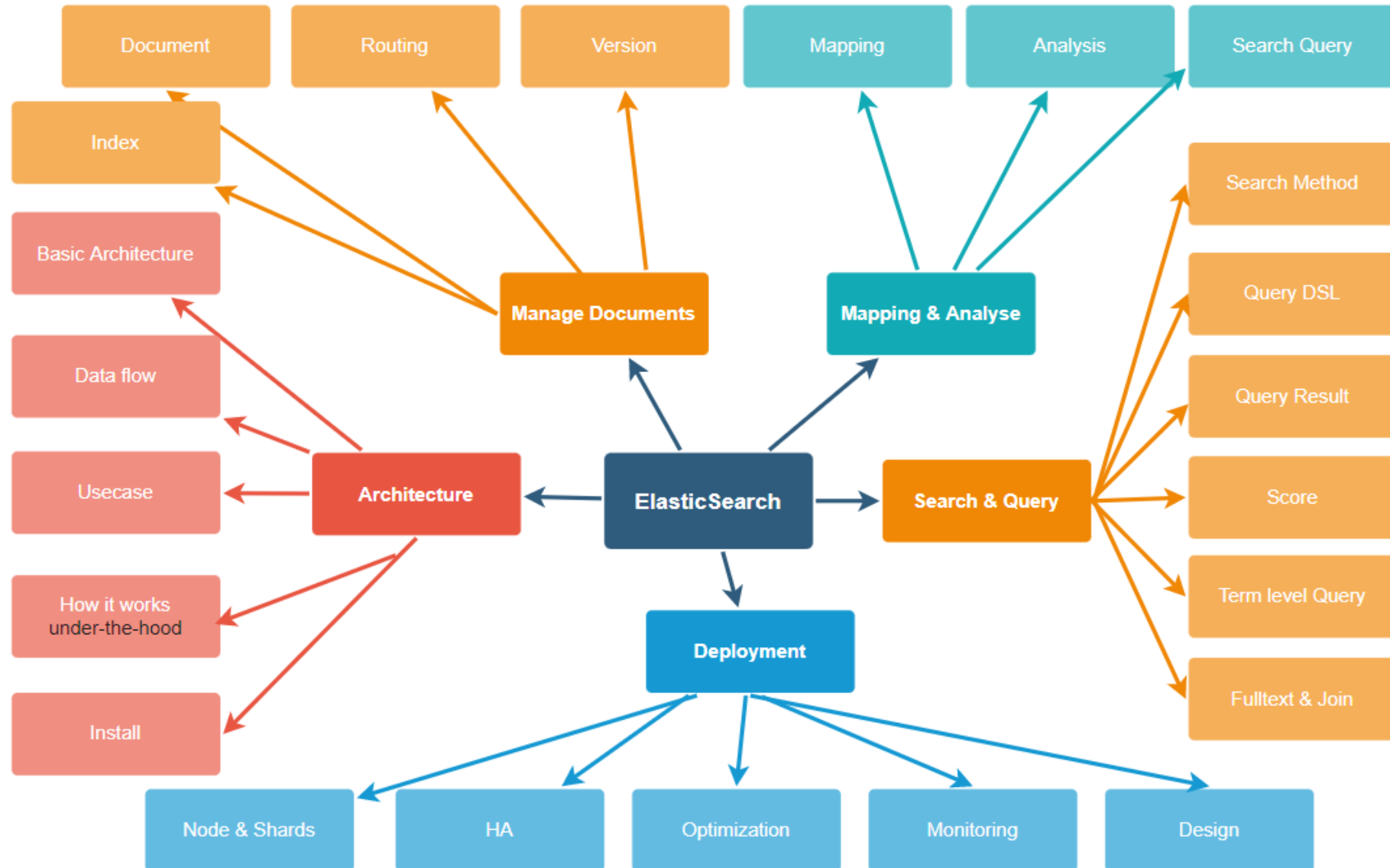





ElasticSearch

Techmaster

Nội dung




Import data sample

 Home

Add Data to Kibana


Use these solutions to quickly turn your data into pre-built dashboards and monitoring systems.



APM

APM automatically collects in-depth performance metrics and errors from inside your applications.


Add APM



Logging

Ingest logs from popular data sources and easily visualize in preconfigured dashboards.


Add log data



Metrics

Collect metrics from the operating system and services running on your servers.

Add metric data



Security analytics

Centralize security events for interactive investigation in ready-to-go visualizations.

Add security events

Add sample data


[Load a data set and a Kibana dashboard](#)

Upload data from log file

[Import a CSV, NDJSON, or log file](#)


Use Elasticsearch data

[Connect to your Elasticsearch index](#)

 Home **Add data**

Add Data to Kibana


All Logging Metrics Security analytics **Sample data**



Sample eCommerce orders

Sample data, visualizations, and dashboards for tracking eCommerce orders.


Add data



Sample flight data

Sample data, visualizations, and dashboards for monitoring flight routes.

Add data




Sample web logs

Sample data, visualizations, and dashboards for monitoring web logs.


Add data

Import data sample

 Home

Add Data to Kibana


Use these solutions to quickly turn your data into pre-built dashboards and monitoring systems.



APM

APM automatically collects in-depth performance metrics and errors from inside your applications.


Add APM



Logging

Ingest logs from popular data sources and easily visualize in preconfigured dashboards.


Add log data



Metrics

Collect metrics from the operating system and services running on your servers.

Add metric data



Security analytics

Centralize security events for interactive investigation in ready-to-go visualizations.

Add security events

Add sample data


[Load a data set and a Kibana dashboard](#)

Upload data from log file

Import a CSV, NDJSON, or log file


Use Elasticsearch data

Connect to your Elasticsearch index

 Home **Add data**

Add Data to Kibana


All Logging Metrics Security analytics **Sample data**



Sample eCommerce orders

Sample data, visualizations, and dashboards for tracking eCommerce orders.


Add data



Sample flight data

Sample data, visualizations, and dashboards for monitoring flight routes.

Add data



Sample web logs

Sample data, visualizations, and dashboards for monitoring web logs.

Add data

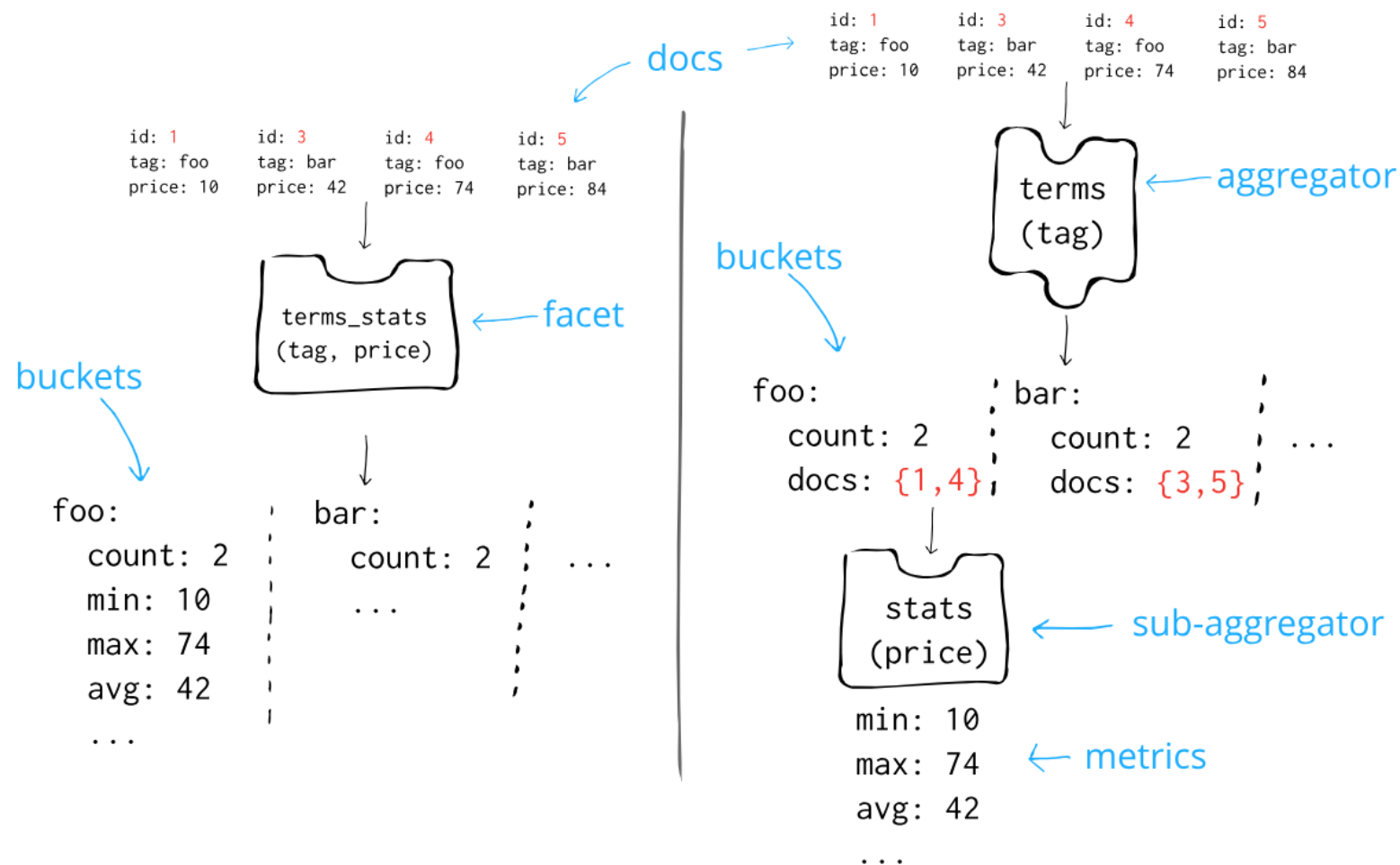
Aggregation



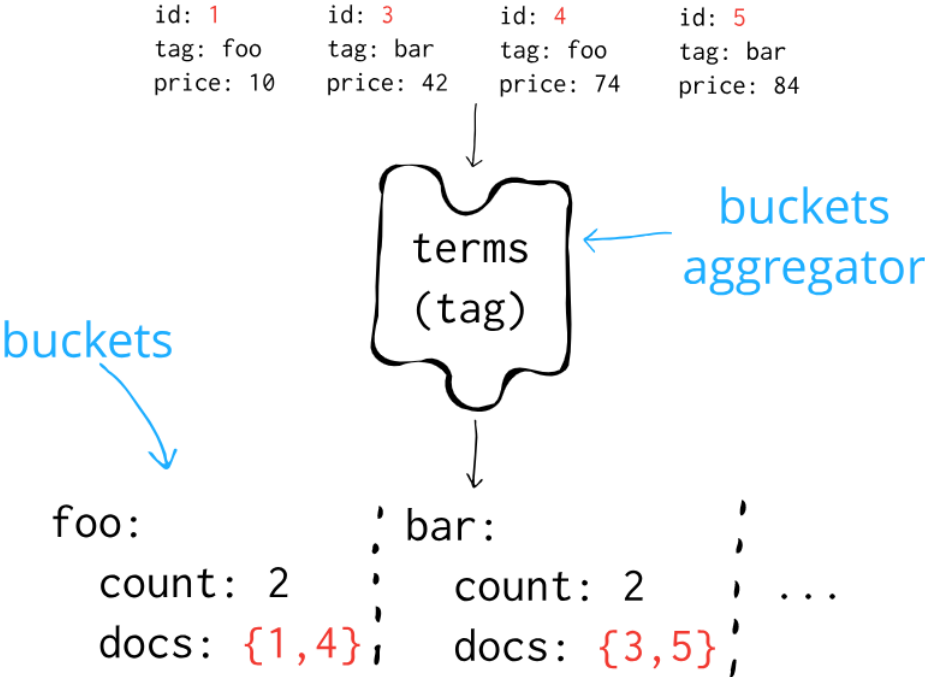
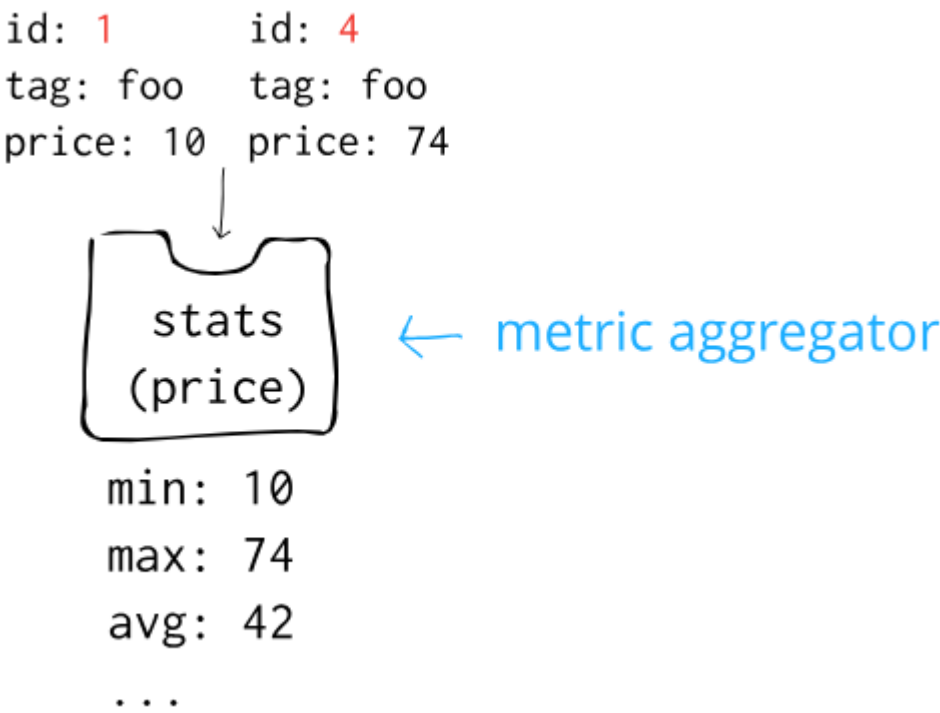
An aggregation summarizes your data as metrics, statistics, or other analytics. Aggregations help you answer questions like:

- What's the average load time for my website?
- Who are my most valuable customers based on transaction volume?
- What would be considered a large file on my network?
- How many products are in each product category?

Aggregation vs Facet



Metric vs bucket



Aggregation Syntax



- ❖ aggs
- ❖ name_of_aggregation
- ❖ type_of_aggregation
- ❖ Field
- ❖ document_field_name

```
"aggs": {  
  "name_of_aggregation": {  
    "type_of_aggregation": {  
      "field":  
        "document_field_name"  
    }  
  }  
}
```


Type of Aggregation



Type

- ❖ Metric aggregations
- ❖ Bucket aggregations
- ❖ Pipeline aggregations
- ❖ Matrix aggregations

Aggregation

- ❖ Cardinality aggregation
- ❖ Stats aggregation
- ❖ Filter aggregation
- ❖ Terms aggregation
- ❖ Nested aggregation

Elasticsearch organizes aggregations into three categories:

- **Metric** aggregations that calculate metrics, such as a sum or average, from field values.
- **Bucket** aggregations that group documents into buckets, also called bins, based on field values, ranges, or other criteria.
- **Pipeline** aggregations that take input from other aggregations instead of documents or fields.

Cardinality aggregation

```
GET /kibana_sample_data_ecommerce/_search
```

```
{
  "size": 0,
  "aggs": {
    "unique_skus": {
      "cardinality": {
        "field": "sku"
      }
    }
  }
}
```



```
1  {
2    "took" : 372,
3    "timed_out" : false,
4    "_shards" : {
5      "total" : 1,
6      "successful" : 1,
7      "skipped" : 0,
8      "failed" : 0
9    },
10   "hits" : {
11     "total" : {
12       "value" : 4675,
13       "relation" : "eq"
14     },
15     "max_score" : null,
16     "hits" : [ ]
17   },
18   "aggregations" : {
19     "unique_skus" : {
20       "value" : 7186
21     }
22   }
23 }
```

Stats Aggregation

```
GET /kibana_sample_data_ecommerce/_search
```

```
{
  "size": 0,
  "aggs": {
    "quantity_stats": {
      "stats": {
        "field": "total_quantity"
      }
    }
  }
}
```

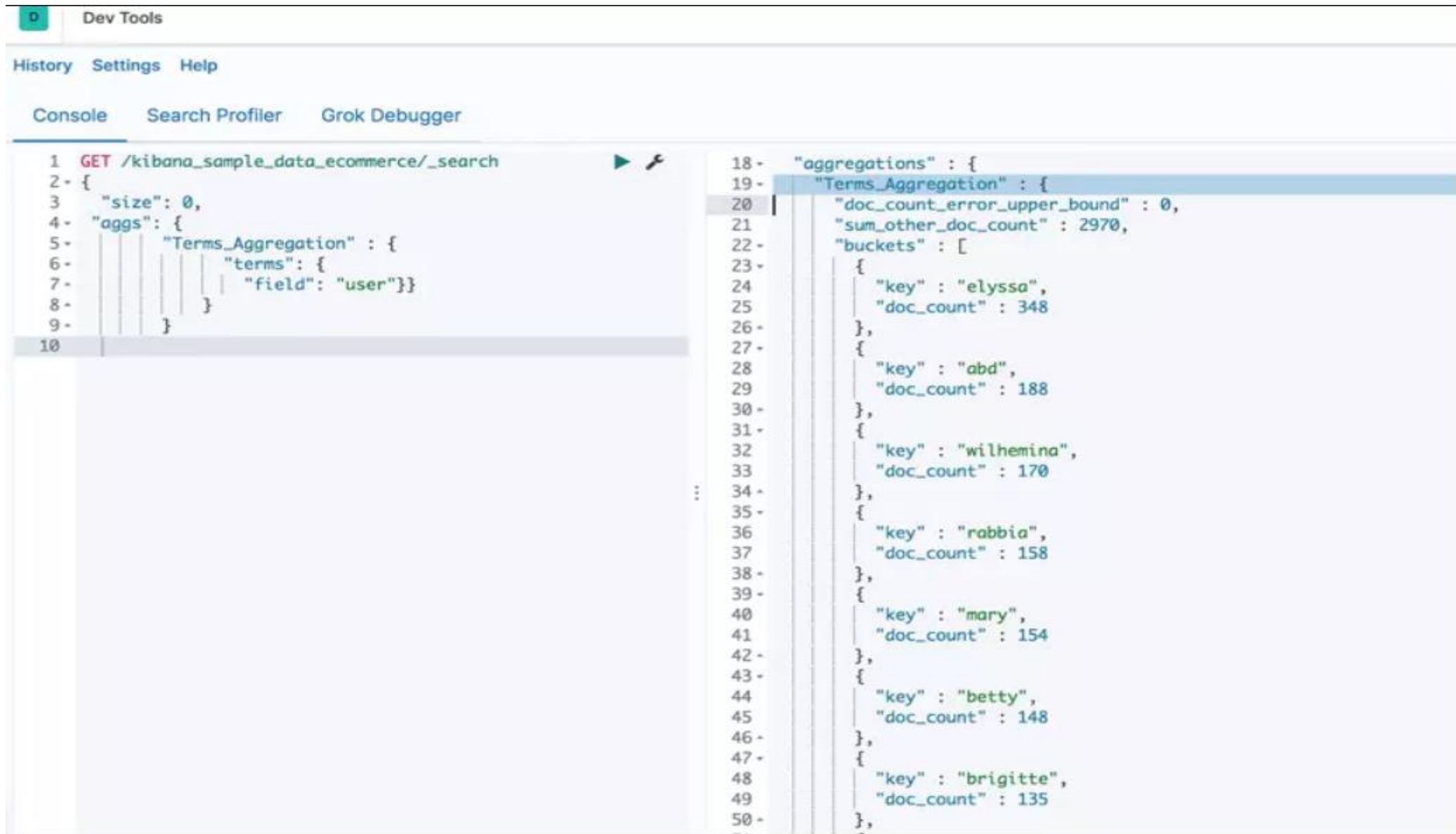
```
2  "took" : 33,
3  "timed_out" : false,
4  "_shards" : {
5    "total" : 1,
6    "successful" : 1,
7    "skipped" : 0,
8    "failed" : 0
9  },
10 "hits" : {
11   "total" : {
12     "value" : 4675,
13     "relation" : "eq"
14   },
15   "max_score" : null,
16   "hits" : [ ]
17 },
18 "aggregations" : {
19   "quantity_stats" : {
20     "count" : 4675,
21     "min" : 1.0,
22     "max" : 8.0,
23     "avg" : 2.1585026737967916,
```

Filter Aggregation

```
GET /kibana_sample_data_ecommerce/_search
{ "size": 0,
  "aggs": {
    "User based filter" : {
      "filter" : {
        "term": {
          "user": "eddie"}}},
    "aggs" : {
      "avg_price" : {
        "avg" : {
          "field" : "products.price" } }
    }
  }
}
```

```
2   "took" : 43,
3   "timed_out" : false,
4   "_shards" : {
5     "total" : 1,
6     "successful" : 1,
7     "skipped" : 0,
8     "failed" : 0
9   },
10  "hits" : {
11    "total" : {
12      "value" : 4675,
13      "relation" : "eq"
14    },
15    "max_score" : null,
16    "hits" : [ ]
17  },
18  "aggregations" : {
19    "User based filter" : {
20      "doc_count" : 100,
21      "avg_price" : {
22        "value" : 34.85423743206522
23      }
24    }
25  }
```

Terms Aggregation



The screenshot displays the DevTools console with a REST client request and its corresponding JSON response. The request is a GET to `/kibana_sample_data_ecommerce/_search` with a body containing a Terms Aggregation on the `user` field. The response shows the aggregation results for the `user` field, listing the top terms and their document counts.

```
1 GET /kibana_sample_data_ecommerce/_search
2 {
3   "size": 0,
4   "aggs": {
5     "Terms_Aggregation": {
6       "terms": {
7         "field": "user"
8       }
9     }
10  }
11 }
12
13
14
15
16
17
18 "aggregations": {
19   "Terms_Aggregation": {
20     "doc_count_error_upper_bound": 0,
21     "sum_other_doc_count": 2970,
22     "buckets": [
23       {
24         "key": "elyssa",
25         "doc_count": 348
26       },
27       {
28         "key": "abd",
29         "doc_count": 188
30       },
31       {
32         "key": "wilhemina",
33         "doc_count": 170
34       },
35       {
36         "key": "rabbia",
37         "doc_count": 158
38       },
39       {
40         "key": "mary",
41         "doc_count": 154
42       },
43       {
44         "key": "betty",
45         "doc_count": 148
46       },
47       {
48         "key": "brigitte",
49         "doc_count": 135
50       },
51     ]
52   }
53 }
```

Nested Aggregation

```
PUT nested_aggregation
{
  "mappings": {
    "properties": {
      "Employee": {
        "type": "nested",
        "properties": {
          "first" : { "type" : "text" },
          "last" : { "type" : "text" },
          "salary" : { "type" : "double" }
        }
      }
    }
  }
}
```

```
PUT nested_aggregation/_doc/1
{
  "group" : "Logz",
  "Employee" : [
    {
      "first" : "Ana",
      "last" : "Roy",
      "salary" : "70000"
    },
    {
      "first" : "Jospeh",
      "last" : "Lein",
      "salary" : "64000"
    },
    {
      "first" : "Chris",
      "last" : "Gayle",
      "salary" : "82000"
    },
    {
      "first" : "Brendon",
      "last" : "Maculum",
      "salary" : "58000"
    },
    {
      "first" : "Vinod",
      "last" : "Kambli",
      "salary" : "63000"
    },
    {
      "first" : "DJ",
      "last" : "Bravo",
      "salary" : "71000"
    },
    {
      "first" : "Jaques",
      "last" : "Kallis",
      "salary" : "75000"
    }
  ]
}
```

Nested Aggregation

Dev Tools

History Settings Help

Console Search Profiler Grok Debugger

```
1 GET /nested_aggregation/_search
2 {
3   "aggs": {
4     "Nested_Aggregation" : {
5       "nested": {
6         "path": "Employee"
7       },
8       "aggs": {
9         "Min_Salary": {
10          "min": {
11            "field": "Employee.salary"
12          }
13        }
14      }
15    }
16  }
```

```
42   "last" : "Maculum",
43   "salary" : "58000"
44 },
45 {
46   "first" : "Vinod",
47   "last" : "Kambli",
48   "salary" : "63000"
49 },
50 {
51   "first" : "DJ",
52   "last" : "Bravo",
53   "salary" : "71000"
54 },
55 {
56   "first" : "Jaques",
57   "last" : "Kallis",
58   "salary" : "75000"
59 }
60 ]
61 }
62 ]
63 ],
64 },
65 "aggregations" : {
66   "Nested_Aggregation" : {
67     "doc_count" : 7,
68     "Min_Salary" : {
69       "value" : 58000.0
70     }
71   }
72 }
```

Joining queries



Performing full SQL-style joins in a distributed system like Elasticsearch is prohibitively expensive. Instead, Elasticsearch offers two forms of join which are designed to scale horizontally

- ❖ nested query
- ❖ has_child and has_parent queries

SQL

```
PUT /library/book/_bulk?refresh
{"index":{"_id": "Leviathan Wakes"}}
{"name": "Leviathan Wakes", "author": "James S.A. Corey",
 "release_date": "2011-06-02", "page_count": 561}
{"index":{"_id": "Hyperion"}}
{"name": "Hyperion", "author": "Dan Simmons", "release_date": "1989
-05-26", "page_count": 482}
{"index":{"_id": "Dune"}}
{"name": "Dune", "author": "Frank Herbert", "release_date": "1965-06
-01", "page_count": 604}
```

```
POST /_sql?format=txt
{
  "query": "SELECT * FROM library WHERE release_date < '2000-01-01'"
}
```



||

1	author	name	page_count	release_date
2	-----+-----+-----+-----			
3	Dan Simmons	Hyperion	482	1989-05-26T00:00:00.000Z
4	Frank Herbert	Dune	604	1965-06-01T00:00:00.000Z
5				

Response data format

format	Accept HTTP header	Description
Human Readable		
csv	text/csv	Comma-separated values
json	application/json	JSON (JavaScript Object Notation) human-readable format
tsv	text/tab-separated-values	Tab-separated values
txt	text/plain	CLI-like representation
yaml	application/yaml	YAML (YAML Ain't Markup Language) human-readable format

```
POST /_sql?format=csv
{
  "query": "SELECT * FROM
library ORDER BY page_count
DESC",
  "fetch_size": 5
}
```

Paginating through a large response


```
POST /_sql?format=json
{
  "cursor": "sDXF1ZXJ5QW5kRmV0Y2gBAAAAAAAAAAAEWYUpOYk1QMhRUEtld3RsNnFtYU1hQQ==:BA
```

[Copy as curl](#)[View in Console](#)

Which looks like:


```
{
  "rows" : [
    ["Dan Simmons",      "Hyperion",      482,    "1989-05-26T00:00:00.000",
    ["Iain M. Banks",    "Consider Phlebas", 471,    "1987-04-23T00:00:00.000",
    ["Neal Stephenson",  "Snow Crash",      470,    "1992-06-01T00:00:00.000",
    ["Frank Herbert",    "God Emperor of Dune", 454,    "1981-05-28T00:00:00.000",
    ["Frank Herbert",    "Children of Dune",   408,    "1976-04-21T00:00:00.000",
  ],
  "cursor" : "sDXF1ZXJ5QW5kRmV0Y2gBAAAAAAAAAAAEWODRMaXBUaVlRN21iTlRyWHZWYUdrdw==:B
```

Filtering using Elasticsearch Query DSL



```
POST /_sql?format=txt
{
  "query": "SELECT * FROM library ORDER BY page_count DESC",
  "filter": {
    "range": {
      "page_count": {
        "gte" : 100,
        "lte" : 200
      }
    }
  },
  "fetch_size": 5
}
```

Passing parameters to a query



```
POST /_sql?format=txt
{
    "query": "SELECT YEAR(release_date) AS year FROM library
WHERE page_count > ? AND author = ? GROUP BY year HAVING
COUNT(*) > ?",
    "params": [300, "Frank Herbert", 0]
}
```

Data management



A *data tier* is a collection of nodes with the same data role that typically share the same hardware profile:

Content tier nodes handle the indexing and query load for content such as a product catalog.

Hot tier nodes handle the indexing load for time series data such as logs or metrics and hold your most recent, most-frequently-accessed data.

Warm tier nodes hold time series data that is accessed less-frequently and rarely needs to be updated.

Cold tier nodes hold time series data that is accessed infrequently and not normally updated.

Frozen tier nodes hold time series data that is accessed rarely and never updated, kept in searchable snapshots.

Data management

On hot nodes

```
node.roles: ["data_hot"]
```

On warm nodes

```
node.roles: ["data_warm"]
```

On cold nodes

```
node.roles: ["data_cold"]
```

```
PUT /myindex
```

```
{
```

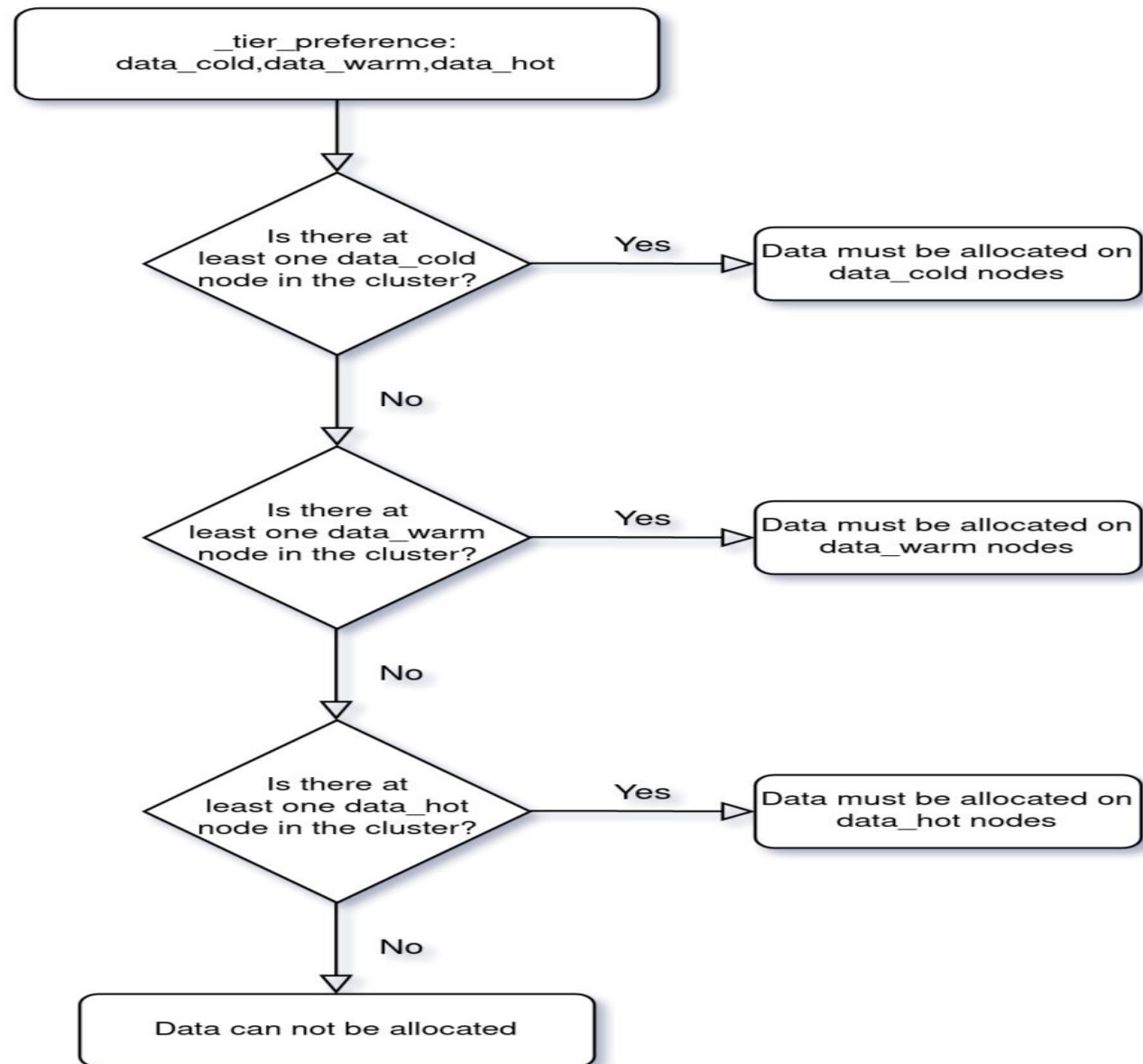
```
  "settings": {
```

```
    "index.routing.allocation.include._tier_preference":
```

```
    "data_cold,data_warm,data_hot"
```

```
  }
```

```
}
```



Elasticsearch security principles



- ❖ Run Elasticsearch with security enabled
- ❖ Run Elasticsearch with a dedicated non-root user
- ❖ Protect Elasticsearch from public internet traffic
- ❖ Implement role based access control

- + Configuring security
- + Updating node security certificates
- + User authentication
- + User authorization
- + Enable audit logging
- + Restricting connections with IP filtering
- + Securing clients and integrations
- + Operator privileges
- + Troubleshooting

Limitations

Configure security for the Elastic Stack

Elastic Security Layers

**Elasticsearch
Development**

Minimal security



**Elasticsearch
Production**

Basic security



Elastic Stack

Basic security

+

TLS for REST

