



Building modern applications using AWS databases

CONTENT

- Preface** 4

- Section 1: Key characteristics of modern applications** 5
 - Built on cloud infrastructures..... 5
 - Built using low-code and no-code technologies..... 6
 - Built using loosely coupled microservices-based application architectures 6
 - Delivers fast time to value with DevOps automation and practices that accelerate software delivery..... 6
 - Built for global operation with local performance..... 7
 - Delivers insights that use embedded machine learning and artificial intelligence 7
 - Built with the highest standards for security, identity, and compliance 7

- Section 2: How AWS databases support modern applications**..... 8

- Section 3: Making best-fit database choices** 9

- Section 4: Meeting performance requirements at scale** 11
 - Automated capacity changes with serverless databases 11
 - Horizontal scaling that automatically distributes load across multiple nodes 14
 - Vertical scaling with a broad choice of instance types and sizes..... 17
 - In-memory benefits of low-latency, high-throughput data access..... 18

- Section 5: Meeting users where they are with global deployments**..... 19
 - Multi-Region read replicas 19
 - Multi-active global tables 19



CONTENT

- Section 6: Maintaining operational stability: availability, upgrades, updates, patches, and security** 20
 - High availability..... 20
 - Recovering from Region-wide outages..... 20
 - Fully managed services 21
 - Blue/green deployments 22

- Section 7: Maintaining the highest levels of security, governance, and compliance** 23

- Section 8: Performing analytics and search on operational data with zero-ETL** 24
 - The challenges with ETL 24
 - AWS zero-ETL future 25

- Section 9: Unlocking vector search for generative AI applications** 26
 - Adding domain specificity using databases with vector capabilities 26
 - Retrieval Augmented Generation to enhance contextual relevance..... 27
 - Storing vectors and operational data together..... 28

- Section 10: Bridging Kubernetes workloads to AWS databases** 30

- Section 11: Migrating your data to AWS** 32

- Conclusion**..... 33

PREFACE

This whitepaper examines the role of the Amazon Web Services (AWS) operational database portfolio in support of modern applications. We start by defining the characteristics of modern applications and the vital technologies that are core to modern applications. We share how the features and benefits of AWS operational databases support these characteristics, while helping you reduce your total cost of ownership. The right database choices can create new and engaging customer experiences, process transactions faster, and spur innovation.

Topics include:

- Making the best-fit database choices for your microservices
- Support for internet scale performance, globally dispersed users, and large data volumes
- Automated operations and capacity management
- High availability and operational stability
- Security, governance, and compliance
- Performing analytics and search on operational data with zero-ETL (extract, transform, and load)
- Using operational databases in generative artificial intelligence (AI) applications
- Deploying databases for container-based applications and Kubernetes
- Data migration for achieving application architecture modernization

Databases play a vital role in the modern application technology stack, and the architecture decisions made today are an investment for the next decade. Making the right choices on the technologies to future-proof your applications requires an understanding of core application and database technologies and trends.

Key characteristics of modern applications

As organizations look to increase the pace of innovation and build new customer experiences, modernizing how you build and operate applications is key. From customer-centric applications that become the main connection points with customers to enterprise software that controls back-office operations, applications are the driving force behind every successful organization. Built with the latest technologies, modern applications are the solution for organizations to innovate faster and improve performance, security, and reliability while lowering their total cost of ownership.

Before we delve into how AWS databases form a critical foundation upon which modern applications are built, it is imperative for us to describe the key characteristics of modern applications.

Built on cloud infrastructures

The move to cloud infrastructures is often motivated by the freedom gained by offloading daily tasks related to maintenance, capacity adjustments, availability, and stability. Organizations depend on cloud providers to handle these undifferentiated tasks. Often, it is difficult to hire and train people in the latest technologies across different geographies. Moving to a cloud infrastructure mitigates this risk.

With cloud infrastructures, you also benefit from the continuous innovation of the cloud provider. The services you use gain features with little or no effort on your part. Also, pre-integration of services provides a faster path to value. For example, cloud technologies like machine learning, analytics, and business intelligence services are easily accessible so that organizations can benefit from these services to compete and innovate.

Another expectation of cloud infrastructures is cloud effectiveness. Cloud infrastructures offer a pricing structure that charges for capacity that is actually used, with no long-term commitments. Moving to a public cloud infrastructure reduces costs while exceeding on-premises performance.

SECTION 1

Built using low-code and no-code technologies

Low-code or no-code software development abstracts away the complexity of writing code by using a visual drag-and-drop interface to build and configure applications. Applications can be built using pre-built visual components like user interface elements, data connections, and building blocks for the underlying logic and business rules. Application development will shift to application composition, assembly, and integration by citizen developers within the teams that use the application.

Built using loosely coupled microservices-based application architectures

Microservices support incremental, rapid release of software, and reduce the risk of failure inherent in the all-or-nothing monolithic approach. With microservices, development teams can focus on a scope that is bounded by a business function, which results in more optimized, hardened code with deeper functionality. A bounded scope empowers each team to develop expertise on the business function they are automating. Armed with this expertise and with agile software development practices, development teams can add deeper functionality faster. Because microservices are isolated services with minimal interdependence, each microservice team can release their service on their own schedule.

However, building decoupled microservices-based applications requires agreed upon interface contracts, governance, and data exchange formats for inter-microservices communications. Adherence to these prerequisites offers microservices development teams the autonomy and freedom to choose the best-fit technology for their microservice. With agreed upon data exchange formats, event-based architectures can serve as the connective tissue for inter-microservice communications. These benefits can come together in a way that accelerates the time to market for innovation.

