How AWS and MongoDB collaborate to unlock the power of Apache Lucene

by Igor Alekseev | on 04 OCT 2021 | in Open Source, Thought Leadership | Permalink | Comments | Share

This post was written by Marcus Eagan, Senior Product Manager MongoDB Atlas Search, MongoDB; and Matt Asay, Head of Evangelism, MongoDB.

Search is essential to delivering exceptional customer experiences, whether those customers are individuals scouring Amazon.com for a new webcam, or enterprises building search into their own applications. For enterprises, there are a number of search alternatives to choose from, but the open source project Apache Lucene has become integral to a wide variety of applications since its introduction in 1999. Hence, when Amazon wanted to upgrade its product search capabilities on Amazon.com, it turned to Lucene. Similarly, when MongoDB customers asked that full-text search be integrated into our distributed database, we also turned to Lucene.

Lucene, however, isn't merely great code. Behind that code is a vibrant community. In this blog post, we discuss some of the ways MongoDB and Amazon's search teams have collaborated on Lucene to tailor it for our own needs while simultaneously improving the project for all.

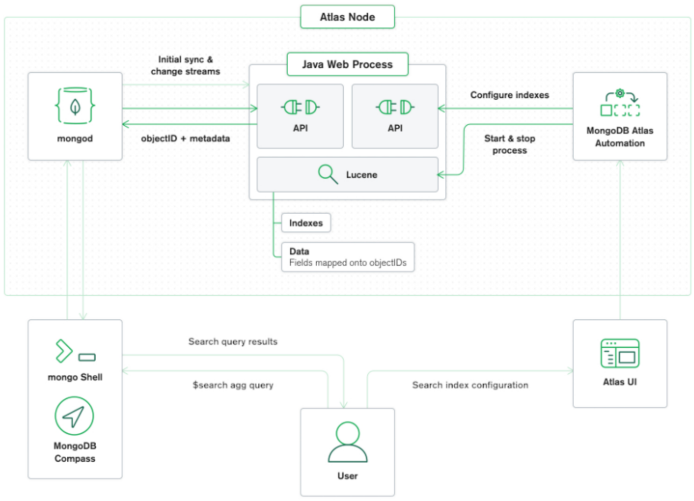
How MongoDB uses Lucene to power Atlas Search

Amazon and MongoDB both use Lucene every day, and the most important use case is no doubt application search, in which the engine is primarily used by humans. At Amazon, the product catalog search millions of users have come to know is powered by Lucene, but that's a relatively recent phenomenon, one explained in the "What Amazon gets by giving back to Apache Lucene" blog post. Lucene is also what enables MongoDB customers to design rich search experiences in their applications on top of data stored in their Atlas databases.

When the MongoDB team first considered building a search product, we evaluated many options, such as starting from scratch, or building on Elasticsearch or with Apache Solr. Ultimately, building with Apache Lucene made sense, given that it provides the foundation for both Elasticsearch and Solr. But more than merely using the project, we also determined that it made sense to contribute actively to Lucene to improve it for our specific use case, in a way that would generally improve Lucene for all users.

Our goal is to make it easy for developers building on MongoDB to add full-text search functionality to their applications, so we designed it to be fully managed and integrated. A Lucene node is embedded with every Atlas cluster and handles search indexing and querying so users don't have to worry about provisioning and running that infrastructure themselves.

Following is an overview the Atlas Search architecture:



This design also allows our engineering teams to implement new features and capabilities in Lucene and pass them onto our customers in the future. So far, these changes only helped MongoDB and our customers. But with Amazon's help, we were able to go farther.

