Madhusudhan Konda

Elasticsearch First Steps

chocolateminds.com

Poll

What is your experience with Elasticsearch?

A) Just starting up (Beginner level)
B) I know a bit (Intermediate level)
C) I’m an expert (advanced level)

Resources

GitHub

https://github.com/madhusudhankonda/

Repository: elasticsearch-first-steps

Wiki: elasticsearch-first-steps/wiki

Search is the new normal!

- Search is a top-class feature
  - No business can survive without a search feature
  - Competitive advantage
  - Not just for end-users. Heavy usage internally by the businesses
  - Full text search, with relevancy.
  - Search and analytics
  - Perhaps machine learning features?
- Github, Netflix, Amazon, Stackoverflow..

Search is not a new feature!

- Database backed search
  - Flight reservations for last month
  - No. of malicious attacks on a website
- Pretty basic - range, filter
  - Relevance/scoring
  - No support for full text
- No advanced features, analytics

Elasticsearch

Not so new kid on the block!

Ultra fast, open source, distributed *Search and Analytics** engine.

*Search, store and analytics

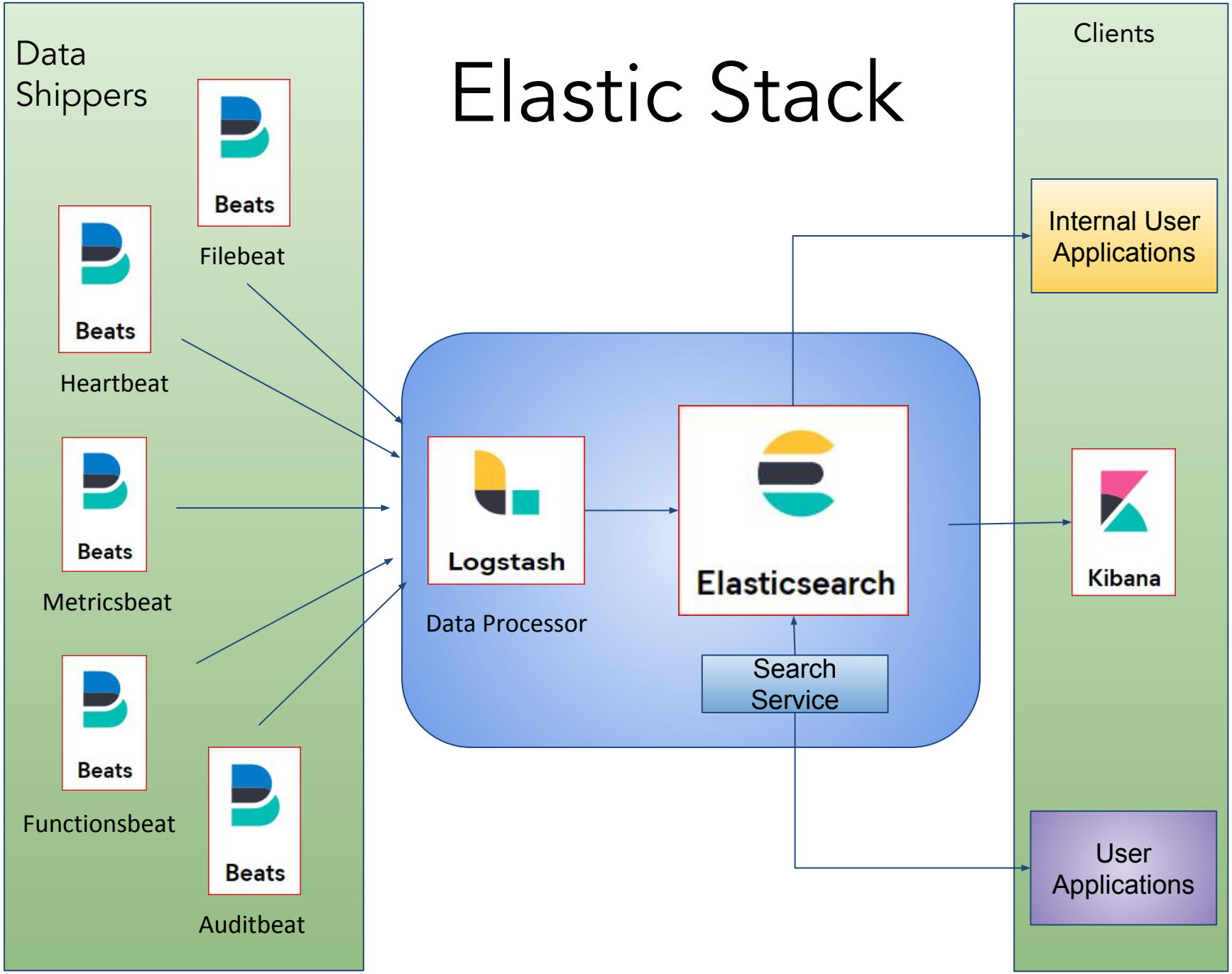
Distributed Database?

Functional and Technical Features

- Full-text
- Relevance,
- Auto-completion
- Suggestion
- Fuzzy search
- Highlighting
- Analytics
- ML

- Blazing fast!
- Distributed, HA
- Auto-scalable
- Fault tolerant
- Multi language support
- Rich set of APIs
- Visualizations

Elastic Stack



Thing of a Beauty!

- Just works!
  - Complexity is beautifully wrapped away
  - Convention over configuration (defaults)
- Consistent APIs
  - REST over HTTP
  - Native Clients (Java, C#, Python, JavaScript..)
- Modular architecture (Elastic Stack)

Elasticsearch in Action

- Installation
  - Binary/archive
  - Docker
- Indexing some data
  - Books Documents
- Searching
  - Search for a book
  - Aggregate ratings

Installing Elasticsearch

- Download elastic.co/downloads
  - Elasticsearch version: 7.8
  - Use Binary/Archive for this class
- Installation
  - Unpack and run Elasticsearch

- Test
  - Visit http://localhost:9200 or
  - `curl localhost:9200` (cmd)

Installing Kibana

- Download elastic.co/downloads
  - Kibana version: 7.8
  - Use Binary/Archive for this class
- Installation
  - Unpack and run Kibana
    http://localhost:5601

Check Check!

- Sample Application - Top Ten Books
- Index a sample document
- Retrieve the document
- Search for it
- Quick tour of Elasticsearch and Kibana Configuration options
- Quick tour of Kibana!

Pulse Check

Has everyone got their setup done?

Exercise 1

- Index a movie document with two fields - `title` and `release_date`.

