Contents hide

Related Articles

- How to Enable 'eq' Filter GraphQL Search for an Custom Attribute of Type Text in Magento 2
- Magento 2 - Useful Elasticsearch Commands
- Magento 2 - Elastic Search Concepts for a Magento Developer
- How Product Data is Pushed to Elasticsearch from Magento
- Elastic Search for Magento Developer

Overview

Once we have the product data pushed to Elasticsearch we can now use ES API's to fetch data.

The data from ES helps to

1. Identify the products ids that are relevant for the specific search query.
2. Filter Navigation / Aggregation

Once we have the products ids, The relevant product data such as pricing, inventory, product attributes are fetched from the mysql directly.

A search request to elastic search contains 3 parts.

1. The Search String
2. Sorting
3. Aggregation (Filter Navigation)

These requests are built based on configuration in search_request.xml file. By default, Magento uses the following query types:

- quick_search_container – Quick Search
- advanced_search_container – Advanced Search
- catalog_view_container – Category page with layered navigation
- graphql_product_search_with_aggregation – Used by GraphQL API when you pass aggregation query
- graphql_product_search – Used by GraphQL API

Code Reference :

- vendor/magento/module-catalog-graph-ql/etc/search_request.xml

The Search Query

There are three types are search query defined in search_request.xml

- boolQuery
- matchQuery
- filteredQuery

With the bool query, you can combine multiple queries into one request and further specify boolean clauses to narrow down your search results.
There are four clauses to choose from:

- Must
  - One or more queries can be specified here. A document MUST match all of these queries to be considered as a hit.
- must_not
  - A document must NOT match any of the queries specified here. It it does, it is excluded from the search results.
- Should
  - A document does not have to match any queries specified here. However, it if it does match, this document is given a higher score.
- Filter
  - These filters(queries) place documents in either yes or no category. Ones that fall into the yes category are included in the hits.
    You can build combinations of one or more of these clauses. Each clause can contain one or multiple queries that specify the criteria of each clause.
    These clauses are optional and can be mixed and matched to cater to your use case. The order in which they appear does not matter either!

Syntax:

```
GET name_of_index/_search
{
  "query": {
    "bool": {
      "must": [
        Filter by attributes, Category Ids, Visibility
      ],
      "should": [
        { The Search String }
      ],
    }
  }
}
```

How Search Query is Generated

The search queries are defined in the below 4 files, Based on the type of search request the respective container is selected.

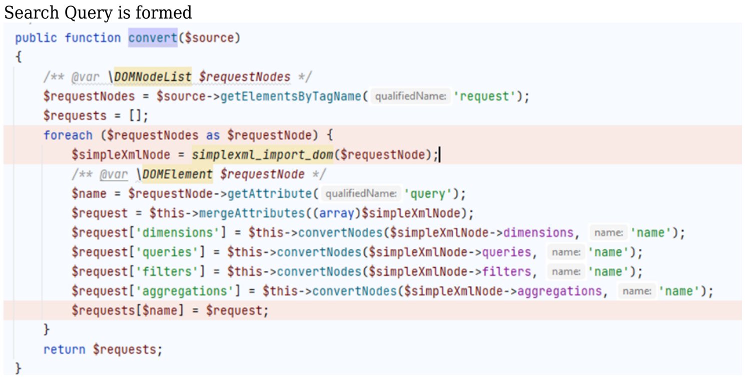
- vendor/magento/module-elasticsearch-catalog-permissions/etc/search_request.xml
- vendor/magento/module-elasticsearch-catalog-permissions-graph-ql/etc/search_request.xml
- vendor/magento/module-catalog-graph-ql/etc/search_request.xml
- vendor/magento/module-catalog-search/etc/search_request.xml

When search request is processed these xml files are processed and the relevant search query is generated and stored in the cache. The process of generating the search aggregation query is explained below

File Reference	Comments
\Magento\Framework\Config\Data::initData	XML file content and parsed and merged

\Magento\Framework\Search\Request\Config\Converter::convert

Search Query is formed



```php
public function convert($source)
{
    /** @var \DOMNodeList $requestNodes */
    $requestNodes = $source->getElementsByTagName( qualifiedName: 'request');
    $requests = [];
    foreach ($requestNodes as $requestNode) {
        $simpleXmlNode = simplexml_import_dom($requestNode);
        /** @var \DOMElement $requestNode */
        $name = $requestNode->getAttribute( qualifiedName: 'query');
        $request = $this->mergeAttributes((array)$simpleXmlNode);
        $request['dimensions'] = $this->convertNodes($simpleXmlNode->dimensions,  name: 'name');
        $request['queries'] = $this->convertNodes($simpleXmlNode->queries,  name: 'name');
        $request['filters'] = $this->convertNodes($simpleXmlNode->filters,  name: 'name');
        $request['aggregations'] = $this->convertNodes($simpleXmlNode->aggregations,  name: 'name');
        $requests[$name] = $request;
    }
    return $requests;
}
```

