

# OpenSearch 2025年 - 2026年

プロジェクト・コミュニティの成長とソフトウェアの進歩

Amazon Web Services Japan G.K.  
Analytics Solutions Architect  
Takayuki Enomoto



# アジェンダ



2025年の  
リリースハイライト



コミュニティ活動の  
振り返り



2026年の  
ロードマップ

# 2025年のハイライト

# Happy Birthday



# OpenSearch



# OpenSearch Project Overview

↓  
**1.3B+**  
総ダウンロード数

□  
**1M+**  
ページビュー

□  
**100+**  
ソリューションプロバイダー

□  
**140+**  
リポジトリ

□  
**3.3K+**  
コントリビューター

□  
**400+**  
参加組織

□  
**7K+**  
フォーラムメンバー

□  
**4K+**  
Slack メンバー

□  
**8週間**  
リリースサイクル

# 2025年リリースと主要アップデート

## リリースタイムライン

1月	2月 2.19	3月
4月	5月 3.0★	6月 3.1
7月	8月 3.2	9月
10月 3.3	11月	12月 3.4

## 主要アップデート領域

### コアエンジン最適化

11.3x 高速化

### コスト効率の改善

ストレージ 58% 削減、ワークロード分離、クエリモニタリング

### 検索の改善

エージェント検索、メモリ最適化ベクトル検索

### 精度・ランキング改善

リスクアーリングの追加、量子化ベクトルの検索精度改善、検索品質評価

### AI/エージェント対応

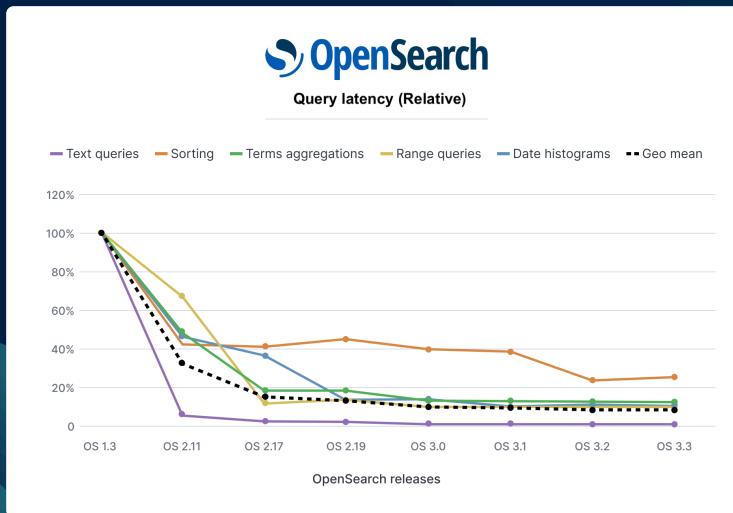
エージェント実装、MCP対応、エージェントメモリ

### Observability

統合UI + PPL刷新

# コアエンジン最適化: 11.3x 高速化を支える技術

## レイテンシ推移 (幾何平均)



OS 1.3 → 3.3: 159ms → 14ms

## 要素技術 (クリックで詳細)

[gRPC/Protobuf](#)

[Concurrent Segment Search](#)

[Approximation Framework](#)

[Range Query](#)

[Skip List](#)

[Star-tree Index](#)

[Streaming Aggregation](#)

[Lucene 10 / JDK 21+](#)

# Approximation Framework: 早期終了による高速化

## 従来の問題と解決策

Lucene は全セグメントを走査し必要以上のドキュメントを処理

→ BKDツリーを活用し、必要件数で走査を停止

**Range:** BKDツリーを走査、size件で停止

**Sort:** match\_all+sortをRange化  
ASC→左から / DESC→右から走査

**search\_after:** 指定位置からのRangeに変換  
前ページの走査をスキップ

## ベンチマーク結果

クエリ種別	改善
Range	90%高速化
Sort (timestamp)	75%高速化
search_after	50x高速化

具体例:

search\_after: 185ms → **8ms**  
desc\_sort: 2,111ms → **6ms**

v3.0で導入、v3.2で全数値型対応

# Skip List: フィルタ付き集計を96%高速化

## 仕組み

Doc Valuesのブロック毎にmin/maxを記録

Block min/max index:

1-100

101-200 ✓

201-300

→ 条件外ブロックをスキップ

## ベンチマーク結果

クエリ

改善

Date Histogram + Filter

96%高速化

+ Metrics集計

46%高速化

## 適用条件

- v3.3で **@timestamp** にデフォルト有効
- v3.2で全数値フィールド対応
- Range Filter + 集計全般で効果

116M HTTPログでのベンチマーク

# Star-tree Index: 集計最大100倍高速化

## 事前集計による高速化

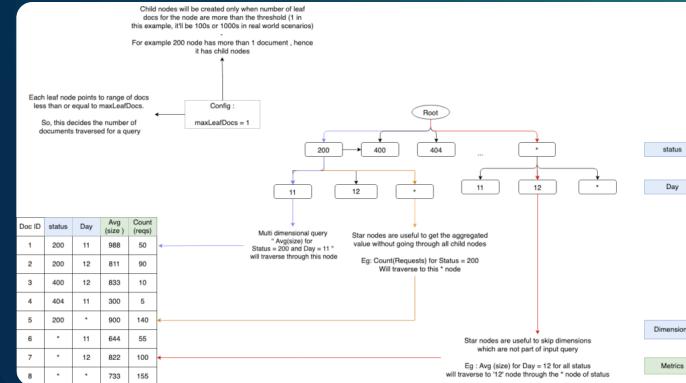
クエリ	通常	Star-tree
status=200 (2億件)	4.2秒	<b>6.3ms</b>
status=400 (3千件)	5ms	<b>6.5ms</b>

## 効果

- クエリ処理量: **100分の1**
- 予測可能なレイテンシ
- 高カーディナリティで特に効果大

## 適用条件

- 追記オンリーのインデックス (更新・削除なし)
- ログ・メトリクス等の時系列データ向け



ディメンション値の組み合わせごとに事前集計

# コスト効率・運用: 主要技術

## ストレージ使用量の削減

- Derived Sourceにより、\_sourceフィールドを保存せず必要時に再構築することでストレージを削減。58%削減、マージも20-48%高速化

## ワークロード管理

- Workload Managementによりリソース使用量の制限・配分を実現
- インデックス、リクエスト種別、ユーザーおよびロール単位でルールを適用することが可能

## クエリ性能分析

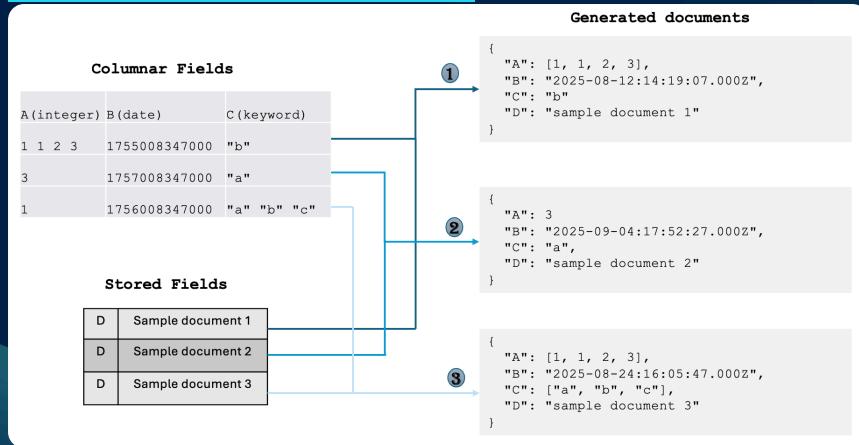
- Query Insights により Top-N クエリの遅延・リソース消費をリアルタイムで可視化し、ボトルネックを特定
- Live queries dashboards により実行中の long-running query の特定も容易に

## 検索・書込の分離

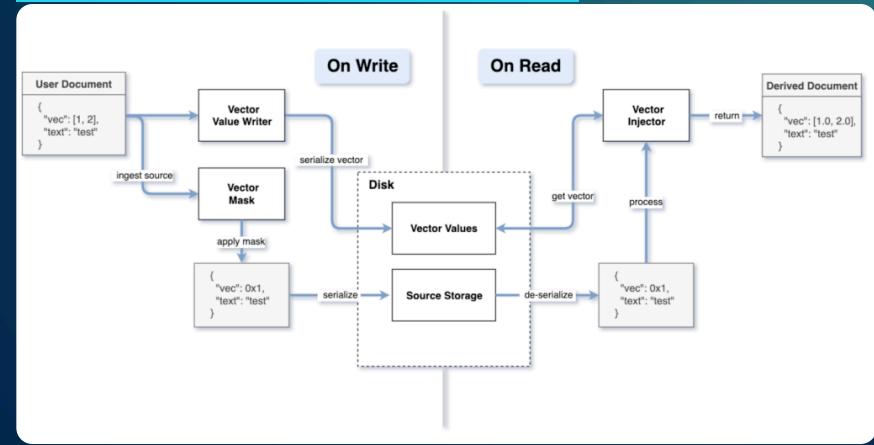
- 検索・書込ノードを分離し、ワークロード特性に応じて異なるハードウェアの割り当てやスケーリングが可能に
- read-only レプリカと write レプリカの概念を追加することで、時系列データの世代管理がより容易に

