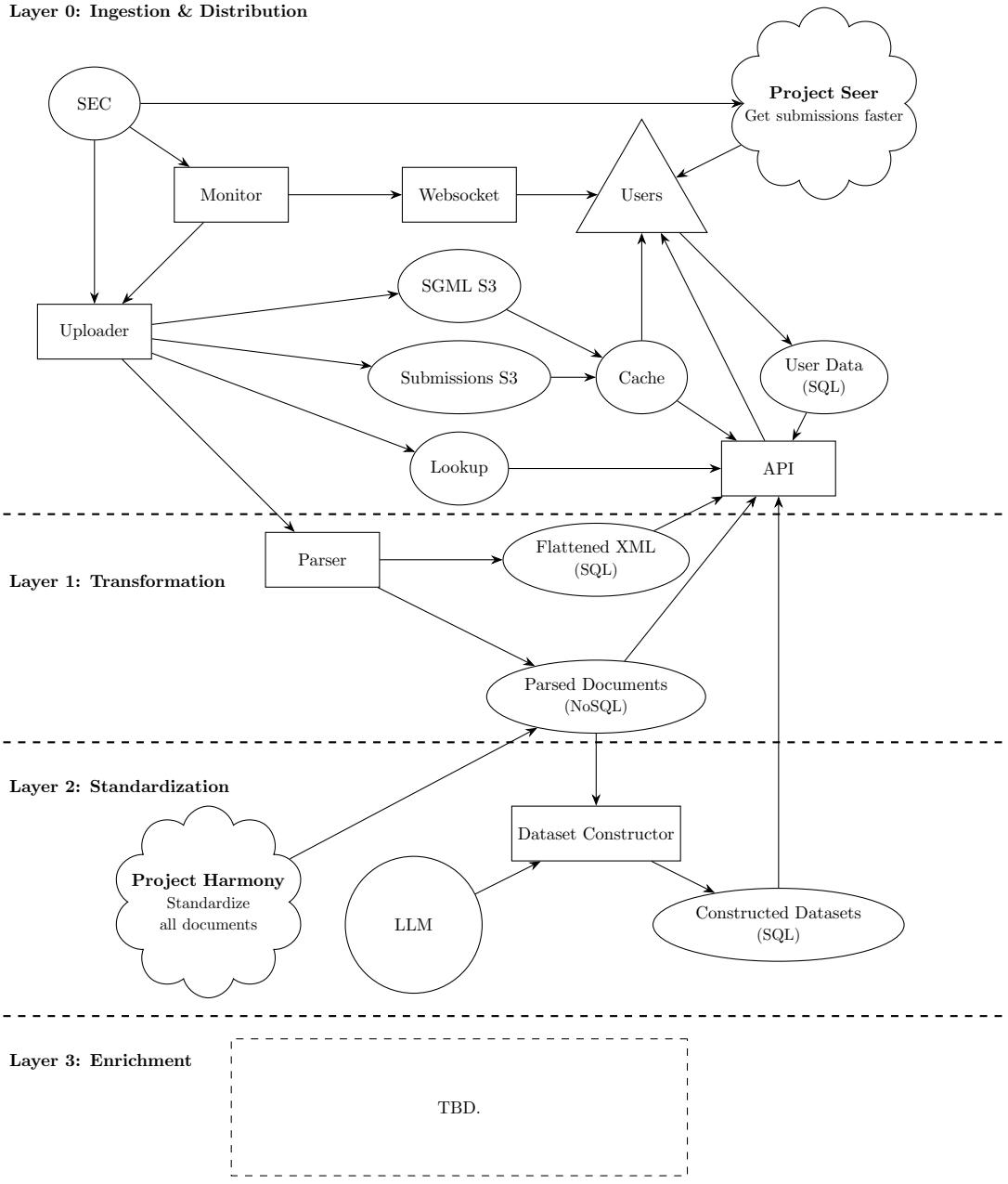
Datamule

John Friedman

May 30, 2025

Working with Securities and Exchange Commission data at scale.

