Building geo search applications with Elasticsearch

Mark Varley

mark@addresscloud.com

By the end of this workshop you should…

- Understand the concepts of NoSQL databases and in what circumstances they are useful
- Understand what problems Elasticsearch aims to solve
- Understand and name the other applications in the Elastic family and describe their use
- Be able to install and run Elasticsearch
- Know how to install Elasticsearch plugins
- Be able to write documents to Elasticsearch
- Understand how to load data from existing data sources
- Understand how indexing works in Elasticsearch at a high level

By the end of this workshop you should also…

- Understand how to query Elasticsearch and the differences between a query and a filter
- Understand how to influence match rates by making changes in the index
- Understand what geo capabilities Elasticsearch has
- Know how to calculate and sort results by distance from a point
- Be able to query by polygon and bounding box
- Understand how to optimise the indexing to speed up geospatial queries
- Understand the limitations of Elasticsearch and when to use a different technology
- **Any requests?**

What this is not…

- A full Elasticsearch training course – this is a user's perspective, I do not work for Elastic!
- A complete overview of the Elasticsearch platform – I use Elasticsearch every day however I do not use every feature, it is huge!
- A guide to running Elasticsearch in Production (though I will give tips where possible)
- A guide to building beautiful mapping applications, I am not a web / app designer!
- A lecture! Questions are welcome throughout

Logistics: tools we will need

- Command terminal
- Java
- Elasticsearch!
- Elasticsearch JDBC
- curl
- Sense (Elasticsearch Chrome plugin)
- An internet connection!
- Sample scripts and data https://github.com/envision-it/elasticsearch-geo or USB stick

Agenda

- **First Hour**
  - Introduction, NoSQL and Elasticsearch 101
  - Hello Elasticsearch
- **Second Hour**
  - Elasticseach deep dive
- **Third Hour**
  - Elasticsearch for geo and aggregations
- **Fourth Hour**
  - Hack-a-thon (time permitting)

What is a NoSQL database?

- **Not Only** SQL
- #nosql originated as a hashtag for a meetup to
- Does not use the relational model
- Normally runs well on clusters
- “Mostly” open-source
- Built for the 21st century web use cases
- “Schema-less”

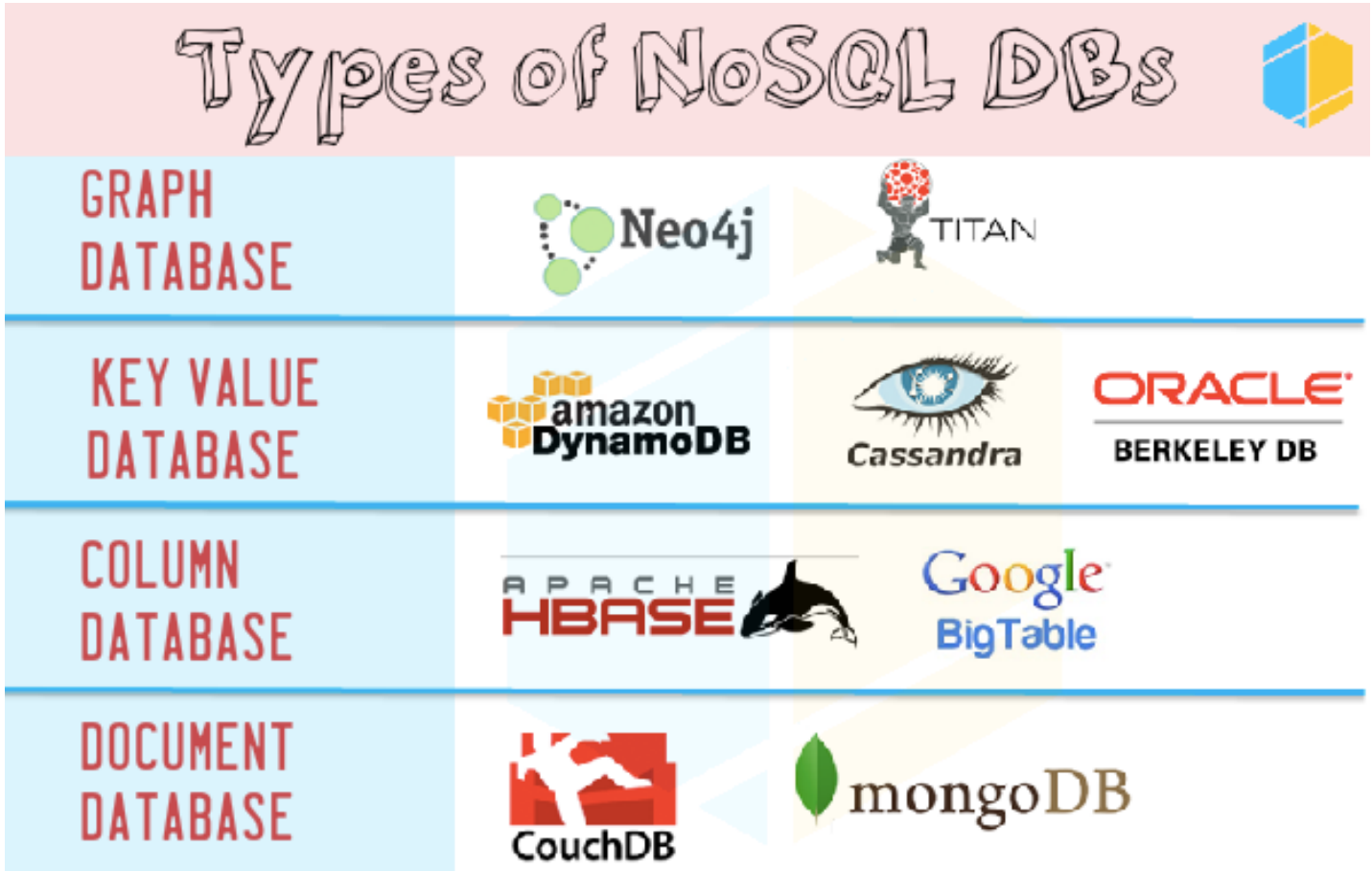
Why NoSQL databases?

