INTRODUCTION TO ELASTICSEARCH

Agenda

- Me
- ElasticSearch Basics
  - Concepts
  - Network / Discovery
  - Data Structure
  - Inverted Index
- The REST API
- Sample Deployment

Me

- Roy Russo
- JBoss Portal Co-Founder
- LoopFuse Co-Founder
- ElasticHQ
  - http://www.elastichq.org
- AltiSource Labs Architect

ElasticSearch in One Slide

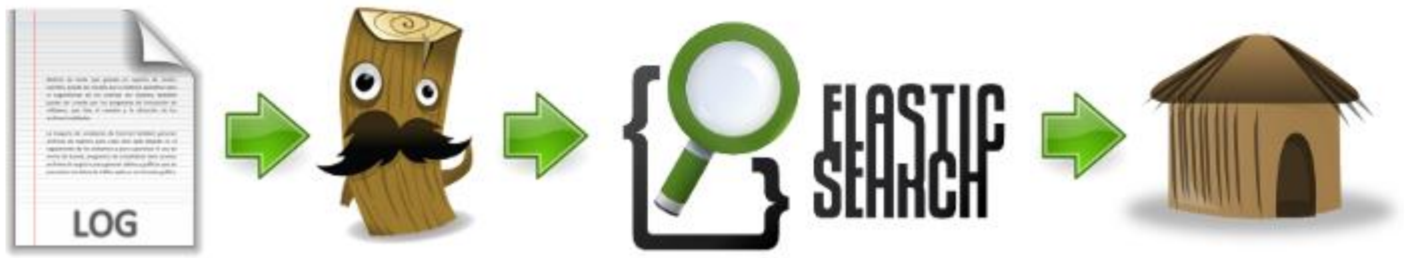
- Document - Oriented Search Engine
  - JSON
  - Apache Lucene
- No Schema
  - Mapping Types
- Horizontal Scale, Distributed
- REST API
- Vibrant Ecosystem
  - Tooling, Plugins, Hosting, Client-Libs

When to use ElasticSearch

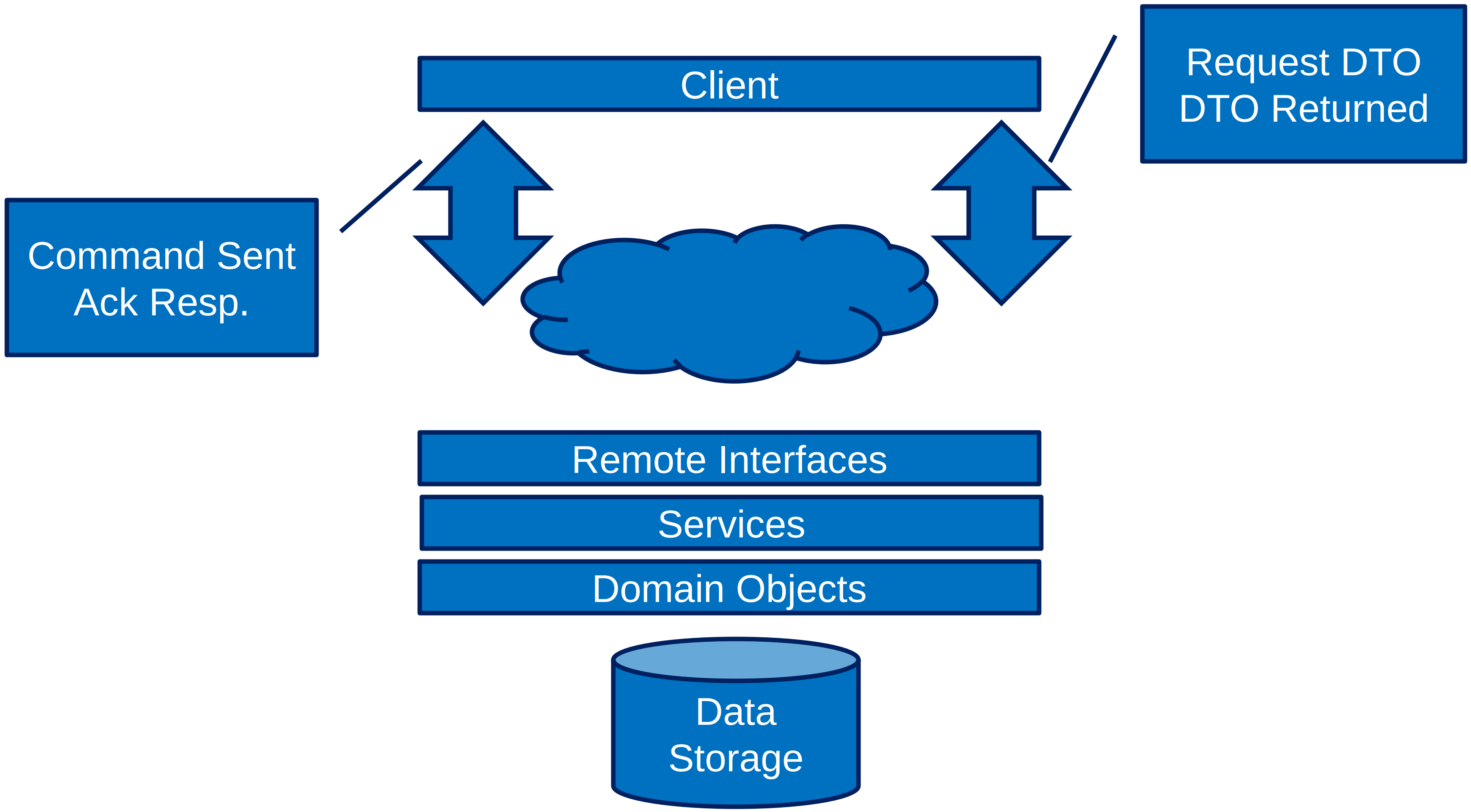
- Full-Text Search
- Fast Read Database
- “Simple” Data Structures
- Minimize Impedance Mismatch

