Reference Architecture: Mainframe Modernization

Creating an Operational Data Layer

November 2022

MongoDB.

Table of Contents

Introduction

Despite its long predicted demise, the mainframe remains a critical asset in the IT infrastructure of many large enterprises. But ongoing reliance on the mainframe does not come without challenges, some of which include:

- **Costs:** Industry analysts found that almost all CIOs with mainframes in their environments cite overall cost as one of their top concerns, with mainframe prices rising up to 11% in the previous year, and data center costs expected to increase 6% year to year.
- **Skills Availability:** CIOs fear a skills shortage as qualified mainframe staff age and retire, resulting in project delays and further cost increases as the scarcity of remaining staff commands higher premiums.
- **The Growth of Digital:** Every organization is re-inventing itself as a digital business. Web, mobile, social, Artificial Intelligence, and Internet of Things applications are driving a deluge of new data. The volume, speed, and diversity of this data are overwhelming mainframe environments. Coupled with pressures to meet new regulatory demands, CIOs are challenged in how quickly they can remake the business for digital while trying to innovate on top of legacy technologies.

With each MIPS costing up to $5,000 per annum, mainframe operating costs can now account for up to 40% of an organization's total IT budget and decrease valuable funds that could be better directed towards driving new digital innovation. To counter spiraling costs and unlock business agility, more enterprises are modernizing workloads from existing mainframes to modern data platforms such as MongoDB.

Purpose & Audience

This reference architecture will illustrate how MongoDB can be used to modernize existing applications and provide a foundation to accelerate new digital initiatives. It is designed for architects seeking to gain a deeper understanding of the scenarios, patterns, and use cases where MongoDB can power a mainframe modernization initiative. This implementation is often alternatively referred to as an operational data layer (ODL) or operational data source (ODS).

Drawing directly from our experience across domains, this document is intended to help you achieve secure, scalable, efficient, low-risk implementations that provide a platform for digital innovation.

It is important to note that this reference architecture concentrates mainly on the modernizing of read operations from the mainframe. For many of our customers, however, the logical extension of this initial step is the eventual deprecation of the mainframe as an operational data store, in favor of a more agile and flexible MongoDB developer data platform. Whether seeking to merely modernize reads from the mainframe or enable complete deprecation, these projects are proven to deliver a significantly

lower TCO and faster time to market for new applications through the use of commodity hardware, more affordable licensing, lower staff costs, and the introduction of modern application development practices.

What is MongoDB?

MongoDB is a developer data platform with an integrated set of database and data services. Not only does it deliver a high-performance, high-reliability, horizontally scalable document database, it also increases developer productivity and enables customers to:

- Leverage data and technology to maximize competitive advantage.
- Reduce the risk associated with mission-critical deployments.
- Accelerate time-to-value.
- Dramatically lower total cost of ownership.

With MongoDB's distributed design and non-tabular data model, developers can deliver applications that were never possible with traditional relational databases at the speed of modern day and age. MongoDB provides a technology foundation that enables development teams through:

1. The storage of **data in flexible, JSON-like documents,** meaning fields can vary from document to document and data structures can be changed over time.
2. The document model, which **maps to the objects in your application code,** making data easy to work with.
3. **A distributed database,** so high availability, horizontal scaling, and geographic distribution are built in and easy to use.

With these capabilities, you can build an operational data layer, providing you with:

- **Flexible, Fast, Transactional Model:** A flexible document data model coupled with dynamic schema and idiomatic native language drivers make it fast for developers to build and evolve applications. With the document model, you can combine data of any structure without giving up sophisticated governance, controls, data access, rich indexing, and ACID transactional guarantees. The schema can be dynamically modified without application downtime. As a result, development teams spend less time preparing data for the database and more time putting data to work.
- **Multi-Region Scalability:** MongoDB can be scaled across geographically distributed data centers and cloud environments, providing new levels of availability and scalability to meet the demands of customers, wherever they are. As deployments grow in data volume and throughput, MongoDB scales easily with no downtime, and without changing the application.
- **Multi-model Feature Set:** for building rich operational and analytical apps. Analytics and data visualization, text search, graph processing, geospatial, in-memory performance, and global replication enable architects to reliably and securely deliver a wide range of real-time applications on a single technology. No need to install multiple databases to meet the needs of different applications. No need to move data into expensive data warehouses or complex and ungoverned data lakes, to extract insight and value from data.

- **Lower TCO:** Application development and operational teams are more productive when they use MongoDB. Single-click management means operations teams are as well.MongoDB runs on commodity hardware in your own data center, in the cloud, or as a fully-managed service, dramatically lowering costs. Finally, MongoDB offers affordable monthly and pay-as-you-go subscriptions, including 24x7x365 global support. Applications can be one tenth the cost to deliver compared to using traditional relational databases.

A survey of over 1,000 C-level executives and 1,000 developers by Stripe and Evans Poll concluded that the biggest hindrance to developer productivity is maintenance of legacy systems and technical debt. By modernizing with MongoDB, you can build business functionality 3-5x faster, scale to millions of users, and cut costs by 70% and more. All by unshackling yourself from legacy systems.

You can learn more about MongoDB by downloading the Architecture Guide.

Mainframe Modernization: What and Why?

Mainframe modernization is the process of replicating commonly accessed data from the mainframe to a separate data store, the operational data layer (ODL), against which queries from consuming applications run.

An illustrative high-level goal state architecture is provided in Figure 1. The individual components and processes are described in more detail throughout this paper.

There are many drivers for introducing an ODL to modernize the mainframe and, depending on the sector within which an organization operates, some may be more important than others. However, it is highly likely that most of the following considerations are significant factors.

Cost savings

Redirecting queries away from the mainframe to the ODL directly reduces million instruction per second (MIPS) costs. Even a reduction of just 20%-30% in MIPS consumption can save $ millions in mainframe operating costs.

What's more, these already significant savings do not account for the increase in data volumes most organizations are experiencing. Data growth is estimated to double every two to three years, which would result in major cost from increased MIPS consumption unless the load can be modernized into an ODL.

New digital initiatives

Organizations are increasingly seeking to engage customers across new digital channels, which in turn drives significant growth in both the number of consumers and the frequency with which mainframe data is accessed. For example,

- Financial institutions provide access to online account balances and policies to customers across new web and mobile channels.
- Retail systems expose a customer's current order status and complete

purchase history rather than just the past month's transactions.
- Government portals enable citizens to review their latest tax statements without contacting a call center.
- Field sales staff access pricing data and stock inventory in real-time while meeting face-to-face with their customers.

In addition to opening up mainframe data to new apps, many organizations are focused on improving the customer experience by integrating data from multiple back-end mainframe systems into a single, 360-degree customer view. This single view helps increase the speed with which customers are served and provides deeper insight into each customer in order to better personalize experiences and drive cross-sell and upsell opportunities.

Trying to serve these new digital initiatives from an existing mainframe platform can present major challenges that drastically reduce the pace of application delivery while escalating cost and risk:

- Drives greater consumption of expensive mainframe MIPS, further increasing costs, especially if larger systems have to be acquired to handle the increased load.
- Limits the organization's ability to grow and scale new services quickly due to its expensive, coarse-grained scale-up model for expanding capacity. The lead time for new hardware means there's no way to react to sudden peaks in demand. As a result, mainframe customers must over-provision hardware to cope with temporary usage spikes, with the hardware remaining relatively idle during normal business demand.
- Reliance on mainframe-based relational databases with rigid schemas inhibits both agile development and the ability to integrate multiple data sources to build a single, consolidated customer view.
- Difficulty in finding experienced mainframe staff who can build new digital apps on top of the mainframe. It is also cost and performance prohibitive to prototype new classes of applications on the mainframe.
- Poor customer experience due to the increased latency of accessing remote mainframe systems. Also, customers often contend with limited mainframe resources shared with other users and with core backend business system processing. Mainframe downtime also negatively impacts customer experience, especially if serving the new digital applications discussed earlier, creating risk of churn.

