elasticsearch.

SEARCH MADE EASY FOR (WEB) DEVELOPERS

Alexander Reelsen
alexander@reelsen.net
@spinscale

AGENDA

- What is so important about search?
- Scalability, Sharding & Replication
- Configuration, Mapping & Analyzers
- Querying, Facetting, Percolation
- Modules, Plugins, Rivers & Tools
- Production setup & living in the trenches

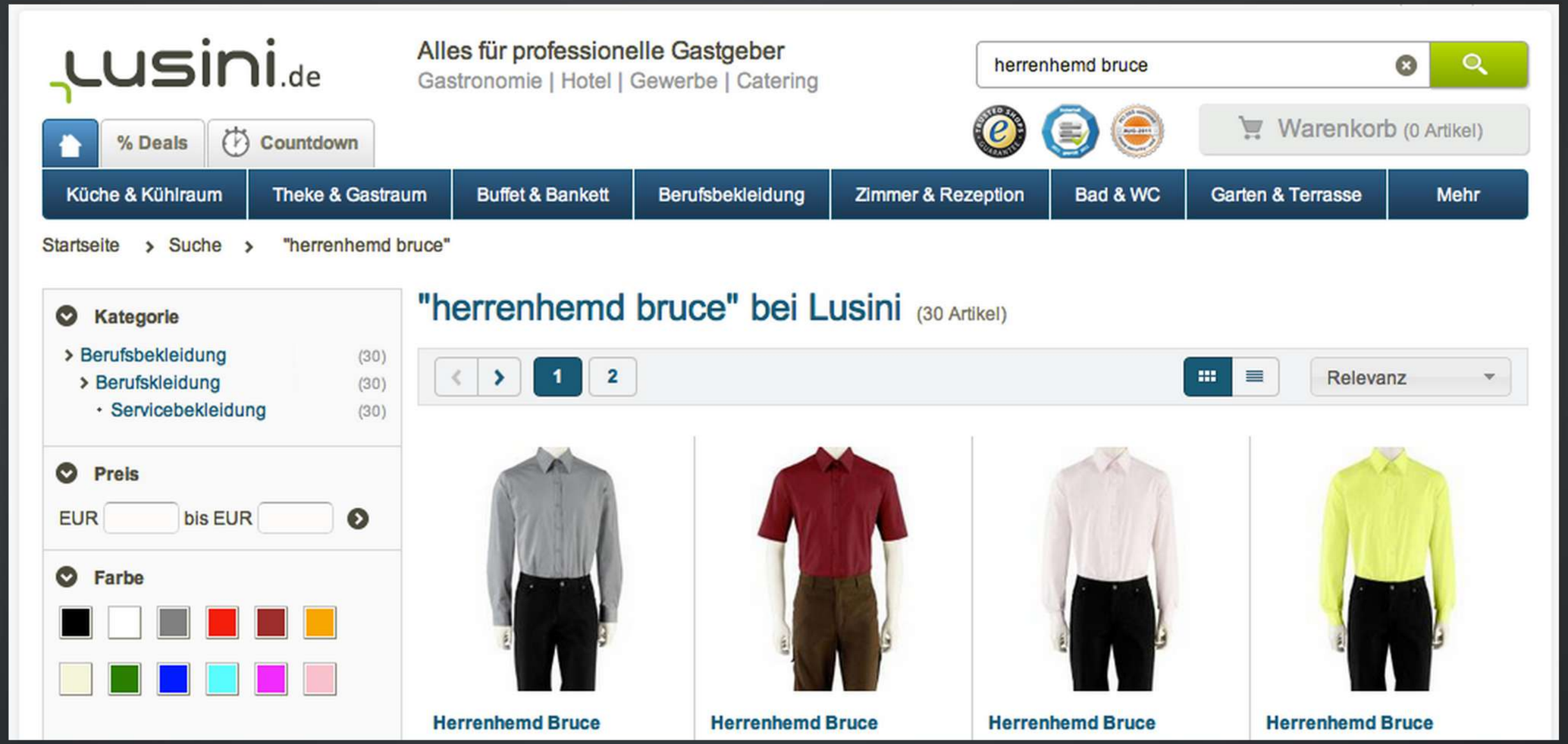
ABOUT ME - ALEXANDER REELSEN

- Studied information systems
- 10 years linux system engineering, converted to software engineering
- Web framework enthusiast, fed up with complex java environment for simple webapps
- Other interests: Scaling web architectures, Web 2.0 (nosql, search)
- Author of Play framework cookbook
- Working at Lusini GmbH, building a b2b ecommerce platform
- Streetball/Basketball

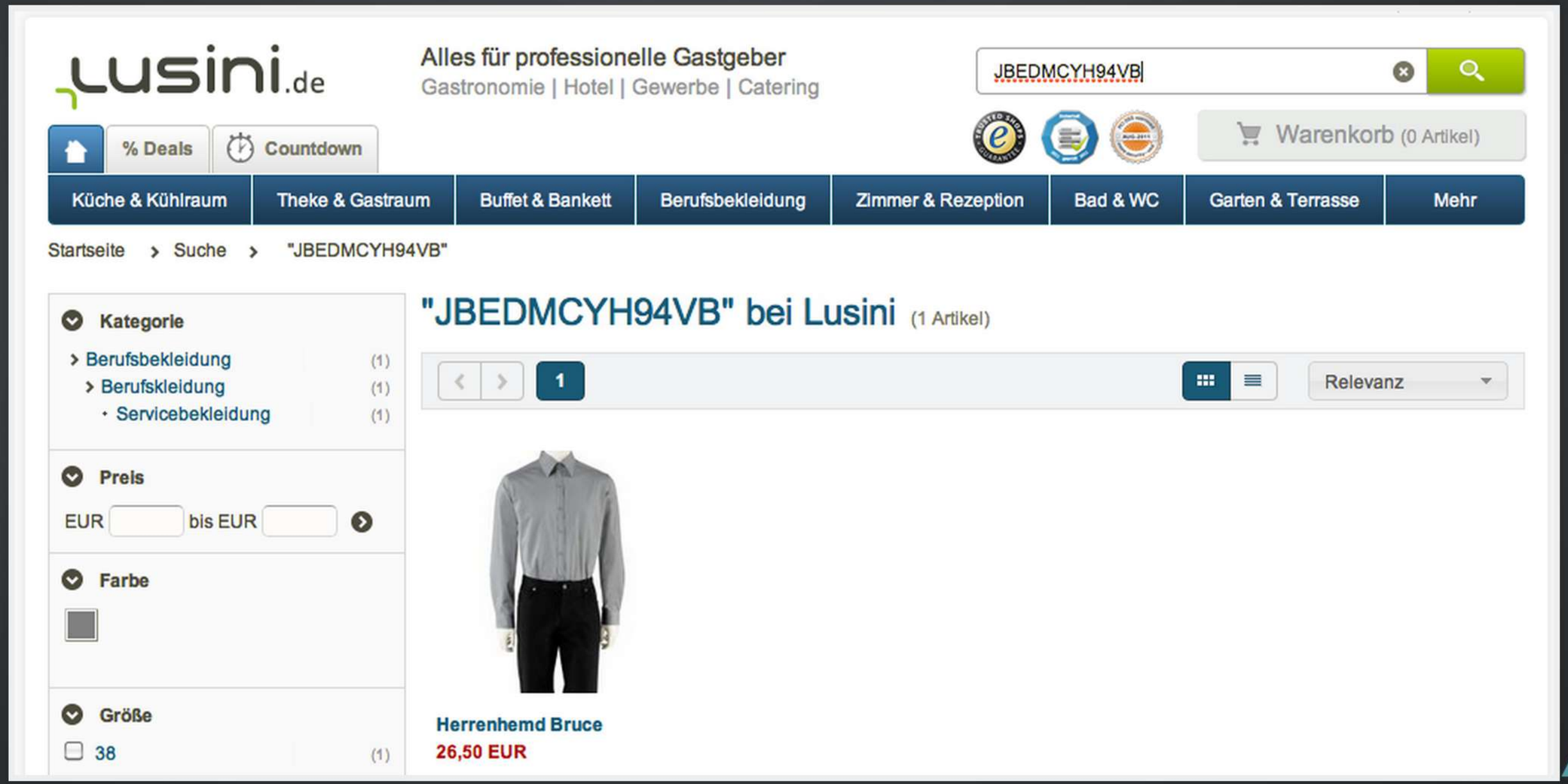
WHAT IS SO IMPORTANT ABOUT SEARCH?

