Implementing an Operational Data Layer

November 2023

MongoDB.

Table of Contents

Introduction

An Operational Data Layer (or ODL) is an architectural pattern that centrally integrates and organizes siloed enterprise data, making it available to consuming applications. It enables a range of board-level strategic initiatives such as Legacy Modernization and Data as a Service, and use cases such as single view, AI-enriched applications, real-time analytics, and mainframe modernization The simplest representation of this pattern is something like the diagram shown in Figure 1. An Operational Data Layer is an intermediary between existing data sources and consumers that need to access that data.

An ODL deployed in front of legacy systems can enable new business initiatives and meet fresh requirements that the existing architecture can't handle – without the difficulty, disruption, and risk of a full rip and replace of legacy systems. It can reduce the workload and cost of source systems, improve availability, reduce end-user response times, combine data from multiple systems into a single repository, serve as a foundation for re-architecting a monolithic application into a suite of microservices, and more. The Operational Data Layer becomes a system of innovation, allowing the business to take an iterative and progressive approach to digital transformation.

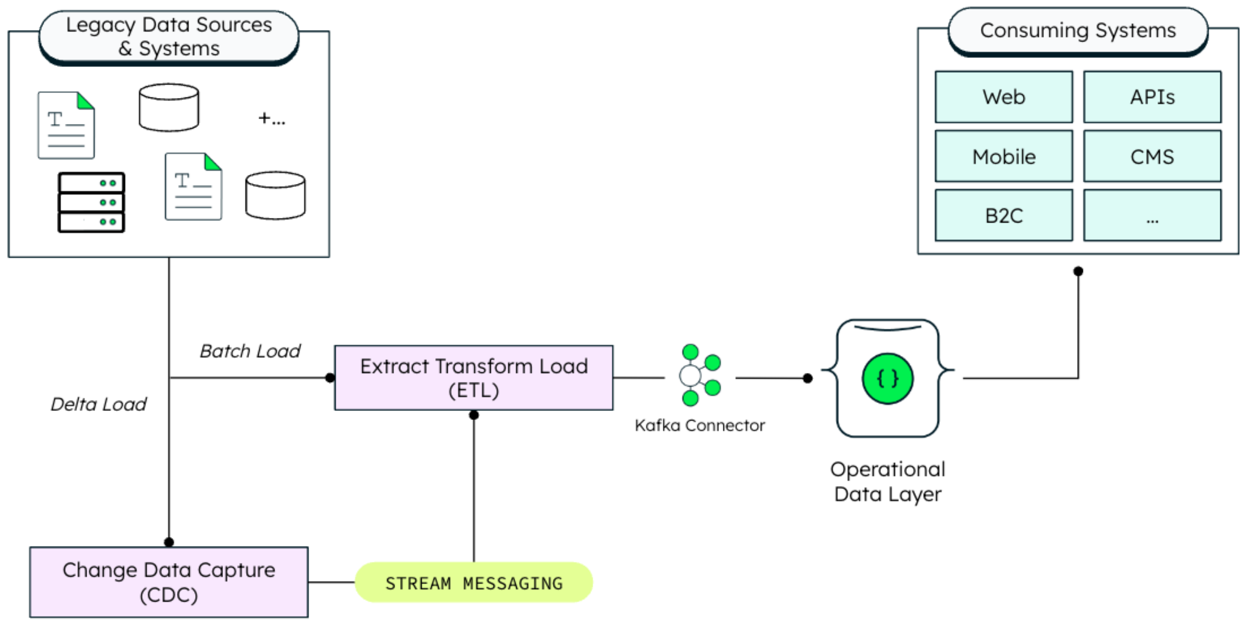

Figure 1: Conceptual model of an Operational Data Layer

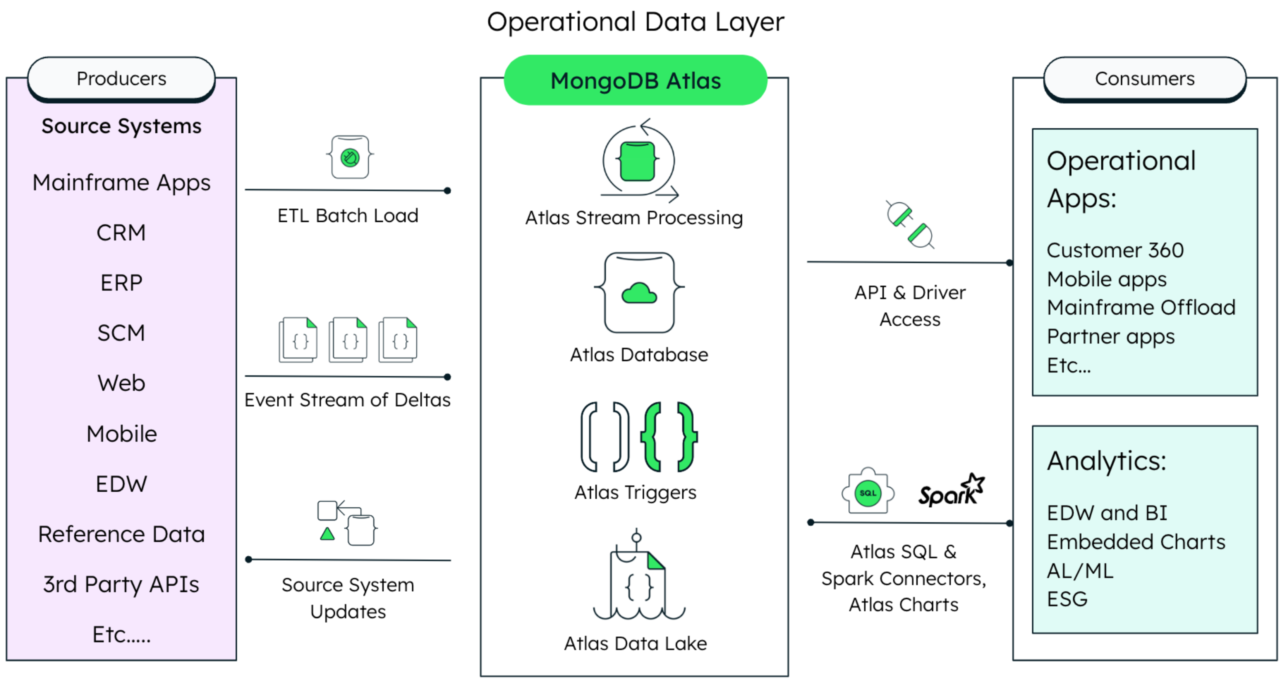

Figure 2: Sample reference architecture for an Operational Data Layer

Other terms sometimes used for this architectural pattern include Operational Data Store, Data Fabric, and Operational Data Hub.