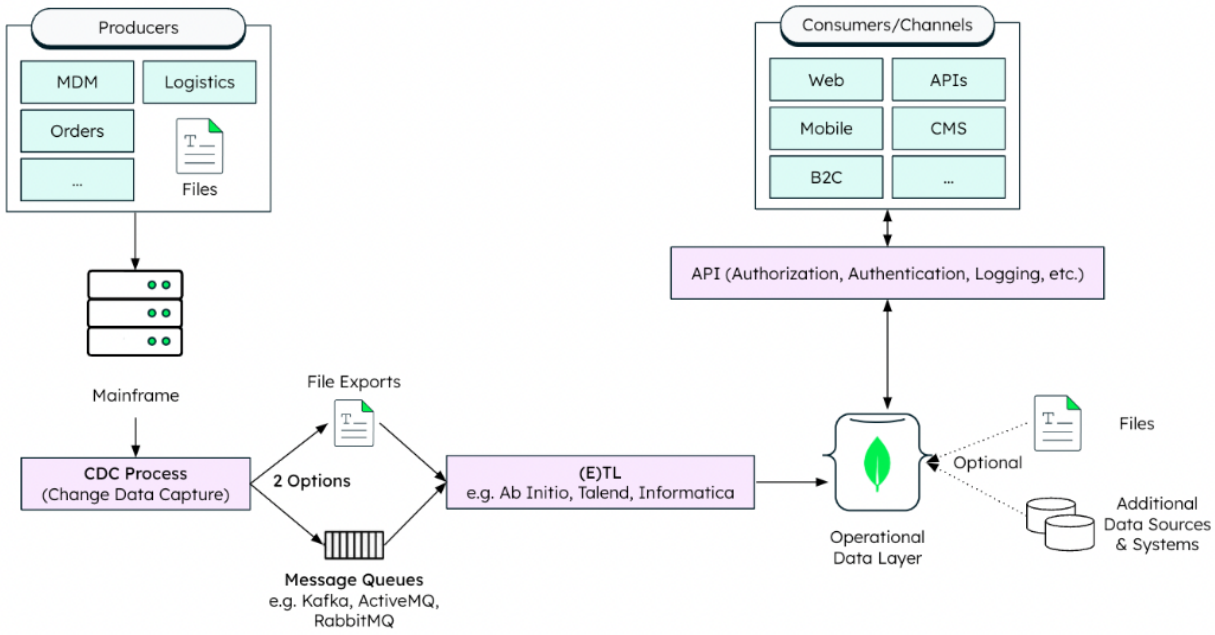


Figure 1. Example of high-level mainframe modernization architecture

Meeting regulatory demands

Organizations are challenged with meeting new regulatory demands cost-effectively and mitigating risk. For example,

- The open banking initiative and the revised payment services directive (PSD2) require exposing customer account information through open APIs to enable greater competition in financial services. Not only does this requirement increase the load on the system, but it also allows potential competitors to directly access the mainframe running your core business processes.
- The EU's General Data Protection Regulation (GDPR) and other privacy regulations demand that only aggregated views of customer data are exposed to prevent disclosing personally identifiable information (PII). An ODL can serve these views while being isolated from raw customer data stored on the mainframe.
- Mainframe downtime resulting from an outage or scheduled maintenance activities – even for a short period of time – can trigger fines in regulated industries such as financial services and telecommunications. Downtime also negatively impacts customer experience, especially if serving the new digital applications discussed earlier, creating a risk of churn.

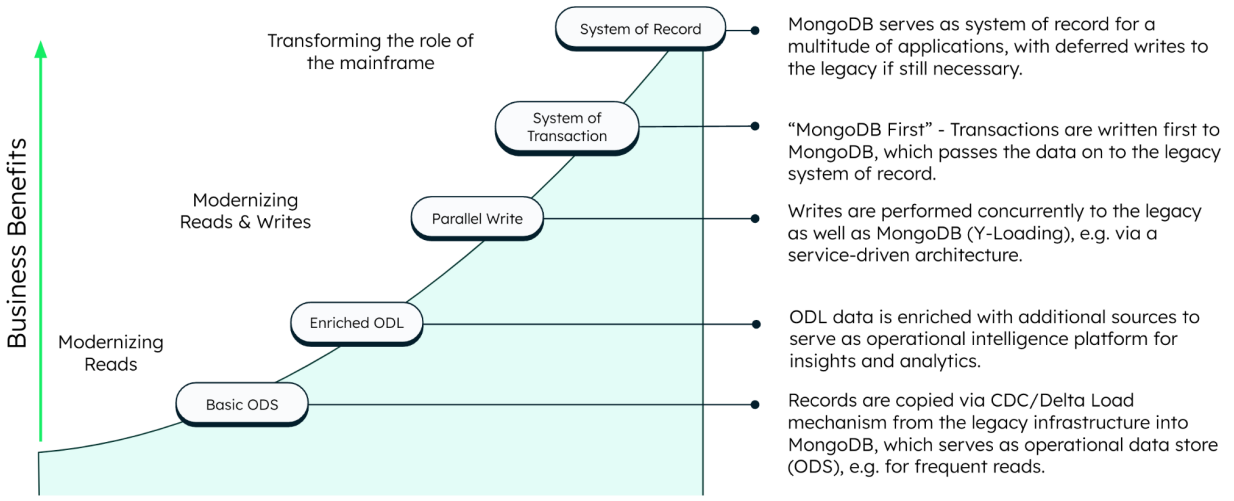


Figure 2. The 5 main phases of mainframe modernization

In the next section of this reference architecture, we will present a set of common approaches and patterns that enable mainframe modernization. We will then discuss the required capabilities of the ODL and present a selection of successful customer mainframe modernization projects.

Common Approaches and Patterns

The main phases of mainframe modernization

Regardless of the degree to which the mainframe is eventually modernized, the end goal is commonly implemented in stages over time, as illustrated at a high level in Figure 2.

Modernization reads

Initial use cases primarily focus on modernizing costly read operations, e.g., highly concurrent, self-service querying of customer data across new digital channels, reading large numbers of transactions for analytics, or retrieving historical views across customer data. Because mainframe data can often be static in nature once it is written, a considerable portion of reads can be modernized to the ODL with relative ease. Using either a change data capture (CDC) or delta load mechanism – described in detail in further sections below – to move data, you can create an ODL alongside the mainframe that can serve read-heavy operations.

This step results in a significant amount of read traffic being modernized from the mainframe and redirected into MongoDB. Depending on the use case, this can vary from 10% to 50% of read traffic reduction on the mainframe. It must be noted that at this stage, all write traffic still goes to the mainframe.

