Whitepaper

Architecture of DirectSearch CORE

In this document, we will explain how Cloudtenna is able to overcome key roadblocks that have historically made cross-silo file services impossible.

File search presents a unique challenge for enterprise platforms. To support large scale search, solutions require specialized infrastructure designed specifically for unstructured data. Engineers at Cloudtenna have developed a brand new global-name-space architecture to make it easy to add advanced cross-silo file management services to any application.

Introduced below is a revolutionary new architecture that separates metadata and content. By separating the document extraction process from the indexing process, Cloudtenna is able to overcome key roadblocks that have historically made cross-silo file services impossible.

KEY CHALLENGES

Search must respect file permissions

For file search, every user receives a personalized experience. Each user has access to a different set of documents, defined by file permissions. This is in contrast to traditional web search engines which search over a corpus of documents that are largely the same for all users. File search is burdened with the additional task of enforcing per-user and per-group file permissions on each and every search query.

Data is scattered across unharmonious file repositories

Files are scattered across all the tools users use in their daily workflow. Universal search must normalize these disparate data sets into a single searchable index. For example, an employee at a large enterprise has access to files on the company's on-premises file server, collaborates with the marketing team using Dropbox, communicates with the engineering team via Slack, and uploads invoices to Salesforce. It is almost impossible to remember where any document is stored. This user needs a birds-eye view to search for documents regardless of where they are stored. File search infrastructure must be able to index content and reconcile file permissions across a wide range of data sources – each which use their own proprietary APIs and file permission paradigms. It's not enough to just index this dispersed content, but this content must be normalized into a single global-name-space.

Content is updated frequently

The most commonly searched for content is hot data – files that have been recently updated. When performing a search, a user is more likely to be looking for a file updated yesterday than a file last touched over a year ago. To make sure recent file updates return in search results, search infrastructure must quickly recognize new content and quickly update the search index. Across an enterprise, content changes often and search infrastructure must be able to keep up.

Security is paramount

For an enterprise, data protection is an absolute bedrock principle – no solution can be adopted that risks exposing sensitive data. As discussed above, file search must only return search results the user has permissions to view. The first challenge is understanding and enforcing file permissions. The second challenge is updating file permissions as to not expose a window of

time when a user can search for files he no longer has permissions to view. If a user is moved from the Sales-group to the Marketing-group, potentially millions of objects in the search database need to be updated to reflect the user's new file permissions. Search infrastructure must account for this update within a reasonable amount of time to prevent a serious security gap.

Rank results by relevancy
We want to help users find the most relevant documents for a given query. This requires being able to leverage machine intelligence at several stages in the search pipeline, from content-specific machine learning (such as image understanding systems) to learning systems that can better rank search results and file recommendations to suit each user's preferences.

Data indexing is expensive
At the heart of every search infrastructure in an index of searchable terms. Collecting and maintaining these keywords is historically a heavy operation. At enterprise scale, these challenges make traditional search prohibitively expensive.

The first challenge is compute requirements. For example, a data crawler is responsible to recognize and re-index file changes in near real-time. In a company of 1,000 employees, it is cost prohibitive to run 1,000 simultaneous crawlers (one for each user) and is downright dangerous because 1,000 simultaneous file-reads is sure to slow the source file server to a crawl – blocking normal direct access for users while the crawlers are performing their index. Unless the usecase does not require new content to be indexed until off-hours, a different lightweight architecture is required.

The second challenge is storage costs. Enterprises typically have millions if not billions of files stored across their many repositories. Search infrastructure stores a full-text-index of each file object so that files can be retrieved via keyword search queries. The cost of this storage is astronomical. Luckily, predictable patterns unlock ways to reduce the required storage footprint. In fact, a lot of that content is duplicates of the same content, just in different locations. To keep costs within budget, file search infrastructure must take advantage of natural language processing (NLP), smart deduplication, and other techniques to manage storage costs.

PRIMARY OBJECTIVES:

- Deliver best-in-class performance, scalability, and reliability to deal with the scale of enterprise file search
- Build a flexible system that would allow our partners to easily customize the document-indexing and query-processing pipelines so that they can tailor the infrastructure for their usecase
- Set strong safeguards to preserve the privacy of users' data

In this blog post we describe the architecture of the DirectSearch CORE and its key characteristics, provide details about the choices we made in terms of technologies and

approaches we chose for the design, and explain how we make use of machine learning (ML) at various stages of the system.

High-level architecture:
DirectSearch consists of three mostly-independent sub-components: Data connectors, our analytics engine (DirectSearch CORE), and a front-end search UI. Cloudtenna core technology and key architecture decisions are housed almost entirely inside the analytics engine. Complexity is extracted out of the data connectors and front-end UI to make it easy for partners to build their own substitute components.

Data connectors
Cloudtenna DirectSearch is designed to provide global file search – meaning that it can ingress data from any and all data sources. Every data source, though, has its own set of APIs, authentication mechanisms, and file permission paradigms. Cloudtenna uses a set of per-service data connectors to communicate with source repositories.