Why Implement an ODL?

An Operational Data Layer is an architectural pattern, but implementing one isn't a goal in and of itself. Organizations choose to build an ODL to unlock the value of previously siloed enterprise data and to power applications that can't be served by existing systems. Some of the most common strategic business initiatives that benefit from an ODL include:

Legacy Modernization

Companies are finding enormous value in new development methodologies, architectural patterns, and technologies, enabling them to build new business functionality 3-5x faster, scale to millions of users wherever they are, cut costs, and more. Legacy systems, however, can hold them back from realizing these benefits. Monolithic code bases, complex dependencies between apps, siloed data, and outdated development processes are a drag on innovation.

There are many ways of modernizing legacy systems, and each has its place. Gradual refactoring of an application using a strangler fig pattern for example can make sense when incremental changes are needed (and feasible), or a wholesale rewrite may be appropriate when a system can safely be cut out of the environment. But in many instances, constructing an Operational Data Layer parallel to legacy systems accelerates and deskrisks modernization.

An ODL can offer a "best of both worlds" approach, providing the benefits of modernization without the risk of a full rip and replace. Legacy systems are left intact – at least at first – meaning that existing applications can continue to work as usual without interruption. New or improved data consumers will access the ODL, rather than subjecting legacy data stores to new workloads that may strain their capacity and expose single points of failure. At the same time, building an ODL offers a chance to redesign the application's data model, allowing for new development and features that aren't possible with the rigid tabular structure of existing relational systems. With an ODL, it is possible to combine data from multiple legacy sources into a single repository where new applications, such as a customer single view or Artificial Intelligence training and inference processes can access the entire corpus of data.

Gradually, existing workloads can be shifted to the ODL, with value delivered at each step. Eventually, the ODL can be promoted to a system of record, and legacy systems can be decommissioned.

Data as a Service

In many organizations, the state of data access is grim. Crucial enterprise data is locked in dozens or hundreds of silos, controlled by different teams, and stuck in systems that aren't able to serve new workloads and access patterns. This is a blocker for innovation and insight, harming the business. For example, building an open banking application for customers to view and manage their account data from any device or platform could require months just to navigate internal processes for getting access to the legacy systems that hold that data, let alone integrate with them.

The solution to this is to invest in a data platform – an Operational Data Layer – that provides data as a service. The ODL itself gathers all important data in one place, while Data Access APIs on top of the ODL provide a common set of methods for working with this data. When development teams need access to existing enterprise data, they can simply leverage the relevant API to access it from the ODL, with no need to open up source systems for each new requirement. An ODL can offer Data as a Service not only to developers, but also to analysts and data scientists; new analyses can be run and new insights generated. When data from multiple source systems is unified in an ODL, consuming applications and AI-powered processes can get the full picture of enterprise data, not available in a world of silos.

Cloud Data Strategy

An Operational Data Layer can be implemented on-premises, in the cloud, or in a hybrid deployment model, as part of a broader Cloud Data Strategy. When legacy on-prem systems can't be migrated but new applications are being deployed in the cloud, a cloud-hosted ODL can provide an intermediary layer. Existing enterprise data is federated to the ODL in the cloud, which makes it available to new cloud-native applications. Source systems remain in place on-premises and can be decommissioned over time as the ODL becomes the system of record. This provides a gradual, non-disruptive approach to cloud transformation.

A cloud-based ODL can also benefit from the on-demand elasticity of cloud infrastructure. It's easy to scale the ODL when an additional set of enterprise data is added, when new consuming applications are created, or when workloads grow.

Use Cases for an ODL

Single View

"Single view" describes a unified, real-time representation of all relevant information on a given entity. Often, this is a customer, and the term "customer 360" is sometimes used. But organizations may also develop single views of products, of financial assets, or other entities relevant for the business. The value of a single view comes from unifying data from multiple sources that otherwise must be accessed separately.

A single view can be exposed to users or consumers, whether internal or external, will benefit from the unified view of data. Internally, this is often customer service representatives, fraud and risk systems, sales and marketing staff, quantitative analysts, and others. Externally, single views can directly power customer service: for example, the full picture of a customer's information can be shown in an online account, and the customer can make changes as needed.

A single view is a perfect fit for an Operational Data Layer. When your goal is to produce a single view, integrating data from multiple sources in a single ODL is the obvious solution. Conversely, if you build an Operational Data Layer for other reasons and already have a store of unified data, it is a simple step to expose that data as a single view via an API for the users who would benefit from it.

Artificial Intelligence and Real-Time Analytics

To compete and win in the digital economy you need to make your applications smarter. Smarter apps use data, AI, and analytics to engage users with natural language, generate insights, and autonomously take action. For these smarter, AI-powered apps to deliver value, they have to work with live data directly within the flow of the app, in real time.

This class of intelligence and analytics is very different from the traditional BI reporting queries consumed by humans staring at dashboards or the predictions generated in batches by offline ML models on historical data. To build this new generation of intelligent applications, we need to do things differently. No longer can we rely on just copying data out of our operational backend systems into centralized analytics data warehouses and data lakes. While this approach is not going away anytime soon, it's not enough on its own. That's because moving data between different systems takes time and creates separation between analyzing application events and taking actions.

