Configuring Document-Level Security for Cloudera Semantic Search

> Note:
> The Cloudera Semantic Search service on Cloudera on premises is in Technical Preview and is under development. You must not use Technical Preview features in production environments.

Introduction

This document outlines the architecture, implementation steps, and transition strategy for enabling Document-Level Security (DLS) in OpenSearch using Apache Ranger. By moving beyond index-level permissions, this solution allows administrators to restrict access to specific documents within an index based on the user's assigned Ranger roles. This guide is designed to help field teams deploy the solution and assist customers in managing the transition without disrupting existing search workloads.

Core architecture and approach

The OpenSearch Ranger plugin implements DLS via **Query Interception**. Rather than modifying low-level storage readers, the plugin intercepts the user's incoming search request and seamlessly rewrites the query to enforce security rules before it executes.

- **Authentication and context:** The plugin relies on the native opensearch-security plugin to authenticate the user.
- **Role Resolution:** The user's identity is passed to Apache Ranger to retrieve their assigned security roles.
- **Query Rewriting:** The plugin wraps the user's original search query in a secure filter. It appends a mandatory check against a specific field within the documents (for example, ranger_role).
- **Execution:** OpenSearch executes the modified query, returning only the documents that match both the user's search intent and their authorized roles.

Implementation requirements

To utilize this feature, the following components must be aligned:

- **Plugin Deployment:** The custom Ranger OpenSearch plugin must be installed on all OpenSearch nodes and ordered to execute strictly after the opensearch-security plugin.
- **Schema Design:** All protected documents must include an array field named ranger_role.
- **Data Ingestion Pipeline:** Upstream applications indexing data into OpenSearch must be updated to append the correct ranger_role values to incoming documents based on business logic.
- **Ranger Policy Configuration:** Administrators must define the corresponding roles within the Apache Ranger UI and assign them to the appropriate users or groups.

Transition plan (zero-downtime rollout)

Introducing strict DLS into an existing production environment requires a phased approach. Enforcing security on a field that does not yet exist in legacy documents will cause those documents to disappear from search results. To prevent this, we utilize a backward-compatible rollout strategy.

Phase 1: Deploying backward-compatible plugin

The initial deployment of the plugin will use a "default-allow for legacy data" configuration. The intercepted query will require that a document either matches the user's role OR does not contain the ranger_role field at all. This ensures legacy data remains accessible while new, secured data is strictly filtered.

Phase 2: Background data backfill

Once the plugin is live, field teams or cluster administrators will run an _update_by_query script to tag existing documents with the appropriate legacy roles. This process runs in the background.

Standard backfill script

JSON

```
None
POST
/target-index/_update_by_query?conflicts=proceed&wait_for_completio
n=false&scroll_size=1000
{
  "query": {
    "bool": {
      "must_not": {
```

```
        "exists": {
          "field": "ranger_role"
        }
      }
    }
  },
  "script": {
    "lang": "painless",
    "source": "ctx._source.ranger_role = params.roles",
    "params": {
      "roles": ["legacy_read_access"]
    }
  }
}
```

Phase 3: Transition to strict enforcement (optional)

Once all legacy documents have been successfully backfilled with a ranger_role via Phase 2, the customer can elect to update the plugin configuration to "strict mode." This removes the backward-compatible logic, ensuring that any document missing the ranger_role field is completely hidden from all users.

Known limitations

Field teams and customers should be aware of the following technical constraints:

- **Hardcoded field**: Currently the field name **"ranger_role"** is hardcoded and can't be configured as required.
- The current DLS implementation is only applicable on search queries. [index/_search and index/_count]
- **Max Clause Count Exceptions:** If an organization assigns an unusually high number of roles to a single user (for example, thousands of distinct roles), the intercepted query may exceed the Lucene indices.query.bool.max_clause_count limit. Roles should be consolidated logically.
- Admin, kibanaserver, opensearch and these users are bypassed from the DLS, along with system index querying entirely bypassed from DLS. And this part of the code is hardcoded into the plugin. Use settings in opensearch.yaml to read the values.
- During Phase 1 of the transition plan, legacy documents lacking the ranger_role field are accessible to any authenticated user capable of querying the index.

- If a user authenticates successfully but has zero roles assigned in Ranger, an improperly formed query could inadvertently return results or throw cluster errors.
- Applying DLS to parent documents does not automatically apply the exact same filter to isolated nested queries unless explicitly handled in the rewriting logic.
- Wrapping every search query adds slight parsing overhead.

Operational support

For field teams encountering issues during deployment:

1. Use the OpenSearch Profile API (GET /index/_search -H 'Content-Type: application/json' -d'{ "profile": true, "query": { "match_all": {} }}') using a test user account to view the rewritten query tree and confirm the ranger_role filter is applied correctly.
2. Monitor the OpenSearch task management API (GET /_tasks) during the Phase 2 backfill to ensure the cluster is not overwhelmed by the Painless script execution. Adjust requests_per_second if CPU utilization spikes.

Example

The following examples demonstrate how to create an OpenSearch index configured for Document-Level Security and how to ingest documents with the required security field.

Create a production index

```
None
curl --cacert cm-auto-global_cacerts.pem -u admin:admin -X PUT
"https://${OPENSEARCH_HOST}:9200/prod-documents" -H "Content-Type:
application/json" -d '
{
"mappings": {
"properties": {
"title": { "type": "text" },
"department": { "type": "keyword" },
"ranger_role": { "type": "keyword" }
}
}
```

```
}'
```

Add new protected data

None

```
curl --cacert cm-auto-global_cacerts.pem -u admin:admin -XPUT
"https://${OPENSEARCH_HOST}:9200/prod-documents/_doc/1" -H "Content-Type:
application/json" -d '

{

"title": "Secret Quantum Engine Blueprint",

"department": "Engineering",

"ranger_role": ["engineering_read"]

}'
```