When to use ElasticSearch - Logs

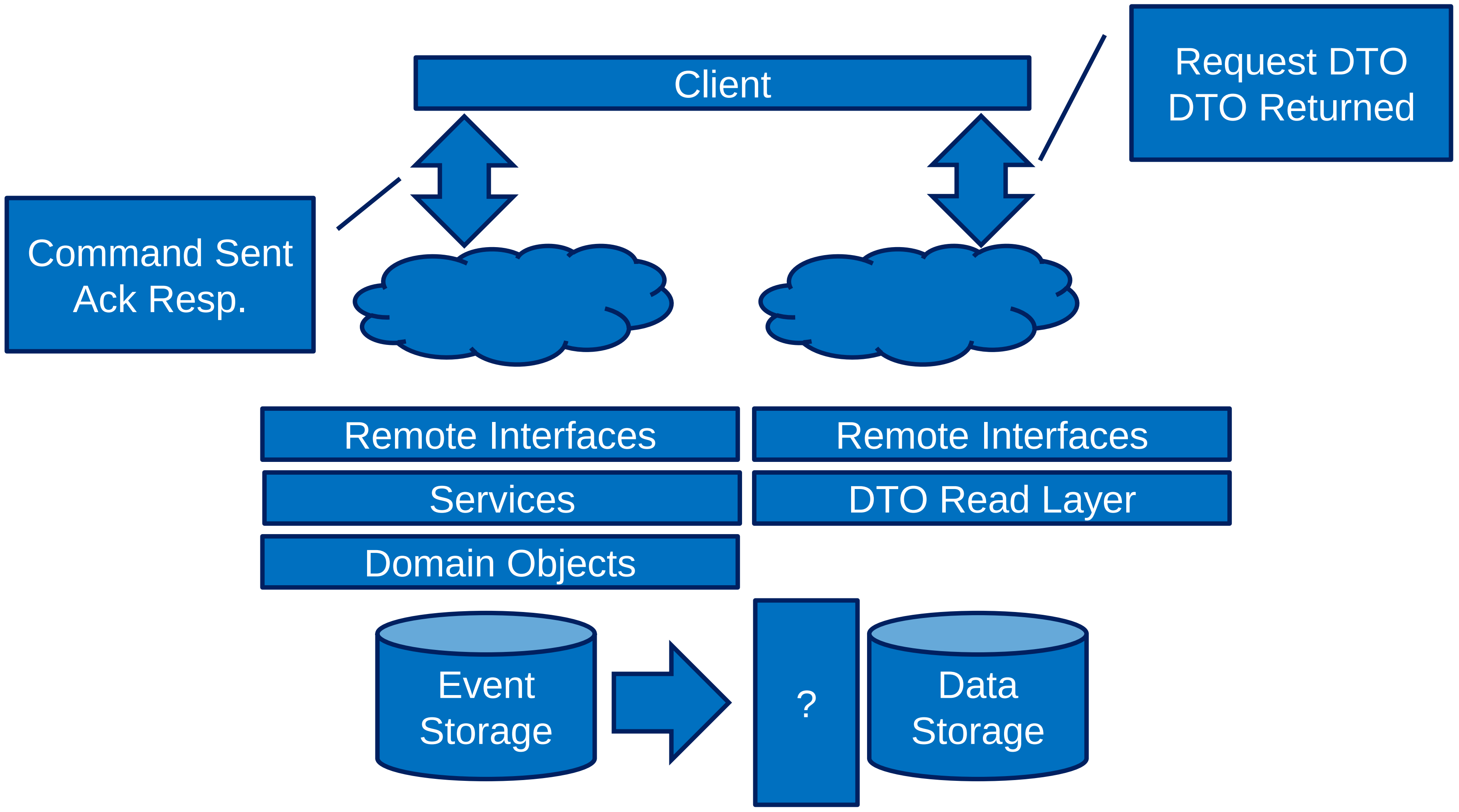
- Logstash + ElasticSearch + Kibana



How to use ElasticSearch - CQRS



How to use ElasticSearch - CQRS

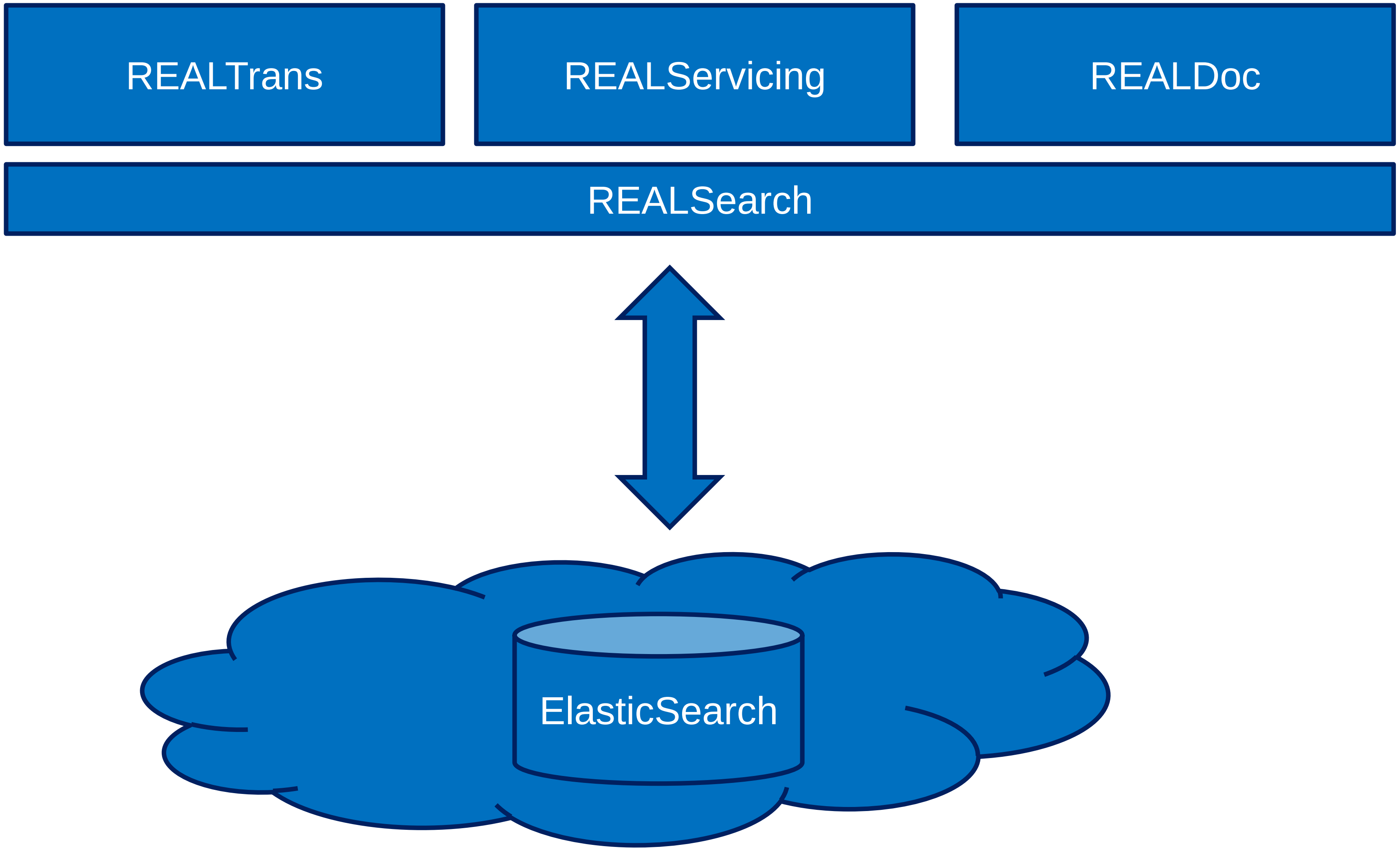


A note on Rivers

- JDBC
- CouchDB
- MongoDB
- RabbitMQ
- Twitter
- And more…

```
"type" : "jdbc",
"jdbc" : {
            "driver" : "com.mysql.jdbc.Driver",
            "url" : "jdbc:mysql://localhost:3306/my_db",
            "user" : "root",
            "password" : "mypassword",
            "sql" : "select * from products"
}
```

