

# ElasticSearch

# ¿Que es ElasticSearch?

- Motor de búsqueda de texto completo basado en Apache Lucene
- Multitenance: Una sola instancia puede servir a varios clientes
- Utiliza Query DSL
- Escrito en Java



# Características de Elasticsearch

- Ejecuta sus operaciones a través de una API REST.
- Especialización en búsquedas Full Text Search.
- Almacenamiento de documentos distribuido y en tiempo real.
- Analíticas en tiempo real.
- Escalable en múltiples servidores con datos estructurados y no estructurados



# Características de Elasticsearch

- Almacena lógicamente sus datos en índices distribuyendo en n shards establecidos por configuración.
- Http API, API java nativa, todas las API routean la operación orientadas a documentos al nodo
- No requiere schema, de ser necesario se puede asignar en la fase de indexación



# Características de Elasticsearch

- Escritura asíncrona y persistente a largo plazo
- Búsquedas en tiempo real
- Operaciones de nivel documental de forma atómica, consistente, aislada y durable.



# Casos de Uso: Wikipedia y The guardian

- Wikipedia utiliza Elasticsearch para proveer full text search sobre los millones de artículos publicados.
- The Guardian implementa elasticsearch para logs de su red y social network data para obtener feedback en tiempo real sobre sus artículos.

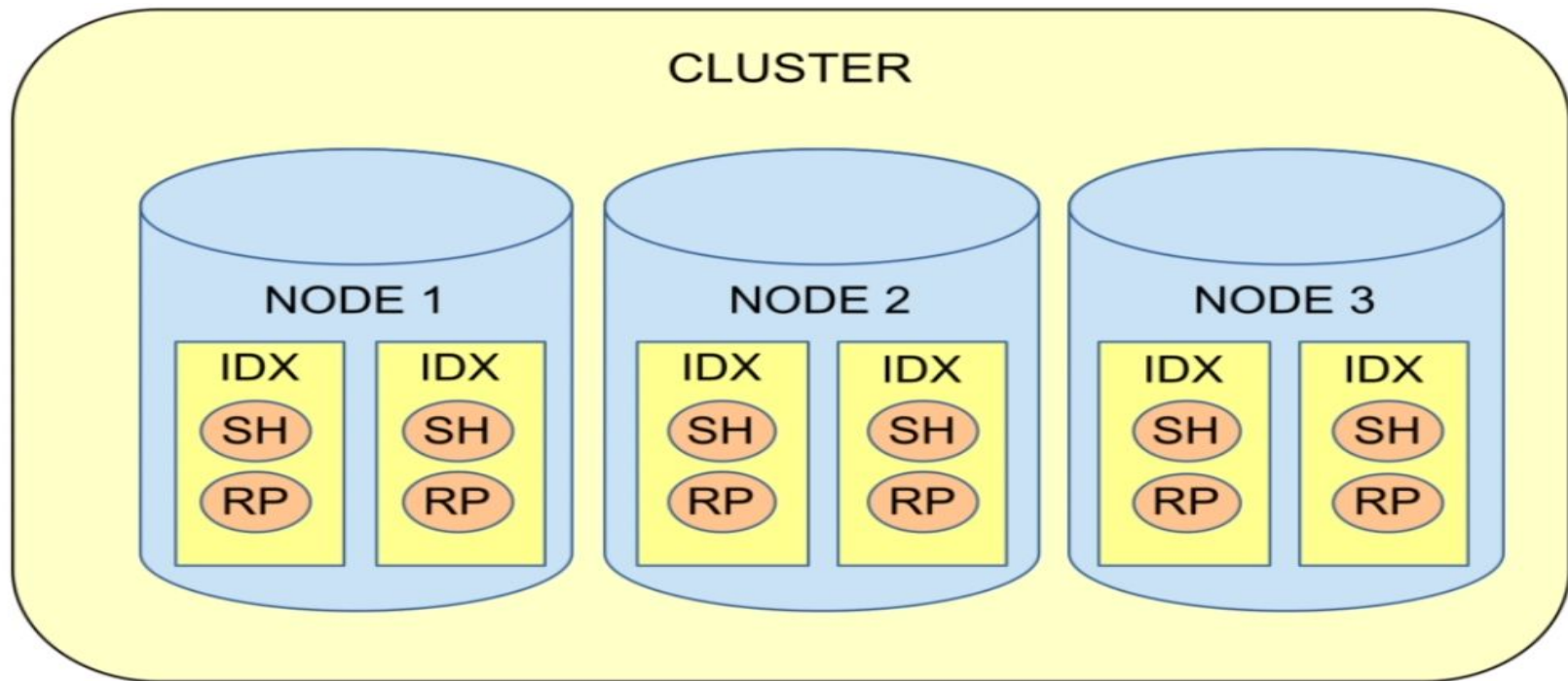


# Casos de Uso: Stackoverflow y Github

- Stackoverflow implementa queries MLT y geolocalización para encontrar preguntas y respuestas relacionadas.
- Github utiliza elasticsearch para consultar más de 130 millones de líneas de código.

