5-Step Guide to Replacing Elasticsearch and Apache Solr with Atlas Search

Application Search: Reduce Complexity, Ship Faster

March 2024

Table of Content

Introduction

Type almost anything into a search bar on sites like Google, Amazon, and Netflix and we are instantly presented with relevant results. Whether we make a typo or enter a partial search term, the search engine figures out what we are looking for. Results are returned conveniently sorted by relevance and are easy to navigate with features like highlighting, filters, and counts.

These fast and intuitive search experiences are so pervasive that we now expect them in every application we use, whether at home or at work – from checking our bank accounts and utility bills to querying back-end business systems for orders, customer information, employee records and more.

However, building these experiences is hard. In many cases, developers have to bolt-on a specialized search engine to their application's database and create a replication mechanism to keep the two systems synchronized.

Apache Lucene-based search engines such as Elasticsearch and Apache Solr were the technologies many developers turned to. While both are powerful, they introduce a huge amount of complexity to the application stack, reducing developer velocity while driving up risk, complexity, and cost. This is because:

1. There are now three separate systems in the application stack: a database, a search engine, and a sync mechanism.

2. Developers have to work with multiple query languages and drivers across database and search to build and evolve their applications. They also have to create transformation logic to handle differing data structures and schema changes between the database and search engine.

3. Operations teams have to care, feed, and secure additional sync and search infrastructure alongside the database.

MongoDB Atlas Search gives you a much better way to build search experiences into your applications. Embedding Apache Lucene — the same technology underpinning Elasticsearch and Solr – you **compress three separate systems into one** and ship new applications and features faster. Customers have reported improved development velocity of **30% to 50%** after adopting Atlas Search.

By bringing together the database, search engine, and sync mechanism into a single, unified and fully managed platform, Atlas Search is the fastest and easiest way to build relevance-based search directly into your applications.

By integrating Search with the database, Atlas Search increases agility and lowers operational overhead while increasing availability, scalability and performance thanks to dedicated infrastructure.

In this guide, we present a 5-step methodology for replacing Elasticsearch and Solr with Atlas Search. The guide steps you through how to:

1. Qualify target workloads for Atlas Search.
2. Migrate your indexes to Atlas Search.
3. Migrate your queries to Atlas Search.
4. Validate and relevance-tune your Atlas Search queries and indexes.
5. Size and deploy your Atlas Search infrastructure.

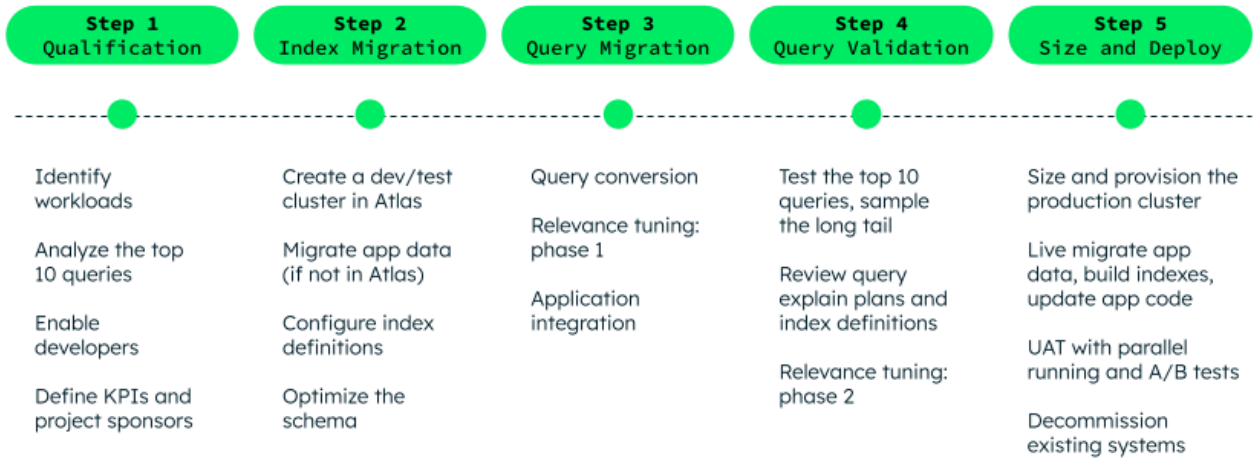

Figure 1. 5-step methodology to replacing Elasticsearch and Solr with Atlas Search

The guide wraps up with examples of customers that have made the switch and provides guidance on how to get started with Atlas Search.

