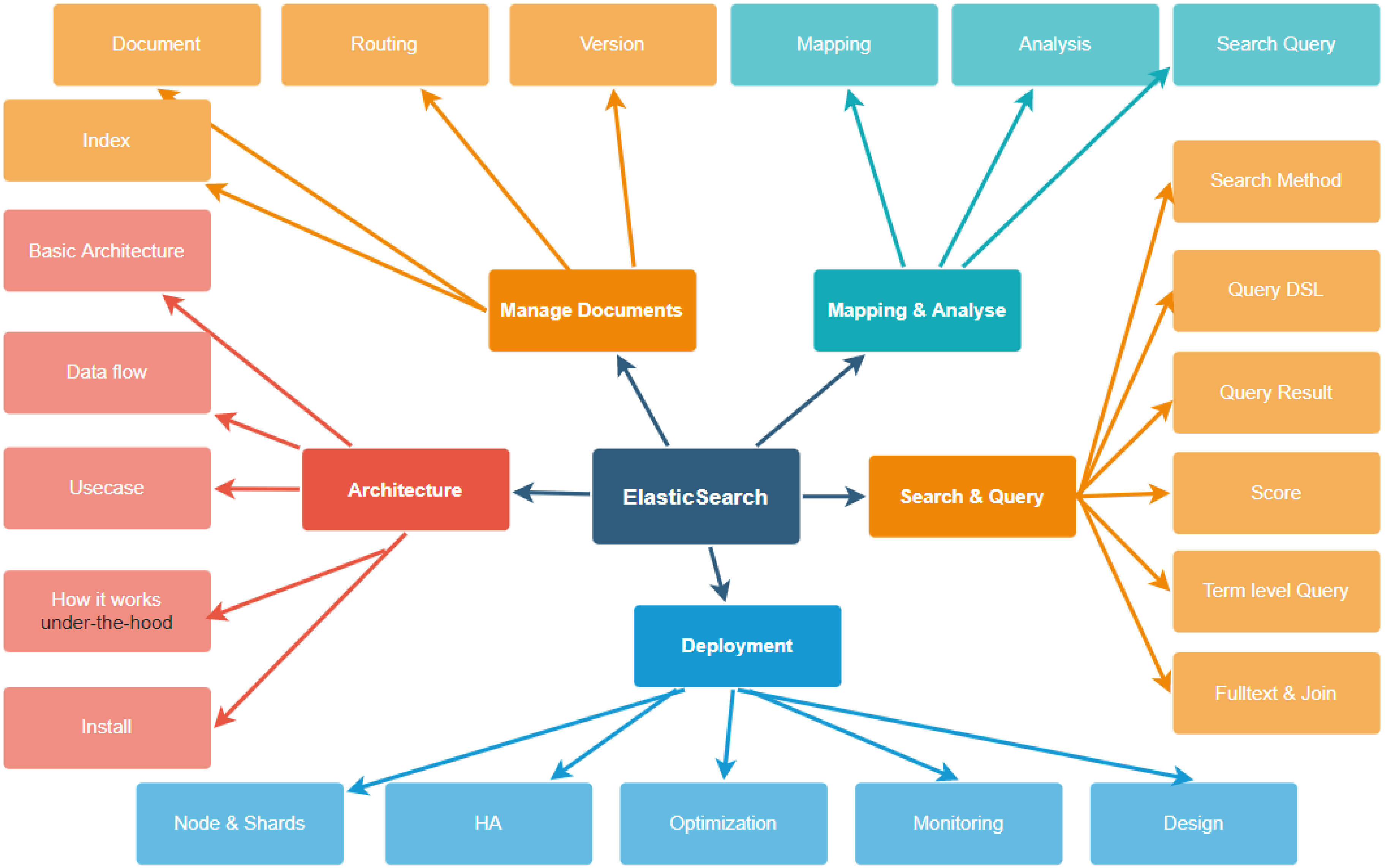
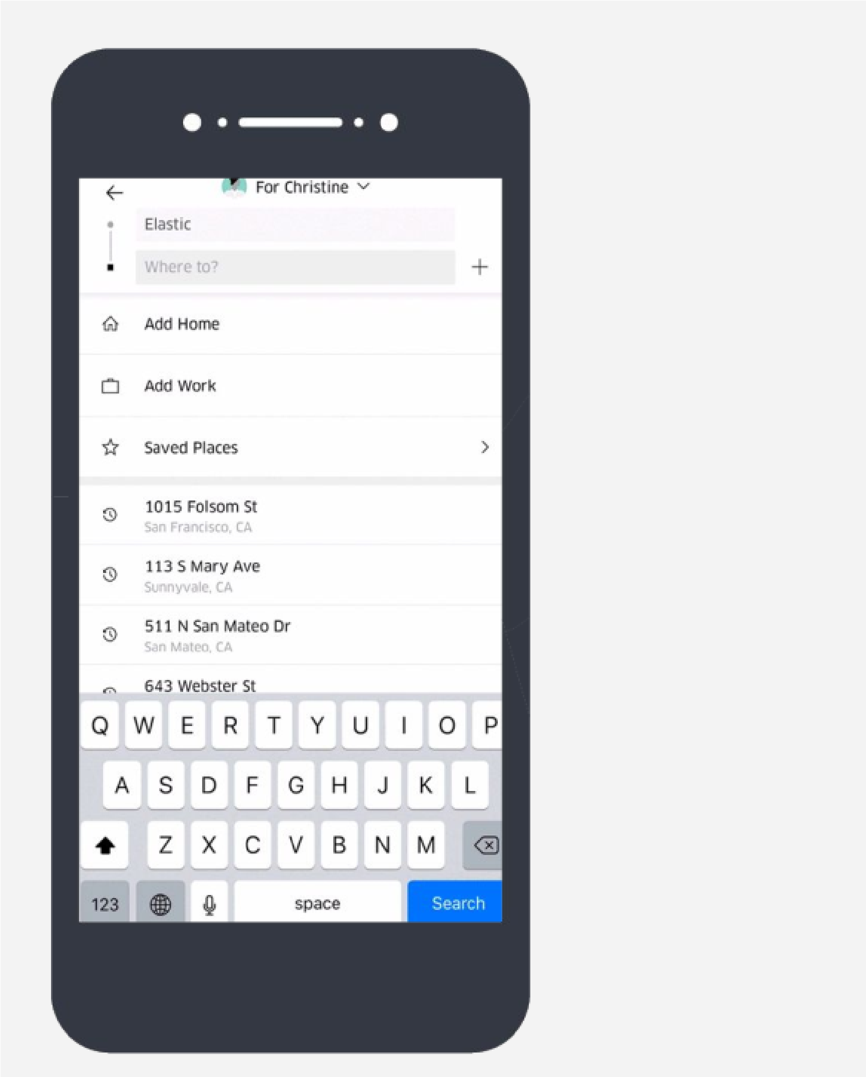
ElasticSearch

Techmaster

Nội dung


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

Introduction



For Christine

Elastic

Where to?

