POLITECNICO
MILANO 1863

SYSTEMS AND METHODS FOR BIG AND UNSTRUCTURED DATA

IR Based Databases - ELK

Marco Brambilla

marco.brambilla@polimi.it

@marcobrambi

Overview – NoSQL Family

Data stored in 4 types:

- Document
- Graph
- Key-value
- Wide-column

Document Database	Graph Databases
Couchbase, MarkLogic, mongoDB	Neo4j, InfiniteGraph – The Distributed Graph Database
Key-value Databases	**Wide Column Stores**
redis, amazon DynamoDB, AEROSPIKE, riak	accumulo, HYPERTABLE INC, Cassandra, APACHE HBASE, Amazon SimpleDB

ELK stack

Kibana

Visualize and Manage

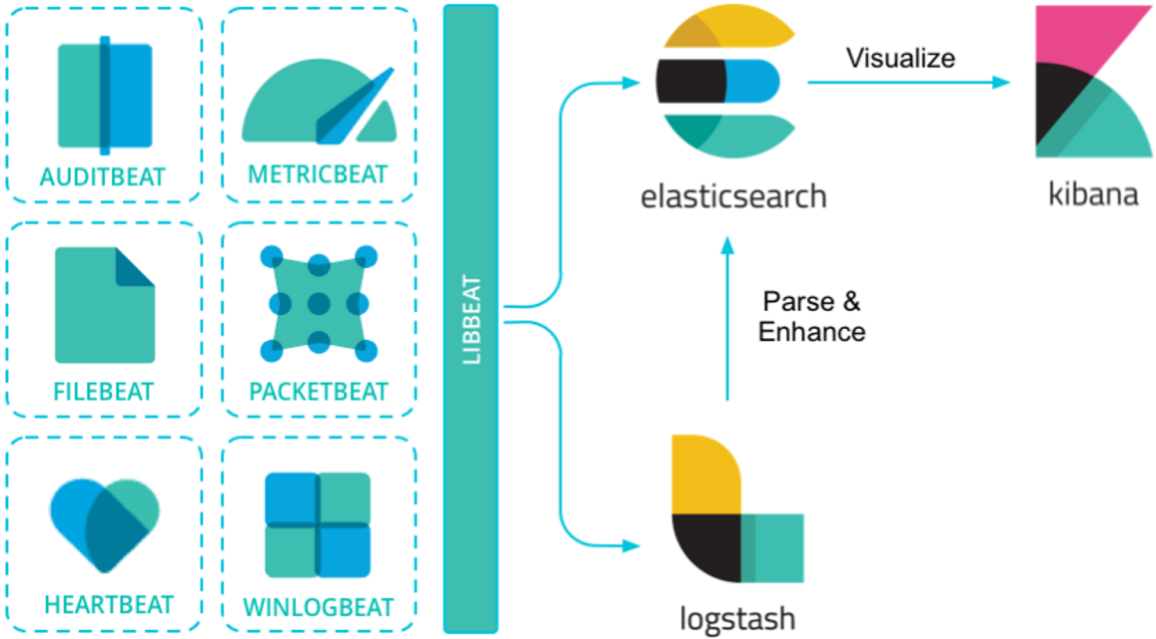
Elasticsearch

Store, Search and Analyze

Logstash + Beats

Ingest

ELK Stack Interactions



Elasticsearch - Introduction

Core of the Elastic Stack

- Search & Analytic engine
  - Near real-time: the time for a document to be searchable after indexing is very short
- Based on Apache Lucene which enables full-text search
- Distributed
- RESTful

Logstash

- Streaming ETL engine of the ELK stack
- Provides centralized data collection, processing and enrichment on the fly
- Data agnostic
- Wide range of integrations and processors
- Ready-to-use monitoring and administrative panes built in Kibana

What is Kibana?

- **Kibana** is an open source data visualization dashboard for Elasticsearch.
- It provides visualization capabilities on top of the content indexed on an Elasticsearch cluster.
- **Kibana** is simple and pretty intuitive to begin with. Despite such simplicity, it is highly customizable, allowing complex and detailed representations.

Elasticsearch

Elasticsearch

Introduction

Elasticsearch - Introduction

Core of the Elastic Stack

- Search & Analytic engine
  - Near real-time: the time for a document to be searchable after indexing is very short
- Based on Apache Lucene which enables full-text search
- Distributed
- RESTful

Elasticsearch - Documents

Elasticsearch stores data structures in JSON documents

- Distributed, can be accessed from any ES node when in a cluster

When stored, new documents are indexed and made fully searchable

Elasticsearch - Indices

Index

- Data organization mechanism
- Define the structure of documents via mappings
- Partitioned in shards

Elasticsearch is based on the concept of **Inverted Index**

Elasticsearch - Shards and Replicas

Shards: distribute operations to

- Increase resistance to faults (e.g. crashes when an indices get too big)
- Improve performance

