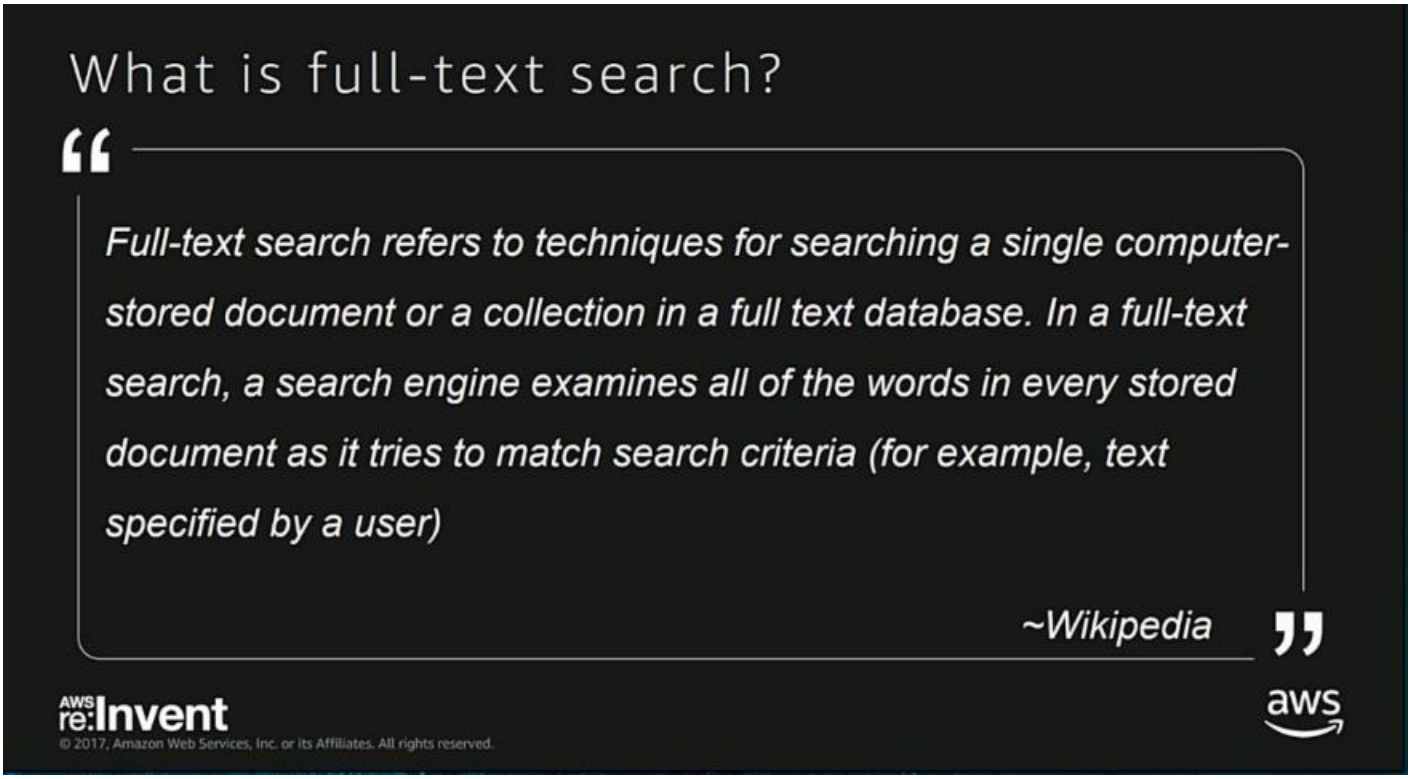
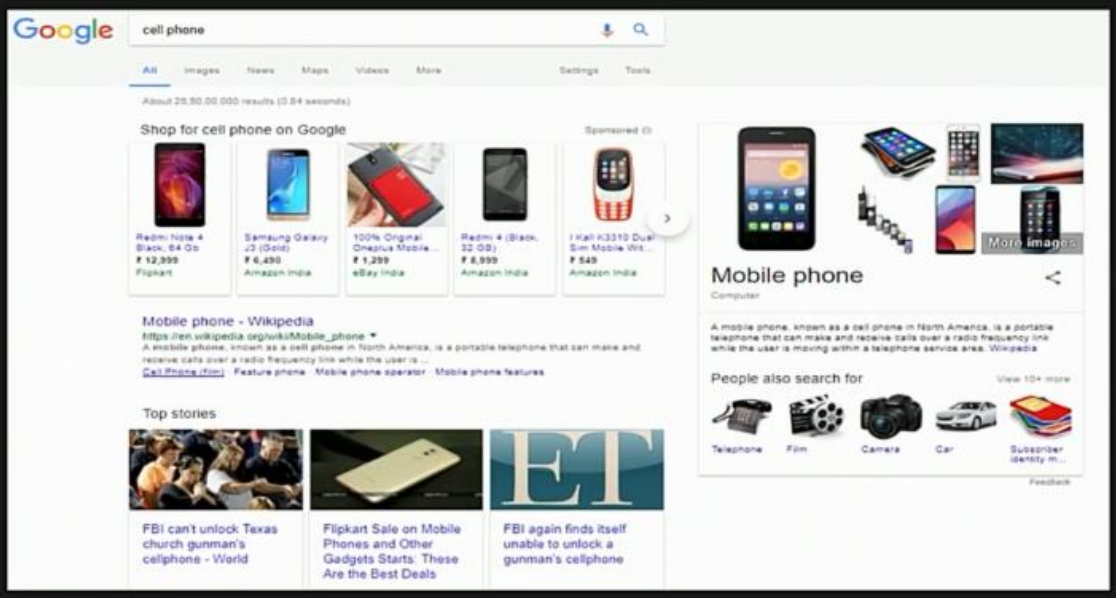
MirrorWeb offers automated website and social media archiving services with full text search capability for all content. The UK government hired MirrorWeb to provide search services across 20 years of archived data from over 4,800 websites. In this session, MirrorWeb discusses the technology stack they built using Amazon Elasticsearch Service (Amazon ES) to search across the 333 million unique documents (over 120 TB) that they indexed within a 10-hour period. They discuss how they moved data from on-premises to Amazon S3 using AWS Snowball and then processed that data using Amazon EC2 Spot Instances, reducing costs by over 90%. They also talk about how they used AWS Lambda to ingest data into Amazon ES. Finally, they share best practices for building a large-scale document search architecture.

