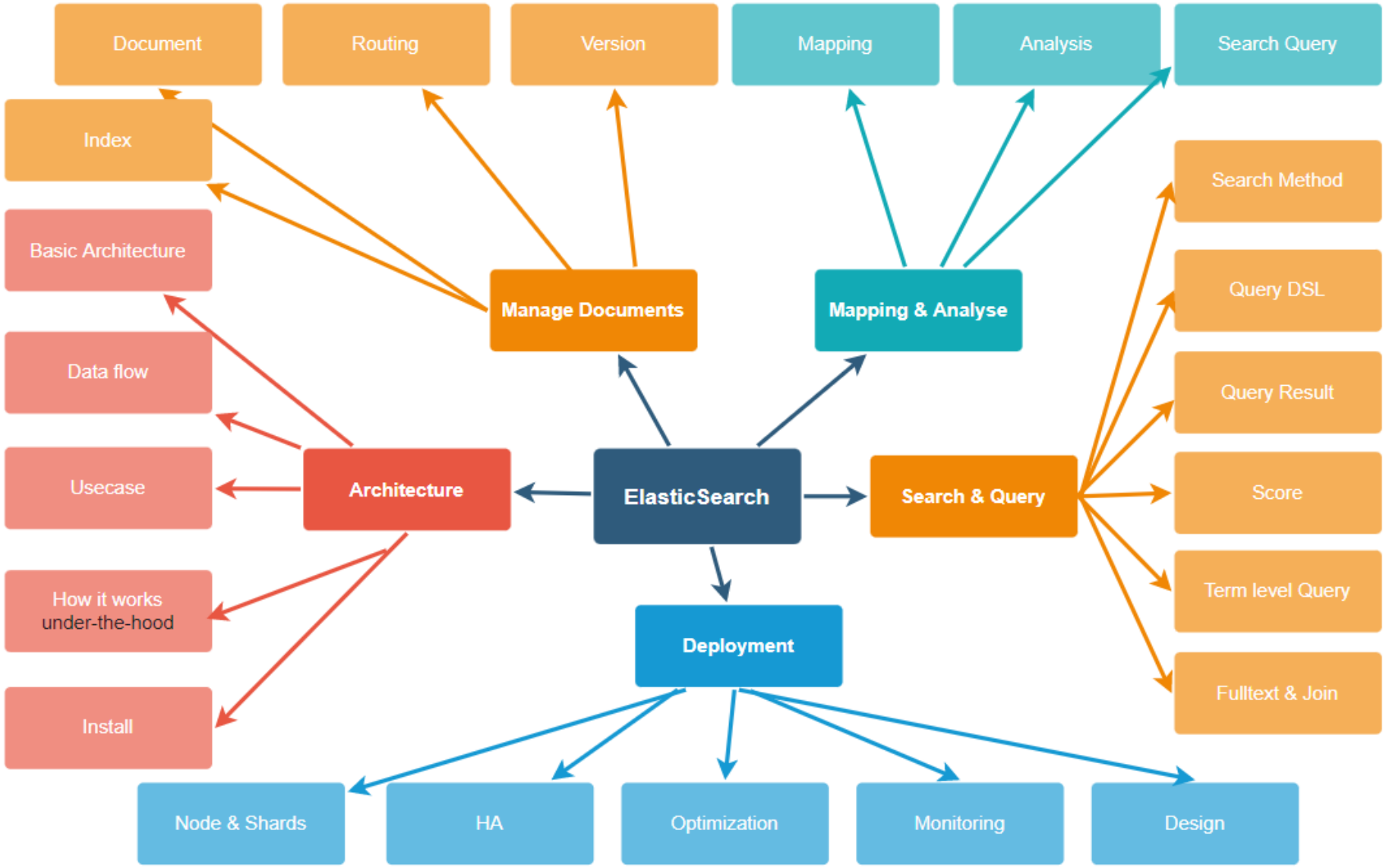
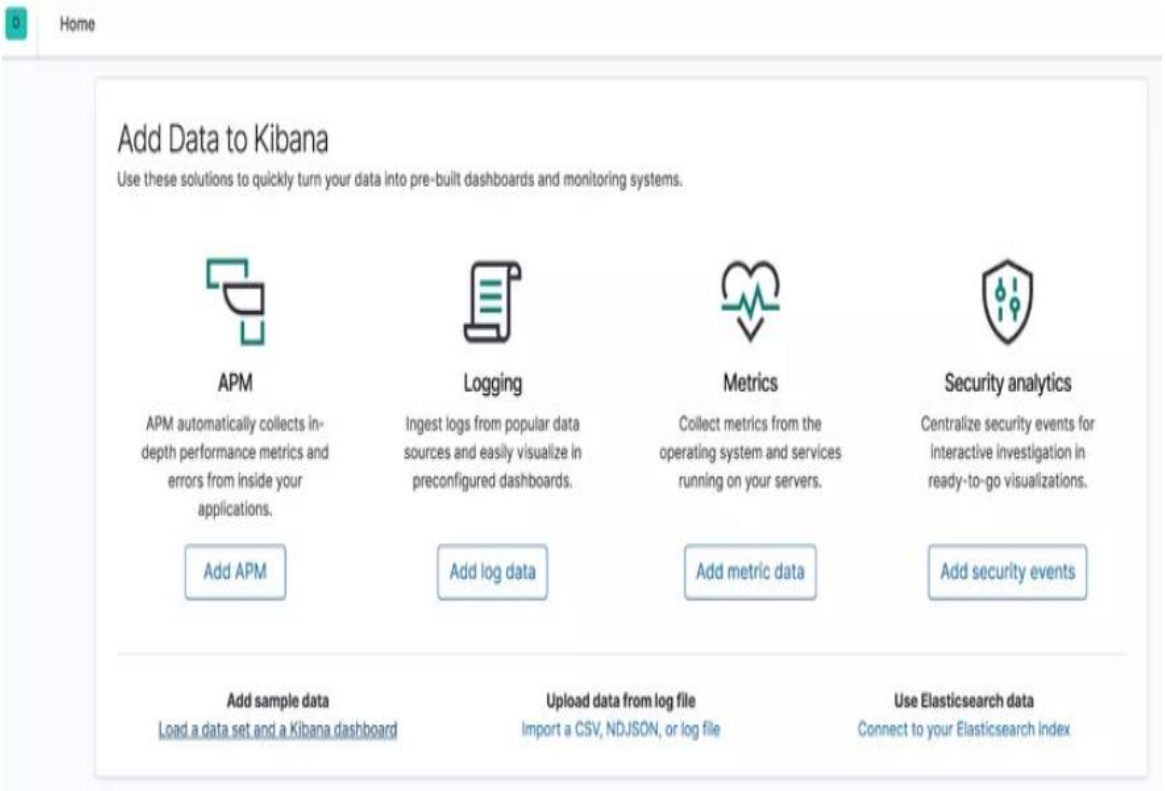
ElasticSearch

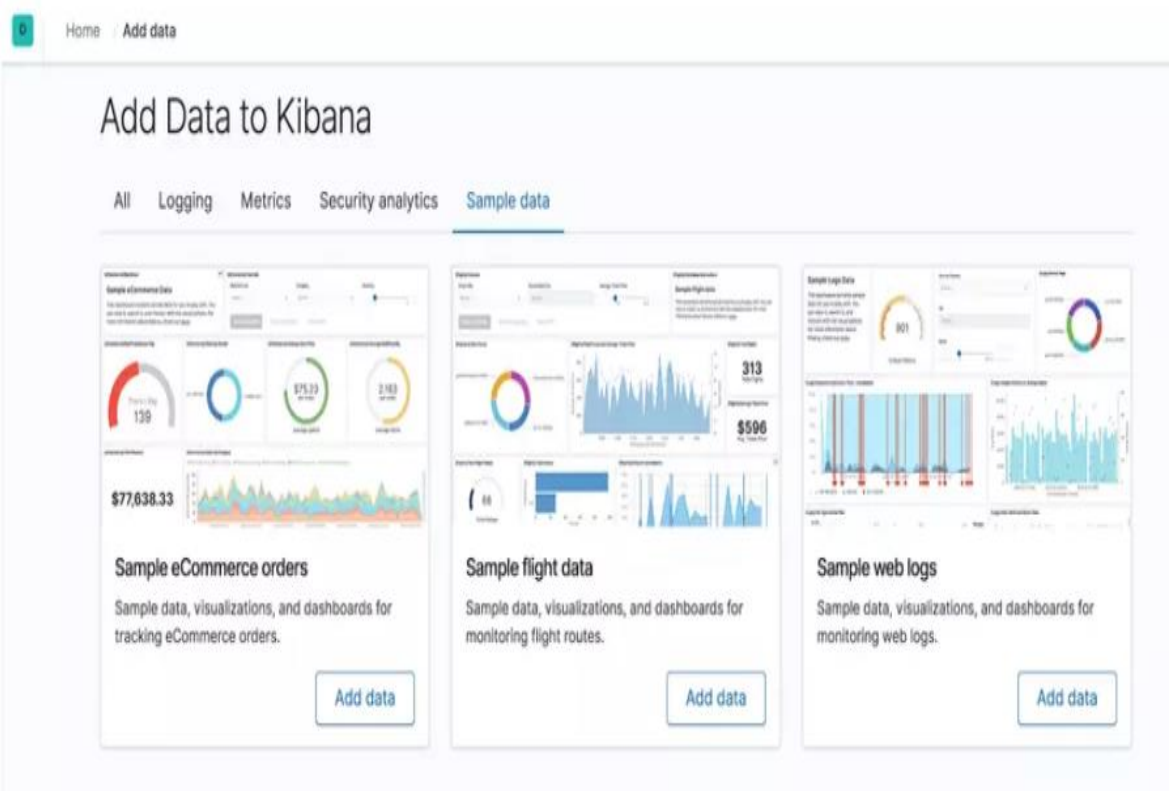
Techmaster

Nội dung

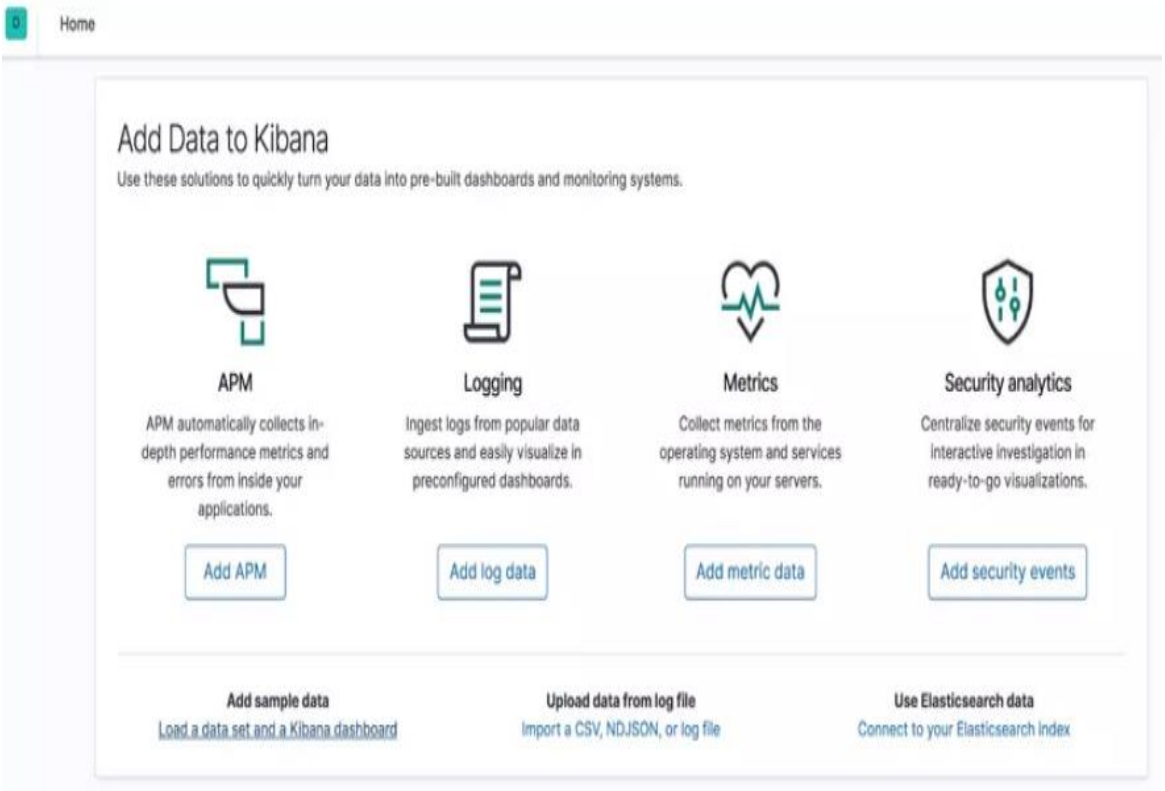

Document
Routing
Version
Mapping
Analysis
Search Query
Index
Search Method
Basic Architecture
Manage Documents
Mapping & Analyse
Query DSL
Data flow
Query Result
Usecase
Architecture
ElasticSearch
Search & Query
Score
How it works under-the-hood
Term level Query
Deployment
Install
Fulltext & Join
Node & Shards
HA
Optimization
Monitoring
Design

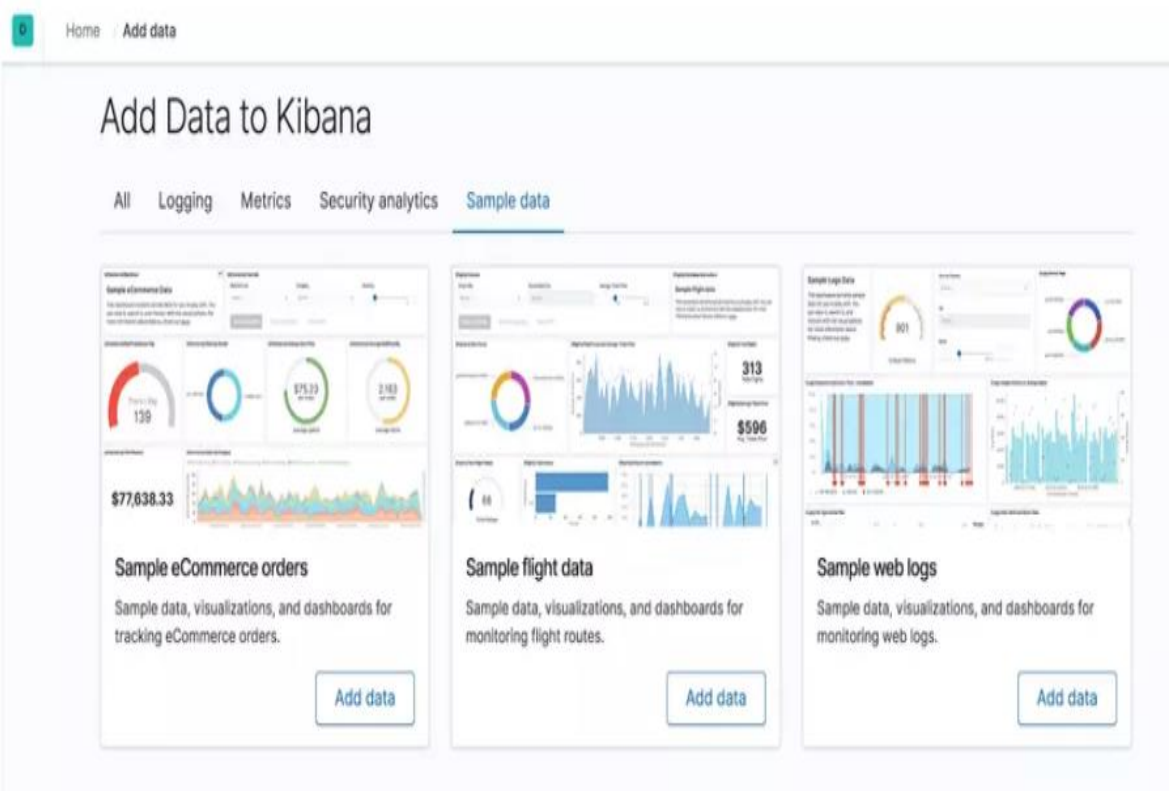
Import data sample





Import data sample



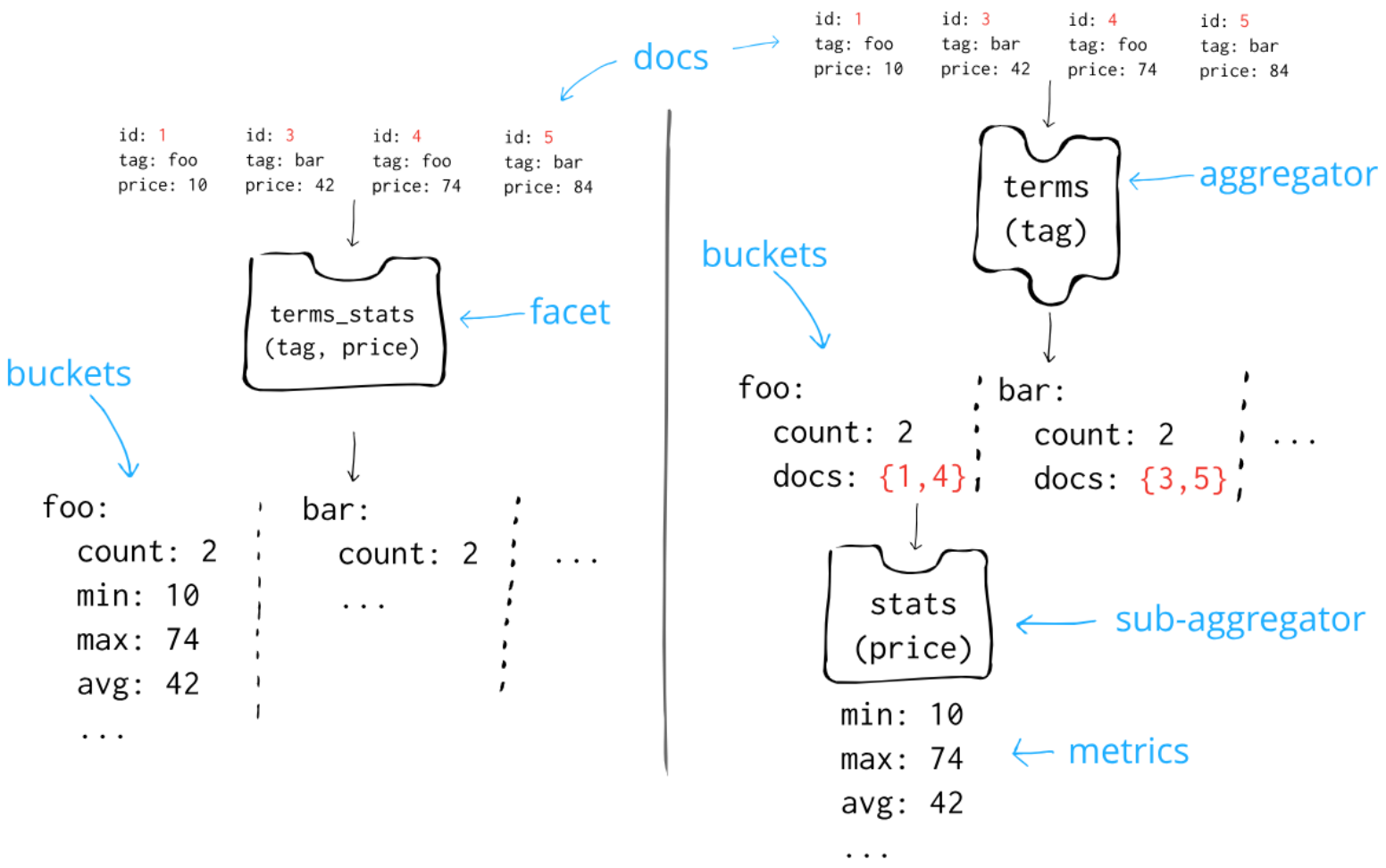


Aggregation

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

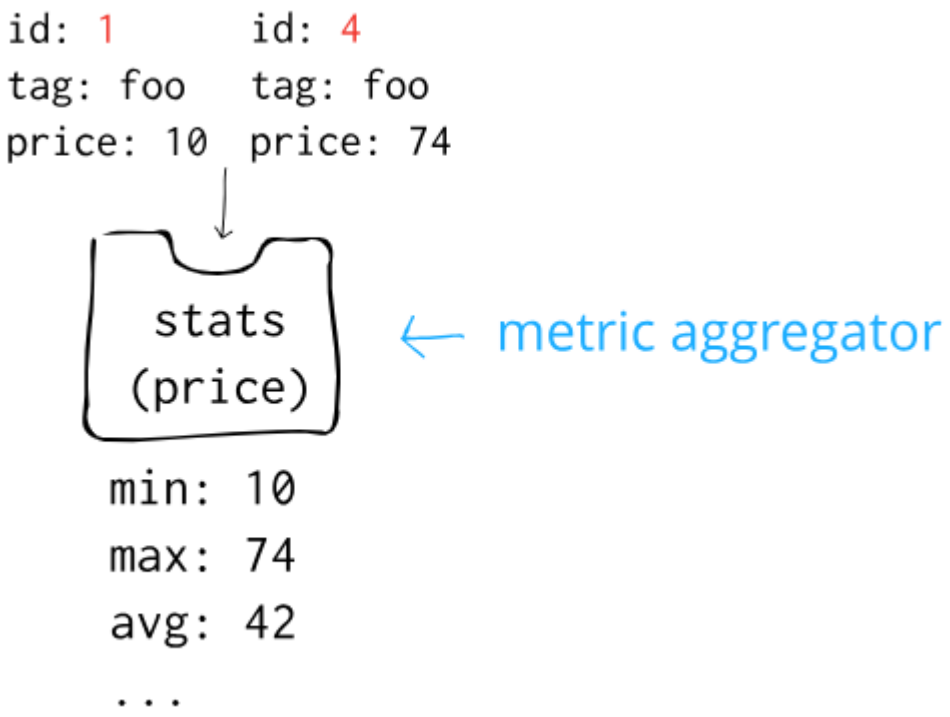
- What's the average load time for my website?
- Who are my most valuable customers based on transaction volume?
- What would be considered a large file on my network?
- How many products are in each product category?

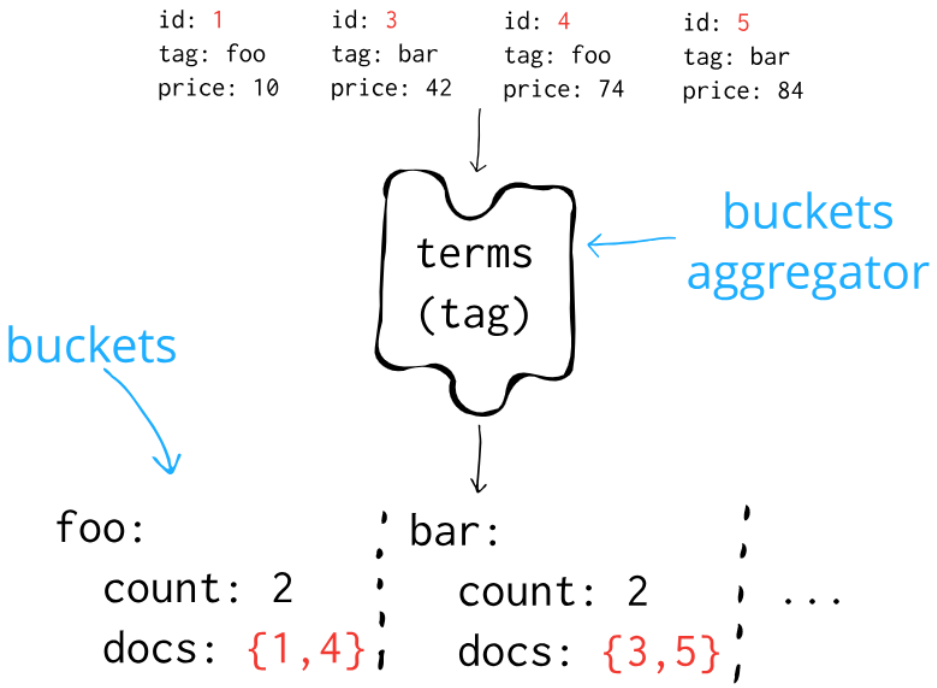
Aggreation vs Facet



Facets vs. aggregations

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"
    }
  }
}
}
```

```
{
  "took" : 372,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 4675,
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [ ]
  },
  "aggregations" : {
    "unique_skus" : {
      "value" : 7186
    }
  }
}
```

Stats Aggregation

```
GET /kibana_sample_data_ecommerce/_search
{
  "size": 0,
 "aggs": {
  "quantity_stats": {
    "stats": {
      "field": "total_quantity"
    }
  }
}
}
```

```
{
  "took" : 33,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 4675,
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [ ]
  },
  "aggregations" : {
    "quantity_stats" : {
      "count" : 4675,
      "min" : 1.0,
      "max" : 8.0,
      "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" } }
            }}}}
```

```
{
  "took" : 43,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 4675,
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [ ]
  },
  "aggregations" : {
    "User_based_filter" : {
      "doc_count" : 100,
      "avg_price" : {
        "value" : 34.85423743206522
      }
```

Terms Aggregation

Dev Tools

History Settings Help

Console Search Profiler Grok Debugger

```
GET /kibana_sample_data_ecommerce/_search
{
  "size": 0,
  "aggs": {
        "Terms_Aggregation" : {
            "terms": {
              "field": "user"}}
        }
    }
```

```
"aggregations" : {
  "Terms_Aggregation" : {
    "doc_count_error_upper_bound" : 0,
    "sum_other_doc_count" : 2970,
    "buckets" : [
      {
        "key" : "elyssa",
        "doc_count" : 348
      },
      {
        "key" : "abd",
        "doc_count" : 188
      },
      {
        "key" : "wilhemina",
        "doc_count" : 170
      },
      {
        "key" : "rabbia",
        "doc_count" : 158
      },
      {
        "key" : "mary",
        "doc_count" : 154
      },
      {
        "key" : "betty",
        "doc_count" : 148
      },
      {
        "key" : "brigitte",
        "doc_count" : 135
      },
```

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

```
GET /nested_aggregation/_search
{
  "aggs": {
    "Nested_Aggregation" : {
      "nested": {
        "path": "Employee"
      },
      "aggs": {
        "Min_Salary": {
          "min": {
            "field": "Employee.salary"
          }
        }
      }
}}}
```

```
      "last" : "Maculum",
      "salary" : "58000"
    },
    {
      "first" : "Vinod",
      "last" : "Kambli",
      "salary" : "63000"
    },
    {
      "first" : "DJ",
      "last" : "Bravo",
      "salary" : "71000"
    },
    {
      "first" : "Jaques",
      "last" : "Kallis",
      "salary" : "75000"
    }
  ]
}
}
]
},
"aggregations" : {
  "Nested_Aggregation" : {
    "doc_count" : 7,
    "Min_Salary" : {
      "value" : 58000.0
```

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'"
}
```

```
 author        |     name       | page_count    |      release_date
---------------+---------------+---------------+------------------------
Dan Simmons    |Hyperion       |482            |1989-05-26T00:00:00.000Z
Frank Herbert  |Dune           |604            |1965-06-01T00:00:00.000Z
```

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": "sDXF1ZXJ5QW5kRmV0Y2gBAAAAAAAAAAEWYUpOYklQMHhRUEtld3RsNnFtYU1hQQ==: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" : "sDXF1ZXJ5QW5kRmV0Y2gBAAAAAAAAAAEWODRMaXBUaVlRN21iTlRyWHZWYUdrdw==: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

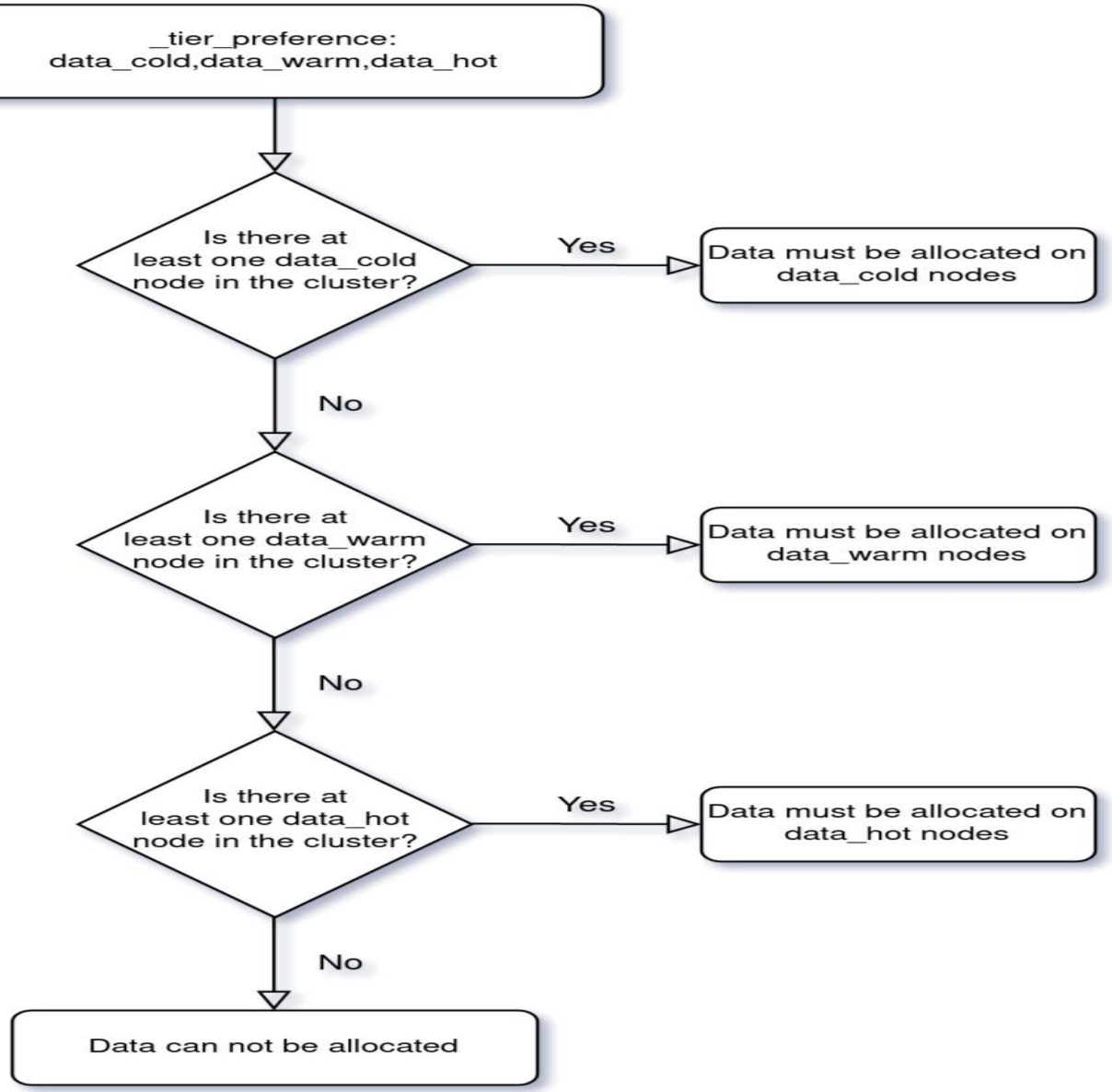
```
# On hot nodes
node.roles: ["data_hot"]

# 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