Aerospike

Hybrid Search

Python-Powered
Precision and Scalability

Art Anderson
Director of Developer Advocacy

Agenda

01 Building a Better Search

Why build a search engine?

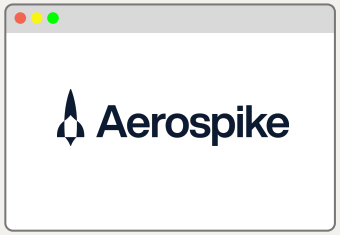
Search on the website was broken, content was hard to find, filtering was non-existent.

- Open the door to more complex search functionality to help users learn and explore.
- Eat a little of our own dog food.
- Why not?

The Plan

The Server

The Data



Scraper

Data Prep

spaCy

API

The Database

Aerospike

The Transformer

Gemini

The Front End

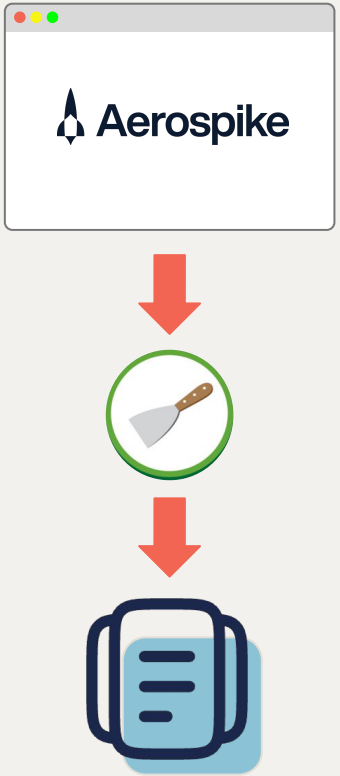
The data

Aerospike.com

- Main website
- Documentation
- Developer Hub
- Blogs
- Knowledge Base

Pages are scraped with Scrapy and converted into markdown.

XPath queries are used to target the relevant content for different page types.



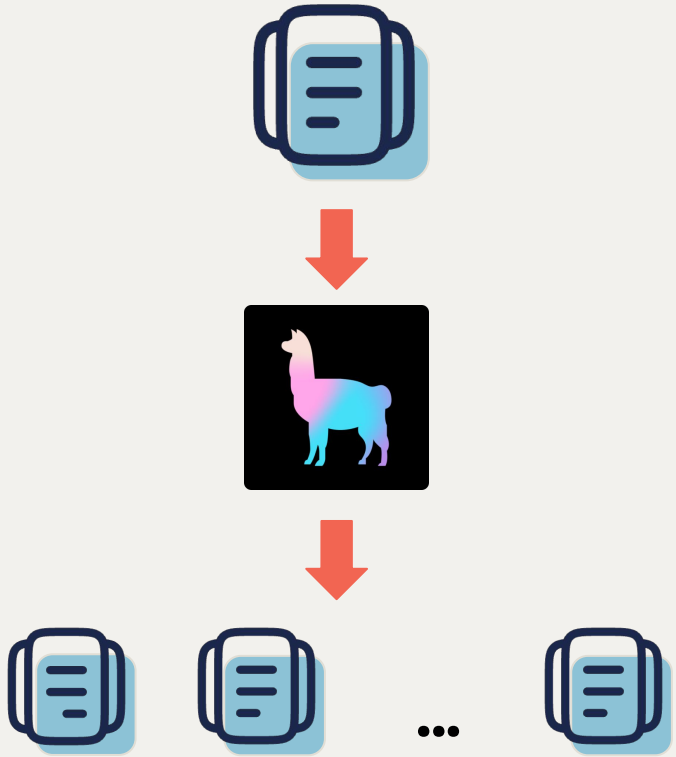
The data

LlamaIndex

- Easy to use
- Light weight
- Comprehensive tool chest

Each page is broken into smaller chunks while maintaining context.

Chunks are prepped for embedding by adding a prefix and metadata.