Instead we have to bring a new class of AI and analytics processing directly to operational data. We call this **application-driven intelligence**. An ODL is a perfect solution to bring enterprise data together from legacy systems, making it available for both model training and for building new AI-powered features in our applications. The Operational Data Layer contains an up-to-the-minute state of data, and services low latency AI inference and analytics queries, either as part of operational processes or for ad-hoc questions that need an answer now.

Using MongoDB's distributed architecture, you can easily isolate different workloads to dedicated nodes within the cluster. For example, short running operational queries against the ODL are routed to transactional nodes while longer running and more complex AI or analytics queries are routed to dedicated analytics nodes. This approach ensures you can service different workloads from a single ODL, and they never compete for resources.

To learn more, review the Application-Driven Intelligence section of MongoDB's AI web page.

Mainframe Modernization

Mainframes have powered backend business processes for decades, but the access patterns and 24/7 requirements of modern applications stretch their limits. As more business moves online and to mobile, exposing the mainframe directly to new digital channels rapidly drives up costs. Even more critically, it can expose a single point of failure when the mainframe periodically needs to be taken offline for maintenance. Beyond the hardware, the databases running on mainframes are typically based on rigid tabular data models, making it very hard to evolve applications to meet new digital business demands. For all of these reasons, organizations can benefit from offloading workloads from the mainframe. Implementing an Operational Data Layer in front of legacy systems allows you to redirect consuming systems to the ODL, improving application availability, helping to meet new regulatory requirements (for example in data sovereignty), and reducing MIPS costs. At the same time, new apps – or new features built into existing apps – benefit from the option to revise the data model, supporting workloads that weren't possible before.

An ODL makes it significantly easier to serve mainframe data to new digital channels without straining legacy systems. For example, a major UK-based global bank built an ODL to offload work from its mainframe, powering web and mobile banking. The ODL became the foundation for Data as a Service, supporting rapid development of new features and applications. Data as a Service is the advantage that lets a traditional bank move at the speed of a digital disruptor.

And More

An ODL is an excellent way of exposing existing data to new applications while mitigating the potential impact to legacy systems. Whenever the business calls for a new application or experience but the changes to (or additional workload on) the data sources would be prohibitively difficult, building an Operational Data Layer can be the solution. To meet these demands, an ODL must be resilient to failure, globally scalable, and cloud-ready, and it must enhance developer productivity to build and evolve apps faster. Once an ODL is in place, it's often a simple matter to point new apps at it, powering multiple innovative developments with a common, enterprise-wide data layer.

Architecture

Consider again the conceptual model of an Operational Data Layer in Figure 3.

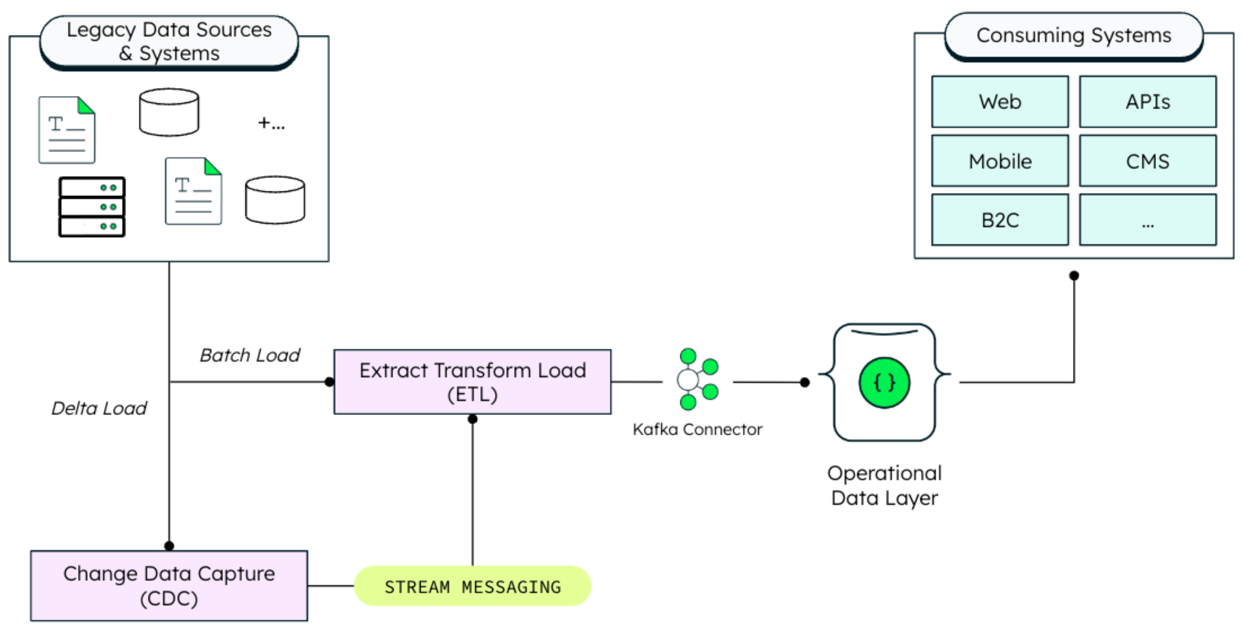

Figure 3: Operational Data Layer model, with data loading shown

Outside of the ODL itself, the chief components in this architecture are the source systems and the consuming systems.

Source Systems & Data Producers

These are usually databases, but sometimes file systems or other data stores. Generally, they are systems of record for one or more applications, either off-the-shelf packaged apps (ERP, CRM, etc.) or internally-developed custom apps.

In some cases, there may be only one source system feeding the Operational Data Layer. Usually, this is if the main goal of implementing an ODL is to add an abstraction layer on top of that single system. This could be for the purpose of caching or offloading queries from the source system, or it could be to create an opportunity to revise the data model for modernization or new uses that don't fit with the structure of the existing source system. An ODL with a single source system is most useful when the source is a highly-used system of record and/or is unable to handle new demands being placed on it; often, this is a mainframe.