Add Home

Add Work

Saved Places

1015 Folsom St
San Francisco, CA

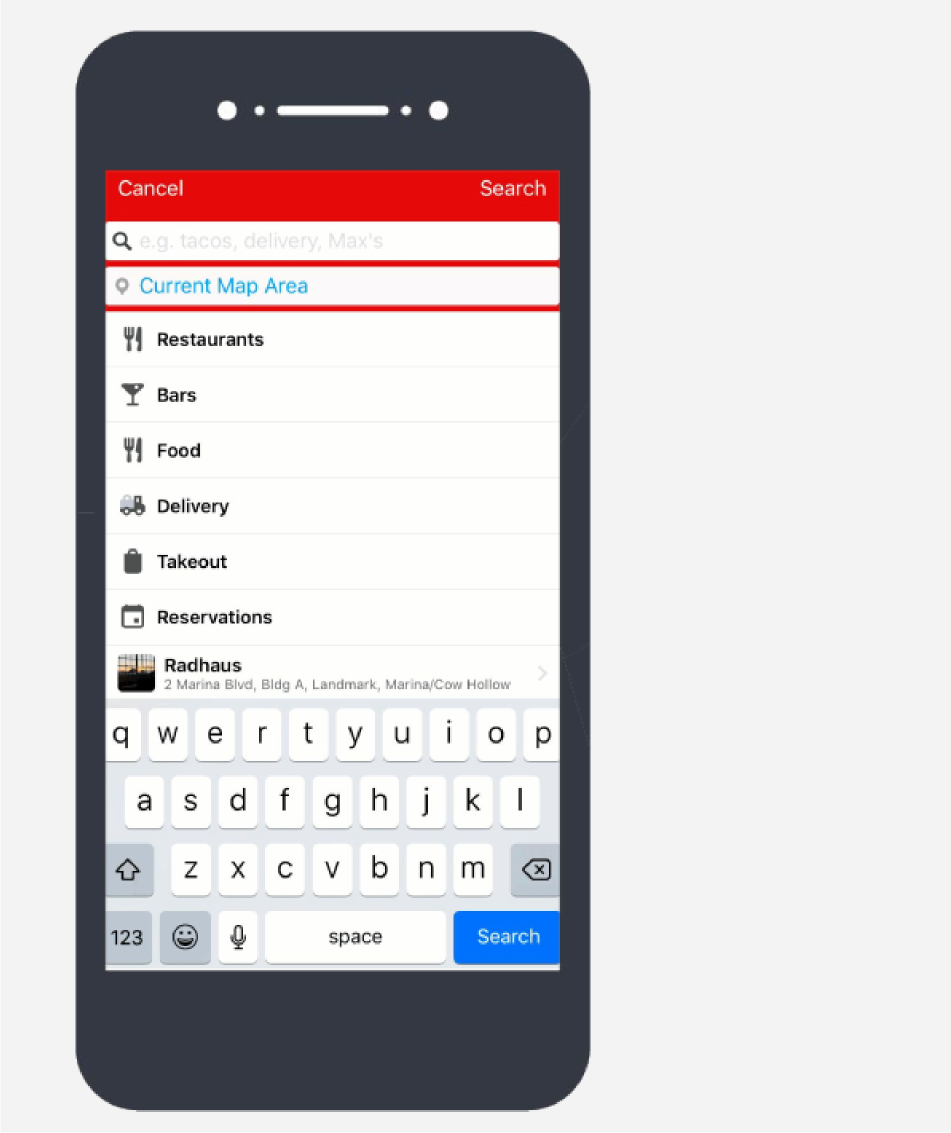
113 S Mary Ave
Sunnyvale, CA

511 N San Mateo Dr
San Mateo, CA

643 Webster St

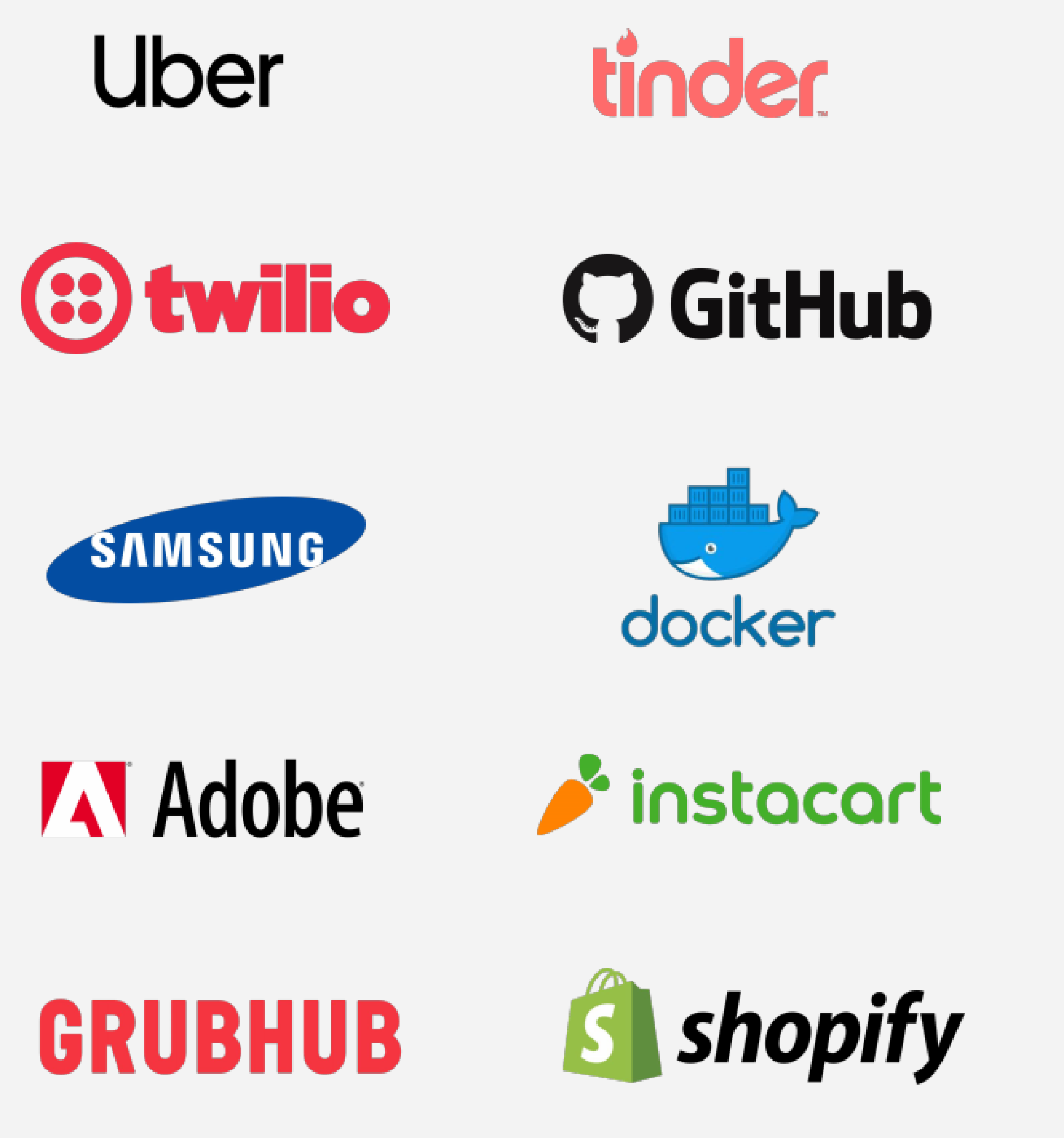
Searching for Rides

Introduction



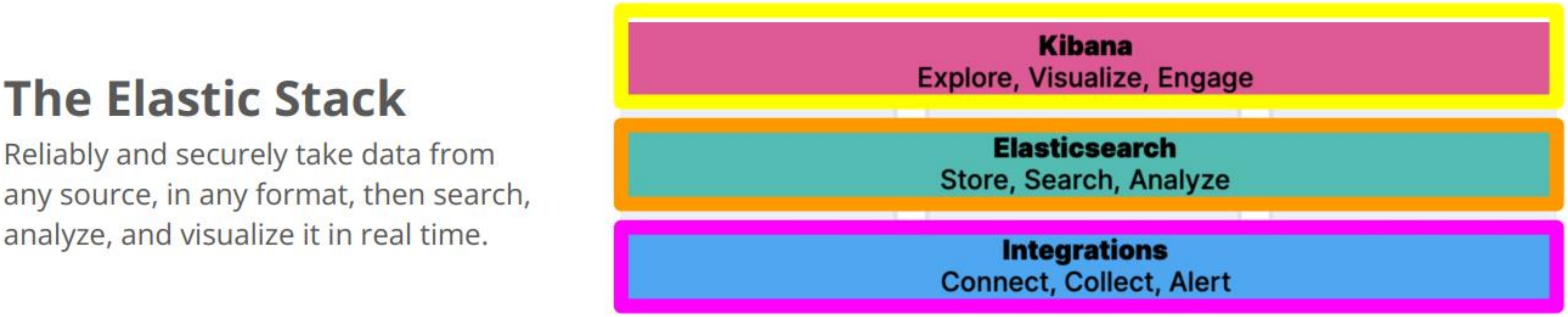
Searching for Restaurants

Introduction



Searching for
Rides

Introduction



The Elastic Stack

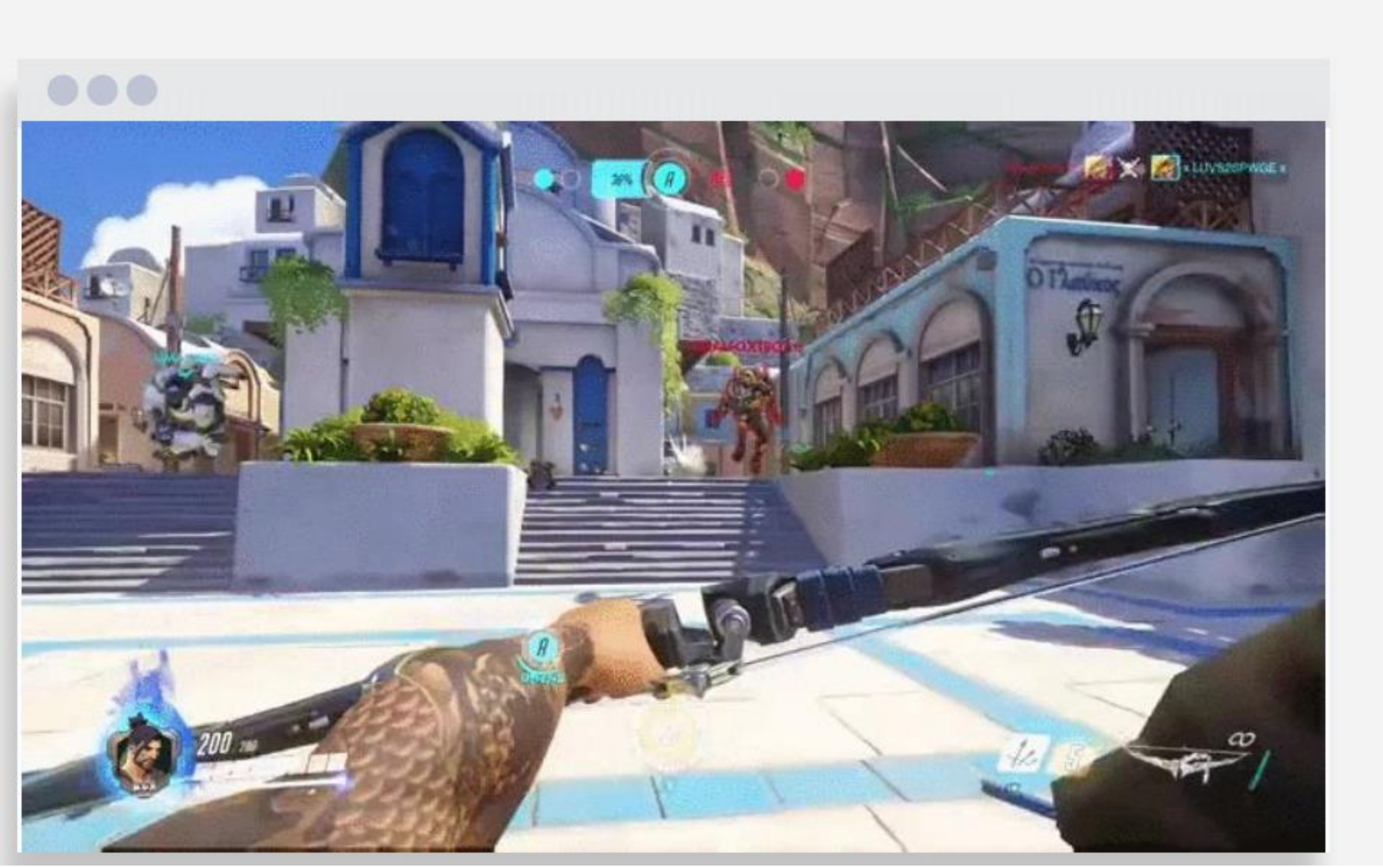
Reliably and securely take data from any source, in any format, then search, analyze, and visualize it in real time.

Kibana
Explore, Visualize, Engage

Elasticsearch
Store, Search, Analyze

Integrations
Connect, Collect, Alert

Use case: Logging

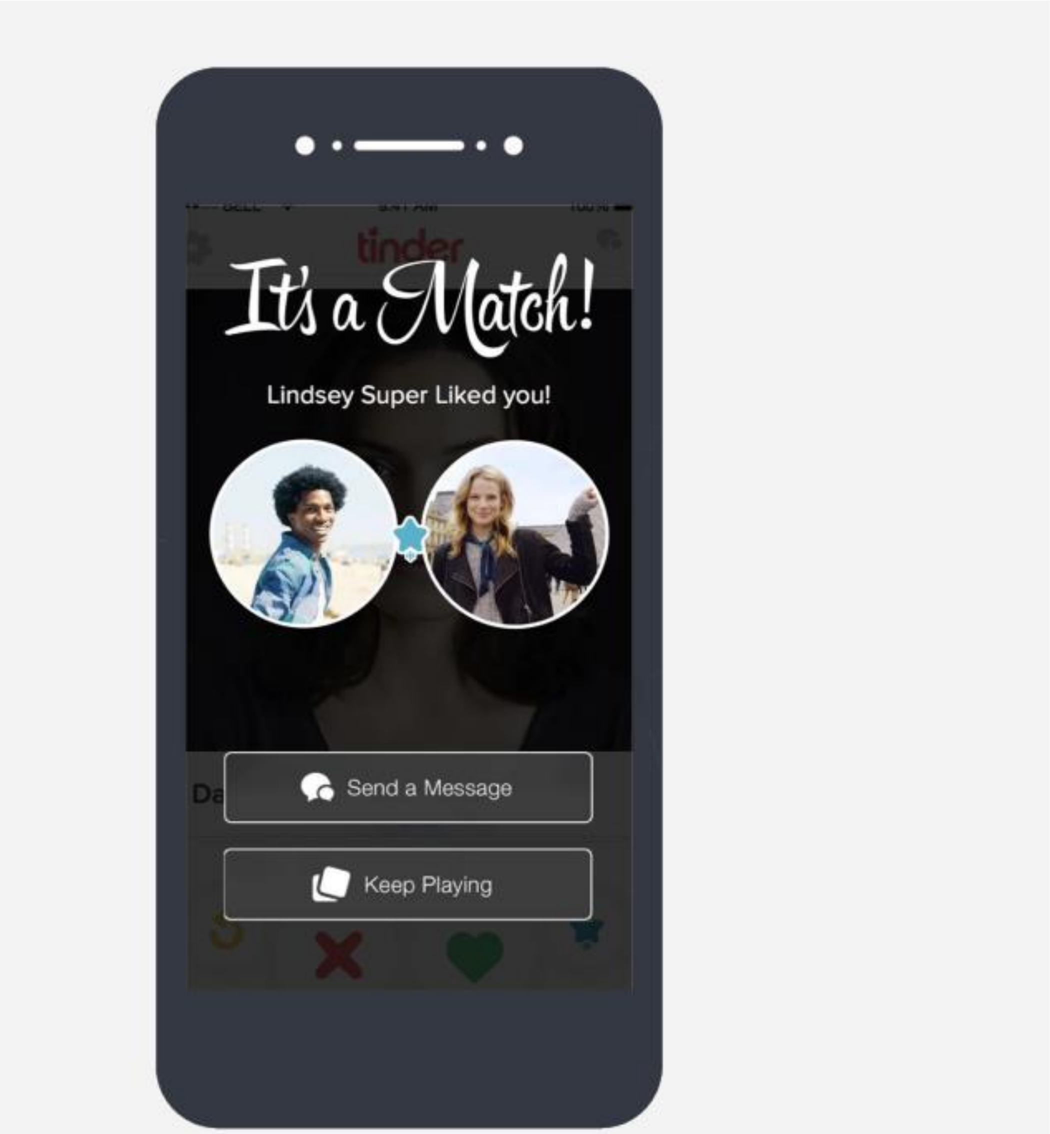


ACTIVISION BLIZZARD

Use case: Metrics

Use case: Security Analytics

Use case: Business Analytics

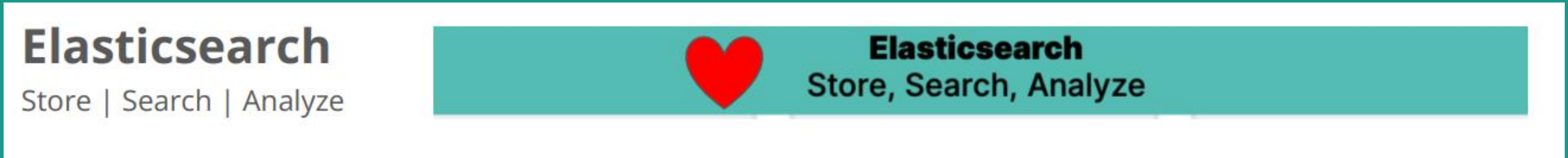


It's a Match!

Lindsey Super Liked you!

tinder™

ElasticSearch



Elasticsearch

Store | Search | Analyze

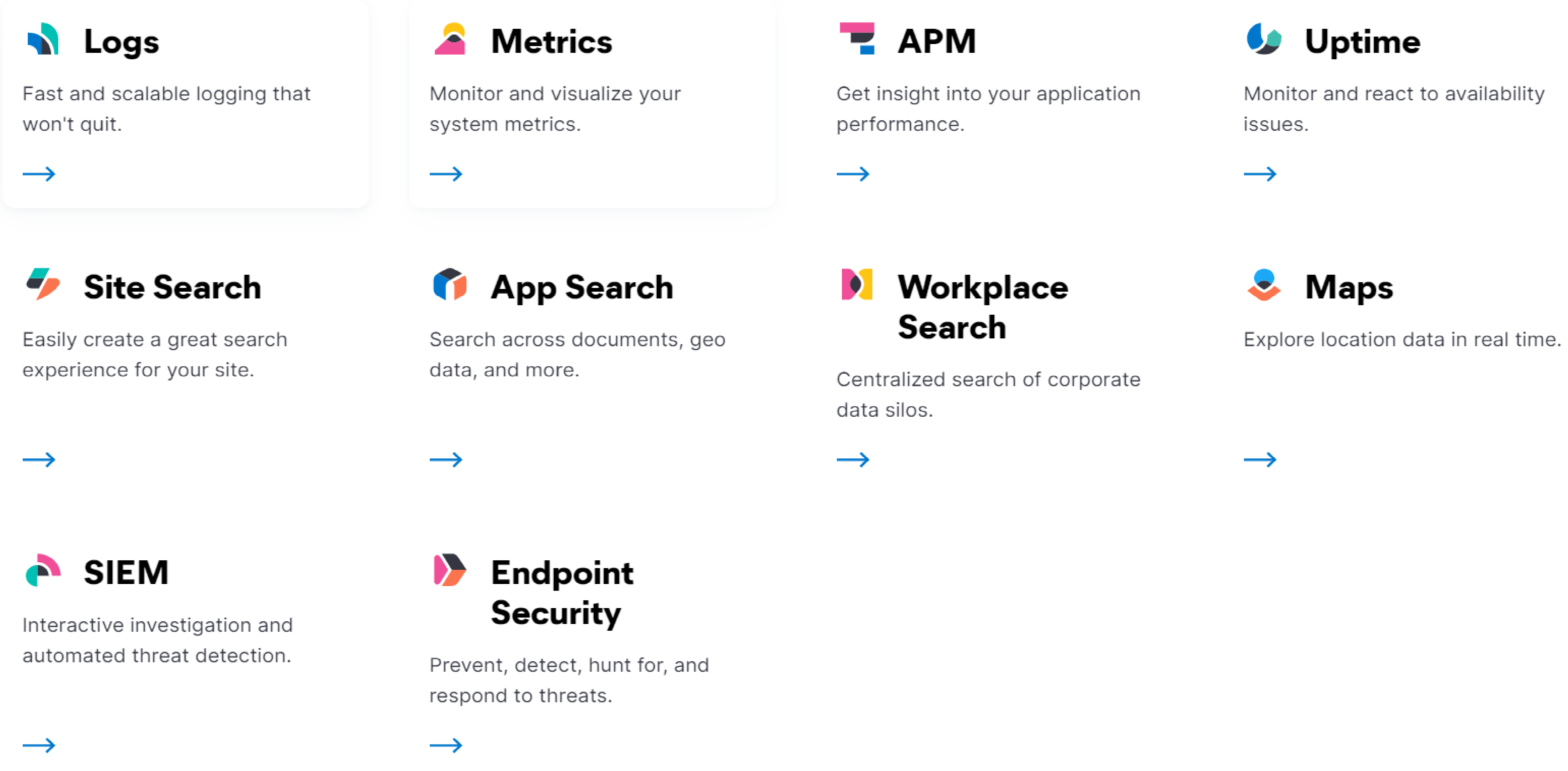
Elasticsearch

Store, Search, Analyze

ElasticSearch

- Elasticsearch is a **near real-time** distributed search and analytics engine with high availability.
- It is used for full-text search, structured search, analytics, or all three in combination.
- It is built on top of the Apache Lucene library.
- It is a schema-free, document-oriented data store.

Usecase



Logs

Fast and scalable logging that won't quit.

→

Metrics

Monitor and visualize your system metrics.

→

APM

Get insight into your application performance.

→

Uptime

Monitor and react to availability issues.

→

Site Search

Easily create a great search experience for your site.

→

App Search

Search across documents, geo data, and more.

→

Workplace Search

Centralized search of corporate data silos.

→

Maps

Explore location data in real time.

→

SIEM

Interactive investigation and automated threat detection.

→

Endpoint Security

Prevent, detect, hunt for, and respond to threats.