1 System Architecture



2 Introduction

Contents

1 System Architecture **1**

2 Introduction **2**
2.1 Goal 2
2.2 Open Source Dependencies 2

3 Technical Specifications **2**
3.1 Layer 0: Ingestion & Distribution 2
3.2 Layer 1: Transformation 6
3.3 Layer 2: Standardization 7
3.4 Layer 3: Enrichment 8
3.5 Implementation Notes 9

4 Conclusion **9**
4.1 Timeline 9
4.2 Funding 9
4.3 Solicitation 9
4.4 Acknowledgements 9

2.1 Goal

The goal is to create comprehensive SEC data infrastructure that makes it easy for developers to build LLM applications and AI agents on top of financial regulatory data.

2.2 Open Source Dependencies

I plan to rely heavily on the open source packages that I have developed since Fall 2024. They are licensed under the MIT license which allows for both commercial and non-commercial use with minimal restrictions.

- `datamule-python`, `https://github.com/john-friedman/datamule-python`, A Python package for working with SEC filings at scale.
- `doc2dict`, `https://github.com/john-friedman/doc2dict`, Quickly convert documents into dictionaries while preserving hierarchy.
- `secsgml`, `https://github.com/john-friedman/secsgml`, Parse SEC Standardized Generalized Markup Language (SGML) files.
- `txt2dataset`, `https://github.com/john-friedman/txt2dataset/`, Convert text into datasets using LLM structured output.

3 Technical Specifications

3.1 Layer 0: Ingestion & Distribution

Monitor

Virtual machines running `monitor_submissions` from `datamule-python` that emits new submissions' location and metadata to the Websocket and Uploader.

- `monitor_submissions` polls the RSS endpoint and EFTS endpoint
- One RSS GET retrieves less than 5kb of data.
- One EFTS GET retrieves less than 10kb of data.
- RSS is the fastest of the two, but misses some submissions. EFTS is slower, but complete.
- EFTS is not sorted, and the minimum interval is one day. To get new data, a full cycle, or between 1 and 200 GETs must be run.

I plan to use a two VM approach using multi-attach EBS for storage. One VM will poll the RSS feed every 200ms, while the other runs an EFTS cycle every minute at 10 requests per second. The VMs will keep a deque of the last 50,000 submissions and write them to separate files on EBS.

Cost Breakdown: $6/month. One AWS t4g.nano costs $3/month on demand. EBS price is $0.09/gb, so costs will be minimal.

Websocket

Virtual machine that users can connect to for receiving real-time updates when new submissions are published. There are on average 75,000 new submissions per month, and the metadata for each submission can be reduced to about 100 bytes. I plan to use Go, as it seems Go will allow 4-5x as many connections as Node.js without issue.

1. Check every 10ms if the RSS or EFTS file has changed (IOPS cost should be minimal).
2. If changed, load file, compare against internal deque of 60,000 (padding for safety).
3. If a new submission is found, emit signal.

Cost Breakdown: $3/month. One AWS t4g.nano costs $3/month on demand. Egress costs $.09/gb, so every additional persistent monthly connection costs $.01.

User Cost: $5/month. Charging for usage would add too much overhead. Not charging would lead to abuse. I plan to add a $5/month subscription option to `https://datamule.xyz` that allows users to maintain up to 5 persistent connections. If people start using this, I will drop costs.

Note: An afforadable websocket has been one of the most asked for features since I started coding `datamule-python`.

Uploader

Virtual machine that:

1. Receives Submission url from Monitor.
2. GETs Submission from the SEC. (About 40gb/month, biggest individual submission is under 250mb).
3. Compresses and uploads raw SGML to SGML S3.
4. Parses SGML into constituent files, compresses and tars it, before uploading to Submissions S3.

Since asynchronous requests are computation light, I think I can fit this into a t4g.nano paying careful attention to memory considerations. However, that would be a bit risky - so I plan to go with the t4g.micro which has 1gib of memory.