ElasticSearch at Work



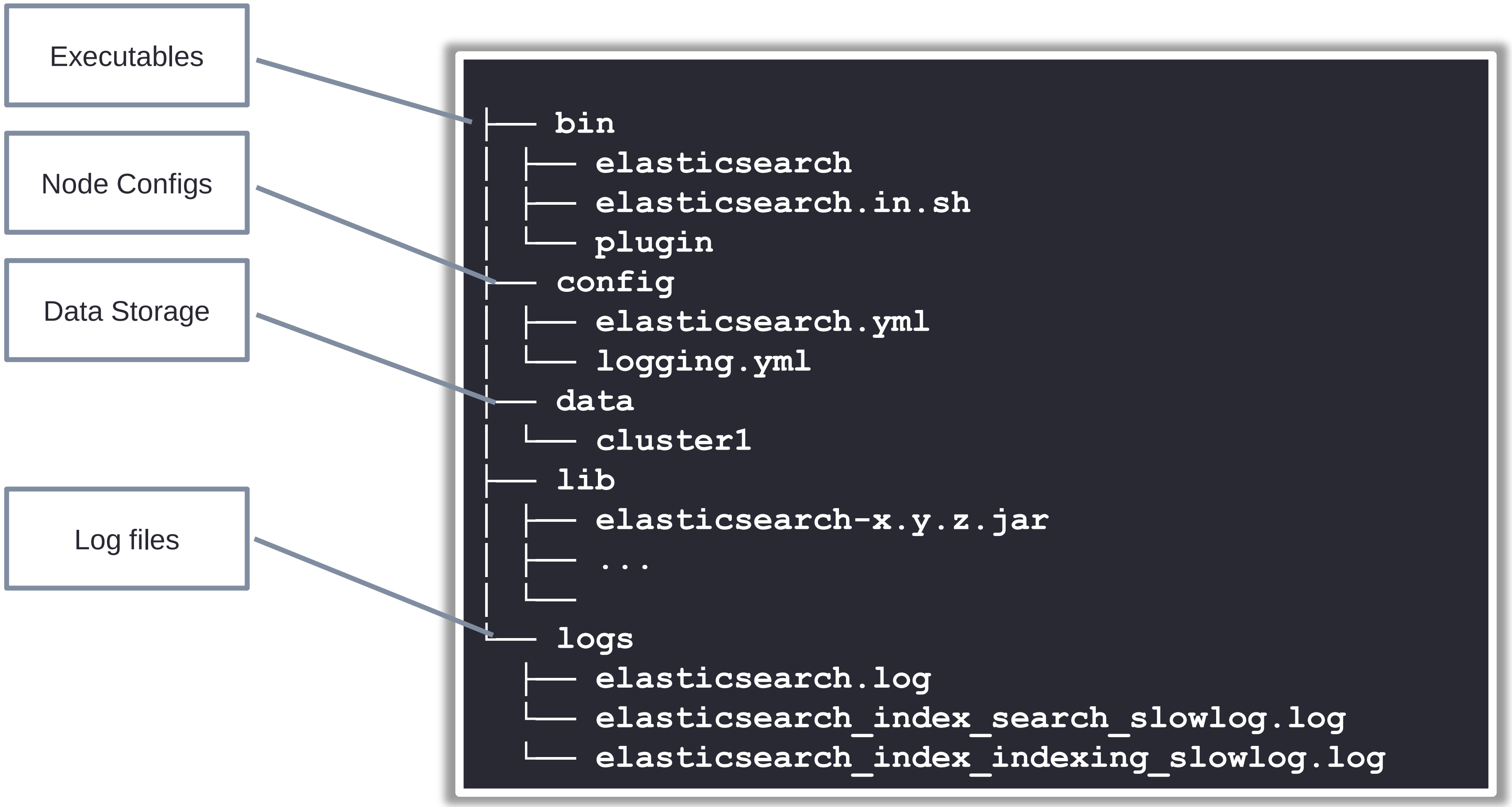
What sucks about ElasticSearch

- No AUTH/AUTHZ
- No Usage Metrics

How the World Uses ElasticSearch

The Basics - Distro

- Download and Run



```
├── bin
│   ├── elasticsearch
│   ├── elasticsearch.in.sh
│   └── plugin
├── config
│   ├── elasticsearch.yml
│   └── logging.yml
├── data
│   └── cluster1
├── lib
│   ├── elasticsearch-x.y.z.jar
│   ├── ...
│   └──
└── logs
    ├── elasticsearch.log
    └── elasticsearch_index_search_slowlog.log
    └── elasticsearch_index_indexing_slowlog.log
```

The Basics - Glossary

- Node = One ElasticSearch instance (1 java proc)
- Cluster = 1..N Nodes w/ same Cluster Name
- Index = Similar to a DB
  - Named Collection of Documents
  - Maps to 1..N Primary shards && 0..N Replica shards
- Mapping Type = Similar to a DB Table
  - Document Definition
- Shard = One Lucene instance
  - Distributed across all nodes in the cluster.

The Basics - Document Structure

- Modeled as a JSON object

```
{
   "genre": "Crime",
   “language": "English",
   "country": "USA",
   "runtime": 170,
   "title": "Scarface",
   "year": 1983
}
```

```
{
   "_index": "imdb",
   "_type": "movie",
   "_id": "u17o8zy9RcKg6SjQZqQ4Ow",
   "_version": 1,
   "_source": {
      "genre": "Crime",
      "language": "English",
      "country": "USA",
      "runtime": 170,
      "title": "Scarface",
      "year": 1983
   }
}
```

The Basics - Document Structure

- Document Metadata fields
  - _id
  - _type : mapping type
  - _source : enabled/disabled
  - _timestamp
  - _ttl
  - _size : size of uncompressed _source
  - _version

The Basics - Document Structure

- Mapping:
  - ES will auto-map (type) fields
  - You can specify mapping, if needed
- Data Types:
  - String
  - Number
    - Int, long, float, double, short, byte
  - Boolean
  - Datetime
    - formatted
  - geo_point, geo_shape
  - Array
  - Nested
  - IP

A Mapping Type

```
"imdb": {
    "movie": {
      "properties": {
        "country": {
          "type": "string“,
          “store”:true,
          “index”:false
        },
        "genre": {
          "type": "string“,
          "null_value" : "na“,
          “store”:false,
          “index:true
        },
        "year": {
          "type": "long"
        }
      }
    }
  }
```

Lucene – Inverted Index

- Which presidential speeches contain the words “fair”
  - Go over every speech, word by word, and mark the speeches that contain it
  - Fails at large scale

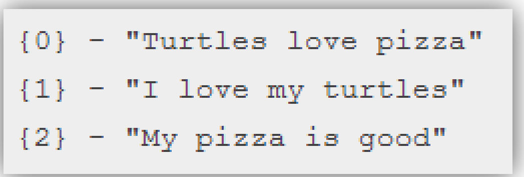
Lucene – Inverted Index

- Inverting
  - Take all the speeches
  - Break them down by word (tokenize)
  - For each word, store the IDs of the speeches
  - Sort all words (tokens)
- Searching
  - Finding the word is fast
  - Iterate over document IDs that are referenced

