OpenSearchCon EUROPE

Writeable Warm

Breaking the Read-Only Barrier in OpenSearch Tiering

Sandeep Kumawat
Software Engineer – AWS OpenSearch

Shubh Sahu
Software Engineer – AWS OpenSearch

Harshita Kaushik
Software Engineer – AWS OpenSearch

What We'll Cover

- **The Foundation**
  - Why data tiering matters
  - Remote-backed storage recap
  - The idea behind Writable Warm
- **The Architecture**
  - Composite Directory
  - Remote-backed storage recap
  - The idea behind Writable Warm
- **Making Warm Tier Efficient**
  - Staggered merges
  - Prefetch and read-ahead
- **Impact**
  - Use cases
  - Future directions and Q&A

Why Data Tiering Matters

OpenSearchCon EUROPE

Data volumes are growing exponentially

Retention requirements keep getting longer

Yet you are paying the **same cost** for both

But not all data is accessed equally

Recent data

actively searched & modified

Older data

maybe once a month

Why not we store older data in cheap storage?
while keeping it queryable AND writable?

Hot vs Warm Tiered Storage

OpenSearchCon EUROPE

- Complete data on local disk (i.e, ssd, ebs)
- Low read and write latencies
- Higher infrastructure cost

Hot Tier

- Minimal data on local disk
- Complete data stored densely on cheaper remote storage (i.e, S3)
- Lower infrastructure cost
- Slightly higher read and write latencies

Warm Tier

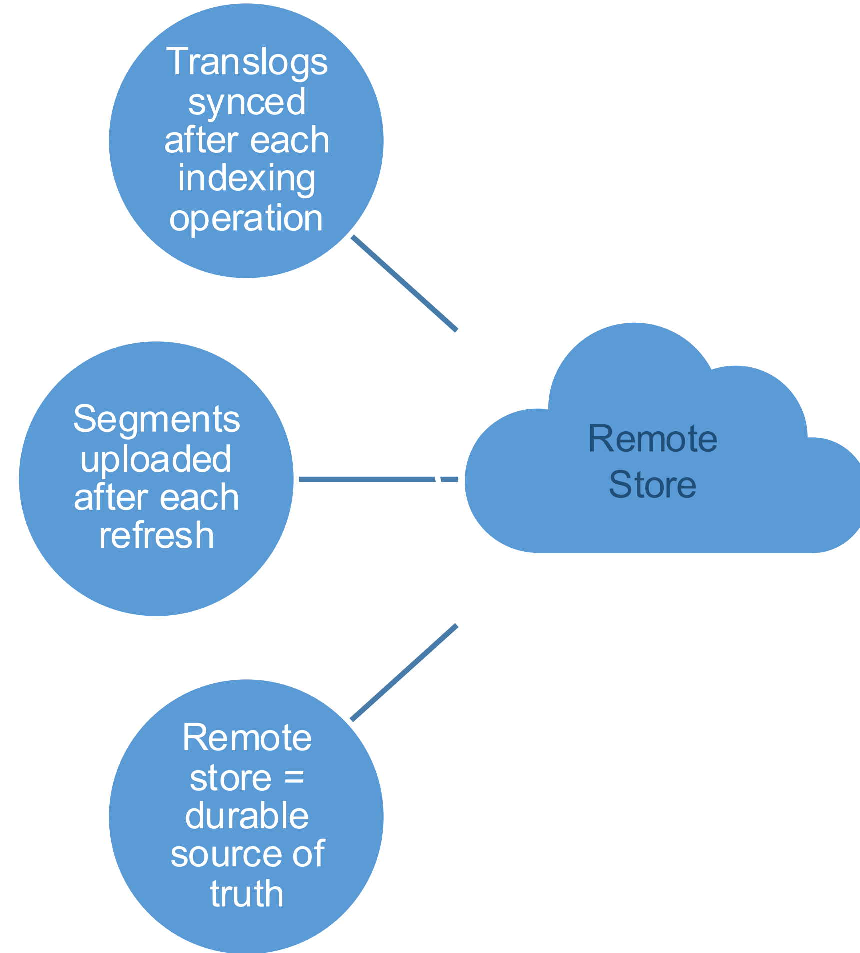
The Backbone : Remote Store

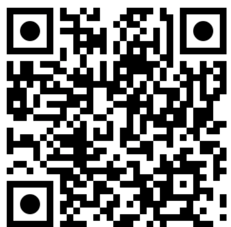
OpenSearch supports remote-backed storage for durability for a while now

