PIEE
Procurement Integrated
Enterprise Environment

Kibana Queries for EDA

July 2023

Prepared by

Shuk Tse

This page intentionally left blank.

Table of Contents

1 Scope

This document provides an introduction to some commonly used Kibana queries for searching EDA (Electronic Document Access) data in PIEE (Procurement Integrated Enterprise Environment).

2 Assumptions

a) User has been granted Kibana access in the PIEE environment.
b) Kibana version is 6.6.0.
c) Kibana queries are run on the Dev Tools page.
d) This is a living document in which Kibana queries are continually being edited and updated.

3 Common Query Syntax Confusions

a) When entering a search parameter, straight quotation marks should be used as they indicate a phrase search.

> Example:
> If the search is for contracts.contract_number=”N609511700000”, you are literally searching for the text “N609511700000” (including the curly double quotes). If the search is for contracts.contract_number="N609511700000", you are only searching for N609511700000 (no quotes).

b) When running range queries on fields of type ‘date’, there is a difference between ‘now’ and ‘now/d”. ‘now’ is the current system time (in UTC) and is always resolved to Unix timestamp in millisecond (i.e. System.currentTimeMillis()). In particular, “now” is not affected by the time_zone parameter.

Search records for the past 24 hours:

```
"gte": "now-1d",
"lt": "now"
```

However, when using “now/d” (i.e. date math rounding), the date is rounded down to the nearest day and the provided time_zone will be considered.

Search records for the current day:

```
"gte": "now/d",
"lt": "now+1d/d"
```

Search records for the previous day:

```
        "gte": "now-1d/d",
        "lt": "now/d"
```

c) Understanding the ‘term’ and ‘match’ query usage is important. A ‘term’ query finds documents based on a precise value such as a contract number, record key or ID. The “term” query only searches for the exact term and does not analyze the search term. A “match” query accepts text/numerics/dates, analyzes them, and constructs a query. To search text field values, one should use the ‘match’ query.

d) Confusion often arise when using square ([ ]) and curly ({ }) brackets. Square brackets surround an array and contain a comma-separated list of values. Curly brackets surround an object and contain a separated list of name/value pairs.

NOTE: A name/value pair consist of a field name (in double quotes), followed by a colon (:), followed by the field value.

4 Kibana Version and Nodes

a) To retrieve the Kibana version number:

```
GET /
```

b) To display Kibana nodes:

```
GET _cat/nodes?v
```

NOTE: The ‘v’ parameter turns on verbose output.

```
GET _cat/nodes?h=ip,port,heapPercent,name
```

NOTE: The ‘h’ parameter forces only those columns to appear.

5 Listing Kibana Indices

a) To list all the Kibana indices in ascending order:

```
GET _cat/indices?v&s=index:asc
```

b) To list all Kibana indices which contains a wildcard phrase (e.g. *close*) and sort by the index field :

```
GET _cat/indices/*close*?v&s=index
```

6 Listing Kibana Aliases

a) To list all the Kibana aliases in ascending order:

```
GET _cat/aliases?v&s=alias:asc
```

b) To list all aliases which contains a specific phrase (e.g. history) in ascending order:

```
GET _cat/aliases/*history*?v&s=alias
```

7 Displaying Index Count and Statistics

a) To get the count of a Kibana index (e.g. contracts):

```
GET contracts/_count
```

b) To get index level statistics for a Kibana index (e.g. contracts):

```
GET contracts/_stats
```

8 Listing Field Mapping

a) To get mapping of a Kibana index:

```
GET doc_history_contract/_mapping
```

9 Match All Query

a) To view all documents for a particular Kibana index:

```
GET
{
  "sort": {
    "load_date": {"order": "desc"}
  },
  "query": {
    "match_all": {}
  }
}
```