Different search use cases, common search goals

Earlier this year, MongoDB product and engineering teams were in discussions about how to build a highly requested feature for Atlas Search: facets. We considered several approaches for implementing this functionality into the product, ultimately landing on one that would use the Lucene facet API—a decision influenced by an email exchange on the Lucene mailing list between Greg Miller of Amazon and Alexander Lukyanchikov of MongoDB. Open source community is not merely about contributing code, but also about sharing experience to guide code decisions, as Miller did in his message:

> Hi Alex-
>
> Amazon's product search engine is built on top of Lucene, which is a fairly large-scale application (w.r.t. both index size, traffic and use-case complexity). We have found taxonomy-based faceting to work well for us generally, and haven't needed to do much to optimize beyond what's already there. As you can imagine, with Amazon's catalog being quite broad, we have a large number of unique facets available for customers to use, which means a single facet-field storing all dimensions can have high cardinality (as is the case by default with taxonomy facets). This is an area where we have experimented a little bit (e.g., "sharding" facets into separate fields to lower cardinality of counting at query-time), but we tend to find Lucene works well "as-is" for the most part in this [space]. The last bit I'll mention here is that, for fields that are numeric and low-cardinality in nature, [LUCENE-7927] added the ability to count these cases a bit more efficiently than trying to apply a taxonomy-based approach.
>
> Happy faceting!

Miller's assistance was exciting for several reasons:

Resources

Open Source at AWS
Projects on GitHub

Follow

AWS Open Source
AWS Cloud
Facebook
LinkedIn
Twitch
Open Source RSS
Email Updates

AWS Events
Discover the latest AWS events in your region
Learn more »

- It told us how Amazon uses Lucene in production and praised its robustness, which derisked our decision to implement it in Atlas Search;
- It shared some of the recent history of the work that Amazon has done to improve the Lucene facet API; and
- It encouraged us to use Lucene facets.

Although we work for different companies of different sizes, we share a common goal: to enable customers to quickly and easily search for the products and information they need. Although the Atlas Search team has done a lot of sophisticated and interesting work on the search product at MongoDB in the past two years, we gratefully acknowledge that much of the heavy lifting for the Lucene facets API was largely done by Amazon engineers before we even started building our product. When Amazon's Miller shared his Lucene insights with us, it demonstrated just how powerful open source community can be.

Combining forces to improve open source communities

Encouraged by Amazon's example in the Lucene community, we've also tried to give back. Open source collaboration is not always a happy fairy tale or comfortable endeavor. Sometimes we have to make hard decisions or do less fun work to improve the project's long-term viability. One example is MongoDB's efforts to replace *master* and *slave* terminologies within the Apache Lucene-Solr project because the terms are inaccurate and offensive. In this we've been joined by other contributors.

Whenever you change that many files in a single pull request for any project, however, risk is introduced. Although a Lucene community member expressed some objection to the change, the community nearly unanimously supported it and we moved on to the next technical debt repayment in the Lucene project with the encouragement of the Amazon team.

Prior to 2021, there was no standard for naming test classes in Apache Lucene. We improved the codebase in that respect with LUCENE-8626. In it, you can see the power of collaboration by users at different companies all around the world.

On the mailing list, Mike McCandless from Amazon had this to say about the effort, specifically a detail around choosing between the suffix or prefix:

> I had thought this issue [of suffix versus prefix] was really a butter side up / Sneetches / Law of Triviality / bike shedding sort of situation, but it is not 🙂
>
> This justification makes sense to me, so I think suffix is indeed better than prefix, as long as we can 1) do the renaming at once, and 2) enforce that consistent naming going forwards.
>
> +1 to rename all of Lucene's tests to use Test.java suffix.

Although McCandless eventually even softened his "at once" position, his support for the improvement was valuable to the momentum of the project because of his previous contributions to Lucene.

The inconsistencies in naming test classes can be comical at first, yet they cause real problems. We must constantly strive to simplify every aspect of a codebase, especially one as complex as Lucene. Standardization has the benefit of enabling users to scan file names faster, add tests that adhere to convention, and simplify logic in build scripts.

To minimize the risk inherent in changing so many files at once, we broke up the effort into three pull requests spanning a few months—and we still had a mistake in a file or two at some point. An outside observer might think: Wasn't this just a find and replace? Yes, but as the author of the change, we still needed to review each of the test files and check for any logic that may have been impacted. And so does the reviewer.

Once again, we were supported by Amazon, this time by another engineer from the Amazon Search team, Mike Sokolov. I'm sure this wasn't the most fun review for him to conduct, but we know he did it right. Buried in the 88 files of monotonous changes, there was an acronymic variable impacted by the change. He noticed the change and made a lighthearted comment in the pull request in his review. We were and are grateful for his time.