- Mismatch between relational structures (tables and joins) and application
- Need to be able to perform quicker
- Need to store a lot more data than ever before
- Need to be able to evolve data structures as applications evolve

Types of NoSQL databases?

- What types of NoSQL databases exist?
- What are examples of each type?

Types of NoSQL databases?

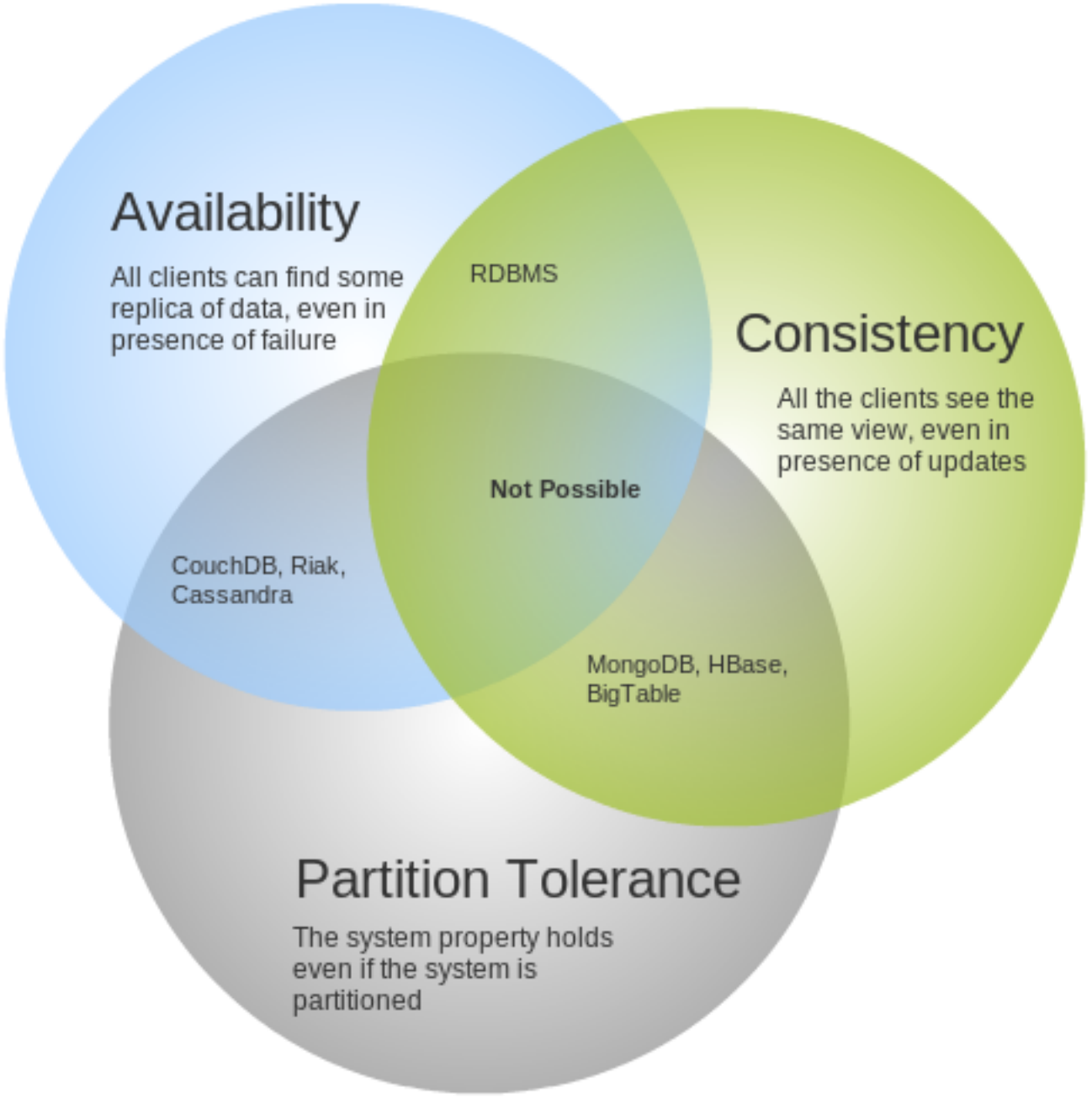


So are Relational DBs dead?

- **No!**
- NoSQL Databases are good for certain use cases but relational databases are going nowhere

