Common Java Client for Elasticsearch/OpenSearch (elk-client)

Introduction:

By using simple approach we are trying to solve the pain points of SDKs. This simple elk-client interacts with Elasticsearch/OpenSearch for all CRUD operations in convenient manner.

It uses the same query which is tested in Dev Tools and replaces the variables with desired values passed by the user.

The ElkRequest uses builder pattern and creating request object for Elasticsearch/OpenSearch CRUD operations is very simple.

Problem with SDKs:

- Always lags with latest versions of the Software
- Difficult to build the query
- Learning curve is steep
- Difficult to debug Query built by SDK

Source code : https://git.nexgen.neustar.biz/ipi/elk-client.git

Maven Dependency : Include following dependency in the pom.xml

```
<dependency>
    <groupId>com.tu</groupId>
    <artifactId>elk-client</artifactId>
    <version>1.0.0</version>
</dependency>
```

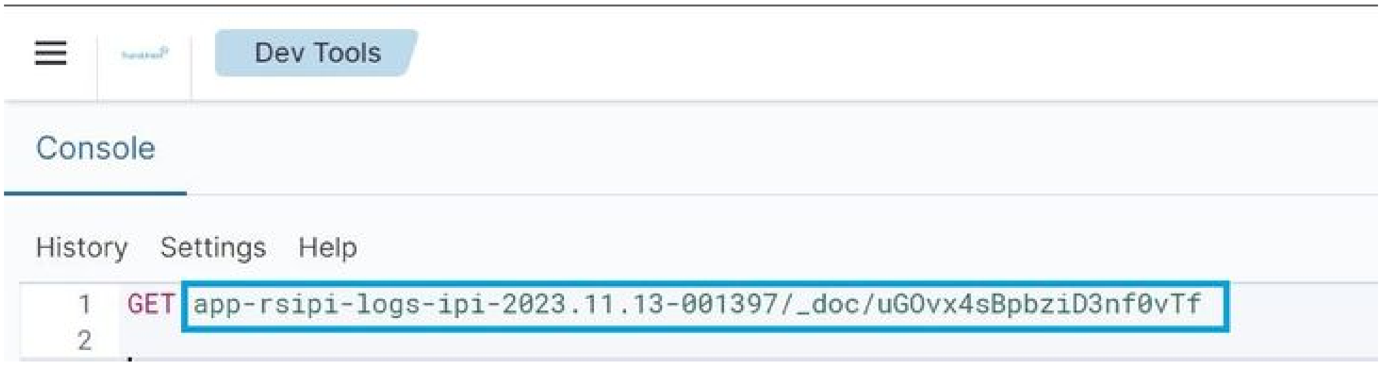
Request :

By calling newBuilder() method, We can configure the ElkRequest.

```
ElkRequest elkRequest = ElkRequest.newBuilder()
                .scheme()     //http or https
                .hostname()   //Hostname
                .port()       //Server port
                .uriSuffix()  //indexes etc, Ex: index/_search?
pretty&size=0
                .username()   //Username
                .password()   //Password
                .query()      //Query
                .queryParam() // Variables : key-value
                .sslContext() // SSLContext parametrs
                .httpRequestConfig() // Http request configs
                .POST()       // HTTP method : GET(), POST(), PUT(),
DELETE(). By default it will be GET().
                .build();
```

- **URI Suffix :** This is the part of URI which contains names of indexes and operations. This is the value from **Dev Tools**(As endpoint)

Elasticsearch/OpenSearch UI Login Menu Management Dev Tools



```
GET app-rsipi-logs-ipi-2023.11.13-001397/_doc/uGOvx4sBpbziD3nf0vTf
```

- **Query :** The string which is used in Dev Tools to query the ELK/Open Search.

Query Example:

```
{
  "size": 0,
  "query": {
    "bool": {
      "must": [
        {
          "range": {
            "event_time": {
              "from": "2023-11-20T01:00:00.000Z", // change to "from":
"@from_timestamp"
              "to": "2023-11-20T01:10:00.000Z",   // change to "to":
"@to_timestamp"
              "include_lower": true,
              "include_upper": false
            }
          }
        }
      ]
    }
  }
}
```

To pass anything as a variable to the query, append with @ followed by variable name.

Example : `@from_timestamp` `@to_timestamp`

- **Query Parameters :** This is a key - value pair and key is nothing but variable name.