The role of the data connectors are to recognize file and user activity, extract content and metadata, and send the metadata to the analytics engine. The analytics engine then later handles the heavy lifting like ACL crunching and deduplication so that the connectors are not burdened with bloated logic. Through this architecture, the data connectors have limited scope which makes them easy to build and maintain.

Cloudtenna provides reference architecture for common file storage connectors including Dropbox, Box, Microsoft OneDrive, Google Drive, Gmail, Outlook, Slack, JIRA, Confluence, and on-premises CIFS filers. By design, partners can use our connector-templates to build their own additional connectors to other services.

Analytics Engine (DirectSearch CORE):
The analytics engine comprises of three core components: the index builder, the retrieval engine, and intelligent ranking engine. Cloudtenna developed theses purpose-built components from the ground up to overcome challenges specific to file-management usecases.

The index builder
As discussed above, the analytics engine works in tandem with a series of data connectors to third-party services. The analytics engine is responsible for receiving content and metadata from the connectors and using this data to build a searchable index. Unlike common solutions that require heavy operations and off-hour batching, DirectSearch CORE uses SPARK in-memory processing which allows for real-time (deterministic) updates. This means that as soon as the analytics engine receives an update – for example that a user has subscribed to a new AD group – the analytics engine immediately begins to build the updated index.

For a deeper technical dive into Cloudtenna's real-time binding, please read our Security Binding white-paper about the advantages of real-time binding vs early/late binding.

The retrieval engine
Upon receiving a query from a user, the first task performed by the analytics engine is to call the DirectSearch CORE identity management (IDM) service to determine the exact set of third-party services and ACLs the user has read access to view. The IDM service solves the user reconciliation problem – associating the querying-user to the multiple disparate third-party services that user has connected. This set defines the "scope" of the query that will be performed by the downstream retrieval engine, ensuring that only content accessible to the user will be searched.

After collecting the list of candidate documents from the search backends, the retrieval engine fetches additional signals and metadata as needed, before sending that information to a separate ranking service, which in turn computes the scores to select the final list of results returned to the user.

The ranking engine
The ranking engine is designed to help surface the documents that the user is most likely to want right now. The retrieval engine may return a result of 75 files. It is the ranking engine's responsibility to further filter the results to boost relevant results to the top of the search results. Think in terms of trying to get the exact website you are looking for to show up on the first page of a Google search.

The ranking engine is powered by a ML model that outputs a score for each document based on a variety of signals. Some signals measure the relevance of the document to the query, while others measure the relevance of the document to the user at the current moment in time (e.g., who the user has been interacting with, or what types of files the user has been working on). The model is trained using "click" data from our front-end. Given searches in the past and which results were clicked on, we can learn general patterns of relevance.

The ranking engine is capable of filtering on a number of pre-built heuristics like file-type and last modified time. The ML-capabilities mean it is possible to build relevancy scores based on an exploding number of additional heuristics.

Front-end UI:
As reference architecture, Cloudtenna delivers a beautiful front-end web-UI designed for cross-silo file search. Partners typically wish to use APIs to embed results directly into their own UI. Cloudtenna's services documented via Swagger for easy integration.

DEEPER DIVE:

Indexing-pipeline

As discussed above, DirectSearch CORE receives updates from data connectors connected to third-party data sources. Every service provides different insight into how to crawl and recognize file updates. As a result, each connector can be optimized to take advantage of the particular end-points available.

- The default way to crawl for new content is to periodically iterate through all files in the data source and check for updates. This works on all data sources.
- Some data sources provide additional endpoints like event notifications and audit logs which can point the crawler to new file updates without the need to the iterate through the entire file tree. If these endpoints are available from the data source, a crawler can recognize file updates much faster and pass the update sooner to the analytics engine for real-time processing into the index.

The indexing process recognizes when a file or user setting needs to be extracted. We separate the indexing process from the document extraction process. This enables support for multiple different data pipelines and other advantages like easy deduplication and fast file delete cleanup.

For most documents, we rely on Apache Tika to transform the original document into a canonical text-based representation, which then gets parsed in order to extract a list of "tokens" (i.e. words) using natural language processing (NLP). This extraction, which strips away content formatting and other unnecessary metadata can reduce storage size by up to 1:17x depending on usecase. The architecture also supports alternative data pipelines a partner may wish to implement, like Tensorflow for image recognition.

Search index architecture

The format of traditional data stores are not well-suited for running searches. This is because it stores extracted content mapped by document id. For search, we need an inverted index: a mapping from search term to list of documents. Essentially, our architecture creates a metadata global-name-space parallel to third-party document stores.

By separating the document extraction process from the indexing process, we gain a lot of flexibility:

- Support for multiple data pipelines
- Smart deduplication to keep storage costs down
- Quickly remove content from the index after a file delete