The 5-step guide is designed to help you plan and execute your migration project. MongoDB's Professional Services team is also available to you as a trusted delivery partner. We can help you through any of the individual steps in the methodology shown above or throughout your entire journey to Atlas Search. The Atlas Search Migrate package provides you with a repeatable migration and testing plan to seamlessly move

your search indexes and queries over to Atlas Search. Throughout the migration process, our consultants support you with query validation, relevance tuning, and sizing. Review the Getting Started section at the end of the guide for more information on our Professional Services offerings. .

What is Atlas Search?

MongoDB has always focused on accelerating and simplifying how developers build with data for any class of application. From our very earliest MongoDB releases, we saw developers needing to expose their data stored in the database to application search.

For simple use cases – where it was enough to just match text in a field – developers were able to use the basic text search operators and index built into the MongoDB database. However these lacked the much more sophisticated speed and relevance tuning features offered by dedicated Lucene-based search engines.

As a result many developers ended up bolting on an external search engine to their database, introducing the complexity we discussed in the Introduction and visualized in Figure 2 below.

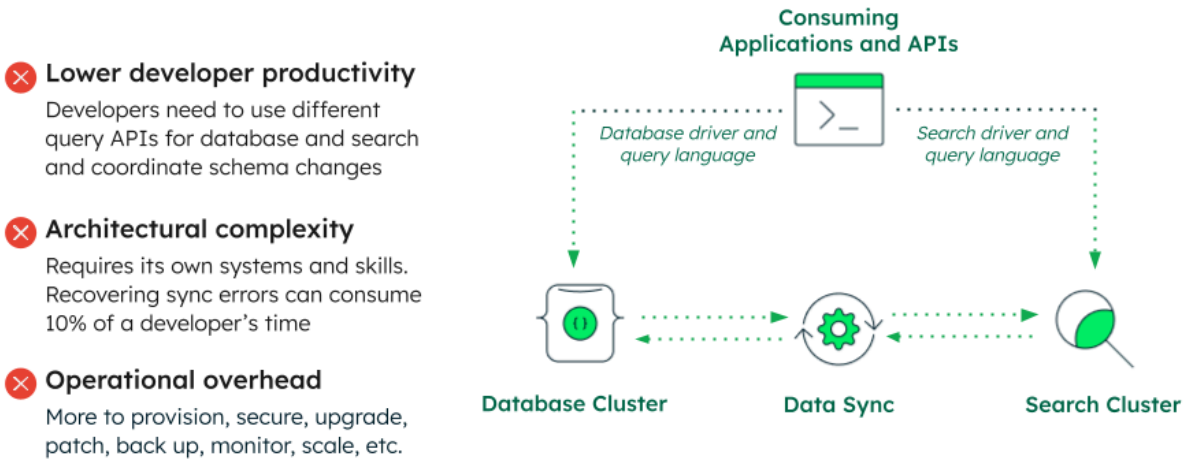

Figure 2. The pain of bolting on a search engine to your database

Working with the MongoDB community, our product designers and engineers ideated on ways to make building application search easier for developers – without compromising on the key features they needed. The result is MongoDB Atlas Search.

Atlas Search embeds an Apache Lucene search index directly alongside the database. With a couple of API calls or clicks in the Atlas UI, you immediately enable fast, intuitive search and information discovery within your applications. By replacing bolt on Elasticsearch and Solr with Atlas Search you get to do three things:

1. **Eliminate the synchronization tax.** Data is automatically and dynamically synced from your Atlas database to your Atlas Search indexes. You avoid having to stand up and manage your own sync mechanism, write custom transformation logic, or remap search indexes as your database schema evolves. You escape the 10% of engineering cycles typically lost to manually recovering sync failures[1], investing that time to innovate for your users instead.

2. **Ship new features faster.** You work with a single, unified API across both database and search operations, simplifying query development. No more context switching between multiple query languages, and with a single driver, build dependencies are streamlined so you release faster. You can test queries and preview results with interactive tools to fine-tune performance and scoring before deploying them directly into application code.