Delivers fast time to value with DevOps automation and practices that accelerate software delivery

Software development is only the beginning of the software development lifecycle. Application code goes through a series of steps for integrating changes and testing to improve the quality of software. Modern DevOps techniques, like continuous integration and continuous deployment, reduce the time it takes to deploy in production from months or weeks to hours.

SECTION 1

Built for global operation with local performance

Modern applications can scale to millions of globally dispersed users and to petabytes of data—with near real-time response. For this to occur, modern applications and their infrastructures must conquer traditional limitations related to network and database latency with data volume capacity that can grow as needed. Databases that support modern applications need to automatically scale up or down both vertically and horizontally with automated load balancing. In addition to scaling up, automatically scaling down is important for releasing capacity that's not being used and not paying for unused capacity.

Delivers insights that use embedded machine learning and artificial intelligence

Modern applications offer analytical features that developers can incorporate without machine learning expertise. These analytical features can provide personalized recommendations, modernize contact centers, improve safety and security, and increase customer engagement by delivering insights.

Built with the highest standards for security, identity, and compliance

Modern applications offer a no-compromise security, identity, and compliance posture. Authenticated users are only allowed to access data that they are authorized to access. Data is protected through encryption, and potential security misconfigurations, threats, or malicious behaviors are quickly identified and remediated. Automated compliance checks provide visibility through ongoing real-time reporting and monitoring. Modern applications use the automation of advanced operational techniques to remove these risks and vulnerabilities and facilitate safe and secure management of infrastructure changes.

How AWS databases support modern applications

To deliver on their benefits, modern applications depend on some key characteristics of AWS databases, such as the ability to:

- Choose the best-fit database technology for each microservice
- Meet performance requirements at scale
- Deploy globally to meet users where they are
- Maintain operational stability with high availability, automation of routine maintenance, automated capacity changes with serverless databases, and the highest levels of security, governance, and compliance
- Use pre-built zero-ETL integrations across data services
- Use vector data in operational databases to extend or build new applications that incorporate generative AI
- Deploy with container-based microservices
- Use AWS tools and technologies for database migrations

The rest of the paper elaborates on each of these characteristics.

SECTION 3

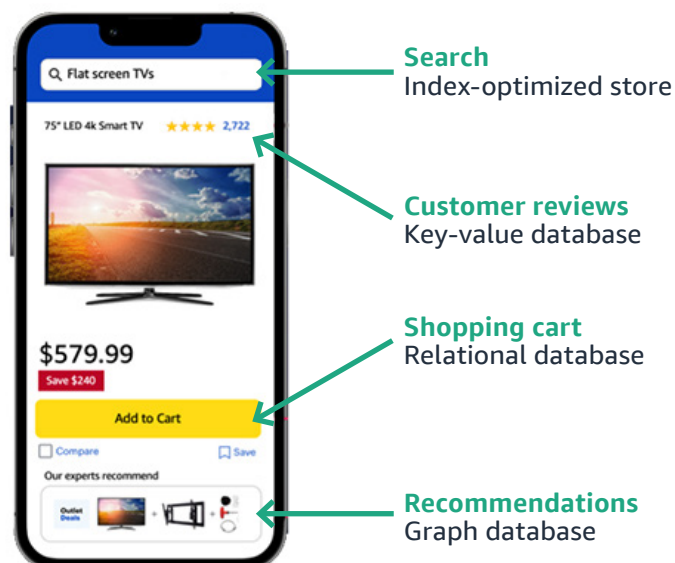
Making best-fit database choices

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.

Relational databases are no longer the single best answer for all database workloads. While relational databases are still essential—in fact, they are still growing—a relational-only approach no longer works in today's world. 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.











A modern UI is a symphony of workloads, each of which has its own unique database requirements. Easy access to a portfolio of purpose-built databases is now a critical success factor for modern applications.

In the example of an ecommerce shopping application, you can use an index-optimized store like [Amazon OpenSearch Service](#) to help users quickly find relevant information. Then, you can use a key-value database for showing customer feedback that uses a five-star rating system. For the purchase button, you can use a relational database to obtain the transactional integrity both for inventory and financial accounting. Then, you can use a graph database like [Amazon Neptune](#) to power personalization algorithms, such as recommendations on what additional things the end customer might want to buy if they're purchasing this item. This is how modern 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.



SECTION 3

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 for Redis	Wide-Column  Amazon Keyspaces	Time-Series  Amazon Timestream	Ledger  Amazon QLDB

AWS has the [broadest set of managed database services](#) of any cloud provider. Altogether, AWS offers more than 15 types of database engines, each built to uniquely address specific customer needs. In addition to [relational databases](#), the AWS database portfolio supports a full range of purpose-built databases including [key-value](#), [document](#), [graph](#), [in-memory](#), [search](#), [wide-column](#), and [time-series](#).

Meeting performance requirements at scale

The need to change database capacity is a frequent occurrence. Often, organizations have some seasonality or event triggers associated with their workloads. 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, which impacts application performance and is felt by customers and other users. If the capacity is set to meet peak workloads, 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. In most cases, manual scaling is not seamless and it involves some business disruption.

Automated capacity changes with serverless databases

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

S&P Dow Jones Indices

A Division of **S&P Global**

Case study: S&P Dow Jones Indices

[S&P Dow Jones Indices](#) (S&P DJI), a business segment of S&P Global Inc., offers innovative indices and a range of leading-edge solutions to help investors identify, measure, and capitalize on global investment opportunities. S&P DJI has nearly 200 databases supporting its core applications, with sizes ranging from 500GB to 15TB.

