JUNE 2025

The Operational Data Layer

How MongoDB empowers enterprises to modernize and innovate

MongoDB

Table of Contents

Abstract

Every large organization inevitably accumulates **“digital dirt”**. Legacy systems evolve, new applications are introduced, and the overall infrastructure becomes increasingly complex. Mergers and acquisitions only add to this complexity.

Applications originally built for specific user groups and access patterns are suddenly expected to interoperate with other systems, support holistic data views, and enable use cases far beyond their initial design. This white paper presents the architectural blueprint for an **Operational Data Layer (ODL)**, an approach that consolidates and organizes siloed enterprise data, making it readily accessible to consuming applications, particularly next-generation AI initiatives. It focuses on how MongoDB empowers enterprises by serving as the core foundation for this ODL and outlines key architectural patterns, data ingestion methods, and access layer considerations.

By **simplifying data modeling and management**, decoupling modern workloads from the limitations of legacy systems, and improving scalability, availability, and responsiveness, an ODL empowers organizations to **innovate faster** and adopt an agile data strategy.

The ODL is a **powerful enabler of innovation**, essential for empowering **next-gen AI**, **generative AI**, and **agentic AI** applications. It is also critical for addressing data fragmentation and overcoming silos, enabling the creation of a unified single view of data (such as customer 360 or product 360) by consolidating data from multiple sources. This white paper also presents a series of real-world, industry-specific examples that demonstrate the impact of an ODL across various business domains.

Agentic AI systems are currently a topic of discussion at nearly all major companies. These are artificial intelligence systems that function with a certain level of autonomy, capable of initiating actions to pursue specific objectives while adapting their behavior in response to changing environments, goals, or inputs. Leveraging MongoDB as the underlying memory layer for these systems offers a highly scalable and flexible solution–one that can evolve alongside shifting or expanding requirements.

Use Cases

Let's explore the most prominent projects and use cases where an Operational Data Layer has proven to be a powerful enabler of innovation, allowing organizations to leverage existing data in new and modern ways.

Empowering next-gen AI applications

Large organizations are aggressively exploring ways to capitalize on the emerging paradigm of large language models (LLMs) and agentic AI–both to streamline existing applications and to create entirely new ones, unlocking fresh revenue streams in the process.

At the heart of this transformation is the **strategic use of data**–leveraging the vast amounts of existing organizational data, combined with the continuous influx of new data generated daily in gigabytes or even terabytes, depending on the industry (e.g., IoT sensor data, telemetrics, call detail records, etc.).

The goal is to build a robust data layer capable of powering real-time intelligence and decision-making–whether it's determining fraud in a split second, acting as an internal advisor that provides natural language insights to employees, or fueling complex agentic AI workflows.

In reality, **many of these AI initiatives are failing today**. A key reason is the incompleteness of the data being used. Much of the necessary information remains trapped in silos, fragmented across systems, and difficult to access in a holistic and unified way. As a result, LLMs often hallucinate, inventing information to fill in the gaps.

This can have serious consequences: A customer service chatbot may provide inaccurate–or even laughably incorrect–advice, rendering it ineffective or embarrassing. Worse still, in high-stakes use cases such as financial risk assessment, flawed outputs can lead to poor decisions, regulatory fines, and ultimately a loss of money, reputation, or trust.

We now highlight four fundamental AI use cases that can significantly benefit from an operational data layer: feeding AI training data, providing context for LLMs to build a RAG architecture, powering agentic AI workflows, and establishing hybrid search experiences.

AI training data

One of the first steps in an AI process pipeline is the creation of custom machine learning models. These models are necessary whenever standard solutions, such as open-source models, are not sufficient to address the specific needs of a business. Instead of relying on generic approaches, organizations must develop models that are tailored to their particular use cases.

This need arises because standard models cannot account for the individual ways in which companies operate. In areas like fraud detection and prevention, for example, generic models often fail because fraud patterns vary widely based on how each organization operates–whether in the channels they use, the roles involved, or the types of products and services they offer. Each of these variables influence the types of fraudulent behaviour that may emerge. To achieve reliable and accurate results, models must be trained with specialized, context-specific data rather than relying on generalized assumptions.

An operational data layer provides the ideal foundation for building such models. It offers access to rich, continuously updated training data, supports iterative improvement of model quality over time, and scales effortlessly with growing data volumes and performance demands. By centralizing and managing enterprise data effectively, the ODL ensures that AI initiatives are not only more precise but also more resilient and future-proof.

Context for RAG

Large language models excel at providing natural language answers and solutions across a wide range of domains. Their capabilities extend beyond text, offering multi-modal outputs such as generated images, audio, and more. LLMs have quickly become a foundation for many AI-driven applications, delivering sophisticated interactions that feel intuitive and human-like.

However, the knowledge LLMs rely on is limited to the data incorporated during their initial training. Every model has a so-called cut-off date, beyond which it has no awareness of new developments. Moreover, LLMs are typically trained only on publicly available information and do not have access to proprietary enterprise data. This creates significant gaps, particularly when applications require the latest insights or organization-specific knowledge. Without additional input, models are prone to hallucination–inventing information to fill these gaps–which undermines trust and effectiveness.

To address this, retrieval-augmented generation (RAG) has emerged as a powerful architectural approach. By supplying relevant, up-to-date information alongside the user's query, RAG ensures that LLMs work with accurate and complete context. MongoDB's native support for vector search and vector embeddings allows models to retrieve semantically similar documents from internal, proprietary sources, delivering highly relevant and reliable responses. Combined with a centralized enterprise knowledge base powered by an ODL, organizations can significantly enhance chatbot performance, automate customer support, and enable real-time, AI-driven decision-making, creating smarter, faster, and more responsive user experiences.

Agentic AI

Agentic AI refers to a class of artificial intelligence systems designed to autonomously make decisions and take actions to achieve specific goals with minimal human intervention. Unlike traditional AI, which often operates within predefined parameters, agentic AI systems exhibit higher levels of autonomy, enabling them to plan, adapt, and execute complex tasks in dynamic environments. These systems integrate various AI techniques, including natural language processing, machine learning, reinforcement learning, and knowledge representation, to function effectively across diverse applications.

The core technical challenge in building such systems is their constant need for relevant, contextualized data–both structured and unstructured, historical and real-time. These agents must integrate insights from diverse formats: logs, sensor streams, transaction records, documentation, conversations, media files, and embeddings. The data is not only vast but highly volatile, evolving rapidly as environments shift and agents interact. Traditional architectures struggle under this strain, especially when composed of isolated systems with narrow interfaces. Data translation friction, inconsistent latency, and poor support for semantic or event-driven access hinder the responsiveness these agentic systems require. Since AI agents operate continuously and adaptively, any bottleneck in accessing or interpreting the operational state directly harms system performance and decision quality.

This is where an ODL becomes essential. It serves as a unified foundation that connects legacy systems, real-time feeds, and diverse data sources–delivering a coherent, queryable interface designed for AI agents. It allows these systems to reason across modalities, find patterns based on meaning rather than structure, and react to state changes without brittle custom integrations. By unifying operational memory into a single layer that supports semantic queries, streaming updates, and multimodal access, the ODL forms the infrastructure backbone for agentic AI. It gives agents the high-fidelity, low-latency, and context-rich access they need to operate autonomously–supporting complex architectures without introducing fragility or opacity.

MongoDB has developed a cross-industry Framework for Rapid AI Agent Deployment.

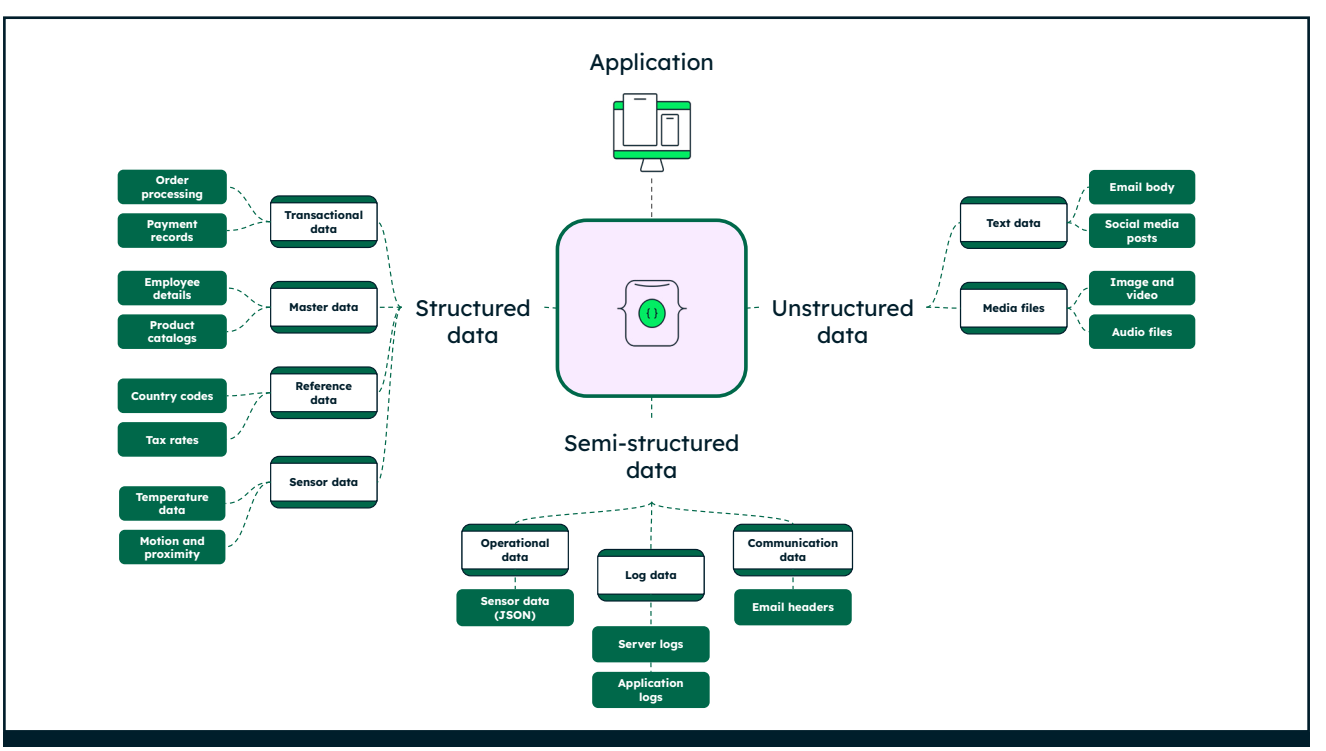

Figure 1: How MongoDB handles structured, semi-structured, and unstructured data.

Advanced search experiences

Modern search experiences have moved far beyond simple keyword matching. Users today expect intelligent, fast, and highly relevant results, not just based on words but on context, intent, and across multiple data formats like text, images, audio, and even video. A seamless, intuitive search experience has become critical for driving engagement, satisfaction, and loyalty across digital platforms.

The reason for this shift is clear: Traditional keyword-based approaches fail to capture the true meaning behind a query. They miss nuance, intent, and the growing expectation for personalization and immediacy. As businesses compete for user attention, delivering smarter, more responsive search capabilities has become a key differentiator, whether for retail, entertainment, e-commerce, or customer support.

This is where an ODL proves invaluable. By combining full-text indexing with vector search in a unified architecture, it enables applications to understand meaning, not just keywords. It supports petabyte-scale data ingestion and high query throughput without performance bottlenecks, enforces data governance across diverse sources, and empowers AI-driven insights—all while ensuring search experiences are faster, more accurate, and more personalized than ever before.

Single view / overcoming silos

Over the years, enterprise data landscapes have become increasingly fragmented. Many applications were built for narrow, internal use cases such as billing, CRM, and support, with little thought given to scale, reuse, or long-term flexibility. Each acquisition introduced new systems and redundant data silos. The result has been a patchwork of disconnected platforms, mismatched schemas, and operational blind spots. These legacy architectures cannot support new audiences, prevent consistent SLAs, and make even basic tasks like assembling a customer profile unnecessarily complex and error-prone.

This fragmentation is more than just inefficient. It actively blocks innovation. Critical upstream projects stall because access to the necessary data is technically impossible, operationally infeasible, or simply too expensive. AI initiatives lack the clean, contextual data they need. Real-time service improvements and cross-functional automation efforts are slowed to a crawl. Teams spend more time waiting for integration work or managing inconsistent datasets than delivering value.

An ODL offers a fundamentally different approach. Instead of starting with the limitations of legacy systems, teams can begin with the goal. Define the outcome: What data is needed? What API should expose it? And in what format it must be structured for efficient development and access? Only then do we trace back to where that data resides—in which systems, tables, and formats—and define change-data capture (CDC) or extract, transfer, load (ETL/ELT) pipelines accordingly. This reverses the traditional model: We design as if building a greenfield app, unconstrained by outdated infrastructure, while still integrating what's necessary from legacy systems. The result is a streamlined, forward-looking architecture that accelerates development, decouples teams from legacy bottlenecks, and makes room for real-time, AI-driven and scalable innovation.

Data-as-a-Service (DaaS)

Data as a Service (DaaS) is a modern approach to data delivery that treats data not as a side effect of business systems, but as a reusable product–available on demand, consistently structured, and consumable through APIs, subscriptions, or query interfaces. It decouples data access from the systems where data is stored, giving developers, analysts, operations teams, and even external partners a unified, governed, and always-on way to work with information. In a DaaS model, data behaves like any other service: discoverable, versioned, and accessible wherever it's needed.

The challenge isn't that organizations lack data. It's that their current architectures were never designed to *deliver* data as a product. Most internal data flows are tightly bound to application logic or limited to one-off extracts for narrow use cases. Data pipelines are handcrafted, context-specific, and hard to reuse. This makes data access brittle and slow, but it also creates deeper organizational friction. Teams duplicate effort, developers reinvent interfaces, and analysts build parallel logic simply because there's no shared foundation to consume data consistently. The cost isn't just technical debt but lost speed, insight, and alignment.

This is where an ODL can help realizing the promise of DaaS. The ODL acts as a unified, centralized layer that aggregates and harmonizes data from multiple sources, transforming fragmented and inconsistent information into a single, coherent view. It provides real-time or near-real-time synchronization with underlying systems, ensuring that the data delivered through DaaS is always current and actionable. The ODL exposes this data through robust APIs, event streams, and virtual views, making it easily consumable by a wide range of applications and users. Crucially, the ODL incorporates comprehensive governance, security, and access controls, so that data is not only accessible but also trustworthy and compliant. Its scalable and resilient architecture allows organizations to expand data services as needs evolve, supporting a growing ecosystem of data consumers without disrupting existing operations. In this way, the ODL transforms DaaS from a theoretical ideal into a practical, reliable, and strategic capability for modern organizations.