  “title”: “Godfather”

  “release_date”:”1982-08-01”
- Retrieve the same document using GET

- Index another movie document but this time with an ID as 99
- Retrieve the document with id 99

Exercise 1 Solution

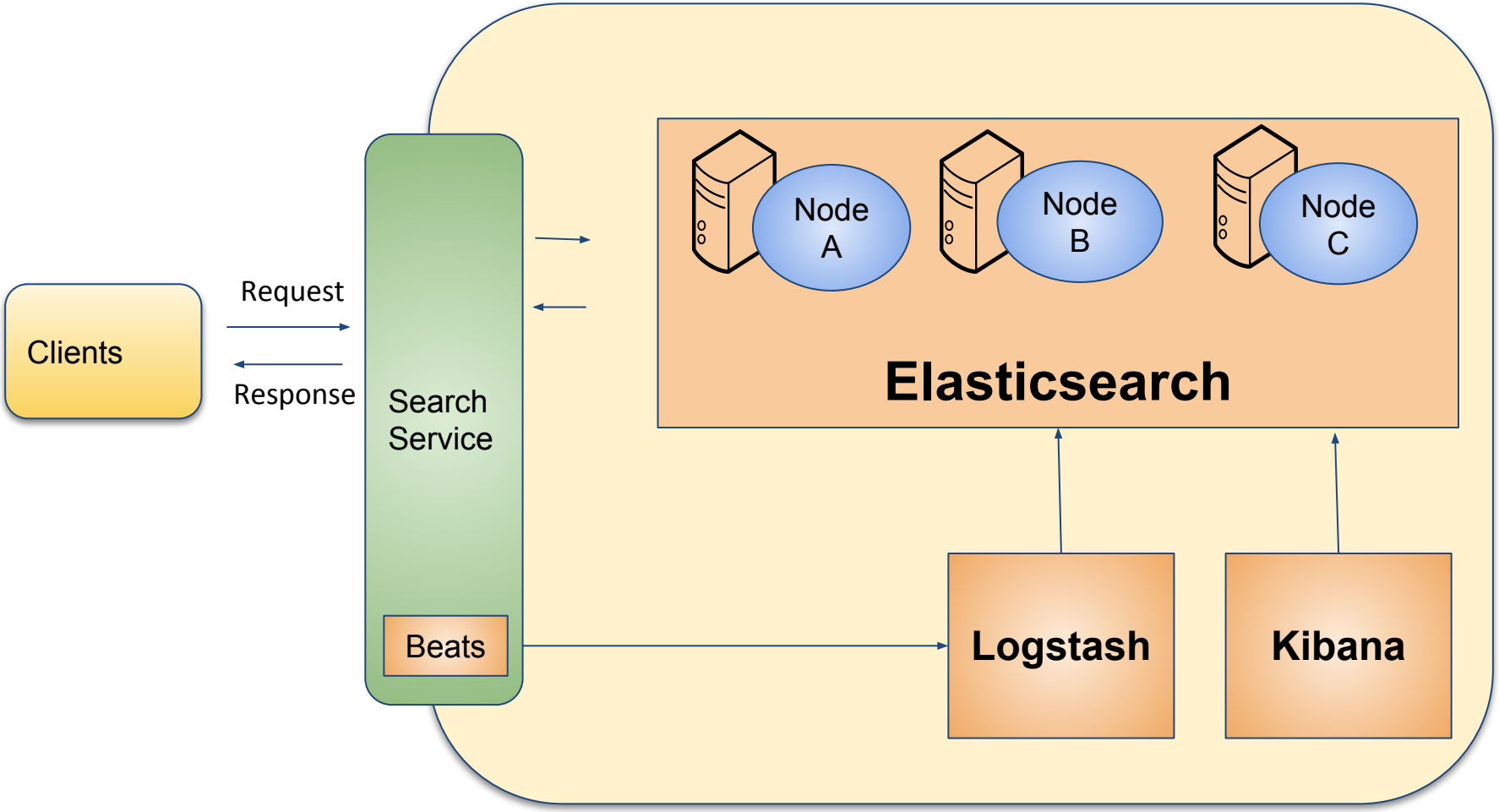
```
POST movies/_doc // no ID?!
{
  “title”: “Godfather”
  “release_date”:”1982-08-01”
}
```

- GET movies/<doc_id>
- PUT movies/_doc/99 //Ah, ID!

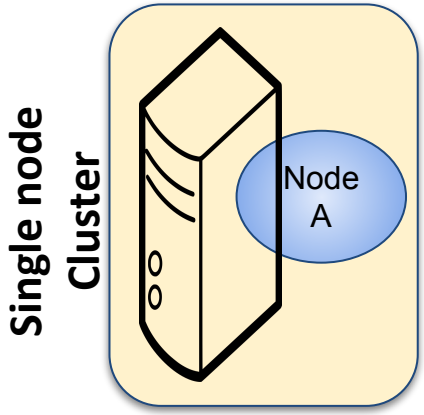
```
{
  “title”: “Godfather II”
  “release_date”:”1986-08-01”
}
```

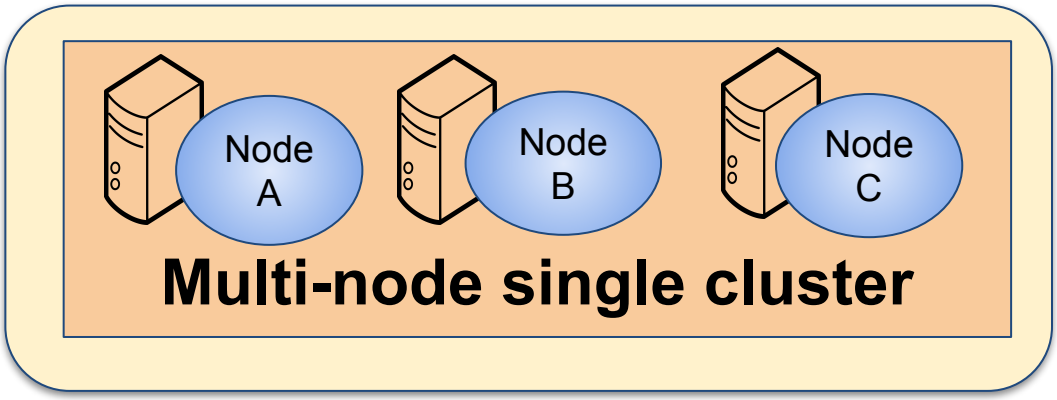
- GET movies/_doc/99

Fundamental Concepts


Clients
Request
Response
Search
Service
Beats
Node
A
Node
B
Node
C
Elasticsearch
Logstash
Kibana