"We began our migration journey with Amazon Web Services (AWS) migrating from on-premises to MySQL on Amazon EC2 and then directly to Amazon Aurora MySQL-Compatible Edition, utilizing AWS Database Migration Services (AWS DMS). We recently adopted Aurora Serverless v2 for a new application that includes machine learning capabilities using AWS SageMaker for sector wide asset classification. Setting up AWS Glue jobs to ingest data from Twitter feeds led to a very dynamic workload for this application."

SECTION 4

Hence we adopted Aurora Serverless v2, which can scale the compute resources dynamically to support processing all the AWS Glue jobs and AWS Lambda functions, while being able to ingest massive volumes of data. We have also enabled write forwarding to keep our application active in more than one region, with Amazon Aurora Global Database.”

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

Automated capacity management is optimized for the specific AWS serverless database, based on the way it allows you to interact with your data. In [Amazon Aurora Serverless v2](#), you deploy clusters with serverless instances. With such instances, the compute capacity, amount of memory, and network throughput of each instance scales automatically (vertical scaling) based on demand. If the workload is read-heavy, you can scale out only reads by adding instances in a serverless configuration (horizontal scaling). Each reader remains within the minimum and maximum values you specify for the cluster. For batch applications or business reporting that experience periodic spikes with a lot of idle periods, this configuration provides better cost-efficiency without impacting performance. Aurora continuously tracks the utilization of resources such as CPU, memory, and network. When capacity is constrained by any of these, the database scales incrementally, in a non-disruptive way, even when there are thousands of active connections and transactions occurring.

Other AWS serverless databases, like Amazon DynamoDB, provide a different experience for achieving the same objectives. For example, DynamoDB capacity and utilization is measured in read and write capacity units. With on-demand capacity mode, you pay for the amount of capacity units your workload consumes at any given point in time, and the service simply adjusts the resources available to match your desired capacity consumption.

Serverless options are available for [Amazon Aurora](#), [Amazon DynamoDB](#), [Amazon ElastiCache](#), [Amazon Neptune](#), [Amazon Keyspaces](#), and [Amazon Timestream](#). In each case, billing is based on usage and results in cost savings as high as 90 percent when compared to provisioning for peak capacity.

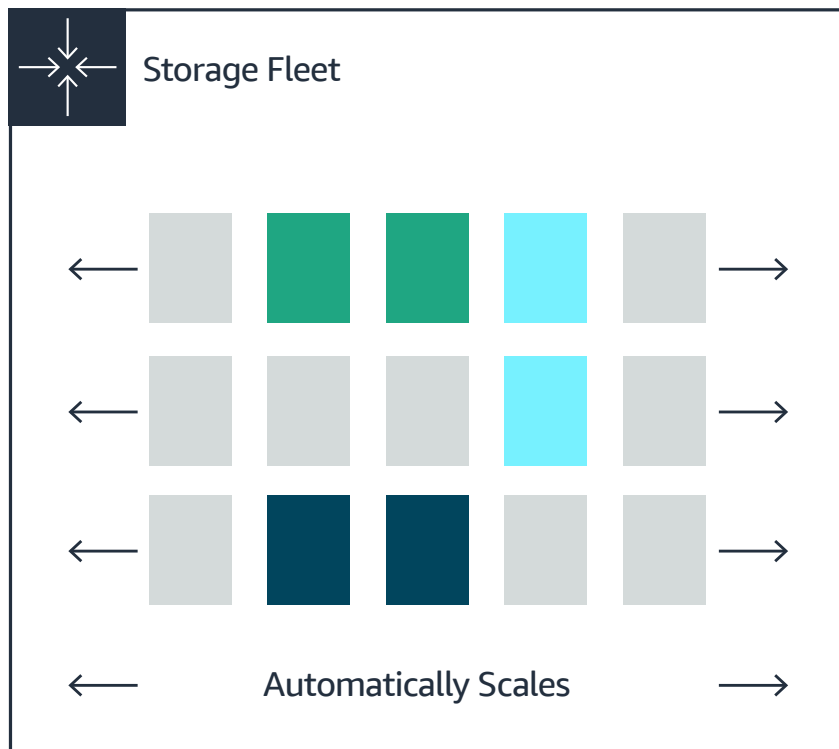
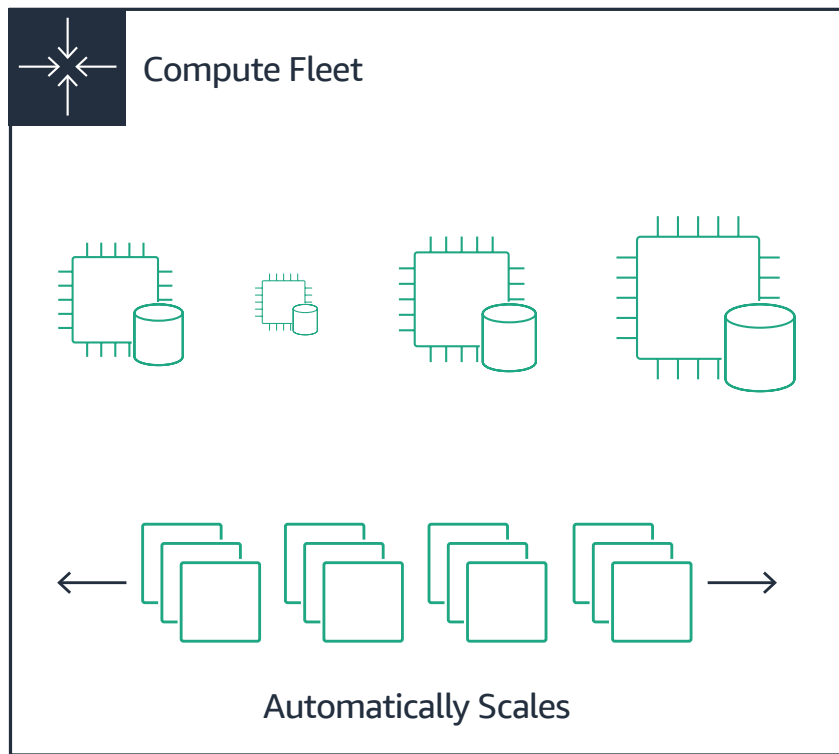