Data governance and data sovereignty

Data governance and sovereignty refer to an enterprise's ability to control, protect, and regulate how data is accessed, shared, stored, and processed, both internally and externally. This includes enforcing who can see what, ensuring that sensitive or personal data is handled appropriately, and supporting compliance with local regulations like GDPR. It's not just about access; it's about accountability, traceability, and trust. True sovereignty means an organization can decide where its data resides, how it's used, and under what legal and technical conditions it's exposed. At the technical level, this encompasses encryption, access controls, auditing, and the ability to deliver only the minimum required data, such as exposing aggregates instead of raw records to downstream consumers.

In the absence of robust governance and sovereignty mechanisms, enterprises face significant risk. Without clear data boundaries and access controls, sensitive customer information can leak across teams, systems, or borders. Regulatory breaches can lead to steep fines,

reputational damage, and legal exposure. Worse still, a lack of control often leads to over-correction: Organizations restrict data access so severely that teams can't move forward. Innovation slows, AI models operate on incomplete inputs, and business users are forced to work around IT in nonsecure, fragmented ways. The result is both risk *and* inefficiency–two forces that pull the organization in opposite directions at the same time.

An ODL provides a clean, enforceable separation between data producers and consumers, allowing governance to be applied centrally and consistently. Instead of connecting each team or application directly to source systems, the ODL acts as a controlled interface–shaping, filtering, and enriching data according to defined policies. Sensitive fields can be masked, access restricted by role, and data aggregated before exposure, ensuring that customer-level details are only shared when strictly necessary. Encryption at rest and in transit, along with detailed audit logging, ensures that every data operation is visible and verifiable. Crucially, the ODL allows these policies to be implemented without slowing down development, enabling real-time access where permitted, and denying it where not.

By shifting governance into a structured, purpose-built layer, enterprises don't have to choose between control and agility. The ODL gives them both. It ensures that data usage complies with regional regulations like GDPR and internal policies while still enabling teams to move fast, collaborate securely, and build with confidence. Data sovereignty becomes operational, not theoretical, embedded into how data flows, decisions are made, and systems interact. This unlocks innovation with guardrails, allowing organizations to act boldly without losing control.

Data APIs

In modern architectures, data APIs have become a strategic mechanism for delivering data, not just within teams but across organizational and even ecosystem boundaries. Internally, they allow developers to access trusted, real-time data without needing direct access to complex source systems. Externally, they enable partners, platforms, or customers to consume selected datasets in a controlled, auditable, and secure way. In both cases, APIs offer clean contracts, schema stability, access control, and observability far beyond what traditional database access or file exports can provide.

Crucially, data APIs also decouple data consumption from the performance and availability limitations of internal source applications. Rather than hitting backend systems directly–which may have variable load, maintenance windows, or no built-in resilience–requests are served through the ODL, which acts as a scalable, real-time buffer and governance layer. This separation allows the API to meet independent, high-availability SLAs–even 99.995% or higher–without being tied to the uptime of legacy infrastructure.

With the ODL managing data access, transformations, and caching, data APIs become stable, fast, and predictable. Developers build confidently on well-defined interfaces. Partners integrate without needing deep system knowledge. Consumers, whether human or machine, get reliable access to the data they need, when they need it, with full compliance and control. Data APIs, powered by the ODL, turn data delivery into a product–consistent, secure, and built to scale.

Architectural Approaches

Designing an ODL with MongoDB involves more than integrating tools and technologies–it requires aligning architectural decisions with the goals of performance, flexibility, scalability, and maintainability. This section outlines core architectural paradigms and patterns that guide a robust ODL implementation using MongoDB.

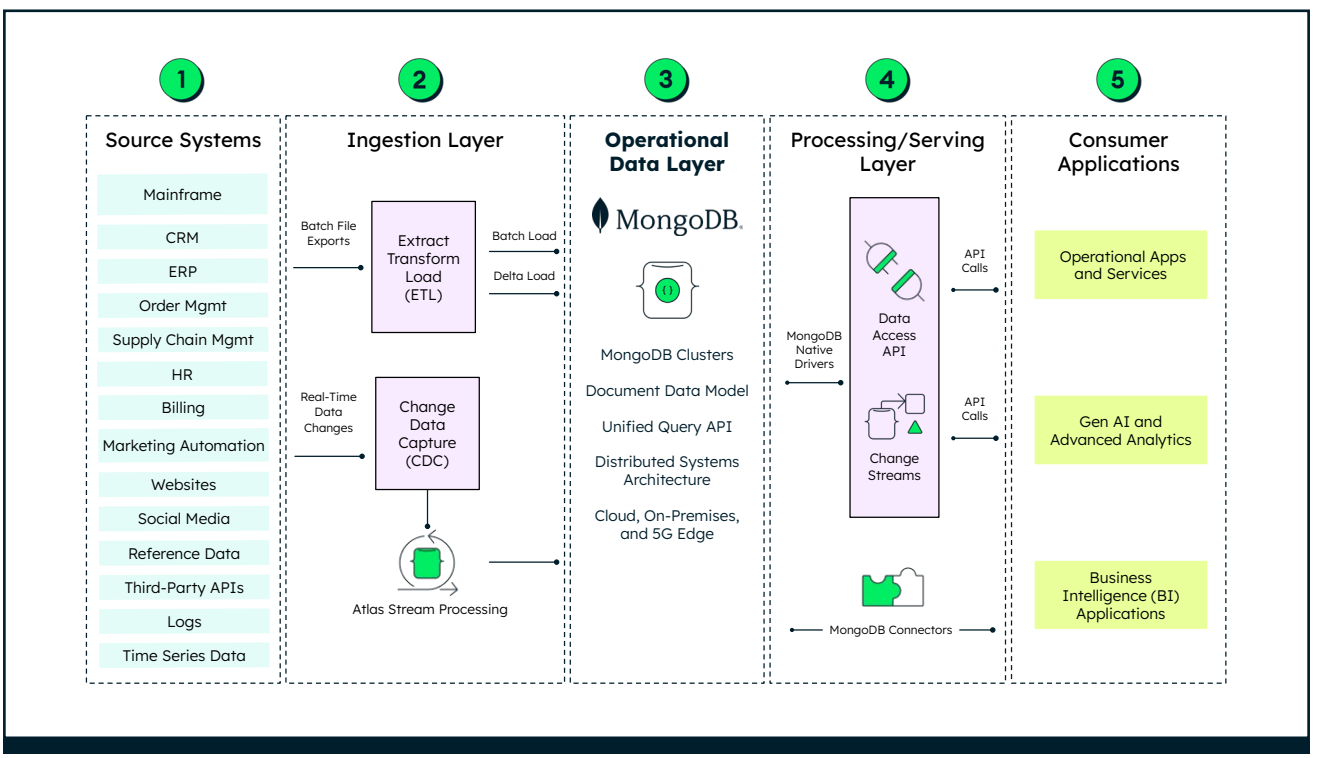

Figure 2. Operational data layer architecture.

The architecture in Figure 2 depicts a MongoDB-centric ODL spanning five conceptual layers:

1. **Source systems**
   These are the systems where data originates–enterprise applications, operational databases, third-party APIs, or legacy systems. Often referred to as "systems of record," they capture critical business transactions and events that must be synchronized into the ODL to enable real-time and contextualized insights.

2. **Ingestion layer**
   Data from the source systems is then ingested into the Operational Data Layer through different ingestion methods, which can be batch-based ETL processes or real-time change-data capture (CDC) mechanisms. The ingestion method is determined by latency requirements and source system capabilities.

3. **Operational data layer**
   This is the core of the architecture, where MongoDB consolidates structured, unstructured, and semi-structured data from multiple sources into a flexible document data model. It supports hybrid workloads–including transactional processing, real-time analytics, and full-text search–while enabling a unified query interface and seamless scaling across cloud, on-premises, and edge environments.

4. **Processing/serving layer**
   This layer enables secure and performant data access through MongoDB native drivers and connectors such as Atlas SQL. It serves data to downstream systems, exposing it in a consistent and governed way to support internal applications, external services, and advanced analytics use cases.

5. **Consumer applications**
   These are the consuming applications and services that leverage data from the ODL. They include operational applications, gen AI and agentic AI systems, and business intelligence tools–each benefiting from real-time, consolidated, and queryable access to enterprise data.

Levels of implementation

Organizations adopting an ODL do not need to follow a rigid sequence of phases. Instead, they can choose the level of integration that aligns with their strategic goals, technical readiness, and desired outcomes. These levels–read-only, enriched, and read-write–represent distinct architectural patterns that can be adopted independently or combined. Each unlocks specific benefits across use cases such as AI enablement, unified views, data services, and governance.

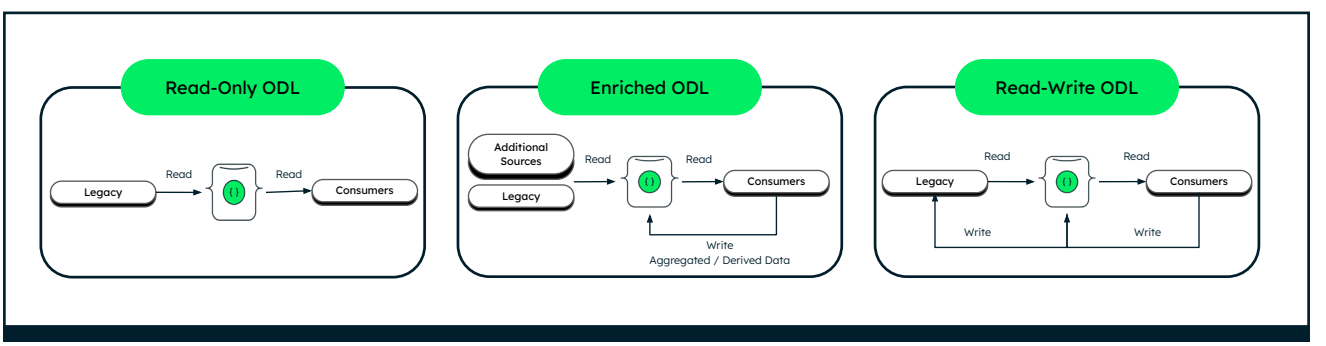

Figure 3. Levels of implementation of an ODL.

Read-only ODL

At this level, the ODL acts as a high-performance read replica of the source systems, often including legacy mainframes or siloed transactional databases. This setup is ideal for organizations looking to offload read traffic, support analytics, or modernize access to critical data via APIs. The ODL essentially acts as a flexible cache in this scenario.

By centralizing operational reads, teams can begin delivering data-as-a-service, making curated, low-latency data available to downstream consumers without impacting source systems. This level also lays the groundwork for data APIs, enabling developers to build modern apps with consistent access to legacy and real-time data.

This approach is common in low-risk modernization initiatives, where the ODL offloads read traffic from source systems, enhancing performance and availability.

Enriched ODL (read with context)

At this level, the ODL evolves beyond being a simple replica of source systems. It is enriched with metadata and reference data, including inputs from external sources such as third-party providers or public web datasets. Rather than just replaying raw operational data, the ODL delivers contextualized, queryable information that supports a wide range of use cases.

For example, an enriched ODL might ingest raw IP address data and augment it with geolocation metadata–such as country, city, and timezone–enabling personalized recommendations or region-specific services.

This enriched context allows organizations to break down data silos and build a single customer view, providing both operational and analytical systems with access to unified, trusted records across channels. The enriched ODL also serves as

a foundation for governed data access, applying policies for data quality while enabling self-service analytics and exploration.

Crucially, this level accelerates the development of gen AI applications by delivering complete, real-time, and contextual inputs to LLMs and AI pipelines–without requiring costly upstream data consolidation.

This approach is especially valuable for organizations pursuing operational intelligence, regulated access to cross-domain data, and rapid AI innovation using rich and timely data.

Read-write ODL

At this level, the ODL participates in writes. Applications write data both to the ODL and to the source system–often using messaging platforms or API gateways to coordinate this dual-write pattern (sometimes called Y-loading or Y-storing).

To maintain consistency across systems, organizations often adopt the transactional outbox pattern or the saga model. These techniques ensure coordinated writes without requiring distributed locking, balancing resilience with practical guarantees of correctness.

This model supports true bidirectional data flow and paves the way for legacy replacement, system rationalization, or even real-time operational platforms powered by the ODL. Data APIs can be fully backed by the ODL, supporting both transactional and analytical behaviors. Generative AI applications benefit from immediate feedback loops, where user interactions or AI-inferred updates are captured directly in the ODL. This level also enables dynamic data governance, enforcing policies not just on reads but on data creation and lineage.

Organizations implement this level when the goal is architectural transformation–moving toward modern, cloud-native operations, reducing dependency on legacy systems, or positioning the ODL as a system of record.

Architectural patterns

Event-driven and microservices architectures

Modern data systems often adopt a **microservices architecture**, where functionality is split into independently deployable and scalable services. These services own their logic and are typically grouped into bounded contexts, each of which may manage its own data store. This promotes agility, modularity, and resilience while allowing multiple related services to interact with a shared domain model.

To support loosely coupled interactions, an **event-driven architecture (EDA)** is often layered on top. In this model, services publish and subscribe to events (e.g., OrderPlaced, PaymentCompleted) via a message broker like Kafka. This asynchronous communication model enhances scalability and evolution by decoupling producers from consumers.