Replicas: copies of shards stored on a different node

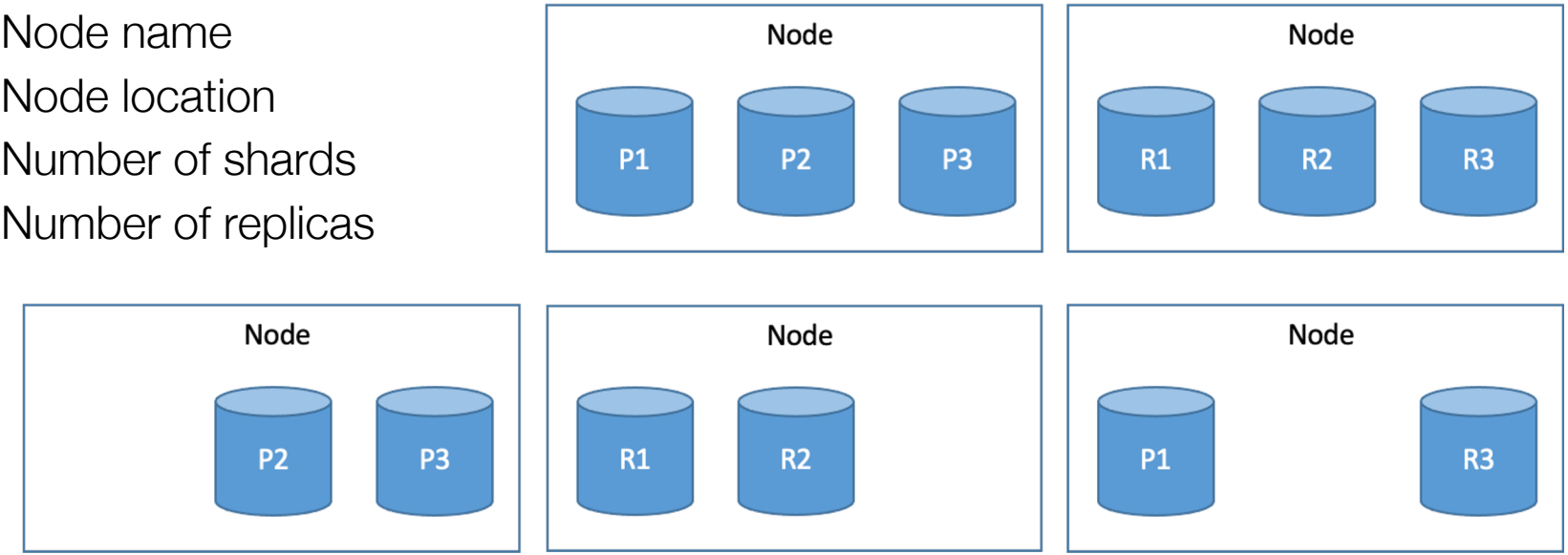
Write operation → Primary → Replica (to copy)

Read operation → Routed to primary or replica

Elasticsearch - Shards and Replicas

Very easy configuration!

- Node name
- Node location
- Number of shards
- Number of replicas

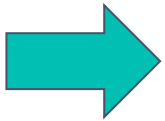


Elasticsearch - Inverted Index

Lists every unique word from every document and identifies which documents a word appears in

Document 1*: "The quick brown fox jumped over the lazy dog"*

Document 2: *"Quick brown foxes leap over lazy dogs in summer"*



"the" -> D1
"quick" -> D1,D2
"brown" -> D1,D2
"fox" -> D1
"jumped" -> D1
"over" -> D1,D2
..
"summer" -> D2

Elasticsearch - TF-IDF

Results are scored with Lucene's Practical Scoring Function, which uses TF-IDF: Term Frequency-Inverse Document Frequency

- Measures the **relevance** of a term inside a document
- Penalizes the words that appear **too often** across different documents

Term Frequency: frequency of term *i* in document *j*

$$\mathrm{tf}_{i,j} = \frac{n_{i,j}}{|d_j|}$$

Number of time term *i* appear in document j

Number of terms in document *j*

Elasticsearch - TF-IDF

Inverse Document Frequency: measures how common a term is across the collection of documents, the lower the less important that term is

$$\mathrm{idf}_i = \log \frac{|D|}{|\{d : i \in d\}|}$$

Number of documents in the collection

Number of documents containing the term *i*

TF-IDF:

$$(\text{tf-idf})_{i,j} = \mathrm{tf}_{i,j} \times \mathrm{idf}_i$$

Elasticsearch - Mapping

By defining a mapping for an index you can tell Elasticsearch the types of your documents' fields

- Decide which fields are searchable
- Enable full-text search, time and geo-based queries

Be careful as you CANNOT change the mapping on an existing index that already has documents!

Elasticsearch - Schemaless

Elasticsearch tries to guess the structure of the documents

- Dynamic mapping, risky (e.g. dates could be parsed as strings)

Is Elasticsearch really schemaless?


```
GET /nyc-restaurants/_mapping
```

```
{
  "nyc-restaurants" : {
    "mappings" : {
      "_meta" : {
        "created_by" : "ml-file-data-visualizer"
      },
      "properties" : {
        "@timestamp" : {
          "type" : "date"
        },
        "ACTION" : {
          "type" : "text"
        },
        "BBL" : {
          "type" : "long"
        },
```

Elasticsearch - Interaction

- Interactions with Elasticsearch happen through requests to REST endpoints
- Actions depend on HTTP verbs
  - GET: used to read document, indices metadata, etc.
  - POST & PUT: often used to create new documents, indices, etc.
  - DELETE

Requests can be sent from

- Command Line
- Software packages like Postman
- Developer Tools built in Kibana

Elasticsearch - Interaction

Beware of the differences between POST and PUT

- POST doesn't require the ID of the resource → duplicates
  - When omitted, Elasticsearch takes care of creating and assigning IDs to documents
  - example: **POST /index_name/_doc**

- PUT requires the ID of the resource → create or update the same one
  - example: **PUT /index_name/_doc/document_id**

Elasticsearch - Interaction

Example

```
POST /my_index/_doc
{
  "author":"Andrea",
  "title":"Set up Elasticsearch and Kibana in 15 minutes!",
  "date":"Tue, 23 Feb 2021 11:40:00 +0000",
  "categories": ["elasticsearch","tutorial","data science","bigdata"],
  "lang":"en-US"
}
```

Elasticsearch - Interaction

Example

```
{
  "_index": "blog",
  "_type": "_doc",
  "_id": ”jYpU_XQBkGeBYJKltRh8",
  "_version": 1,
  "result": "created",
  ...
}
```

ID auto-generated by Elasticsearch

```
{
  "_index": "blog",
  "_type": "_doc",
  "_id": "my_id",
  "_version": 1,
  "result": "created",
  ...
}
```

User-specified ID

Elasticsearch - Comparison with Relational DBs

RDBMS

- Table
- Row
- Column
- Schema

Elasticsearch

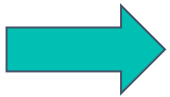
- Index
- Document
- Field
- Mapping

Creating and index

PUT /index_name

Example:

```
PUT /my_index
```



```
{
  "acknowledged" : true,
  "shards_acknowledged" : true,
  "index" : "my_index"
}
```