Case study: Careem

“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)*





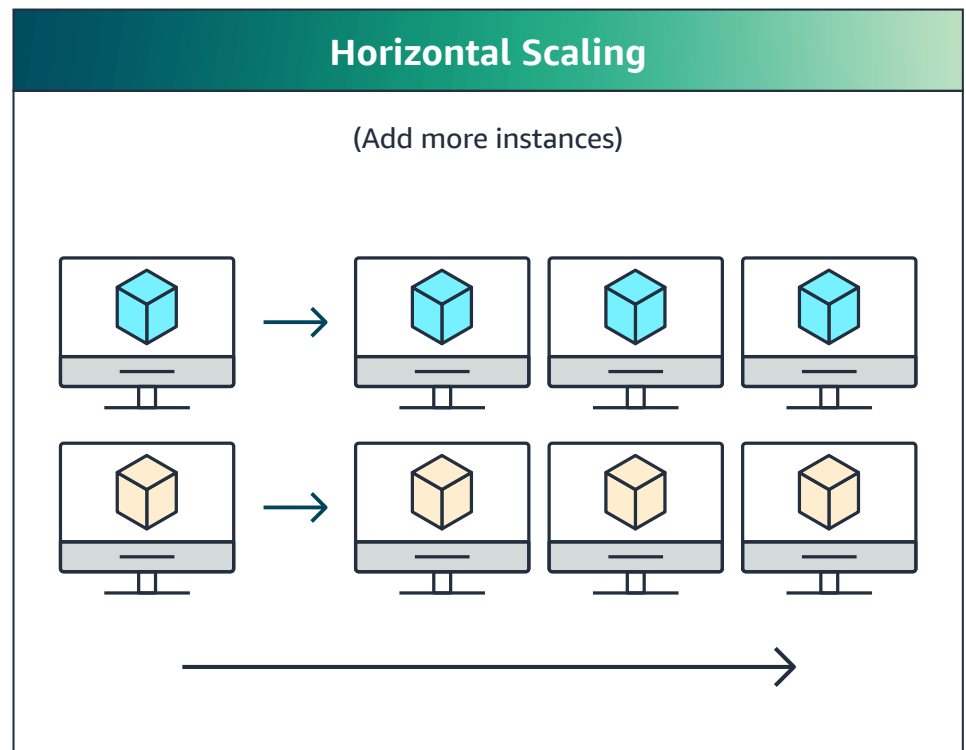
SECTION 4

Horizontal scaling that automatically distributes load across multiple nodes

Sharding for scaling writes, reads, and data volume

Changing the capacity of an AWS database can be accomplished in multiple ways. Vertical scaling re-hosts the database on a larger virtual machine. Horizontal scaling adds nodes to a cluster to scale out. Either or both in combination can be used to scale a database. Vertical scaling is often an easier choice with relational databases, which have multiple tables with interrelationships and referential integrity constraints across tables. It can be difficult to break up a relational data model into slices of data to be spread across multiple virtual machines. However, modern offerings like [Amazon Aurora Limitless Database](#) have conquered the challenge of horizontally scaling a relational database within a Region.

Non-relational databases, on the other hand, are relatively unstructured and have a simpler data model. The common practice with non-relational databases has always been to scale them horizontally. Many of the serverless databases covered earlier are also horizontally scalable. Because these databases are fully managed, AWS takes care of horizontally scaling your databases by using techniques like sharding and read replicas.



SECTION 4

The data volume benefits of sharding

A common approach traditionally used by non-relational databases for scaling the volume of data is a divide and conquer technique known as sharding. With sharding, data is split into smaller subsets (shards) and distributed across a number of physically separated nodes. 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. Sharding takes advantage of built-in data mapping and routing logic to send requests to the appropriate nodes, also known as shards.

Although sharding can be used with relational databases, it has traditionally been much simpler to set up for non-relational databases. With the large number of tables typical of relational databases, care must be taken to ensure that each node or shard has the same key ranges that are needed for joining tables locally, and to avoid queries that span multiple shards to the extent possible. These challenges are being addressed with recent approaches, like Amazon Aurora Limitless Database, by automating the process of creating and removing shards as the data volume grows or shrinks.

Because 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 nodes or shards. When nodes are added or removed, sharding automatically rebalances the data in a cluster. Sharding can increase the overall capacity of a cluster into petabytes of data.

Sharding for increasing the write and read performance

In addition to increasing the overall capacity of a cluster, sharding is effective in online transaction processing (OLTP) environments where the high volume of writes or transactions can go beyond the capacity of a single database instance, and write performance is of concern. Each shard only handles a subset of the write requests, increasing the overall write requests a cluster can handle.