Distributed Clusters

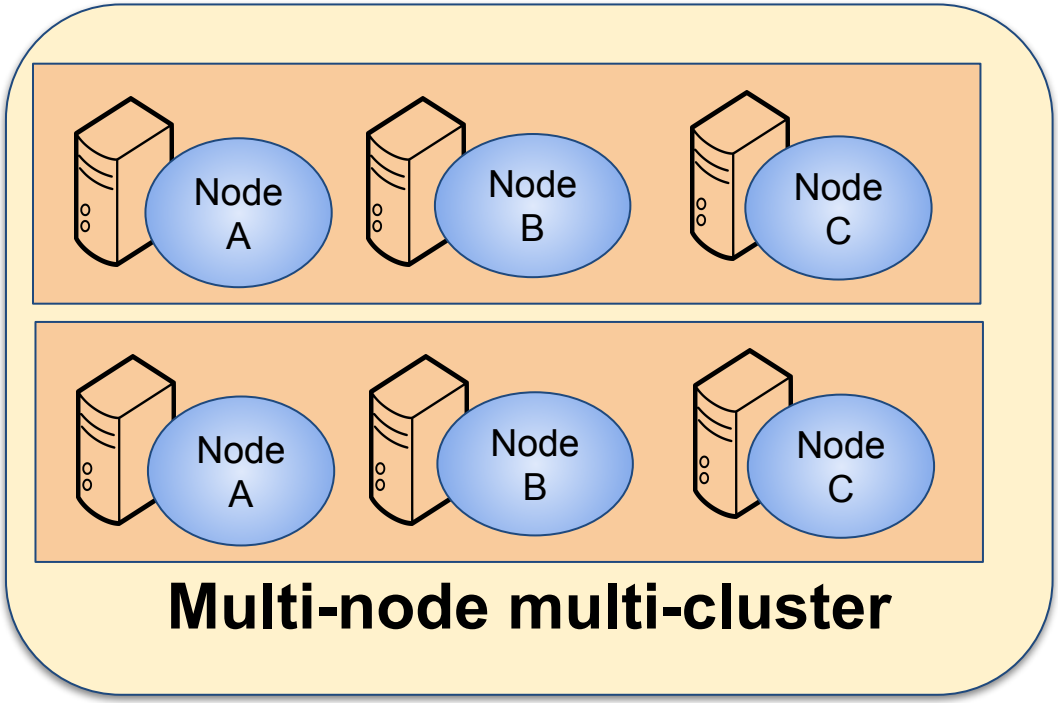


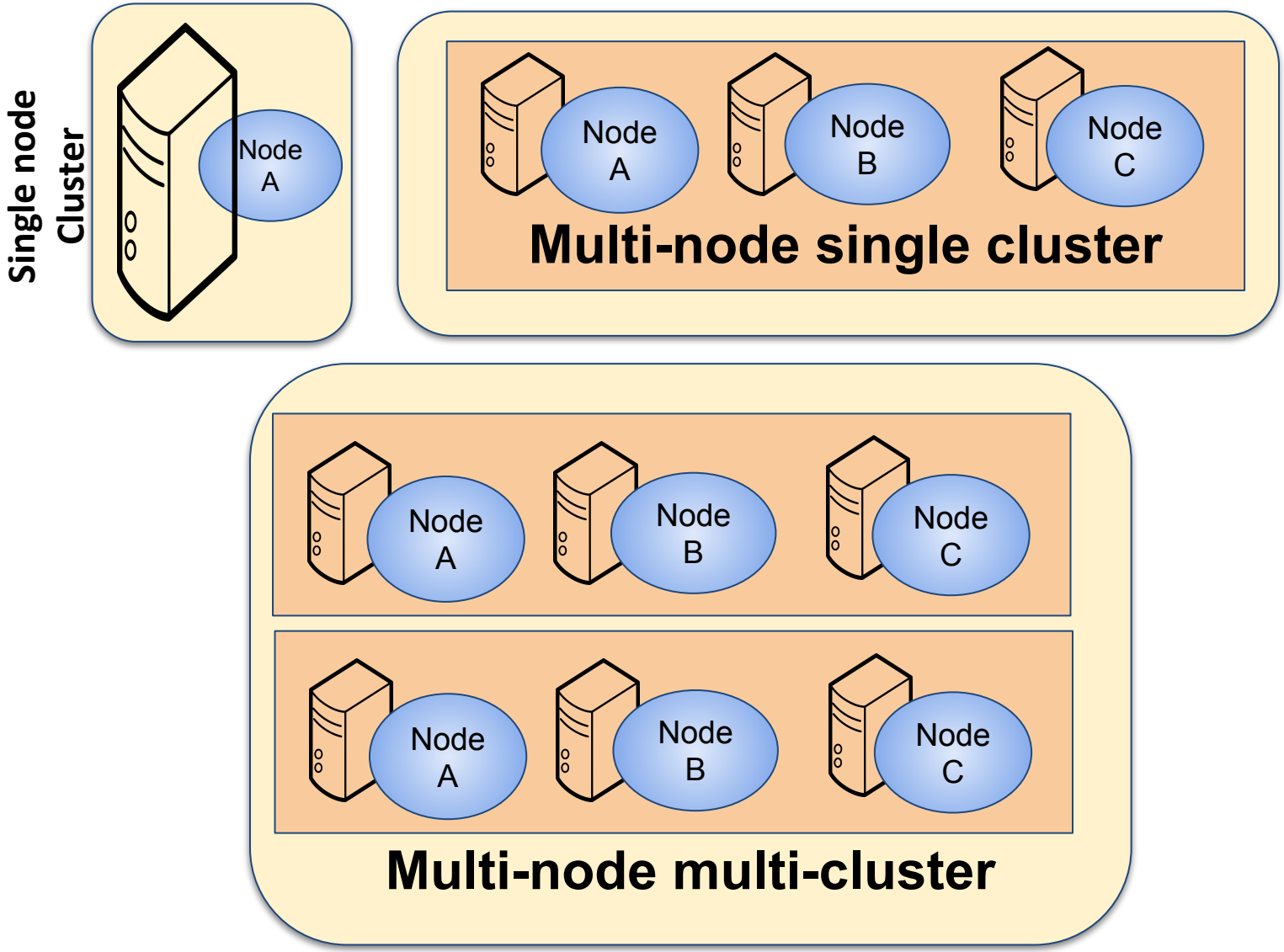


Node Roles

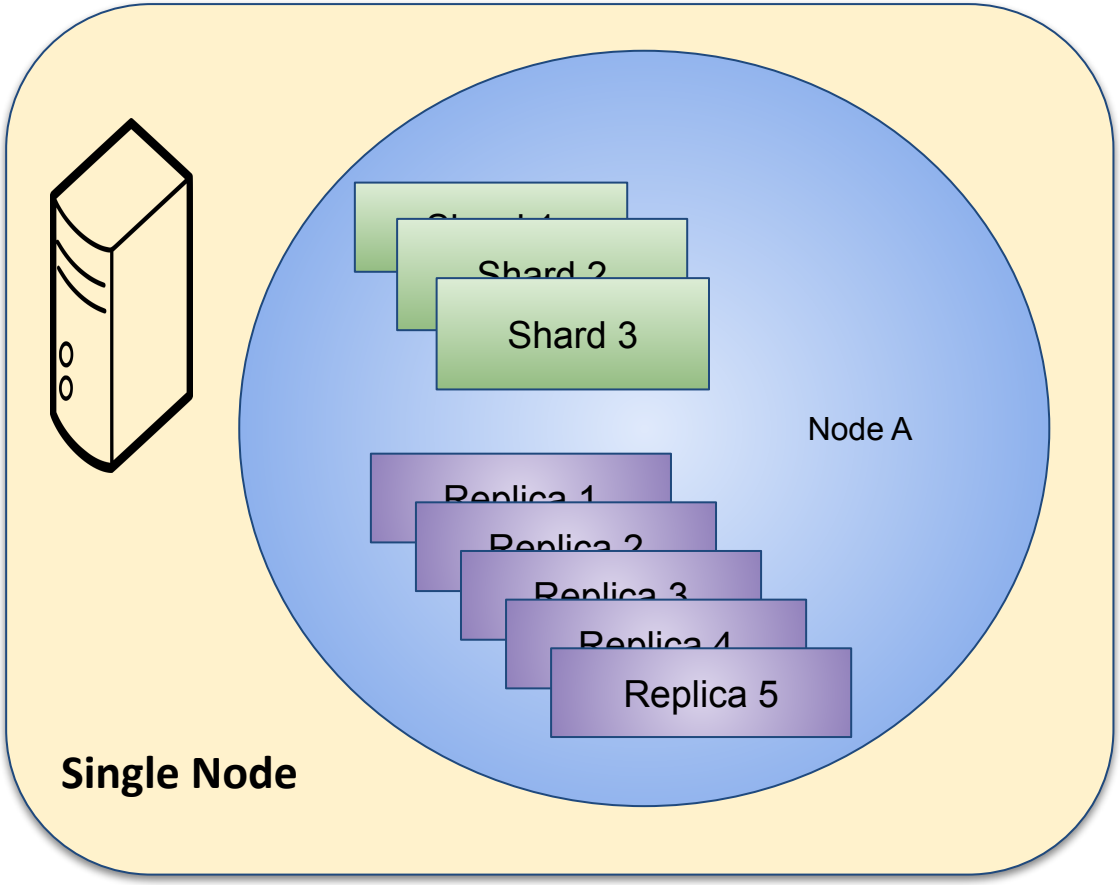
- master
- data
- coordinator
- ingest

- Horizontal and Vertical scaling
- Auto-scales





chocolateminds.com

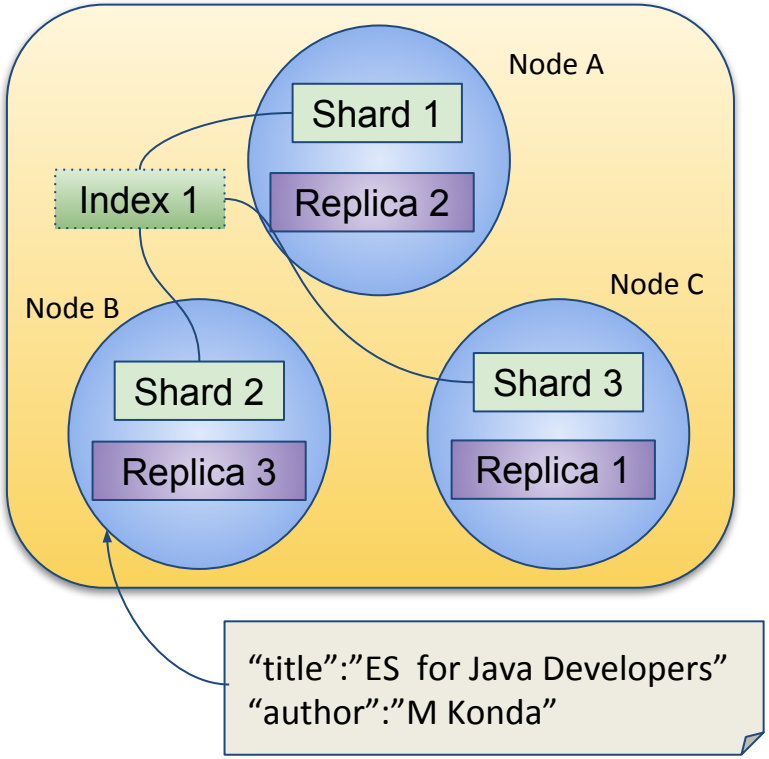

Shard 3
Node A
Replica 5
Single Node

Index and Shards