CAP Theorem

- Where is Elasticsearch?



So is Elasticsearch a NoSQL Database?

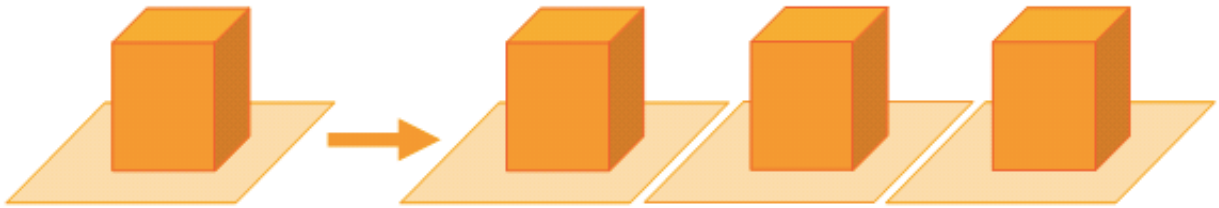
- **Yes and No!**
- Shares many characteristics with a document DB such as MongoDB
- But built and optimised for a specific task: indexing and searching big datasets
- Elasticsearch is often used as a secondary database, data is piped into ES for search

Where did Elasticsearch come from?

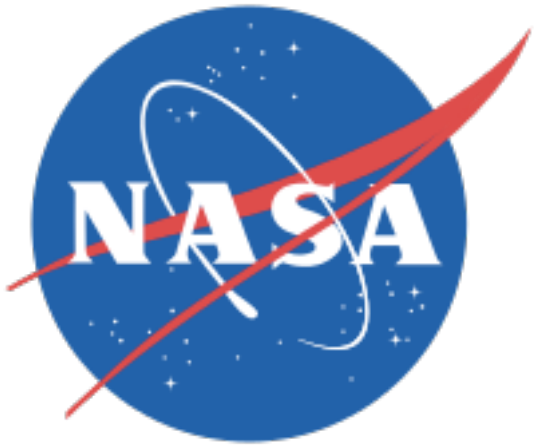
Lucene

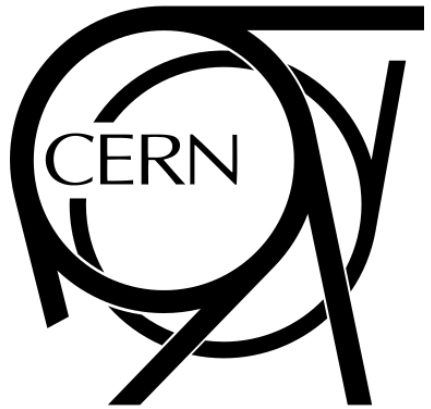
REST API

{JSON}



Who uses it?