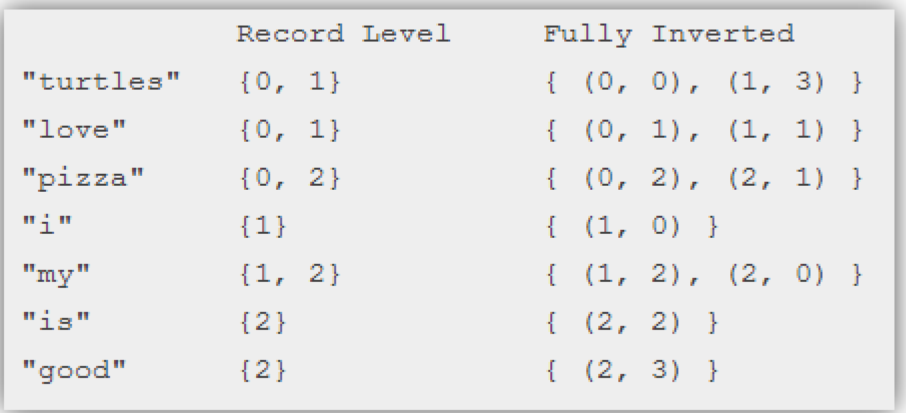
Token	Doc Frequency	Doc IDs
Jobs	2	4,8
Fair	5	1,2,4,8,42
Bush	300	1,2,3,4,5,6, …

Lucene – Inverted Index

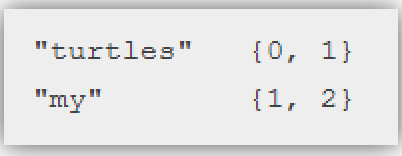
- Not an algorithm
- Implementations vary



```
{0} - "Turtles love pizza"
{1} - "I love my turtles"
{2} - "My pizza is good"
```



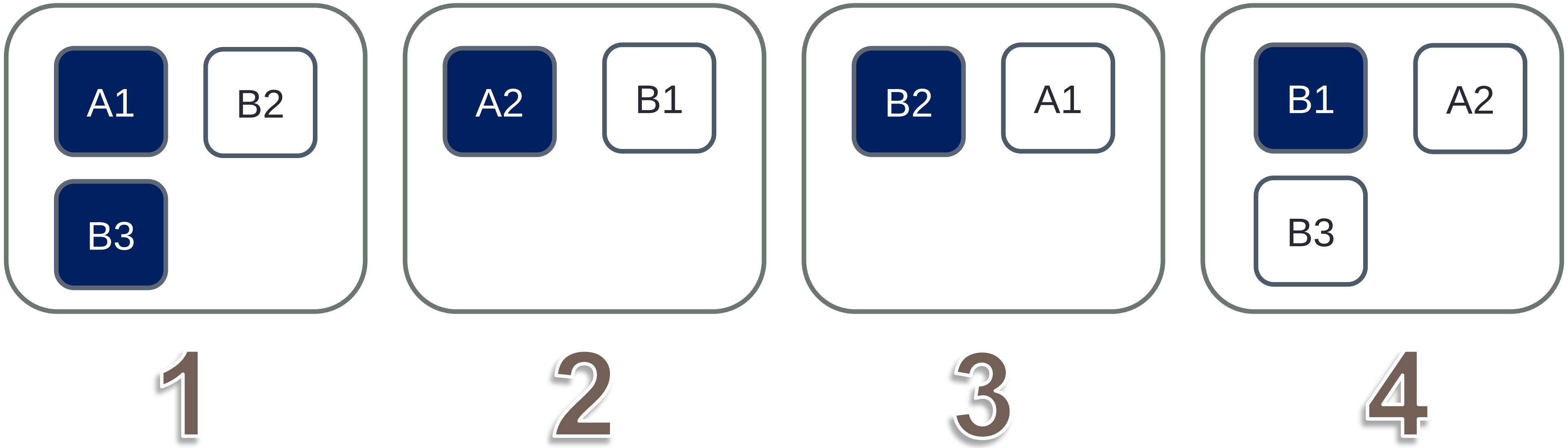
	Record Level	Fully Inverted
"turtles"	{0, 1}	{ (0, 0), (1, 3) }
"love"	{0, 1}	{ (0, 1), (1, 1) }
"pizza"	{0, 2}	{ (0, 2), (2, 1) }
"i"	{1}	{ (1, 0) }
"my"	{1, 2}	{ (1, 2), (2, 0) }
"is"	{2}	{ (2, 2) }
"good"	{2}	{ (2, 3) }



"turtles"	{0, 1}
"my"	{1, 2}

Cluster Topology

- 4 Node Cluster
- Index Configuration:
  - “A”: 2 Shards, 1 Replica
  - “B”: 3 Shards, 1 Replica