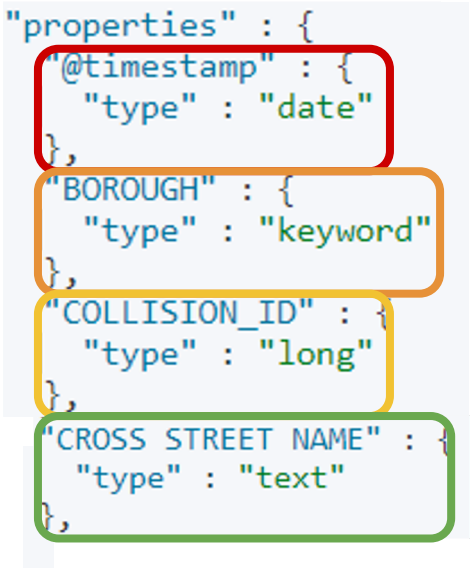
How to define a mapping

PUT index_name/_mapping

Request body: mapping structure

```
PUT /my_index/_mapping
{
  "properties": {
    "field_name": {
      "type": "text"
    }
  }
}
```

How to define a mapping



Date is used for dates you can specify the format.

Keyword is used for structured content such as email, tags, postcode etc…

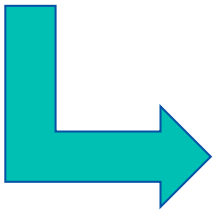
Long is used for 64-bit integers.

Text is used for full-text search. With **analyzer** you can specify how the field is pre-processed.

Retrieving a document

Query:

```
GET /nyc-restaurants/_doc/pZpozncBDaS7kSRanan2
```




```
"_index" : "nyc-restaurants",
"_type" : "_doc",
"_id" : "pZpozncBDaS7kSRanan2",
"_version" : 1,
"_seq_no" : 382695,
"_primary_term" : 1,
"found" : true,
"_source" : {
  "DBA" : "BENTO HOUSE",
  "RECORD DATE" : "02/22/2021",
  "VIOLATION CODE" : "22G",
  "PHONE" : "9173063883",
  "Latitude" : 40.760198166197,
  "BBL" : 4049800055,
  "BUILDING" : "136-75",
  "BORO" : "Queens",
  "CUISINE DESCRIPTION" : "Chinese",
  "CAMIS" : 50071889,
  "Community Board" : 407,
  "INSPECTION TYPE" : "Administrative Miscellaneous / Initial Inspection",
  "BIN" : 4112507,
  "ZIPCODE" : 11354,
  "INSPECTION DATE" : "02/17/2021",
  "STREET" : "ROOSEVELT AVENUE",
  "Longitude" : -73.827606355065,
  "Census Tract" : 87100,
  "@timestamp" : "2021-02-17T00:00:00.000+01:00",
  "ACTION" : "Violations were cited in the following area(s).",
  "Council District" : 20,
  "NTA" : "QN22",
  "location" : "40.760198166197,-73.827606355065"
```

Language analyzer

- It removes stopwords of a given language
- Perform stemming
- Elasticsearch provides several analyser for different languages

Other built-in analyzers

Standard Analyzer

The standard analyzer divides text into terms on word boundaries, as defined by the Unicode Text Segmentation algorithm. It removes most punctuation, lowercases terms, and supports removing stop words.

Simple Analyzer

The simple analyzer divides text into terms whenever it encounters a character which is not a letter. It lowercases all terms.

Whitespace Analyzer

The whitespace analyzer divides text into terms whenever it encounters any whitespace character. It does not lowercase terms.

Other built-in analyzers

Stop Analyzer

The stop analyzer is like the simple analyzer, but also supports removal of stop words.

Keyword Analyzer

The keyword analyzer is a “noop” analyzer that accepts whatever text it is given and outputs the exact same text as a single term.

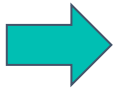
Pattern Analyzer

It performs the analysis according to a custom regexp pattern

Test an analyzer

If you are unsure on what analyzer to use you can test it.

```
POST /_analyze
{
  "analyzer": "standard",
  "text": "The 2 QUICK Brown-Foxes jumped over the lazy dog's bone."
}
```

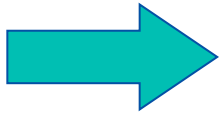


```
{
  "tokens" : [
    {
      "token" : "the",
      "start_offset" : 0,
      "end_offset" : 3,
      "type" : "<ALPHANUM>",
      "position" : 0
    },
    ...
  ]
}
```

Search - Query

Search - Query

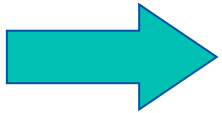
```
GET /nyc-restaurants/_search
{
  "query": {
    "match": {
      "INSPECTION TYPE" : "Cycle Inspection / In
    }
  }
}
```




```
{
  "_index" : "nyc-restaurants",
  "_type" : "_doc",
  "_id" : "apVnzncBDaS7kSRahykR",
  "_score" : 0.53188044,
  "_source" : {
    "DBA" : "U LIKE CHINESE TAKE OUT",
    "SCORE" : 20,
    "RECORD DATE" : "02/22/2021",
    "VIOLATION CODE" : "04L",
    "PHONE" : "2129426668",
    "Latitude" : 40.867494224968,
    "BBL" : 1022350020,
    "BUILDING" : "4926",
    "BORO" : "Manhattan",
    "CUISINE DESCRIPTION" : "Chinese",
    "CAMIS" : 50016036,
    "VIOLATION DESCRIPTION" : "Evidence of mice or live mice pres
      /or non-food areas.",
    "Community Board" : 112,
    "INSPECTION TYPE" : "Cycle Inspection / Initial Inspection",
```

Search - Query

```
GET /nyc-restaurants/_search
{
  "query": {
    "match": {
      "BORO": "Queens"
    }
  }
}
```

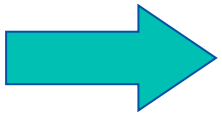



```
{
  "_index" : "nyc-restaurants",
  "_type" : "_doc",
  "_id" : "aZVnzncBDaS7kSRahykR",
  "_score" : 1.4879451,
  "_source" : {
    "DBA" : "NEW YORK SHAKE & BURGER",
    "SCORE" : 10,
    "RECORD DATE" : "02/22/2021",
    "VIOLATION CODE" : "02G",
    "PHONE" : "7185231900",
    "Latitude" : 40.696199622634,
    "BBL" : 4100310020,
    "BUILDING" : "147-01",
    "BORO" : "Queens",
```

Search - Query

Borough (BORO) was mapped as a **keyword**.
A keyword **does not have any analyser**!