- No search, no google, no bing, no twitter, no amazon, no ebay, ...
- Functional requirements: Relevance (finds the *right* stuff)
- Non-functional requirements: Scalability, performance, concurrent updates
- Solutions: Google commerce search, Sphinx, SearchBlox, Solr, elasticsearch, IndexTank, Sensei DB

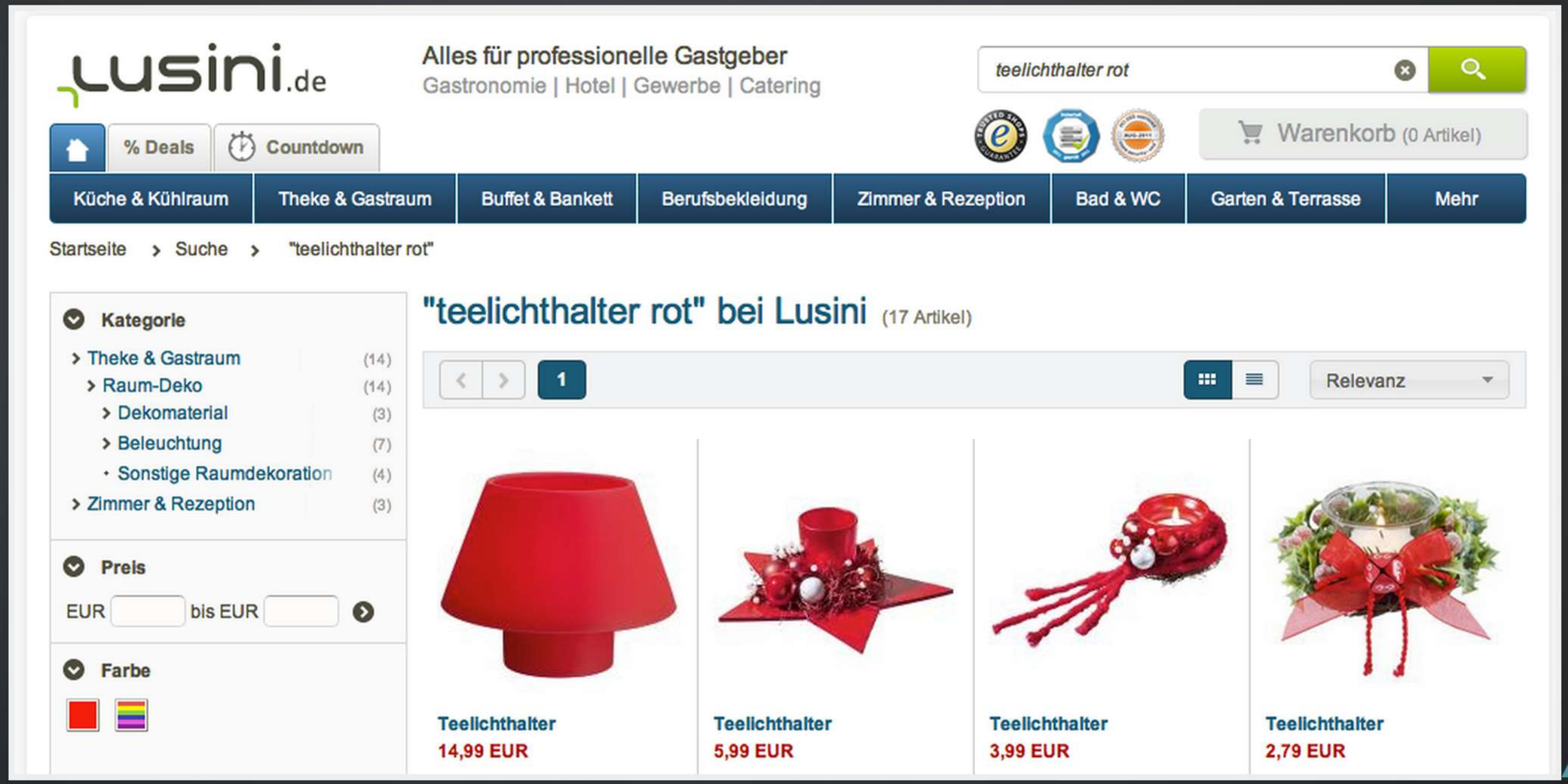
SEARCH IS MORE THAN TEXT SEARCH



SEARCH MUST SEARCH FOR IDS



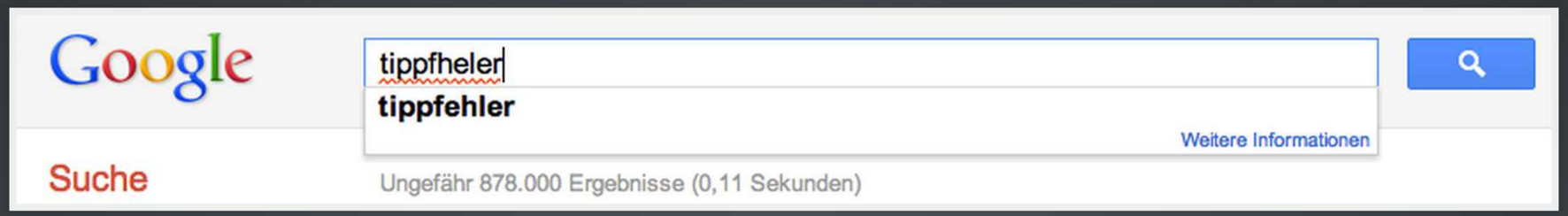
SEARCH MUST SEARCH FOR COLORS



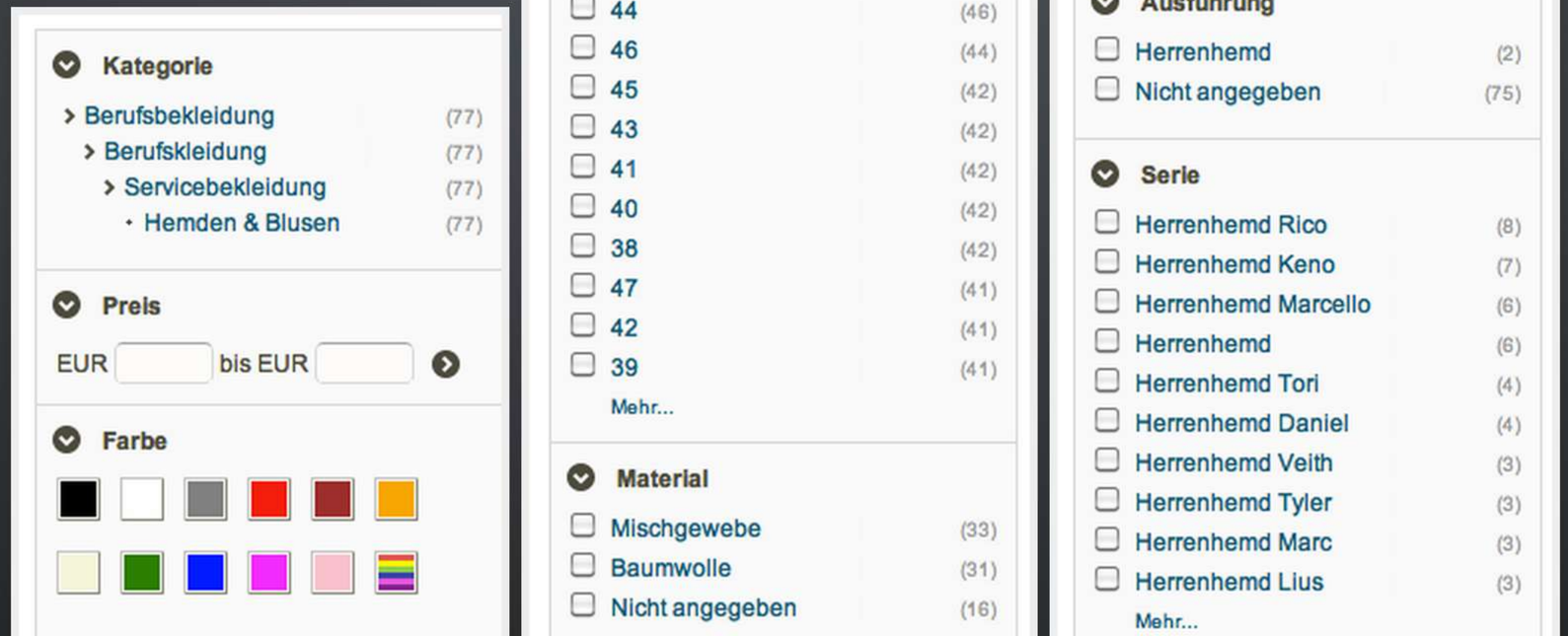
SEARCH MUST SEARCH FOR BRANDS

SEARCH MUST ADVICE

SEARCH MUST BE INTELLIGENT



SEARCH MUST AGGREGATE



WHY AN OWN SEARCH ENGINE?

- Because you can - telling this your CTO doesn't work.
- Your data, your search - noone spying...
- Customize your search - Rank your own style
- Customize your data - Extend your search?!
- Best support and in-sourced know-how - Lower TCO
- No blackbox - Lower TCO