Building a Cluster

Start Cluster…

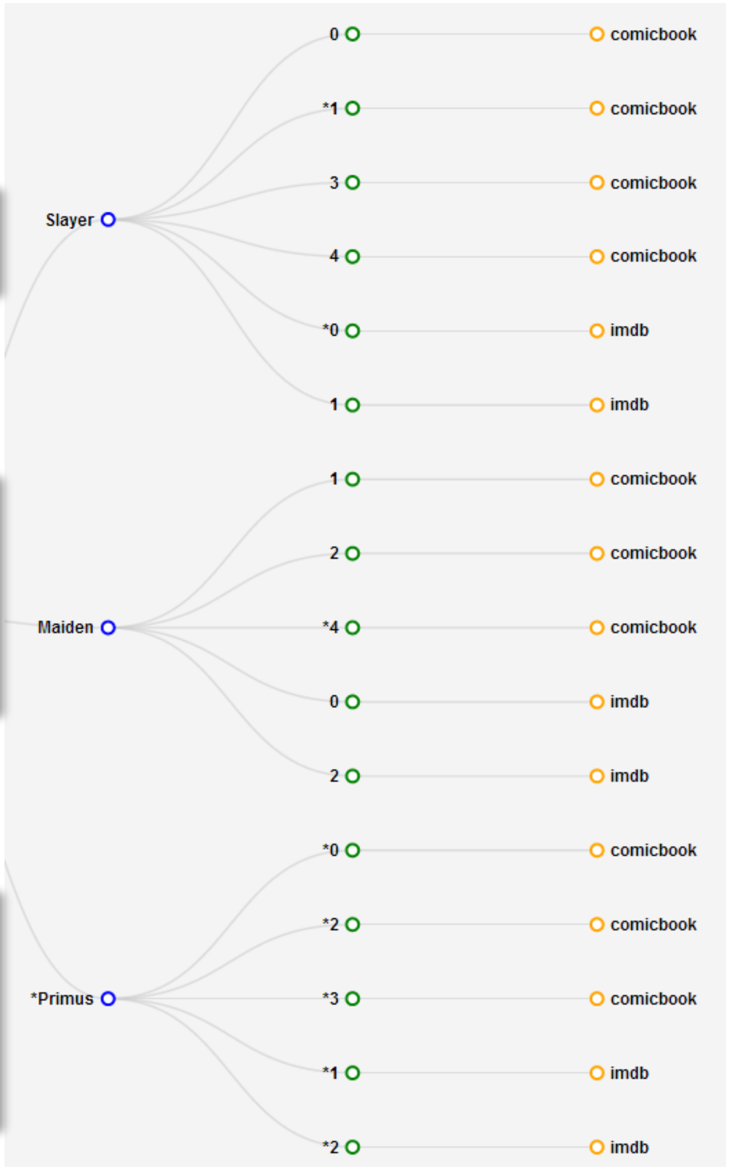
```
start cmd.exe /C elasticsearch -Des.node.name=Primus
start cmd.exe /C elasticsearch -Des.node.data=true -Des.node.master=false -Des.node.name=Slayer
start cmd.exe /C elasticsearch -Des.node.data=true -Des.node.master=true -Des.node.name=Maiden
```

Create Index…

```
curl -XPUT 'http://localhost:9200/imdb/' -d '{
    "settings" : {
        "index" : {
            "number_of_shards" : 3,
            "number_of_replicas" : 1
        }
    }
}'
```

Index Document…

```
curl -XPOST 'http://localhost:9200/imdb/movie/' -d '{
      "genre": “Comedy",
      "language": "English",
      "country": "USA",
      "runtime": 99,
      "title": “Big Trouble in Little China",
      "year": 1986
}'
```

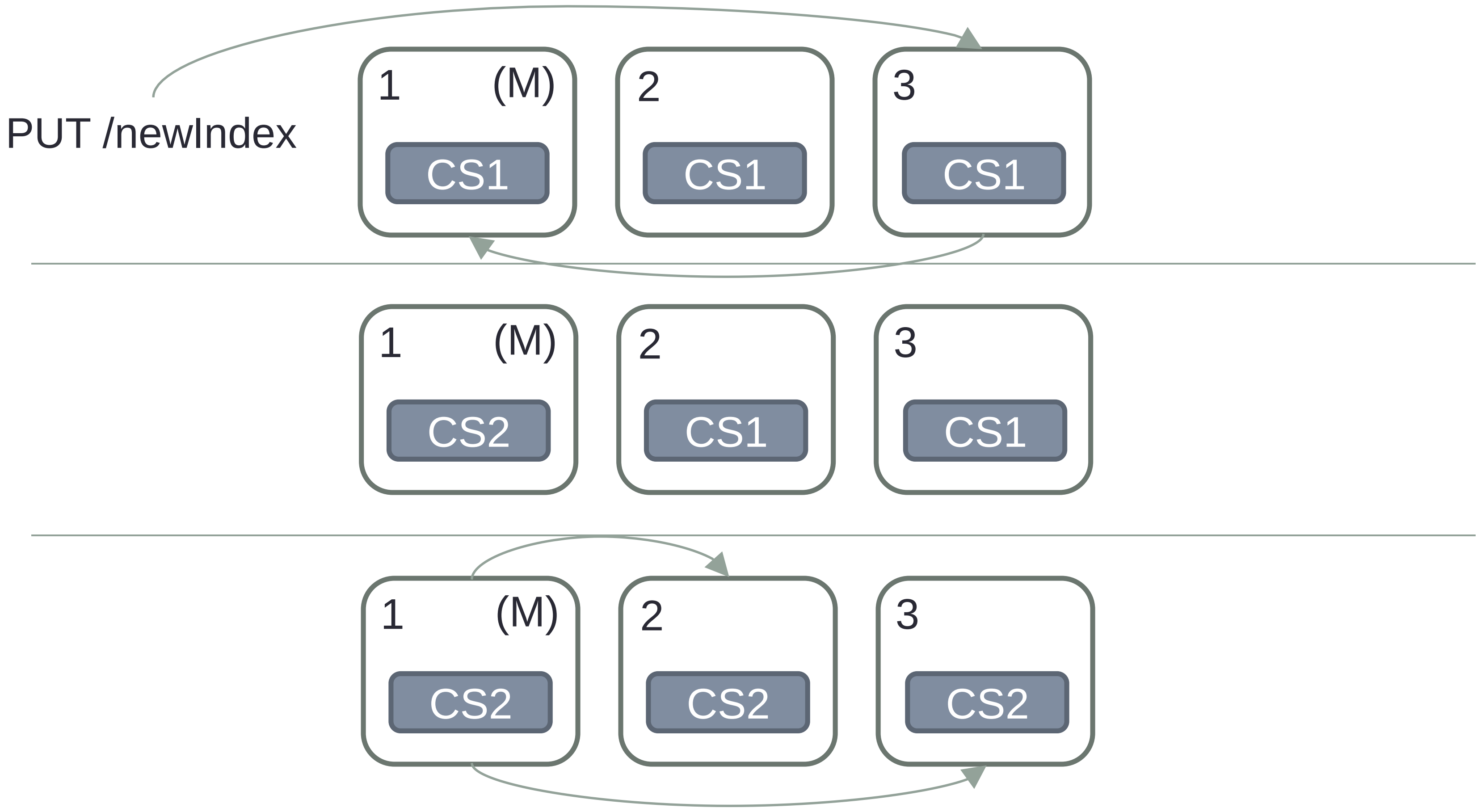


Cluster State

- Cluster State
  - Node Membership
  - Indices Settings and Mappings (Types)
  - Shard Allocation Table
  - Shard State
- cURL -XGET http://localhost:9200/_cluster/state?pretty=1'