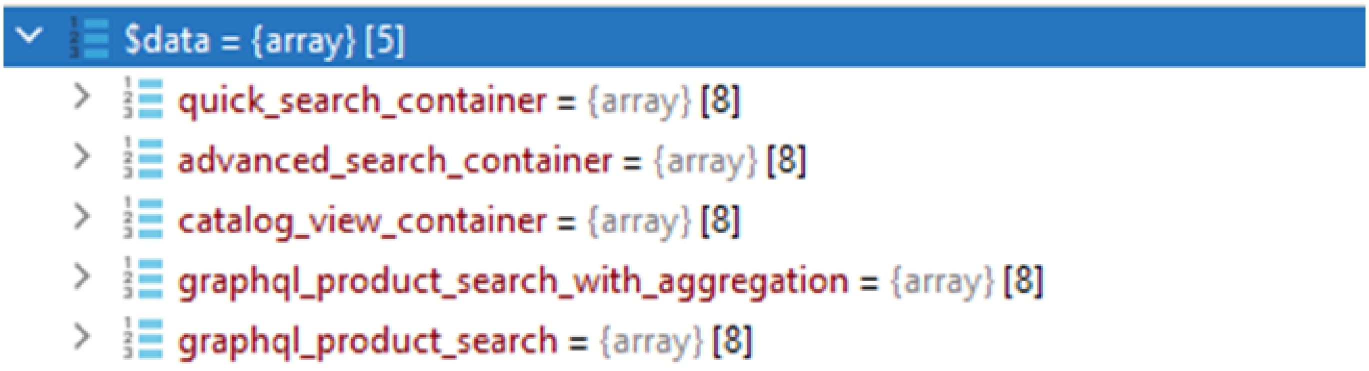
\Magento\CatalogSearch\Model\Search\Request\SearchModifier::modify

At this point we will have the category and price bucket generated but filterable attribute bucket are not generated yet

\Magento\CatalogSearch\Model\Search\RequestGenerator::generate
\Magento\CatalogSearch\Model\Search\RequestGenerator::generateRequest

At this point all the filterable bucket attributes are generated.
$request['aggregations'][$bucketName] = $generator->getAggregationData($attribute, $bucketName);

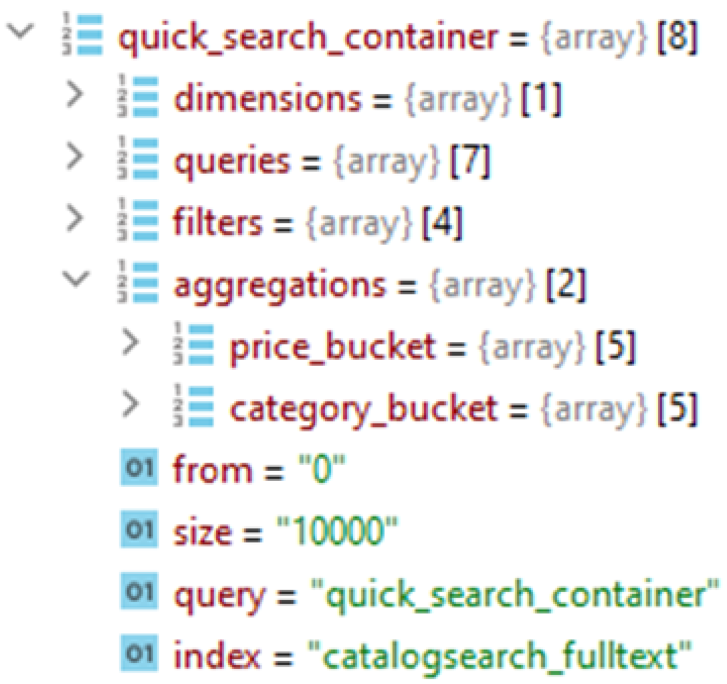
The ouput of these parsed xml files is stored in the $data array



```
$data = {array} [5]
    quick_search_container = {array} [8]
    advanced_search_container = {array} [8]
    catalog_view_container = {array} [8]
    graphql_product_search_with_aggregation = {array} [8]
    graphql_product_search = {array} [8]
```

