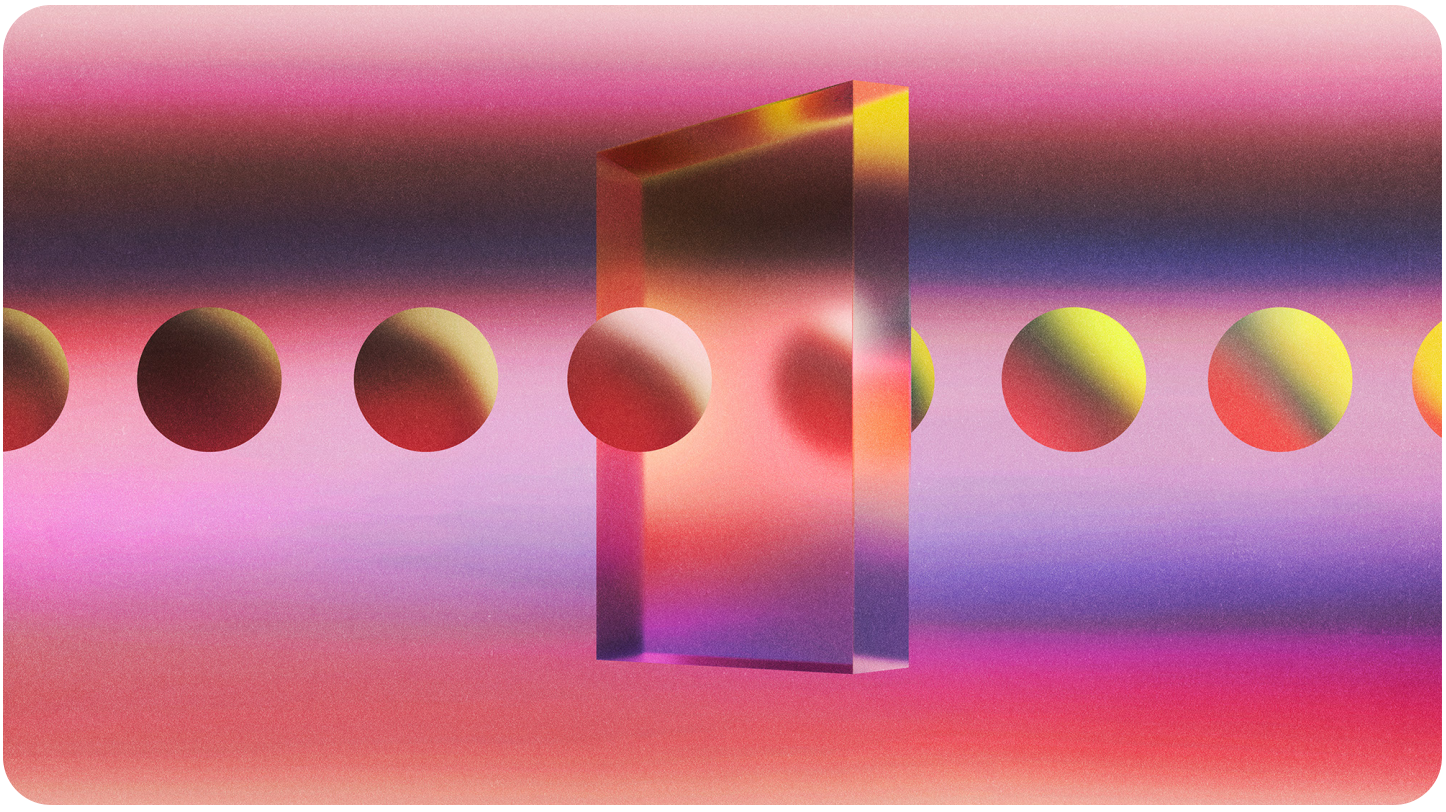
AWS Controllers for Kubernetes (ACK) adopt the approach of managing AWS databases as external resources. With ACK, you can take advantage of AWS managed services for your K8s applications without needing to define resources outside of the K8s cluster or run services that provide supporting capabilities like databases, caches, or message queues within the K8s cluster. Each ACK service controller manages resources for a particular AWS service, and is packaged into a separate container image that is published in a public repository.

