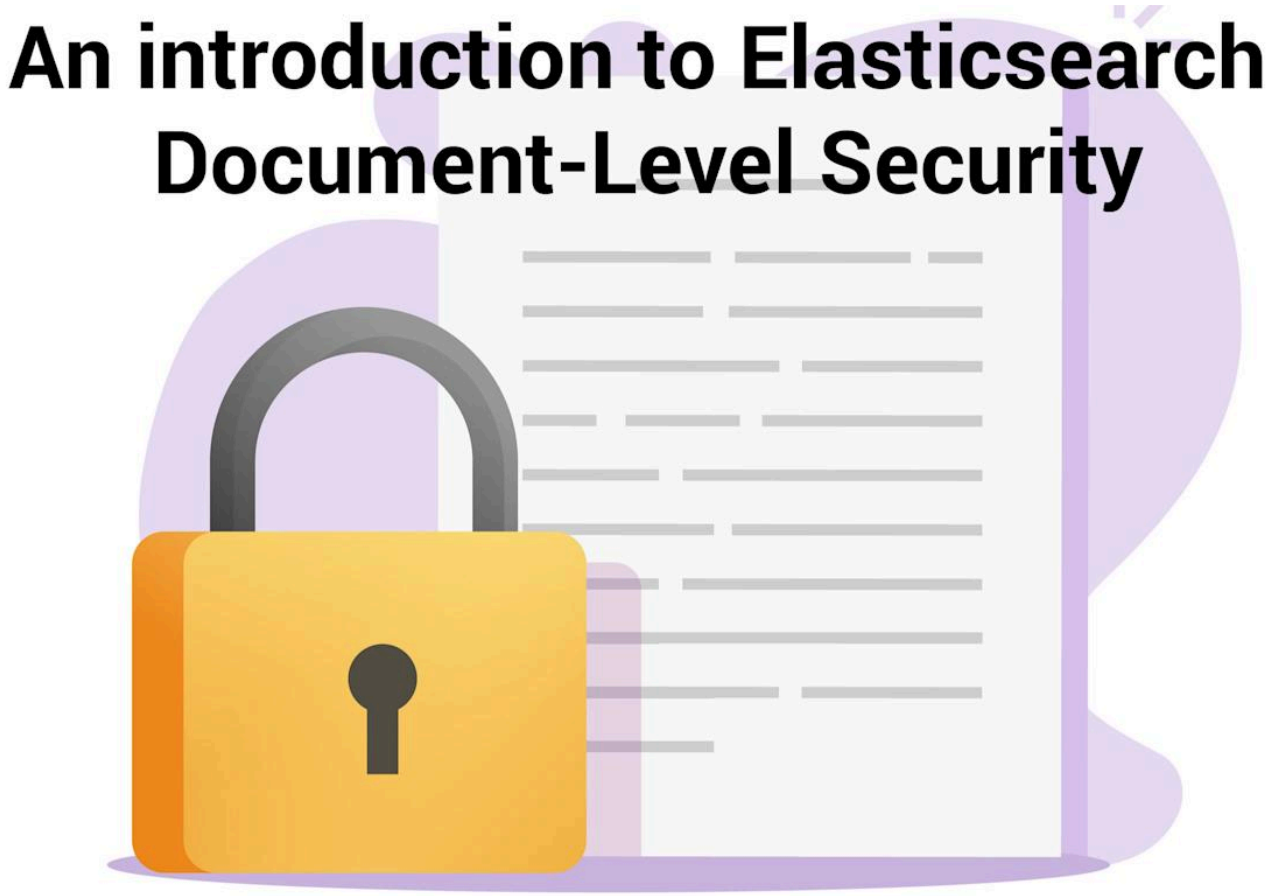
Elasticsearch Document-Level Security

Jochen Kressin · 2020-10-13



Search Guard provides both Document- and Field-Level Security for Elasticsearch and Kibana. Document-level security controls what documents a user can see in an Elasticsearch index. Field-level security does the same for fields. This article looks at Document-level security, how it is different from filtered aliases, and how it can improve control over your data.

Access control on Document-level

Search Guard provides role-based access control for Elasticsearch and Kibana. You can define which roles have access to indices and what they can do with those indices' documents. However, often this is not granular enough.

Suppose you store many different documents in an index. For example, log files from different systems like productions, staging, or development. Or PII data where access is governed by GDPR compliance regulations. You want to be able to control which documents are visible for what role.

Elasticsearch filtered aliases

In Elasticsearch, you can add a filter to any index alias. The filter is a regular Elasticsearch filter query which, as the name implies, filters documents from the result set. Isn't that what we are looking for? Not quite.

Filtered aliases seem like an excellent solution to the problem, but are inherently insecure and inflexible. An alias is just a pointer to a concrete, underlying index. To access data, a user needs to have access permission to this concrete index. Instead of using the alias that filters out documents, the user could just use the concrete index name, circumventing the filter altogether.

The power of Document
-Level security

Instead of using a filtered alias which operates on the Elasticsearch layer, Document-level security (DLS) goes all the way down to Lucene. What does that mean?

> It does not matter if you use an index or an alias. DLS always applies.

It does not matter if you use an index or an alias. DLS always applies. For Elasticsearch, it looks like filtered documents are not existent / deleted. Thus you can use all Elasticsearch features like aggregations in conjunction with DLS without limitations. DLS queries are flexible and support variable substitution for implementing dynamic queries based on user attributes as well.

A simple example

Imagine you have PII documents in an index called *humanresources* like:

```
{
    "FirstName": "ELLIOT",
    "LastName": "CASTRO",
    "Designation": "SVP",
    "Salary": 154000,
    "Address": "327 West Orchard RoadMiami Gardens, FL 33056",
    "Gender": "Male",
    "Age": 61,
    "MaritalStatus": "Married",
    "Interests": "Train Collecting,Collecting Artwork,Reading,Playing music,Aircraft Spotting"
}
```

The *Designation* field can contain values like *SVP*, *Manager*, *DevOps*, *Developer*, *QA Engineer* etc.

Our goal is to set up a role that can read (but not write) those documents, but we want to filter out all records where the Designation is, say, *SVP*.

We set up a Search Guard role and add a Document-level security query for the *humanresources* index pattern like:

```
sg_dls:
  cluster_permissions:
    - SGS_CLUSTER_COMPOSITE_OPS_RO
  index_permissions:
    - index_patterns:
      - 'humanresources'
      allowed_actions:
        - SGS_READ
      dls: '{ "bool": { "must_not": { "match": { "Designation": "SVP"  }}}}'
```

(Of course, you can also use the REST API or the Kibana Configuration UI to set up the role).

This role's DLS query is a simple Elasticsearch query that filters all documents where the *Designation* field is *SVP*. This applies to the index and all aliases that point to it. Suppose you run, for example, an Elasticsearch term aggregation on the *Designation* field. For Elasticsearch, it looks like there are zero documents where *Designation* is *SVP*.

Dynamic DLS queries

Document-Level Security queries are dynamic in nature. Instead of using a simple, fixed query, you can leverage variable substitution to make the query dynamic.

This can be basic variables like the user name:

```
sg_dls:
  ...
  index_permissions:
    - index_patterns:
      - 'humanresources'
      allowed_actions:
        - SGS_READ
      dls: '{"term" : {"creator" : ${user.name|toJson}}}'
```

Or multi-valued attributes like the user's roles any attributes from LDAP, JSON web tokens or the internal users database.

```
sg_dls:
  ...
  index_permissions:
    - index_patterns:
      - 'humanresources'
      allowed_actions:
        - SGS_READ
      dls: '{"terms" : { "role" : ${user.roles|toJson}}}'
```

Watch out for our next article, which explains how to leverage variable substitution in DLS queries to the fullest.

Where to go next

- Read our documentation about Document-Level security for Elasticsearch
- Read our blog post about attribute based document access
- Watch out for the next article about multi-valued user attributes, which unleashes the full power of DLS

Image: Shutterstock / Saxarinka

This article was published on the Search Guard blog: https://search-guard.com/blog/elasticsearch-document-level-security/