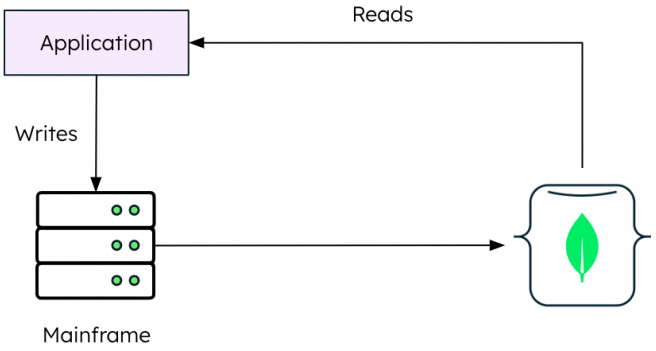


Figure 3: Read modernizing, with writes still directed to the mainframe

Modernizing reads and data enrichment

A logical next step is to enrich the data in the ODL by decorating it with metadata, or merging it with other adjacent data sources – a typical use case being to more efficiently enable advanced analytics or create a "single customer view." For example, financial transaction data could be subsequently categorized by enriching it using external, 3rd party information such as utility bill payments, retail purchases, etc. A retail bank could, for instance, then make it much easier for their customers to determine their spending on each category, i.e. on utilities over the last n months.

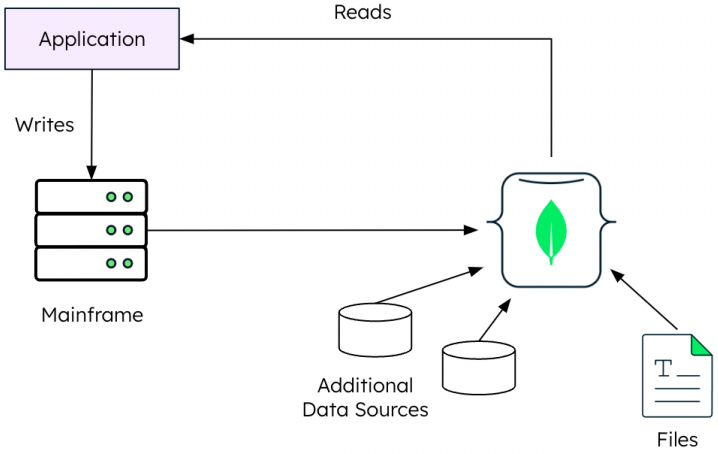


Figure 4: Read modernizing with data enrichment

Enrichment can potentially result in even more use cases being modernized from the mainframe. The degree to which this is feasible is obviously determined by the specific application, but it can

vary between 25% to 75% of read traffic modernized from the mainframe. Note that, as before, all write traffic still goes to the mainframe.

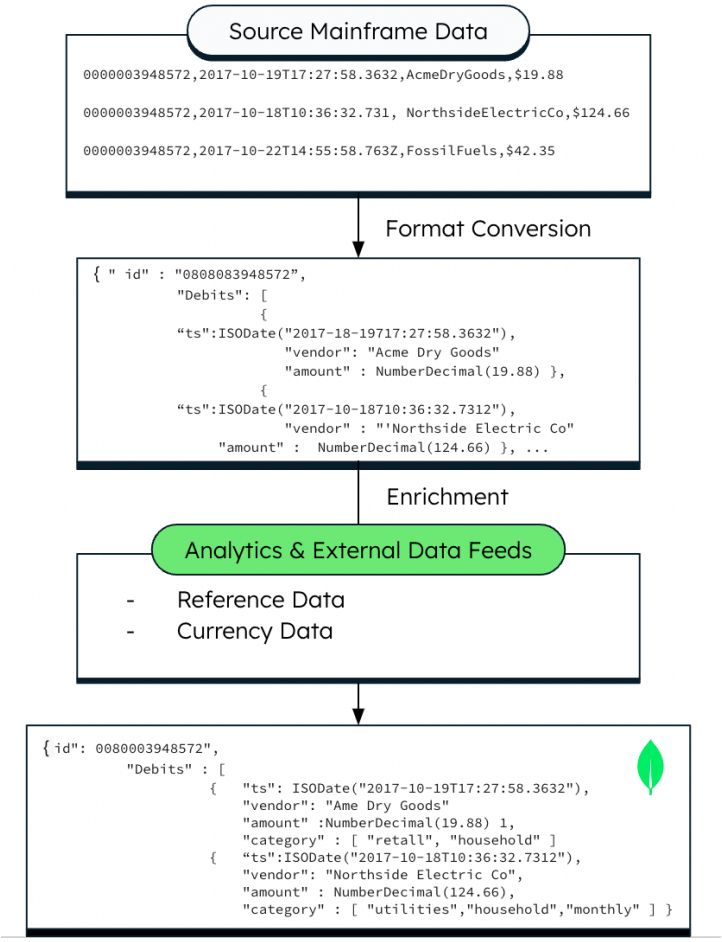


Figure 5: Document sample: Modernizing and enriching mainframe data to create enhanced customer experiences

Modernizing reads and writes

By introducing a smarter architecture to orchestrate writes between both the mainframe and the MongoDB ODL concurrently, it is possible to augment the change data capture (CDC) or delta load mechanisms we describe below. And of course, those writes against MongoDB can be protected with multi-document ACID transactional guarantees.

This can introduce a number of significant advantages, both technical and commercial:

- Real-time view of the data - ODL users are immediately consuming the newest version of the data, rather than waiting for updates to propagate from source systems to the ODL.
- Reduced application complexity - read and write operations no longer need to be segregated between different systems.
- Enhanced application agility - with traditional relational databases running the source systems on the mainframe, it can take weeks or months' worth of developer and database administrator (DBA) effort to update schemas. ORMs, and application code to support new business functionality. MongoDB's flexible data model with a dynamic schema makes the addition of new fields a runtime operation, allowing organizations to evolve applications more rapidly.

This pattern is often also referred to as "Y-loading" and can help lay the foundations for a more transformational shift of the role of the mainframe in your enterprise architecture. Moreover, only those writes that must be exposed to legacy mainframe applications need to be written back onto the mainframe. As a result, further savings can be realized, as new applications that would otherwise have had to access the mainframe can now operate against the ODL.

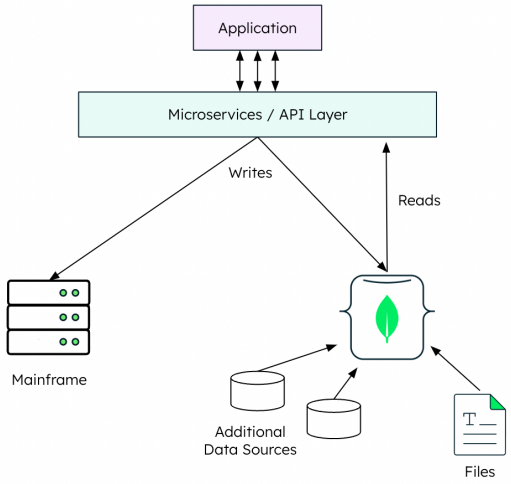


Figure 5: Read and write modernizing, with dual write (Y-loading) approach

Typically 40% to 80% of reads go to MongoDB. Additionally, some of the writes are now against data that is not stored in the mainframe, which results in a write modernizing of, for example, 10% to 25%.