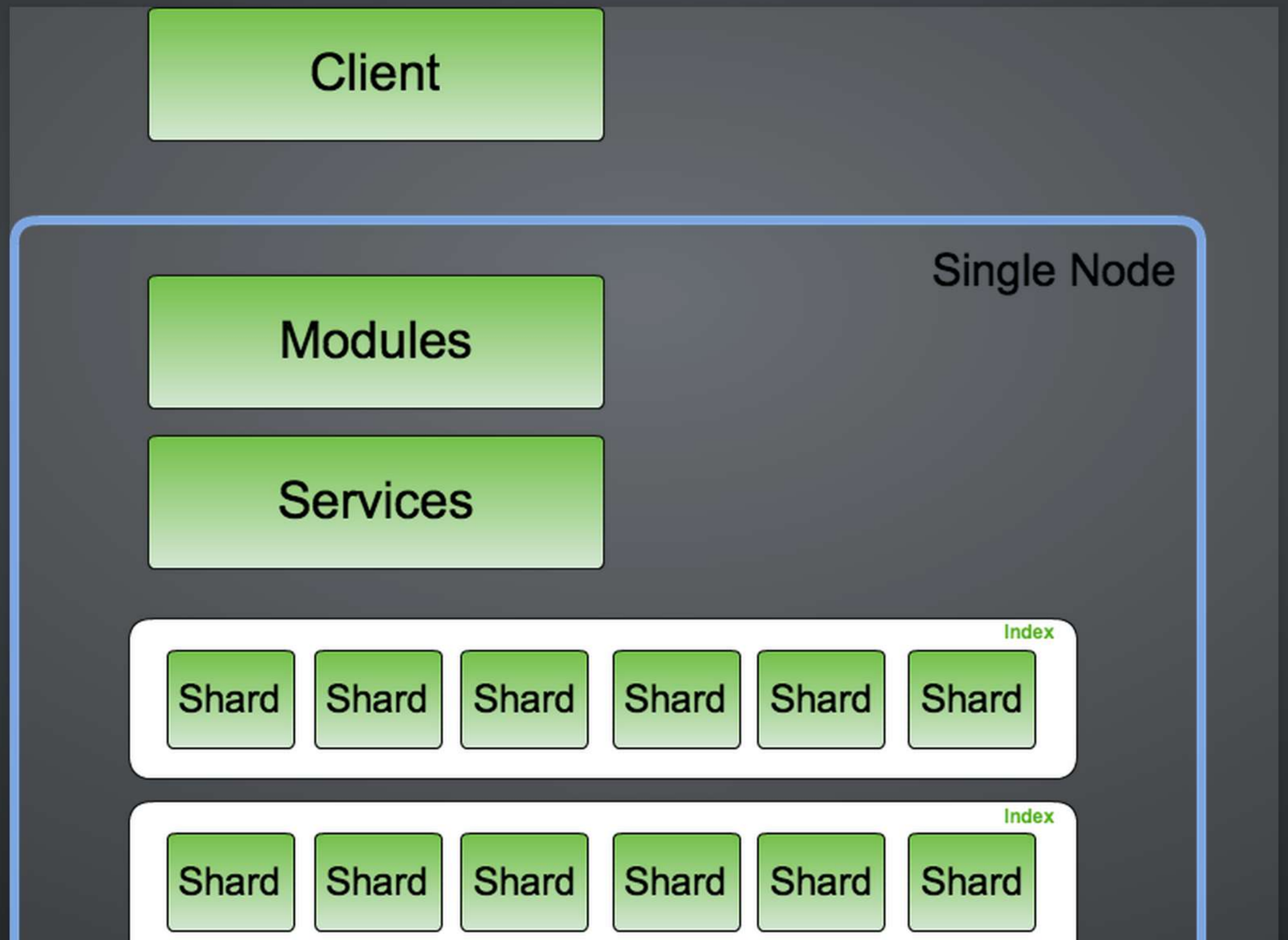
ELASTICSEARCH IN TEN SECONDS

- Java, based on Apache Lucene
- Scales out, replicates, shards, fail-over
- Schema-free
- Document-based
- Every interaction can be done via HTTP & JSON
- References: Mozilla, StumbleUpon, Sony, Infochimps, Assistly, Klout

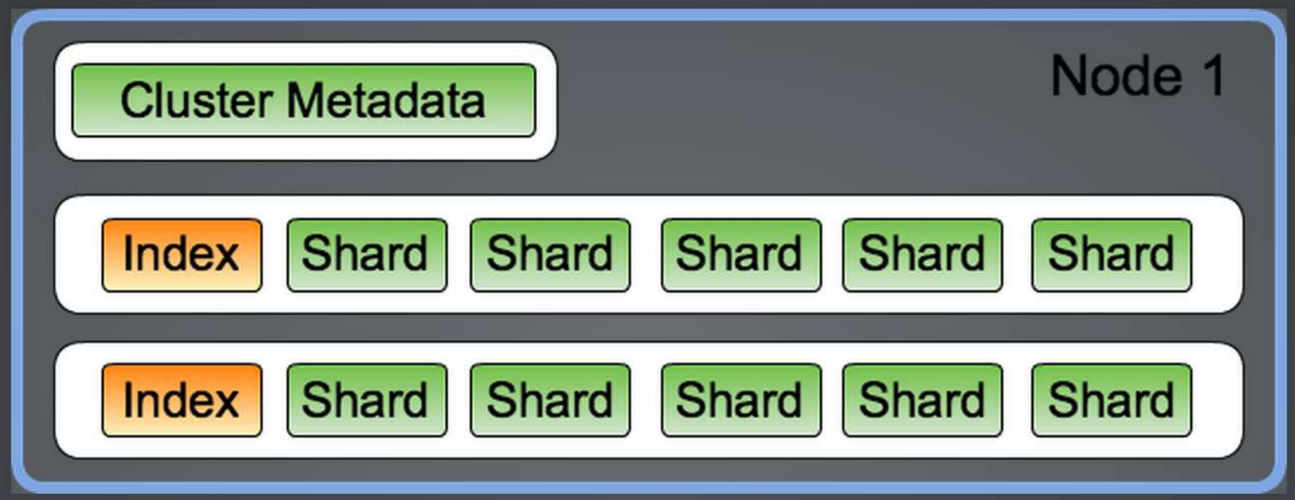
STANDING ON THE SHOULDERS OF GIANTS

- Lucene, JBoss Netty, Jackson, log4j
- Google Guice, Google Guava, MVEL, Groovy
- Jodatime, JLine, snakeyaml
- hamcrest, testng
- sigar via JNA

