ELASTIC STACK

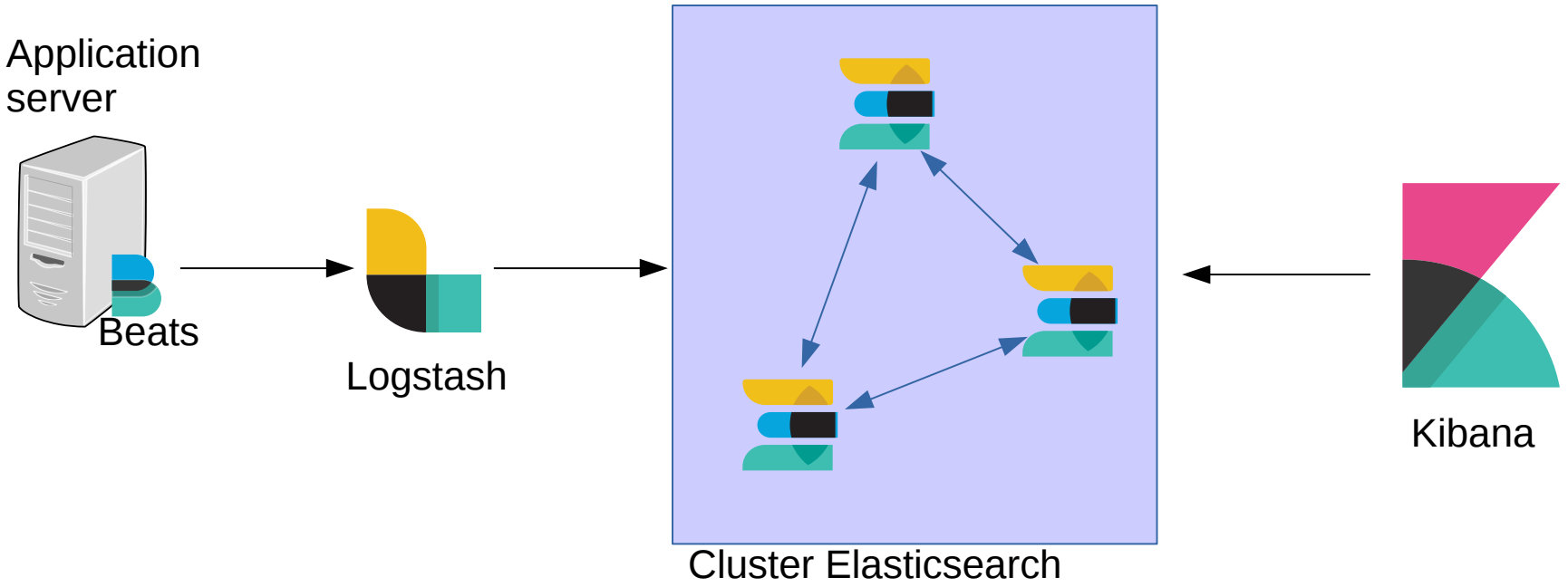
TABLE OF CONTENTS

LOGISTICS

- Schedule
- Lunch & pauses
- Other questions ?

INTRODUCTION

ELASTIC STACK AT A GLANCE



USES CASES

- Business applications
  - Transaction / processus number and duration
  - Operations history
  - Application calls correlations
- Performances / Health monitoring
  - Requests number / durations / response times
  - Errors
  - Systems, databases, applications, ... logs correlation
- Security
  - Fraud / intrusion detection
  - Traceability

TAKING ADVANTAGE OF LOGS

- Used to be fastidious
  - Which server (cluster) produced the log I am interested in ?
  - Where do I find the log files ?
  - Which users has habilitation to access the logs ?
  - Limited set of basic tools `ls`, `tail`, `grep`...
  - File search, with long term synthesis is hard to obtain this way

THE NEW LOGGING WAYS WITH ELASTIC STACK (1/6)

1. Crawl for logs on each server to collect them
2. Push to centralized servers
3. Enrich and structure the logs
4. Index and make logs searchables and agregatables
5. Request and view synthetics reports

THE NEW LOGGING WAYS WITH ELASTIC STACK (2/6)

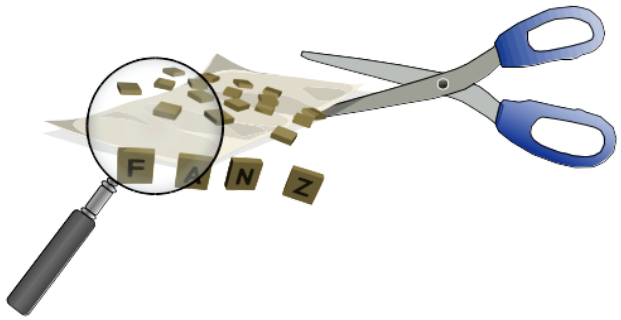
Crawlers and shippers have to ship messages to the ***centralized system***

- Beats lightweight data shippers:
  - Filebeat, Metricbeat,
  - Packetbeat, Winlogbeat,
  - Auditbeat, Heartbeat
- Logstash (used as a shipper)
- Web applications via sockets or brokers
- Apache Nutch
- BASH / Java / / ... Batches

THE NEW LOGGING WAYS WITH ELASTIC STACK (3/6)

Tools have to parse/transform ***centralized logs*** to ***enrich*** and ***structure*** them

- Logstash can parse/transforme logs with its ***One (default)*** or more ***Pipelines*** composed of one to many ***input***, ***filter*** and ***output*** plugins like:
  - File input/output
  - Syslog input/output
  - Brokers input/output
  - Elasticsearch input/output
  - Grok filter
  - Mutate filter
  - Date filter
  - Geoip filter
  - ...



THE NEW LOGGING WAYS WITH ELASTIC STACK (4/6)

Tools have to parse/transform ***centralized logs*** to ***enrich*** and ***structure*** them

- ***Elasticsearch ingest nodes*** can parse/transform logs with their ***Pipelines*** composed of

  one to many ***processors*** like:

  - ***Convert Processor***, ***Date Processor***, ***Date Index Name Processor***, ***Fail Processor***, ***Foreach Processor***, ***Grok Processor***, ...
  - ***Attachment Processor***

    external file attachments extractor plugin based on Apache Tika working with docs, pdfs, ppt, xls, ... file formats
  - ***Geoip Processor***

    external plugin for geoip processing of IPv4 and IPv6 Ip addresses
  - ***User-Agent processor***

    external plugin for user agents processing
  - ...

THE NEW LOGGING WAYS WITH ELASTIC STACK (5/6)

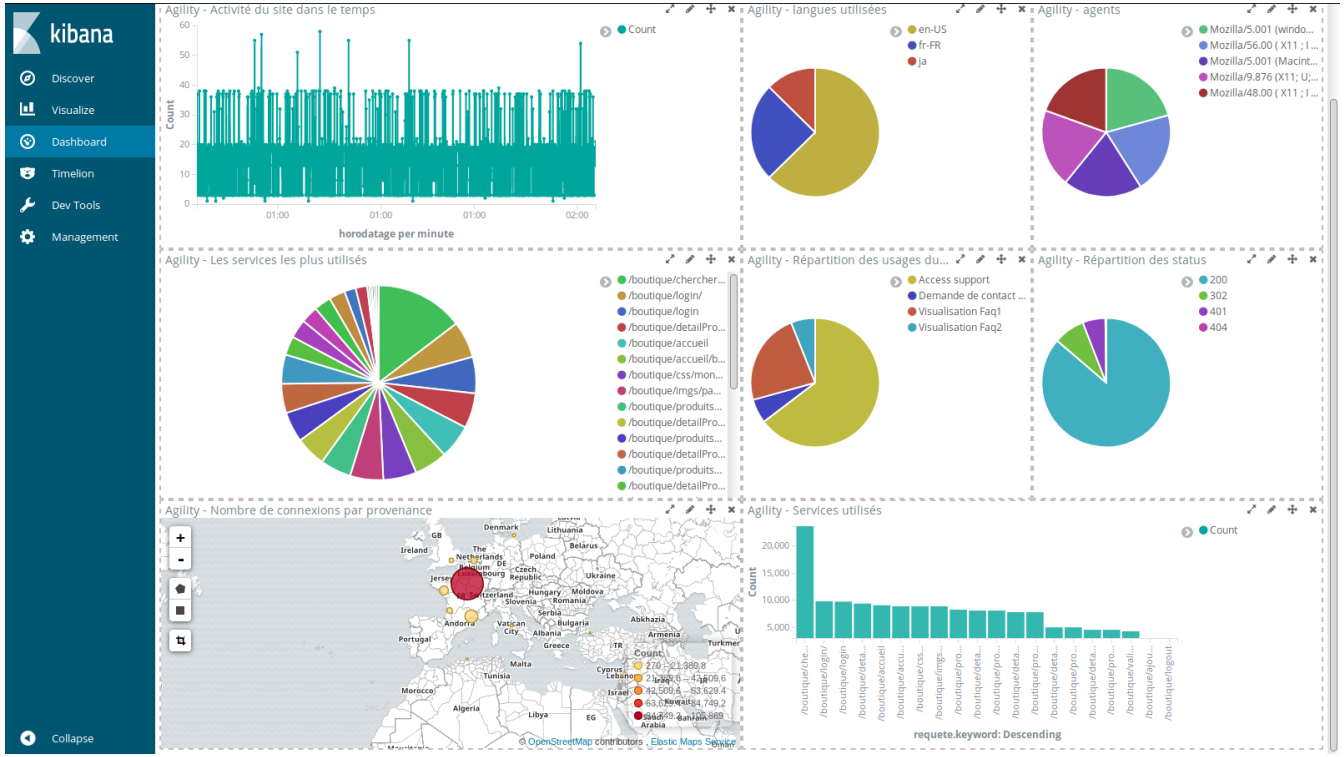
Once ***structured / enriched and centralized***, logs have to be ***indexed*** to be easily made ***searchable*** and ***aggregatable***...

This is the ***Elasticsearch*** search engine job !

```
{
   "type": "accesslog",
   "@timestamp": "1517404596045",
   "verb": "GET",
   "path": "/application/index.html",
   "status": 200,
   "durationMs": 54,
   [...]
}
```

THE NEW LOGGING WAYS WITH ELASTIC STACK (6/6)

Now graphically build ***aggregation requests*** and build your ***own views*** on data and prepare reporting with thematic ***dashboards***



BEATS

Lightweight **data collectors and shippers**

- Read specific data types on the data generators (applications/servers/loadbalancers,...)
- Replace the obsolete logstash shipping method
- Written in **Go** and configured via **Yaml** files

LOGSTASH

A tool which behaves like an **ETL** to transform event data

- Read in → Transform / Filter → Write out
- Historically dedicated to logs
- Written and configured in JRuby

ELASTICSEARCH

A **document oriented nosql database** as well as a powerful seach engine

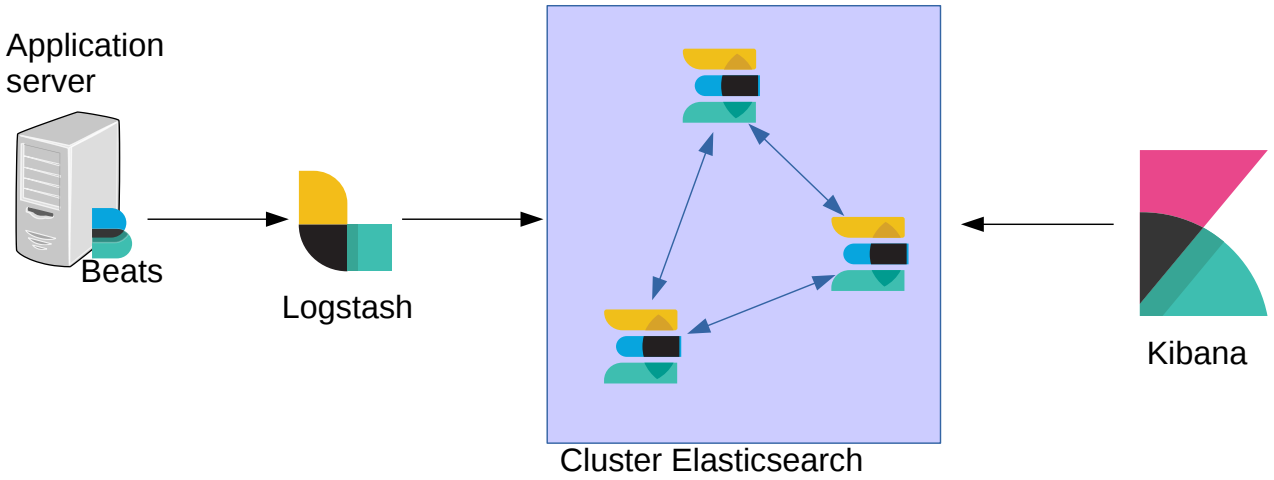
- Based on the well known **Apache Lucene library**
- Based on a distributed architecture enabling:
  - Workload distribution
  - Fault tolerance
- Accessible via APIs
  - JSON/HTTP REST API
  - Java Transport/REST APIs
- Written in Java