→

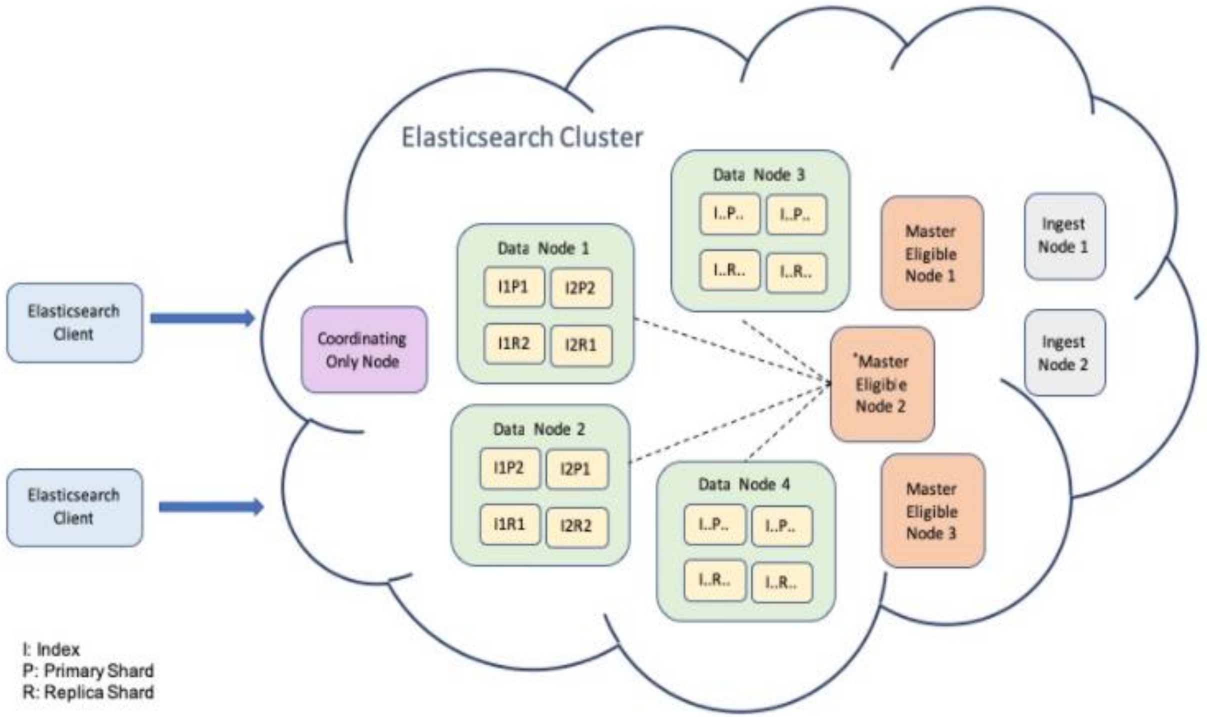
Basic Architecture

Cluster

- An **Elasticsearch cluster** is a group of one or more Elasticsearch nodes that are connected together

- **Nodes** in an Elasticsearch cluster are connected to each other, and each node contains a small chunk of cluster data.

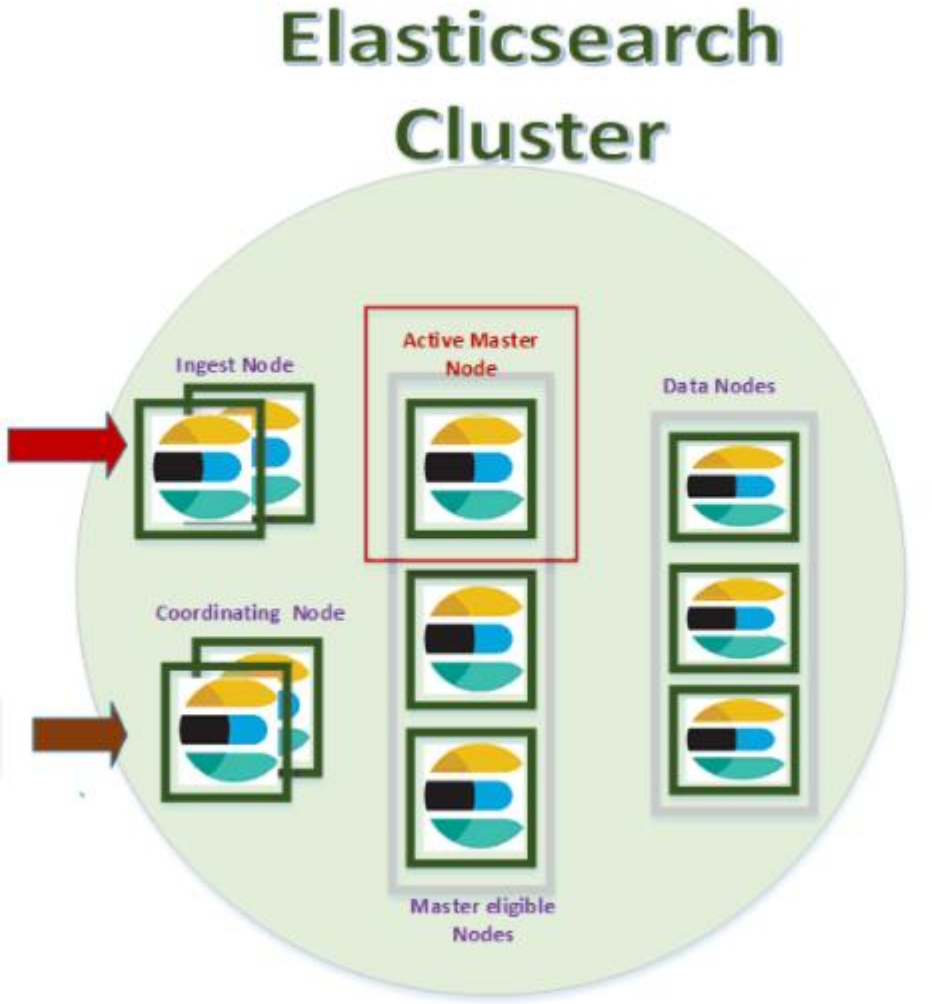
- The nodes participate in the overall cluster processes in charge of searching and indexing



Node

- Node is an instance, not a machine. You can run multiple nodes on a single machine

- Node Type
  - **Master node**: controls the ES cluster
  - **Data node:** contains data and the inverted index.
  - **Coordinating node:** as a load balancer
  - **Ingest node:** transform and enrich the document before indexing

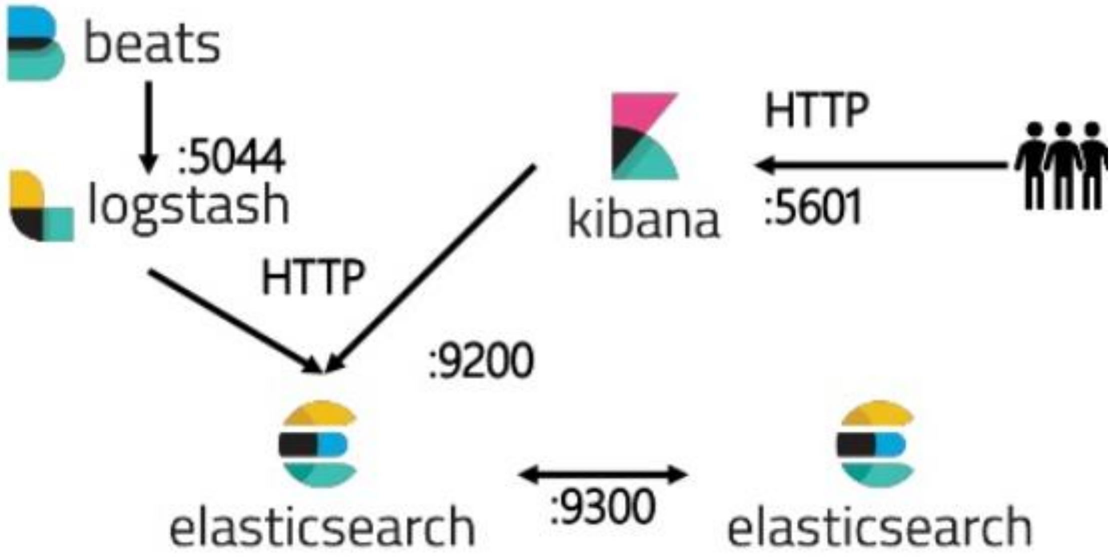


Node port

- **Port 9200**: used to filter requests coming from outside the cluster. This process meets requests coming through the REST APIs used for querying, indexing, and more.

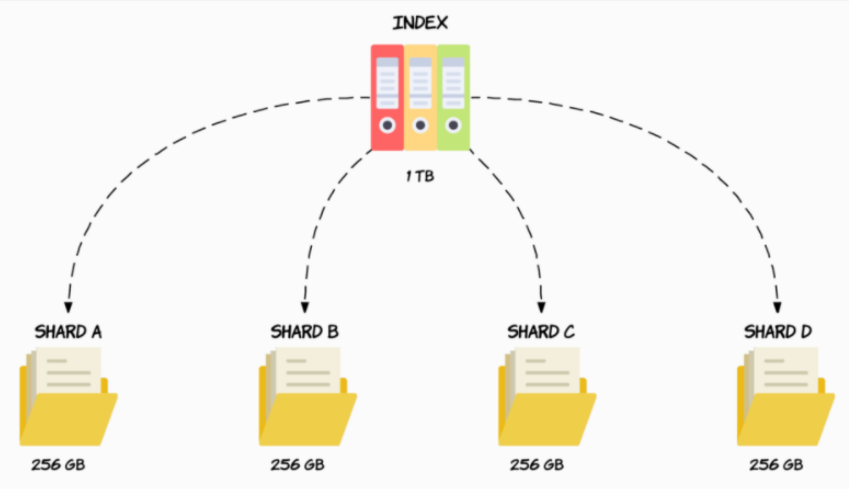
- **Port 9300**: used for inter-node communication. This occurs in the transport layer

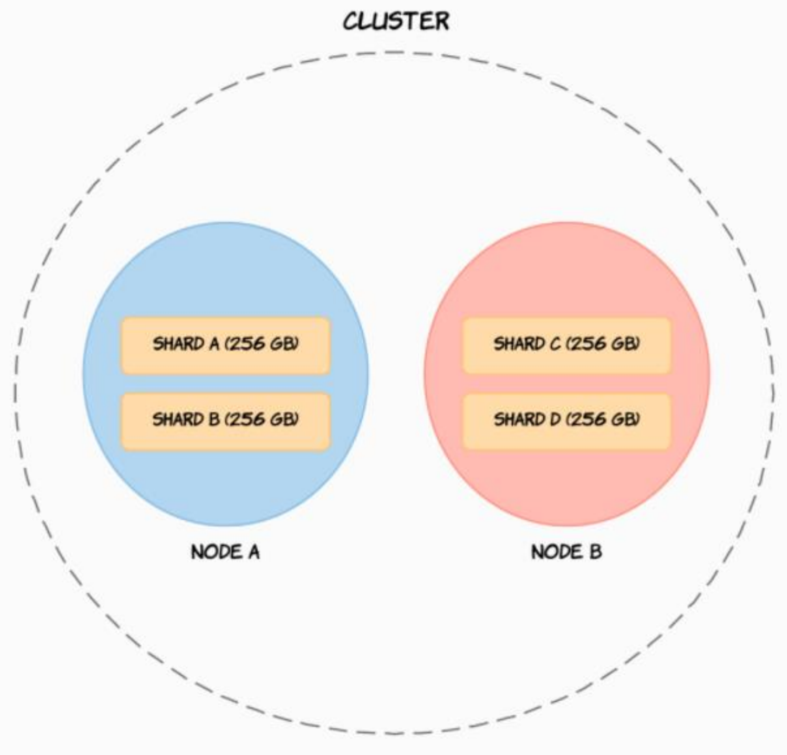
- Why does elasticsearch use 2 different ports?



Shard

- Elasticsearch is extremely scalable due to its **distributed architecture**. One of the reasons this is the case, is due to something called sharding.
- Where an the size of an index exceeds the **hardware limits** of a single node, sharding comes to the rescue
- **A shard** will contain a subset of an index' data and is in itself fully functional and independent, and you can kind of think of a shard as an "**independent index.**"
- Question: number shards per index, shard size ???? (insert, query, disaster ???)

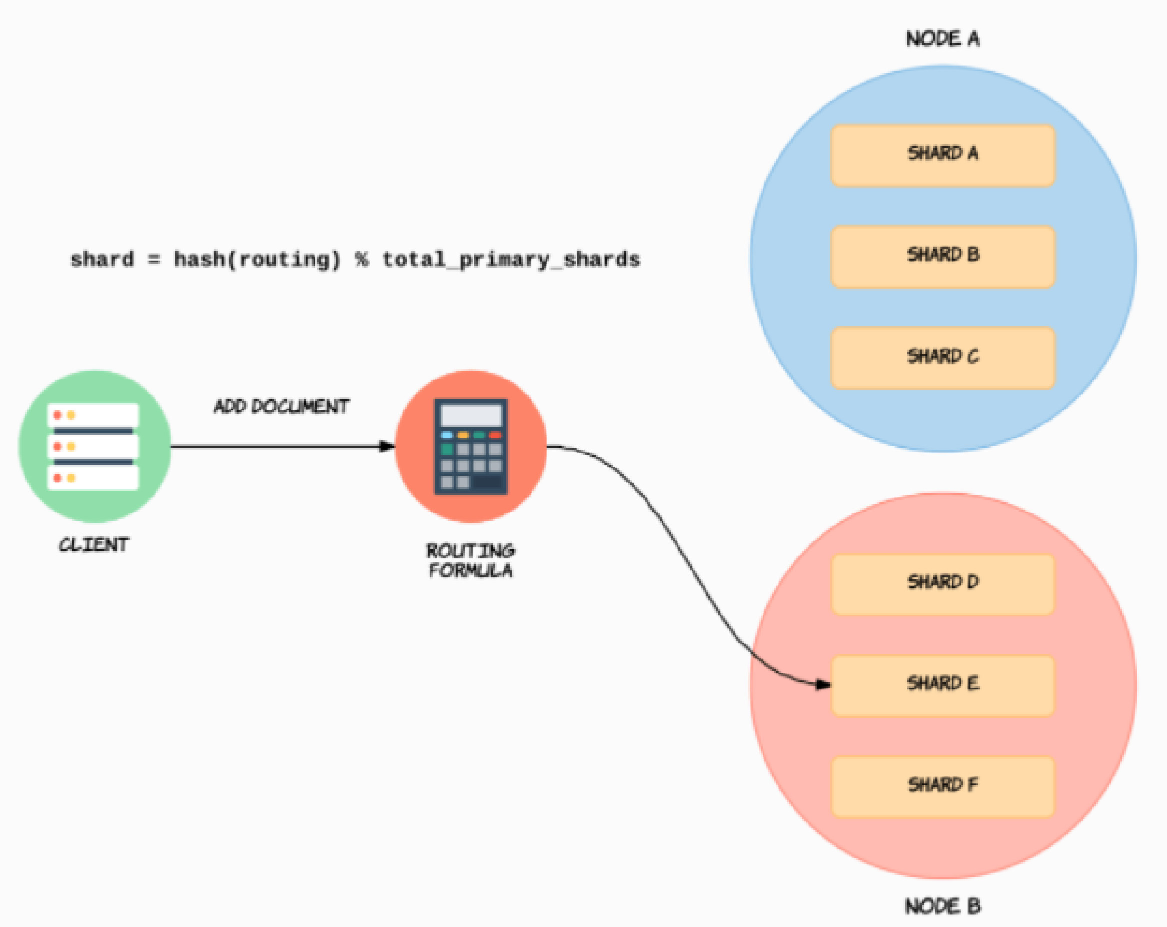




Distributing Documents across Shards (Routing)