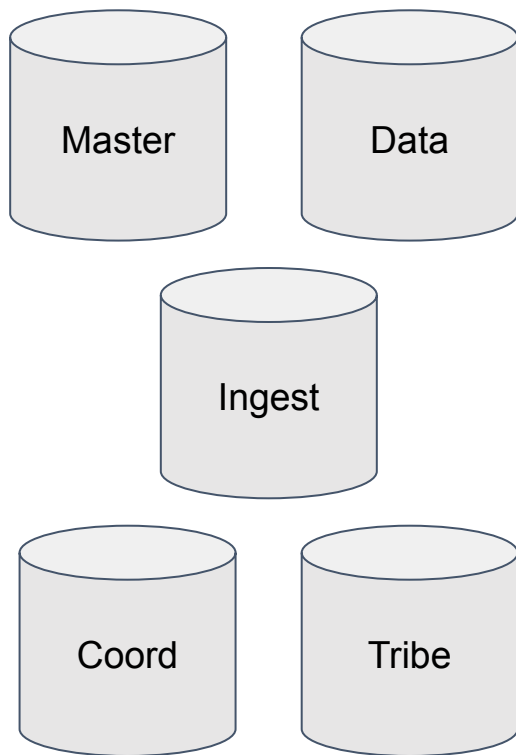


# Arquitectura





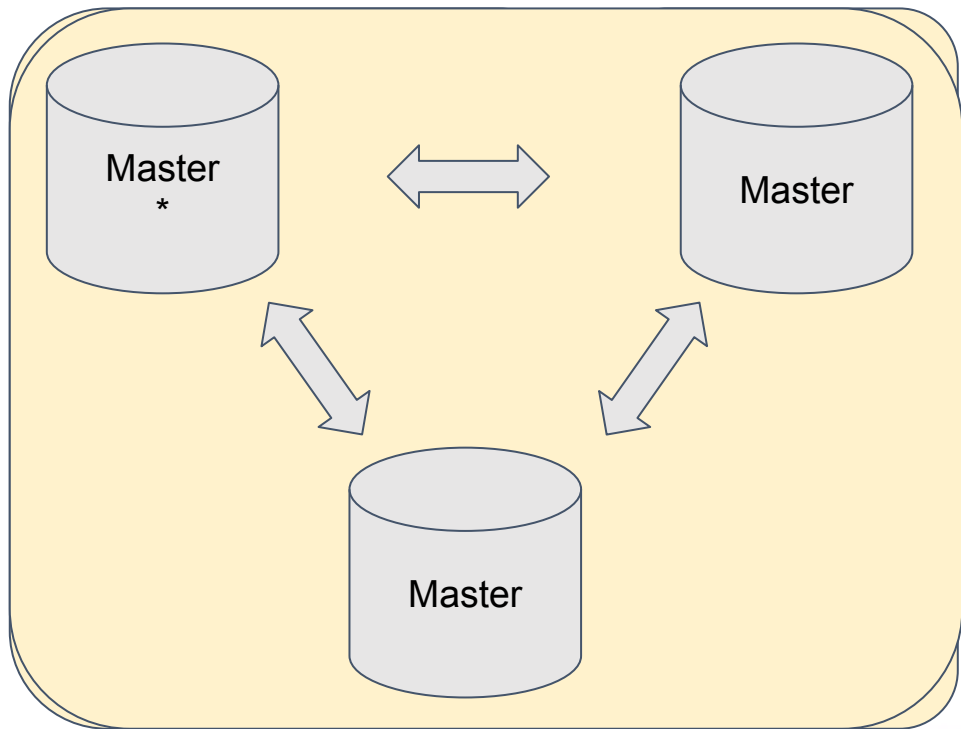
# Tipos de nodos



- Master: Tienen como responsabilidad gestionar el cluster y asegurar su integridad
- Data: Son los nodos normales que contienen los datos y ejecutan las búsquedas
- Ingest: Se encarga de la ingesta de datos a elasticSearch
- Coord: Su única función es enrutar peticiones dentro del cluster y agregar datos de consultas distribuidas
- Tribe: Cumplen una función de fachada, agregando varios clusters de manera transparente

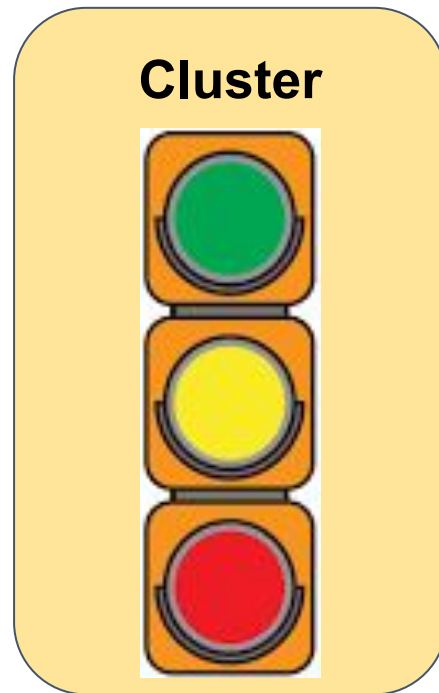
# Split Brain

- Un nodo master pierde comunicación y genera su propio cluster
- Se pierde lo indexado en un nodo
- Solución: Configurar el mínimo de nodos master un cluster  
**minimum\_master\_nodes**