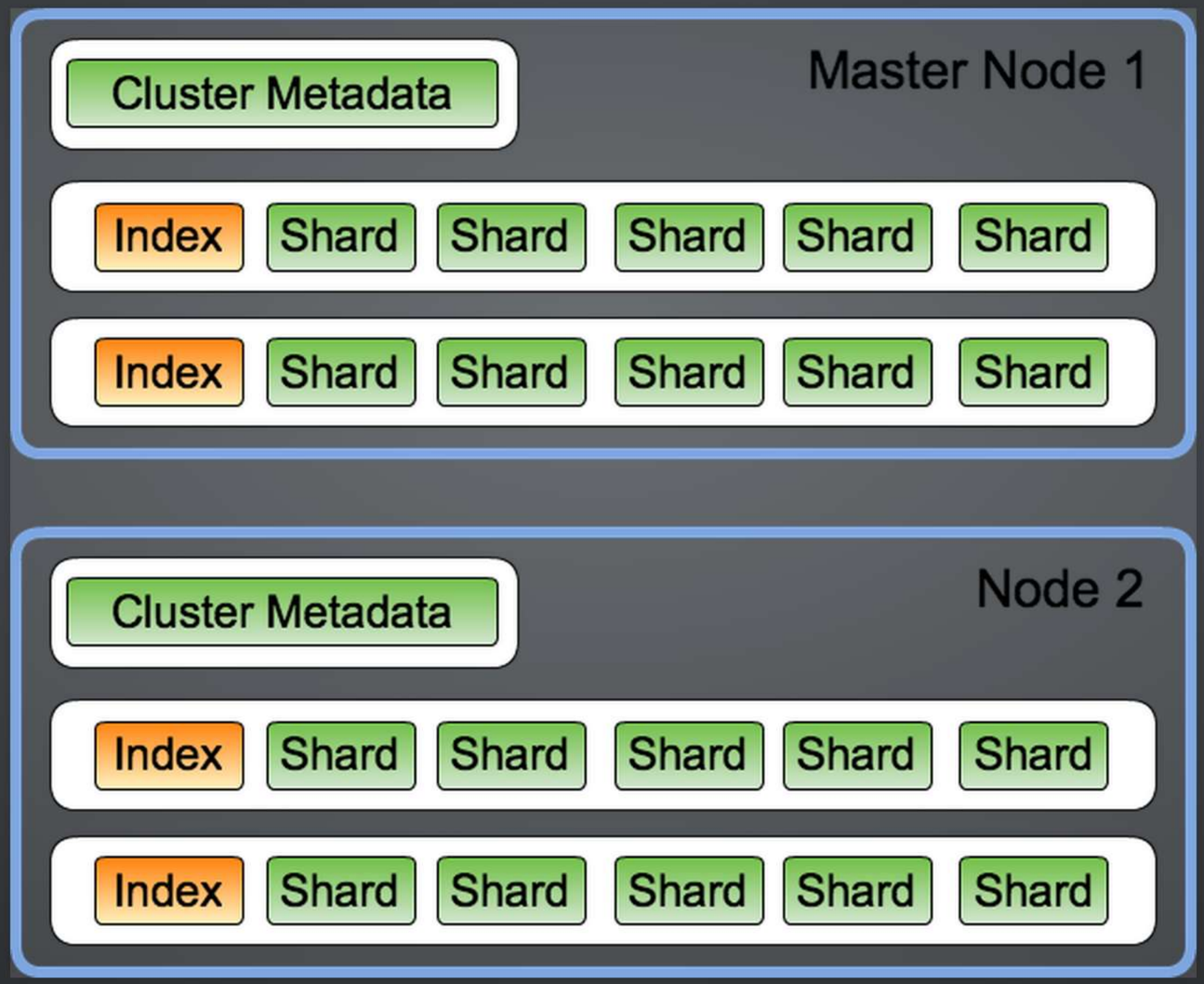
ELASTICSEARCH ARCHITECTURE



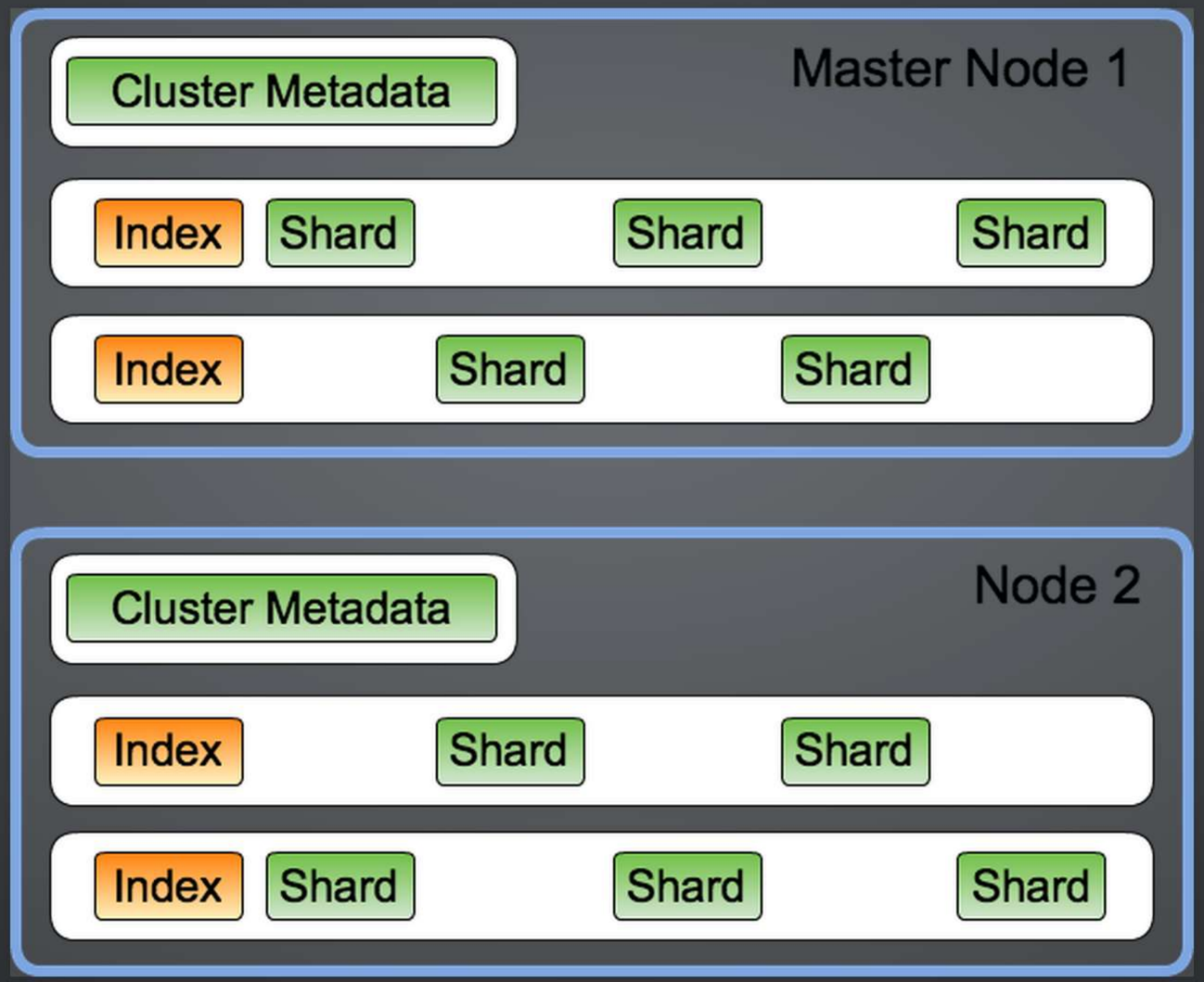
SINGLE NODE SETUP



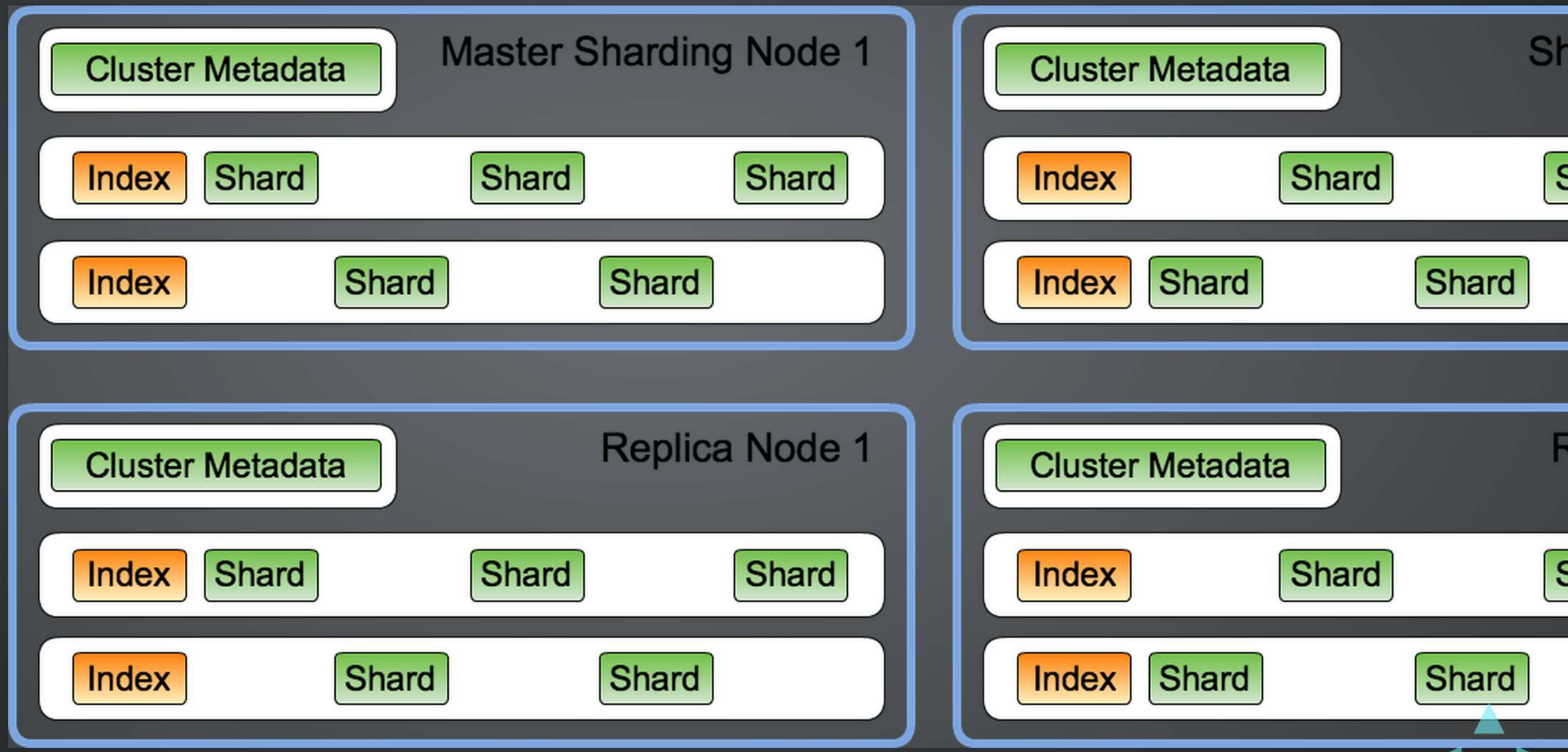
REPLICATION



SHARDING


Cluster Metadata
Master Node 1
Index
Shard
Shard
Shard
Index
Shard
Shard
Cluster Metadata
Node 2
Index
Shard
Shard
Index
Shard
Shard
Shard

REPLICATION & SHARDING



INSTALLATION - TAKES TWO MINUTES

- Download from github or elasticsearch.org zip file
- Unzip
  `unzip elasticsearch-0.19.1.zip`
- Run
  `elasticsearch/bin/elasticsearch -f`
- Check by connecting with your browser to http://localhost:9200

CONFIGURATION

- `config/elasticsearch.yml` or `config/elasticsearch.json`
- Application-wide settings (zen discovery, available analyzers)
- index default configurations (number of shards)
- Seperate logging file: `config/logging.yml` (simplified log4)

CONFIGURATION

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

http:
  max_content_length: 100000

index:
  number_of_shards: 1

  analysis:
    analyzer:
      default:
        type: standard

    lowercase_analyzer:
      type: custom
      tokenizer: standard
      filter: [standard, lowercase]
```

DATA REPRESENTATION IN JSON

```
{
        "id": "1",
        "name" : "MacBook Air",
        "price": 1099,
        "descr" : "Some lengthy never-read description",
        "attributes" : {
                "color" : "silver",
                "display" : 13.3,
                "ram" : 4
        }
}
```

INDEX YOUR PRODUCT

```
curl -X PUT localhost:9200/products/product/1 -d '{
        "id": "1",
        "name" : "MacBook Air",
        "price": 1099,
        "descr" : "Some lengthy never-read description",
        "attributes" : {
        "color" : "silver",
        "display" : 13.3,
        "ram" : 4
        }
}'
```

http://localhost:9200/products/product/1

JSON AS QUERY LANGUAGE

http://host:9200/products/product/_search

```
{ "query" : { "term" : { "name": "MacBook Air" }}}
```

```
{ "query" : { "prefix" : { "name": "Mac" }}}
```

```
{ "query" : { "range" : { "price" : { "from" : 1000, "to": 2000 } } } }
```

```
{ "from": 0, "size": 10, "query" : { "term" : { "name": "MacBook Air" }}}
```

```
{ "sort" : { "name" :  { "order": "asc" } }, "query" : { "term" : { "name": "MacBook Air" }}}
```

JSON AS QUERY LANGUAGE

