Discovering python search engine

—

José Manuel Ortega - Pycones 2017

Agenda

- **Introduction to search engines**
- **ElasticSearch,whoosh,django-hystack**
- **ElasticSearch example**
- **Other solutions & tools**
- **Conclusions**

Search engines

Search engines

- Document based
- A document is the unit of searching in a full text search system.
- A document can be a json or python dictionary

Core concepts

Core concepts

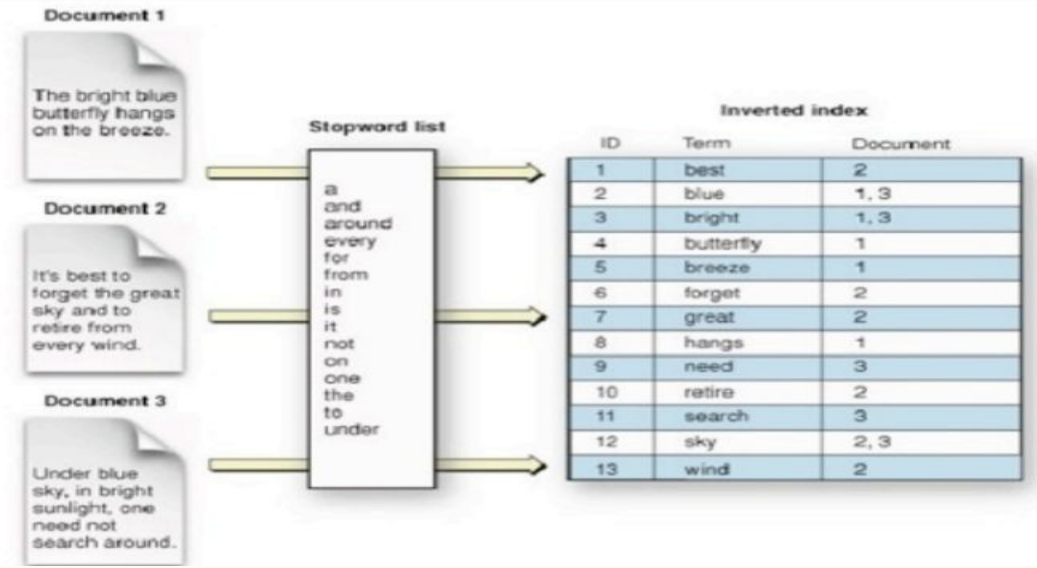
- **Index:** Named collection of documents that have similar characteristics(like a **database**)
- **Type:**Logical partition of an index that contains documents with common fields(like a **table**)
- **Document**:basic unit of information(like a **row**)
- **Mapping**:field properties(datatype,token extraction). Includes information about how fields are stored in the index

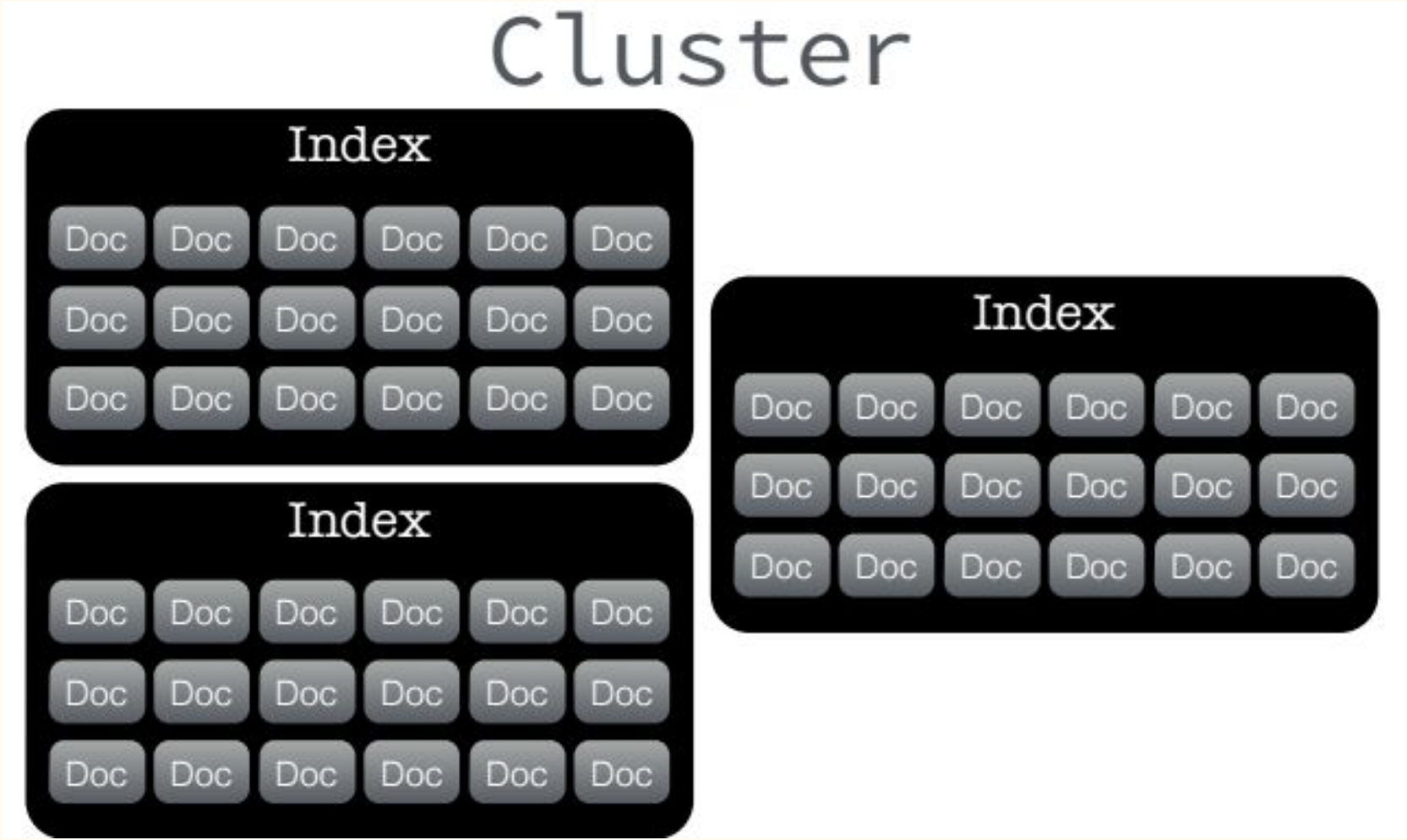
Core concepts

- **Relevance** are the algorithms used to rank the results based on the query
- **Corpus** is the collection of all documents in the index
- **Segments**:Sharded data storing the inverted index.Allow **searching in the index in a efficient way**

Inverted index

- Is the heart of the search engine
- Each inverted index stores position and document IDs

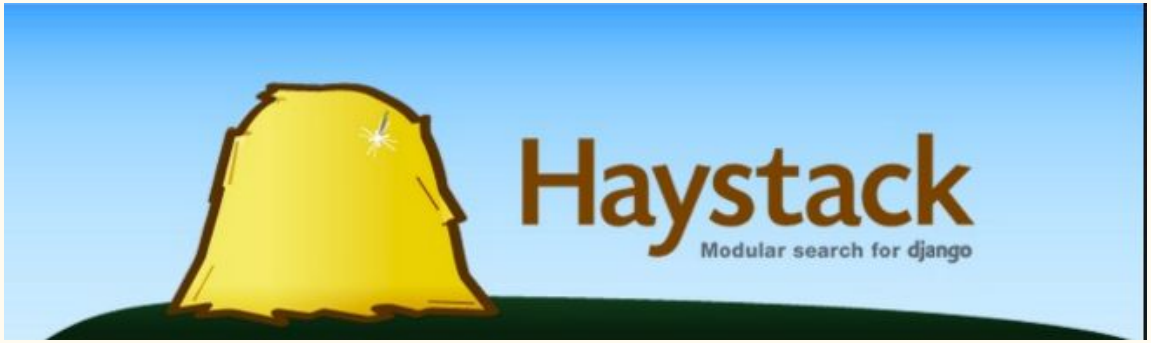



Cluster
Index
Doc Doc Doc Doc Doc Doc
Doc Doc Doc Doc Doc Doc
Doc Doc Doc Doc Doc Doc
Index
Doc Doc Doc Doc Doc Doc
Doc Doc Doc Doc Doc Doc
Doc Doc Doc Doc Doc Doc
Index
Doc Doc Doc Doc Doc Doc
Doc Doc Doc Doc Doc Doc
Doc Doc Doc Doc Doc Doc