Architecting Lucene at scale

There's an important detail to our Lucene implementation in Atlas Search that makes us different from Amazon. Tens of thousands of new clusters are created on MongoDB Atlas every week, and the majority of these are shared clusters that are hosted on multi-tenant architecture. In other words, we need the ability to quickly spin up managed Lucene clusters in infrastructure that is shared across customers.

If we aren't careful, some of those customers could hurt themselves and others by adding millions of fields to their search index definitions and stressing the system. This could result in a catastrophic impact to thousands of Atlas customers and their application end users.

Historically, there had been a way to prevent this problem from happening in Lucene—a field limits API available in the IndexWriter, the class that transforms structured data into Lucene structured data to enable fast searching and a host of other features. However, this feature was deprecated in Lucene on July 17, 2019.

In January 2021, the Lead Software Engineer of Atlas Search opened his first Lucene JIRA ticket, LUCENE-9680. In it, he proposed that we undeprecate the field limit API in order to support our multi-tenant architecture. The default mode is still to allow as many Lucene fields as one would like, but for our shared infrastructure architecture, we leverage this API to prevent mapping explosions for our users. Both McCandless and Sokolov from Amazon participated in the discussion, and Sokolov reviewed the pull request.

What's next for MongoDB and Amazon Search engineers

The future of the open source collaboration between MongoDB's Atlas Search team and Amazon's Search team will go far beyond improving facets implementations, the consistency of class names, and supporting more stable multi-tenancy deployments. This is only the beginning of our partnership. Many other search teams contribute to open source Lucene, and there's a lot more work to do. If you walk away from this blog post feeling, "all this sounds really easy," we implore you to chip in.

Maintaining an open source project may not always be relevant for the products of our companies, but sometimes we must do work for the health of the project so that all of us can continue to offer solutions that millions of people rely on everyday. In a future blog post, we hope to discuss a multi-year project started initially by the Amazon team that we feel will cement Lucene as the bleeding edge of open source search.

In the meantime, you can try MongoDB Atlas Search today for free on AWS to unlock the power of Lucene for your own applications. Or if you need a dedicated Atlas cluster because you have volume of inserts or have high performance requirements, enable the provisioned IOPS feature to complement Lucene indexing.

If you see opportunities to add features that will benefit the greater Lucene community, let us know so we can prioritize those efforts. In open source we all benefit more than we contribute. In the case of Lucene, both Amazon and MongoDB benefit much, so it's a privilege that we are also able to contribute back, especially when we get to do it together.

Matt Asay

Matt Asay leads evangelism at MongoDB. Previously. Asay was a Principal at Amazon Web Services and Head of Developer Ecosystem for Adobe. Prior to Adobe, Asay held a range of roles at open source startups, including MongoDB, Nodeable, Strobe, Canonical, and Alfresco. Asay is an emeritus board member of the Open Source Initiative (OSI) and holds a J.D. from Stanford, where he focused on open source and other IP licensing issues.

Marcus Eagan

Marcus Eagan is the Senior Product Manager for Atlas Search at MongoDB, an open source contributor to a variety of projects, and an angel investor to a number of startups building the next generation of infrastructure. Prior to MongoDB, Marcus led Developer Tools and Evangelism at Lucidworks, led a few software engineering teams at Ford Motor Company, and built an IoT Security startup on MongoDB.

The content and opinions in this post are those of the third-party author and AWS is not responsible for the content or accuracy of this post.

Igor Alekseev

Igor Alekseev is a Senior Partner Solution Architect at AWS in Data and Analytics domain. In his role Igor is working with strategic partners helping them build complex, AWS-optimized architectures. Prior joining AWS, as a Data/Solution Architect he implemented many projects in Big Data domain, including several data lakes in Hadoop ecosystem. As a Data Engineer he was involved in applying AI/ML to fraud detection and office automation. Igor's projects were in variety of industries including communications, finance, public safety, manufacturing, and healthcare. Earlier, Igor worked as full stack software engineer/tech lead.

View Comments

AWS Podcast
Subscribe for weekly AWS news and interviews
Learn more »

AWS Partner Network
Find an APN member to support your cloud business needs
Learn more »

AWS Training & Certifications
Free digital courses to help you develop your skills
Learn more »