AWS Whitepaper

Choosing an AWS NoSQL Database

Choosing an AWS NoSQL Database: AWS Whitepaper

Table of Contents

Choosing an AWS NoSQL Database

Publication date: **April 25, 2023** (*Document revisions*)

Over the past decade, modern applications have started using NoSQL databases to manage the three Vs commonly associated with Big Data applications:

- **Volume** — Amount of data
- **Velocity** — Speed with which data is generated
- **Variety** — Structured and unstructured data generated by humans or machines

NoSQL does not describe one single technology. it is a variety of non-relational technologies and architectures aiming to address the challenges of new use cases. Some of the popular types of NoSQL databases include key-value, document, graph, and wide-column. These databases are becoming more popular as organizations create larger volumes and a greater variety of unstructured data.

NoSQL databases help developers manage the challenge of ever-expanding diversity of data types and data models, and are highly effective at handling unpredictable data, often with faster query speeds. Different NoSQL database options are available, and finding the right NoSQL database remains challenging because there are many options with varying architectures.

To help navigate the selection process, this whitepaper provides guidance on selecting the right NoSQL database for specific use cases. It is intended for cloud architects, database architects and application developers building modern applications on AWS and considering using a NoSQL database.

Introduction

For decades, the predominant data model used for application development was relational databases. The relational model has been the mainstay of data storage systems since its inception by Edgar F. Codd in 1969. Commercially, vendors such as IBM, Oracle, and Microsoft have had tremendous success with their database products, which are all based on the relational model.

The amount of data that exists today is growing at an unprecedented rate, and this has put an increasing amount of pressure on the database systems that are tasked with storing and managing this data. Modern business systems manage increasingly large volumes of heterogeneous data. This

is driven in part by the adoption of microservices, the Internet of Things (IoT), the amount and size of the data being captured, and the requirements for many types of specialized analytics.

To address these challenges, NoSQL databases have gained increased acceptance when developing modern business systems. NoSQL databases are purpose-built for specific data models, and allow for flexible schemas to meet these modern storage and heterogeneous data requirements.

At AWS, we recognize that different NoSQL database technologies solve different problems. Consider this AWS Bookstore Demo Application as an example. It contains multiple experiences such a shopping cart, product search, recommendations, and a "top sellers" list.

For each of these use cases, the app makes use of a purpose-built database, so the developer never has to compromise on functionality, performance, or scale. However, selecting the right NoSQL database can be a complex and challenging process due to the different types of NoSQL databases available, each with its own strengths and weaknesses.

To make an informed decision, it is important to consider several key factors, including the data model, scalability, consistency, availability, and durability. This whitepaper provides guidance on selecting the right NoSQL database for specific use cases by offering a detailed comparison of the five most common AWS noSQL databases and their features, insights into the strengths and limitations of different databases, understand their use cases, and use the decision framework to make an informed decision on which database to choose based on your specific needs.

Are you Well-Architected?

The AWS Well-Architected Framework helps you understand the pros and cons of the decisions you make when building systems in the cloud. The six pillars of the Framework allow you to learn architectural best practices for designing and operating reliable, secure, efficient, cost-effective, and sustainable systems.

Using the AWS Well-Architected Tool, available at no charge in the AWS Management Console, you can review your workloads against these best practices by answering a set of questions for each pillar.

For more expert guidance and best practices for your cloud architecture—reference architecture deployments, diagrams, and whitepapers—refer to the AWS Architecture Center.

Types of NoSQL databases

To support specific needs and use cases, NoSQL databases use a variety of data models for managing and accessing the data. The following section describes some of the common NoSQL database categories:

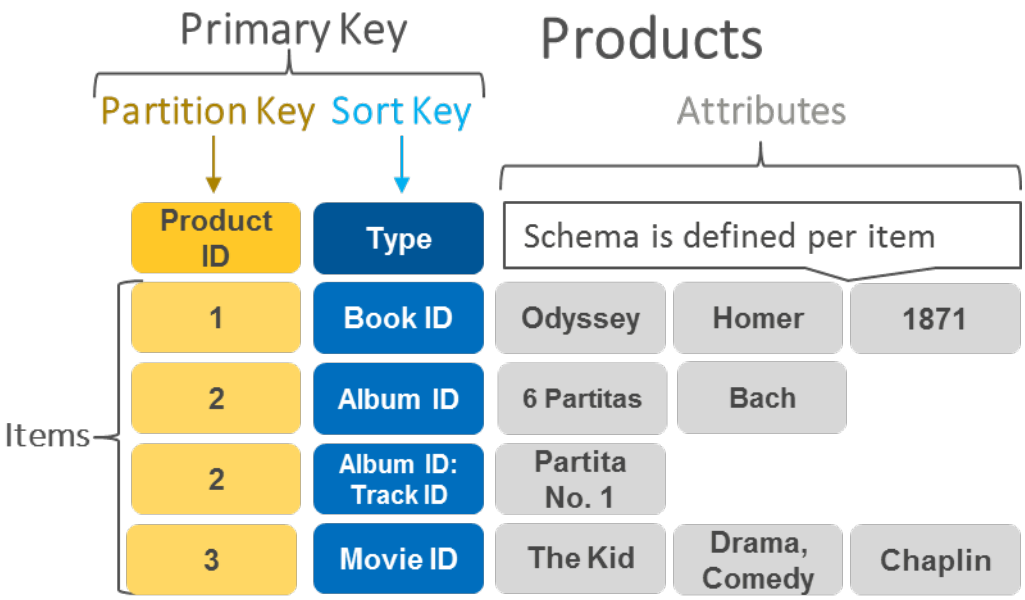
- Key-value pair
- Document-oriented
- Column-oriented
- Graph-based
- Time series

These types of databases are optimized specifically for applications that need large data volumes, flexible data models, and low latency. To achieve these objectives, NoSQL databases employ various techniques, and it's important to note that not all database options prioritize the same set of factors that are mentioned here:

- Consistency
- Atomicity, Consistency, Isolation, and Durability (ACID) transactions
- Query language and data access richness (simplified Create, Read, Update, and Delete (CRUD)-style operations with known predictable cost)
- Sharding/partitioning of data sets on primary identifier keys
- Shifting the burden of data and schema validation to the application (removing referential integrity enforcement by the database and so on)

Here's a brief overview of most popular NoSQL data models.