J.P.Morgan

What are the typical use cases?

What are the drawbacks?

- Security
- Transactions
- Maturity of Tools
- Large Computations
- Data Availability
- Durability

Hello World / Hello Elasticsearch

- Pair up!
- Introduce yourselves
- Install Elasticsearch
- Fire up Elasticsearch
- Access http://localhost:9200
- “You Know, for Search…”
- Install Kopf plugin: bin/plugin install lmenezes/elasticsearch-kopf

Our First Index

curl -XPOST 'http://localhost:9200/movies/movie' -d '{ "title" : "Star Wars: Episode IV - A New Hope", "released": 1977 }'

curl -XPOST 'http://localhost:9200/movies/movie' -d '{ "title" : "Star Wars: Episode V - The Empire Strikes Back", "released": 1980 }'

curl -XPOST 'http://localhost:9200/movies/movie' -d '{ "title" : "Star Wars: Episode VI - Return of the Jedi", "released": 1983 }'

curl 'http://localhost:9200/movies/movie/_search?q=title:jedi&pretty=true'

What just happened?

- Auto-generated **index**, **type** and mappings and IDs
- Created documents and added these to the index
- Assigned Shards and Replicas
- Set the Cluster status to yellow
- How do we make it green?

- (Btw these curl commands are pretty cumbersome, let’s use Sense)

Inverted Index

- The quick brown fox jumped over the lazy dog
- Quick brown foxes leap over lazy dogs in summer

Term	Doc_1	Doc_2
Quick		X
The	X	
brown	X	X
dog	X	
dogs		X
fox	X	
foxes		X
in		X
jumped	X	
lazy	X	X
leap		X
over	X	X
quick	X	
summer		X
the	X	

- Search: “quick brown”

Term	Doc_1	Doc_2
brown	X	X
quick	X	
Total	2	1

Inverted Index cont…

- Lower case the words (terms) The > the
- Stem-reduce foxes > fox
- Synonyms: jump / leap > jump
- **Both indexed text and query string must be normalized into the same form**
- This is analysis

Term	Doc_1	Doc_2
brown	X	X
dog	X	X
fox	X	X
in		X
jump	X	X
lazy	X	X
over	X	X
quick	X	X
summer		X
the	X	X

Queries and Filters

- Elasticsearch provides a full Query DSL based on JSON to define queries
- Query context:
  - How well does this document match this query clause?
  - Each document is scored
- Filter context:
  - Does this document match this query clause?
  - No score applied – if document does not match it is excluded
  - Think of SQL WHERE= clause

- Tip: Applying filters correctly will have a huge impact on performance

A simple query

```
POST movies/movie/_search
{
   "query": {
      "match": {
         "title": "star"
      }
   }
}
```

Why does Episode 4 score lower? Explain yourself Elasticsearch!

A simple filter

```
POST movies/movie/_search
{
   "filter": {
      "range": {
         "released": {
            "from": 1980
         }
      }
   }
}
```

What about the scores?

Scoring in Elasticsearch

- Elasticsearch uses the Boolean model to find matching documents, and a formula called the practical scoring function to calculate relevance
- Is is based on the following:

  - **Term frequency** - how often does the term appear in this document?
  - **Inverse document frequency** - how often does the term appear in all documents in the collection?
  - **Field-length norm** - how long is the field? The shorter the field, the higher the weight

- It is possible at index time to control which of these to apply
- It is also possible to perform custom scoring at query time using scripts

Putting them together

```
POST movies/movie/_search
{
   "query": {
      "filtered": {
         "query": {
            "match": {
               "title": "star"
            }
         },
         "filter": {
            "range": {
               "released": {
                  "from": 1980
               }
            }
         }
      }
   }
}
```

Combining queries

```
POST movies/movie/_search
{
   "query": {
      "bool": {
         "must": [
            {
               "match": {
                  "title": "star"
               }
            }
         ],
         "should": [
            {
               "match": {
                  "title": "the"
               }
            }
         ]
      }
   }
}
```

Loading data into Elasticsearch

- Direct API posts
- Bulk API
- Logstash
- JDBC Importer (replaces “rivers”)
- ogr2ogr
- Elasticsearch official SDKs
- Third party libraries
- Roll your own!

