SNIA | Cloud Storage Technologies

Ceph: Object Storage Meets Vector Search

Live Webinar
July 16, 2026
10:00 am PT / 1:00 pm ET

Christina Bassani
Seagate

Kyle Bader
IBM

Prashanth Rao
LanceDB

Today's Presenters

Christina Bassani

Moderator
Sr Director, Chief of Staff
Office of the CTO
Seagate

Kyle Bader

Chief Architect
Data and AI, Ceph
IBM Storage

Prashanth Rao

AI Engineer
LanceDB

SNIA Cloud Object Storage Plugfests sponsored by CST

SNIA | Cloud Storage Technologies

https://snia.org/groups/cst

Committed to the adoption, growth and standardization of **intelligent data storage usage** in cloud infrastructures

- One fee for SNIA Membership – Participate in ANY group!
- Join SNIA groups at https://www.snia.org/member_com

The SNIA Community

200
industry leading organizations

2,000
active contributing members

50,000
IT end users & storage pros worldwide

SNIA | CST A SNIA Community

SNIA Legal Notice

- The material contained in this presentation is copyrighted by SNIA unless otherwise noted.
- Member companies and individual members may use this material in presentations and literature under the following conditions:
  - Any slide or slides used must be reproduced in their entirety without modification
  - SNIA must be acknowledged as the source of any material used in the body of any document containing material from these presentations.
- This presentation is a project of SNIA.
- Neither the author nor the presenter is an attorney and nothing in this presentation is intended to be, or should be construed as legal advice or an opinion of counsel. If you need legal advice or a legal opinion please contact your attorney.
- The information presented herein represents the author's personal opinion and current understanding of the relevant issues involved. The author, the presenter, and the SNIA do not assume any responsibility or liability for damages arising out of any reliance on or use of this information.

  NO WARRANTIES, EXPRESS OR IMPLIED. USE AT YOUR OWN RISK.

Agenda

- Introduction to vector search and S3 Vectors
- What is Lance?
- LanceDB technical dive
- Ceph design goals and parameters
- Adding S3 Vector to Ceph object
- OSS library selection
- Hybrid search strategy

Audience Poll

- Rate your familiarity
  - I am new to nearest neighbor search and vectors databases
  - I am familiar with nearest neighbor search and vector databases
  - I am familiar with ANN and have used vector search before

What are Vectors and Vector Search

What is a vector?

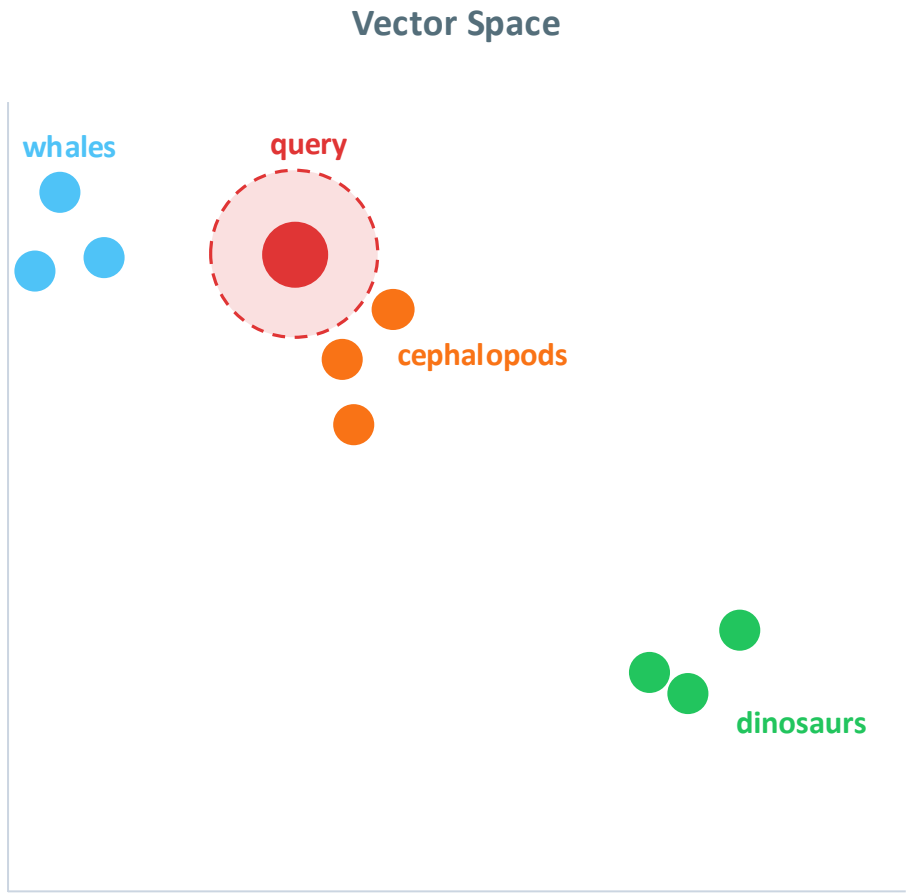
An array of float32 numbers encoding the semantic meaning of any object — text, image, audio — as a point in high-dimensional space.

