New Java API vs old API

API client (from 6.8 to 7.17) - 8.x

In Elasticsearch provided High level rest client(HLRC) now it completely removed after 7.17

Transport client

But in ElasticSearch 8.x **Java Api client** is available

still we can enable compatibility mode HLRC version 7.17 in 8.x , sample code is available in below reference document

but in this mode HLRC sends additional headers that instructs **ES8.x to behave like 7.x server**

We can use Es 7.12 builder style code pointing to ES 8.12 Es search with help of migrator

Then required dependencies to go with above approach is

we can use **HLRC version 7.x with the Java API Client version 8.x**

co.elastic.clients:elasticsearch-java:8.12.2

org.elasticsearch.client:elasticsearch-rest-high-level-client:7.17.4

reference **:https://www.elastic.co/guide/en/elasticsearch/client/java-api-client/current/migrate-hlrc.html**

Reference : https://www.elastic.co/guide/en/elasticsearch/client/java-api-client/7.17/migrate-hlrc.html.

https://www.elastic.co/guide/en/elasticsearch/client/java-api/6.8/transport-client.html#transport-client

Observation : **Need to re-write the Java api client over Transport client** ..

8.x api client observations:

Elastic search api client

it is further organised into feature groups. These groups are called "NameSpaces"

like : API conventions , commons options, connector apis , graphExplore, Machine Learning , search api ect...

api client come with 2 flavors : Blocking and Asynchronous clients

Synchronous (blocking) -> we use ElasticsearchClient

Asyc (non-blocking)-> ElasticsearchAsyncClient

How you create builder object in the java api client ES8.x

https://www.elastic.co/guide/en/elasticsearch/client/java-api-client/8.12/building-objects.html

Now ES api avialable with all variant queries,aggregation, field mapping , analyzer under one union its easy to use .

Java api client ingestion (json files)

https://www.elastic.co/guide/en/elasticsearch/client/java-api-client/8.12/loading-json.html

Indexing before .. 6.8 ES

-> to index a doc we need to pass json . there are several different ways to json document

using jsonString , jackson ObjMapper, Xcontentfactory.jsonBuilder to construct a Byte Array .

https://www.elastic.co/guide/en/elasticsearch/client/java-api/6.8/java-docs-index.html

bulk

https://www.elastic.co/guide/en/elasticsearch/client/java-api/6.8/java-docs-bulk.html

Now in 8.x

you can use java objects to pass or json too

single doc ingestion

https://www.elastic.co/guide/en/elasticsearch/client/java-api-client/8.12/indexing.html

Bulk ingestion

https://www.elastic.co/guide/en/elasticsearch/client/java-api-client/8.12/indexing-bulk.html

Search in 6.x

QueryBuilder

client. prepareSearch which we are using in the current applications

```
      .prepareSearchScroll ..
 client.prepareMultisearch() ...
```

simple single doc search

```
SearchResponse response = client.prepareSearch("index")
       .setSearchType(SearchType.DFS_QUERY_THEN_FETCH)
       .setQuery(QueryBuilders.termQuery("multi", "test"))
       .get();
SearchResponse response = client.prepareSearch().get();
```

In 8.x search sample Functional style

```
GetResponse<Product> response = esClient.get(g -> g
    .index("products")
    .id("bk-1"),
    Product.class
);
```

we have different search techniques in 8x

https://www.elastic.co/guide/en/elasticsearch/client/java-api-client/8.12/searching.html

it has many variant structuring ,Metrics, bucketing etc..

8.x **builder style**

```
TermQuery query1=  QueryBuilders.term().field("status").value(1).build();
                Query queryTerm1= TermQuery.of(t->t.field("status").value(1))._toQuery();
                Query termQueryModel=TermQuery.of(t->t.field("part_type").value("MODEL"))._toQuery();
Query partsQuery= queryBuilder.must(termQueryModel).should(termQuer) .build()._toQuery();
InnerHits innerHits = InnerHits.of(i-> i.ignoreUnmapped(false).docvalueFields(docValueFieldslist));
                            SearchRequest request1= new SearchRequest
                          .Builder().
                          index(part_index)
                          .source(config)  .query(partsQuery)
                                        .build();
   SearchResponse<Object> response=client.search(request1, Object.class);
                              List<Hit<Object>> hits=response.hits().hits();
                  for(Hit<Object> hit:hits)
                            partPojoList.add(hit.source());
```

1

[1] Sandeep Kumar Rayala

[2]

[2]