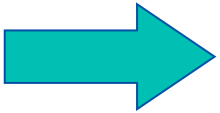
```
GET /nyc-restaurants/_search
{
  "query": {
    "match": {
      "BORO": "queens"
    }
  }
}
```



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

Count - Query

```
GET /nyc-restaurants/_count
{
  "query": {
    "match": {
      "SCORE" : 13
    }
  }
}
```



```
{
  "count" : 32930,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  }
}
```

Match - Errors

Be careful when you use the **match** identifier!


```
{
  "error" : {
    "root_cause" : [
      {
        "type" : "parsing_exception",
        "reason" : "[match] query doesn't support multiple fields, found [SCORE] and [BORO]",
        "line" : 5,
        "col" : 16
      }
    ],
    "type" : "parsing_exception",
    "reason" : "[match] query doesn't support multiple fields, found [SCORE] and [BORO]",
    "line" : 5,
    "col" : 16
  },
  "status" : 400
}
```


You can only match 1 field at time

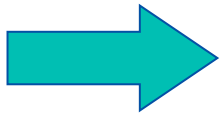
Search - Filter

- Exact match
- For fields that are not analyzed
- No relevance
- The filter is either satisfied or not
- Cacheable

Search - Filter

```
GET /nyc-restaurants/_search
{
  "query": {
    "bool": {
      "filter": {
        "term": {
          "VIOLATION CODE": "04M"
        }
      }
    }
  }
}
```

IN Query




```
"hits" : {
  "total" : {
    "value" : 8711,
    "relation" : "eq"
  },
  "max_score" : 0.0,
  "hits" : [
    {
      "_index" : "nyc-restaurants",
      "_type" : "_doc",
      "_id" : "aJVnzncBDaS7kSRahykR",
      "_score" : 0.0,
      "_source" : {
        "DBA" : "XIFU FOOD",
        "SCORE" : 9,
```

Search - Filter

```
GET /nyc-restaurants/_search
{
  "query": {
    "bool": {
      "must": [
        {}
      ],
      "must_not": [
        {}
      ],
      "should": [
        {}
      ]
    }
  }
}
```

AND

NOT

OR

```
"must_not": [
  {
    "term": {
      "VIOLATION CODE": {
        "value": "02G"
      }
    }
  }
],
```

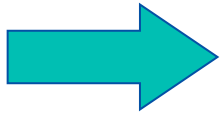
```
"must": [
  {"range": {
    "INSPECTION DATE": {
      "gte": "02/23/2020",
      "lte": "02/23/2021"
    }
  }}
],
```

```
"should": [
  {"term": {
    "BORO": {
      "value": "Queens"
    }
  }}
]
```

Search - Filter

```
GET /nyc-restaurants/_search
{
  "query": {
    "bool": {
      "must": [
        {"range": {
          "INSPECTION DATE": {
            "gte": "02/23/2020",
            "lte": "02/23/2021"
          }
        }}
      ]
    }
  }
}
```

AND

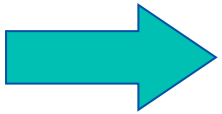


```
"hits" : {
  "total" : {
    "value" : 9510,
    "relation" : "eq"
  },
  "max_score" : 1.0,
  "hits" : [
    {
      "_index" : "nyc-restaurants",
      "_type" : "_doc",
      "_id" : "upVnzncBDaS7kSRahykR",
      "_score" : 1.0,
      "_source" : {
        "DBA" : "TACOS Y QUESADILLAS MEXICO",
        "SCORE" : 19,
        "RECORD DATE" : "02/22/2021",
        "VIOLATION CODE" : "02G",
        "PHONE" : "7182714260",
        "Latitude" : 40.743890908308,
```

Search - Filter

```
GET /nyc-restaurants/_search
{
  "query": {
    "bool": {
      "must": [
        {"range": {
          "INSPECTION DATE": {
            "gte": "02/23/2020",
            "lte": "02/23/2021"
          }
        }}
      ],
      "must_not": [
        {
          "term": {
            "VIOLATION CODE": {
              "value": "02G"
            }
          }
        }
      ]
    }
  }
}
```

NOT

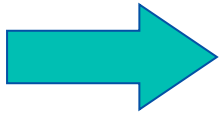


```
"hits" : {
  "total" : {
    "value" : 8923,
    "relation" : "eq"
  },
  "max_score" : 1.0,
  "hits" : [
    {
      "_index" : "nyc-restaurants",
      "_type" : "_doc",
      "_id" : "L5VnzncBDaS7kSRahyoR",
      "_score" : 1.0,
      "_source" : {
        "DBA" : "QUICK GRILL JAPAN",
        "RECORD DATE" : "02/22/2021",
        "VIOLATION CODE" : "20D",
        "PHONE" : "7186821786",
        "Latitude" : 40.613111633522,
        "BBL" : 5015440100,
        "BUILDING" : "1445",
```

Search - Filter

```
GET /nyc-restaurants/_search
{
  "query": {
    "bool": {
      "must": [
        {"range": {
          "INSPECTION DATE": {
            "gte": "02/23/2020",
            "lte": "02/23/2021"
          }
        }}
      ],
      "must_not": [
        {
          "term": {
            "VIOLATION CODE": {
              "value": "02G"
            }
          }
        }
      ],
      "should": [
        {"term": {
          "BORO": {
            "value": "Queens"
          }
        }}
      ]
    }
  }
}
```