  Example:

  ```
  queryParam("from_timestamp", "2023-11-20T01:00:00.000Z")
  
  queryParam("to_timestamp", "2023-11-20T01:10:00.000Z")
  ```

- **SSL Context :** Instances of this class represent a secure socket protocol implementation which acts as a factory for secure socket factories or SSLEngines. This class is initialized with an optional set of key and trust managers and source of secure random bytes.
  Every implementation of the Java platform is required to support the following standard SSLContext protocols:TLSv1, TLSv1.1 and TLSv1.2

Default elk-client uses following implementation:

```
TrustStrategy acceptingTrustStrategy = (X509Certificate[] chain, String
authType) -> true;
        SSLContextBuilder sslContextBuilder = SSLContexts.custom()
                .loadTrustMaterial(null, acceptingTrustStrategy);
        SSLContext sslContext = sslContextBuilder.build();
```

Reference : https://docs.oracle.com/en/java/javase/17/docs/api/java.base/javax/net/ssl/SSLContext.html

- **Http Request Configurations:** Immutable class encapsulating request configuration items.

Default elk-client uses following configuration:

```
RequestConfig requestConfig = RequestConfig
                    .custom()
                    .setConnectionRequestTimeout(300000)
                    .setConnectTimeout(300000)
                    .setSocketTimeout(300000)
                    .build();
```

Reference : https://hc.apache.org/httpcomponents-client-4.5.x/current/httpclient/apidocs/org/apache/http/client/config/RequestConfig.html

Client : ElkClient executes the request with classname. The response entity will be mapped to the provided object type.

```
ElkResponse<object_type> elkResponse = ElkClient.execute(elkRequest,
object_class)
```

Example:

```
ElkResponse<Root> elkResponse = ElkClient.execute(elkRequest, Root.
class);  // Object type is Root
Root root = elkResponse.getEntity() // getEntity method returns the
Root object
```

Response : ElkResponse contains following variables.

```
public class ElkResponse<T> {
    private final int statusCode;
    private final T entity;
    private final String message;
    private final Exception exception;
    }
```

`statusCode` : Http status code. incase of exception value will be 0.

`entity` : Output from the ELK/ OpenSearch of type T.

`message` : This contains reason for failure. Ex: `No such host is known (hostname)`