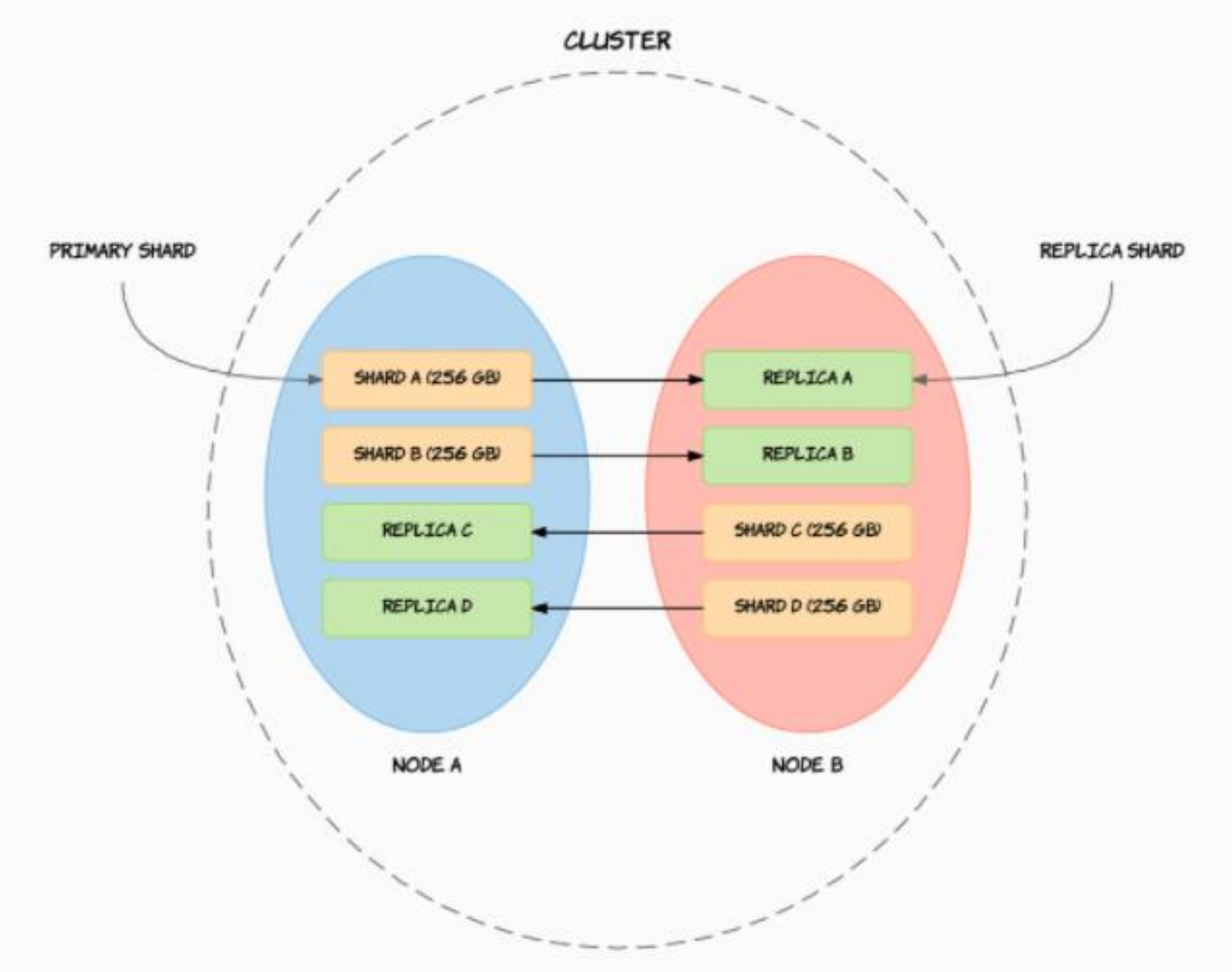
- The “routing” value will equal a given document’s ID. This value is then passed through a hashing function, which generates a number that can be used for the division. The remainder of dividing the generated number with the number of primary shards in the index, will give the shard number. This is how ES determines the location of specific documents.

- When executing search queries (i.e. not looking a specific document up by ID), the process is different, as the query is then broadcasted to all shards



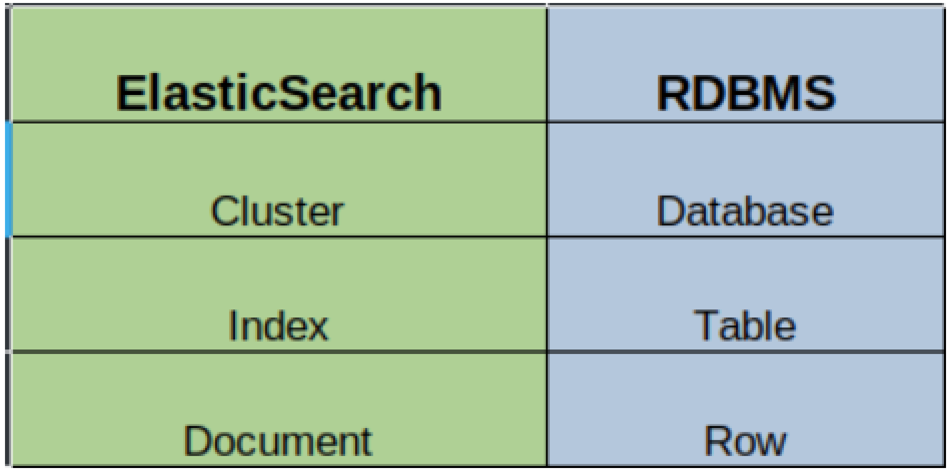
Replication

- Replica set is a copy of primary shard
- Replication purposes
  - Ensure high availability
  - Increased query throughput
- How is replication work and how to design replica ?



Rdbms vs Elasticsearch

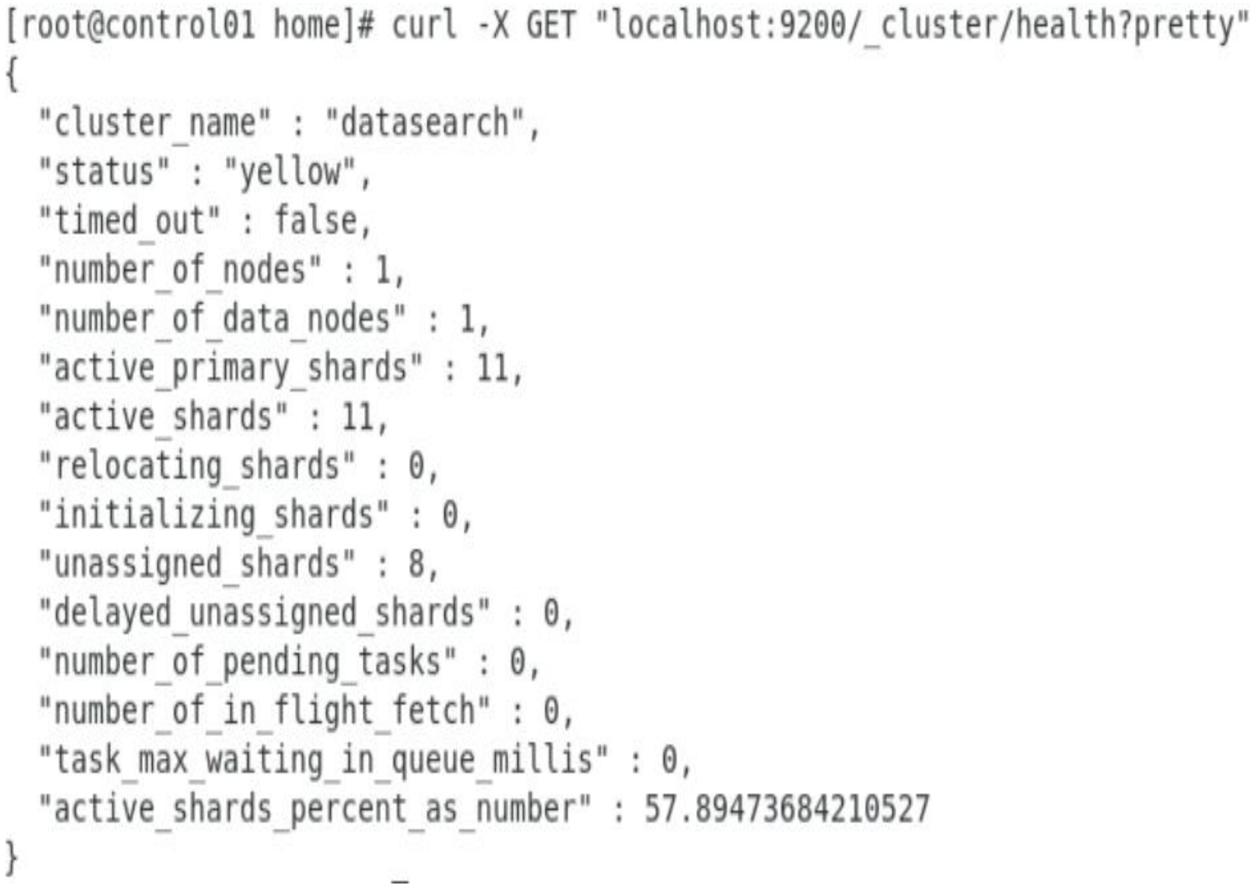
- While SQL and Elasticsearch have different terms for the way the data is organized (and different semantics), essentially their purpose is the same.



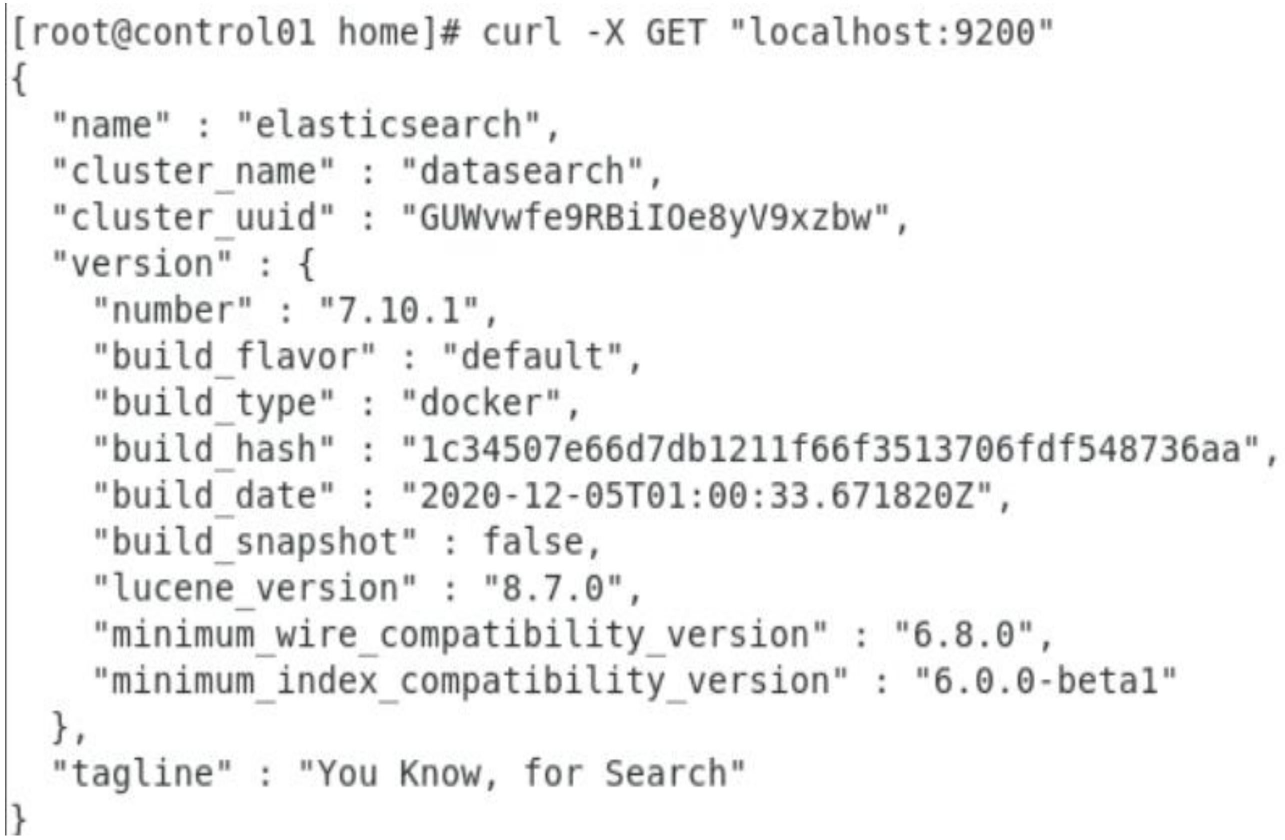
ElasticSearch	RDBMS
Cluster	Database
Index	Table
Document	Row

Inspecting Cluster

- The “curl”



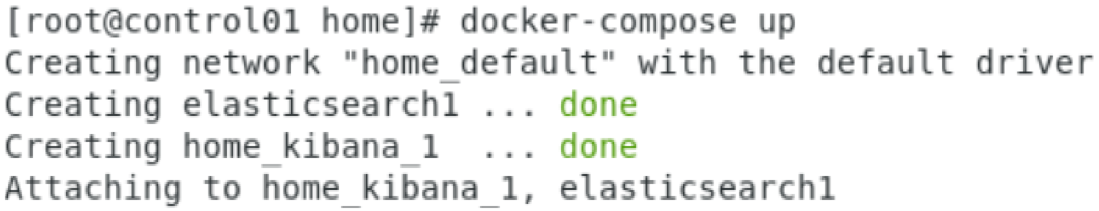
```
[root@control01 home]# curl -X GET "localhost:9200/_cluster/health?pretty"
{
  "cluster_name" : "datasearch",
  "status" : "yellow",
  "timed_out" : false,
  "number_of_nodes" : 1,
  "number_of_data_nodes" : 1,
  "active_primary_shards" : 11,
  "active_shards" : 11,
  "relocating_shards" : 0,
  "initializing_shards" : 0,
  "unassigned_shards" : 8,
  "delayed_unassigned_shards" : 0,
  "number_of_pending_tasks" : 0,
  "number_of_in_flight_fetch" : 0,
  "task_max_waiting_in_queue_millis" : 0,
  "active_shards_percent_as_number" : 57.89473684210527
}
```



