|  |  |
| --- | --- |
| Title: Architecture Decision around choice of NoSQL Database | |
| Reference | AD-001-001 |
| Issue | APPLICATION-XYZ requires a scalable NoSQL database to store and retrieve data. |
| Underlying Business / Technical Requirements | 1. The database must allow data storage in JSON format. 2. Database must be secure and support encryption at rest and in transit. 3. Data retrieval must be fast and data should be available in < 10 milliseconds. 4. Database should support sharding or must be distributed to handle huge amount of data and traffic. 5. Database must be highly available. |
| Assumptions, Constraints, Enterprise Architecture Alignment | Chosen product must align with Enterprise Architecture Guideline of preferring PaaS and SaaS over COTS or any custom solution.  NoSQL database will be used for data which is not tightly related with other entities. Application will still use relational database for rightly related data. |
| Options | |  |  |  | | --- | --- | --- | | Amazon DynamoDB | MongoDB | Azure Cosmos DB | | Fully managed | Available as Fully Managed as well as Self-hosted | Fully Managed | | Cloud specific | Cloud agnostic | Cloud specific | | DynamoDB supports JSON-like documents but stores data in key-value or document format. You can store, query, and index JSON data. | Built natively for JSON-like document storage using BSON (Binary JSON). MongoDB offers rich querying and indexing features tailored for document structures. | Offers support for multiple APIs, including the MongoDB API and the Core (SQL) API, both of which handle JSON-like document storage effectively. | | Fully managed with built-in encryption at rest and in transit via AWS Key Management Service (KMS). Fine-grained access control using IAM policies. | Supports encryption at rest and in transit. In managed MongoDB Atlas, encryption is enabled by default. Advanced security features like field-level encryption are also supported. | Encryption at rest and in transit is enabled by default. It integrates with Azure Key Vault for additional security. | | Offers low-latency access (single-digit ms). DynamoDB Accelerator (DAX) further optimizes performance for read-heavy workloads. | Performance depends on the architecture and indexing strategy. MongoDB Atlas provides excellent response times but may require tuning for high-performance needs. | Consistently delivers <10ms latency for reads and writes globally, making it suitable for applications with strict performance SLAs. | | Natively distributed and automatically partitions data for scalability. No manual sharding is required. | Supports manual sharding, allowing you to partition data across multiple nodes. MongoDB Atlas automates sharding in a managed environment. | Natively distributed, providing global data distribution with automatic sharding. Highly suited for large-scale, multi-region applications. | | Built for high availability and fault tolerance. Data is automatically replicated across multiple availability zones in a region. | High availability is achieved through replica sets. Managed MongoDB Atlas simplifies this setup. | Provides multi-region replication. | | Dynamo DB read and write unit both are separate. So if you are writing more than reading then you can scale the writing unit independently. | MongoDB can scale reads and writes through replica sets and sharding, but the scaling is interconnected and not fully independent. | Cosmos DB has a request unit to interact with the database that is consist of reading and writing both and you scale that unit. You cannot scale both of them independently. It has a single Request unit for both read and writes operations. | | Offers partition keys to decide the placement of documents.  Dynamo Db offers an optional sort key. So the data stored in a particular partition will be sorted. | MongoDB uses the shard key to determine the placement of documents in the cluster. | Offers partition keys to decide the placement of documents. | | Dynamo DB, you can use AWS Amplify Datastore which eliminates the need for the backend in mobile clients. | MongoDB supports mobile clients through its solution called MongoDB Realm, which provides an offline-first, synchronized, and secure data layer for mobile applications. | Cosmos Db does not support mobile clients and needs a backend for it. | | Dynamo DB has a limited number of querying capabilities. | MongoDB provides powerful querying capabilities that make it highly suitable for a wide range of applications. | Cosmos Db has more flexibility as it supports multiple databases and real-time feeds through Azure Functions. Azure CosmosDB has SQL-like queries but is more complex than DynamoDB queries. | | Amazon DynamoDB offers Global Tables that can be used to replicate the data in multiple regions automatically. | There is no such provision in Azure CosmosDB. | There is no such provision in Azure CosmosDB. | |
| Recommended Option | Amazon DynamoDB |
| Justification | 1. Read units can scale independently, which will better as application requires more reads than writes. 2. Out of the box sort key will be easier to use for sorting. 3. Easy integration with Lambda and other AWS services. 4. As organization is already using AWS ecosystem, there will be less learning curve and overhead for using DynamoDB. |
| Implication | * Amazon DynamoDB must be approved for use by Security Team. * Guidelines for usage must be developed and circulated. * Amazon DynamoDB must be brought into MY-COMPANY-NAME AWS Services Catalogue. |
| Notes |  |
| Any follow-up Action |  |