How MongoDB Enables Real-Time Data with Event-Driven Architecture

October 2019

Table of Contents

Introduction

The world moves in real time, and the demands on individuals, groups, and organizations are constantly changing. These truths may appear self-evident, but many enterprises are still trying to meet their data needs in ways that don't match this new reality. To keep up with the pace of business, companies must become agile organizations fueled by applications that access, analyze, and distribute data in real time. Those applications must run on a data fabric that offers the elasticity to scale up and down as needed, efficiently and cost-effectively empowering people with the insights they need to do their jobs better and thus help their companies compete.

A traditional request-driven data architecture is, conceptually at least, an anti-pattern to this new imperative. Request-driven architectures, true to their name, require users to make requests and wait until the requested information becomes available. By contrast, an event-driven architecture proactively makes a stream of data from source systems available in real time. Consuming applications and services subscribe to topics of interest and consume the data at their own pace. To put it another way, request-driven architectures are system-focused, while event-driven architectures are user-focused, providing timely, contextual insights.

This paper discusses why event-driven architectures are the natural evolution of how the world stores and accesses data, shows how MongoDB can help your organization establish event-driven architecture, and offers named examples of customers working with MongoDB to adopt this model and run their businesses in real time.

What Has Changed: Operating in Real Time

Charles W. Bachman designed what is widely considered the first database management system (DBMS) in 1960, his Integrated Database System. In 1970, Edgar F. Codd wrote a series of papers on ways to construct databases that coalesced into a paper titled A Relational Model of Data for Large Shared Data Banks,[1] and the years that followed saw the emergence of the first commercial relational databases. For decades the purpose of databases was simply to collect data, and relational DBMSs did what they were designed to do: store structured data, such as simple customer records, that was easily organized into tabular formats and readily queried with SQL.

Although its origins date back to the 1980s, the Internet became widely available in the early to mid 1990s. It revolutionized the way the world lives, works, and plays, and in the process exposed the inherent limitations of traditional system architectures, especially relational databases: they could not keep up with the sheer amount of data coming at them, and were never designed to manage the unstructured data generated from Web, mobile, social and Internet of Things (IoT) applications. So the focus shifted to addressing the limitations of the RDBMS by expanding to accommodate these new data types and sources, thus addressing two of the so-called "three Vs of data": volume and variety. The first NoSQL databases emerged from places like Google and Amazon in 2004-05. Companies began collecting as much data as they could afford to store, and shortly thereafter the term "big data" appeared, describing massive datasets that were impossible to process and manage using traditional data management tools.

[1] Dataversity, *A Brief History of Database Management,* available here

Somewhere along the way the world figured out that 'big data' could also mean 'slow data.' Ironically, and ominously, while the pace of business was accelerating, sitting on massive amounts of data was slowing organizations down, leaving them less equipped to keep up. Simply solving for volume and variety left databases with a major blind spot: the third of the three Vs, velocity.

So today the focus has shifted again, this time to address all three Vs through stream processing of all relevant data, of all structures and from all sources, to enable businesses to operate in real time. General agreement exists that real time is one of three data speeds, but definitions vary:

- The most commonly agreed definition is that real time is the ability to ingest and act on data in 100-200 milliseconds; near real time, anything up to 10 seconds; and batch, anything over 10 seconds. Some, however, believe the bar for real time is (conceptually) higher: 15-20 milliseconds.
- The computer science definition of real time is merely that a given operation is guaranteed to execute within a defined duration.
- Beyond one-dimensional benchmarks of speed, other considerations enter into the picture. Chief among them: does a system deliver real-time data when it is being accessed by a realistic number of concurrent users — or, using a convenient benchmark favored by some providers, when accessed by a single user?

Bottom Line on Real Time: You Need Faster Data

While speeds and feeds are important indicators of the capabilities of a database or platform, these varying definitions have clouded the issue. The pace of business today, however, is clarifying matters in a hurry. If your organization wants or needs to take any action and that action is held up waiting for data, you need faster data. Prosperity and survival dictate that you can no longer afford to wait on data to arrive. Your people, wherever they are at any moment, need real-time data, and real-time analytic insights, to do their jobs better. Everything they do contributes to (or detracts from) customer experience – which is fast becoming the most important competitive differentiator in the digital economy, surpassing price and product as the key differentiator for your brand.[2]

The Major Driver of Real Time Today is Microservices

The key driver today in making the need for real-time data an organizational imperative is the emergence of microservices. Your ability to achieve the benefits of real-time data depends on the ability of your development organization to rapidly build and deploy new business applications by embracing agile ways of working, such as creating an agile and DevOps culture, and adopting a microservices architecture.

Microservices architectures break up monolithic applications into small, discrete services or functions, typically packaged and isolated within containers, which communicate with each other through well defined, network-based APIs. A microservices design approach creates self-sufficient sprint teams that are empowered to bring new capabilities online independently of each other, then over time evolve and upgrade them without impacting adjacent microservices. This helps you simplify, migrate, and manage complex data changes.

As beneficial as microservices can be, they require the ability to work with large volumes of data that

[2] Walker Research, *Customers 2020: A Progress Report,* cited here and available here

change frequently, and that is a challenge many existing systems cannot meet. Adopting a microservices architecture relies on a database with a flexible data model that can support these rapid changes, coupled with the scale needed to manage ever faster and larger data sets.

In short: things happen in real time. Organizations must now operate in real time. The challenge facing software architects is how to help their companies do so. The need has long existed to move data quickly between systems, but the emergence of microservices means that the ability to do so has never been more crucial than it is right now. And "quickly" has now accelerated to real time.

Core Building Block of Real Time is an Event

While various definitions of an event exist, for data management purposes an event has three discrete characteristics:

1. Atomic: something happened, and the event is the atomic unit representing what occurred.
2. Related: an event rarely stands alone; it is nearly always part of a stream or sequence of events.
3. Behavioral: these streams or sequences of events create an accumulation of facts that captures behavior.

The way, then, to enable companies to operate in real time is to help them capture, manage, and act on events as they occur. This is not to say that event management is merely reactive; events also become parts of broader sequences and patterns that are indicators of other things that are about to occur. The smart organization that captures and acts on events can predict and proactively prepare for what is to come. A cohesive model has emerged to enable organizations to best leverage events: event-driven architecture.

What Is Event-Driven Architecture?

As mentioned at the outset, conventional request-driven architectures require users to make requests and wait on the information they need. This often creates a tightly-coupled system wherein one application asks another for some needed information, and that application provides it. By contrast, in an event-driven architecture, applications and services publish events in real time as they become available to a streaming event messaging queue. Instead of playing request-and-wait, other applications services subscribe asynchronously to event types or topics of interest and consume them at their own speed. An application or service publishing events to the queue is termed a producer, and a subscribing application or service is termed a consumer. Although industry diagrams commonly portray them as such, there is not a fixed set of "producers" and another fixed set of "consumers;" one of the benefits of event-driven architecture is that it is a decoupled ecosystem that allows applications and services to act interchangeably to provide and consume data when and where needed in real time. Indeed, many applications and services may both produce and consume data.