1. **Key-value** — A key-value data store is a type of database that stores data as a collection of key-value pairs. In this type of data store, each data item is identified by a unique key, and the value associated with that key can be anything, such as a string, number, object, or even another data structure.



An example of data stored as key-value pairs.

AWS offers Amazon DynamoDB as a key-value managed database service.

2. **Document** — In a document database, the data is stored in documents. Each document is typically a nested structure of keys and values. The values can be atomic data types, or complex elements such as lists, arrays, nested objects, or child collections (for example, a collection in the document database is analogous to a table in a relational database, except there is no single schema enforced upon all documents).

   Documents are retrieved by unique keys. It may also be possible to retrieve only parts of a document--for example, the cost of an item--to run queries such as aggregation, querying using examples based on a text string, or even full-text search. Most document databases also allow you to define secondary indexes.

   You can transfer the application code object model directly into a document using several different formats. The most commonly used are JavaScript Object Notation (JSON), Binary JavaScript Object Notation (BSON), and Extensible Markup Language (XML).

Key	Document
101	{ "ID": "1001", "ItemsOrdered":[{ "ItemID": "1", "Quantity": "2", "cost": "1000", }, { "ItemID": "1001", "Quantity": "2", "cost": "1000", }], "OrderDate": "05/11/2019" }
102	"ID": "1002", "ItemsOrdered":[{ "ItemID": "2890", "Quantity": "11", "cost": "10000", }] "OrderDate": "05/11/2019" }

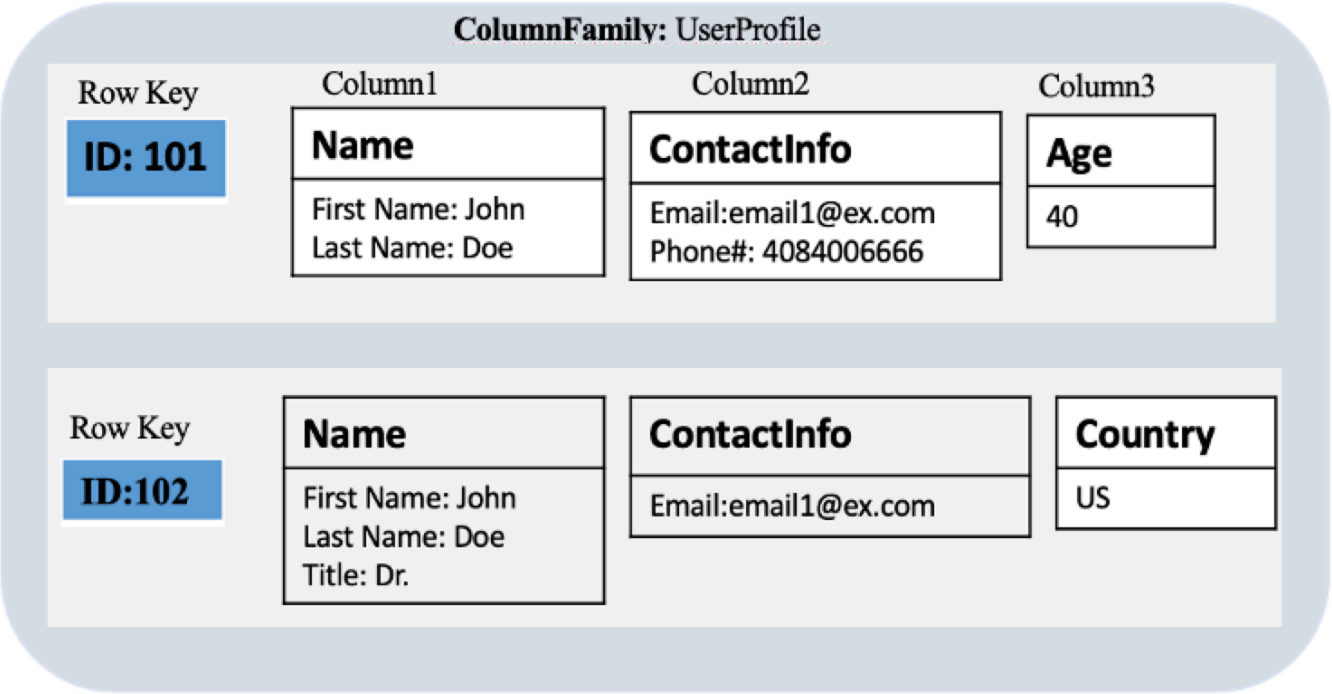
An example of a document data model

AWS offers a specialized document database service called Amazon DocumentDB (with MongoDB compatibility).

3. **Wide-column** — A wide column data store is a type of NoSQL database that stores data in columns rather than rows, making it highly scalable and flexible. In a wide column data store, data is organized into column families, which are groups of columns that share the same attributes. Each row in a wide column data store is identified by a unique row key, and the columns in that row are further divided into column names and values.