elastic

WHOOSH

Python search library



lucene

Apache Solr

ElasticSearch

Elasticsearch

Search & Analyze Data in Real Time

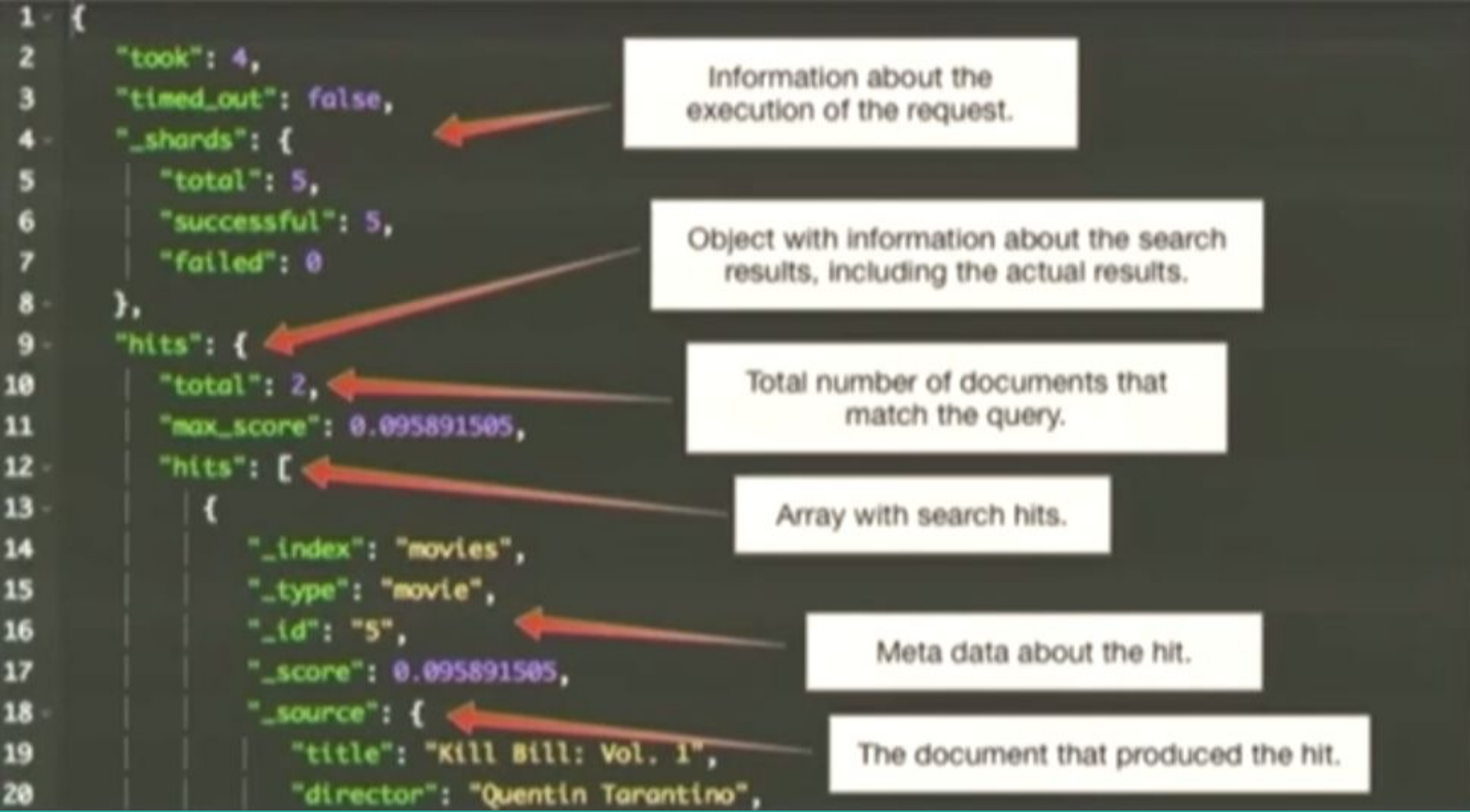

GitHub


KLOUT


stackoverflow

- Open source search server based on Apache Lucene
- Written in Java
- Cross-platform
- Communications with the search server is done through HTTP REST API
- curl -X<GET|POST|PUT|DELETE> http://localhost:9200/<index>/<type_document>/id

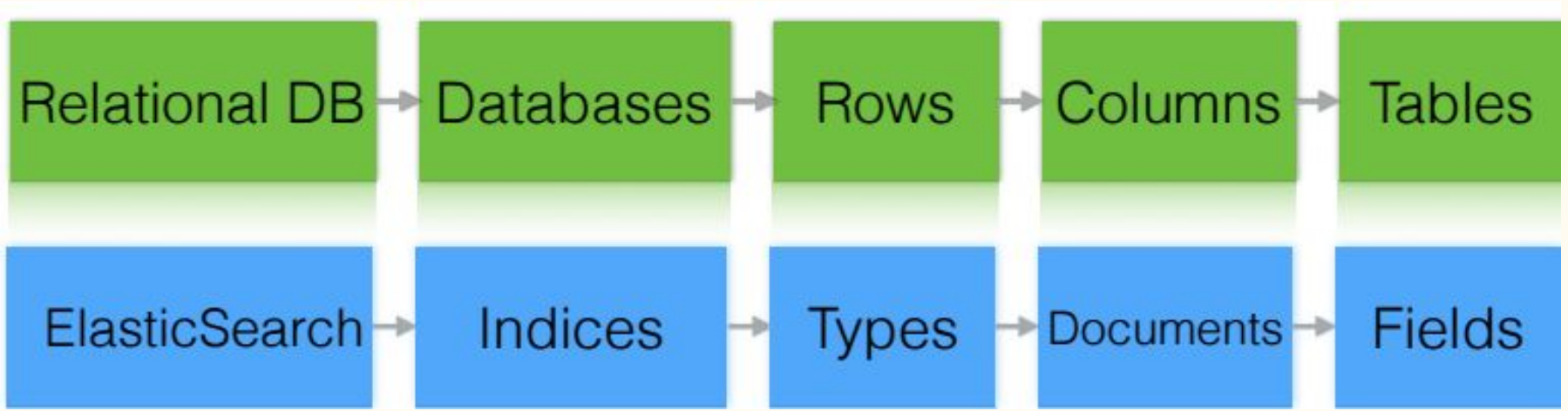
- You can add a document without creating an index
- ElasticSearch will create the index,mapping type and fields automatically
- ElasticSearch will infer the data types based on the document’s data

