



elastic



# logistics



- **Class Hours:**
  - Start time is 8:30am
  - End time is ~4:00pm
  - Class times may vary slightly for specific classes
  - Breaks mid-morning and afternoon (10 minutes)
- 
- **Lunch:**
  - Lunch is 11:45am to 1:00pm
  - Yes, 1 hour and 15 minutes
  - Extra time for email, phone calls, or simply a walk.



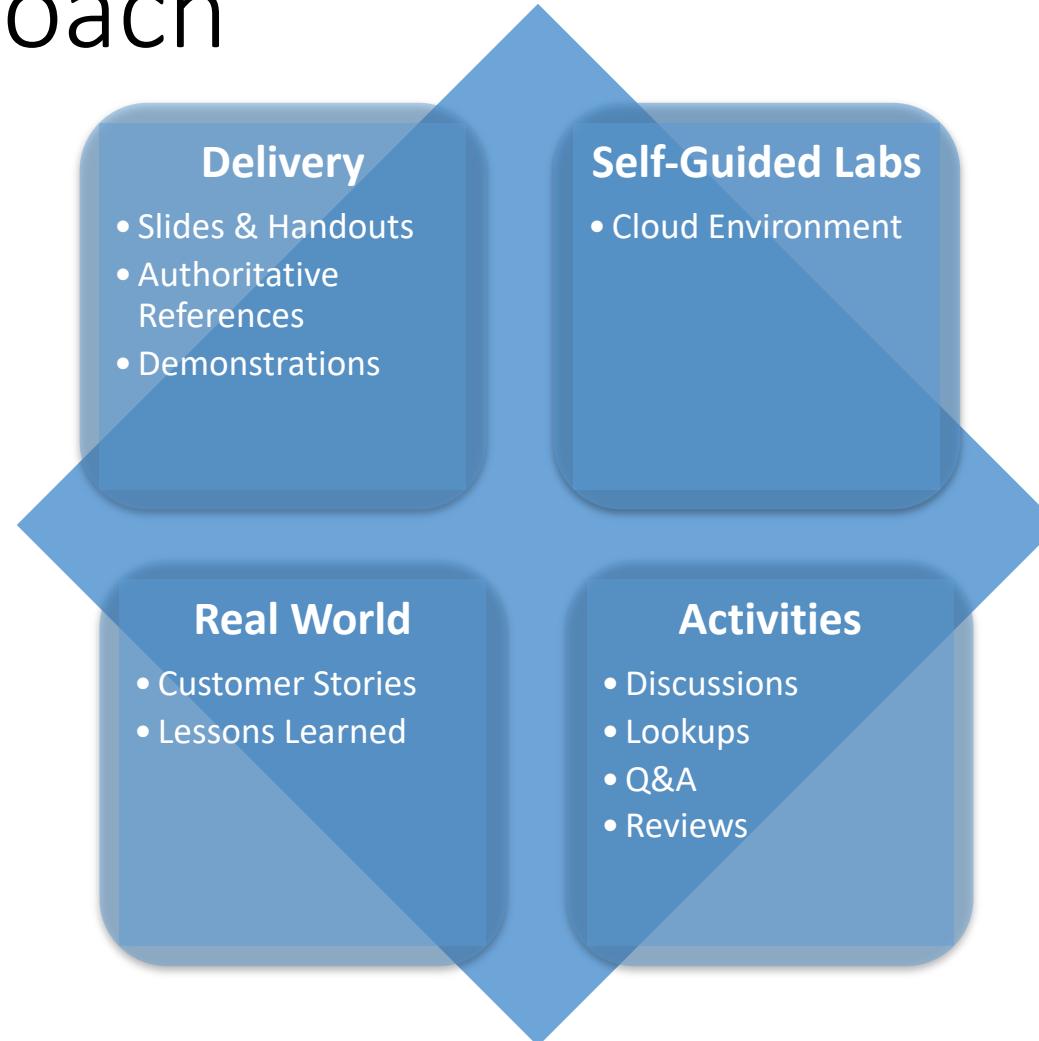
- **Telecommunication:**
- Turn off or set electronic devices to vibrate
- Reading or attending to devices can be distracting to other students

- **Miscellaneous**
- Courseware
- Bathroom

# course objectives

- Get familiar with Elasticsearch as a distributed database and search engine
- Install and configure Elasticsearch
- Discuss the Elasticsearch architecture
- Understand Elasticsearch Domain-specific Language
- Understand difference between search and filter
- Setup Logstash to transform log lines into JSON
- Setup Kibana as a graphical front-end
- Best practices for scaling Elasticsearch

# course approach



End of Day Office Hours - 1:1 Assistance



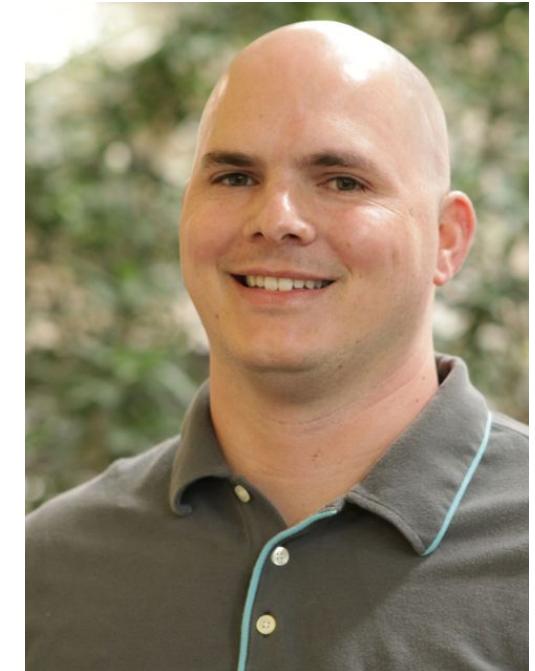
# the training dilemma



# meet the instructor

Jason Smith

Cloud Consultant with a Linux sysadmin background.  
Focused on cloud-native technologies: automation,  
containers & orchestration



Twitter  
[@jruels](https://twitter.com/jruels)

github  
<https://github.com/jruels>

mail  
[jason@innovationinsoftware.com](mailto:jason@innovationinsoftware.com)

LinkedIn  
<https://www.linkedin.com/in/jruels/>

Expertise  
• Cloud  
• Automation  
• CICD  
• Docker  
• Kubernetes

# introductions

- Name
- Title or Job Role if your title doesn't tell me what you do
- Which statement best describes your Elasticsearch experience?
  - a. I am ***currently working*** with Elasticsearch on a project/initiative
  - b. I ***expect to work*** with Elasticsearch on a project/initiative in the future
  - c. I am ***here to learn*** about Elasticsearch outside of any specific work related project/initiative
- Expectations for course (please be specific)
- Why are you here?

# elastic stack overview

# Elastic Stack – Evolution



- 2010 - Released as Open Source project
- 2012 - Elasticsearch company founded
- 2015 - Rebranded as Elastic
  - Highly scalable
  - Open Source
    - Enterprise support available
  - Built for searching and analyzing large datasets

# Elastic Stack



- 2009 - Released as Open Source project
- 2015 - Added to Elastic family
- 2015 - Log forwarder released

- Open Source data collection engine
- Real-time pipelining capabilities
- Collect logs from multiple input sources and send to Elasticsearch

# Elastic Stack



elasticsearch

- 2011 - Released as Open Source project
- 2013 - Added to Elastic family



logstash

- Browser based analytics & search dashboard for Elasticsearch
- Visualize Elasticsearch data
- Highly customizable



kibana

# Elastic Stack



elasticsearch



logstash



kibana



INNOVATION  
SOFTWARE  
LEARN. EMPOWER. INNOVATE.