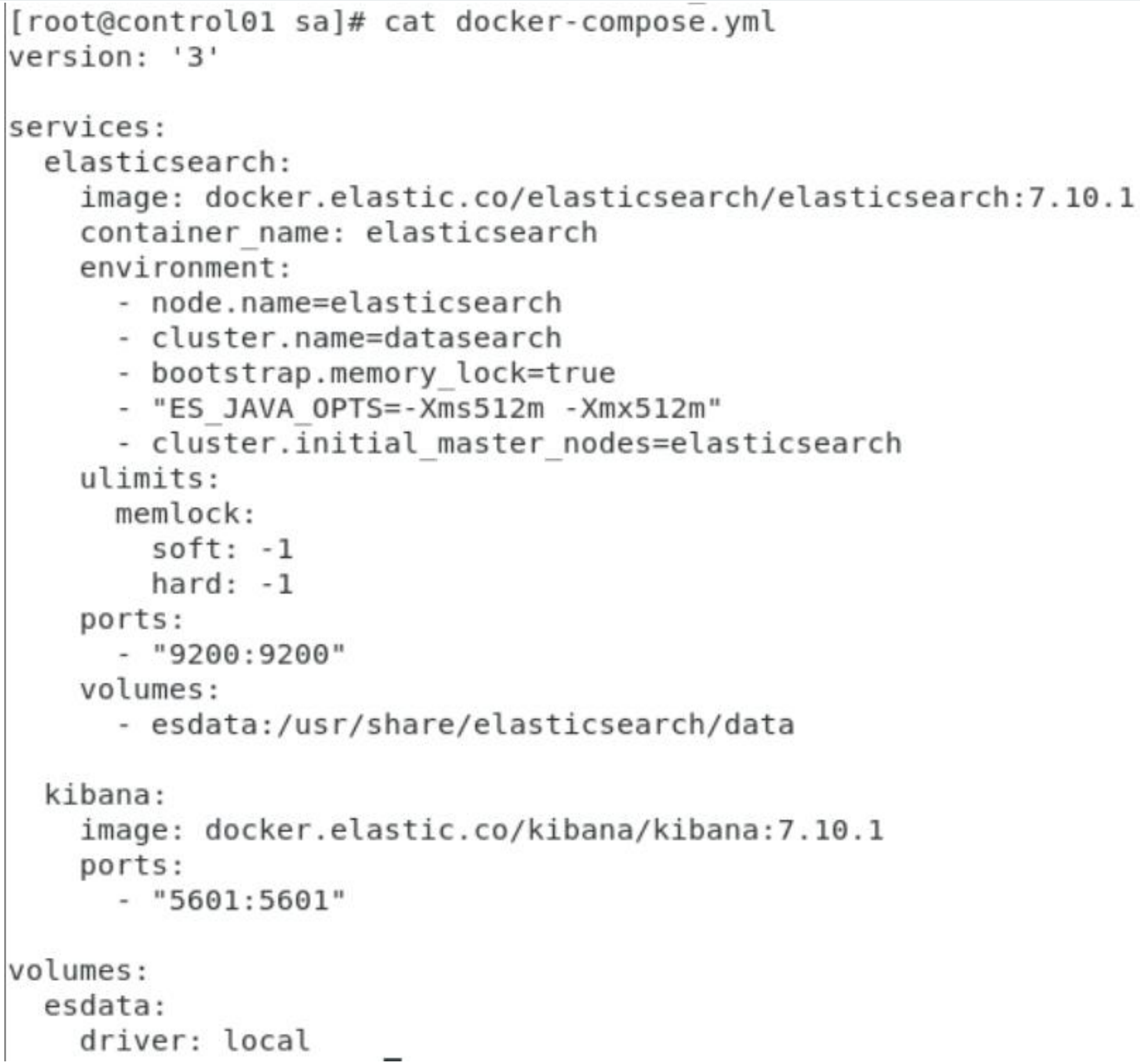
```
[root@control01 home]# curl -X GET "localhost:9200"
{
  "name" : "elasticsearch",
  "cluster_name" : "datasearch",
  "cluster_uuid" : "GUWvwfe9RBiIOe8yV9xzbw",
  "version" : {
    "number" : "7.10.1",
    "build_flavor" : "default",
    "build_type" : "docker",
    "build_hash" : "1c34507e66d7db1211f66f3513706fdf548736aa",
    "build_date" : "2020-12-05T01:00:33.671820Z",
    "build_snapshot" : false,
    "lucene_version" : "8.7.0",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

Installation

- ➢ docker-compose up



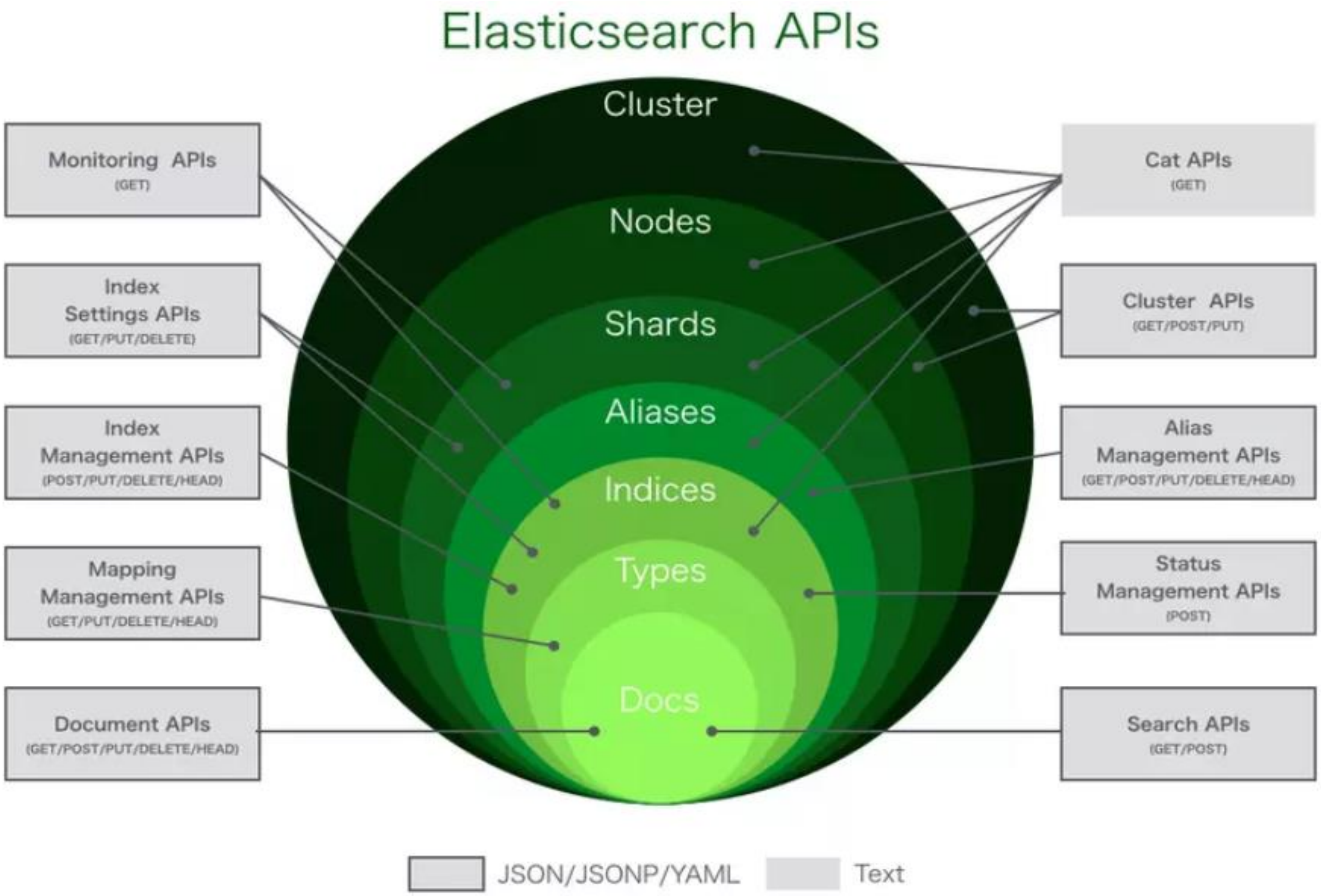
```
[root@control01 home]# docker-compose up
Creating network "home_default" with the default driver
Creating elasticsearch1 ... done
Creating home_kibana_1  ... done
Attaching to home_kibana_1, elasticsearch1
```



```
[root@control01 sa]# cat docker-compose.yml
version: '3'

services:
  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:7.10.1
    container_name: elasticsearch
    environment:
      - node.name=elasticsearch
      - cluster.name=datasearch
      - bootstrap.memory_lock=true
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
      - cluster.initial_master_nodes=elasticsearch
    ulimits:
      memlock:
        soft: -1
        hard: -1
    ports:
      - "9200:9200"
    volumes:
      - esdata:/usr/share/elasticsearch/data

  kibana:
    image: docker.elastic.co/kibana/kibana:7.10.1
    ports:
      - "5601:5601"

volumes:
  esdata:
    driver: local
```

Manage Document

API


Elasticsearch APIs
Cluster
Nodes
Shards
Aliases
Indices
Types
Docs
Monitoring APIs
(GET)
Index Settings APIs
(GET/PUT/DELETE)
Index Management APIs
(POST/PUT/DELETE/HEAD)
Mapping Management APIs
(GET/PUT/DELETE/HEAD)
Document APIs
(GET/POST/PUT/DELETE/HEAD)
Cat APIs
(GET)
Cluster APIs
(GET/POST/PUT)
Alias Management APIs
(GET/POST/PUT/DELETE/HEAD)
Status Management APIs
(POST)
Search APIs
(GET/POST)
JSON/JSONP/YAML
Text

Index api

- In Elasticsearch, an index (plural: indices) contains a schema and can have one or more shards and replicas. An Elasticsearch index is divided into shards and each shard is an instance of a Lucene index.

- Indices are used to store the documents in dedicated data structures corresponding to the data type of fields. For example, text fields are stored inside an inverted index whereas numeric and geo fields are stored inside BKD trees.

【index api】

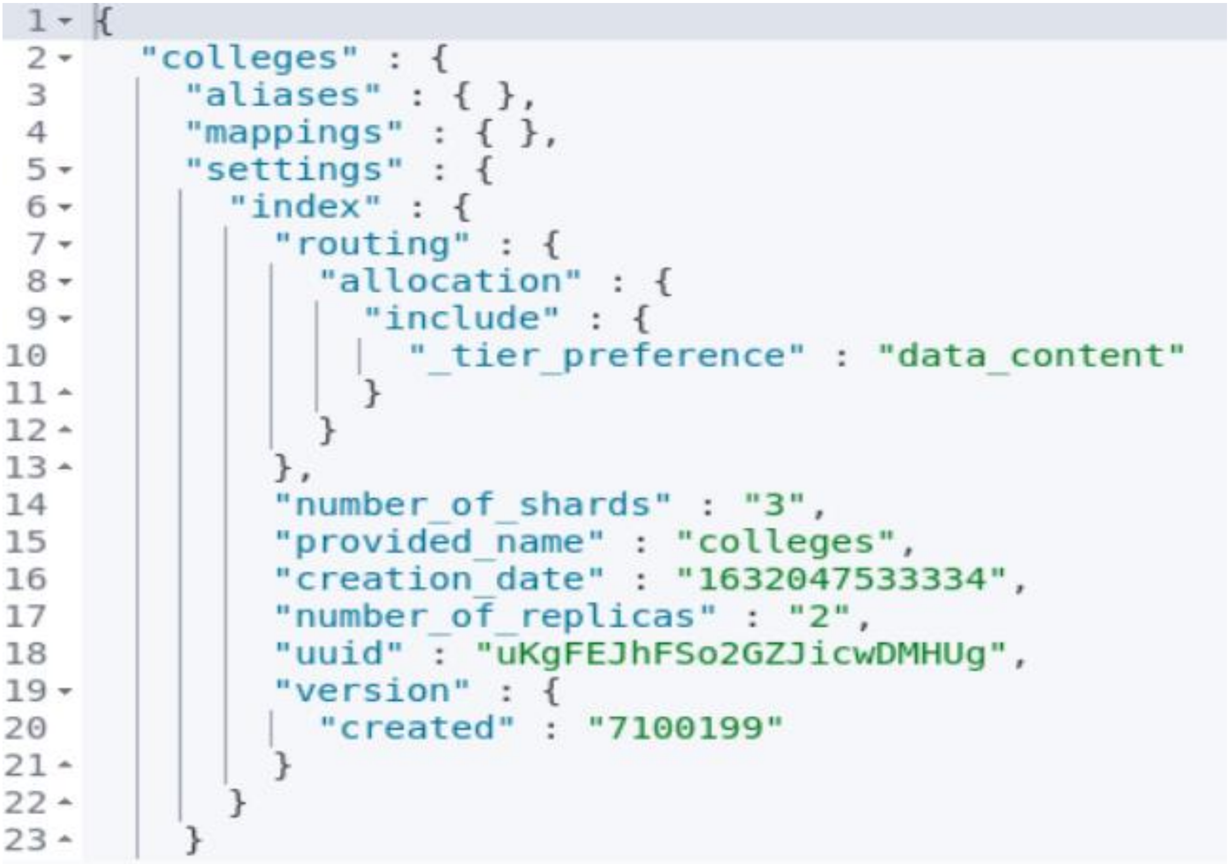
```
PUT /{index} # create index
DELETE /{index} # delete index
HEAD /{index} # confirm exist
index
POST /{index}/_close # block
write index POST /{index}/_open
# open write index
```

Index api

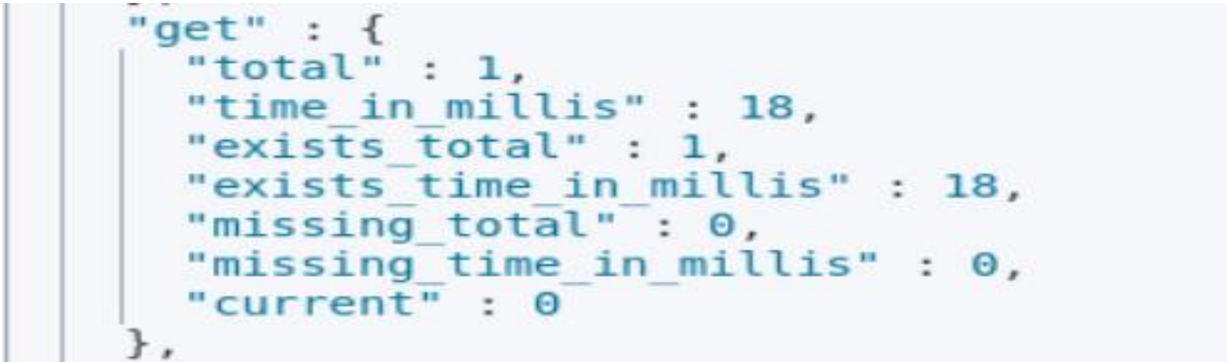
```
PUT colleges
{
  "settings" : {
     "index" : {
        "number_of_shards" : 3,
        "number_of_replicas" : 2
     }
  }
}
GET colleges
HEAD colleges
GET /colleges/_settings
POST colleges/_flush
DELETE /colleges
GET /colleges/_stats/
```

Flush

The flush process of an index makes sure that any data that is currently only persisted in the transaction log is also permanently persisted in Lucene. This reduces recovery times as that data does not need to be reindexed from the transaction logs after the Lucene indexed is opened.