# Derived Source: ストレージコスト削減 (3.0/3.2)

## 一般フィールド (3.2)



## ベクトルフィールド (3.0)



`_source` を保存せず `doc_values` 等から動的に再構築

- ストレージ **41-58%削減** / Merge **20-48%高速化**

ベクトルを1バイトプレースホルダーに置換

- ストレージ **3倍削減** / 3.0以降デフォルト有効

# Workload Management: リソースの制限・配分 (2.18)

## Dashboards UI

The screenshot shows the 'Workload groups' section of the OpenSearch Data Administration interface. It displays the following information:

- Total workload groups: 1
- Total groups exceeding limits: 0
- Last updated: 8/7/2025 @ 2:06:29 AM
- Search bar: Search workload groups
- Refresh button
- Table headers: Workload group name, CPU usage, Memory usage, Total completions, Total rejections, Total cancellations, Top N Queries
- Data row: DEFAULT\_WORKLOAD\_GROUP (CPU usage: 0%, Memory usage: 0%, Total completions: 64, Total rejections: 0, Total cancellations: 0)
- Page settings: Rows per page: 10

CPU / メモリ使用量をワークロードグループ別に可視化

## ルール適用単位

- インデックスパターン（例: logs-\*）
- リクエスト種別 / ユーザー / ロール

## 制御可能なリソース

- CPU 使用率 / メモリ使用量
- 閾値超過時に reject または cancel

## 効果

重いクエリが他のワークロードに影響しない

# Query Insights: クエリ性能の可視化 (2.12)

## Top-N Queries

The screenshot shows the 'Query insights - Top N queries' section of the OpenSearch Dashboards. The table displays the following columns: ID, Type, Timestamp, Latency, CPU Time, Memory Usage, Index, Search Type, Coordinator Node ID, and Total Shards. The table lists several queries, each with its timestamp, latency, CPU time, memory usage, index, search type, coordinator node ID, and total shards. The rows are numbered 1 through 10. The interface includes a navigation bar with tabs for 'Live queries', 'Top N queries' (selected), and 'Configuration'. It also features filters for 'Type', 'Indices', 'Search Type', 'Coordinator Node ID', and buttons for 'Load Test', 'Show Errors', and 'Refresh'.

ID	Type	Timestamp	Latency	CPU Time	Memory Usage	Index	Search Type	Coordinator Node ID	Total Shards	
1	query	Oct 12, 2025 @ 2:41:42 PM	8386.00 ms	43.68 ms	80744.00 B	allbase	query then fetch	1 <td>2</td> <td>1</td>	2	1
2	query	Oct 12, 2025 @ 3:40:00 PM	3010.00 ms	1.56 ms	80744.00 B	my_index	query then fetch	1 <td>2</td> <td>1</td>	2	1
3	query	Oct 12, 2025 @ 2:39:54 PM	8018.00 ms	3.70 ms	80744.00 B	my_index	query then fetch	1 <td>2</td> <td>1</td>	2	1
4	query	Oct 12, 2025 @ 3:40:14 PM	3017.00 ms	1.58 ms	80744.00 B	my_index	query then fetch	1 <td>2</td> <td>1</td>	2	1
5	query	Oct 12, 2025 @ 3:40:21 PM	3016.00 ms	2.47 ms	80744.00 B	my_index	query then fetch	1 <td>2</td> <td>1</td>	2	1
6	query	Oct 12, 2025 @ 3:40:31 PM	3016.00 ms	2.32 ms	80744.00 B	my_index	query then fetch	1 <td>2</td> <td>1</td>	2	1
7	query	Oct 12, 2025 @ 3:40:31 PM	3016.00 ms	2.37 ms	80744.00 B	my_index	query then fetch	1 <td>2</td> <td>1</td>	2	1
8	query	Oct 12, 2025 @ 3:40:34 PM	3015.00 ms	2.30 ms	80744.00 B	my_index	query then fetch	1 <td>2</td> <td>1</td>	2	1
9	query	Oct 12, 2025 @ 3:40:34 PM	3015.00 ms	2.35 ms	80744.00 B	my_index	query then fetch	1 <td>2</td> <td>1</td>	2	1
10	query	Oct 12, 2025 @ 3:40:34 PM	3015.00 ms	2.55 ms	80744.00 B	my_index	query then fetch	1 <td>2</td> <td>1</td>	2	1

レイテンシ / CPU / メモリ消費の上位クエリを特定

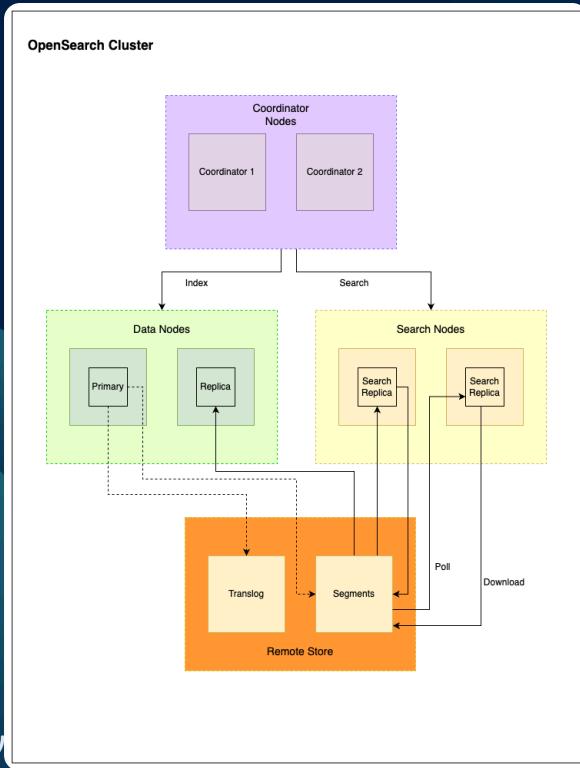
## 主な機能

- **Top-N 分析:** 重いクエリをランキング表示
- **グルーピング:** 類似クエリをまとめて分析
- **履歴エクスポート:** ローカルインデックスに保存

## Live Queries (3.1)

- 実行中クエリのリアルタイム監視
- Long-running クエリの即座の特定

# Reader/Writer 分離: 柔軟なスケーリング (3.0 experimental)



## メリット

- 独立スケーリング: 検索負荷に応じて Search ノードだけ増減
- コスト最適化: Writer は高 I/O、Reader は高メモリ
- 障害分離: Writer 障害が検索に影響しない

## ベンチマーク結果

- クエリレイテンシ **34%** 改善
- コスト **8%** 削減

# 検索: 新機能・機能改善

## 新機能

- **Agentic Search:** 自然言語から最適クエリを自動生成
- **Semantic Field/Highlighting:** パイプライン不要のセマンティック検索
- **BM25F:** 複数フィールドのスコアリング改善
- **マルチベクトル対応:** Late Interaction で精度向上
- **Template Query:** プレスホールダ変数対応

## 機能改善

- **Hybrid Query:** Collapse / Pagination / Score Explanation
- **ML Inference Extension:** クエリ外の外部パラメーターも ML モデルに渡せる
- **Terms Lookup by Query:** 別インデックスへのクエリ結果で動的フィルタ（サブクエリ的）

# 検索: 精度・ランキング / 性能

## 精度・ランキング

- **RRF / Z-score 正規化:** ハイブリッド検索のスコア統合改善
- **OpenSearch Asymmetric Distance:** クエリ精度を維持した量子化
- **Random Rotation:** 量子化時の情報損失を軽減
- **Maximal Marginal Relevance:** 結果の多様性確保
- **Learning to Rank:** ML モデルによるリストアーリング
- **Search Relevance Workbench:** 検索品質の評価・比較ツール

## 検索性能

- **ベクトル検索**
  - Lucene-on-Faiss: スループット 2x
  - NVIDIA cuVS: インデックス構築 9x 高速化
- **スパース検索**
  - Seismic: 最大 100x 高速化
- **ハイブリッド検索**
  - Bulk Scorer: レイテンシ 85% 削減
- **ML 推論**
  - Skip Existing: 未変更入力スキップ、コスト 70% 削減