Sharding is also effective for read-intensive workloads, provided that the queries are simple. Reading and joining data from multiple shards can erode the performance benefits of sharding. The inability to offer a consistent, global image of all data limits the sharded database architecture's ability to play an active role in an online analytic processing (OLAP) environment, where data analytic functions are usually performed on the whole dataset.

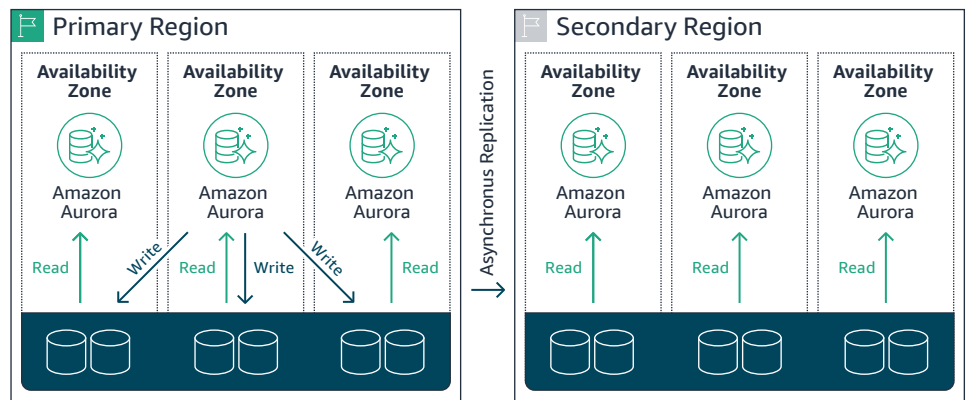
As examples, [Amazon Aurora Limitless Database](#), [Amazon DocumentDB Elastic Clusters](#), [Amazon ElastiCache](#), and [Amazon MemoryDB](#) offer sharding across Availability Zones (AZs).

SECTION 4

Optimizing reads with read replicas

Most application workloads are read-intensive rather than write-intensive. Improving the read performance of your database can go a long way toward improving overall performance.

Any single instance of a database is throughput constrained. For most applications, a majority of requests to the database are read requests. This presents an opportunity to overcome the throughput constraints of a single instance by scaling horizontally and adding secondary read replicas. Also, geographically dispersed users suffer the impact of poor response time because of network lag. Adding local read replica instances near end users significantly reduces the impact from network lag.



Case study: Atlassian

Using Amazon RDS and Aurora, which can both automatically scale, the company can improve performance and reduce costs. Atlassian uses both Amazon Aurora Read Replicas and [Amazon RDS Read Replicas](#)—which make it easy to elastically scale out beyond the capacity constraints of a single database instance for read-heavy database workloads—during times of peak traffic. The company can then downsize read replicas when they aren't needed, resulting in significant cost savings.

“Using Amazon RDS for PostgreSQL and Amazon Aurora PostgreSQL-Compatible Edition reduces the complexity of the scaling process. It’s definitely been a positive experience for Atlassian.”

Arul Shaji Arulappan, Principal Engineer, Atlassian

AWS databases support the addition of read replica instances within a cluster (across Availability Zones) or across Regions using multi-Region replication. Reads from the database are distributed across read replicas to spread the



SECTION 4

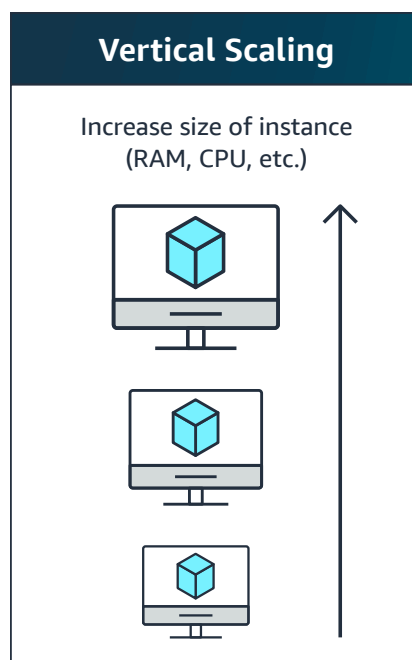
load and ensure high read performance. As replication lag is low, the read replicas are a near real-time representation of the data at the primary with little to no impact on your primary database. This approach also improves write performance as reads are offloaded from the primary.

Vertical scaling with a broad choice of instance types and sizes

With vertical scaling, the database's instance type and size can be changed to scale your database to the requirements of your target workload. You can vertically scale up AWS database instances with the click of a button. AWS provides a broad selection of instance types and sizes that are composed of varying combinations of CPU, memory, storage, and networking capacity, which allows you to scale your database to the requirement of your target workload. There is minimal downtime when you're scaling your instance up in a Multi-AZ environment because the standby database gets upgraded first, then a failover occurs to the newly sized database.

Each database engine offers different choices. You can access the pricing page for your database engine of interest from the database [category page](#) to find the available instance choices.

Selecting a serverless option is an attractive alternative for automated scaling operations (see [Automated capacity changes with serverless databases](#)). With AWS serverless databases and on-demand capacity scaling modes, your database scales automatically to match the workload.



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

In-memory caches and databases provide a good solution for use cases that require ultra-fast data access. AWS offers two options for in-memory data access: Amazon ElastiCache and MemoryDB. Both have Redis-compatible engines. In addition, ElastiCache has a Memcached engine.