KIBANA

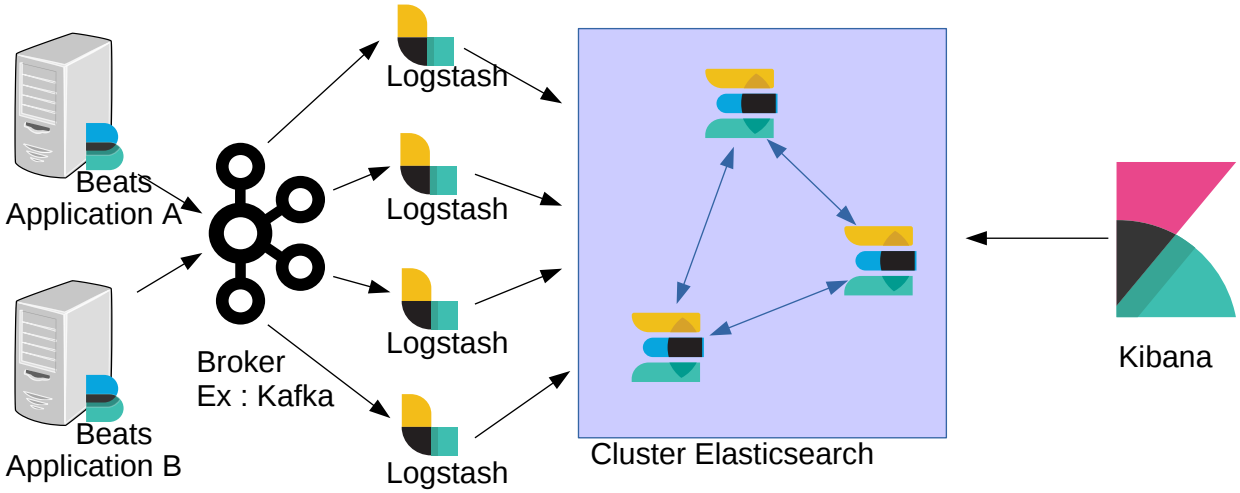
A **web interface** to graphically analyze your requests and build graphical views on the results

- Explore data
- Graphically build Elasticsearch requests
- Produce views and dashboards : maps, graphics, tables...
- Written in JavaScript

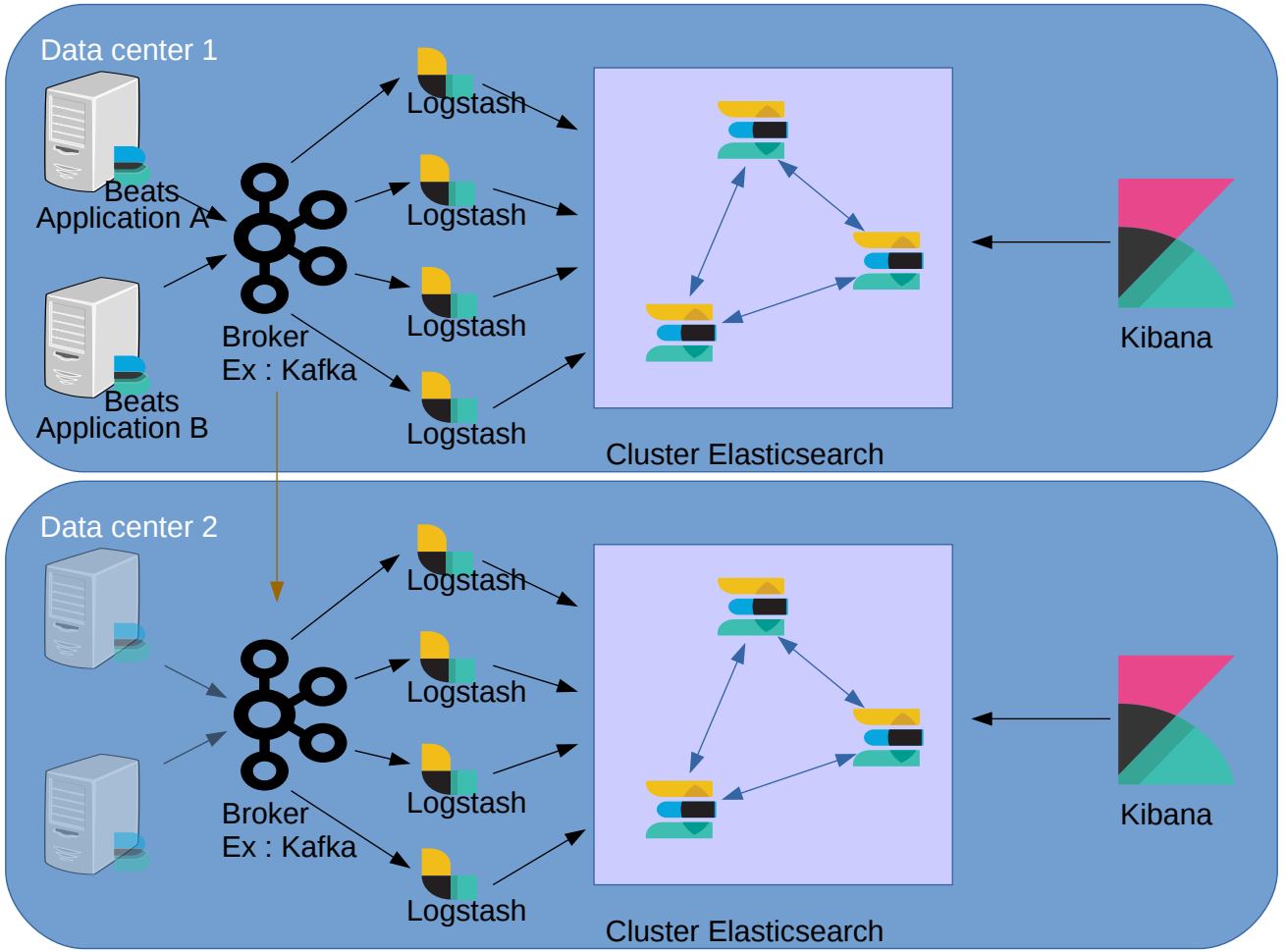
SIMPLE ARCHITECTURE



RESILIENT ARCHITECTURE



ULTRA-RESILIENT ARCHITECTURE



ELASTICSEARCH

REST API - URL CANONICAL FORMAT

Curl URL sample

```
curl -XPOST http://elasticsearch-host:port/index/_doc/id/_action?param=42
-d ''{ "clef": "valeur" }'
```

Kibana development tools URL sample

```
POST index/_doc/id/_action?param=42
{ "clef": "valeur" }
```

- **Http Verbs** : `GET`, `POST`, `PUT`, `DELETE`
- **Port** : 9200 (défault)
- **Index** :
  - A list like `first,second`
  - An alias
  - A wildcard exrpession like `logstash-*`
- **Action** : `_search`, `_mapping`, `_settings`, `_analyze`, `_source`...
- **Body** : the document content depends on the chosen action
- Optional : port, index, id, action, parameters

DEFINITIONS - DOCUMENT, TYPE AND INDEX

- A **Document** is a json instance of what we want to index/search

```
{
  "title": "Spring Batch in Action",
  "auteurs": ["Arnaud Cogoluègnes", "Olivier Bazoud"],
  "price": 20.80,
  "editor": {
    "code":"manning",
    "name": "Manning Publications Co."
  },
  "publication_date":"01/10/2011"
}
```

contains typed **fields** : `string`, `number`, `object`, `array`, `boolean`, ... and their respective values

- A **Document type** is a data structure defining **field types** and **index/search optimization options** also referred as a **mapping**

- An **Index** is a thematic document set storage

DEFINITIONS - MAPPING (1/2)

- Defines an **index** documents **shared structure**
- Only one per index since Elasticsearch v6.0.0
- Is **dynamically** (schemaless by default) or **manually** set (via Put Mapping API)
- For each **document type field**, indicates its
  - **Name**
  - **Type**
  - **Indexation** options
  - **Analysis** options

DEFINITIONS - MAPPING (2/2)

- Impacts
  - Index **physical structure**, **performances** and **size**
  - Query results speaking of **ergonomy** and **relevancy**
- Mainly immutable as types and options have a direct impact on the physical aspect of an index once created:
  - Options requiring new datastructures for an **already known/indexed field** are forbidden
  - New Analyzers can not be applied to **already mapped/indexed** field as it would be impossible to tidy new documents as old ones

> *As a result, mapping definitions are used to change over time and new functionalities in an iterative mode. As a consequence reindexation is a normal process.*

REST API - MAPPING

```
PUT /library/_mapping/_doc
{ "properties": {
    "title": {
      "type": "text",
      "analyzer": "standard",
      "fields": {
        "fr":    {"type": "text", "analyzer": "french"},
        "raw":   {"type": "text", "index": "false"}}},
    "price":{ "type":"double" },
    "publication_date": { "type": "date", "format": "yyyy-MM-dd" },
    "editor": {
      "type":"object",
      "properties": {
        "code": {"type": "keyword"},
        "name":  {"type": "text", "index": false}
      }
    }
  }
}
```

- Simple types: `string`, `integer`, `long`, `float`, `double`, `boolean`, `date`
- Structured types: `arrays`, `object` or `nested`
- Others: `ip`, `geo_point`, `geo_shape`

DEFINITIONS - ANALYZE (1/3)

- **Analyze** process only apply to `text` fields to `tokenize` them

> *"Tokenization is the act of breaking up a sequence of strings into pieces such as words, keywords, phrases, symbols and other elements called tokens.*
>
> *Tokens can be individual words, phrases or even whole sentences.*
>
> *In the process of tokenization, some characters like punctuation marks are discarded. The tokens become the input for another process like parsing and text mining."*

DEFINITIONS - ANALYZE (2/3)

Tant va la cruche à l'eau qu'à la fin elle se brise
→ tant,va,cruch,eau,fin,bris

- `Tokenizes` texts in tokens
- `Transforms or strips` accents, char cases, special chars or html elements
- Drops meaningless linking words (`stopwords`)
- Manages synonyms, conjugation/plural forms of verbs and words (`stemming`)
- Often/always `language specific`
- Used by default at the indexing `AND` search time (depends on mapping options)
- Can not be changed after documents have been indexed
- Is applied to one or more fields depending of mapping options

DEFINITIONS - ANALYZE (3/3)

Elasticsearch makes it possible to treat **one source document field** with **different analyzers** to enable different kinds of token indexation and search contexts.

For this, you have to map a field with the **mapping multi-fields** syntax:

```
PUT /library/_mapping/_doc
{ "properties": {
    "title": { // Title is a multi-field
      "type": "text",
      "analyzer": "standard",
      "fields": { // title, title.fr and title.raw are analyzed differently
        "fr":    {"type": "text", "analyzer": "french"},
        "raw":   {"type": "text", "index": "false"}}},
    "price":{ "type":"double" },
    "publication_date": { "type": "date", "format": "yyyy-MM-dd" },
    "editor": {
      "type":"object",
      "properties": {
        "code": {"type": "keyword"},
        "name":  {"type": "text", "index": false}
      }
    }
  }
}
```

ANALYZER, TOKENIZER AND TOKEN FILTERS...

- **Analyzer**
  - `standard`, `french`, `english`, `whitespace`, `keyword`, `custom`...
  - Processing order: N Char filters → 1 Tokenizer → N Token filters
- **Char filter** : Transform **text** at a **char level**
  - html_strip, mapping, ...
- **Tokenizer** : Do the **tokenization** work to generate **tokens** (**mandatory**)
  - standard, whitespace, letters, pattern, keyword, ...
- **Token filter** : Add / modify or delete at a **token level**
  - standard, asciifolding, lowercase, stop, synonym, stemmer, ngram, ...

