# Citadel Search: Open Source Enterprise Search

Pablo Panero[1], Ismael Posada Trobo[2], Carina Rafaela de Oliveira Antunes[3] and Andreas Wagner[4]

***Abstract*— Nowadays, the amount of digital content that is created increases at a very fast pace. In many of these cases, the produced content is meant to be publicly available, and it will be discoverable through most of the existing search engines. Even though enterprises and organisations are no exception in the amount of produced content, not all of it can be made publicly available and should only be discoverable and accessible if the user fulfills certain authorization and authentication requirements. In addition, the organisation-specific nature of the information takes us into the field of enterprise search.**

**Citadel Search is an Open Source enterprise search solution that makes use of state of the art technology such as Elasticsearch and the Invenio Framework for large-scale digital repositories. This enables users to create tailored data models for different information sources, have fine grain access control over this information, and obtain relevant results upon search queries.**

**This paper describes the design and architecture of Citadel Search and its components. Furthermore, it presents implementation details and results of using Citadel Search for several large document collections at CERN. Namely, CERN's Engineering Data Management Service (EDMS), the Indico application for event organisation, archival and collaboration, and finally CERN's large information space made of more than 14000 Web sites.**

## I. INTRODUCTION

Using a centralized enterprise search solution that aggregates all data presents several challenges [1]. In addition, CERN's *search as a service* paradigm adds more challenges that need to be tackled. For example, providing support for different data and access control models, isolation among collections and different levels of scalability. Hence a high level of customizability is required to fulfil the users' needs.

As it can be seen in Figure 1, the global Citadel Search system could be generalized as a set of different instances, being therefore isolated, and where each one would use a specific data and access control model. Each instance would have its own persistent storage database, document index, cache and message broker. However, all these instances must also expose their content through a central entry point, though still honoring each instance-specific access control rules. This would allow the re-use of all indexed content from a global enterprise search portal.

[1]CERN pablo.panero@cern.ch
[2]CERN ismael.posada.trobo@cern.ch
[3]CERN carina.oliveira.antunes@cern.ch
[4]CERN andreas.wagner@cern.ch

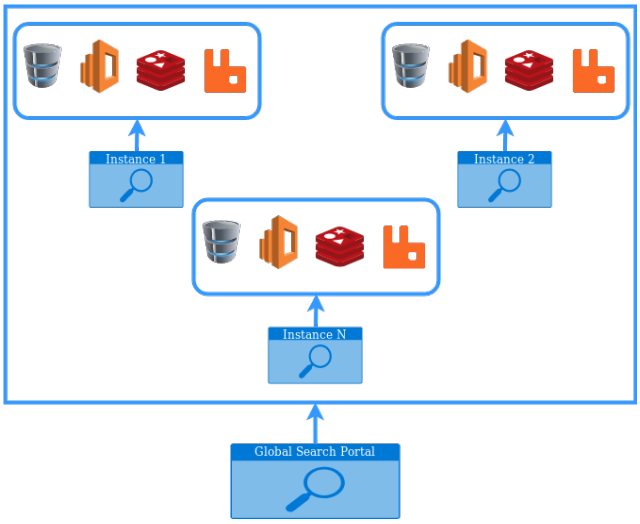


**Fig. 1:** Citadel Search components overview, with three specific instances connected to the global search portal.

## II. DESIGN AND IMPLEMENTATION

From a zoomed-out perspective, it can be seen that there are two main things required to fulfil the previously stated needs: A place to store vast amounts of heterogeneous content (from the data model point of view) and a method to query it, obtaining the most relevant results.

In conclusion, what is needed is a *large scale repository, which provides indexing and search capabilities*. This is the definition of what the Invenio Framework [2] provides, and therefore Citadel Search has been built on top of it.