# Elastic Stack



elasticsearch



logstash



kibana

- 2015 - Beats tools released
- Open platform for single-purpose data shippers



beats



INNOVATION  
SOFTWARE  
LEARN. EMPOWER. INNOVATE.

# Elastic Stack



elasticsearch



logstash



kibana

- 2016 - X-Pack released
  - Renamed in 2018 to Elastic Stack
- Features
- Tools to enable monitoring, alerting, reporting, machine learning and much more



x-pack



beats



INNOVATION  
SOFTWARE  
LEARN. EMPOWER. INNOVATE.

# Elastic Stack



elasticsearch

“ELK Stack”



logstash



Features



kibana

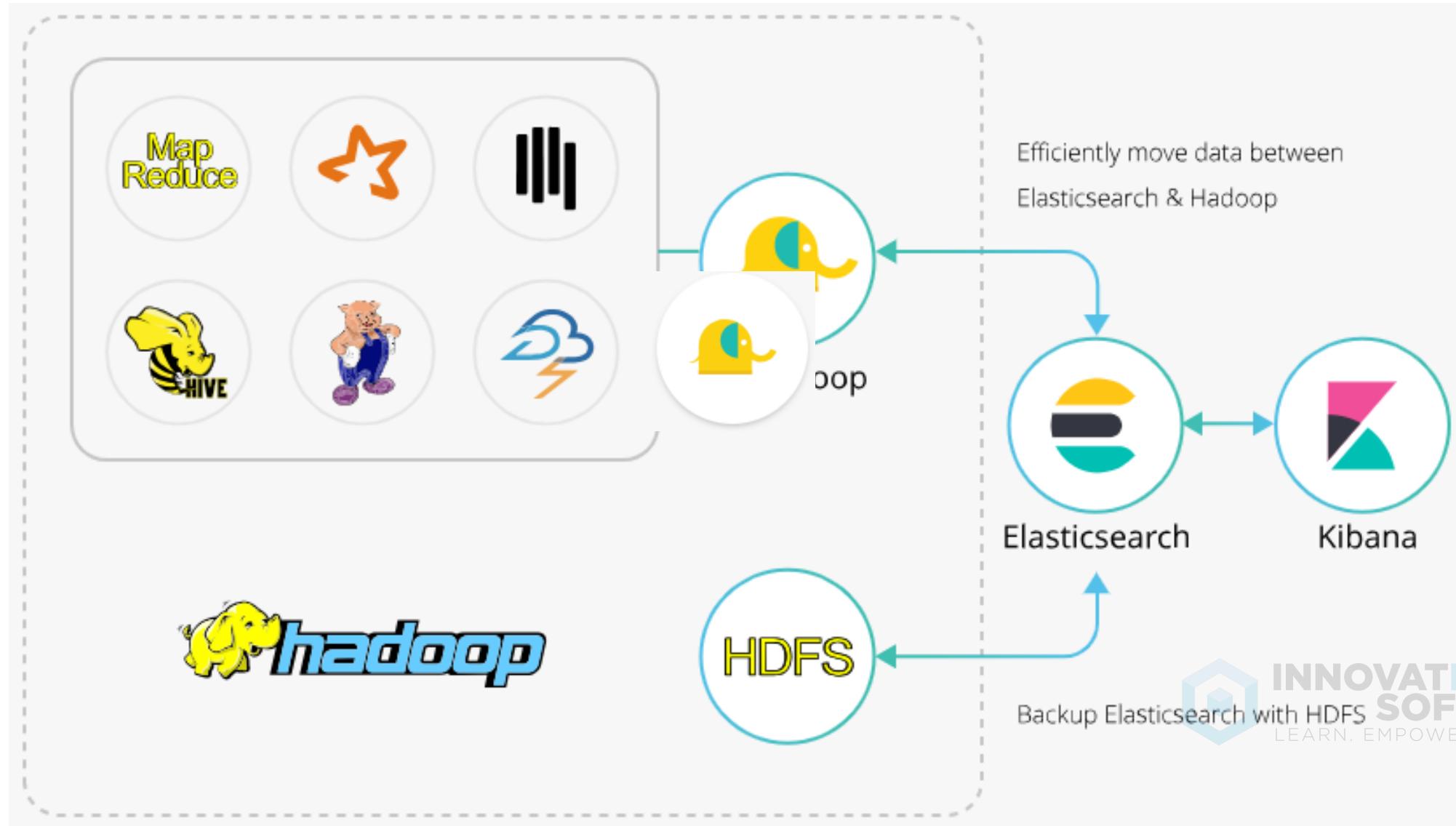


beats



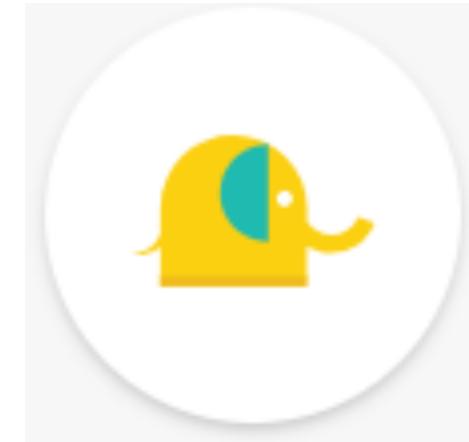
INNOVATION  
SOFTWARE  
LEARN. EMPOWER. INNOVATE.

# Elasticsearch - Hadoop



# ES-Hadoop

- ES-Hadoop is a single distributable binary that allows the use of HDFS as an archive repo for Elastic Stack
- Works with open source and distributed versions of Hadoop, e.g. Cloudera & Databricks
- Index data from Hadoop into Elasticsearch for high performance querying and aggregation of Big Data



# Elastic Stack use cases

## Ecommerce

Utilize Elasticsearch as your storefront search mechanism storing product or service information and enabling features like typeahead

## Internet of Things

Store streams of device data in Elastic Stack and visualize that information for customers, e.g. show all the active snowplows in the city on an interactive map

## DevOps

Collect log and transactional data to identify patterns or anomalies in the day to day operations of your applications, processes, services and APIs

## Data Visualization

Store data in Elasticsearch and build custom dashboards in Kibana to show the effects of changes, both expected and unexpected

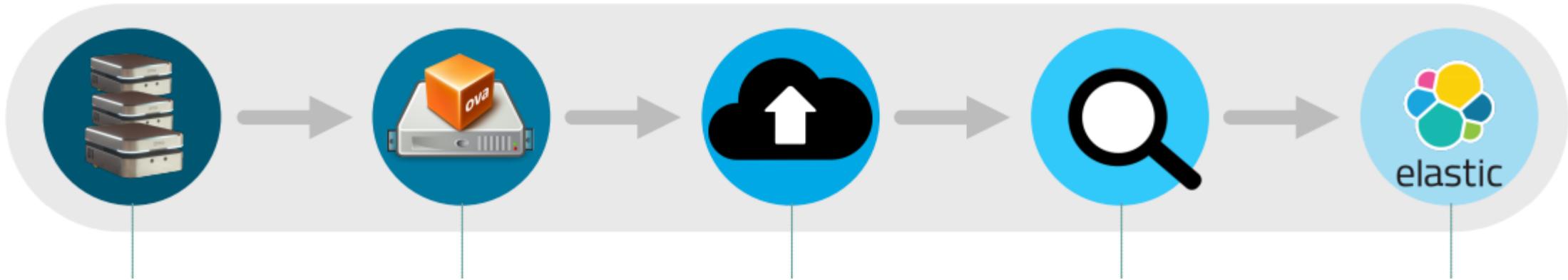
## Alerting

Pattern reverse search against queries to identify conditions that meet a certain criteria, e.g. tell me when a flight drops below X\$ during a set timeframe