Elements of Event-Driven Architecture

Event-driven architecture requires a great many complex data movements and interactions, but in its simplest form, it can be broken down as shown in Table 1:

Element	What it Does
Event Producers	Event producers generate events that place event messages into a message queue or stream. Events can capture a myriad of occurrences that are virtually limitless and are defined by the application: ● Purchases ● GPS positioning coordinates ● Periodic sensor readings such as current temperature ● A user sending a tweet with a specific hashtag ● User actions on a site, captured as a log entry for each click
Event Types and Topics	The system shepherds events into logical streams called event types or topics, and these are partitioned to support scale-out functionality.
Event Consumers	Event consumers listen for, and react to, events appended to these topics. A given application can and often does fulfill the role of both producer and consumer.
Decoupled architecture	A decoupled architecture is a framework for complex work that allows components to remain completely autonomous and unaware of each other. A decoupled system allows changes to be made to any one system without having an effect on any other system. Producers and consumers are decoupled from each other. This removes dependencies between components, which creates a number of advantages, including: ● Making the system elastic so it can grow and shrink in response to changing demand. ● Allowing each service to evolve independently, such as adding new functionality without impacting the rest of an application. That includes the ability to scale different microservices independently of each other. ● Making it easy and risk-free to add new producer and (especially) consumer apps. ● Enabling consuming apps to go offline – for example, overnight – then reconnect and replay what they have missed, with no additional coding or load on the producing apps.
Microservices	A microservices architecture breaks up applications into discrete services or functions, typically isolated within containers. Microservices communicate with each other through well defined, network-based APIs.
Event streaming platform[3]	Each microservice will typically read and write directly to its own database, but the event streaming platform is the glue that binds microservices together, providing the channels that allow events to pass between microservices and connecting microservices into complete

[3] Technologists and market observers use various terms to describe this entity, but what is important is the functionality it brings to the event-driven architecture equation

Element	What it Does
	applications. Applications can query streams of data in-flight, then persist both raw events and analytics into a database to make them available to other apps.
Database	In most microservices architectures, each microservice has its own database, but the centralized database is the foundation for event-driven architecture; this database: • Aggregates and persists events. • Enriches event streams with data from other sources, including historical data. • Provides a central repository for multiple event streams, enabling applications and users to benefit from the data across all microservices. • Serves as a data source for operational and analytical apps. • Provides a unified view of state across the business, without having to replay and join multiple independent streams.

Table 1: Elements of Event-Driven Architecture

Event-driven architectures deliver significant value to the business. There is tremendous pressure for applications to immediately react to changes as they occur. Capturing and acting on events in real time enables systems to react automatically and immediately to events. Competitive tracking gives a company the opportunity to rapidly position itself to outflank competitors. Detecting operational errors immediately lets the company minimize fallout and take quick corrective action. These benefits translate into not only operational excellence and cost savings but also enhanced customer experience as the company optimizes customer-facing processes. More broadly, an event-driven architecture helps the organization improve business agility.

Event-driven architecture is innovative, but its foundational principles are well established. Microservices are the realization of the vision set forth in 2005 in Service-Oriented Architecture (SOA), with its decoupled ecosystem of self-describing, self-registering services and self-subscribing consumers. Taken together, streaming and microservices are a pattern based on The Reactive Manifesto,[4] which defines a reactive system as one that is message-driven, remains responsive even as you elastically scale to meet varying workload demands, and is resilient in the face of failures.

Detailed examples follow in this paper, but a quick real-world example of event-driven architecture at work in retail would be an e-commerce site. As shoppers place items into their carts and check out, these events create a stream that invokes microservices managing inventory, promotions & coupons, credit check, fraud detection, shipping, post-purchase tracking and follow-up, invoicing, marketing list building, retargeting, and more.

Figure 1 shows event-driven architecture at work in an e-commerce application, where microservices are managing a range of functions such as those shown here.

[4] Available here

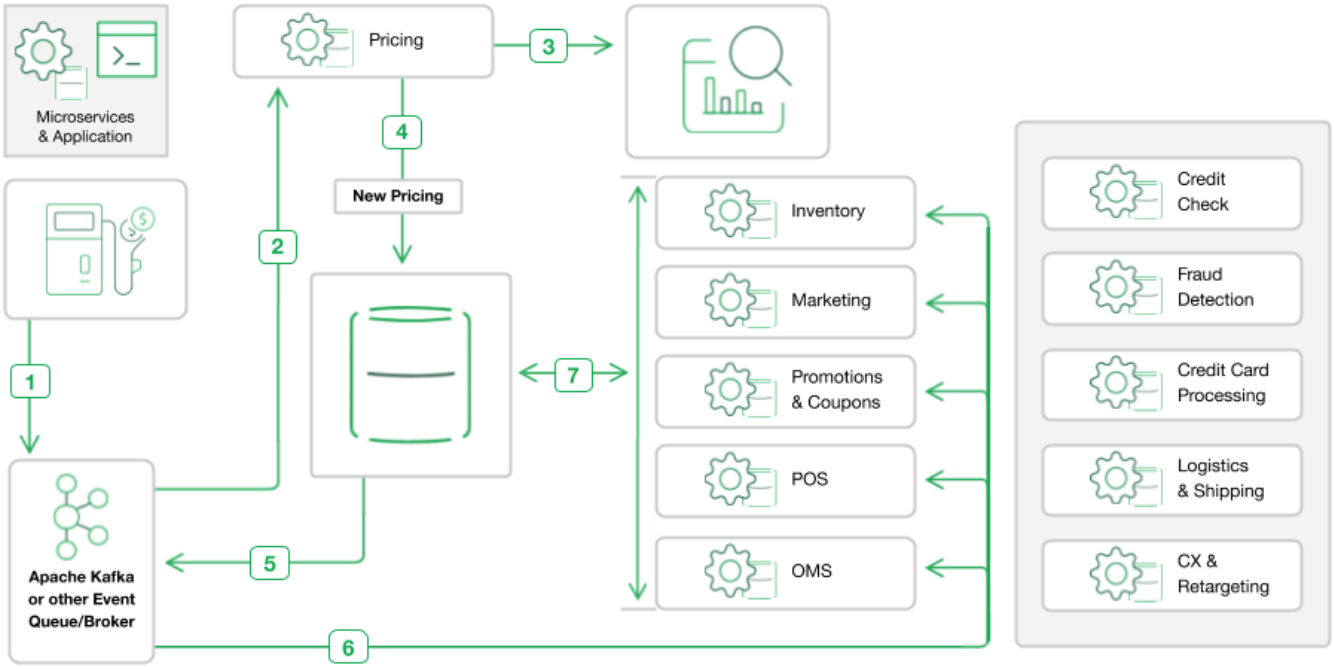

Figure 1: Event-Driven Architecture in e-commerce

Figure 1 illustrates a price change scenario. Fuel costs to ship some items have gone up, which could impact pricing:

[1] This produces events about the cost increase and places them into the message stream where the event queue makes them available. All microservices are listening for messages of interest.

[2 - 3 - 4] Pricing consumes the event, analyzes it against existing data, and produces events conveying the new pricing into the message stream.

[5 - 6] The database pushes those messages to the event queue, which then makes them available to all consumers who are listening for messages of interest. Microservices directly impacted by pricing changes, such as those that manage inventory, marketing, promotions & coupons, point of sale (POS), and the e-commerce provider's order management system (OMS), consume the price change events and update their individual databases accordingly.