Transforming the role of the mainframe:"ODL-first"

With a shift towards writing to MongoDB first, before writing to the mainframe (if at all), you are further changing the meaning of "system

of record" and "mainframe" within the organization. This also further simplifies the architecture, possibly eliminating the need for distributed transactions that may otherwise be required to guarantee data consistency between the mainframe and ODL

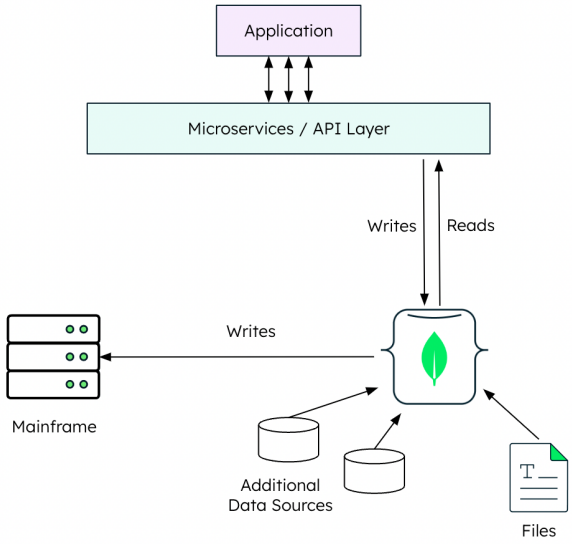


Figure 6: Read and write modernizing with ODL as initial target for application write traffic

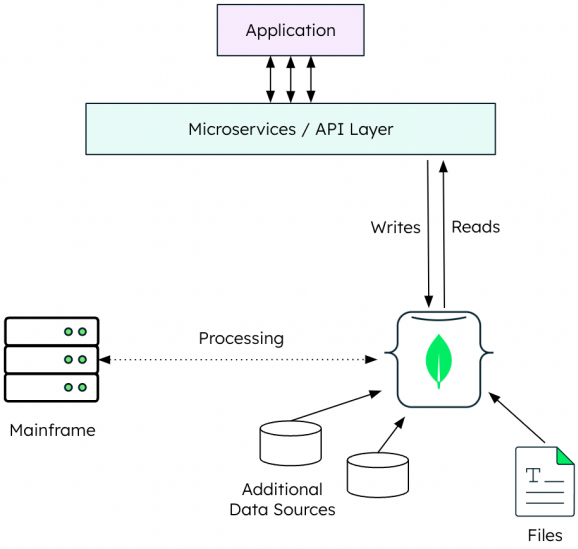


Figure 7: ODL as System Of Record

In this architecture, transactions are first executed on MongoDB, before they are executed on the mainframe. The estimated saving here is that 60% to 90% of all reads go to MongoDB.

Taken a step further, the MongoDB ODL can evolve to serve as the system of record, with writes only optionally being passed on to the mainframe solely to serve downstream legacy app dependencies. And as those legacy applications themselves are modernized, there is the possibility of decommissioning the mainframe entirely.

The estimated saving here is that 90% to 100% of all reads go to MongoDB. At this stage, all writes are directed against MongoDB, with only some being propagated back to the mainframe because of legacy application requirements. Depending on the use cases, this approach may improve write savings in the range of 50% to 90%.

Enabling Technologies for Mainframe Modernization

Regardless of how far an organization is intending to go in its mainframe transformation journey – whether it is just modernizing read operations or writes - there are a number of foundational technologies and enabling patterns that are essential to delivering a successful project. See figure 1 earlier in the paper for a high-level view of the reference architecture to support mainframe modernization.

When making choices about technologies and approaches for modernizing, requirements around the timeliness of data will factor heavily into the decision process. For example, does the organization require data to be synchronized in real-time (often sub-second, or whatever "real-time" means to the business), or will batch updates suffice, perhaps on an hourly or daily basis? In many cases, there may be different approaches for ODL "bootstrapping" (i.e., batch) versus subsequent delta loads (i.e., "real-time" stream/ queue-based synchronization).

Change data capture, queuing and ETL

Fundamental to successfully mirroring data from the mainframe into an ODL is the ability to detect changes to the relevant source data stored on the mainframe. However, as highlighted above, there is often also a need to replicate modifications made to data in the ODL back into the mainframe.

CDC is essentially the act of:

- Identifying data manipulation language (DML) changes to the database (inserts, updates, deletes).
- Capturing the specific data change(s).
- Triggering some action(s) using the changed data.

Depending on the type of mainframe, there may be a number of CDC mechanisms available, with some mainframe databases providing native capabilities such as IBM InfoSphere CDC for Db2. Other commercial options include Ab Initio or Attunity Replicate.

In some cases, however, CDC may be a completely manual process, requiring active programmatic monitoring of the underlying database's logging and auditing capabilities.

Within the context of a mainframe modernization project, a queuing system or message broker, such as Kafka, is often used to decouple the source mainframe system from the destination system (the MongoDB ODL) and to allow for greater deployment flexibility.

Messaging systems are also typically deployed in a highly available topology, distributed across several nodes for redundancy and scale.

The key benefits of putting a message queue between the CDC system and the operational databases are increased resiliency and deployment agility. Increased resiliency is provided by using a durable pub/sub queue as a caching mechanism, which can also assist with consistency and/or reconciliation concerns between the source mainframe and target ODL. Deployment agility allows for other related systems to easily 'produce' or 'consume' data into or from the queue to expand the available services. This is extremely useful when developing modern agile applications where new requirements may be added, or existing functionality modified, with very little notice.

You can learn more about using MongoDB and Kafka here.

An alternative solution to that based on queuing is to use a more traditional ETL-based

approach. Coming in various flavors, both proprietary commercial and freely available open-source, ETL tools are often capable of leveraging not only simple batch-based file transfer approaches but also CDC approaches.

Commonly used ETL tools include Talend, Ab Initio, Attunity Replicate, and Informatica.

Synchronizing writes into the mainframe with change streams

If MongoDB is to be used to ingest writes and then push them back into the mainframe, then change streams provide another potential solution for cross-platform synchronization. In the context of mainframe modernization, change streams can be used to propagate write operations from MongoDB to the mainframe. Change streams also enable developers to build real-time apps that can view, filter, and react to data changes as they occur in the MongoDB database. It makes it simple to stream data changes and trigger actions wherever they are needed.

Implemented as an API on top of MongoDB's operation log (oplog), consumers can open change streams against collections and filter on relevant events using the $match, $project, and $redact aggregation pipeline stages. The consumer can register for notifications whenever a MongoDB document or collection is modified, enabling the mainframe to apply new data in real-time. These changes can be consumed from directly or via a message broker such as Kafka.

Exposing data from the MongoDB ODL

Once the mechanism for the replication of changes to MongoDB ODL is online, it is ready to be exposed to consumers. This is typically done through APIs, such as RESTful web services, that abstract access to the underlying data of the ODL.

In modern microservice, it is good practice to use the API approach as any number of consuming applications – whether customer-facing web and mobile services or backend enterprise and analytics applications – can be repointed to the web service API with no or minimal modification to the application's underlying logic and code. Native MongoDB drivers can be used to write such APIs and custom modules. These drivers are idiomatic and support all popular programming languages and frameworks to make development fast and natural. Available drivers include Java, Javascript, .NET, Python, Perl, PHP, Scala and others, in addition to 30+ community-developed drivers.

MongoDB Atlas App Services can be used to further accelerate the development of an API layer to the ODL. App Services provides a suite of managed cloud services including Atlas Device sync, serverless cloud functions, declarative access rules, flexible Data API, GraphQL API and more. You can use App Services to write and host a full application in a managed cloud environment backed by MongoDB Atlas - getting your apps to market faster while reducing operational costs and efforts.

Your existing applications access their data using the same MongoDB drivers, but new applications can use new RESTful APIs built using features of App Services:

- Create the API using App Services HTTP service and webhooks.
- Control data access using rules; these rules can easily interact with existing authentication systems.
- App Services functions can be used to enrich the data to meet new application requirements.