10 Match Query

a) To find a matching value(s) in a specified Kibana index:

```
GET
{
  "query": {
    "term": {
      "contract_number": "N609511700000"
    }
  }
}
```

```
GET
{
  "query": {
    "bool": {
      "must": [
        {"term": {"contract_number": "HC10280100000"}},
        {"term": {"delivery_order_number": "HC10280100000"}}
      ]
    }
  }
}
```

11 Prefix Query

a) To list all contracts starting with ‘S0000’ in the contracts index:

```
GET
{
  "sort": [
    { "contract_number": { "order" : "asc"}}
  ],
  "query": {
    "prefix": {
      "contract_number": "S0000"
    }
  }
}
```

12 Finding Distinct Counts

a) To find all distinct values and their counts for a given column:

```
 GET
{
  "size":"0",
  "aggs": {
    "uniq_pds_status": {
      "terms": {"field":"pds_status"}
    }
  }
}
```

NOTE: The size key omits the results/hits (except the total).

13 Finding Duplicate Values

a) To find duplicate values and their counts for a given field:

```
GET
{
  "size": 0,
   "aggs": {
     "duplicateCount": {
     "terms": {
        "field": "contract_number",
        "min_doc_count": 2
      },
      "aggs": {
        "duplicateDocuments": {
        "top_hits": {}
      }
    }
   }
  }
}
```

NOTE: The above will return all values of the field contract_number which occur in at least two documents. Top hits aggregation will return the actual documents.

14 GEX Ingestion Listing

a) To list GEX ingestion by file status message over the last 24 hours:

```
GET
{
  "size": 0,
  "query": {
    "range": {
      "ingestion_date": {
      "gte": "now-1d",
        "lt": "now"
      }
    }
  },
  "aggs": {
    "daily_received": {
    "terms": {
      "field": "gex_file_status_message",
      "size": 10000
      }
    }
  }
}
```

NOTE: Substitute ‘gex_file_status_message’ with ‘gex_file_status’ will list GEX ingestion by file status over the 24 hours.

b) To list GEX ingestion by file status message for failed files only:

```
GET
{
  "size": 0,
  "query": {
    "bool": {
      "must": [
        {
          "match" : {"gex_file_status" : "failed"}
        }
      ]
    }
  },
  "aggs": {
    "daily_received": {
    "terms": {
      "field": "gex_file_status_message",
      "size": 100
      }
    }
  }
}
```

NOTE: The above “match” query clause is included in square brackets in case other index patterns needed to be added to the same search.

15 GEX Ingestion History

a) To list GEX ingestion by file error over the last 24 hours:

```
GET
{
 "size": 0,
 "query": {
   "range": {
     "received_date": {
     "gte": "now-1d",
       "lt": "now"
     }
   }
 },
 "aggs": {
   "daily_received": {
   "terms": {
       "field": "file_error",
       "size": 100
     }
   }
 }
}
```

b) To list GEX ingestion by file error per day between a given date range:

```
GET
{
 "query": {
   "bool": {
     "filter": {
       "range": {
         "received_date": {
         "gte": "01/01/2018",
           "lt": "2020",
           "format": "dd/MM/yyyy||yyyy"
         }
       }
     }
   }
 },
 "aggs": {
   "daily_load_by_file_error": {
   "date_histogram": {
       "field": "received_date",
       "interval": "day"
      },
      "aggs": {
       "file_error": {
       "terms": {
         "field": "file_error"
        }
       }
      }
     }
    }
   }
```

NOTE: Dates can be parsed using the format parameter specified on the date field.

c) To display all error_stacktrace that are not null:

```
GET
{
 "query": {
   "bool": {
      "must_not": {
        "match" : {
          "error_stacktrace" : "ZZZULL"
        }
      },
      "must": {
        "exists" : {
          "field": "error_stacktrace"
        }
      }
   }
 }
}
```

d) To get a count of how many documents are ingested between a given date range:

```
GET
{
 "query": {
  "range": {
    "ingestion_date": {
    "gte": "2019/04/01",
    "lt": "2019/06/30"
    }
   }
 },
 "aggs": {
  "daily_load_by_day": {
  "date_histogram": {
     "field": "ingestion_date",
     "interval": "day"
    }
   }
 }
}
```