SAMPLE - USING A CUSTOM ANALYZER

```
PUT library
{ "settings": {
    "analysis": {
      "filter": {
        "ngram_filter": {
          "type" : "edge_ngram",
          "min_gram" : "2",
          "max_gram" : "10" }},
      "analyzer": {
         "ngram_analyzer": {
          "type": "custom",
          "tokenizer": "standard",
          "filter": ["standard", "lowercase", "ngram_filter"] }}}},
  "mappings": {
    "properties": {
        "title": {
          "type": "text",
          "analyzer": "ngram_analyzer" }
      }
  }
}
```

REST API - TESTING ANALYZERS (1/2)

```
//Testing an existing analyzer
POST _analyze
{
  "analyzer": "whitespace",
  "text":     "The quick brown fox."
}

//Testing a custom analyzer
POST _analyze
{
  "tokenizer": "standard",
  "filter":  [ "lowercase", "asciifolding" ],
  "text":     "The quick brown fox."
}

//Testing a settings and mapping defined analyzer
POST library/_analyze
{
  "field": "title", //Use the title field analyzer
  "text":     "The quick brown fox."
}
```

REST API - TESTING ANALYZERS (2/2)

```
//Whitespace analyzer test response
{ "tokens": [
    { "token": "The",
      "start_offset": 0,
      "end_offset": 3,
      "type": "word",
      "position": 0 },
    { "token": "quick",
      "start_offset": 4,
      "end_offset": 9,
      "type": "word",
      "position": 1 },
    { "token": "brown",
      "start_offset": 10,
      "end_offset": 15,
      "type": "word",
      "position": 2 },
    { "token": "fox.",
      "start_offset": 16,
      "end_offset": 20,
      "type": "word",
      "position": 3 }
  ]}
```

REST API - TWO INDEXATION MODES

```
 //Index a document with a given id, a business one, to enable CRUD uniquely
//identified document management
//CRUD stands for Create/Update or Delete operations. Ex: Invoices management

PUT /library/_doc/1
{"title":"Spring Batch in Action"}

//Index a document with a self generated id when manipulating documents as a
//whole set only. i.e. using full text search only
//Logs are good candidates for this kind of indexation

POST /library/_doc
{"title":"Elasticsearch in Action"}


//When the id is known, we have direct access to it for CRUD operations

//Getting the unique document
GET /library/_doc/1

//Deleting a unique document
DELETE /library/_doc/1
```

REST API - BULK INDEXATION

- Bulk indexation can be used to optimize indexation performances:
  - It has more chance to fill the buffer before refresh interval event occurs
  - It uses less http/tcp connections to carry the same payload

```
//Bulk indexation is a suit of json command objects followed by its json payload
//Results status are given in the same order as the bulk request is forged
//The document transaction level implies some indexations can fail while others do not

POST /_bulk
{ "index": {"_index":"library", "_id":"1"}} //Command
{ "title": "title": "Spring Batch in Action", ...}          //Payload
{ "index": {"_index":"library", "_id":"2"}} //Command
{ "title": "title": "Elasticsearch in Action", ...}         //Payload
```

LUCENE SEGMENTS AND MERGES

- A Lucene segment is an ***immutable inverted vector index***

Term	Docs
spring	1
batch	1
action	1,2
java	2

- As the number of segments increases, ***consumes*** more and more ***resources*** (file pointers, caches, memory...)
- To overcome this problem, Elasticsearch regularly ***merges*** the segments
- These merge operations comply with a ***merge policy*** optionally chosen at the creation of the index (this an ***expert*** setting)
- A ***merge*** consists of:
  - Deleting several segments
  - Deleting the associated caches
  - Building a bigger ***immutable*** segment to contain the previously deleted segments without keeping the documents ***marked*** as destroyed

REFRESH

- **Near Real Time**
  - There is a small ***delay*** between the moment the document is ***indexed*** and the moment it is ***available*** through search
  - Default is ***1 second***
  - Configurable for each index with the ***refresh_interval*** setting

- Before a bulk indexing (***_bulk***), it is recommended to increase the ***refresh_interval*** in order to prevent too many index refreshes

REFRESH AND LUCENE SEGMENTS

- Before being sent to Lucene, indexed documents are ***analyzed*** and ***stored*** in an in-memory buffer so these documents ***cannot*** be ***searched*** instantly
- This buffer comes with a ***Translog*** file which is kept ***12 hours***(default) with a maximum size of ***512 mb***(default). Used to persist commands in the buffer and optimze chances to get quicker replica's recoveries.
- The buffer is ***flushed*** to a Lucene segment when:
  - the buffer is ***full***
  - during a Lucene ***reopen***
  - on an Elasticsearch ***refresh***:
    - ***without*** Lucene commit,
    - triggered according to the ***index.refresh_interval*** setting value or though the API ***_refresh***
  - on an Elasticsearch ***flush***:
    - triggers a Lucene ***commit***,
    - ***triggered*** automatically by Elasticsearch or though the API ***_flush***

REST API - SEARCH REQUEST SAMPLES

```
 # Search with the Lucene Syntax
GET /library/_doc/_search?q=batch

# Search with the JSON Syntax
POST /library/_doc/_search
{
  "query":{
    "query_string":{
      "query":"batch"
    }
  }
}
```

REST API - SEARCH RESPONSE SAMPLE


```
{
  "took": 2,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": 177095,
    "max_score": 0.076713204,
    "hits": [
      {
        "_index": "library", "_type": "_doc", "_id": "1",
        "_score": 0.076713204,
        "_source": {"title": "Spring Batch in Action"}
      }
    ], ...
  } ...
}
```


REST API - LUCENE SEARCH SYNTAX (1/3)

```
GET /library/_doc/_search?q=batch&size=100&sort=title.keyword:asc
```

q stands for **query string** query and accepts the Lucene syntax:

```
batch          //search on all elligible fields by default
title:batch    //search on the title field
```

- By default, Elasticsearch search upon all **term query eligible fields** as **index.query.default_field** index setting has '*' as its default value
- **index.query.default_field** can be changed to match a more specific field if needed
- **Query string** query is a **full text search**, so its terms are **analyzed** as stated by the mapping for **indexation and/or search** operations

REST API - LUCENE SEARCH SYNTAX (2/3)

- Logical operators

```
 Erreur Error
message:Erreur OR level:Error
message:Erreur || level:Error

+Erreur +Error
message:Erreur AND level:Error
message:Erreur && level:Error

-Trace
NOT message:Trace
```

- Grouping syntax

```
 +(message:Erreur level:Error) +user:jdoe
(message:Erreur OR level:Error) AND user:jdoe
level:(Erreur Error)
```

REST API - LUCENE SEARCH SYNTAX (3/3)

- Searching phrases

```
message:"Erreur Grave"
message:"Erreur Grave"~2
```

- Searching with **Fuzziness**

```
message:Ereur~1
```

- Searching on **intervals**

```
duration:[100 TO 500]
price:[* TO 50]
date:[2015-05-01 TO 2015-06-01} //} stands for mathematical interval sign '['
```

- Searching using **Wildcards/Regexp** (not advised for performance reasons)

```
message:"Err* Text?"
message:/[Ee]rr(eu|o)r/
```

REST API - JSON SEARCH SYNTAX

```
POST /library/_doc/_search
{
  //By default full text search use the same analyzer for index and search time
  "query": { "match": {"title": "action"}},

  //By default, sorts upon _score
  "sort": [{"title": "asc"},"_score"],

  //By default, page size is 10
  "from": 0, "size": 100,

  "aggregations": {"stats_prix": {"stats": {"field": "price"}}},
  "highlight": {"fields": {"title": {}}}
}
```

REST API - SEARCH RESPONSE

```
{
  "took": 360, "timed_out": false,
  "_shards": {"total": 5, "successful": 5, "failed" 0},
  "hits": {
    "total": 2, "max_score": null,
    "hits": [ {
      "_index": "library", "_type": "_doc", "_id": "1",
      "_score": 0.15342641,
      "_source": {
        "title": "Spring Batch in Action", ...
      },
      "highlight": {
        "title": [ "Spring Batch in <em>Action</em>" ]
      }
    }, ... ]
  },
  "aggregations": {
    "stats_prix": {
      "count": 2, "min": 20.8, "max": 31.6, "avg": 26.2, "sum": 52.4
    }
  }
}
```

REST API - JSON SYNTAX SEARCH QUERIES (1/3)

Analyzed search queries:

`match`, `match_phrase` and `multi_match` are full text search

```
{ "query": {
    "match": {
      "title": "spring"
    }
  }
}
```

`query_string` is a full text search with a Lucene syntax

```
{ "query": {
    "query_string": {
      "query": "spring AND batch"
    }
  }
}
```

`match_all` is equivalent to **no search** query and returns all index documents (of course paginated)

REST API - JSON SYNTAX SEARCH QUERIES (2/3)

Not Analyzed search queries:

`term`, `terms` are exact match queries

```
{ "query": {
    "term": {
      "editor.code": "manning"
    }
  }
}
```

`prefix`, `wildcard`, `regexp` are substring matching queries

`fuzzy` is a term proximity search query (based on Levenstein distance)

`range` is an interval search query used for numbers, dates, texts or keywords

```
{ "query": {
   "range": {
    "price": {
      "gt": 10,
      "lte": 30 }
    }
  }
}
```

REST API - JSON SYNTAX SEARCH QUERIES (3/3)

Bool search query aggregates **sub queries** and plays a great role on **scoring** and **caching** options

```
{
"query": {
  "bool": {
    "must": [
    //Must appear in docs, score contribution, implicit 'and' operators
      { "match": { "title": "spring" }},
      { "match": { "authors": "cogoluegnes" }}
    ],
    "filter": {
    //Must appear in docs, no score contribution, implicit 'and' operators
      "range": { "price": { "lte": 30 } }
    },
    "should": [
    //Shoul appear in docs, each match increases score, implicit 'or' operators
      { "match": { "editor": "eni" }}
    ],
    "must_not": {
    //Must not appear in docs, no score contribution, implicit 'or' operators
    //Consider ordering the most restrictive clause first.
      "match": { "editor": "manning" }
    }
  }
}}
```

REST API - SEARCH FILTER AND QUERY CONTEXTS

- In a **Query** context
  - Prefer using full text search queries → analyzed text
  - Impacts scoring

- In a **Filter** context like **filter**, **bool:must_not**, **bool:filter** or **constant_score**
  - Prefer using exact match search queries
  - As it **reduces query perimeters** → Increase performances
  - As it is **cacheable** → Increase performances
  - As there is **no score computation** → Increase performances

DEFINITIONS - AGGREGATIONS USAGES

- **Refining** results lists
  - Show pre-filtering options
  - Grouping results per type / tab
- **Analyze** data
  - Computes counts, sum, stats... on buckets of documents
  - Enables showing adequat diagrams or views on these computations results

> *NB: Aggregations apply to `query` results for all matching documents* **without pagination limits**

DEFINITIONS - AGGREGATION TYPES (1/2)

- **Bucketing** aggregations
  - `terms`, `histogram`, `date_histogram`, `range`, `filter`...
  - Can contain sub-aggregations like bucketing or metric ones
  - Classifies documents in groups/sub-groups of elements called **buckets** (i.e : By prices, By terms, By days ...)
  - Contains a default count metric aggregation to count buckets elements

- **Metric** aggregations
  - `min`, `max`, `stats`, `extended_stats`...
  - Computes the chosen metric in the containing bucket
  - No containing bucket → computes on all index's documents
  - Apply on numbers, dates, documents in a bucket...

DEFINITIONS - AGGREGATION TYPES (2/2)

- **Pipeline** aggregations
  - `min_bucket`, `max_bucket`, `stats_bucket`, `derivative`...
  - Computes statistics based on buckets of other computed bucketing aggregation
  - Works at a bucket level instead of documents one

DEFINITIONS - AGGREGATIONS SAMPLES

Titre	Format	Prix
Spring Batch in Action	Paper	47.17
Spring in Action	PDF	48.49
Camel in Action	eBook	47.00
Hibernate in Action	PDF	40.56
Lucene in Action	eBook	41.72
Elasticsearch in Action	eBook	43.50

Format	Nb	Prix min	Prix max	Prix moyen
Paper	1	47.17	47.17	47.17
PDF	2	40.56	48.49	44.52
eBook	3	41.72	47.00	44.07

REST API - AGGREGATIONS REQUEST SAMPLE

```
{
  //Query section
  "query": { "match": {"title": "action"}},
  //Aggregations section
  "aggregations": {
    "titre_terms": {
      "terms": {
        "field": "title",
        "size": 5 }},
    "prix_histo": {
      "histogram": {
        "field": "price",
        "interval": 2 }},
    "prix_avg": {
      "avg": {
        "field": "price"
      }
    }
  }
}
```

REST API - AGGREGATIONS RESPONSE SAMPLE

```
{
  //Query result section
  "hits": { ... }
  //Aggregations result section
  "aggregations":{
    "titre_terms": {
      "buckets": [
        { "key": "action", "doc_count": 6 },
        { "key": "spring", "doc_count": 2 },
        { "key": "batch",  "doc_count": 1 },...]},
    "prix_histo": {
      "buckets": [
        { "key": 40, "doc_count": 2 },
        { "key": 42, "doc_count": 1 },
        { "key": 44, "doc_count": 0 }, ...]},
    "prix_avg": {
      "value": 44.74
    }
  }
}
```

REST API - AGGREGATIONS (1/2)

- `terms` classifies by terms

```
{ "aggregations": {
    "title_terms": {
      "terms": {
        "field": "title",
        "size": 10}}}}
```

- `range`, `date_range` classify by range of values

→ Ranges coming from the expressed request

```
{ "aggregations": {
    "price_range": {
      "range": {
        "field": "price",
        "ranges": [{ "to": 20},{"from": 20,"to": 40},{"from": 40}]}}}}
```

- `histogram`, `date_histogram` classify by range of values

→ Ranges coming from real docs values

```
{ "aggregations": {
    "title_terms": {
      "histogram": {
        "field": "price",
        "interval": 5 }}}}
```

REST API - AGGREGATIONS (2/2)