How are they created?

Embedding models (OpenAI, Cohere, sentence-transformers) map raw data to fixed-length numeric arrays, placing semantically similar items close together.

Approximate Nearest Neighbor (ANN)

Finds the K most semantically similar items to a query vector — the engine behind semantic search, recommendations, and RAG pipelines.



Similar items cluster together in high-dimensional space

SNIA | CST A SNIA Community

What is S3 Vectors?

Vector Bucket

New bucket type. Only s3vectors:* actions are permitted. Supports SSE-S3 and SSE-KMS. Internally maps to a LanceDB dataset directory.

Index

A LanceDB table inside a bucket. Bound to a distance metric and dimension count at creation time via CreateIndex.

Vectors

Dense float32 arrays stored with string metadata tags. Up to 10 K/V pairs, 2 KB filterable, 40 KB total per vector.

Query

Top-K ANN search via QueryVectors. Optional metadata tag predicate narrows the candidate set before the ANN phase.

Bucket Directory Structure (hidden)

```
s3://foo/              ← Vector Bucket
├── bar/               ← Index 'bar'
│   ├── data/
│   ├── indices/
│   ├── _latest.manifest
│   └── schema.arrow
└── baz/               ← Index 'baz'
```

ARN Formats

```
arn:aws:s3vectors:region:ACCT:bucket/my-bucket

arn:aws:s3vectors:region:ACCT:bucket/my-bucket/index/my-index
```


Lance Format: On-Disk Data & Index + Versioning

Columnar + Multimodal Storage + Arrow-native

Vectors, metadata, text, and multimodal blobs live as columns, so queries can project only the data they need.

Versioning is part of the format itself

Writes create fragments and update manifests. Versions track table state without rewriting the whole dataset.

Random access + scans

The format is optimized for row-level random access while preserving high-throughput scan behavior.

Indexes beside data

Vector, full-text, and scalar indexes are stored with the dataset rather than in an external service.

Lance is the Open Lakehouse Format Layer

Open data layer

Lance powers the underlying storage layer: vectors, scalar columns, manifests, schema, versions, and indexes as inspectable disk-native data.

LanceDB Table Management layer

Tables, index creation, vector queries, and writes sit on top of a flexible, open lakehouse format instead of hiding data inside an opaque service with a proprietary format.

Fit for Ceph

Ceph remains the storage and control plane. Lance format serves provides vector-specific layout and retrieval mechanics.

Why it matters

The vector index is persisted data with lifecycle hooks, not a sidecar database Ceph must deploy and operate separately.

