MongoDB®

Transforming Customer Experience With MongoDB Atlas Search

October 2021

Introduction

The search bar is truly magical. Whether we are shopping for groceries, buying a new home, browsing the web to find answers to our burning questions, servicing our customers, looking for our next job, or seeking suggestions for our next vacation, the search bar helps us navigate and discover exactly what we are looking for – all just a simple, natural language query away.

Conditioned by years of internet search engines, our users expect the applications they rely on at home and at work to provide search functionality. But building full-text search into applications is hard. Developers have had to turn to one of two approaches, both of which come with major downsides and trade-offs:

1. Use the search features built into your database to directly query stored data. However, these features are limited, failing to provide the rich search functionality users have come to expect.
2. Bolt on a specialized search engine alongside your database, synchronizing data between the two. Now users get the rich search experience they expect but the application stack has gotten much more complex and unwieldy. All of this translates to reduced developer velocity, compromised customer experience, and escalating costs.

MongoDB Atlas Search gives you a much better way. It combines the power of Apache Lucene – the same technology underpinning the world's most popular search engines – with the developer productivity, scale, and resilience of the MongoDB Atlas database.

A couple of API calls or clicks in the Atlas UI and you instantly expose your data to sophisticated, relevance-based search experiences that boost engagement and improve customer satisfaction. Your data is immediately more discoverable, usable, and valuable. And it's all fully managed for you in the cloud, removing operational burden. Customers have reported **30% to 50% improvements** in time to market for new application functionality by adopting Atlas Search.

In this white paper, we dig deeper into the challenges of implementing search today and how that's transformed with Atlas Search. We discuss the ideal use cases for Atlas Search, along with those requirements where you may be better served considering alternative approaches. And we wrap up with how you can get started with Atlas Search.

The Trouble With Search

Although search is essential for every modern application, building it isn't easy. Application owners need to consider how both the speed of search and the relevance of search results will evolve over time. The more demanding each becomes, the more sophisticated their search needs will be.

To address search requirements, developers typically either try to contort their database to handle search queries, or they turn to specialized search engines, bolting them on to their application's database. Before choosing which approach to take, it is important to recognize that databases and search engines are fundamentally different technologies, designed to do different things.

Database Design Goals

Databases are very powerful when users know upfront *exactly what they want to query* – for example, returning the account balance for a specific customer or booking a specific hotel in a city. Databases provide indexes to make these queries fast, but you need to know your users' query patterns in advance so that the right indexes can be defined on the underlying data.

Beyond retrieving data, databases must also optimize for the demands of transactional and analytical "systems of record" applications, prioritizing data correctness and integrity, concurrency, resilience, and scalability.

Search Engine Design Goals

Search engines are very powerful when users want to retrieve information with natural language search terms and are *open to suggestions* in the results that are returned to them. In essence, a search engine has to infer *intent* from a user's query, providing users with the ability to explore related information.

For example, a search engine will improve discoverability to users researching city vacations in different destinations that meet their desired criteria, such as cities that are situated on the coast and less than three hours' flight time away. Alternatively, the user might want to find articles with information related to a specific topic or get recommendations for a movie to stream.

For these more open-ended questions, specialized inverted indexes that contain the position of each word in a document are required. Each piece of content is extracted, analyzed, and converted to a set of terms that are then scored and indexed. The search engine follows the same process when parsing a query to match the search term to the most relevant top-k documents and then returns them as a set of suggestions to the user.

A search engine should also enable the application owner to customize how data is indexed, correct user typos, surface related information, and tune result sets to provide the most relevant, highly scored results first.

Option 1: Using Database Search

Because the database stores the application data that needs to be searched, using its built-in query and indexing features would seem to be a simple solution. There is no need to replicate data out of the database into a separate system which then has to be independently maintained. Also, developers can continue to work with a single query language and driver they are already familiar with.

Many databases, including MongoDB Atlas, offer built-in index and query features for text search or more general regular expressions (regex operators), which can be useful for simple search needs (e.g., where there is a limited set of matching documents or infrequent exact matches).

However, as search requirements evolve, developers may quickly encounter feature limitations and compromised application performance that impacts user experience.

Limited Functionality, High Complexity

Databases lack many of the features that are expected for sophisticated search experiences. Features that are either absent or limited in functionality can include searching across different data types, relevance tuning, fast faceted navigation and counts, fuzzy search, autocomplete, highlighting, and broad international language support. The importance of each of these features is discussed in more detail in the "Key Atlas Search features" section of this white paper.

It is also not uncommon to find that queries needing to filter many database records are composed of long chains of subqueries. These are hard to write, test, debug, and maintain, adding further friction to development.

Performance Overhead

As users' search queries become broader and more sophisticated, developers need to create more indexes against the data. Failing to do so means queries have to scan every record in the database to find the required matches, crippling performance.

Indexes don't come for free, however. As data is written to the database, the index has to be maintained alongside the base data. Multiple indexes cause more write amplification and consume additional memory, CPU, and I/O, all of which impose a performance overhead on the database for regular operations as well as for search queries.

Some databases do offer specialized inverted indexes that can reduce the number of secondary indexes that need to be created and maintained. However, these also impose performance penalties to the database, especially when indexed data is being frequently updated. This is because the index has to be rebuilt to merge new data, making it inaccessible to queries until the process – which itself is CPU and I/O intensive – is complete.