- `min`, `max`, `avg`, `stats`, `extended_stats` compute metrics on the containing bucket

```
{ "aggregations": {
    "price_stats": {
      "stats": {
        "field": "price"}}}}
```

- `filter` defines a single bucket for matching documents

```
{ "aggregations": {
    "term_filter": {
      "filter": {
        "term": {
          "editor.code": "manning" }}}}}
```

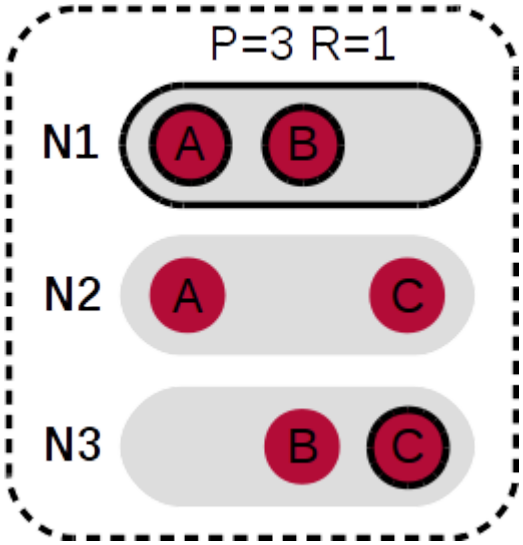
REST-API - NESTED AGGREGATIONS SAMPLE

```
POST library/_search
{
  "aggregations": {
    "byEditor": {
      "terms": {
        "field": "editor.code"
      },
      "aggregations": {
        "avg_price": {
          "avg": {
            "field": "price" }
        },
        "byAuthor": {
          "terms": {
            "field": "auteurs" }
          }
        }
      }
    }
  }
}
```

DEFINITIONS - CLUSTER, NODE, SHARD, REPLICA

- **Cluster** is a set of Elasticsearch nodes working together
- **Node**
  - An Elasticsearch process runing on a machine
  - A master node per cluster
- **Index**
  - Known from all the cluster's nodes
  - Partitioned in P shards :

    → Indexed datas can be spread on 1 to P nodes
- **Shard** is an index portion
  - A document stands only in one shard
  - A node can manage 1 to P shards of an index
  - A shard can be replicated R times
- **Replica** is physically the same as a Shard, but is considered as a copy
  - Enables fault tolerence
  - Increase performances by distributing search request on the cluster



REST API - SHARDING AND REPLICATING

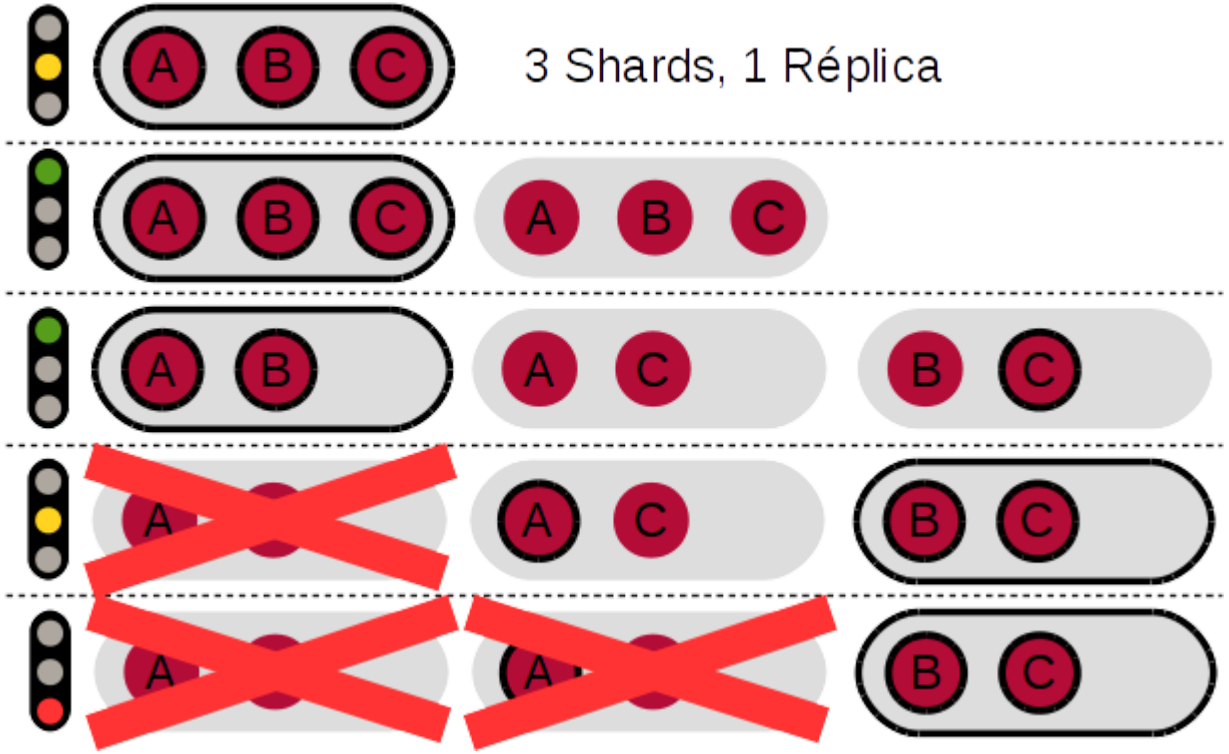
- Index level configuration

```
PUT /library
{ "settings": {
    "index": {
      "number_of_replicas":"1",
      "number_of_shards":"5" }}}
```

- Once an index is created, only replication factor can be changed

```
PUT /library/_settings
{ "index": {
    "number_of_replicas":"2" }}}
```

HIGH AVAILIBILITY



REST API - ALIASES

- **Aliases**
  - Can **route** to the current index (See Blue/Green deployment doctrine)
  - Can group indexes used at the same time (~ SQL Union)

```
PUT /library/_alias/lib

POST /_aliases
{ "actions" : [
    { "add" : {
        "index" : "library",
        "alias" : "lib"
      }
    }
  ]
}
```

REST API - INDEX TEMPLATE

```
 //With this command, the index template will be used on each index creation when it's name
matches the 'logstash-*' index patterns

PUT /_template/logstash
{
  "order": 0,
  "index_patterns": "logstash-*",
  "settings": {
    "index": {
      "number_of_replicas": 1,
      "number_of_shards": 3 }},
  "aliases": {
    "logstash": {}},
  "mappings": {
    "properties": {
        ...
    }
  }
}
```

DATE MATH INDEX NAME RESOLUTION (1/2)

Sometimes you will have to request indices without knowing their real names as they are computed at the index creation time.

→ Without alias, it is difficult to script index actions that would be valid over time passing without changing hard coded references

- Consider using Elasticsearch Date math index name resolution
  - Expressed this way: **<static_name{date_math_expr{date_format|time_zone}}>**
  - **date_math_expr** is a dynamic date math expression that computes the date dynamically. See below
  - **/** symbol has to be encoded with `%2F` to match URL encoding rule
  - **date_format** is the computed date rendering format and defaults to `YYYY.MM.dd`

DATE MATH INDEX NAME RESOLUTION (2/2)

```
//Search for 2 days old logs
GET /<logstash-{now%2Fd-2d}>/_search
```

See https://www.elastic.co/guide/en/elasticsearch/reference/6.2/date-math-index-names.html for more details

Samples matrix

Date math expression	Result	Sample
logstash-{now/d}	current day	logstash-2024.03.22
logstash-{now/d - 2d}	2 days ago	logstash-2024.03.20
logstash-{now/M}	begining of the current month	logstash-2024.03.01
logstash-{now/M{YYYY.MM}}	current month	logstash-2024.03

BACKUP & PURGE

- **Backup** is always incremental
  - Repository: an accessible file system like NAS, NFS, DFS... for all nodes
  - Snapshot command archives of one or more indices in a repository
  - Restore command restores one or more indicies in a repository
- **Purge** is a manual operation consisting of deleting old/unused indices
- Companion tools:
  - Schedulers like Cron...
  - REST API or Elasticsearch Curator

INSTALLING AND RUNNING

- Pre-requisites : Oracle Java 8
- Installers : .zip, tar.gz., .deb, .rpm packages
- Configurations : `elasticsearch.yml`, `jvm.options` and `log4j2.properties`
- Start : 1 Elasticsearch process with
  - sudo systemctl start elasticsearch
  - bin/elasticsearch
- Connect to 9200 HTTP port or 9300 Transport TCP port depending on your client

CONFIGURATION

- Set `-Xmsset -Xmx` to 50% of physical RAM in the `jvm.options` file

→ To have efficient nodes, we recommend to never use a RAM bigger than 64Gb as we reserve only half of it to:

```
- take advantage of OOPs optimizations under a 32Gb JVM
- make gc shorter (scanning more than 32Gb can take a long time)
```

- Set the most important things in `elasticsearch.yml`

```
 # Clustering
cluster.name: my-cluster
node.name: node1

discovery.seed_hosts: ["node1-host:9200", "node2-host"]
cluster.initial_master_nodes: ["node1-host:9200", "node2-host"]

# Réseau
network.host: 192.168.0.1 //The public ip/host address of the node
```

PLUGINS

- Usages ?
  - Discovery : AWS, Azure, GCE
  - Analyze : ICU, Tika, Asiatic languages
  - Scripting : JavaScript, Python
  - Security : Stack Features (X-Pack), Search Guard
  - Alerting : Stack Features (X-Pack)

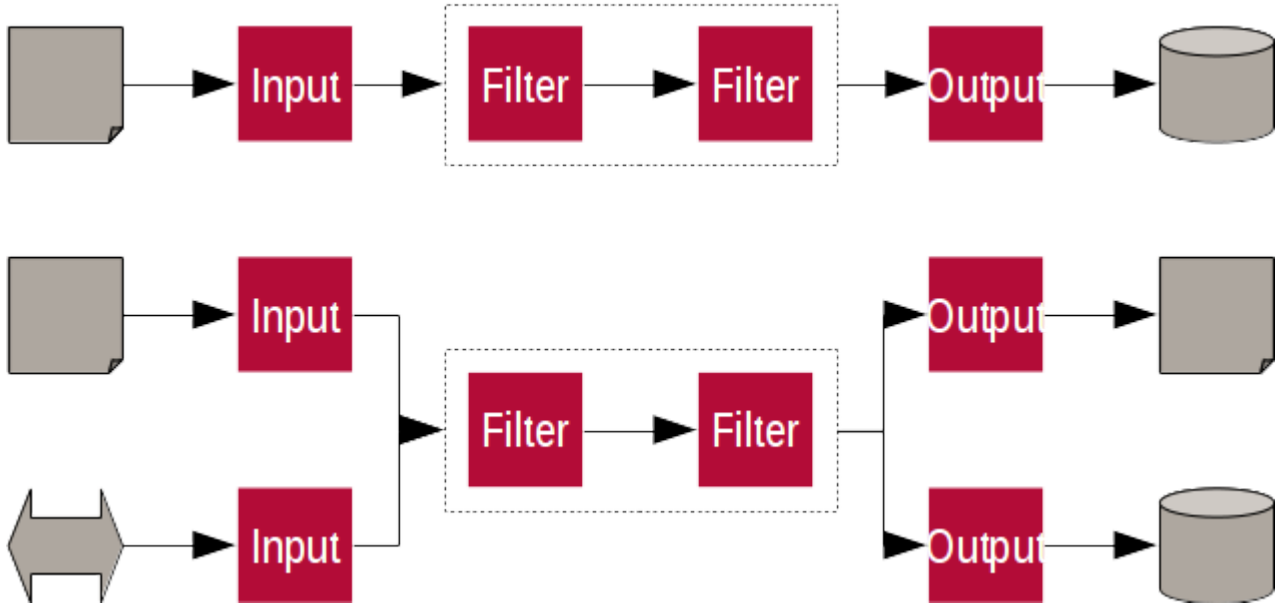
```
 # Plugin installation sample
bin/elasticsearch-plugin install mapper-size
```

- Plugins list: https://www.elastic.co/guide/en/elasticsearch/plugins/current/index.html

Lab 1

LOGSTASH

DEFINITIONS - EVENT PROCESSING PIPELINE



- An **Input**: generates/gets events
- A **Filter**: Modifies events to enrich their data or datatypes informations
- An **Output**: Ships events to a recipient
- **Inputs/Outputs Codecs**: decode/encode data in a native way for **line**, **multiline** or **json** events

DEFINITIONS - EVERYTHING IS EVENT

- An **Event** is a timestamped datastructure aimed at containing meaningful fields split/added/converted while transiting in its event processing pipeline
- All input events start with three fields
  - **@timestamp**: is a mandatory input timestamp
  - **@version**: is a version indicator
  - **message**: the entire data received by the pipeline Input
- An **Event** can be manipulated through the Event API:
  - During the **Input** stage
  - During **Filters application** stage
  - During **Output** stage

```
{
  "@timestamp": "2015-05-13T20:42:33.333Z",
  "@version": "1",
  "message": "Hello Logstash!",
}
```