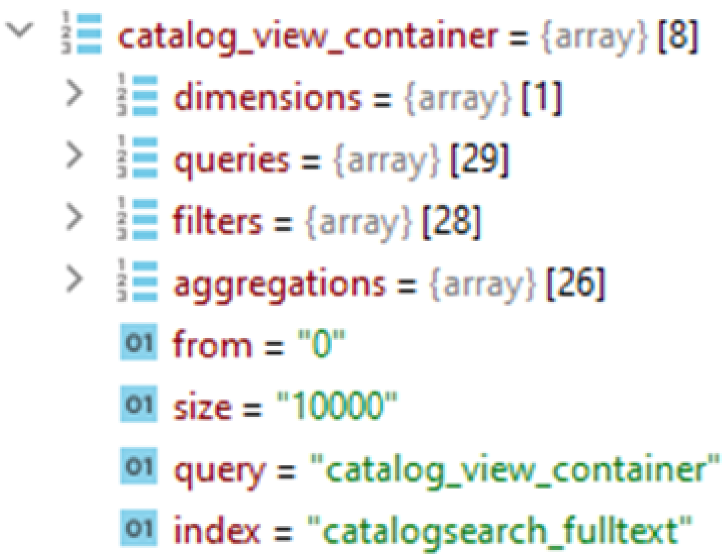
Container	Sample Data

Quick
Search
Container



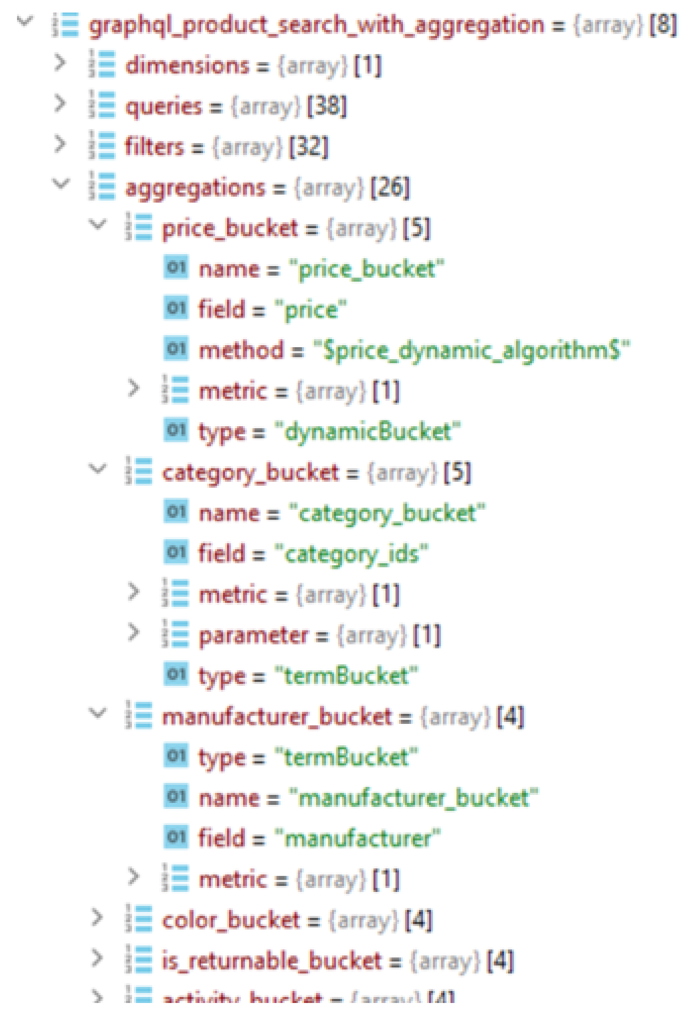
```
quick_search_container = {array} [8]
  dimensions = {array} [1]
  queries = {array} [7]
  filters = {array} [4]
  aggregations = {array} [2]
    price_bucket = {array} [5]
    category_bucket = {array} [5]
  from = "0"
  size = "10000"
  query = "quick_search_container"
  index = "catalogsearch_fulltext"
```

Catalog
View
Container



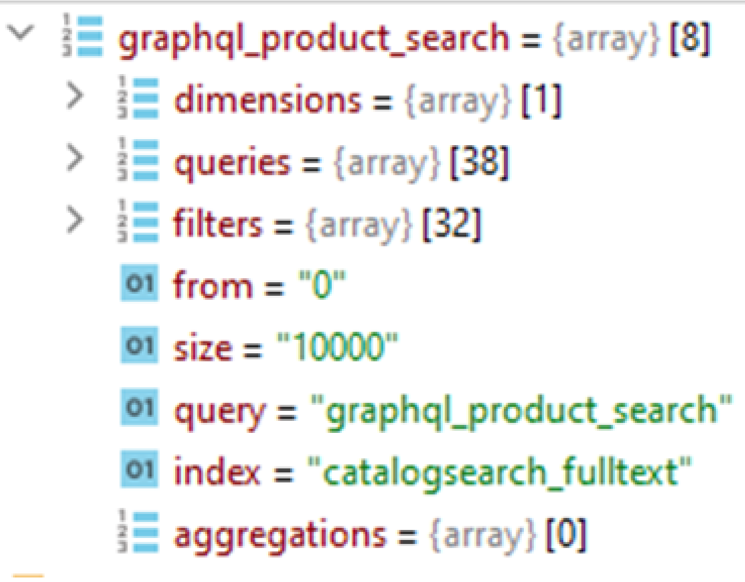
```
catalog_view_container = {array} [8]
  dimensions = {array} [1]
  queries = {array} [29]
  filters = {array} [28]
  aggregations = {array} [26]
  from = "0"
  size = "10000"
  query = "catalog_view_container"
  index = "catalogsearch_fulltext"
```

GraphQl Product Search With Aggregation



```
graphql_product_search_with_aggregation = {array} [8]
  dimensions = {array} [1]
  queries = {array} [38]
  filters = {array} [32]
  aggregations = {array} [26]
    price_bucket = {array} [5]
      name = "price_bucket"
      field = "price"
      method = "$price_dynamic_algorithm$"
      metric = {array} [1]
      type = "dynamicBucket"
    category_bucket = {array} [5]
      name = "category_bucket"
      field = "category_ids"
      metric = {array} [1]
      parameter = {array} [1]
      type = "termBucket"
    manufacturer_bucket = {array} [4]
      type = "termBucket"
      name = "manufacturer_bucket"
      field = "manufacturer"
      metric = {array} [1]
    color_bucket = {array} [4]
    is_returnable_bucket = {array} [4]
    activity_bucket = {array} [4]
```

