# NIXIESEARCH:


# Running Lucene over S3

## And why we are building our own serverless search engine

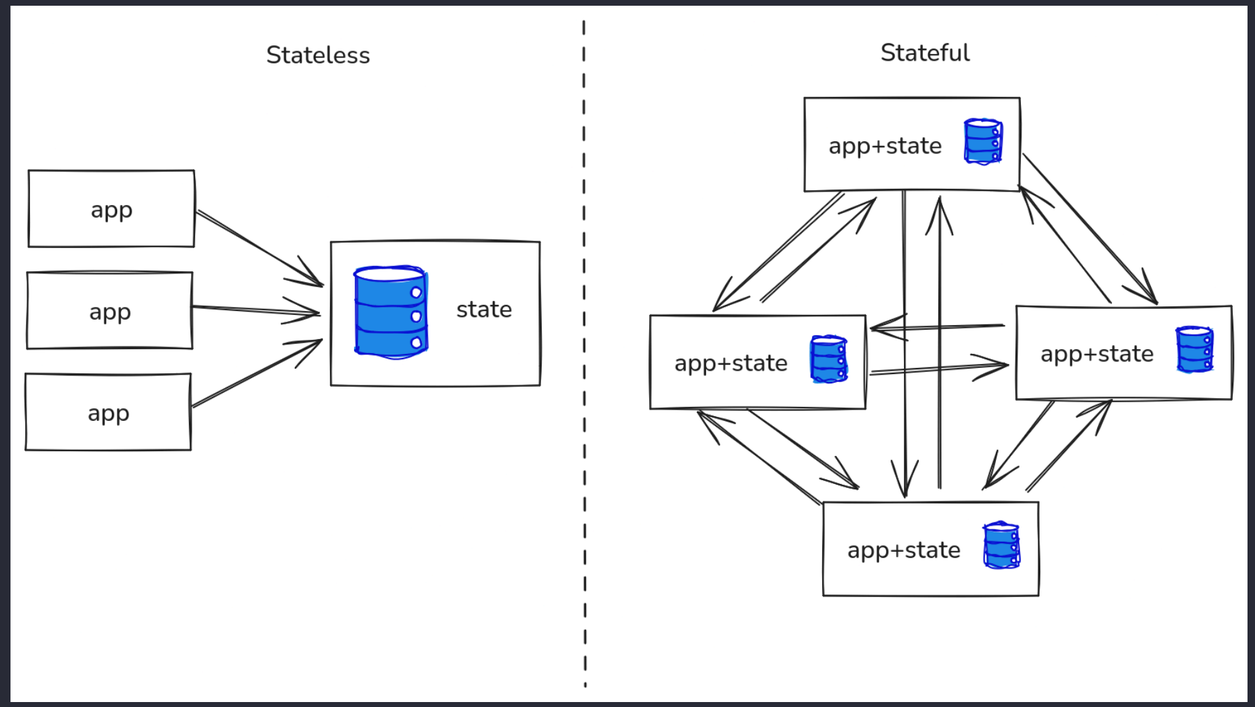
Haystack EU 2024 | Roman Grebennikov

# whoami

- PhD in CS, quant trading, credit scoring
- **Findify**: e-commerce search, personalization
- **Delivery Hero**: food search, LLMs
- **Opensource**: Metarank, lightgbm4j, flink-scala-api

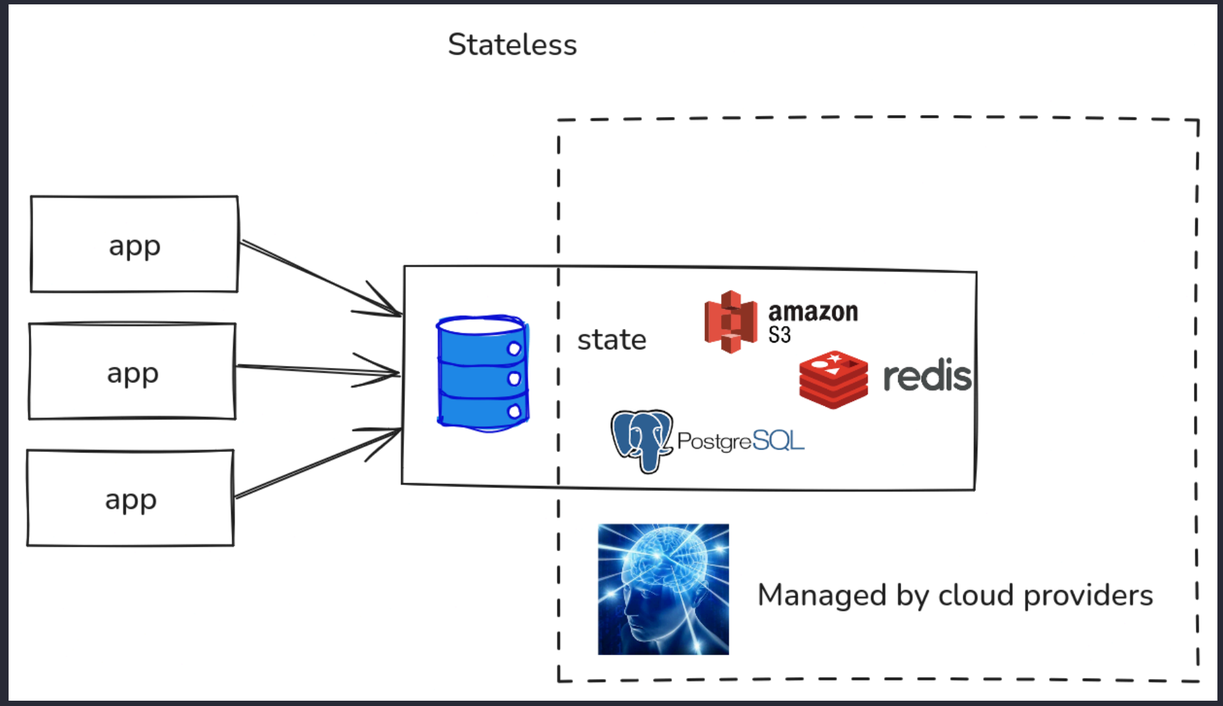

When you're on-call and try to guess what will break in your search today
ingestion rate too high
split brain on restart
disk is full
shard split caused OOM
cluster state is borked
rolling restart on a yellow cluster
prize wheel

# Curse of stateful apps

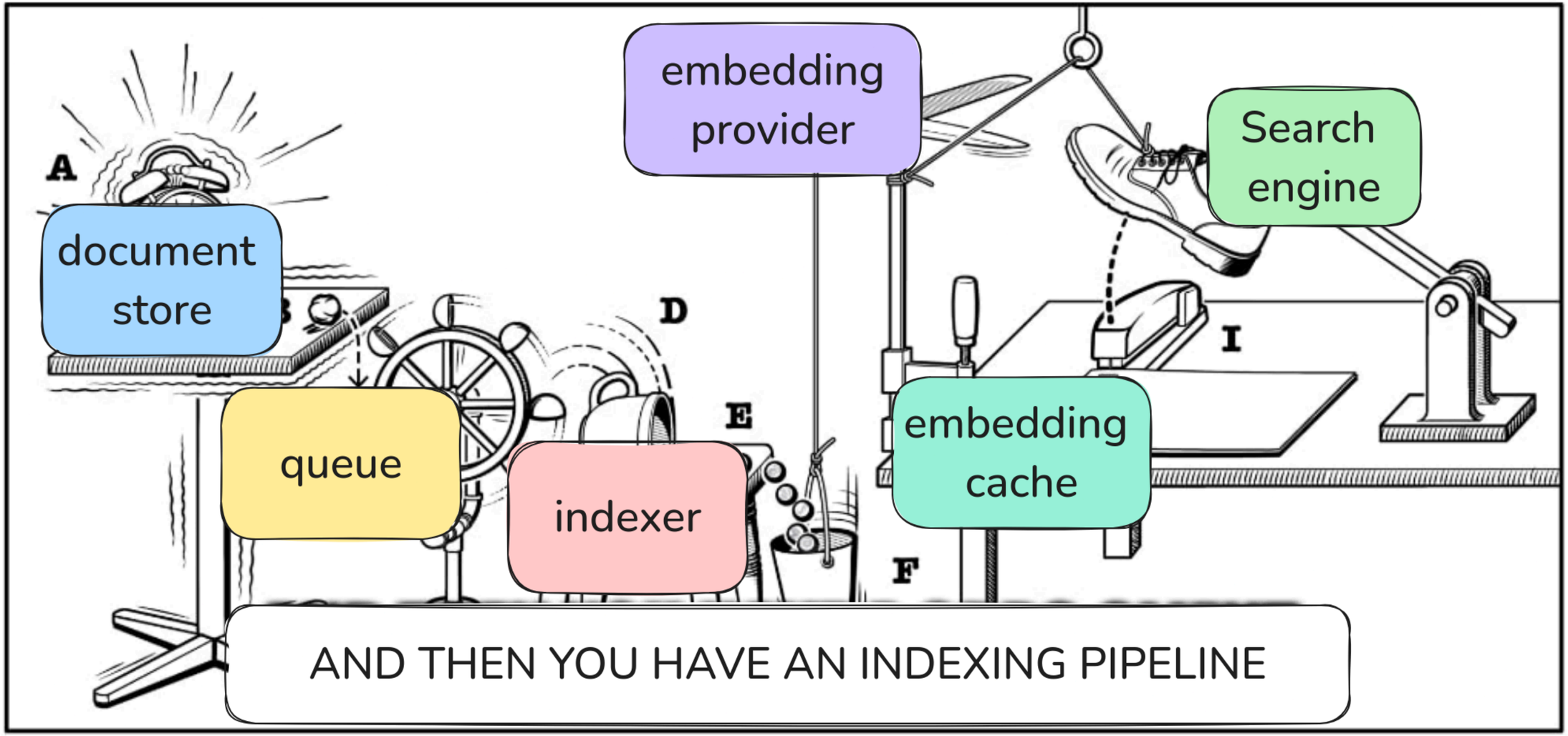



- Depends on per-node storage
- Up/Down-scaling requires rebalance
- "state inside" - good luck if borked

# Blessing of stateless apps




- Managed by cloud providers (and not by you)
- Easier to modify (data + metadata)


embedding provider
Search engine
document store
queue
indexer
embedding cache
AND THEN YOU HAVE AN INDEXING PIPELINE

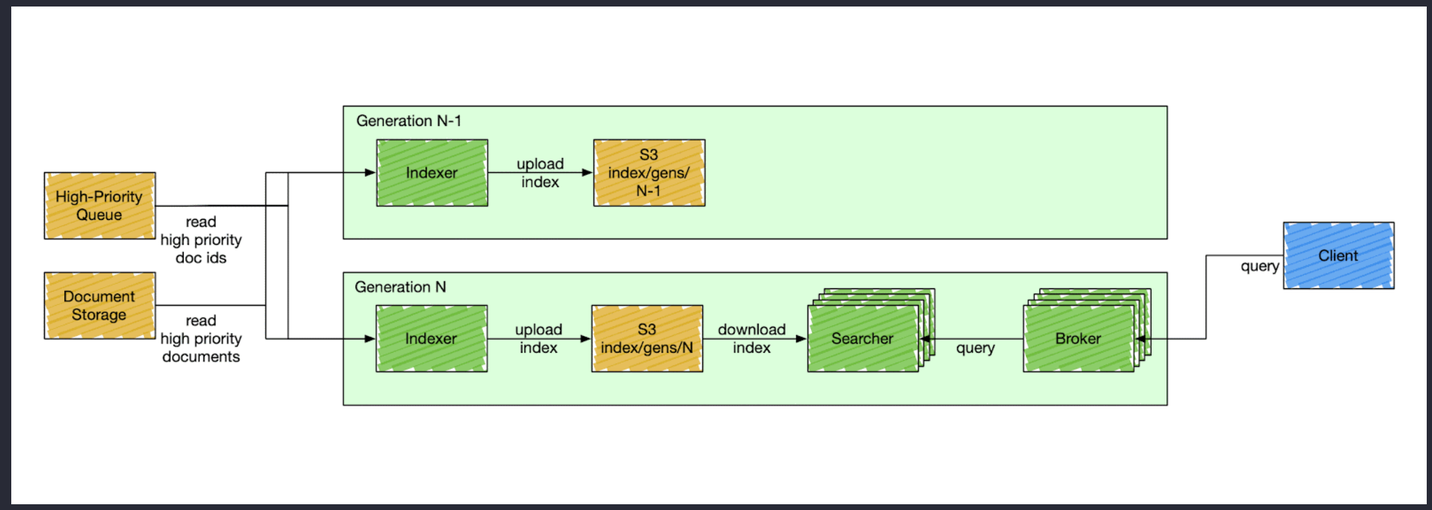
# Stateless search?