16 Contract Awards

a) To list contract awards and contract attachments over the last month:

```
GET
{
 "size": 0,
 "query": {
  "bool": {
    "should": [
    {
     "range": {
       "ingestion_date": {
       "gte": "now-1M/M",
         "lt": "now/M"
       }
     }
    },
    {
     "bool" : {
       "must" : [
         { "wildcard" : {"file_name" : "*.pdf*"}},
         { "term" : {"file_ingest_type" : "award pdf"}}
       ]
     }
    },
        {
          "bool" : {
            "must_not": {
              "wildcard": {"file_name" : "*.csv*"}
            },
            "must" : {
              "term" : {"file_ingest_type" : "contract attachment"}
            }
          }
        }
        ],
        "minimum_should_match": 2
      }
    },
    "aggs": {
      "reccount": {
        "terms": {"field": "file_ingest_type"}
      }
    }
}
```

17 Mass Contract Closeout (CCO) Queries

a) To check the ingestion count for a specified CCO file. The count will be the total processed records plus one for the actual spreadsheet:

```
GET
{
  "sort": {
    "received_date": {"order": "desc"}
  },
  "query": {
    "bool": {
      "must": [
          {"term": {"ingest_type": "MASS CLOSEOUT"}},
          {"term": {"file_name": "<Excel_file_name>_<sheet_tab_name>.csv"}}
      ]
    }
  }
}
```

NOTE: The file extension is .csv as the NiFi code automatically converts the CCO Excel spreadsheet to a CSV file.

b) To obtain the processed (or failed) CCO count for a specific spreadsheet:

```
GET
{
 "sort": {
   "received_date": {"order": "desc"}
 },
 "query": {
   "bool": {
     "must": [
         {"term": {"ingest_type": "MASS CLOSEOUT"}},
         {"term": {"file_name": "<Excel_file_name>_<sheet_tab_name>.csv"}},
         {"term": {"file_error": "N"}}
     ]
   }
 }
}
```

c) To confirm the closeout status for a specific contract/DO:

```
GET
{
 "query": {
   "bool": {
         "must": [
             {"term": {"contract_number": "SPM7LX11D9037"}},
             {"term": {"delivery_order_number": "182W"}}
           ]
   }
 }
}
```

NOTE: To identify the parent-child relationship, one can match the contract.parent_record_key (child) with the contract.record_key (parent).

d) To determine the count for unique error details for a specific CCO spreadsheet:

```
GET
{
 "size": 0,
 "sort": {
   "received_date": {"order": "desc"}
 },
 "query": {
   "bool": {
     "must": [
         {"match": {"ingest_type": "MASS CLOSEOUT"}},
         {"match": {"file_name": "<Excel_file_name>_<sheet_tab_name>.csv"}},
         {"match": {"file_error": "Y"}}
     ]
   }
 },
 "aggregations": {
   "group_by_error": {
   "terms": {
     "field": "file_error_detail",
     "order":
       {"_key": "asc"}
     }
   }
 }
}
```

NOTE: The "sum_other_doc_count" value plus the remaining doc counts should equal to the number of mass CCO errors. By default, a Terms aggregation gives the top ten most popular terms and their counts. A sum_other_doc_count field represents the "Other" items.

e) To confirm the closed_date in the contracts index:

```
GET
{
  "size": 100,
  "sort": {
    "closed_date": {"order": "desc"}
  },
  "query": {
    "term": {
      "contract_number_delivery_order_number": "HC10280400000"
    }
  }
}
```

f) To confirm the doc_history_contract index for the closeout contract with a date range filter:

```
GET
{
  "size": 100,
  "sort": {
    "ingestion_date": {"order": "desc"}
  },
  "query": {
   "bool": {
     "must": [
       {"bool": {
        "should": [
          {"term": {"contract_number": "ZZZULL"}},
          {"term": {"delivery_order_number": "ZZZULL"}},
          {"match": {"comments": "Closed Date Updated by <Excel_filename>.xlsx"}}
        ],
        "minimum_should_match": 1
        }
       },
       {"range": {
         "ingestion_date": {
            "gt": "2019/05/01",
            "lte": "2019/05/30"
         }
        }
       }
   ]
   }
  }
}
```

NOTES:
Null values in the contract_number and delivery_order_number fields are mapped to "ZZZULL".