# Agentic Search: 自然言語からクエリを生成 (3.2+)

Configure agent EXPERIMENTAL

Agent My conversational agent

Name My conversational agent

Description Enter description

Type Conversational

Model Bedrock Claude 4.5

Tools

- > Query Planning - required
- > Search Index
- > List Index
- > Index Mapping
- > Web Search

MCP servers

+ Add MCP server

> Advanced settings

Learn more

Enable

Enable

Enable

Enable

Enable

Enable

Learn more

エージェント設定画面 ( Dashboards > AI Search Flows )

## Agentic Query

```
GET /products/_search?search_pipeline=agentic-pipeline
{ "query": { "agentic": {
    "query_text": "赤い靴で5000円以下" }}}}
```

## 変換後の Query DSL

```
{ "query": { "bool": { "must": [
    { "match": { "color": "red" }},
    { "match": { "category": "shoes" }},
    { "range": { "price": { "lte": 5000 }}}]}}}
```

# Semantic Field: セマンティック検索をより簡単に (3.3+)

## 特徴

- semantic フィールドに model\_id を指定するだけ
- Ingest Pipeline / Search Pipeline 不要
- knn\_vector / rank\_features が自動生成される
- Dense / Sparse 両モデルに対応

## 考慮事項

- モデル差し替え時はインデックスの再作成が必要  
(従来の Neural Search はパイプライン変更のみでモデル更新可能)

```
// マッピング定義
"content": {
  "type": "semantic",
  "model_id": "my-model"
}
```

```
// 自動生成されるフィールド
"content_semantic_info": {
  "embedding": {
    "type": "knn_vector",
    "dimension": 384,
    "method": { "engine": "faiss" }
  },
  "model": { "id": "...", "name": "..." }
}
```

# Terms Lookup by Query: 動的フィルタリング (3.3)

## 概要

別インデックスへのクエリ結果を使って動的にフィルタ (サブクエリ的な機能)

## ユースケース

- ユーザーの購入履歴から関連商品を除外
- 権限テーブルに基づくドキュメントフィルタ
- リアルタイムの在庫状況でフィルタ

## クエリ例

```
GET products/_search
{
  "query": {
    "terms": {
      "product_id": {
        "index": "user_purchases",
        "query": {
          "term": { "user_id": "user123" }
        },
        "path": "purchased_product_ids"
      }
    }
  }
}
```

# Maximal Marginal Relevance (MMR): 結果の多様性確保 (3.2)

## 特徴

- 関連性と多様性のバランスを自動調整
- 冗長な結果を排除し、情報の網羅性を向上
  - 例: 「Python入門」で検索 → 同じ本の異なる版ではなく、様々な入門書を返す

## MMRの仕組み

$$\text{MMR} = (1 - \lambda) \times \text{関連性} - \lambda \times \text{既選択との類似度}$$

- 関連性: クエリとの類似度
- 多様性: 既に選択された結果との差異
- $\lambda$  (**diversity**): 1に近いほど多様性重視

## 設定例

```
{  
  "query": { "knn": { "my_vector": {  
    "vector": [0.12, 0.54, 0.91],  
    "k": 10 } } },  
  "ext": { "mmr": { "diversity": 0.7 } }  
}
```

## 効果

- RAG での情報の網羅性向上
- 商品検索での多様な選択肢提示

# Learning to Rank: ML モデルによるリスクアーリング (2.19)

## 特徴

- XGBoost / RankLib の軽量 ML モデルでリスクアーリング
- Feature Set: 検索クエリを特徴として定義
- Feature Logging: 学習データ収集機能
- 元は外部プラグイン → 2.19 で公式プラグインに

## ユースケース

- クリックスルー率を考慮したランキング
- ビジネスルールと関連性の組み合わせ
- A/B テストによるモデル改善

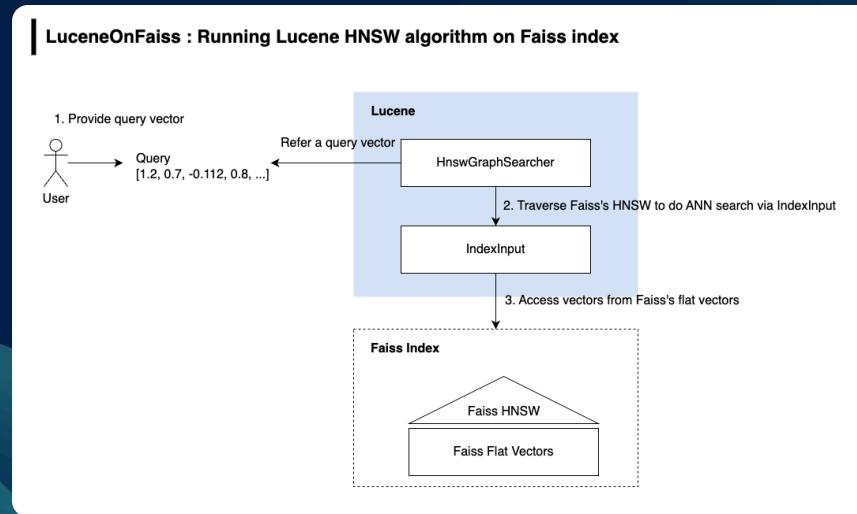
## 使用例

```
// Feature Set の定義
PUT _ltr/_featureset/my_features
{ "featureset": { "features": [
  { "name": "title_match",
    "params": [ "keywords" ],
    "template": { "match": { "title": "{{keywords}}" } }
  }]
}}
```

```
// LTR を使った検索
POST my_index/_search
{ "query": { "sltr": {
  "params": { "keywords": "検索語" },
  "model": "my_model" }}}}
```

# Lucene-on-Faiss: メモリ効率的なベクトル検索 (3.1)

## 処理の流れ



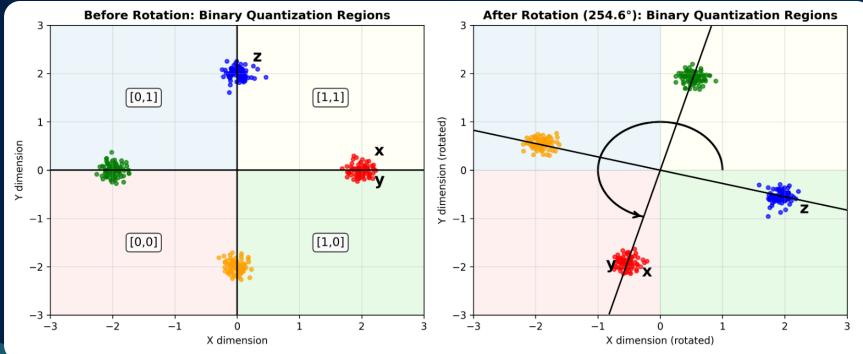
- Lucene が次の訪問ノードを決定。Faiss インデックスから必要なベクトルのみ読み込み
- Faiss 上で 距離計算を行い、結果を Lucene に返

## 特徴

- 訪問するノードのベクトルのみをディスクからオンラインデマンドで読み込むことでメモリを削減。
  - 30GB の Faiss インデックスを 8GB メモリのインスタンス (t2.large) で処理することができた。
- Lucene の早期終了ロジックにより、検索速度に改善がみられる場合も
  - 32x 量子化されたインデックスのベンチマークでは QPS が約 2 倍向上した。