- Index is a logical structure
- Index is distributed to multiple shards
- Shard is Lucene Instance
- Replica is a copy
  - Serves read requests
- Doc's home using routing algorithm:

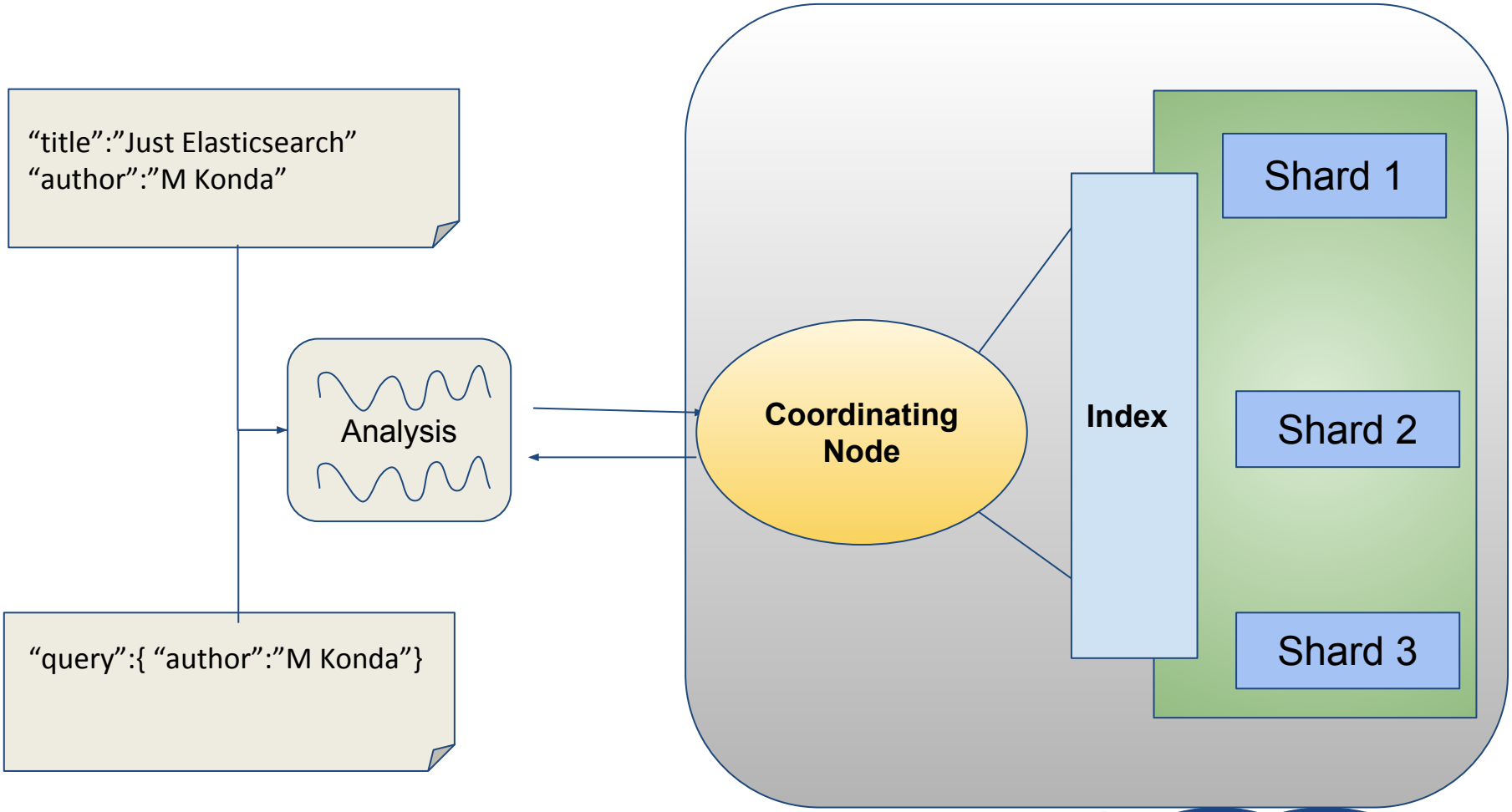
```
shard = hash(ID) % num_of_shards
```

- Evenly distributed; no hotspots



Once instantiated shards can't be changed for an index (!)