LOGSTASH RUBY CONFIGURATION SAMPLE

```
input {
  stdin {
  }
}
filter {
  grok {
    match => { "message" => "%{TIME:time} %{GREEDYDATA:message}" }
    overwrite => [ "message" ]
  }
}
output {
  file {
    codec => json {}
    path => "output.json.log"
  }
}
```

LOGSTASH RUBY CONFIGURATION SYNTAX

```
The configuration of a plugin consists of the plugin name followed by a block of settings for
that plugin:
input {
  file {
    path => "/var/log/apache/access.log"
    type => "apache"
  }
}
```

A plugin can require that the value for a setting be a certain type:

Type	Sample
string (password/path/uri)	my_uri => 'http://user:pass@example.net' **or** my_password => "password"
boolean	ssl_enable => true
number	port => 33
bytes (string field that represents a valid unit of bytes)	my_bytes => "1113" # 1113 bytes **or** my_bytes => "10MiB" # 10485760 bytes
codec	codec => "json"
list	path => ["/var/log/messages", "/var/log/*.log"]
hash (a collection of key value pairs separated by spaces or newlines)	match => { "field1" => "value1" "field2" => "value2" }

LOGSTASH RUBY CONFIGURATION SYNTAX

- To use **escape sequences** in quoted strings, you will need to set **config.support_escapes** to **true** in your logstash.yml

Text	Result
\r	carriage return (ASCII 13)
\n	new line (ASCII 10)
\t	tab (ASCII 9)
\	backslash (ASCII 92)
\"	double quote (ASCII 34)
\'	single quote (ASCII 39)

- Lines starting with **#** are comments

EVENT MANIPULATION SYNTAX

- Once **input stage section** has created an **event**, it is possible to access its fields in **filters or output stage sections**:

Acces type	Sample
Read event top field	fieldname or [fieldname]
Read event nested field	[top-fieldname][nested-fieldname]
Read event metadata field	[@metadata][fieldname]
Setting a value to an event top field	add_field => { fieldname => "Value" }
Setting a value to an event nested field	add_field => { "[top-fieldname][nested-fieldname]" => "Value" }
Setting a value to event metadata field	add_field => { "[@metadata][fieldname]" => "Value" }

Metadatas are special fields which will not be printed with the rest of the event by outputs but they can help parameterizing some ouput plugins or conditional behaviors

SPRINTF FORMAT AND EVENT FIELDS

```
output {
  statsd {
    increment => "apache.%{[response][status]}"
  }
}
```

- The field reference format is also used in what Logstash calls sprintf format. This format enables you to refer to field values from within other strings.

> *For example, the statsd output has an increment setting that enables you to keep a count of apache logs by status code*

SPRINTF FORMAT TO ACCESS @TIMESTAMP VALUE

```
output {
  file {
    path => "/var/log/%{type}.%{+yyyy.MM.dd.HH}"
  }
}
```

- you can convert the timestamp in the @timestamp field into a string

> *Instead of specifying a field name inside the curly braces, use the +FORMAT syntax where FORMAT is a time format.*

ENVIRONMENT VARIABLES AND CONFIGURATION

```
input {
  file {
    path => "${APACHE_LOGS_DIR:/var/log/httpd}/**/access*.log"
  }
}
```

- It is possible to use **environment variable values** with the following format: **${ENV_VAR:default_value}**

- **Please note that**:
  - Replacements are case-sensitive
  - Replacements are made on Logstash startup and will be therefore immutable while running
  - References to undefined variables raise a Logstash configuration error unless default values have been specified

OVERVIEW - INPUTS AND OUTPUTS

- File, Stdin / Stdout
- **Logs**: Syslog, Gelf
- **Brokers**: RabbitMQ, Redis, Kafka, ZeroMQ, SQS
- **Metric**: Ganglia, Graphite, Nagios, StatsD, CollectD, JMX
- **Messaging**: Mail, XMPP
- **Network**: TCP, UDP

OVERVIEW - INPUT COMMON SETTINGS

All inputs have the following **settings**:

- **add_field**, adds a field to an event
- **codec** (line) specifies the decoding codec
- **enable_metric** (true) disables or enables metric logging
- **id** an id for this Input configuration to help monitoring Logstash activities
- **tags** adds any number of arbitrary tags to your event.
- **type** adds a type field to all events handled by this input

OVERVIEW - OUTPUT COMMON SETTINGS

All outputs have the following **settings**:

- **codec** (line) specifies the encoding codec
- **enable_metric** (true) disables or enables metric logging
- **id** an id for this Output configuration to help monitoring Logstash activities

OVERVIEW - CODECS

- **Codec**: encode (Output) or decode(Input) Events content
  - **line** (default Logstash codec)
  - **multiline**
  - **json**
  - **rubydebug**:
    - encode events to a pretty and colorized json debug output
    - can print metadatas field with **metadata => true** codec option

- All **codecs** can specify their own charset

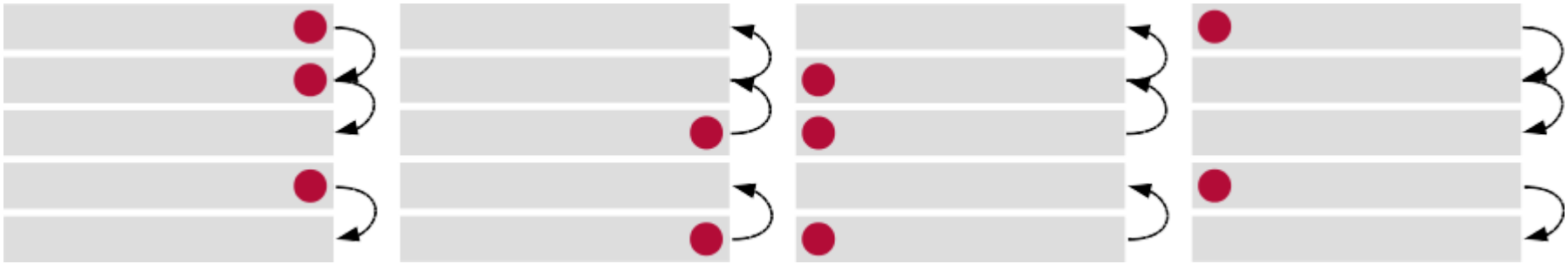
```
input {
  file {
    path => "file.log"
    codec => line {
      charset => "UTF-8"
    }
  }
}
```

OVERVIEW - MULTILINE SPECIFICITIES

- Multiline is a specific codec where you have to **express a rule** to indicate to Logstash what **line styles** mark the **limits** of a multiline event

```
input {
  # For instance, writing the following configuration means each line starting
  # with a space character should merge with the previous one:

  stdin {
    codec => multiline {
      pattern => "^\s"
      negate => false
      what => "previous"
    }
  }
}
```



- Please take advantages of `(?m)`, `$` and `^` in regular expressions

OVERVIEW - FILTERS

- **Filters**: Modify Events
  - GROK (On the shelf regular expressions splitting events message in named fields)
  - Date, UUID, URL, Syslog Priority
  - JSON, CSV, XML, Key / Value
  - Transformation : Mutate, Ruby (scripting)
  - Drop, Split, Clone
  - Consolidation : DNS, GeoIP, User agent, Elasticsearch, Environment, Translate

OVERVIEW - FILTER COMMON SETTINGS

All Filters have the following **settings**:

- **add_field**, adds a field to an event
- **add_tag**, adds a tag to the **tags field** of the event
- **enable_metric** (true) disables or enables metric logging
- **id** an id for this Input configuration to help monitoring Logstash activities
- **periodic_flush** (false), calls the filter flush method at regular interval
- **remove_field**, on filtering success, removes a field from an event
- **remove_tag**, on filtering success, removes a tag from the **tags field** of the event

PIPELINE PROCESSING DEFINITION

- An Event processing pipeline is defined by:
  - A list of **Inputs used to collect** events
  - Some **Filters** used to process events
  - Some **Outputs** used to ship out events
- An Event processing pipeline can define **conditional processing paths** depending on **processed events values** or **processing states** like parse failures
- An Event processing pipeline can have technical settings like:
  - **queue** types and options (seen further)
  - number of **workers** affected to it
  - batch size and delay options
  - stopping security options
  - paths to data/plugins or config files
  - logging and debuging options

PIPELINE CONDITIONAL PROCESSING CONFIGURATION

- Conditional expressions can be used in pipeline configuration:
  - Equality or comparison operators : `==, !=, <, >, <=, >=`
  - Regular expression operators : `=~, !~`
  - Inclusion operators: `in, not in`
  - Logical operators : `and, or, !`

```
filter {
  if [type] == "access_log" {
    grok {
      # Access Log Grok Pattern
    }
    if [status] == 200 or "_grokparsefailure" in [tags] {
      drop { }
    }
  }
}
```

PIPELINE TECHNICAL IMPORTANT SETTINGS (1/5)

- **pipeline.workers** is the number of workers that will, in parallel, execute the filter and output stages of the pipeline

  Defaults to the number of the host's CPU cores

  Can be set to a greater value while outputs make a lot of I/Os and the CPU can enter idle phase. Think to test performances for tuning !

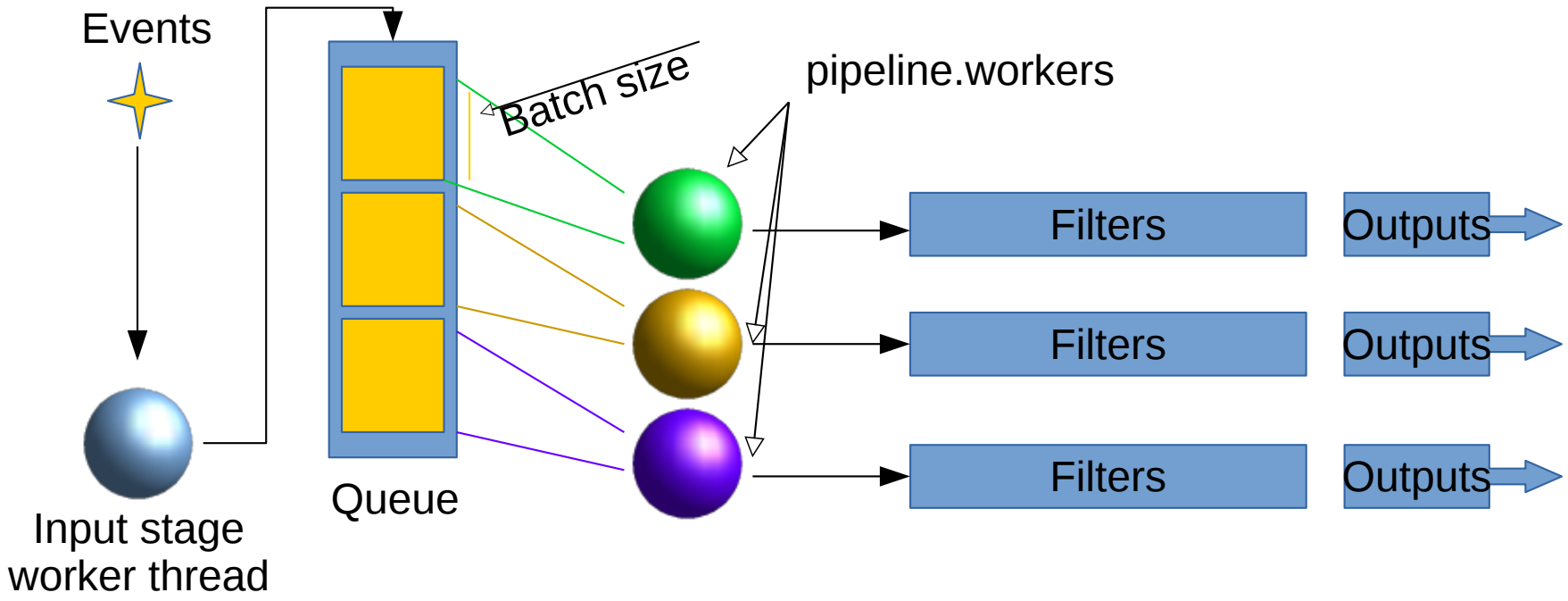
- **pipeline.batch.size** is the maximum number of events an individual worker thread will collect from inputs before attempting to execute its filters and outputs

  Defaults to 125

- **pipeline.batch.delay** is how long in milliseconds to wait for each event before dispatching an undersized batch to pipeline workers

  Defaults to 50 milliseconds

PIPELINE TECHNICAL IMPORTANT SETTINGS (2/5)



Execution model:

- Each pipeline **Input stage** has its **own worker thread** to queue events
- Filters/outputs share a **worker thread** for each **batch of events** taken from the **queue** to process them
- Downstream bottleneck will block upstream

PIPELINE TECHNICAL IMPORTANT SETTINGS (3/5)

As data flows through the event processing pipeline, Logstash may encounter situations that **prevent it** from **delivering events** to the configured output.