- JSON is the universal data format for applications, RESTful APIs, MongoDB, and MongoDB App Services. Because of this, there is no need to waste development and runtime resources converting between formats.

As a fully-managed serverless platform running in the cloud, no additional infrastructure is required, and Atlas App Services automatically scales as the use of the ODL's APIs increases. Learn more about Atlas App Services here.

The following sample code example is a Java snippet showing how to implement a Kafka listener which can process messages from a Kafka queue and insert corresponding documents into MongoDB

Required Platform Capabilities

The data platform used to manage the ODL modernization from the mainframe provides the core technology foundation for the project, and is therefore critical to determining success or failure.

Relational databases, once the default choice for enterprise applications, are unsuitable for ODLs used in mainframe modernization use cases. The data layer is forced to simultaneously accommodate the schema complexity of all source mainframe databases, requiring significant upfront schema design effort. Any subsequent changes in any of the source systems' schema – for example, when adding new application functionality – will break the database schema. The schema must be updated, often causing application downtime. Onboarding new data sources from the mainframe multiplies the complexity of adapting the relational database's rigid, tabular schema.

Relational databases also struggle to meet the performance SLAs of the system. Typically, the mainframe data set will be normalized across multiple tables, which must then be JOINed to materialize the modernized data to the consumer. This process can add significant query latency while also inhibiting scalability as the ODL grows to onboard new data sources and serve new applications.

MongoDB and its document model provide a mature, proven alternative to relational databases for enterprise applications, including ODL projects. MongoDB is the leading developer data platform today. MongoDB is at the center of legacy modernization initiatives across a range of organizations, including Travelers, Cisco, eharmony, RBS, Sega, China Eastern, and many more.

As discussed below, the required capabilities demanded by a mainframe modernization project are well served by MongoDB, providing a technology foundation that enables development teams through:

1. The document data model – presenting them the **best way to work with data**
2. A distributed systems design – allowing them to **intelligently put data where they want it.**
3. A unified experience that gives them the **freedom to run anywhere**– allowing them to future-proof their work and eliminate vendor lock-in.

For more details on each of these areas, please refer to the MongoDB architecture guide.

The power of document-oriented databases

MongoDB is the pioneer of what has come to be called NoSQL databases, which developed because RDBMS systems based on SQL did not support the scale or rapid development cycles needed for creating modern applications.

NoSQL is an umbrella term; it includes document-oriented databases like MongoDB, columnar databases, in-memory databases, and more.

In MongoDB, records are stored as documents in compressed BSON files. The documents can be retrieved directly in JSON format, which has many benefits:

- It is a natural form to store data.
- It is human-readable.
- Structured and unstructured information can be stored in the same document.
- You can nest JSON to store complex data objects.
- JSON has a flexible and dynamic schema, so adding fields or leaving a field out is not a problem.
- Documents map to objects in most popular programming languages.

Most developers find it easy to work with JSON because it is a simple and powerful way to describe and store data.

Perhaps most importantly, the developer controls the database schema. Developers adjust and reformat the database schema as the application evolves without the help of a database administrator. When needed, MongoDB can coordinate and control changes to the structure of documents using schema validation.

MongoDB created Binary JSON format (BSON) to support more data types than JSON. This new format allows for faster parsing of the data. Data stored in BSON can be searched and indexed, tremendously increasing performance. MongoDB supports a wide variety of indexing methods, including text, decimal, geospatial, and partial.

Using MongoDB enables your team to go further and faster when developing software applications that handle data of all sorts in a scalable way.

Intelligent insights, delivered in real time

When designing the schema of a database, it is impossible to know in advance all the queries. An ad hoc query is a short-lived command whose value depends on a variable. Each time an ad hoc query is executed, the result may be different depending on the variables in question.

Optimizing the way in which ad-hoc queries are handled can make a significant difference at scale, when thousands to millions of variables may need to be considered. This is why MongoDB, a document-oriented, flexible schema database, stands apart as the cloud database platform of choice for enterprise applications that require real-time analytics. With ad-hoc query support that allows developers to update ad-hoc queries in real-time, the improvement in performance can be game-changing.

MongoDB supports field queries, range queries, and regular expression searches. Queries can return specific fields and also account for user-defined functions. This is made possible because MongoDB indexes the documents and provides a very powerful query mechanism, MongoDB Query Language (MQL).

Enabling business intelligence (BI) and advanced analytics

The ODL, among other things, serves as the foundation for analytics and BI. MongoDB provides integration with popular analytics frameworks and SQL-based BI tools to power advanced analytics and help you build insightful visualizations.

MongoDB Connector for BI and MongoDB Charts

We often see that enterprises use business intelligence tools, such as Tableau, MicroStrategy, and Qlik, to visually analyze and understand their data. These tools expect to see the data in tabular format. The MongoDB Connector for BI lets you use such SQL- based BI and analytics tools without sacrificing the benefits of MongoDB's flexible data model to store data in rich, multi-dimensional documents and quickly build new functionalities.

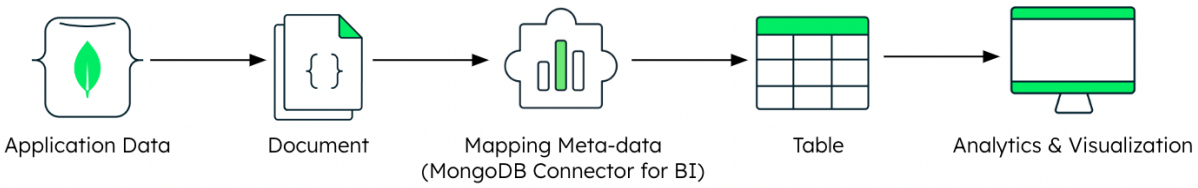


Figure 9: MongoDB Connector for BI to integrate with relational BI tools

The MongoDB Connector for BI acts as a layer that passes queries and data between a MongoDB instance and your reporting tool. It stores no data and purely serves to bridge your MongoDB server with business intelligence tools. To learn more, read the documentation here.

For users who do not have an existing investment in BI tools, MongoDB Charts is a tool to create visual representations of your MongoDB data. Because Charts natively understands the MongoDB document model, you can create charts from data that varies in shape or contains nested documents and arrays, without needing to first map the data into a flat, tabular structure.

Using Charts, you can quickly and easily visualize your data, place multiple charts onto a single dashboard, and then share that dashboard with key stakeholders to support collaborative analysis and decision making. When you connect to a live data source, MongoDB Charts will keep your visualizations and dashboards up-to-date with the most recent data. Charts will automatically generate an aggregation pipeline from your chart design, which is then executed on your MongoDB server. With MongoDB's workload isolation capabilities – enabling you to separate your operational from analytical workloads in the same cluster – you can use Charts for a real-time view without having any impact on production workloads.

MongoDB Connector for Apache Spark

An increasing number of businesses are leveraging Apache Spark with MongoDB to perform advanced analytics. The MongoDB Connector for Apache Spark makes it easy and efficient. The connector exposes all of Spark's libraries, including Scala, Java, Python, and R. MongoDB data is materialized as DataFrames and Datasets for analysis with machine learning, graph, streaming, and SQL APIs.

The MongoDB Connector for Spark can also take advantage of MongoDB's aggregation pipeline and rich secondary indexes to extract, filter, and process only the range of data it needs – for example, analyzing all customers located in a specific geography. This is very different from less mature non-tabular data stores that do not offer either secondary

indexes or in-database aggregations. In these cases, Apache Spark would need to extract all data based on a simple primary key, even if only a subset of that data is required for the Spark process. This means more processing overhead, more hardware, and longer time-to-insight for the analyst.