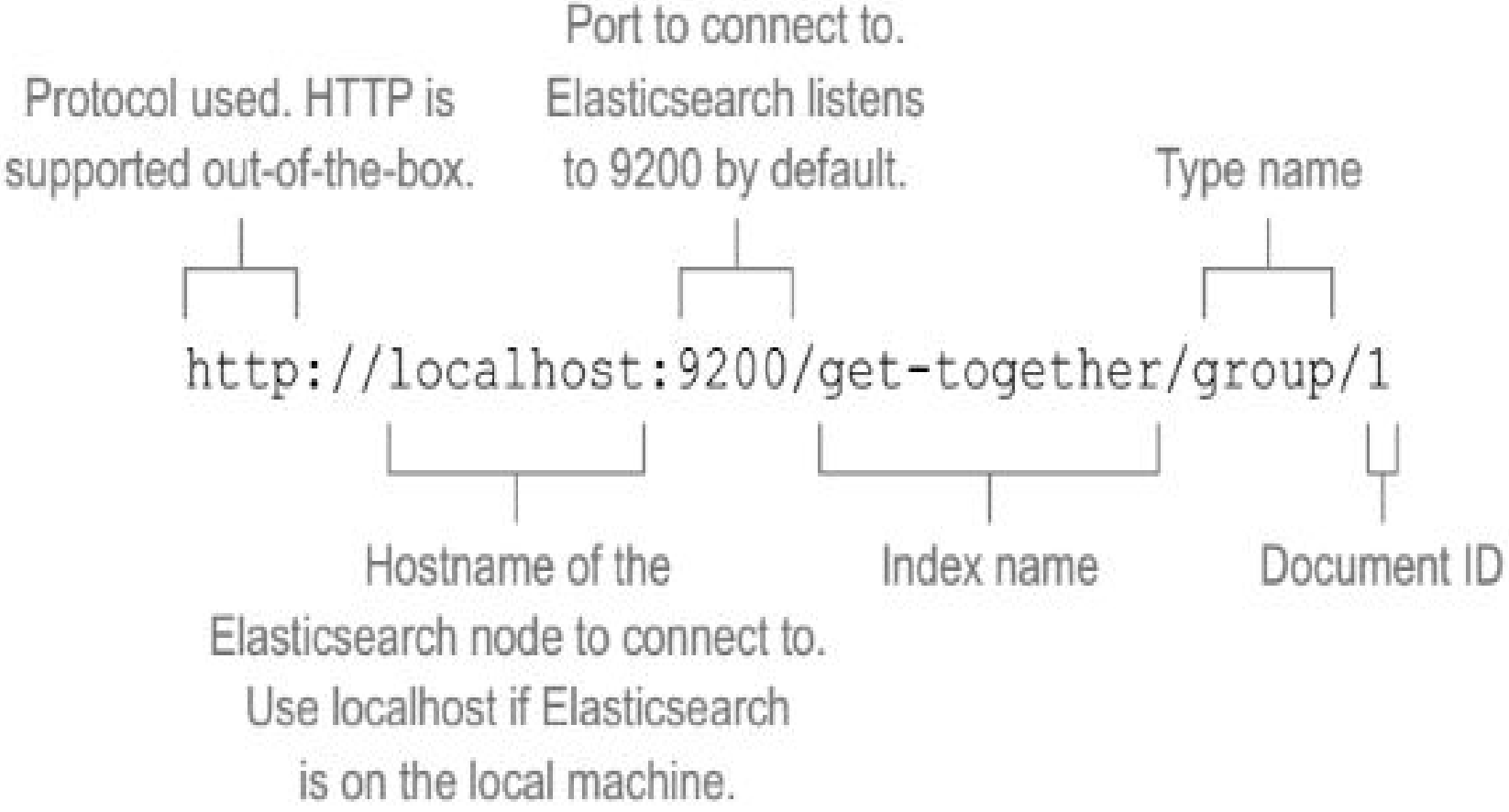

{
"took": 4,
"timed_out": false,
"_shards": {
"total": 5,
"successful": 5,
"failed": 0
},
"hits": {
"total": 2,
"max_score": 0.095891505,
"hits": [
{
"_index": "movies",
"_type": "movie",
"_id": "5",
"_score": 0.095891505,
"_source": {
"title": "Kill Bill: Vol. 1",
"director": "Quentin Tarantino",
Information about the execution of the request.
Object with information about the search results, including the actual results.
Total number of documents that match the query.
Array with search hits.
Meta data about the hit.
The document that produced the hit.

Metadata Fields

- Each document has metadata associated with it
- **_index:**Allows matching documents based on their indexes.
- **_type:**Type of the document
- **_id:**Document id(not indexed)
- **_uid**:_type + _id(indexed)
- **_source:**contains the json passed in creation time of the index or document(not indexed)
- **_version**

ElasticSearch vs Relational DB




Protocol used. HTTP is supported out-of-the-box.
Port to connect to. Elasticsearch listens to 9200 by default.
Type name
http://localhost:9200/get-together/group/1
Hostname of the Elasticsearch node to connect to. Use localhost if Elasticsearch is on the local machine.
Index name
Document ID

Creating an Index

```
curl -XPUT ‘localhost:9200/myindex’-d {
    “settings”:{..}
    “mappings”:{..}
}
```

```
curl -XPUT http://localhost:9200/blog -d '{
    "settings" : {
        "index" : {
            "number_of_shards" : 1,
            "number_of_replicas" : 0
        }
    },
    "mapping": {
        "posts": {
            "properties": {
                "title": "string",
                "published": "date"
            }
        }
    }
}'
```

```
curl -XPOST http://localhost:9200/blog/posts -d '{
    "title": "Discover python search engine",
    "published": "2017-09-22"
}'


curl -XPOST http://localhost:9200/blog/posts/_search?
pretty= true -d '{
    "query": {
        "match": {
            "title": "python"
        }
    }
}'
```

```
import json
import requests

url = 'http://localhost:9200/pycones/talk/1'

document = {
  'title': "Discovering python search engine",
  'description' : "Full text search"
  }

requests.post(url, data=json.dumps(document))
```

Searching a document

```
curl -XGET 'localhost:9200/myindex/mydocument/_search?q=elasticSearch'

curl -XGET 'localhost:9200/myindex/mydocument/_search?pretty' -d{
    "query":{
        "match":{
            "_all":"elasticSearch"
        }
    }
}
```

Query DSL

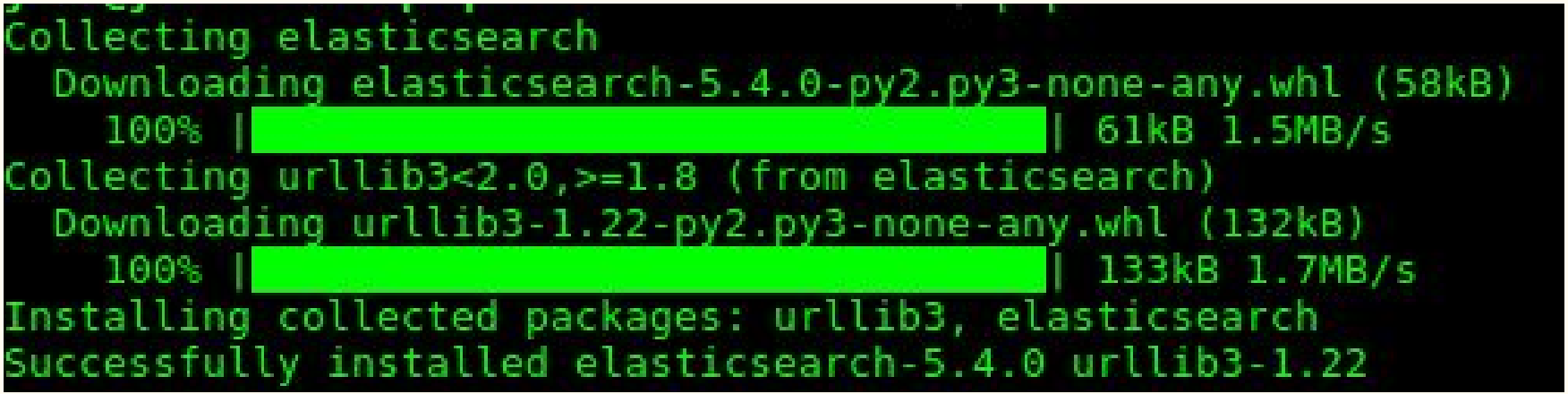
```
import requests
import json

def search(uri, term):
    """Simple Elasticsearch Query"""
    query = json.dumps({
        "query": {
            "match": {
                "title": term
            }
        }
    })
    response = requests.get(uri, data=query)
    results = json.loads(response.text)
    return results
```

Searching a document

- **Search can get much more complex**
  - **Multiple terms**
  - **Multi-match(math query on specific fields)**
  - **Bool(true,false)**
  - **Range**
  - **RegExp**
  - **GeoPoint,GeoShapes**