3. **Remove operational heavy-lifting.** The fully-managed Atlas platform automates provisioning, replication, patching, upgrades, scaling, security, and disaster recovery while providing deep performance visibility into both database and search. By working with a single system, you avoid an exponential increase in the number of system components you need to design, test, secure, monitor, and maintain.

4. **Ability to scale search and database independently.** Switching to dedicated infrastructure means higher availability, with a lower likelihood of downtime due to load balancing during any outage event. In some cases we've seen 60% faster query time for some users' workloads for complex workloads, leveraging Search Nodes.

[1] Based on interviews with engineering teams that have replaced bolt on search engines and the associated sync mechanism.

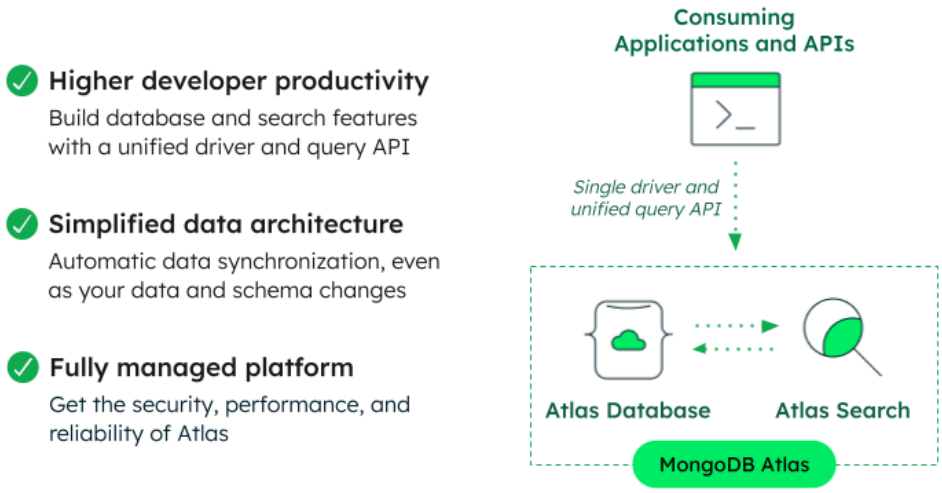

Figure 3. Dramatic architectural simplification with integrated database, sync, and search in MongoDB Atlas

So how do you make the switch from your bolt on Elasticsearch or Solr search engine to Atlas Search? The following five steps provide you with a proven and repeatable methodology for migration.

Step 1: Qualification

The first step in the migration process is to qualify whether your workload is appropriate for Atlas Search. This qualification needs to be based on your current application requirements and those you are planning for in the future.

The following sections outline those use cases where Atlas Search is a good fit, and those where you are better placed sticking with your existing solution.

Identifying Appropriate Atlas Search Use Cases

Atlas Search is designed for application search where you would previously bolt-on an external search engine to the application's database. These use cases fall into three main categories:

1. **Product catalog and content search:** The search bar is the primary interface for users to navigate the product catalog or content metadata stored in the MongoDB Atlas database.

Example customers using Atlas Search for catalog and content search include Keller Williams – one of the world's largest real estate agents; StoreHippo – India's largest eCommerce SaaS platform for mid and large sized enterprises who replaced Algolia and Azure Cognitive Search with Atlas Search for its product catalog service; and CNFT.IO, the first and largest NFT marketplace trading on the Cardano blockchain.

2. **In-app search:** Line-of-business applications supporting internal users or customer self-service portals where search is a supporting function used to enhance information discovery. Examples include searching across customer orders and inventory, insurance claims, credits and debits in an online banking application, sales or employee data, and more.

   Example Atlas Search customers include Current – one of the United States' fastest growing challenger banks; a global stock exchange powering credit risk applications; and a multinational convenience store chain for inventory management and customer self-scan checkout systems.

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 such as fuzzy matching and autocomplete. These capabilities make it much easier to query disparate data ingested from multiple sources into the single, 360-degree view.

   Example Atlas Search customers powering single view include a global top 10 insurer and one of Europe's largest energy providers.