   Unlike traditional relational databases, which have a fixed number of columns and data types, wide column data stores allow for a variable number of columns and support multiple data types. The most significant benefit of having column-oriented databases is that you can store

large amounts of data within a single column. This feature allows you to reduce disk resources and the time it takes to retrieve information from it.

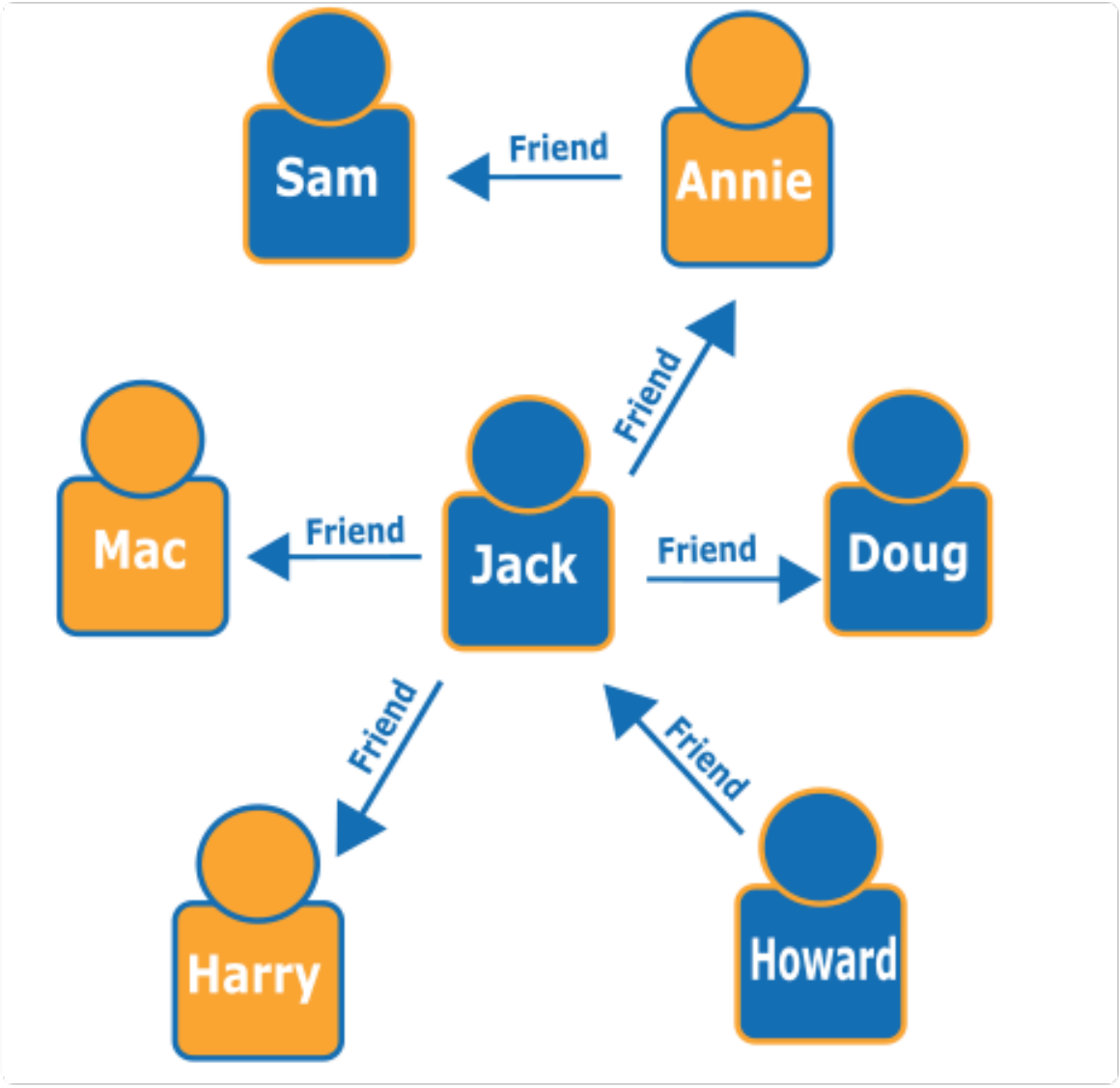


An example of the kind of data you might store in a wide-column data store

AWS offers Amazon Keyspaces (for Apache Cassandra) as a wide-column managed database service.

4. **Graph** — Graph databases are used to store and query highly connected data. Data can be modeled in the form of entities (also referred to as nodes, or vertices) and the relationships between those entities (also referred to as edges). The strength or nature of the relationships also carry significant meaning in graph databases.

   Users can then traverse the graph structure by starting at a defined set of nodes or edges and travel across the graph, along defined relationship types or strengths, until they reach some defined condition. Results can be returned in the form of literals, lists, maps, or graph traversal paths. Graph databases provide a set of query languages that contain syntax designed for traversing a graph structure, or matching a certain structural pattern.