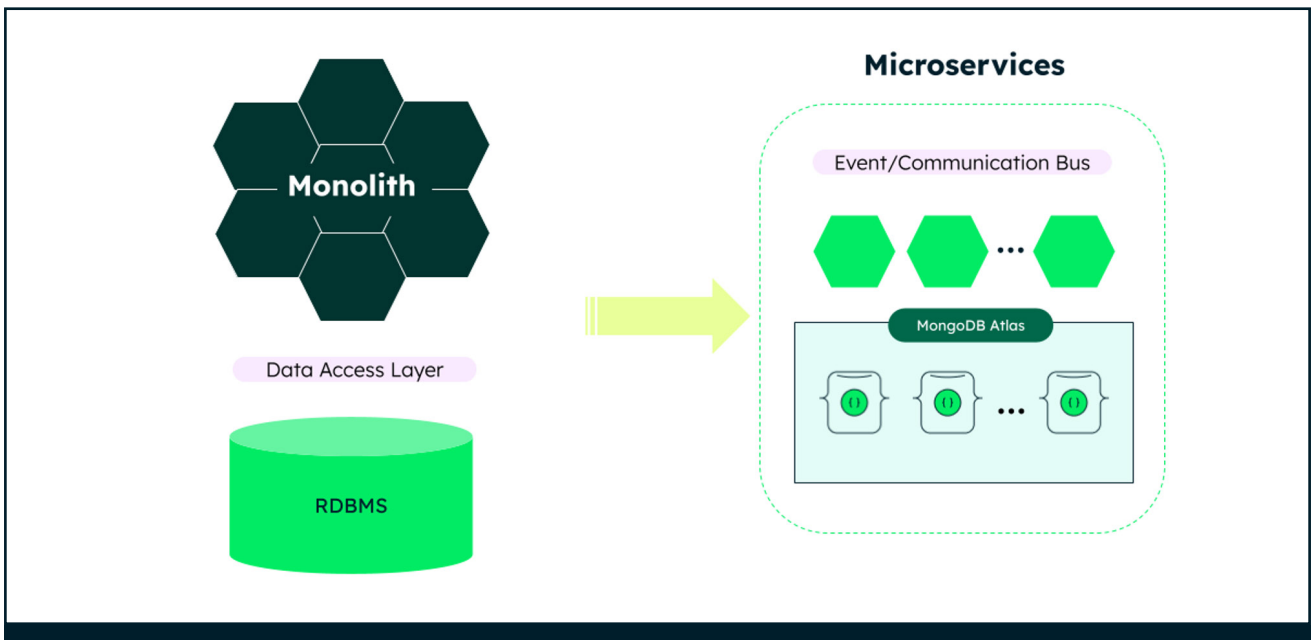

Figure 4. Comparison between monolithic and microservices architectures.

An ODL is the ideal foundation for implementing microservices and event-driven architectures because it provides a unified, real-time access layer that bridges the data silos typically introduced by service-level autonomy. While microservices benefit from owning their own data for encapsulation and independence, this often leads to fragmentation and duplication when services need to share or correlate data. The ODL solves this by acting as a central hub that aggregates, harmonizes, and serves operational data across services without compromising their autonomy. It ensures that each service can access up-to-date, relevant information without tight coupling or direct dependency on other services' internal databases.

This is especially powerful in event-driven systems, where the ODL can serve as both a source and sink of enriched contextual data—enhancing event payloads, supporting idempotency, and enabling reactive patterns such as event replay, compensation, and temporal queries. In short, the ODL provides the high-performance, scalable data backbone that modern microservices architectures require to operate efficiently and evolve independently.

Workload isolation: OLTP and OLAP

In traditional data architectures, analytical (OLAP) workloads often compete with transactional (OLTP) operations for the same resources, leading to degraded performance and reliability. This is especially problematic when real-time responsiveness is critical for transactional systems.

Workload isolation solves this by separating the two types of workloads—typically within the same database infrastructure—so that heavy analytics don't interfere with operations like inserts, updates, or low-latency queries. In MongoDB, this can be achieved through several architectural patterns that allow database operations to become more infrastructure and data center aware, enabling geographic and operational separation.

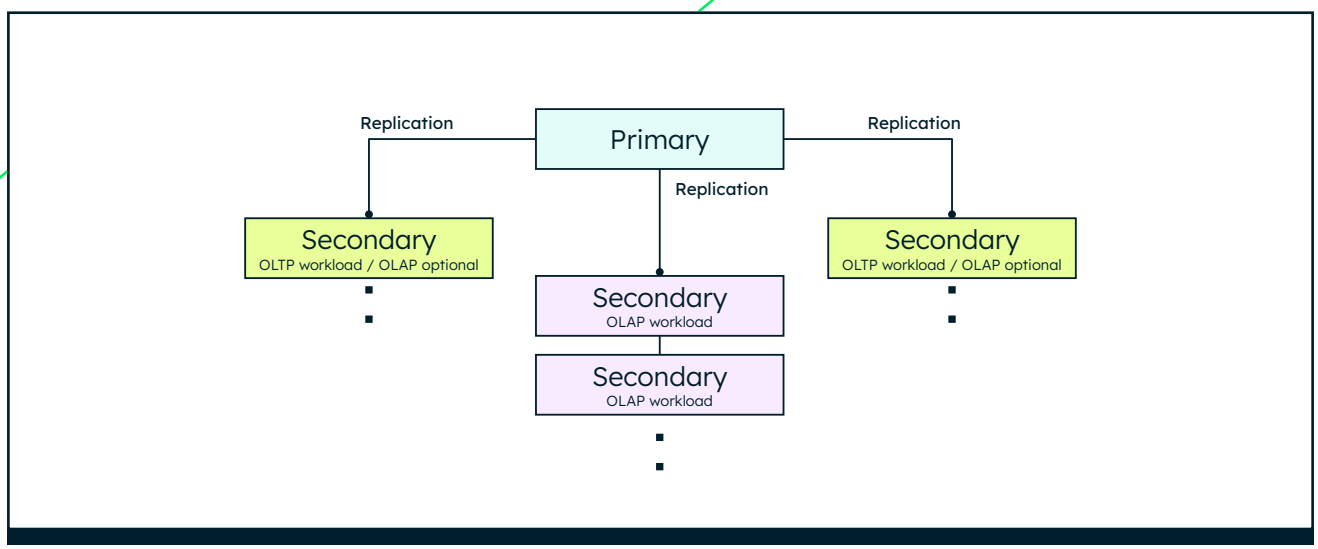

Figure 5. OLTP vs OLAP workload isolation diagram.

Workload isolation patterns in MongoDB

Replication

One common approach is to use replication. MongoDB replica sets allow for direct read operations from secondary nodes, preserving the primary node for write-heavy transactional operations. This is done using read preferences like secondaryPreferred. With this setup, analytical workloads such as reporting or aggregations can be offloaded to secondaries, while operational workloads remain isolated on the primary. Sharding can further enhance this pattern by segmenting data across multiple shards, based on data access patterns or domains.

Shard key-based isolation

Another pattern is shard key–based isolation. A shard key is crucial for distributing data across shards in a sharded cluster and plays a key role in query isolation. MongoDB sharding enables horizontal scalability by distributing data and load across nodes. Selecting the right shard key can significantly impact performance by allowing for efficient routing of queries to specific shards, minimizing the number of shards that need to be scanned. If shard keys are chosen to reflect usage profiles–such as region or workload type–this can ensure that OLTP and OLAP traffic is directed to different parts of the infrastructure. Using zone sharding, it's possible to map certain types of workloads to dedicated hardware, allowing for a clean separation between operational and analytical use cases.

This separation of workloads is not CQRS in the strict architectural sense, but it aligns with the same principle: isolating operational and analytical concerns to optimize for each. While CQRS typically refers to a software pattern that decouples read and write models at the application level, shard key–based isolation achieves a similar outcome at the infrastructure level. By mapping different traffic profiles–such as high-throughput writes versus complex analytical reads–to distinct zones or shard ranges, MongoDB allows systems to scale more effectively and avoid resource contention. In this way, infrastructure-level isolation can complement or even substitute for full CQRS implementations, especially when the underlying platform supports intelligent query routing and workload-aware data placement.

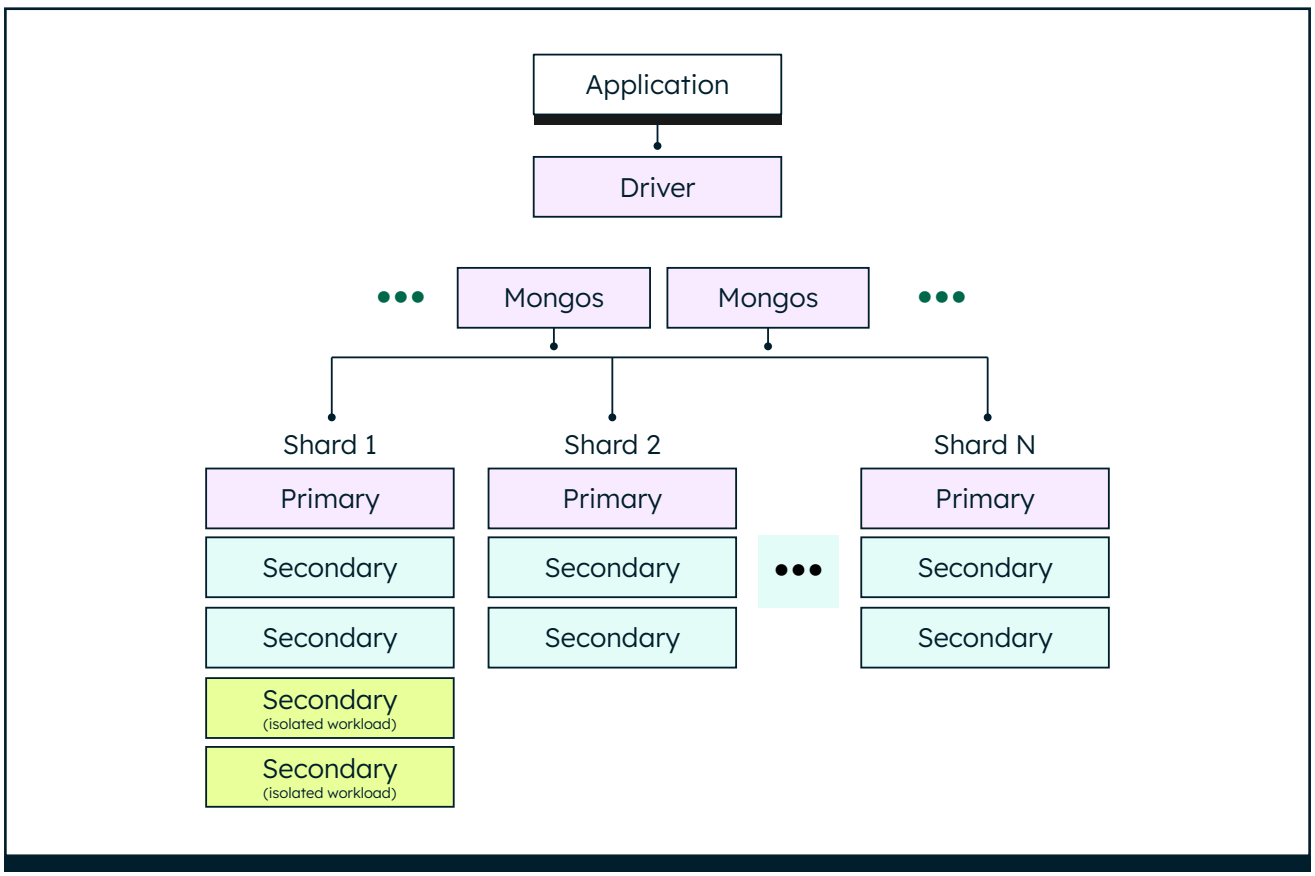

Figure 6. Sharded cluster.

Atlas Data Federation

MongoDB Atlas also provides Data Federation, which allows querying the data stored in multiple sources, including cloud object storage–like S3 bucket storage, without loading that data into MongoDB itself. This is useful when historical or low-access-frequency data needs to be queried without affecting the core operational database. For example, audit logs or user history can remain in cold storage, while being accessed in real time through federated queries, eliminating the need for duplication in the ODL.

Geographical workload segregation

Geographical workload segregation is another useful strategy. MongoDB supports region-aware routing, allowing queries to be directed to the nearest available node. OLTP workloads can be served from local infrastructure to ensure performance, while OLAP queries can be routed to centralized or lower-cost regions where they won't interfere with transactional performance. This is particularly effective for global applications that have different performance or compliance requirements in different geographies. In addition to performance benefits, geo-sharding also supports data sovereignty by enabling data to reside within specific jurisdictions–an increasingly common requirement in regulated industries and regions with strict data residency laws. (Visit: Cloud Providers and Regions.)

These workload isolation techniques not only improve performance and scalability, but also help reduce infrastructure costs by aligning resource usage with workload characteristics.

Unifying fragmented architectures with MongoDB

Modern architectures built on microservices and event-driven systems offer flexibility and scalability, but they also introduce fragmentation. Data ends up scattered across different services, each with its own persistence layer, making consistency, observability, and operational intelligence harder to maintain.

MongoDB helps address this by acting as a central aggregation and processing layer–a purpose-built operational data layer that reunifies data coming from different services, systems, and workloads.

In **event-driven environments**, MongoDB can act as a sink for Kafka or other event brokers, capturing transactional signals from across the system in real time. These events can be stored and made queryable for downstream applications, dashboards, or analytical processes.

In **microservices-based systems**, each service often manages its own data independently. MongoDB can serve as a common data access layer, where data from services like orders, inventory, or payments can be joined, searched, and analyzed together. Thanks to its flexible document model, MongoDB can support a variety of schemas without requiring rigid definitions or a central monolith.

Workload isolation also plays a role here. Since MongoDB supports both OLTP and OLAP patterns natively–through replica sets, sharding, read preferences, and Atlas Data Federation–teams can build an ODL that powers both transactional APIs and real-time analytics, without duplicating data or sacrificing performance.

In doing so, MongoDB becomes the operational backbone of distributed systems–unifying fragmented data into a consistent, scalable, and high-performance layer that supports both business operations and data-driven decision-making.

Data ingestion and modelling

MongoDB's document-oriented model supports constantly evolving schemas, making it ideally suited for an operational data layer. Its inherent flexibility enables iterative and agile development, ensuring that data structures remain aligned with dynamic application requirements and changing data integrations.

Schema design patterns

MongoDB provides a set of established schema design patterns developed in collaboration with customers across industries. These patterns serve as practical tools to help developers fully leverage the power of the document model. They offer a structured way to think in documents–bringing out the strengths of document-oriented storage by aligning data models with real-world application needs. Rather than relying on ad hoc modeling, developers can use these patterns to build scalable, efficient, and maintainable access to enterprise data in the ODL.