### A. Architecture

As a consequence of using the Invenio Framework, the infrastructure architecture of Citadel is similar to other Invenio instances. Names of the specific technologies used at CERN can be found between parentheses:

- Application server: It runs the Citadel Search application, and therefore the logic behind this enterprise search system (uWSGI). Citadel Search is a Python application supporting Python version 3.6 and above.
- Persistent Storage: All indexed documents are stored in JSON format in a SQL relational database (PostgreSQL). The reason behind the use of a SQL database is its transactional capabilities, which ensures data consistency. In addition, single document look ups

(by primary key) are usually much more efficient than when performed over a document store.
- Document storage, search and indexing: Indexing, querying and getting relevant documents from those queries is left to the document storage / search engine (Elasticsearch).
- File Storage: In order to store files and other binary content, Citadel makes use of object storage. Thanks to Invenio, multiple storage back-ends can be used at the same time (Local, migration to S3 planned).
- Caching: For fast temporary storage (user sessions, rendered pages, etc.) Citadel makes use of in-memory caching (Redis).
- Background processing: In order to perform long-running and bulk operations the system makes use of multiple queues (RabbitMQ).

In addition, a load balancer can be set up in front of the application servers in order to scale out easily. In the production deployments at CERN, the Nginx web server is set up as a load balancer.

More details can be found in the official Invenio documentation [3]. Moreover, due to the interoperability provided by the Invenio Framework technologies such as MySQL, Apache, HAProxy, among others could also be used to run Citadel Search.

There are currently eight instances of Citadel Search deployed at CERN. This deployment is done on CERN's software container application platform based on OKD/OpenShift, and a container template has been made available [4].

### B. Customizable data model

Citadel Search makes use of what the Invenio Framework calls *record*. Namely, an object abstraction that can contain anything that can be represented in JSON format. In consequence, all documents are ingested as a collection of key value pairs following the JSON Schemas [5] specifications. This is what it is called *data model*.

Moreover, the loading, serialization and de-serialization process is configurable. Consequently, a custom mechanism to process records could be set in place. Allowing the user to send data in a different format than the one specified in the data model, calculating and/or transforming the missing values in order to match the data model. This is useful for metadata enrichment and access control.

In addition, a similar schema is created to store the documents in Elasticsearch. This is called *mapping*, and contains the specification of how the fields are stored and indexed (e.g. which strings are to be treated as full text and which as exact match, which are date values, in which language should the stemming be performed).

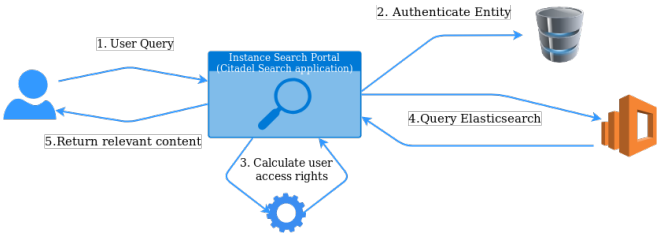


**Fig. 2:** Citadel Search access control flow.

### C. Access Control

As mentioned in the introduction, access control is one of the biggest challenges that needs to be tackled. Both authorization and authentication have to be addressed at many levels.

Note that, as shown in Figure 2, Citadel Search aims at providing enterprise search capabilities **as a service**. Meaning that the users of the system are entities (e.g. services with their own users) responsible for the content they index, and therefore, in charge of controlling who has access to what. Citadel will authenticate the entity (e.g. service), and provide the necessary mechanisms to enforce each particular access control model.

In Figure 2 the workflow that is followed to process a search query from a user is shown. Nonetheless, before explaining each one of the steps in detail, note that there is a first step that is carried out (but only once and by the entity). This is the step needed to authenticate the entity (e.g. service), that will later on query the instance in the name of its users. This authentication happens in the administration web interface. The log in process can be done using basic authentication or any of the OAuth clients that have been integrated within Invenio [1] [6]. In addition, a Python signal allows the customization of the authorization rules. Once the entity is authenticated and authorized, it will obtain a token, namely an API key, that will be used to authenticate itself when interacting with RESTful API.

