ElasticSearch

Introduction and Lessons Learned

WHAT DO I DO?

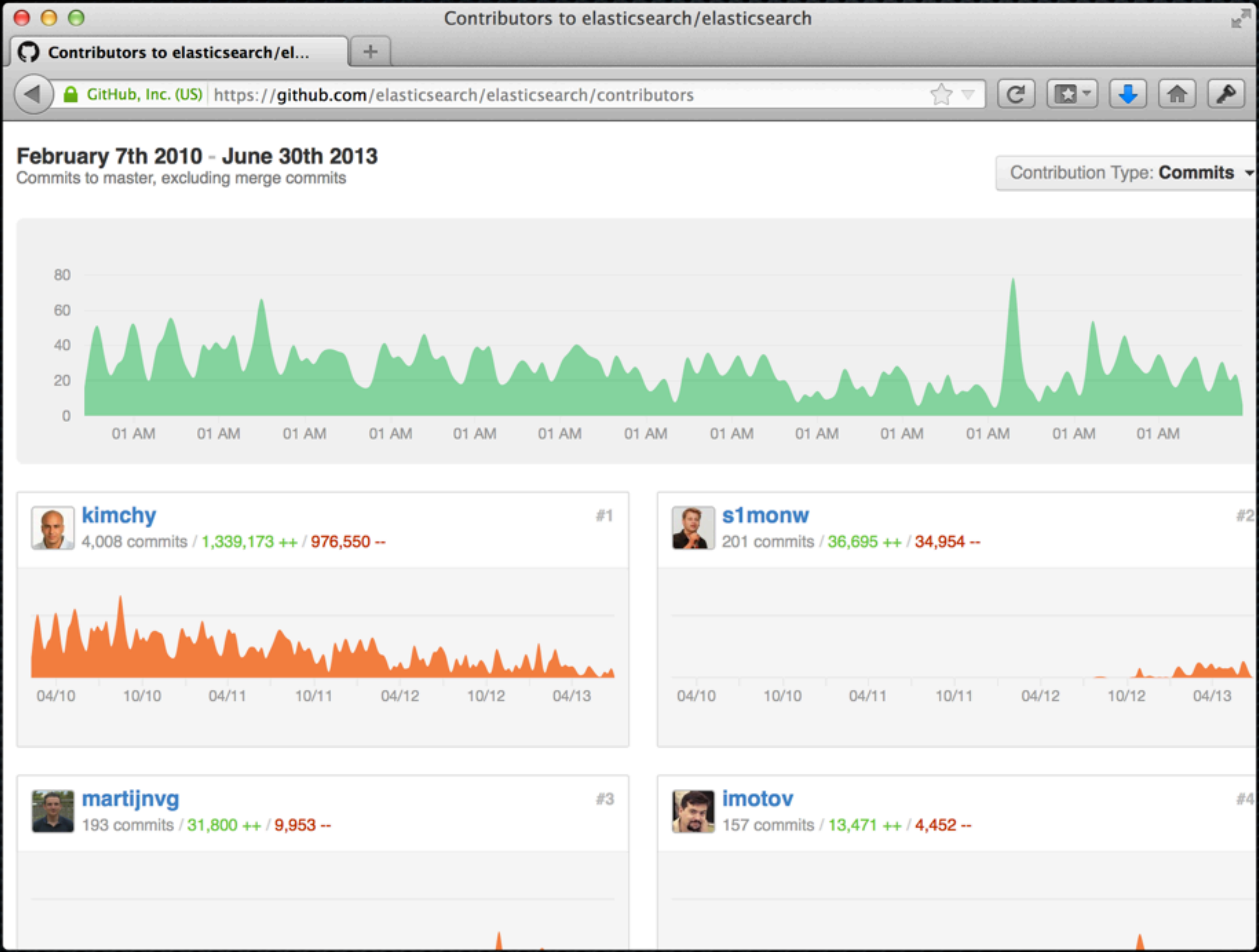
- Working as a senior Python developer for Artirix.
- Building backend systems and services.
- Organiser of Python Glasgow.

ARTIRIX.

Maximising the Value of Content, Data & Information

elasticsearch

- Open Source - Apache Licence.
- Backed by the ElasticSearch company.
- Careful feature development.
- Primary Author is Shay Banon.



Contributors to elasticsearch/elasticsearch

Contributors to elasticsearch/el...