```
{
  "colleges" : {
    "aliases" : { },
    "mappings" : { },
    "settings" : {
      "index" : {
        "routing" : {
          "allocation" : {
            "include" : {
              "_tier_preference" : "data_content"
            }
          }
        },
        "number_of_shards" : "3",
        "provided_name" : "colleges",
        "creation_date" : "1632047533334",
        "number_of_replicas" : "2",
        "uuid" : "uKgFEJhFSo2GZJicwDMHUg",
        "version" : {
          "created" : "7100199"
        }
      }
    }
  }
```



```
"get" : {
  "total" : 1,
  "time_in_millis" : 18,
  "exists_total" : 1,
  "exists_time_in_millis" : 18,
  "missing_total" : 0,
  "missing_time_in_millis" : 0,
  "current" : 0
},
```

Indexing document

- In Elasticsearch, an index (plural: indices) contains a schema and can have one or more shards and replicas. An Elasticsearch index is divided into shards and each shard is an instance of a Lucene index.

- Indices are used to store the documents in dedicated data structures corresponding to the data type of fields. For example, text fields are stored inside an inverted index whereas numeric and geo fields are stored inside BKD trees.

【index api】

```
PUT /{index} # create index
DELETE /{index} # delete index
HEAD /{index} # confirm exist
index
POST /{index}/_close # block
write index POST /{index}/_open
# open write index
POST /products/_doc
{
 "name" : "Coffee Maker" ,
 "price": 64,
 "in_stock": 10
}
```

Create document

```
POST /products/_doc
{
 "name" : "Coffee Maker" ,
 "price": 64,
 "in_stock": 10
}
```

```
PUT /products/_doc/101
{
 "name" : "Coffee Maker" ,
 "price": 101,
 "in_stock": 10
}
```

```
{
  "_index" : "products",
  "_type" : "_doc",
  "_id" : "q_OhHX0BHhtkMm9JUVTl",
  "_version" : 1,
  "result" : "created",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 0,
  "_primary_term" : 1
}
```

Retrieving document by ID

```
GET /products/_doc/100}
```

```
{
  "_index" : "products",
  "_type" : "_doc",
  "_id" : "101",
  "_version" : 1,
  "result" : "created",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 7,
  "_primary_term" : 1
}
```

Updating documents

```
POST /products/_update/100
{
  "doc": {
    "in_stock":3
  }
}
```

```
{
  "_index" : "products",
  "_type" : "_doc",
  "_id" : "100",
  "_version" : 7,
  "result" : "updated",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 8,
  "_primary_term" : 1
}
```

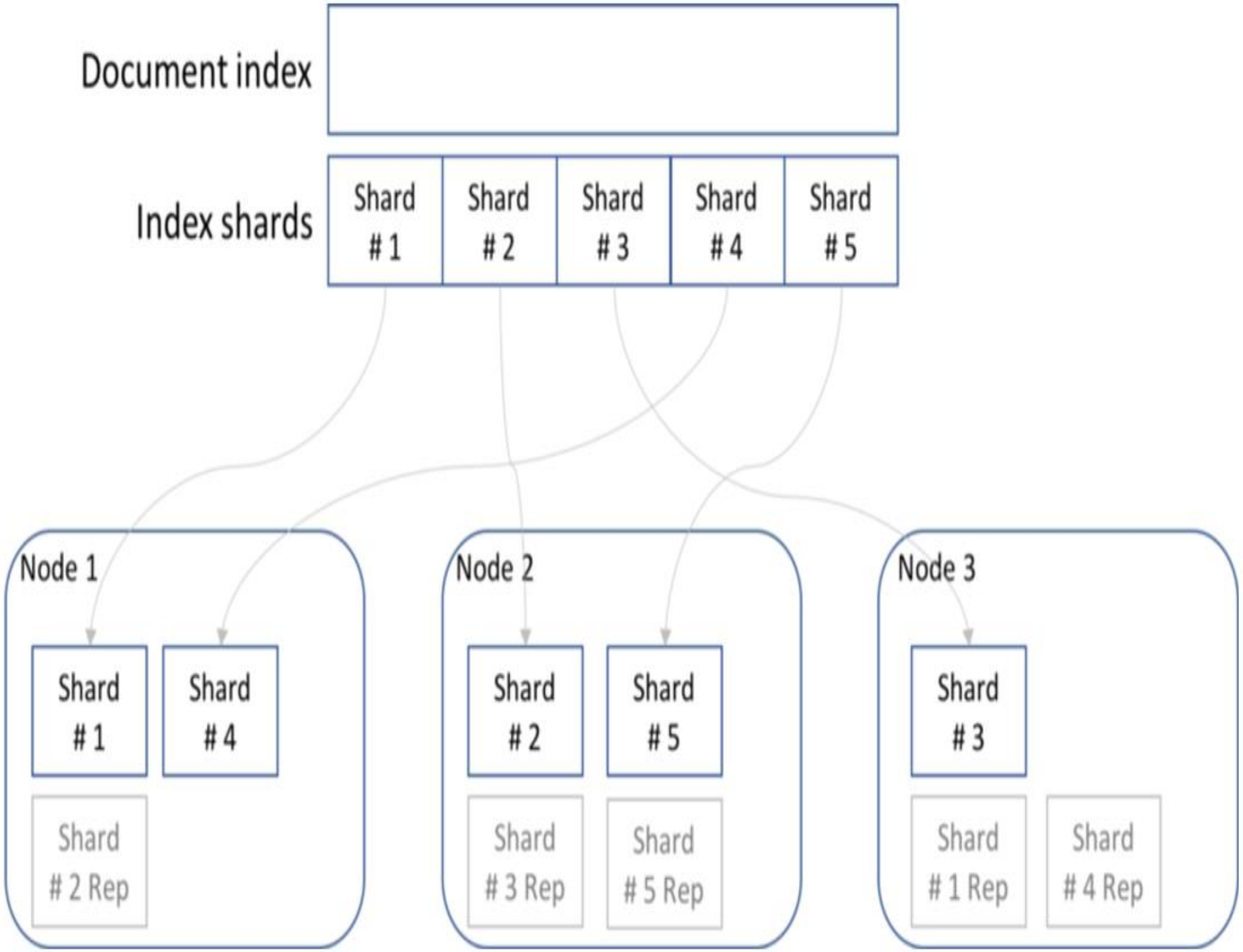
Script

```
POST /products/_update/100
{
  "script": {
    "source":"ctx._source.in_stock+=10"
  }
}
POST /products/_update/100
{
  "script": {

"source":"ctx._source.in_stock+=params.
quantity",
    "params": {
      "quantity":4
    }
  }
}
```