ElasticSearch python client

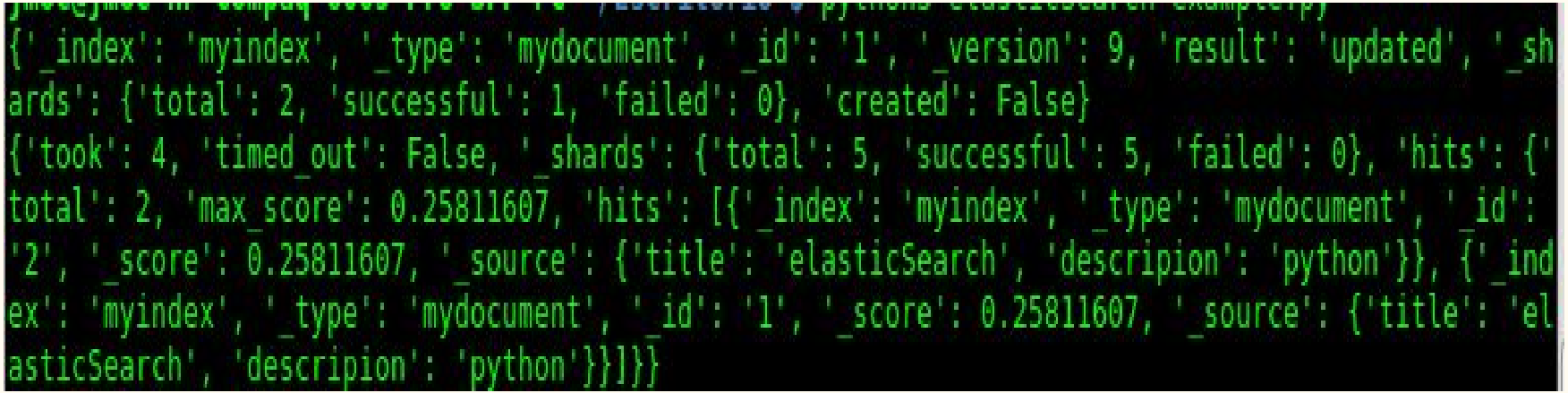
- The official low-level client is elasticsearch-py
  - pip install elasticsearch



ElasticSearch-py API

```python
from elasticsearch import Elasticsearch

es= Elasticsearch()

document = {"title":"elasticSearch","descripion":"python"}

##create inded if doesnt already exist
response = es.index(index="myindex",doc_type="mydocument",id=1,body=document)
print(response)

##search for "python" in all fields in all documents
query = {'query':{'match':{'_all':'python'}}}
response_search = es.search(index="myindex",doc_type="mydocument",body=query)
print(response_search)
```

ElasticSearch-py API



```
import csv
from elasticsearch import ElasticSearch

#by default connect with localhost:9200
es = ElasticSearch()

#create an index in es,ignore status code 400
es.indices.create(index='myindex',ignore=400)

with open('data.csv') as file:
  reader = csv.DictReader(file)
  for line in reader:
    es.index(index='myindex',doc_type,body=line)
```

```
def check_index_existence(index_name):
    '''
    Function for checking the existence of an index
    :param index_name: Name of the index to be created
    :param doc_type: Name of document type to be created
    :return: False if index does not exist else True
    '''
    if not es.indices.exists(index=index_name):
        return False
    else:
        return True
```

```
def index_document(index_name, doc_type, doc_id):
    '''
    Function for index a document
    :param index_name: Name of the index
    :param doc_type: Name of document type
    :param doc_id: Unique id to be set for the document
    :return:
    '''
    doc1 = {
            'name': 'Elasticsearch Essentials',
            'category': ['Big Data', 'search engines',
              'Analytics']
            }
    es.index(index=index_name, doc_type=doc_type, body=doc1,
      id=doc_id)
```

```
def get_document(index_name, doc_type, doc_id):
    '''
    Function for fetching a single document
    :param index_name: Name of the index
    :param doc_type: Name of document type
    :param doc_id: Document id to be retrieved
    :return:
    '''
    response = es.get(index=index_name, doc_type=doc_type,
      id=doc_id, ignore=404)
    print response
    #Access fields from the response
    print response.get('_source').get('category')
```

```
def check_document_existence(index_name, doc_type, doc_id):
    '''
    Function for checking if a document exists ot not in the
      given index
    :param index_name: Name of the index
    :param doc_type: Name of document type
    :param doc_id: Document id whose existence need to be
      checked
    :return:
    '''
    if not es.exists(index=index_name, doc_type=doc_type, id
      =doc_id):
        return False
    else:
        return True
```

```
def search_documents(index_name, doc_type, query=None):
    '''
    Function for performing search using Query-DSL queries
    :param index_name: Name of the index
    :param doc_type: Name of document type
    :param query: Query in json format
    '''
    if query is None:
        query = {
         "query":{
           "match":{"text":"Elasticsearch"}
         }
        }

    response = es.search(index=index_name, doc_type=doc_type
      , body=query, size=2, request_timeout=20)
    for hit in response['hits']['hits']:
        print json.dumps(hit.get('_source'))
```

Geo queries

- Elastic search supports two types of geo fields
  - **geo_point**(lat,lon)
  - **geo_shapes**(points,lines,polygons)
- **Perform geographical searches**
  - Finding points of interest and GPS coordinates

```
def create_index_with_geopoint_mapping(index_name,doc_type):
    """
    Function to create index with nested mapping
    :param index_name: Name of the index to be created
    :param doc_type: Name of document type to be created
    """
    doc_mapping = {
      "properties": {
       "location": {
            "type": "geo_point"
            }
        }
    }
    body = dict()
    mapping = dict()
    mapping[doc_type] = doc_mapping
    body['mappings'] = mapping
    es.indices.create(index=index_name, body = body)
```

```
def find_by_distance(index_name, doc_type):
    query = {
      "query": {
          "filter": {
            "geo_distance": {
              "distance": "12km",
              "location": {
                "lat": 28.67,
                "lon": 77.42
              }
            }
          }
      }
    }
    response = es.search(index=index_name, doc_type=doc_type
      , body=query)
```

 **jmortega** python discover python search engine

..

backend	python discover python search engine
data	python discover python search engine
frontend	python discover python search engine
Makefile	python discover python search engine
README.rst	python discover python search engine
mapping.json	python discover python search engine
requirements.txt	python discover python search engine
runbackend.py	python discover python search engine

List of Talks

Title	Author	
Modelo de Redes Neuronales Recurrentes para el Análisis y Predicción de Series Temporales	J.C. González-Avella y J. M. Tuduri	Delete
Discovering python search engine	Jose Manuel Ortega	Delete
Usando Python y SocketIO para aplicaciones en tiempo real	Oscar Ramírez	Delete
High-impact refactors while keeping the lights on	Diego Muñoz	Delete
Mi coche ya es mayor de edad, creo que puede conducir él solito.	Ricardo Guerrero Gómez-Olmedo	Delete
City attractiveness seen through Twitter	Antònia Tugores	Delete
Factories, what the hell?	Miguel Jiménez	Delete
Machine Learning en producción	Luis Mesas	Delete

Talks Home

machine learning

Title	Author
Machine Learning en producción	Luis Mesas

Insert New Talk Full Talk List

Machine Learning en producción

Author:	Luis Mesas
Type:	Talk
Description:	Una vez tenemos el modelo diseñado por los data scientist hay que ponerlo en pro, pero ninguna de las herramientas que han usado son viables en producción y tienen problemas de escalabilidad ¿Que hacemos ahora?. En esta charla veremos arquitecturas aplicadas a 2 casos de uso típicos: arquitectura de ejecución única reutilizable y arquitectura de uso contínuo con bucle de feedback.