GitHub, Inc. (US) https://github.com/elasticsearch/elasticsearch/contributors

February 7th 2010 - June 30th 2013

Commits to master, excluding merge commits

Contribution Type: **Commits**

kimchy #1

4,008 commits / 1,339,173 ++ / 976,550 --

s1monw #2

201 commits / 36,695 ++ / 34,954 --

martijnvg #3

193 commits / 31,800 ++ / 9,953 --

imotov #4

157 commits / 13,471 ++ / 4,452 --

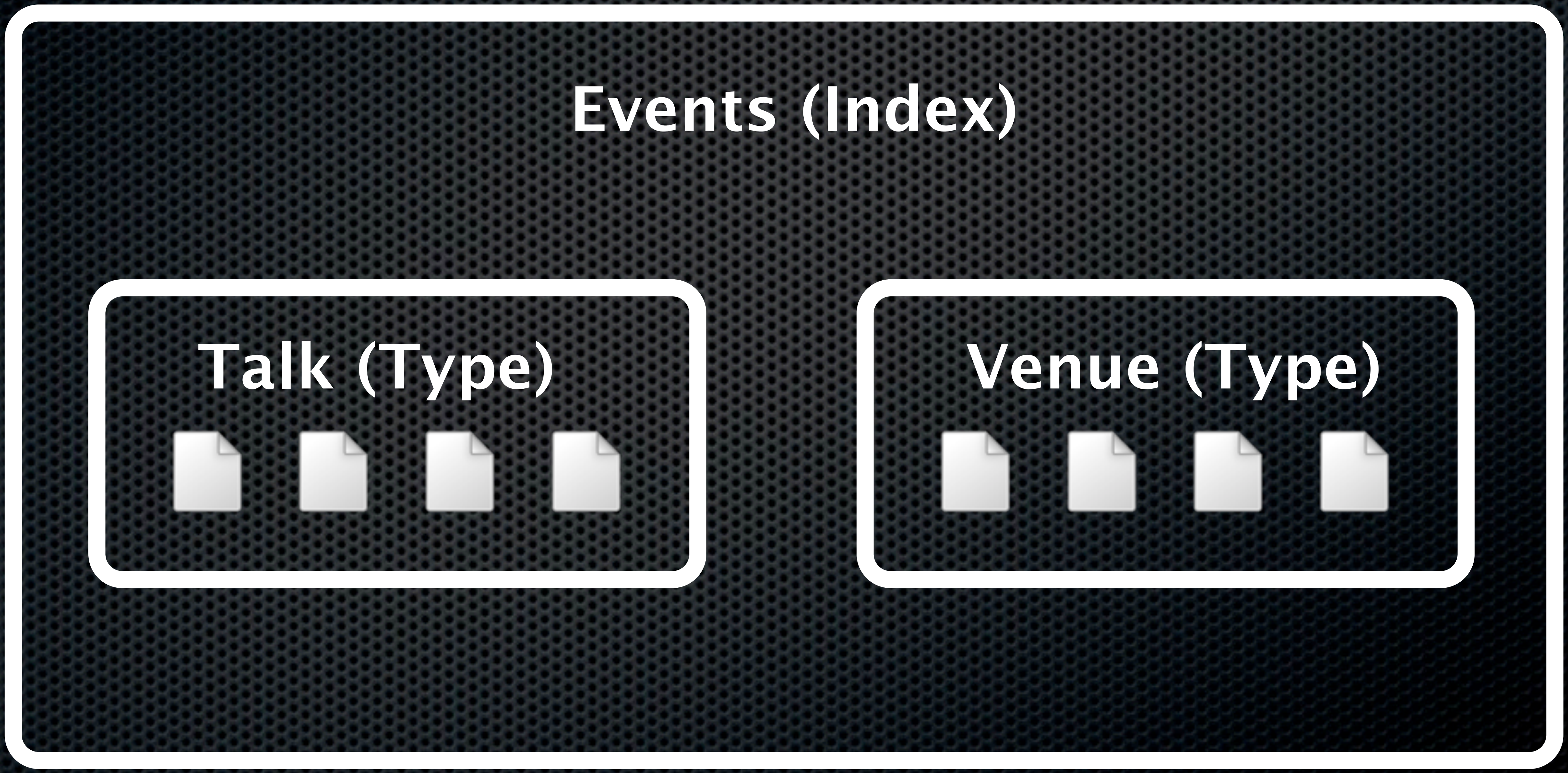
elasticsearch

- Full text search
- Big data
- Faceting
- GIS
- Clustering
- Logging and more.

Data Model

- Document store – JSON everywhere.
- Speaks HTTP (and thrift.)
- Schemaless (kinda.)
- Indexes, Types and Documents.

Data Model



Events (Index)

Talk (Type)

Venue (Type)

Getting started.