How Lance Becomes LanceDB

Two layers, the same source of truth

- LanceDB provides a higher-level API for vector indexing, search and table operations
- Lance format provides a lower-level API for the same functionality

Layer 02 • Database API + Catalog

LanceDB Tables

High-level API • Director namespaces • Simple query / index / write operations

vector ANN • full-text • scalar filters • merge_insert • compaction

→ Directory catalog maps database/table names onto Lance dataset directories

Retrieval & Data Management

Layer 01 • Format

Lance OSS format

Open source • columnar • object-store-native

Fragments • manifests • schema • indexes • fast scans & random access • versioning

→ Stores bytes, indexes and versions on object storage / Native SSD paths

Storage & Indexing

Same bytes, same table —Lakehouse layer over an open format

Why LanceDB – Technical Deep Dive

IVF-PQ Index

Composite of Inverted File Index (IVF) and Product Quantization (PQ). IVF divides the vector space into cells; PQ compresses vectors to reduce index size. Tunable nprobes per query — higher values increase recall, lower values reduce latency. Target: 100–800 ms.

SIMD Acceleration (AVX2)

K-means clustering during index generation is accelerated via AVX2 SIMD. Minimally requires AVX2 — relevant for modern x86 Ceph OSD nodes.

merge_insert for PutVectors

Atomic upsert — checks existence and inserts or updates in a single call. Avoids race conditions that would arise from separate read-then-write. A background optimize() call rebuilds the IVF-PQ index without blocking concurrent merge_insert or query calls.

GPU Acceleration Path

Index generation can be GPU-accelerated via PyTorch (accelerator='cuda'). Ceph gateway nodes typically lack accelerators, but this could change. Might need to look into native approach with cuVS, though.

Why the API Mapping Feels Natural

S3 Vectors concept

Vector bucket
Index
Vector
PutVectors
QueryVectors

LanceDB / Lance concept

- Catalog namespace -> table directory -> row -> merge_insert -> vector query

Less translation glue

Ceph does not need to invent a storage format, table lifecycle, query engine, and index maintenance path from scratch.

Schema fixed where S3 fixes it

Index creation binds dimension count and metric; LanceDB tables naturally carry that schema and index metadata.

Index lifecycle has a home

Manifests, data files, and index files persist through Ceph/RADOS, while LanceDB owns table and index operations.

Index Options and Query-Time Tuning

Distance metric

LanceDB supports l2, cosine, and dot for indexed vector search. Ceph's S3 Vectors path can focus on l2/cosine.

nprobes

Query-time recall/latency dial. More probes scan more partitions; fewer probes reduce work. Auto-tuning can be the default.

IVF-PQ

IVF partitions narrow candidates; PQ compresses vectors to reduce footprint and improve disk/cache behavior.

Refine + filters

refine_factor reranks extra candidates; filtered queries can adapt probes when too few rows survive a filter.

L2 vs Cosine “Distance” vs “Direction”

Both can rank nearest neighbors, but they have different ideas of “close”

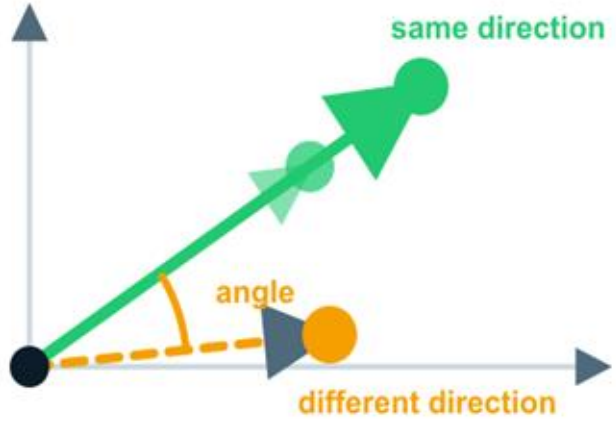
L2 distance

Measures straight-line distance between coordinates.

Good when "near" should mean small coordinate difference.

Cosine distance

Measures angle, so direction matters more than length.



Good when "similar" means same direction, even if lengths differ.

Practical rule of thumb:

- Start with the metric recommended by the embedding model provider
- For normalized vectors, L2 and Cosine tend to produce similar rankings

SNIA | CST A SNIA Community

IVF-PQ nprobes Tuning

nprobes controls how many ANN partitions are searched at query time.

IVF-PQ index

IVF splits vectors into partitions.