Remote store can be any blob store: Amazon S3, Azure Blob Storage, Google Cloud Storage etc.

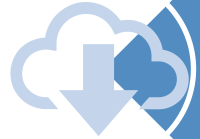
Remote storage always has all the data

Today every node also keeps 100% locally, but do we always need it ?



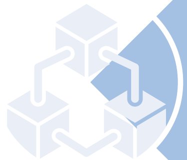


Introducing Writable Warm



What if a shard could operate without all data being present locally?
Keep essential data on disk. Fetch the rest from remote store on demand.

A warm index is an OpenSearch index where the entire index data is not guaranteed to be present locally. The shards are always open and assigned, the index is active



Writable Warm = partial local data + on-demand remote fetch + full write support

The Architecture

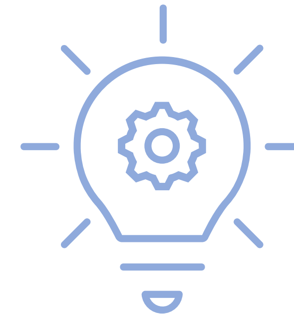
Building blocks of writable warm

Multiple Store Interaction ?

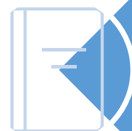
- Lucene expects a local directory.
- Warm tier: data is split between local and remote.

Local Store Interaction

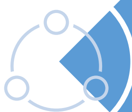
Remote Store Interaction



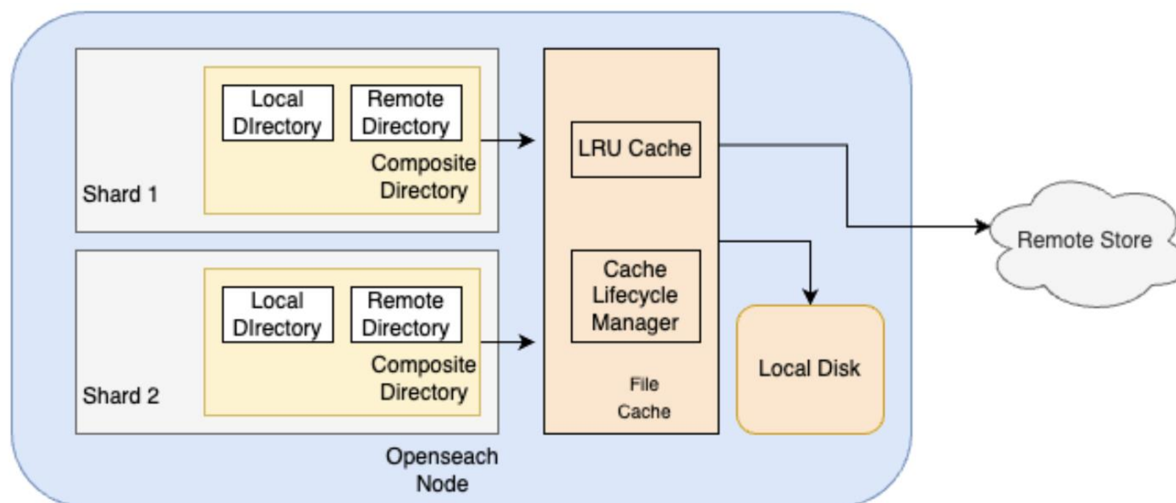
Composite Directory



A single directory abstraction over local + remote storage.