Atlas Search is part of MongoDB Atlas, the multi-cloud developer data platform. MongoDB Atlas combines transactional processing, relevance-based search, real-time analytics, mobile edge computing with cloud sync, and cloud data lake in an elegant and integrated data architecture. Through a flexible document data model and unified query interface, Atlas provides a first-class developer experience to power almost any class of application. At the same time it meets the most demanding requirements for resilience, scale, and data privacy.

The Transforming Customer Experience with Atlas Search white paper provides more details on target use cases for Atlas Search, and how its integration into the Atlas developer data platform helps you meet a complete set of requirements for building modern applications.

Query Analysis

Atlas Search provides the features needed to deliver rich and personalized experiences to your users. These include fuzzy matching, autocomplete, lightning fast facets and counts, highlighting, custom scoring, geospatial queries, and synonyms, all backed by support for multiple analyzers and languages.

The quickest way to qualify Atlas Search for your specific use case is to identify the 10 most common search queries your application is running today. In many cases these top 10 queries often represent ~80% of all search operations. You can identify these queries by inspecting the logs of your existing Elasticsearch or Solr engine. From there you can map the operators used in these top 10 queries to the Atlas Search feature set.

For some application search use cases both Elasticsearch and Solr can be configured with custom plug-ins that extend core Lucene functionality. These are typically used to add support for additional tokenizers, filters, and file formats. If you are using custom plug-ins in your current application search deployment, you should contact your MongoDB representative. They can work with one of our search specialists to determine whether that functionality can be replicated with Atlas Search.

Technology Enablement

To help with the qualification there are many resources available that will get your developers up to speed with Atlas Search.

The Atlas Search documentation provides reference materials and tutorials, while the MongoDB Developer Hub provides sample apps and code. You can spin up Atlas Search at no-cost on the Atlas Free Tier and follow along with the tutorials using our sample data sets, or load your own data for experimentation within your dedicated sandbox.

For a more curated experience we also offer a one day instructor-led Atlas Search training class (search for DS110). The class helps identify appropriate use cases and features for Atlas Search. It then introduces students to key Atlas Search concepts including how to build indexes, write queries, tune relevance, and size their systems. It is a highly effective and efficient way of building your team's foundational skills in Atlas Search in just 8-hours of expert instruction and hands-on exercises.

KPIs and Sponsorship

For any migration project to be successful, it is important that its Key Performance Indicators (KPIs) are defined up front and project sponsors be appointed. Both the business and technology sides of the organization should be represented. The sponsors need to have both the credibility to prioritize the project in their teams and the authority to allocate resources to it.

From the business side of the organization, representation will typically come from the application's Line-of-Business (LoB) owner. For a product catalog application, this could be the head of merchandising, while for a customer single view, it might be the head of customer service.

For the technology side of the house, representation should include:

- Architects responsible for designing and integrating systems across your technology estate.
- Developers responsible for the application code, and if applicable, search engineers who tune search relevance (more common in product catalog use cases).
- DevOps engineers responsible for maintaining the application infrastructure.

KPIs should be unified across the teams, but may be articulated in different ways. From replacement projects we have worked on, the following table demonstrates how KPIs have been aligned across teams (note that the metrics quoted below are for illustration only).

Business KPI	**Technology KPI**
30% improvement in time to market for new features	30% improvement in developer productivity
Improved customer satisfaction via higher quality search results surfacing new data within 10 seconds of ingestion	30% reduction in database -> search sync latency and time taken to recover from sync errors
30% reduction in opportunity cost and business risk in delivering new features	30% reduction in operational overhead

While it is often assumed that reducing TCO is a critical KPI of Elasticsearch and Solr replacements, from our experience the project is usually driven by a desire to reduce opportunity costs – specifically accelerating the delivery of new features that provide competitive advantage to the business. These reductions are realized by the developer velocity improvements and architectural simplification Atlas Search delivers. Financial savings are important, but tend to be a second order consideration.

Which Search Use Cases Should You Qualify Out?

As noted earlier, Atlas Search is designed for application search use cases. By design, it is tightly integrated with the MongoDB Atlas platform. Therefore all data has to first be stored in MongoDB database collections in order to then create the required search indexes against it.

Atlas Search is not currently designed for log analytics typically used in DevOps observability or security and threat hunting applications. Atlas Search is also not suitable for enterprise-wide search systems. In these scenarios, Elasticsearch and Solr provide built-in connectors and agents to crawl and extract data from multiple internal source systems, index them, and then make data and analytics searchable with bespoke tools.