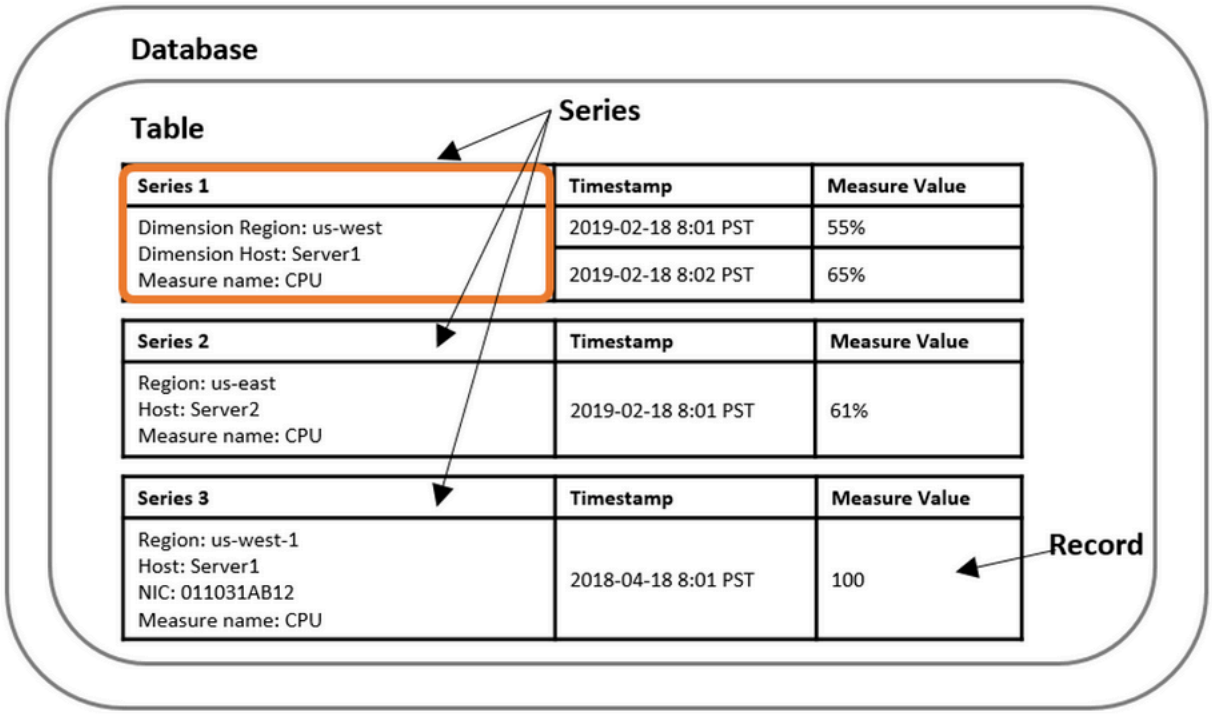


An example of a social network graph. Given the people (nodes) and their relationships (edges), you can find out who the "friends of friends" of a particular person are—for example, the friends of Howard's friends.

AWS offers Amazon Neptune as a managed graph database service.

5. **Time series** — A time series database is designed to store and retrieve data records that are sequenced by time, which are sets of data points that are associated with timestamps and stored in time sequence order. Time series databases make it easy to measure how measurements or

events change over time; for example, temperature readings from weather sensors or intraday stock prices.



An example of a series data model

AWS offers Amazon Timestream as a managed time series database service.

Understanding Amazon NoSQL data stores

AWS provides the broadest selection of managed NoSQL databases, allowing you to save, grow, and innovate faster. With Amazon NoSQL databases, you get high performance, enterprise-grade security, automatic, and instant scalability. The following table lists some of the AWS managed NoSQL database services offered, and their key characteristics:

Table 1 — AWS database service comparison

AWS database service	Use cases	Strengths	Security
Amazon DocumentDB (with MongoDB compatibility)*	User profile/personalization, catalogs, mobile, and content managemen	Flexible schema and indexing, ad hoc queries on any attributes,	Capability to ena encryption at res transit

AWS database service	Use cases	Strengths	Security
• Mid TB range • Data format: JSCO, BSON, XML • NoSQL type: document • Consistency: strong/ev entual	t, retail and marketing (for example, tracking customers who purchase similar items)	including nested attribute s	
Amazon DynamoDB* • High TB range • Data format: JSON, BSON, or XML • NoSQL type: key-value , document • Consistency: strong/ev entual	• User preferences • Session management • Shopping cart • Product catalog • High-traffic web apps • Near real-time bidding	• Performance at scale • Serverless • Simple data model	Encrypts all data default, row/colu security
Amazon Keyspaces (for Apache Cassandra) • Data format: JSON • NoSQL type: wide column • Consistency: one, local_one, local-quo rum	High scalable apps for: • Equipment maintenance • Fleet management • Route optimization	• Extreme write speeds with relatively less velocity reads • Being serverless, allocates storage and read/write throughput directly to tables	• Tables are encr default • Capability to e encryption at r transit

AWS database service	Use cases	Strengths	Security
Amazon Neptune* • Mid TB range • Data format: Germalin, RDF, open Cypher • NoSQL type: graph • Consistency: immediate consistency	• Recommendations • Social patterns • relationship traversal • Fraud detection • Risk assessment	• Highly connected data is locally indexed and purpose-built to answer questions about relationships • Optimized for efficient storage and retrieval	Capability to ena encryption at res transit
Amazon Timestream* • NoSQL type: TimeSerie s • Consistency: eventual	• Server metrics • Application performance monitoring • Network data • IoT apps • Sensor data • Events • Clicks • Financial forecasting • Many other types of analytics data	Analytics over time series data	Encrypts all data default
Amazon ElastiCache for Memcached • Low TB range • NoSQL type: in-memory, key-value	• Caching repeat requests • Sticky sessions (to store session state)	• Simple caching model • Multi-threaded performance	Capability to ena encryption at res transit

AWS database service	Use cases	Strengths	Security
Amazon ElastiCache for Redis • Low TB range • NoSQL type: in-memory, key-value	• Gaming leaderboards • Geospatial applications	• Complex data structures • Sorting and ranking • Pub/sub messaging • Geospatial capabilities	Capability to ena encryption at res transit
Amazon MemoryDB for Redis • NoSQL type: in-memory, database • Consistency: strong/eventual	• High concurrency • Streaming media • Data feeds	• Durable database • Complex data structures	Capability to ena encryption at res transit

* ACID compliant

NoSQL databases

- Amazon DynamoDB
- Amazon Keyspaces (for Apache Cassandra)
- Amazon Neptune
- Amazon Timestream
- Amazon ElastiCache
- Amazon DocumentDB (with MongoDB compatibility)
- Amazon MemoryDB for Redis

Amazon DynamoDB

Amazon DynamoDB is a fully managed NoSQL database service. Some key capabilities of DynamoDB include:

- **High performance** — Designed to provide single-digit millisecond latency for read and write operations at any scale.