## Advanced Analytics

Run analytics against existing data to enable the use of ad-hoc queries for decision support and business intelligence

# Elastic for Internet of Things (IoT)



## Things

Devices send data streams to be captured, analyzed, and aggregated

## Smart Hubs

Data is filtered, enriched and aggregated at the hub

## Network

Data is sent to the centralized environment for identification of patterns and anomalies

## Analysis

Data is analyzed for origin, content and relevance and accepted or rejected

## Storage

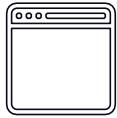
Accepted data is indexed and stored in Elastic to allow for querying and visualization



**INNOVATION  
SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

# elasticsearch basics

# logical concepts of elasticsearch



## documents

Documents are the things you're searching for. They can be more than text – any structured JSON data works. Every document has a unique ID, and a type.



## types

A type defines the schema and mappings shared by documents that represent the same sort of thing. (A log entry, an encyclopedia article, etc.)

Elasticsearch is moving away from ‘types’. In Elasticsearch 6 only one type is allowed per index.



## indices

An index powers search into all documents within a collection of types. They contain inverted indices that let you search across everything within them at once.



**INNOVATION  
SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

# what is an inverted index

Inverted index		
<b>Document 1:</b>	space:	1, 2
Space: The final frontier. These are the voyages...	the:	1, 2
	final:	1
	frontier:	1
<b>Document 2:</b>	he:	2
He's bad, he's number one. He's the space cowboy with the laser gun!	bad:	2
	...	

# It's not quite that simple.

TF-IDF means Term Frequency \* Inverse DocumentFrequency

Term Frequency is how often a term appears in a given document

Document Frequency is how often a term appears in all documents

Term Frequency / Document Frequency measures the relevance  
of a term in a document

# using indices



## RESTful API

Elasticsearch fundamentally works via HTTP requests and JSON data. Any language or tool that can handle HTTP can use Elasticsearch.



## client API's

Most languages have specialized Elasticsearch libraries to make it even easier.



## analytic tools

Web-based graphical UI's such as Kibana let you interact with your indices and explore them without writing code.



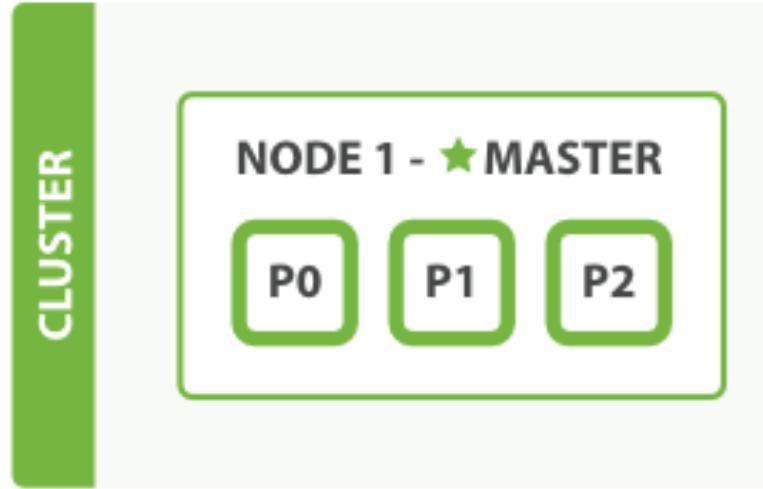
**INNOVATION  
SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

# near real time (NRT)

- Elasticsearch is a **near real time search** platform
- there is a slight **latency** (normally one second) from the time you index a document until the time it becomes searchable

└ sandbox cluster ┘

# elasticsearch cluster



# getting started

- when you start an instance of Elasticsearch, you are starting a node
- If you are running a single instance of Elasticsearch, you have a cluster of one node
- all primary shards reside on the single node
- no replica shards can be allocated, therefore the cluster state remains yellow
- the cluster is fully functional, but is at risk of data loss in the event of a failure

# node type basics

# Elasticsearch node types

## Master Eligible

Nodes only responsible for lightweight cluster-wide actions such as creating or deleting an index, tracking which nodes are part of the cluster, and deciding which shards to allocate to which nodes.

## Machine Learning

Run ML jobs on these nodes.

Requires X-Pack

## Data

Data nodes hold the shards that contain the documents we have indexed. Data nodes handle data related operations like CRUD, search, and aggregations.

## Ingest

Ingest nodes are dedicated to the execution of data pre-processing pipelines, composed of one or more ingest processors.

## Alerting

Nodes that run alerting jobs.

## Coordinating

Nodes that route requests, handle the search reduce phase, and distribute bulk indexing.