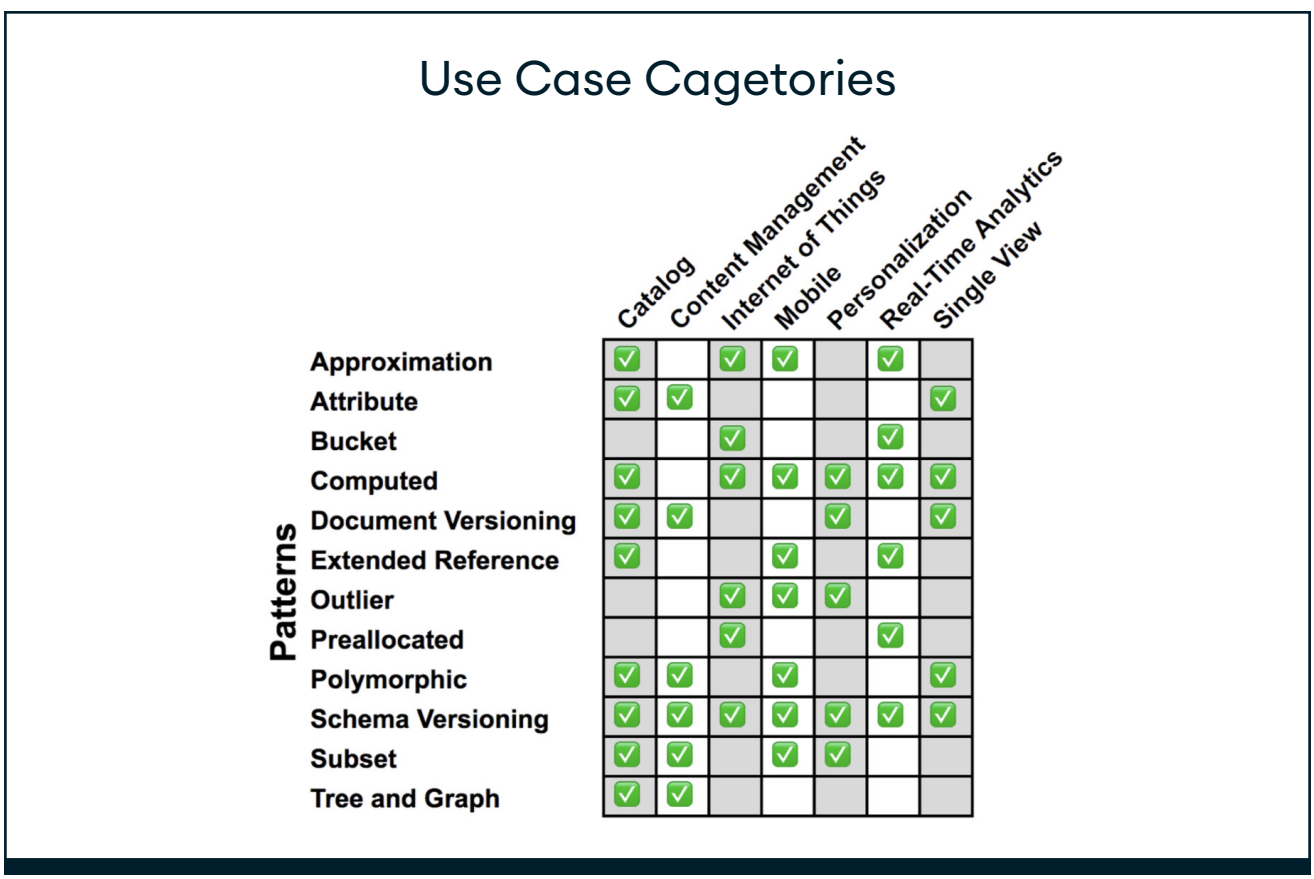

Figure 7. Schema design patterns.

Embedding vs. referencing

Relational database systems enforce rigid referencing through normalization rules and foreign key constraints to maintain structural integrity. MongoDB, however, provides the flexibility to choose between embedding and referencing on a case-by-case basis, facilitating a more nuanced approach to data modeling. Embedding is best when you need to access related data together frequently and efficiently, while referencing is better for managing large, complex datasets independently and minimizing data duplication.

Developers can embed data to enhance query performance and document clarity or employ referencing through MongoDB's native `_id` (primary key) attribute to manage complex relationships and maintain the distinctness of top-level entities. This flexibility enables precise optimization of data models, achieving an optimal balance between clarity, performance, and scalability.

To further dig into the art of data modelling, MongoDB provides a comprehensive set of learning material for further reading. Also, MongoDB has developed the Relational Migrator, which is a tool that helps automate the process of migrating from a relational, tabular database like Oracle or PostgreSQL to MongoDB.

Ingestion methods

A foundational element of any operational data layer is its ability to effectively ingest data from diverse sources. With data flowing from enterprise applications, APIs, IoT devices, event streams, and legacy systems, choosing the right ingestion method is critical to ensure data is processed, stored, and made available in a timely, scalable, and efficient manner. From batch processing to real-time streaming and hybrid approaches, modern data ingestion strategies are designed to support varied operational requirements and use cases.

Batch ingestion

Batch ingestion is a foundational technique used when data can be processed in scheduled intervals. Typically, this involves loading large volumes of data, such as hourly transaction logs or daily exports, into MongoDB from files or other databases. This approach is ideal for consolidating updates from systems that do not support real-time integration or for initial data loads.

MongoDB import tools (mongoimport)

One of the simplest tools for batch ingestion into MongoDB is mongoimport. This command-line utility supports importing JSON, CSV, or TSV files directly into a collection, making it especially useful for loading structured files during development or testing. For example, the following command loads a CSV file containing sales data into a MongoDB collection:

```
mongoimport --uri "mongodb+srv://USERNAME:PASSWORD@CLUSTER_URL" \
            --collection sales \
            --type csv \
            --file daily_sales.csv \
            --headerline
```

This command connects to a MongoDB deployment via the URI, specifies the file type, and uses the CSV headers to define field names.

While effective for small-scale use cases, mongoimport is typically not suitable for complex or large-scale pipelines.

Bulk write API for programmatic batch control

To support larger or more programmatically controlled workflows, MongoDB provides the bulk write API. This allows developers to insert multiple documents in a single operation, offering efficiency and control in scripted batch pipelines. Consider a Node.js application that programmatically inserts product data. Developers can construct an array of operations and execute them in bulk using the API, which significantly reduces the overhead compared to individual inserts.

```
const { MongoClient } = require("mongodb");
async function batchInsert() {
    const uri = "mongodb+srv://USERNAME:PASSWORD@CLUSTER_URL";
    const client = await MongoClient.connect(uri);
    const db = client.db("ecommerce");
    const collection = db.collection("sales");

    const data = [
        { item: "shirt", quantity: 10, price: 20 },
        { item: "pants", quantity: 15, price: 30 },
        { item: "shoes", quantity: 100, price: 50 }
    ];

    const operations = data.map(doc => ({ insertOne: { document: doc } }));
    await collection.bulkWrite(operations);

    console.log("Batch Data Inserted");
    client.close();
}
  batchInsert();
```

Beyond these tools, MongoDB also integrates with a broader ecosystem of platforms and utilities that facilitate batch ingestion. MongoDB Compass, our graphical user interface, provides guided import options suitable for non-technical users. For more advanced workflows, MongoDB Atlas Data Federation allows querying and materializing data from external sources like AWS S3. Developers can also build custom ingestion pipelines using native MongoDB drivers or leverage connectors available in ETL platforms like Apache NiFi, Talend, or Informatica.

ETL vs. ELT

When building data pipelines, organizations often adopt either ETL (extract, transform, load) or ELT (extract, load, transform) strategies, depending on their infrastructure and data processing needs.

In a traditional ETL pipeline, data is first extracted from one or more sources, transformed externally (cleaned, enriched, or reshaped) and then loaded into the destination system, such as MongoDB. This is useful when incoming data needs to be pre-processed before it is ready for operational use.

In contrast, ELT flips the transformation step. Data is extracted from its source, loaded directly into MongoDB in its raw form, and then transformed within MongoDB itself. This approach takes advantage of MongoDB's robust Aggregation Pipeline and integration with tools like Apache Airflow to handle transformations post-ingestion.

MongoDB supports both paradigms flexibly. In an ETL setup, developers can use platforms like Apache Nifi or Talend to clean and shape data before writing it to MongoDB. In an ELT context, connectors such as Airbyte, the MongoDB Kafka Connector, and the MongoDB Spark Connector enable fast loading of raw data into MongoDB, where downstream transformations can be performed either natively or through orchestration tools.

To illustrate this, consider a scenario using Apache Spark. A Spark job can read data from a CSV file, then write that data directly to a MongoDB collection using the Spark Connector:

```
from pyspark.sql import SparkSession

spark = SparkSession.builder\
    .appName("Example")\
    .config("spark.mongodb.output.uri",
"mongodb+srv://USERNAME:PASSWORD@CLUSTER_URL/ecommerce.sales")\
    .getOrCreate()

data = spark.read.csv("daily_sales.csv", header=True)
data.write.format("mongodb").mode("append").save()
```

This hybrid integration allows teams to use Spark for powerful data transformations while benefiting from MongoDB's flexibility as a storage engine.

Real-time ingestion and streaming

In operational workloads requiring immediate data access–such as fraud detection, personalization engines, or telemetry–real-time ingestion is critical. MongoDB integrates tightly with Apache Kafka to support event-driven architectures. Using the MongoDB Kafka Connector, developers can configure a Kafka sink that listens to a topic and writes streaming records directly into MongoDB collections with minimal latency.

A typical Kafka Sink Connector configuration looks like the following:

```
connector.class=com.mongodb.kafka.connect.MongoSinkConnector
topics=transactions
key.converter=org.apache.kafka.connect.storage.StringConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
connection.uri=mongodb+srv://USERNAME:PASSWORD@CLUSTER_URL
database=banking
collection=transactions
```

With this setup, transactional events from a payment platform, for example, are streamed in real time into MongoDB, enabling live processing and analytics downstream.

Beyond Kafka, MongoDB supports custom stream ingestion pipelines using code. Developers can implement a Node.js Kafka consumer to listen for messages on a topic–say, IoT sensor data–and insert them into a MongoDB collection as they arrive. This is useful for telemetry, event logging, or system monitoring.

```
const kafka = require('kafka-node');
const { MongoClient } = require('mongodb');

const consumer = new kafka.Consumer(client, [{ topic: "sensorData", partition: 0 }], {
autoCommit: true });

consumer.on('message', async function(message) {
    const mongoClient = await MongoClient.connect("mongodb+srv://USERNAME:PASSWORD@
CLUSTER_URL");
    const collection = mongoClient.db("iot").collection("readings");

    const sensorEvent = JSON.parse(message.value);
    await collection.insertOne(sensorEvent);

    console.log("Inserted Sensor Data:", sensorEvent);
    mongoClient.close();
});
```

This stream processing approach allows MongoDB to continuously receive and store data with near real-time responsiveness, making it ideal for high-throughput pipelines.

Change data capture

Change-data capture (CDC) is a powerful technique used to monitor and track data changes–such as inserts, updates, and deletes–in a source database and propagate those changes to downstream systems in real time. Depending on the implementation context, CDC can serve either as a real-time ingestion method or as part of a broader stream processing architecture.

When used for real-time ingestion, CDC enables immediate propagation of data changes to support time-sensitive workflows. For example, in banking, CDC can power fraud detection mechanisms by instantly flagging suspicious transaction patterns as they occur. Similarly, in retail environments, CDC can help maintain up-to-date inventory records across distributed systems by synchronizing stock levels in real time.

In stream processing scenarios, CDC feeds these change events into platforms like Apache Kafka, enabling continuous transformation, enrichment, or aggregation before further consumption. This is particularly useful in use cases such as ingesting IoT telemetry, where sensor updates are reflected in MongoDB collections with minimal latency, or monitoring telecom networks, where real-time analysis of service events can uncover and address outages as they happen.

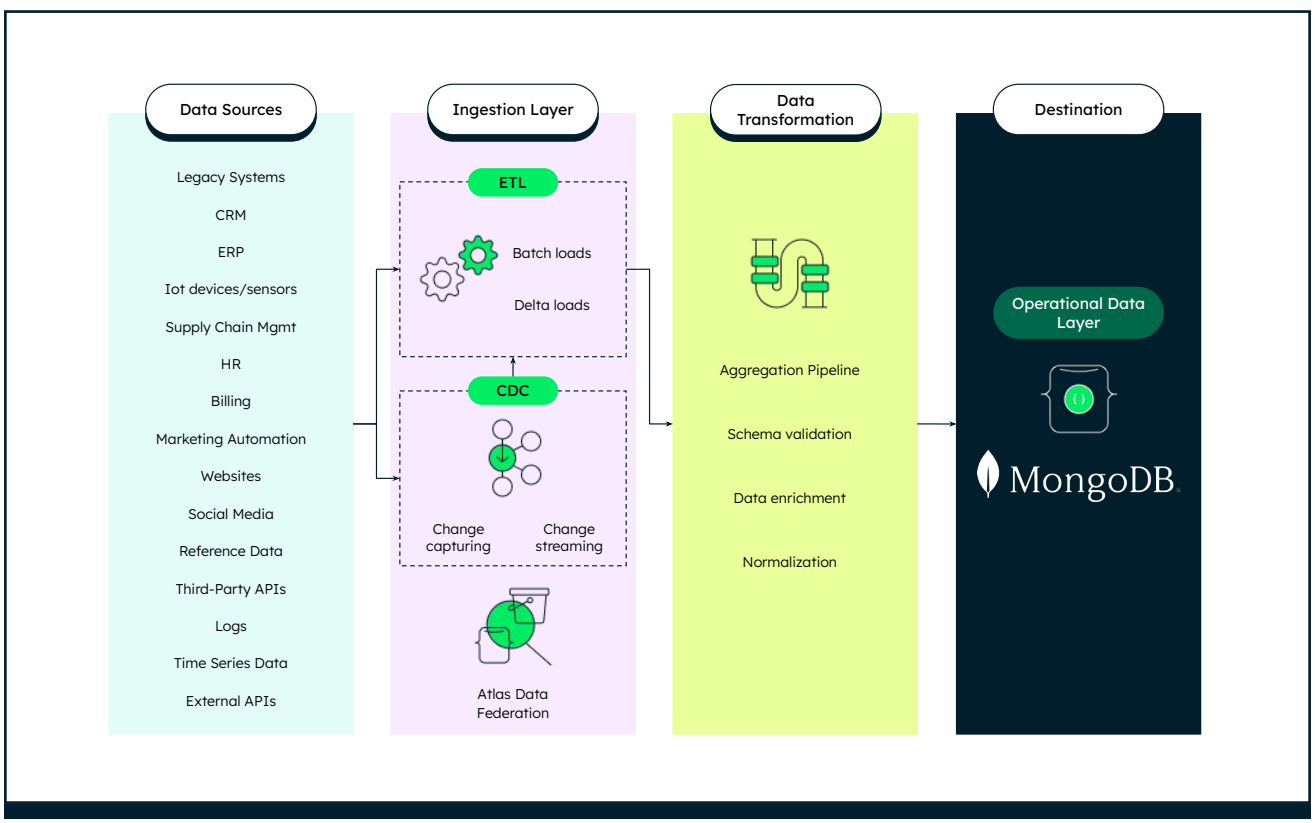

Figure 8. Ingestion and modeling flow.