http://host:9200/products/product/_search


```
{ "took":206,"timed_out":false,
"_shards":{"total":1,"successful":1,"failed":0},
"hits":{ "total":1,"max_score":2.098612,
  "hits":[ {
    "_index":"products1","_type":"product","_id":"1",
    "_score":2.098612, "_source" : {
      "id": "1",
      "name" : "MacBook Air",
      "price": 1099,
      "descr" : "Some lengthy never-read description",
      "attributes" : {
        "color" : "silver",
        "display" : 13.3,
        "ram" : 4
      }
}}]}}
```


CONFIGURATION - MAPPING

- On indexing the JSON document is parsed and all data types are extracted
- Mapping fields to datatypes is done automatically on first indexing
- Can be configured on a per-type basis
- Strings can have their own analyzer
- Sample types: float, long, boolean, date (+formatting), object
- One field can have multiple fields analyzed differently (lowercase, query)

SAMPLE MAPPING

```
{
    "product": {
        "properties": {
            "ProductId":            { "type": "string", "index": "not_analyzed" },

            "ProductEnabled":       { "type": "boolean" },
            "PiecesIncluded":       { "type": "long" },
            "LastModified":         { "type": "date", "format": "yyyy-MM-dd HH:mm:ss.SSS" },

            "AvailableInventory":   { "type": "float" },
            "Price":                { "type": "float" },

            "LongDescription":      { "type": "string", "include_in_all" : true },

            "ProductName" : {
                "type" : "multi_field",
                "include_in_all" : true,
                "fields" : {
                    "ProductName":  { "type": "string", "index": "not_analyzed" },
                    "lowercase":    { "type": "string", "analyzer": "lowercase_analyzer" },
                    "suggest" :     { "type": "string", "analyzer": "suggest_analyzer" }
                }
            }
        }
    }
}
```

CONFIGURATION - ANALYZERS

- An analyzer consists of a Tokenizer and an arbitrary amount of filters
- Example:

```
suggest_analyzer:
  type: custom
  tokenizer: whitespace
  filter: [standard, lowercase, shingle]
```

- Stripping html code:

```
char_filter: html_strip
```

JAVA API - CREATING A CLIENT

```
Settings settings = ImmutableSettings.settingsBuilder().
    put("cluster.name", clusterName).build();