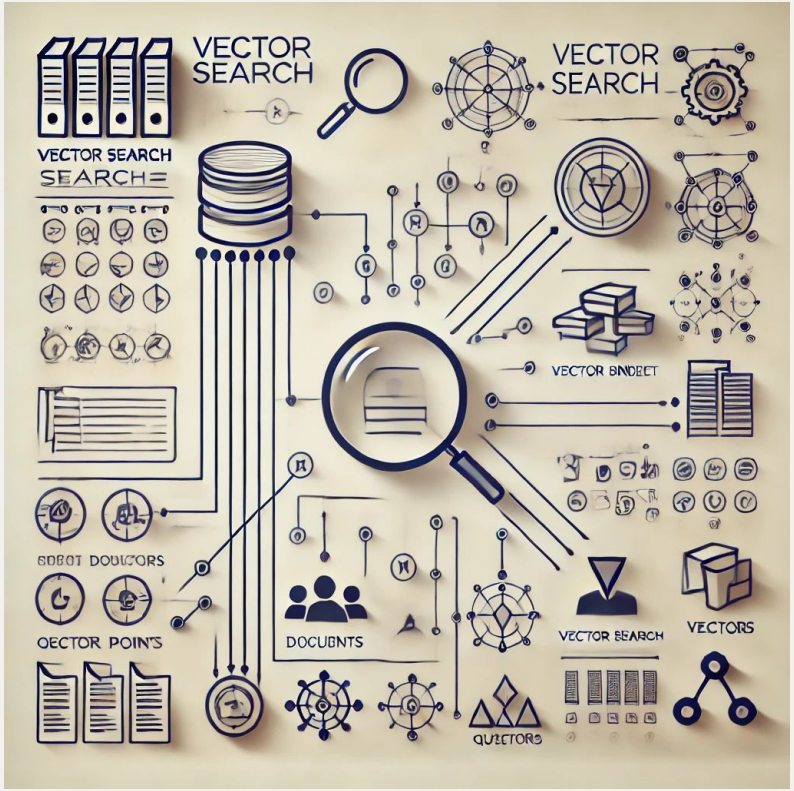
02 Semantic Search

Search with meaning

Search with vectors

Vector embeddings capture the semantic meaning of source data.

- Reduces high volume data according to meaning.
- Similar to a “lossy” algorithm like JPEG compression.



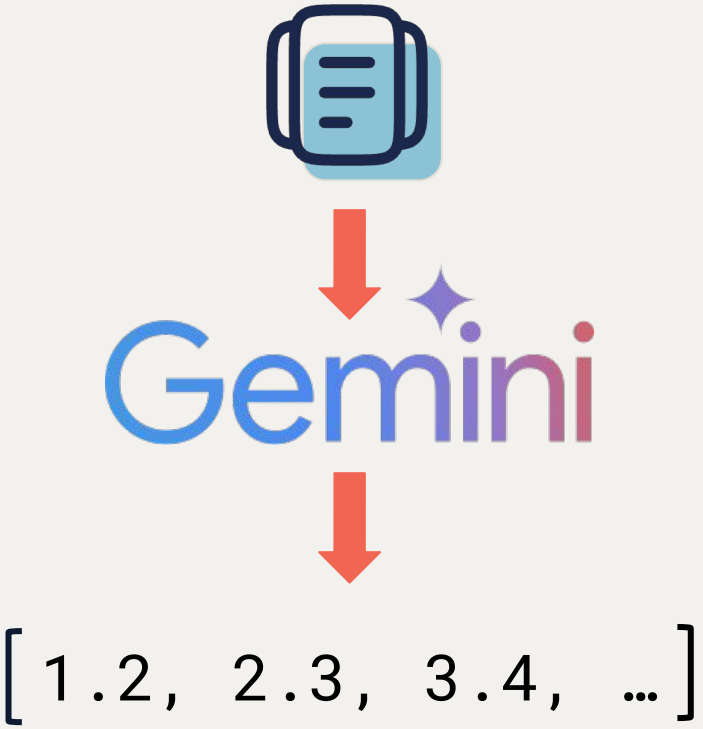
Generating vectors

Gemini

- 768 dimensions
- Fast and easy!

Gemini generates a vector for each chunk, capturing semantic meaning.