More often, there are multiple source systems. In this case, the ODL can unify disparate data sets, providing a complete picture of data that would not otherwise be available.

When planning an Operational Data Layer, start by identifying the applications that generate the required source data and their associated databases, along with the business and technical owners of the applications. For each data source, it can be helpful to appoint a data steward. The steward needs to command a deep understanding of the source database, with specific knowledge of:

- The schema of the source data, plus an understanding of which tables store the required attributes, and in what format
- The clients and applications that generate the source data
- The clients and applications that currently consume the source data.

The data steward should also be able to define how the required data can be extracted from the source database to meet the ODL's requirements (e.g., frequency of data transfer), without affecting either current producing or consuming applications.

Consuming Systems

An ODL can support any consuming systems that require access to data. These can be either internal or customer-facing. Existing applications can be updated to access the ODL instead of the source systems, while new applications (often delivered as domains of microservices) will typically use the ODL first and foremost. The requirements of a single application may drive the initial implementation of an ODL, but usage usually expands to additional applications once the ODL's value has been demonstrated to the business.

An Operational Data Layer can also feed AI and analytics systems, providing insights that weren't possible without a unified data system. AI model training tools along with business reporting tools can connect to an ODL for an up-to-the-minute view of the company – without interfering with operational workloads – while the data can also support generative AI and inference features within AI-augmented apps. .

When identifying consuming systems that will rely on the ODL, consider:

- How their business processes operate, including the types of queries executed as part of their day-to-day responsibilities, and the required Service Level Agreements (SLAs)
- The specific data they need to access
- The sources from which the required data is currently extracted, or would be extracted

Data Loading

For a successful ODL implementation, the data must be kept in sync with the source systems. Once the source systems' producers have been identified, it's important to understand the frequency and quantity of data changes in producer systems. Similarly, consuming systems should have clear requirements for data currency. Once you understand these, it's much easier to develop an appropriate data loading strategy.

1. **Batch extract and load.** This is typically used for an initial one-time operation to load data from source systems. Batch operations extract all required records from the source systems and load them into the Operational Data Layer for subsequent merging. If none of the consuming systems requires up-to-the-second level real-time data and overall data volumes are low, it may also suffice to refresh the entire data set with periodic (daily/weekly) data refreshes. Batch operations are also good for loading data from producers that are reference data sources, where data changes are typically less frequent – for example, country codes, office locations, tax codes, and similar. Commercial ETL tools or custom implementations are used for carrying out batch operations: extract data from producers, transform the data as needed, and then load it into the ODL. If after the initial load the development team discovers that additional refinements are needed to the transformation logic, then the related data from the ODL may need to be dropped, and the initial load repeated.

2. In the process of **Delta extract and load**, there's a crucial step involving an event producer typically added to the source system. This event producer commonly leverages log files to trigger Change Data Capture (CDC) events when data in the source system changes in any way. These CDC events are then subsequently streamed to the MongoDB Atlas-based Operational Data Layer (ODL). This event-driven approach ensures that the source systems and the ODL remain synchronized.

   To further enhance this data integration process, the MongoDB Connector for Apache Kafka plays a pivotal role, serving both as a source and a sink. This setup is designed to maintain a current and synchronized Operational Data Layer (ODL) that reflects changes in source systems almost instantly. It's worth noting that the frequency of these delta operations can vary significantly. In some cases, changes are captured and propagated at regular intervals, such as every few hours, while in other cases, they are event-based, streaming to the ODL as soon as new data is committed to the source systems.

Matching, merging, and reconciling data from disparate systems as part of the data load remains an important aspect, and it represents a topic of its own within this broader data integration framework.

Data Flow and Maturity Model

Often, an Operational Data Layer evolves over time. From a focused and simple start, it grows in scope and strategic importance, delivering increased benefits to the business. In fact, it is a best practice to begin an ODL implementation with a limited scope and grow it over time. An Operational Data Layer that tries to incorporate all possible data sources from the beginning may stall before it proves its value; it's much better to demonstrate the ODL's capabilities with a small set of sources and consumers, then iterate from there, incorporating best practices that have been developed along the way.

One of our observations of working with many enterprises on their ODL projects is that as more data sources are integrated, the ODL becomes valuable for serving reads from a broadening range of consumers. Over time, it begins accepting writes, and ultimately can become a system of record and system of innovation in its own right.

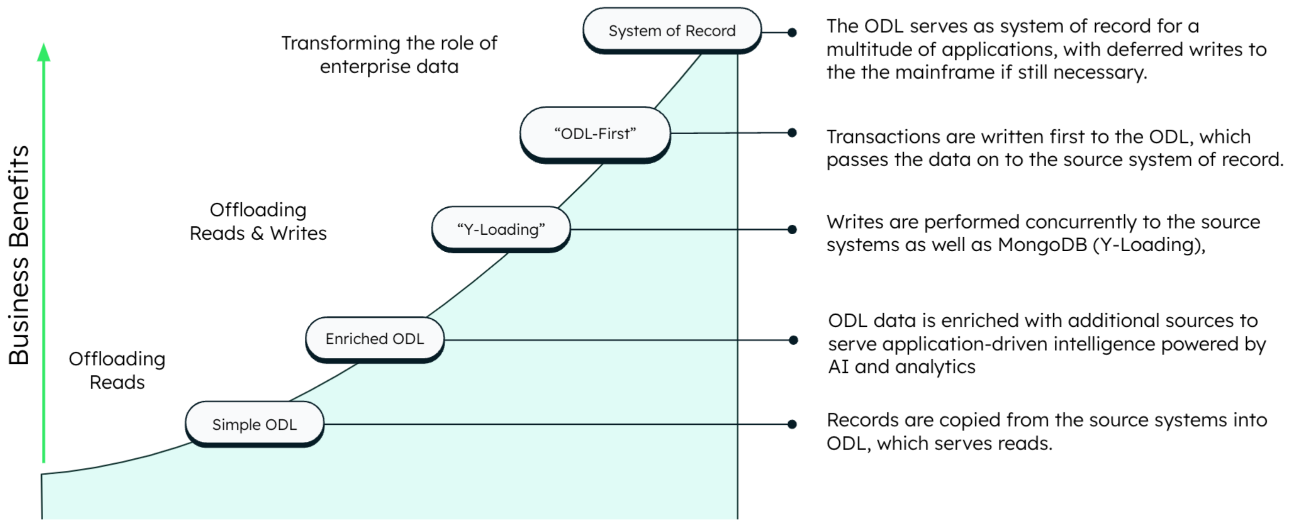
Business Benefits
Transforming the role of enterprise data
Offloading Reads & Writes
Offloading Reads
System of Record
"ODL-First"
"Y-Loading"
Enriched ODL
Simple ODL
The ODL serves as system of record for a multitude of applications, with deferred writes to the the mainframe if still necessary.
Transactions are written first to the ODL, which passes the data on to the source system of record.
Writes are performed concurrently to the source systems as well as MongoDB (Y-Loading),
ODL data is enriched with additional sources to serve application-driven intelligence powered by AI and analytics
Records are copied from the source systems into ODL, which serves reads.