All nodes function as coordinating nodes

# Elasticsearch node settings

## Master Eligible

node.master: true

node.data: false

node.ingest: false

## Data

node.master: false

node.data: true

node.ingest: false

## Ingest

node.master: false

node.data: false

node.ingest: true

## Coordinating

node.master: false

node.data: false

node.ingest: false

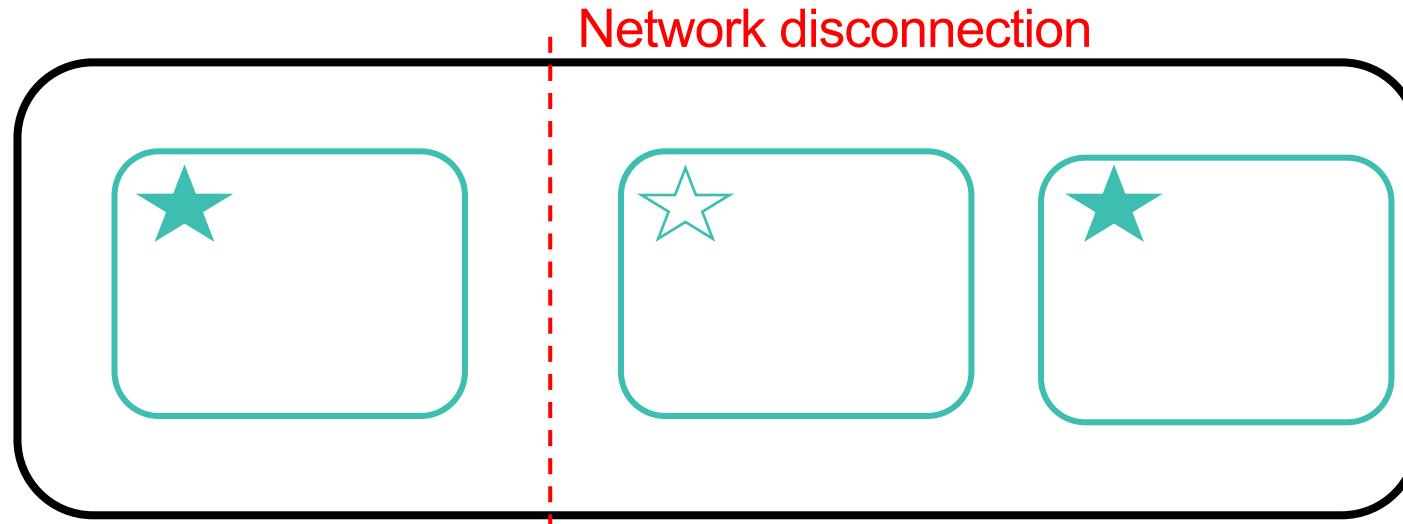


**INNOVATION  
SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

# Architecture

# split brain

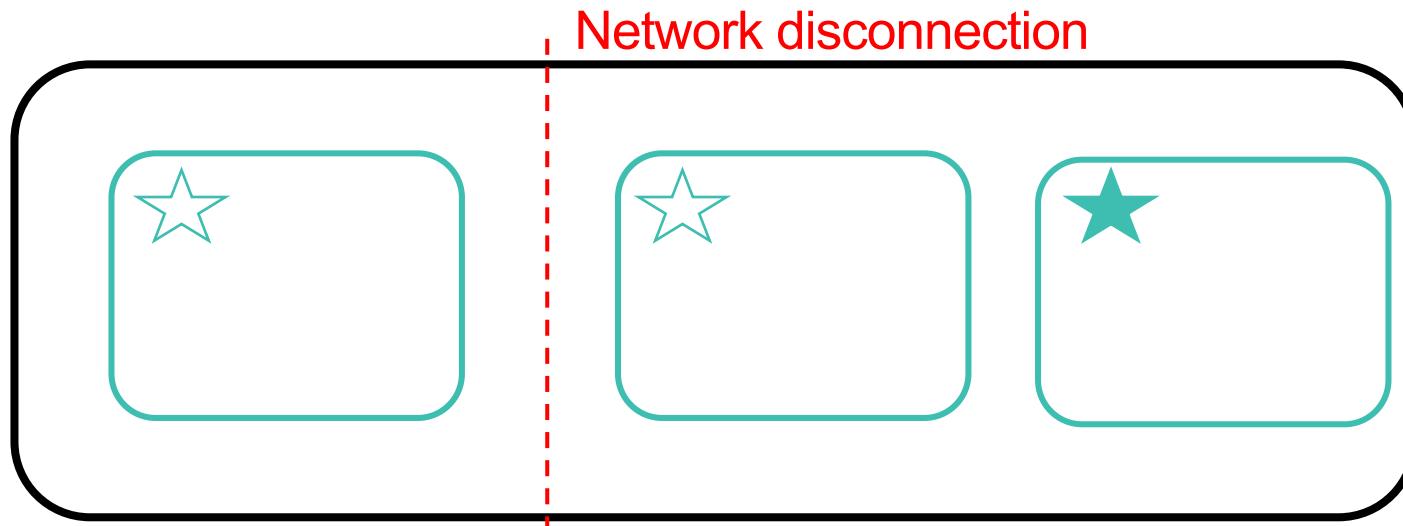
Cluster with 3 master eligible nodes.



If network is partitioned nodes try to elect themselves master, which leads to two masters.

# avoiding split brain

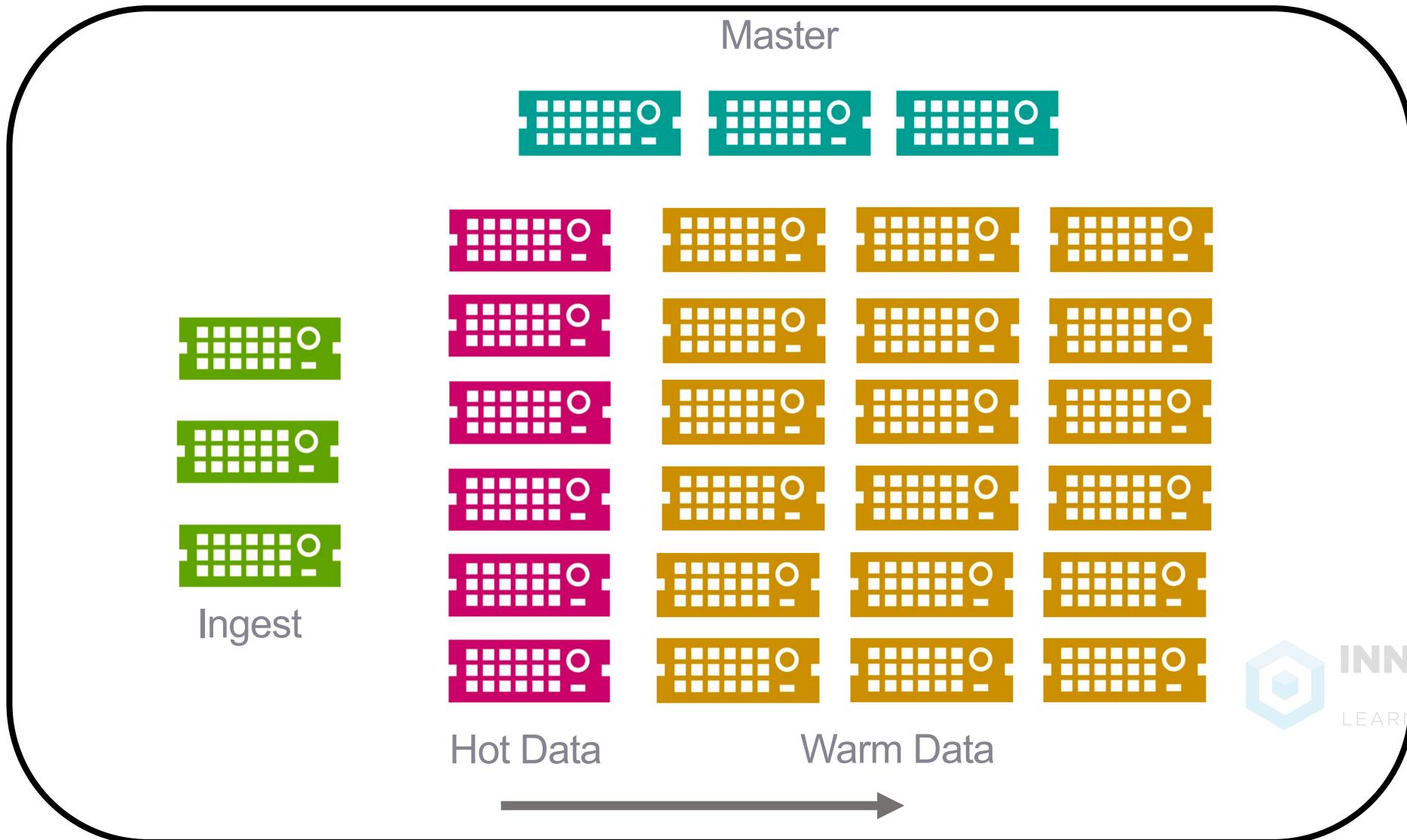
Cluster with 3 master eligible nodes.