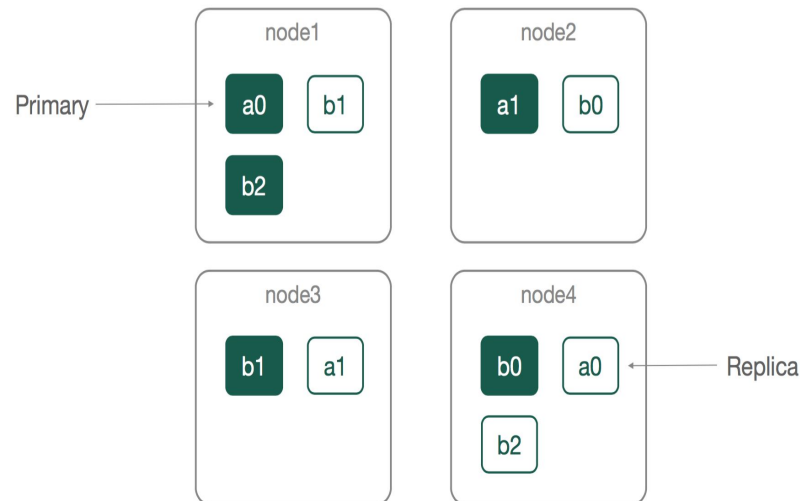
# Estados de un cluster

- Verde: Todos los shards, réplicas y primarios se encuentran activos.
- Amarillo: Todos los shards primarios activos, pero no todos los replica.
- Rojo: No todos los shards primarios se encuentran activos.



# Shards: ¿Que es un Shard?

- Container físico donde se almacenan los datos. Pueden ser primarios o réplicas.
- Tamaño limitado por el hardware disponible en el cluster.
- Los shards replica son copias de un shard primario que asegura alta disponibilidad y agiliza las búsquedas.



# Índices: ¿Que es un índice?

- Organización Lógica para los shards físicos, que almacenan una parte o la totalidad del índice.
- Los Shards se encuentran distribuidos en los nodos del cluster.
- Elasticsearch se encarga de mantener los shards del índice balanceados entre los nodos del cluster.

# Índices: ¿Que es un índice?

- Al crear el índice se selecciona la cantidad de shards primarios y de réplicas que le corresponden al mismo.

```
POST /blogs
{
  "settings" : {
    "number_of_shards" : 3,
    "number_of_replicas" : 1
  }
}
```

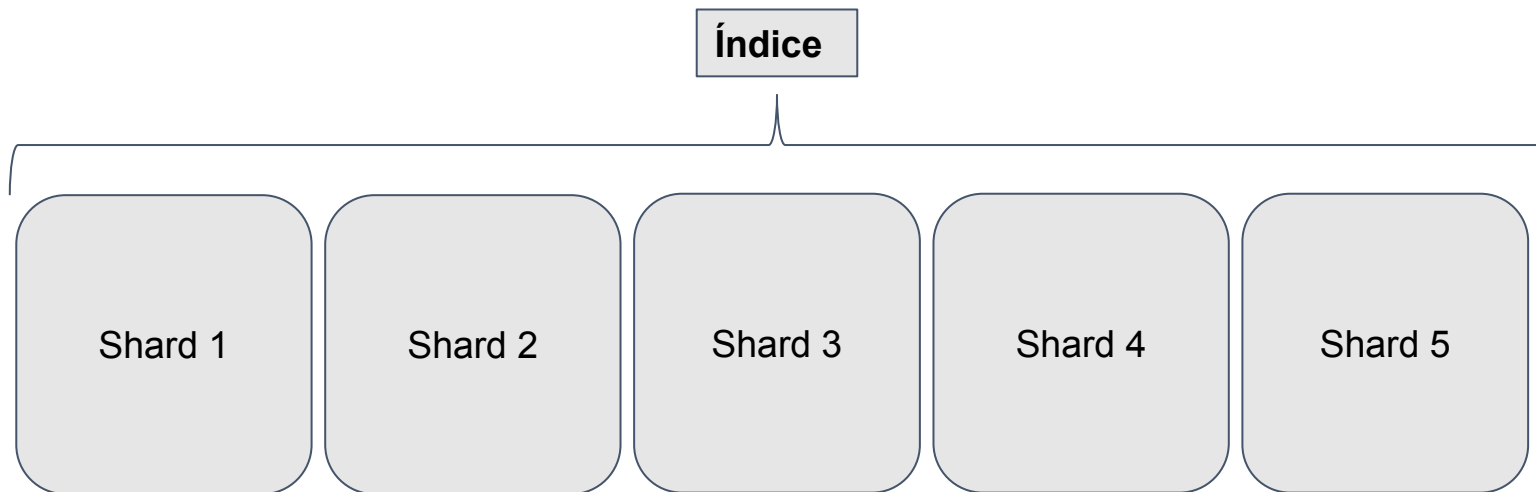
# Documentos

- Son la entidad mínima de almacenamiento en Elasticsearch.
- Tiene formato JSON
- Estos se encuentran almacenados en los índices, en consecuencia también están almacenados en los shards

```
{
  "_index" : "indice",
  "_type" : "_doc",
  "_id" :
  "zcrm_24B6W92TnjdoP41",
  "_score" : 1.0,
  "_source" : {
    "usuario" : "DBLandIT"
  }
}
```

# Índices + Shards + Documentos

- Cada índice se divide en shards.