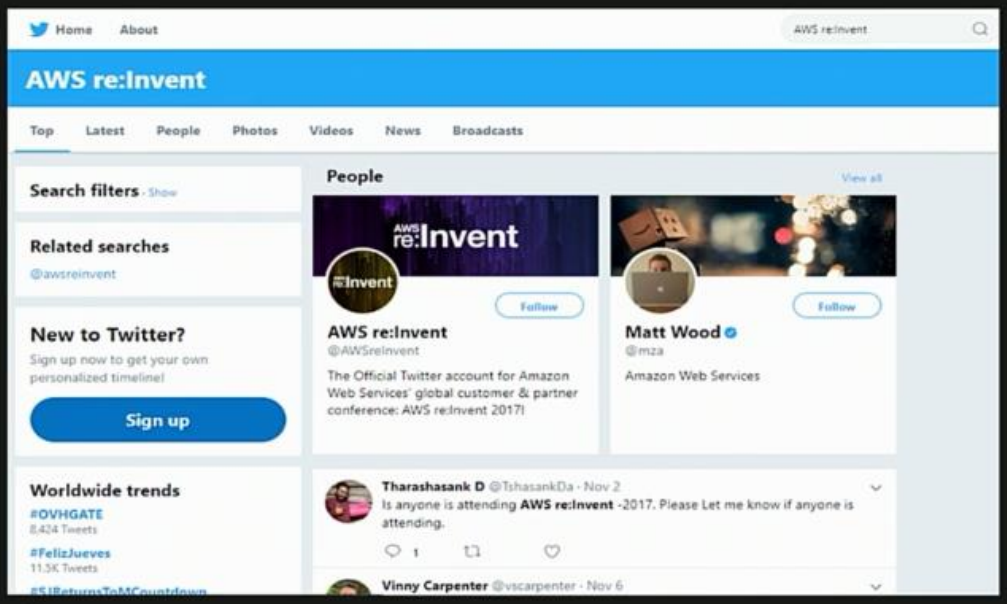


Web search



E-commerce

Social media search

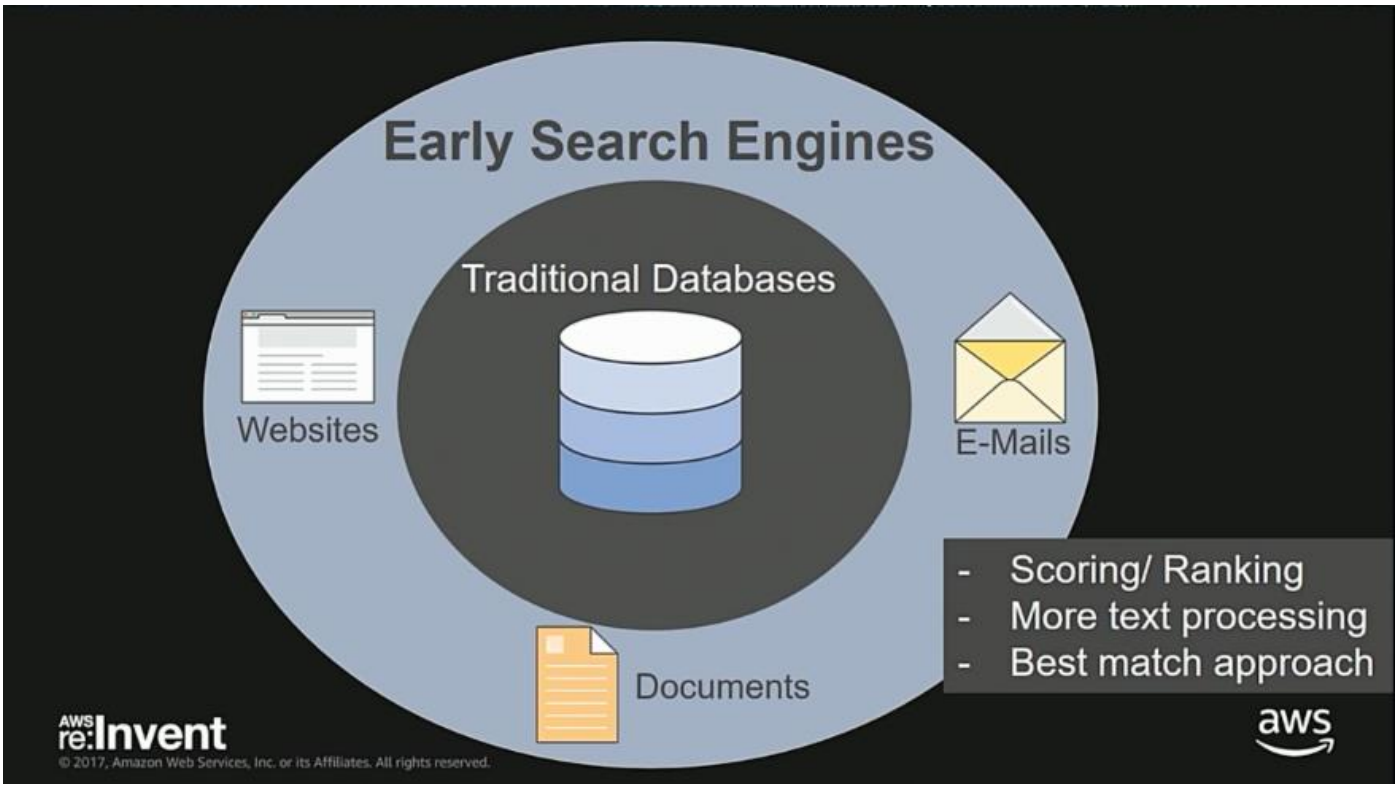


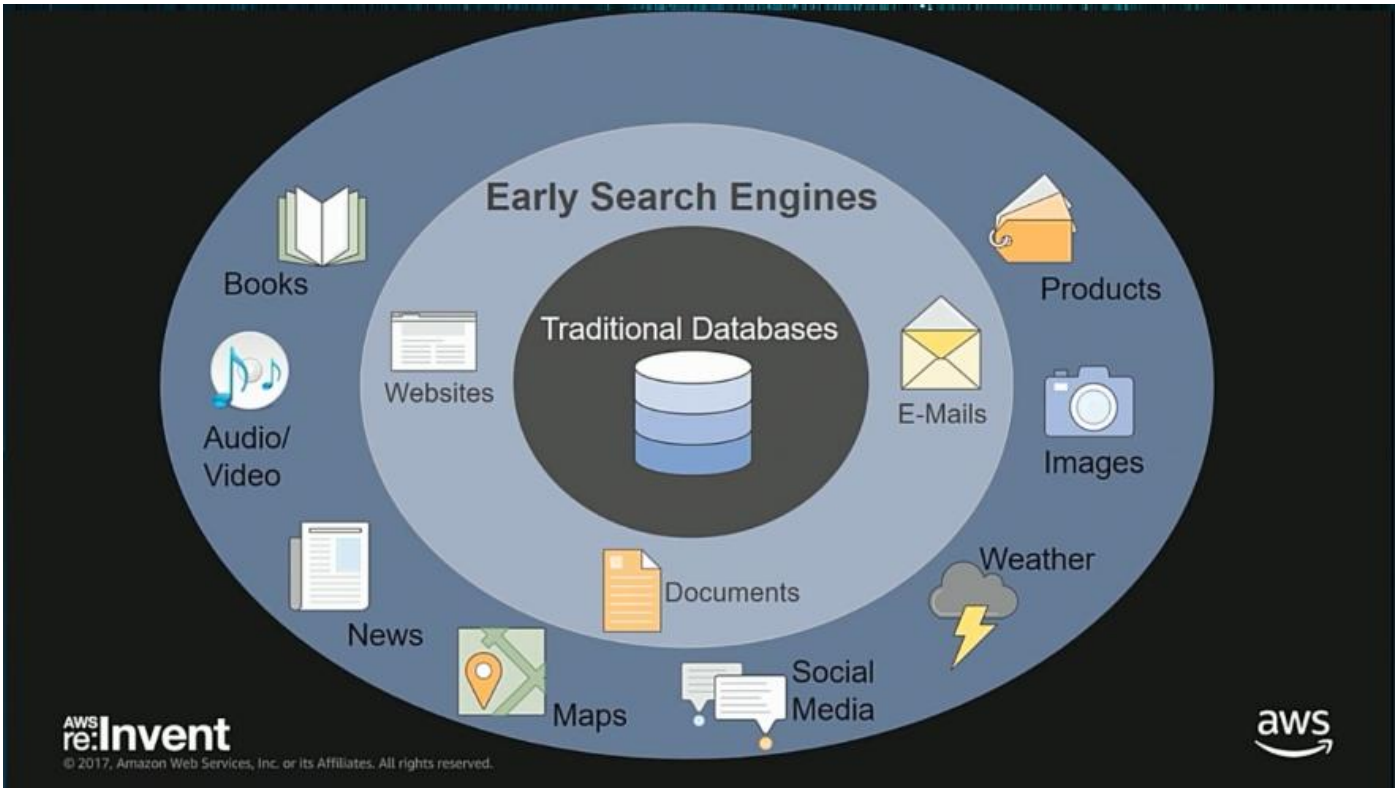
Evolution of full-text search

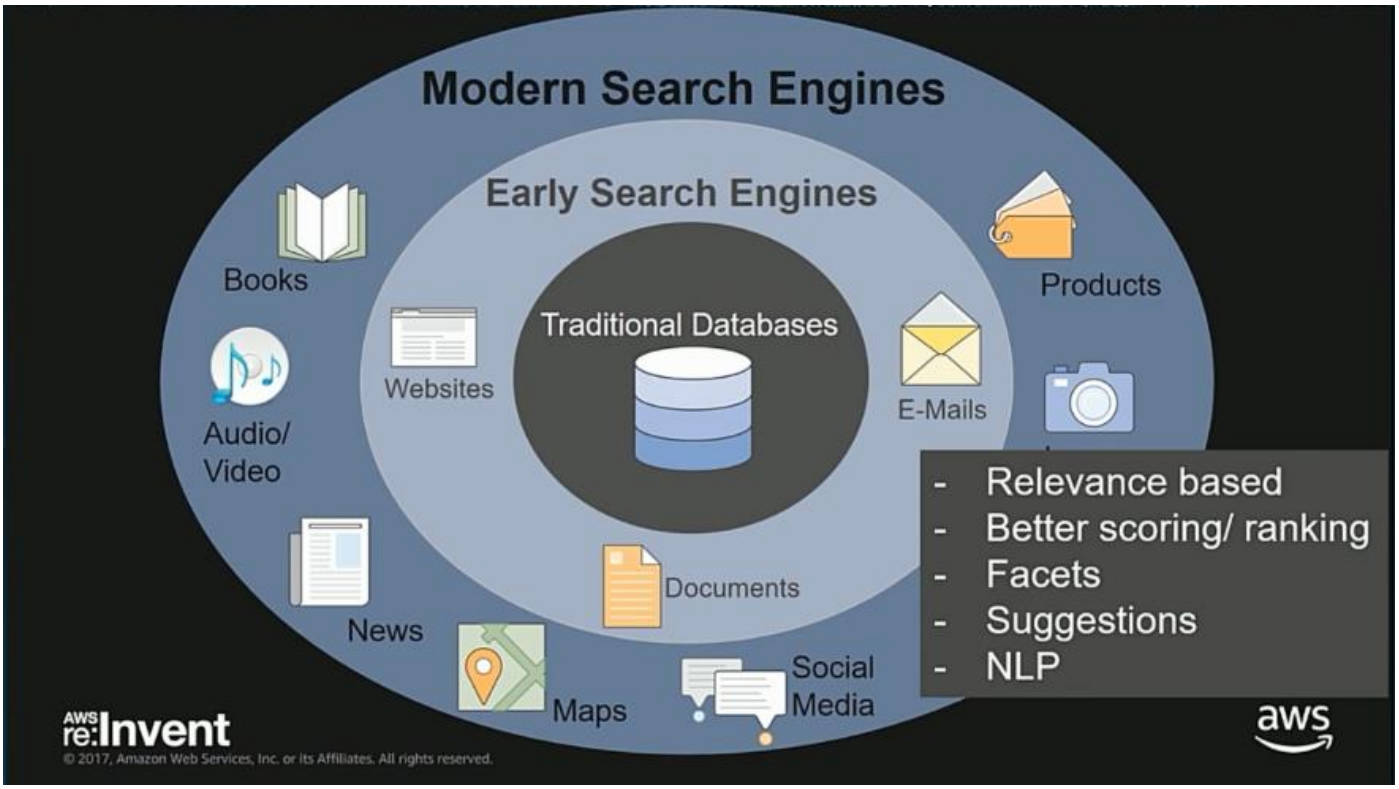
Traditional Databases

```
SELECT *
FROM BooksTable
WHERE Title LIKE '%search%'
```

- Simple string matching
- Query returns all matches
- Small volume of data
- Transactional model


Early Search Engines
Traditional Databases
Websites
E-Mails
Documents
- Scoring/ Ranking
- More text processing
- Best match approach
AWS re:Invent
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
aws


Early Search Engines
Traditional Databases
Books
Products
Websites
E-Mails
Audio/
Video
Images
Weather
Documents
News
Social
Media
Maps
AWS re:Invent
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
aws



Search → Integral part of all products and services

Structured data → Unstructured/ Semi-structured data

Data Volume ↑

Performance Requirements ↑

Text Analysis ↑

Popular search engines

Rank Nov 2017	Rank Oct 2017	Rank Nov 2016	DBMS	Database Model	Score Nov 2017	Score Oct 2017	Score Nov 2016
1.	1.	1.	Elasticsearch	Search engine	119.41	-0.82	+16.84
2.	2.	2.	Solr	Search engine	69.16	-1.97	+0.80
3.	3.	3.	Splunk	Search engine	64.87	+0.51	+10.14
4.	4.	4.	MarkLogic	Multi-model	11.55	-0.26	+1.33
5.	5.	5.	Sphinx	Search engine	5.88	-0.14	-1.11

Source: DB-Engines Ranking - https://db-engines.com/en/ranking/search+engine

- Open source
- High performance, distributed
- Analytics and search
- Easy ingestion and visualization
- Fast time to value

AWS re:Invent

aws

Amazon Elasticsearch Service

Amazon Elasticsearch gives you a managed interface for Elasticsearch

Benefits of Amazon Elasticsearch Service

Supports Open-Source APIs and Tools
Drop-in replacement with no need to learn new APIs or skills