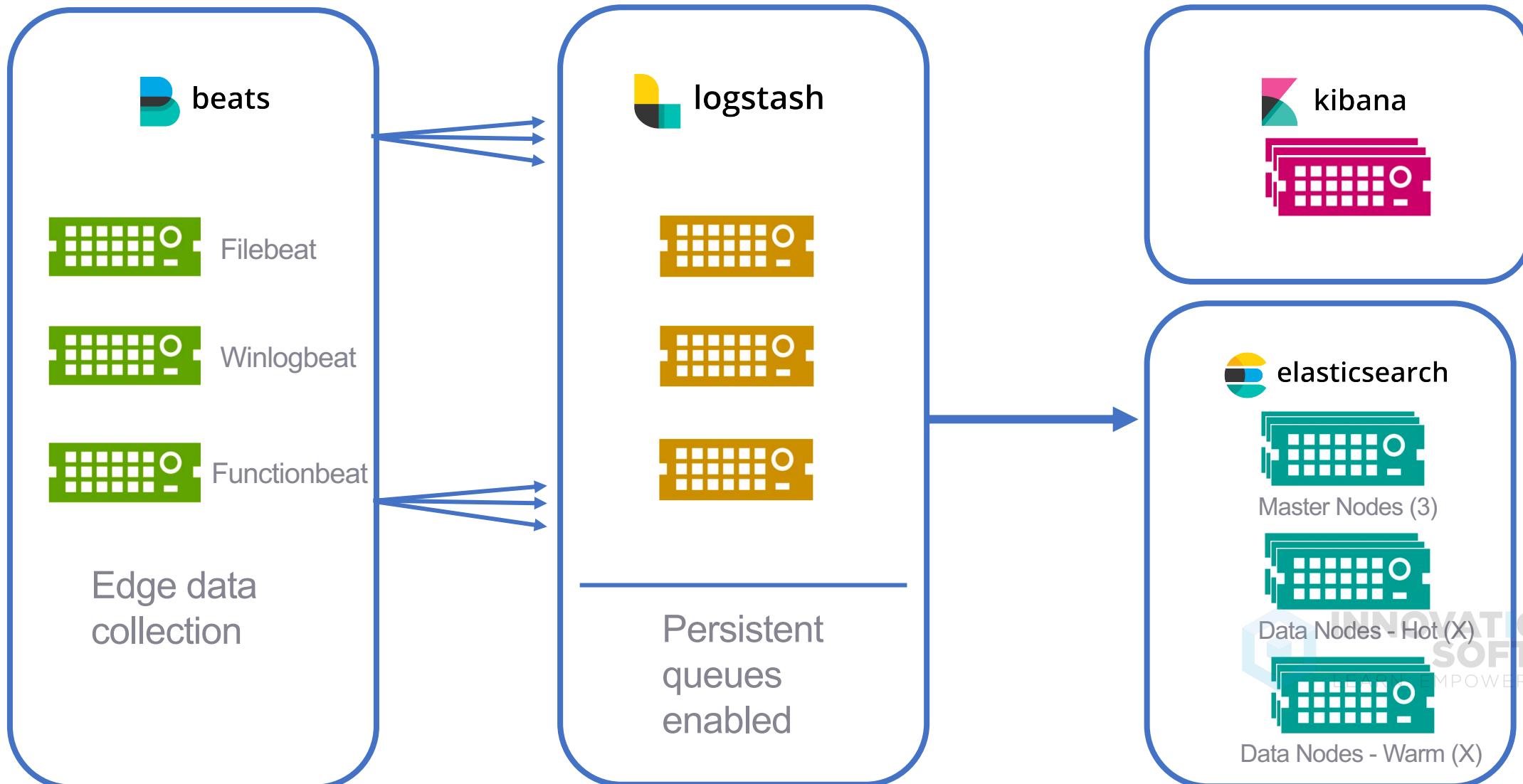
Master eligible node needs at least `minimum\_master\_nodes` votes to win election.

For 3 nodes recommendation is `minimum\_master\_nodes=2`

# Elasticsearch large cluster



# Elasticsearch cluster



# REST: a quick intro.

# Anatomy of a HTTP request

METHOD: the “verb” of the request. GET, POST, PUT, or DELETE

PROTOCOL: what flavor of HTTP (HTTP/1.1)

HOST: what web server you want to talk to

URL: what resource is being requested

BODY: extra data needed by the server

HEADERS: user-agent, content-type, etc.

# example: GET request for google.com

GET /index.html

Protocol: HTTP/1.1

Host: [www.google.com](http://www.google.com)

No body

Headers:

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.1.5) Gecko/20091102 Firefox/3.5.5 (.NET CLR 3.5.30729)

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8

Accept-Language: en-us,en;q=0.5

Accept-Encoding: gzip,deflate

Accept-Charset: ISO-8859-1,utf-8;q=0.7,\*;q=0.7

Keep-Alive: 300

Connection: keep-alive

Cookie: PHPSESSID=r2t5uvjq435r4q7ib3vtdjq120

Pragma: no-cache

Cache-Control: no-cache

# RESTful API's

pragmatic definition: using HTTP requests to communicate with web services

examples:

GET requests retrieve information (like search results)

PUT requests insert or replace new information

DELETE requests delete information

# REST fancy-speak

## Representational State Transfer

Six guiding constraints:

- client-server architecture
- statelessness
- cacheability
- layered system
- code on demand (ie, sending Javascript)
- uniform interface

# why REST?

- language and system independent
- highly scalable

# the curl command

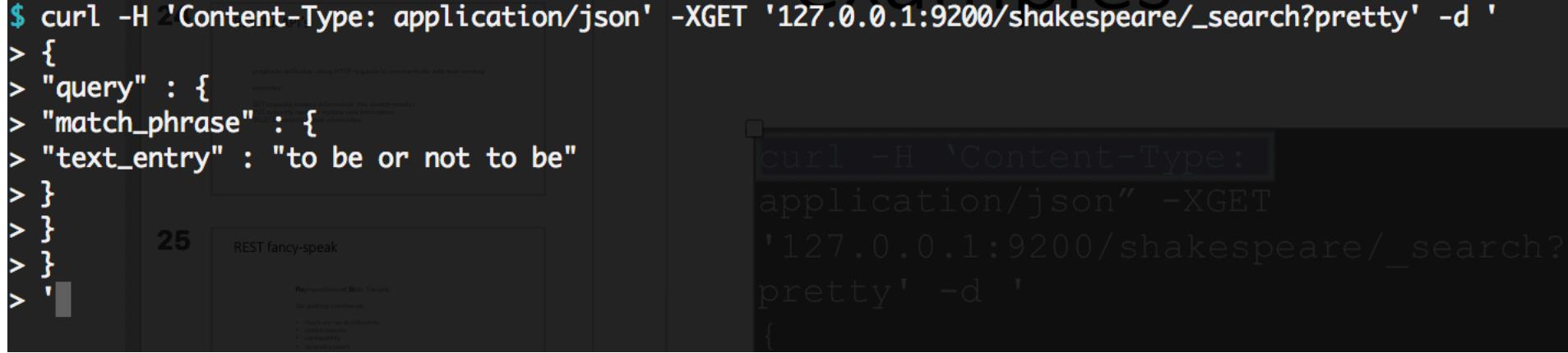
A way to issue HTTP requests from the command line

From code, you'll use whatever library you use for HTTP / REST in the same way.

```
curl -H "Content-Type: application/json" <URL>-d '<BODY>'
```