MongoDB partnerships for CDC in mobile applications

MongoDB also extends CDC capabilities to mobile and edge environments through partnerships with PowerSync, Ditto, and Ably, providing offline-first and real-time sync functionality for distributed apps.

PowerSync allows developers to sync only the most relevant data changes to mobile clients, reducing bandwidth utilization and ensuring timely updates. In a delivery app, for instance, drivers can receive new orders in real time, directly from the operational MongoDB backend.

Ditto enables offline data capture and peer-to-peer sync, allowing mobile users to work even without connectivity. Once the device is back online, changes are reconciled automatically with MongoDB, making it ideal for collaborative or field apps.

Ably supports global real-time pub/sub messaging, helping propagate MongoDB updates to client devices at scale. A ride-sharing app, for example, might use Ably to notify riders and drivers of trip status changes instantaneously.

Together, these integrations allow MongoDB to serve as a real-time backend for both cloud-native and edge-native applications, providing seamless data sync, high performance, and resilient offline capabilities.

Access layer architecture

As discussed in the previous sections, once data is unified and governed within the ODL, the question becomes how to make it securely and efficiently available to consumers–from internal services and analytics engines to customer-facing applications and AI models. This is where the access layer comes into play.

The access layer is not a monolithic gateway but a structured architectural tier responsible for brokering data access across domains while enforcing organizational policies. It acts as the outer boundary of the ODL, shaping how internal data surfaces to the outside world under strict security controls, performance guarantees, and multi-modal interface support.

A well-architected access layer must fulfill four essential requirements. First, it must support protocol agnosticism, enabling flexibility across REST, GraphQL, gRPC, and WebSockets. Second, security must be intrinsic: zero-trust principles, identity-based policies, and cryptographic safeguards must be enforced at every point of entry and communication. Third, performance isolation is crucial–operational and analytical workloads must coexist without interference. And finally, the system must be evolution-proof, with versioned APIs and adaptable schema management to avoid breaking downstream integrations.

Layered architecture and access routing

To meet these goals, the access layer is logically composed of three integrated tiers: an API gateway, a service mesh, and a data proxy layer. Each tier plays a distinct role in brokering access between data producers and consumers.

At the outermost boundary, the **API gateway** serves as the primary enforcement point. It handles authentication, rate limiting, routing, and protocol translation. For ODL environments, both Kong and Traefik are well suited options. Kong offers dynamic plugin support and proven high throughput, while Traefik provides a lightweight, middleware-driven architecture with native service discovery and seamless integration into modern containerized environments. In either case, the gateway inspects tokens, enforces per-tenant quotas, and emits detailed telemetry to observability platforms, all while shielding internal systems from direct exposure.

Within the system boundary, the **service mesh** ensures that microservices communicate securely and reliably. It enforces mutual TLS using SPIFFE identities, applies retry logic and circuit breaking, and propagates tracing headers consistently. Mesh frameworks such as Istio and Linkerd allow administrators to apply fine-grained

traffic policies and observability across every service call, even in fast-moving environments.

The innermost **data proxy layer** mediates access to the underlying databases. It implements intelligent connection pooling, request caching, and workload-aware routing. This is especially important in hybrid environments where OLTP, OLAP, and RAG-based workloads share infrastructure but demand different response times and consistency guarantees.

Access patterns by consumer type

OLTP applications, such as fraud detection or customer portals, require single-digit millisecond response times. Queries are routed to the primary node of a MongoDB replica set for consistent reads and writes, with optional caching at the application layer or via `mongos` in sharded clusters.

OLAP and BI applications process large datasets for aggregations and time-series analysis. Queries are routed to read-only secondary nodes using MongoDB's `secondaryPreferred` read preference, leveraging the aggregation framework for complex analytics. Atlas Data Federation or `$out/$merge` pipelines can integrate external data sources, with proper indexing ensuring performance without impacting OLTP workloads.

AI applications, including RAG and agentic interfaces, rely on semantically enriched data. Queries are routed to MongoDB Atlas Vector Search indexes using `$vectorSearch` for semantic scoring, with embeddings generated by external models (e.g., OpenAI) or Voyage AI, included and automatically managed in MongoDB Atlas since Q3 2025. Hybrid search combines vector and keyword queries for precision, while `$lookup` or application-level merging integrates operational metadata, with careful synchronization ensuring near-real-time data consistency.

The following example code demonstrates role-based query routing for OLTP, OLAP, and AI/RAG workloads in MongoDB, using Flask and JWT authentication to direct requests to primary nodes, secondary nodes with aggregations, or vector search indexes, respectively.

```
@app.route('/entities', methods=['GET'])
@jwt_required()
def get_entities():
    role = get_jwt_identity().get("role", "default")
    if role == "analyst":
        data = list(db.documents.aggregate([...],
                read_preference=ReadPreference.SECONDARY_PREFERRED)) or [{}]
        return jsonify({"source": "OLAP", "data": data})
    elif role == "ai-agent":
        prompt = request.args.get("prompt", "...")
        data = list(db.documents.aggregate([{"$vectorSearch": {...}}])) or [{}]
        return jsonify({"source": "AI", "data": data, "prompt": prompt})
    else:
       data = db.documents.find_one({"status": ...}) or {}
       return jsonify({"source": "OLTP", "data": data})
```

Data change streams

MongoDB Change Streams enable event-driven architectures by capturing real-time changes to documents in collections (such as inserts, updates, and deletes) without the need for polling. This makes it easy for developers to build responsive applications that can react to data changes dynamically, such as triggering workflows, updating caches, or detecting anomalies. Here's an example that listens for real-time changes in a transactions collection:

```
const { MongoClient } = require("mongodb");
async function watchChanges() {
    const uri = "mongodb+srv://USERNAME:PASSWORD@CLUSTER_URL";
    const client = await MongoClient.connect(uri);
    const collection = client.db("banking").collection("transactions");

    const changeStream = collection.watch();

    changeStream.on("change", (change) => {
        console.log("Next Change:", change);
        // Process real-time event (e.g., flag fraudulent activity)
    });
}
  watchChanges();
```

Encryption and certificate management

TLS must be enforced for all external access points. Many organizations use Let's Encrypt with the ACME protocol to automatically provision and renew certificates.

For regulated environments or where higher assurance levels are required, DigiCert offers a commercial alternative to Let's Encrypt, providing extended validation (EV) certificates, managed PKI services, and detailed compliance reporting. Organizations may adopt a hybrid strategy using Let's Encrypt for internal or non-critical services and DigiCert for customer-facing or compliance-sensitive APIs.

Security enforcement and data trust

Large enterprises should implement a layered approach to security in the access layer, extending beyond TLS and authentication. Tools like Open Policy Agent (OPA) can provide dynamic, policy-based access control at the API level. Keycloak offers a robust solution for managing identities, issuing JWTs and embedding role-based claims.

HashiCorp Vault can securely manage secrets and inject credentials into runtime environments.

For enhanced security, enterprises may consider integrating Thales CipherTrust or Luna HSMs to offload cryptographic operations and store root keys in tamper-proof hardware. These tools can support compliance with data sovereignty and key management regulations, particularly in finance, telecom, and public sector applications. Together, such solutions help enforce least privilege and enable auditable access, aligning with regulatory standards.

A foundation for scalable, secure access

When implemented correctly, the access layer enables differentiated service levels across consumer types without duplicating data or increasing risk. It allows organizations to decouple their backend systems from consumer-facing APIs, introduce caching and query acceleration, enforce compliance, and scale independently.

More than a routing layer, this architecture becomes the execution surface for enterprise-wide policy governing not just who accesses which data but how, when, and under what guarantees.

Business Domains

Everything discussed so far applies broadly across major industries. In the following sections, we will shed light on the specific necessities, challenges, usage scenarios, and benefits associated with implementing an Operational Data Layer. By examining real-world demands and pain points, we aim to clarify how an ODL can unlock value, streamline data access, and support modern digital initiatives across diverse enterprise landscapes.

Financial services

The financial services industry is undergoing rapid transformation, driven by regulatory pressures, technological innovation, and evolving customer expectations. From payments modernization to AI-driven transformation, financial institutions face diverse operational challenges requiring agile, scalable, and intelligent data infrastructure.

Payments modernization

Payments sit at the core of digital finance, yet many institutions are constrained by fragmented systems and outdated infrastructure. Integrating modern payment rails–ACH, SEPA, RTP, SWIFT, blockchain protocols–requires a unified, high-performance data layer.

MongoDB serves as the operational data layer for a "payments data hub," enabling financial institutions to consolidate data from various payment rails into a unified, real-time store. This payment ODL provides a platform for critical workflows such as payment validations, verification of payee (VoP), and reconciliation across multiple systems–enhancing operational efficiency and reducing error rates.

Key capabilities of MongoDB make it the perfect fit for payments modernization:

- **Search and matching:** Advanced search capabilities, including fuzzy, semantic, and hybrid searches, help validate and verify payee information, flag duplicate entries, and rapidly match transaction details across datasets.
- **Scalability:** MongoDB scales effortlessly to support high transaction volumes, ensuring uninterrupted service even during peak payment periods, such as holidays or promotional campaigns.
- **Multi-modal data handling:** MongoDB's support for unstructured, semi-structured, and structured data allows institutions to manage diverse payment data formats without transformation.
- **Speed:** Real-time performance ensures payments are processed, validated, and reconciled in milliseconds–ideal for enabling frictionless user experiences.

In addition to streamlining payments infrastructure, MongoDB provides critical security features to protect sensitive payment data, such as encryption at rest and in transit, tokenization for sensitive fields, and fine-grained access controls. By implementing MongoDB as the ODL, financial institutions reposition themselves as agile, payment-first innovators ready to meet evolving consumer and regulatory demands.

AI-driven financial services transformation

AI is reshaping financial services with personalized banking, conversational agents, dynamic portfolio optimization, and more. Delivering these capabilities at scale requires a data platform that's fast, flexible, and built for intelligent automation.

MongoDB underpins these AI-powered solutions by providing the robust operational data layer necessary to support high-speed data ingestion, retrieval, and analysis. Key use cases include:

AI-personalized digital banking

MongoDB enables intelligent systems such as virtual financial assistants, chatbots, and recommendation engines that deliver hyper-personalized customer experiences. Features like Atlas Vector Search empower these systems to decipher intent, retrieve relevant insights, and provide actionable recommendations based on real-time customer profiles.

- Retrieval-augmented generation (RAG)**:** By vectorizing and indexing financial data, MongoDB powers RAG workflows that enable generative AI systems to synthesize answers based on meaning-driven (semantic) search patterns. For example, RAG can assist customers in understanding mortgage terms, optimizing savings plans, or finding relevant policy documents. See figure 9:

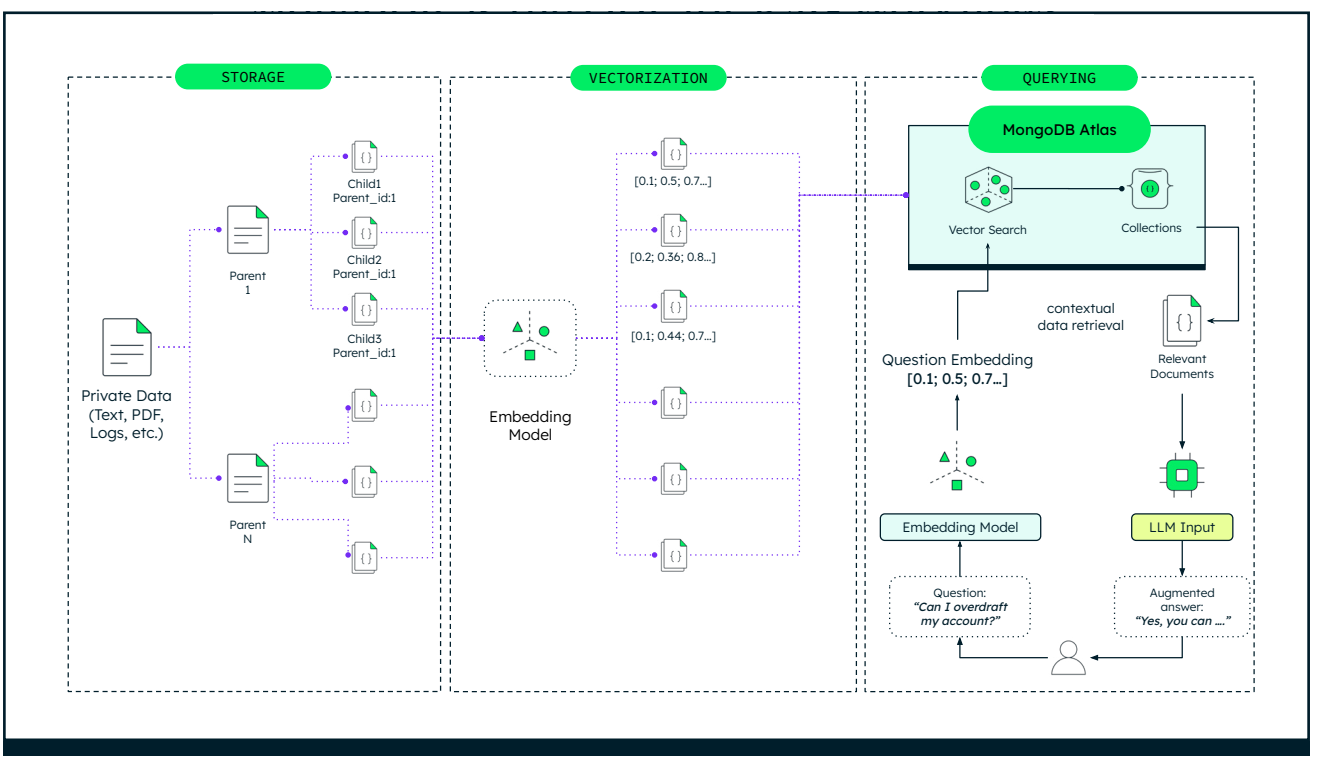

Figure 9. MongoDB as the ODL for RAG Workflows.