Let’s go to the movies

- Import the movies.csv file using the JDBC importer
- Note, the version of the loader must match the version of Elasticsearch or bad things can happen!
- Note the default mapping, not very useful
- Some fields are not very useful to us either
- Let’s amend and try again
- We have a number of data types available: https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping-types.html

Exercises

- Find all movies about fish, find all movies about fishes
- Find all movies made between 1980 and 1989
- Find all movies about fish made in 1988
- Find all action movies less than 90 minutes long
- Find all action movies with a budget of greater than $100m
- Find all star wars films but not the Phantom Menace
- Find all films with an mpaa rating of “PG”
- Find all films with the title “zigzag”, now try “zig zag”
- Find all films called “Blood Money”

Controlling Analysis

- Strings are analyzed by default
- For terms such as the mpaa rating we do not want this
- We can specify how the the mappings are defined at index time
- We can also control how tokenizing works so we could look for “ZigZag” and break out to “Zig Zag” at index time
- Let’s re-index

And/Or

- If we search for “Blood Money” we get all films with “Blood” or “Money” in the title
- We can control this at query time in a number of ways:
    - Operator: and/or
    - minimum_should_match

Dealing with Order

- We would expect a film called “Dirty Dancing” to score higher than one called “Dancing Dirty”
- However a match query does not distinguish between term order by default
- We have a number of options:

    - Match phrase
    - Shingles

Multi Match Queries

- In addition to the bool query, Elasticsearch provides the multi-match query as a short-hand:
  - **best_fields**: (default) Finds documents which match any field, but uses the _score from the best field
  - **most_fields**: finds documents which match any field and combines the _score from each field.
  - **cross_fields**: treats fields with the same analyzer as though they were one big field. Looks for each word in any field
  - **phrase**: runs a match_phrase query on each field and combines the _score from each field
  - **phrase_prefix**: runs a match_phrase_prefix query on each field and combines the _score from each field

Dealing with Language

- We saw with the “fish” / “fishes” example that elasticsearch does not recognise common linguistic features such as plurals by default
- Using a “stemmer” combined with common “stop words” can help greatly
- Elasticsearch has many language analyzers available, these are presets: combinations of stemmers and stop words that are optimised for a certain language

Fuzzy Logic

- The fuzzy query uses similarity based on Levenshtein edit distance for string fields, and a +/- margin on numeric and date fields
- Fuzziness can be the number of replacements or AUTO
- An optional prefix_length can be applied, this helps enormously with performance!

Ordering Results

- By default results are returned in descending order of score
- We can control the ordering by applying a custom order
- Note that ordering results can be expensive so use with caution
- Beware using analysed fields in the order as results can be unusual
- Multi field at analysis time can overcome these issues (use a .raw field)

Aggregations

- Aggregations are a powerful feature of Elasticsearch and drive many analytical use cases
- If well designed aggregations can be very fast operating across multiple servers, shards and indexes
- We will explore a simple example here however there are some excellent tutorials available that explore this in more depth
- It is also possible to perform geographic aggregations on geohash and geodistance which could be used to provide server side clustering capability to web applications

Geo Support

- Elasticsearch has supported geo since it's earliest days
- It supports two types:
    - **Geo-points** allow you to find points within a certain distance of another point, to calculate distances between two points for sorting or relevance scoring, or to aggregate into a grid to display on a map
    - **Geo-shapes**, are used purely for filtering. They can be used to decide whether two shapes overlap, or whether one shape completely contains other shapes
- We will be focussing on the geo_point datatype today but will use a geo_shape to perform a query against a point layer