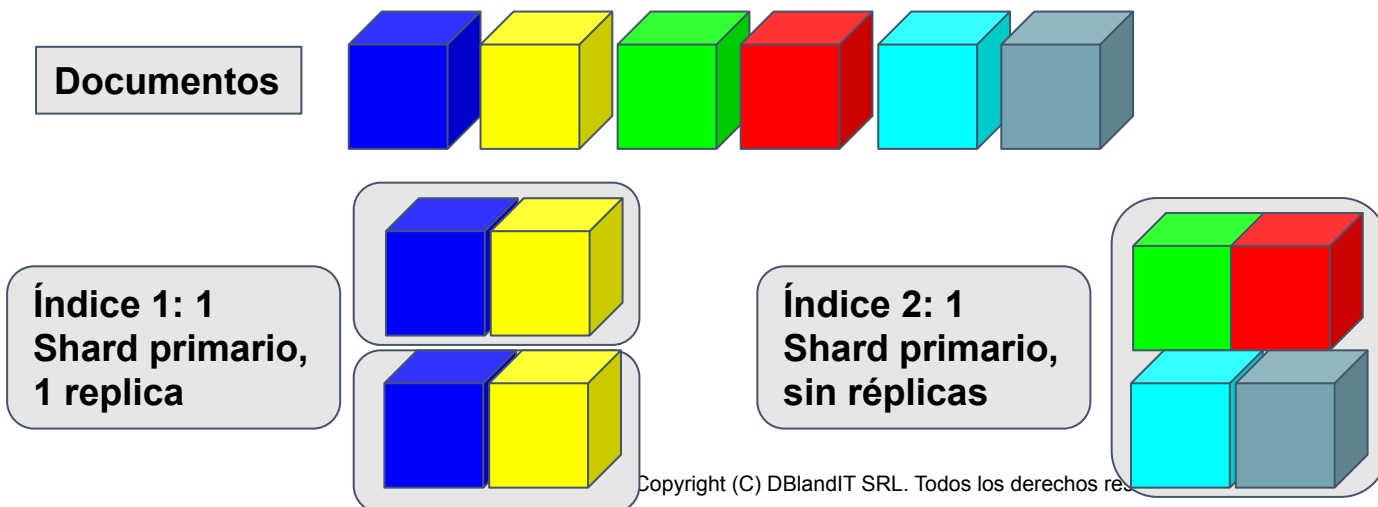


# Índices + Shards + Documentos




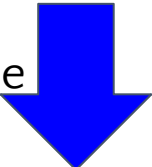

- Cada Shard se compone de un primario y uno o más réplicas.
- Dentro de los shards se almacenan los documentos del índice.
- Las escrituras solo son aceptadas por los shards primarios.
- Los shards réplicas proveen alta disponibilidad para las lecturas.
- La cantidad máxima de documentos almacenados en un índice será mayor cuantos más shards primarios se le asignen.

# Índices + Shards + Documentos

- Cada documento se almacenará en un único shard primario del índice a que se lo inserte.
- La cantidad máxima de documentos almacenados en un índice será mayor cuantos más shards primarios se le asignen.

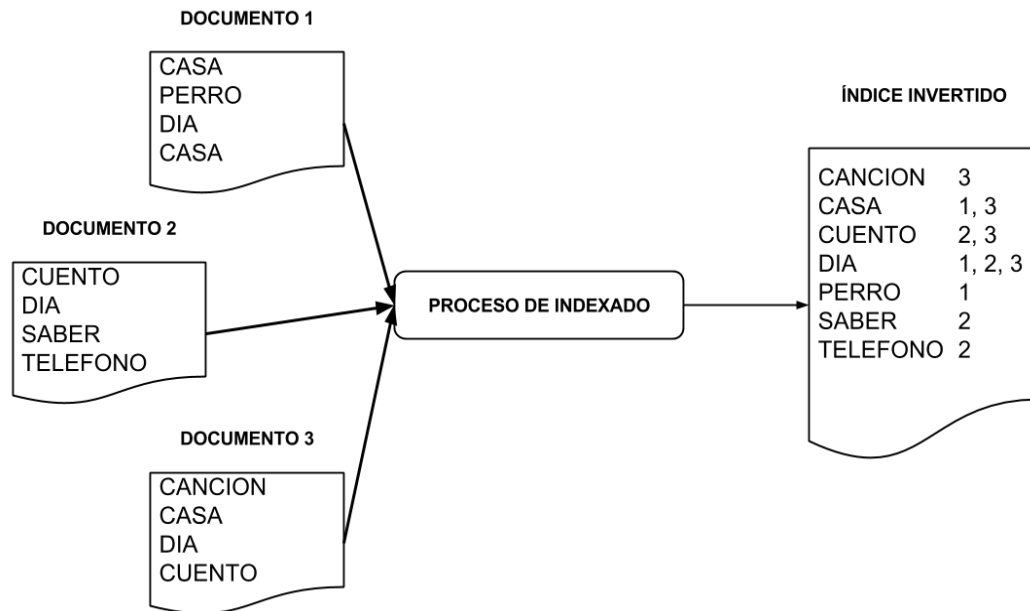


# Políticas de ciclo de vida

- Hot Phase: Rollover sobre el índice en caso de que el actual cumpla una condición designada (Índices read-write)  Read  Write
- Warm Phase: Para índices de solo lectura asigna hardware de menor rendimiento, como también encoger el índice para mayor rendimiento de la búsqueda  Read
- Cold Phase: Índices con poca actividad asignar hardware de menor rendimiento y disminuye el número de réplicas  Read
- Delete Phase: Define cuando es seguro eliminar el índice  Index