# ADC + Random Rotation: 量子化精度の改善 (3.2)

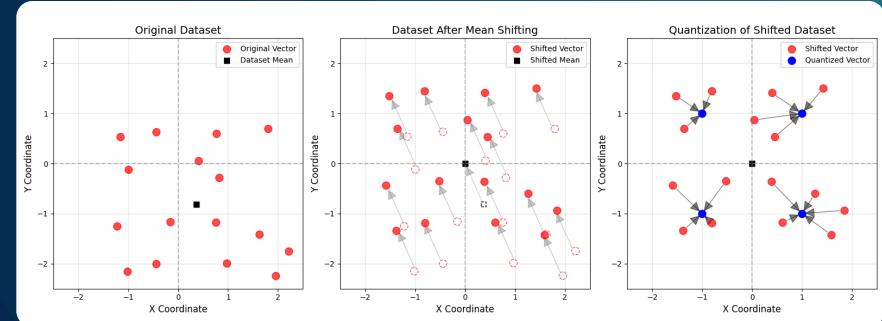
## Random Rotation



課題: 量子化時に特定次元の情報が失われやすい

解決: ランダム回転で情報を全次元に分散 → **Recall** が最大 5% 向上

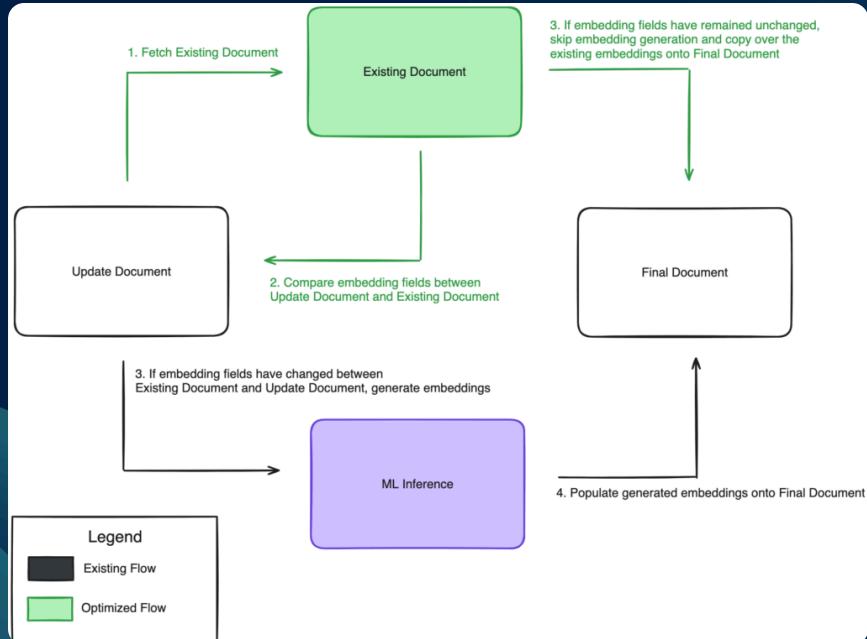
## ADC (Asymmetric Distance)



課題: クエリもドキュメントも量子化すると誤差が2倍

解決: クエリはフル精度のまま距離計算 → **Recall** が最大 10% 向上

# Skip Existing: ML推論コストの削減 (3.2)



## 特徴

- 入力フィールドが変更されていなければ既存エンベディングを再利用
- `text_embedding`, `sparse_encoding`, `text_image_embedding` に対応
- 変更なしドキュメントで処理時間 **84% 削減**

## 設定例

```
{  
    "text_embedding": {  
        "model_id": "...",  
        "field_map": { "text": "embedding" },  
        "skip_existing": true  
    }  
}
```

# 検索品質: Search Relevance Workbench

## Query Set Comparison - 複数クエリでパターンを発見

### 1. Query Set 作成

The screenshot shows the 'Create Query Set' interface. It includes fields for 'Name' (e.g., 'SRM Blog Post Query Set'), 'Description' (e.g., 'A query set for the SRM blog post with six queries'), and a 'Manual Queries' section containing a JSON object representing the queries.

```
query_set: { "queries": [ { "query": "1", "ref": "SRM_Blog_Post_1", "category": "bullet_point", "description": "bullet_point", "is_err": "false" }, { "query": "2", "ref": "SRM_Blog_Post_2", "category": "bullet_point", "description": "bullet_point", "is_err": "false" }, { "query": "3", "ref": "SRM_Blog_Post_3", "category": "bullet_point", "description": "bullet_point", "is_err": "false" }, { "query": "4", "ref": "SRM_Blog_Post_4", "category": "bullet_point", "description": "bullet_point", "is_err": "false" }, { "query": "5", "ref": "SRM_Blog_Post_5", "category": "bullet_point", "description": "bullet_point", "is_err": "false" }, { "query": "6", "ref": "SRM_Blog_Post_6", "category": "bullet_point", "description": "bullet_point", "is_err": "false" } ] }
```

### 2. Search Config 定義

The screenshot shows the 'Search Configuration' interface. It includes fields for 'Search Configuration Name' (e.g., 'keyword'), 'Index' (e.g., 'ecommerce'), and a 'Query' section containing a JSON object representing the search configuration.

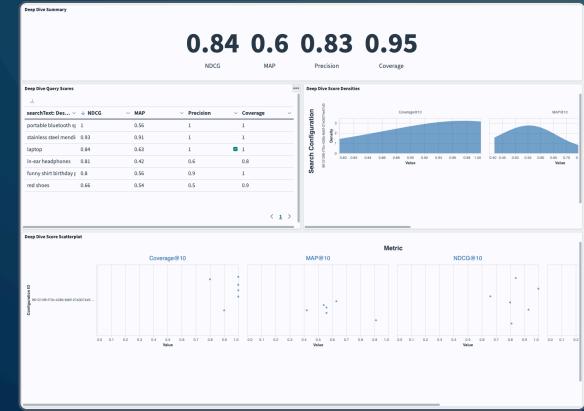
```
search_config: { "name": "keyword", "index": "ecommerce", "query": { "query": "{'query': '1', 'ref': 'SRM_Blog_Post_1', 'category': 'bullet_point', 'description': 'bullet_point', 'is_err': 'false'}", "ref": "SRM_Blog_Post_1", "category": "bullet_point", "description": "bullet_point", "is_err": "false" } }
```

比較する検索設定を2つ定義  
• クエリ本文 + パイプライン  
• 例: keyword vs hybrid search

### 評価対象のクエリ集合を定義

- 実ユーザーのクエリログからサンプリング
- 問題のあるクエリを手動で追加も可

### 3. 実験結果



- Jaccard / RBO: 結果の変化を測定
- nDCG: Judgment で定量評価
- Judgment: Explicit / UBI / LLM

# 検索品質: User Behavior Insights (UBI)

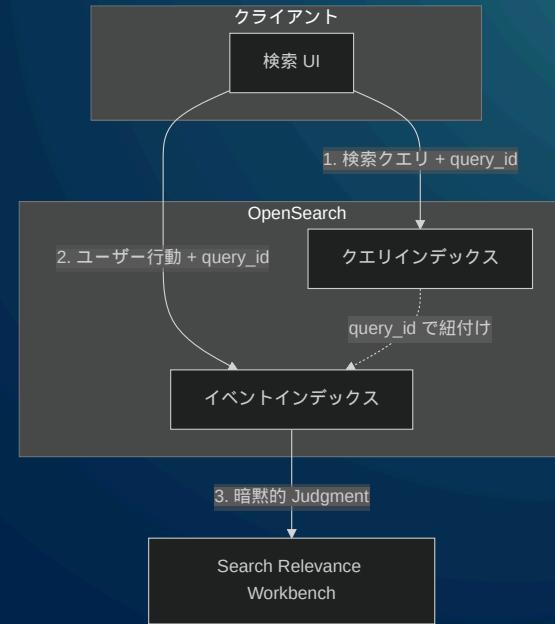
検索クエリとユーザーアクションを紐付けて記録

## 主な機能

- イベント収集: クエリ / 結果 / イベント ( click, purchase 等 ) を記録
- CTR 分析: クリック率やゼロクリック検索を検出
- A/B テスト: Team Draft Interleaving で設定を比較

## SRW との連携

- クリック等のユーザー行動を **暗黙的 Judgment** として活用
- 人手のラベリング不要で nDCG 評価が可能に



出典: OpenSearch Documentation

# AI/エージェント: OpenSearch をエージェントに組み込む

## MCP サーバー提供

- 外部エージェントから OpenSearch を呼び出し可能
- ローカル / ビルトイン MCP サーバー

## 会話メモリ

- エージェントの会話履歴を OpenSearch に保存
- コンテキスト維持による精度向上

## ノーコード構築

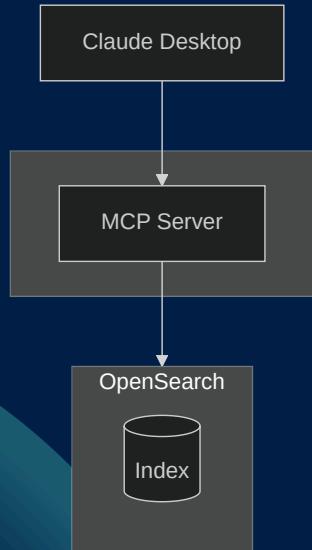
- **AI Search Flows:** RAG / エージェントをノーコード構築
- **Processor Chains:** 検索パイプラインの入出力を加工

