OpenSearch Offline Batch Ingestion at Uber

Uber

About Us

Monika Agarwal

Staff Software Engineer @ Uber

Tarun Kishore

Software Engineer II @ Uber

Agenda

1. Search @ Uber: platform evolution
2. Why offline ingestion is needed at Uber
3. Architecture for OpenSearch offline ingestion
4. Technical Challenges and Benefits
5. Uses of OpenSearch Offline Ingestion @ Uber

Search @ Uber

Search powers critical flows: rider matching, Uber Eats, restaurant search, and Pindrop search. The platform evolved as scale, latency, and reliability needs grew.

1

Elasticsearch

Started with Elasticsearch for search infrastructure and early platform needs.

2

In-house engine

Built purpose-specific search capabilities as scale and reliability requirements increased.

3

Lucene NRT engine

Built another in-house engine on Lucene near-real-time indexing.

4

OpenSearch

Adopting OpenSearch to bring standard APIs, ecosystem benefits, and open-source collaboration.

Current focus: move Uber search use cases to OpenSearch while carrying forward learnings from in-house systems.

Why offline ingestion is needed at Uber

Offline ingestion

Build the base search index outside the OpenSearch cluster, restore it, then apply live updates when needed.

Uber search use cases generally fall into 3 categories:

Batch-only

Complete rebuild

- Full index refreshed in batch
- No continuous update stream required
- Useful for periodic or full rebuilds

Best fit: offline ingestion

Hybrid

Historical base + live updates

- TB-scale historical index
- Kafka updates keep it fresh
- Most common large-scale shape

Best fit: offline + live ingestion

Live-only

Continuous updates only

- No large historical bootstrap
- Freshness driven by Kafka
- Direct pull-based ingestion path

Best fit: Kafka / API ingestion

Offline ingestion isolates bulk ingestion from serving

Use the right ingestion path for each workload: live-only through Kafka, and batch/hybrid through an offline base-index build.

Why not use Kafka/API for every base build

- Full corpus replay can be slow and operationally expensive
- Bulk API indexing competes with queries on serving clusters
- Large backfills create merge pressure and latency risk

Offline ingestion path

- Build Lucene/OpenSearch-compatible artifacts outside serving clusters
- Generate snapshot metadata and restore into OpenSearch
- For hybrid workloads, Kafka live ingestion continues on top

Build the base index offline. Restore it into OpenSearch.
Keep it fresh with streaming ingestion.

Architecture Overview

Build

Distributed offline index construction

Merge

Shard merge and snapshot generation

Restore

Snapshot restore into production clusters

Reverse Engineering OpenSearch Internals

Major Technical Challenges

- Reproducing InternalEngine behavior outside runtime clusters
- Shard indexing lifecycle
- Lucene segment handling
- Snapshot metadata generation

Why Difficult

- Required APIs are internal
- Behavior tied to OpenSearch cluster runtime
- No public interfaces for offline segment creation

Distributed Indexing

Spark-Based Parallel Processing

Documents partitioned across:

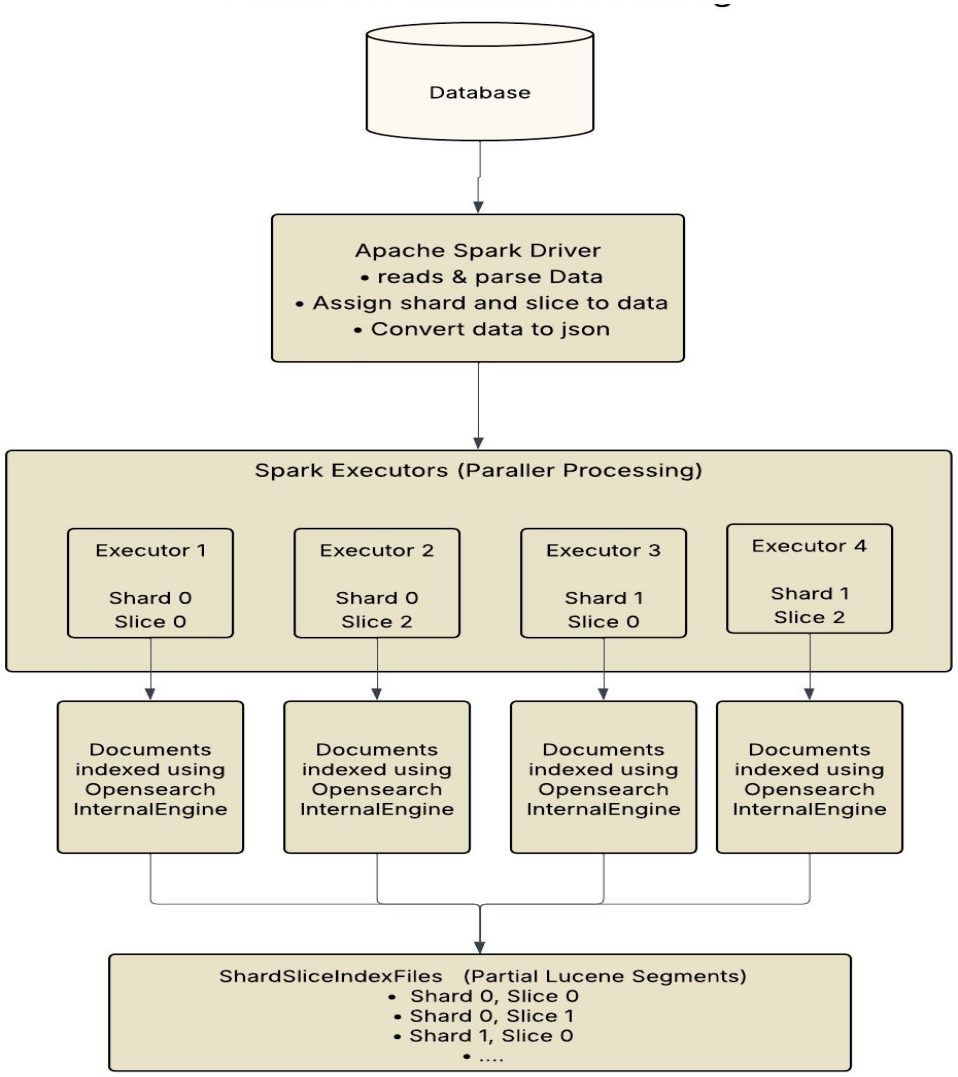
- Shards
- Slices per shard

Executors Perform

- Document parsing
- Analyzer execution
- Segment creation
- Force merge operations

Benefits

- Horizontal scalability
- Reduced cluster indexing pressure



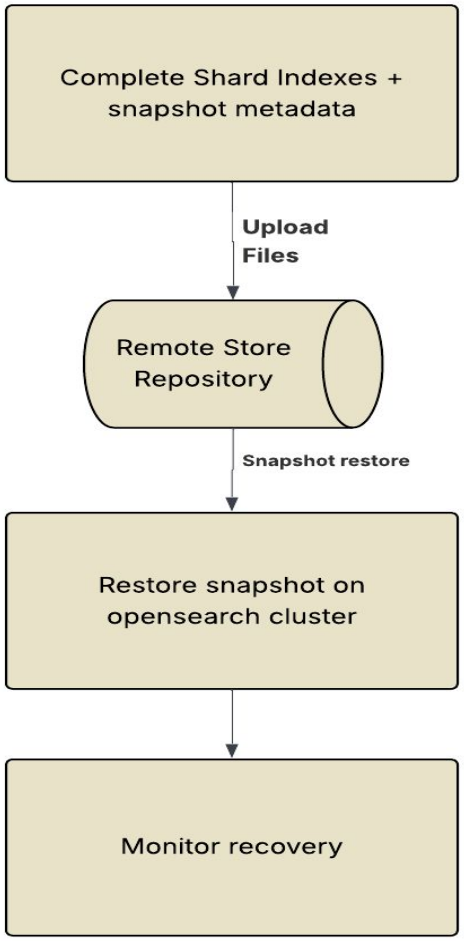
Snapshot Generation Pipeline

Offline Snapshot Creation

- Generate shard indexes offline
- Build OpenSearch snapshot metadata
- Upload repository-compatible snapshots

Result

Standard OpenSearch restore APIs can be used directly.



Ensuring OpenSearch Compatibility

Compatibility Requirements

- Lucene segment compatibility
- Mapping correctness
- Analyzer consistency
- Snapshot structure correctness

Validation Areas

- Restore behavior
- Shard recovery
- Search correctness
- Replication consistency

Testing Approach

- End-to-end restore tests
- Segment-level validation
- Query result comparison

Deployment Flow

Serving clusters restore indexes instead of building them — atomic, fast, and minimally disruptive.

1. **Upload Snapshot**
   Push generated shard indexes to remote snapshot repository
2. **Register Repository**
   Register or open the snapshot repository in OpenSearch
3. **Restore Index**
   Use standard OpenSearch restore APIs — atomic and fast
4. **Validate & Serve**
   Serve immediately; atomic rollback available if needed

Scalability Benefits

Number of Documents	Offline Ingestion (multi-cluster)	Online Ingestion (multi-cluster)
1M	~ 2 minutes	30 minutes
10M	~ 16 minutes	24 minutes
400M	~ 4 hours	4.6 hours

Key Advantages

- Spark distributed indexing scales horizontally
- Throughput scales linearly with executors
- Production clusters focus on serving traffic
- Eliminates online indexing bottlenecks
- Supports billion-scale document ingestion

Operational Impact at Uber

Platform Benefits

- Same snapshot across multiple clusters
- Preprod/shadow validation before rollout
- Simplified rebuild workflows
- Improved disaster recovery

Infrastructure Benefits

- Significantly reduced cluster indexing load
- Production clusters serve traffic during rebuilds
- Atomic deployments with instant rollback
- 2.5B doc index was built and deployed to 2 clusters in 6 hours.

Current Scope & Future Direction

Current Strengths

- Large-scale batch indexing
- Pull-based ingestion workflows
- Native OpenSearch compatibility
- Multi-cluster replication

Potential Future Work

- Plugin support (k-NN / vector indexing)
- Reduced version coupling
- Real-time hybrid ingestion
- Broader cluster type support

Key Takeaways

Safer Backfills

Offline indexing isolates rebuild workload from serving

Solves Pull Ingestion

Purpose-built for pull-based pipeline ingestion

Reduces Cluster Load

Eliminates merge pressure and online indexing overhead

Scalable with Spark

Spark executors scale horizontally with document volume

Native Compatibility

Standard OpenSearch restore APIs — no custom tooling

Multi-Cluster Ready

Same snapshot validated across pre-prod and production

Thank You

Uber