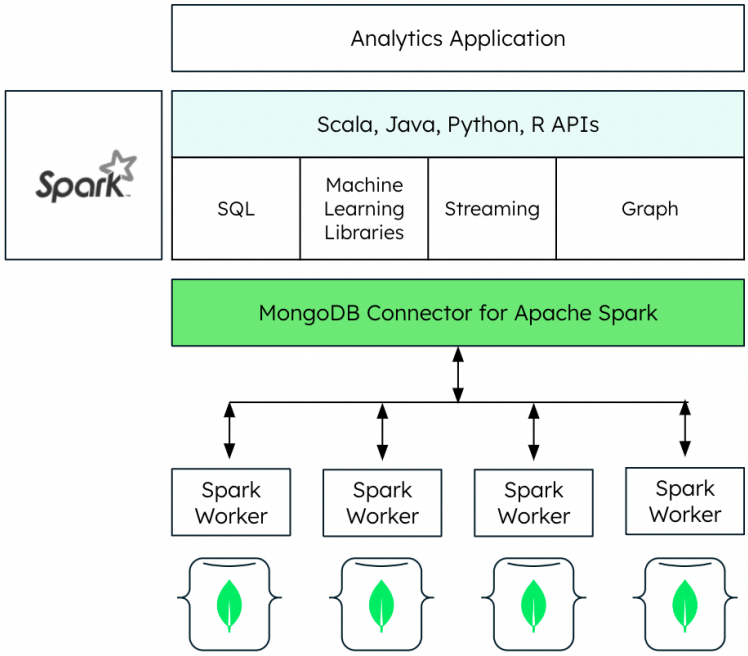


Figure 10: Enable advanced analytics MongoDB Connector for Spark

To further maximize performance across large, distributed datasets, the MongoDB Connector for Apache Spark can co-locate resilient distributed datasets (RDDs) with the source MongoDB node, thereby minimizing data movement across the cluster and reducing latency. To learn more about the MongoDB Connector for Apache Spark refer to the MongoDB Spark Connector documentation here.

Please note that if you are running analytics, it is a good practice to provision MongoDB replica sets with dedicated analytics nodes. This allows data scientists and business analysts to simultaneously run exploratory queries and generate reports and machine learning models against live data without impacting nodes serving the modernized mainframe data to operational applications.

Intelligently distribute data for scale and availability

Successful mainframe modernization projects can demand rapid scalability of the ODL. As new data sources and attributes, along with additional consumers such as applications, channels, and users, are onboarded, the demands for processing and storage capacity quickly grow.

To address these demands, MongoDB provides horizontal scale-out for the ODL platform on low-cost, commodity hardware using a technique called sharding, which is

transparent to applications. Sharding distributes data across multiple database instances. Sharding allows MongoDB deployments to address the hardware limitations of a single server, such as bottlenecks in CPU, RAM, or storage I/O, without adding complexity to the application. MongoDB automatically balances modernized mainframe data in the cluster as the data set grows or the size of the cluster increases or decreases.

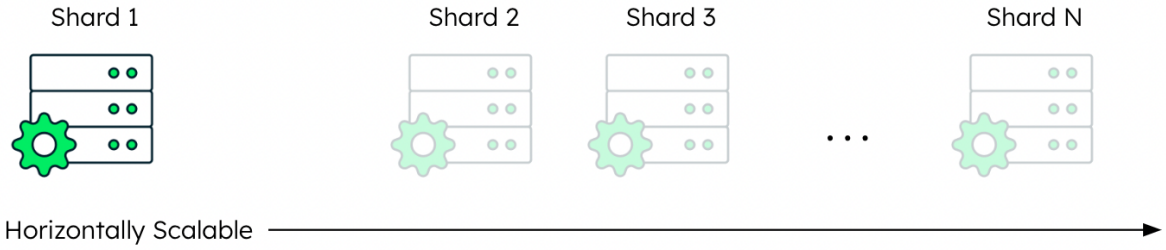


Figure 11: MongoDB scale-out as the ODL grows

MongoDB maintains multiple replicas of the data to maintain database availability. Replica failures are self-healing, and so the ODL remains unaffected by underlying system outages or planned maintenance. Replicas can be distributed across regions for disaster recovery and data locality to support global user bases. Find out more about scalability with MongoDB Atlas here.

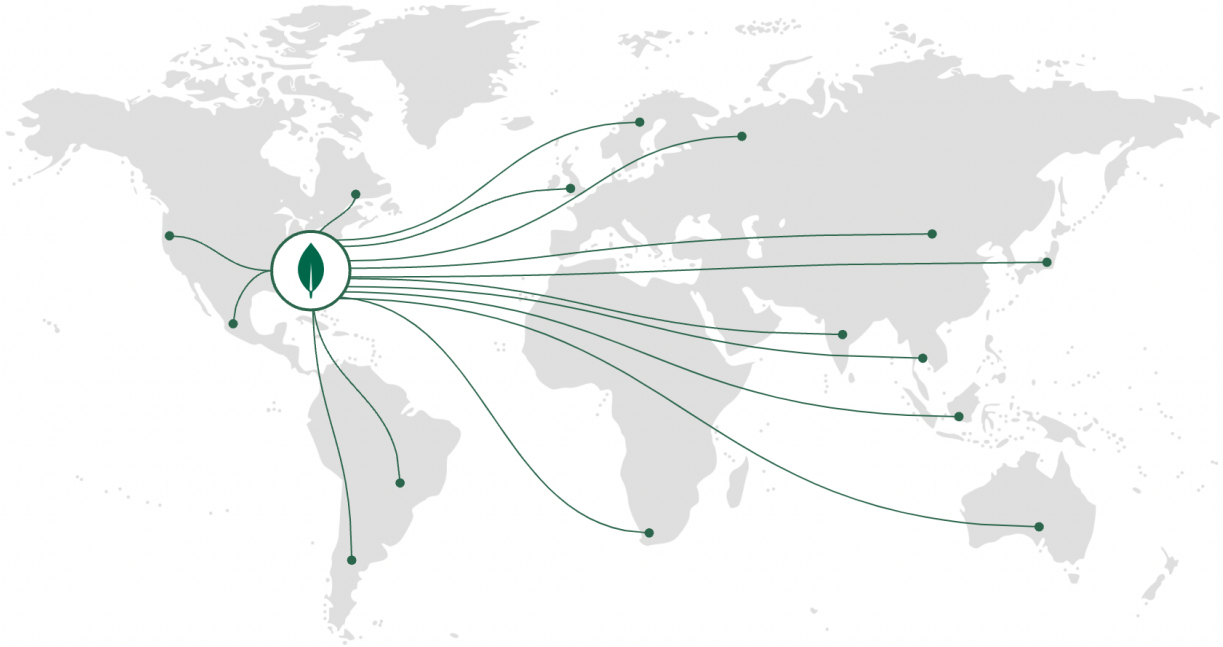


Figure 12: Global distribution of the ODL

Freedom to run anywhere: Enterprise deployment model

MongoDB can be run on a variety of platforms – from commodity x86 and ARM-based servers, through to IBM Power and even IBM zSeries mainframes, which can be useful for those who aren't ready to modernize apps, but want to evaluate a modern data platform on their existing hardware. You can deploy MongoDB on servers running in your own data center or in public and hybrid clouds. With the MongoDB Atlas service, we can run the ODL for you on any of the leading cloud provider platforms.