- **Scalability** — The ability to automatically scale throughput capacity in response to demand, so you can start small and scale as needed.
- **Flexibility** — Supports both document and key-value data models, and provides rich data types such as lists and maps, making it easy to store any type of data.
- **Durability** — Provides consistent, single-digit millisecond performance for read and write operations, even in the face of hardware failures.
- **Global tables** — DynamoDB global tables provides multi-Region data replication which makes it easy to build globally distributed applications.
- **Integrations** — Integration with other AWS services such as AWS Lambda, Amazon Simple Storage Service (Amazon S3), Amazon DynamoDB Streams, and Amazon Kinesis, making it easy to build serverless and data-driven applications. Amazon DynamoDB integrates with Amazon CloudWatch Contributor Insights to provide information about the most accessed and throttled items in a table or index. DynamoDB delivers this information to you using CloudWatch Contributor Insights rules, reports, and graphs of report data.

Amazon Keyspaces (for Apache Cassandra)

Amazon Keyspaces (for Apache Cassandra) (Amazon Keyspaces) is a fully managed, Apache Cassandra-compatible database service. Some key features of Amazon Keyspaces include:

- **Apache Cassandra compatibility** -— Full compatibility with Cassandra, allowing you to use your existing Cassandra applications and tools with minimal changes.
- **Scalability** — Designed to handle millions of requests per second and terabytes of data, making it suitable for high-scale applications.
- **Serverless** — Instead of deploying, managing, and maintaining storage and compute resources for your workload through nodes in a cluster, Amazon Keyspaces allocates storage and read/write throughput resources directly to tables.
- **Global distribution** — Supports global distribution of data, allowing you to store and access data from multiple Regions, reducing latency and improving application performance.
- **Monitoring and management** — Provides an easy-to-use, web-based console for monitoring and managing your database, as well as integration with Amazon CloudWatch for metrics and alerts.
- **Integration with other AWS services** — Integration with other AWS services such as Amazon S3, Amazon Redshift, and Amazon EMR, making it easy to build data-driven applications.

- **Highly available and secure** — Data is replicated automatically across multiple AWS Availability Zones using a replication factor of three. Amazon Keyspaces encrypts all customer data at rest by default, and is integrated with AWS IAM to help you manage access to your tables and data.

Amazon Neptune

Amazon Neptune is a fully managed graph database service. Neptune makes it easy to build and run applications that work with highly connected datasets, including for ID, graph/C360, security, fraud, and knowledge graph applications. Some key features of Amazon Neptune include:

- **High performance** — Provides low-latency and high-throughput performance for both read and write operations, making it suitable for real-time applications.
- **Scalability** — Neptune can handle billions of vertices and edges, and is designed to automatically scale to meet the demands of your application.
- **Compatibility** — Supports the popular graph query languages, including Apache TinkerPop Gremlin and SPARQL, making it easy to use with existing applications and tools.
- **Durability** — Automatically replicates data across multiple Availability Zones (AZs) for high availability (HA) and data durability.
- **Integration with other AWS services** — Integration with other AWS services such as Amazon S3, Amazon OpenSearch Service, and Amazon SageMaker, making it easy to build data-driven applications.
- **Management and monitoring** — Provides an easy-to-use, web-based console for monitoring and managing your database, as well as integration with Amazon CloudWatch for metrics and alerts.

Amazon Timestream

Amazon Timestream is a managed time series database. It is specifically designed to handle time-stamped data, such as IoT device data and operational logs, and provides a fast, scalable and cost-effective way to store and analyze large amounts of time series data.

Some of the key features of Amazon Timestream include:

- **Scalable storage** — Automatically scales storage as your data grows, so you don't have to worry about running out of space.

- **Fast querying** — Provides fast and efficient querying of your time series data, allowing you to quickly and easily analyze your data.
- **Integrations** — Integration with other AWS services, such as Amazon Kinesis, Amazon S3, and Amazon QuickSight, making it easy to collect, store and analyze your data.
- **Cost-effective** — Provides cost-effective storage and analysis of your time series data, with the ability to choose between standard and memory-optimized performance tiers.

Amazon ElastiCache

Amazon ElastiCache is a web service that makes it easy to deploy, operate, and scale an in-memory cache in the cloud. It provides a simple way to cache frequently-used data in memory, reducing the need to repeatedly fetch this data from a slower disk-based data store such as a relational database.

ElastiCache supports two popular in-memory cache engines: Memcached and Redis. These engines can be used to significantly improve the performance of web applications, mobile apps, gaming platforms

Some of the key features of Amazon ElastiCache include:

- **Easy setup** —Providing a simple, one-click setup process, making it easy to get started with in-memory caching.
- **Scalable performance** —Automatically scales cache nodes as your application's needs change, so you can easily adjust cache performance to meet the demands of your application.
- **High availability** —Provides built-in replication and failover capabilities, ensuring high availability and durability of your cache data.
- **Integrations** —integration with other AWS services, such as Amazon Elastic Compute Cloud (Amazon EC2), Amazon Relational Database Service (Amazon RDS), and Amazon S3, making it easy to use caching in your overall application architecture.