# Proceso de indexación: Índice invertido

- Proceso por el cual Elasticsearch guarda sus documentos
- Fundamental para asegurar la eficiencia de búsquedas de texto completo
- Sobre cada campo de cada documento se genera un índice invertido



# Proceso de indexación: Tokenización

- Va separa nuestros campo en términos para un posterior análisis
- Por default los separa en términos con espacios en blanco

`{"title": "Torrente, el brazo tonto de la ley"}`

Torrente  
el  
brazo  
tonto  
de  
la  
ley

# Proceso de indexación: Análisis

- Analiza cada término y busca sinónimos adecuados para cada palabra
- Singulariza y elimina el género
- Elimina los términos que no son útiles para un búsqueda (StopWords)

Torrente  
el  
brazo  
tonto  
de  
la  
ley



**torrent, arroyo, riachuelo**  
**el**  
**braz, extremidad**  
**tont, bob, mem, neci**  
**de**  
**la**  
**ley, norma, precepto**

# Proceso de indexación: Resultado Final

{“title”:“*Torrente, el brazo tonto de la ley*”}  
{“title”:“*Ciudad sin ley*”}  
{“title”:“*Dos tontos muy tontos*”}  
{“title”:“*Ciudades modernas*”}



*Doc.1*  
*Doc.2*  
*Doc.3*  
*Doc.4*



term	id
braz	1
ciudad	2,4
dos	3
ley	1,2
modern	4
tont	1,3
torrent	1

# Proceso de indexación: Doc Values

- Formato adicional de almacenamiento
- Registros en forma de columnas
- Se utiliza para: agregaciones, ciertos filtros, clasificación, scripts que refieren campos

Doc	Terms	
Doc_1	hello, world, perl	
Doc_2	hello, world, java	
Doc_3	We, need, more, golang, tutorials	



# Index Template

- Declaración de tipos de properties(campos)
- Cantidad de shards del índice
- Patrones a los que se aplica el template
- Evita la duplicación de campos innecesaria

```
{  
  "mappings": {  
    "properties": {  
      "title": { "type": "text" },  
      "name": { "type": "text" },  
      "age": { "type": "integer" },  
      "created": {  
        "type": "date",  
        "format": "strict_date_optional_time||epoch_millis"  
      }  
    }  
  }  
}
```

# Index Template

```
{
  "mappings": {
    "properties": {
      "title": { "type": "text" },
      "name": { "type": "text" },
      "age": { "type": "integer" },
      "created": {
        "type": "date",
        "format": "strict_date_optional_time||epoch_millis"
      }
    }
  }
}
```

Name	Type	Format	Searchable	Aggregata...	Excluded
host.os.version	string		●		
host.os.version.keyword	string		●	●	
input.type	string		●		
input.type.keyword	string		●	●	
ip	string		●		
ip.keyword	string		●	●	
level	string		●		
level.keyword	string		●	●	
log.file.path	string		●		
log.file.path.keyword	string		●	●	
Rows per page: 10 ▾			< 1 ... 6 7 <b>8</b> 9 10 >		

# Configuración de gestión de recursos JVM

- Heap Size JVM: El heap de memoria debe ser siempre menor al 50% del total de la misma.
- Elasticsearch usa memoria para otras operaciones por eso requiere el resto de la memoria.
- El límite de memoria para la JVM es 26GB, para evitar inconvenientes con el garbage collector.



# Configuración JVM (jvm.options)

- `XmsNg` Tamaño mínimo del heap de memoria (N = cantidad de GB)
- `XmxNg` Tamaño máximo del heap de memoria (N = cantidad de GB)
- Recomendable que Maximo y Minimo sean iguales

# Xms represents the initial size of total heap space  
# Xmx represents the maximum size of total heap space

`-Xms1g`  
`-Xmx1g`

# Configuración Elasticsearch (elasticsearch.yml)

- `cluster.name` nombre del cluster, por default “elasticsearch”
- `node.name` nombre descriptivo del nodo, por defecto “node-n”
- `node.attr.rack` Especificación adicional de la posición del nodo en el servidor

```
# Use a descriptive name for your cluster:

cluster.name: my-application

# ----- Node
# -----

# Use a descriptive name for the node:

node.name: node-1

# Add custom attributes to the node:

node.attr.rack: r1
```

# Configuración Elasticsearch (elasticsearch.yml)

- `path.data` Directorio donde guarda los datos de los índices
- `path.logs` Directorio donde guarda los logs
- Fundamental para migraciones a nuevas versiones de Elasticsearch

```
# Path to directory where to store the data (separate  
multiple locations by comma):
```

```
path.data: /path/to/data
```

```
# Path to log files:
```

```
path.logs: /path/to/logs
```