Beyond the performance overhead of index maintenance, it is important for application owners to consider the additional workload the database now has to support. In addition to handling

the core data persistence and processing demands of the application, the database also has to support search operations. To avoid resource contention between these two workloads, the database needs to be carefully sized and closely monitored and scaled, driving up operational overhead and cost.

Option 2: Bolting On a Specialized Search Engine

If the database's internal search features are not adequate to satisfy the desired user experience, then another option is to bolt on a dedicated search engine alongside the database.

This will provide the more sophisticated search features demanded by customers, but it will impose additional constraints on developers and ops teams while driving up data duplication and technology sprawl (see Figure 1).

Impacts on Developer Productivity

It is critical in today's digital economy for developers to build and evolve applications at speed. Introducing a search engine alongside the database means developers now have two separate systems they need to work with, which slows them down.

With this approach, developers have to learn how to work with two entirely different query languages to access the database and the search engine. This increases their learning curve and means frequent context switching when building application functionality, both of which impact their productivity while complicating testing and ongoing maintenance.

Because this approach requires two different APIs/drivers, application dependencies become much more complex, reducing the pace and frequency of releasing applications to production.

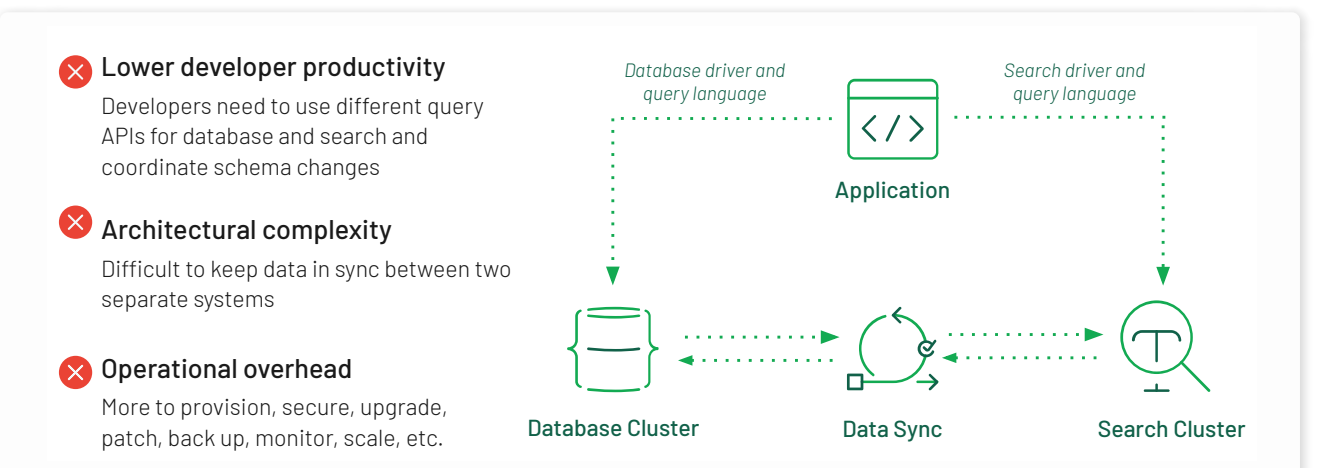

Figure 1: Database cluster, data sync, and search cluster create complexity and architectural sprawl, inhibiting developer productivity and increasing operational load.

DevOps Burden

Doubling up with a database and search engine also adds time, cost, and complexity to operations and site reliability engineering (SRE) teams.

Now they have an additional system in their technology stack that needs constant care and feeding: It has to be provisioned, secured, monitored, scaled, patched, and backed up with its own tooling and APIs. It also means working across multiple vendors, making issue resolution more complex. Every new project means another dataset living in its own silo, adding to data sprawl and governance overhead.

Synchronization Overhead

To surface relevant and up-to-date search results, the database and search engine need to be kept synchronized, duplicating data between systems.

This means engineering teams need to create a synchronization mechanism that replicates data from the database to the search engine. Typically they will create a data pipeline with custom filtering and transformation logic built on top of messaging systems such as Apache Kafka, or using packaged connectors from specialized providers. Whether building or buying, the process takes time and adds ongoing costs. The synchronization mechanism also has to be deployed onto its own nodes, creating additional hardware sprawl.

Once the synchronization mechanism has been deployed, it needs to be monitored and managed, adding more engineering overhead.

It is important that replication to the search engine keeps pace with database writes so that search results do not excessively lag the database and break application SLAs. Monitoring the replication process is necessary to identify and remediate synchronization issues. This becomes especially complex if the search index falls so far behind the database that it has to be resynced from scratch, causing potential application downtime.

New application features that necessitate changes to the database's schema often need both the synchronization logic and the search engine schema to also be updated at the same time. This creates more dependencies that slow down the pace of rolling new features to production.

Performance Overhead and Impact to User Experience

Beyond the overhead of creating and managing the synchronization mechanism, using two systems can impact application performance. Complex queries may need to return data from both the search engine and the database, requiring coordinated query routing and additional network hops between the two systems. Network round-trips add latency that impacts user experience.

Why Not Just Use a Search Engine as a Database?

With search engines storing and querying data, some engineering teams may consider eliminating the database altogether and just using the search engine for data persistence. At first glance, this would address many of the constraints discussed above, presenting a single system to develop against and to operationalize, while eliminating the overhead of data synchronization.