Figure 4: Maturity model of an Operational Data Layer

Phase 1: Simple ODL, Offloading Reads.

Initial use cases for an ODL are often tightly scoped, with one (or a few) source systems and one (or a few) consuming systems. The goal is usually to serve only read operations – the ODL can reduce workload on source systems to cut costs, provide high availability during source system downtime, improve performance, or handle long-running analytics and ML model building queries. Whatever the motivation, the ODL has the potential to easily offload a significant amount of read traffic from overburdened source systems.

Most organizations look to realize the value of always-on offloading from the beginning, improving performance and reducing costs. Sometimes, an ODL may start with a more limited scope, in which the source system serves all reads normally, while the ODL is used to replicate data to remote regions and only used to serve users in those countries. Usually, the ODL in these cases quickly transitions to serving certain workloads at all times.

In this phase, all write traffic continues to go to source systems, and data changes are then pushed into the ODL via the delta load mechanisms described above.

Phase 2: Enriched ODL for New Use Cases

Once the ODL has proven its value, a logical next step is to enrich its data by adding useful metadata or integrating new (related) data sources. A typical use case for this is to enable AI and advanced analytics across a fuller picture of data or create a single customer view. For example, credit card transactions could be enriched by categorizing purchases according to third party information. A card issuer could, for instance, then make it much easier for customers to determine their spend on each category – e.g., travel purchases over the last n months, or to detect fraud by better identifying unusual patterns of behavior.

With this enrichment, the ODL can not only offload more reads from source systems, but also enable use cases that weren't possible before.

Phase 3: Offloading Reads and Writes

The ODL's scope can be expanded by introducing a smarter architecture to orchestrate writes between both source systems and the ODL concurrently. In this phase, when a given consuming system performs a write, it goes to both ODL and source system, either directly from application logic or via a stream messaging system, API layer, or other intermediary from which both repositories can receive the write. This pattern is also referred to as "Y-loading", and can help lay the foundations for a more transformational shift of the ODL's role in your enterprise architecture.

Some organizations move directly to phase 4 below, but Y-loading can allow you to run both systems in parallel and road-test the ODL before using it as the primary system for writes.

Phase 4: ODL First

By default, all writes are directed to the ODL. Where necessary, changes are routed using change streams and the Kafka source connecter from the ODL back to the source systems, either so that legacy applications can continue to rely on a source system before being ported to the ODL, or merely as a fallback, in case it should be needed. The secondary write to the source system can be accomplished with a CDC system listening to the ODL or a similar system, like MongoDB Atlas Triggers.

Phase 5: System of Record

Ultimately, the Operational Data Layer can evolve to serve as the System of Record. Once all consuming systems for a given legacy source have been ported to the ODL, and its stability has been proven, the source system can be decommissioned for cost savings and architectural simplicity.

Why MongoDB for an Operational Data Layer?

MongoDB meets the required capabilities for an Operational Data Layer. When you choose MongoDB as the foundation for an ODL, you're investing in the best technology, people, and processes for your system of innovation.

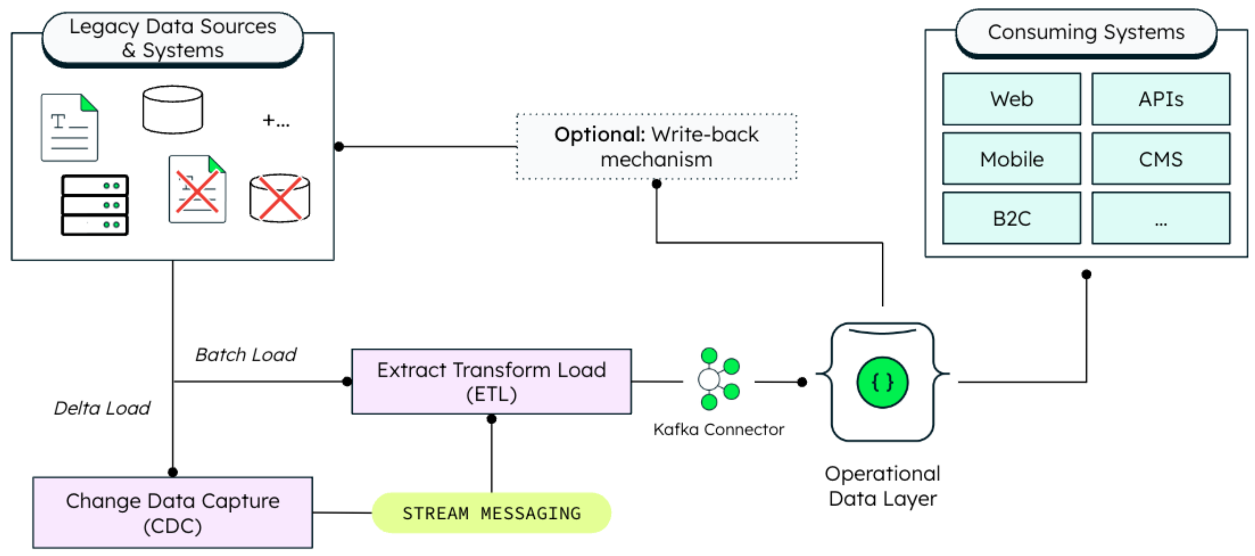

Figure 5: Operational Data Layer, accepting writes and optionally pushing them back to source systems, with select source systems decommissioned