Go to Search | See All Talks | Insert New Talk

Insert New Talk

Talk title:

Machine learning y datascience con python con sklearn y pyspark

Author:

author

Type:

☑ **Talk** ☑ **Workshop**

Talk Description:

description

Save Cancel

Create basic database (use data from data/pycones2017.data):

```
make index
```

Run backend service:

```
make backend
```

Run frontend service:

```
make frontend
```

Point your browser to:

```
http://localhost:8000
```

and search for a talk.

```
def get(self):
    print("Call for: GET /talks")
    url = config.es_base_url['talks']+'/_search'
    query = {
        "query": {
            "match_all": {}
        },
        "size": 100
    }
    resp = requests.post(url, data=json.dumps(query))
    data = resp.json()
    talks = []
    for hit in data['hits']['hits']:
        talk = hit['_source']
        talk['id'] = hit['_id']
        talks.append(talk)
    return talks
```

Whoosh

- Pure-python full-text indexing and searching library
- Library of classes and functions for indexing text and then searching the index.
- It allows you to develop custom search engines for your content.
- Mainly focused on index and search definition using schemas
- Python 2.5 and Python 3

Schema

```
class whooshSCHEMA(fields.SchemaClass):
  title = fields.TEXT(stored=True,sortable=True)
  content =  fields.TEXT(spelling=True)
  date = fields.DATETIME(stored=True)
  summary = fields.STORED
  url=fields.ID(stored=True, unique=True))

WHOOSH_SCHEMA = fields.Schema(
  title=fields.TEXT(stored=True,sortable=True),
  content=fields.TEXT(spelling=True),
  date = fields.DATETIME(stored=True),
  summary = fields.STORED,
  url=fields.ID(stored=True, unique=True))
```

Create index and insert document

```
#To create an index basically you need a writer object
ix = index.create_in("index",schema=WHOOSH_SCHEMA)
writer = ix.writer()

writer.add_document(title="pycones 2017",content="python conference",
   date = datetime(2017,9,22),
   summary = "discovering python search engine",
   url="http://pycones.es")

writer.add_document(title="python 2017",content="pycones2017",
   date = datetime(2017,9,22),
   summary = "discovering python search engine",
   url="http://pycones.es")

writer.commit()
```

Searching single field

```python
#searching in the index by a single field
from whoosh import qparser

queryParser = qparser.QueryParser("content",schema = ix.schema)
query = queryParser.parse("python")
with ix.searcher() as searcher:
  results = searcher.search(query)
  print(results)
  for result in results:
    print(result)
```

```
<Top 1 Results for Term('content', 'python') runtime=0.00055894400065881 2>
<Hit {'date': datetime.datetime(2017, 9, 22, 0, 0), 'summary': 'discovering py
thon search engine', 'title': 'pycones 2017', 'url': 'http://pycones.es'}>
```

Searching multiple field

```
#searching in the index by a multiple field
from whoosh.qparser import MultifieldParser, OrGroup

queryParser = MultifieldParser(["title",
                                "content"],
                               schema=ix.schema,
                               group=OrGroup)
query = queryParser.parse("python")
with ix.searcher() as searcher:
  results = searcher.search(query)
  print(results)
  for result in results:
    print(result)
```

```
<Top 2 Results for Or([Term('title', 'python'), Term('content', 'python')]) ru
ntime=0.0008421520014962888>
<Hit {'date': datetime.datetime(2017, 9, 22, 0, 0), 'summary': 'discovering py
thon search engine', 'title': 'python 2017', 'url': 'http://pycones.es'}>
<Hit {'date': datetime.datetime(2017, 9, 22, 0, 0), 'summary': 'discovering py
thon search engine', 'title': 'pycones 2017', 'url': 'http://pycones.es'}>
```

Django-haystack

Haystack

Modular search for django

- More Like This
- Faceting
- Spatial Search
- Stored (non-indexed) fields
- Highlighting
- Spelling Suggestions
- Boost

Find the needle you're looking for. Download | Documentation

Search doesn't have to be hard. Haystack lets you write your search code once and choose the search engine you want it to run on. With a familiar API that should make any Djangonaut feel right at home and an architecture that allows you to swap things in and out as you need to, it's how search ought to be.

Haystack is BSD licensed, plays nicely with third-party apps without needing to modify the source and supports Solr, Elasticsearch, Whoosh and Xapian.

Haystack 2.6.0!

Posted on 2017/01/04

Changelog is available here: http://django-haystack.readthedocs.io/en/v2.6.0/changelog.html.

Haystack 2.0.0!

Posted on 2013/05/12 by Daniel

After two (far too long) years, Haystack v2.0.0 has been

- Multiple backends (you have a Solr & a Whoosh index, or a master Solr & a slave Solr, etc.)
- An Elasticsearch backend
- Big query improvements
- Geospatial search (Solr & Elasticsearch only)
- The addition of Signal Processors for better control
- Input types for improved control over queries
- Rich Content Extraction in Solr

Branch: **master** ▾ **django-haystack** / haystack / **backends** /

Create new file | **Find file** | **History**

 **acdha** Whoosh: prevent more_like_this from hitting an uninitialized variable ... Latest commit 7d42273 7 days ago

..		
__init__.py	PEP-8	4 months ago
elasticsearch2_backend.py	Refactoring the code in pull request #1336 . This pull request is to …	9 months ago
elasticsearch_backend.py	fix: contains filter, add endswith filter	a year ago
simple_backend.py	Avoid unsafe default value on backend clear() methods	2 years ago
solr_backend.py	Merge branch '1504-solr-6-by-default'	4 months ago
whoosh_backend.py	Whoosh: prevent more_like_this from hitting an uninitialized variable	7 days ago

```
# Add Haystack to INSTALLED_APPS
INSTALLED_APPS = settings.INSTALLED_APPS + (
    'haystack',
)
HAYSTACK_CONNECTIONS = {
    'default': {
        'ENGINE': 'haystack.backends.solr_backend
          .SolrEngine',
        'URL': 'http://localhost:9001/solr/example'
    },
    'elasticsearch': {
        'ENGINE': 'haystack.backends.elasticsearch_backend
          .ElasticsearchSearchEngine',
        'URL': 'http://localhost:9200',
        'INDEX_NAME': 'example_project'
    },
    'whoosh': {
        'ENGINE': 'haystack.backends.whoosh_backend
```

```
import datetime
from haystack import indexes
from haystack import site
from blog.models import BlogEntry

class BlogEntryIndex(indexes.SearchIndex,
indexes.Indexable):

    text = indexes.CharField(document=True, use_template
      =True)
    author = indexes.CharField(model_attr='author')
    pub_date = indexes.DateField(model_attr='pub_date')

  def get_queryset(self):
    return BlogEntry.objects.filter(pub_date= datetime
      .datetime.now(), status=BlogEntry.LIVE_STATUS)

site.register(BlogEntry, BlogEntryIndex)
```

```
import datetime
from haystack import indexes
from haystack import site
from blog.models import BlogEntry

class BlogEntryIndex(RealTimeSearchIndex):

    text = indexes.CharField(document=True, use_template
      =True)
    author = indexes.CharField(model_attr='author')
    pub_date = indexes.DateField(model_attr='pub_date')

  def get_queryset(self):
    return BlogEntry.objects.filter(pub_date= datetime
      .datetime.now(), status=BlogEntry.LIVE_STATUS)

site.register(BlogEntry, BlogEntryIndex)
```

```
#Create search_sites.py inside my Django site, at the same
  level as settings.py

#This causes haystack to look for search_indexes.py inside
  each app in the Django site.

import haystack
haystack.autodiscover()
```

```
from haystack.query import SearchQuerySet

def search_entries(request):
    entries = SearchQuerySet.autocomplete
    (content_auto=request.POST.get('search_text',''))

    return render_to_response('ajax_search.html',{'entries'.entries})
```

- **Create the index**
  - Run **./manage.py rebuild_index** to create the new search index.
- **Update the index**
  - **./manage.py update_index** will add new entries to the index.
  - **./manage.py rebuild_index** will recreate the index from scratch.