> *For example, the data might contain unexpected data types, or Logstash might terminate abnormally.*

To prevent data loss and ensure that events flow through the pipeline without interruption, Logstash provides the following **data resiliency features**:

- Memory Queues → queue.type: memory (default)
- Persistent Queues → queue.type: persisted
- Dead Letter Queues → dead_letter_queue.enable: true (defaults to false)

PIPELINE TECHNICAL IMPORTANT SETTINGS (4/5)

How a persistent queue works:

1. When events are ready to process, Inputs push them to the queue
2. The queue registers events in pages splitting them when too big:
   - The **head** page is the current page (append-only)
   - The **tail** pages are old terminated pages (imutables)
3. The queue regularly **flushes** events and queue states in a **checkpoint file**:
   - **fsync** the head page
   - **mark** acknowledged events
   - **delete** tail pages where all events are acknowledged
4. pipeline.workers (filters+outputs) take batch of events to process
5. Each event ending the pipeline process is acknowledged to the queue

> **queue.checkpoint.writes** *specifies the maximum number of events to write to disk before forcing a checkpoint (defaults to* **1024***)*
>
> *To ensure maximum durability and avoid losing data, set it to 1 at the cost of lower perfomances*

PIPELINE TECHNICAL IMPORTANT SETTINGS (5/5)

How a dead letter queue works:

1. When an event can not be processed event informations are written to the **dead letter queue** with:
   - the original event content
   - metadata describing the encountered error
   - informations about the plugin who wrote the event to the queue
   - a queue insertion timestamp
2. To **reprocess** those events, just create a new pipeline with a **customized dead_letter_queue input**

DEFINING MULTIPLE PIPELINES (1/2)

It is possible to run **more than one pipeline** in the **same process** if we provide Logstash with a **pipelines.yml** file under the logstash **path.settings** setting

```
 //pipelines.yml sample
//This sample contains a list of dictionaries, where each dictionary describes a
//pipeline, and each key/value pair specifies a setting for that pipeline.

- pipeline.id: my-pipeline_1
  path.config: "/etc/path/to/p1.config"
  pipeline.workers: 3
- pipeline.id: my-other-pipeline
  path.config: "/etc/different/path/p2.cfg"
  queue.type: persisted
```

Please note that using -e or -f command line options will make Logstash ignoring the pipelines.yml file and logging a warning about it

DEFINING MULTIPLE PIPELINES (2/2)

Multiple **pipelines** are especially **useful** when:

- event flows do not share the same inputs/filters and outputs
- conditional processing to separate flows becomes too complex
- different durabilities or performances settings are needed for different flows

> *→ Be careful to resources competition between pipelines*
>
> *→ **Persistent queues** and **dead letter queues** are **isolated** per pipeline with their locations namespaced by the **pipeline.id** value*

INPUT SAMPLES - STANDARD INPUT INPUT

```
input {

  # To test or debug Logstash event processing configuration, use stdin

  # Configuration autoreload does not work with this input as it binds to the
  # console to wait for user keyboard inputs

  stdin {
    codec => line { }
    type => "access_log"
    tags => ["access","apache"]
    add_field => {
      "webserver" => "apache"
    }
  }
}
```

INPUT SAMPLES - FILE INPUT

```
input {
    file {
      path => "/var/log/httpd/**/access*.log"
      sincedb_path => "/var/lib/logstash/httpd/access.log.db"
      start_position => "end"
    }
}
```

- **path** an array of file paths which must be absolute ones
- **sincedb** database file keeps track of the current position of log files
- **start_position** can be **beginning** or **end** and choose where Logstash starts initially reading files. Be aware of performances potential impacts...
- **File rotation** is detected and handled but

> *→ be careful when using start_position:beginning as rotating file with a copy of the files will reprocess them*
>
> *→ note that network shared directory can greetly impact performances*

OUTPUT SAMPLES - STANDARD OUTPUT OUTPUT

```
output {

  # To test or debug Logstash event processing configuration, use stdout
  # and the rubydebug codec

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

OUTPUT SAMPLES - ELASTICSEARCH OUTPUT

```
output {

  # The elasticsearch output plugin generates a new index template if not
  # already set in elasticsearch

  # Uses bulk API to send batch of documents (batch size is limited to 20Mb)

  # Bulk document indexation response status of 400 and 404 are sent to the
  # dead_letter queue if enabled. Elsewise Logstash generate a warning log
  # 409 responses are dropped and generate a warning log

  elasticsearch {
    hosts => ["es-node-1","es-node-2"]
    index => "logstash-%{+YYYY.MM.dd}"
    document_type => "%{type}"
    pipeline => "my-ingest-pipeline"
  }
}
```

OUTPUT SAMPLES - ELASTICSEARCH OUTPUT

- With a daily index `logstash-YYYY.MM.DD` you may have to shard cautiously...

> *for instance, using default sharding value of 5 shards would produce 1825 shards a year which consumes a lot of resources and file handles on the host system*

→ Try to change to monthly logs indices `logstash-YYYY.MM`

→ Try 3 shards per index to reduce resources and file handles consumption

OUTPUT SAMPLES - ELASTICSEARCH OUTPUT

- When Logstash wants to log in an inexistant index, `logstash-2018.01.31` for instance:
  - Logstash creates an index template with mapping and settings to handle/define its own fields options (version, @timestamp, geoip, ...)
  - Logstash parameters this template so that it can be applied to all indices named `logstash-*`
  - Elasticsearch will create the index with this mapping template but all other fields use dynamic mapping making it less accurate and efficient for search...

→ The solution is to manage your own **Index Template** with specific:

- settings (Shards / replicas, analyzers)
- mappings
- aliases

FILTER SAMPLES - MUTATE FILTER

```
filter {
    mutate {
      add_field => ["env", "prod"],
      add_tag => ["marqueur"],
      convert => [ "size", "integer" ]
    }
}
```

Mutate command names	Description
`convert`	converts to `integer`, `float`, `boolean`
`gsub`	search/replaces with a regular expression
`join`, `split`	converts an array to `string` or string to array
`lowercase`, `uppercase`	converts character case
`merge`	concatenates two field
`rename`	renames a field
`replace`, `update`	modifies a field value
`strip`	suppresses unused spaces surrounding strings

FILTER SAMPLES - GROK FILTER

```
 # application log sample
# 2016-01-01 10:52:39 DEBUG c.z.Class:200 - A useful debug message

filter {
  grok {
    # The match operation will process 'event.message' and will try to apply a
    # regular expression composed of named subpatterns to set event fields
    # with 'event.message' subpatterns matching part.
    # For instance the following LOGLEVEL subpattern of the grok matching
    # request will try to set 'event.level' field value to the 'event.message'
    # LOGLEVEL matching part...

    match => { "message" =>
      "%{DATESTAMP:timestamp} %{LOGLEVEL:level} \
       %{JAVACLASS:class}:%{NUMBER:line} - \
       %{GREEDYDATA:message}"}
    patterns_dir => "/etc/logstash/patterns"

    # As we try to set a new value to 'event.message', which would create an
    # array of values, we tell grok to drop the old 'event.message' value
    overwrite => ["message"]

    tag_on_failure => ["_grokparsefailure"]
  }
}
```

FILTER SAMPLES - GROK FILTER PATTERNS

```
USERNAME [a-zA-Z0-9._-]+
USER %{USERNAME}
```

→ These are grok patterns samples you can find here:

https://github.com/logstash-plugins/logstash-patterns-core/tree/master/patterns

Nom	Description
USERNAME, HOSTNAME	Code like user name, server name, env, ...
INT, NUMBER	Integer like duration, size
IP, IPORHOST	IPv4 or IPv6
PATH, URIPATH	File path or URL
YEAR, MONTH, DAY... TIMESTAMP_ISO8601, SYSLOGTIMESTAMP	Dates and timestamps
COMMONAPACHELOG, COMBINEDAPACHELOG	Access Logs au format Apache
LOGLEVEL	Info, Warn, Error
JAVACLASS	Java class name
DATA, GREEDYDATA	Unspecified matching like '.*'

FILTER SAMPLES - GROK FILTER USAGE HELPERS

- Ruby regular expressions are not java/sed/go regular expressions...

→ Please take a look at the ruby regular expression documentation: https://github.com/kkos/oniguruma/blob/master/doc/RE

- Test your expressions with specialized online sites:
  - http://rubular.com
  - https://regex101.com

- Use grok matching pattern writer helpers:
  - https://grokdebug.herokuapp.com
  - http://grokconstructor.appspot.com

- write regular expressions of quality, think performances Catastrophic Backtracking

FILTER SAMPLES - DATE

```
filter {
  date {
    match => [ "timestamp",
      "dd/MMM/YYYY:HH:mm:ss Z", "ISO8601" ]
    locale => "en_US"
    timezone => "Europe/Paris"
    tag_on_failure => ["_dateparsefailure"]
  }
}
```

The date filter is used for **parsing dates from fields**, and then using that date or timestamp as **the logstash @timestamp** for the event

- Expressions are expressed in JodaTime format
- Do not forget to manage the locale part of dates (prefer 04 to Apr|Avr|Abr)
- Do not forget to manage timezones

> *For logs consistency, your server times* **must** *be synchronized using* **Network Time Protocol**

USECASES - ACCESS LOGS

```
input {
  file {
    path => " /var/log/httpd/access*"
    type => "access"
  }
}
filter {
  if [type] == "access" {
    grok {
      match => { "message" => "%{NGINXACCESS}" }
    }
    date {
      match => ["timestamp", "dd/MMM/yyyy:HH:mm:ss Z"]
      locale => "en-US"
    }
    geoip {
      source => "clientip"
      target => "geoip"
    }
    useragent {
      source => "agent"
    }
  }
}
```

USECASES - SYSLOG (1/2)

```
//Log sample: Kernel, IPTables / NetFilter, Exim / SendMail / PostFix...
Jul  9 16:12:17 myserver rsyslogd: [origin software="rsyslogd" swVersion="7.4.4" x-pid="9378"
x-info="http://www.rsyslog.com"] start


//RSysLog configuration
*.*     @@127.0.0.1:5000
```

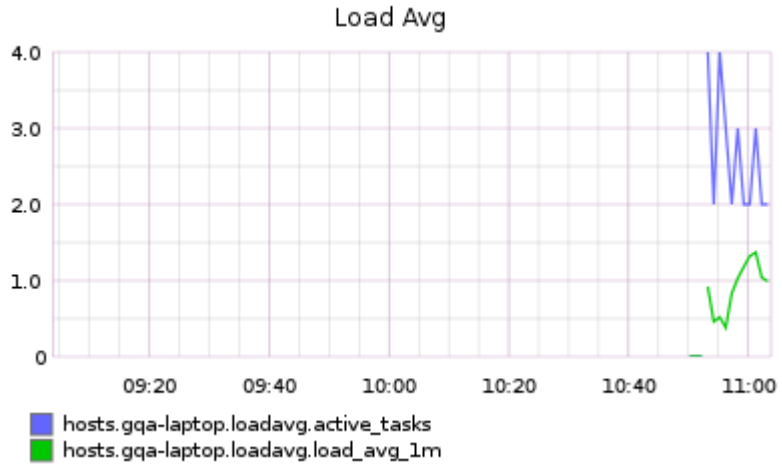
USECASES - SYSLOG (2/2)

```
input {
  tcp {
    port => 5000
    type => "syslog"
  }
}

filter {
  if [type] == "syslog" {
    grok {
      match => { "message" => "%{SYSLOGTIMESTAMP:syslog_timestamp} % \
                  {SYSLOGHOST:syslog_hostname} %{DATA:syslog_program}(?:\[% \
                  {POSINT:syslog_pid}\])?: %{GREEDYDATA:syslog_message}" }
      add_field => [ "received_at", "%{@timestamp}" ]
    }
    syslog_pri { }
    date {
      match => [ "syslog_timestamp", "MMM  d HH:mm:ss", "MMM dd HH:mm:ss" ]
    }
  }
}
```

USECASES - SYSTEM METRICS (1/2)

```
$cat /proc/loadavg
1.13 1.24 1.22 4/928 9934
```



USECASES - SYSTEM METRICS (2/2)

```
input {
  exec {
    type => "system-loadavg"
    command => "cat /proc/loadavg"
    interval => 30
  }
}
filter {
  grok {
    type => "system-loadavg"
    pattern => "%{NUMBER:load_avg_1m} %{NUMBER:load_avg_5m} % \
                    {NUMBER:load_avg_15m} %{NUMBER:active_tasks}/% \
                    {NUMBER:total_tasks}"
  }
}
output {
  graphite {
    host => "graphiteserver"
    port => 2003
    metrics => [
      "hosts.%{host}.loadavg.load_avg_1m","%{load_avg_1m}",
      "hosts.%{host}.loadavg.active_tasks","%{active_tasks}"
    ]
  }
}
```

USECASES - BEATS (1/2)

Beats tool suite is made of **lightweight specialized data shippers** written with the **Go language**

- **Metricbeat** collects **metrics** from your systems and services
- **Filebeat** crawls datas from **log files** (replaces Logstash Forwarder)
- **Packetbeat** collects data traveling over the **wire**
- **Auditbeat** collects the same data as **auditd** on **Linux systems**
- **Winlogbeat** collects **Windows event logs**
- **Heartbeat** collects **isAlive requests** responses

USECASES - BEATS (2/2)