Coming back to the steps shown in Figure 2:

1) User Query: The user will query for the terms that she is looking for.
2) Authenticate the entity: The back-end (RESTful API) will authenticate the entity based on the API key that is passed in the *"Authorization"* HTTP header.
3) Calculate user access rights: Based on the users rights (e.g. groups to which he belongs) the entity will calculate the access rights. Note that this step is the entity's responsibility. However, later on this will be converted to an Elasticsearch filter that will enforce CRUD permissions over the documents.
4) Query Elasticsearch: Citadel Search (back-end) will receive the user's query and its access rights. It will

[1]Even though as of 2019 it is restricted the aforementioned methods, it is in the roadmap to support other ones such as SAML.

translate them into an Elasticsearch query + filter and will perform the search query. It returns the results to the entity.
5) Return relevant content to the user: The entity will mold the results according to the specific needs and it is then shown to the user.

Therefore Citadel Search enforces access control at five levels:

- Web User Interface: The web interface must only be accessed by authenticated users. The administration panel should only be accessed by authorized users.
- RESTful API: Only authenticated and authorized users should be able to query the exposed endpoints.
- Instance level: Owners or administrators of the instance, i.e. one or more users of the system, must have superuser privileges over the whole instance.
- Collection level: Owners of the collection, i.e. one or more users of the system, must have superuser privileges over a specific collection.
- Document level: A document must provide CRUD access control. This means that create, read, update and delete operations might be allowed for a potentially different set of users.

### D. Search

One of the requirements of any enterprise search is to provide an advanced method to perform queries. Citadel Search uses the same query parser then Elasticsearch's *query string*. It allows to query for specific content in specific fields, the use of wildcards (both * and *?*), range filters for numeric and date values, and the construction of complex hierarchies of *AND* and *OR* statements. Some examples can be found in Citadels official user documentation [7].

### E. Storage

As it has been mentioned in the *architecture* subsection, Citadel uses three types of storage: In-memory, relational, and document oriented.

The in memory database is used to store users' sessions and cache rendered pages.

The SQL relational database is used due to its transactional capabilities, which ensures data consistency. In addition they are usually highly reliable as compared to some NoSQL solutions. Almost all accesses to the relational database are primary key look ups, which are usually very efficient in databases.

Any other search queries are sent to the search engine cluster which provides much better performance than a relational database. The chosen search engine is Elasticsearch due to it being fully JSON-based, and thus it fits well together with storing records internally in the database as JSON documents. Furthermore Elasticsearch is highly scalable and provides very powerful search and aggregation capabilities, such as geospatial queries.

When indexing content Citadel Search offers two options: Direct indexing, which will directly index a record when handling a request, and thus make the record immediately available for searches. Alternatively, Citadel Search also supports bulk indexing which is significantly more efficient when indexing large number of records. The bulk indexing works by the application sending a message to the message queue, and at regular intervals a background job will consume the queue and index the records. Also, several bulk indexing jobs can run concurrently at the same time on multiple worker nodes and thus it can achieve very high indexing rates during bulk indexing.

Finally, files and binary content needs to be stored outside of the databases. Citadel Search, provides object storage by mimicking the Amazon's S3 storage API [8] (e.g. using the concept of *buckets*).

### F. Full-text search

An important feature of an enterprise search solution is to provide powerful search over the indexed documents' metadata. Nonetheless, it is also important to be able to search inside the documents content itself. For metadata only documents (e.g. certain engineering files, which only contain an identifier, a list of responsible people names and a free text field with a description) this is already made possible by the Invenio Framework built-in indexing of documents.

However, for binary [2] files its contents need to be extracted. For this purpose Apache Tika, a toolkit to detect and extract metadata and text from over a thousand different file types, such as PPT, XLS, and PDF, among others, provides OCR (Optical Character Recognition) capabilities. Citadel Search stores the metadata of the file in what is called *record* along with a reference to the bucket where the file was stored, while the extracted content of the file is indexed in order to make it searchable. Note that the file will be stored, through the bucket, in the configured object storage (local, S3, etc.).