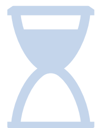


Routes reads/writes to the right store



RFC: github.com/opensearch-project/OpenSearch/issues/12781

Composite Directory



But… local disk fills up on the warm tier as writes continue.

- New files cached as full files in File Cache
- Readers hold references → no eviction
- Disk usage keeps growing

How do we solve this?

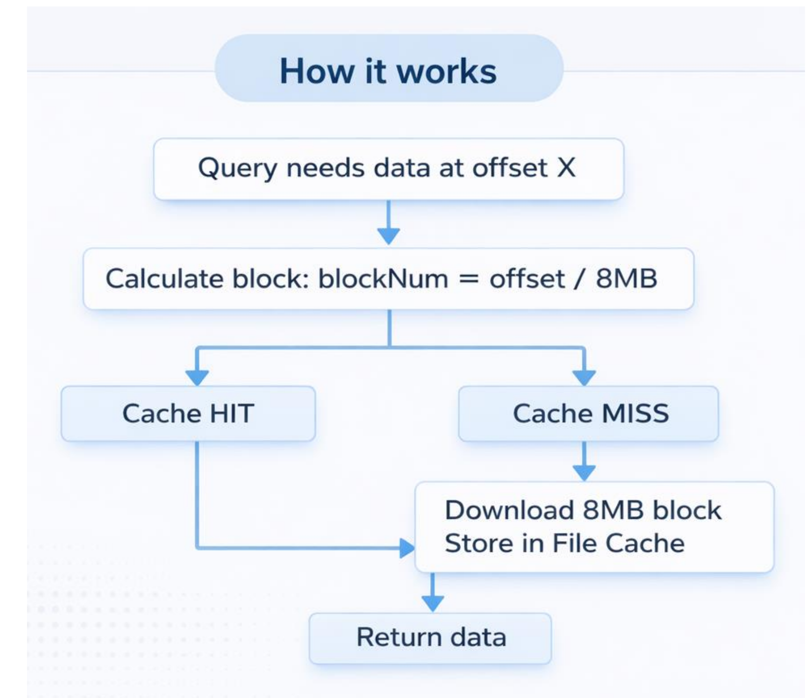
- Move to block-based access (no full file reads)
- Evict full files after upload to remote store

Block Index Input

OpenSearchCon EUROPE

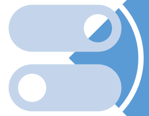
Stop downloading full files. Start fetching blocks.

- Segment files can be of up to a few GB scale
- Downloading full files for every read is impractical
- Fetch data in small blocks (8MB)



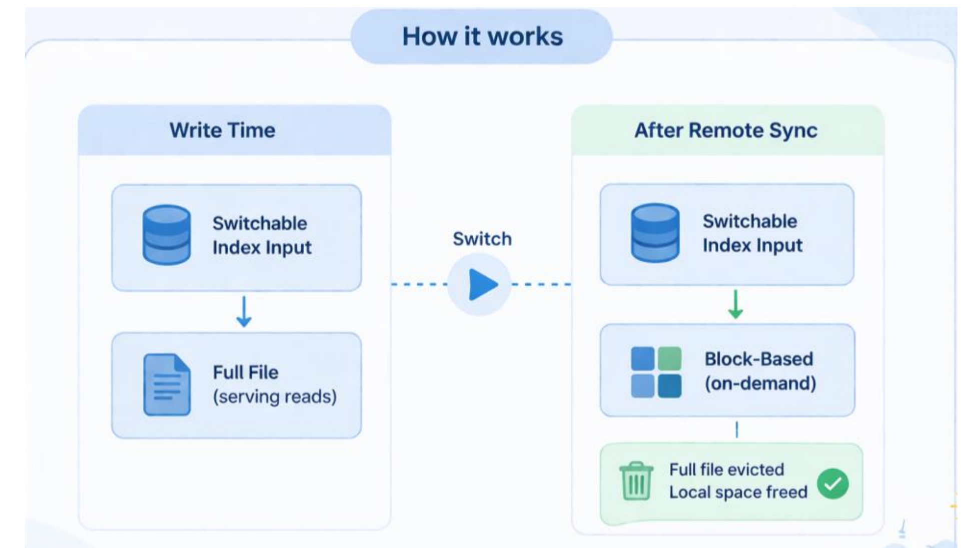
Switchable Index Input

OpenSearchCon EUROPE



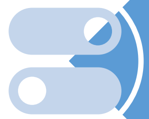
Switch from full files → block-based access without disruption