MongoDB Enterprise Advanced is the production-certified, secure, and supported version of MongoDB, offering:

- **Advanced security**. Robust access controls via LDAP, Active Directory, Kerberos, x.509 PKI certificates, and role-based access controls to ensure a separation of privileges across applications and users. Data anonymization can be enforced by read-only views to protect sensitive, personally identifiable information. Data in flight and at rest can be encrypted to FIPS 140-2 standards, and an auditing framework for forensic analysis is provided. You can learn more from the MongoDB Security Architecture.
- **Automated deployment and upgrades** . With Ops Manager, operations teams can deploy and upgrade distributed MongoDB clusters in seconds, using a powerful GUI or programmatic API.
- **Point-in-time recovery**. Continuous backup and consistent snapshots of distributed clusters allow seamless data recovery in the event of system failures or application errors without the latency of rehydrating the ODL from the source mainframe systems.

MongoDB Atlas: Fully-managed, on-demand cloud service

An increasing number of companies are moving to the public cloud to not only reduce the operational overhead of managing infrastructure, but to also provide their teams with access to on-demand services that give them the agility they need to meet faster application development cycles. This move from building IT to consuming IT as a service is well aligned with parallel organizational shifts, including agile and DevOps methodologies and microservices architectures. Collectively, these seismic shifts in IT help companies prioritize developer agility, productivity, and time to market.

MongoDB offers the fully managed, on-demand, and elastic MongoDB Atlas service, in the public cloud. Atlas enables customers to deploy, operate, and scale MongoDB databases on AWS, Azure, or GCP in just a few clicks or programmatic API calls. MongoDB Atlas is available through a pay-as-you-go model and billed on an hourly basis. It's easy to get started - use a simple GUI to select the public cloud provider, region, instance size, and features you need. MongoDB Atlas provides:

- Automated database and infrastructure provisioning so teams can get the database resources they need, when they need them, and can elastically scale whenever they need to.
- Security features to protect your data, with network isolation, fine-grained access control, auditing, and end-to-end encryption, enabling you to comply with industry regulations such as HIPAA.
- Built-in replication both within and across regions for always-on availability.
- Global sharding allows you to deploy a fully managed, globally distributed database that provides low latency,

responsive reads and writes to users anywhere, with strong data placement controls for regulatory compliance.
- Fully managed, continuous and consistent backups with point in time recovery to protect against data corruption, and the ability to query backups in-place without full restores.
- Fine-grained monitoring and customizable alerts for comprehensive performance visibility.
- Automated patching and single-click upgrades for new major versions of the database, enabling you to take advantage of the latest and greatest MongoDB features.
- Live migration to move your self-managed MongoDB clusters into the Atlas service or to move Atlas clusters between cloud providers.
- Widespread coverage on the major cloud platforms with availability in over 50 cloud regions across Amazon Web Services, Microsoft Azure, and Google Cloud Platform. MongoDB Atlas delivers a consistent experience across each of the cloud platforms, ensuring developers can deploy wherever they need to, without compromising critical functionality or risking lock-in.

MongoDB Atlas can be used for everything from a quick proof of concept, to dev/test/QA environments, to powering production applications. The user experience across MongoDB Atlas and on-premise deployments is consistent, ensuring that you easily move from your own facilities to the public cloud and between providers as your needs evolve.

Executing mainframe modernization: Application modernization factory

Beyond the ODL platform technology, MongoDB has worked with many companies to support mainframe modernization initiatives. The application modernization factory (AMF) is a professional services engagement that provides advisory consulting, program governance, and application lifecycle expertise.

Working with MongoDB consultants, the first step in the AMF process is to identify application stakeholders and then build an inventory and characterization of existing apps, before identifying the best-fit candidates for modernization. From there, we scope the project, quantify the economic value of change, and provide a roadmap for delivery.

We support the modernization of applications throughout the software development lifecycle, harnessing patterns and technologies such as agile and DevOps, microservices, cloud computing, and MongoDB best practices. We partner with your teams to accelerate the assessment, prioritization, and redesign of legacy apps and work with them through the modernization efforts of redevelopment, consolidation, and optimization. To learn more, review the MongoDB Legacy Modernization page.

Mainframe Modernization in Action

Global retail and investment banking group

The bank has reduced its reliance on its legacy mainframe systems by modernizing customer data to a new agile, scalable, and resilient ODL built on MongoDB.

Like many large banks, it faced a significant challenge in meeting customer demand for new digital banking services

while relying on legacy infrastructure. In addition, with many core banking applications running on the mainframe, the platform itself had become a single point of failure. Two mainframe outages in the previous 12 months prevented customers from making payments. Not only had this resulted in reputational brand damage, but it also caught the attention of industry regulators.

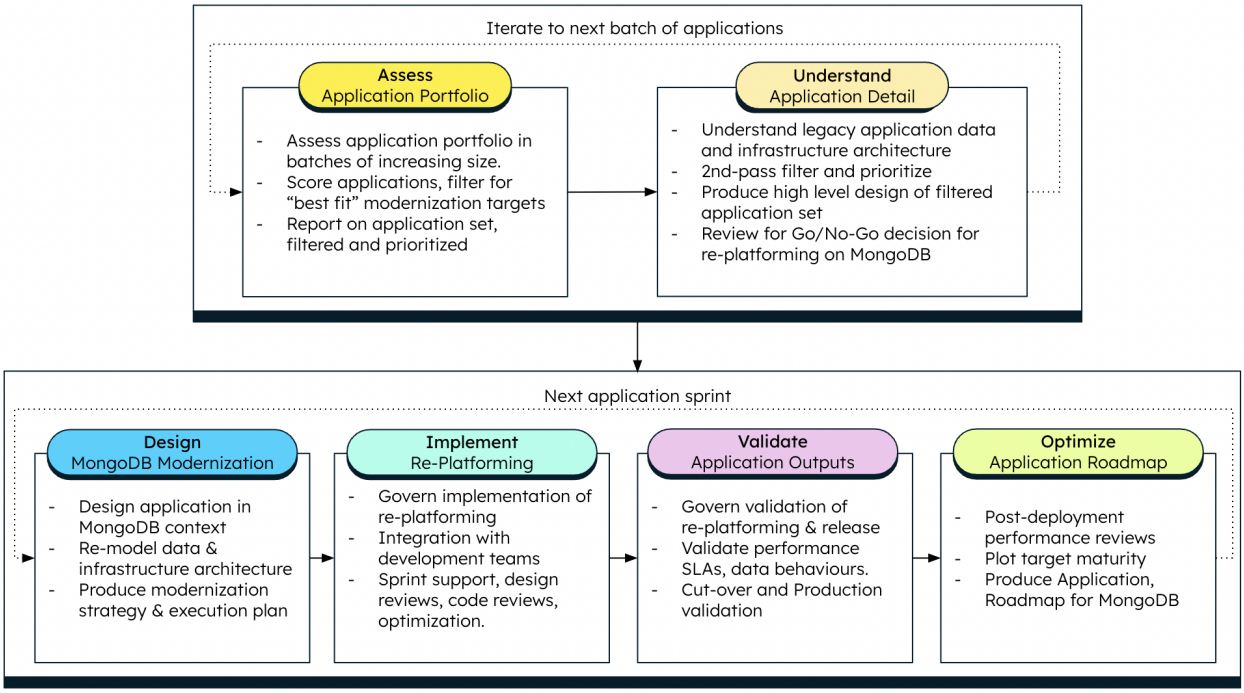


Figure 13: MongoDB application modernization factory accelerating and de-risking mainframe modernization

To mitigate future risk, the bank explored ways it could make customer data accessible even when the mainframes were down. An analysis of customer journeys through the bank's digital channels revealed that 92% of all traffic was generated by just 25 interaction types, and 85% of these were read-only, i.e., for customers to view their balances and review transactions. This analysis confirmed that creating a new real-time, synchronized, read-only copy of customer account data in an ODL was a viable approach to improving resilience.