Pipeline:

1. Asynchronous streaming SEC SGML to EBS
2. Asynchronous uploads to SGML S3
3. Parsing SGML sequentially and writing to EBS
4. Renaming parsed file when done to prevent write coordination issues.
5. Asynchronous uploads to Submissions S3
6. File Cleanup (Logic here should be changed upon addition of Parser)

I plan to use the `Submission` class within `datamule-python` which uses the SGML processor from `secsgml` to parse the SGML files quickly. Requests to the SEC will utilize an average rate limit of 300 per minute, with allowed bursts of 10requests/second. The SEC allows 5requests/second for long durations, but is fine with 10/s as long as it is for a duration of a few minutes.

Cost Breakdown: $6/month. One AWS t4g.micro costs $6/month on demand. Egress costs $.09/gb, so another $3-$4 per month.

SGML S3

Wasabi S3 bucket containing raw SGML files from the SEC. Files above a certain threshold are compressed using zstandard.

Cost Breakdown: $20/month. Wasabi costs $7/tb.

Submissions S3

Wasabi S3 bucket containing SGML files split into their constituent files combined into a tar file. constituent files above 1kb are compressed using zstandard. the first file is a metadata.json from the Submission header in the original SGML as well as the list of documents, their type, sequence, and filename.

Note that the metadata.json's document file is modified to include byte locations of the files to enable HTTP range requests.

This format is designed to enable both fast downloads of the entire submission as well as individual documents within the submission.

Intended use-case for specific documents:

1. User requests specific document within a submission.
2. Conditional GET for first 16kb is sent
3. If submission is under 16kb in size, for instance most form 4s, the entire submission is returned, and all but the document discarded.
4. If submission is over 16kb size, read the metadata.json for the specific documents bytes locations
5. GET requests using HTTP ranges for each document

Cost Breakdown: $30/month. Wasabi costs $7/tb.

Cache

S3 buckets are proxied behind Cloudflare with caching enabled to eliminate egress fees.

Endpoints:

- SGML: `https://library.datamule.xyz/original/nc/`
- Tar: `https://library.datamule.xyz/tar/nc/`

Submissions are stored in accession number with no dashes format. See: `https://library.datamule.xyz/original/nc/000147793223000200.sgml.zst`.

This means that submissions can be accessed for free. The API only charges for lookup costs, e.g. select all 10-Ks filed in 2025.

Note: Some of the historical correspondence files, were discarded during the upload process due to naming collisions. This will be addressed in the future.

Lookup

Lookup is currently a Cloudflare D1 database with only select values such as cik, type, and date. To save on costs earlier in the project, I only stored one cik per accession. This was a bad idea as it turns out that was important. I plan to rewrite Lookup soon.

- Rough estimates put the size of the metadata around 100gb (20,000,000 submissions)
- Existing Lookup is about 500mb, 1gb with indices.
- A 20kb 10-K metadata file converts to 5kb (documents) csv, and 1kb (submission metadata) json file.
- Ballpark: 20gb of relational data and 50gb of nonrelational data.

Submission metadata has two sets of information:

1. Submission specific metadata such as acceptance-datetime. (150bytes)
2. Company/individual specific metadata which changes rarely across submissions (800bytes).

By storing only changes in company/individual metadata, we reduce the size of nonrelational data, probably by an order of magnitude.

So, now we're at: 20gb document relational data, 200mb of submission specific relational metadata, 5gb of company/individual specific nonrelational data.

Relational: AWS Aurora v2 with scale to zero is tempting, but the 15s cold start will make user experience unpleasant. Using db.t3.micro for RDS should cost $10-$15 per month.

Nonrelational: Looks like db.t4g.medium for the nonrelational database with proper indices should be fine. $54/month.

Note: It may be possible to standardize the remaining nonrelational data. Will look into it.

Cost Breakdown: $70/month. db.t3.micro + db.t4g.medium.

User Cost: $1/100,000 results. This seems high enough to be above cost of provision, and is currently what I charge.

Note: The new architecture of Submissions S3 will be integrated into `datamule-python` to enable free users to download submissions extremely fast, but without the historical depth (Starts in 2001, not 1995) and without complex queries.

Users

Users can interact with the API by creating an account at `https://datamule.xyz/dashboard` (Google OAuth) and loading up their balance (Stripe). The easiest way to interact with the API is by using the python package, `datamule-python`: `https://github.com/john-friedman/datamule-python`.