GraphQl Product Search

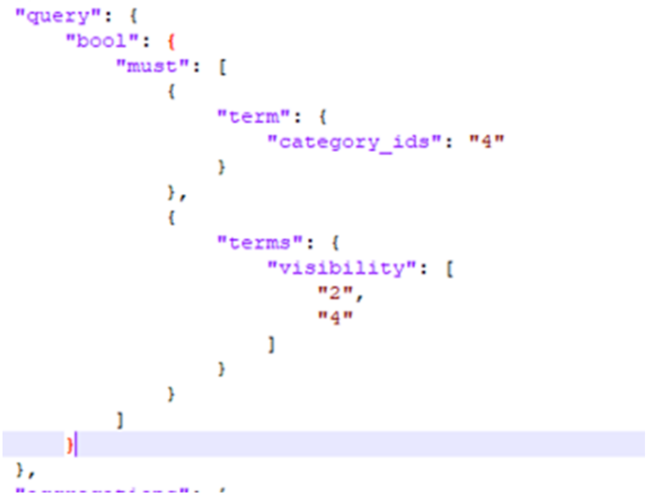


```
graphql_product_search = {array} [8]
  dimensions = {array} [1]
  queries = {array} [38]
  filters = {array} [32]
  from = "0"
  size = "10000"
  query = "graphql_product_search"
  index = "catalogsearch_fulltext"
  aggregations = {array} [0]
```

Lets see some actual elastic search requests that gets triggered in real time.

Category Page Query

To List all the Products from a specific Category

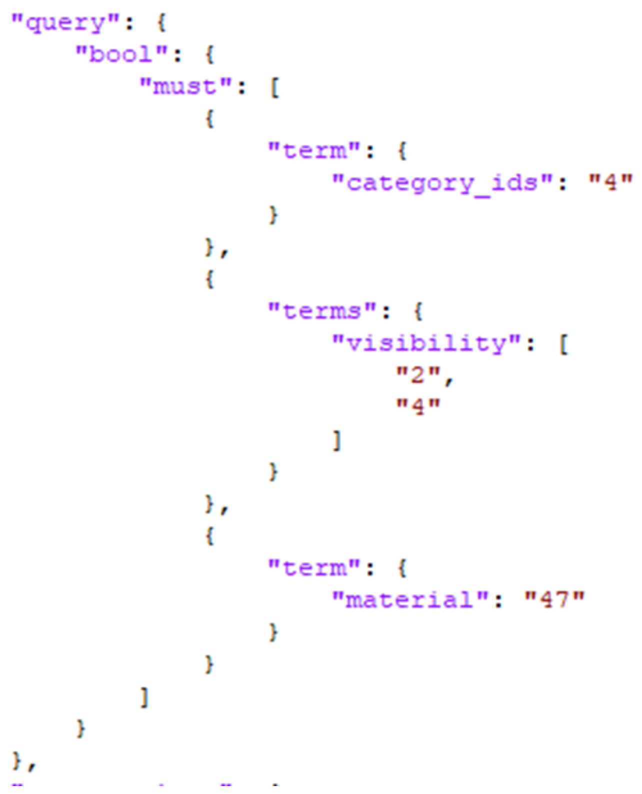
Frontend URL	**Elastic Search Request**
Category Page http://commerce.development/gear/bags.html	URL: http://localhost:9200/commerce_product_1/document/_search?track_total_hits=true Body: 

```
"query": {
    "bool": {
        "must": [
            {
                "term": {
                    "category_ids": "4"
                }
            },
            {
                "terms": {
                    "visibility": [
                        "2",
                        "4"
                    ]
                }
            }
        ]
    }
},
```

Category Page with Filter Selected
http://commerce.development/gear/bags.html?material=47

Api Request:
http://localhost:9200/commerce_product_1/document/_search?track_total_hits=true

Request Body:

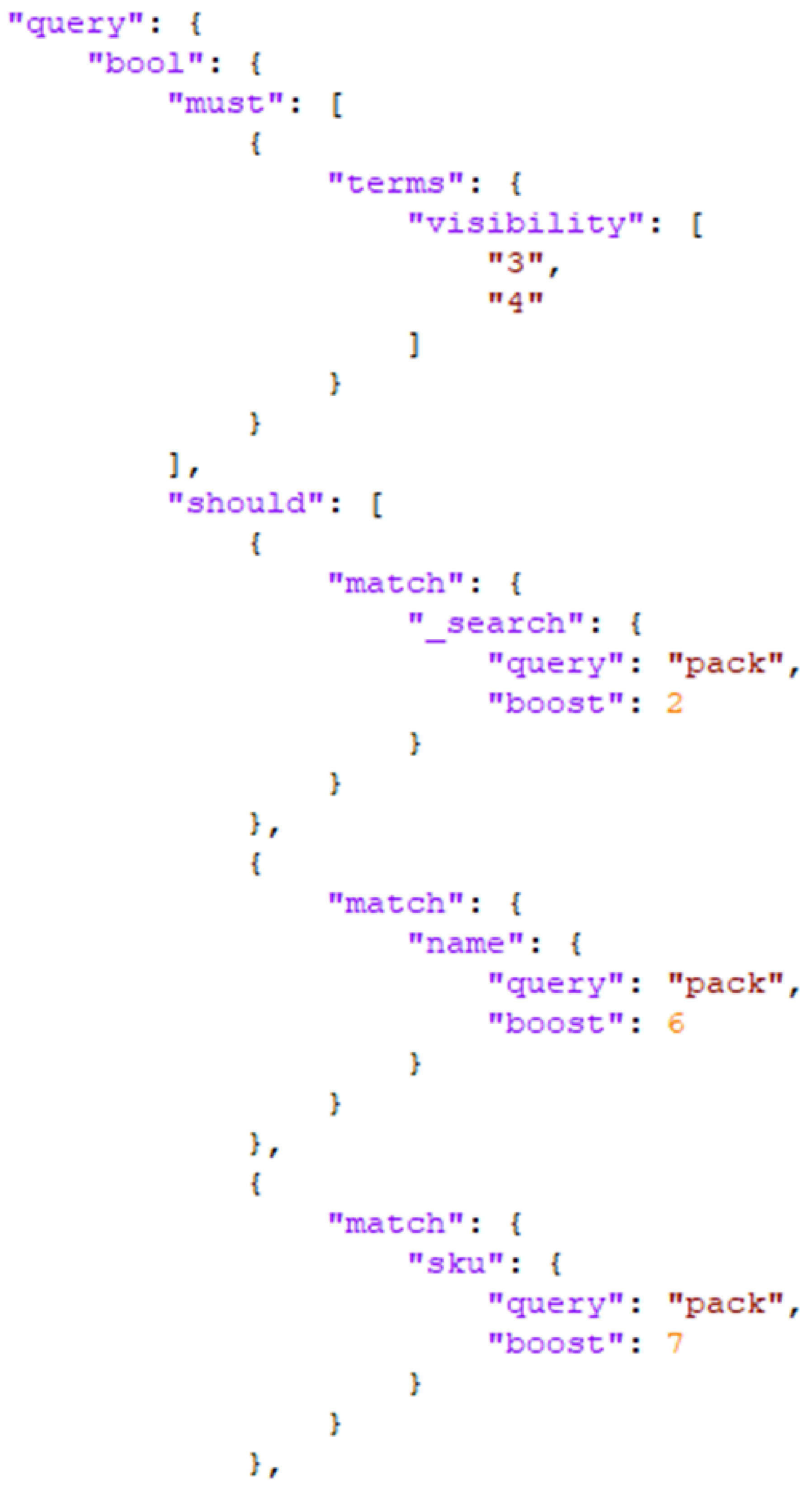


```
"query": {
    "bool": {
        "must": [
            {
                "term": {
                    "category_ids": "4"
                }
            },
            {
                "terms": {
                    "visibility": [
                        "2",
                        "4"
                    ]
                }
            },
            {
                "term": {
                    "material": "47"
                }
            }
        ]
    }
},
```

Search Page Query

If you are searching for a string “pack” the below query is issues to elastic search

Frontend URL : http://commerce.development/catalogsearch/result/?q=pack



```
"query": {
    "bool": {
        "must": [
            {
                "terms": {
                    "visibility": [
                        "3",
                        "4"
                    ]
                }
            }
        ],
        "should": [
            {
                "match": {
                    "_search": {
                        "query": "pack",
                        "boost": 2
                    }
                }
            },
            {
                "match": {
                    "name": {
                        "query": "pack",
                        "boost": 6
                    }
                }
            },
            {
                "match": {
                    "sku": {
                        "query": "pack",
                        "boost": 7
                    }
                }
            },
```

In Short difference between PLP and Search Page query is the query clause

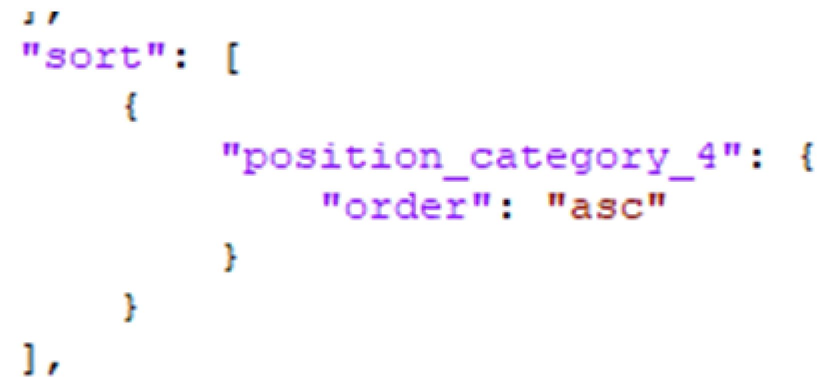
- PLP -> must
- Search Page -> must & should

Sorting

By default magneto supports position, product name and price sorting. Additional attributes can be added for sorting. In that case the respective mapping will get updated note that a full reindexing is required in that case.