`exception` : Exception details

Test Cases :

```
package org.testelk;

import com.tu.elk.client.ElkClient;
import com.tu.elk.dto.ElkResponse;
import com.tu.elk.request.ElkRequest;
import org.apache.http.client.config.RequestConfig;
import org.apache.http.conn.ConnectTimeoutException;
import org.apache.http.ssl.SSLContextBuilder;
import org.apache.http.ssl.SSLContexts;
import org.apache.http.ssl.TrustStrategy;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.testelk.dto.Root;

import javax.net.ssl.SSLContext;
import java.security.KeyManagementException;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.cert.X509Certificate;

import static org.junit.jupiter.api.Assertions.*;

class ELKClientTest {

    private static ElkRequest.Builder requestBuilder;

    @BeforeEach
    void init() {
        requestBuilder = ElkRequest.newBuilder()
                .scheme("https")
                .hostname("olselkprod-coordinator-01.laas.onedev.
neustar.biz")
                .port(9200)
                .username(user_name)
                .password(password);
    }

    @Test
    void getSuccessCall() {
        ElkRequest elkRequestGet = requestBuilder
                .uriSuffix("app-rsipi-logs-ipi-2023.11.13-001397/_doc
/ng7Dx4sBpbziD3nff_WM")
                .GET()
                .build();
        ElkResponse<String> responseGet = ElkClient.execute
(elkRequestGet, String.class);
        assertNotNull(responseGet.getEntity());
        assertEquals(200, responseGet.getStatusCode());
    }

    @Test
    void postSuccessCall() {
        ElkRequest elkRequest = requestBuilder
                .uriSuffix("app-rsipi-logs-ipi-*/_search?pretty&size=0")
                .query(ELKConstants.LiveQuery)
                .queryParam("from_timestamp", "2023-11-20T01:00:00.000
Z")
                .queryParam("to_timestamp", "2023-11-20T01:10:00.000Z")
                .POST()
                .build();

        ElkResponse<Root> responsePost = ElkClient.execute(elkRequest,
Root.class);
        assertNotNull(responsePost.getEntity());
        assertEquals(200, responsePost.getStatusCode());
    }

    @Test
    void getFailureCall() {
        ElkRequest elkRequestGet = requestBuilder
                .username("dummy")
                .uriSuffix("app-rsipi-logs-ipi-2023.11.13-001397/_doc
/ng7Dx4sBpbziD3nff_WM")
                .GET()
                .build();


        ElkResponse<String> responseGet = ElkClient.execute
(elkRequestGet, String.class);
        assertNull(responseGet.getEntity());
        assertEquals(401, responseGet.getStatusCode());
    }

    @Test
    void exceptionCall() {
        ElkRequest elkRequestGet = requestBuilder
                .hostname("test-olselkprod-coordinator-01.laas.onedev.
neustar.biz")
                .username("dummy")
                .uriSuffix("app-rsipi-logs-ipi-2023.11.13-001397/_doc
/ng7Dx4sBpbziD3nff_WM")
                .GET()
                .build();


        ElkResponse<String> responseGet = ElkClient.execute
(elkRequestGet, String.class);
        assertNotNull(responseGet.getException());
        assertEquals("No such host is known (test-olselkprod-
coordinator-01.laas.onedev.neustar.biz)", responseGet.getMessage());
    }

    @Test
    void postWithSSLContext() throws NoSuchAlgorithmException,
KeyStoreException, KeyManagementException {

        TrustStrategy acceptingTrustStrategy = (X509Certificate[]
chain, String authType) -> true;
        SSLContextBuilder sslContextBuilder = SSLContexts.custom()
                .loadTrustMaterial(null, acceptingTrustStrategy);
        SSLContext sslContext = sslContextBuilder.build();

        ElkRequest elkRequest = requestBuilder
                .uriSuffix("app-rsipi-logs-ipi-*/_search?pretty&size=0")
                .query(ELKConstants.LiveQuery)
                .queryParam("from_timestamp", "2023-11-20T01:00:00.000
Z")
                .queryParam("to_timestamp", "2023-11-20T01:10:00.000Z")
                .sslContext(sslContext)
                .POST()
                .build();

        ElkResponse<Root> responsePost = ElkClient.execute(elkRequest,
Root.class);
        assertNotNull(responsePost.getEntity());
        assertEquals(200, responsePost.getStatusCode());
    }

    @Test
    void postWithHttpRequestConfig() {
        int timeout = 30000;
        RequestConfig requestConfig = RequestConfig
                .custom()
                .setConnectionRequestTimeout(timeout)
                .setConnectTimeout(timeout)
                .setSocketTimeout(timeout)
                .build();
        ElkRequest elkRequest = requestBuilder
                .uriSuffix("app-rsipi-logs-ipi-*/_search?pretty&size=0")
                .query(ELKConstants.LiveQuery)
                .queryParam("from_timestamp", "2023-11-20T01:00:00.000
Z")
```

```
                .queryParam("to_timestamp", "2023-11-20T01:10:00.000Z")
                .httpRequestConfig(requestConfig)
                .POST()
                .build();

        ElkResponse<Root> responsePost = ElkClient.execute(elkRequest, 
Root.class);
        assertNotNull(responsePost.getEntity());
        assertEquals(200, responsePost.getStatusCode());

    }

    @Test
    void postWithHttpRequestConfigConnectionTimeout() {
        int timeout = 3;
        RequestConfig requestConfig = RequestConfig
                .custom()
                .setConnectionRequestTimeout(timeout)
                .setConnectTimeout(timeout)
                .setSocketTimeout(timeout)
                .build();
        ElkRequest elkRequest = requestBuilder
                .uriSuffix("app-rsipi-logs-ipi-*/_search?pretty&size=0")
                .query(ELKConstants.LiveQuery)
                .queryParam("from_timestamp", "2023-11-20T01:00:00.000
Z")
                .queryParam("to_timestamp", "2023-11-20T01:10:00.000Z")
                .httpRequestConfig(requestConfig)
                .POST()
                .build();

        ElkResponse<Root> responsePost = ElkClient.execute(elkRequest, 
Root.class);
        assertNull(responsePost.getEntity());
        //responsePost.getException().printStackTrace();
        assertTrue(responsePost.getException() instanceof 
ConnectTimeoutException);

    }

    @Test
    void postWithHttpRequestConfigAndSSLContext() throws 
NoSuchAlgorithmException, KeyStoreException, KeyManagementException {
        int timeout = 30000;
        RequestConfig requestConfig = RequestConfig
                .custom()
                .setConnectionRequestTimeout(timeout)
                .setConnectTimeout(timeout)
                .setSocketTimeout(timeout)
                .build();
        TrustStrategy acceptingTrustStrategy = (X509Certificate[]
chain, String authType) -> true;
        SSLContextBuilder sslContextBuilder = SSLContexts.custom()
                .loadTrustMaterial(null, acceptingTrustStrategy);
        SSLContext sslContext = sslContextBuilder.build();

        ElkRequest elkRequest = requestBuilder
                .uriSuffix("app-rsipi-logs-ipi-*/_search?pretty&size=0")
                .query(ELKConstants.LiveQuery)
                .queryParam("from_timestamp", "2023-11-20T01:00:00.000
Z")
                .queryParam("to_timestamp", "2023-11-20T01:10:00.000Z")
                .httpRequestConfig(requestConfig)
                .sslContext(sslContext)
                .POST()
                .build();

        ElkResponse<Root> responsePost = ElkClient.execute(elkRequest,
Root.class);
        assertNotNull(responsePost.getEntity());
        assertEquals(200, responsePost.getStatusCode());

    }

}
```