For these use cases, it is better to use MongoDB as one of your data sources alongside your existing Elasticsearch or Solr search engine.

Step 2: Index Migration

The next step in the project is to migrate your indexes to Atlas Search. You should start by creating a Dev/Test environment in MongoDB Atlas. This should be configured with the appropriate amount of storage to accommodate your test data set.

Atlas Search automatically indexes the data stored in the MongoDB Atlas database, so the next task in Step 2 is to make sure your data is available in Atlas.

If your application data is already stored in MongoDB, then the data migration effort is minimal. If the data is stored in another database, then there is more work needed. We will consider both scenarios in turn.

Application Data Stored in MongoDB

If your application database is currently running in MongoDB Atlas, then you can move straight to the tasks below describing how to identify fields to index and optimize your schema.

If you are self-managing MongoDB, then you first need to migrate your data into the Atlas dev/test environment. At this stage of the process, you may choose to load just a subset of your data. There are a variety of tools available to help move data into Atlas. You can restore a backup taken from your existing deployment or live migrate data from that existing deployment into your new dev/test cluster. Restoring a backup is a simpler process at this development stage of the project. Live migration is discussed later in the deployment step of the project.

Application Data Stored in Another System

If your application data is stored in anything other than MongoDB, you will first need to migrate it to a MongoDB Atlas database. The complexity of this migration will depend on your current data store.

Some teams may store their application data as well as indexes in Elasticsearch or Solr. Like MongoDB, both systems store data as JSON documents. So all you need to do during the development phase is dump the existing data out as a JSON file and then use mongoimport to load it into MongoDB. Refer to Step 5 later in this paper which discusses how to perform a live migration of data into MongoDB Atlas.

If you are migrating from a relational database, then you will need to remodel your schema from tabular structures to rich JSON documents. The MongoDB documentation provides guidance and best practices for data modeling in MongoDB. To support the migration you can also use the Professional Services discussed later in the Getting Started section.

Identify and Configure Fields to Index

With your data in the Atlas database, the next task is to identify the fields you need Atlas Search to index.

By default, Atlas Search will create a dynamic mapping that automatically indexes all fields within your collection. This is best used when you are getting started with Atlas

Search for a new application or have a document schema that frequently changes. While dynamic mapping is extremely convenient and flexible, be aware that using it will consume more RAM and disk storage. The alternative to dynamic mapping is to use static mapping, which allows you to selectively define specific fields to index[2].

Rather than reverse engineering your existing queries to identify which fields to index, a quick and convenient way to determine static mappings is to review the field mappings already created for your Elasticsearch or Solr index. You can then apply those to the index definitions you create for Atlas Search.

You should take the opportunity to rationalize your indexes by comparing the existing Elasticsearch and Solr field mappings with the search engine logs. This will identify those fields that are actually being queried. As your application has evolved over time, it's not uncommon to find that some indexed fields are no longer being queried. These can therefore be skipped in the index definition you create for Atlas Search, resulting in more efficient indexes..

By analyzing the existing field mappings you will also identify the analyzers being used to tokenize field values and whether any synonym mappings have been created:

- Atlas Search provides a number of built-in analyzers as well as the flexibility to create your own custom analyzer. This can be configured to meet the specific indexing requirements of your application.
- You can also recreate synonym mappings in Atlas Search and apply them when you define or modify your index.

Schema Optimization

A key advantage of the document data model over traditional relational databases is the ability to model data in the same structure as objects in your application. By nesting related data in sub-documents and arrays, documents bring together objects that would otherwise be normalized across separate tables. This approach eliminates expensive join operations and makes documents much more intuitive for developers to work with.

While Atlas Search indexes common nested fields, it does not currently index numeric, date, or boolean values if they are stored in an array. If you need to index these values, you have two options:

1. Flatten your schema by moving the array values into a top level field and indexing that.

[2] To learn more about dynamic and static index mapping, review the Atlas Search documentation.

2. Convert the array values to string data types before indexing them. The MongoDB $convert operator can recast the data types for you.

Unlike rigid relational database schema, MongoDB's document schema is dynamic and flexible so you can make these changes to your data model without first having to run an expensive ALTER_TABLE command.