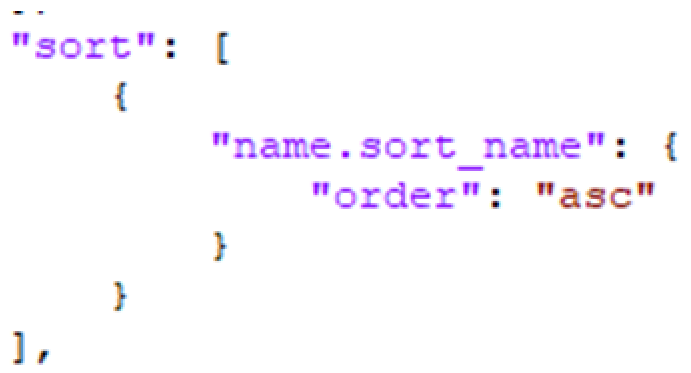
Below are the few sort query examples.

Sort By Position



```
"sort": [
    {
        "position_category_4": {
            "order": "asc"
        }
    }
],
```

Sort By Name

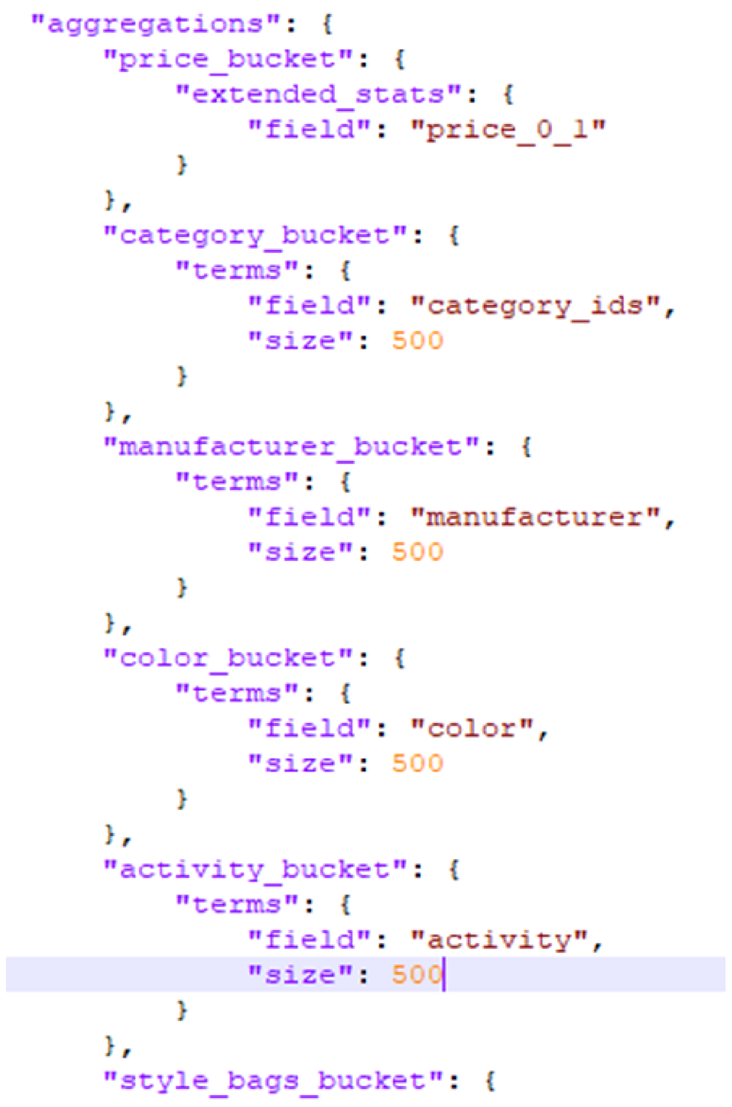


```
"sort": [
    {
        "name.sort_name": {
            "order": "asc"
        }
    }
],
```

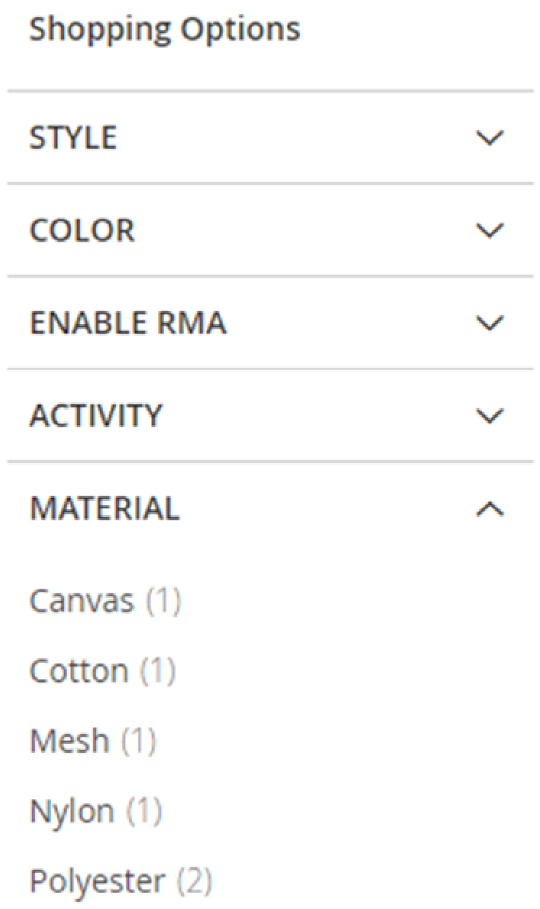
Filter Navigation / Aggregation

The data for the filter navigation is generated using the aggregation buckets in the search query

