Database selection

1. Brief Comparison of Selected Databases

Characteristic	MongoDB	PostgreSQL	M
Database Type	NoSQL (document-based)	SQL (relational)	S
Data Model	JSON-like documents (BSON)	Tables and relationships	T
Schema	Flexible (no fixed schema)	Rigid schema	R
Scalability	High (horizontal, sharding)	High (horizontal and vertical)	H
Complex Queries	Limited, not ideal for JOINs	Very robust and efficient	R
Transactions	Basic support (limited multi-document)	Full ACID support	F
Speed	Very fast for large volumes of data	Moderate at large volumes	M
Indexing Support	Advanced aggregation and compound indexes	Extensive indexing and optimization	S
Caching Capability	Built-in caching (with extensions)	External caching mechanisms needed	E n
Handling of Non-relational Data	Ideal for unstructured/semi-structured	Not recommended	N
Integration with NestJS	Seamless with @nestjs/mongoose	Excellent with @nestjs/typeorm	E
Typical Use Case	Semi-structured data, logs, high scalability	Complex data integrity, reports	M r

---

2. Selection of the Two Databases Most Relevant to the Requirements

Based on the requirements for flexibility in handling semi-structured data, scalability, and NestJS integration, the two most relevant databases for a deeper comparison are:

- **MongoDB**: Its flexibility for handling semi-structured data and high horizontal scalability, along with seamless integration with NestJS using Mongoose, makes it a strong candidate.
- **PostgreSQL**: Although relational, PostgreSQL's support for complex queries, strong data integrity, scalability, and robust integration with NestJS via TypeORM makes it a powerful option for more structured scenarios.

---

3. Detailed Comparison Between MongoDB and PostgreSQL

Characteristic	MongoDB	PostgreSQL
Database Type	NoSQL (document-based)	SQL (relational)
Data Model	JSON-like documents (BSON)	Tables and relationships
Schema Flexibility	Fully flexible, schema-less	Rigid schema with support for JSONB
Scalability	High scalability (horizontal, sharding)	High scalability (horizontal and vertical)
Complex Queries	Limited for complex joins and transactions	Very strong support for complex queries and joins
Transactions	Basic support (multi-document)	Full ACID compliance (robust transactions)
Performance for Reads/Writes	Optimized for high-write operations	Balanced for both reads and writes
Indexing and Query Optimization	Supports compound indexes and aggregation pipelines	Supports a wide range of indexes and query optimizations
Caching	Native support for in-memory caching	Requires external caching like Redis
Handling Semi-structured Data	Natively supports flexible document storage	Supports JSONB for semi-structured data, but less native than MongoDB
Integration with NestJS	Excellent with Mongoose ODM	Excellent with TypeORM ORM
Use Case	Ideal for logs, tracking, semi-structured data	Ideal for complex queries, data integrity, reports

Database Cost

Postgres SQL on Azure

Instancia	Núcleos virtuales	Memoria	Pago por uso
B1ms	1	2 GiB	$12.41/mes
B2ms	2	8 GiB	$99.28/mes
B2s	2	4 GiB	$49.64/mes
B4ms	4	16 GiB	$198.56/mes
B8ms	8	32 GiB	$397.12/mes
B12ms	12	48 GiB	$595.68/mes
B16ms	16	64 GiB	$794.24/mes
B20ms	20	80 GiB	$992.80/mes

MongoDB on Azure

Cluster Tier	Storage	RAM	vCPUs	Base Price
M10	8 GB	2 GB	1 vCPU	$0.08/hr
M20	32 GB	4 GB	1 vCPU	$0.19/hr
M30	32 GB	8 GB	2 vCPUs	$0.51/hr
M40	64 GB	16 GB	4 vCPUs	$0.99/hr
M50	128 GB	32 GB	8 vCPUs	$1.95/hr
M60	128 GB	64 GB	16 vCPUs	$3.73/hr
M80	256 GB	128 GB	32 vCPUs	$7.42/hr
M200	256 GB	256 GB	64 vCPUs	$13.65/hr
M300	512 GB	384 GB	48 vCPUs	$16.63/hr
M400	512 GB	432 GB	64 vCPUs	$20.74/hr
M600	4000 GB	640 GB	80 vCPUs	$46.87/hr