OR

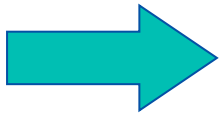


```
"hits" : {
  "total" : {
    "value" : 8923,
    "relation" : "eq"
  },
  "max_score" : 2.487945,
  "hits" : [
    {
      "_index" : "nyc-restaurants",
      "_type" : "_doc",
      "_id" : "c5VnzncBDaS7kSRahyoR",
      "_score" : 2.487945,
      "_source" : {
        "DBA" : "PEGGY DEMPSEYS",
        "SCORE" : 12,
        "RECORD DATE" : "02/22/2021",
        "VIOLATION CODE" : "04N",
        "PHONE" : "7183263707",
        "GRADE DATE" : "02/27/2020",
        "Latitude" : 40.722747793365,
        "BBL" : 4027510026,
        "BUILDING" : "6414",
        "BORO" : "Queens",
        "CUISINE DESCRIPTION" : "American",
```

Search - Filter

```
GET /nyc-restaurants/_search
{
  "query": {
    "bool": {
      "filter": {
        "prefix": {
          "VIOLATION DESCRIPTION": "flies"
        }
      }
    }
  }
}
```

Prefix Filter




```
"hits" : {
  "total" : {
    "value" : 10000,
    "relation" : "gte"
  },
  "max_score" : 0.0,
  "hits" : [
    {
      "_index" : "nyc-restaurants",
      "_type" : "_doc",
      "_id" : "gpVnzncBDaS7kSRahykR",
      "_score" : 0.0,
      "_source" : {
        "DBA" : "NUMERO 28",
        "SCORE" : 20,
        "RECORD DATE" : "02/22/2021",
        "VIOLATION CODE" : "04N",
        "PHONE" : "2127728200",
        "Latitude" : 40.769530136088,
        "BBL" : 1014490024,
        "BUILDING" : "1431",
        "BORO" : "Manhattan",
        "CUISINE DESCRIPTION" : "Italian",
        "CAMIS" : 41642694,
        "VIOLATION DESCRIPTION" : "Filth flies or
```

Filter and Query combination

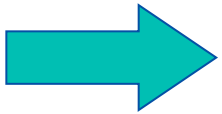
- You can combine filter and queries
- It’s more **efficient** than doing only queries
- You can increase the **relevance** of a document at query time

Filter and Query combination

```
GET /nyc-restaurants/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "SCORE": "20"
          }
        }
      ],
      "filter": [
        {
          "term": {
            "BORO": {
              "value": "Manhattan"
            }
          }
        }
      ]
    }
  }
}
```

Query

Filter



```
"hits" : {
  "total" : {
    "value" : 3935,
    "relation" : "eq"
  },
  "max_score" : 1.9322574,
  "hits" : [
    {
      "_index" : "nyc-restaurants",
      "_type" : "_doc",
      "_id" : "apVnzncBDaS7kSRahykR",
      "_score" : 1.9322574,
      "_source" : {
        "DBA" : "U LIKE CHINESE TAKE OUT",
        "SCORE" : 20,
        "RECORD DATE" : "02/22/2021",
        "VIOLATION CODE" : "04L",
        "PHONE" : "2129426668",
        "Latitude" : 40.867494224968,
        "BBL" : 1022350020,
        "BUILDING" : "4926",
        "BORO" : "Manhattan",
        "CUISINE DESCRIPTION" : "Chinese",
        "CAMIS" : 50016036,
```

Multi index search

Elasticsearch allows to search in **multiple index** at the same time

- /_search
- index_1,index_2/_search
- prefix*/_search

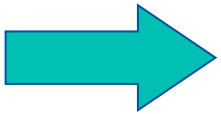
Aggregations

How can we perform aggregations in Elasticsearch? Let's consider the following SQL query.

SELECT score, count(*)

FROM nyc-restaurants

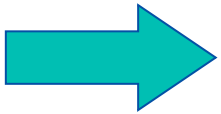
GROUP BY score



```
GET /nyc-restaurants/_search
{
  "size": 0,
  "aggs": {
    "score_groups": {
      "terms": {
        "field": "SCORE"
      }
    }
  }
}
```

Aggregations

```
GET /nyc-restaurants/_search
{
  "size": 0,
  "aggs": {
    "score_groups": {
      "terms": {
        "field": "SCORE"
      }
    }
  }
}
```



```
"aggregations" : {
  "score_groups" : {
    "doc_count_error_upper_bound" : 0,
    "sum_other_doc_count" : 184789,
    "buckets" : [
      {
        "key" : 12,
        "doc_count" : 41375
      },
      {
        "key" : 13,
        "doc_count" : 32930
      },
      {
        "key" : 10,
        "doc_count" : 21892
      },
      {
        "key" : 11,
        "doc_count" : 21330
      },
```

Aggregations

Elasticsearch provides many types of aggregations

- Metrics aggregations
- Bucket aggregations
- Pipeline aggregations
- Matrix aggregations

Logstash

Logstash

Introduction

POLITECNICO
MILANO 1863

Logstash

- Streaming ETL engine of the ELK stack
- Provides centralized data collection, processing and enrichment on the fly
- Data agnostic
- Wide range of integrations and processors
- Ready-to-use monitoring and administrative panes built in Kibana

Beats

- Platform for data shippers
- Collect and ship logs and metrics from hosts or containers
- Many available
  - Filebeat
  - Metricbeat
  - Packetbeat
  - Heartbeat

Other Beats can be explored here

Logstash - Working with Beats

- Beats focus on data collection and shipping
- Logstash focuses on processing and data normalization

Logstash can also receive data from devices for which Beats are not deployed

- Via TCP, UDP, HTTP protocols
- Pool-based inputs like JDBC

List of the available input plugins

Logstash - Processing

- Filter plugins (a.k.a. Processors)
  - Help with data wrangling
  - Use them to build pipelines that structure, normalize and enrich data
- Examples:
  - Derive geographic coordinates from IP addresses
  - Exclude sensitive fields