```
//filebeat.yml content
filebeat:
  prospectors:
    -
      document_type: apache
      paths:
        - "/var/log/apache/*"
output:
  logstash:
    hosts: ["logstashhost:5044"]

-------------

//Filebeat command line sample
filebeat -e -c filebeat.yml

-------------

//Logstash beats Input
input {
    beats {
      port => 5044
    }
}
```

HIGH-AVAILABILITY BROKERS - REDIS

```
//Redis output sample
output {
  redis {
    host => "redismq"
    data_type => "channel"
    key => "log_channel"
  }
}

//Redis input sample
input {
  redis {
    host => "redismq"
    data_type => "channel"
    key => "log_channel"
    threads => 1
  }
}
```

HIGH-AVAILABILITY BROKERS - RABBITMQ

```
//RabbitMQ output sample
output {
  rabbitmq {
    host => "rabbitmq"
    exchange => "log_exchange"
    exchange_type => "direct"
    durable => true
    key => "log_key"
  }
}

//RabbitMQ input sample
input {
  rabbitmq {
    host => "rabbitmq"
    exchange => "log_exchange"
    queue => "log_queue"
    key => "log_key"
    durable => "true"
    threads => 1
  }
}
```

HIGH-AVAILABILITY BROKERS - KAFKA

```
//Kafka output sample
output {
  kafka {
    bootstrap_servers => "kafka1:9092"
    topic_id => "logstash"
    partition_key_format => "%{type}"
  }
}

//Kafka input sample
input {
  kafka {
    zk_connect => "zookeeper:2181"
    topic_id => "logstash"
  }
}
```

GOOD PRACTICES (1/2)

- **Correlation**: Use correlation ids to find services events relationships
  - Transmit request or user correlation ids across all systems
  - Use **Mapped Diagnostic Context**, thread contextualized datas of Logback and Log4J libraries to make legacy code logging correlation ids by just changing log patterns to use context data
- **Parsing**
  - Use constant log format with specific field and line separators
  - Prefer machine readable to human readable logs: epoch+TZ vs dates
- **Development**
  - Slice your pipeline configurations per log type to facilitate tests and deployment
  - Do not forget unit testing also works on Logstash configurations (RSpec)
    https://www.elastic.co/blog/logstash-functionality-through-testing
  - Do not forget documentation is your friend (really !)

GOOD PRACTICES (2/2)

- **Deployment**
  - Automate your deployments with Chef, Puppet, Ansible or Docker
- **Performances**
  - Try to use text events instead of json ones as:
    - it takes longer to your application to generate json than text
    - it is a bigger **payload** to send on the wire - may **consume brandwidth**
    - Nevertheless parsed by **json input** to add @timestamp and @version
  - Try to manage multiline codecs at the sender side
  - Try to send log messages through brokers directly when possible
  - Try to not send log messages through SocketAppenders as a blocked reader will block the sender
  - Are the stacktraces always needed ? Is it possible to change your application log level settings while running ?

INSTALLATION

- Install the way you want: tar.gz., .deb or .rpm packages
- Configure your pipeline and settings: Ruby file with **.conf** and

```
 //Start Logstash
logstash -f logstash.conf
logstash -f /etc/logstash.d/**/*

//Modularize, slice and order by name your pipeline configuration files
/etc/logstash.d/01_input_filter_apache.conf
/etc/logstash.d/02_input_filter_syslog.conf
[...]
/etc/logstash.d/90_filter_output_commun.conf
```

- Automatic config reloading

```
 logstash -f logstash.conf --config.reload.automatic
logstash -f logstash.conf --config.reload.automatic --config.reload.interval 3
```

DEBUGGING

```
# Testing a configuration file
logstash -f logstash.conf -t

# Generating verbose (info) and debug logs
logstash -f logstash.conf --verbose -l /var/log/logstash/logstash.log
logstash -f logstash.conf --debug   -l /var/log/logstash/logstash.log
```

- There are many others ways to debug

```
output {
  if "_grokparsefailure" in [tags] or "_dateparsefailure" in [tags] {
    file {
      path => "/var/log/logstash/logstash_parsefailure_%{type}.log"
    }
  } else {
    elasticsearch {
      ...
    }
  }
}
```

PLUGINS