Easy to Use
Deploy a production-ready Elasticsearch cluster in minutes

Scalable
Resize your cluster with a few clicks or a single API call

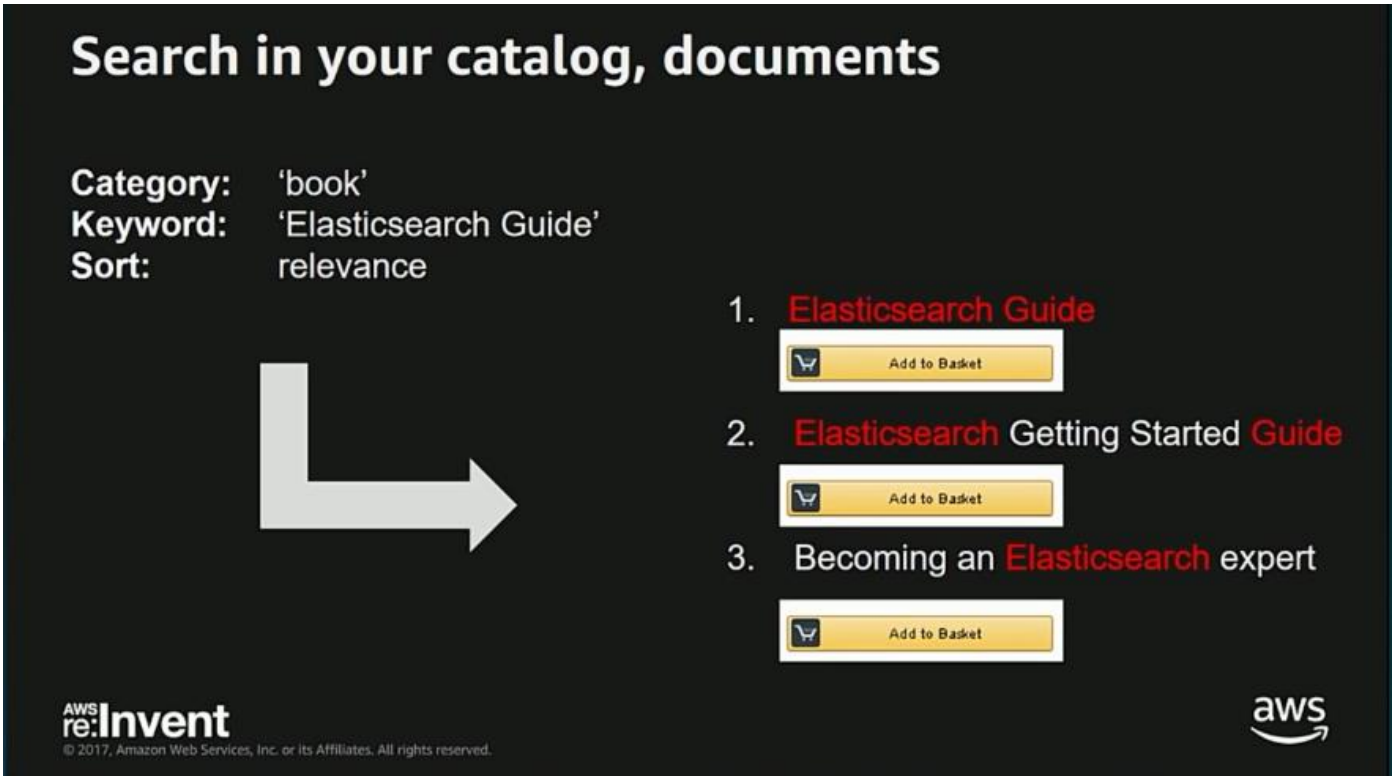
Secure
Deploy into your VPC and restrict access using security groups and IAM policies

Highly Available
Replicate across Availability Zones, with monitoring and automated self-healing

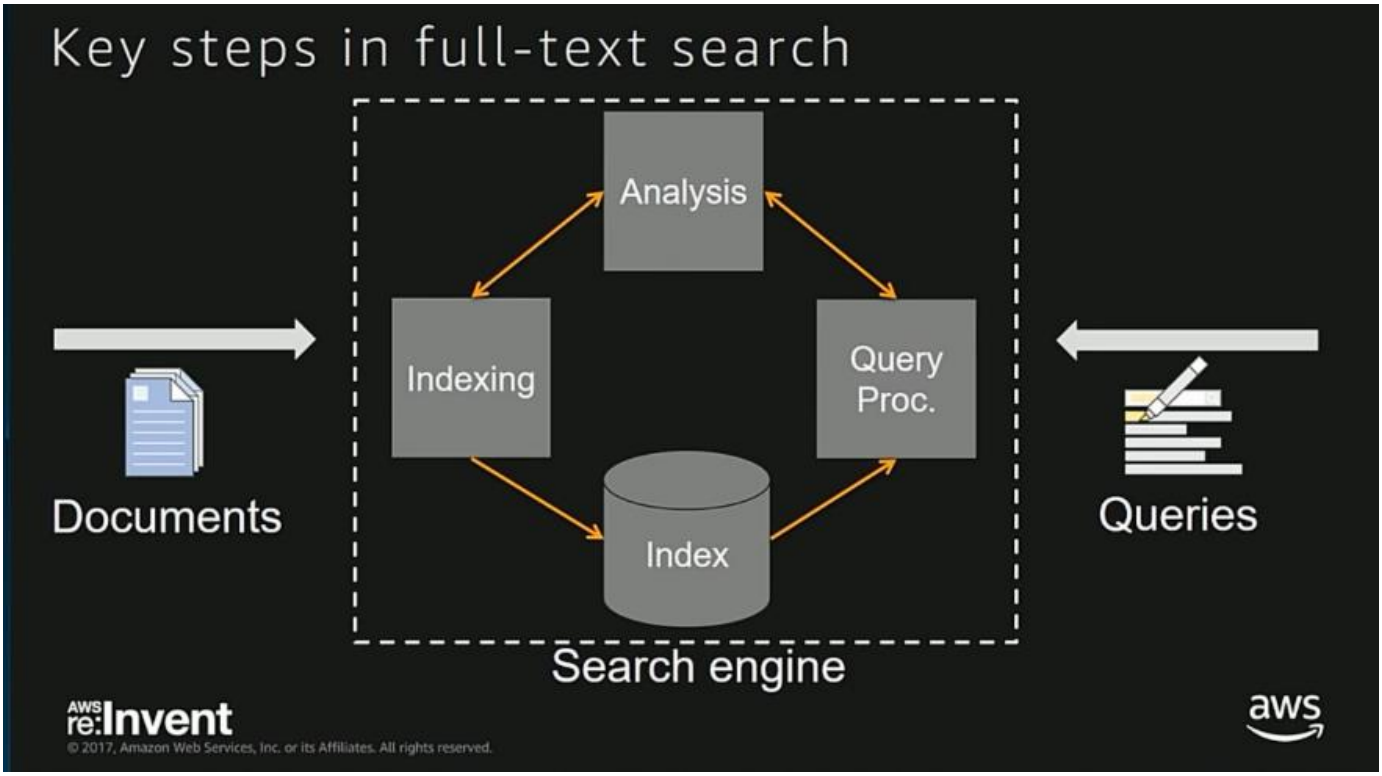
Tightly Integrated with Other AWS Services
Seamless data ingestion, security, auditing and orchestration