[7] The centralized database aggregates and persists events, enriches event streams with data from other sources, including historical data, and provides a central repository for multiple event streams. This enables applications and users to benefit from all data across all microservices and provides a unified view of state across the e-commerce provider's enterprise.

Other examples of enterprise-driven architecture in real-world situations include trading applications that need to be updated in real time as stock, ETF, and options prices change; creating an IoT data pipeline that generates alarms whenever a connected vehicle moves outside of a geo-fenced area; or updating dashboards, analytics, and search engines as operational data changes. The possibilities are endless.

The Database is Core to Event-Driven Architecture

The database plays a central role in event-driven architecture. An event messaging queue streams data that is sequentially written as events into commit logs, which allows real-time data movement between your services.

While events flow through Kafka in an append-only stream, a complete event-driven architecture can aggregate, persist, and serve them to consuming apps and users in real time, enriching event streams with associated data sources and historical data. The database is best at this, materializing views of the data and application state to provide the lowest latency queries and deeper analysis to consuming applications, without having to replay and regenerate data from multiple streams. Although each microservice is provisioned with its own database, the core database provides a central repository for multiple independent event streams. Thus the platform serves as a data source driving operational and analytical apps, and perhaps even more importantly, provides a complete end-to-end view of the business.

Use Case Spotlight: Fraud Detection & Remediation

The value of an event-driven architecture shines through in many use cases across industry verticals, and a great example is a credit card fraud detection and customer care applications. With so many providers issuing credit cards and creating a range of financing options for their customers, this example can apply to many industries. With access to more accurate and up-to-date data, systems can respond to emerging conditions and anomalous events as they occur; users get what they want immediately; and businesses make better decisions, gain operational simplicity, and deliver better customer experiences.

In our spotlight scenario, an identity thief has stolen a legitimate customer's credit card information. The differences between how this plays out with legacy batch-loading data systems versus real-time treatment built on an event-driven architecture are striking and have real business implications.

THE OLD WAY: Batch Jobs Perform Periodic Data Analysis

In our scenario, an identity thief steals a legitimate user's credit card information and uses it, perhaps many times and likely up to the user's remaining credit limit. Based on ongoing usage patterns and location data, after running a nightly batch job the system flags fraudulent charges and locks the user's account. The next time the user tries to use the card, it is mysteriously declined. The user calls the bank, learns the bad news, gives the CSR account verification information, waits three days for a new card to arrive in the mail — and worries what may happen next. While worrying, the user may pull out another credit card they rarely use and wonder if that card would be less risky to use going forward.

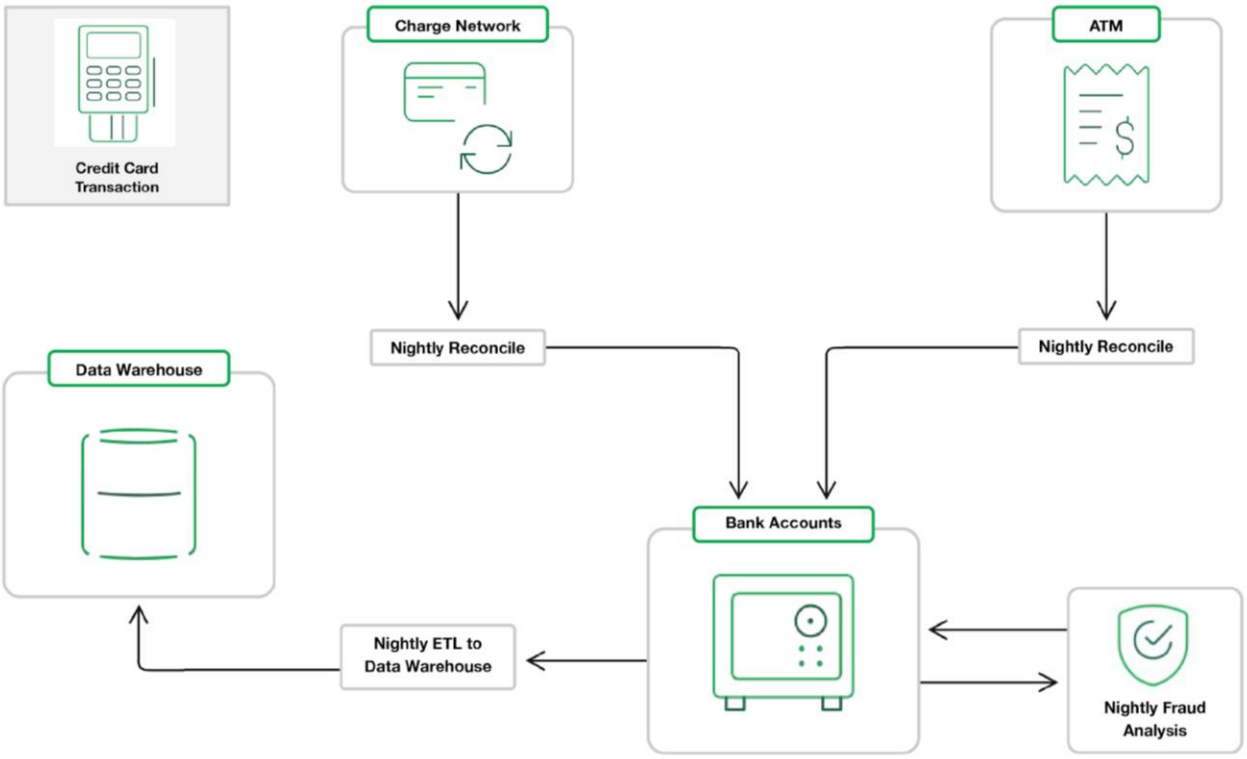

Figure 2: The Old Way

The fundamental technology gap in legacy systems that allows this to happen is illustrated in Figure 2: components are not sending events to one another in real time, but rather batching them up and processing them all at once. This means suspicious activity is not detected until a batch data job runs and performs its nightly reconciliation with the charge network. Maintaining transactional and historical data siloed in separate systems adds complexity and further slows the process.

The customer is surprised and worried over declined charges and wonders what is going on with the account. He or she also must go through extra effort, probably for some time into the future, reviewing statements and disputing charges, addressing angry merchants who may blame the user for the issue, and updating any accounts that were set up to autobill the old card to the new card number. The business bears the cost of fraudulent charges or raises an insurance claim to cover losses, and bears the added support burden of picking up the pieces with the customer.

THE NEW WAY: Real-Time Data Changes the Game

If an identity thief obtains the user's credit card information, a charge is recorded as an event in a transaction stream in real time and touches off a fraud detection algorithm. This enables a fraud management service based on an event-driven architecture to detect suspicious activity as it happens and apply various treatments, including declining the transaction and notifying the merchant, card issuer, and user. The issuer may notify authorities to initiate action against the thief while deciding whether to issue a new card and assuring the user the situation is under control.

Likewise, whenever the legitimate user goes to use the card, as shown in Figure 3, the system constantly checks for potential fraud. Depending

on the issuer, the user can receive an alert, confirm the legitimacy of the transaction, and use the card without incident.

Here, too, event-driven architecture delivers significant value to the business. As soon as a transaction event occurs, the fraud detection service calls on multiple data sources to verify the integrity of the transaction, avoiding losses to the credit card issuer.