Step 3: Query Migration

Query Conversion

Developers coming from Elasticsearch or Solr backgrounds will find both the JSON document structure of the Atlas Search query syntax and the consistent naming of the query operators immediately familiar.

Similarly, developers who have worked with the MongoDB database will see that Atlas Search is integrated into the regular aggregation pipeline via the $search stage. While $search introduces new query operators, the learning curve will be low.

The Atlas Search query documentation includes reference material on all of the operators available to you. The hands-on section of the documentation takes you through tutorials and code samples showing how to write common search queries. The tutorials and sample code include guidance on how to use Atlas Search features such autocomplete, facets, and synonyms; how to tune relevance; and run geospatial queries. Both the reference material and tutorials enable your teams to quickly convert queries written against Elasticsearch or Solr to Atlas Search.

Relevance Tuning

As well as converting query code, you should also use this step of the project to ensure you are appropriately tuning relevance for your common application queries. You should review how your existing search solution is scoring query results and apply equivalent scoring rules to your Atlas Search queries.

Relevance tuning is not a one-off, "set it and forget it" exercise. We will come back to it in the latter steps of the project.

Application Integration

Whether your application accesses Elasticsearch and Solr directly through programmatic drivers or via an API abstraction, repointing them to Atlas Search is straightforward:

- All of MongoDB's official drivers support Atlas Search, making it accessible and idiomatic to developers using any of today's most popular programming languages.
- The Search Tester, available from the Atlas UI, is useful for getting started with simple queries. By entering a search term you can review results, tune relevance, and then export the generated query code directly into your application.
- As an alternative to the drivers, your applications can also access Atlas Search via the MongoDB GraphQL API and the REST-like MongoDB Data API. These provide an even broader choice of ways to integrate your application with Atlas Search.

Step 4: Query Validation

With your indexes and queries migrated to Atlas, the next step is to validate query correctness.

Test the Most Common Queries and Sample the Long Tail

The quickest way to get started with validation is to reuse your 10 most common queries identified back in Step 1 (query qualification) and run them against Atlas Search to compare results – evaluating both relevance sorting and the number of results returned.

Bear in mind that if you had just loaded a snapshot of your data in Step 2, then Atlas Search will not have the latest indexed data available to Elasticsearch or Solr. However this exercise will still provide you with solid insight into the validity of results returned by Atlas Search and how they have been scored. This provides the opportunity to tune relevance as needed. As with any search engine, tuning is an iterative process repeated through the lifecycle of the application, and will likely require evolving your code to optimize field mappings and scoring rules.

When you are satisfied with the results from the top 10 queries, then sample the long tail of less frequently run queries. The number of queries you sample will be dependent on your application. Some of these long tail queries may be especially important for

application functionality, even if they are not run that often. There are a couple of ways you can identify these queries to ensure they are tested:

1. Work with the application owner and lead developers who should have knowledge of the queries.
2. Review whether the queries are included as part of your existing automated regression test coverage.

You can also use the testing phase of the project to identify common queries that failed to return results with your existing search engine. Use this analysis to tune your Atlas Search indexes, synonyms, and queries accordingly.

Explain Yourself!

As well as verifying relevance, you should also validate the speed of your queries, ensuring you can achieve the performance requirements of the application.

The Atlas Search explain method returns the query plan and associated execution statistics. Output from `explain` will show you what Lucene queries were run under the hood, parameters such as the analyzer used, boost coefficient and filter values, and execution time of a given query.

Step 5: Deploy

With your search queries validated, now is the time to deploy Atlas Search to production.

Cluster Sizing & Performance Optimization

The first task in this stage is to size your Atlas deployment to meet your application's throughput and latency SLAs. Atlas Search sizing will be dictated by a number of factors including:

- Query volume.
- The number and size of documents and fields indexed.
- Write load from the application.
- Specific analyzers configured.

- Features used such as autocomplete, highlighting, and synonyms

The performance considerations section of the documentation provides practical guidance and best practices on how to optimize Atlas Search performance by configuring your index definitions and operators. It also provides the same for constructing queries to optimize filtering, scoring and sorting of the results set.

If you started out with Atlas Search by configuring your index with dynamic mapping, now is a good time to redefine your index with static mappings. At this stage, you will have a clearer understanding of which specific fields need to be indexed. This exercise will immediately reduce your index size, lowering memory, CPU, and disk overhead.