### G. Aggregation

A drawback of having different instances, with different data and access control models, is that it is challenging to provide a single point of entry to search all the collections. It is not user-friendly having to perform the same query in two or more places to find a document, it goes against the principle of *enterprise search*.

However, using the concept of *alias* that Elasticsearch provides, and exposing the different clusters. Many collections can be exposed to a central entry point, the *global search portal*. Note that in order to make this possible some data model conventions must be followed:

- All searchable content must be set inside the *_data* field. This convention helps when querying more than once collection (instance). It ensures that only

[2]For simplicity, understanding as binary files everything that is more than plain metadata in JSON format. Including non-binary files such as txt.

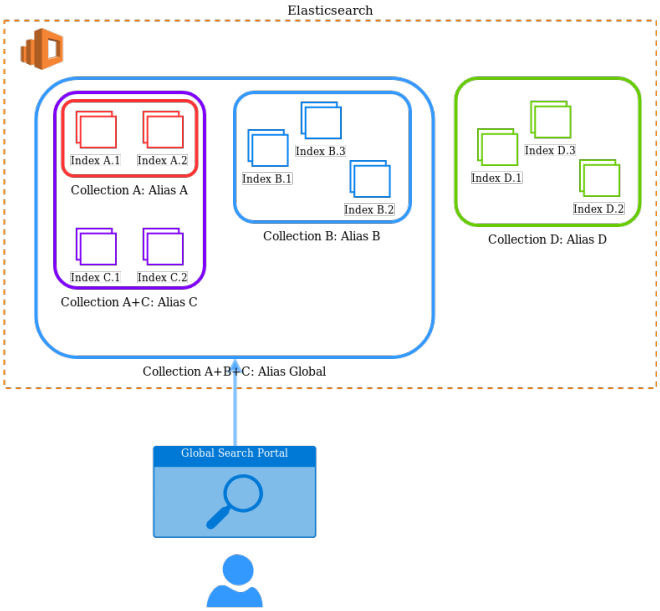


**Fig. 3:** Using Elasticsearch aliases to provide instances' isolation.

globally relevant fields will be queried, avoiding collection-specific metadata that might produce non-desired results.
- Collection specific filter fields, i.e. those that are not useful for full-text search such as dates must be placed outside the *_data* field.
- Since every instance can have different access control models (e.g. using groups, single user permissions by email address, or a simple public/private mechanism), the global entry point will only return as result those that are public and/or follow CERN's conventions (which is the use of e-groups).

Another option to save computing resources usage when the collections of the instances are small, or even in order to avoid exposing Elasticsearch instances to the internet and having to set up cross-cluster search, is to use the same concept of alias to isolate instances inside the same Elasticsearch cluster. This is shown in 3.

As it can be seen, inside the same Elasticsearch cluster there are four collections: A, B, C and D. Each collection provides an alias which means that querying that alias will return only the enclosed collections. For example entity A will query *A* and will obtain results belonging to A. In addition, C has an agreement with A, and therefore a query to *C* will include results belonging to A and C. When a user queries the global search portal, documents belonging to *A*, *B* and *C*. However, documents from *A* will not be duplicated since an alias does not imply copying over data, but just referencing it. Finally, note that entity D, did not wanted to have their contents searchable in the global entry point (or did not comply with the data and access control conventions) and therefore the alias does not include its collection.

### H. Harvesting

The idea behind *as a service*, is that the content is pushed by each interested entity to their own instances and then, upon agreement, used by the global search portal. Nonetheless, it is neither optimal nor user friendly to ask every website owner to push its content to its own search instance. Therefore, CERN has built a web crawler based on Scrapy.

This crawler is able to extract the content of all public websites, render those that are built using JavaScript frameworks (using Splash), and finally index it (in a specific search instance) in a format that complies with the CERN conventions and in consequence making it available in the global search portal.

