Comparison of AWS Database Services

Service	Type	Cost (Estimated)	Performance	When to Use	Key Differences
RDS	Relational Database	Moderate to High	Varies by engine (MySQL, PostgreSQL, etc.), high read/write throughput with proper scaling	Use for traditional relational database workloads with complex queries and transactions	Supports multiple engines (MySQL, PostgreSQL, MariaDB, etc.), fully managed, scales vertically
AuroraDB	Relational Database	Moderate to High	High performance (5x MySQL, 3x PostgreSQL)	Use for high-performance relational workloads needing scalability and fault tolerance	Compatible with MySQL and PostgreSQL, auto-scaling, replicated storage across 3 AZs
DynamoDB	NoSQL (Key-Value Store)	Low to Moderate (based on read/write capacity)	Millisecond latency for reads and writes, scales horizontally	Use for low-latency applications with unstructured or semi-structured data	Serverless, fully managed, scales horizontally, great for key-value and document data models
DocumentDB	NoSQL (Document Store)	Moderate to High	High performance for JSON document workloads	Use for applications requiring MongoDB compatibility for document-based data	Managed MongoDB-compatible document store, scales horizontally with replica sets
Keyspaces	NoSQL (Wide-Column Store)	Low to Moderate (pay-per-request)	Low latency for high-throughput workloads	Use for Cassandra-compatible applications requiring massive scale and high availability	Managed Cassandra-compatible service, scales horizontally, great for wide-column data model
QLDB	Ledger Database	Moderate	High throughput, immutable and cryptographically verifiable transactions	Use for applications needing immutable, verifiable transaction logs (e.g., supply chain, finance)	Immutable ledger, ensures cryptographically verifiable transaction history

Neptune	Graph Database	Moderate to High	Low-latency graph queries at scale, optimized for graph traversal	Use for graph-based applications like social networks, fraud detection, and recommendation engines	Supports both property graph (Gremlin) and RDF graph (SPARQL) models
Timestream	Time Series Database	Low to Moderate	Optimized for ingesting, storing, and querying time-series data	Use for IoT, DevOps, or industrial telemetry workloads requiring time-series data	Fully managed, scales automatically, optimized for time-series workloads
ElastiCache	In-Memory Cache (Redis, Memcached)	Low to High (depending on instance type)	Microsecond latency, great for caching	Use for caching, session management, and real-time analytics	Managed Redis or Memcached, great for improving application performance via caching
MemoryDB	In-Memory Database (Redis)	Moderate to High	Microsecond latency, optimized for high availability and durability	Use for real-time applications that need persistent in-memory data, such as gaming, chat, and leaderboards	Managed Redis with high availability and durability features
Redshift	Data Warehouse (SQL-based)	Moderate to High	Optimized for high-performance querying and reporting	Use for large-scale analytics, BI reporting, and complex SQL querying on petabyte-scale data	Columnar storage, massive parallel processing (MPP), great for large-scale data warehousing

Cost

- RDS, AuroraDB, Redshift: Typically higher cost because of instance-based pricing and managed services with high performance.
- DynamoDB, Timestream: Pay-per-request pricing, making it cost-effective for low to medium traffic, but costs increase with higher throughput.
- ElastiCache, MemoryDB: Costs are based on the instance type and storage, with MemoryDB generally more expensive due to high availability and durability.
- Keyspaces, QLDB: Moderately priced, with Keyspaces offering pay-per-request pricing and QLDB pricing based on transaction volume and data storage.
- DocumentDB: Moderate to high cost due to the need for instance-based pricing and storage.

Performance Considerations

1. DynamoDB, ElastiCache, MemoryDB: Offer the highest performance due to their in-memory nature. High throughput with horizontal scaling.
2. AuroraDB, DocumentDB, Keyspaces, Neptune, Timestream: Provide high performance for most workloads.
3. RDS: Performance can vary depending on instance type and workload. Traditional relational database performance, but may require manual tuning.
4. QLDB: Provides high performance for ledger-based operations.
5. Redshift: Optimized for large-scale analytics.
6. Neptune: Optimized for graph queries, delivering low-latency graph traversal performance.

When to Use

- RDS/AuroraDB: For complex relational workloads with transactional support and complex joins/queries.
- DynamoDB: For highly scalable, low-latency key-value workloads.
- DocumentDB: For MongoDB-compatible applications that need a managed document database solution.
- QLDB: When you need cryptographic verification of transactional data.
- Neptune: For applications with complex relationships that require graph traversal and queries.
- Timestream: For IoT, time-series data, and telemetry that require fast ingestion and analysis of time-based data.
- ElastiCache/MemoryDB: For in-memory caching and real-time applications requiring microsecond-level access.

Additional Considerations:

- Aurora vs. RDS: Aurora offers better scalability and availability, especially for high-demand environments.
- DynamoDB vs. DocumentDB: DynamoDB is better for key-value workloads, whereas DocumentDB is more suitable for document-based, MongoDB-compatible use cases.
- Timestream vs. QLDB: Timestream is best for time-series data, whereas QLDB is for immutable, verifiable transaction logs.
- ElastiCache vs. MemoryDB: MemoryDB is for real-time, durable, in-memory data use cases, while ElastiCache is more for caching and session management.