# examples

```
$ curl -H 'Content-Type: application/json' -XGET '127.0.0.1:9200/shakespeare/_search?pretty' -d '  
> {  
>   "query" : {  
>     "match_phrase" : {  
>       "text_entry" : "to be or not to be"  
>     }  
>   }  
> }' 25 REST fancy-speak  
Representational State Transfer  
The goal of REST:  
• open server architecture  
• client-server  
• scalability
```



```
curl -H 'Content-Type:  
application/json' -XGET  
'127.0.0.1:9200/shakespeare/_search?  
pretty' -d '  
{
```

# examples

```
{  
    "took" : 59,  
    "timed_out" : false,  
    "_shards" : {  
        "total" : 5,  
        "successful" : 5,  
        "skipped" : 0,  
        "failed" : 0  
    },  
    "hits" : {  
        "total" : 1,  
        "max_score" : 13.874454,  
        "hits" : [  
            {  
                "_index" : "shakespeare",  
                "_type" : "doc",  
                "_id" : "34229",  
                "_score" : 13.874454,  
                "_source" : {  
                    "type" : "line",  
                    "line_id" : 34230,  
                    "play_name" : "Hamlet",  
                    "speech_number" : 19,  
                    "line_number" : "3.1.64",  
                    "speaker" : "HAMLET",  
                    "text_entry" : "To be, or not to be: that is the question:"  
                }  
            }  
        ]  
    }  
}
```

why REST?  
the curl command  
Sundog Learning Academy  
In order to issue HTTP requests from the command line, you can use an easy-to-use curl command. Run curl -X GET http://localhost:9200/\_search?pretty to see the raw JSON response from Elasticsearch.

Click to add notes

redis-slave:  
image: gcr.io/google\_samples/

Slide 13 of 278 English (United States)



INNOVATION  
SOFTWARE

LEARN. EMPOWER. INNOVATE.

# the httpie command

A way to issue HTTP requests from the command line  
Simpler syntax than curl, defaults to ‘pretty’ output.

```
http <VERB> <URL>-d‘<BODY>’
```

# examples

```
$ http GET localhost:9200/shakespeare/_search <<<'{
> "query" : {
>   "match_phrase" : {
>     "text_entry" : "to be or not to be"
>   }
> }
> }
```

# examples

```
HTTP/1.1 200 OK
content-encoding: gzip
content-length: 280
content-type: application/json; charset=UTF-8

{
  "_shards": {
    "failed": 0,
    "skipped": 0,
    "successful": 5,
    "total": 5
  },
  "hits": {
    "hits": [
      {
        "_id": "34229",
        "_index": "shakespeare",
        "_score": 13.874454,
        "_source": {
          "line_id": 34230,
          "line_number": "3.1.64",
          "play_name": "Hamlet",
          "speaker": "HAMLET",
          "speech_number": 19,
          "text_entry": "To be, or not to be: that is the question:",
          "type": "line"
        },
        "_type": "doc"
      }
    ],
    "max_score": 13.874454,
    "total": 1
  },
  "timed_out": false,
  "took": 16
}
```



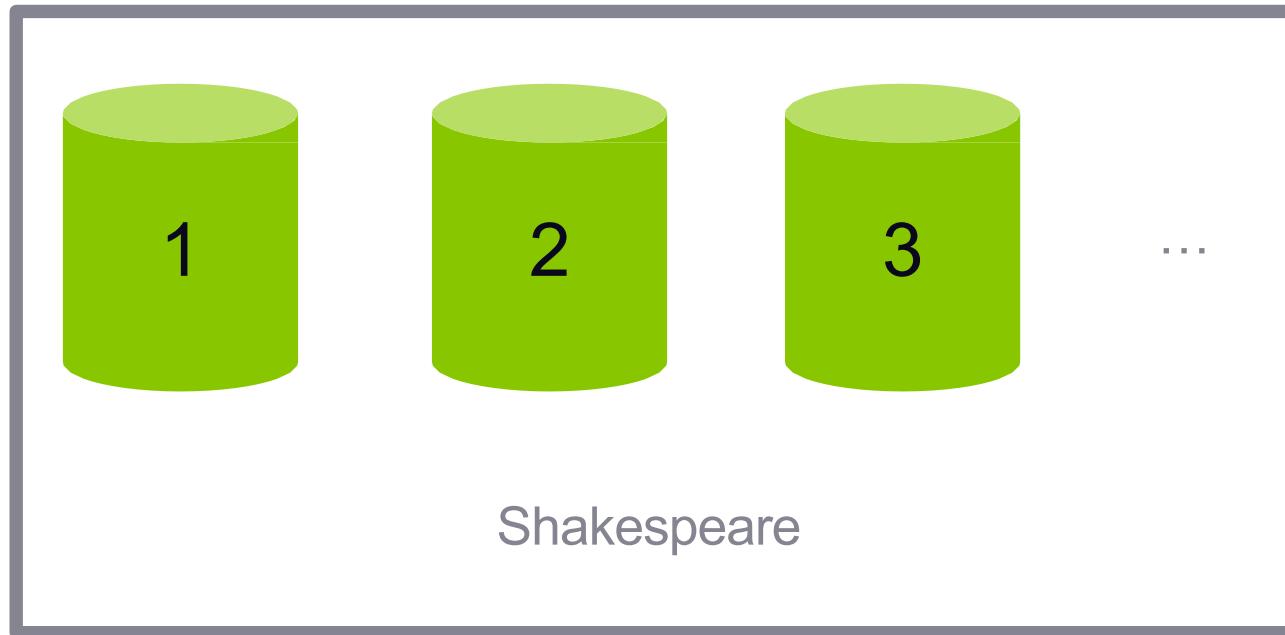
# lab01: lab setup

- Confirm access to VM
- Install Elasticsearch
- Upload some data

# how elasticsearch scales

# an index is split into shards

Documents are **hashed** to a particular **shard**.

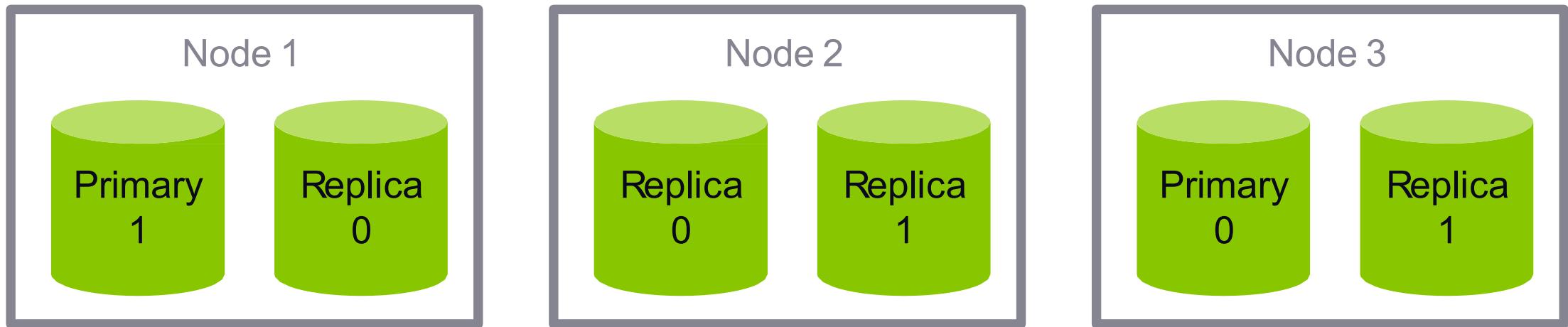


Each shard may be on a different **node** in a **cluster**.  
Every shard is a self-contained Lucene index of its own.