```
//Install plugins
bin/logstash-plugin install logstash-filter-xxx

//Listing plugins with version informations
bin/logstash-plugin list --verbose
```

- Usages ?
  - Input, Codec, Filter, Output
  - All is plugin with Logstash
    - Standard plugins → provided
    - Additional plugins → to install
  - Based on RubyGems

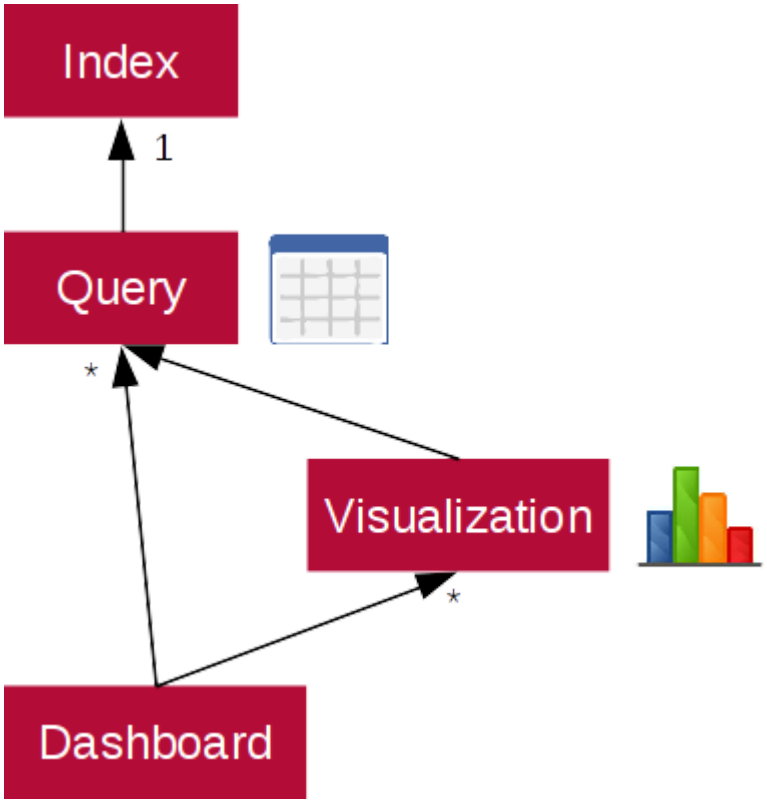
Lab 2

KIBANA

WORKFLOW

1. Manage **settings** and **index patterns** to tell kibana:
    - Which mapping it will have to use in requests
    - Which fields are eligibles in certain types of queries
2. **Discover** Index → Query
    - Indicate criterias to filter data
    - Live browse index data
    - Create the query you like
3. **Visualize** Query → Aggregation → Visualization
    - Choose your saved queries or/and a visualization type
    - Build and parametrize your aggregations on its results
4. **Dashboard** Visualization → Dashboard
    - Build your synthetic dashboards from your saved queries and visualizations

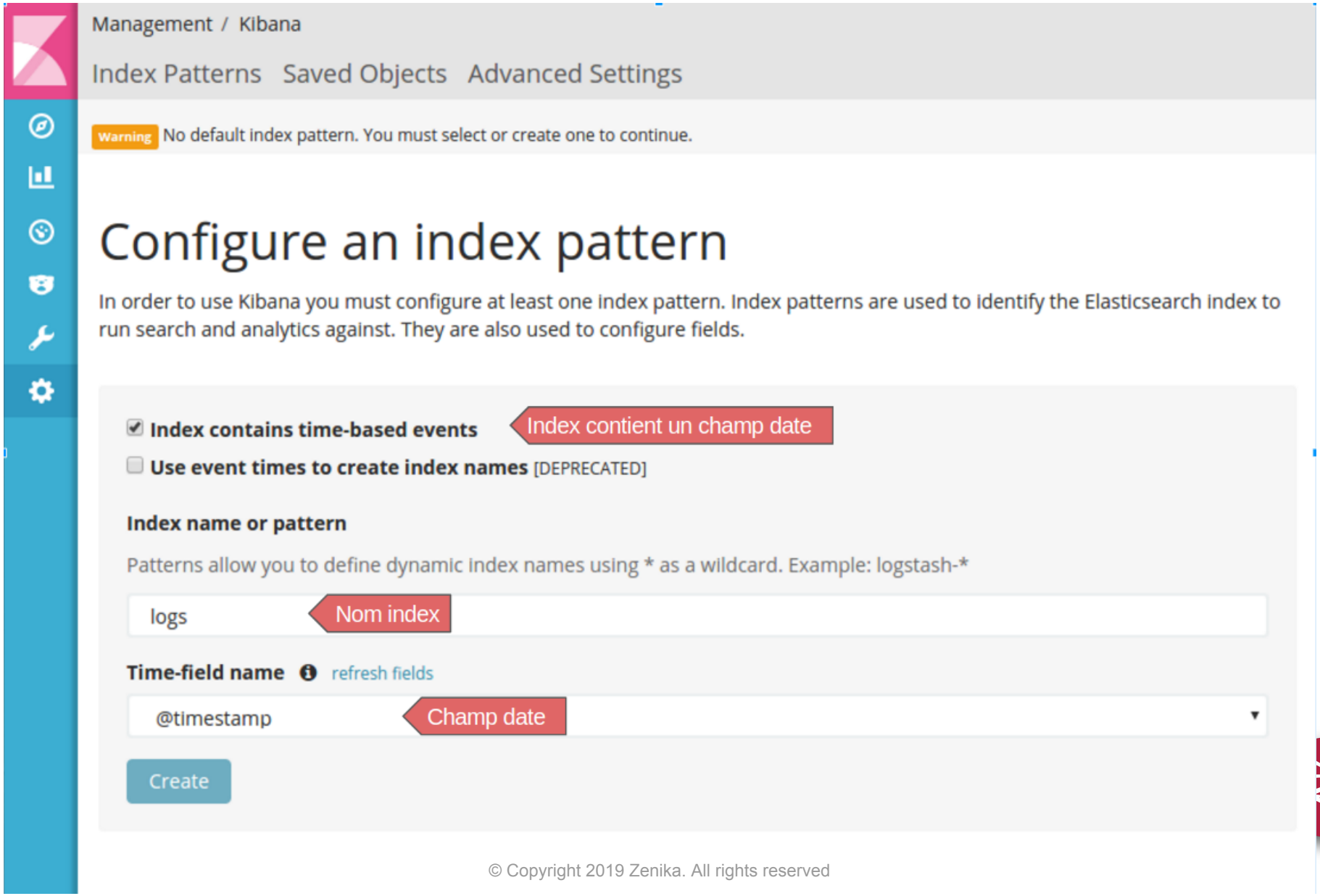
MODEL



MANAGE KIBANA SETTINGS

- Things to **manage** in Kibana :
  - Index patterns
    - Detecting field mappings
    - Add scripted fields
    - Add source fields filters (to hide them from discovery results)
  - Saved Objects (a place to import/export saved queries/views/dashboards)
  - Advanced settings

MANAGE KIBANA SETTINGS

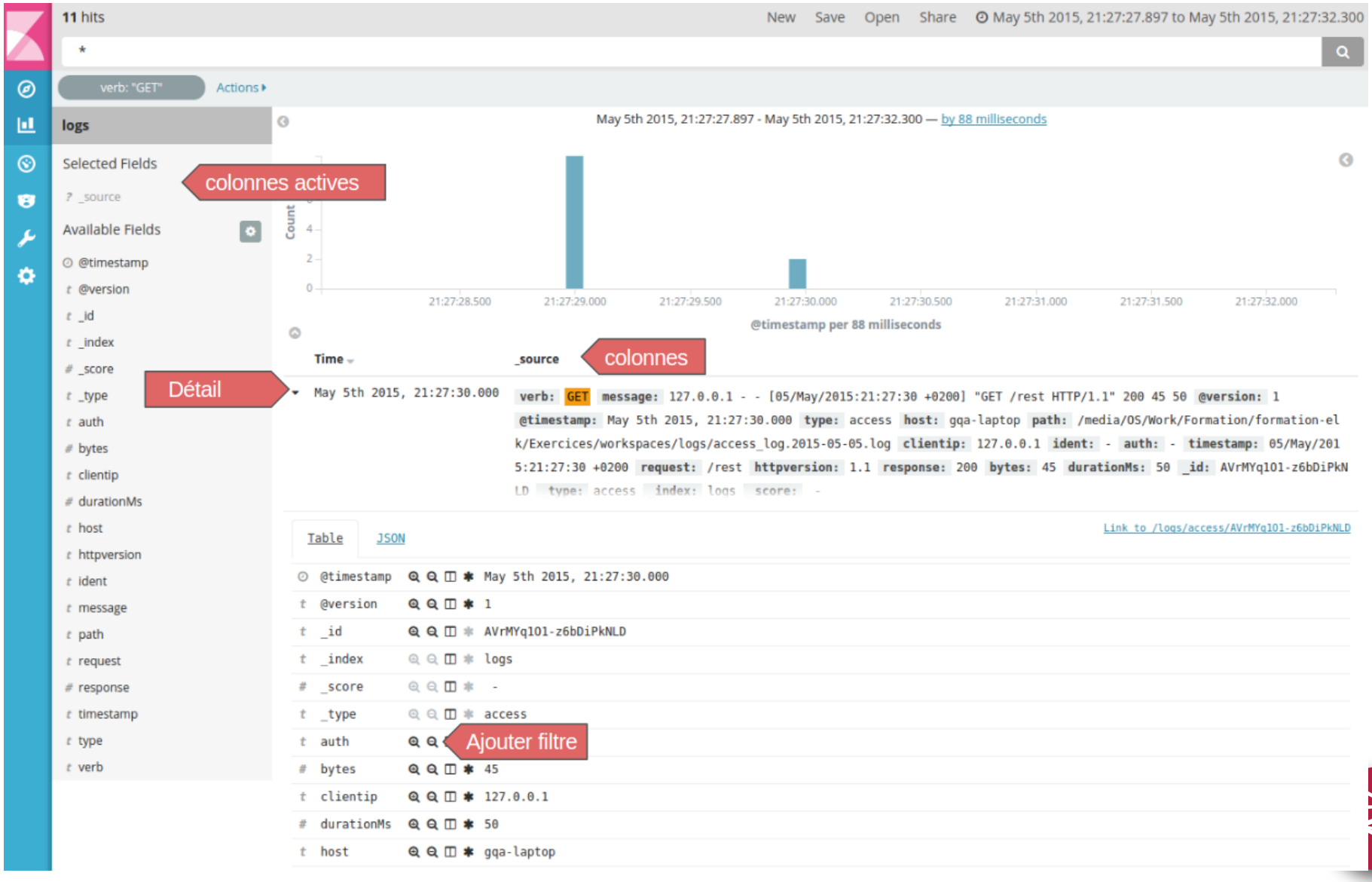


DISCOVER

- This tab lets you:
  - Add a date interval criteria
  - Add a lucene syntax textual request
  - Add or complete dynamically the request criteria list from the GUI
  - See a result sample of your query
  - Change the result table columns list
  - Open/save named queries
  - See the details of json requests/results

DISCOVER

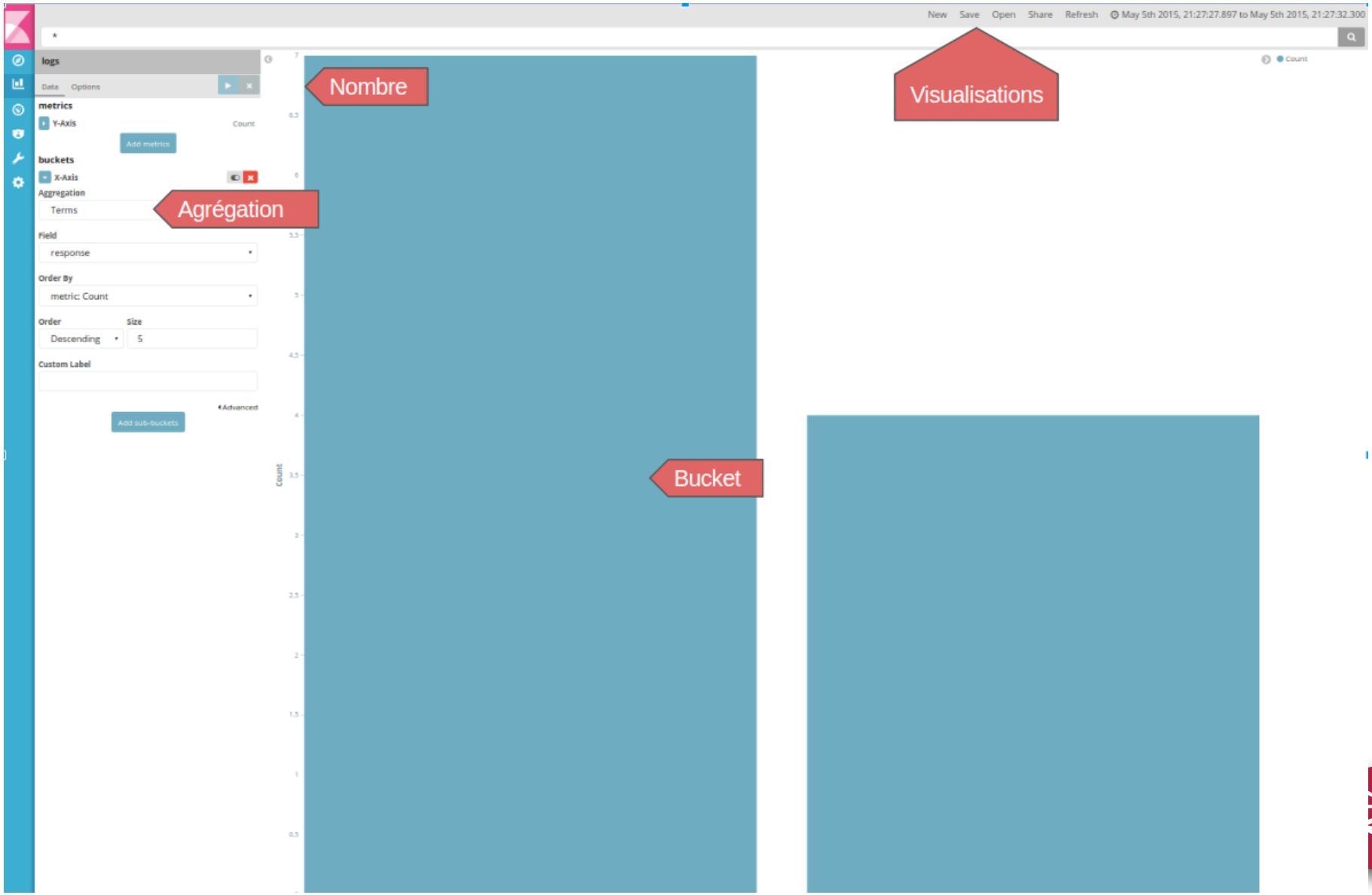
DISCOVER



VISUALIZE

- This tab lets you:
  - Select a view type depending of what you want to show
  - Open an existing query
  - Create a new basic query by selecting the target index (i.e filtered by time range only)
  - Build and parametrize your aggregation depending on visualization type and index field mappings
  - Open/save named visualization
  - See the details of json aggregation requests/results

VISUALIZE



TIMELION (1/3)

- **Timelion**, pronounced "**Timeline**", brings together totally **independent data sources** into a **single** interface, driven by a simple, one-line **expression language** combining data retrieval, time series combination and transformation, plus visualization.

- Timelion is answering questions like:
  - How many pages does each unique user hit over time?
  - Whats the difference between this Friday and last Friday?
  - What % of Japan’s population came to my site today?
  - What’s the 10 day moving average of the S&P 500?
  - How what is the cumulative sum of all searches made in the last 2 years?

TIMELION (2/3)

- Every Timelion expression starts with a data source function

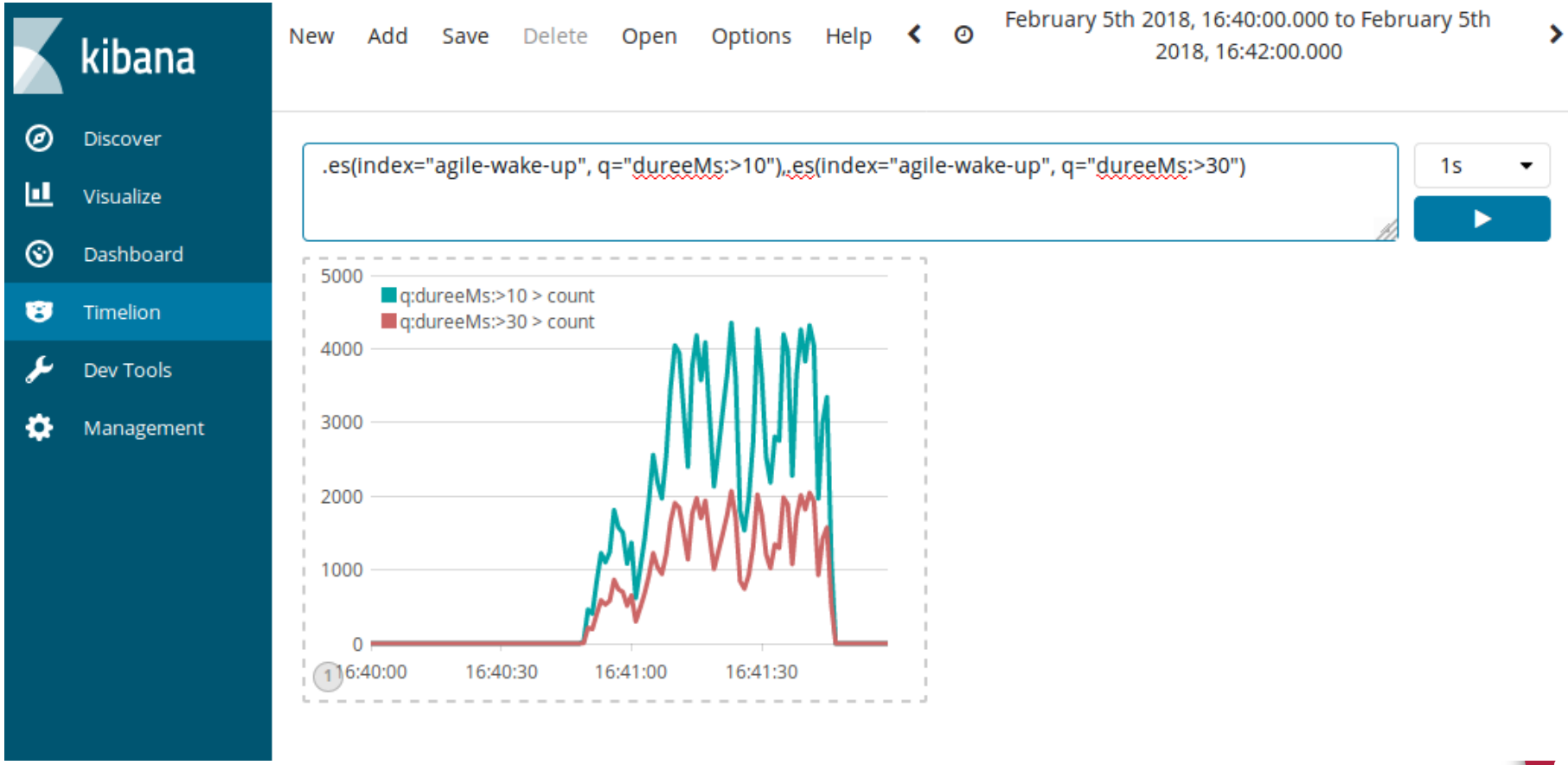
```
.es(*) counts everything in Elasticsearch over time
```

- **wbi** external datasources function exists to get data from the **World Bank Open Data**
- Each datasource function represents time series data
- Comma separated datasource functions will show multiple time series on the same chart
- Other functions exist to tranform the data like:
  - Cumulative average / Moving average
  - trend
  - derivative
  - changing range with shape adjustment
  - ...
- Functions are **chainable**

TIMELION - SAMPLE (3/3)

```
//This timelion query will show two series
//One counting http requests during more than 10 ms over time
//The second one counting http requests during more than 30 ms over time

.es(index="agile-wake-up", q="dureeMs:>10"),.es(index="agile-wake-up", q="dureeMs:>30")
```



DASHBOARDS

- This tab lets you:
  - Add saved queries (shown as a table in the dashboard)
  - Add saved visualizations diagrams with their legend
  - Add saved timelion series with their legend
  - Organize their order and size
  - Open/save named dashboards

DEV TOOLS

- This tab lets you:
  - Create json requests on the left side of the screen:
    - play them
    - use autocomplete to help forging complex requests
    - Autoindent them
    - Persistent while local storage is not emptied
  - See the requests results on the right side

INSTALLATION

- Install : paquets .tar.gz., .deb, .rpm
- Configure the **kibana.yml** file as needed
- Start a Kibana process

CONFIGURATION

- Kibana settings:
  - **elasticsearch.url** tells Kibana about the Elasticsearch node to request on
  - **kibana.index** to give a custom name to the Kibana index
- Security options
  - Use Stack Features (X-Pack) or Search Guard
  - Use a reverse proxy for authentication and SSL processing
- High availability
  - Instanciate Kibana for each Elasticsearch client node
  - Use a reverse proxy and load balance over kibana instances

Lab 3

CONCLUSION