Developers can use their knowledge of the K8s resource model to work with AWS databases, just like any other K8s resource. ACK enables K8s users to describe the desired state of AWS resources using the K8s API and configuration language. ACK resources are defined using YAML-formatted manifest files to both initially define the resource configuration and to modify it. After the manifest file is created, the resource it defines is created by using the file name as the input argument to the Kubernetes “kubectl apply” command. To change a resource configuration, you simply edit the appropriate parameters in the existing resource manifest file, then call the “kubectl apply” command in the same manner as the initial resource creation. Manifest files can be version controlled, alongside your application code, so that changes over time are easily tracked and attributed to changes to application code.

Bridging Kubernetes workloads to AWS databases

Also, ACK is declarative, so you can define the desired state and allow the controller to take the necessary steps without defining an imperative list of steps. The K8s control loop manages the state of your cluster as well as the configuration you passed in for your AWS resource. Periodically, an ACK service controller will look for any drift and attempt to remediate. A single consolidated approach using ACK makes it easier to adopt GitOps for automating your deployments.

ACK service controllers run in a container on any K8s distribution on premises or in the cloud. Hence, they are not limited to Amazon Elastic Kubernetes Service (Amazon EKS). Controllers are currently available for RDS, Aurora, DynamoDB, ElastiCache, MemoryDB, Keyspaces, and more.