Indexing and Search Mechanics



Single doc retrieval with an ID will be forwarded to the shard containing it

Term queries will NOT be analysed

Inverted Index

- Heart and soul of a Search Engine
- Per full-text field
- A Hashmap like data structure
- List of words and their occurences in the documents
- Faster search
- Additional info: Field length, word position etc

Word	Doc Num
imagination	1, 2
is	1
more	1
important	1
than	1

Inverted Index

- Stop words
  - a, the, at
- Stemming

  -fight, fighter, fought

Example

quote is a full-text field!

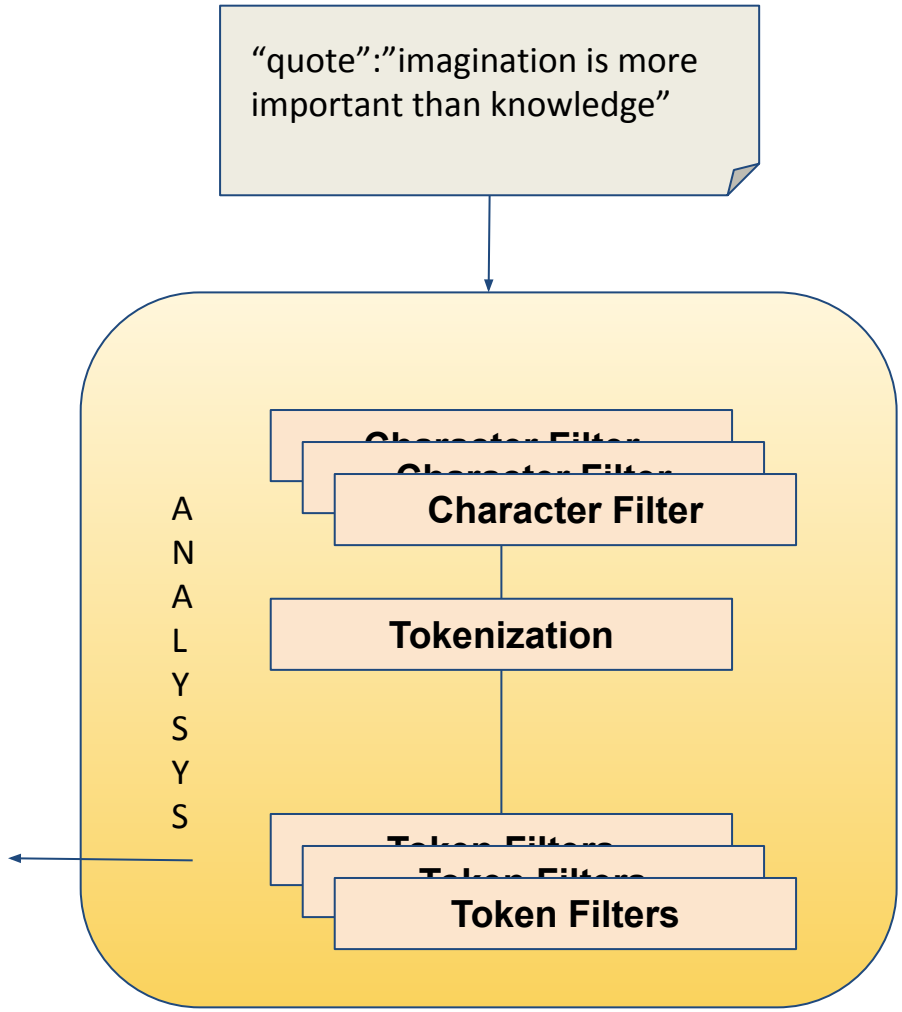
```
{
    "author":"Albert Einstein",
    "quote":"Imagination is more important than knowledge"
}
```

```
{
    "author":"John Lennon",
    "quote":"Reality leaves a lot to the imagination"
}
```

```
{
    "author":"Napoleon Bonaparte",
    "quote":"Impossible is a word to be found only in the dictionary of fools"
}
```

Building of an Inverted Index

- Analyser
  - Char Filters (0..n)
  - Tokenizers (1)
  - Token Filters (0..n)



Inverted Index

Word	Doc Num
imagination	1, 2
is	1
more	1
important	1
than	1

Poll

If we set `number_of_shards` as 3 and `number_of_replicas` as 2 for our newly created index, how many shards and replicas will be created altogether when an index is created?

a) 6
b) 9
c) 12

Mapping and Text Analysis

Mapping

- Definition of your data
  - keywords, text
  - Format of the date, ip
  - nested, completion
- Field data types
  - string,keyword, date etc
- Dynamic and Explicit mapping
- Schemaless..but..
- Mapping templates

You know more about your data. Don't let Elasticsearch guess it.

- string, keyword
- int/long/float..
- date
- boolean
- range
- ip
- object
- nested
- geo_point, geo_shape ...

Date Types

- Strings of a particular format
- “2020-05-11” “2020/05/11”
- integer in seconds since the epoch
- long in milliseconds since the epoch

```
PUT movies/_doc/1
{
    "title":"Godfather",
    "release_date":"1972/08/24"
}

GET movies/_mapping
..
"release_date" : {
  "type" : "date",
  "format" : "yyyy/MM/dd HH:mm:ss||yyyy/MM/dd||epoch_millis"
  }
```

Explicit mapping

```
PUT books_with_mapping
{
  "mappings": {
    "properties": {
      "title":{
        "type": "text"
      },
      "release_date":{
        "type": "date"
      },
      "rating":{
        "type": "integer"
      }
    }
  }
}
```

```
PUT books_with_multifield_mapping
{
  "mappings": {
    "properties": {
      "author":{
        "type": "text",
        "fields": {
          "orig":{ // author.orig
            "type":"keyword"
          }
        }
      }
    }
  }
}
```

No arrays as a data type - every fields accepts “arrays”