As the bank began the project of modernizing customer data to MongoDB, it realized it was now able to enrich the original service provided by the mainframe. Rather than display just the last 300 transactions for each account – which is too limited for business customers – it could now efficiently retrieve the entire transaction history, and provide customers with rich search over the data.

Building upon these early successes, the bank has started to create entirely new services on top of the ODL. For example, landing pages required 15 or more expensive queries across the backend mainframe databases to present a single view to the customer of their personal accounts, savings, mortgages, and more. This is now being served by a single query to the

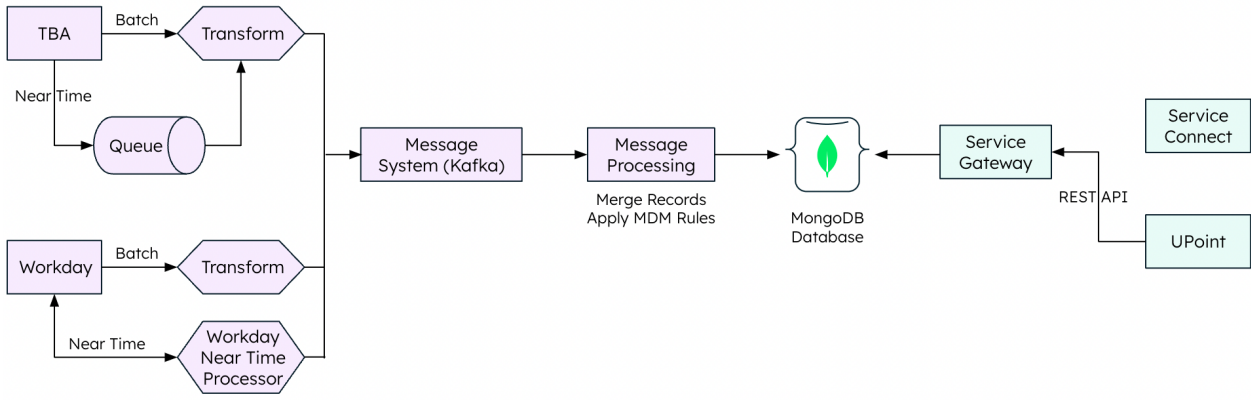
ODL, increasing performance to improve customer experience, and reducing mainframe costs.

In addition to single view, the bank is using the ODL to present personalized offers to its customers when they log into their accounts. The ODL also allows the bank to better prepare for the new PSD2 payments services directive by enabling customer data to be accessed via APIs to third-party platforms, for example, checking account balances via Facebook.

In summary, the mainframe modernization project has enabled the bank to move from using the ODL purely as a system for resilience to one where the ODL is now the system of innovation. The bank has been able to reduce costs and risk while enabling new digital initiatives that were just not possible on the mainframe. You can learn more from the press coverage on the project.

Alight Solutions (Aon Hewitt) mainframe modernization and single view

Alight Solutions, formerly part of Aon PLC, is the leading provider of outsourced benefits administration, cloud-based HR and financial solutions, serving close to 40 million employees from over 1,400 organizations, including nearly 50% of the Fortune 500.



Phase 2: Real time and batch data pipeline to create a single view with Kafka & MongoDB. (Image courtesy of Alight Solution's MongoDB World presentation)

To meet the demands of the business, the architecture team was tasked with building a cross-platform single customer view to unlock greater data insights and improve application responsiveness while also reducing mainframe costs. Retrieving customer data from frontend consumer interfaces such as Salesforce, the UPOINT HR portal, and mobile apps meant accessing multiple backend source systems running on a mainframe platform and Workday. Each request added to the ongoing mainframe MIPS costs, and scaling multiple systems to support customer growth was proving difficult. Query latency was typically 0.5 of a second, which negatively impacted the customer experience.

After researching multiple options, the Alight team decided MongoDB was the best choice for its single view platform, hosting data modernized from the mainframe:

- With its flexible document data model, MongoDB offered the schema flexibility needed to accommodate multi-structured,

polymorphic customer data from the source systems.
- Its expressive query language and secondary indexes enabled business users to access customer data in whichever way they needed.
- MongoDB distributed, self-healing architecture enabled Alight to operationalize the single view, ensuring high scalability and always-on availability with minimal overhead to the ops team.

Application performance was a key driver for the project. With customer data aggregated into a single document, query latency was reduced from 500 milliseconds when accessing the mainframe to less than 2 milliseconds from MongoDB, representing a 250x improvement!

The project has been delivered over three phases. From the initial phase that delivered a working solution in less than six months, through to the current phase that provides more granular insights into each customer's benefit classes.

In phase 1, data was extracted from the mainframe in batches, transformed in Hadoop, and then loaded into MongoDB. Phase 1 was a vital first step in demonstrating the possibilities of mainframe modernization and building a single view, but Hadoop's fragility and reliance on multiple independent technologies complicated the data transformation process. In addition, change data capture was slow, and the data merging and matching rules were too simplistic, affecting data quality.

In phase 2, Alight replaced Hadoop with Apache Kafka, enabling changes in source data to be sent as messages in both batches and in real-time streams. Data could then be merged by applying more sophisticated master data management (MDM) rules, before being loaded into MongoDB, where it was exposed to consuming systems via a REST API.

In the third phase, Alight engineers redesigned their data structures to model customer data by benefit class, enabling greater deployment flexibility for scalability and high availability. Data versioning was also added to track changes to each customer record over time.

The benefits the single view project has delivered to the business so far include:

- **Customer experience:** faster responsiveness, and ease in delivering new functionality.
- **Lower costs**: fewer calls to the mainframe, and reductions in peak consumption.
- **Unlocked innovation:** greater insight into customer data, and improved agility in delivering new services.

To learn more, take a look at how Alight Solutions (Aon Hewitt) improved customer experience by over 250x with mainframe modernization and single view here.

Conclusion

Modernizing data into an ODL is a challenging undertaking. However, by partnering with a vendor that combines proven methodologies, tools, and technologies, organizations can innovate faster with lower risk and cost. MongoDB is that vendor.

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)MongoDB Realm (mongodb.com/realm)

Legal Notice

This document includes 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 financial guidance for the first fiscal quarter and full year fiscal 2021; the anticipated impact of the coronavirus disease (COVID-19) outbreak on our future results of operations, 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 press release 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: our limited operating history; our history of losses; failure of our database 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; 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 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; the price volatility of our common stock; the financial impacts of the coronavirus disease (COVID-19) outbreak on our customers, our potential customers, the global financial markets and our business and future results of operations; the impact that the precautions we have taken in our business relative to the coronavirus disease (COVID-19) outbreak may have on our business and those risks detailed from time-to-time under the caption "Risk Factors" and elsewhere in our Securities and Exchange Commission ("SEC") filings and reports, including our Quarterly Report on Form 10-Q filed on December 10, 2019, as well as future filings and reports by us. 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.