AWS re:Invent

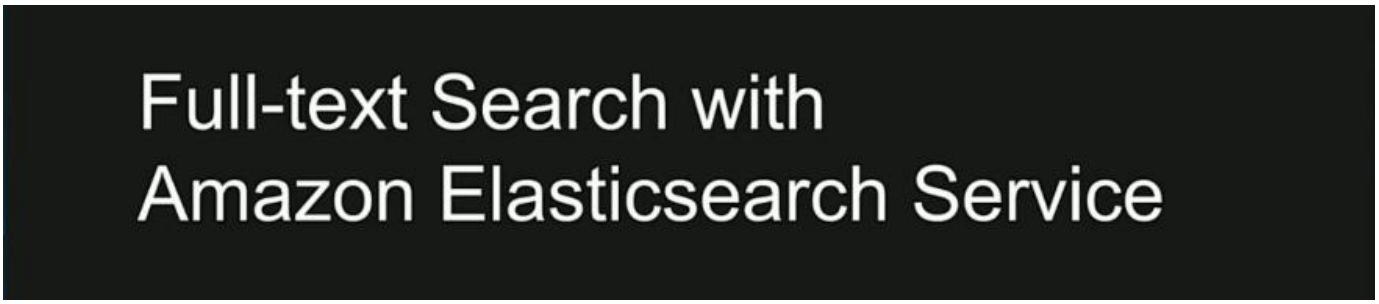
aws

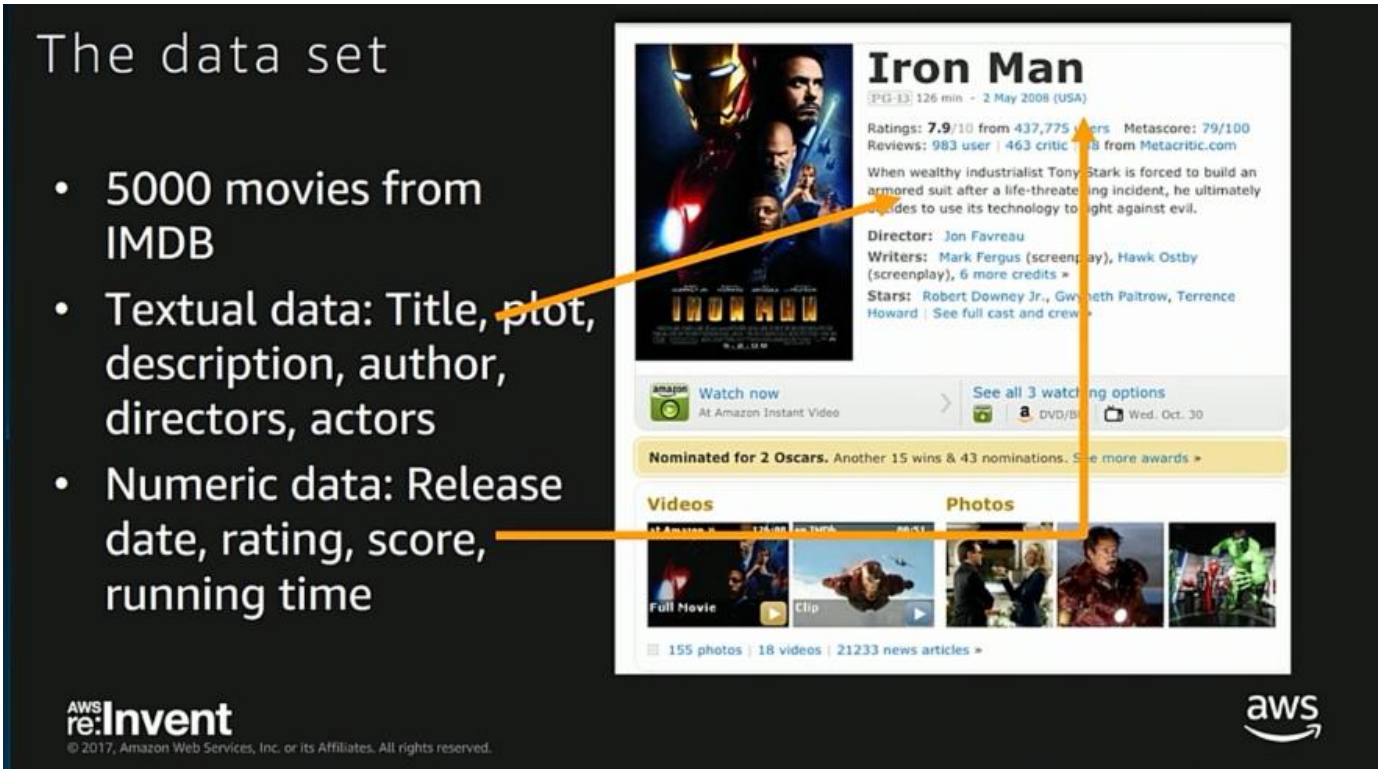


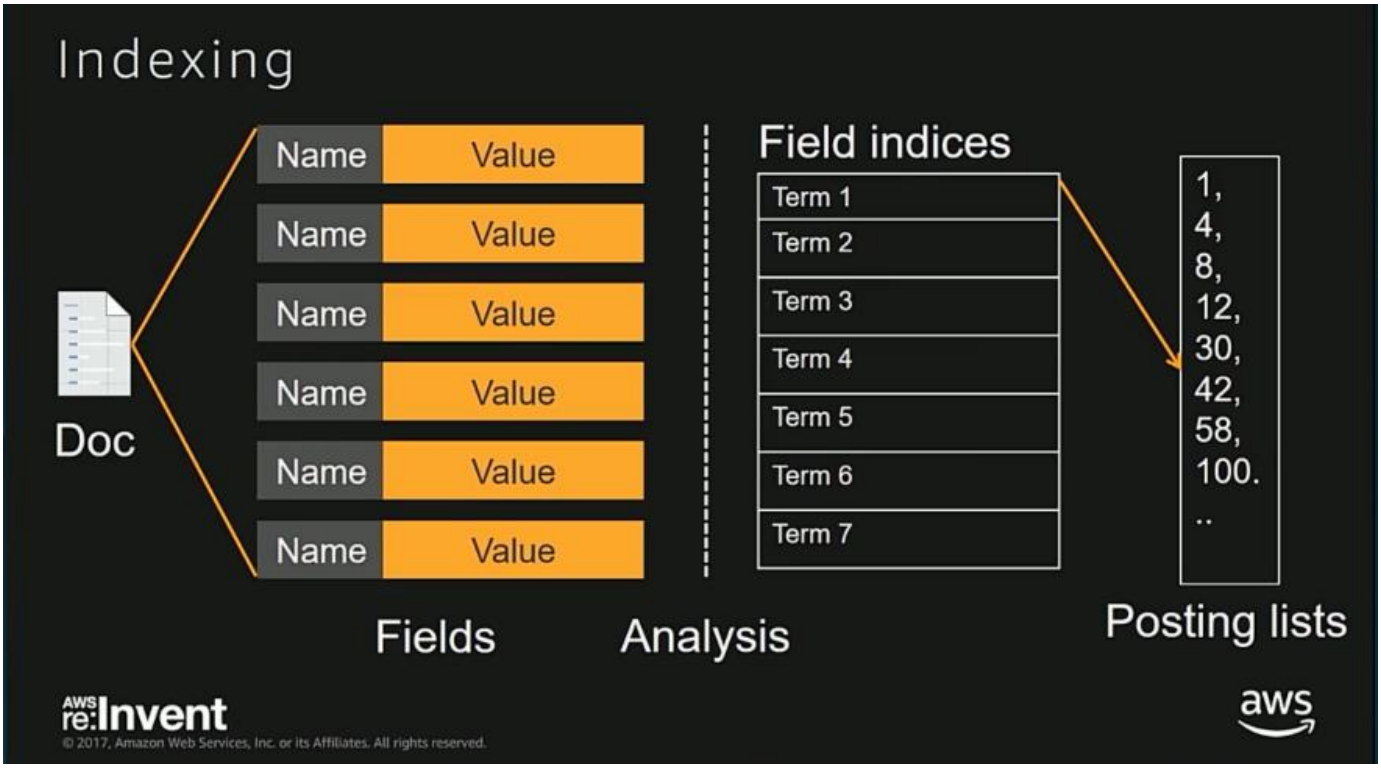
Just with a few clicks in the AWS Elasticsearch console, you can pick your cluster size and instances and the ES service will do the rest. Now you can ingest data into your ES cluster, set up your indexes, and start using the search service.