Elasticsearch.guru

Exercise 2

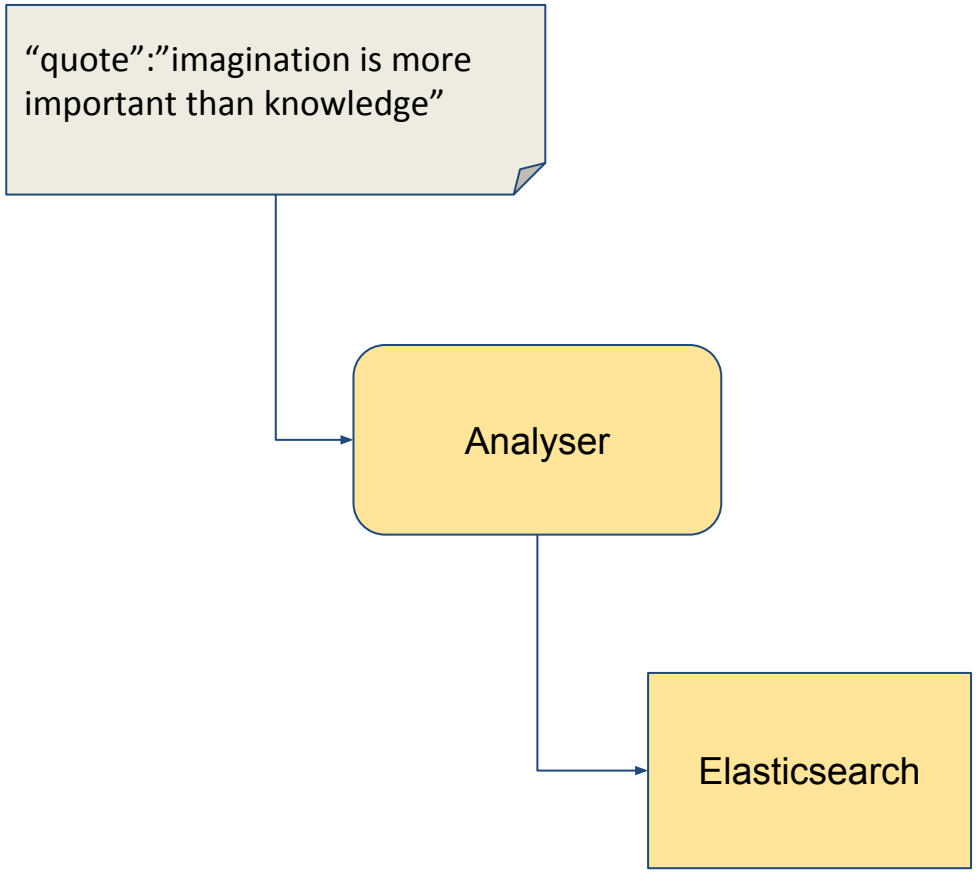
- Create a mapping for index `movies`:
- `title` is both `text` and a `keyword` type
- Synopsis is `text` type
- release_date is `date` type but format is `dd-MM-yyyy`

Exercise 2 Solution

```
PUT movies
{
  "mappings":{
    "properties":{
      "title":{
        "type":"text",
        "fields": {
          "keyword":{
            "type":"keyword"
          }
        }
      },
      "synopsis":{
        "type": "text"
      },
      "release_date":{
        "type": "date",
        "format": "dd-MM-YYYY"
        }}}}
```

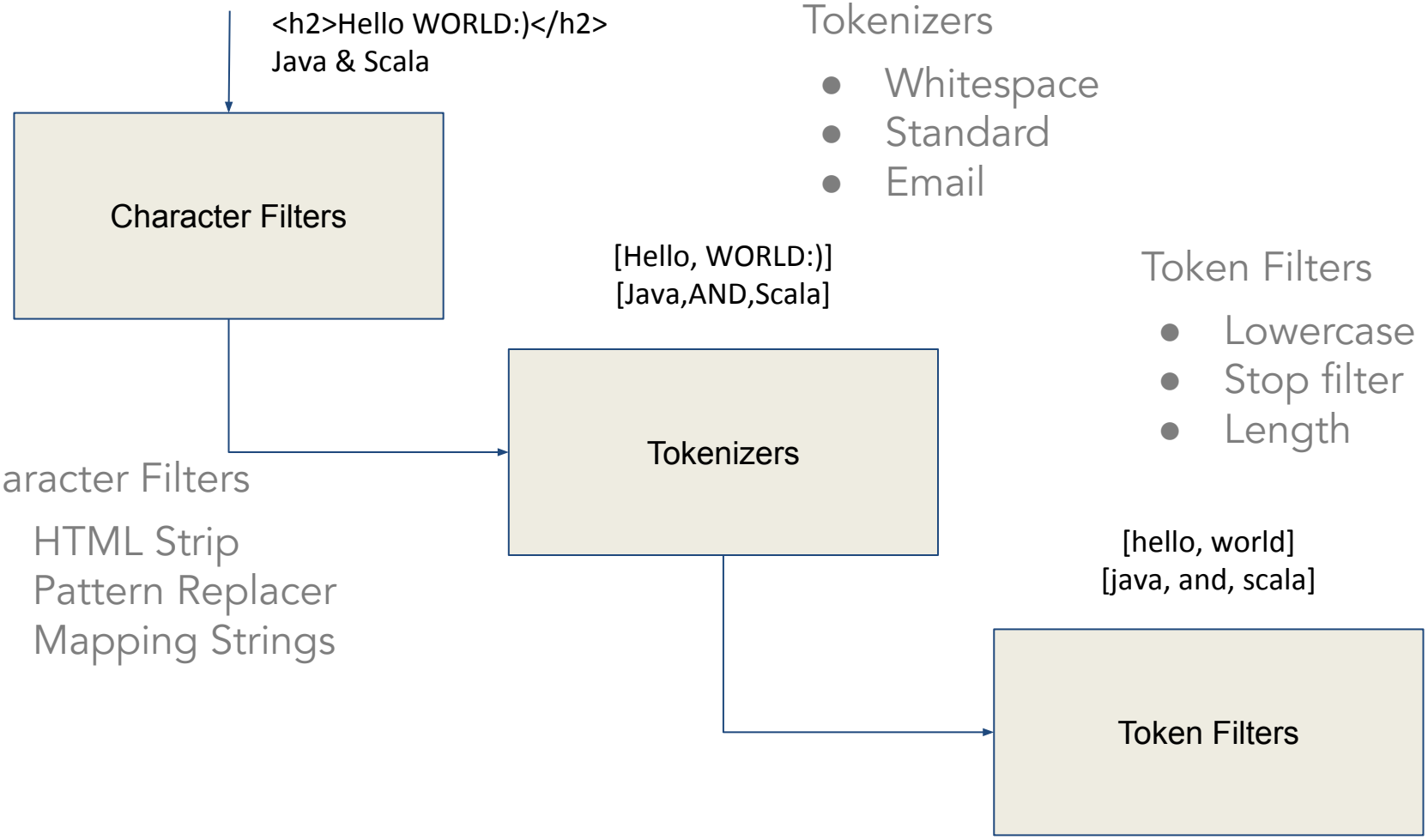
```
POST movies/_doc
{
  "title":"Godfather",
  "synopsis":"Moving story about..",
  "release_date":"01-01-1980"
}
```

Analysers



- In built Analysers
  - Standard
  - Simple
  - Whitespace
  - Stop
- Custom Analysers
  - Build your own

Low Level Blocks



Tokenizers

- Whitespace
- Standard
- Email

Token Filters

- Lowercase
- Stop filter
- Length

Character Filters

- HTML Strip
- Pattern Replacer
- Mapping Strings

Indexing Operations

Index

- Collection of documents
  - Logical grouping
  - Spread between shards
- Automatically gets created when indexing a document
- Three parts:
  - Aliases
  - Settings
  - Mappings

```
PUT cars/_doc/1
GET cars
{
  "cars" : {
    "aliases" : { },
    "mappings" : { },
    "settings" : { }
  }
}
```

Index Settings

- Settings: configuration of the indices
  - How many shards, replicas
  - Allow/disallow writes and/or reads
  - Refresh interval, codecs
- Static Settings
  - Pre-creation; Number of shards
  - Cannot be changed once index is created (scrap that index, reindex and reapply!)
- Dynamic Settings: replicas, allow/disallow writes
  - _settings API

Why can't ES change Shards?