PQ stores compressed vectors.

partitions | PQ codes

query-time knob

nprobes

how many IVF partitions to search

lower	faster, less work
higher	more candidates, better recall

Candidate + refine path

1. Search selected partitions
2. Score compressed candidates
3. Optional refine
4. Return top-k

Although more probes improve recall; they still do not make ANN exact.

```
# Usually leave nprobes unset for auto-tuning
results = table.search(query_vector)
.nprobes(20) # manual recall/latency dial
.refine_factor(10) # rerank on full vectors
.limit(10).to_pandas()
```

Tuning rule of thumb

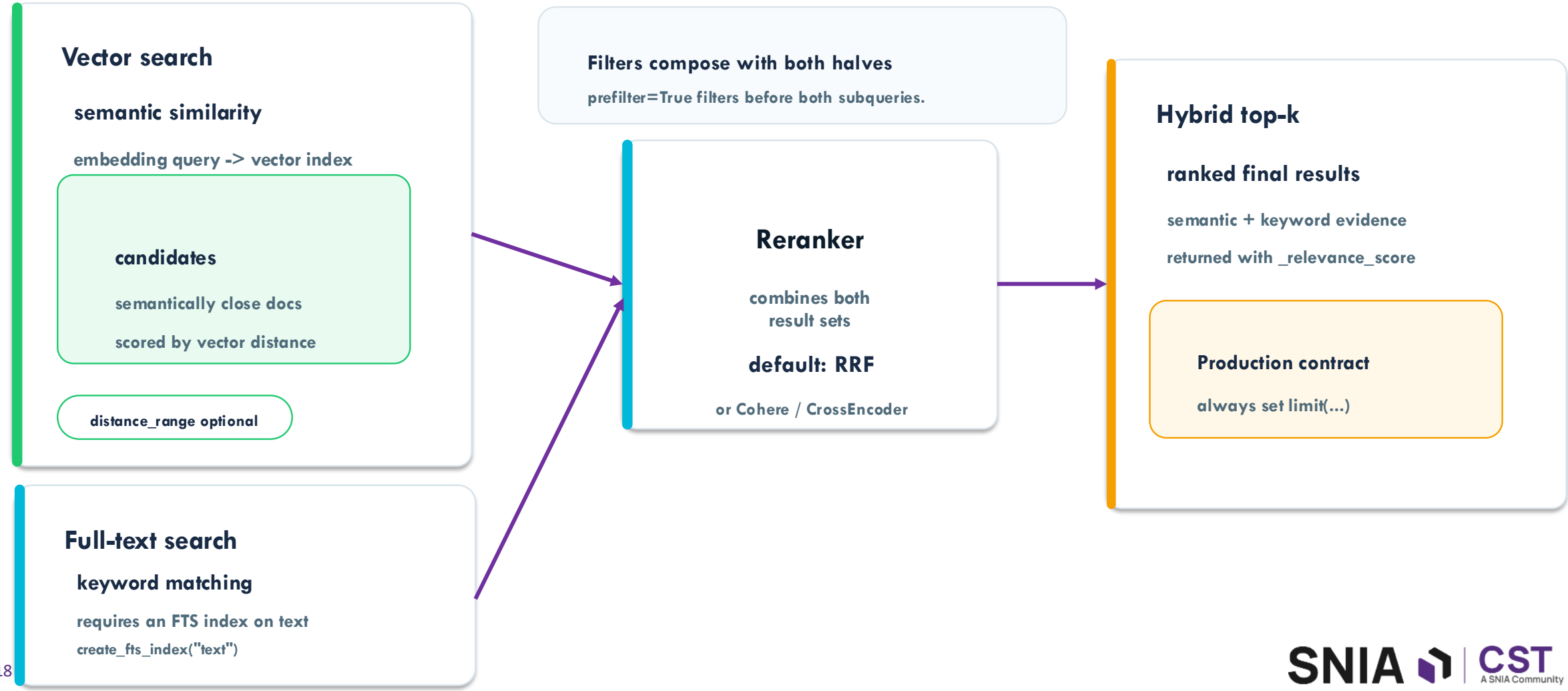
If recall is low, increase nprobes.