Technology

MongoDB is the best way for an ODL to work with data

MongoDB's developer data platform integrates all of the data services you need to build modern applications in a unified developer experience. It handles transactional workloads, AI-enriched apps with vector and full-text search, time-series and stream data processing, app-driven analytics,, and more, all while working with a single copy of the data and a consistent interface, reducing data infrastructure sprawl and complexity.

Intuitive Developer Experience

Build and iterate quickly with an intuitive and flexible database that matches how your applications work with data and can be modified at any time as application requirements evolve.

Match the data structures in your application code with a powerful model at the data layer. MongoDB's document data model is intuitive and easy to use, while powerful and versatile enough to contain and model all of the data types and access patterns that your application needs.

Move faster with MongoDB's flexibility: schemas don't need to be defined up front, allowing you to easily evolve your data model alongside application requirements. When necessary, it's easy to impose schema controls for data governance.

Build with MongoDB in the same ways you build your applications. Connect using idiomatic drivers for more than a dozen popular languages, or use API access layers for HTTP and GraphQL. Work with native MongoDB developer tooling and extensions for the IDEs you already use.

One Platform, Many Workloads

Address any application's data needs quickly with an integrated collection of data and application infrastructure capabilities in a unified developer data platform.

Manage all of the data models and access patterns your application needs in a single database, and query with a single API that handles everything from basic CRUD and transactions to composable aggregation pipelines and vector search.

Eliminate the time and costs of data plumbing; instead of requiring you to connect all your data services together manually with ETL, CDC, and other data movement tools, MongoDB Atlas automatically synchronizes and moves data between services (database, search, mobile, archival tiers, and more).

Operational Ease of Use

Confidently run your most important database workloads with automated operations and integrations into your deployment workflows.

Simplify deployment across application and data layers by building MongoDB into your existing CI/CD workflows: MongoDB Atlas can be controlled by UI, API, CLI, and integrations with Kubernetes and Infrastructure as Code tools.

Rest assured that your data layer will stay up and running even under the most intense workloads with built-in replication and failover, auto-scaling, and the ability to deploy in 100+ regions across three cloud providers.

Deliver fast end user experiences while controlling costs with intelligent performance optimization, plus built-in monitoring and alerting that integrates with your observability tooling.

Flexibility. A flexible data model is essential to integrate multiple source systems into a single ODL. With MongoDB, there's no need to pre-define a schema. Documents are polymorphic: fields can vary from document to document within a single collection. For example, all documents that describe customers might contain the customer ID and the last date they purchased a product or service, but only some of these documents might contain the user's social media handle or location data from a mobile app. This makes it possible to merge data from source systems storing records on overlapping but non-identical sets of entities. On the consuming side, MongoDB's flexibility makes it easy to alter the data model as needed to meet the requirements of new applications being built on the ODL.

Speed. Using MongoDB for an ODL means you can get better performance when accessing data, and write less code to do so. In most legacy systems, accessing data for an entity, such as a customer, typically requires JOINing multiple tables together. JOINs entail a performance penalty, even when optimized – which takes time, effort, and advanced SQL skills. The situation is even worse when you consider that for a given requirement, a consuming system may need to access multiple legacy databases.

In MongoDB, a document is a single place for the database to read and write data for an entity. This locality of data ensures the complete document can be accessed in a single database operation that avoids the need internally to pull data from many different tables and rows. For most queries, there's no need to JOIN multiple records. If the MongoDB-based ODL integrates data from multiple source systems, the performance benefits of accessing that unified data are even greater.

Versatility. Building upon the ease, flexibility, and speed of the document model, MongoDB enables developers to satisfy a range of application requirements, both in the way data is modeled and how it is queried.

The flexibility and rich data types of documents make it possible to model data in many different structures, representative of entities in the real world. The embedding of arrays and subdocuments makes documents very powerful for modeling complex relationships and hierarchical data, with the ability to manipulate deeply nested data without the need to rewrite the entire document. But documents can also do much more: they can be used to model flat, table-like structures, simple key-value pairs, text, geospatial data, the nodes and edges used in graph processing, and more.

With MongoDB's expressive query language, documents can be queried in many ways, from simple lookups and range queries to creating sophisticated processing pipelines for data analytics and transformations, faceted search, JOINs, geospatial processing, and graph traversals.

This versatility is crucial for an Operational Data Layer. As an ODL evolves over time, it typically incorporates new source systems, with data model implications that weren't planned from the outset. Similarly, new consuming systems that connect to the ODL will have access patterns and query requirements that haven't been seen before. An Operational Data Layer needs to be versatile enough to meet a wide variety of requirements.

Data Access and APIs. Consuming systems require powerful and secure access methods to the data in the ODL. If the ODL is writing back to source systems, this channel also needs to be handled.

MongoDB's drivers provide access to a MongoDB-based ODL from the language of your choice. Most organizations building an ODL choose to develop a common Data Access API; this layer communicates to the ODL via a driver, in turn exposing a common set of data access methods to all consumers. This API layer can be custom built, or MongoDB App Services can be used to expose access methods with a built-in rules engine for fine-grained security policies, giving precise control over what data each consumer can access, down to the level of individual fields.

When writes are issued to the ODL but must also be propagated back to source systems, this can be accomplished in one of two ways. First, the API layer described above could simultaneously send writes to the MongoDB ODL and the relevant source system. Second, writes can be issued only to the ODL, while Atlas Triggers or MongoDB Change Streams watch for changes in the ODL and notify source systems. (Alternatively, this could be accomplished with a commercial CDC tool).