- Agentic AI-powered investment portfolio management: AI agents use MongoDB's scalable architecture for predictive modeling, risk analysis, and investment strategy optimization. With real-time data integration and semantic search capabilities, portfolio managers can adjust allocations dynamically and respond to market changes with greater precision. See figure 10:

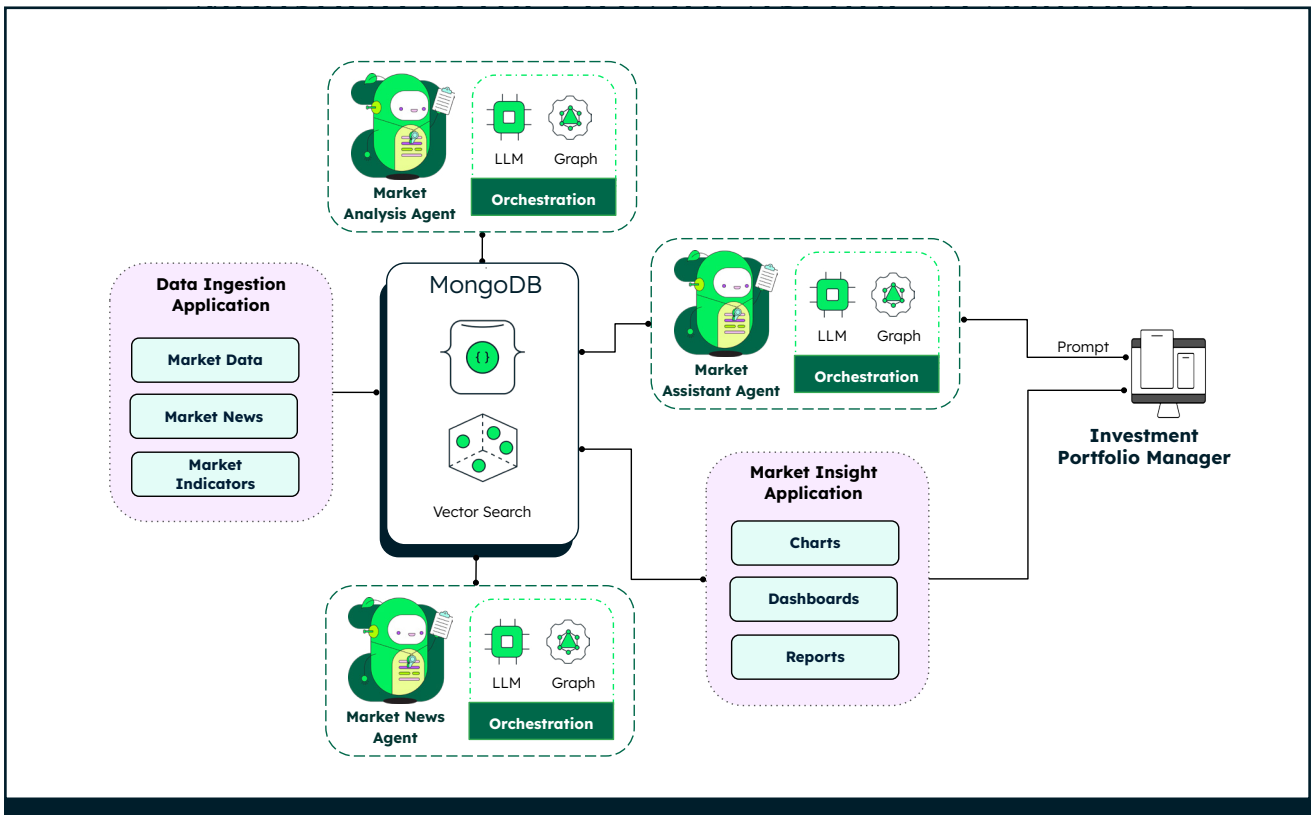

Figure 10. MongoDB as the ODL for Agentic AI Solutions.

MongoDB's ability to integrate structured and unstructured data gives AI applications access to rich, contextualized datasets. That means more accurate models, faster insights, and better customer outcomes, all within a secure, compliant environment.

Open finance

Open finance is transforming the financial landscape by creating opportunities for innovation and competition. It allows third-party financial service providers to access and use consumer financial data from banks and other financial institutions through APIs. This enables the delivery of hyper-personalized experiences and advanced services, such as personal finance management (PFM) apps and account aggregation. However, this also introduces new challenges in security, compliance, and interoperability that organizations need to address to stay ahead.

To overcome these challenges, MongoDB Atlas functions as the operational data store (ODS) for open finance, offering a flexible, scalable and secure data platform. Its document model allows the integration of diverse data types and data sources, natively storing data in a binary representation called BSON (binary JSON) that is compatible with JSON, making it ideal for open finance applications. MongoDB supports a wide range of extensions and connectors to create RESTful API development, empowering developers with an easy-to-use and intuitive data platform that boosts productivity and performance. MongoDB Atlas also ensures enterprise-grade security with built-in encryption, role-based access controls, and auditing to comply with regulations such as GDPR, PSD2 and FiDA.

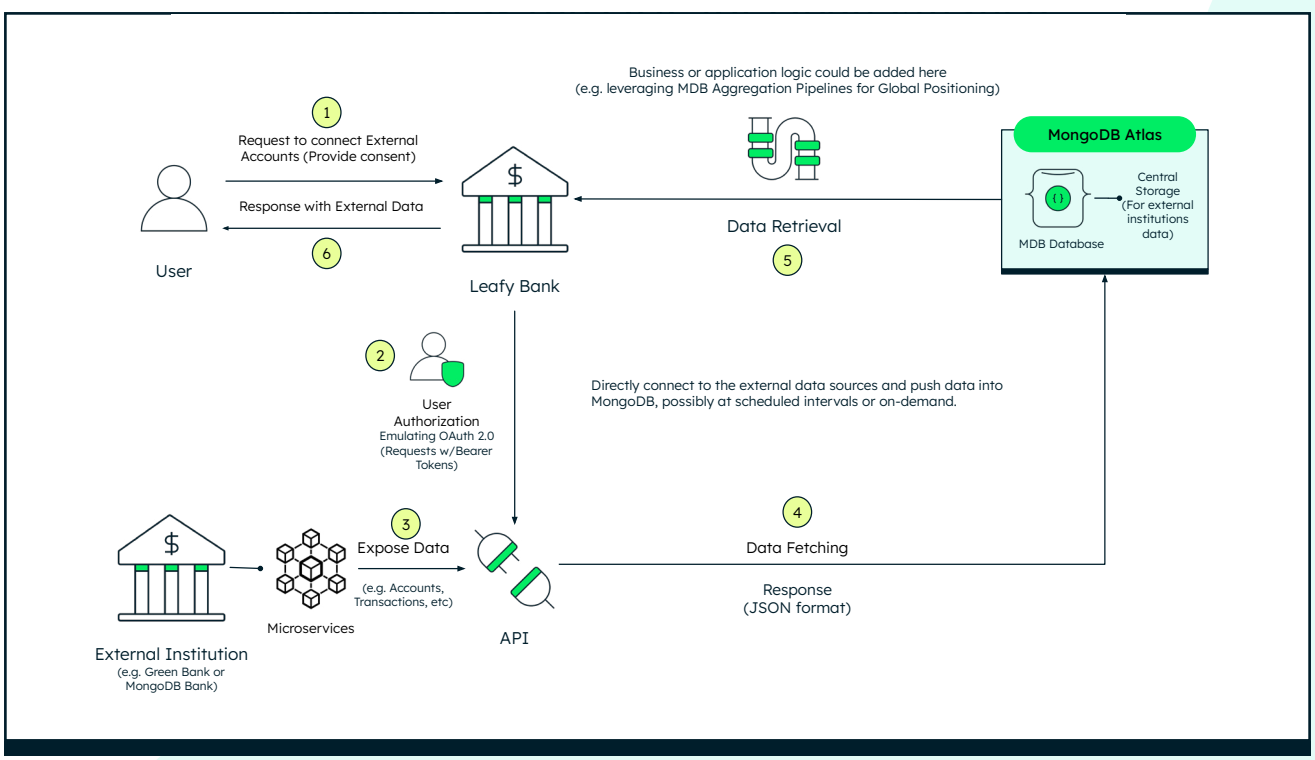

Figure 11. MongoDB as the ODL for Open Finance.

Financial crime mitigation and compliance

Fraud prevention and compliance are mission-critical but increasingly difficult to manage as transaction volumes rise and capabilities like AML and KYC become more critical. Traditional systems struggle to scale, adapt, and respond to threats in real time.

MongoDB helps financial institutions modernize their fraud and compliance operations with a unified data architecture. Using MongoDB's flexible document model, teams can consolidate customer profiles, transactions, and third-party data into a single, queryable view, reducing blind spots across detection workflows. Features like change streams enable real-time monitoring, allowing institutions to detect suspicious activity as it happens and accelerate incident response.

Advanced workloads are also supported through MongoDB Atlas Vector Search, which enhances AI-powered detection models by surfacing subtle anomalies and improving precision. Built-in enterprise security, including encryption, role-based access controls, and audit logging, ensures alignment with standards like PCI DSS, GDPR, CCPA, and PSD2.

With MongoDB as the ODL, financial organizations can build adaptive fraud ecosystems that minimize risk, reduce false positives, and support regulatory compliance.

Retail

Retail is one of the fastest moving industries and a fast adopter of new technology. Today's retailers are pushed to innovate in a competitive market with slim margins to find new ways to attract customers and keep their costs down. Fragmented systems and data silos often prevent retailers from reaching their full potential and understanding their customers and their businesses.

An ODL is often implemented to turn these challenges into opportunities by unlocking data to be used across the organization and bridging between disparate ecommerce and in-store technology estates. Retailers look to combine data to get the full view of their customers and their businesses through customer 360 and unified commerce transformations.

On the other side of the business, retailers struggle to combine data across a disparate supply chain and ERP estate. Traditional ERP technologies were not designed for the complexity of modern retail–for example omnichannel ordering or buy online, pick up in store. They also weren't designed to handle the peak seasonal traffic as the ecommerce world has expanded and cannot keep up with the performance expectations of the modern customer. Acting as a real-time data engine, an ODL delivers agility for fast development of new services while maintaining the stability of core systems. By breaking down silos, integrating speed-layer innovation, and ensuring operational reliability, an ODL empowers retailers to drive growth, enhance customer loyalty, and stay agile in a rapidly evolving marketplace.

Unified commerce

Today's shoppers demand fluid and connected experiences. Retail success depends on delivering frictionless, omnichannel journeys where every customer interaction–whether online, in-store, or mobile–is seamlessly integrated. Unified commerce is a transformative business strategy that centralizes sales channels, inventory, customer data, and fulfillment systems into a single platform. This integration unlocks real-time visibility and enables fluid transitions across touchpoints while delivering key benefits:

- **Seamless customer experience:** By connecting all channels, businesses can offer consistent and personalized shopping experiences.
- Improved efficiency: A centralized platform streamlines operations, reduces errors, and optimizes workflows.
- **Better data insights:** Unified commerce provides comprehensive visibility into customer behavior and business performance, supporting data-driven decisions.
- **Enhanced personalization:** A unified view of customer data allows businesses to tailor marketing efforts, product recommendations, and customer interactions with precision.
- **Streamlined inventory management:** Accurate inventory tracking across channels prevents stockouts and improves fulfillment accuracy at scale.

Implementing unified commerce effectively hinges on a robust operational data layer (ODL). An ODL bridges fragmented systems and powers the real-time capabilities retailers need, such as dynamic pricing, live inventory tracking, and intelligent order orchestration. It provides the agility to modernize legacy architectures while ensuring stability in core operations. By enabling scalability, the ODL supports peak performance during high-demand periods and accelerates innovation, allowing retailers to swiftly adapt to evolving customer expectations and market conditions. Unified commerce, powered by an ODL, is more than a strategy. It is the foundation for delivering exceptional experiences, driving growth, and achieving sustained success in the modern retail landscape.

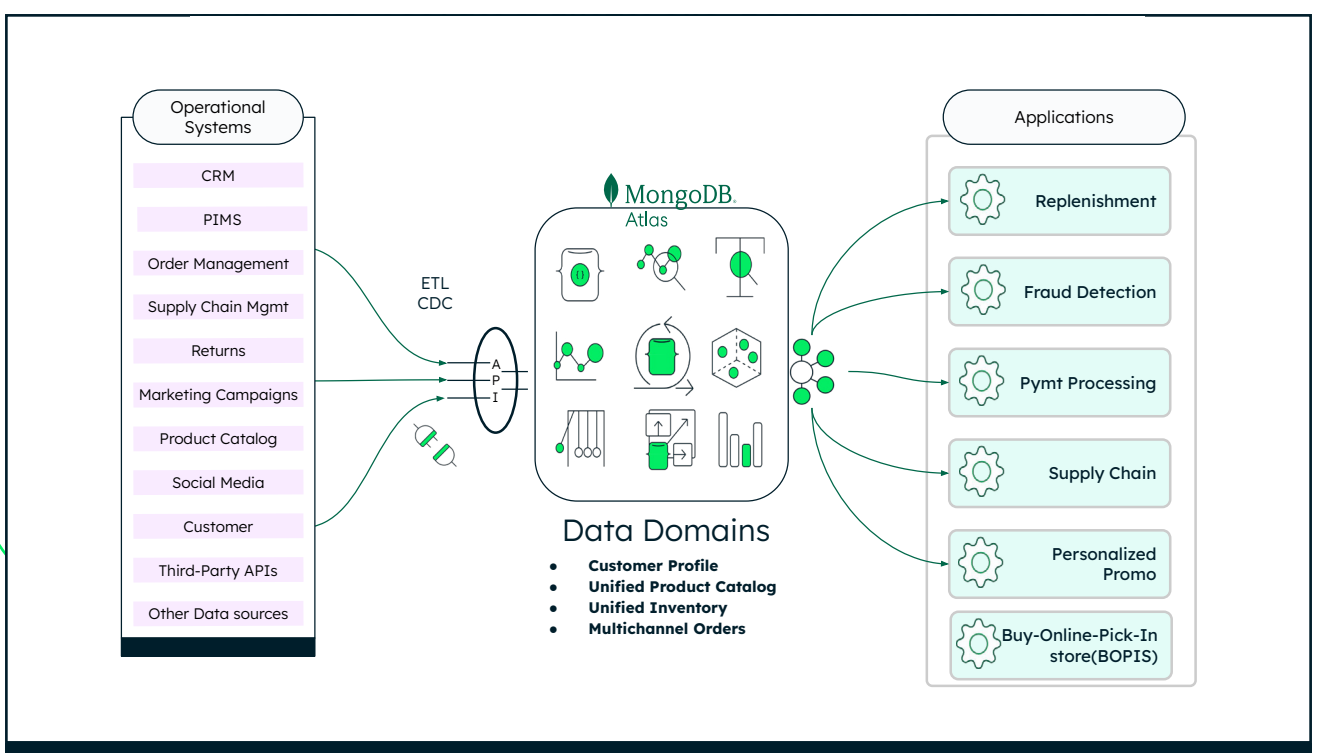

Figure 12. Domain-driven design for unified commerce.