Amazon DocumentDB (with MongoDB compatibility)

Amazon DocumentDB (with MongoDB compatibility) (Amazon DocumentDB) is a fully managed, scalable, and highly available document database service. It is designed to be compatible with the MongoDB API, allowing you to use existing MongoDB applications and tools with Amazon DocumentDB.

DocumentDB supports a range of use cases, such as content management systems, e-commerce applications, and mobile backends. It allows you to store and retrieve JSON-like documents, with support for complex queries, indexing, and aggregation. It also supports transactions, allowing you to group multiple write operations into a single atomic unit of work.

Some of the key features of Amazon DocumentDB include:

- **Compatibility** — DocumentDB is compatible with existing MongoDB drivers and tools, and applications can be used with DocumentDB with little or no change.
- **Scalability** — DocumentDB Elastic Clusters scale within minutes to handle millions of reads and writes with petabytes of storage capacity, helping you cost-effectively meet the needs of your most demanding document workloads.
- **Flexibility** — Supports a flexible data model that allows you to store and retrieve JSON-like documents with varying structure and complexity. This makes it well-suited for a wide range of use cases, such as content management, user profiles, product catalogs, and more.
- **Durability** — Designed to provide high durability and availability for your data. It provides automatic backup and recovery, point-in-time recovery, and data replication across multiple Availability Zones for high availability and disaster recovery (DR). DocumentDB automatically backs up your data and transaction logs to Amazon S3, which is designed for 99.999999999% durability. This helps ensure that your data is protected against data loss or corruption, even in the event of a disaster or outage.
- **Global clusters** — DocumentDB global clusters provides disaster recovery from Region-wide outages and enables low-latency global reads.
- **Integration with other AWS services** — You can integrate with AWS Glue to import and export data from and to DocumentDB to other AWS services such as Amazon S3, Amazon Redshift, and Amazon OpenSearch Service.

Amazon MemoryDB for Redis

Amazon MemoryDB for Redis (MemoryDB) is a fully managed, in-memory database service. It is designed to provide high performance and low latency for applications that require fast and frequent access to data.

Some of the key features of MemoryDB include:

- **Compatibility** — Compatibility with Redis, an open-source, in-memory data store that is widely used for caching, near real-time analytics, and other high-performance applications. MemoryDB

supports the same set of Redis data types and parameters, and requires no code change to migrate from Redis.

- **Scalability** — Designed to be highly available and durable, with automatic failover and data replication across multiple Availability Zones for high availability and disaster recovery.
- **Data durability** — Data is stored across multiple Availability Zones, while ensuring single-digit millisecond response using AWS proprietary architecture design.
- **Support for security features** such as Amazon Virtual Private Cloud (Amazon VPC), encryption with AWS Key Management Service (AWS KMS), and authentication and authorization with Redis ACLs.
- **Flexibility** — Provides a number of features and capabilities to help you optimize your application's performance, including read and write replicas, Multi-AZ deployments, automatic scaling, and flexible pricing options based on usage. MemoryDB is well-suited for a wide range of use cases, including caching, near real-time analytics, and session stores. It is particularly useful for applications that require fast and frequent access to data, such as gaming, e-commerce, and advertising.

Amazon NoSQL databases integrate with AWS Identity and Access Management (AWS IAM) for access control and security. IAM allows you to manage access to your NoSQL databases by creating policies that define permissions for specific users, groups, or roles. You can use IAM to control access to specific tables or resources within your NoSQL databases, as well as to enforce fine-grained permissions for read and write operations.

Decision making

This section outlines a decision framework you can use to help you determine which AWS-managed NoSQL database service, or combination of database services would fit your workload needs best.

While there is no simple formula you can follow that is comprehensive enough to apply generally, there are a few important questions related to your application that should be answered during the selection process:

What type of data is your application planning to persist (such as JSON structures, telemetry data, image files, geospatial data)?

Different databases allow you to access stored data differently. If you plan to store unstructured data such as images or encoded payloads, you need a data store that can store and retrieve the entire unstructured binary payload fast, but may need a rich set of data access features to introspect the unstructured data.

Conversely, a catalog system needs a richer feature set to access data based on patterns, but also allow for flexibility to expand the set of attributes collected for each item in the catalog. These capabilities may be more important than the absolute fastest way to retrieve data access.

What performance requirements and service-level commitments have you made to your end users (for example, a service level agreement that guarantees microsecond or millisecond-level response latency for queries)?

If your workload requires extremely high read performance with a response time measured in microseconds rather than single-digit milliseconds, then you may consider using in-memory caching solutions alongside your database, or a database that supports in-memory data access.

Also consider how predictable your performance needs to be. A database such as Amazon DynamoDB can deliver consistent, predictable response latencies to reads and writes, but it does so because it supports a small number of query patterns that have a known cost. It's a great fit for point queries accessing one or a very small number of records at high frequency.