Search Query	**Frontend Display**



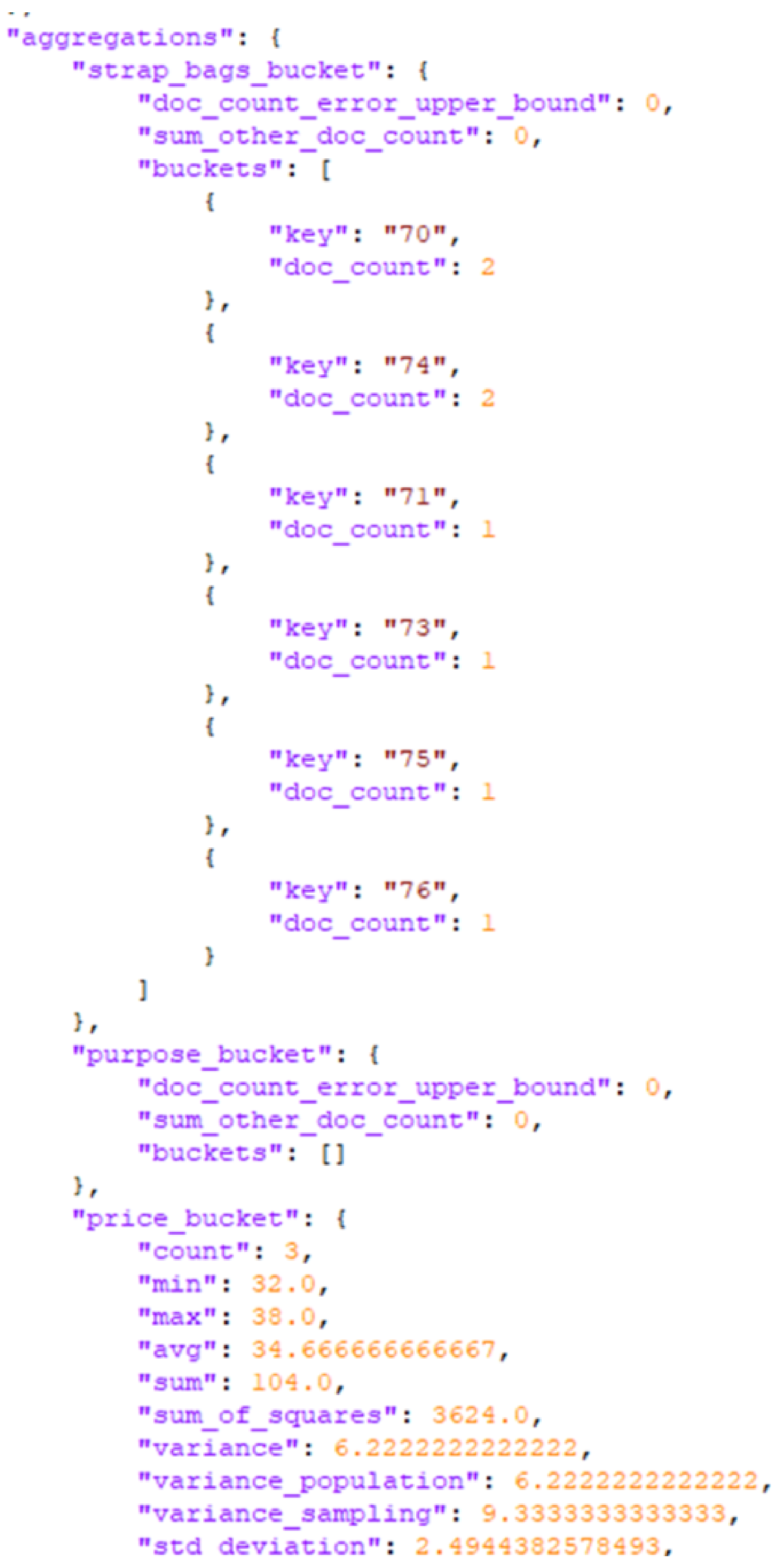
```
"aggregations": {
    "price_bucket": {
        "extended_stats": {
            "field": "price_0_1"
        }
    },
    "category_bucket": {
        "terms": {
            "field": "category_ids",
            "size": 500
        }
    },
    "manufacturer_bucket": {
        "terms": {
            "field": "manufacturer",
            "size": 500
        }
    },
    "color_bucket": {
        "terms": {
            "field": "color",
            "size": 500
        }
    },
    "activity_bucket": {
        "terms": {
            "field": "activity",
            "size": 500
        }
    },
    "style_bags_bucket": {
```



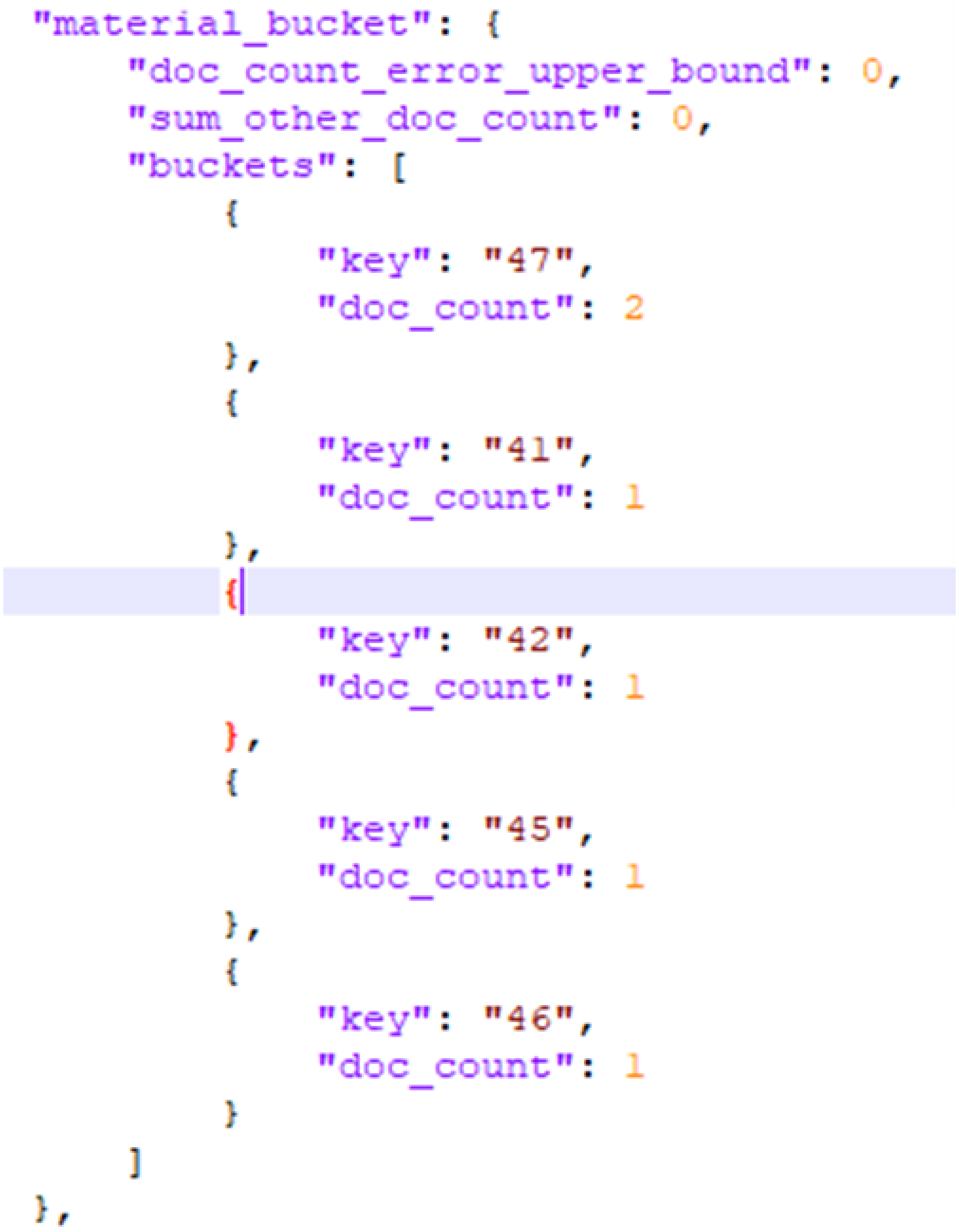
There are primarily three types of aggregation buckets

- Category Bucket
- Price Bucket
- Filterable Product Attributes Bucket

The Response from the aggregation bucket from the elastic search will be like the below



```
"aggregations": {
    "strap_bags_bucket": {
        "doc_count_error_upper_bound": 0,
        "sum_other_doc_count": 0,
        "buckets": [
            {
                "key": "70",
                "doc_count": 2
            },
            {
                "key": "74",
                "doc_count": 2
            },
            {
                "key": "71",
                "doc_count": 1
            },
            {
                "key": "73",
                "doc_count": 1
            },
            {
                "key": "75",
                "doc_count": 1
            },
            {
                "key": "76",
                "doc_count": 1
            }
        ]
    },
    "purpose_bucket": {
        "doc_count_error_upper_bound": 0,
        "sum_other_doc_count": 0,
        "buckets": []
    },
    "price_bucket": {
        "count": 3,
        "min": 32.0,
        "max": 38.0,
        "avg": 34.666666666667,
        "sum": 104.0,
        "sum_of_squares": 3624.0,
        "variance": 6.2222222222222,
        "variance_population": 6.2222222222222,
        "variance_sampling": 9.3333333333333,
        "std deviation": 2.4944382578493,
```



```
"material_bucket": {
    "doc_count_error_upper_bound": 0,
    "sum_other_doc_count": 0,
    "buckets": [
        {
            "key": "47",
            "doc_count": 2
        },
        {
            "key": "41",
            "doc_count": 1
        },
        {
            "key": "42",
            "doc_count": 1
        },
        {
            "key": "45",
            "doc_count": 1
        },
        {
            "key": "46",
            "doc_count": 1
        }
    ]
},
```

These data from the response is parsed and displayed in the Frontend Filtered Navigation