Consuming systems aren't limited to operational applications. Many data analytics tools can use MongoDB's connectors to access the ODL. The MongoDB SQL Connector allows analysts to connect to a MongoDB ODL with their BI and visualization tools of choice; alternatively, MongoDB Charts can connect directly to the ODL for native visualization. The Connector for Apache Spark exposes MongoDB data for use by all of Spark's libraries, enabling advanced analytics such as machine learning processes.

MongoDB lets you intelligently distribute an ODL

Consuming systems depend on an ODL. It needs to be reliable, scalable, and offer a high degree of control over data distribution to meet latency and data sovereignty requirements.

Availability. Availability is always crucial for mission-critical applications. An ODL may need to meet even higher standards because it is often implemented precisely in order to provide improved availability in the event of source system outages.

MongoDB maintains multiple copies of data using replica sets. Replica sets are self-healing as failover and recovery is fully automated, so it is not necessary to manually intervene to restore a system in the event of a failure, or to add additional clustering frameworks and agents that are needed for many legacy relational databases.

Scalability. Even if an ODL starts at a small scale, you need to be prepared for growth as new source systems are integrated, adding data volume, and new consuming systems are developed, increasing workload. To meet the needs of an ODL with large data sets and high throughput requirements, MongoDB provides horizontal scale-out on low-cost, commodity hardware or cloud infrastructure using sharding. Sharding automatically partitions and distributes data across multiple physical instances, or shards, all in a completely application-transparent way. To respond to fluctuating workload demand, nodes can be added or removed from the ODL in real time, and MongoDB will automatically rebalance the data accordingly, without manual intervention.

Workload Isolation. An ODL must be able to safely serve disparate workloads. It should be able to serve analytical queries on up-to-date data, without having an impact on production applications. This can obviate the need for a new data warehouse, data mart, or just extracting a new cut of data whenever analysts call for it.

MongoDB's replication provides a foundation for combining different classes of workload on the same MongoDB cluster, each operating against its own copy of the data. With workload isolation, business analysts can run exploratory queries and generate reports, and data scientists can build machine learning models without impacting operational applications. Within a replica set, one set of nodes can be provisioned to serve operational applications, replicating data in real time to other nodes dedicated to serving analytic workloads.

Data Locality. An ODL works best if data is distributed according to the needs of its consuming systems. MongoDB allows precise control over where data is physically stored in a single logical cluster. For example, data placement can be controlled by geographic region for latency and governance requirements, or by hardware configuration and application features to meet specific classes of service for different consuming systems. Data placement rules can be continuously refined as required, and MongoDB will automatically migrate the data to its new zone.

MongoDB gives you the freedom to run anywhere

Portability. MongoDB runs the same everywhere: on-premises in your data centers, on developers' laptops, in the cloud, or as an on-demand fully managed Database as a Service: MongoDB Atlas. Wherever you need to deploy MongoDB, you can make full use of its capabilities. This also offers the flexibility to change deployment strategy over time – e.g., to move from on-prem to the cloud – or even to deploy clusters in hybrid environments.

Global Coverage. MongoDB's distributed architecture allows a single logical cluster to be distributed around the world, situating data close to users. When you use MongoDB Atlas, global coverage is even easier; Atlas supports 110+ regions across all the major cloud providers. Global cluster support enables the simplified deployment and management of a single geographically distributed Operational Data Layer. Organizations can easily control where data is physically stored to allow low-latency reads and writes and meet the data sovereignty requirements demanded by data privacy regulations like the GDPR. Data can also be easily replicated and geographically distributed, enabling multi-region fault tolerance and fast, responsive reads of all the data in an ODL, from anywhere.

No Lock-In. With MongoDB, you can reap the benefits of a multi-cloud strategy. Since Atlas clusters can be deployed on all major cloud providers, you get the advantage of an elastic, fully-managed service without being locked into a single cloud provider. If business conditions require you to change the underlying infrastructure, it's easy to do so. Of course, there's no requirement to use MongoDB Atlas. You can always deploy MongoDB on-prem, manage it yourself in the cloud, or use MongoDB Atlas – and change between deployment methods over time if needed. When you build an Operational Data Layer with MongoDB, you're fully in charge of how and where you run it. MongoDB also provides you with Cluster-to-Cluster sync that can be used from on-prem to the cloud.

For more details on MongoDB technology, read the MongoDB Architecture Guide.

People and Process

Of course, implementing an Operational Data Layer does not only depend on technology. Success also relies on skilled people and carefully designed processes.

MongoDB's Data Layer Realization methodology is a tried and tested approach to constructing an Operational Data Layer. It helps you unlock the value of data stored in silos and legacy systems, driving rapid, iterative integration of data sources for new and consuming applications. Data Layer Realization offers the expert skills of MongoDB's consulting engineers, but also helps develop your own in-house capabilities, building deep technical expertise and best practices.