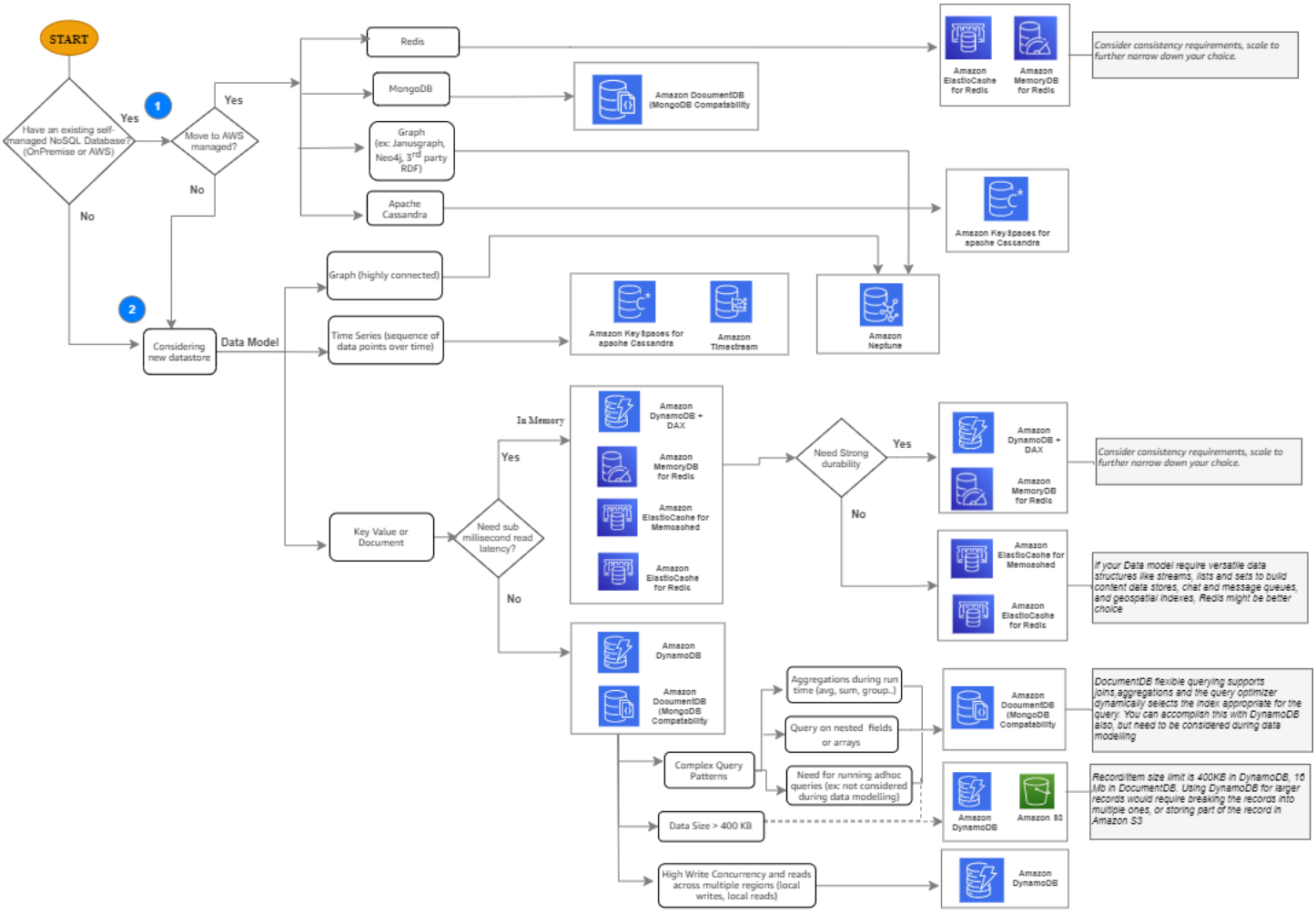
If your data access patterns require accessing a variable or unpredictable number of records or volume of data, your performance will also have more variability. Consider also that modern architectures are implemented using decoupled microservices, each with different date access requirements, compounding the end-to-end latency or performance of the end user request.

What are your resiliency requirements?

Workloads with high availability requirements (such as mission-critical applications that can't tolerate any downtime) can span multiple Regions to provide further resiliency in case a specific AWS region becomes unreachable. For example, you can use DynamoDB global tables to deploy globally across supported Regions and read or write to the local copies of the tables concurrently. Amazon DocumentDB also supports multiple Regions through Amazon DocumentDB global clusters, but you can only write in the primary Region cluster.

After you address these questions, you can use the following decision tree for further direction on how to narrow down your choices. The decision tree covers two scenarios:

1. If you're already using a NoSQL database on premises and would like to consider migrating to a fully managed scalable, highly available AWS NoSQL database service, start your review of our decision tree at Step 1.
2. If you want to modernize your application and are considering a NoSQL database, you can use the decision tree to choose the most appropriate AWS-managed NoSQL database service for your use case based on your requirements by starting at Step 2, You can start eith the data model that is appropriate for your use case.


START
Have an existing self-managed NoSQL Database? (OnPremise or AWS)
Yes
1
Move to AWS managed?
Yes
No
No
2
Considering new datastore
Data Model
Redis
MongoDB
Graph (ex: Janusgraph, Neo4j, 3rd party RDF)
Apache Cassandra
Graph (highly connected)
Time Series (sequence of data points over time)
Key Value or Document
Amazon ElastiCache for Redis
Amazon MemoryDB for Redis
Consider consistency requirements, scale to further narrow down your choice.
Amazon DocumentDB (MongoDB Compatability)
Amazon Keyspaces for apache Cassandra
Amazon Neptune
Amazon Timestream
In Memory
Yes
Need sub millisecond read latency?
No
Amazon DynamoDB + DAX
Amazon MemoryDB for Redis
Amazon ElastiCache for Memcached
Amazon ElastiCache for Redis
Need Strong durability
Yes
No
Amazon DynamoDB + DAX
Amazon MemoryDB for Redis
Consider consistency requirements, scale to further narrow down your choice.
Amazon ElastiCache for Memcached
Amazon ElastiCache for Redis
If your Data model require versatile data structures like streams, lists and sets to build content data stores, chat and message queues, and geospatial indexes, Redis might be better choice
Amazon DynamoDB
Amazon DocumentDB (MongoDB Compatability)
Aggregations during run time (avg, sum, group..)
Query on nested fields or arrays
Complex Query Patterns
Need for running adhoc queries (ex: not considered during data modelling)
Data Size > 400 KB
Amazon DocumentDB (MongoDB Compatability)
DocumentDB flexible querying supports joins,aggregations and the query optimizer dynamically selects the index appropriate for the query. You can accomplish this with DynamoDB also, but need to be considered during data modelling
Amazon DynamoDB
Amazon S3
Record/Item size limit is 400KB in DynamoDB, 16 Mb in DocumentDB. Using DynamoDB for larger records would require breaking the records into multiple ones, or storing part of the record in Amazon S3
High Write Concurrency and reads across multiple regions (local writes, local reads)
Amazon DynamoDB