This model must be used to generate embeddings on the query as well.



Storing the data

[1.2, 2.3, 3.4, …]

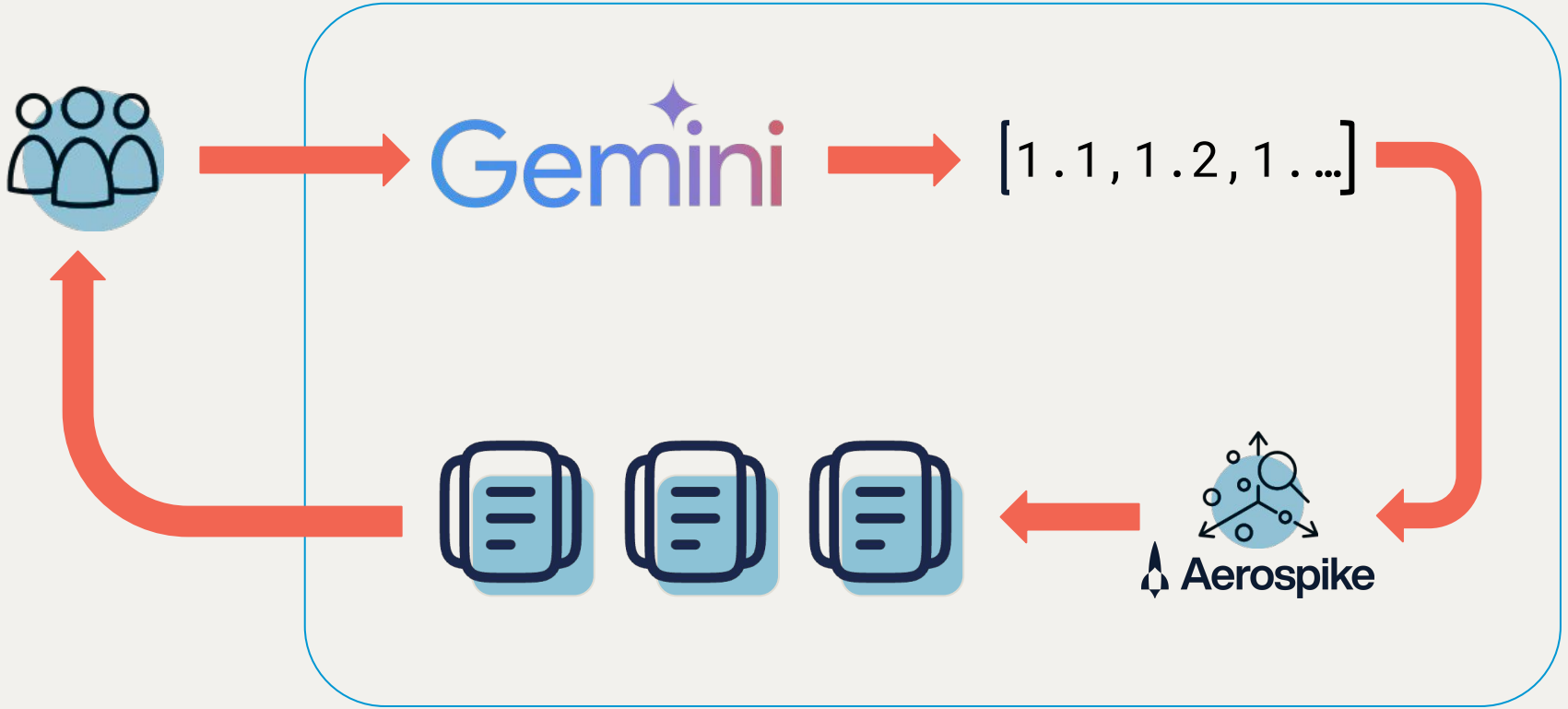
Aerospike Vector Search

- HNSW
- Speed and scale
- It's Aerospike!

Aerospike stores each chunk along with document information.

Cosine similarity is used to query the data. The most similar chunks are returned for the search.