For best performance we recommend the following:

- Select an Atlas cluster tier that has sufficient RAM to maintain the search index in memory. To avoid resource contention with the database, the search index should occupy no more than 40% of available RAM.

- Maintain sufficient storage space on disk to support at least 2x the index size. The additional capacity allows the existing index to continue to serve queries while any modifications to the index definition are performed in the background. When the new index is built, the existing index is automatically dropped.

- Ensure sufficient CPU capacity is provisioned to both service queries and minimize synchronization lag between the database and search index.

You can monitor a full range of metrics from the Atlas metrics tab, including CPU, memory, and disk consumption, index size and the number of unique fields indexed, along with replication latency between the database and Atlas Search index.

MongoDB read preferences offer a further potential performance optimization. By utilizing read preferences in your driver you can control how search queries are routed across your Atlas cluster. Selecting a secondary read preference will send search queries to a replica. This isolates those search queries from database operations that are hitting the primary node, eliminating potential resource contention.

MongoDB Atlas' native sharding enables you to horizontally scale-out your workloads across multiple instances. If you are indexing very large collections, sharding your cluster may be required. Lucene has a hard limit of 2 billion documents, and so sharding will be

necessary if you expect to exceed this limit. We would not recommend waiting until the 2 billion hard limit is reached. Best practice would be to anticipate sharding when you reach 500 million documents. Sharding early will ensure your cluster has sufficient capacity to redistribute and balance your data while still serving your application, avoiding downtime.

Go Live, Go Retire

If your production application is already using Atlas as its database, then you are ready to promote Atlas Search to serve your search queries. Build your Atlas Search index using the index definitions determined in Step 2, update your code with the queries developed and tuned in Steps 3 and 4, and update your application's connection string.

We recommend running both your existing search solution and Atlas Search in parallel for several project sprints. Use A/B testing to compare search results and secure sign off through User Acceptance Testing.

If your application data is not yet in Atlas, then you will need to migrate the latest data from your existing database to Atlas, keeping both systems in sync for parallel operation. The migration process will depend on where your data is currently stored:

- If you are self-managing MongoDB in your own facilities or in the cloud, use the Atlas Live Migration service. This will take a copy of your data and load it into your Atlas cluster and then synchronizes all data changes between the source and target systems. When the two databases are in sync, change the connection string in your application and cut over.

- If your application data is not in MongoDB, repeat the dump and import process from Step 2. Alternatively, use an ETL process that applies any required data remodeling logic. For a live migration to your new Atlas environment, you will need to stream all data changes from your source system after the initial load. Consider using the MongoDB Kafka Connector to sink data changes from your migration data pipeline to the Atlas database.

When Atlas Search is meeting your application requirements, you can go ahead and decommission (retire) your redundant search engine and sync mechanism!

Monitoring and Management

It is often said that development of relevance-based search has a beginning but no end. As application functionality evolves, or as you start to serve new users and markets, you must continually tune relevance to ensure users can find what they are searching for.

As with any search technology you should regularly review search results to find opportunities to increase relevance by modifying analyzers, field mappings, scoring rules, and creating new synonyms. Run `explain` whenever you modify your indexes and queries.

As your application grows, it may become necessary to scale-up your deployment to higher Atlas cluster tiers. You should monitor system consumption from the Atlas metrics tab to ensure you have sufficient resources to maintain application SLAs. You can configure alerts in Atlas to fire notifications when any of the Atlas Search metrics exceeds predetermined thresholds. You should also consider configuring cluster auto-scaling to dynamically increase and decrease compute and memory capacity in response to application load.

Customers Making the Switch to Atlas Search

The 5-step methodology presented in this guide is based on working with customers across all industry sectors and geographies. The following highlights several customers who have made the switch by replacing bolt on search engines with Atlas Search.

Product Catalog Search in Automotive: Scaling to 160 Countries

One of the world's largest auto manufacturers uses MongoDB Atlas to unify its after-sales parts product catalog and content management system storing manuals and maintenance procedures.

The MongoDB Atlas database was selected to store the data set comprising millions of documents extracted from 40 different backend operational systems. The company originally bolted on Elasticsearch to Atlas to power information discovery across the catalog and CMS, but reevaluated that decision with the release of Atlas Search.