For read performance, both ElastiCache and MemoryDB provide ultra-fast microsecond response time and throughput improvements over disk storage. ElastiCache recently achieved a notable [performance record](#)—the ability to 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, while MemoryDB is an in-memory database. Like other caches, ElastiCache is an ephemeral cache. 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 for Redis](#) is a better fit. MemoryDB is an in-memory database and offers both in-memory performance and durability. When data is written to MemoryDB for Redis, it also synchronously writes data to a durable Multi-AZ transaction log before it 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.

Meeting users where they are with global deployments

Globally distributed users expect local response time from their databases, rather than being subjected to network latency. Global data distribution can locate data near users, so that they experience local response time. There are two approaches to global data distribution: (1) maintaining local copies of data for read response time and (2) using multi-active data distribution for the response time of both reads and writes.

Global distribution of data also provides a mechanism for disaster recovery by failing over to a secondary region if the application cannot connect to its regional endpoint.

Multi-Region read replicas

Beyond Multi-AZ read replicas, the AWS operational databases that support globally distributed read replicas are [Amazon RDS](#), [Amazon Aurora](#), [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.

Multi-active global tables

Multi-active global tables are a technique for scaling both reads and writes across regions. Unlike sharding, multi-active global tables are copies of the entire dataset.

[DynamoDB global tables](#) use a multi-Region, multi-active approach—every region's replica is active. 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). This approach opens up the possibility of update conflicts, which are resolved by using the “last writer wins” technique.

Cross-region replication usually occurs within one second. It is easy to convert a table to a global table and add or remove regions from a global table. Data is secure because it is encrypted in transit and at rest.

Maintaining operational stability: availability, upgrades, updates, patches, and security

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. AZs are isolated from each other. Data redundancy is maintained through data replication across 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. Aurora 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.

Recovering from Region-wide outages

AWS databases can be hosted in multiple locations worldwide. These locations are composed of isolated AWS Regions. AWS Regions span multiple AZs. Databases can be set up to replicate data from the primary Region to secondary Regions.

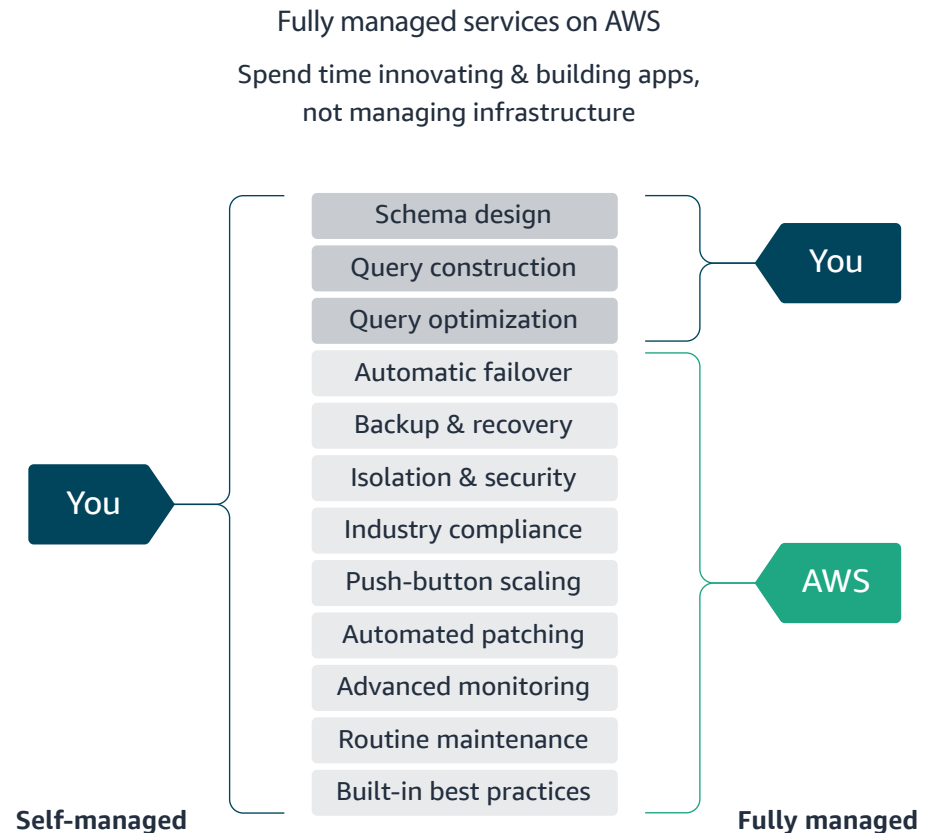
Amazon Aurora Global Database is designed for globally distributed applications, allowing a single Aurora database to span multiple AWS Regions. It replicates your data with no impact on database performance, enables fast local reads with low latency in each Region, and provides disaster recovery from Region-wide outages. If your primary Region suffers a performance degradation or outage, you can promote one of the secondary Regions to take read/write responsibilities. An Aurora cluster can recover in less than one minute, even in the event of a complete regional outage.

SECTION 6

With [Amazon DynamoDB global tables](#) and [Amazon Keyspaces Multi-Region Replication](#), data is replicated across Regions in a multi-active configuration that allows writers in multiple Regions. If an application is unable to connect to a regional database endpoint, the full dataset can be accessed from a secondary Region. Both DynamoDB global tables and Keyspaces Multi-Region Replication are designed for up to 99.999 percent availability.