Cluster State

- Changes in State published from Master to other nodes



Discovery

- Nodes discover each other using multicast.
  - Unicast is an option

```
discovery.zen.ping.multicast.enabled: false

discovery.zen.ping.unicast.hosts: ["host1", "host2:port", "host3"]
```

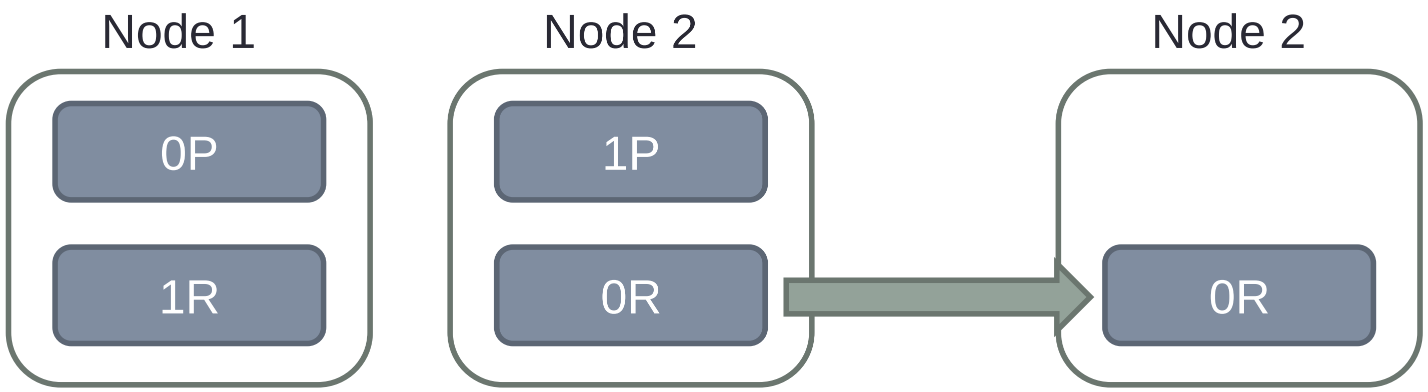
- Each cluster has an elected master node
  - Beware of split-brain

The Basics - Shards

- Primary Shard:
  - First time Indexing
  - Index has 1..N primary shards (default: 5)
  - # Not changeable once index created
- Replica Shard:
  - Copy of the primary shard
  - Can be changed later
  - Each primary has 0..N replicas
  - HA:
    - Promoted to primary if primary fails
    - Get/Search handled by primary||replica

Shard Auto-Allocation

- Add a node - Shards Relocate



- Shard Stages
  - UNASSIGNED
  - INITIALIZING
  - STARTED
  - RELOCATING

The Basics – Searching

- How it works:
  - Search request hits a node
  - Node broadcasts to every shard in the index
  - Each shard performs query
  - Each shard returns results
  - Results merged, sorted, and returned to client.
- Problems:
  - ES has no idea where your document is
  - Broadcast query to 100 nodes
  - Performance degrades

The Basics - Shards

- Shard Allocation Awareness
  - cluster.routing.allocation.awareness.attributes: rack_id
  - Example:
    - 2 Nodes with node.rack_id=rack_one
    - Create Index 5 shards / 1 replica (10 shards)
    - Add 2 Nodes with node.rack_id=rack_two
    - Shards RELOCATE to even distribution
    - Primary & Replica will NOT be on the same rack_id value.
  - Shard Allocation Filtering
    - node.tag=val1
    - index.routing.allocation.include.tag:val1,val2

```
curl -XPUT localhost:9200/newIndex/_settings -d '{
    "index.routing.allocation.include.tag" : "val1,val2"
}'
```

Nodes

- Master node handles cluster-wide (Meta-API) events:
  - Node participation
  - New indices create/delete
  - Re-Allocation of shards
- Data Nodes
  - Indexing / Searching operations
- Client Nodes
  - REST calls
  - Light-weight load balancers

REST API

- Create Index
  - action.auto_create_index: 0
- Index Document
  - Dynamic type mapping
  - Versioning
  - ID specification
  - Parent / Child (/1122?parent=1111)

REST API – Versioning

- Every document is Versioned
- Version assigned on creation
    - Version number can be assigned

REST API - Update

- Update using partial data
- Partial doc merged with existing
- Fails if document doesn't exist
- "Upsert" data used to create a doc, if doesn't exist

```
{
   "upsert" : {
      "title": "Blade Runner"
   }
}
```

REST API

- Exists
  - No overhead in loading
  - Status Code Result
- Delete
- Get
  - Multi-Get

```
{
"docs" : [
  {
   "_id" : "1"
   "_index" : "imdb"
   "_type" : "movie"
  },
  {
   "_id" : "5"
   "_index" : "oldmovies"
   "_type" : "movie"
   "_fields" " ["title", "genre"]
  }
]
}
```