```
{
  "_index" : "products",
  "_type" : "_doc",
  "_id" : "100",
  "_version" : 10,
  "_seq_no" : 11,
  "_primary_term" : 1,
  "found" : true,
  "_source" : {
    "name" : "Coffee Maker",
    "price" : 67,
    "in_stock" : 20
  }
}
```

Shard map

- All the nodes in the Elasticsearch cluster contain metadata about which shard lives on which node.

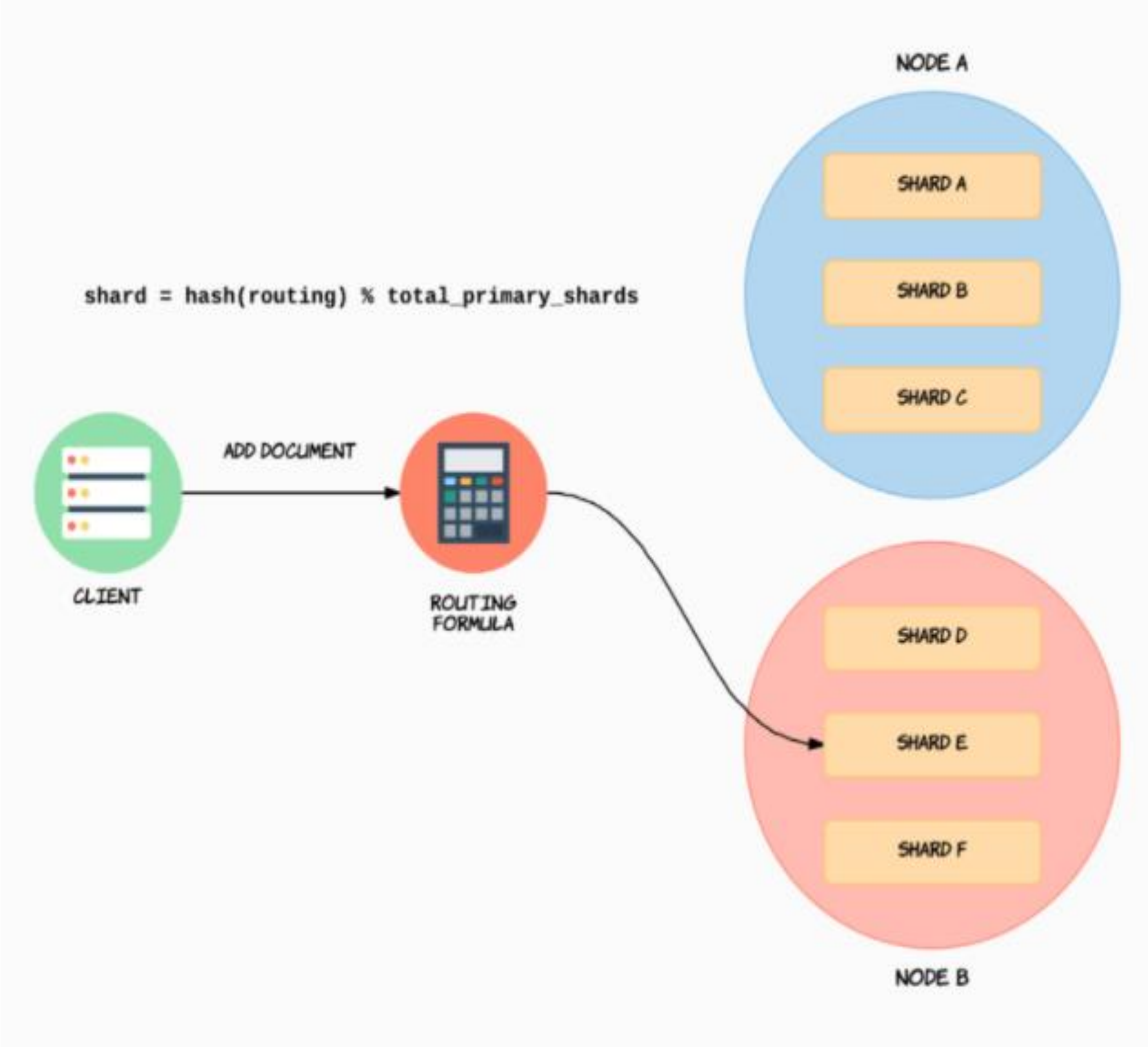


Understand routing

- How does Elasticsearch know where to store documents
- How are documents found once they have been indexed?
- The answer is routing
- Routing is process of resolving a shard for a document

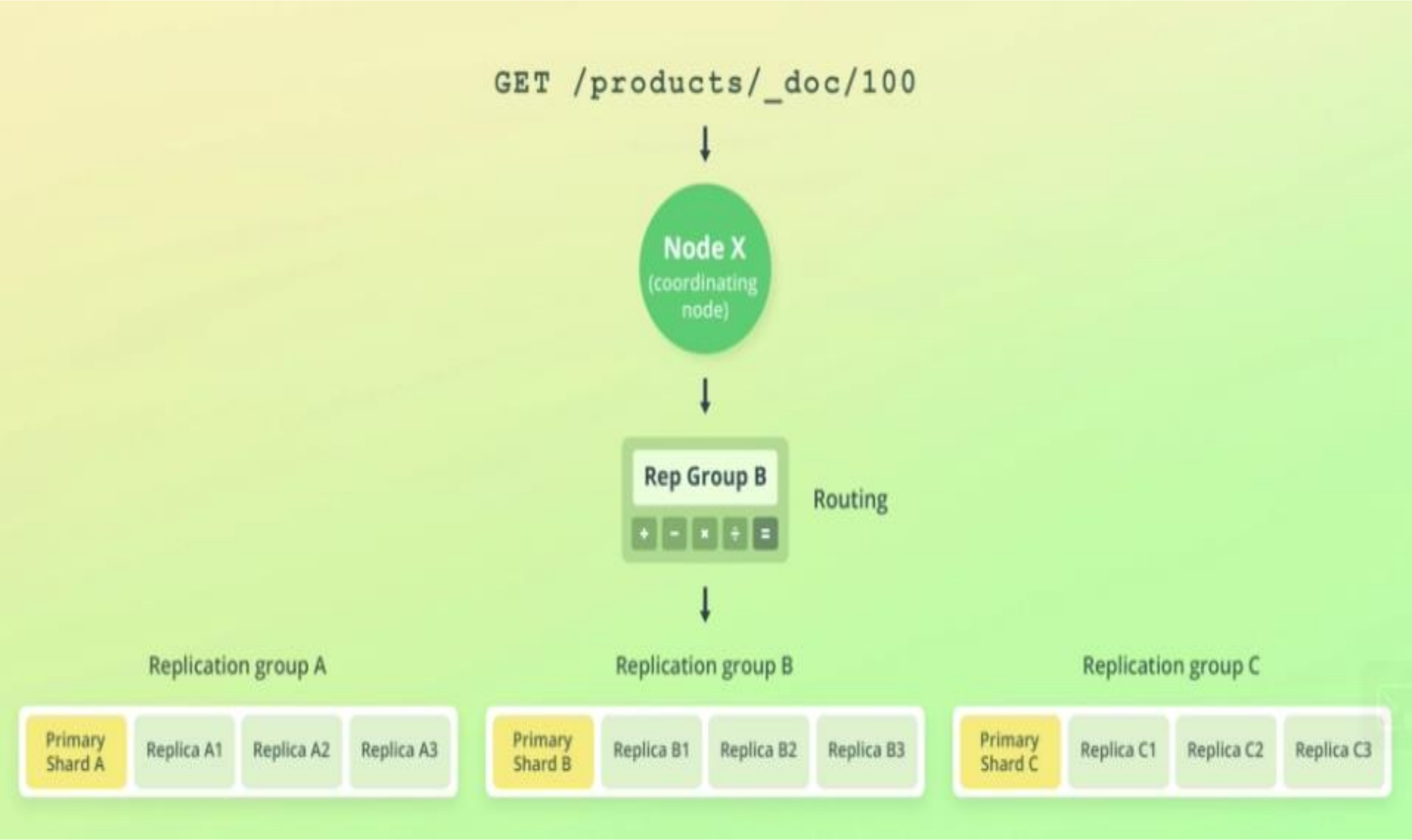
shard_num = hash(_routing)% num_primary_shards

Note: murmur3 hashing



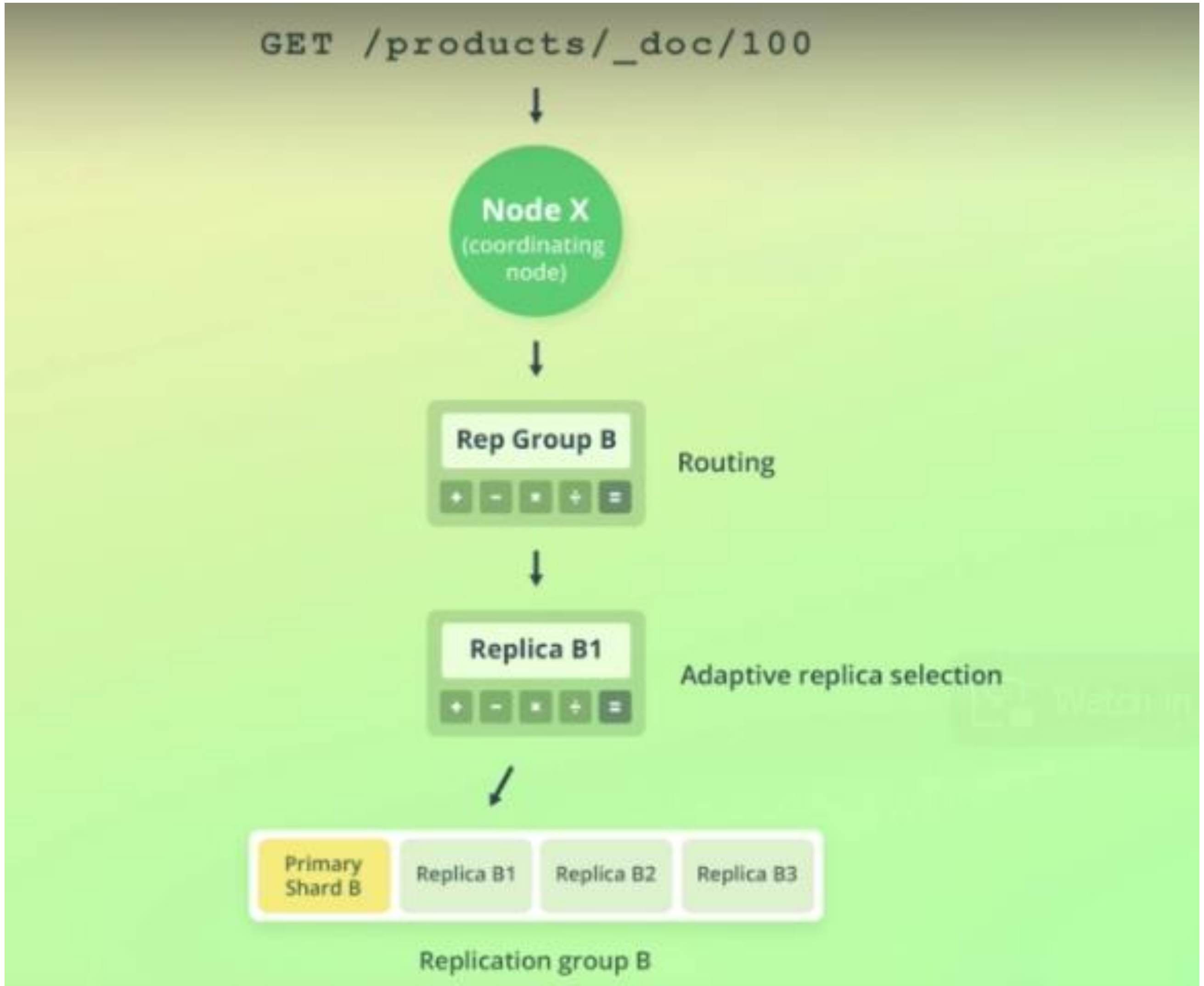
How ES reads data

- Step 1: Chose replica group



How ES reads data

- Step 2: Chose shard



Summary

- Request handled by a coordinating node
- ARS (Adaptive Replica Selection) helps reduce query response time
- Coordinating node collects the response and send to clients

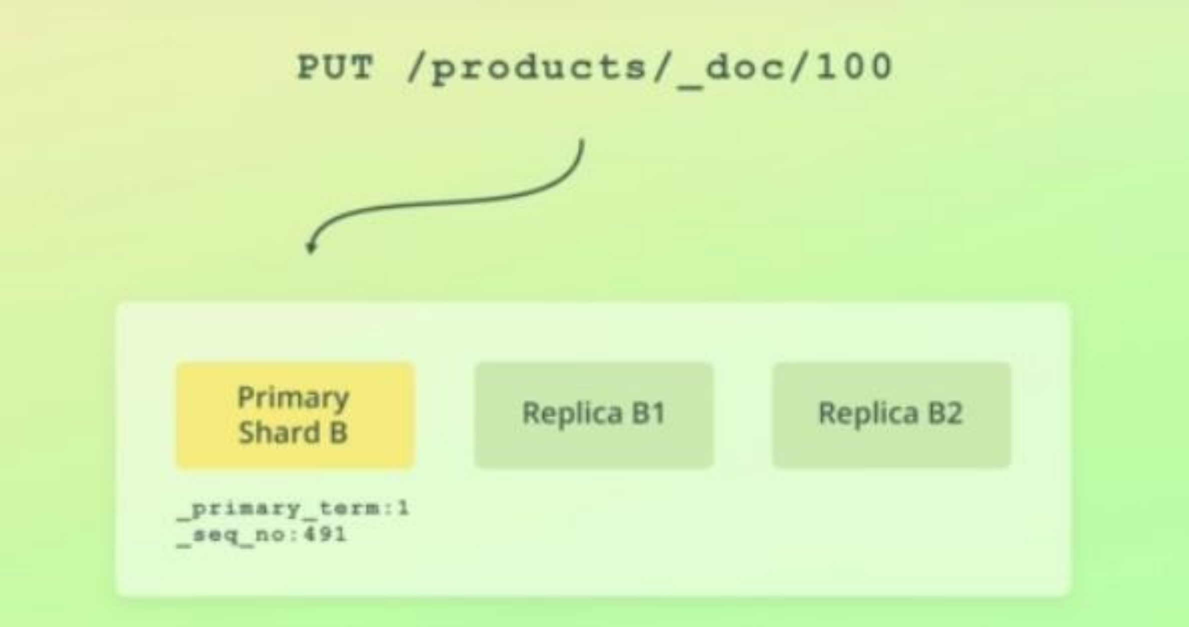
Primary term & Sequence Number

Primary term

- Distinguish old and new primary shards
- Count how many times primary shard has changed

Sequence number

- Incremented for each write operation
- Enable ES to order write operation

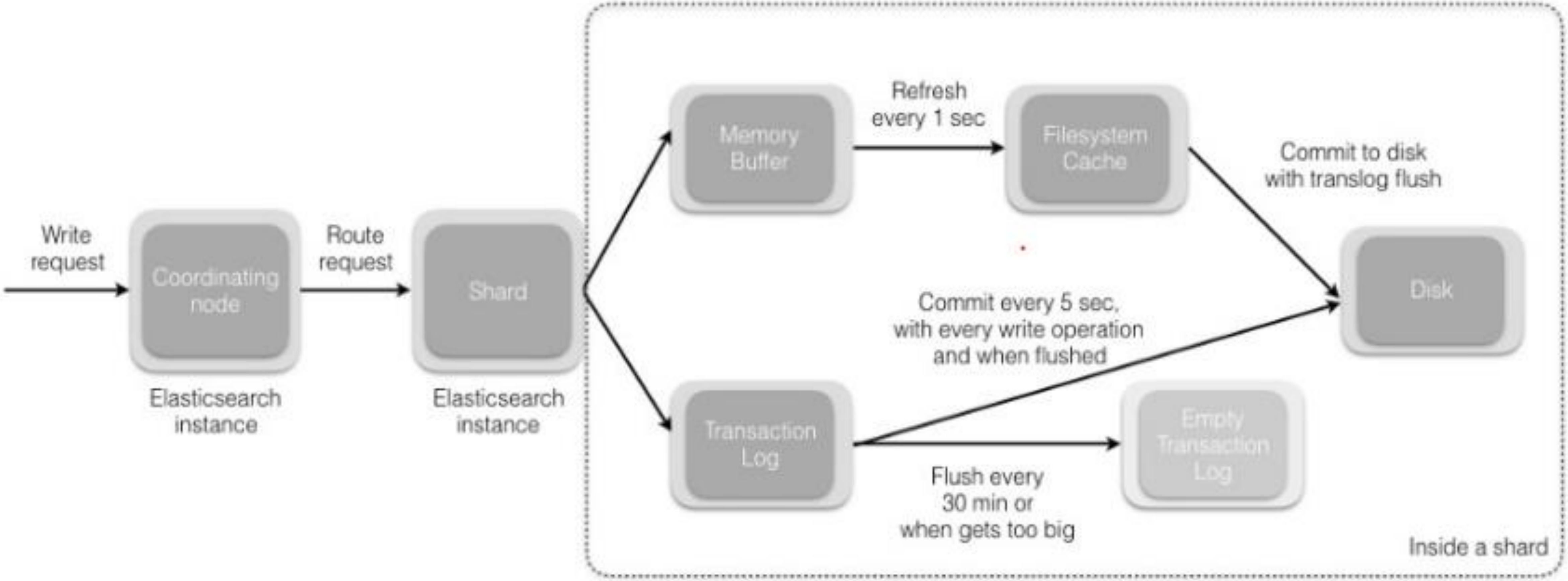