# primary and replica shards

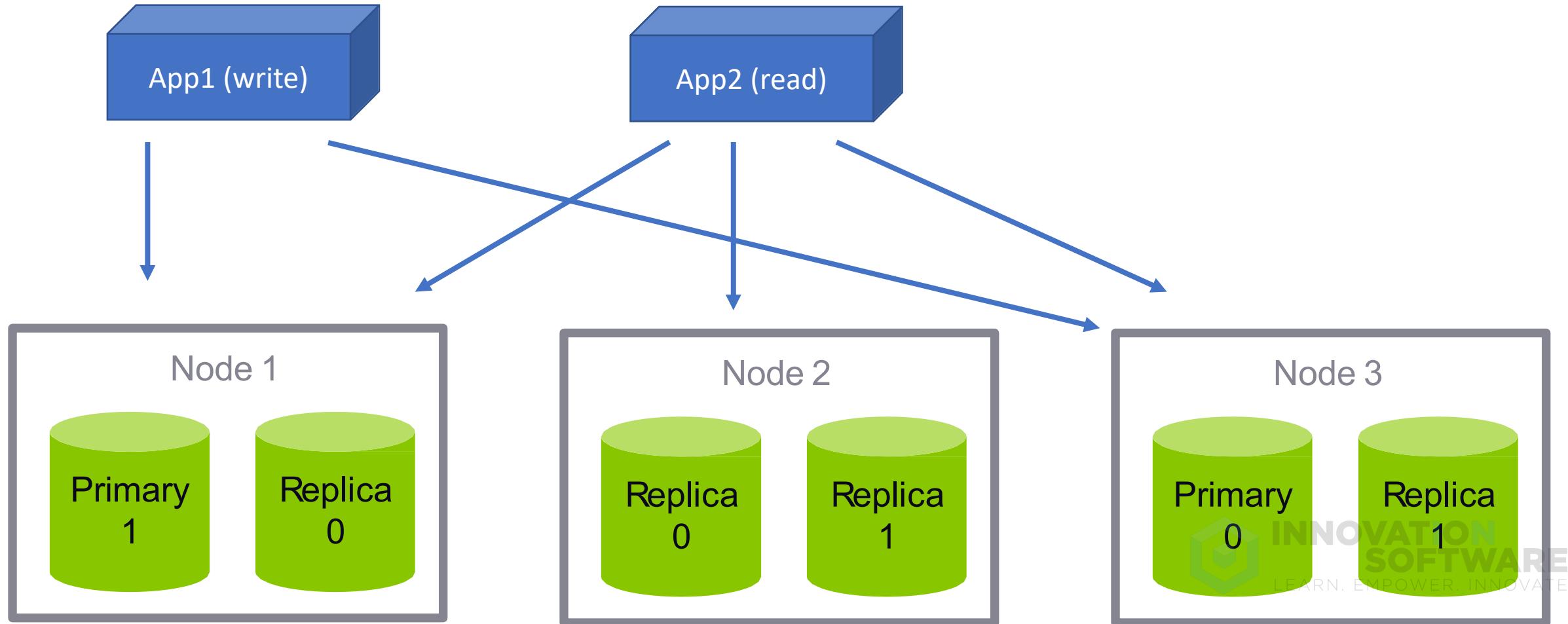
This **index** has two **primary shards** and a **replication factor of 2**.

Your application should round-robin requests among nodes.



**Write** requests are routed to the primary shard, then replicated  
**Read** requests are routed to the primary or any replica

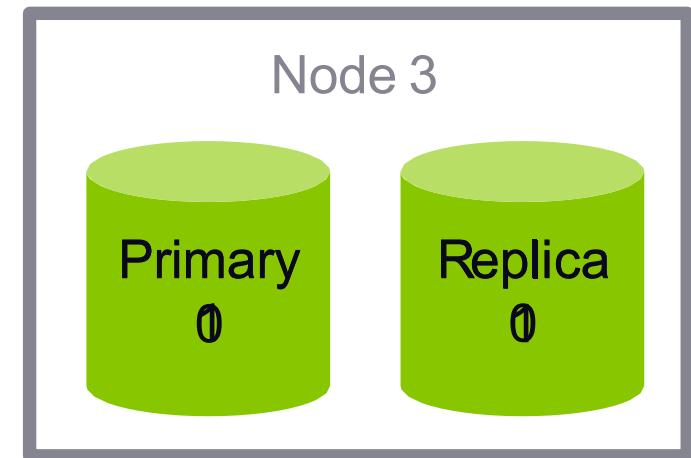
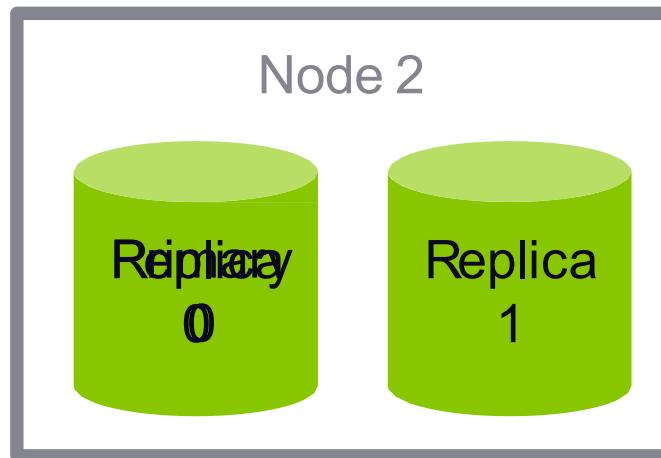
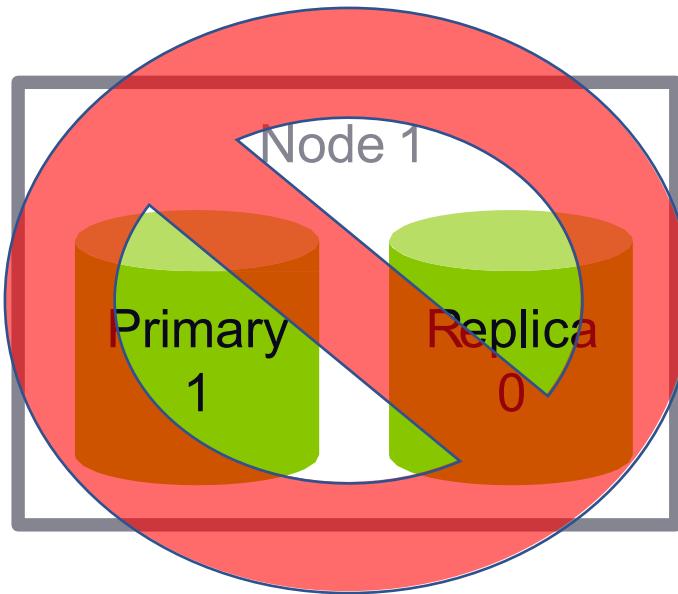
# primary and replica shards



# primary and replica shards

This **index** has two **primary shards** and two **replicas**.

Your application should round-robin requests among nodes.



**Write** requests are routed to the primary shard, then replicated

**Read** requests are routed to the primary or any replica

# The number of primary shards cannot be changed later.

Not as bad as it sounds – you can add **more replica shards** for more read throughput.

Worst case you can **re-index** your data.