# Configuración Elasticsearch (elasticsearch.yml)

- `bootstrap.memory_lock`  
Define si la memoria asignada en “jvm.options” está bloqueada
- Evita que se ejecute swapping sobre la memoria asignada a elastic

```
#  
# Lock the memory on startup:  
#  
bootstrap.memory_lock: true  
#  
# Make sure that the heap size is set to about half  
the memory available  
# on the system and that the owner of the process is  
allowed to use this  
# limit.  
#  
# Elasticsearch performs poorly when the system is  
swapping the memory.  
#
```

# Configuración Elasticsearch(elasticsearch.yml)

- `node.max_local_storage_nodes`  
Cantidad de nodos que pueden correr en el mismo servidor
- `node.master` Habilita el nodo para ser maestro
- `node.data` Habilita el nodo como nodo de datos
- Por defecto la configuración viene en **true** para los tres tipos requeridos

```
node.max_local_storage_nodes: 2
```

```
node.master: true
```

```
node.data: true
```

```
node.ingest: true
```





# Configuración Elasticsearch(elasticsearch.yml)

- `gateway.recover_after_nodes`  
Cantidad de nodos a esperar para reiniciar el cluster, no tiene en cuenta el “**after\_time**”
- `gateway.expected_nodes` Cantidad de nodos a esperar para reiniciar el cluster
- `gateway.recover_after_time`  
Tiempo que va a esperar los nodos para el reinicio del cluster

```
gateway.recover_after_nodes: 3

gateway.expected_nodes: 5

gateway.expected_master_nodes: 2

gateway.expected_data_nodes: 4

gateway.recover_after_time: 5

gateway.recover_after_master_nodes: 3

gateway.recover_after_data_nodes: 2
```

# Configuración Elasticsearch(elasticsearch.yml)

- `discovery.seed_hosts` Declara la cantidad de nodos/instancias que van a integrar el cluster
- `cluster.initial_master_nodes` Selecciona una lista de nodos/instancias de las que se depende para ser elegibles como master en el cluster

```
# Pass an initial list of hosts to perform discovery
when this node is started:
# The default list of hosts is ["127.0.0.1", "[::1]"]
#
discovery.seed_hosts: ["host1", "host2"]
#
# Bootstrap the cluster using an initial set of
master-eligible nodes:
#
cluster.initial_master_nodes: ["node-1", "node-2"]
```

# Index Template: Index Patterns

- `index_patterns` Patrones a los cuales se aplica el template
- `settings` Define configuraciones generales de los índices, cantidad de shards y cantidad de réplicas

```
{  
  "index_patterns": ["te*", "bar*"],  
  "settings": {  
    "number_of_shards": 1  
  }  
}
```

# Index Template: \_source

- `_source` Define si se guarda el cuerpo original de documento JSON
- Al deshabilitarlo se ahorra espacio pero limita las queries
- Se lo puede incluir parcialmente si fuese necesario

```
"mappings": {  
  "_source": {  
    "enabled": false  
  },  
  "properties": {  
    "host_name": {  
      "type": "keyword"  
    },  
    "created_at": {  
      "type": "date",  
      "format": "EEE MMM dd HH:mm:ss Z yyyy"  
    }  
  }  
}
```

# Index Template: Properties

- `properties` Define el tipo de datos de los campos que van a formar el índice
- Evita la duplicación de campos, y define nuestros campos full text
- Define tipos de datos especiales para queries especializadas

```
"properties": {  
  "host_name": {  
    "type": "keyword"  
  },  
  "created_at": {  
    "type": "date",  
    "format": "EEE MMM dd HH:mm:ss Z yyyy"  
  }  
}
```

# Queries

# Tipos de Queries

- Leaf Query Clauses
  - Incluye operaciones como “term”, “range”, “match”, etc.
  - Devuelven el resultado de una query específica.
- Compound Query Clauses
  - Se componen de dos o más queries de las mencionadas en el punto anterior.
  - Retornan resultados específicos y realizar queries más complejas.



# Queries

- Match Query
- Multimatch Query
- Term Query
- Wildcard Query
- Fuzzy Query
- Bool Query
- Specialized Queries
- Span Queries
- Aggregation
- Otros...



# Match Query

- Búsquedas full text de Elasticsearch
- Recibe como parámetros el campo y la cadena a buscar en el mismo
- El resultado va a contener todos los documentos que contengan la cadena buscada en el campo
- Si cometemos errores de tipeo, va a buscar al índice y nos va devolver los campos, calculando la distancia de *Damerau-Levenshtein*



# Match Query

- 1er Query: Va a devolver todos los documentos que contengan en su campo `action` "CREATE USER".
- 2da Query: Va a devolver todos los documentos que contengan en su campo `action` "CREATE USOR".
- 3er Query: Devuelve todos los documentos que contienen "USER" en la frase del campo `action`.

```
#Match Query escrita correctamente
GET logstash-2019.06.24-000001/_search
{
  "query": {
    "match": {
      "action": "CREATE USER"
    }
  }
}

#Match Query escrita con un error
GET logstash-2019.06.24-000001/_search
{
  "query": {
    "match": {
      "action": "CREATE USOR"
    }
  }
}

#Match Query escrita parcialmente
GET logstash-2019.06.24-000001/_search
{
  "query": {
    "match": {
      "action": "USER"
    }
  }
}
```