This is the simplest form of a search architecture. You simply ingest documents to the search engine, and then make queries to the search engine to get results. Inside the search engine is where the real work happens. First, the search engine takes the document that you feed it and processes/analyzes them and extract the key terms that it needs to index for you. When you now do a query against the search engine, it figures out what are the key terms that it needs to search against, goes into the index and pulls out the necessary documents, sorts them based on relevance and returns it to you.







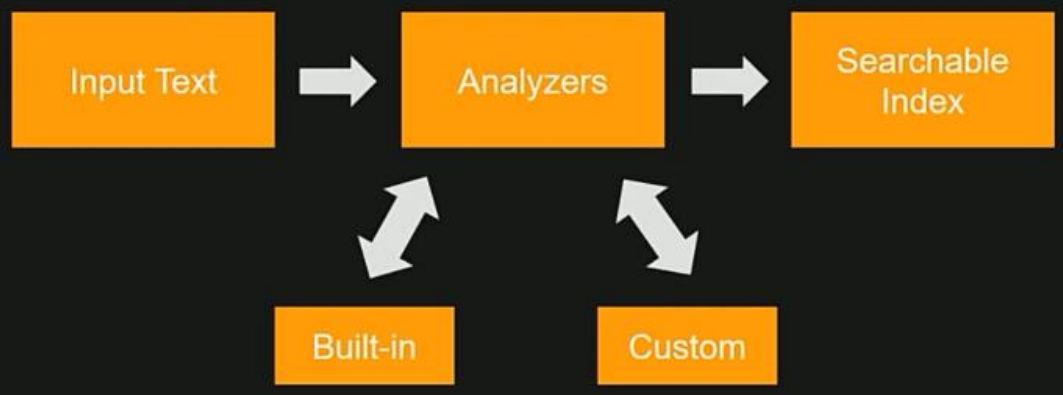
ES use the data structure called inverted indexes to create and store its indexes. A JSON document is sent to ES as name/value pairs. ES does Analysis by looking at each of the fields and extract the terms that it needs to index. For each document, ES will build a list of the document terms that match the document.

Indexing a movie

```
PUT /movies/movie/12345
{
    "title" : "Iron Man",
    "plot" : "When wealthy industrialist Tony Stark is forced to build an armored
suit after a life-threatening incident, he ultimately decides to use its
technology to fight against evil.",
    "directors" : [ "Jon Favreau" ],
    "release_date" : "2008-04-14T00:00:00Z",
    "rating" : 7.9,
    "genres" : [ "Action", "Adventure", "Sci-Fi" ],
    "image_url" : "http://ia.media-imdb.com/images/M/MV5BMTczN2._V1_SX400_.jpg",
    "rank" : 171,
    "running_time_secs" : 7560,
    "actors" : [ "Robert Downey Jr.", "Gwyneth Paltrow", "Terrence Howard" ],
    "year" : 2008
}
```


We have created an index called movies and we have a document called 12345 as above, the document has multiple fields like title, plot, etc.

Text Analysis



Indexing a movie

```
PUT /movies/movie/12345
{
    "title" : "Iron Man",
    "plot" : "When wealthy industrialist Tony Stark is forced to build an armored
suit after a life-threatening incident, he ultimately decides to use its
technology to fight against evil.",
    "directors" : [ "Jon Favreau" ],
    "release_date" : "2008-04-14T00:00:00Z",
    "rating" : 7.9,
    "genres" : [ "Action", "Adventure", "Sci-Fi" ],
    "image_url" : "http://ia.media-imdb.com/images/M/MV5BMTczN2._V1_SX400_.jpg",
    "rank" : 171,
    "running_time_secs" : 7560,
    "actors" : [ "Robert Downey Jr.", "Gwyneth Paltrow", "Terrence Howard" ],
    "year" : 2008
}
```

Text Analysis - Tokenization

```
GET _analyze
{
    "tokenizer" : "whitespace",

    "text" : "When wealthy
    industrialist Tony Stark is
    forced to build an armored
    suit after a life-
    threatening incident, he
    ultimately decides to use
    its technology to fight
    against evil."
}
```

```
"tokens": [
  {
    "token": "When",
    "start_offset": 0,
    "end_offset": 4,
    "type": "word",
    "position": 0
  },
  {
    "token": "wealthy",
    "start_offset": 5,
    "end_offset": 12,
    "type": "word",
    "position": 1
  }
  ...
```

AWS re:Invent

aws

Text Analysis – Downcasing

```
GET _analyze
{
    "tokenizer" : "whitespace",

    "filter" : ["lowercase"],

    "text" : "When wealthy
    industrialist Tony Stark is
    forced to build an armored
    suit after a life-
    threatening incident, he
    ultimately decides to use
    its technology to fight
    against evil."
}
```

```
when wealthy industrialist tony
stark is forced to build an
armored suit after a life-
threatening incident, he
ultimately decides to use its
technology to fight against
evil.
```

AWS re:Invent

aws

Text Analysis – Stop word removal

```
GET _analyze
{
    "tokenizer" : "whitespace",

    "filter" : ["lowercase",
    { "type": "stop",
      "stopwords": ["a", "an", "is", "to"]}],

    "text" : "When wealthy industrialist Tony
    Stark is forced to build an armored suit
    after a life-threatening incident, he
    ultimately decides to use its technology
    to fight against evil."
}
```

when wealthy industrialist tony stark ~~is~~ forced ~~to~~ build ~~an~~ armored suit after ~~a~~ life-threatening incident, he ultimately decides ~~to~~ use its technology ~~to~~ fight against evil.