- **Pros:**
  - **Easy to setup**
  - **Looks like Django ORM but for searches**
  - **Search engine independent**
  - **Support 4 engines (Elastic, Solr, Xapian, Whoosh)**
- **Cons:**
  - **Poor SearchQuerySet API**
  - **Difficult to manage stop words**
  - **Loose performance, because extra layer**
  - **Django Model based**

Other solutions

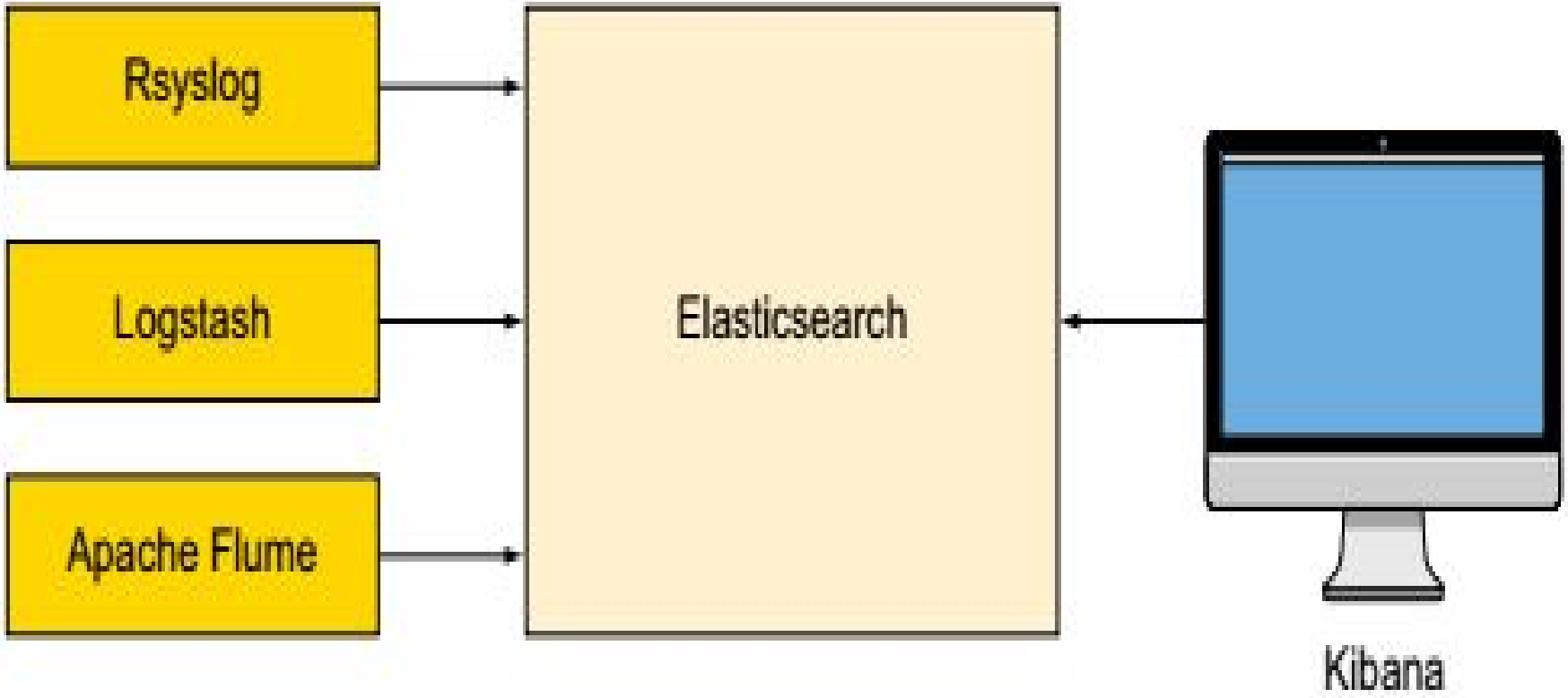
Other solutions

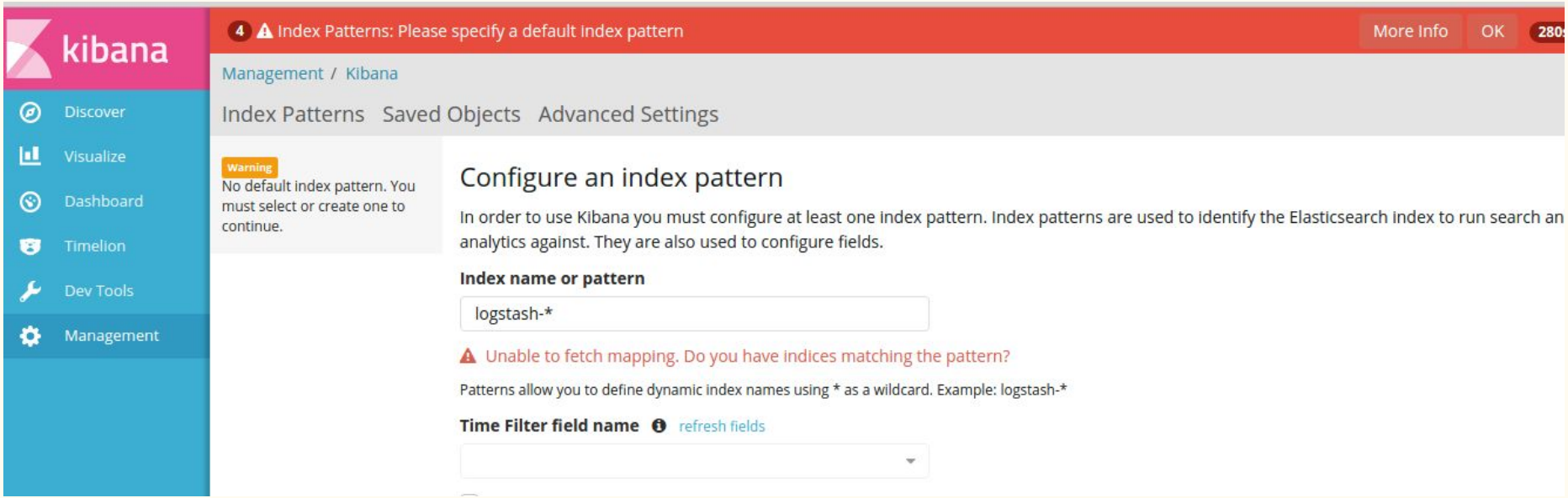
- https://xapian.org
- https://docs.djangoproject.com/en/1.11/ref/contrib/postgres/search/
- https://www.postgresql.org/docs/9.6/static/textsearch.html

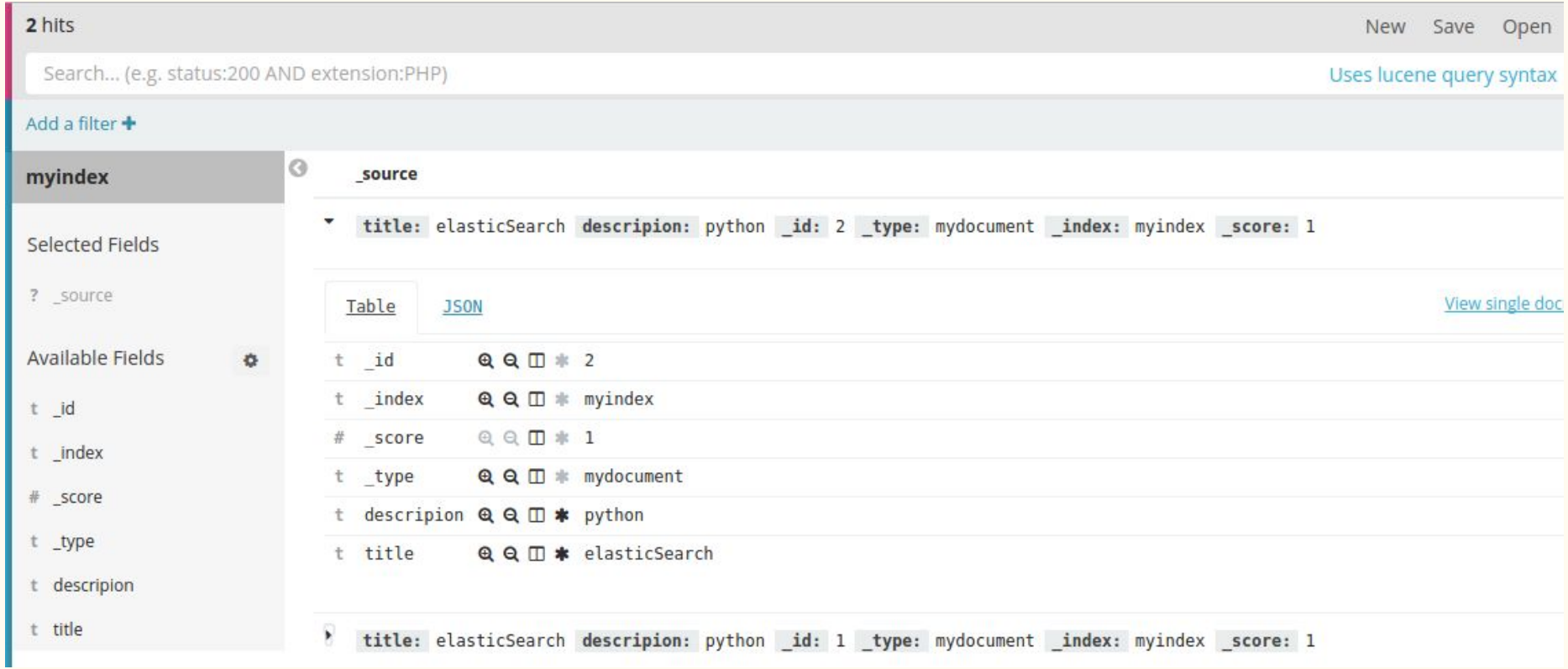
pysolr

```
import pysolr

# Setup a Solr instance. The timeout is optional.
solr = pysolr.Solr('http://localhost:8983/solr/', timeout=10)

# How you'd index data.
solr.add([
    {
        "id": "doc_1",
        "title": "A test document",
        "_doc": [
            { "id": "doc_1_1", "title": "document_1_1" }
        ]
    },
])

results = solr.search('document')
```