If latency matters, decrease it.

SNIA | CST A SNIA Community

Hybrid Search: Combining Vector Search + FTS

LanceDB combines semantic search based on vector similarity with keyword-based searches (FTS), then reranks the merged candidates



Ceph Design Goals

01

Cost-Effective Storage

Designed for billions of vectors. Target ≤1:3 ratio of raw vector data to total on-disk footprint (data + IVF-PQ indexes). Leverages Ceph's erasure coding and tiering.

02

Zero Extra Infrastructure

No new services to deploy or monitor. Vector capabilities live entirely inside the RGW request pipeline — if you run Ceph, you get vector search automatically.

03

Sub-Second ANN Search

Target query latency of 100–800 ms. Achieved via IVF-PQ on-disk indexes with tunable nprobes — trading recall precision for latency as needed.

Design Parameters

Parameter	Value	Notes
Indexes per vector bucket	**≤ 10,000**	*Each index independently queryable with its own metric and dimensions*
Vector dimensions	**1 – 4,096**	*Dense float32 only; sparse vectors are out of scope for v1*
Data type	**float32**	*Single-precision IEEE 754 — matches native embedding model output*
Top-K per query	**≤ 30**	*Maximum nearest neighbors returned per QueryVectors call*
Space amplification	**≤ 1 : 3**	*Raw vector data to total on-disk footprint including IVF-PQ index files*
PutVectors batch size	**500 / 20MB**	*Per call limit — whichever is smaller; max 5 calls/index/second*

Adding S3 Vectors to Ceph

1

New Resources & IAM Actions

- Vector bucket type
- Index resource type
- Vector CRUD ops
- IAM policy actions
- S3-compatible routing
- SSE-S3 / SSE-KMS

2

RGW Request Handling

- Parse & validate
- Enforce dimensions
- Check batch limits
- Rate limiting
- Dispatch to engine

3

LanceDB (Rust via FFI)

- Build IVF-PQ indexes
- Cosine & L2 distance
- merge_insert writes
- Background optimize
- Metadata filtering

4

RGW Storage Abstraction (SAL)

- Persist Lance datasets
- Inherit replication
- Leverage tiering & EC
- No new daemons
- Existing auth & quotas

OSS Library Assessment

FAISS

DISQUALIFIED

Meta Research · MIT License

Distance metrics

L2, IP, Cosine

Storage model

In-memory first

Language

C++ / Python

On-disk ANN

✗ Not primary use case

Integration effort

High — no built-in I/O

Verdict

Optimized for GPU RAM, not disk. Out of scope.

DiskANN

VIABLE

Microsoft · MIT License

Distance metrics

L2, Cosine, Inner Product

Storage model

Disk-native (SSD-optimized)

Language

C++

On-disk ANN

✓ Core design

Integration effort

High — no query engine, format, or I/O layer

Verdict

Good algorithm, but needs substantial glue code.

LanceDB

SELECTED

LanceDB Inc. · Apache 2.0

Distance metrics

L2, Cosine

Storage model

Lance columnar, disk-native

Language

Rust (FFI / C API)

On-disk ANN

✓ IVF-PQ, AVX2 SIMD

Integration effort

Low — full-stack library

Verdict

Best fit: metrics, storage, hybrid search, clean API.

Metadata Filtering Strategy

The Challenge

Different vectors within the same index can have different metadata keys. Flattening every key into its own column would produce an unbounded number of columns — impractical to pre-create.

LanceDB Scalar Index Types

- btree
- bitmap
- label_list (equality only after flattening, no range queries)

Frequency-based flattening

An accumulator in the PutVectors path tracks the most frequently seen filterable keys per index. Top-N keys are dynamically flattened into dedicated columns with scalar indexes.

Query-driven indexing

Monitor which keys are most frequently used as filters in QueryVectors calls. Proactively create scalar indexes on hot keys — prioritizes latency for real-world query patterns.