Fully managed services

With fully managed databases, your operational burden is significantly reduced. Daily maintenance tasks can take up an inordinate amount of time on undifferentiated heavy lifting. AWS managed databases handle all of the fundamental operations—like provisioning, high availability and durability, patching, and upgrades—without downtime, setup, configuration, automated backups, and failover. 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.



SECTION 6

Blue/green deployments

Some changes, like major version upgrades and schema changes, are under your control and subject to your timing requirements. New approaches to making these changes follow a process that resembles a DevOps approach for managing the software development lifecycle (SDLC).

One such DevOps-inspired process is blue/green deployments. This approach can be applied to databases as part of making changes to production. Despite a series of promotion steps and tests, the performance of a database in a real-life production environment after a change is still somewhat unpredictable. 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 MySQL in 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 database can be updated 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](#). 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.

Maintaining the highest levels of security, governance, and compliance

AWS maintains the highest standards in security, governance, and compliance. Data in your databases can be encrypted using encryption keys you manage through AWS Key Management Service (AWS KMS). AWS databases also support secure connections via Transport Layer Security (TLS) protocol. AWS databases are integrated with Amazon CloudWatch for monitoring and logging, so that customers can granularly monitor their consumed capacity to better understand the cost of their fleet.

[AWS Identity Services](#) help you securely manage identities, resources, and permissions at scale. With AWS, you have identity services for your workforce and customer-facing applications to get started quickly and manage access to your workloads and applications. AWS detection and response services help you identify potential security misconfigurations, threats, or unexpected behaviors, so you can quickly respond to potentially unauthorized or malicious activity occurring within your environment.

[Amazon GuardDuty](#) is a threat detection service that continuously monitors your AWS accounts and workloads for malicious activity and delivers detailed security findings for visibility and remediation.

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

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. They touch small subsets of data, unlike analytical or search-oriented data stores that process much larger volumes of data for each query. 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 has to be updated and redeployed. Data pipelines that are built using ETL 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.

SECTION 8

AWS zero-ETL future

The complexities of ETL are why AWS is re-imagining the future with zero-ETL integrations. AWS has a growing portfolio of seamless connections between data services, reducing the dependency on ETL. Also, AWS has already built integrations between our most popular data stores:

- Amazon Aurora MySQL zero-ETL integration with Amazon Redshift
- Amazon Aurora PostgreSQL zero-ETL integration with Amazon Redshift
- Amazon RDS for MySQL zero-ETL integration with Amazon Redshift
- Amazon DynamoDB zero-ETL integration with Amazon Redshift
- Amazon DynamoDB zero-ETL integration with Amazon OpenSearch Service

Zero-ETL integrations open up near real-time analytics and search use cases on petabytes of operational data without maintaining 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 also access the machine learning 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.

Unlocking vector search for generative AI applications

Generative AI holds the promise of ushering in a new wave of innovation. Generative AI applications rely on foundation models (FMs), including large language models (LLMs). These models are trained on vast datasets, such as all of the content accessible on the internet, and serve as foundational models for various use cases.

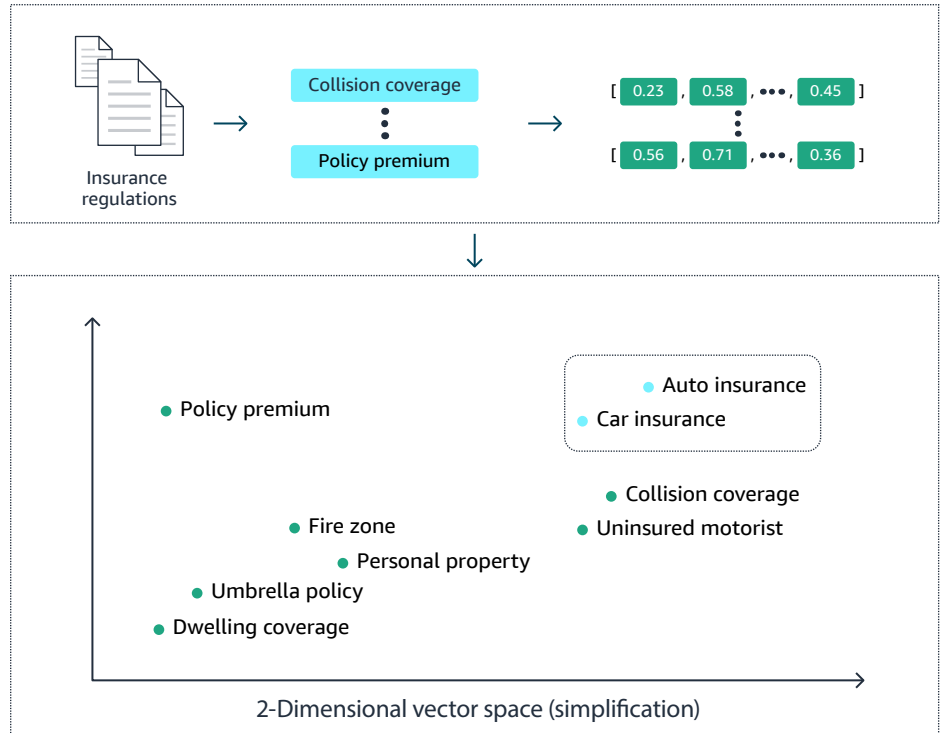
While a generative AI application relying purely on an FM has access to broad real-world knowledge, it needs to be augmented with domain-specific data to produce accurate and relevant results. These requests often contain inaccuracies, referred to as hallucinations, which occur more frequently the more domain-specific the interaction is. Customizing generative AI applications for domain-specific accuracy is crucial, often requiring additional context.