But as noted earlier, databases and search engines are different technologies designed to do different things.

Beyond serving application queries, databases are designed around a core set of data persistence and processing capabilities. These demand data integrity, consistency, and durability; balanced performance across reads and writes; concurrency; availability; security; disaster recovery; and more.

With a specialized architecture and indexing focused on fast, relevance-based information retrieval, dedicated search engines have a different set of design goals that compromise many of the capabilities that make databases so essential.

One of the industry's largest search engine vendors cautions against using a search engine as a database both in press articles and in core product documentation.

Customers have reported improved development velocity of 30% to 50% after adopting Atlas Search.

MongoDB Atlas Search: A Better Approach to Building Search

MongoDB Atlas Search makes it easy to build fast, relevant, full-text search on top of your data in the cloud. A couple of API calls or clicks in the Atlas UI and you instantly expose your data to sophisticated search experiences that boost engagement and improve satisfaction with your applications. Your data is immediately more discoverable, usable, and valuable.

By embedding an Apache Lucene search index directly alongside the database, data is automatically synchronized between the two, developers work with a single API, there is no separate system to run and pay for, and everything is fully managed for you, relieving operational burden. The MongoDB application data platform radically simplifies your data architecture, enabling you to gain a competitive advantage by innovating faster while reducing cost, risk, and complexity.

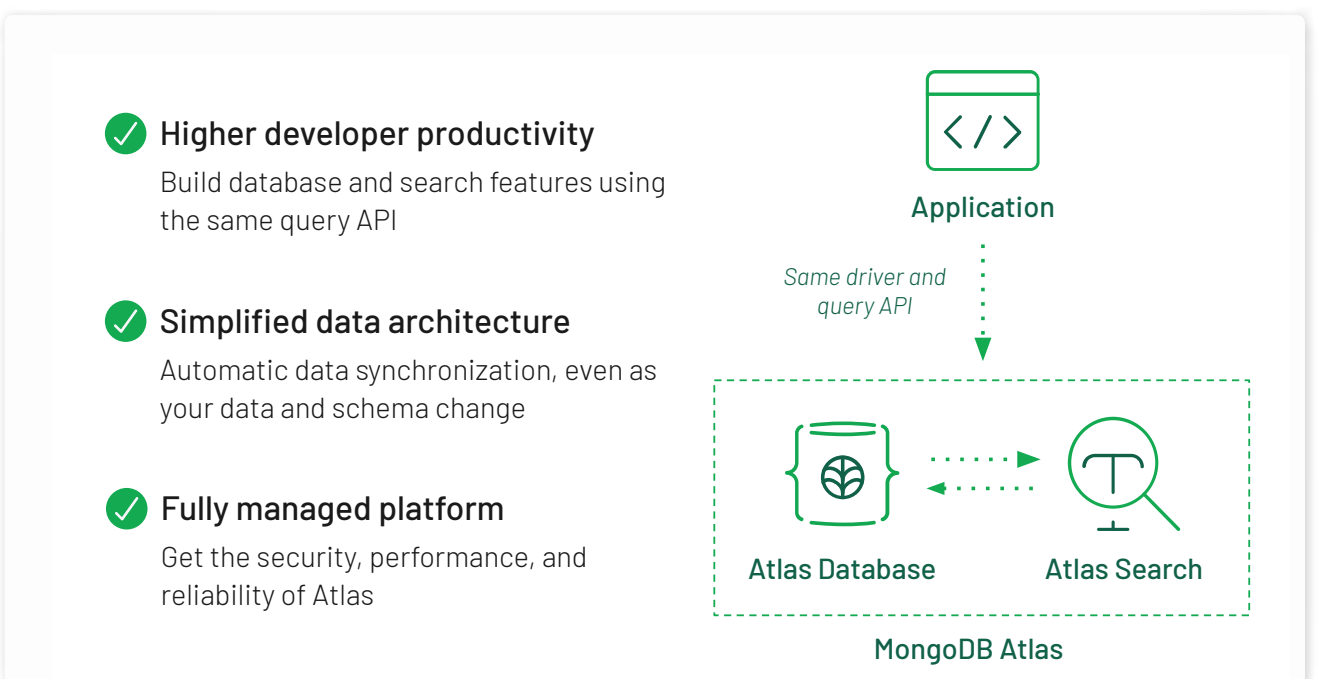

Figure 2: An integrated and unified platform for database and search improves developer productivity and operational efficiency.

How Does Atlas Search Work?

Index Creation and Data Synchronization

As soon as you create a search index, an Apache Lucene process is deployed alongside your database on each node of the Atlas cluster. Document fields that you want indexed are automatically synced from the database to Lucene. When the initial sync is complete, Atlas Search opens a change stream against the MongoDB database from which it receives notifications of all relevant data changes as they happen.

It then applies these changes to Atlas Search, keeping the indexes fresh in close to real time, using MongoDB's native, event-driven data streaming pipeline (see Figure 3).

This process is fully automated and transparent, so there is no need to write and maintain your own custom data sync, and there are no additional hardware or shadow copies of your data to manage. Because the database and search index processes are colocated on the same physical node, network hops are eliminated. This minimizes replication lag so you are serving fresher results to your users. Additionally, query performance is improved, returning results faster.