Post-filtering

Inflate top-k request size for ANN search and apply JSON filters to the result, returning top-k survivors. Maintains compatibility with S3 Vectors. Extension to provide filterableMetadataKeys during CreateIndex for optimized pre-filtering.

Examples

```
ubuntu@ip-172-31-12-186:~/s3-vectors-demo$ kubectl exec -n rook-ceph s3v-demo -- aws s3vectors create-vector-bucket --vector-bucket-name movies
{
    "vectorBucketArn": "arn:aws:s3vectors:::bucket/movies"
}
ubuntu@ip-172-31-12-186:~/s3-vectors-demo$ kubectl exec -n rook-ceph s3v-demo -- aws s3vectors create-index --vector-bucket-name movies --index-nam
e movies --data-type float32 --dimension 5 --distance-metric cosine
{
    "indexArn": "arn:aws:s3vectors:::bucket/movies/index/movies"
}
ubuntu@ip-172-31-12-186:~/s3-vectors-demo$ kubectl exec -n rook-ceph s3v-demo -- aws s3vectors put-vectors --vector-bucket-name movies --index-name
 movies --vectors '[
    {
      "key": "star-wars",
      "data": {"float32": [0.9, 0.1, 0.8, 0.2, 0.7]},
      "metadata": {"genre": "scifi", "title": "Star Wars", "year": 1977}
    },
    {
      "key": "blade-runner",
      "data": {"float32": [0.85, 0.15, 0.75, 0.3, 0.65]},
      "metadata": {"genre": "scifi", "title": "Blade Runner", "year": 1982}
    },
    {
      "key": "the-notebook",
      "data": {"float32": [0.1, 0.9, 0.2, 0.8, 0.15]},
      "metadata": {"genre": "romance", "title": "The Notebook", "year": 2004}
    },
    {
      "key": "titanic",
      "data": {"float32": [0.2, 0.85, 0.15, 0.75, 0.2]},
      "metadata": {"genre": "romance", "title": "Titanic", "year": 1997}
    },
    {
      "key": "alien",
      "data": {"float32": [0.88, 0.12, 0.82, 0.18, 0.72]},
      "metadata": {"genre": "scifi", "title": "Alien", "year": 1979}
    }
  ]'
```


Examples

```
ubuntu@ip-172-31-12-186:~/s3-vectors-demo$ kubectl exec -n rook-ceph s3v-demo -- aws s3vectors list-vectors --vector-bucket movies --index-name mov
ies
{
    "vectors": [
        {
            "key": "alien"
        },
        {
            "key": "blade-runner"
        },
        {
            "key": "star-wars"
        },
        {
            "key": "titanic"
        },
        {
            "key": "the-notebook"
        }
    ]
}
```


Examples

```
ubuntu@ip-172-31-12-186:~/s3-vectors-demo$ kubectl exec -n rook-ceph s3v-demo -- aws s3vectors get-vectors --vector-bucket movies --index-name movi
es --keys star-wars blade-runner --return-data --return-metadata
{
    "vectors": [
        {
            "key": "blade-runner",
            "data": {
                "float32": [
                    0.8500000238418579,
                    0.15000000596046448,
                    0.75,
                    0.30000001192092896,
                    0.6499999761581421
                ]
            }
        },
        {
            "key": "star-wars",
            "data": {
                "float32": [
                    0.8999999761581421,
                    0.10000000149011612,
                    0.800000011920929,
                    0.20000000298023224,
                    0.699999988079071
                ]
            }
        }
    ]
}
```


Key takeaways

- Object interfaces are fluid versus ossified like POSIX
- Object systems are extending into tabular data, catalogs, with indexes
- Hybrid systems can provide a batteries included experience
- There are real benefits of storing vector embeddings and indexes data adjacent.

Q&A

Thanks for Viewing this Webinar

- Please rate this webinar and provide us with feedback
- This webinar and a copy of the slides are available at the SNIA Educational Library
- A Q&A blog from this webinar will be posted at snia.org/blog
- Follow us on LinkedIn and X for future webinar dates

THANK YOU!

