Milestone 5: Hybrid Architecture Implementation Report

1. Executive Summary: The Business Problem

AetherMart is "drowning in data" from new, diverse channels (web clickstreams, IoT devices, social media, chat logs). The Project.docx business case identifies several key challenges that our current architecture cannot solve:

- **Maria (Marketing):** Needs a "360-degree customer view" to analyze "granular customer journeys" but this data doesn't fit in the current SQL tables.
- **Sarah (Product):** Is frustrated by the "rigid structure" of the Products table, which slows down the launch of new, diverse products (e.g., a smart speaker vs. a digital consultation).
- **Alex (CTO):** Needs to handle this new data *volume* and *variety* without creating more "data silos" and wants to enable real-time analytics.
- **David (Architect):** Proposed a "hybrid data architecture"—integrating a specialized NoSQL database—to solve this.

Our goal for Milestone 5 was to design, implement, and validate this hybrid architecture by integrating a MongoDB NoSQL database with our existing MariaDB cluster.

2. Implementation Process: Building the Hybrid Architecture

We successfully built a secure, multi-instance, heterogeneous environment within AWS. The process was as follows:

1. **Provisioning (The "Hybrid" Part):** A new, separate AWS EC2 t2.micro instance (AetherMart-MongoDB-Server) was launched. This is critical: it isolates the NoSQL workload from the live transactional MariaDB database, preventing resource contention.
2. **Network Security (Layer 1):** A new AWS Security Group (MongoDB-SG) was created and applied to the new instance. This cloud-level firewall is our primary security defense.
3. **Installation & Configuration:**
    - MongoDB Server (v7.0) was installed and configured on the new EC2 instance.
    - The mongod.conf file was modified (bindIp: 0.0.0.0) to allow it to listen for network connections within our private AWS network.
4. **Application Security (Layer 2):**
    - We enabled Role-Based Access Control (RBAC) in mongod.conf by setting

security: authorization: "enabled".
    - We created the aethermart_admin user in the admin database with full read/write privileges.
5. **Data Integration (The "ETL" Part):**
    - On our original MariaDB server, we installed the pymysql and pymongo Python drivers.
    - We created a "bulk" ETL script (migrate_to_mongo.py) that securely connects to *both* databases, Extracts data from MariaDB, Transforms it (cleans data, converts types, enriches with new fields), and Loads it into MongoDB.

3. Security Strategy: Defense-in-Depth

This architecture directly addresses Sarah's concerns about PII by implementing security at multiple layers.

Layer 1: Network-Level Isolation (The Key POC)

Our primary security measure is the MongoDB-SG firewall, which makes the database invisible to the public internet. Its "Inbound Rules" are:

Type	Protocol	Port Range	Source	Description
SSH	TCP	22	**My IP**	Allows *only me* to manage the server.
Custom TCP	TCP	27017	**[MariaDB Private IP]**	Allows *only our application server* to talk to the database.

Justification: This lockdown means the only entity that can talk to our MongoDB database is the trusted, internal application server. This is a best-practice design that massively reduces the attack surface.

Layer 2: Application-Level Security (Completed)

The firewall protects the *server*; RBAC protects the *data*. We have **completed** this step:

- **Authentication is Enabled:** By setting security: authorization: "enabled", all anonymous access to the database is **blocked**.

- **Admin User Created:** We created the aethermart_admin user.
- **Scripts Secured:** Our Python scripts (migrate_to_mongo.py and test_hybrid.py) were updated to pass these new credentials, allowing them to authenticate and perform their tasks.

This two-layer approach ensures that even if an attacker breached our private network, they would still need valid credentials to access or modify any data.

4. Data Migration Strategy: A Table-by-Table Analysis

We did not perform a "full migration." We built a hybrid system, making strategic decisions for each table in the aethermart_db schema.

Table(s)	Decision	Justification (The "Why")
Customers	**ENRICH & COPY**	**(Solves Maria's Problem)** Kept in MariaDB for Orders integrity. A new, enriched copy was created in MongoDB (customer_profiles) to hold the 360-degree view (logs, preferences).
Products & Categories	**DENORMALIZE & ENRICH**	**(Solves Sarah's Problem)** Products kept in MariaDB for stock/price. A new product_catalog collection was created in MongoDB, denormalizing the category_name and adding a flexible specifications field.
Reviews	**MOVE (RECOMMENDED)**	This data is semi-structured and has no hard foreign key dependencies. It's a perfect candidate to move to MongoDB, allowing for new features like photo/video uploads and upvotes.

`Orders` & `Order_Items`	**STAY IN MARIADB**	**(Critical)** These are the heart of the transactional (OLTP) system. They are ACID-dependent and must remain in a relational database.
`Suppliers`	**FUTURE CANDIDATE**	A good candidate for migration. A many-to-many junction table in SQL becomes a simple array in a MongoDB document, simplifying the schema.
`Audit_Logs`	**FUTURE CANDIDATE**	Log tables are a perfect fit for MongoDB, which excels at storing and querying massive volumes of "write-once, read-rarely" data.

5. POC Validation: How We Solved the Business Case

We ran "test" queries in the `mongosh` shell to prove our solution solves the business problems.

Business Problem	Proof-of-Concept (POC) Test
Maria's "360-Degree View"	We ran `db.customer_profiles.findOne(...)` and showed a single JSON document containing that customer's SQL data (name, email) *plus* new fields for `recent_activity_log` and `customer_preferences`.
Sarah's "Flexible Schema"	We ran `updateOne()` on two *different* products. We gave one product `color` and `wifi_standard` specs, and the other `duration_hours` and `platform` specs. This *proves* we can have different attributes for different products in the same collection.

Alex's "No Data Silos"	We ran an `aggregate()` query (`db.customer_profiles.aggregate(...)`) to group customers by state. This *proves* the new database is not a "dumb silo" but a powerful, queryable analytics store, augmenting our M4 data warehouse.

6. Future Enhancements

This new hybrid architecture unlocks new capabilities for AetherMart, which we can propose as future enhancements:

1. **Real-Time Analytics Dashboard:** Our M4 Data Warehouse is for *historical* analytics. We can now use MongoDB to build a *real-time* dashboard for Sarah, showing sales *as they happen*.
2. **Complete the Migration:** Finish the strategic migration by moving the `Suppliers` and `Audit_Logs` tables to MongoDB to further simplify the architecture and improve performance.