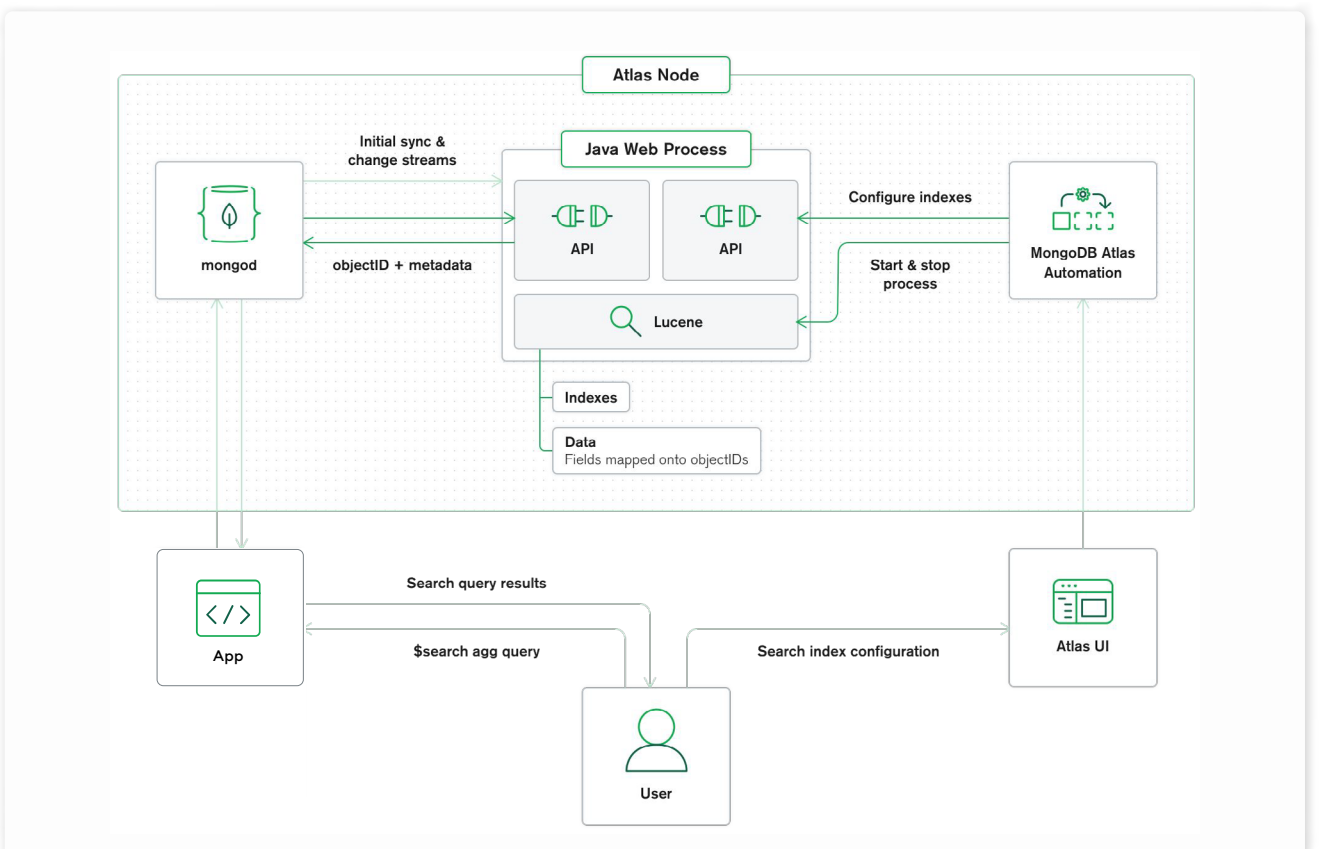

Figure 3: MongoDB Atlas Search architecture.

You can reconfigure or refine your search index at any time – for example, by modifying the fields you want to index or changing index analyzers. Atlas Search makes this process transparent and non-disruptive to your users. With no-downtime indexing, your applications can continue to read and write to your database and search cluster while the index is being rebuilt. Once Atlas Search rebuilds the index, the old index is replaced in the background without any further action from you.

Developer Experience

The advanced search features offered by Apache Lucene are exposed to developers via regular MongoDB Query API syntax, drivers, and tools.

Developers use a single, unified API and driver to work across both database and search, reducing context switching, simplifying their code, and eliminating unnecessary dependencies. As a result, they are building, testing, iterating, and shipping applications and new features faster.

Operational Experience

Atlas Search is an integrated service within the MongoDB Atlas application data platform. It is fully managed for you, with Atlas handling provisioning, replication, patching, upgrades, scaling, security, backup, and disaster recovery. Because Atlas Search sits right alongside your database, operations teams use the same APIs and UI, dramatically simplifying operational processes.

Through the Atlas tooling, you are able to monitor and visualize Atlas Search CPU, memory, and disk consumption alongside database metrics. You can also configure alerts to trigger when consumption exceeds predefined thresholds and configure auto-scaling so that your Atlas resources are automatically scaled up and down in response to application demand.

MongoDB Atlas is a global, multi-cloud service available in more than 80 regions around the world, so you get unmatched data distribution across Amazon Web Services (AWS), Microsoft Azure, and Google Cloud, all in a single cluster. You can take advantage of the best features of each cloud; deliver low-latency, highly resilient search to users in any country; and avoid lock-in to any one single cloud provider.