Other tools


Rsyslog
Logstash
Apache Flume
Elasticsearch
Kibana

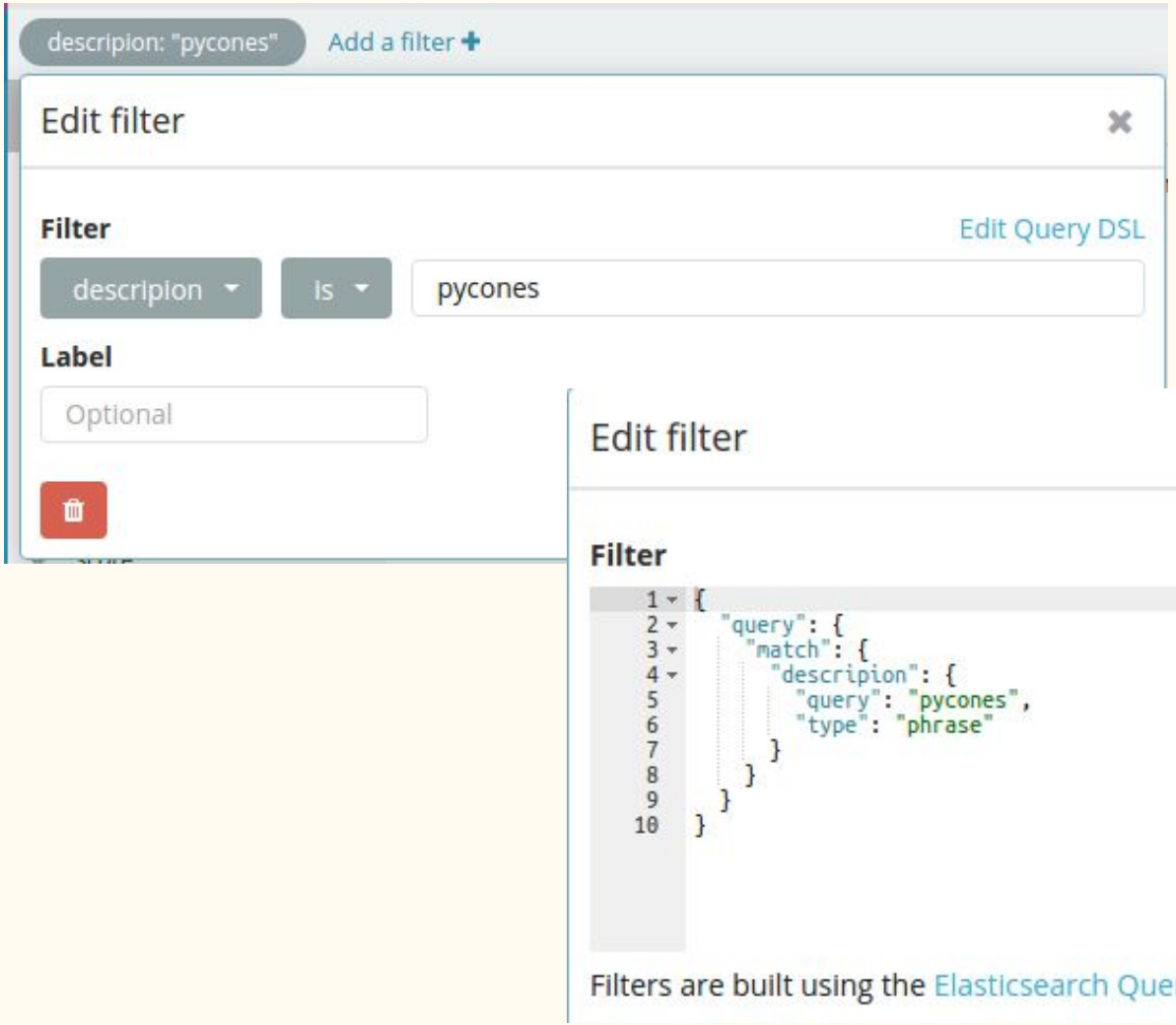

kibana
Discover
Visualize
Dashboard
Timelion
Dev Tools
Management
4 Index Patterns: Please specify a default index pattern
More Info
OK
Management / Kibana
Index Patterns
Saved Objects
Advanced Settings
Warning
No default index pattern. You must select or create one to continue.
Configure an index pattern
In order to use Kibana you must configure at least one index pattern. Index patterns are used to identify the Elasticsearch index to run search an analytics against. They are also used to configure fields.
Index name or pattern
logstash-*
Unable to fetch mapping. Do you have indices matching the pattern?
Patterns allow you to define dynamic index names using * as a wildcard. Example: logstash-*
Time Filter field name
refresh fields


2 hits
New
Save
Open
Search... (e.g. status:200 AND extension:PHP)
Uses lucene query syntax
Add a filter +
myindex
Selected Fields
? _source
Available Fields
t _id
t _index
_score
t _type
t descripion
t title
_source
title: elasticSearch descripion: python _id: 2 _type: mydocument _index: myindex _score: 1
Table
JSON
View single doc
t _id 2
t _index myindex
_score 1
t _type mydocument
t descripion python
t title elasticSearch
title: elasticSearch descripion: python _id: 1 _type: mydocument _index: myindex _score: 1

Console

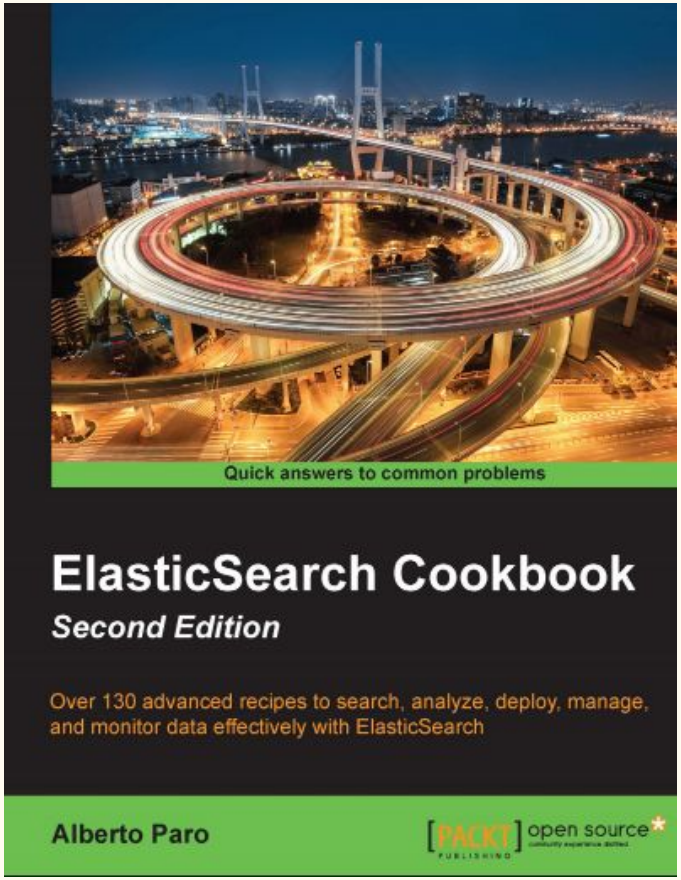
```
GET _search
{
  "query": {
    "multi_match":
        {
          "query": "pycones",
          "fields":["title","descripion"]
        }

  }
}
```