## エージェント実行基盤

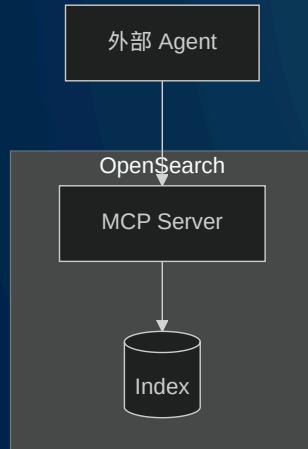
- OpenSearch 上でエージェントを作成・実行
- **Plan-Execute-Reflect** で複雑タスクを分解

# MCP: AIエージェントとの連携 (3パターン)

## 1. ローカル MCP

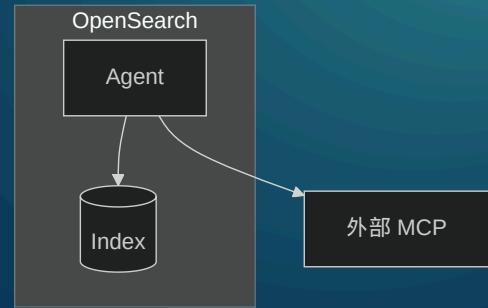


## 2. ビルトイン MCP



3.0+ で設定不要

## 3. MCP Client

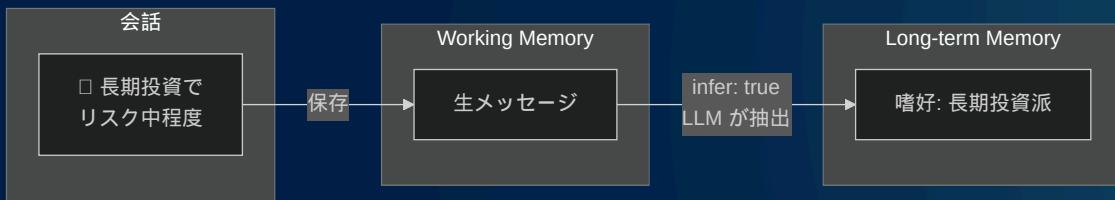


外部ツールを呼び出し

```
pip install opensearch-mcp-server-  
py
```



# Agentic Memory: 短期記憶と長期記憶の処理 (3.3)



## Working Memory ( 短期 )

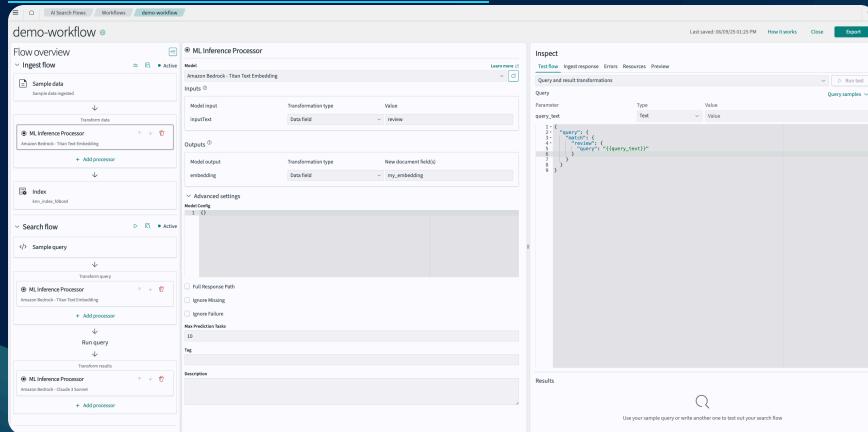
- 会話の生データをそのまま保存
- セッション中のコンテキスト維持
- トークン制限内で直接参照

## Long-term Memory ( 長期 )

- LLM が**重要情報を自動抽出**
- SEMANTIC / USER\_PREFERENCE / SUMMARY
- セッションを超えて永続化

# AI Search Flows: ノーコードでAI検索構築

## ワークフロービルダー



IDE風UI: フロー概要 / 設定 / テスト

## プリセットテンプレート

- Semantic / Hybrid / Multimodal Search
- RAG with Vector Retrieval
- Agentic Search (3.4)**

## できること

- Ingest/Search パイプライン構築
- MLモデル連携設定
- サンプルデータでテスト

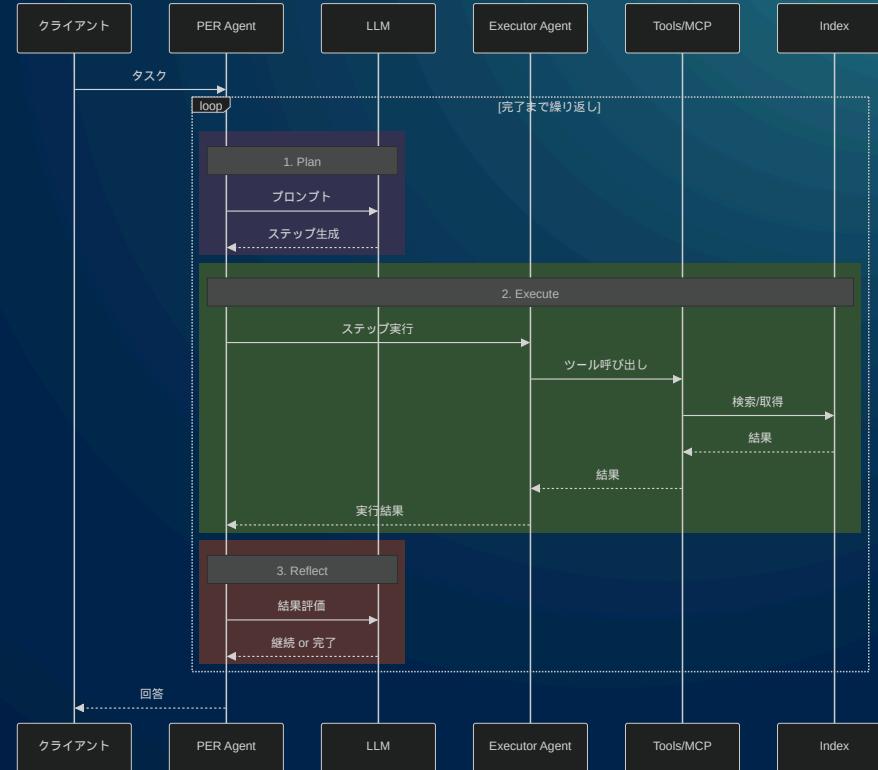
# Plan-Execute-Reflect Agent: 自律的な問題解決(3.0)

## 特徴

- 複雑なタスクを自動でステップに分解
- 中間結果に基づき計画を動的に修正
- 非同期実行・MCP連携対応

## ユースケース

- マイクロサービスのトラブルシューティング
- ログ・トレース・メトリクスの相關分析



# Observability: 主要技術 (1/2)

## Discover UI 刷新

- ログ / トレース / 可視化を統合した新 UI  
( 3.3 プレビュー、3.4 デフォルト )
- **自動チャート選択:** データパターンに応じた最適な可視化
- **Discover Traces:** クリックで PPL クエリを構築 ( 3.3 )
- **AI 支援:** コンテキスト認識チャット UI ( 3.3 experimental )

## PPL 強化

- **Apache Calcite** エンジン導入 ( 3.0 )、デフォルト化 ( 3.3 )
- lookup / join / subsearch ( 3.0 )
- flatten / expand / json\_extract ( 3.1 )
- timechart / bin / regex / rex / spath ( 3.3 )
- chart / streamstats ( 3.4 )
- 集約関数 / フィルタのプッシュダウン最適化 ( 3.2 )

# Observability: 主要技術 (2/2)

## Trace Analytics 刷新

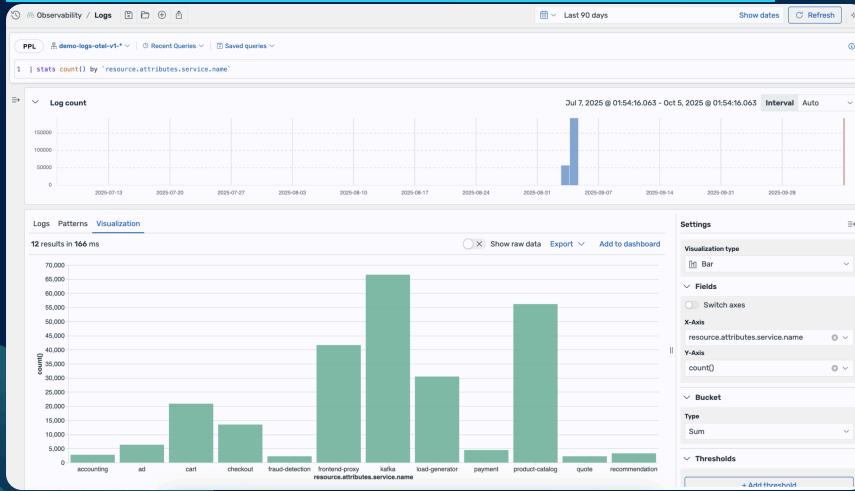
- **OpenTelemetry 完全互換** ( 3.2 )
- React Flow でサービスマップを可視化 ( 3.3 )
- カスタムインデックス名 / フィールドマッピング対応 ( 3.1 )
- クロスクラスター検索でマルチクラスタ分析 ( 3.1 )