Key Atlas Search Features

Having an integrated and fully managed platform isn't useful if it doesn't provide the features you need to build compelling search experiences for your users.

Based on Apache Lucene – the same search library underpinning Elasticsearch and Apache Solr – Atlas Search provides the essential capabilities required for building search across multiple use cases, including those described below.

User Experience

- **Fuzzy search:** Providing typo tolerance, fuzzy search automatically surfaces relevant results even if the user incorrectly spells the search term or tries to use an inexact match. This gives more precision (accuracy) to the search results with lower user effort.
- **Autocomplete:** Also called type-ahead, autocomplete provides suggested search terms. This is especially powerful in helping users complete their search query faster with less typing, or when a user is unsure of the precise search term to use.
- **Highlighting:** By extracting snippets from a document and displaying them with a document's title, highlighting is an effective way of showing users why a result matched their search term, making it faster for them to identify the most relevant results.
- **Geospatial-aware search:** Accomplished with Atlas Search's geoShape and geoWithin operators, geospatial-aware search supports use cases that require distance-based sort orders in results sets.

Relevance

- **Custom scoring:** Enables developers and search engineers to tune the relevance of search results, either boosting or burying specific documents from the results set. This is useful when merchandizing specific offers or surfacing promoted content to users.
- **Synonyms:** Provides context-sensitive search by allowing alternative words to find related content. For example, "bike" also returns results for "bicycle" and "cycling"; "NoSQL" also returns "non-relational." This can be very powerful at expanding the recall (breadth) of relevant search results.
- **Analyzers:** Control how search terms are indexed and queries are parsed, such as where to break up word groupings, whether to consider punctuation and capitalization, and how to handle special characters and different languages. Atlas Search offers a number of built-in analyzers, along with the ability to create your own. Collectively, these analyzers allow you to customize your search experience for specific industries and locales.
- **Rich query API:** Combine search operators with other MongoDB stages in aggregation pipelines so you can build powerful application functionality that blends, transforms, and enables analytics against your data.

Speed

- **Facets:** Simplifies information navigation and discovery by grouping related search results into categories – for example, displaying vacation options by destination, trip type, and price band. Faceting also provides fast counts of all documents matching each category, helping users identify relevant results faster.

- **Index intersection:** Complementing MongoDB's powerful indexing and query planner, Atlas Search's index intersection allows complex ad hoc queries to use multiple term indexes in Lucene simultaneously to filter query results, providing higher performance when interrogating data using multiple predicates.

Beyond the ubiquitous search bar, these features also power many pieces of application functionality that don't require user input. Examples include social media feeds, trending topics, and content personalization and recommendations.

You can learn more about each of these features and how to get started with them from the Atlas Search documentation.

Beyond Search: A Complete Application Data Platform

Atlas Search is built on top of MongoDB, the most popular and widely used modern database in the market. MongoDB has become so popular because engineering teams can build and ship applications faster than other data platforms. You can get started with both MongoDB Atlas and Atlas Search in minutes on a fully managed service that handles operations for you – on any cloud you choose.

What makes MongoDB Atlas database and Atlas Search the right choice for you?

1. The document data model is **intuitive and flexible.** Documents map directly to the objects in your code so they are much easier and more natural to work with. You can store, index, and search data of any structure and modify your schema at any time as you add new features to your applications.
2. You work with **data as code.** The MongoDB Query API and drivers are idiomatic to your programming language. Ad hoc queries, indexing, full-text search, and real-time aggregations provide powerful ways for accessing, grouping, transforming, searching, and analyzing your data to support any class of workload.
3. With a distributed architecture, your database and search engine is **resilient and globally scalable.** Replication with self-healing recovery keeps your applications highly available while giving you the ability to isolate operational and search workloads on separate nodes within a single cluster. Native sharding provides elastic and application-transparent horizontal scale-out to accommodate your workload's growth, along with geographic distribution for data residency controls. These controls ensure that data is kept close to users for low latency and to comply with data sovereignty mandated by modern privacy regulations.

Database and search capabilities are the foundation of the MongoDB application data platform, providing a unified developer experience for modern applications that span from cloud to edge. You can easily extend the value of your data by using additional services, including:

- MongoDB Charts. Create, share, and embed visualizations of search results without having to move data into separate analytics or BI tools.
- MongoDB Realm Application Services. Benefit from a serverless and fully managed GraphQL endpoint that directly accesses Atlas Search from your search bar. GraphQL makes it fast to query and return results straight from a single endpoint, avoiding data over-fetching that is typical with REST APIs.
- MongoDB Realm Sync and mobile database. Simplify code when building offline-first applications that require data synchronization with your Atlas cloud back end.
- MongoDB Atlas Data Lake. Query and combine MongoDB Atlas application data with other data assets stored on Amazon S3. With Atlas Online Archive, you can configure custom retention policies to automatically tier aged data out of hot MongoDB storage onto low-cost S3 object storage. Data remains accessible with federated database queries that span both hot and cold data tiers using a single connection string from your application.

Use Cases for Atlas Search

Previewed in 2019 and in general availability as of 2020, Atlas Search has been widely used by customers of all sizes and across all industry sectors.

Customers can use Atlas Search at any stage of their application lifecycle. Some are building new applications that use Atlas Search from the start. Some are extending existing MongoDB workloads with new search functionality, while others are replacing existing database-plus-search bolt-ons with the unified Atlas platform.