1. Prepare block-based input in background
2. Atomic switch to block-based access
3. Full file evicted → space freed

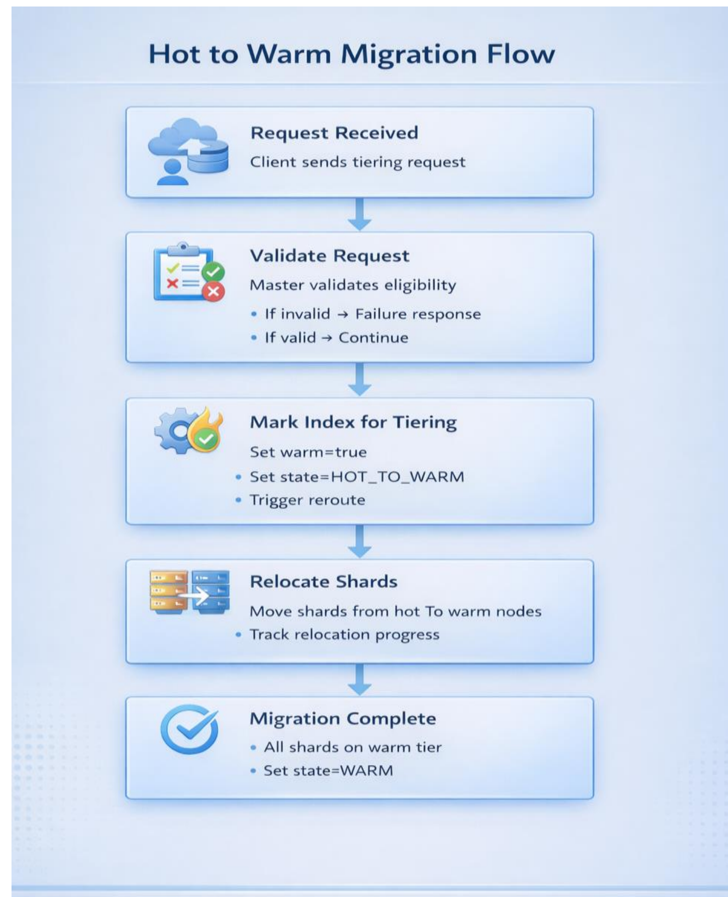


Migration Flow

OpenSearchCon EUROPE



Migration is just shard relocation - no heavy operations



Making Warm Tier Efficient

Search and Memory Optimizations

The Challenges

OpenSearchCon EUROPE

- Dense shard packaging
- File fetch from remote

Dense Shard Packaging

OpenSearchCon EUROPE

Why?

- On remote-backed indices, segments accumulate over time
- **Memory footprint :** n-segments = ~703KB per shard vs 1-segment = ~67KB. That's **90%** more memory with unmerged segments
- **GC pressure** : n-segments causes 92% more young GC collections and 182% more GC time.

Solution

- **Segment Merges**
- Lucene's background merges → might not reach desired segment count
- Force merges → **CPU/IO** resource intensive, disrupt production workloads, and dominate migration costs
- Need for more intelligent background merges

Staggered Merges

OpenSearchCon EUROPE

Pre-migration

- **Opportunistic merges** during low cluster load.
- **Triggered periodically** based on **node capacity & index characteristics** (not all at once).
- Reduces segment count *without blocking migration*.

During migration

- Instant shard relocation → zero downtime**.**
- No force merges.