## III. Discussion and Future Work

Citadel Search is new project that aims to became a powerful enterprise search solution, in order to tackle as many use cases as possible. It has been tested already on collections of approximately half a million documents and performs well. Nonetheless there is still much room for improvements.

In the front-end world the future work includes providing a user-friendly state of the art web interface that includes features such as: auto-complete and *"did you mean"* suggestions, promoted results and complex term highlighting. Currently a pre-alpha version has been developed using the React-SearchKit [9].

Currently, a change in the data model implies the edition of the JSON schema and its related Elasticsearch mapping, invoking its creation in the Elasticsearch cluster, setting up its aliases, and finally carrying out the data re-indexing if needed. However, this is, once again, not very user friendly. The plan is to enable administrators to create, modify and remove collections (JSON schema and Elasticsearch mapping) and re-index them (if no conflicts are found between versions) from the web user interface.

In terms of query relevance, CERN faces a challenge due to the homogeneous nature of its content (not in the data model, but in its content). Since CERN's main mission is physics related, it is obvious that some terms such as *LHC*, *ATLAS*, *CMS* or *Higgs* will appear in many of the documents, and the word *CERN* will appear in many more, if not in all. Therefore some fields such as the *title* need to be given more importance. This can be achieved through field *boosting* at query time. However, with schema evolution this might become a challenge to keep and maintain. Therefore, it is in the plan to perform an integration with a suitable analytics platform (e.g. Matomo) and feed the ranking system with it. In addition, machine learning techniques such as *learning to rank* [10] will be explored.

Finally, the harvesting crawler could be enhanced in order to feed private content of websites. However, the challenge

there is to set up the correct access rights. Even though websites are provisioned in CERN servers and access control files such as *.htaccess* or Apache's configuration files could be parsed, it would require a great amount of effort to develop, and even more to maintain since web technologies evolve fast and change faster. Therefore, the plan is to set those access rights in a central repository from where the crawler could obtain that information.

## IV. Source Code

Citadel Search is Open Source under MIT license. The source code is available at `https://github.com/inveniosoftware-contrib/citadel-search`.

## References


[1] D. Hawking. Challenges in enterprise search. In *Proceedings of the 15th Australasian database conference-Volume 27*, pages 15–24, 2004.

[2] Invenio Software. Invenio framework. *online:* `https://inveniosoftware.org/products/framework/`.

[3] Invenio Software. Infrastructure architecture. *online:* `https://invenio.readthedocs.io/en/latest/architecture/infrastructure.html`.

[4] Citadel Search. Citadel search openshift template. *online:* `https://gitlab.cern.ch/webservices/cern-search/citadel-search-openshift`.

[5] JSON Schema ORG. Json schema specifications. *online:* `https://json-schema.org/specification.html`.

[6] Invenio Software. Infrastructure oauth client. *online:* `https://invenio-oauthclient.readthedocs.io`.

[7] Citadel Search. Citadel search query. *online:* `https://citadelsearch.docs.cern.ch/usage/operations/#query-documents`.

[8] Amazon. Amazon s3. *online:* `https://docs.aws.amazon.com/AmazonS3/latest/API/Welcome.html`.

[9] Invenio Software. React searchkit. *online:* `https://inveniosoftware.github.io/react-searchkit/`.

[10] 019s. Elasticsearch learning to rank. *online:* `https://elasticsearch-learning-to-rank.readthedocs.io/en/latest/`.


## V. Acknowledgements


Building an enterprise search system and making it into an production service within only a year would not have been possible without the help of Ismael Posada Trobo, Eduardo Alvarez Fernandez, Andreas Wagner and Bruno Silva da Sousa for their invaluable knowledge on enterprise search and CERN specific insights. And to Diego Rodriguez Rodriguez, Alexandros Ioannidis, Nicola Tarocco, Lars Holms Nielsen, Esteban Gabancho, Jose Benito Gonzalez Lopez and the rest of the Invenio Software team for their patience and time to help understanding the Invenio Framework and again their invaluable knowledge in large-scale repositories deployment.