There are three core use cases that Atlas Search is powering today:

1. **Product catalog** and **content search:** We define these as "search-first" use cases because the search bar is the primary interface for users to interact with the service.
2. **In-application search:** These are business applications supporting internal users or customer self-service portals where search is a supporting function used to enhance the application experience.

3. **Single view (i.e., customer 360):** As with application search, users interact with the single view via search as a supporting function. What's different is that the single view application itself relies on specific Atlas Search capabilities that make it much easier to unify disparate data ingested from multiple sources.

In the following section, we provide a definition of each use case, along with the required capabilities mapped to Atlas Search features and examples of customers using Atlas Search today.

Product Catalog and Content Search

With online and mobile sales volumes growing at around 50% every year – growth that has been accelerated by the COVID-19 pandemic – it is vital for companies to deliver the best possible experience that drives conversions when customers browse and search their ecommerce product catalogs.

At the same time, the increasing ubiquity of high-speed internet connectivity and smart mobile devices is changing the content management landscape. Sites have to create engaging, relevant, and immersive experiences enlivened with rich media assets and user-generated content, all of which have to be discoverable with low latency for any device, anywhere on the planet. Content search use cases include websites, digital and social media, online publications, research and training materials, documentation, user forums, and image repositories.

MongoDB is already widely used for both product catalog and content management, with application owners taking advantage of MongoDB's flexible document data model, distributed architecture, and rich query API. The relevance of each of these is summarized in the following table. The addition of Atlas Search means organizations can deliver rich and intuitive search experiences without having to bolt on an external search engine.

As "search first" use cases, product catalog and content search are the most demanding of all search applications, relying on all of the capabilities discussed in the earlier "Key Atlas Search Features" section. Table 1 highlights the most important requirements.

Required Capabilities	Why MongoDB?
User search experience 1. Quickly find the most relevant matches to products or content using flexible search terms in a variety of languages. 2. Intuitively research and compare different product and content categories. 3. Receive a concise summary of product or content directly within the search results. 4. Perform geospatial-aware search (for use cases requiring distance-based sort orders). 5. Boost preferred search results for merchandising or content promotion.	**Rich search capabilities** 1. Fuzzy search, autocomplete, synonyms, and analyzers help users get the right search results faster. 2. Faceted search and counts help users efficiently navigate categorized search results. 3. Highlighted extract snippets help users understand a document's relevance. 4. Geospatial search allows users to filter and return results by location. 5. Custom scoring returns sponsored or preferred documents higher up in the results set.
Store and query complex data 1. Handle massive variability in catalog and content-management attributes, metadata, media assets, and user-generated content. 2. Quickly update the schema as new products and content are added.	**Document data model** 1. Store complex, multi-structured data in a polymorphic and flexible schema with support for rich data types. 2. Avoid disruptive schema migrations via a dynamic schema that instantly adapts to accommodate new data models.
User personalization and data insights 1. Serve personalized product or content recommendations that improve sales conversions and reduce bounce rates. 2. Monitor sales performance and content consumption in real time.	**Real-time analytics** 1. Capture clickstreams and sales attributions in MongoDB time series collections, exposing events to data science tools using the MongoDB Spark connector or R and Python drivers to tune scoring, expand synonyms, etc. 2. Create and share dashboards for real-time insights and reporting with MongoDB Charts.
Application resilience and performance 1. Benefit from always-on, low-latency search. 2. Never slow down under peak loads generated by promotions, seasonal shopping events, or new publications.	**Distributed architecture** 1. Replica sets for built-in redundancy and self-healing recovery. 2. Deploy multi-region clusters for wider geo-resilience and data colocation close to users. 3. System resources auto-scale up and down in response to user demand; scale out as data volume and customer base grows via native sharding.

Table 1: Required capabilities for "search first" applications.

Atlas Search in Action

Keller Williams is one of the largest real estate companies in the world. Employing 190,000 people, the company closed on 1.2 million homes in 2020, representing sales volumes in excess of $400 billion. The business relies on its web and mobile sites, underpinned by two core MongoDB Atlas databases to connect buyers with properties and agents.

- The Fast Facets database supports faceted search so customers and agents can quickly browse information about multiple properties for sale in a specific area.
- The Master Dataset database, or MDS, contains all of the property details that come into play once a prospective buyer is interested in a particular property and wants to drill down to learn more.

Atlas Search is a key part of the Keller Williams platform, providing fast and intuitive geospatial-aware search that matches properties to each customer's search criteria.

Atlas Search powers a diverse range of catalog and content management use cases across many industry sectors.

One of the world's **largest auto manufacturers** uses MongoDB Atlas to unify its after-sales parts product catalog and the content management system that stores manuals and maintenance procedures. It replaced Elasticsearch with MongoDB Atlas to reduce data duplication and simplify its architecture, freeing developers to build new applications faster and cut operational overhead. Atlas Search is used by its B2B marketplace that serves hundreds of thousands of users across 160 countries and 30 languages. Autocomplete, synonyms, and fuzzy search enable users to quickly find the parts and manuals they are looking for. The scalability of Atlas and the rich search functionality provided by Atlas Search become even more valuable as the company opens up its marketplace to B2C channels and millions of new users.