Considerations

Best practices and limitations

- For implementation best practices and limitations, refer to the AWS documentation for the respective database service.

- If your use cases use relatively static schemas, perform complex table lookups, require accessing data across multiple keys and might experience high service throughputs it might be a better fit for Amazon RDS offerings.

Some customer journeys and lessons learned

- Amazon DynamoDB on Production: FinBox's Compilation of Lessons Learned in a Year
- Refer to this case study to learn how McAfee's use case of migrating from Microsoft SQL server to DynamoDB to power their next generation messaging platform to drive ecommerce business
- Watch this video on why Amazon Fulfillment choose Amazon DocumentDB to power their inventory authority platform (IAP), considerations for performance and scale, and some learnings from their experience
- Watch this video to hear about FINRA's story on how they modernized their data collection platform used by FINRA customers from a relational database using XML to Amazon DocumentDB.
- Idea to product: PricewaterhouseCoopers launches Check-In within three months on Amazon Keyspaces
- Watch this video to learn about the key features of ElastiCache for Redis and dive deep into how Groupon uses ElastiCache for deal curation.
- Blog post on how Near was able to reduce latency by four times and achieve 99.9% uptime of its critical RTB platform applications by moving to ElastiCache.
- LexisNexis presentation on using graph to store relationships between legal documents using Amazon Neptune.
- Cox Automotive presentation on using graph to store relationships between user identities on their web platforms to power marketing and advertising.

References

- Scale and performance characteristics of Timestream – Deriving near real-time insights over petabytes of time series data with Amazon Timestream.
- This blog post provides you with a quick summary and set of resources for common topics so you can quickly ramp up on Amazon DocumentDB.
- This blog post provides improved performance characteristics of Amazon Keyspaces, lightweight transactions API, advanced design patterns, and operational best practices.
- AWS Online Tech Talks: ElasticCache best practices
- Getting started with Amazon Neptune by creating a graph of all of your AWS resources.
- How to migrate an application from using GridFS to using Amazon S3 and Amazon DocumentDB.
- Graph data model lets you traverse through relationships without requiring joins and indexes. For more information, refer to the "How Do I Know I Need an Amazon Neptune Graph Database?" video.
- Graph data model lets you traverse through relationships without requiring joins and indexes. For more information, refer to "How Do I Know I Need an Amazon Neptune Graph Database?".
- Complex data models (such as arrays, nested fields, and deep relationships) let you consider a wider range of application needs. For more information, refer to the "When to use DocumentDB vs DynamoDB" video.
- DynamoDB provides extreme scale for certain data access patterns. For more information, refer to "How to determine if Amazon DynamoDB is appropriate for your needs".
- Refer to this tech talk to learn about DocumentDB use cases, and how Amazon DocumentDB cluster architecture provides better performance, scalability, and availability.
- Amazon MemoryDB for Redis is a durable, in-memory database for workloads that require an ultra-fast Redis-compatible primary database. If you require sub-millisecond performance and need to add persistence and durability, consider using MemoryDB rather than in-memory cache for Redis. Refer to this tech talk to learn about Amazon MemoryDB for Redis.

Developer references

- Why purpose-built database? This hands-on tutorial will help you get an idea of how AWS NoSQL databases can help run your specific workloads.

Training and guidance

- To ensure that development teams were comfortable with transitioning to Amazon, it essential to train the teams on AWS NoSQL databases and cloud-based design patterns (tech talks, workshops, and Immersion Days.)

Conclusion

NoSQL databases have become increasingly popular over the years due to their scalability, flexibility, and ability to handle large volumes of complex data. This whitepaper has provided an overview of the different types of NoSQL databases, including document-based, key-value, column-family, and graph databases, as well as their unique strengths and weaknesses.

It has also explored the various NoSQL database services offered by Amazon Web Services, including Amazon DynamoDB, Amazon Keyspaces, Amazon Neptune, Amazon DocumentDB, and Amazon MemoryDB for Redis. We hope it helps you make an informed decision on which database to choose based on your specific needs.

Contributors

Contributors to this document include:

- Ashish Bhatia, Senior Solution Architect, Amazon Web Services
- Malathi Pinnamaneni, Senior Solution Architect, Amazon Web Services

Document revisions

To be notified about updates to this whitepaper, subscribe to the RSS feed.

Change	Description	Date
Minor update	Updated Amazon Keyspaces information. Minor editorial corrections throughout.	July 28, 2023
Initial publication	Whitepaper published.	April 25, 2023

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents current AWS product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided “as is” without warranties, representations, or conditions of any kind, whether express or implied. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

AWS Glossary

For the latest AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.