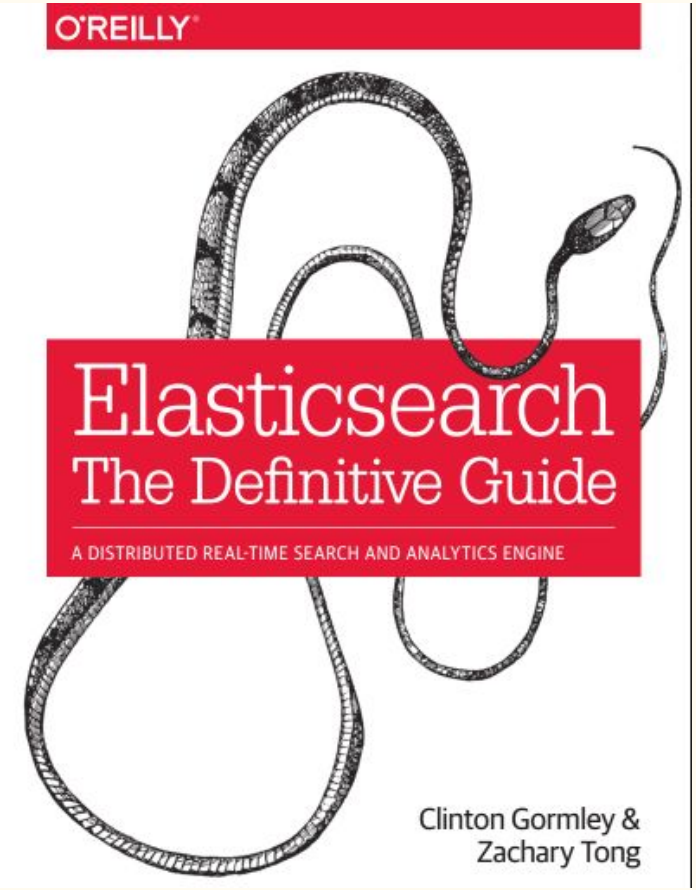
```
{
  "took": 10,
  "timed_out": false,
  "_shards": {
    "total": 6,
    "successful": 6,
    "failed": 0
  },
  "hits": {
    "total": 2,
    "max_score": 0.25811607,
    "hits": [
      {
        "_index": "myindex",
        "_type": "mydocument",
        "_id": "2",
        "_score": 0.25811607,
        "_source": {
          "title": "pycones 2017",
          "descripion": "discoverting python search engine"
        }
      },
      {
        "_index": "myindex",
        "_type": "mydocument",
        "_id": "1",
        "_score": 0.25811607,
        "_source": {
          "title": "python conference",
          "descripion": "pycones 2017"
        }
      }
    ]
  }
}
```

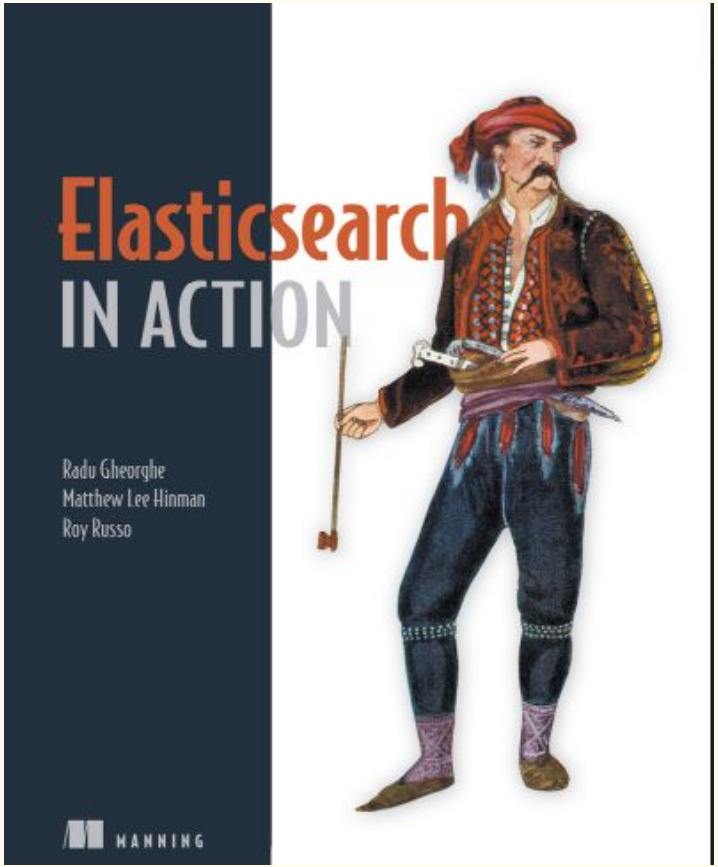

descripion: "pycones"
Add a filter +
Edit filter
Filter
Edit Query DSL
descripion
is
pycones
Label
Optional
Edit filter
Filter
Search filter values
{
"query": {
"match": {
"descripion": {
"query": "pycones",
"type": "phrase"
}
}
}
}
Filters are built using the Elasticsearch Query DSL.

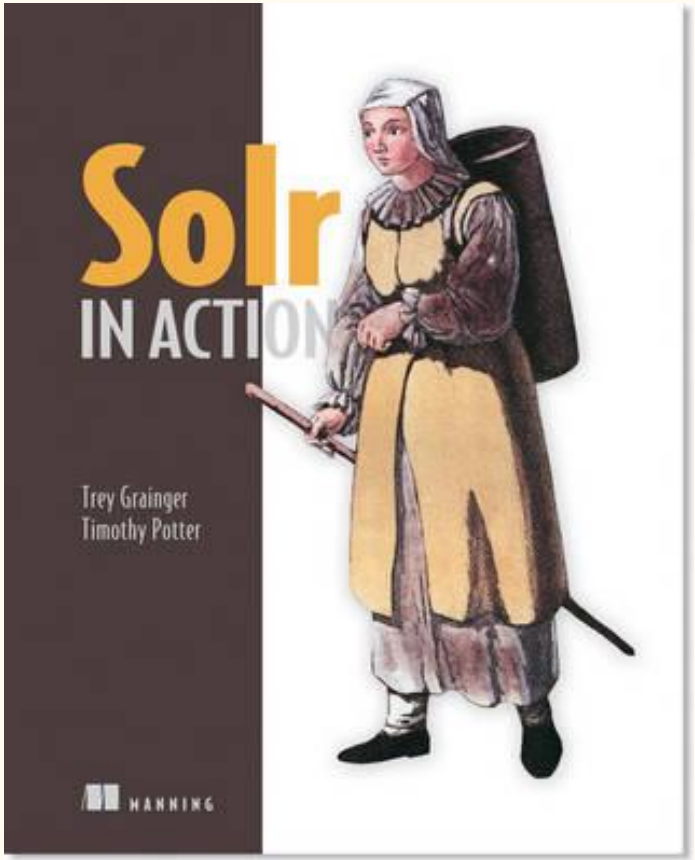
References

- http://elasticsearch-py.readthedocs.io/en/master/
- https://whoosh.readthedocs.io/en/latest
- http://django-haystack.readthedocs.io/en/master/
- http://solr-vs-elasticsearch.com/
- https://wiki.apache.org/solr/SolPython
- https://github.com/django-haystack/pysolr


Quick answers to common problems
ElasticSearch Cookbook
Second Edition
Over 130 advanced recipes to search, analyze, deploy, manage, and monitor data effectively with ElasticSearch
Alberto Paro
[PACKT] open source
PUBLISHING


O'REILLY
Elasticsearch
The Definitive Guide
A DISTRIBUTED REAL-TIME SEARCH AND ANALYTICS ENGINE
Clinton Gormley &
Zachary Tong


Elasticsearch
IN ACTION
Radu Gheorghe
Matthew Lee Hinman
Roy Russo
MANNING


Solr
IN ACTION
Trey Grainger
Timothy Potter
MANNING

Thanks!

jmortega.github.io

@jmortegac