Customer data platform 360 (single) view

Unified commerce helps retailers connect systems and channels, creating consistent experiences across all customer touchpoints. But to deliver truly meaningful and personalized interactions, retailers need more than just connected systems. They need a complete, real-time view of their customers.

This is where customer 360, or single view, comes in. It unifies data from e-commerce purchases, in-store sales, CRM platforms, and more into one accurate and governed profile. Without this real-time, complete understanding of customer behavior and preferences, personalized marketing and strong customer relationships are nearly impossible.

With a unified customer-360 profile, retailers can offer personalized promotions based on the full customer journey. Support teams gain visibility into past interactions, helping them resolve issues faster and improve satisfaction. This leads to higher loyalty, retention, and better campaign results.

An operational data layer is essential to making this happen. It consolidates data from multiple sources–CRM, POS, marketing systems, and external data–while handling identity matching, deduplication, and compliance. The ODL delivers a reliable, real-time single view that powers smarter targeting, better decision-making, and faster adaptation to changing customer needs.

By breaking down data silos and enabling seamless access to unified data, the ODL drives the personalized experiences and intelligent operations retailers need to stay competitive. In today's market, Customer 360 isn't just a goal–it's a critical foundation for growth and success.

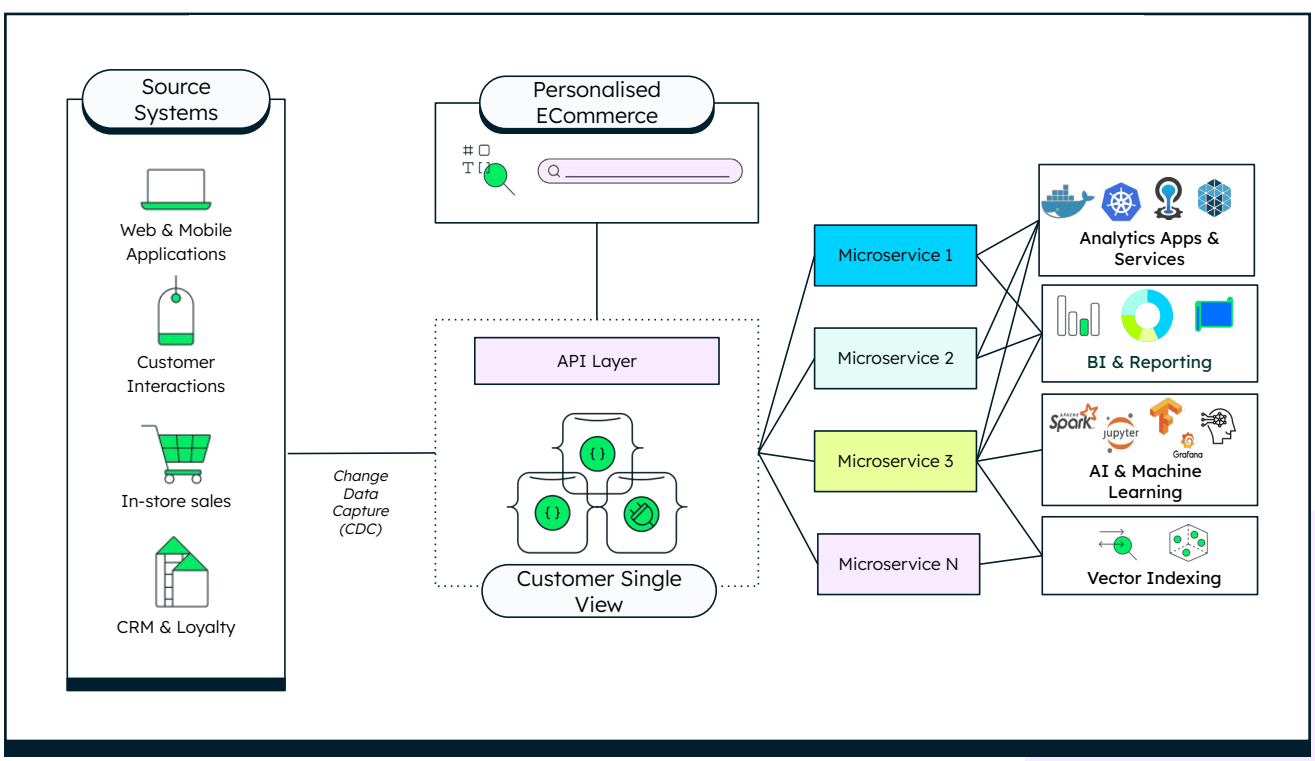

Figure 13. Customer data platform 360 (single) view diagram.

ODL speedlayer to innovate over ERP

Modern retail moves at lightning speed, from inventory shifts and pricing changes to order fulfillment and fraud detection. Yet many retailers are constrained by monolithic enterprise systems such as an SAP ERP, which, while robust and reliable, often hinder rapid development due to their complexity, cost, and rigidity. To overcome this, building an operational data layer–often referred to as a speed layer–on MongoDB offers a pragmatic and strategic solution.

An ODL complements ERP core services by acting as a high-performance data intermediary that interacts with SAP without disrupting its stability. This approach allows the core ERP to remain the system of record while the ODL becomes the system of engagement where customer-facing applications, digital services, and operational dashboards can be developed and deployed rapidly. By decoupling front-end innovation from backend constraints, the speed layer allows retailers to respond rapidly to changing customer demands, whether it's peak-season surges or real-time restocking.

The speed layer enables low-latency access to critical data aggregated from SAP and other systems, reducing the need for costly and slow queries directly on the ERP backend. This not only enhances application responsiveness but also shields the core from performance degradation under load.

By centralizing operational data, retailers can also enable advanced analytics, AI/ML initiatives, and omnichannel experiences without overloading SAP. In today's modern retail world as AI and generative AI need real-time access to data, building an AI-capable layer to execute modern workflows is a necessity.

With an ODL powering the speed layer, retailers can accelerate innovation while maintaining core system stability, delivering agility at the edge without disrupting critical operations. This balance positions retailers to scale efficiently, stay competitive, and offer both operational excellence and next-gen customer experiences.

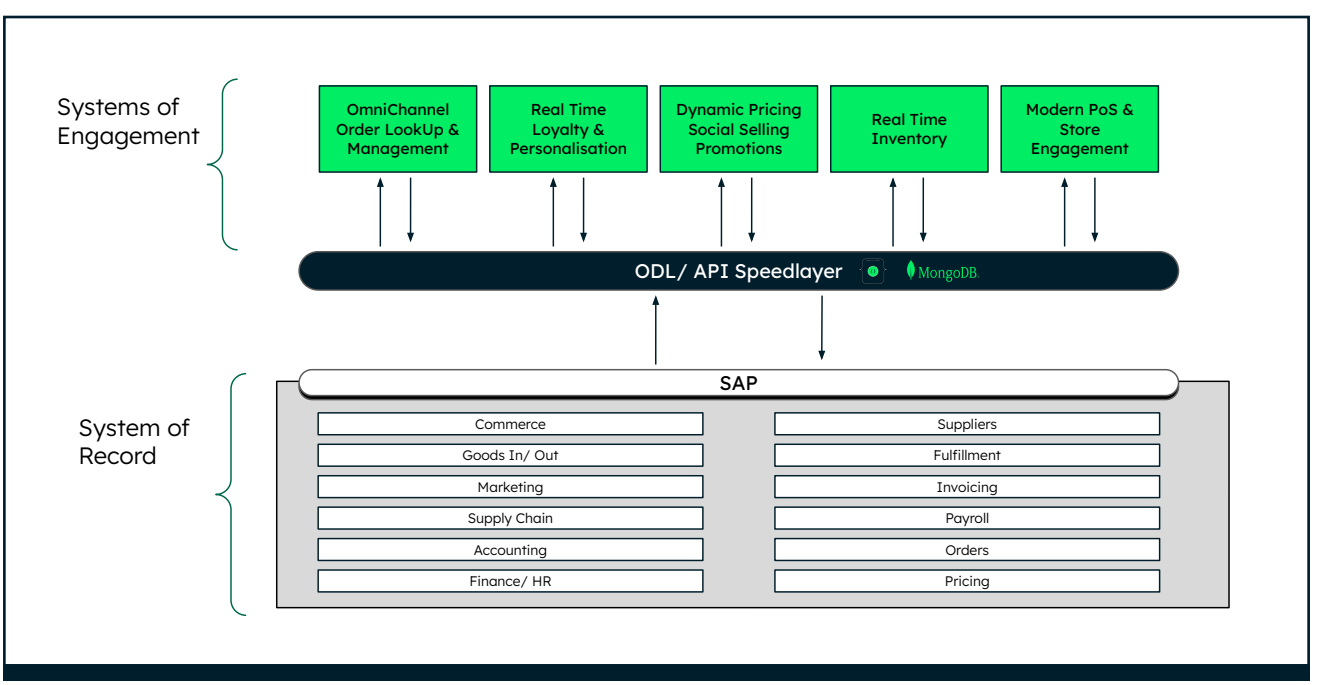

Figure 14. MongoDB as a speed layer to innovate over ERP systems diagram.

Telco

Telecom is no longer just about connectivity. It's about enabling intelligent, real-time services at a massive scale. As networks modernize, telcos must handle fast-changing data across billions of endpoints, support radically distributed architectures, and consistently deliver low-latency responses. An operational data layer provides the foundation for this shift, acting as a real-time, governed data backbone that bridges legacy complexity with next-generation demands.

5G and edge computing

5G brings the core capabilities for transforming enhanced mobile broadband (eMBB) for data-heavy apps, massive Machine-Type Communication (mMTC) for dense IoT deployments, and ultra-Reliable Low Latency Communication (uRLLC) for mission-critical interactions. But these require not just network speed. They demand real-time access to the right data. The ODL, powered by MongoDB, enables this by centralizing operational data, decoupling service logic from backend systems, and supporting the dynamic, scalable access patterns modern apps need. This architectural separation allows telcos to launch and evolve services faster while meeting strict SLAs for performance, reliability, and resilience.

Edge computing extends that vision. Data from connected devices is increasingly processed at the edge, close to users, factories, vehicles, or infrastructure where latency matters most.

The ODL acts as the control plane across this distributed topology. It aggregates data locally, syncing with core systems intelligently and applying governance policies consistently. This unlocks the full potential of telco-led IoT ecosystems. From private 5G in smart factories, to autonomous driving, smart cities, home automation, and remote healthcare, telcos are no longer just transport providers, but real-time data platforms. With the ODL in place, IoT data becomes accessible, actionable, and secure, powering use cases that would be impossible with rigid, centralized architectures.

Agentic AI for network assurance and customer care

Another emerging application of the ODL in telecommunications lies in supporting agentic AI. By bringing together customer data across all touchpoints–support interactions, complaints, network usage, and retail activity such as purchases or rate plan adjustments–the ODL enables intelligent, context-aware querying. Built on MongoDB, it allows AI agents to operate on a unified, real-time view of the customer, powered by advanced aggregations and flexible pipelines. This not only enhances responsiveness in customer-facing scenarios but also creates new operational value. Customer data from business support systems (BSS) can now be correlated with operational support system (OSS) telemetry, helping to identify root causes of network issues with unprecedented clarity and speed.

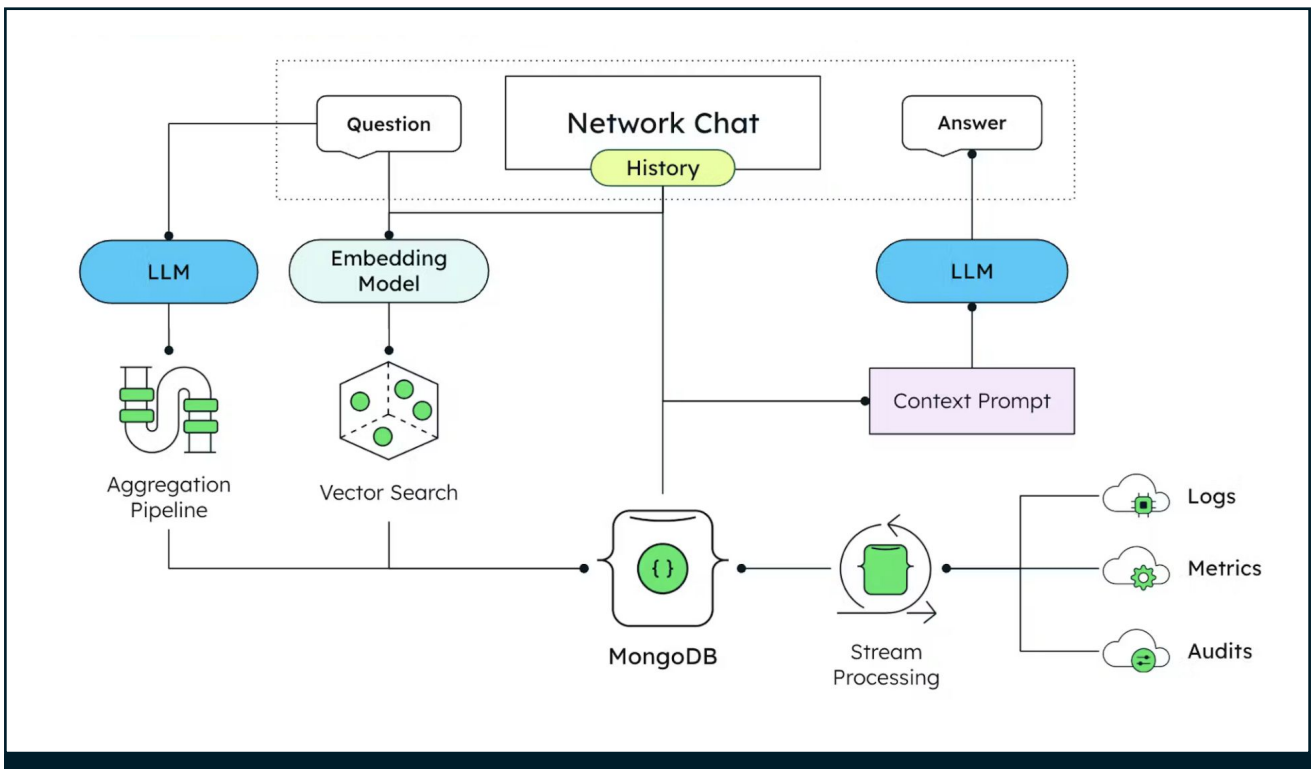

Figure 15. MongoDB as the ODL component of a network chat application

This shift also opens the door to entirely new monetization strategies. As telcos evolve into data service platforms, the value isn't just in connectivity, it's in enabling ecosystems. The ODL makes it possible to expose curated, policy-compliant data products to partners, enterprises, and developers via secure, well-governed APIs. Smart-city operators can access real-time infrastructure data, automotive OEMs can tap into anonymized mobility insights, and industrial vendors can build on top of private 5G environments. With versioned APIs, usage metering, and access controls, telcos can turn their data assets into subscription-based offerings without exposing internal systems or violating sovereignty. The result is faster innovation, shared value creation, and a clear path from infrastructure investment to revenue growth.