Post migration

- **Continuous merges** on warm tier.
- Optimizes segments **long-term** for ongoing writes.
- Maintains **memory efficiency** + **query performance**.

Merges get triggered based on :

- Configurable CPU threshold (e.g., <70%)
- Idle Ingestion traffic for X minutes (e.g., 30m)
- ISM policy hints : warm migration configured
- Scheduled Off-peak windows (e.g., 10PM for 10 hours)

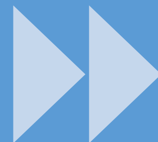
Prefetch and Read-Ahead

OpenSearchCon EUROPE

Cache misses hit remote storage latency. Prefetch stays ahead of the query.

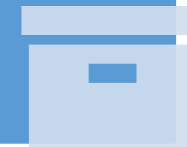
- Optimizes aggregation queries (doc values)
- When block B is fetched, async prefetch B+1 to B+N
- Exploits sequential access patterns
- Subsequent reads served from cache

Read-Ahead Prefetch

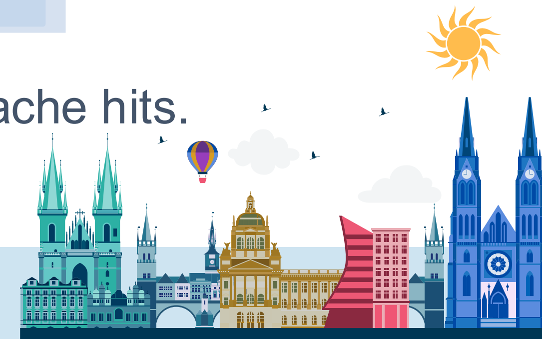


- Optimizes search result retrieval (fetch phase)
- Identifies which blocks contain stored fields for matching docs
- Downloads them before the fetch phase begins
- Fetch phase reads entirely from cache

Stored Fields Prefetch



Both mechanisms download blocks before they're needed, turning cache misses into cache hits.



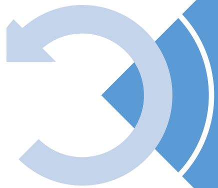
Real World Impact

Use Cases and Future

When to Use Writable Warm

If you have time-series data with long retention and your older indices are eating storage costs, Writable Warm is built for you!

Log analytics, observability, metrics with **high ingestion and long retention**



Late-arriving data that needs to land in older indices

Search latency on older data is **not critical**

Future Directions

OpenSearchCon EUROPE

Seamless Cold Tier	• Extend the warm model to cold/archival storage. Implicit attach/detach, data flows down automatically
Tiered Hot Indices	• Bring partial data locality to hot tier. Configure what percentage of an index stays local vs remote. Same building blocks: Composite Directory, Block Index Input, File Cache
Auto-Tiering	• Policy-driven tier transitions based on access patterns. No manual intervention.

We'd love to hear which of these matters most to you. Come talk to us!

Key Takeaways

Write Support on Warm Tier	• Append, update, delete. Background merges continue. No more read-only constraint
Built on Remote Storage	• Leverages existing remote-backed storage for fast tier transitions in seconds
Operational Simplicity	• Migration = shard relocation. Automatic failure recovery. Prefetch masks remote latency.
Cost Efficiency	• Higher storage density. Dramatically faster migrations with minimal CPU impact

Open Source and Community

OpenSearchCon EUROPE

Built in the open, for the OpenSearch community

What	Where
Writable Warm RFC	github.com/opensearch-project/OpenSearch/issues/12809
META tracking	github.com/opensearch-project/OpenSearch/issues/13149
Composite Directory	github.com/opensearch-project/OpenSearch/issues/12781
Block-Level Fetch	github.com/opensearch-project/OpenSearch/issues/9987
Remote Storage RFC	github.com/opensearch-project/OpenSearch/issues/2700
Tiering Lifecycle	github.com/opensearch-project/OpenSearch/issues/21016

Get Involved

Review the RFC · Share your use cases · Contribute code · Join the discussion

OpenSearchCon
EUROPE