- Routing
  - Process of allocating a home to your doc
  - Doc belongs to one and only one shard
  - Spreads the load evenly
- Routing Algorithm (recap)
  - Doc's home

```
shard = hash(id) % num_of_shards
```

  - Index and Search retrieval

Index Templates

- Templated index creation
  - org wide settings across clusters
  - glob pattern
- _template API
- No retrospective changes!
- Multi-cluster (Indices Modules)

```
PUT /_template/dev_template
{

"index_patterns":["*_orders","flights_*"],
  "settings":{
    "number_of_shards":5,
    "number_of_replicas":2
  }
}

//Creating failed_order uses *_orders
PUT failed_orders

//Creating flights_landed uses flights_*
PUT flights_landed
```

Exercise 3

- Create a template with
  - index pattern: *_movies
  - number_of_shards:3
  - number_of_replicas:1
- Create an index: classic_movies and check if the template pattern was applied

Exercise 3 Solution

```
1. PUT _template/movie_template
2. {
3.   "index_patterns":["*_movies"],
4.   "settings":{
5.     "number_of_shards":3,
6.     "number_of_replicas":1
7.   }
8. }
PUT classic_movies
PUT comedy_movies
```

Exercise 3.1 is available in github!

Aliases (_alias API)

- Alternate names
- Single Alias -> multiple Indices
- Aggregating data from multiple indices
- Zero downtime during reindexing op

```
//Create power_cars alias to cars index
PUT classic_cars/_alias/cars

//multi-index aliasing
PUT vintage_cars

//Create power_cars alias to cars index
PUT vintage,classic_cars/_alias/cars_alias

//Create alias based on wildcards
PUT *cars*/_alias/all_cars_alias
```

- Best practice is to follow a naming convention like *alias or *_a
- Index name cannot be set as alias name!

Searching on an Alias

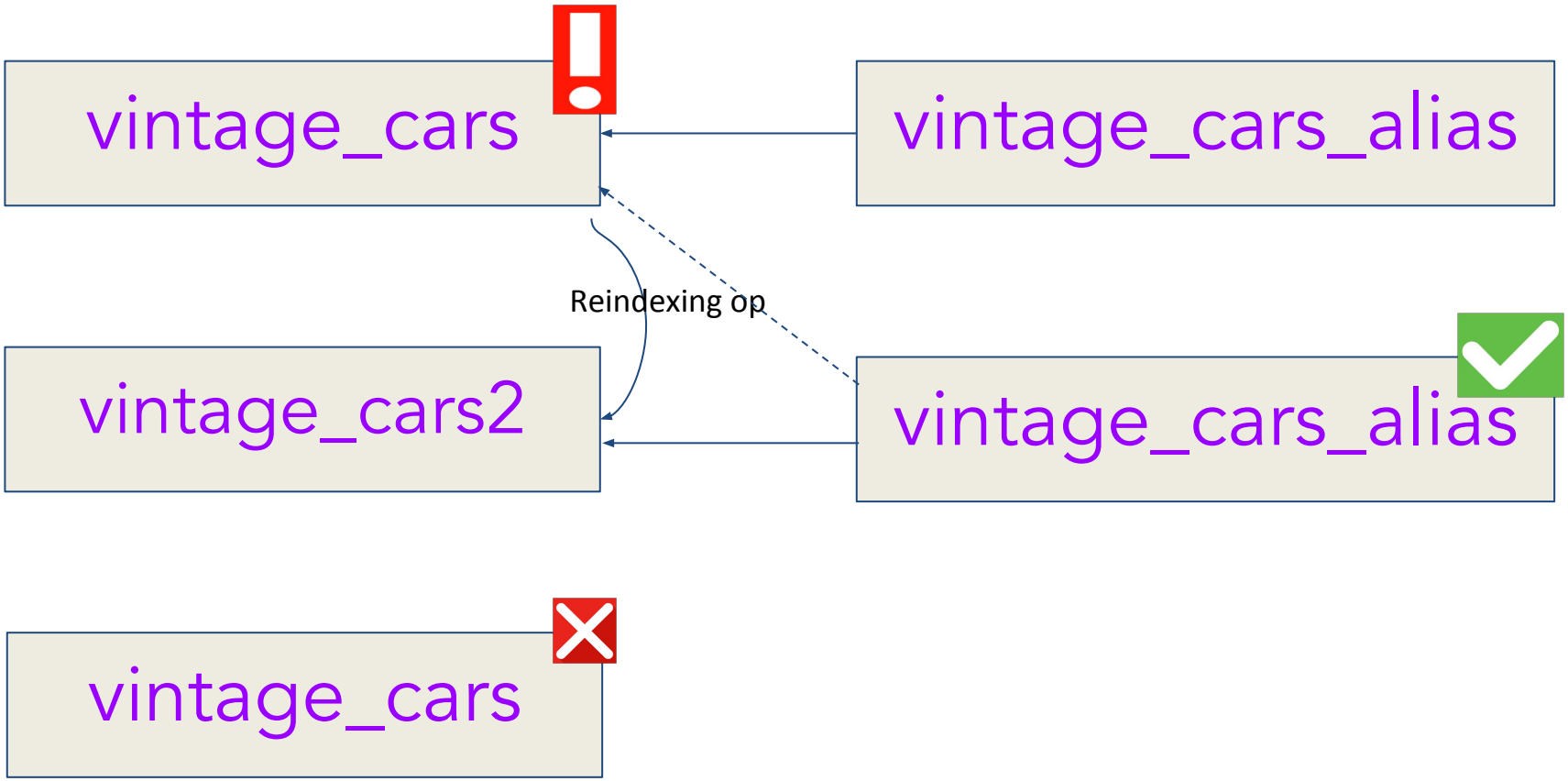
```
// Search across all *cars indices - a bit of mouthful
GET power_cars,vintage_cars,old_cars,classic_cars/_search

// Search on a single alias
GET all_cars/_search

// Find all the aliases
GET vintage_cars/_alias

GET _cat/aliases
```

Alias in a PROD env

Superb feature!
But often
overlooked :(

- Index: vintage_cars
- Alias: vintage_cars_alias -> alias to vintage_cars
- vintage_cars has a problem!!
- New index: vintage_cars_new -> reindexed from vintage_cars index
- Delete alias: delete vintage_cars_alias and recreate against vintage_cars_new index

Alias in a PROD env



Aliases (_aliases) API

```
DELETE vintage_cars,vintage_cars2
PUT vintage_cars
PUT vintage_cars2
GET vintage_cars2
PUT vintage_cars/_alias/vintage_cars_alias

POST /_aliases
{
    "actions" : [
        { "remove": { "index": "vintage_cars", "alias": "vintage_cars_alias" } },
        { "add":  { "index": "vintage_cars2", "alias": "vintage_cars_alias" } }
    ]
}
```