User Data

Stores minimal user data (to save costs) in a Cloudflare D1 database.

API

Cloudflare workers connect user requests to databases. As I expand access to SQL/NoSQL databases, some work should be done here to prevent SQL injection attacks.

Project Seer

Delivers SEC metadata approximately 30 seconds faster and submissions 2 seconds faster than commercially available alternatives by taking advantage of how information propagates across the SEC website. I plan to test if this works, if it does, I'll probably open-source the code.

Cost Breakdown: $3/entity. One AWS t4g.nano instance is about $3 per month, less if reserved.

3.2 Layer 1: Transformation

Parser

Virtual machine that parses documents using `doc2dict`. PDF, HTML, etc filetypes are converted into nested dictionaries and uploaded to a non-relational database. XML files are flattened into multiple `Tables` and stored in a relational database.

- Read submissions tar file from EBS
- Parses files
- Update NoSQL and SQL databases

Cost Breakdown: $3/month. One AWS t4g.nano instance is about $3 per month.

Flattened XML DB

Ballpark, there will be 3tb of data to store split across 200+ tables. I think Aurora Serverless V2 is a good option here. It scales to 0.5 ACUs($43/month), and each ACU-hour costs $0.12. So a user query with decent indices costs ballpark $0.00007. If a user requests the entire Form 3,4,5 tables (a couple gb) should cost me about $0.18 in egress and maybe $.07 in ACUs.

I can further reduce costs if I setup an additional stage to handle egress compression.

Cost Breakdown: $350/month. $300 in storage, $50 for Aurora baseline, and $.09gb egress.

User Cost: variable. 10x markup on egress, and ACUs should be enough to insulate me from unexpected costs while being extremely cheap.

Parsed Documents DB

Ballpark, the SEC has 10tb of document data. After applying `doc2dict`, we should have about 1tb of data in dictionary representation. For instance all 10-K root forms converted to dictionary take up about 20gb.

I think AWS Opensearch serverless or AWS DocumentDB autoscaling is probably the best bet here. Should be possible to get under $1,000 per month in costs.

User Cost: TBD.

3.3 Layer 2: Standardization

Project Harmony

Goal:

1. Standardize headers within document type and company
2. Standardize headers across companies and document type
3. Convert tables within documents to a standardized form with contextual metadata.

Standardize tables within documents is straightforward. `doc2dict` supports basic table cleaning, but the result, and text above and below the table need to be fed into an LLM for proper context. I need to do the cost estimates, but this should be fairly cheap to apply across the entire SEC corpus. Probably under $500.

Standardizing headers within document type and company can probably be done minimal ML - maybe clustering. Basically, we use the headers from multiple documents of the same type, exploit the difference in how they are recorded and use that to fix inaccuracies in the nesting algorithm. The probable result is some (slower) parser that does a lot of string comparisons. This has the added benefit of making older SEC documents that lack the information needed for nesting, nestable. For example, 10-Ks in .txt form have detectable headers (text on its own line, possibly with two line breaks on each side), but lack information on which headers should be nested under other headers.

Note: I've already begun standardizing SEC "standardized sections". For example, "ITEM 1.A RISK FACTORS" and "Item 1A" in 10-K document types are already mapped as part of ***datamule-python****. There are hundreds of forms and even more document types. Unfortunately, the SEC lacks documentation for many smaller submission types. The easiest way to solve this, is by using a de facto not de jure approach where I construct a list of all headers for a document type, and then determine the de facto headers, before creating mapping dicts for each one. A list of submission types that I manually standardized over two days is visible here:* ***https://john-friedman.github.io/datamule-python/datamule-python/sec/submission_types/***

Standardizing headers across companies and document type (e.g. 10-K vs S-1) is a more challenging problem. For example, should 'Sales in East Asia' be considered the same as 'Sales in Japan and Korea'. Not sure how to think about it, and what information is useful to create. I probably should get some real world use-cases to construct benchmarks. Should be fairly solvable.

Dataset Constructor

Converts sections from the parsed NoSQL database into datasets using `txt2dataset`. For example, an auto-updating executive compensation dataset using item 5.02 in 8-Ks.

I plan to release the prompts and schema in an open source repository.

Cost Breakdown: Depends on the dataset. IIRC Executive Compensation dataset costs under $50 to construct.

User Cost: $10 per dataset. Convenience fee.

LLM

LLM selection is still under evaluation.

3.4 Layer 3: Enrichment

I would like to enrich the dataset.

- Increasing information density
- Connecting data across time, companies, and document types.

This is a long-term idea that I don't know how to approach currently.

Increasing Information Density

There is a lot of boilerplate language in SEC documents. If the useless information can be reduced, the data becomes much more information dense. This is useful for improving the output and reducing the costs of LLMs using the SEC corpus.

I'm not sure how to do this, and am actively looking for advice. My current idea is a mix of:

1. Train classifiers to strip useless language (preprocessing)
2. Use weaker LLMs to summarize sections within the dictionaries.

I'm not sure how 1. would work as I currently know little about classifiers. 2. is more straightforward.

- There are 50,000 tokens in a 10-K in dictionary representation.
- There are 200,000 10-K root forms, or about 10 billion tokens.
- Gemini 2.0 Flash Lite (cheaper model) costs $0.075 per million tokens of input, and $0.30 / million tokens of output
- Assuming 1/10 reduction in size (generous) that would cost $1,000.
- Ballpark, 10-K root form text is 2% of total text in the SEC corpus.
- Summarizing the entire corpus would therefore cost $500,000.

I'm making a lot of educated guesses, but basically this is expensive, even if I figure out a way to reduce costs by an order of magnitude.

Connecting data across time, companies, and document types.

I'm not sure how to think about this. Some possibilities are to precompute context, and store it as metadata within documents or construct a network structure that corresponds to documents relations with each other.

The issue here is that I need to know more about:

1. data structures and what modern infrastructure can support.
2. what are the bottlenecks for real world use cases.

I don't want to build something that is not useful.

Conclusion

If I want to build Layer 3: I need to raise funding, do a lot of reading, and talk to both experts and prospective users.

3.5 Implementation Notes

- The database implementations have been setup to conserve my limited resources. If the database responses are too slow for your use case, let me know and I will swap in a more powerful machine.
- Much of this project is powered by `doc2dict`. `doc2dict` is a package I introduced in May 2025. I expect it to significantly mature over the next few months.

4 Conclusion

4.1 Timeline

1. Layer 0: Funded. Should be done by July.
2. Layer 1, SQL DB: Funded, should be done by July.
3. Layer 1, NoSQL DB: Funded, but it's expensive. 10-K MVP should be done by July, will open to select users for feedback before proceeding due to cost.
4. Layer 2, Project Harmony: Funded. Should be done by end of August.
5. Layer 3: In thinking stage.

4.2 Funding

Existing Credits

- 25k AWS Activate
- 5k MongoDB
- 2k Google Cloud

Thank you George, and On Deck Founders for your support.

4.3 Solicitation

If you are/know someone who:

- has extensive SEC domain knowledge
- has research experience with LLMs
- or is building something in the interaction between SEC data and LLMs

I would very much like to talk with them. My email is `johnfriedman@datamule.xyz`.

4.4 Acknowledgements

Rui A, Victor Barres, Aquiba Benarroch, Alex Braid, Rick Davis, Yonatan Delelegn, Steven Glinert, Cornelius Graubner, Jeremy Griffith, Christian Grupp, Justin Guzman, Andrew Hoog, Emerson Hsieh, Morris Hsieh, Christian Kalin, Jesse Knight, Richard Li, Darren Liccardo, George Lin, Didier Lopes, James Maslek, Ryan Martin, Brian Mensink, Yosef Mihretie, Kapil Phadnis, Clint Rhea, Miquel Ruiz, Jared Rutner, Daniel Shaar, Benqing Shen, Derek Snow, Jake Sussman, Christine Tan, Daniel Taylor, Robert Tolan, Aleksi Tukiainen, Daniel Lovera, and many open source contributors. Thank you very much.