Geo Point

- There are four ways of expressing a geo-point in Elasticsearch: object, string, geohash or array
- Somewhat confusingly, strings are express “lat,lon” whereas arrays are [lon,lat] – we will be using the object format to avoid ambiguity
- In previous versions of Elasticsearch it was necessary to fine tune the indexing to achieve acceptable performance however since v2.x the defaults should be fine for most use cases
- Geo indexes were previously memory intensive however with 2.x and the move to doc values by default good performance can be achieved with disk
- For details of the inner workings of the geo types the following article is recommended: https://www.elastic.co/blog/supercharging-geopoint

Geo Point Parameters

- **geohash** - Should the geo-point also be indexed as a geohash in the .geohash sub-field?
- **geohash_precision** - The maximum length of the geohash to use for the geohash and geohash_prefix
- **geohash_prefix** - should the geo-point also be indexed as a geohash plus all its prefixes?
- **ignore_malformed** - if true, malformed geo-points are ignored. If false (default), malformed geo-points throw an exception and reject the whole document
- **lat_lon** - should the geo-point also be indexed as .lat and .lon sub-fields? Accepts true and false (default)

- Note: using lat_lon can give a good perfomance uplift on geo_bounding_box and geo_distance queries and is recommended even though not the default

Let’s load some geo data!

- We will use the geonames cities15000 dataset
- We will need to add a header record to get the csv loader to work
- We also need to provide the column datatypes as before
- We need to defined mappings for the geo_point to be able to work with this
- We also need to query the csv file to return location.lat and location.lon mappings

Bounding Box Filtering

- A query allowing to filter hits based on a point location using a bounding box
- The simplest and fastest of the Geo filters available
- The bounding box is specified as a top_left and bottom_right object
- Options:
  - **ignore_malformed** - set to true to accept geo points with invalid latitude or longitude (default is false)
  - **type** - set to one of indexed or memory to defines whether this filter will be executed in memory or indexed
- If we have set lan_lon=true in our mapping we should set type=indexed to get the performance uplift

Geo Distance Filtering

- Filters documents that include only hits that exists within a specific distance from a geo point
- In the same way the geo_point type can accept different representation of the geo point, the filter can accept it as well
- Accepts the following options:
  - **distance**: the radius of the circle centred on the specified location, can be expressed in various units
  - **distance_type**: how to compute the distance. Can either be sloppy_arc (default), arc (slightly more precise but significantly slower) or plane (faster, but inaccurate on long distances and close to the poles)
  - **optimize_bbox**: whether to use the optimization of first running a bounding box check before the distance check
  - **ignore_malformed**: as before with indexing, default is false

Geo Polygon Filtering

- A query allowing to include hits that only fall within a polygon of points
- Options are as with bounding box
- Accepts an object representing a series of points from which a polygon is contrsucted
- This kind of query is very expensive in Elasticsearch currently!
- Work is underway to optimise the handling of shapes in Elasticsearch and it is is likely in the future that the geo_point and geo_shape mappings will undergo considerable work and will very likely be merged

Sorting by Distance

- Search results can be sorted by distance from a point
- Options are:
  - **order** – asc or desc
  - **unit** – m, km etc
  - **distance_type** - how to compute the distance. Can either be sloppy_arc (default), arc (slightly more precise but significantly slower) or plane (faster, but inaccurate on long distances and close to the poles).

- Scoring by Distance is usually a better solution for this kind of requirement however this requires an in-depth look at custom scoring and is out of the scope of this workshop

Exercises

- Experiment with running the various different types of geo filters and geo sorts we have covered
- Combine these with the full text queries we learnt previously
- This is the power of Elasticsearch and the real ue case we are exploring: the combination of advanced full text searching and geo to deliver near real-time results!

Hackathon

- Work in pairs, ideally at least one developer to each pair
- Build a simple application using the Elasicsearch API to display results on the map for the geonames data
- Be prepared to present your results
- There are no prizes except the glory and admiration of your colleagues!

Top Tip: Type Ahead (Auto Complete)

- There are a number of ways of achieving this
- Prefix queries will work but are slow
- Completion suggester is highly optimised and makes use of memory
- nGrams provide a good solution but need to be considered at index time

Top Tip: Production Optimisation

- Config file tweaks
- Warmers API
- Disable refresh for big indexing jobs
- Use the bulk API for large loads
- Optimize API
- Avoiding split brain
- Load balancing
- Excluding superfluous fields
- Use prefix with fuzzy logic
- Securing behind a proxy