Humanitix powers search across its events catalog and content with Atlas Search. Donating profits from ticket sales to educational projects that help disadvantaged children around the world, Humanitix always seeks to minimize costs. Bringing database management and search together in MongoDB Atlas creates a much simpler, integrated architecture that scales as the organization expands into new geographic markets.

One of North America's fastest-growing **medicinal supply companies** uses MongoDB Atlas as the back end to its ecommerce platform. Its product catlaog runs on MongoDB and uses Atlas Search and its geospatial operators to connect customers with local dispensaries in their area.

The world's leading **3D geometric deep-learning software company** uses Atlas Search as part of its digital twin platform used by manufacturers in their engineering assemblies and designs. The metadata for more than 30 billion 3D-rendered components is stored in the MongoDB database and indexed by Atlas Search, making it fast and easy for engineers to find what they need with exact search precision.

In-Application Search

Whether building applications to support internal business users or portals for customer self-service, the search bar has become an essential feature of any UI, serving to enrich the overall application experience. With the added power of search, users and customers can quickly navigate order, inventory, payments, claims, audit logs, employee information, accounts data, and more.

If search isn't available or isn't implemented well, internal users will waste a lot of time trying to find the right information and customers will defect to competitors who can better serve them.

Although search requirements for general-purpose in-app experiences are not as broad as they are for product catalogs and content management, there are still several critical capabilities you need to provide alongside the database powering the application (see Table 2).

Required Capabilities	Why MongoDB?
User search experience 1. Quickly find relevant records, allowing for typos, similar terms, and incomplete query criteria. 2. Locate the right record from a large corpus of information using complex query criteria. 3. Expose high-performance search directly from the application's UI.	**Rich search capabilities** 1. Fuzzy search, synonyms, and autocomplete help users get to the right results faster with less effort. 2. Index intersection evaluates multiple predicates to efficiently return matching records. 3. MongoDB Realm allows developers to simply click to enable a GraphQL endpoint behind their search bar, providing direct access to Atlas Search.
Balance fast application development evolution with data integrity 1. Store records without lengthy upfront schema design or complex object-relational mapping (ORM) abstractions. 2. Quickly build and deploy new application functionality. 3. Ensure data quality and accuracy for business-critical applications.	**Document data model with transactional guarantees** 1. Use documents that match the objects developers work with in code. 2. Dynamic schema instantly adapts to accommodate new app features and data structures, avoiding disruptive schema migrations. 3. Strong data consistency, backed by ACID transaction guarantees. Schema validation provides centralized governance over your data models.
Powerful application functionality and business insights 1. Enable everything from simple lookups through to sophisticated data transformations and aggregations via support for multiple access patterns and query types. 2. Deliver insights to applications in real time. 3. Monitor business performance and automate optimizations via dashboards, BI, and machine learning. 4. Tier and query data anywhere.	**Fast queries and real-time analytics** 1. Support multiple query types with the expressive MongoDB query API and secondary indexing. Use aggregation data pipelines for in-database data preparation, readying data for analytics. 2. Group and count related data with Atlas Search facets. Increase query speed by caching common result sets with Materialized Views. 3. Create and embed data visualizations with MongoDB Charts. SQL integration for BI tools with the BI Connector. AI automation with the Spark Connector and idiomatic R and Python drivers. 4. Tier aged business data to S3 by using Atlas Online Archive, then federate queries across storage tiers via Atlas Data Lake.
Resilience, performance, and regulatory compliance 1. Ensure always-on applications. 2. Scale to meet business growth. 3. Protect data security and preserve user privacy.	**Distributed architecture with end-to-end security** 1. See capabilities list in Table 1. 2. See capabilities list in Table 1. 3. Complete security protection with fine-grained data access controls; encryption of data in flight, in use, and at rest; audit log for forensic analysis of system activity; and data sovereignty policies to meet modern privacy regulations.

Table 2: Required capabilities for application search.

Atlas Search in Action

Current is one of the United States' fastest-growing challenger banks, serving several million customers and doubling in size every six months. Its core banking platform runs on MongoDB Atlas and Google Cloud. Every transaction is stored in the MongoDB database, with Atlas Search enabling users to quickly browse each payment and track rewards points. Atlas Search is also used to connect account holders, providing faster and easier access for peer-to-peer payments. Current had initially considered using Elasticsearch, but saw the opportunity to simplify its technology stack and eliminate the overhead of data synchronization between separate systems by using Atlas Search.

One of the world's largest **home fitness companies** uses Atlas Search to help its course instructors and in-store staff curate and construct playlists from the company's internal web portal. Fuzzy search and faceted navigation help the organization quickly find the right mix of tracks for each session and log consumption in the database so it can pay royalties to each artist.

A **multinational convenience store chain** with more than 70,000 locations around the world uses Atlas Search in its internal web portal. Store managers use fuzzy search to quickly track internal inventory and to browse internal reports that track sales data. The company is also using Atlas Search in a new self-checkout system currently in trial in select stores. If the point-of-sale system fails to capture the barcode, the system scans the item's shape and weight and passes the results to Atlas Search, which returns a list of likely options to the customer, making checkout even more convenient.

One of the world's largest **stock exchanges** is using Atlas Search to power its internal credit risk application, improving developer velocity by over 30% compared to its previous solution. Prior to the addition of Atlas Search, market data feeds had to be pre-processed in application code before being loaded into MongoDB Atlas, where they could then be exposed to users for basic regex search.