Adding domain specificity using databases with vector capabilities

One approach to adding domain specificity to generative AI applications relies on encoding domain-specific data into n-dimensional vectors. A vector, which is an array of numbers, symbolizes data in a mathematical form. Each dimension in the array corresponds to a specific feature or attribute of the data. Vectors are derived through embedding models applied to raw data. Vectors serve as a structured data format that is easily readable by machine learning models such as FMs, enabling you to use diverse data types—such as text, images, videos, and audio—as part of your generative AI applications.

Vectors facilitate the representation of complex relationships between data elements in machine learning. They allow your models to more easily find relationships between similar words or phrases. For instance, “cat” is closer to “kitten,” whereas “dog” is closer to “puppy.” This means your FMs can produce more relevant responses to prompts from your end users. Such semantically meaningful vectors are termed embeddings, where words, phrases, sentences, images, audio, video, or other data elements are mapped to vectors, allowing for enhanced contextual analysis. Similarity searches involve the process of grouping these embeddings to find similar items. For instance, ecommerce platforms transform product images into vectors that embed context, enabling accurate recommendations based on customer preferences. Vectors enable efficient nearest-neighbor searches in multi-dimensional spaces. Powered by algorithms like k-nearest neighbor (k-NN), Hierarchical Navigable Small World (HNSW), and Inverted File Index (IVF), AWS databases that support vectors as a data type offer rapid lookup capabilities, along with core database features like scalability, availability, and security.

SECTION 9



Many use cases, including natural language processing and recommendation systems, rely upon 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. Vector search can also add more current information that is absent from LLMs, giving LLMs an external memory.

Retrieval Augmented Generation to enhance contextual relevance

Retrieval Augmented Generation (RAG) is a technique for building context around a query based on vector similarity searches. The vector similarity searches find vectors that represent related and up-to-date information from your private data sources and augment the user's prompt with this information as added context. RAG involves the conversion of the user's prompt into a vector, which can then be used to search for contextually similar vectors in a database with vector capabilities. The outcome of these searches typically yields a ranked list of vectors with the highest similarity scores to the vector that represents the prompt. Subsequently, the original source or index linked to each vector can also be retrieved. The retrieved vectors enrich the user's prompt, grounding the LLM

SECTION 9

in the specific domain context and increasing the accuracy and contextual relevance of the LLM's output based on the desired context. By augmenting prompts with contextual data, LLMs can respond more accurately, reducing the likelihood of generating irrelevant or inaccurate content.

Knowledge Bases for Amazon Bedrock

[Amazon Bedrock](#) is a fully managed service that simplifies the process of building and scaling generative AI applications with FMs. Knowledge Bases for Amazon Bedrock 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).

The Knowledge Bases for Amazon Bedrock 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 Amazon Aurora or Amazon OpenSearch Service 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.

Storing vectors and operational data together

Vectors, as a data type, can seamlessly integrate into the databases that customers currently use. This eliminates the need to migrate data to a specialized vector database while still harnessing vector capabilities within existing architectures. AWS databases with vector capabilities allow the storage of vast amounts of vector embeddings, perform similarity searches without data movement, and use RAG use cases. This integration not only preserves your data but also grants you access to 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, leading to significant changes in your existing applications. Moving to a new data source and managing it on your own can be cumbersome. 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 enablement of familiar operational databases benefits operations teams significantly. Storing vectors alongside your application data eliminates the need for data processing pipelines or complex application logic to combine data across different systems, and reduces overhead and licensing costs.

SECTION 9

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 for Redis
- Amazon DocumentDB
- Amazon OpenSearch Service

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.

Bridging Kubernetes workloads to AWS databases

As AWS managed services, AWS databases mirror Kubernetes (K8s) by providing a similarly fluid ability to scale their workload. But how do the worlds of K8s and fully managed 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 API 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.

SECTION 10

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 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 a GitOps based approach to 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 Amazon RDS, Amazon Aurora, Amazon ElastiCache, Amazon MemoryDB, Amazon DynamoDB, Amazon Keyspaces, and [more](#).

Migrating your data to AWS

Migrating databases is a delicate process. Organizations need a practical approach to migrating their 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 other important evolutionary paths.

AWS offers tools and experts to help assess, plan, and build the right migration path for your company. [AWS Database Migration Service](#) (AWS DMS), which includes [integrated schema conversion](#), helps users migrate databases to AWS while maintaining uninterrupted operations on the source database. AWS 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.

SAMSUNG

Case study: Samsung Electronics

In less than 18 months, [Samsung Electronics migrated its global Samsung Account data to Amazon Aurora](#). “AWS had lots of tools and services to help the migration—AWS DMS is one example,” says Salva Jung, Principal Architect and Engineering Manager, Samsung. Samsung Electronics used AWS Database Migration Service (AWS DMS) to initiate the transition of the data and ensure the source database remained operational so that end users could still access their Samsung Accounts as usual. In addition, AWS DMS replicated the large-scale heterogeneous database, duplicated 2 or 3 TB of user data in 3–4 days, and routed user traffic one by one from the IDC to the cloud. Samsung Electronics completed the migration with minimal downtime. “We had some downtime but not much,” says Jung. “The important thing is that we detected problems quickly and minimized the user impact.”

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.

CONCLUSION

Modern 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 broadest choice. 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. Because 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 the least of your concerns.

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.

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.