InetSocketTransportAddress addr =
    new InetSocketTransportAddress(host, port)

Client client = new TransportClient(settings).
    addTransportAddress(addr);
```

STARTING AN EMBEDDED SERVER

```
File config = new File("elasticsearch-local.yml");
String config = FileUtils.readFileToString(config);

Builder settingsBuilder = ImmutableSettings.settingsBuilder().
    loadFromSource(config);

Node node = NodeBuilder.nodeBuilder().
    settings(settingsBuilder).node();

Client client = node.client();
```

EXECUTING A QUERY

```
CountRequestBuilder countRequestBuilder =
    new CountRequestBuilder(client)
        .setQuery(QueryBuilders.termQuery("foo", "bar"))
        .setIndices("products")
        .setTypes("product");

CountResponse response =
    countRequestBuilder.execute().actionGet();
long count = response.count();
```

SEARCH API OVERVIEW

- Index, Delete, Delete-By-Query, Get, Multiget, Bulk
- Search/Count queries (term query, prefix query, id, fuzzy…)
- Geo-based queries, TTL
- More like this, Highlighting
- Facetting, Percolation, Scripting

SEARCH - FACETTING

- Facetting adds aggregated information to a standard search query
- Term: Group results by a term
- Range: Group by price or date ranges
- Histogram: Group results in equally sized buckets, also as date histogram
- Statistical: Include statistical data like min, max, sum, avg & some more
- Geo distance: Group results around a coordinate

FACET QUERY

```
SearchRequestBuilder searchRequestBuilder = new SearchRequestBuilder(client)
    .setIndices("products")
    .setTypes("product");

searchRequestBuilder.setQuery(QueryBuilders.prefixQuery("ProductName.suggest", "macbook"));

searchRequestBuilder.addFacet(FacetBuilders.termsFacet("categoryFacet").field("CategoryId"));

SearchResponse searchResponse = searchRequestBuilder.execute().actionGet();

TermsFacet facet = searchResponse.getFacets().facet(TermsFacet.class, "categoryFacet");
List entries = facet.entries();
String term = entries.get(0).term();
int count = entries.get(0).count();
```

SEARCH - SCRIPTING

THIS IS WHERE YOUR OWN INTEGRATION BEATS ALL OTHERS

- Score down all your products without an image
- Dont include them in your results
- Score up products by an attribute like its product quality or stock
- Apply math operations on fields to change score

SEARCH API - PERCOLATION

IMPLEMENT A PRICE AGENT FOR FREE!

```
curl -X PUT localhost:9200/_percolator/products/pricecheck -d '{
"query" : {
  "bool" : {
    "must" : { "term" : { "name" : "MacBook Air" } },
    "must" : { "range" : { "price" : { "from" : 200, "to" : 999 } } }
    }
  }
}'
{"ok":true,"_index":"_percolator","_type":"products","_id":"pricecheck","_version":1}

curl -X PUT 'localhost:9200/products/product/1?percolate=*' -d '{ "price": 1000, "name" : "MacBook Air"
 }'
{"ok":true,"_index":"products","_type":"product","_id":"1","_version":1,"matches":[ ]}