Multiple actions:

- add
- remove
- remove_index

Exercise 4: Aliases

- Create an alias `movies_alias` to the `movies` index using _alias API
- Create multiple aliases using _aliases API:
  - remove movies_alias from `movies` index
  - add a `all_movies` alias to `comedy_movies` and `movies` indices

Exercise 4: Aliases Solution

PUT movies

POST movies/_alias/movies_alias

Lesson

Working with Documents

Working with Documents

- JSON document - basic unit of information
- Indexing: Single PUT/POST, Bulk indexing
- Reading/Retrieving: GET
  - Manipulate results/responses
    - Include/exclude fields
    - Wildcard fetching
- Delete
- Update/ Scripted Updates
- Upsert

Exercise 5: Document APIs

- Index a sample document
- Add an additional field using `_update` (clue: use "doc" object in the request body)

Delete all documents (caution, you'll lose your work!) using `delete_by_query` API

Search

Elasticsearch Search Features

- Term and Full-text queries
- Prefix searching
- Phrase matching
- Suggestions and completion suggesters
- More like this queries
- Geo queries (lat/lon, geo shapes)
- Highlighting
- Fuzzy searching
- Specialized queries ...

Search Context

- Filter context
  - Does the doc match to the criteria
  - Yes/No
  - Executed mostly on structured data like numbers, dates, ranges
- Query context
  - How well the document matches?
  - Scored results
  - Full/free text searching
  - Relevancy

Filter Contexts are Cached

Relevancy

Do you really need?!

- A score card!
- A number indicating how close the document matches the query
- Every document will be scored for a query
- Google ranks pages according to relevancy
- Searching “Virgin” or “Windows” - relevant results?

Similarity Algorithms

Fortunately we don't need to deal with these algos directly!

- Okapi BM25
- TF/IDF (Term Frequency / Inverse Document Frequency
- TF: Frequency of the search word in the document.
  - Higher the frequency, higher the weight
- IDF: Frequency of the word across all the documents (index). More frequent less weight
  - Higher the frequency - lower the weight
  - Uncommon words carry more weight

Search Types

Use Query DSL if possible!

- URL Search
  - `GET localhost:9200/_search?q=name:John`
  - light weight
  - Error prone, not a great experience
- Query DSL: JSON based payload

```
GET localhost:9200/people/_search
{
   "query":{
      "match":{"name":"John}
   }
}
```

Term-level Search

Filters on structured data

- No analysis process.
- Match exact words
- Dates, numbers, IDs etc
- Yes/No - Available or unavailable

Relational databases run these type of queries!

Types:

- term
- terms
- ids
- exists
- range
- prefix
- fuzzy
- wildcard

Term-level Examples

```
// TERM Query
GET books/_search
{
  "query": {
    "term": {
      "author.keyword": "Cay S. Horstmann"
    }
  }
}

// TERMS Query
GET books/_search
{
  "query": {
    "terms": {
      "author.keyword": [
        "Cay S. Horstmann",
        "Joshua Bloch"
      ]
    }
  }
}
```

```
// Range Query
GET books/_search
{
  "query": {
    "range": {
      "amazon_rating": {
        "gt": 4.5,
        "lt": 5
      }
    }
  }
}

// IDs Query
GET books/_search
{
  "query": {
    "ids": {
      "values": [1,3,5]
    }
  }
}
```

Full-text Search

Queries on unstructured data

- Analysis process same as indexing time
- Each doc stamped with score
- Relevancy is important

Types:

- match/match_all
- multi_match
- match_phrase
- match_phrase_prefix

Full-text Search Examples

```
// MATCH_ALL Query
GET books/_search
{
  "query": {
    "match_all": {}
  }
}
```

```
// MATCH Query
GET books/_search
{
  "query": {
    "match": {
      "author": "cay"
    }
  }
}
```

```
// MATCH_PHRASE Query
GET books/_search
{
  "query": {
    "match_phrase": {
      "synopsis": "simple language"
    }
  }
}
```

```
// Multi_MATCH with highlights Query
GET books/_search
{
"query": {
  "multi_match": {
    "query": "simple language",
    "fields": ["synopsis","title"]
  }
} ,"highlight": {
  "fields": {
    "synopsis": {}
  }
}
}
```

Poll

1) Filter queries produce no scores for the results

a) True
b) false

2) The _score in the query response represents

a) Term Frequency
b) Relevancy Score
c) Elasticsearch version

Compound Queries

- Combination of leaf queries
- Boolean query
  - Finds documents based on combined queries
  - Can have one or more boolean clauses:
    - must (AND)
    - should (OR)
    - must_not (NOT)
    - filter
  - must & should -> query context - produces score

Boolean Query

```
GET books/_search
{
  "query": {
    "bool": {
      "must": [
        {"match": {"author": "Joshua"}},
        {"match": {"title":"Java"}}
      ],
      "should": [
        {"range": {"amazon_rating": {"gt": 4}}},
        {"match": {"tags": "C"}}
      ],
      "must_not": {"term": {"edition": 3}}
    }
  }
}
```

Build very complex queries!

Exercise 6: Search

- Index a set of books using books.json
- Create a search query for fetching all the documents
- Create a term level search query for fetching `author.keyword` = 'Joshua' - does it return results? Why not?

Analytics

Top notch analytical features out of the box!

- Analytics is the other side of the coin!
- Grouping of data, rolling up
- Powerful and ultra fast
- Four categories of aggregations
  - Metrics:
    - sum, avg, max/min, sum, stats, top hits
  - Buckets:
    - histograms, terms, range aggs
  - Pipeline:
    - mov avg, cum sum, derivatives
  - Matrix:
    - matrix stats (experimental)

Metrics Aggs: avg, sum & stats

```
// Average book rating
GET books/_search
{
  "size": 0,
  "aggs": {
    "rating_aggs": {
      "avg": {"field": "amazon_rating"}
    }
  }
}
// Sum
      "sum": {"field": "price_usd"}
// Stats (multi-value)
      "stats": {"field": "orders"}
```

Buckets Aggs: Histogram

```
// Histogram of book rating
POST books/_search
{
  "size": 0,
  "aggs": {
    "rating_histogram": {
      "histogram": {
        "field": "amazon_rating",
        "interval": 0.5
      }
    }
  }
}
```

Buckets Aggs: Terms

```
## bucket aggregation - terms
POST books/_search
{
  "size": 0,
  "aggs": {
    "top_terms": {
      "terms": {
        "field": "tags.keyword",
        "size": 10
      }
    }
  }
}
```

Exercise 7: Aggs

- Index set of books
- Fetch average of ratings
- Create a query for a histogram for rating field with frequency as 1
- Create a query for stats on amazon_rating field

Visualizations

- Web application on top of Elasticsearch
- In depth analytics on your data
- Visualizations
  - Dashboards with Pie charts, Histograms, Heatmaps, Tag clouds.
- Cluster management
- Wide variety of use cases : performance, debug logs, anomaly detection (ML), security threat detection

Thank you!

Notes, Exercises, Questions and code is available on:

https://github.com/madhusudhankonda/elasticsearch-first-steps