```
PUT /products/_doc/100
```

```
_primary_term:1
_seq_no:491
```

- PT & SN are key when ES needs to recover from a primary shard failure
- Global & Local checkpoint

How the write request and data flows



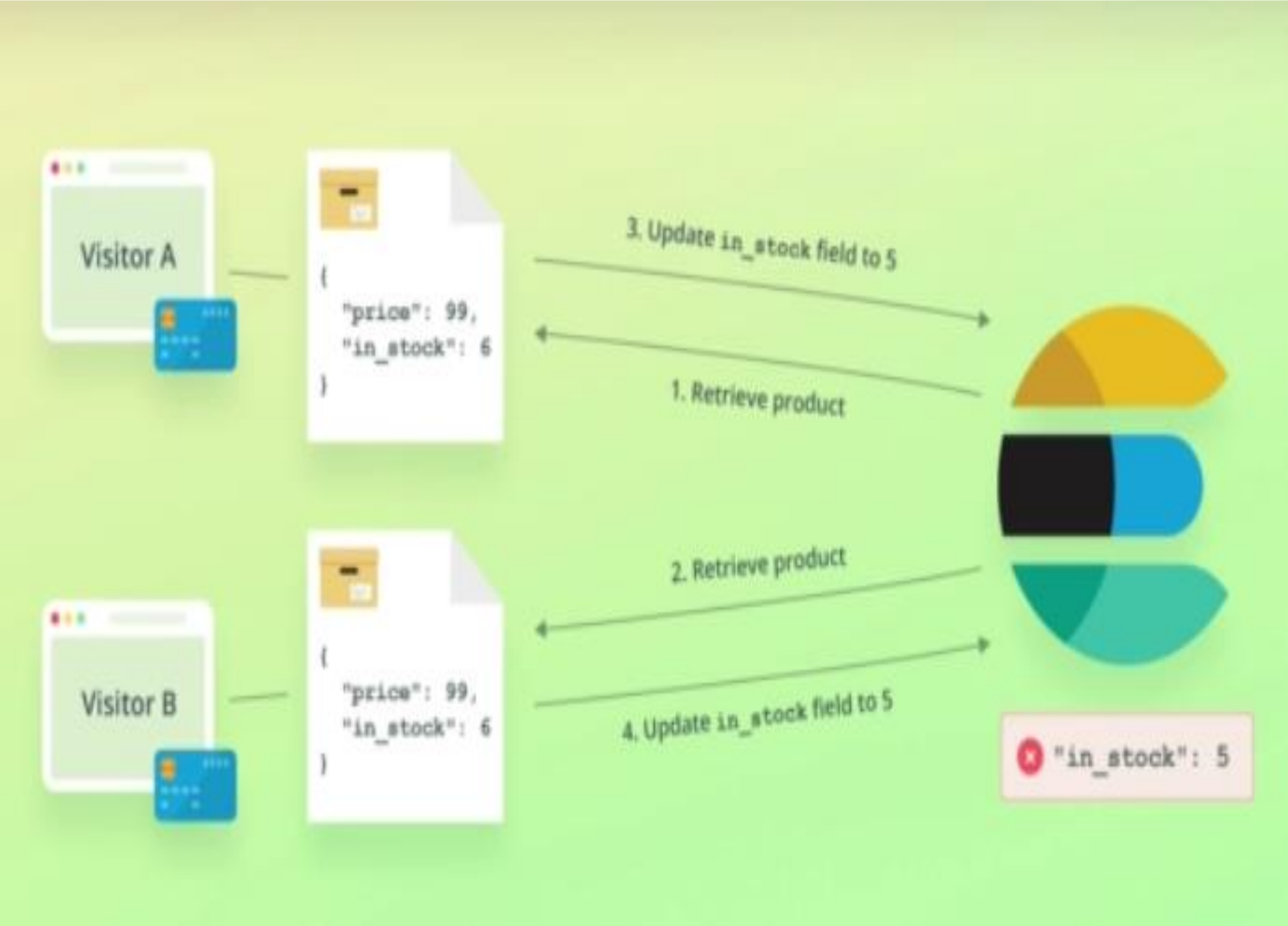
Document Versioning

- ES stores an _version metadata filed with every document
  - The value is an integer
  - Incremented by one when modifying a document
  - The value is retained for 60 seconds when deleting a document
  - Previously the way to do optimistic concurrency control
  - ES may delete older version

Optimistic Concurency control

Introduction

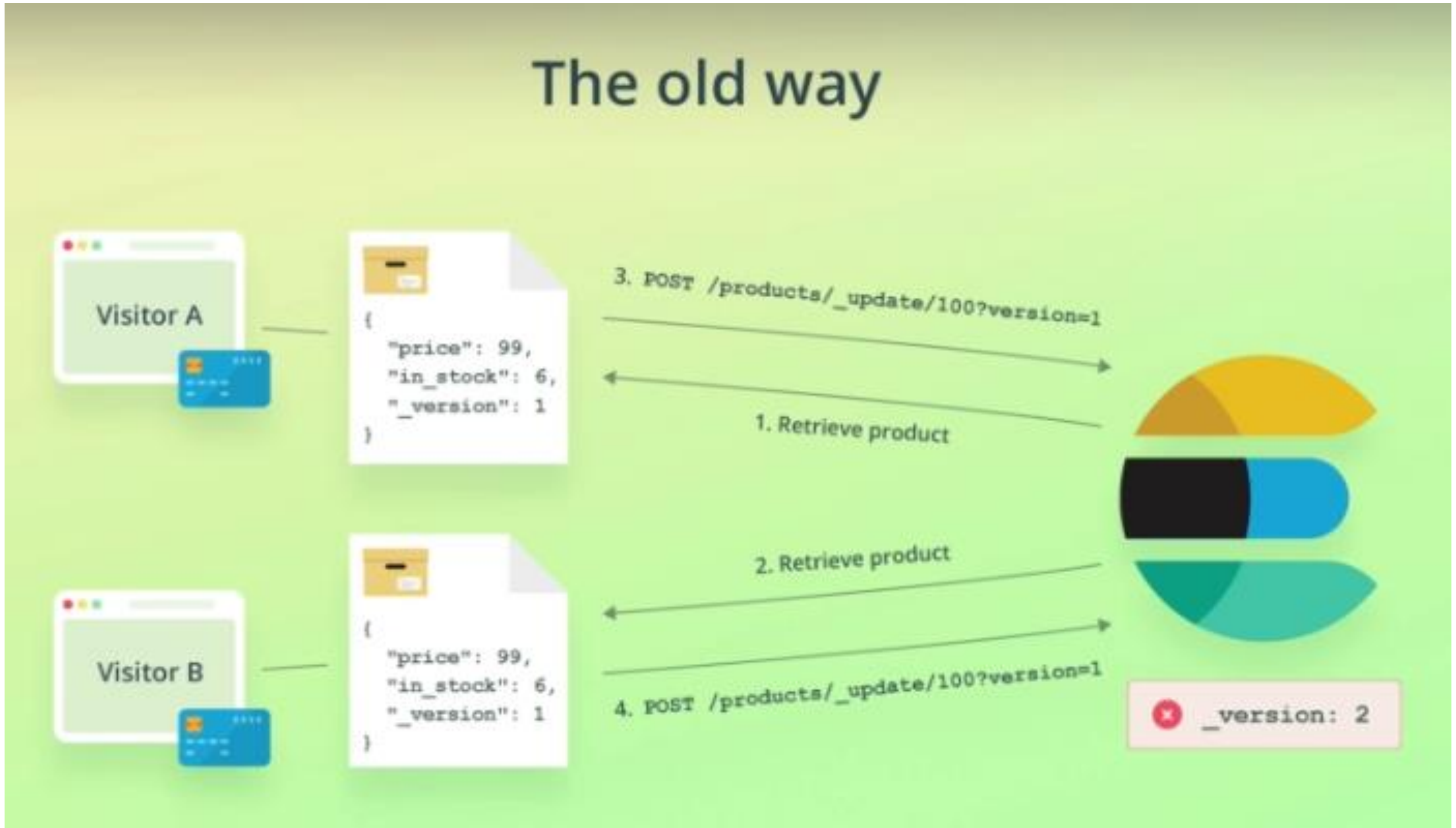
- Prevent overwriting documents inadvertently due to concurrent operations
- There are many scenarios



Th old way

Introduction

- Prevent overwriting documents inadvertently due to concurrent operations
- There are many scenarios



```
{
  "price": 99,
  "in_stock": 6,
  "_version": 1
}
```

```
{
  "price": 99,
  "in_stock": 6,
  "_version": 1
}
```

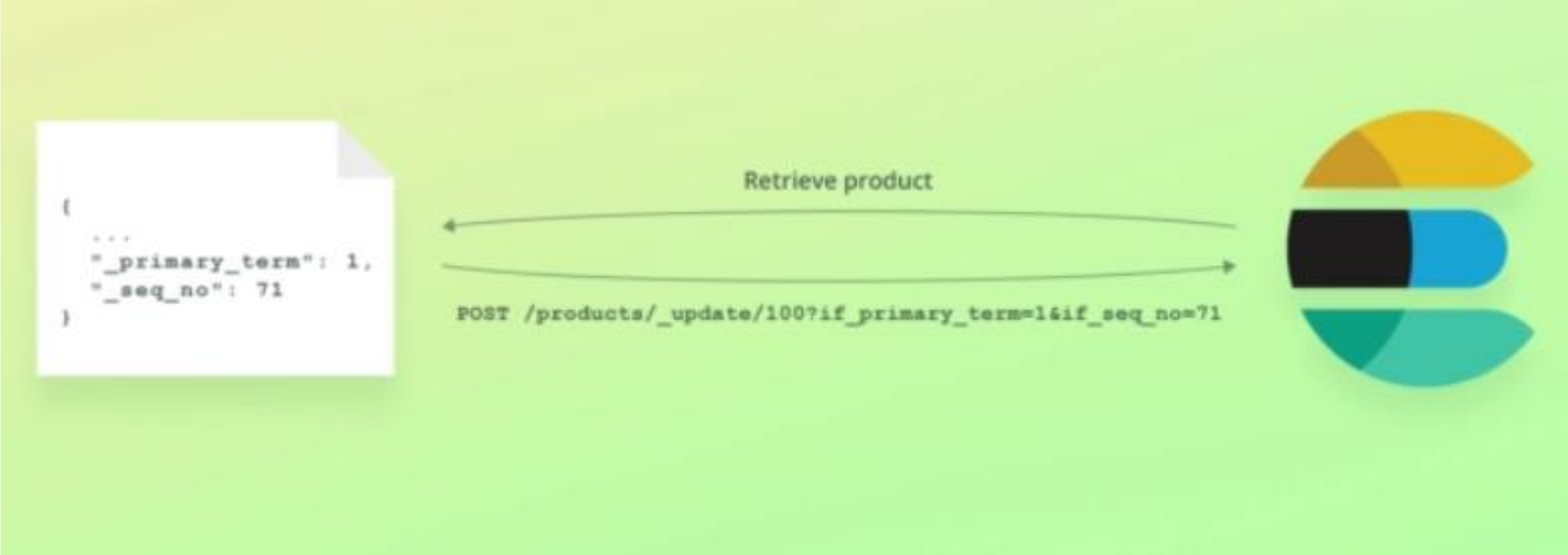
```
3. POST /products/_update/100?version=1
4. POST /products/_update/100?version=1
```

```
_version: 2
```

The new way

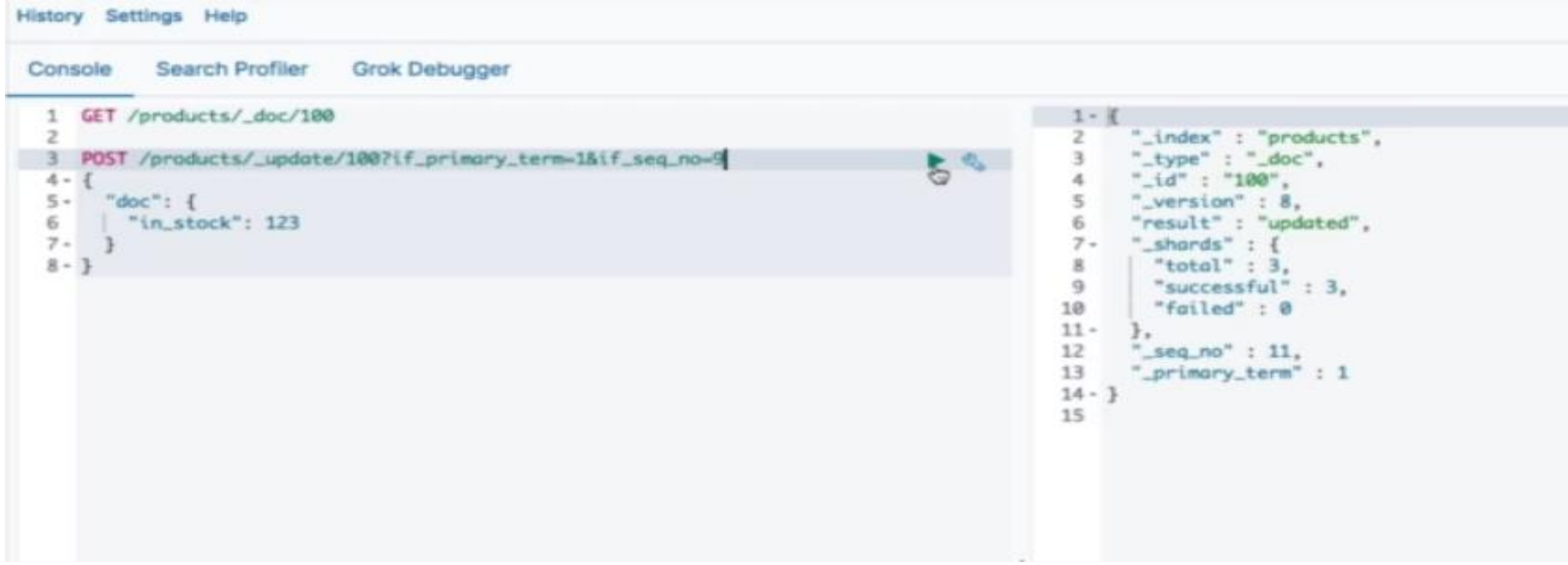
Handle failures

- Discussion



```
{
  ...
  "_primary_term": 1,
  "_seq_no": 71
}
```

```
POST /products/_update/100?if_primary_term=1&if_seq_no=71
```



```
GET /products/_doc/100

POST /products/_update/100?if_primary_term=1&if_seq_no=9
{
  "doc": {
    "in_stock": 123
  }
}
```

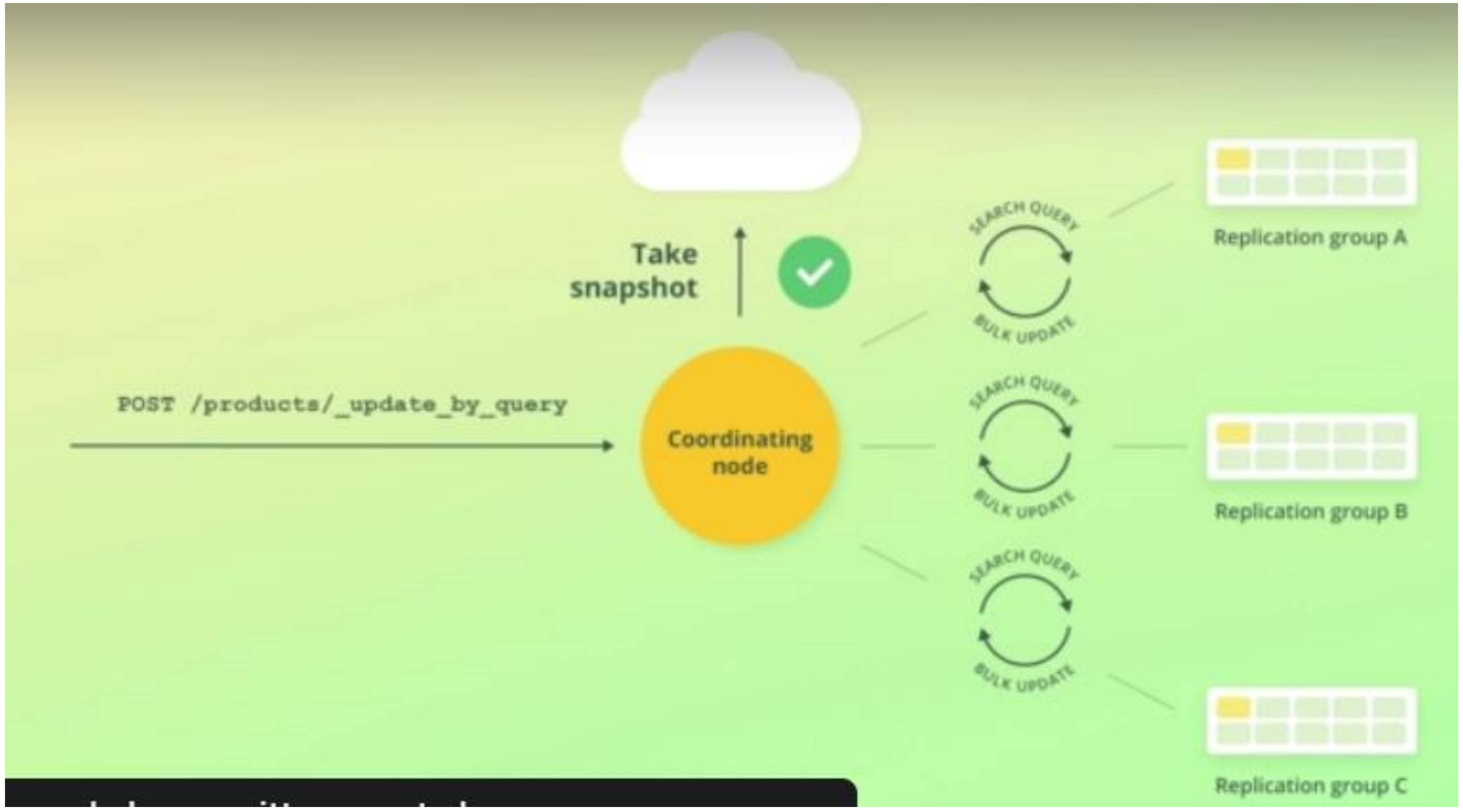
```
{
  "_index" : "products",
  "_type" : "_doc",
  "_id" : "100",
  "_version" : 8,
  "result" : "updated",
  "_shards" : {
    "total" : 3,
    "successful" : 3,
    "failed" : 0
  },
  "_seq_no" : 11,
  "_primary_term" : 1
}
```

Update by query

- How to update multiple documents within a single query (Similar to an update where query in a RDBMS)

- The query uses 3 concepts that
  - Primary terms
  - Seq Num
  - Optimistic concurents control

Discussion role of snapshot



```
POST /products/_update_by_query
```