# Match\_All

- Devuelve todo el contenido del índice “my\_index”
- Si el índice a consultar fuera muy grande se ajusta a la ventana de resultados máxima seteada en la configuración del mismo

```
GET /my_index/_search
{
  "query":{"match_all": {}}
}
```

# Multimatch Query

- Análogo a las match query con múltiples campos
- Compara la cadena seteada en query contra los campos en el array "fields"

```
42
43 #Multimatch query
44 GET logstash-2019.06.24-000001/_search
45 {
46   "query":{
47     "multi_match": {
48       "query": "USER",
49       "fields": ["action","thread"]
50     }
51   }
52 }
53
```

# Term-Level Queries

- Buscan documentos en base al valor preciso de un atributo.
- No realizan un análisis de los terminos de busqueda.
- Los resultados de las term queries coinciden exactamente con el valor preciso buscado.



# Term Queries

- Evalúan por igualdad un campo con su valor preciso
- No evalúa si valor fue cargado parcialmente, solo por igualdad

```
GET /kibana_sample_data_ecommerce/_search
{
  "query": {
    "term": {
      "currency": {
        "value": "EUR"
      }
    }
  }
}
```

# Terms Queries

- Evalúan múltiples valores contra el mismo campo
- Si el valor del campo está dentro del array retorna el documento

```
GET /kibana_sample_data_ecommerce/_search
{
  "query": {
    "terms": {
      "customer_gender": ["MALE", "FEMALE"]
    }
  }
}
```

# Wildcard Queries

- Matchea un campo contra un patrón
- Devuelve los documentos que cumplan con el patrón asignado en value

```
#WildCard Query
GET logstash-2019.06.24-000001/_search
{
  "query": {
    "wildcard": {
      "user": {
        "value": "*man",
        "boost": 1.0
      }
    }
  }
}
```



# Fuzzy Query

- Admite errores de tipeo
- Busca en el índice evaluando por la distancia de *Damerau-Levenshtein* si el valor del campo es aceptable

```
#Fuzzy Query
GET logstash-2019.06.24-000001/_search
{
  "query": {
    "fuzzy": {
      "user": {
        "value": "botman",
        "boost": 1.0
      }
    }
  }
}
```

# Bool Queries

- Ejecutan múltiples queries simultáneamente
- Evalúan por negación, veracidad y requisitos mínimos
- `filter` realiza la búsqueda omitiendo el cálculo del `score`

```
#Bool Queries
GET logstash-2019.06.24-000001/_search
{
  "query": {
    "bool": {
      "must": {
        "term": { "user" : "batman" }
      },
      "filter": {
        "term": { "logtype" : "login" }
      },
      "must_not": {
        "range": {
          "duration" : { "gte" : 0, "lte" : 1 }
        }
      },
      "should": [
        { "term" : { "code" : 202 } }
      ],
      "minimum_should_match" : 1,
      "boost" : 1.0
    }
  }
}
```

# Specialized Queries

- Destinadas a resolver casos muy específicos
- Requieren Data-Types especiales, especificados en el `index template`
- Permite realizar consultas en diversos formatos

```
GET /_search
{
  "query": {
    "more_like_this" : {
      "fields" : ["title", "description"],
      "like" : "Once upon a time",
      "min_term_freq" : 1,
      "max_query_terms" : 12
    }
  }
}
```

# Span Queries

- Consultas posicionales de bajo nivel
- Control experto sobre el orden y los términos especificados
- Se utilizan para consultas muy específicas sobre patentes y documentos legales

```
GET /_search
{
  "query": {
    "span_first" : {
      "match" : {
        "span_term" : { "user" : "kimchy" }
      },
      "end" : 3
    }
  }
}
```

# Elastic SQL

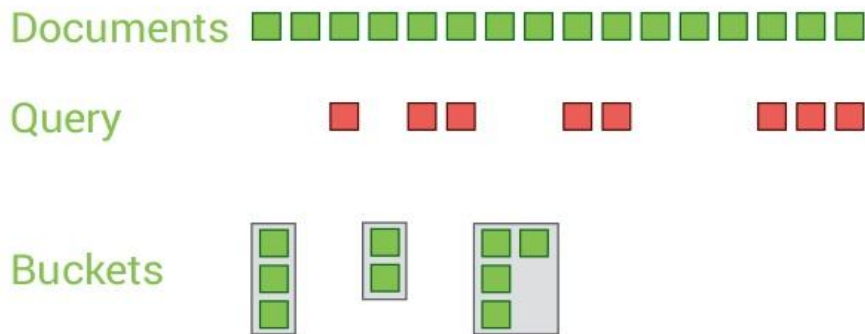
- Lenguaje adicional SQL para hacer consultas
- `format` Formato en que retorna el resultado

```
POST /_sql?format=txt
{
  | "query": "SELECT * FROM library ORDER BY page_count DESC LIMIT
    5"
}
```

# Aggregation Bucketing

- No agregan datos
- Filtra los documentos y los separa en buckets
- API que permite consultar cada bucket por separado