REST API - Search

- Free Text Search
  - URL Request
  - http://localhost:9200/imdb/movie/_search?q=scar*
- Complex Query
- http://localhost:9200/imdb/movie/_search?q=scarface+OR+star
- http://localhost:9200/imdb/movie/_search?q=(scarface+OR+star)+AND+year:[1981+TO+1984]

REST API - Search

- Search Types:
  - http://localhost:9200/imdb/movie/_search?q=(scarface+OR+star)+AND+year:[1941+TO+1984]&search_type=count
  - http://localhost:9200/imdb/movie/_search?q=(scarface+OR+star)+AND+year:[1941+TO+1984]&search_type=query_then_fetch
  - Query and Fetch (fastest):
    - Executes on all shards and return results
  - Query then Fetch (default):
    - Executes on all shards. Only some information returned for rank/sort, only the relevant shards are asked for data

REST API – Query DSL

http://localhost:9200/imdb/movie/_search?q=(scarface+OR+star)+AND+year:[1981+TO+1984]

Becomes…

```
curl -XPOST 'localhost:9200/_search?pretty' -d '{
   "query" : {
      "bool" : {
         "must" : [
         {
            "query_string" : {
               "query" : "scarface or star"
            }
         },
      {
     "range" : {
          "year" : { "gte" : 1931 }
                }
            }
      ]
    }
  }
}'
```

REST API – Query DSL

- Query String Request use Lucene query syntax
  - Limited
  - Instead use “match” query

```
curl -XPOST 'localhost:9200/_search?pretty' -d '{
   "query" : {
       "bool" : {
           "must" : [
           {
               “match" : {
                   “message" : “scarface star"
               }
           },
       {
      "range" : {
           “year" : { "gte" : 1981 }
                     }
               }
         ]
…
```

Automatically builds a boolean query

REST API – Query DSL

- Match Query

```
{
  “match”:{
    “title”:{
      “type”:“phrase”,
      “query”:“quick fox”,
      “slop”:1
    }
  }
}
```

- Boolean Query
  - Must: document must match query
  - Must_not: document must not match query
  - Should: document doesn’t have to match
    - If it matches… higher score

```
{
  "bool":{
    "must":[
      {
        "match":{
          "color":"blue"
        }
      },
      {
        "match":{
          "title":"shirt"
        }
      }
    ],
    "must_not":[
      {
        "match":{
          "size":"xxl"
        }
      }
    ],
    "should":[
      {
        "match":{
          "textile":"cotton"
        }
```

REST API – Query DSL

- Range Query
  - Numeric / Date Types
- Prefix/Wildcard Query
  - Match on partial terms
- RegExp Query

```
{
  "range":{
    "founded_year":{
      "gte":1990,
      "lt":2000
    }
  }
}
```

REST API – Query DSL

- Geo_bbox
  - Bounding box filter
- Geo_distance
  - Geo_distance_range

```
{
  "query":{
    "filtered":{
      "query":{
        "match_all":{

        }
      },
      "filter":{
        "geo_distance":{
          "distance":"400km"
          "location":{
            "lat":40.73,
            "lon":-74.1
          }
        }
```

```
{
  "query":{
    "filtered":{
      "query":{
        "match_all":{

        }
      },
      "filter":{
        "geo_bbox":{
          "location":{
            "top_left":{
              "lat":40.73,
              "lon":-74.1
            },
            "bottom_right":{
              "lat":40.717,
              "lon":-73.99
            }
…
```

REST API – Bulk Operations

- Bulk API
  - Minimize round trips with index/delete ops
  - Individual response for every request action
    - In order
  - Failure of one action will not stop subsequent actions.
  - localhost:9200/_bulk

```
{ "delete" : { "_index" : “imdb", "_type" : “movie", "_id" : "2" } }\n
{ "index" : { "_index" : “imdb", "_type" : “actor", "_id" : "1" } }\n
{ "first_name" : "Tony", "last_name" : "Soprano" }\n
...
{ “update" : { "_index" : “imdb", "_type" : “movie", "_id" : "3" } }\n
{ doc : {“title" : “Blade Runner" } }\n
```

Percolate API

- Reversing Search
  - Store queries and filter (percolate) documents through them.
  - Useful for Alert/Monitoring systems

```
curl -XPUT localhost:9200/_percolator/stocks/alert-on-nokia -d '{
"query" : {
   "boolean" : {
       "must" : [
           { "term" : { "company" : "NOK" }},
           { "range" : { "value" : { "lt" : "2.5" }}}
        ]
     }
   }
}'
```

```
curl -X PUT localhost:9200/stocks/stock/1?percolate=* -d '{
    "doc" : {
        "company" : "NOK",
        "value" : 2.4
      }
}'
```

Clients

- Client list: http://www.elasticsearch.org/guide/clients/
  - Java Client, JS, PHP, Perl, Python, Ruby
- Spring Data:
  - Uses TransportClient
  - Implementation of ElasticsearchRepository aligns with generic Repository interfaces.
  - ElasticSearchCrudRepository extends PagingandSortingRepository
  - https://github.com/spring-projects/spring-data-elasticsearch

```
@Document(indexName = "book", type = "book", indexStoreType = "memory", shards = 1, replicas = 0, refreshInterval = "-1")
public class Book {
…
}

public interface ElasticSearchBookRepository extends ElasticsearchRepository<Book, String> {
}
```

B’what about Mongo?

- Mongo:
  - General purpose DB
- ElasticSearch:
  - Distributed text search engine

… that’s all I have to say about that.

Questions?