## その他の強化

- Discover から異常検出器を直接起動 ( 3.0 )
- 異常検出の結果フラット化でダッシュボード改善 ( 2.19 )
- スパイク / ディップ検出の細分化設定 ( 2.19 )
- ML Commons の OpenTelemetry メトリクス統合 ( 3.1 )

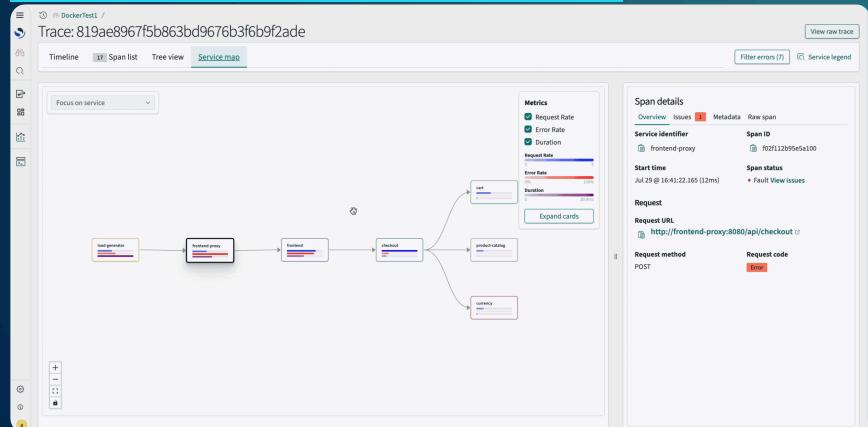
# 新 Discover UI: 統合オブザーバビリティ体験

## ログ・トレース・メトリクスを統合



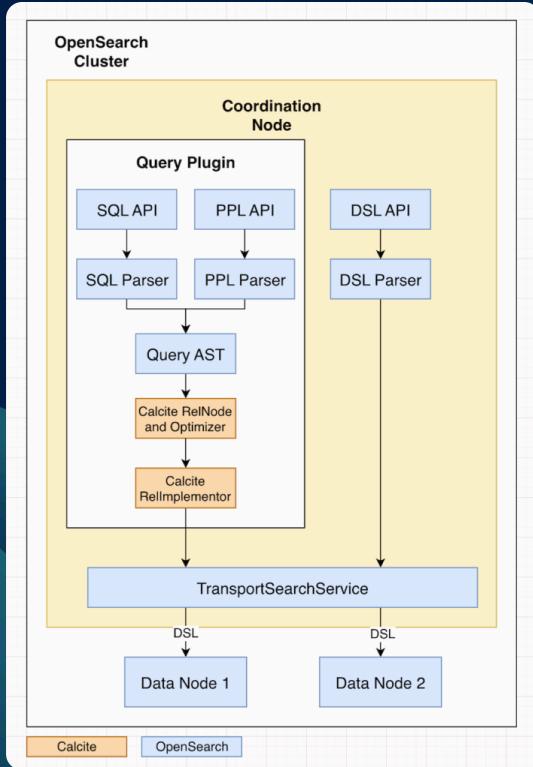
サービス別ログボリュームの自動可視化

## React Flow サービスマップ



インタラクティブなトレース可視化

# PPL on Apache Calcite: クエリエンジンの刷新 (3.0)



## 特徴

- **Apache Calcite** ベースの新クエリエンジン
- 3.0 で導入、3.3 でデフォルト化
- SQL/PPL の統一的な最適化基盤
- 集約関数 / フィルタのプッシュダウン (3.2)

## 新機能

- `lookup` / `join` / `subsearch` (3.0)
- `flatten` / `expand` / `json_extract` (3.1)
- `timechart` / `bin` / `rex` / `spath` (3.3)
- `chart` / `streamstats` (3.4)

# 運用ツールのアップデート

## Data Prepper

データ収集・変換・ルーティングパイプライン

- SQS / Kinesis / Lambda ソース追加
- MySQL / PostgreSQL の CDC 対応 (Aurora/RDS のみ)
- Kafka シンク、条件付きルーティング

## Migration Assistant

Elasticsearch → OpenSearch 移行ツールキット

- EKS / Helm デプロイ対応
- トラフィックリプレイ改善
- メタデータ移行の自動化

## OpenSearch Benchmark

パフォーマンス測定・負荷テストツール

- Redline Testing: 限界性能の自動検出
- カスタムワークロード作成の簡素化
- レポート機能の強化

## Prometheus Exporter

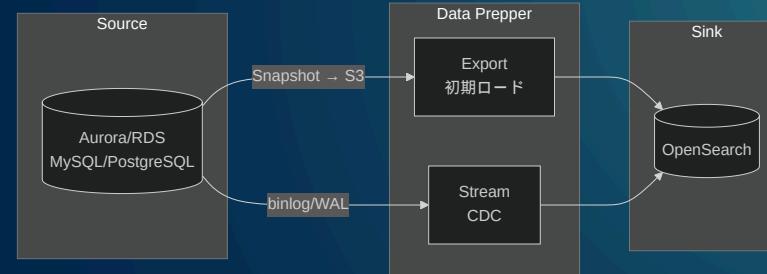
OpenSearch メトリクスを Prometheus 形式で公開

- OpenSearch プロジェクトに移行 (3.2)
- 公式サポート・メンテナンス体制
- Grafana ダッシュボード連携

# Data Prepper: RDS/Aurora CDC 対応

## 特徴

- MySQL / PostgreSQL の CDC (Change Data Capture)
- Aurora / RDS に対応
- Export: 初期ロード (S3 経由)
- Stream: binlog / WAL からリアルタイム同期
- INSERT / UPDATE / DELETE を検出



## ユースケース

- RDB → OpenSearch のリアルタイム検索
- データレイク構築

# Migration Assistant: 柔軟な移行オプション

## 3つの移行シナリオ

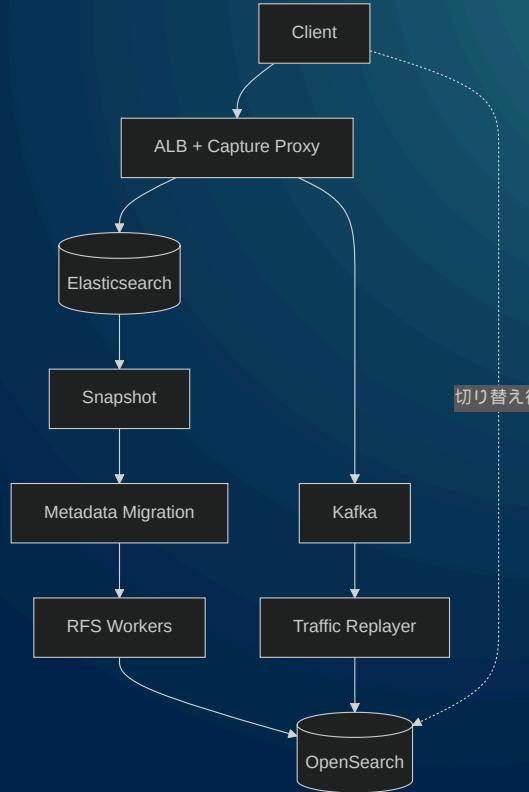
1. **Backfill** のみ - Snapshot → RFS
2. **Live Capture** のみ - Traffic Replay
3. 両方 - ゼロダウンタイム移行

## EKS / Helm 対応

- クラウド非依存 ( AWS 以外でも利用可 )
- Argo Workflows による自動化
- Pod 単位のスケーリング

## メタデータ変換

- dense\_vector → knn\_vector
- flattened → flat\_object



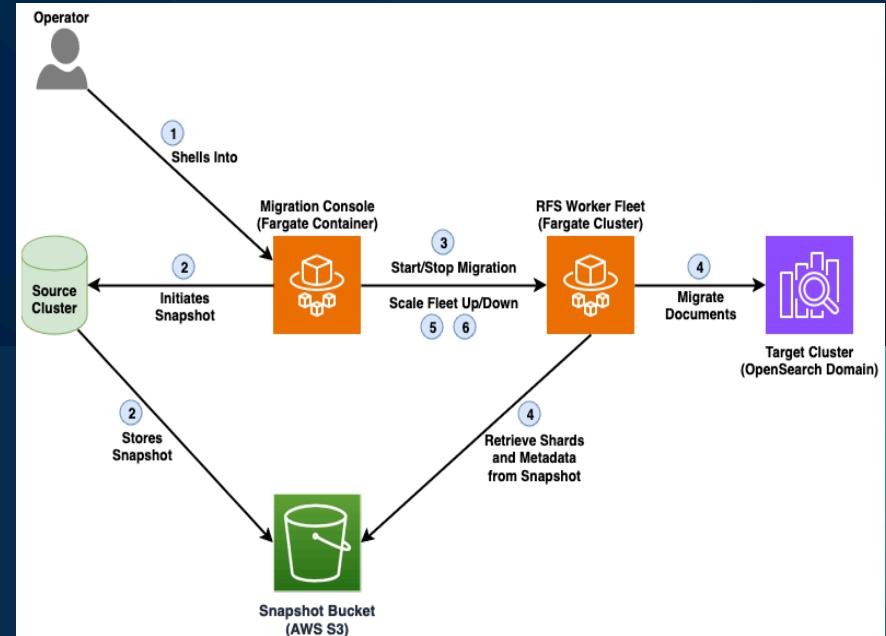
# Reindex-from-Snapshot (RFS): 高速データ移行

通常の Snapshot Restore との違い

	Snapshot Restore	RFS
OpenSearch ジョン 跨ぎ	×	複数 OK
ES 6.x → OS 2.x	×	□
ソースクラスタ負荷	高い	ゼロ
シャード並列処理	×	□

## 仕組み

Snapshot 内の Lucene ファイルから `_source` を直接抽出し、ターゲットに Bulk API で再インデックス



出典: AWS Blog

# 2025 活動振り返り

Foundation・コミュニティ・イベント

# OpenSearch Software Foundation とは

2024年9月 Linux Foundation 傘下で設立。ベンダー中立なガバナンスでプロジェクトの長期的な持続性の確保。



## Governing Board

戦略・予算・方針を決定



## Technical Steering Committee (TSC)

技術的方向性を決定

↑ 理事を任命

## Members

組織として参加・資金提供

↑ 報告

## Technical Advisory Groups (TAGs)

# Foundation メンバー

組織としてプロジェクトに参加し、運営資金を提供。投票権・理事任命権の取得。

## Premier Members



## General Members



New



New



New



New



New

# Governing Board

戦略・予算・方針の決定。Premier Memberが理事を任命、General Memberから代表を選出。



**Carl Meadows**  
AWS  
Chair



**Ben Slater**  
NetApp  
General Member



**Andrew Ross**  
AWS  
TSC Representative



**Ed Anuff**  
DataStax  
General Member



**Verena Lommatsch**  
SAP  
Premier Member



**Shanshan Song**  
Uber  
Premier Member



**Yakun Li**  
ByteDance  
General Member

# Technical Steering Committee (TSC)

技術的監督・ロードマップの決定。現TSCメンバーの投票による選出。

所属	メンバー ('YY=加入年)	所属	メンバー ('YY=加入年)
Apple	Mikhail Stepura '25	Independent	Amitai Stern '25
AWS	<b>Chair</b> Andrew Ross '24 Pallavi Priyadarshini '24 Prudhvi Godithi '25	OSC	Eric Pugh '25
ByteDance	Yakun Li '24	Paessler	Jonah Kowall '25
Eliatra	Nils Bandener '25	Salesforce	Bryan Burkholder '24
IBM	Samuel Herman '25	SAP	Karsten Schnitter '24
		Uber	Yupeng Fu '24 Shubham Gupta '24 Michael Froh '25

# Technical Advisory Groups (TAGs)

TSCに報告する長期グループ。特定技術領域のニーズを監督・調整。ミーティングは公開、誰でも参加可能。

## Build TAG

ビルド・CI/CD関連

## Observability TAG

ログ・メトリクス・トレース

OpenTelemetry, Prometheus,  
Jaeger, Fluent Bit と連携

**2025年9月発足**

AWS, Uber, SAP, Apple, Paessler

## Security TAG

セキュリティ関連

# Ambassadors

個人としてコミュニティ活動を推進するリーダーの認定。2025年9月にプログラム開始。



**Amanda Katona**  
NetApp



**Charlie Hull**  
The Search Juggler



**Dotan Horovits**  
AWS



**Eric Pugh**  
OSC



**Itamar Syn-Hershko**  
BigData Boutique



**Kassian Rosner Wren**  
NetApp



**Kris Freedain**  
AWS



**Laysa Uchoa**  
Nordcloud

応募条件: コード・コンテンツ・ドキュメント等で貢献実績があり、コミュニティ活動に1年間コミットできる方

応募プロセス: 年2回募集（3月・9月）→ 貢献実績を記載して応募 → Foundation審査 → 1年任期

# User Group の成長

技術セッション・ハンズオン・ユースケース共有・コントリビューター育成を各地で実施



**19**

Countries

**37**

Groups

**7,933**

Members

# OpenSearchCon

OpenSearchプロジェクトの公式  
カンファレンス

- 技術セッション・ワークショッ  
プ
- 最新機能の発表
- コミュニティ交流

[全プレイリスト](#)

2025 5都市

**Europe**

4-5月

[Videos](#)

**India**

6月

[Videos](#)

**N.America**

9月

[Videos](#)

**Korea**

11月

[Videos](#)

**Japan**

12月

[Videos](#)

2024 3都市

**Europe**

ベルリン

[Videos](#)

**India**

ベンガルール

[Videos](#)

**N.America**

サンフランシスコ

[Videos](#)

2023

**N.America** シアトル [Videos](#)

2022 初開催

**N.America** シアトル [Videos](#)

# OpenSearchCon Japan 2025

**131**

In-Person Attendees

**42**

Organizations

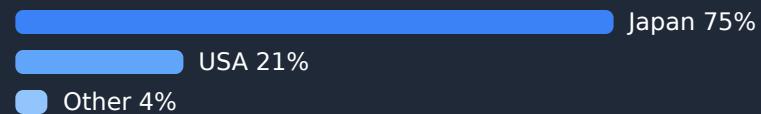
**22**

Conference Sessions

## Industry



## Attendees by Region



## Primarily Attended For



# 2026年のロードマップ

# 2026年ロードマップ概要

## □ Streaming Query

- v3.5: Streaming Aggregations デフォルト化
- Apache Arrow形式でブロック処理
- メモリ効率向上・応答時間短縮

## □ Composable Engine

- Substrait IR による統一論理プラン
- DataFusion / Velox 実行エンジン
- DSL/SQL/PPL の共通最適化

## ⚡ gRPC API 拡張

- 50+ Aggregation 対応
- Python SDK (PyPI公開済)
- REST API との機能パリティ

## □ Core Performance

- Skip-list: サブ集計対応
- Bulk Collection API (Lucene 10.4)
- **Intra-segment 並列検索**

## □ Vector Search

- BFLOAT16 対応 (メモリ50%削減)
- Memory-Optimized 検索の改善
- Disk-based v2 (BPGP/Gorder-PQ)
- GPU転送最適化で2x高速化

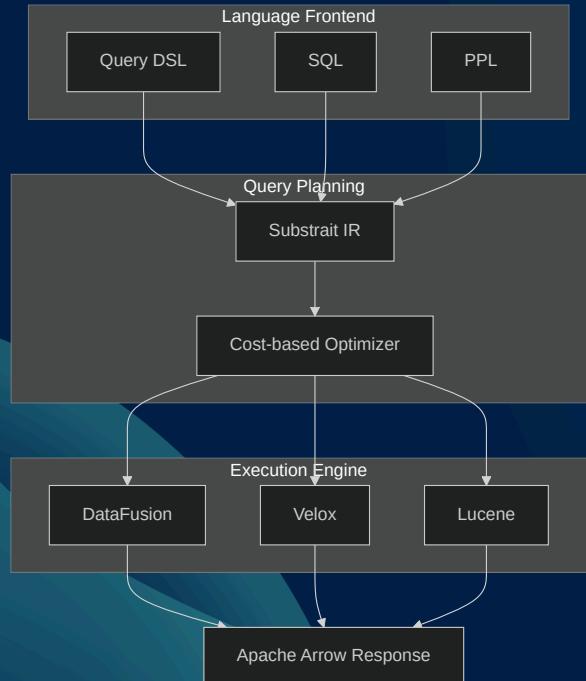
## □ 拡張性

- Vector Engine 統一 I/F
- k-NN / JVector 共通化
- Neural Search プラグイン分離

[□ GitHub Roadmap](#) | [□ Performance Blog \(2025/12\)](#)

# Composable Query Engine

## 新アーキテクチャ



## 現状の課題

- Luceneとの密結合による制約
- 大規模集計でのメモリボトルネック
- DSL/SQL/PPLで重複した式エンジン

## コミュニティの動き

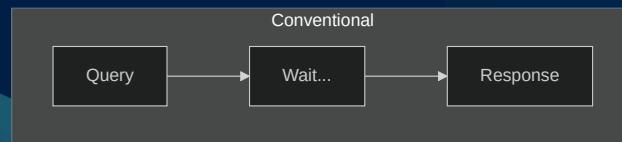
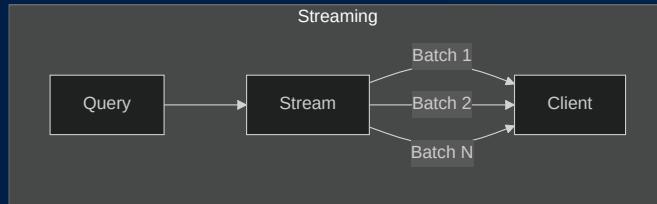
- **ByteDance**: OLAPプラグイン提供
- **Velox4J**: Velox統合の提案
- **Segmentless設計**: ベクトル検索向け

## 互換性

- 既存Luceneベースの集計は維持
- **opt-in**プラグインとして実装

# ストリーミングクエリーアーキテクチャ

## 従来 vs ストリーミング



従来: 全処理完了まで待機 → メモリ大

新方式: Arrow Batchで逐次返却 → 低メモリ

## 期待される効果

指標

改善

応答時間

最大2倍高速

メモリ使用量

大幅削減

部分結果

即時返却可能

## 3.5での目標

- ストリーミング集計をデフォルト有効化
- クエリプランニング改善 / Virtual Threads

## 対応集計

Numeric terms / Cardinality count (今後拡張)

# BFloat16 対応: 範囲制限なしで50%メモリ削減

## 浮動小数点形式の比較

形式	ビット	範囲	メモリ
FP32	32	$\pm 3.4 \times 10^{38}$	100%
FP16	16	$\pm 65,504$	50%
<b>BFloat16</b>	16	$\pm 3.4 \times 10^{38}$	<b>50%</b>

FP16 は範囲制限があり、範囲外の値があると使えない  
→ BFloat16 なら FP32 と同じ範囲で 50% 削減

## BFloat16 とは

Google が機械学習用に開発した16bit浮動小数点形式。FP32 の上位16bitをそのまま切り取った構造で、精度を犠牲に範囲を維持

FP32: 1bit符号 + 8bit指数 + 23bit仮数

BFloat16: 1bit符号 + 8bit指数 + 7bit仮数

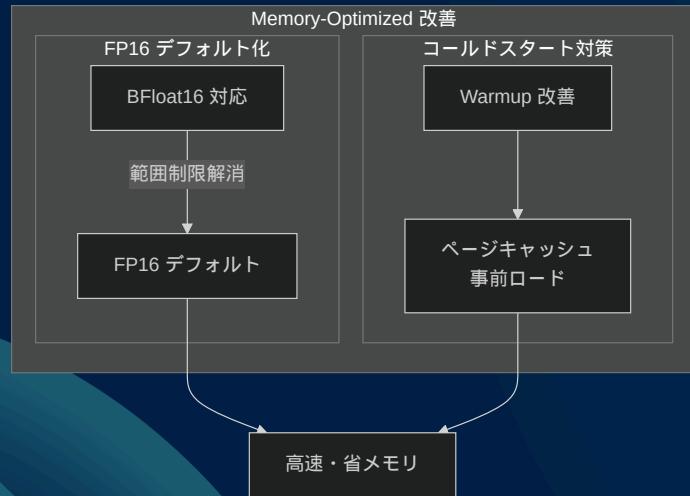
仮数部 7bit → 精度 2～3桁だが距離計算には十分

## ハードウェアアクセラレーション

Intel AVX512 BF16 命令セットにより、新世代プロセッサでハードウェアレベルの高速化が可能

# Memory-Optimized Vector Search 改善

## 改善計画



## FP16 デフォルト化

v3.4.0 で Native SIMD scoring が導入され、FP16 のパフォーマンスが大幅改善。BFloat16 対応により範囲制限が解消されれば、FP16 をデフォルト化して **メモリ50%削減** が可能に

## コールドスタート対策

従来は Warmup API を呼んでもセクションオフセットの解析のみで、最初のクエリで cold start が発生していた

Warmup 改善により 4KB 単位でページを touch し、カーネルがページキャッシュに事前ロード → テールレイテンシ削減

# Disk-based Vector Search 改善

## Disk-based Vector Search とは

低メモリ環境向けの 2 フェーズ検索。量子化インデックス（メモリ）で候補絞り込み → フル精度ベクトル（ディスク）で再スコアリング

圧縮率	エンジン	量子化方式
4x	Lucene	Scalar Quantization
8x～32x	Faiss	Binary Quantization

## 課題

- 第2フェーズにおけるレイテンシが課題。ベクトルがディスク上に散在するためランダム I/O が発生する。
- Lucene では Binary Quantization がサポートされていない。

Faiss が唯一の選択肢となるが 32x 圧縮時の Recall に課題がある

## 改善の方向性

### Faiss 向け: Vector Reordering

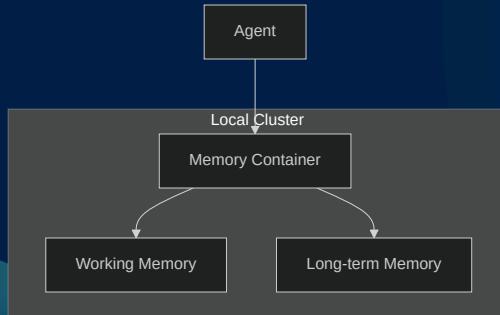
- Bipartite Graph Partitioning によるベクトル再配置
- Gorder-PQ によるグラフ隣接関係の最適化
- シーケンシャル I/O でレイテンシ改善

### Lucene 向け: Better Binary Quantization

- RaBitQ ベースの高精度 1bit 量子化に対応
- Faiss の Bianry Quantization と比較して高 Recall

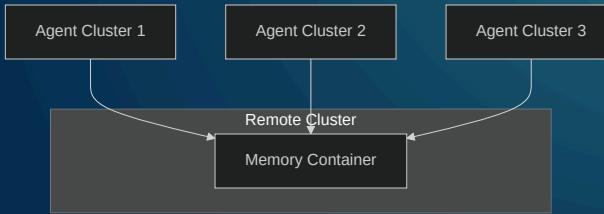
# AI/ML ロードマップ: Remote Agentic Memory (3.5)

## 3.3: Local Agentic Memory



- 同一クラスター内でのみメモリ保存
- エージェントとメモリが密結合

## 3.5: Remote Agentic Memory



- 複数クラスターでメモリ共有
- コンピュートとストレージの分離
- マネージド環境での柔軟な構成

`memory_configuration.endpoint` でリモートクラスターを指定

# 2026年 OpenSearchCon

## イベント

## 日程

## 場所

OpenSearchCon China 3月17-18日 上海 (Hilton Shanghai Hongqiao)

OpenSearchCon Europe 4月16-17日 プラハ

OpenSearchCon India 6月15-16日 ムンバイ (KubeCon India併催)

OpenSearchCon North America 9月22-24日 サンノゼ, CA

# まとめ

## 2025年の技術的進化

領域	主な成果
<b>Vector / AI</b>	FP16、Lucene-on-Faiss、Agent Framework、MCP
<b>Search</b>	Workbench、UBI、Seismic
<b>Performance</b>	Star-tree Index、Derived Source
<b>Observability</b>	Discover UI刷新、PPL Calcite

## 2026年の注目領域

領域	ロードマップ
<b>Query Engine</b>	Composable Query Engine
<b>Streaming</b>	Apache Arrow/Flight
<b>Vector</b>	BFloat16、Memory-Optimized、Disk-based v2
<b>AI/ML</b>	Remote Agentic Memory

コミュニティの成長: ダウンロード13億回 / コントリビューター3,300名 / 参加組織400社

# コミュニティに参加しよう！



**Slack**



**Forum**



**GitHub**

## 今日から始められること

- Playground で試す: playground.opensearch.org
- ドキュメントを読む: docs.opensearch.org
- User Group Japan に参加

## コントリビュート

- Issue報告・PR: GitHub
- Ambassadorに応募（2月・9月）□ Now Open!
- OpenSearchCon で発表



# Thank you!

引き続きセッションをお楽しみください