OSX

```
$ brew install elasticsearch
$ elasticsearch –f –D es.config=
   /usr/local/opt/elasticsearch/config/elasticsearch.yml
```

```
$ curl -s -XGET 'localhost:9200/'
{
  "ok" : true,
  "status" : 200,
  "name" : "Gigantus",
  "version" : {
    "number" : "0.90.2",
    "snapshot_build" : false,
    "lucene_version" : "4.3.1"
  },
  "tagline" : "You Know, for Search"
}
```

API Hierarchy

- http://host:port/[index]/[type]/[_action/id]
  - /my_index/_status
  - /my_index/_mapping
  - /my_index/my_type/_status
  - /my_index/my_type/_search
  - /my_index,my_other_index/_search
  - /_cluster/health

Indexing

```
curl -XPUT localhost:9200/events/talk/123 -d '
{"title": "ElasticSearch: Introduction."}
' | python -m json.tool

{
    "_id": "123",
    "_index": "events",
    "_type": "talk",
    "_version": 1,
    "ok": true
}
```

Fetching

```
curl -XGET localhost:9200/events/talk/123

{
    "_id": "123",
    "_index": "events",
    "_source": {
        "title": "ElasticSearch: Introduction."
    },
    "_type": "talk",
    "_version": 1,
    "exists": true
}
```

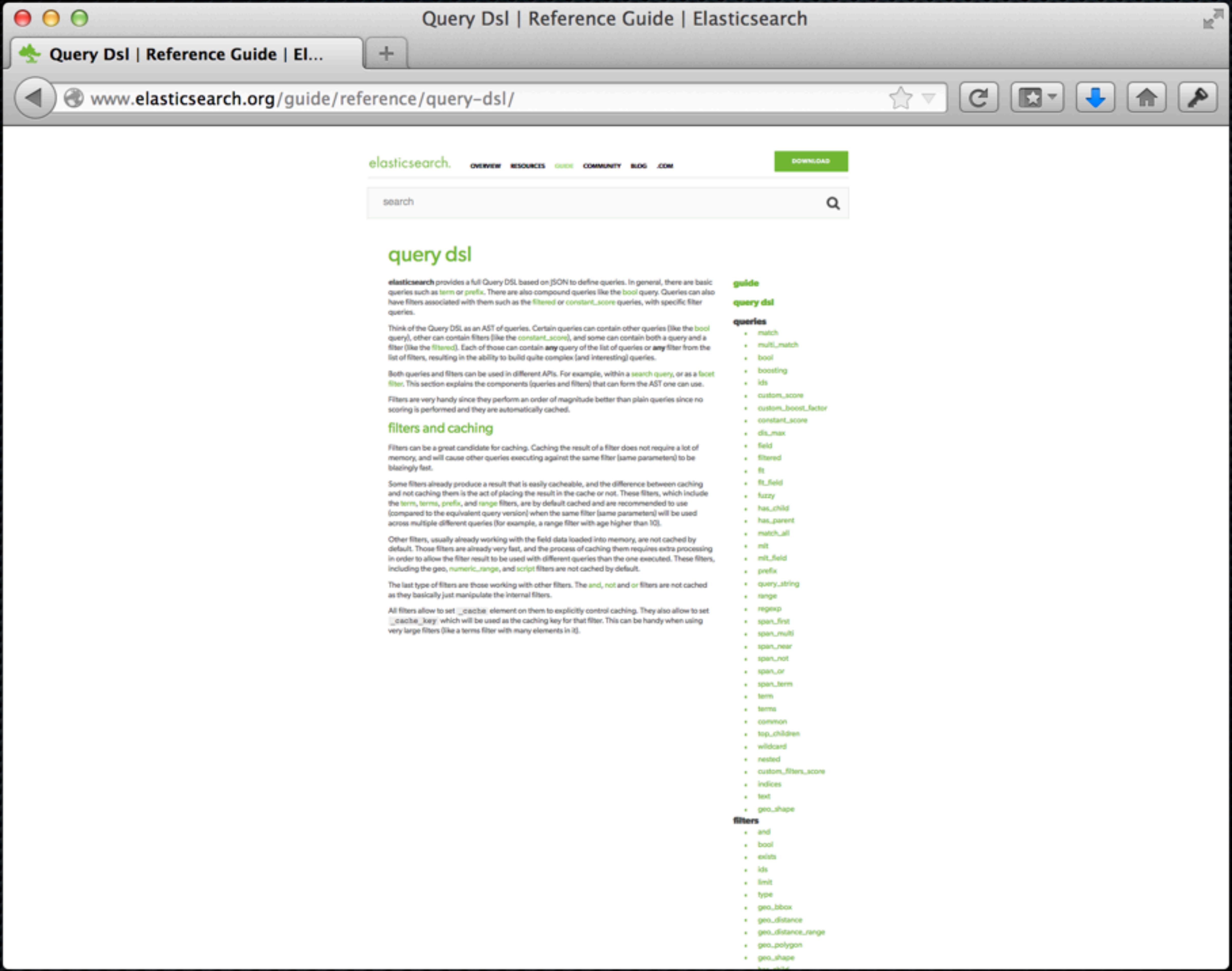
Searching

```
curl -XGET 'localhost:9200/events/_search?q=_id:123'
{
"_shards": { "failed": 0, "successful": 5, "total": 5},
"hits": {
    "hits": [
        {
            "_id": "123", "_index": "events",
            "_score": 1.0,
            "_source": {
                "title": "ElasticSearch: Introduction."
            },
            "_type": "talk"
        }
    ],
    "max_score": 1.0,
    "total": 1
},
```