curl -X PUT 'localhost:9200/products/product/2?percolate=*' -d '{ "price": 999, "name" : "MacBook Air"
}'
{"ok":true,"_index":"products","_type":"product","_id":"2","_version":1,"matches":["pricecheck"]}
```

INDICES API

- Aliases, Analyze
- Create, Delete, Exists, Open, Close, Optimize, Refresh, Flush, Settings
- Get, Put, Delete Mapping
- Get, update settings
- Snapshot
- Index templates (mappings + settings)
- Stats, Status
- Segments, Clear cache

CLUSTER API

- Health, State, Settings
- Nodes Info, Nodes Stats, Nodes Shutdown

MODULES

- REST, Thrift, Memcached, ZeroMQ
- JMX
- Scripting (MVEL, javascript, groovy, python, native)
- Discovery: EC2, Zen
- Cluster, Indices, Network, Transport

PLUGINS

- Analysis: Smart Chinese, ICU, IK, Mmseg, Hunspell
- Transport: Memcached, Thrift, ZeroMQ, Servlet
- Scripting: javascript, groovy, python
- Site plugins: BigDesk, Elasticsearch Head
- Misc: Mapper attachments, Hadoop, AWS cloud, Mock Solr, Suggester, PartialUpdate

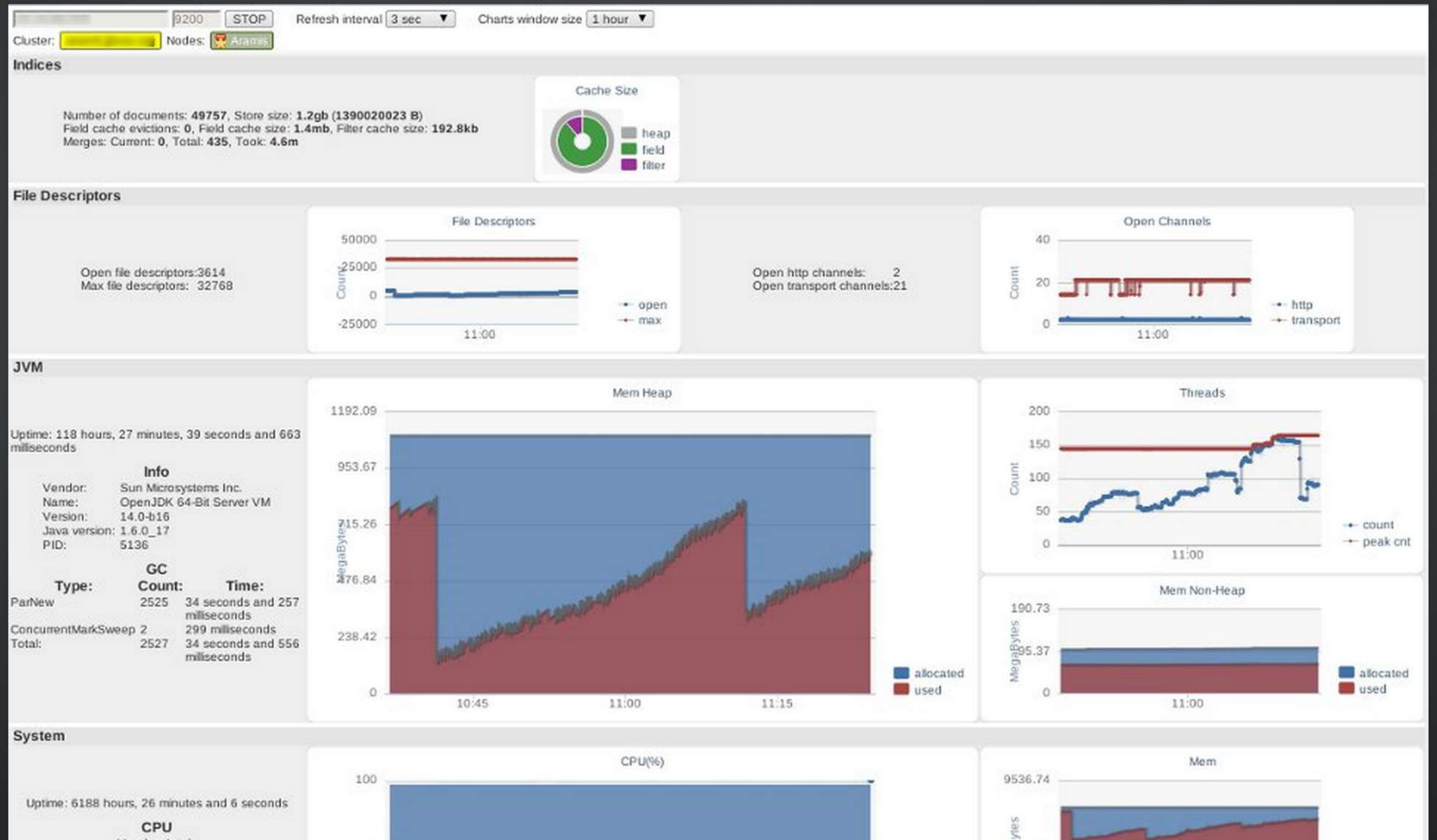
RIVERS

- Interface to import data into elasticsearch
- CouchDB, Wikipedia, Twitter, RabbitMQ