In point of fact, not only is capturing and acting on events in real time a better way, it is the *only* way to effectively implement fraud detection and remediation. Detecting potentially anomalous events, and analyzing them against account-specific insights, in real time, significantly reduces the risk of fraud, and allows data scientists to quickly update their models as new threat patterns emerge. Immediate detection allows the company to be proactive and positive.

Finally, and from a business standpoint, perhaps most importantly: with the new, the customer experience is much better and far more likely to enhance customer satisfaction and retention.

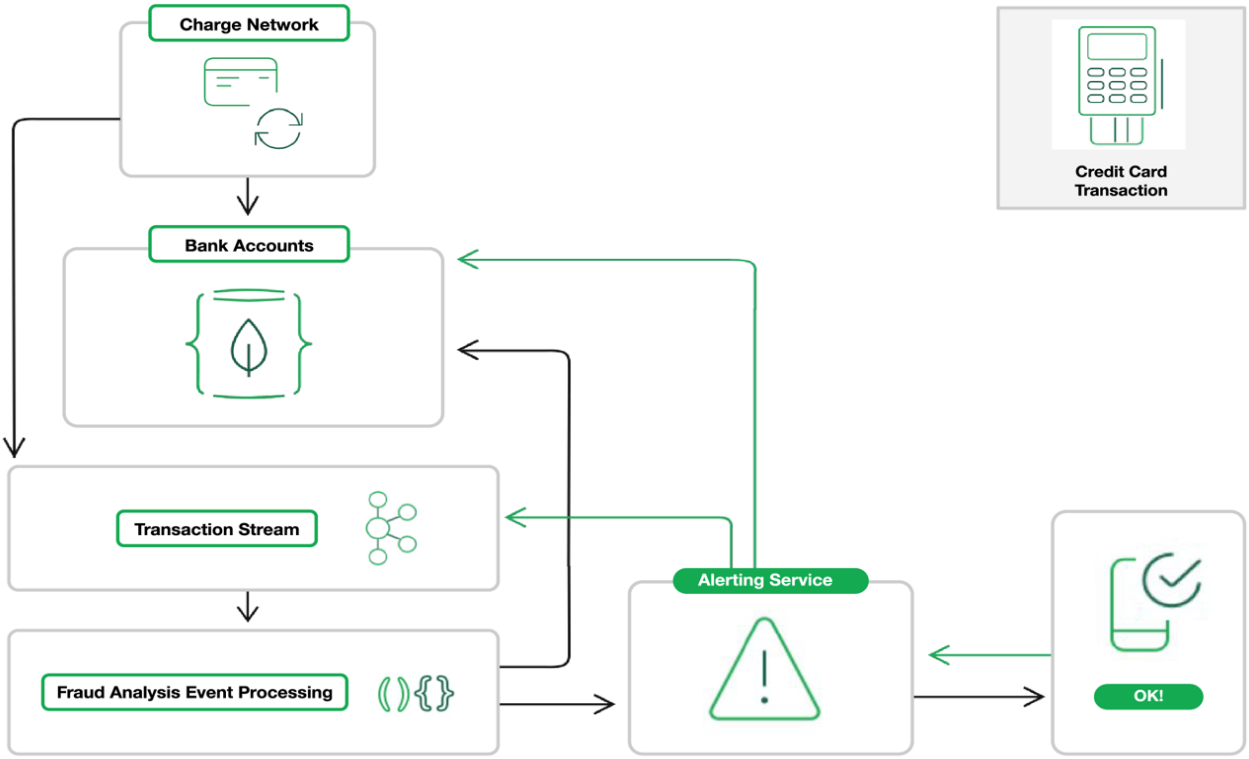

Figure 3: The New Way

While our examples above are illustrative, many companies beyond those who issue credit cards are in fact "financing" their users, which means fraud detection is important to a wide range of companies. For example, with cloud services, customers often consume a service and the provider bills them at the end of the month. A company can find itself chasing customers who commit intentional non-payment (INP) theft by using cloud services and disappearing, or simply refusing to pay. Event-driven architecture provides event streams that reveal user behavior patterns much faster, which enables prevention, detection, enforcement, and containment. By filtering on events such as usage above predefined charging and time thresholds, and choice of payment method, some companies providing cloud services have reduced or eliminated their INP-based losses by adopting an event-driven architecture.

Examples of other use cases illustrating the benefits of event-driven architecture follow further along in this paper.

Microservices and Event-Driven Architectures: Why MongoDB

Microservices are a common architectural pattern for modern app development, and many are built around event-driven architecture, which allows data to seamlessly move between the various microservices in an application. Microservices built with agile and DevOps methodologies, and deployed to elastically scalable cloud platforms, enable business agility.

Trying to implement microservices in a legacy relational database incurs the pain and friction of having to define a schema in the database and re-implement that same schema again to effect object-relational mapping (ORM) at the application layer. Then your development team has to repeat the process, first for each microservice and then every change to the data model as application functionality evolves.

With MongoDB, data modeling for microservices is easy. MongoDB helps you move at the speed your users demand. This gives you the power to launch new digital initiatives and bring modernized applications to market faster, running reliably and securely at scale, unlocking insights and intelligence ahead of your competitors. MongoDB is ideally suited to deploy your microservices architecture:

1. The document model is the most natural way to work with data in your database. It is simple for any developer to learn how to code against MongoDB, and as a result, industry surveys show it is wildly popular amongst developers.[5] This makes it much easier to hire and on-board new developers as you add more microservice teams. JSON documents are the language of APIs, which are how events commonly enter the system, and are also how event data is passed around between the microservices within the system. JSON documents are how you work with MongoDB, so there is no need to convert between models.
2. The database-per-microservice model adopted by most organizations means you may end up with scores of databases. With MongoDB that is no problem: MongoDB Atlas, the global cloud database for MongoDB, makes it easy to stand up fully managed and fully automated databases. You save a huge amount of ops effort and are not forcing every sprint team to include a DBA.
3. That said, however, not all of the data in a microservice's database is of interest solely

[5] Latest example at time of publication: Stackoverflow, *Developer Survey Results 2019,* available here and cited here: Mat Keep, MongoDB Blog, MongoDB: Most Wanted Database by Developers for the 4th Consecutive Year

to that microservice. MongoDB can serve as a centralized data source accessible to all microservices, so you can perform analytics across data from all of them and benefit from a unified view of your business.

4. In an event-driven microservices architecture, you have many instances, often running in a public cloud. This means it is a case of "when, not if" nodes will fail. MongoDB replica sets give you redundancy and self-healing recovery out of the box. In the cloud, each microservice, or each logical grouping of related microservices, is provisioned with its own MongoDB replica set, and deployed across multiple availability zones on any of the major cloud providers.
5. You can synchronize updates across serverless and microservices architectures by triggering an API call when a document is inserted or modified.
6. Different types of data models are sometimes appropriate for different microservices; these data models, may include documents, tabular, key-value, geospatial, text, and graph. With MongoDB you can work with all of these data models using a flexible document structure and rich query language to implement polyglot persistence without having to rely on a multitude of different database technologies.