After a Proof of Concept demonstrated Atlas Search was able to meet application requirements, the company's architecture team took the decision to replace Elasticsearch.

By making the switch, the company has reduced data duplication and simplified its architecture by eliminating the fragile sync mechanism and transformation logic. As a result, its developers can build new features faster, for example adding the ability for users to search by specific vehicle error conditions and immediately find the appropriate parts and installation manuals needed to remedy the problem. At the same time, operational overhead has been cut.

Atlas Search is used by the company's B2B marketplace serving hundreds of thousands of users across 160 countries and in 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.

Customer 360 in Insurance: 90% Faster Release Velocity

As one of Europe's largest insurance companies, Helvetia is increasing its competitiveness by modernizing with cloud-native services. The company turned to the MongoDB developer data platform to deliver a single, 360-degree real time customer view. The single view unifyies siloed account history, quotation, policy and service data, exposing it to users via fast and reliable full-text search.

The engineering team initially built out its single view platform with Elasticsearch bolted onto the Atlas database. While Elasticsearch was well regarded by the engineering team, its complex integration with the database was slowing down speed of innovation. It was failing to provide the most recent customer data, impacting user experience and customer satisfaction. The engineering team had to grapple with:

1. Complex data synchronization pipelines built with Apache Kafka. Sync failures and schema changes both demanded constant Elasticsearch reindexing, adding to development overhead and impacting customer experience with stale or incomplete search results.
2. Working with multiple query languages and drivers to query the database and search indexes, slowing developers down.
3. DevOps overhead in managing and monitoring three separate systems. It could take hours to diagnose and remediate application issues.

As a result of these issues, the company's engineering leadership took the decision to switch to Atlas Search. To start the migration, the company's engineers first reviewed the Elasticsearch index definitions, and mapped them to Atlas Search. With a couple of API calls, the configured index was deployed within MongoDB Atlas, and data was automatically synchronized between the database and search index. After rewriting their queries to use the familiar MongoDB Query API, the company's developers were able to verify the performance and reliability of search results, ensuring no compromise to application functionality..

This deep integration between the MongoDB Atlas database and Atlas Search enables the company's developers to dramatically compress their release cycles.

"Coordinating the inter-team dependencies and executing the reindex process meant it would take us 3 to 5 days to roll out new application features with Elasticsearch. Because Atlas Search is embedded right alongside the database, everything is automated for us. Now we've got feature releases down to 3 hours, representing a time saving of over 90%."

Johannes Mangold, Lead Solution Architect, Helvetia

Learn more by reading the Helvetia case study.

In-Application Search for Retail Banking

Sometimes development teams start out on a new project with the assumption that they need to bolt-on a search engine to their database, but then discover Atlas Search as an alternative approach.

An example is Current, 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.

"We avoid the heartache of developing the Elasticsearch data sync pipeline, which would require us to revisit the code periodically as requirements change, each time with less

context than before. By translating well-understood Lucene patterns into the MongoDB aggregation pipeline, we were able to sever the reliance on another technology while simplifying queries." TREVOR MARSHALL | CTO, CURRENT

Getting Started with Atlas Search

Elasticsearch and Solr are popular and proven search engines, but introduce significant complexity when they are bolted on to a database to power application search. MongoDB Atlas Search provides a faster and easier way to deliver rich, relevance-search based features for your applications.

This guide has presented a repeatable 5-step methodology to replacing Elasticsearch and Solr. The best way to get started with Atlas Search is to try it out on a free cluster today.

MongoDB's Professional Services team is available as a trusted delivery partner to help you and your teams bring applications built with Atlas Search to life. Whether you need support with one specific part of the methodology outlined in this paper, or you need guidance throughout the entire journey, we're here to provide the right type of services you need to be successful with your search project.

Flex Consulting is available directly from the Atlas UI, providing your teams access to consulting engineers for short, remote sessions to target specific technical hurdles. We can also stand up an interim development team to build on your behalf with our Jumpstart service. With Jumpstart, we can build new or migrate existing apps and bring them to production while empowering your teams with the skills they need to continuously evolve your new search applications in the future.

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.

US 866-237-8815 • INTL +1-650-440-4474 • info@mongodb.com.