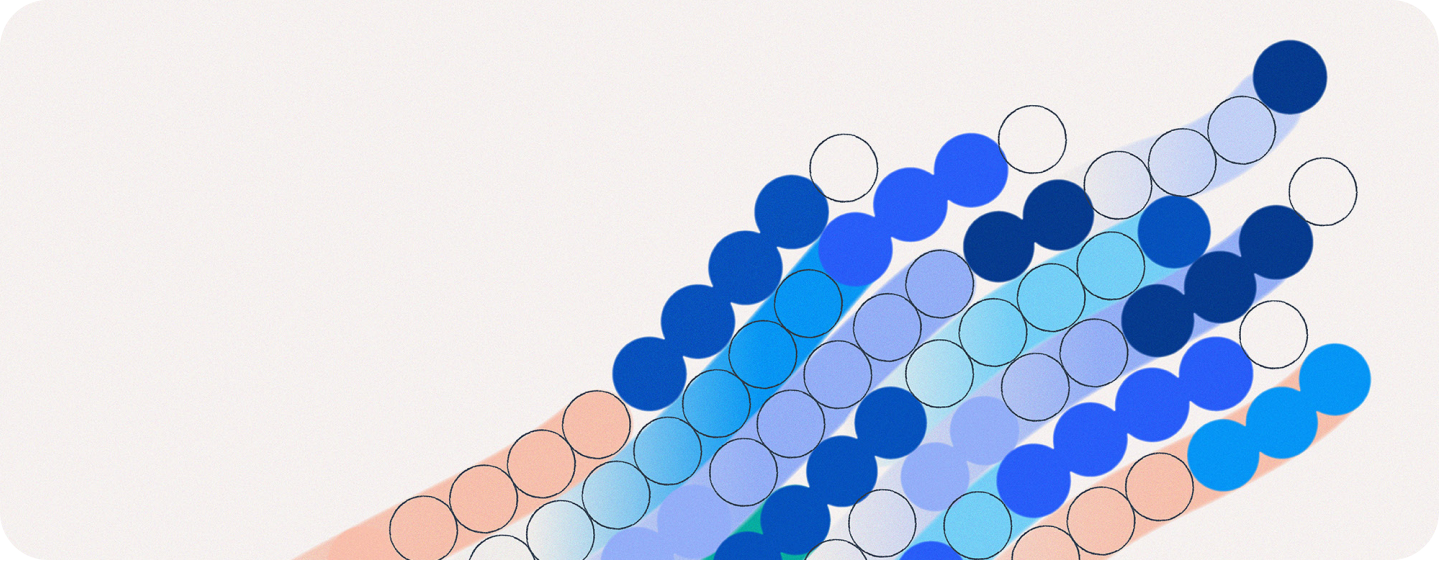


07 Migrating your data to AWS

Migrating your data to AWS

Migrating to the cloud is more important than ever so you can take advantage of your data for generative AI and machine learning. We realize you need a practical approach to migrating your existing on-premises database to the same type of database in the cloud (homogeneous migration) or switching to a different type of database in the cloud (heterogeneous migration). Cloud-to-cloud database migrations are another important evolutionary path.

AWS offers tools and experts to help assess, plan, and build the right migration path for your company. The AWS Database Migration Service (AWS DMS), which includes Schema Conversion with generative AI, helps you migrate databases to AWS while maintaining uninterrupted operations on the source database. DMS supports an extensive list of sources and targets. This method is useful when you have to migrate the database code objects—including views, stored procedures, and functions—as part of the database migration, or have to convert between different database engines or data models. This solution is applicable to databases of any size. It keeps the database available for the application during migration and allows you to perform validation of the migrated data, while the data is getting replicated from source to target, thereby saving time on data validation.



Case study: S&P Dow Jones Indices

S&P Dow Jones Indices
A Division of S&P Global

S&P Dow Jones Indices is a division of S&P Global Inc., an American publicly traded corporation headquartered in New York City. The organization provides solutions for tracking market performance, evaluating portfolios, and developing investment strategies. Its indices number in the hundreds of thousands and span every major asset class, as well as markets around the globe. S&P DJI faced the challenge of managing vast amounts of data while improving database performance.

To reduce maintenance time, S&P DJI re-platformed to Amazon Relational Database Service (Amazon RDS), gaining benefits like business continuity and high availability. To enhance performance, S&P DJI migrated some on-premises MySQL databases to Amazon Aurora MySQL-Compatible Edition and others to Aurora PostgreSQL-Compatible Edition. S&P DJI also adopted Amazon Aurora Global Database and Aurora Serverless for scalability and efficiency, particularly for applications involving machine learning and large-scale data ingestion. AWS Database Migration Service (AWS DMS) and AWS Schema Conversion Tool (AWS SCT) enabled seamless migrations, cutting migration time to just 10 hours.

Read the full story →

"After migrating to Aurora, we could increase the efficiency of our operations, our quality of service, and the speed and performance of many of our databases, and also cut down our operational costs significantly. This is just a new beginning. We are going to continue to use many AWS services."

Shivakumar Bangalore
Senior Director of Database Engineering, S&P Global Inc.

08 Conclusion

Conclusion

The AWS database portfolio is the result of an ongoing, focused investment strategy to provide customers with feature-rich and cost-effective database products.

Our commitment to innovation is motivated by our belief that our innovation can clear the path for your innovation.

Data-driven applications mark a shift in how applications are designed and built. Unlike applications built on monolithic architectures, these applications are based on distributed microservices-based architectures. The decomposition of monoliths into microservices facilitates an independent best-fit database choice for each microservice, and AWS offers the right tool for the job. The workload-specific focus of each database is key to how AWS can optimize each database for performance at scale. In addition, each database is future-proofed with the seamless addition of features that support emerging use cases like generative AI, and ample headroom for growth in the number of users, the geographical dispersion of users, and data capacity. As every AWS database is fully managed, database operations, maintenance, and capacity adjustments are automated and non-disruptive. AWS continues to invest in making the maintenance of your databases effortless.

To get started, you can gain free hands-on experience with many of our databases—check out AWS Free Tier. AWS also offers Optimization and Licensing Assessment (OLA) to help you evaluate options to migrate to the cloud. When you complete this form to request an assessment, the AWS OLA team can help you. You can also learn more about AWS databases by heading over to the database category page where you'll find related content, additional documentation, and links to each of the database service pages.