How MongoDB Enables Event-Driven Architecture

Today MongoDB is at the core of many event-driven systems. MongoDB provides a database you can consume as a service or run yourself on-prem or in the cloud, with connectors and tools that make it incredibly easy to move data between systems, to analyze and visualize data, and to make data accessible to users wherever they are at any moment.

As shown in Figure 4, by starting with the core MongoDB data platform and binding in complementary technologies, MongoDB provides the data persistence heart of an event-driven architecture.

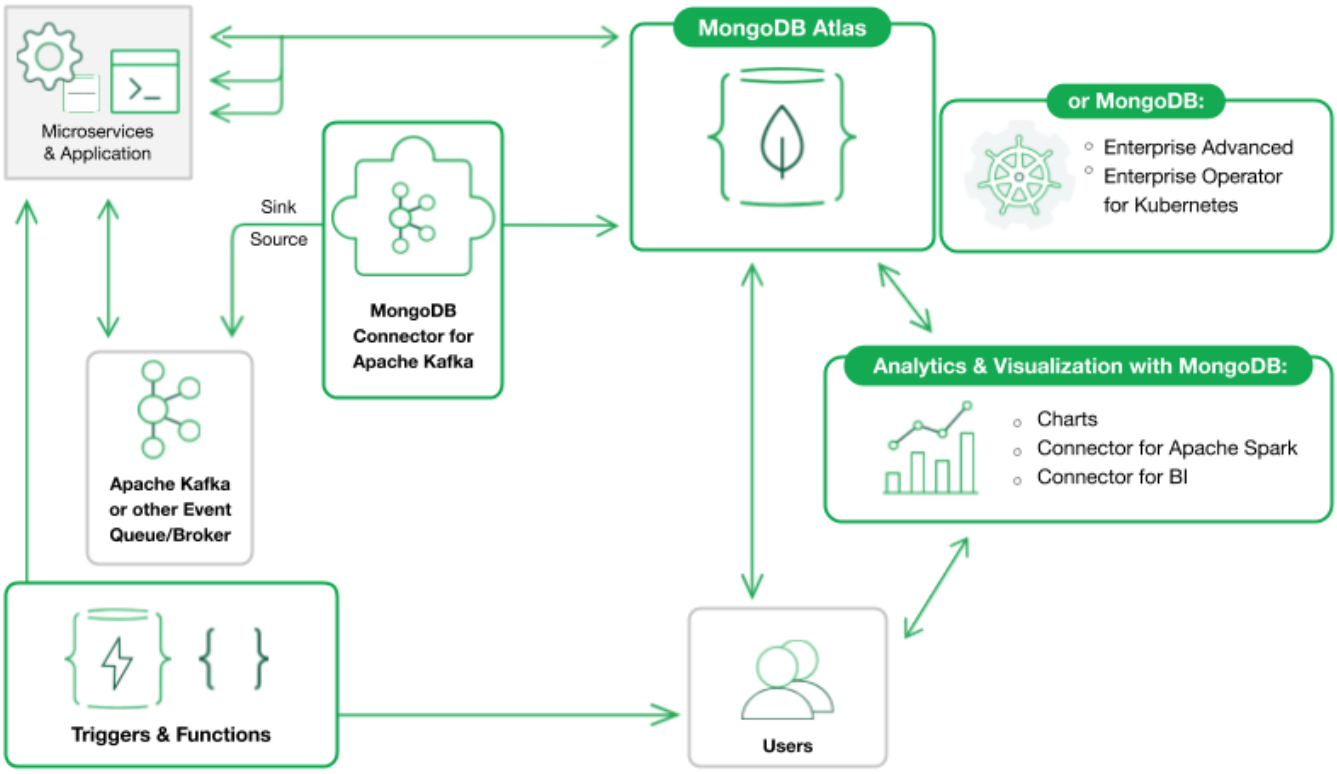

Figure 4: MongoDB At the Heart of Event-Driven Architecture

MongoDB Atlas

MongoDB Atlas is our on-demand, fully-managed and global database as a service (DBaaS) that runs on all leading cloud platforms to let you focus on apps instead of ops. A leader in the DBaaS space,[6] Atlas radically simplifies development and operations, provides a persistence layer for an event-driven architecture, and distributes data to consumers (systems and users) whenever and wherever needed. Atlas lets you scale at the click of a button, in response to an API call, or fully automatically as loads grow, to keep up with the demands of your microservices, with no downtime, full security, and complete data protection. Alternatively, you can deploy MongoDB Enterprise Advanced on your own self-managed on-premises or public cloud infrastructure.

MongoDB Change Streams: Addressing Change Data Capture

MongoDB Change Streams enables applications to stream real-time data changes in the database by leveraging MongoDB's underlying replication capabilities. Change Streams can notify your application of all writes to documents (including deletes) and provide access to all available information immediately as changes occur in the database. Downstream applications can register for notifications whenever a document or collection is modified, filtering on relevant events using aggregation pipeline stages $match, $project, and $redact. This enables them to act on new data in real time without constantly querying the entire collection to identify changes. It also avoids delays

[6] Latest example at time publication: The Forrester Wave™: *Database-As-A-Service,* available here and cited here: Seong Park, MongoDB Blog, *MongoDB Named A Leader in The Forrester Wave™: Database-As-A-Service Q2 2019*

and higher system overhead that constant request/response polling can introduce due to the database being regularly checked even if nothing has changed.

Atlas Triggers and Functions: The Benefits of Change Streams, Consumed Serverlessly

Business applications can consume change streams directly via a message queue, but MongoDB's Atlas Triggers are a feature that provides a serverless way of consuming change stream events. With Triggers, you don't have to stand up your own application server to run your change data capture process. Change streams flow change data to Atlas Triggers to create responsive, event-driven pipelines.

MongoDB Functions enable developers to run simple JavaScript code in a serverless platform, making it easy to implement application logic, securely integrate with cloud services and microservices, and build APIs. Atlas Triggers can execute Functions according to predefined schedules or in real time in response to changes in the database or user authentication events. Examples include:

- Sending a text message or email to notify a customer that his or her balance has fallen below a predefined threshold.
- Reacting to a customer order event by updating inventory levels, sending a delivery request to your warehouse system, and recording the order in the customer's profile.

MongoDB Enterprise Operator for Kubernetes

MongoDB gives you the freedom to run anywhere. Wherever you want to run your event-driven architecture, we can be your database of choice. Atlas is the preferred route for many, but we also give you the flexibility to run MongoDB yourself, controlling it right alongside your microservices orchestrated with Kubernetes.

MongoDB's Enterprise Operator for Kubernetes lets you deploy and manage the full lifecycle of your containerized application and MongoDB clusters all from within your Kubernetes environment and get a consistent experience wherever you run them: on-premises, hybrid, or public cloud. Kubernetes Operators are application-specific controllers that extend the Kubernetes API to create, configure, and manage instances of stateful applications such as databases. On self-managed infrastructure, whether on-premises or in the cloud, you can use our Kubernetes Operator and the MongoDB Ops Manager to automate and manage MongoDB clusters. You have full control over your MongoDB deployment from a single Kubernetes control plane. You can use with the Operator with upstream Kubernetes or with any popular Kubernetes distribution such as Red Hat OpenShift and Pivotal Container Service (PKS).