Querying the data



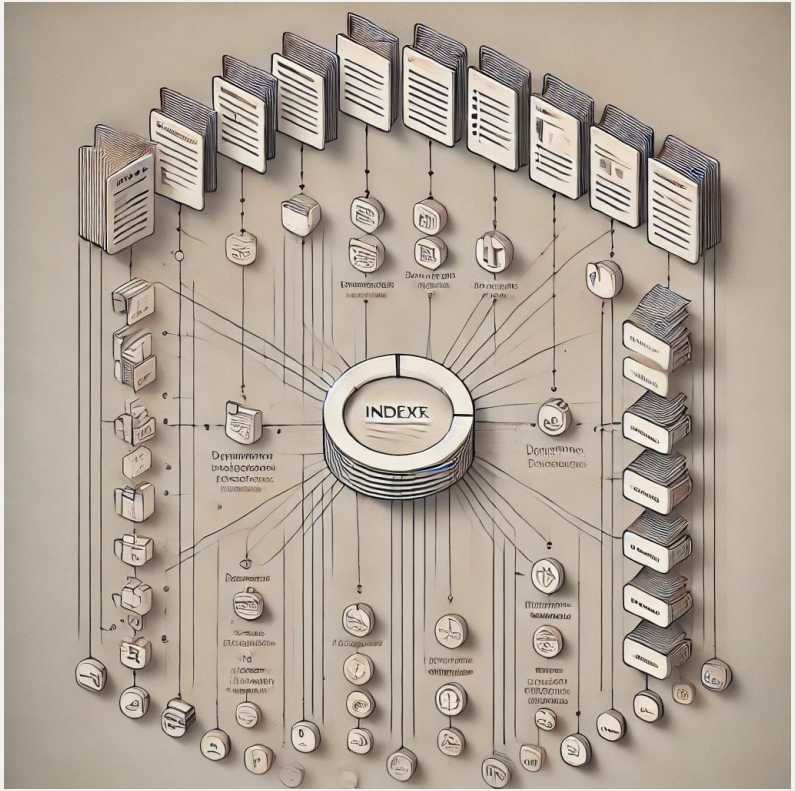
03 Keyword Search

Flip things inside out

Inverted Index

An inverted index is a data model commonly used in search engines. The index is organized by keywords. Each keyword maintains a list of the documents in which it appears.

- Fast full-text search across large data sets.
- Easy to build, but requires complex processing to add to and update the index.



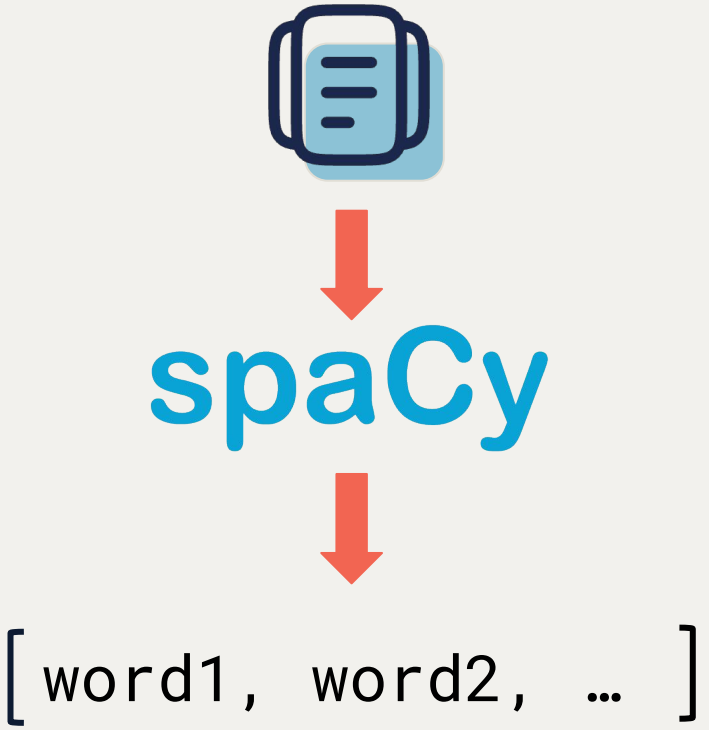
Identifying keywords

spaCy

- Tokenization
- Stop word removal
- Lemmatization

Each chunk goes through the spaCy filter producing a list of tokens used to create the inverted index.