Manufacturing

Unified namespace

A unified namespace (UNS) can streamline operations by enabling seamless data flow between operational technology (OT) and information technology (IT) systems, supporting real-time decision-making and process optimization.

A UNS acts as a centralized, contextualized data repository, fostering communication across industrial systems and eliminating the inefficiencies of traditional point-to-point data exchanges, such as data silos and integration complexity. By incorporating MongoDB as an operational data layer within a UNS, manufacturers can aggregate structured and unstructured data from diverse sources like IoT devices, MES, SCADA, and ERP systems. MongoDB's flexible document model supports real-time access, scalability, and robust analytics. Features such as Atlas Stream Processing, Change Streams, and the Aggregation Framework enable event-driven architectures (see above), enhancing responsiveness in industrial operations. Additionally, MongoDB's support for time series data, metadata storage, and real-time querying facilitates manufacturing intelligence, predictive maintenance, and process optimization within a unified, scalable data architecture.

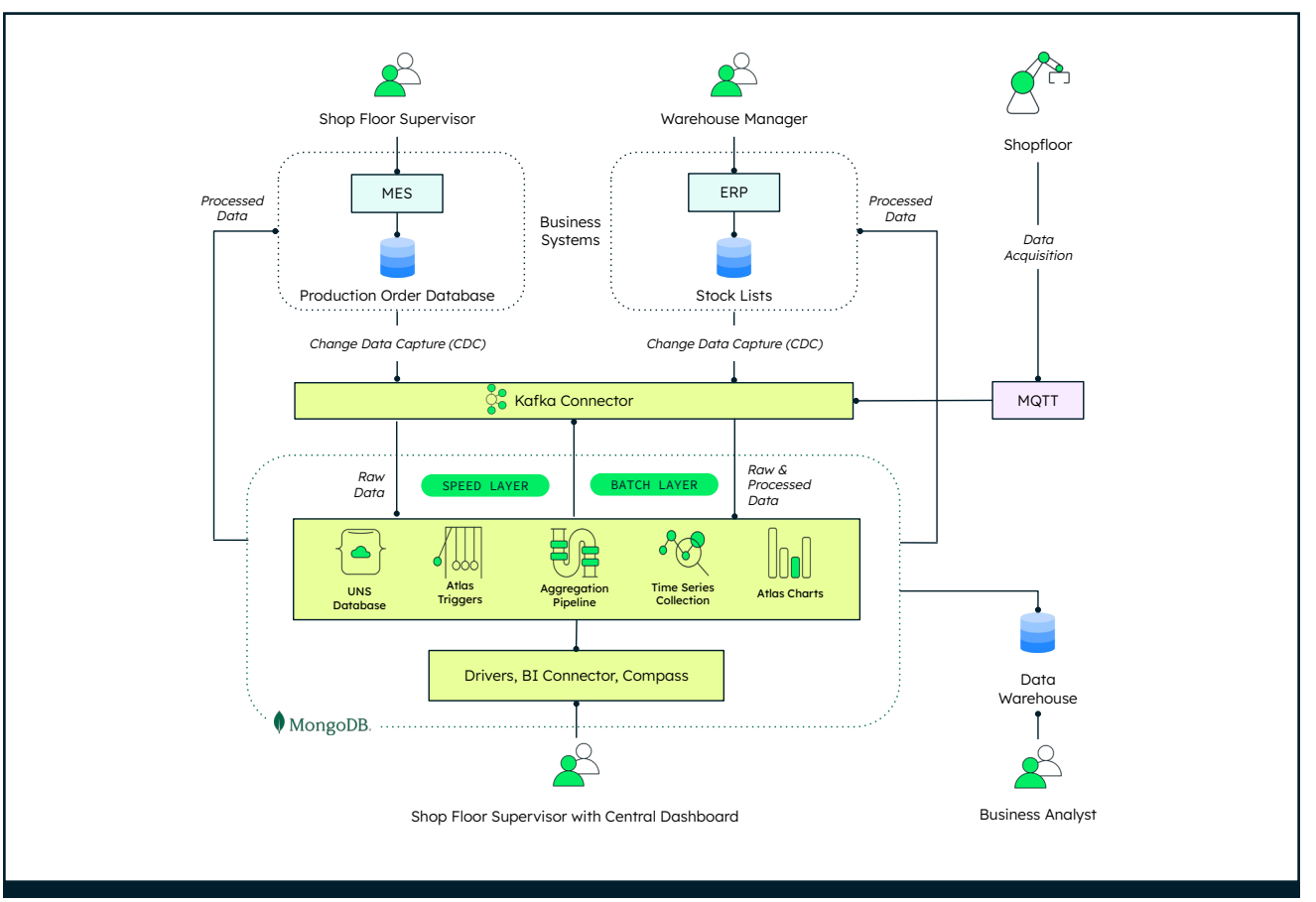

Figure 16. Shop floor supervisor with central dashboard

Read further about why MongoDB is a perfect fit for a UNS and on how to build an industrial unified namespace architecture with MongoDB and Arcstone on the MongoDB website.

Healthcare

Healthcare organizations worldwide face a common and growing challenge: delivering real-time, unified views of patient data across multiple, fragmented systems while ensuring scalability, security, and flexibility. Traditional relational database architectures, often composed of legacy systems accumulated over decades, struggle to meet these modern requirements. The resulting silos, performance bottlenecks, and data management complexities hinder innovation and negatively impact patient outcomes.

MongoDB uniquely addresses these challenges. With its flexible document model, MongoDB empowers healthcare providers to aggregate and unify diverse datasets–clinical, administrative, operational, and patient-generated–in real time. Its schema-agnostic architecture simplifies the integration of evolving data standards and new data sources, reducing the complexity of traditional data warehouses and accelerating time-to-insight.

Real-time patient 360 view

Two critical use cases stand out for an operational data layer in healthcare built on MongoDB. First, the creation of a real-time patient 360-view, enabling healthcare providers to consolidate data from various healthcare entities into a unified patient record accessible instantly by clinicians, administrators, and patients. This dramatically enhances clinical decision-making, patient engagement, and care coordination. Second, Fast Healthcare Interoperability Resources (FHIR) integration and data enrichment significantly boosts interoperability by seamlessly incorporating standard healthcare data (FHIR resources) with additional enriched metadata, vectors, and AI-generated insights. This integration not only improves data exchange and collaboration but also facilitates advanced clinical analytics and AI-driven healthcare solutions.

Healthcare organizations leveraging MongoDB for these ODL applications benefit from significant improvements in patient care outcomes, streamlined operational efficiency, real-time analytics capabilities, enhanced data security, and vastly improved data interoperability across diverse systems and standards.

Customer examples

MongoDB's impact can be seen with **Osakidetza**, the Basque Country's healthcare system. Osakidetza needed a unified application capable of delivering a real-time single view of patient data sourced from diverse systems–medical histories, hospitalizations, consultations, laboratories, and more. Its existing systems, burdened with data silos and slow batch processes, were unable to provide real-time insights necessary for timely care. MongoDB's document model provided the ideal platform, offering scalability, speed, and advanced security. As a result, Osakidetza now benefits from real-time data loading, unified patient views, improved diagnosis and care quality, and enhanced patient service through faster automated data integration.

Similarly, **CatSalut**, the Catalan Health Service, adopted MongoDB to integrate data from SAP Argos, dialysis equipment, and various clinical applications into a high-performance central repository. MongoDB's scalability and real-time capabilities dramatically reduced batch processing times, enabling clinicians and health centers across the region to benefit from a common, timely patient view, thus significantly enhancing diagnostic accuracy and patient care outcomes.

SATUSEHAT, managed by the Indonesian Ministry of Health's Digital Transformation Office (DTO), is another example of the critical impact made by MongoDB. This national digital health platform faced challenges including rapid data growth, query complexity, strict data residency regulations,

and diverse data types–from structured ICD-10 records to complex DICOM imaging. MongoDB's robust scalability, native support for heterogeneous data, and advanced security features allowed SATUSEHAT to unify and manage data from over 10,000 healthcare facilities, positioning Indonesia as a global leader by incorporating WHO-standard international patient summaries (IPS).

These cases highlight a broader industry trend. Many healthcare customers worldwide initially adopted MongoDB for its powerful operational data layer capabilities. However, observing its profound impact, these customers increasingly position MongoDB as their foundational data store. By consolidating their data strategies around MongoDB, they streamline operations, enhance agility, and significantly reduce, or entirely eliminate, their reliance on costly, rigid legacy systems. MongoDB has proven not only to meet the immediate operational needs of healthcare organizations but also pave the way for future innovations and sustainable growth.

Insurance

Insurance companies are increasingly leveraging MongoDB operational data layers to transform their digital infrastructure and enhance operational efficiency. These modern data platforms serve as the connective tissue between legacy systems and innovative applications, enabling insurers to consolidate fragmented domains of information across policy administration, claims processing, and underwriting systems into a unified, canonical format and schema.

By implementing an ODL that takes advantage of MongoDB's document model, insurance organizations can break down traditional data silos, accelerate application development, and more quickly respond to changing market conditions. The flexibility of MongoDB's schema design allows insurers to adapt to evolving regulatory requirements and rapidly introduce new products while maintaining the high-performance, scalable infrastructure necessary to process millions of transactions daily. This technological foundation empowers insurance companies to deliver more personalized customer experiences, optimize risk assessment, and gain competitive advantages in an increasingly data-driven marketplace.

Customer example

Humana, the leading health insurance provider in the U.S., leveraged MongoDB for healthcare interoperability modernization. Rather than attempting disruptive legacy system replacements, Humana built a cloud-native core data fabric using MongoDB Atlas. This ODL approach facilitated rapid implementation of FHIR standards, enabling real-time, bidirectional data flows. This not only improved patient experiences but aligned strategically with Humana's transition toward value-based, personalized healthcare services.

Legal Notice

This document may include certain "forward-looking statements" within the meaning of Section 27A of the Securities Act of 1933, as amended, or the Securities Act, and Section 21E of the Securities Exchange Act of 1934, as amended, including statements concerning our future growth and the potential of MongoDB Atlas; and our ability to transform the global database industry and to capitalize on our market opportunity. These forward-looking statements include, but are not limited to, plans, objectives, expectations and intentions and other statements contained in this document that are not historical facts and statements identified by words such as "anticipate," "believe," "continue," "could," "estimate," "expect," "intend," "may," "plan," "project," "will," "would" or the negative or plural of these words or similar expressions or variations. These forward-looking statements reflect our current views about our plans, intentions, expectations, strategies and prospects, which are based on the information currently available to us and on assumptions we have made. Although we believe that our plans, intentions, expectations, strategies and prospects as reflected in or suggested by those forward-looking statements are reasonable, we can give no assurance that the plans, intentions, expectations or strategies will be attained or achieved. Furthermore, actual results may differ materially from those described in the forward-looking statements and are subject to a variety of assumptions, uncertainties, risks and factors that are beyond our control including, without limitation: the effects of the ongoing military conflicts between Russia and Ukraine and Israel and Hamas on our business and future operating results; economic downturns and/or the effects of rising interest rates, inflation and volatility in the global economy and financial markets on our business and future operating results; our potential failure to meet publicly announced guidance or other expectations about our business and future operating results; our limited operating history; our history of losses; failure of our platform to satisfy customer demands; the effects of increased competition; our investments in new products and our ability to introduce new features, services or enhancements; social, ethical and security issues relating to the use of new and evolving technologies, such as artificial intelligence, in our offerings or partnerships; our ability to effectively expand our sales and marketing organization; our ability to continue to build and maintain credibility with the developer community; our ability to add new customers or increase sales to our existing customers; our ability to maintain, protect, enforce and enhance our intellectual property; the effects of social, ethical and regulatory issues relating to the use of new and evolving technologies, such as artificial intelligence, in our offerings or partnerships; the growth and expansion of the market for database products and our ability to penetrate that market; our ability to integrate acquired businesses and technologies successfully or achieve the expected benefits of such acquisitions; our ability to maintain the security of our software and adequately address privacy concerns; our ability to manage our growth effectively and successfully recruit and retain additional highly-qualified personnel; and the price volatility of our common stock. These and other risks and uncertainties are more fully described in our filings with the Securities and Exchange Commission ("SEC"), including under the caption "Risk Factors" in our Annual Report on Form 10-K for the year ended January 31, 2024, filed with the SEC on March 15, 2024, and other filings and reports that we may file from time to time with the SEC. Except as required by law, we undertake no duty or obligation to update any forward-looking statements contained in this release as a result of new information, future events, changes in expectations or otherwise.

Resources

For more information, please visit mongodb.com or contact us at sales@mongodb.com.

Case Studies

Presentations

Free Online Training

Webinars and Events

Documentation

MongoDB Atlas database as a service for MongoDB

MongoDB Enterprise Download

About MongoDB

MongoDB enables innovators to unleash the power of software and data. Whether deployed in the cloud or on-premises, organizations use MongoDB for trading platforms, global payment data stores, digital end-to-end loan origination and servicing solutions, general ledger system of record, regulatory risk, treasury, and many other back-office processes. At the core of our modern data platform is the most versatile cloud database service on the market, MongoDB Atlas, which can run in any cloud or even across multiple clouds to get the best from each provider with no lock-in.

To learn more about MongoDB, visit MongoDB.com →

About the Authors

Benjamin Lorenz started his career with MongoDB in 2016. He has been instrumental in growing the Central European sales region and was involved in numerous strategic customer projects. In his current role as the Principal for Telco, Media & Content, Benjamin supports customers worldwide in their digital transformation projects and helps establish new, data-powered revenue streams. He lives with his family in Frankfurt, Germany.

Andrea is a Senior Specialist on the Industry Solutions team at MongoDB. In her current role, Andrea focuses on developing thought-leading financial services applications and resources to educate and guide enterprises on their path to modernization and innovation with MongoDB. She is passionate about delivering impactful, data-driven solutions and advancing technological innovation.