Query DSL

- Filters
  - Fast
  - Cached
  - Boolean
- Queries
  - Fuzzy
  - Scored

```
{
    "bool": {
        "must": {
            "range": {
                "year": {"from": 2011, "to":2013}
            }
        },
        "must_not": {
            "term": {"language": "PHP"}
        },
        "should": [
            {
                "term": {"tag": "elasticsearch"}
            },
            {
                "term": {"tag": "python"}
            }
        ],
        "minimum_number_should_match": 1,
        "boost": 1.0
    }
}
```


Query Dsl | Reference Guide | Elasticsearch
Query Dsl | Reference Guide | El...
www.elasticsearch.org/guide/reference/query-dsl/
elasticsearch.
OVERVIEW
RESOURCES
GUIDE
COMMUNITY
BLOG
.COM
DOWNLOAD
search

query dsl

elasticsearch provides a full Query DSL based on JSON to define queries. In general, there are basic queries such as term or prefix. There are also compound queries like the bool query. Queries can also have filters associated with them such as the filtered or constant_score queries, with specific filter queries.

Think of the Query DSL as an AST of queries. Certain queries can contain other queries (like the bool query), other can contain filters (like the constant_score), and some can contain both a query and a filter (like the filtered). Each of those can contain **any** query of the list of queries or **any** filter from the list of filters, resulting in the ability to build quite complex (and interesting) queries.

Both queries and filters can be used in different APIs. For example, within a search query, or as a facet filter. This section explains the components (queries and filters) that can form the AST one can use.

Filters are very handy since they perform an order of magnitude better than plain queries since no scoring is performed and they are automatically cached.

filters and caching

Filters can be a great candidate for caching. Caching the result of a filter does not require a lot of memory, and will cause other queries executing against the same filter (same parameters) to be blazingly fast.

Some filters already produce a result that is easily cacheable, and the difference between caching and not caching them is the act of placing the result in the cache or not. These filters, which include the term, terms, prefix, and range filters, are by default cached and are recommended to use (compared to the equivalent query version) when the same filter (same parameters) will be used across multiple different queries (for example, a range filter with age higher than 10).

Other filters, usually already working with the field data loaded into memory, are not cached by default. Those filters are already very fast, and the process of caching them requires extra processing in order to allow the filter result to be used with different queries than the one executed. These filters, including the geo, numeric_range, and script filters are not cached by default.

The last type of filters are those working with other filters. The and, not and or filters are not cached as they basically just manipulate the internal filters.

All filters allow to set `_cache` element on them to explicitly control caching. They also allow to set `_cache_key` which will be used as the caching key for that filter. This can be handy when using very large filters (like a terms filter with many elements in it).

guide

query dsl

queries

- match
- multi_match
- bool
- boosting
- ids
- custom_score
- custom_boost_factor
- constant_score
- dis_max
- field
- filtered
- flt
- flt_field
- fuzzy
- has_child
- has_parent
- match_all
- mlt
- mlt_field
- prefix
- query_string
- range
- regexp
- span_first
- span_multi
- span_near
- span_not
- span_or
- span_term
- term
- terms
- common
- top_children
- wildcard
- nested
- custom_filters_score
- indices
- text
- geo_shape

filters

- and
- bool
- exists
- ids
- limit
- type
- geo_bbox
- geo_distance
- geo_distance_range
- geo_polygon
- geo_shape

Reverse Indexes

The quick
brown Fox

jumps over
the lazy dog

The brown
fox jumps

quick	1
brown	1, 3
fox	1, 3
jumps	2, 3
lazy	2
dog	2

Some Lessons!

- Indexing is really fast.
- Use with another canonical storage database.
- Bulk index around 5Mb at a time.
- Run the latest version Oracle Java.
- Define your schema.
- OOM can be a problem.
- Lots of facets = lots of memory.
- ID’s not guaranteed to be unique with routing.
- Don’t write Java plugins - hard to keep relevant.
- Avoid using “Rivers” - use the Java API instead.

Third Party Code

- Head
- Paramedic
- Segmentation Spy
- Kibana
- Loads of others...

Python Integration

- pyes - oldest, a bit hairy
- pyelasticsearch - newer, nicer, low level
- elasticutils - built on pyelasticsearch, feels ORM’y
- django-haystack - Very easy integration with Django

Questions?

Follow me on Twitter: d0ugal

artirix.com
dougalmatthews.com
speakerdeck.com/d0ugal