MongoDB Connector for Apache Kafka

Confluent Platform, including Apache Kafka and Kafka Connect, is designed as an event messaging queue for massive streams of data that sequentially writes events into commit logs, allowing real-time data movement between your services and data sources. The MongoDB Connector for Apache® Kafka® simplifies building robust, reactive pipelines to move events between systems. As illustrated in Figure 5, you can use MongoDB as:

- A **sink (consumer)** to ingest events from Apache Kafka topics directly into MongoDB collections, exposing the data to your services for efficient querying, enrichment, and analytics, as well as for long-term storage.
- A **source (producer)** for Kafka topics; in this mode, data is captured via Change Streams within the MongoDB cluster and published straight into Kafka topics.

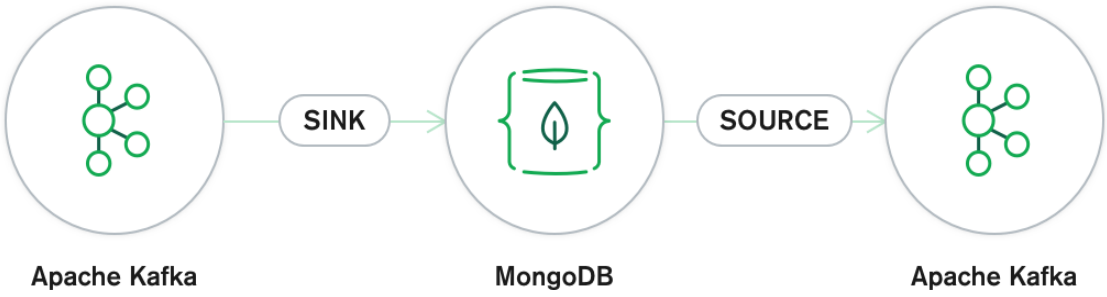

Figure 5: MongoDB Can Serve as Both a Sink and a Source for Apache Kafka

These capabilities enable consuming apps to react to data changes in real time using a fully event-driven programming style. MongoDB and Apache Kafka work together to enable you to build microservices and event-driven architectures with ease. Our Kafka Connector is developed and supported by MongoDB engineers and verified by Confluent as a first-class component of Confluent Platform.

Analytics and Visualization with MongoDB

Key features that enable your teams to obtain analytic insights in MongoDB include:

MongoDB Charts

MongoDB Charts is the fastest and easiest way to visualize event data directly from MongoDB in order to act instantly and decisively based on the real-time insights your event-driven architecture is delivering. Leveraging the analytical functionality of MongoDB's aggregation framework, MongoDB Charts lets you connect to any MongoDB instance as a data source, create charts and graphs, build dashboards, and share them with other users for collaboration. MongoDB's materialized views enable you to maintain rolling snapshots of data, e.g., sales data, that can be visualized in live dashboards in Charts to monitor sales performance. Charts natively supports the richness of the document model, including nested and hierarchical data, making data manipulation user-friendly by presenting operations that make sense for each data type. Charts lets you build standalone dashboards or embed them straight into your web apps.

Charts supports workload isolation, working against secondary replicas or dedicated analytics nodes in MongoDB Atlas clusters to avoid contending for resources with operational apps. You either create standalone dashboards or embed charts directly into your Web apps, building richer user experiences while reducing development time and effort. All MongoDB Charts are live updated and deeply customizable, making them the perfect fit for your modern, styled web applications and sites.

MongoDB Charts also helps you make sense of geospatial data, such as latitude and longitude coordinates, or polygons on the earth's surface. Geolocation capabilities are already a part of the MongoDB platform, and Charts extends and enhances them by letting you perform geolocation analysis without having to write any code.

MongoDB Connector for Apache Spark

Apache® Spark® performs analytics using a cluster of nodes and is used for batch workloads that previously were run with Apache® Hadoop®. Because of Spark's ability to store the data in memory rather than on disk, users get much faster execution. Spark can also perform streaming analytics, reusing the same application analytics code used in Spark batch jobs. The MongoDB Connector for Apache Spark lets you access Spark libraries and enables streaming, machine learning,

and real-time analytics to help you query streaming data in real time, and persist results to MongoDB. You can also use the Spark connector to extract data from MongoDB, and blend it with streaming data being processed by Spark.

MongoDB Connector for BI

If you're sending data to third-party analysis tools via ETL, you're operating with stale data and negating the real-time advantages that are the main reason to adopt event-driven architecture in the first place. The MongoDB Connector for BI creates a direct connection from MongoDB to third-party SQL-based analysis tools to ensure that your teams are always making the best decisions based on the freshest data, even if they are visualizing the data in third-party reporting tools.

MongoDB Professional Services: Accelerating Event-Driven Architecture

MongoDB's Professional Services team helps you combine people, processes, and technology to make the move to event-driven architecture and run your business in real time. MongoDB Professional Services works with you to build an in-house Innovation Accelerator to achieve faster time-to-market across multiple business applications and use cases. Establishing event-driven architecture requires streamlining your development processes with agile ways of working such as DevOps and microservices patterns. Professional Services delivers Innovation Solution Kits that provide design patterns for common use cases, with reference architectures and technical standards to help you make the move to microservices and establish your event-driven architecture.

Other Use Cases

Additional use cases offer examples of how MongoDB enables event-driven architecture to deliver technological advantages and business value. This section outlines just a few.

Time Series and IoT

Time series applications capture and measure state changes over time and are often used to forecast future changes and establish operational system thresholds. Rather than appending only the most recent data and storing solely the latest application state, time series applications store all changes of state (e.g., `[x=150, x++, x+=12, x-=2]`) in the original order so they can be replayed to re-create the final state and be analyzed to measure the changes that have occurred. IoT applications are the most widely deployed examples of time series applications. They must cope with massive numbers of events being generated by a multitude of devices. Existing data systems are challenged simply to keep up, much less to capture events in sequence and make sense of not just the IoT-specific data but what it means when analyzed along with all other relevant data across the enterprise.

Event-driven architecture creates a data pipeline where these producers – the devices, sensors, and gateways in your IoT networks – generate streams of events, and event consumers listen and react to those events. MongoDB supports event-driven architecture that can accommodate the complex and quickly changing time series data generated by fleets of sensors and connected devices. MongoDB and Kafka working together are well suited to this design approach because MongoDB can store arbitrarily long sequences of events, while Kafka can quickly and efficiently provide them in the correct sequence to a consuming application via MongoDB's Kafka Connector. Kafka can ingest data very quickly, acting as a buffer during spikes in sensor data ingress, such that business apps can process events at their own pace rather than having to discard readings during busy times.

MongoDB persists and aggregates events, serving them to consuming apps in real time, using Atlas Triggers to flag data of interest. It can also enrich event streams with associated data sources and historical data. This means that instead of having to replay and regenerate data from multiple streams,

your existing apps can access and act on enriched event streams in real time. When you add new apps, you can so without changing anything else in your architecture, and MongoDB can replay all events so the new app can learn from historical events as well as new ones. Two prominent IoT-focused use cases illustrating the value of event-driven architecture are predictive maintenance and telematics.

