Document- and field-level security with Search Guard

Jochen Kressin · 2016-08-17

Document- and field-level security gives you very fine-grained control over which documents and fields a certain user is allowed to see.

Document-level security (DLS) means filtering out complete documents of a search result based on the role of the user. Field-level security (FLS) means filtering out fields from documents based on the role of the user.

Scenario

Let's assume we have an index called *human*resources_, which contains sensitive information about employees, including their actual salary. A document in this index might look like this:

```
{
    "_type" :     "employee",
    "first_name": "John",
    "last_name":  "Doe",
    "age" :        45,
    "department":  "Management",
    "salary":      120000
}
```

We want to be able to restrict access to these documents, based on the role of the user. Let's assume we have the following roles:

- head*of*hr
- hr_employee
- hr_trainee

We want to configure the following permissions:

- The role *head*of*hr* is able to see all documents, and all field
- The role *hr*employee_ is able to see only documents of employees that do no have a management position
- the role *hr*trainee *can also see only non-management documents, but only the field* first*name*, *last*name *and* department_.

In order to do so, we can use **document-level security** for **filtering out documents** based on a certain criterion, and **field-level-security** for defining the **visible fields**.

Both settings are applied per-role and per-index, and are configured in sg*roles.yml. We start by setting up the three roles, and initially allow complete access to the index* human*resources*:

```
head_of_hr:
  indices:
    'human_resources':
      '*':
        - '*'
hr_employee:
  indices:
    'human_resources':
      '*':
        - '*'
hr_trainee:
  indices:
    'human_resources':
      '*':
        - '*'
```

Document-level security

Now we want to filter out all documents where the *department* field is *Management*, and we want to do so for the *hr*employee*and the* hr*trainee* role.

Document-level security works by simply specifying a regular Elasticsearch query, using the query DSL. Only documents that match this query are included in the search result. We use a Bool query with a must_not clause for this (more on Bool queries can be found in the Elasticsearch docs).

```
{
  "query": {
    "bool": {
      "must_not": { "match": { "department": "Management"  }}
    }
  }
}
```

We now can use this exact query for configuring the document-level security setting of Search Guard:

```
hr_employee:
  indices:
    'human_resources':
      '*':
        - '*'
    _dls_: '{"query": { "bool": { "must_not": { "match": { "department": "Management"  }}}}}'
```

Note that the **dls** parameter is on the same level as *human*resources*, so it applies for this index only. Whenever a user with role* hr*employee* tries to access documents, the specified dls-query is executed and the results filtered accordingly.

For Search Guard 5, please remove the leading "query" key from the DSL query:

```
_dls_: '{ "bool": { "must_not": { "match": { "department": "Management"  }}}}'
```

DLS and FLS are applied on Lucene-level, not Elasticsearch-level. This means that the final documents handed from Lucene to Elasticsearch are already filtered according to the DLS and FLS settings for added security.

Field-level security

Field-level security is even simpler. You just need to specify which fields the document should contain, a.k.a. white-list these fields. FLS is configured on the same level as DLS. In our example, we want to limit the access of the *hr*trainee *role also to non-management documents, but only the fields* first*name*, *last*name *and* department_ should be visible. The settings look like this:

```
hr_trainee:
  indices:
    'human_resources':
      '*':
        - '*'
    _dls_: '{"query": { "bool": { "must_not": { "match": { "department": "Management"  }}}}}'
    _fls_:
      - 'first_name'
      - 'last_name'
      - 'department'
```

Advanced: Username substitution in DLS queries

Sometimes it is handy to use the authenticated users username in DLS queries. This is also possible with Search Guard, by using the variable *{user.name}*. You can use it in any part of the DLS query, and also multiple times if needed. What can you do with that?

Let's extend our scenario a little bit. Let's say our employee documents have an additional field *owner*, which contains the username of the employee:

```
{
    "_type" :     "employee",
    "owner":      "jdoe",
    "first_name": "John",
    "last_name":  "Doe",
    "age" :        45,
    "department":  "Management",
    "salary":      120000
}
```

Every employee should be able to see his/her own document in the index. Given that the *owner* field contains the username that was used to authenticate against Search Guard, we can use this DLS query for the *employee* role:

```
employee:
  indices:
    'human_resources':
      '*':
        - '*'
    _dls_: '{"term" : {"owner" : "${user.name}"}}'
```

Summary

Document- and field-level security can be used to implement very fine-grained control over which documents and fields a particular role can see. By defining a regular query using the Elasticsearch query DLS, you can filter documents on a per-index and per-role basis. By defining FLS fields, you can white-list the fields a particular role can see. For DLS, you can also use the *${user.name}* variable in the query, which gets substituted with the authenticated user's username.

Image: Shutterstock / N K

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