Atlas Search significantly reduced the complexity of the data-ingestion pipeline by allowing raw data to be loaded and indexed, using standard features like fuzzy search and autocomplete to help internal users quickly find top trending articles for each company they are following.

Single View

A single view, sometimes called Customer 360, aggregates data from multiple source systems into a central repository to create a single view of a customer. By creating this single, real-time view, organizations enhance business visibility and enable new classes of analytics to better understand and serve their customers.

Even before the availability of Atlas Search, MongoDB was well established for single-view use cases. LCL, a subsidiary of Crédit Agricole and one of the major retail banks in France, uses MongoDB to build a single view of its customers, cutting development

time to market by 40%. Alight Solutions, formerly a part of Aon Hewitt, migrated from its legacy mainframes and built a single employee view on MongoDB, improving application performance and customer experience by 250x.

These and many other customers have been able to take advantage of MongoDB's flexible document data model, rich query API, and distributed systems architecture – all captured in Table 2 above – to build their single customer view.

Atlas Search adds two critical new capabilities to single view applications: fuzzy search and autocomplete. One of the toughest challenges in single view projects is data reconciliation – the process of unifying the identity of a customer ingested from multiple source systems. It is not uncommon, for example, that in a CRM system the customer has one identifier, in an after-sales system they have another identifier, and in a billing system, yet another. Fuzzy search and autocomplete are incredibly powerful for reconciling these disparate identities into a single customer record for the application's user.

One of the **global top 10 insurers** is using Atlas Search on top of the single view built on the MongoDB database to reconcile fragmented customer records. Data is ingested from the company's back-end databases and Master Data Management (MDM) system, where it is then exposed to 10 different applications – from call center to internal business processes to its customer self-service portal.

Fuzzy search enables disconnected customer details and policy information to be unified within the application's UI. Without fuzzy search, the insurer's development team would have had to write its own application-side logic to try to match these records at runtime. With Atlas Search handling this requirement, the insurer's users and customers get higher performance, and internal development effort is reduced.

A **major European energy provider** is using Atlas Search for a very similar use case. All customer data from the company's backend CRM and ERP systems is ingested into the single view stored in MongoDB. Atlas Search is used to expose this data to internal business users and to customers via its self-service web portal. As the company expands into smart home solutions, the single view will become even more critical for user experience, enabling customers to track energy consumption and cost down to the level of individual devices.

Is Atlas Search Always the Right Solution?

As the use cases above demonstrate, Atlas Search is a highly capable solution for a broad spectrum of search requirements.

Because Atlas Search is tightly integrated with MongoDB Atlas, all data first has to be loaded to MongoDB database collections in order to create the required search indexes. There are some highly specialized applications that need to aggregate and search across multiple data repositories – databases, data warehouses, object stores, file systems, message queues, API gateways, and so on. Examples include DevOps observability platforms; security-event and threat-hunting applications that are continuously ingesting and searching log data from source systems; and enterprise-wide document management systems.

For these use cases, it is better to use MongoDB as one of your data sources alongside a dedicated search engine. In these scenarios, some dedicated search engines provide a number of built-in connectors and agents to extract data from multiple source systems, index them (typically with Lucene), and then make data searchable with custom-built tools.

Getting Started With Atlas Search

Atlas Search is designed as a self-service platform. You have to be running a MongoDB Atlas database (version 4.2 and up). From there you can use your own data or load one of our sample data sets to try it out. Create a search index from the Atlas UI, CLI, or API, and then start querying your data.

Atlas Search is available with all Atlas clusters – including free clusters – so you can evaluate it at no cost. Our Getting Started tutorial steps you through the process. Atlas Search documentation provides a complete reference on how to configure, manage, and query search indexes, along with performance recommendations. The MongoDB Developer Hub and MongoDB YouTube channel provide a wealth of articles and tutorials for beginners through to expert users.

Our professional services team can also support you at any stage of your application lifecycle. They can partner with you throughout a project to implement sophisticated solutions, including Atlas Search and other components of the MongoDB application data platform, as well as help drive longer-term strategic initiatives.

With Flex Consulting, our consulting engineers help teams address questions about Atlas Search in short, technical sessions:

- The **Design and Develop** track helps you apply best practices and patterns as you start out. Our consultants work with you to configure and tune your indexes, map fields, choose the right analyzers, and use more advanced Atlas Search features such as synonyms, autocomplete, custom scoring, and highlights, alongside Atlas Search operators.
- The **Migrate and Simplify** track is designed to move data into MongoDB Atlas from existing databases and external search engines.
- **Optimize** is useful for existing Atlas Search deployments. It will help you find opportunities to further improve search performance, recall, and precision by analyzing execution statistics and explain plans, tuning index mappings, and optimizing query design and system sizing, along with advanced topics such as rightsizing edgeGrams and nGrams for autocomplete.

Whether you are building a new application, extending an existing MongoDB workload, or looking to simplify your application estate, Atlas Search makes it easy to get started, and makes your user experiences more engaging and delightful. Try Atlas Search on a free cluster today and see for yourself.

Safe Harbor

The development, release, and timing of any features or functionality described for our products remains at our sole discretion. This information is merely intended to outline our general product direction and it should not be relied on in making a purchasing decision nor is this a commitment, promise or legal obligation to deliver any material, code, or functionality.

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