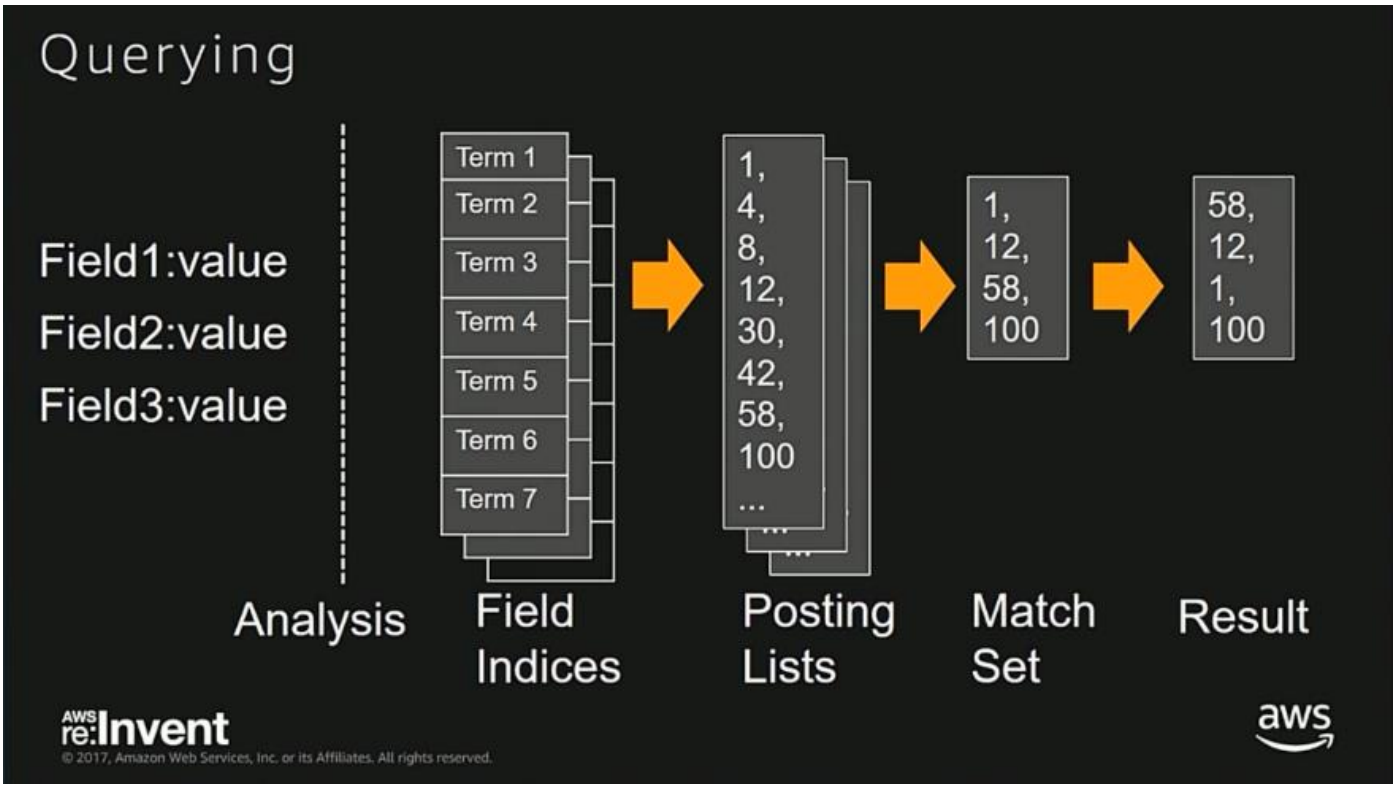
We are customizing it further by specifying the ***stopwords*** that we want to get stripped off.

Text Analysis – Stemming

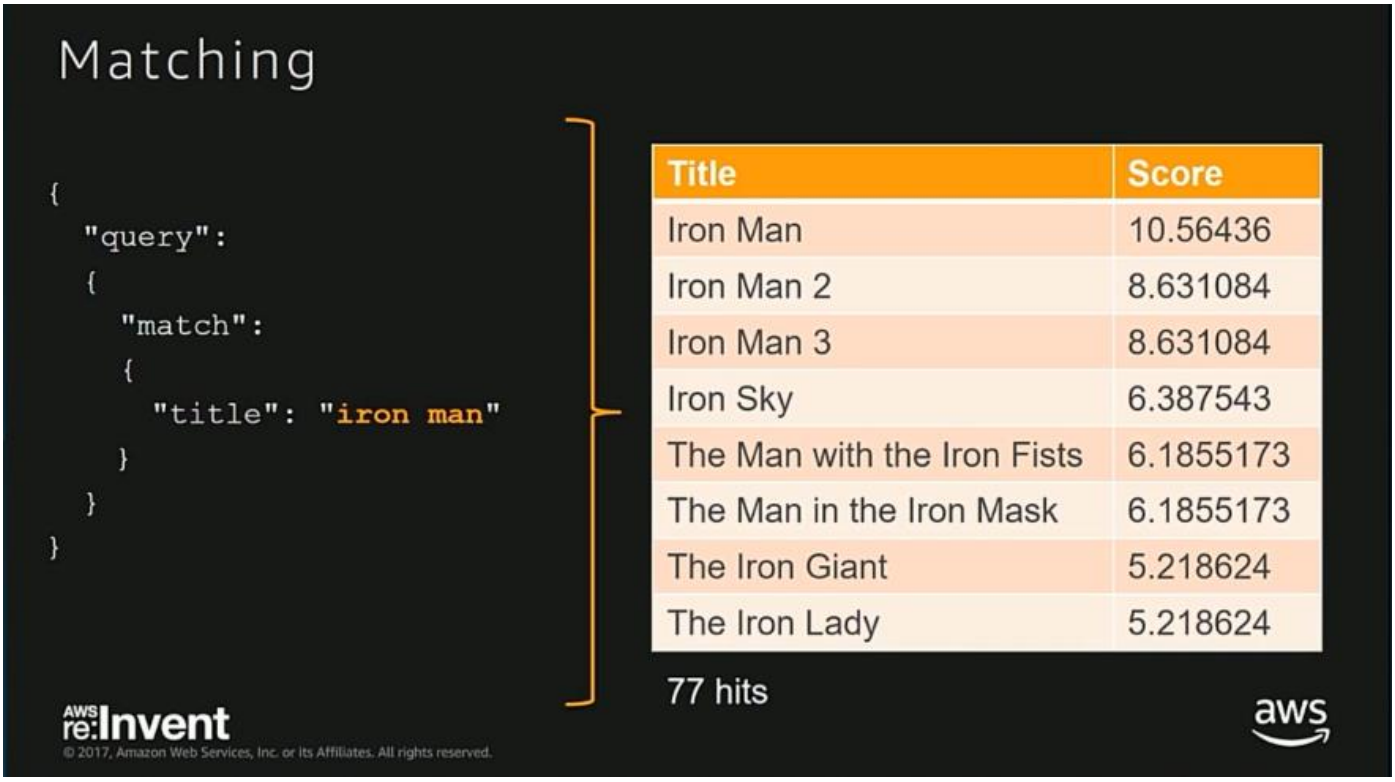
```
GET _analyze
{
    "tokenizer" : "whitespace",

    "filter" : ["lowercase",
    { "type": "stop",
      "stopwords": ["a", "an", "is", "to"]},
    {"type": "stemmer", "name": "lovins"}],

    "text" : "When wealthy industrialist
    Tony Stark is forced to build an armored
    suit after a life-threatening incident,
    he ultimately decides to use its
    technology to fight against evil."
}
```

when **wealth industr ton** stark **forc** build **armor** suit after **lif threat incid**, he **ultim dec us** it **technolog** fight against evil

We are using a ***stemmer*** called ***lovins*** that translate some of the words into their root forms



Querying is all about specifying terms to match against. ES first analyzes your query by transforming it to the normalized form, then it looks for all the documents that match, then it ranks the results and returns the results



Matching – Whole phrase

```
{
  "query":
  {
    "match_phrase":
    {
      "title": "iron man"
    }
  }
}
```

Title	Score
Iron Man	10.56436
Iron Man 2	8.631084
Iron Man 3	8.631084

3 hits

Compound Queries

```
{
  "query": {
    "bool": {
      "must": [
      {
        "match": {"title": "iron man"}
      }],
      "filter": [
      { "range": {"rating": { "gte": 7 }
      }}]
    }
  }
}
```

Title	Rating	Score
Iron Man	7.9	10.0832
Iron Man 3	7.4	8.3512
Iron Man 2	7	8.2564
The Iron Giant	7.8	5.3499
Rain Man	8	4.5572

33 hits

Aggregations – Faceted drill down

```
{
  "query": {
    "match": {"title": "iron man"}
  },
  "aggs" : {
    "rating" : {
     "range": {
        "field": "rating",
        "ranges" : [{"from": 7, "to": 8},
                    {"from": 8}]
        }
      }
    }
}
```

Rating	Count
From 7 to 8	27
From 8	6

77 hits

We specify facets and get back result buckets

Scoring / Ranking

- Default scoring algorithm
  - TF = Term Frequency – for each document
  - IDF = Inverse Document Frequency – across all documents
  - Field length
- A number of additional options such as:
  - Field value based ranking
  - Via rank functions that include document information

Scoring – Boosting the score

```
{
  "query": {
    "multi_match": {
      "query": "James Bond",
      "fields": ["title", "plot"]
     }
  }
}
```

Top ranked result = “Casino Royale”
Score = 10.46

```
"fields": ["title^5", "plot"]
```

Top ranked result = “Bond 24”
Score = 32.63

AWS re:Invent

aws

We increased the weight given to the title score and it will prioritize the title matches more than other available fields

All in all ...