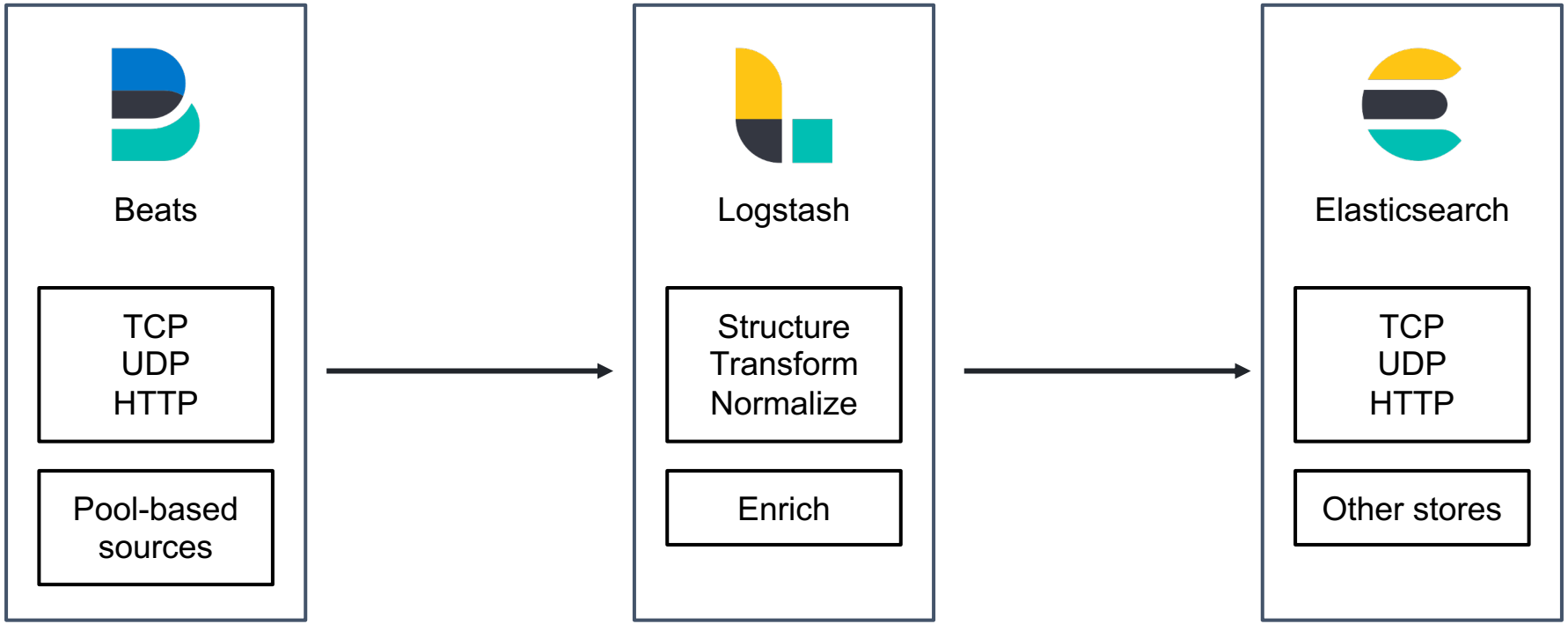
List the available filters

Logstash - Emission

- Plugins are available to emit data to Elasticsearch, to other data stores or via TCP, UDP and HTTP protocols

List of the available output plugins

Logstash - Data flow



Modules

- Provided by Logstash and Beats
- Available only for specific data types
- Data-to-dashboard functionalities
  - Automated parsing and enrichment in Logstash
  - Custom schema in Elasticsearch
  - Default dashboards in Kibana

Logstash

Logstash

Building blocks

Logstash - Events

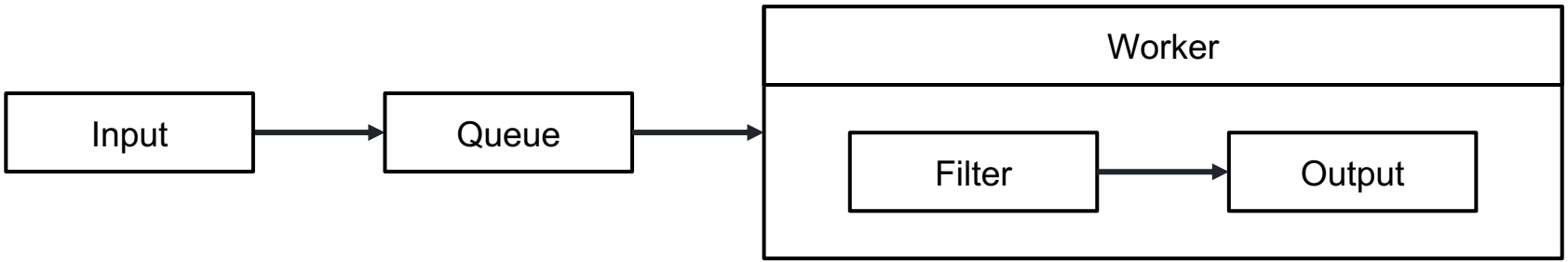
- Primary unit of data in Logstash
- Similar to JSON documents
- Flow through pipelines

```
{
  “@timestamp" => 2021-16-02T01-01-01,
  “message" => “hello",
  “other_field" => {
    “nested_field" => 5678
  }
}
```

Logstash - Pipeline

Logical flow of data

- Supports multiple inputs
- Single queue to buffer data
  - Either in-memory or persistent
- Workers ensure server-side scalability



Logstash - Instance

Logstash instance = Logstash process

Multiple pipelines are supported in a single Logstash instance

- Separate data flows

Logstash - Codecs

- Used to change the data representation of an event
  - Serialization and de-serialization
- Can be used as part of an input or output

```
input{
  file {
    path => “file.log”
    codec => json
  }
}
```

Parse JSON data in *file.log*

```
output{
  stdout {
    codec => rubydebug
  }
}
```

Format console output with Ruby Amazing Print

Logstash - Filters examples

mutate: field manipulation filter

- Convert types
- Add, rename, replace, copy fields
- Upper and lowercase transformations
- Join arrays
- Split field into arrays

split: divide a single event into multiple events

drop: delete an event

Logstash - Enrichment filters examples

geoip and **dns**: enrich IP addresses

useragent: record information like browser type from web logs

translate: use local data to translate parts of the events

elasticsearch: query Elasticsearch

jdbc: query databases that support Java

Logstash - Pipeline example

```
input {
  beats { port => 5043 }
}
filter {
  mutate { lowercase => [“message"] }
}
output {
  elasticsearch {}
}
```

- Receive data from a host that uses Beats
- Process that input via a filter
- Output directly to Elasticsearch

Logstash - Message management

- “At Least Once” message delivery
  - In most conditions messages are delivered *exactly* once
  - Unclean shutdowns could lead to duplicates

- DLQ - Dead Letter Queue
  - Events that are not processable
  - Due to “at least once” policy these are undeliverable
  - Avoids losing those events and halting the pipeline, freeing processing resources for subsequent events

Kibana

What is Kibana?

Kibana is an open source data visualization dashboard for Elasticsearch.

It provides visualization capabilities on top of the content indexed on an Elasticsearch cluster.

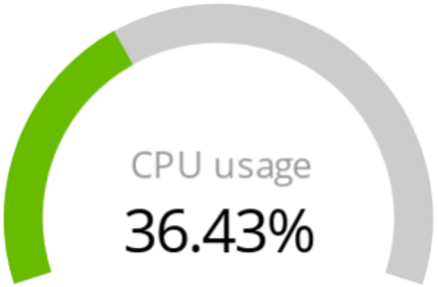
Kibana is simple and pretty intuitive to begin with. Despite such simplicity, it is highly customizable, allowing complex and detailed representations.

Kibana has many different kinds of subscriptions, although the main differences are between the free one and the others. The details about the differences are available on the Kibana official website.

What is Kibana?

Through **Kibana** you can aggregate, organize, filter and represent data using many different data types and representations.

A wide variety of graphs is available, although some data types, analyses and graphs are only available through a paid subscription.

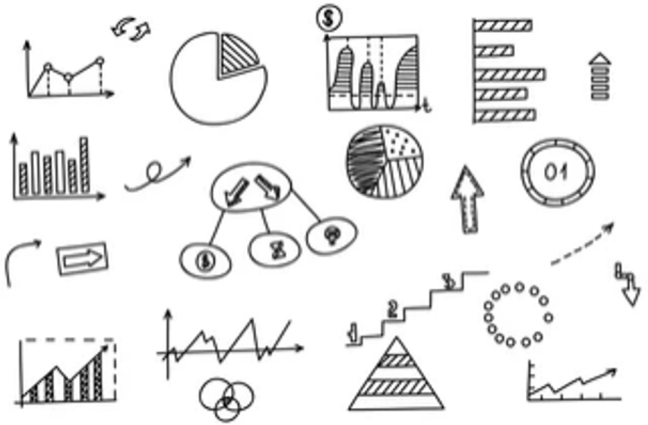


What is Kibana?

Kibana is really powerful. It allows the creation of customized dashboards combining together basic representations, location-based analyses, time series, machine learning and graphs/networks.

Agenda

- Data Ingestion
  - Add Data
  - Agents & Fleets
  - Data Upload
  - Geospatial Data
- Data Overview
- Dashboard Creation
  - Interface Overview
  - Lens
  - Maps
  - Time Series Visual Builder (TSVB)
  - Filtering

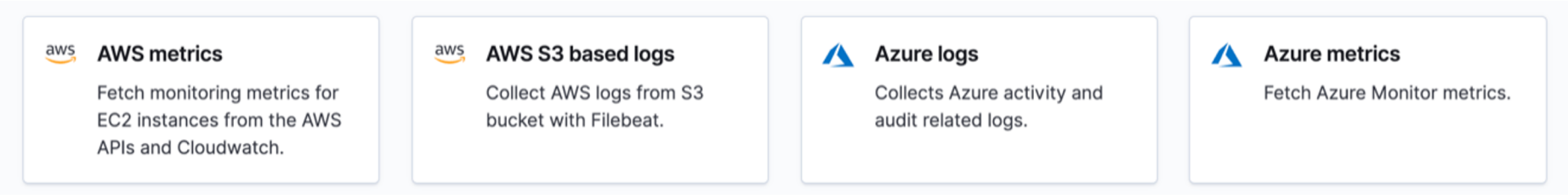


Data Ingestion - Add Data

Following, an overview of the upload processes listed before is provided.

Add Data

Import data from other services through some modules to periodically collect the data and send it to Elasticsearch. The picture below provides some examples of such modules.



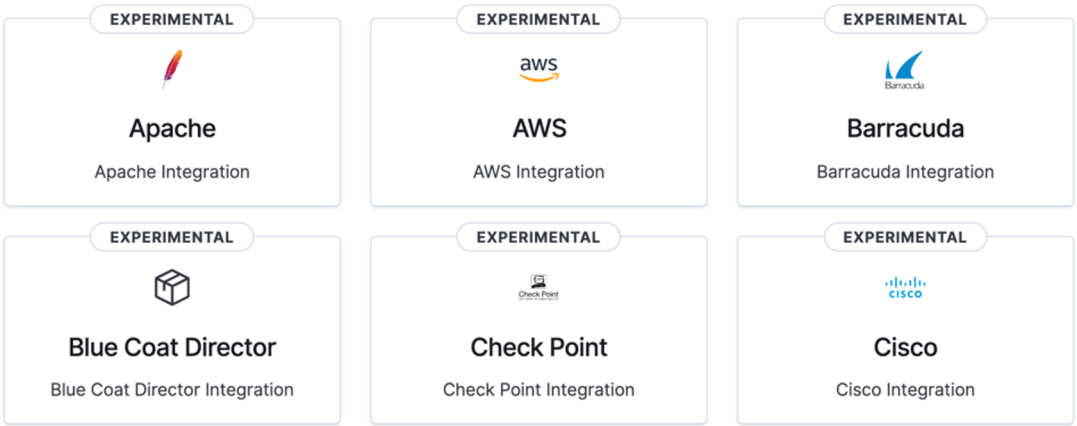
Data Ingestion - Elastic Agents

Add Elastic Agent

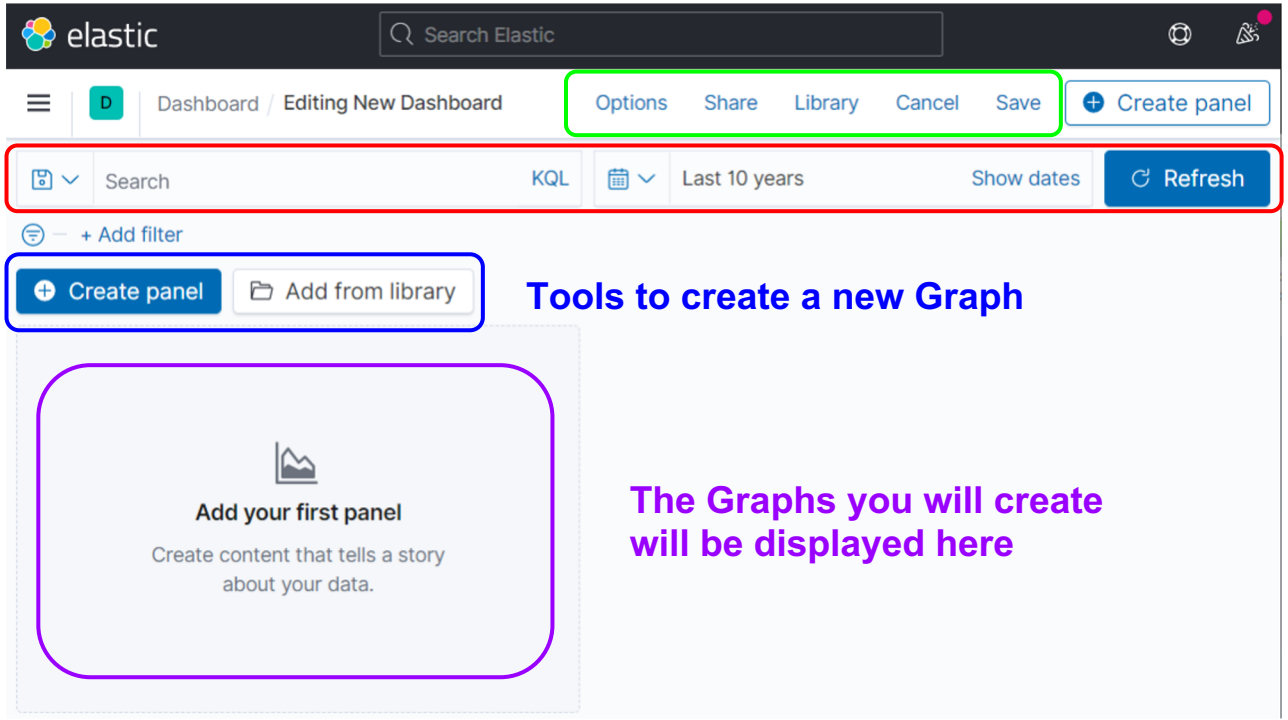
It offers a centralized way to set up your integrations through integrations modules. The integrations ship with dashboards and visualizations, so you can quickly get insights into your data.

Through **Fleet** mode, you can add and manage integrations for popular services and platforms, providing an easy way to collect your data.

The picture on the right exemplifies some of the available modules.



Practice - Create your First Dashboard



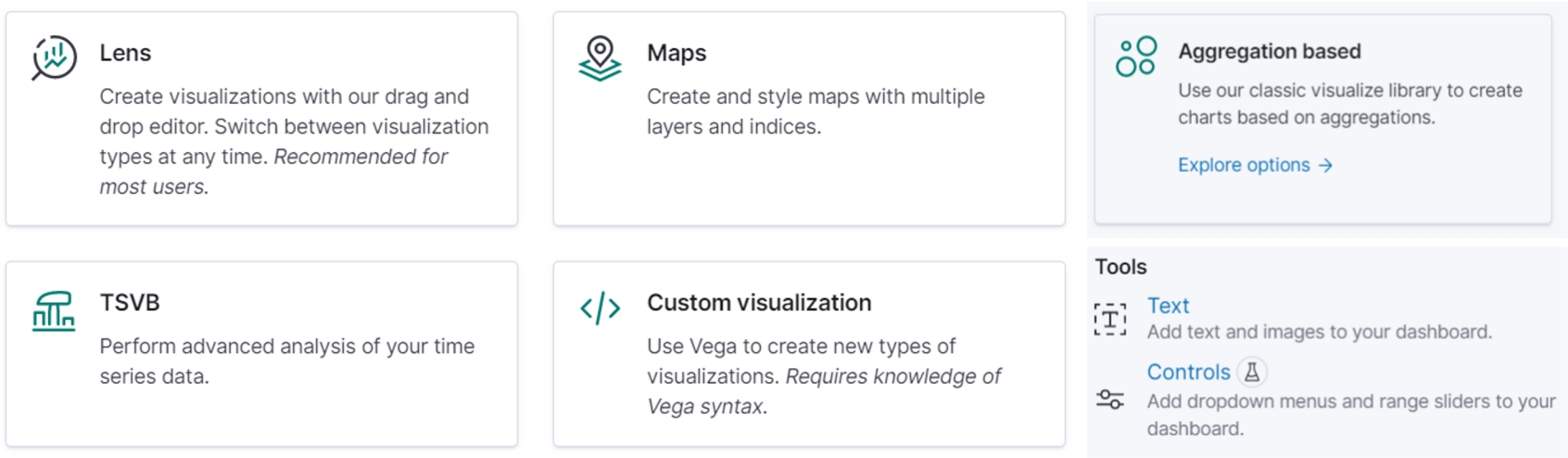
Dashboard Management

Time and Fields Filters

Practice - Create your First Dashboard

Press on “**Create Panel**” in the top-right (or top-left) of the screen.

A box with many different options will be displayed, as shown in the picture below.



Conclusions

Kibana is a very interesting tool to explore, organize and visualize huge amounts of data.

Kibana is easy to learn thanks to its intuitiveness and the tools it provides, but really hard to master. You should have notice it from the complexity it allows to manage while creating all the different panels.

But… Our journey isn't over yet. Now we will provide you with a user study and a dataset to test your skills.

POLITECNICO
MILANO 1863

SYSTEMS AND METHODS FOR BIG AND UNSTRUCTURED DATA

IR Based Databases - ELK

Marco Brambilla

marco.brambilla@polimi.it

@marcobrambi