Predictive Maintenance

All machines eventually break down. When machines are at the heart of manufacturing and other industrial processes, those breakdowns can wreak havoc on production, schedules, resources, delivery commitments, customer relationships, and more. Simply waiting until machines break down and only then attempting repair or replacement is unacceptable, but predicting when it will happen, with the specific knowledge to take evasive action, is incredibly difficult. The complexity of the systems, equipment, and parts involved in these processes can overload legacy data architectures, rendering proactive approaches impossible.

Event-driven architecture can be deployed as the foundation for predictive maintenance applications. Kafka streams time series event data from assembly lines and parts inventories into MongoDB to drive machine learning models that detect degrading conditions in equipment and systems that predict problems. The company can either repair equipment or swap it out for replacement parts before it shuts down assembly lines. Atlas Triggers can alert engineers to potential equipment failures, and analysis of both real-time and historical trending data can help the company adopt optimal equipment operating modes and manufacturing parameters.

Telematics

The average vehicle now has more than 30,000 parts, and every one of them can impact operation, safety, and performance. Parts and systems are now monitored and managed by more than 100 sensors in the average vehicle, and that number is expected to double in the next two to three years. Diagnostics from a vehicle's sensors must be received and processed back at base, in the backend, and distributed in real time to microservices managing everything from vehicle systems to dealer and service records and more. And since connected cars are quickly gaining the smartphone-like ability to make payments, other microservices come into play to manage processes including credit check and the transaction itself. Legacy data architectures and their batch processing operations are no match for the amount of data being generated by all of these vehicles.

Event-driven architecture supports telematics applications. Kafka streams event data from all of those sensors into MongoDB to drive machine learning models monitoring performance of parts and equipment; real-time road, traffic, and weather conditions; location and movement of the vehicle, and more. This helps to ensure driver safety, optimize automotive performance, and keep maintenance schedules on track.

MongoDB’s IoT Reference Architecture provides a deeper exploration of MongoDB at the core of event-driven architecture. Learn more →

E-commerce

Engaging buyers and keeping them engaged throughout their buying journeys is critical. MongoDB processes events in the customer journey so the system can respond instantly with personalized offers. The system’s responses are based on real-time inventory, available offers, customer behaviors and purchasing patterns, seasons, time of day, weather, and other factors. Accessing event data in real time enables the e-commerce site to make offers with high takeup potential and update inventory the moment it changes to avoid disappointing shoppers with stockouts on items, for example, that the system may have just shown as available.

When a shopper presses the 'Buy Now' button, the system must communicate with various microservices managing inventory, credit check, ordering, shipping, and more to complete the transaction and ensure a good customer experience. MongoDB provides the means for microservices to pass messages to each other, with multiple microservices publishing and subscribing to the same topics so each microservice has the data it needs. The database persists the sequences of messages and application state at any moment in time. This enables a wide range of actions, e.g., notifying an Atlas Trigger when an order has been completed or dispatched, and the Atlas Trigger calling a notification service to send an email, text, or social media message to a shopper.

Figure 6 is the corollary to Figure 1, this time showing where MongoDB products and features fit into event-driven architecture for e-commerce.

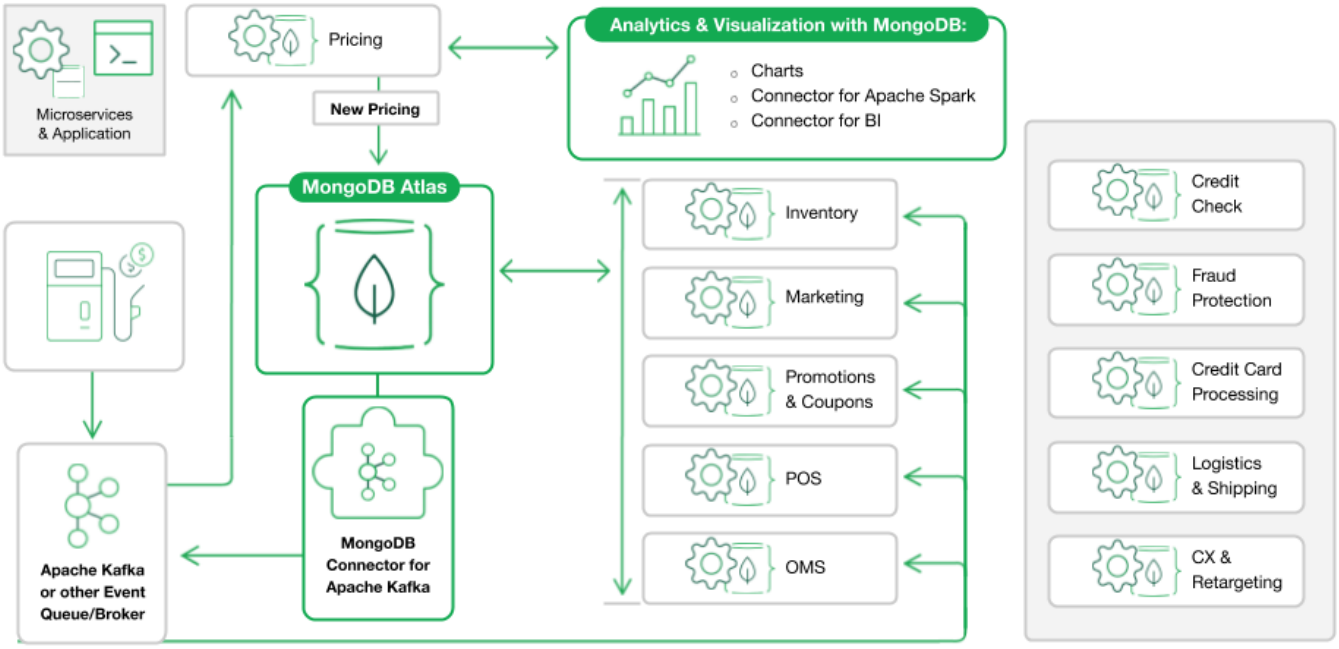

Figure 6: MongoDB in e-commerce Event-Driven Architecture

The MongoDB Connector for Apache Kafka first serves as a Kafka sink: the events about the increased fuel costs become part of a Kafka topic inserted into a MongoDB Collection. From there the microservice that manages the pricing leverages MongoDB's advanced analytical querying capabilities such as the MongoDB Aggregation Framework to calculate new pricing, and this new pricing data can be further analyzed and visualized using MongoDB Charts, the MongoDB Connector for Apache Spark, and the MongoDB Connector for BI. MongoDB Change Streams flows change data to Atlas Triggers and Functions to push the change data into a Kafka topic using the source capability of the Kafka Connector. This enables other microservices that depend on the pricing data to consume events about the pricing changes, using a fully reactive, event-driven programming pattern.

Operational Data Layer

MongoDB and Kafka can work together in an event-driven architecture to create an Operational Data Layer (ODL). An ODL is an architectural pattern that centrally integrates and organizes all siloed enterprise data and makes it accessible to users or apps that need to access the data. An ODL can combine data from multiple legacy sources into a single repository where new applications can access all data from all sources to develop single views of customers, products, financial assets, or other relevant entities. Unifying all data into one repository helps the organization make smarter decisions and take action faster.