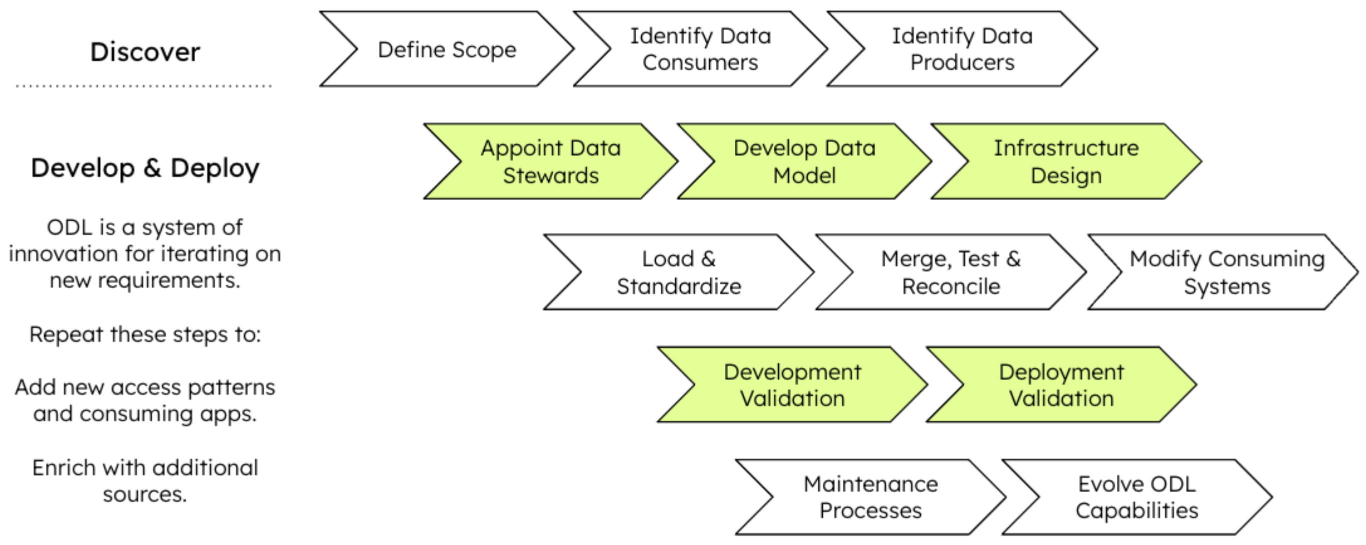

Figure 6: MongoDB methodology for implementing an Operational Data Layer

This process for constructing an Operational Data Layer has been successfully implemented with many customers. Starting with clear definitions of project scope and identifying required producing and consuming systems is the first step to ensure success. Based on these findings, we assign data stewards for clear chains of responsibility, then begin the process of data modeling and infrastructure design. Data is loaded and merged into the Operational Data Layer, then data access APIs are built and consuming systems are modified to use the ODL instead of legacy systems. We conduct validation of both development capabilities using the ODL and the deployment architecture, then optimize ODL usage, implementing maintenance processes and evolving the ODL based on the next wave of required functionality. This process is iterative, repeating in order to add new access patterns and consuming apps or enrich the ODL with new data sources.

A successfully implemented ODL is a springboard for agile implementation of new business requirements. MongoDB can help drive continued innovation through a structured program that facilitates prototyping and development of new features and applications.

Operational Data Layer Success Stories

Nationwide

Nationwide, the world's largest building society with over 15 million members, recognized the need to adapt to the evolving demands of online and digital banking. Their reliance on outdated mainframe technology was a roadblock to innovation. To address this, Nationwide initiated a move to "offload" the mainframe by replicating data to a real-time event-streaming platform called Speed Layer. This platform consolidated and enriched data, enabling faster development and real-time customer services.

To support this transformation, Nationwide sought a modern, flexible, and secure database. After a thorough evaluation, they selected MongoDB Atlas, a globally trusted cloud database service, and deployed it on AWS. MongoDB's reputation for versatility, functionality, resilience, and security made it the ideal choice for Nationwide's evolving needs.

Learn more here.

Toyota

Toyota Financial Services (TFS), a global subsidiary of Toyota Motor Corporation, is undergoing a transformative shift from off-the-shelf software to in-house development. In 2017, they established a data lake on MongoDB, now an integral part of their agile developer culture. Schuelke, Division Information Officer, for Enterprise API Services at Toyota Financial Services, a key figure in this transformation, recognized the value of MongoDB's developer community and online support. TFS migrated to Atlas, MongoDB's developer data platform, reducing infrastructure management time by 40%. They're also exploring Atlas Search and Atlas Charts.

Today, Atlas is the backbone of TFS's data fulfillment service, consolidating data from mainframe applications and powering the event log system, Elvis, for quick response to millions of weekly events. TFS now operates on a robust and flexible database platform, safeguarding sensitive data and enabling rapid responses. This shift positions TFS to diversify into the dynamic mobility sector, moving beyond traditional finance lending.

Learn more here.

Humana

Humana, a large healthcare enterprise with complex legacy systems, faced the challenge of modernizing their core platform quickly. They opted for a cloud-native core data fabric with a focus on interoperability. Utilizing MongoDB Atlas and an API-first approach based on the FHIR standard, they designed a solution to access and serve healthcare data rapidly. This core data fabric streamlined data from various sources into a standardized format for interoperable APIs.

Humana's two-tiered strategy involved optimizing data via the core data fabric, merging it into MongoDB for data velocity. They then partnered with Microsoft Azure's FHIR-focused cloud storage for interoperability. By being among the first healthcare insurers to fully adopt FHIR, Humana improved patient experiences with real-time data integration, differentiating themselves through service and outcomes, not just pricing.

Learn more here.

Conclusion

An Operational Data Layer makes your enterprise data available as a service on demand, simplifying the process of building transformational new applications. It can reduce load on source systems, improve availability, unify data from multiple systems into a single real-time platform, serve as a foundation for re-architecting a monolith into microservices, unlock the value of data trapped inside legacy systems, and more. An ODL becomes a system of innovation, allowing an evolutionary approach to legacy modernization.

Implementing an ODL can be a challenging undertaking. It requires technical skills, the right tools, and coordination among many different parts of the business. By working with MongoDB, you get access to the right technology, people, and process for success.

To discuss building an Operational Data Layer with MongoDB, contact us.

About MongoDB

MongoDB empowers 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 developer data platform is the most advanced 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

Resources

For more information, please visit mongodb.com or contact us at sales@mongodb.com.
Case Studies (mongodb.com/customers)
Presentations (mongodb.com/presentations)
Free Online Training (university.mongodb.com)
Webinars and Events (mongodb.com/events)
Documentation (docs.mongodb.com)
MongoDB Atlas database as a service for MongoDB (mongodb.com/cloud)
MongoDB Enterprise Download (mongodb.com/download)

Legal Notice