The comments field is case sensitive and the provided text has to match exactly.

To tie history back to the contract, use the contract.parent_record_key to match the contracts.record_key.

g) To confirm a record for each processed CCO file has been written to the contract_close_file index. The total records processed within the file will also be displayed:

```
GET
{
  "query": {
    "term": {
      "file_name": "<Excel_file_name>_<sheet_tab_name>.csv"
    }
  }
}
```

18 Date Range Aggregation

a) To get a count of how many documents are ingested between a specified date range:

```
GET
{
 "query": {
   "range": {
     "ingestion_date": {
     "gte": "2019/04/01",
     "lt": "2019/06/30"
     }
   }
 },
 "aggs": {
   "daily_load_by_day": {
   "date_histogram": {
      "field": "ingestion_date",
      "interval": "day"
     }
   }
 }
}
```

19 PDS IPR Monthly Reports

a) Below is an example of PDS IPR Monthly Reports for a service/agency loaded in previous month:

```
GET
{
 "_source": [ "schema_version", "pds_status" ],
 "query": {
   "bool": {
      "must": [
        {"bool": {
         "should": [
           {"prefix": {"issue_by_dodaac": "E"}},
           {"prefix": {"issue_by_dodaac": "F"}},
           {"prefix": {"issue_by_dodaac": "J"}}
         ],
         "minimum_should_match": 1
         }
        },
        {"range": {
          "signature_date": {
          "gte": "now-1M/M",
            "lt": "now/M"
          }
        }}
      ],
      "must_not": [
        {"bool": {
         "should": [
           {"match": { "aco_mod": "ZZZULL"}},
           {"match": { "pco_mod": "ZZZULL"}}
         ],
         "minimum_should_match": 2
        }}
      ]
    }
  },
  "aggregations": {
    "group_by_pds_status": {
    "terms": {
       "field": "pds_status",
       "order":
         {"_key": "asc"}
      }
    }
  }
}
```

20 Elasticsearch SQL

Elasticsearch offers an SQL feature included in X-Pack, an Elastic Stack extension, to execute SQL queries against Elasticsearch indices and return results in tabular format:

NOTE: The "SELECT" statement must be one continuous line without any line breaks. Also, join or complex predicates are not supported.

a) Below are some Elasticsearch SQL examples:

```
POST /_xpack/sql?format=txt
{
 "query":
   " SELECT contract_number ,pds_schema_version, load_date FROM contracts
WHERE contract_number LIKE 'SPM74%' AND load_date > '2016/03/01'  "
}
```

```
POST /_xpack/sql?format=txt
{
  "query":
  " SELECT contract_number, delivery_order_number, aco_mod, pco_mod FROM
conformance_pds ORDER BY status_change_date DESC LIMIT 10 "
}
```

```
POST /_xpack/sql?format=txt
{
  "query":
    "SELECT contract_number, delivery_order_number, count(*)
contract_count FROM conformance_pds WHERE contract_number LIKE 'fa2%'
GROUP BY contract_number, delivery_order_number HAVING count(*) > 1
}
```

b) To convert an SQL query into a regular Elasticsearch query:

```
POST /_xpack/sql/translate
{
  "query":
    "select schema_version, creation_date from conformance_pds where pds_status
= 'Waiting' and (aco_mod is not null or pco_mod is not null) and (creation_date >
'2019/05/31' and creation_date < '2019/07/01') and delivery_order_number='ZZZULL'
"
}
```