4. Justification for Choosing MongoDB

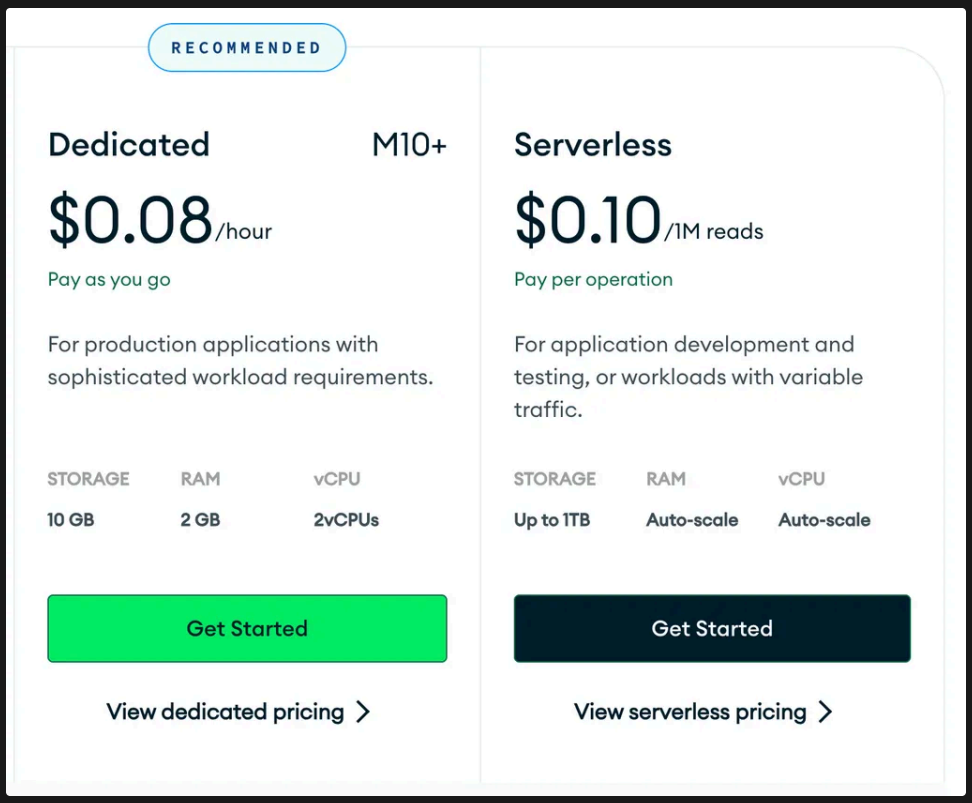
After the detailed comparison, **MongoDB** emerges as the better option for this middleware project due to the following key factors:

1. **Schema Flexibility**: MongoDB is perfectly suited for handling semi-structured and unstructured data, with its schema-less nature allowing for evolving data models. While PostgreSQL offers support for JSONB, MongoDB is designed specifically for document storage, making it a more natural fit for unstructured data.
2. **Scalability**: MongoDB excels in environments where rapid horizontal scalability is necessary, thanks to its sharding capabilities. This is essential if the middleware needs to manage large distributed data volumes efficiently.
3. **Integration with NestJS**: MongoDB's seamless integration with NestJS via @nestjs/mongoose allows for easy data management and querying through an Object Document Mapper (ODM), reducing development friction and ensuring smooth implementation.

4. **Performance for Write-heavy Workloads**: MongoDB is optimized for high-write workloads, making it ideal for logging, tracking, and storing input/output data, which is crucial in middleware operations where data is constantly being generated and stored.
5. **Handling Semi-structured Data**: Since the middleware will process data with varying degrees of structure, MongoDB's document model provides the best solution. It allows flexible storage and efficient querying of semi-structured and unstructured data, avoiding the need for complex migrations that a relational database like PostgreSQL would require.

In conclusion, MongoDB's flexibility, performance, and scalability make it the best choice for this middleware project, especially considering the unstructured nature of the data and the need for efficient scaling.

Recommendation: Serverless Database vs. M10 Instance



Comparison Criteria:

1. **Cost Efficiency**:
    - **M10**: Fixed cost at $0.08 per hour, amounting to approximately $57.60 per month for continuous operation. This is predictable and stable, ideal for consistent workload levels.
    - **Serverless**: Pay per operation, specifically $0.10 per million reads. The cost-effectiveness hinges on fluctuating or unpredictable workloads. For low to moderate usage levels, this could significantly reduce costs.
2. **Scalability**:
    - **M10**: Requires manual scaling decisions to adjust resources as your needs grow, which could involve some downtime or delay.
    - **Serverless**: Automatically scales the RAM and vCPU based on the workload demands, which is advantageous during unpredictable traffic spikes, ensuring that the database can handle loads without manual intervention.
3. **Operational Simplicity**:
    - **M10**: Involves some management regarding scaling and maintenance, which could require additional overhead or expertise.
    - **Serverless**: Reduces operational overhead by eliminating the need to manage database capacity, focusing your efforts more on development and less on database administration.
4. **Predictability**:
    - **M10**: Provides a stable environment with predictable performance, which is crucial for applications requiring consistent database interaction and quick response times.
    - **Serverless**: While offering flexibility in resource utilization, it may introduce some unpredictability in costs and performance, especially if the workload significantly varies from month to month.

Recommendations Based on Usage Patterns:

- **For Consistent Workloads**: If your middleware consistently uses the database at a predictable rate (e.g., steady logging, session management), the **M10** is likely more economical and reliable.
- **For Variable Workloads**: If your project experiences significant fluctuations in usage (e.g., batch processing at certain times, unpredictable user-driven events), the