This must be used for filtering the query as well.



Creating the index

[word1, word2, …]

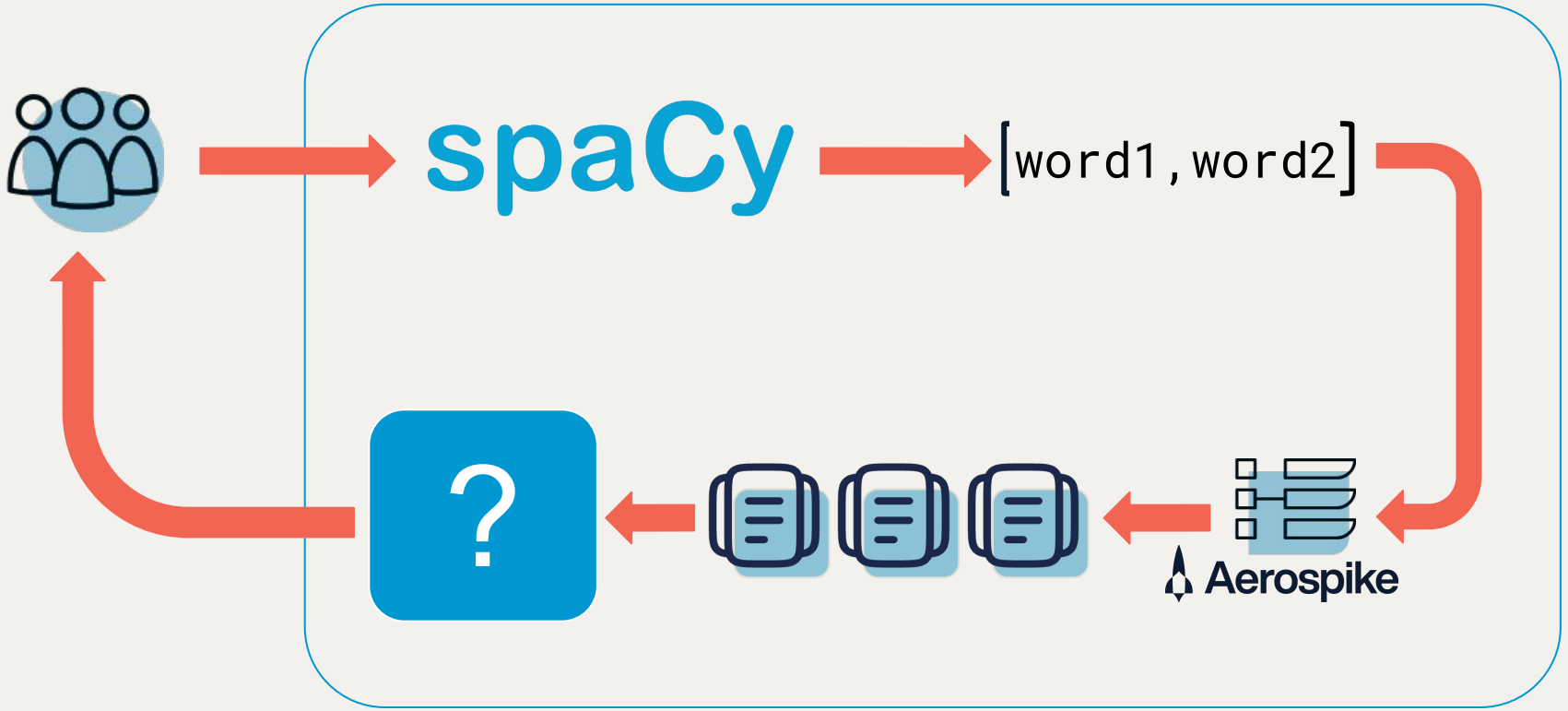
Aerospike Key-Value

- Fast
- Handles complex data types well
- Same storage as vectors

Each keyword is a record containing a map.

Map keys are the document identifier, values record the keyword frequency and position along with other metadata.