- ✓ Elasticsearch is a great technology for full-text search
  - ✓ Key operations - Indexing, Analysis, Querying
  - ✓ Several options to extend the core functionality
- ✓ Amazon Elasticsearch Service makes things a lot easier
- ✓ AWS ecosystem helps drive even more value

AWS re:Invent

aws

MirrorWeb

Philip Clegg,
CTO, MirrorWeb

MirrorWeb

Introduction to MirrorWeb and the UK Government Web Archive

AWS re:Invent

aws

Who are MirrorWeb ?

Website Archiving

Social Media Archiving

Public Sector and Regulated Industries

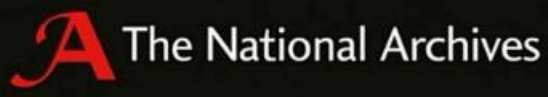
AWS re:Invent

MirrorWeb

aws

Public Archives

UK Government
Web Archive



http://webarchive.nationalarchives.gov.uk

UK Parliament
Web Archive

http://webarchive.parliament.uk

MirrorWeb

aws

What are Web Archives?

- Website data stored in ISO standard WARC format
- Play back requires the creation of an index of WARC file contents – CDX File

AWS re:Invent
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

MirrorWeb

aws

CDX Indexing

- An index for a web archive (WARC or ARC) is know as a CDX file.
- A CDX file is typically a sorted plain-text file with each line representing info about a single capture in an archive.
- CDX file is generated for every WARC file
- Typically produced with Apache Hadoop but we did it with AWS Lambda

MirrorWeb

aws

What is the UK Government Web Archive?

- 20 Years of historic archives
- Over 120TB data
- Over 4800 archived Government sites
- Archived Government Twitter accounts
- Thousands of archived YouTube Videos

MirrorWeb

aws

UK Government Web Archive Project

- Collect the historic archives from the UK National Archives
- Develop a public facing website
- Full replay of all archived sites
- Full text search across the entire historic archive

The National Archives

MirrorWeb

aws

Indexing 120TB of Web Archives

Indexing 120TB of Web Archives

- 120TB of Web Archives in 100MB files
- Only government domains are in scope – filtering stage
- Only certain mime types are in scope – filtering stage
- The data set contains many duplicate pages

MirrorWeb

aws

We spin up a very large cluster to do the initial ingest and then scaled it back down when finished and only need to provide query results using the index.

There are other OSS tools that can also be used for indexing web archives but none of them pushes automatically to ES

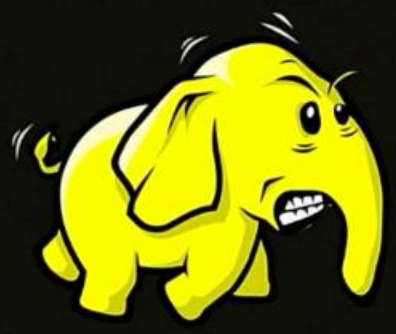
Ideal Steps with Amazon EMR

MirrorWeb

aws

Indexing 120TB of Web Archives

- So, we hired a Hadoop contractor!
- And they quoted us for 1-2 weeks of work
- It soon turned in to 6-8 weeks of work…



AWS re:Invent
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

MirrorWeb

aws

“Hadoop is great for batch data processing!”

MirrorWeb

aws

“…on small amounts of large files”

AWS re:Invent
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

MirrorWeb

aws

We had 1.2 million ~100mb compressed archives.

MirrorWeb
aws

Since we had our files as 100MB chunks that were also gzipped, we had to move the whole files into HDFS in order to process them and we couldn't pull in small sized files in batches into HDFS directly from S3.

So we decided to think outside the box

MirrorWeb
aws

WarpPipe.

Clustered data processing, for the cloud.

MirrorWeb
aws

This is like a rewrite of Hadoop for the cloud.

WarpPipe

- Speed - High Concurrency
- Security - Transit and at-rest Encryption.
- Per worker Amazon S3 Access
- Cluster scalable on the fly without data loss

MirrorWeb

aws

Actual Steps with WarpPipe and Amazon EC2

MirrorWeb

aws

Optimizing Elasticsearch Cluster for Ingest

- 9 node ES Cluster using r4.4xlarge instances - 144 Cores
- Elasticsearch index set to 1 shard per CPU totaling 144 Shards
- No replica shards
- Use of the bulk ingest API to to insert groups of records per worker

MirrorWeb

aws

Once the extract worker got to 2MB in size, we push that chunk of data into ES

It was fast…

Averaging at 146 Million Documents per hour!

MirrorWeb

aws

How did we De-duplicate?

- Deduplication by producing MD5 digest of the document url and document WARC md5 digest
- Ex: md5 of following string to produce ElasticSearch _id

```
https://www.gov.uk/:XXQJCWTY66GZQVZLAUSNFDLUMJYG6L5U
```

MirrorWeb

aws

What was the result of Deduplication?

- 1.4 Billion Documents Indexed by WarpPipe
- Deduplication by sharding url and document md5 digest
- 333 Million Unique documents in Elasticsearch
- Index reduced from 8TB 2.9TB due to deduplication

MirrorWeb

aws

Optimizing Elasticsearch Cluster for Search

- Use the ES shrink index API to reduce shards from 144 to 12 shards
- Cluster downgraded to 6 r4.xlarge instances
- 3 x m3.large master nodes
- Add a replica index to improve speed and redundancy

Amazon Elasticsearch Service

What was the cost?

- 10 hours of r4.xlarge at standard price would have cost $2,960
- We used a maximum of 1000 r4.xlarge EC2 instances for WarpPipe ingest workers
- Our Spot purchased cost was around $187
- 136 Hours of r4.4xlarge.elasticsearch instances - $237.86

Lessons Learnt

- You can easily bring down Elasticsearch with 1000 servers hitting it!
- Tuning the Elasticsearch cluster for indexing is important.
- There is a 40TB soft limit on Amazon EBS volumes
- Spot purchasing can save loads of money

AWS re:Invent

MirrorWeb
aws

Concluding Statistics

- 93% faster than the UK Web Archive's Hadoop Cluster
- 70% reduction in cost through the use of Spot purchases
- 60% reduction in index size due to deduplication

AWS re:Invent
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
MirrorWeb
aws

Closing Thoughts

Elasticsearch for full-text search

- Highly distributed, scalable, extensible service
- Rich options for Text Analysis
- Supports a plethora of search features
  - Filtering
  - Suggestions
  - Facets
  - Highlights
  - Aggregations
  - Adjustable ranking and more

AWS re:Invent

aws

AWS re:Invent

THANK YOU!

AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws