# Search Index (Earlybird) main classes

> \*\*TL;DR\*\* Earlybird (Search Index) find tweets from people you follow, rank them, and serve them to Home.

## What is Earlybird (Search Index)

[Earlybird](http://notes.stephenholiday.com/Earlybird.pdf) is a \*\*real-time search system\*\* based on [Apache Lucene](https://lucene.apache.org/) to support the high volume of queries and content updates. The major use cases are Relevance Search (specifically, Text search) and Timeline In-network Tweet retrieval (or UserID based search). It is designed to enable the efficient indexing and querying of billions of tweets, and to provide low-latency search results, even with heavy query loads.

## High-level architecture

We split our entire tweet search index into three clusters: a \*\*realtime\*\* cluster indexing all public tweets posted in about the last 7 days, a \*\*protected\*\* cluster indexing all protected tweets for the same timeframe; and an \*\*archive\*\* cluster indexing all tweets ever posted, up to about two days ago.

Earlybird addresses the challenges of scaling real-time search by splitting each cluster across multiple \*\*partitions\*\*, each responsible for a portion of the index. The architecture uses a distributed \*inverted index\* that is sharded and replicated. This design allows for efficient index updates and query processing.

The system also employs an incremental indexing approach, enabling it to process and index new tweets in real-time as they arrive. With single writer, multiple reader structure, Earlybird can handle a large number of real-time updates and queries concurrently while maintaining low query latency. The system can achieve high query throughput and low query latency while maintaining a high degree of index freshness.

## Main Components

\*\*Partition Manager\*\*: Responsible for managing the configuration of partitions, as well as the mapping between users and partitions. It also handles index loading and flushing.

\*\*Real-time Indexer\*\*: Continuously reads from a kafka stream of incoming tweets and updates the index (tweet creation, tweet updates, user updates). It also supports tweet deletion events.

\*\*Query Engine\*\*: Handles the execution of search queries against the distributed index. It employs various optimization techniques, such as term-based pruning and caching.

\*\*Document Preprocessor\*\*: Converts raw tweets into a document representation suitable for indexing. It handles tokenization, normalization, and analysis of tweet text and metadata. See our ingestion pipeline `src/java/com/twitter/search/ingester` for more write-path processing.

\*\*Index Writer\*\*: Writes tweet documents to the index and maintains the index structure, including \*\*posting lists\*\* and \*\*term dictionaries\*\*.

\*\*Segment Manager\*\*: Manages index segments within a partition. It is responsible for merging, optimizing, and flushing index segments to disk, or flush to HDFS to snapshot live segments.

\*\*Searcher\*\*: Executes queries against the index, using techniques like caching and parallel query execution to minimize query latency. It also incorporates scoring models and ranking algorithms to provide relevant search results.

The most important two data structures for Earlybird (or Information Retrieval in general) including:

\* \*\*Inverted Index\*\* which stores a mapping between a Term to a list of Doc IDs. Essentially, we build a hash map: each key in the map is a distinct Term (e.g., `cat`, `dog`) in a tweet, and each value is the list of tweets (aka., Document) in which the word appears. We keep one inverted index per field (text, UserID, user name, links, etc.)

\* \*\*Postings List\*\* which optimize the storage a the list of Doc IDs mentioned above.

See more at: https://blog.twitter.com/engineering/en\_us/topics/infrastructure/2016/omnisearch-index-formats

## Advanced features

Earlybird incorporates several advanced features such as facet search, which allows users to refine search results based on specific attributes such as user mentions, hashtags, and URLs. Furthermore, the system supports various ranking models, including machine learning-based scoring models, to provide relevant search results.

## Directory Structure

The project consists of several packages and files, which can be summarized as follows:

\* At the root level, the primary focus is on the Earlybird server implementation and its associated classes. These include classes for search, CPU quality factors, server management, index config, main classes, server startup, etc.

\* `archive/`: Directory deals with the management and configuration of archived data, specifically for Earlybird Index Configurations. It also contains a `segmentbuilder/` subdirectory, which includes classes for building and updating archive index segments.

\* `common/`: Directory holds utility classes for logging, handling requests, and Thrift backend functionality. It also has two subdirectories: `config/` for Earlybird configuration and `userupdates/` for user-related data handling.

\* `config/`: Directory is dedicated to managing tier configurations specifically for archive cluster, which relate to server and search query distribution.

\* `document/`: Handles document creation and processing, including various factories and token stream writers.

\* `exception/`: Contains custom exceptions and exception handling classes related to the system.

\* `factory/`: Provides utilities and factories for configurations, Kafka consumers, and server instances.

\* `index/`: Contains index-related classes, including in-memory time mappers, tweet ID mappers, and facets.

\* `ml/`: Houses the `ScoringModelsManager` for managing machine learning models.

\* `partition/`: Manages partitions and index segments, including index loaders, segment writers, and startup indexers.

\* `querycache/`: Implements caching for queries and query results, including cache configuration and update tasks.

\* `queryparser/`: Provides query parsing functionality, including files that cover query rewriters and lhigh-frequency term extraction.

\* `search/`: Contains read path related classes, such as search request processing, result collectors, and facet collectors.

\* `segment/`: Provides classes for managing segment data providers and data reader sets.

\* `stats/`: Contains classes for tracking and reporting statistics related to the system.

\* `tools/`: Houses utility classes for deserializing thrift requests.

\* `util/`: Includes utility classes for various tasks, such as action logging, scheduled tasks, and JSON viewers.

## Related Services

\* The Earlybirds sit behind Earlybird Root servers that fan out queries to them. See `src/java/com/twitter/search/earlybird\_root/`

\* The Earlybirds are powered by multiple ingestion pipelines. See `src/java/com/twitter/search/ingester/`

\* Earlybird segments for the Archives are built offline by segment builders

\* Also, Earlybird light ranking is defined in `timelines/data\_processing/ad\_hoc/earlybird\_ranking`

and `src/python/twitter/deepbird/projects/timelines/scripts/models/earlybird`.

\* Search common library/packages

## References

See more:

\* "Earlybird: Real-Time Search at Twitter" (http://notes.stephenholiday.com/Earlybird.pdf)

\* "Reducing search indexing latency to one second" (https://blog.twitter.com/engineering/en\_us/topics/infrastructure/2020/reducing-search-indexing-latency-to-one-second)

\* "Omnisearch index formats" (https://blog.twitter.com/engineering/en\_us/topics/infrastructure/2016/omnisearch-index-formats)