- RSS, MongoDB
- Hint: When writing your own river, make sure you are implementing streaming

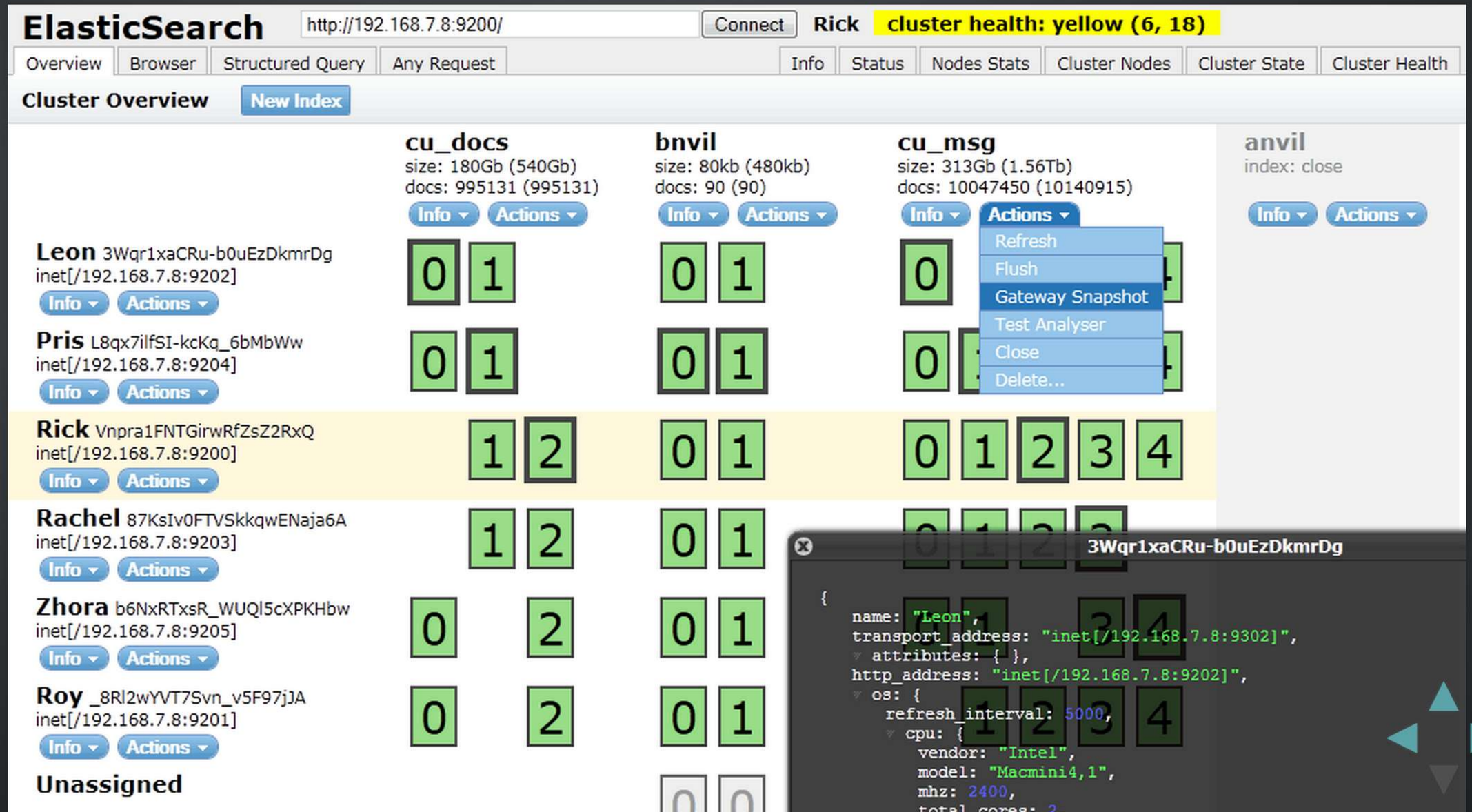
TOOLS

- BigDesk, Elasticsearch Head
- Chef, puppet
- RPMs and debian packages
- daikon CLI

BIGDESK SCREENSHOT



ELASTICSEARCH-HEAD SCREENSHOT



LANGUAGE SUPPORT & SOFTWARE

- java, groovy, python, perl, ruby, erlang, .net, clojure
- Integrations: grails, django, rails, catalyst, flume, terrastore, hadoop, symfony2, drupal, couchdb, play framework, node.js
- Software: Graylog2
- Elasticsearch as SaaS: bonsai.io

RUNNING IN PRODUCTION

- 220k products, one index, one shard (due to result grouping)
- Almost all queries have a big facetting query part (with filters)
- Don't expose your search engine to the internet!
- Write your own river
- Be prepared to upgrade every now and then

THANKS FOR LISTENING!

QUESTIONS?

Slides available at
http://spinscale.github.com/

alexander@reelsen.net
@spinscale

DOCUMENTATION & CREDITS

- elasticsearch.org
- @elasticsearch
- Elasticsearch google group

- Presentation done with reveal.js
- Cool zooming done with zoom.js