## Aggregation & Filtering



# Aggregation Bucketing

GET emails/\_search

```
{
  "size": 0,
  "aggs": {
    "interactions": {
      "adjacency_matrix": {
        "filters": {
          "grpA": { "terms": { "accounts": ["hillary", "sidney"] } },
          "grpB": { "terms": { "accounts": ["donald", "mitt"] } },
          "grpC": { "terms": { "accounts": ["vladimir", "nigel"] } }
        }
      }
    }
  }
}
```



```
"buckets": [ "buckets": [
  {
    "key": "grpA", "key": "grpA",
    "doc_count": 2 "doc_count": 2
  },
  {
    "key": "grpA&grpB", "key": "grpA&grpB",
    "doc_count": 1 "doc_count": 1
  },
  {
    "key": "grpB", "key": "grpB",
    "doc_count": 2 "doc_count": 2
  },
  {
    "key": "grpB&grpC", "key": "grpB&grpC",
    "doc_count": 1 "doc_count": 1
  },
  {
    "key": "grpC", "key": "grpC",
    "doc_count": 1 "doc_count": 1
  }
]
```

# Aggregation Metrics

- Agregan métricas sobre los documentos devueltos por la query: promedios, sumas, conteos, etc.
- `single-value numeric` agrega un solo valor
- `multi-value numeric` agrega múltiples valores

```
id: 1      id: 4  
tag: foo   tag: foo  
price: 10  price: 74
```



← metric aggregator

```
min: 10  
max: 74  
avg: 42  
...
```



# Aggregation Metrics

```
#Bool Queries
GET logstash-2019.06.24-000001/_search
{
  "query": {
    "bool": {
      "must": {
        "term": { "user" : "batman" }
      },
      "filter": {
        "term": { "logtype" : "login" }
      },
      "must_not": {
        "range": {
          "duration": { "gte" : 0, "lte" : 1 }
        }
      },
      "should": [
        { "term": { "code" : 202 } }
      ],
      "minimum_should_match" : 1,
      "boost" : 1.0
    }
  },
  "aggs": {
    "byUser": {
      "terms": {
        "field": "user.keyword"
      }
    }
  }
}
```



```
"aggregations" : {
  "byUser" : {
    "doc_count_error_upper_bound" : 0,
    "sum_other_doc_count" : 0,
    "buckets" : [
      {
        "key" : "Batman",
        "doc_count" : 3
      }
    ]
  }
}
```

# Aggregation Pipeline

- Funcionan con los resultados de otra agregación
- Padre: Reciben el resultado de una agregación anterior y puede calcular nuevos buckets
- Hermano: Son capaces de computar múltiples agregaciones al mismo nivel

```
{
  "aggs" : {
    "sales_per_month" : {
      "date_histogram" : {
        "field" : "date",
        "calendar_interval" : "month"
      },
      "aggs": {
        "sales": {
          "sum": {
            "field": "price"
          }
        }
      }
    },
    "max_monthly_sales": {
      "max_bucket": {
        "buckets_path": "sales_per_month>sales"
      }
    }
  }
}
```

# Delete API

- API para remover documentos en función de su id
- Acepta parámetros de enrutamiento y versionado
- Permite el uso de condicionales según número de secuencia o término primario

```
DELETE /twitter/_doc/1
```

```
{
  "_shards" : {
    "total" : 2,
    "failed" : 0,
    "successful" : 2
  },
  "_index" : "twitter",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 2,
  "_primary_term": 1,
  "_seq_no": 5,
  "result": "deleted"
}
```

# Delete by Query API

- Realiza múltiples búsquedas y eliminaciones masivas
- En caso de fallar 10 veces se aborta el “delete\_by\_query” pero no hace rollback sobre lo que eliminó
- Permite hacer slicing con los ids de los documentos a eliminar

```
POST twitter/_delete_by_query
{
  "query": { ❶
    "match": {
      "message": "some message"
    }
  }
}
```

# Update API

- Actualizar un documento en función a su id
- Upserts seteando el parámetro “doc\_as\_upsert” en true
- Variedad de updates a través de scripting

```
PUT test/_doc/1
{
  "counter" : 1,
  "tags" : ["red"]
}

POST test/_update/1
{
  "script" : {
    "source": "ctx._source.counter += params.count",
    "lang": "painless",
    "params" : {
      "count" : 4
    }
  }
}
```

# Update by Query API

- Actualización de documentos en función al resultado de una query
- Soporta scripting para actualización de campos
- Agregar nuevos campos en un documento

```
POST twitter/_update_by_query
{
  "script": {
    "source": "ctx._source.likes++",
    "lang": "painless"
  },
  "query": {
    "term": {
      "user": "kimchy"
    }
  }
}
```

# Bulk API

- Es la **API de operaciones masivas por excelencia**
- Soporta operaciones de `index`, `create`, `update` y `delete`
- Capacidad de manejar paralelismo y configurar tamaño de bulk para ajustar la performance y realizar un tuning sobre las operaciones en el cluster

# Bulk API

- Cada lenguaje tiene su soporte en el conector correspondiente (por ej: JAVA, Python)
- Un ejemplo:

```
POST _bulk
{ "index" : { "_index" : "test", "_id" : "1" } }
{ "field1" : "value1" }
{ "delete" : { "_index" : "test", "_id" : "2" } }
{ "create" : { "_index" : "test", "_id" : "3" } }
{ "field1" : "value3" }
{ "update" : { "_id" : "1", "_index" : "test" } }
{ "doc" : { "field2" : "value2" } }
```



# Preguntas?