- Lucene world: NRTSearch, OpenSearch
- Non-Lucene: quickwit, turbopuffer


Main point: decouple search and storage

# Industry goes stateless

- **Uber**: Lucene: Uber's Search Platform Version Upgrade [1]
- **Doordash**: Introducing DoorDash's in-house search engine [2]
- **Amazon**: E-Commerce search at scale on Apache Lucene [3]




[1]: https://www.uber.com/en-NL/blog/lucene-version-upgrade/

[2]: https://careers.doordash.com/blog/introducing-doordashs-in-house-search-engine/

[3]: https://www.youtube.com/watch?v=EkkzSLstSAE

# open-source?

# Nixiesearch


- Started as a POC for Lucene over S3
- Went further: RAG, local inference, hybrid search

# Lucene and S3: 2007

kimchy.github.com / _posts / 2007-11-16-lucene-and-amazon-s3.textile

kimchy fix images — f08b1d4 · 15 years ago

Preview | Code | Blame — 19 lines (8 loc) · 1.87 KB — Raw

| layout | title |
| --- | --- |
| post | Lucene and Amazon S3 |

I spent some time trying to have the ability to store Lucene index on Amazon S3 service. Amazon S3 is a really cool idea, and having the ability to store Lucene index on top of it will provide a simple way to allow storing Lucene index in a distributed environment supporting HA. It will also make a lot of sense for applications deployed on Amazon EC2, since working with S3 from EC2 is free.

It was pretty simply to implement Lucene Directory interface on top of Amazon S3. A bucket is considered to be a Lucene index, and each file has one file object that holds its meta data, and 0 or more file objects holding portions of it (naturally, it is configurable). This, with Compass support for such storage, and Compass local cache support, should provide minor performance overhead when switching from local file system to S3.

Even before I embarked on this quick hacking session, the main thing I was concerned about was how to implement locking on top of S3. There is no formal locking API for it, but I heard somewhere that bucket creation is atomic. Assuming that it is, a very simple locking support can be done (creating a bucket and succeeding indicates a lock obtained, failure means it is locked already, deletion of a bucket releases the lock). Sadly, this is not the case and bucket creation is certainly not atomic. Funnily enough, it does not even fail when trying to create an already existing bucket.

So for now I shelved the implementation. It would be great if the good people at Amazon would allow for simple locking support. I understand that this is not simple to do in a distributed environment (hey, I work at GigaSpaces), but it must be there in some form, it will make S3 much a more attractive offer.

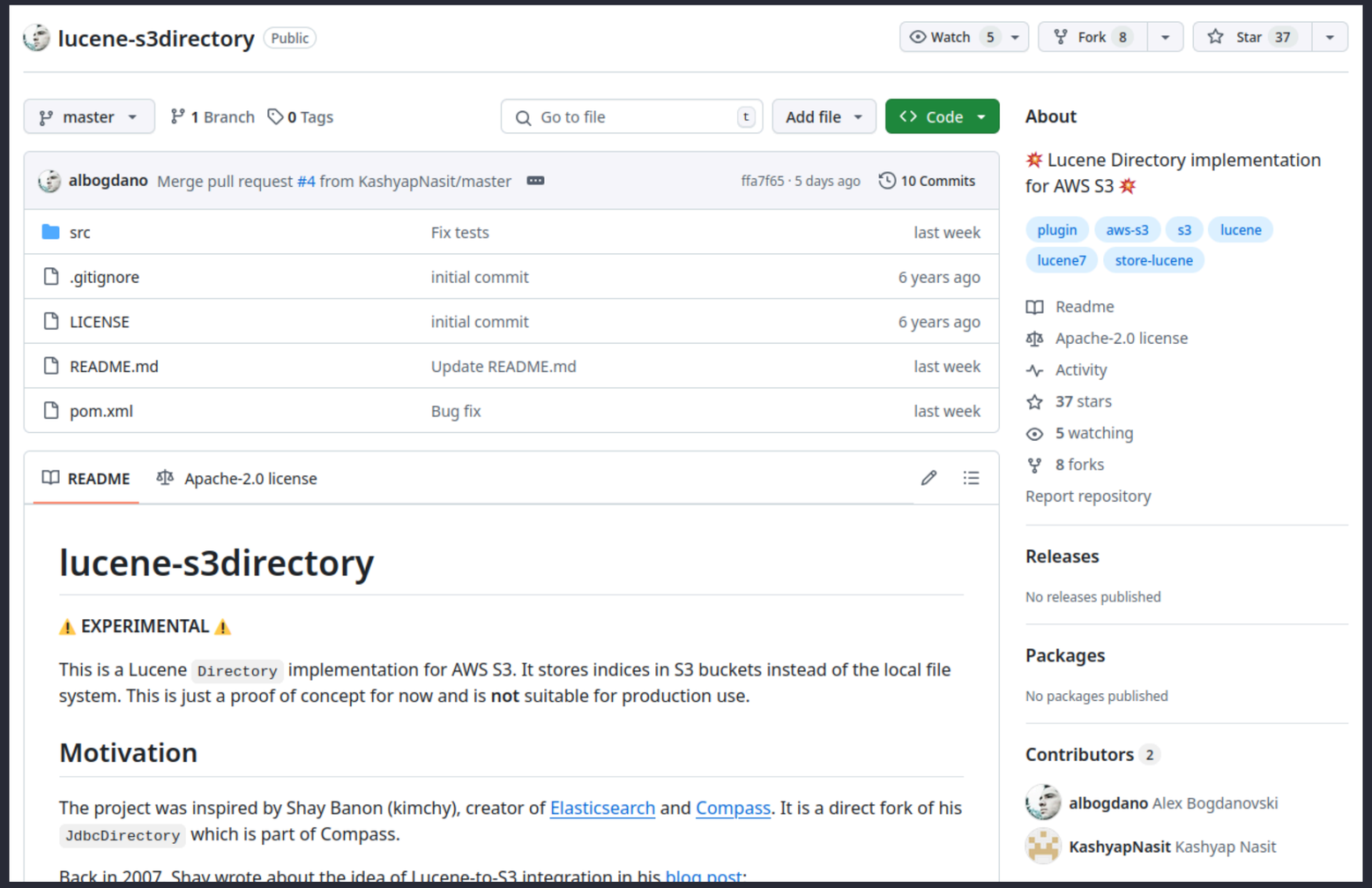
# Lucene Directory

IO abstraction for data access:

- **lawful good**: MMapDirectory, ByteBuffersDirectory
- **chaotic evil**: JDBCDirectory

```
public abstract class Directory implements Closeable {
  public abstract String[] listAll();
  public abstract void deleteFile(String name);
  public abstract long fileLength(String name);
  public abstract void rename(String source, String dest);
  public abstract IndexOutput createOutput(String name, IOContext context);
  public abstract IndexInput openInput(String name, IOContext context);
  public abstract void close();
}
```

# Lucene S3 Directory



lucene-s3directory Public

Watch 5 Fork 8 Star 37

master 1 Branch 0 Tags Go to file t Add file Code

albogdano Merge pull request #4 from KashyapNasit/master ffa7f65 · 5 days ago 10 Commits

| | | |
|---|---|---|
| src | Fix tests | last week |
| .gitignore | initial commit | 6 years ago |
| LICENSE | initial commit | 6 years ago |
| README.md | Update README.md | last week |
| pom.xml | Bug fix | last week |

README Apache-2.0 license

## lucene-s3directory

⚠️ EXPERIMENTAL ⚠️

This is a Lucene `Directory` implementation for AWS S3. It stores indices in S3 buckets instead of the local file system. This is just a proof of concept for now and is **not** suitable for production use.

### Motivation

The project was inspired by Shay Banon (kimchy), creator of Elasticsearch and Compass. It is a direct fork of his `JdbcDirectory` which is part of Compass.

Back in 2007, Shay wrote about the idea of Lucene-to-S3 integration in his blog post:

About

Lucene Directory implementation for AWS S3

plugin aws-s3 s3 lucene lucene7 store-lucene

Readme
Apache-2.0 license
Activity
37 stars
5 watching
8 forks
Report repository

Releases

No releases published

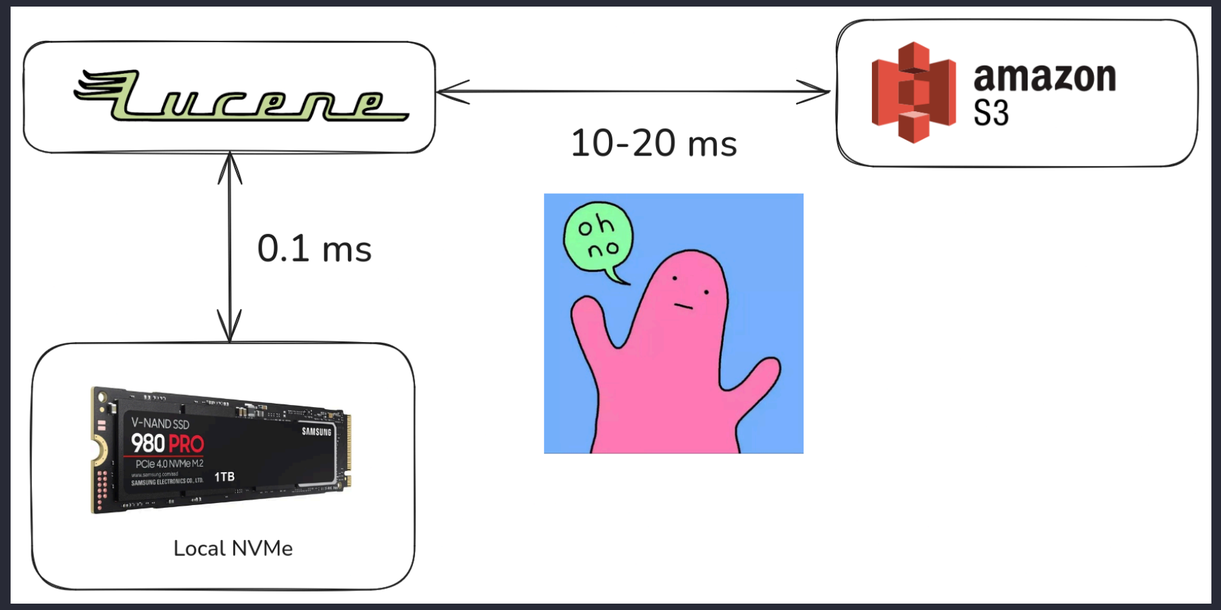
Packages

No packages published

Contributors 2

albogdano Alex Bogdanovski
KashyapNasit Kashyap Nasit

# S3 - a remote block store!




## Performance

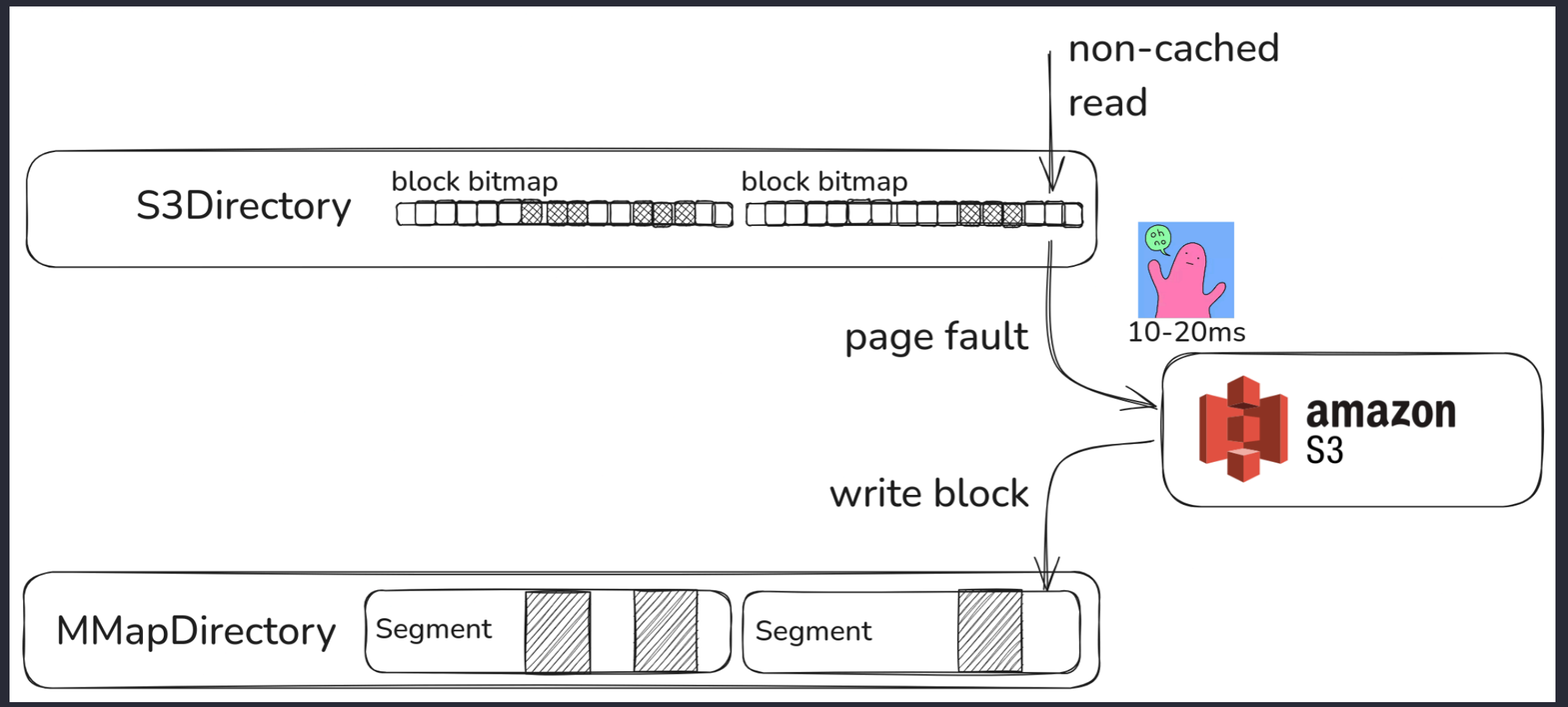
Performance is not great. Each request to AWS takes a lot of time - TLS handshake, signature calculation, etc. I tried to do my best to optimize the code but I'm sure it can be optimized further. Contributions are welcome.

`S3DirectoryBenchmarkITest.java`:

```
RAMDirectory Time: 225 ms
FSDirectory Time : 62 ms
S3Directory Time : 16859 ms
```

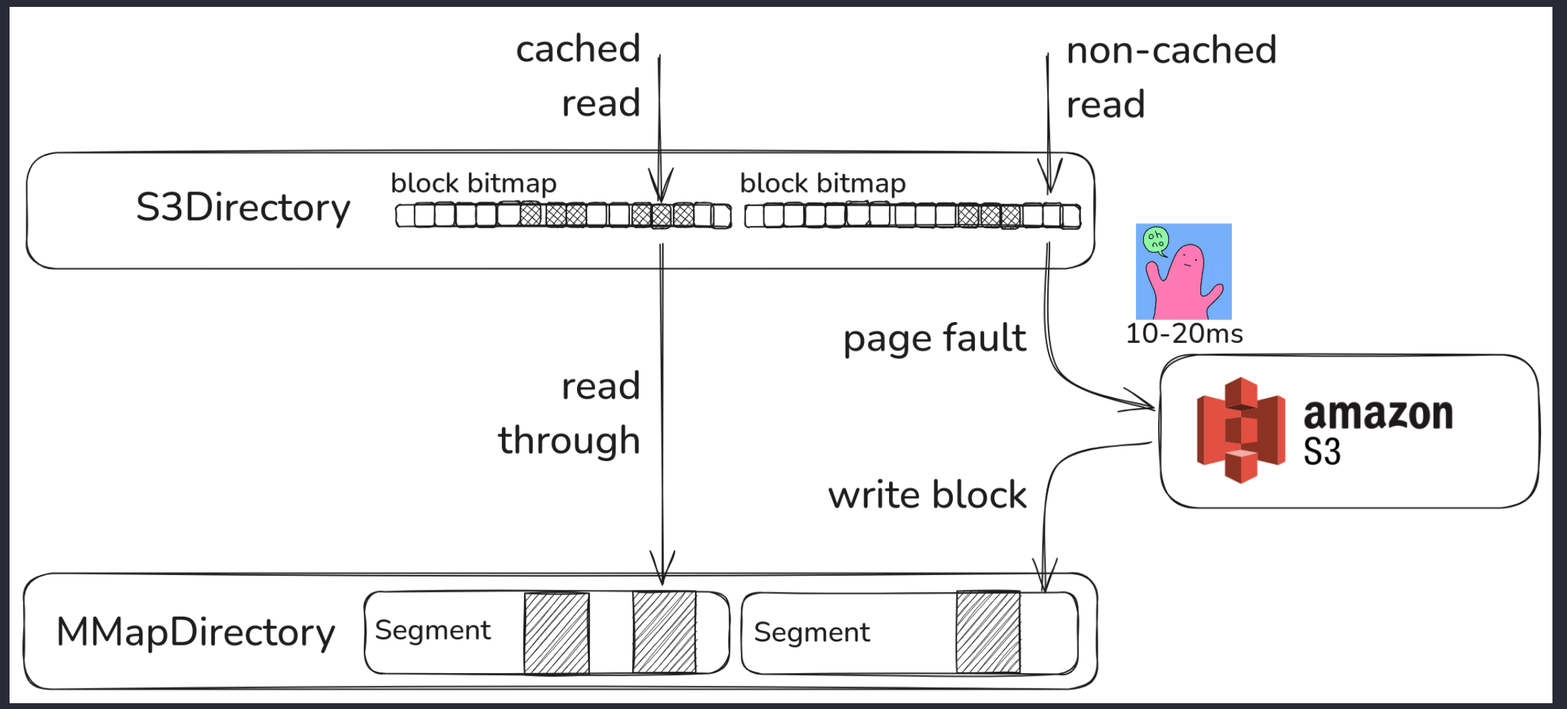
# Nixie v0.0.1: S3Directory

Own read-only S3Directory, with block caching



# Nixie v0.0.1: S3Directory

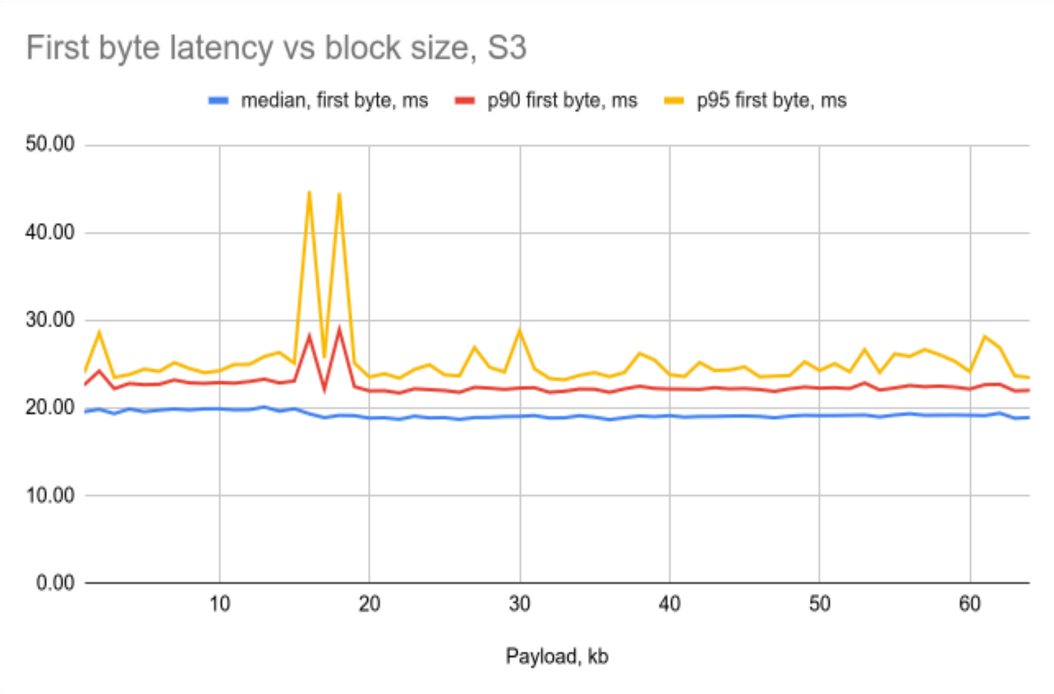
Own read-only S3Directory, with block caching

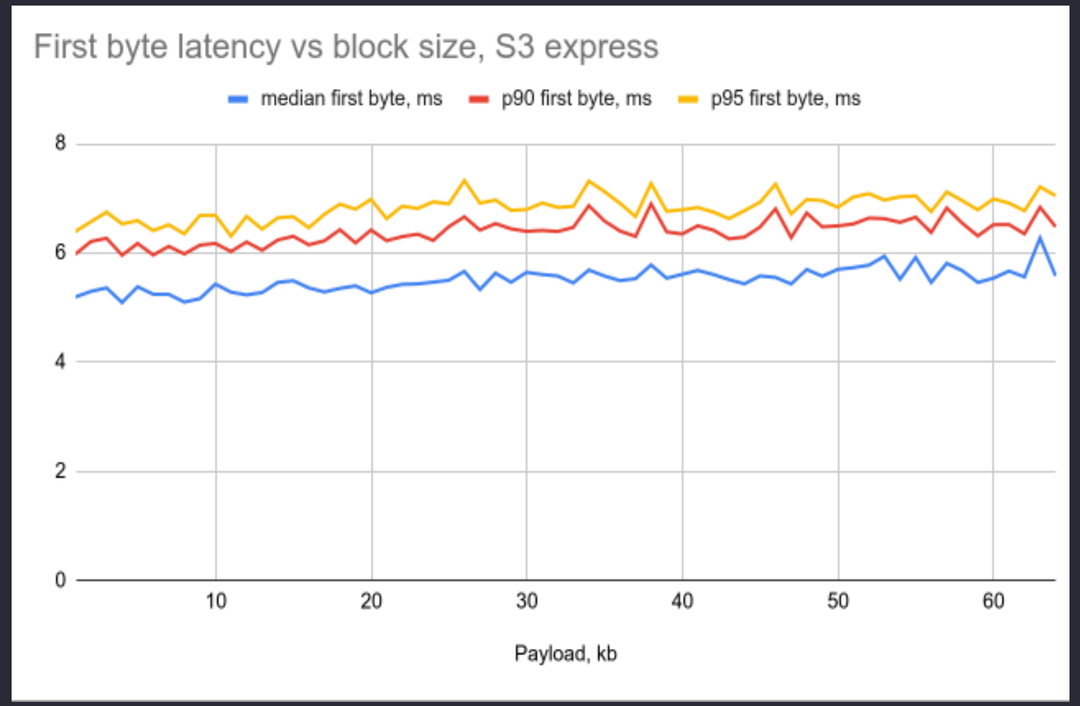



- **pros:** no storage, stateless nodes
- **pros:** easy up-down scaling, state - file on S3
- **con:** LATENCY

# S3 vs S3-Express latency

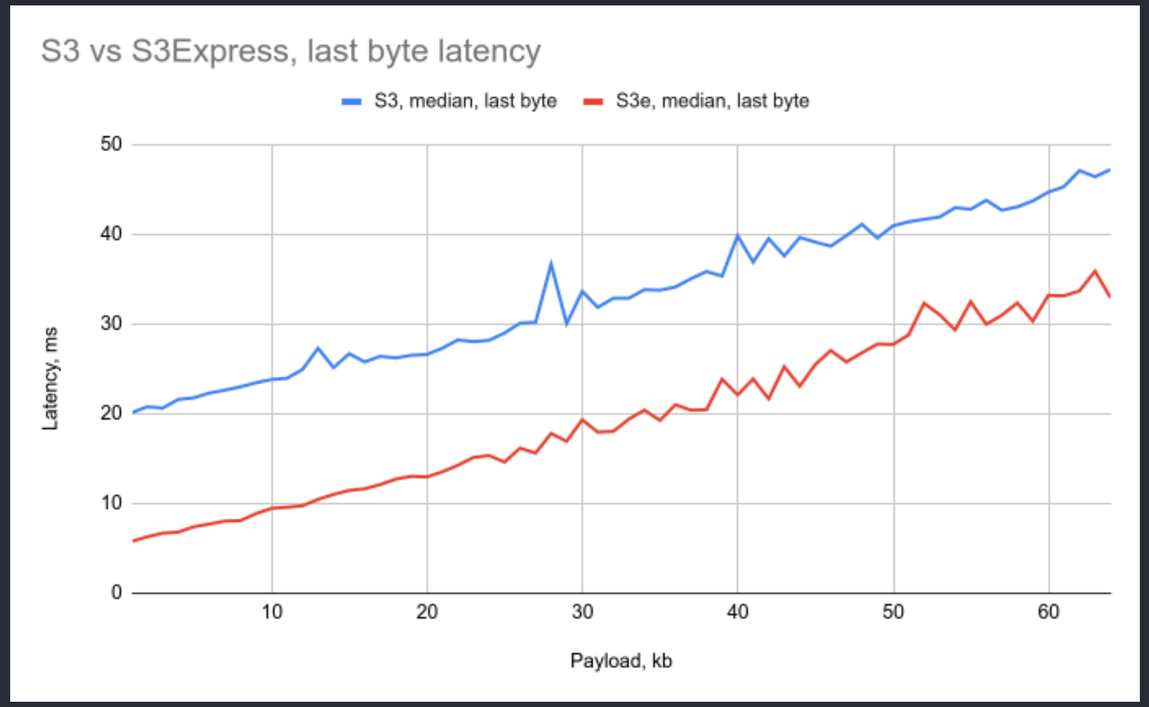
S3 Express: low-latency single-AZ S3




**first-byte latency**: 20ms vs 5ms

# S3 vs S3-Express latency




**last-byte latency**: const_delay + transfer

# Experiment setup

- MSMARCO dataset, 10k, 100k, 1m documents
- HNSW search over e5-base-v2 (768 dims), 1 segment
- Default Lucene HNSW settings (M=16, efC=100)

| | docs=10k | size=30Mb | | |
|---|---|---|---|---|
| Block size | Read ops | Read rate | Op latency, ms | Request #1, ms |
| 4096 | 105 | 17.48% | 6.76 | 709.36 |
| 8192 | 68 | 22.64% | 8.03 | 545.80 |
| 16384 | 42 | 27.96% | 11.59 | 486.72 |
| 32768 | 23 | 30.62% | 17.99 | 413.85 |
| 65536 | 12 | 31.96% | 32.91 | 394.89 |

| | docs=100k | size=300Mb | | |
|---|---|---|---|---|
| Block size | Read ops | Read rate | Op latency, ms | Request #1, ms |
| 4096 | 604 | 0.82% | 6.76 | 4,080.49 |
| 8192 | 443 | 1.21% | 8.03 | 3,555.72 |
| 16384 | 363 | 1.98% | 11.59 | 4,206.61 |
| 32768 | 319 | 3.48% | 17.99 | 5,739.86 |
| 65536 | 288 | 6.29% | 32.91 | 9,477.24 |

| | docs=1M | size=3Gb | | |
|---|---|---|---|---|
| Block size | Read ops | Read rate | Op latency, ms | Request #1, ms |
| 4096 | 488 | 0.07% | 6.76 | 3,296.82 |
| 8192 | 366 | 0.10% | 8.03 | 2,937.68 |
| 16384 | 304 | 0.17% | 11.59 | 3,522.89 |
| 32768 | 263 | 0.29% | 17.99 | 4,732.23 |
| 65536 | 247 | 0.54% | 32.91 | 8,128.05 |

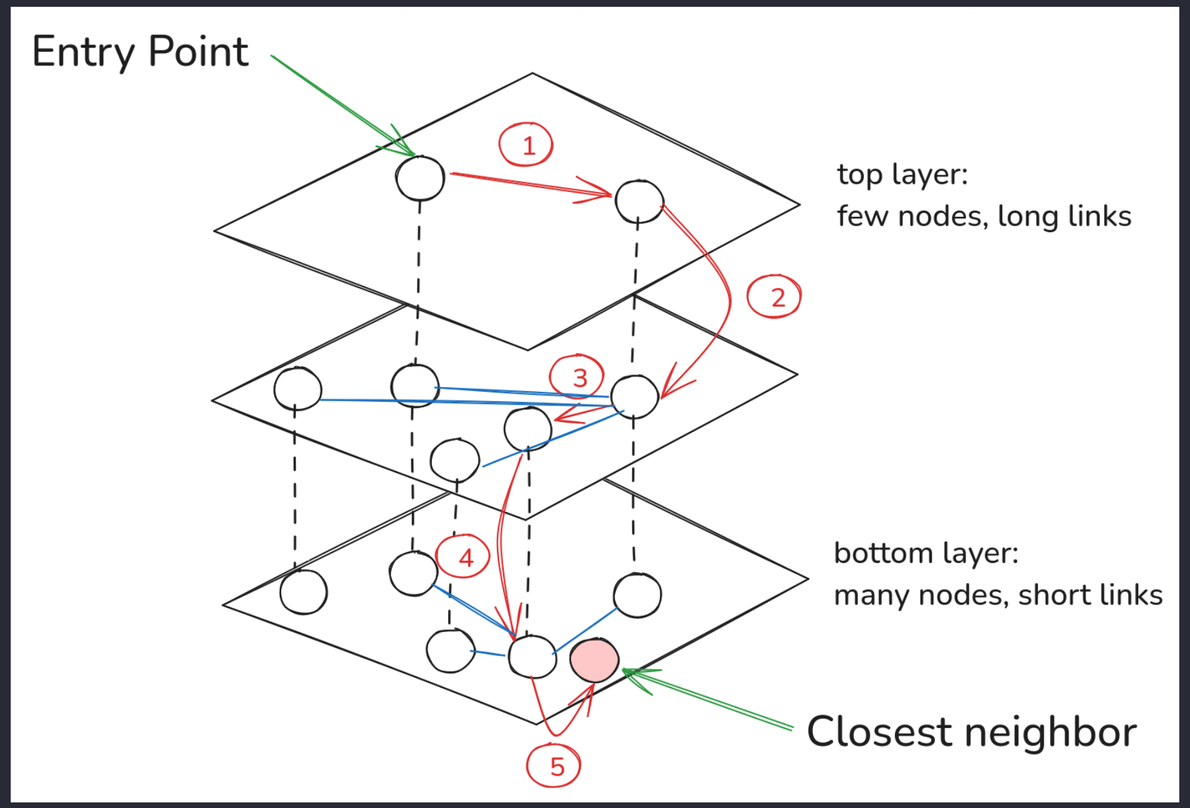
| | docs=10k | size=30Mb | | |
|---|---|---|---|---|
| Block size | Read ops | Read rate | Op latency, ms | Request #1, ms |
| 4096 | 105 | 17.48% | 6.76 | 709.36 |
| 8192 | 68 | 22.64% | 8.03 | 545.80 |
| 16384 | 42 | 27.96% | 11.59 | 486.72 |
| 32768 | 23 | 30.62% | 17.99 | 413.85 |
| 65536 | 12 | 31.96% | 32.91 | 394.89 |

| | docs=100k | size=300Mb | | |
|---|---|---|---|---|
| Block size | Read ops | Read rate | Op latency, ms | Request #1, ms |
| 4096 | 604 | 0.82% | 6.76 | 4,080.49 |
| 8192 | 443 | 1.21% | 8.03 | 3,555.72 |
| 16384 | 363 | 1.98% | 11.59 | 4,206.61 |
| 32768 | 319 | 3.48% | 17.99 | 5,739.86 |
| 65536 | 288 | 6.29% | 32.91 | 9,477.24 |

| | docs=1M | size=3Gb | | |
|---|---|---|---|---|
| Block size | Read ops | Read rate | Op latency, ms | Request #1, ms |
| 4096 | 488 | 0.07% | 6.76 | 3,296.82 |
| 8192 | 366 | 0.10% | 8.03 | 2,937.68 |
| 16384 | 304 | 0.17% | 11.59 | 3,522.89 |
| 32768 | 263 | 0.29% | 17.99 | 4,732.23 |
| 65536 | 247 | 0.54% | 32.91 | 8,128.05 |

- 10k docs: #1 request reads 30% of the index
- 1M docs: **3 SECOND LATENCY?**

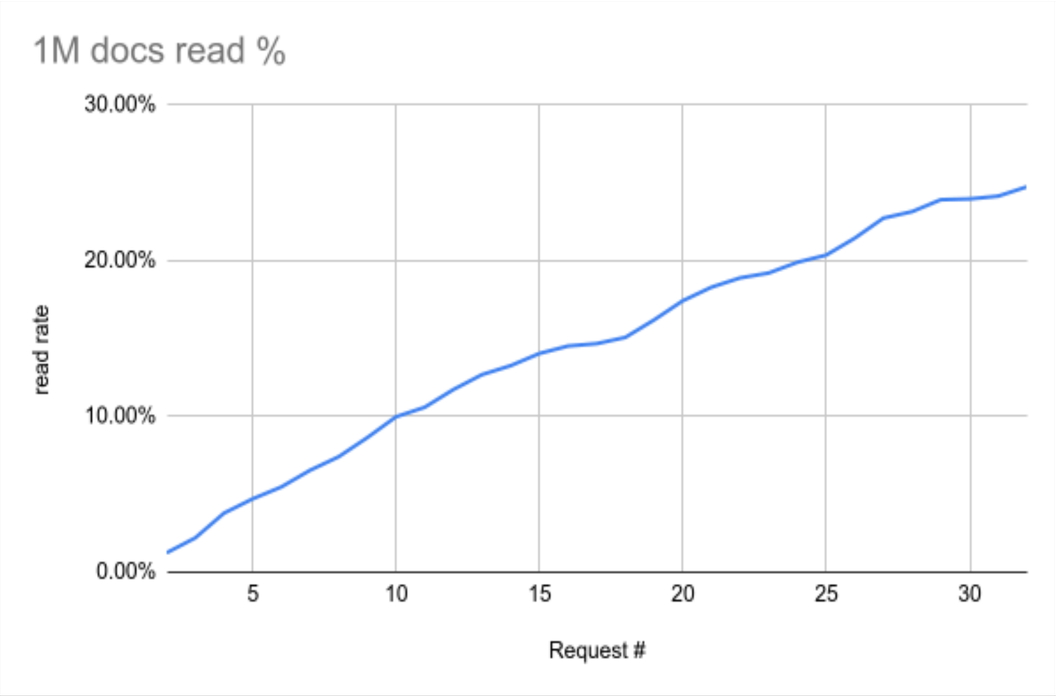
# HNSW for dummies




Each probe = sequental random read = +5ms

# N-th request latency

1M documents, bs=8192:


Not getting better after request #32 😭

# Lucene v10 I/O concurrency

## Improve Lucene's I/O concurrency #13179

New issue

Open · 6 of 9 tasks · **jpountz** opened this issue on Mar 13 · 11 comments

**jpountz** commented on Mar 13 • edited · Contributor

### Description

Currently, Lucene's I/O concurrency is bound by the search concurrency. If `IndexSearcher` runs on N threads, then Lucene will never perform more than N I/Os concurrently. Unless you significantly overprovision your search thread pool - which is bad for other reasons, Lucene will bottleneck on I/O latency without even maxing out the IOPS of the host.

I don't think that Lucene should fully embrace asynchronousness in its APIs, or query evaluation would become overly complicated. But I still expect that we have a lot of room for improvement to allow each search thread to perform multiple I/Os concurrently under the hood when needed.

Some examples:

- When running a query on two terms, e.g. `apache OR lucene`, could the I/O lookups in the `tim` file (terms dictionary) be performed concurrently for both terms?
- When running a query on two terms and start offsets in the `doc` file (postings) have been resolved, could we start loading the first bytes from these postings lists from disk concurrently?
- When fetching the top N=100 stored documents that match a query, could we load bytes from the `fdt` file (stored fields) for all these documents concurrently?

This would require API changes in our `Directory` APIs, and some low-level `IndexReader` APIs (`TermsEnum`, `StoredFieldsReader`?).

- [x] Add IndexInput#prefetch. #13337
- [x] Use `IndexInput#prefetch` for terms dictionary lookups. #13359
- [x] Use IndexInput#prefetch for postings, skip data and impacts #13364
- [x] Add prefetching for doc values and norms. #13411
- [x] Add prefetching support to stored fields. #13424
- [ ] [https://github.com/ Add prefetching support to term vectors. #13758
- [ ] Add prefetching support to KNN vectors
- [ ] Add prefetching support to points
- [x] Set `ReadAdvice#NORMAL` on files that have a forward-only access pattern. #13450

21 8

Assignees
No one assigned

Labels
type:enhancement

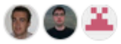
Projects
None yet

Milestone
10.0.0

Development
No branches or pull requests

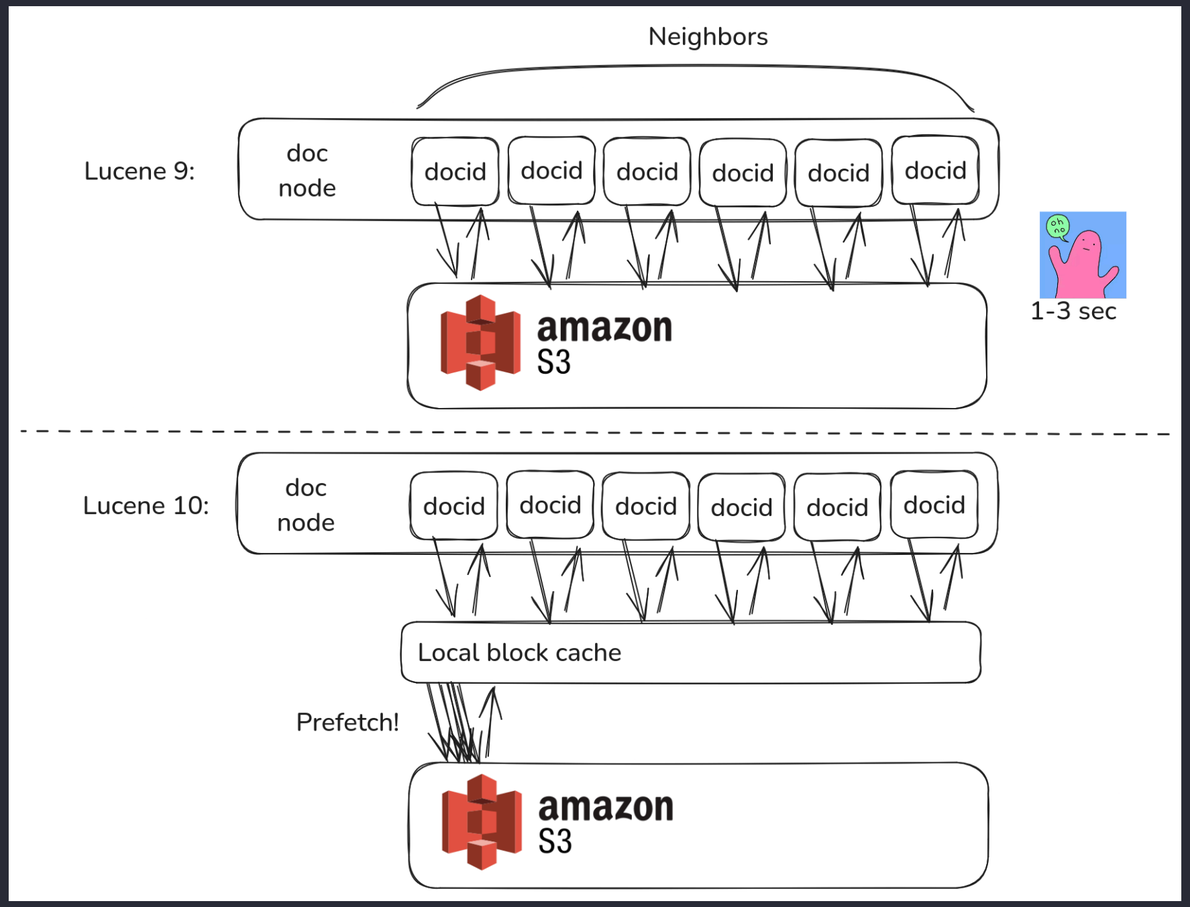
Notifications Customize
Subscribe
You're not receiving notifications from this thread.

3 participants

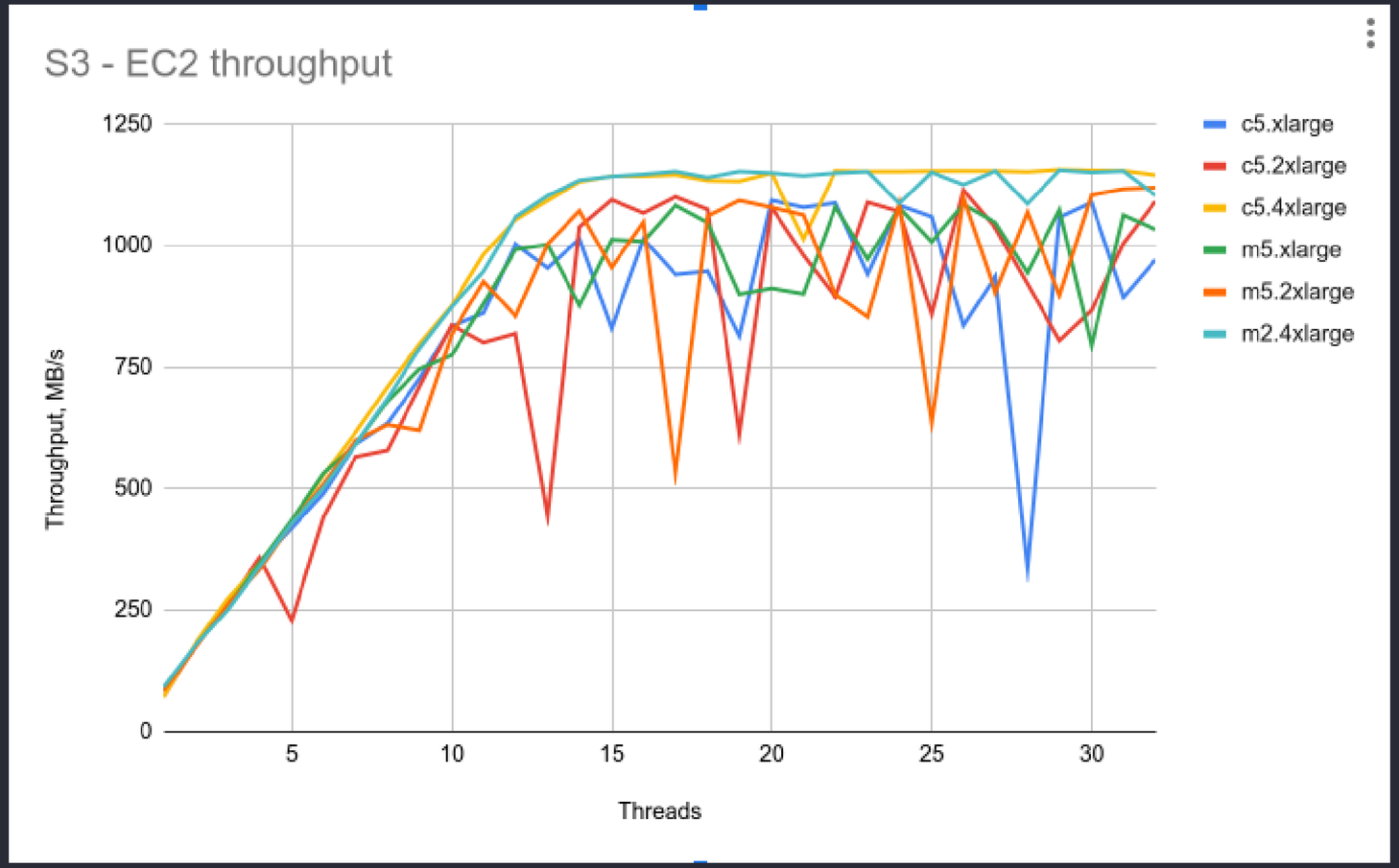


# LUCENE-13179 TLDR

- Sequential IO is slow, let's make it concurrent
- **IndexInput.prefetch** - hint for future reads



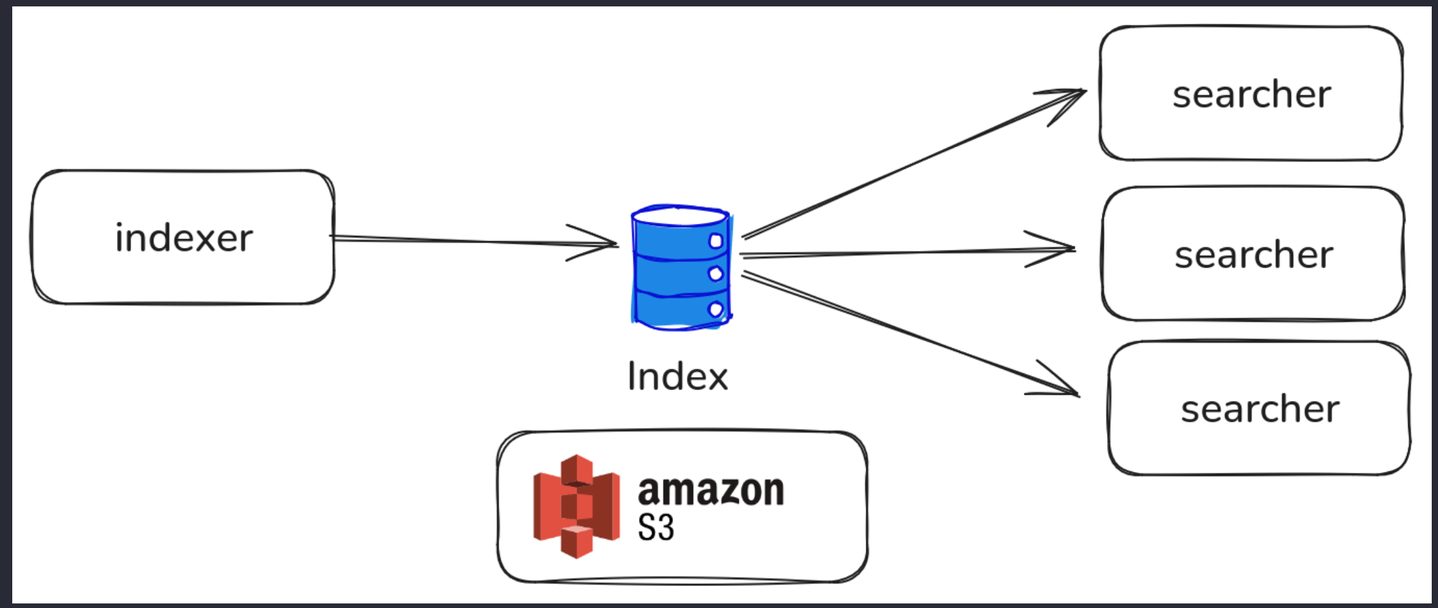
# S3: 👎 latency, 👍 concurrency




- 1 GB/s. Can fetch 10GB index in 10 seconds

# Serverless dilemma

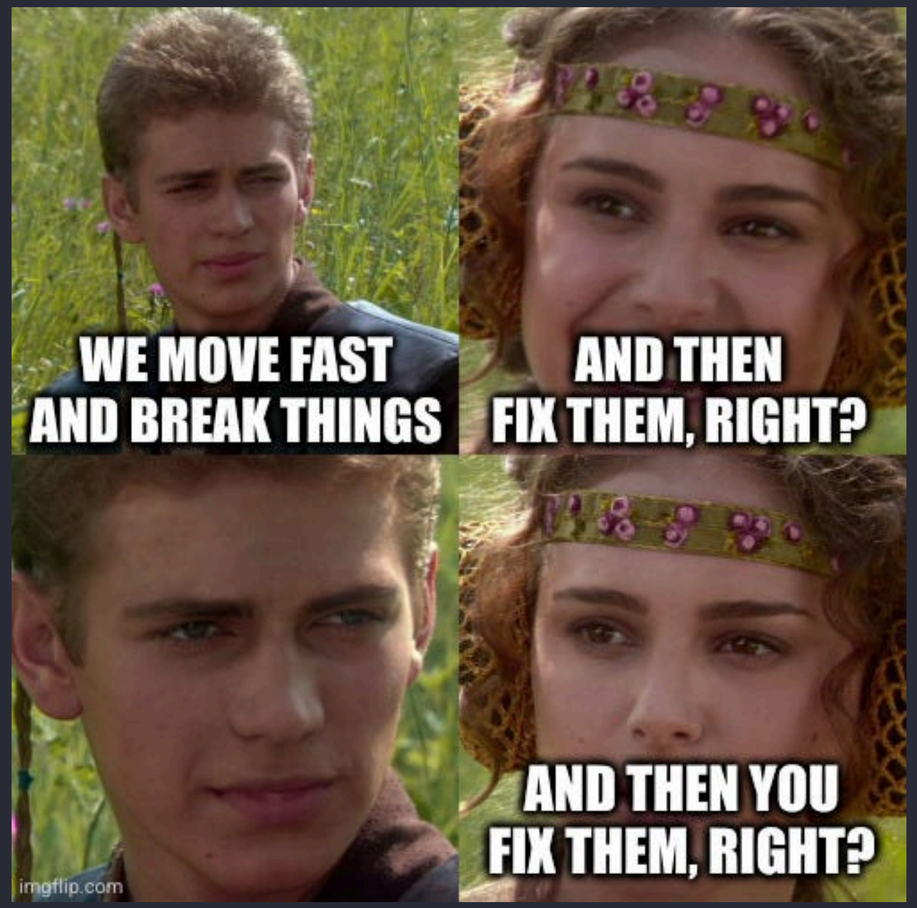
- No cold start: need to sync the index
- Fast startup: high search latency




Nixiesearch: segment replication

# How far does stateless go?

- Stateless index: on S3 ✅
- Immutable configuration 🤔



# Immutable config? 🤔

Regular backend app config change:

- Commit config to git
- PR review, CI/CD blue-green deploy

# Immutable config? 🤔

Mapping change for an index:

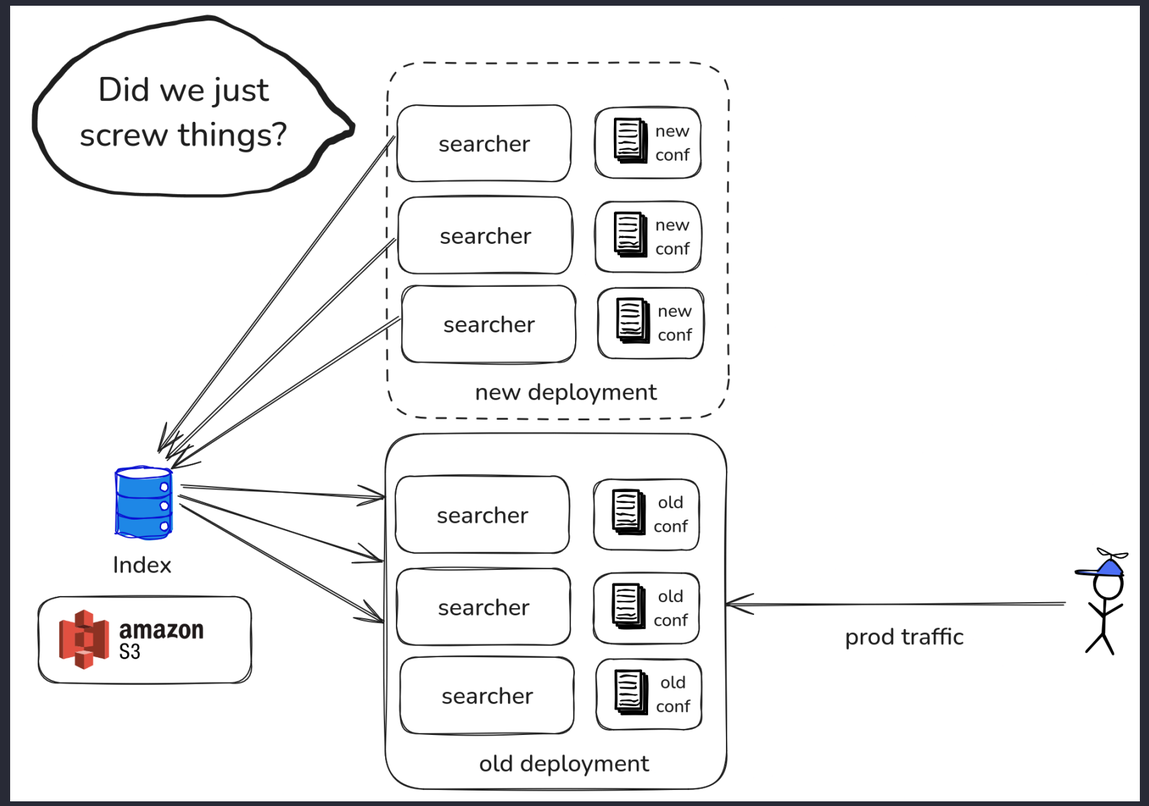
- Send HTTP POST request to prod cluster
- Earth shaking, light goes off
- You hear siren sounds

# Config management in Nixie

```yaml
inference:
  embedding:
    text:
      provider: onnx
      model: nixiesearch/e5-small-v2-onnx

schema:
  helloworld:
    fields:
      title:
        type: text
        search:
          type: semantic
          model: text
      price:
        type: int      # can be also float/long/double
        filter: true
        facet: true
        sort: true
```

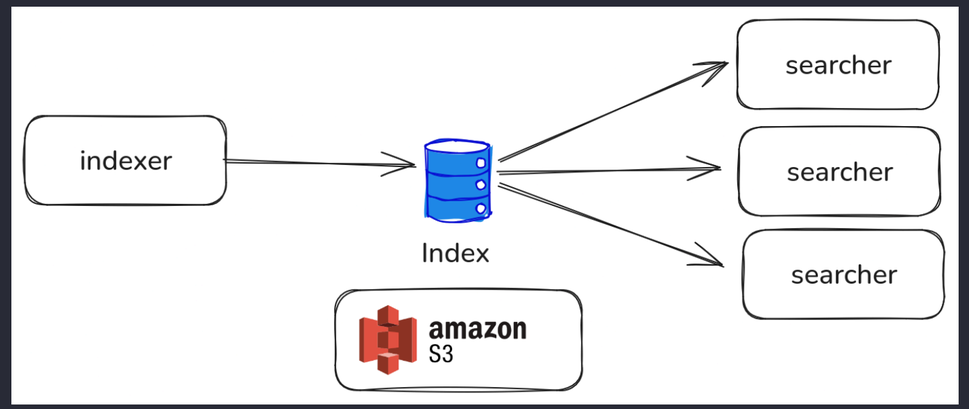
# Search is not special




- No way to change the runtime config
- Index schema, system settings = just conf

# When things go bad

- Index + config compatible = healthcheck OK
- int -> string = need to reindex ☢️

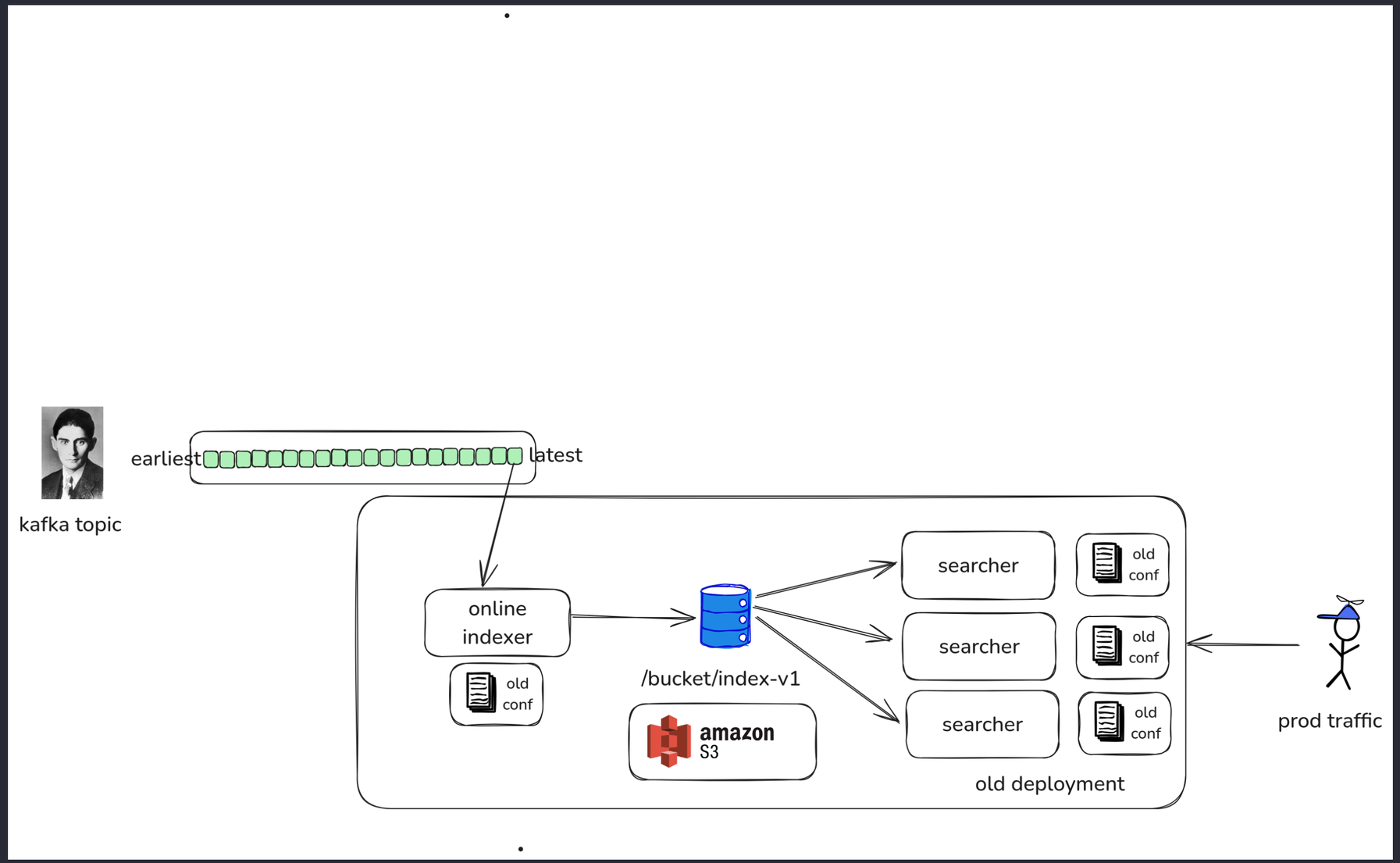



index = directory on S3

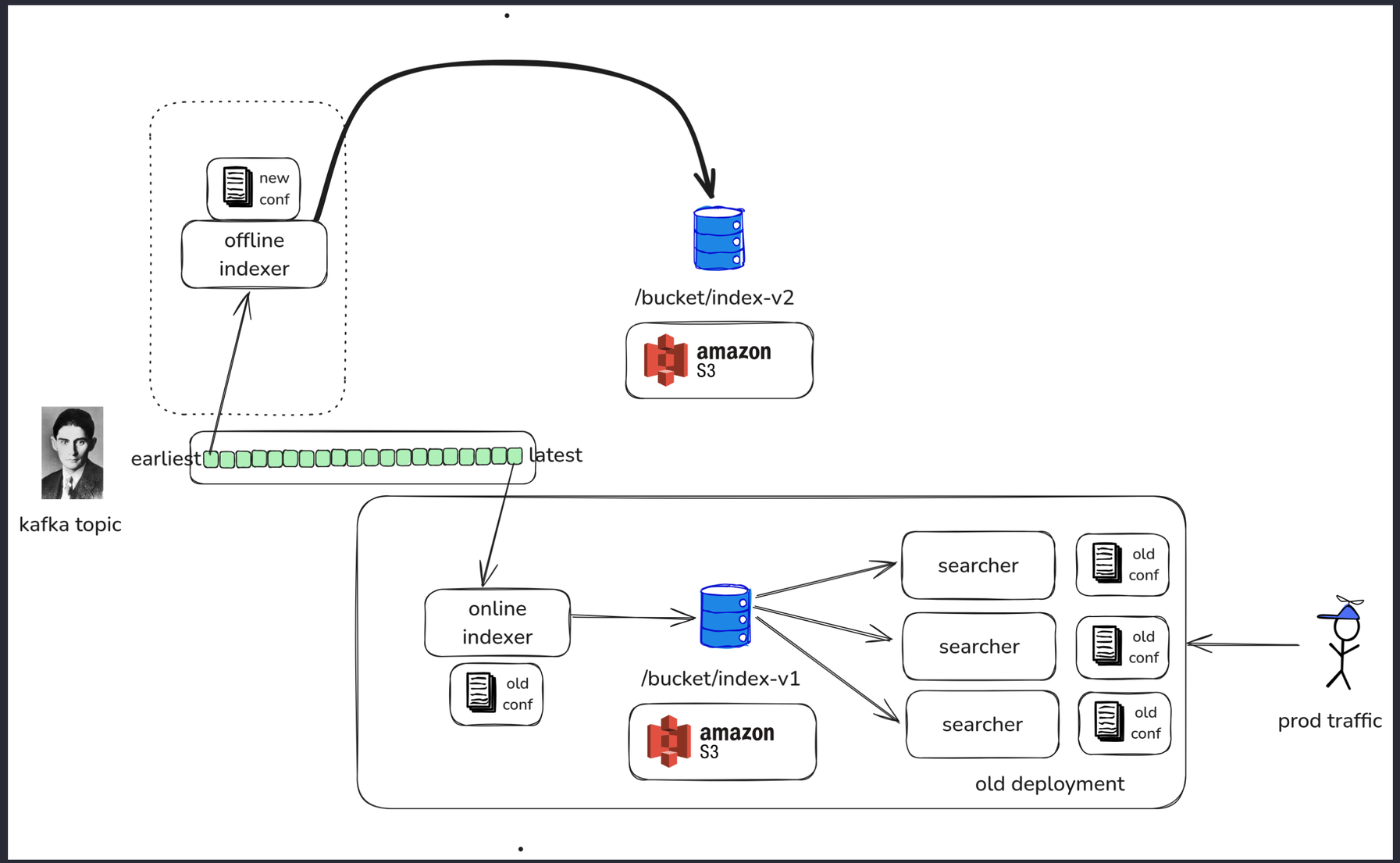
# Push vs pull indexing


- **Push**: same cluster, control backpressure
- **Pull**: separate service, offline

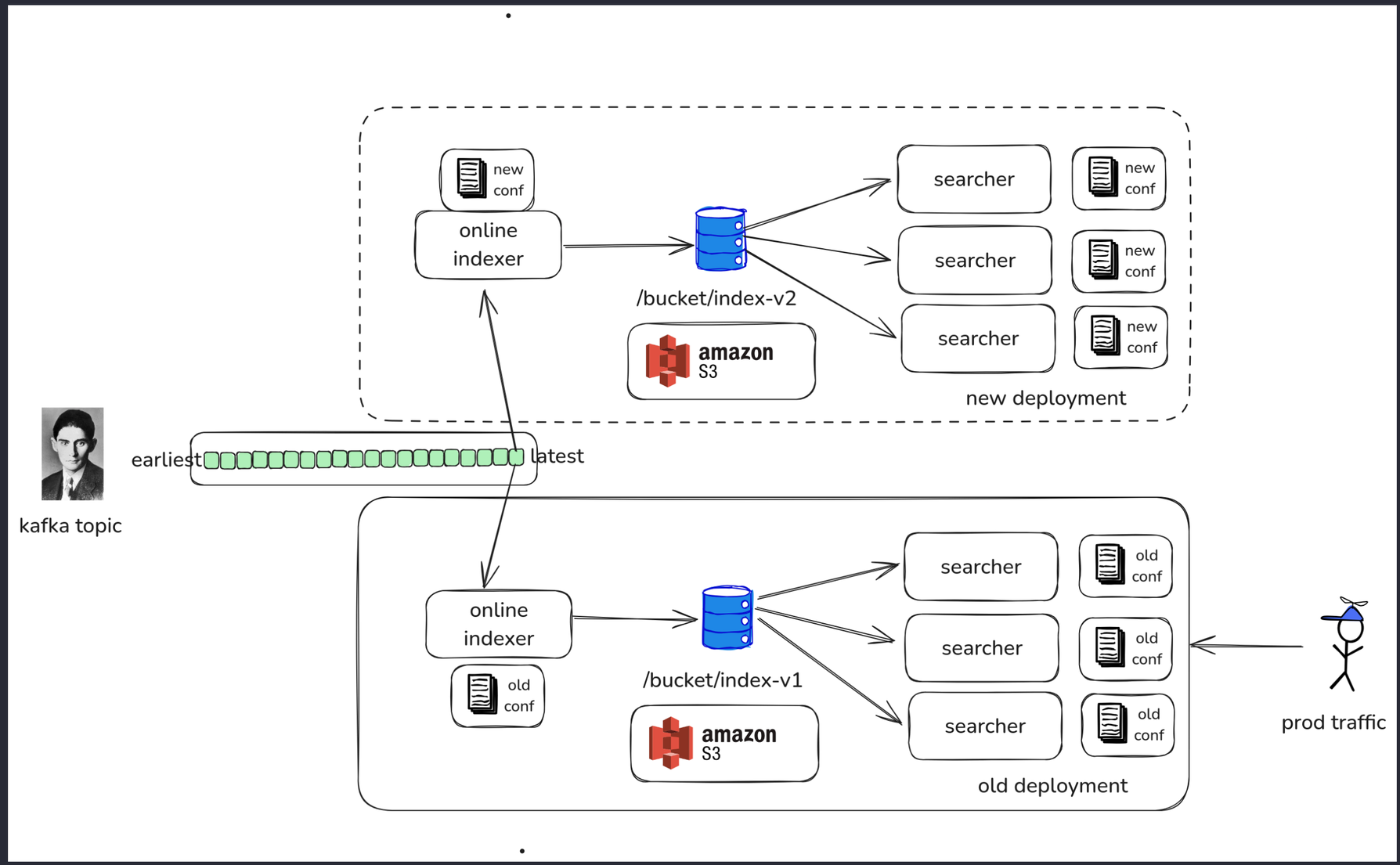
# Offline reindex



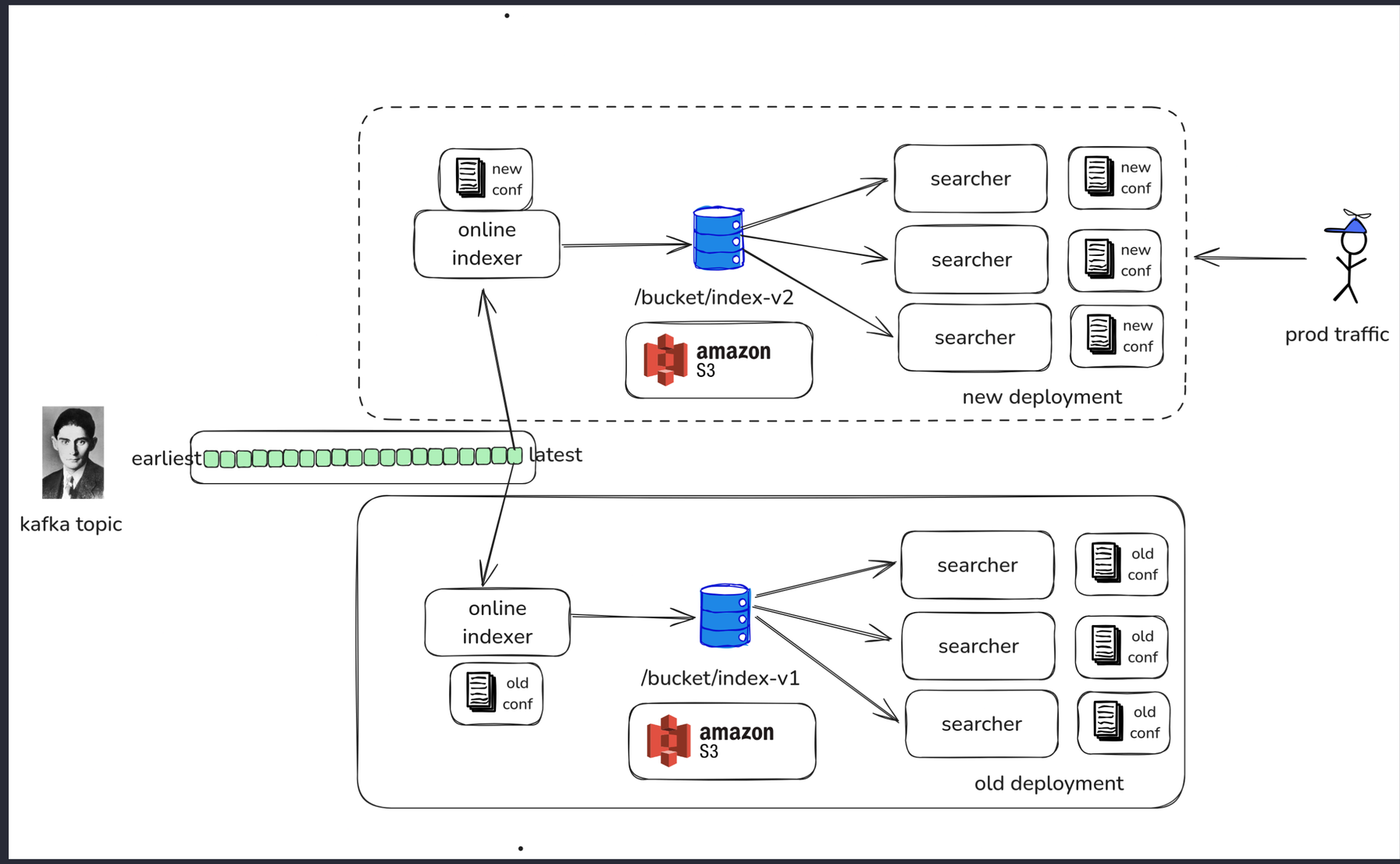
# Offline reindex



# Offline reindex

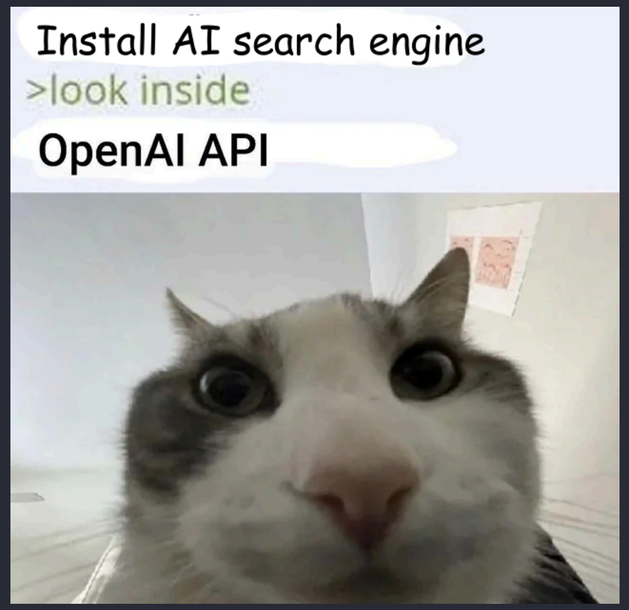


# Offline reindex



# demo

# Text in, text out, 100% local




- Embeddings and LLM inference is local
- ONNX, GGUF - GPU for indexing 🤔

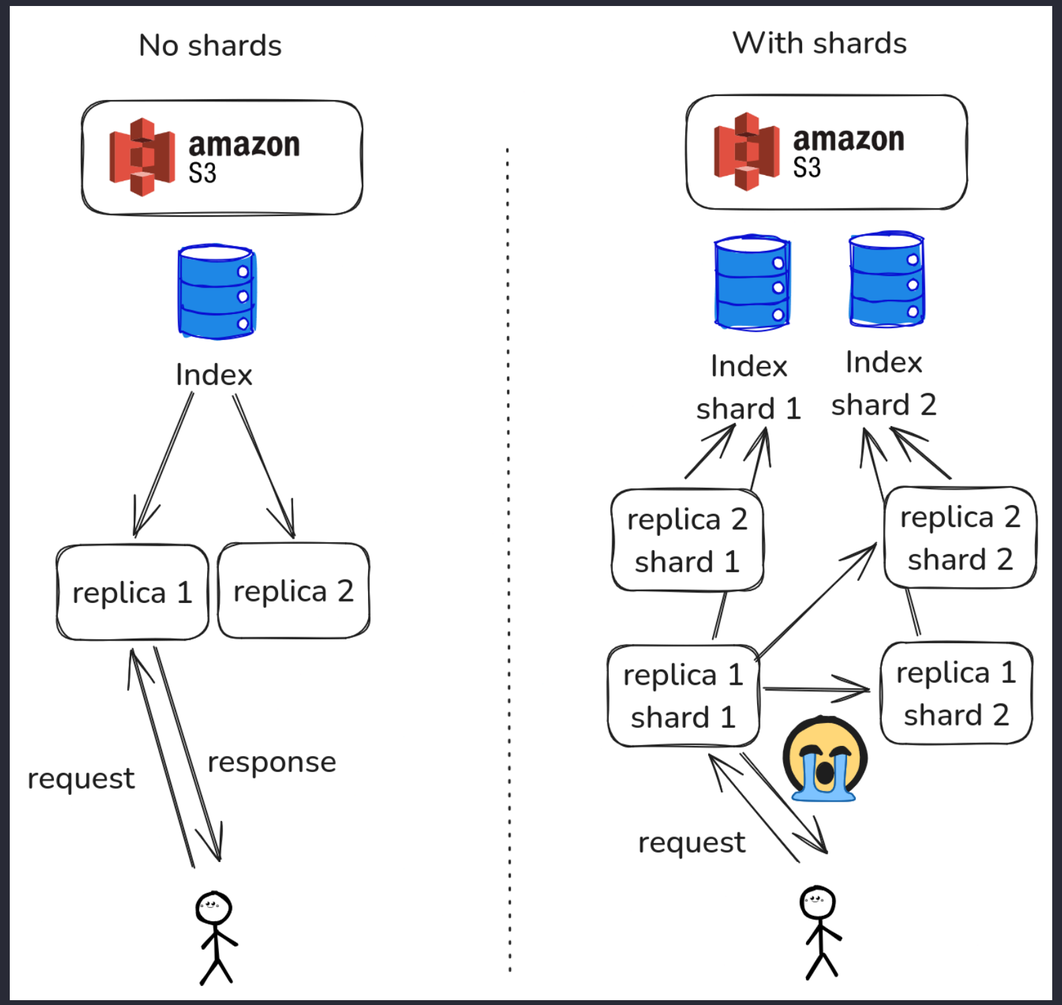
```
docker run --gpus=all nixiesearch/nixiesearch:0.3.3-amd64-gpu index
```

# Why local inference?

1. **Latency**: CPU ONNX e5-base-v2 inference is 5ms
2. **Privacy**: data is not leaving your perimeter

Optional: openai, cohere, mxb, google providers

# The Bad, The Ugly




- No sharding support (yet)

# No sharding, really?


- 1M docs MSMARCO index: 3GB
- 1B index: 99% it's logs/APM/traces

# Future: reranking support

Single retrieval pipeline:

- Cross-encoder: rerank top-N candidates
- retrieve -> rerank -> summarize
- Local ONNX, optional external providers

# Future: ingestion pipeline

Common NLP tasks, automated:

- Transform a field before indexing
- Split to chunks, summarize
- Contextual embeddings for RAG
- Automated category detection

EC2 g4.large with T4 GPU = 300$/month

# Future: domain adaptation

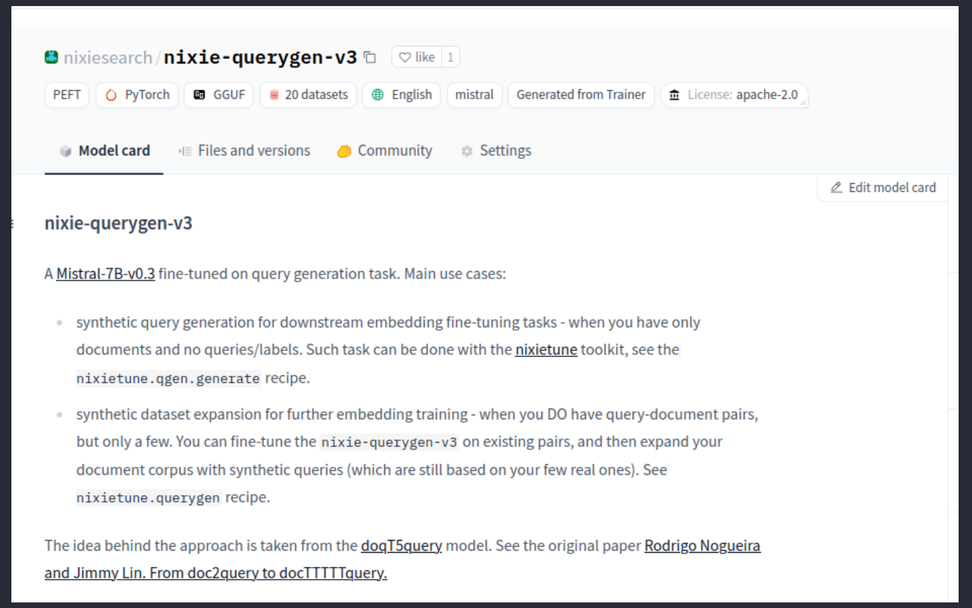
Different search engines, same embedding, same results

- **Fine-tuning is dead**: you need training data
- **You have only docs**: can you generate queries and labels?

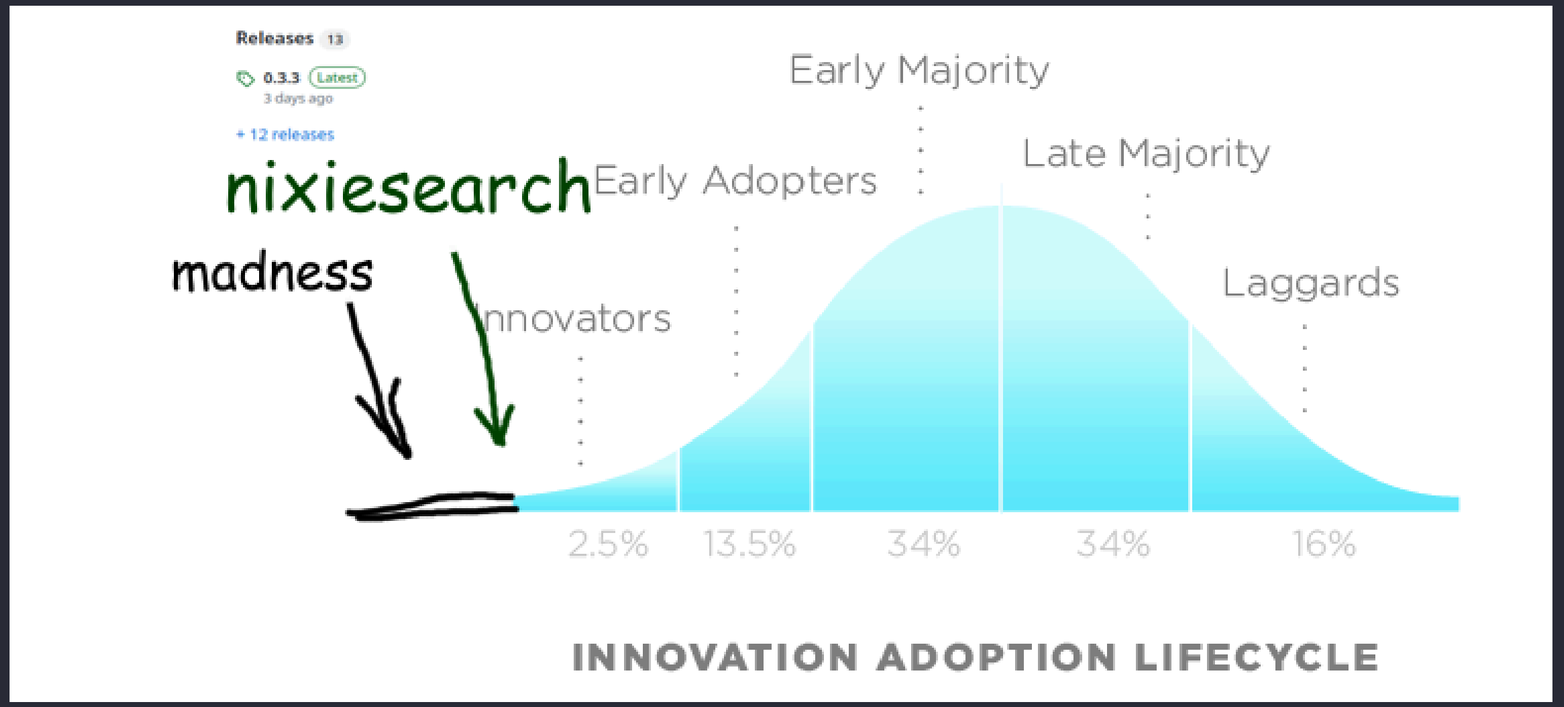
# Future: domain adaptation

Fine-tuning on LLM-generated synthetic data

- For each document **generate a query**: it's a positive
- For query+positive, **mine negatives**
- **Fine-tune** the embedding model



# Expect a bumpy ride




- Some features might be missing (like sorting)
- Docs are imperfect - but they exist!
- There ~~may~~ will be breaking changes

# Links


- Github: nixiesearch/nixiesearch
- Docs microsite: nixiesearch.ai
- Slack: nixiesearch.ai/slack