An enterprise data warehouse (EDW) or Hadoop platform can also unify data – for daily or weekly batch analysis. Both leave an organization operating with stale data. By contrast, an ODL provides real-time data to support operational processes and user queries. Similar to the way microservices can operate independently of each other, an ODL can expose existing data to new applications while mitigating the impact to legacy systems. Kafka captures any changes in source systems and streams them into MongoDB to keep the ODL fresh, and conversely, if users write to the ODL, Change Streams and Kafka can push that back in real time to update the source systems. MongoDB's white paper, Implementing an Operational Data Layer, discusses the topic in depth. Learn more →

Artificial Intelligence

Training AI models is time- and resource-intensive, and at a time when concerns about climate change and sustainable practices are at all-time highs, some assert that simply training a single AI model creates what to many would be an unthinkable carbon footprint. Slashing the amount of time it takes to train AI models, then, may be the single most important step toward making AI a cost-effective (and palatable) proposition, and event-driven architecture is poised to do exactly that.

By enabling you to analyze data in real time, event-driven architecture can accelerate by orders of magnitude your assessments of how AI models are performing, so you can access additional datasets quickly and get models field-ready faster. You can access training and model data programmatically via MongoDB's R and Python drivers, or via our Spark Connector, so you are always updating your models with the freshest data. Machine learning algorithms can be computationally expensive and may not be able to keep up with incoming events at the busiest times. MongoDB and Kafka work together to act as a buffer when a learning algorithm is unable to keep up with spikes in event volumes.

Web Activity Tracking and Log Aggregation

Website activity tracking was the original use case for Kafka when it was developed at LinkedIn. In event-driven architecture, site activity such as pages visited, engagement duration, ads displayed, and many other metrics are captured in Kafka topics (one topic per data type). These topics can be consumed by applications such as monitoring, real-time analysis, or archival for offline analysis. Insights from the data are persisted in MongoDB where they can be analyzed alongside data from all other sources.

Log aggregation traditionally involved collecting physical log files from multiple servers and storing them in a central location (e.g., HDFS), with the data analyzed by periodic batch jobs. Modern event-driven architectures use MongoDB and Kafka to combine real-time log feeds from multiple sources so the data can be continuously monitored and analyzed. Events can be logged and their insights and consequences can be immediately understood and acted upon.

MongoDB Event-Driven Architecture Customer Snapshots

Following are examples of companies working with MongoDB to run their businesses in real time through event-driven architectures.

Ticketek

Ticketek, Australia's largest sports and live entertainment ticketing company, chose MongoDB over a range of databases including Oracle, Microsoft SQL Server, MySQL, and several NoSQL providers to enhance the customer experience for millions of patrons. MongoDB Atlas serves as the data layer for the event-driven architecture that powers Ticketek's multi-channel ticket sales and distribution network. Ticketek's strategy is to banish batch processing from the industry and create an advanced data ecosystem through open integration and real-time data streaming with clients and partners. Every ticket sale passes through an e-commerce platform backed by fully managed MongoDB Atlas databases. The platform feeds a network of real-time dashboards that provide insights into ticket sales performance and trends. Learn more →

EG

EG, the UK's leading commercial property data service, made MongoDB the center of its event-driven architecture. EG runs its business on a microservices-inspired architecture it terms "pods." Each pod maintains a local instance of the data it needs, pulling from a centralized system of record that stores all property details, transactions, locations, market data, availability, and more. All data product pods and the system of record run on MongoDB. With its legacy relational databases, EG could deploy only one application update per month, but with MongoDB, it is deploying new business functionality 50-60 times per month. Learn more →

ao.com

ao.com, a leading online electrical retailer, uses event-driven architecture built on MongoDB. AO has many sources of customer information spread across many different departments, each using its own technology. The company recognized the need to bring this data together into an Operational Data Layer to provide a 360-degree single customer view. Now apps in its source system update customer data, which is written as events to Kafka and pushed to MongoDB Atlas to create one source of truth for all customer data. MongoDB then serves as a source for apps including customer service, fraud detection, and GDPR compliance. With MongoDB, ao.com can

scale up and out on demand, with no application downtime; it can perform complex queries in milliseconds improving customer experience and accelerating fraud detection from hours to seconds. With MongoDB, AO can react in real time to anything that happens in its production environment. Learn more →

Man AHL

AHL, a subsidiary of The Man Group, which is one of the world's largest hedge fund investment firms, used MongoDB to create a single platform for all of its financial data. The system receives data for up to 250 million ticks per second from multiple financial sources and writes it to Kafka. Kafka consolidates and buffers events before they are stored in MongoDB, where the data can be analyzed by quantitative researchers.
Learn more →

comparethemarket.com

comparethemarket.com, a leading price comparison provider, uses MongoDB as the default operational database across its microservices architecture. While each microservice uses its own MongoDB database, the company needs to maintain synchronization between services, so key data changes are written to a Kafka topic. Those events are written to MongoDB to enable real-time personalization and optimize the customer experience. Learn more →

Conclusion

The world runs in real time, and businesses must match that pace by becoming agile organizations. Their ability to do so rests on empowering their development teams to rapidly build new business functionality through agile ways of working, such as DevOps and microservices patterns. Microservices are now the major driver in companies making the move to real time, and event-driven architecture enables them to get there. By starting with the MongoDB platform and binding in complementary technologies, companies can avail themselves of all of the advantages of an event-driven architecture, helping them align teams effectively, innovate rapidly, and meet the challenges of ultra-competitive global markets.

We Can Help

We are the company that builds and runs MongoDB. More than 14,200 organizations rely on our commercial products. We offer software and services to make your life easier:

MongoDB Atlas is the database as a service for MongoDB, available on AWS, Azure, and GCP. It lets you focus on apps instead of ops. With MongoDB Atlas, you only pay for what you use with a convenient hourly billing model. Atlas auto-scales in response to application demand with no downtime, offering full security, resilience, and high performance.

MongoDB Enterprise Advanced is the best way to run MongoDB on your own infrastructure. It's a finely-tuned package of advanced software, support, certifications, and other services designed for the way you do business.

MongoDB Atlas Data Lake allows you to quickly and easily query data in any format on Amazon S3 using the MongoDB Query Language and tools. You don't have to move data anywhere, you can work with complex data immediately in its native form, and with its fully-managed, serverless architecture, you control costs and remove the operational burden.

MongoDB Charts is the best way to create visualizations of MongoDB data anywhere. Build visualizations quickly and easily to analyze complex, nested data. Embed individual charts into any web application or assemble them into live dashboards for sharing.

MongoDB Realm is a serverless application platform that provides an elastically scalable infrastructure for handling requests and coordinating cloud service and database interactions. Realm takes care of the infrastructure, allowing you to focus on the high-value parts of your applications.

MongoDB Cloud Manager is a cloud-based tool that helps you manage MongoDB on your own infrastructure. With automated provisioning, fine-grained monitoring, and continuous backups, you get a full management suite that reduces operational overhead, while maintaining full control over your databases.

MongoDB Consulting packages get you to production faster, help you tune performance in production, help you scale, and free you up to focus on your next release.

MongoDB Training helps you become a MongoDB expert, from design to operating mission-critical systems at scale. Whether you're a developer, DBA, or architect, we can make you better at MongoDB.

Resources

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

Case Studies

Presentations

White Papers

Free Online Training

Webinars and Events

Documentation

mongoDB

US +1 866.237.8815 • Rest of World +1 650.440.4474 • info@mongodb.com