Querying the data



BM25 (Best Match 25)

$$\text{score}(D,Q) = \sum_{i}^{n} \underbrace{IDF(q_i)}_{\text{The Inverse Document Frequency}} \cdot \frac{\overbrace{f(q_i,D)}^{\text{How many times does the i}^{th}\text{ term appear in document D}} \cdot (k_1+1)}{f(q_i,D) + \underbrace{k_1}_{\text{Control for term frequency}} \cdot (1 - \underbrace{b}_{\text{Control for document length}} + b \cdot \frac{|D|}{\underbrace{avgdl}_{\text{The length of document D over the average document length}}})}$$

$$\sum_{i}^{n} \log \frac{|D|}{|d : q_i \in d|}$$

The document

The query terms

04 Ranking and Sorting

Putting it all in order

Semantic re-ranking

Pros
- Fast
- Cheap (Google credits!)

Cons
- Pretty terrible results

Pros
- Good results

Cons
- Slow
- Cost

Neither option seemed to be a good fit for our application…

Reciprocal Rank Fusion (RRF)

$$RRF(d) = \Sigma(r \in R)\ 1\ /\ (k + r(d))$$

The document

The sum across all retrievers

The retrievers

The constant (usually 60)

The rank of the current document

05 Tying things together

The server and API

Fully built in Python using FastAPI and uvicorn. Gunicorn is used to manage the worker processes in production.

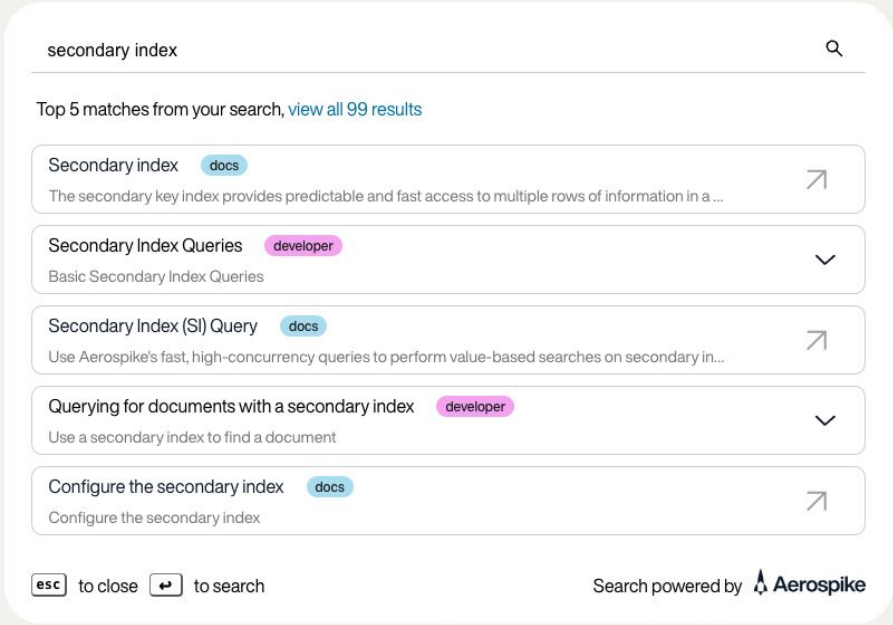
- Fast to develop/Easy to work with
- Excellent performance!


```python
@app.get("/rest/v1/search/")
async def search(
    q: str,
    count: int = 5,
    search_type: str = "hybrid",
    chat: bool = False,
    page: int = 0,
    pageSize: int = 10,
    filters: str = ""
):
    # Do stuff!
```


The front end

The website is built using the Astro framework and uses React for interactivity.

- Initial modal pop-up from anywhere on the site.
- Full search page available with filtering options.



06 The Future

Where do we go from here?

Looking forward

A perfect world where improved search has transformed daily life. People enjoy seamless access to information in a harmonious, futuristic setting, blending nature with technology…

Nah, we'll probably just build a RAG.

The Aerospike Community

Join the conversation and become a part of our community!

Try the Aerospike Vector Database for free!

Aerospike

Thank you