The number of shards can be set up front via a PUT command via **REST** / HTTP

```
PUT /testindex
{
  "settings": { "number_of_shards": 3
    , "number_of_replicas": 1
  }
}
```

# number of shards quiz

How many total shards does the code on the right create?

```
PUT /testindex
{
  "settings": { "number_of_shards": 3
    , "number_of_replicas": 1
  }
}
```

# primary and replica shards

This index has 3 primary shards and 3 replicas.



「quiz time」

The schema for  
your documents  
are defined by...

- The index
- The type
- The document itself



INNOVATION  
SOFTWARE  
LEARN. EMPOWER. INNOVATE.

01

The schema for  
your documents  
are defined by...

- The index
- The type
- The document itself



INNOVATION  
SOFTWARE  
LEARN. EMPOWER. INNOVATE.

01

# 02

What purpose do inverted indices serve?

- They allow you to search phrases in reverse order
- They quickly map search terms to documents
- They load balance search requests across your cluster

# 02

What purpose do inverted indices serve?

- They allow you search phrases in reverse order
- They quickly map search terms to documents
- They load balance search requests across your cluster

# 03

- 8
- 15
- 20

An index configured for 5 primary shards and 3 replicas would have how many shards in total?



INNOVATION  
SOFTWARE  
LEARN. EMPOWER. INNOVATE.

# 03

- 8
- 15
- 20

An index configured for 5 primary shards and 3 replicas would have how many shards in total?



INNOVATION  
SOFTWARE  
LEARN. EMPOWER. INNOVATE.

# 04

- true
- false

Elasticsearch is built  
only for full-text search  
of documents.



LEARN. EMPOWER. INNOVATE.

# 04

- true
- false

Elasticsearch is built  
only for full-text search  
of documents.



LEARN. EMPOWER. INNOVATE.

# examining movielens

# movielens

**movielens** is a free dataset of movie ratings gathered from movielens.org.

It contains user ratings, movie metadata, and user metadata.

Let's download and examine the data files from movielens.org



# lab02: download MovieLens data

- Visit GroupLens website
- Download small data set
- Look through sample data files

Instructor led lab

# creating mappings

# what is a mapping?

a mapping is a **schema definition**.

elasticsearch has reasonable defaults, but sometimes you need to customize them.

```
curl -H "Content-Type: application/json" -XPUT 127.0.0.1:9200/movies -d '  
{'  
  "mappings": {  
    "movie": {  
      "properties" : {  
        "year" : {"type": "date"}  
      }  
    }  
  }'  
'
```

**NOTE: Content-Type is now required in Elasticsearch 6.x+**

# elasticsearch 5 syntax

In Elasticsearch 5 it was possible to send a REST request without the Content-Type. Elasticsearch would then “sniff” the content and set the type based on that.

```
curl 'http://localhost:9200/_search' -d'  
{  
  "query" : {  
    "match_all" : {}  
  }  
}'
```

# elasticsearch syntax

There are two reasons this changed.

## Clarity

Sending plain text content to API  
that doesn't support it returns.

Content-Type header [text/plain] is not supported

In Elasticsearch 5 if you sent plain text that started with a curly brace and the letter “a” it would assume it was JSON, but when it tried to parse that, it would fail and the error message would look more like:

Unexpected character ('a' (code 97)) : was expecting  
double-quote to start field name

# elasticsearch syntax

There are two reasons this changed.

## Security

```
<html>
  <body>
    <script src="https://code.jquery.com/jquery-3.2.1.min.js"
           type="text/javascript"></script>
    <script type="text/javascript">
      $(function() {
        $.ajax({
          url: "http://localhost:9200/visitors/doc/",
          type: 'POST',
          data: JSON.stringify({ browser: navigator.userAgent,
                                date: new Date() }),
          contentType: 'text/plain'
        });
      });
    </script>
  </body>
</html>
```

- JSON sent as text/plain
- Cross Origin Resource Sharing

# common mappings

## field types

text, keyword, byte, short, integer, long, float, double, boolean, date

```
"properties": {  
    "user_id" : {  
        "type": "long"  
    }  
}
```

## field index

do you want this field to be queryable? true / false

```
"properties": {  
    "genre" : {  
        "index": "false"  
    }  
}
```

## field analyzer

define your tokenizer and token filter. standard / whitespace / simple / english etc.

```
"properties": {  
    "description" : {  
        "analyzer": "english"  
    }  
}
```



**INNOVATION  
SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

# more about analyzers

## character filters

remove HTML encoding, convert & to and

## tokenizer

split strings on whitespace / punctuation / non-letters

## token filter

lowercasing, stemming, synonyms, stopwords

# choices for analyzers

## standard

splits on word boundaries, removes punctuation, lowercases. good choice if language is unknown

## simple

splits on anything that isn't a letter, and lowercases

## whitespace

splits on whitespace but doesn't lowercase

## language (i.e. english)

accounts for language-specific stopwords and stemming



**INNOVATION  
SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

# lab03: create year mapping

- Log into VM
- Use curl to create mapping for year field.
- Confirm it was created successfully

\* Instructor led lab

# [ hacking curl ]

# make life easier

From your home directory:

Two ways

Alias

```
alias curl="/usr/bin/curl -H 'Content-type: application/json'"
```

Or wrap the command

```
mkdir bin
```

```
cd bin
```

```
vi curl (Hit I for insert mode)
```

```
#!/bin/bash
/usr/bin/curl -H "Content-Type: application/json" "$@"
```

Esc – wq! – enter

```
chmod a+x curl
```

# make life easier

Delete an index

```
curl -XDELETE 127.0.0.1:9200/movies
```

# remember



Without this hack, you need to add

`-H "Content-Type: application/json"`

to every curl command!

The rest of the course assumes you have this in place.

import  
**one document**



**INNOVATION**  
**SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

# insert

```
curl -XPUT 127.0.0.1:9200/movies/movie/109487 -d '  
{  
  "genre" : ["IMAX","Sci-Fi"],  "title" : "Interstellar",  "year" : 2014  
}'
```



**INNOVATION**  
**SOFTWARE**  
LEARN. EMPOWER. INNOVATE.



UNLEASHED



INNOVATION  
SOFTWARE  
IMPRESSIVE. INNOVATE.

import  
many  
documents

# json bulk import

```
curl -XPUT 127.0.0.1:9200/_bulk -d '
```

```
{ "create" : { "_index" : "movies", "_type" : "movie", "_id" : "135569" } }
{ "id": "135569", "title" : "Star Trek Beyond", "year":2016 , "genre":["Action", "Adventure", "Sci-Fi"] }
{ "create" : { "_index" : "movies", "_type" : "movie", "_id" : "122886" } }
{ "id": "122886", "title" : "Star Wars: Episode VII - The Force Awakens", "year":2015 , "genre":["Action", "Adventure", "Fantasy", "Sci-Fi", "IMAX"] }
{ "create" : { "_index" : "movies", "_type" : "movie", "_id" : "109487" } }
{ "id": "109487", "title" : "Interstellar", "year":2014 , "genre":["Sci-Fi", "IMAX"] }
{ "create" : { "_index" : "movies", "_type" : "movie", "_id" : "58559" } }
{ "id": "58559", "title" : "Dark Knight, The", "year":2008 , "genre":["Action", "Crime", "Drama", "IMAX"] }
{ "create" : { "_index" : "movies", "_type" : "movie", "_id" : "1924" } }
{ "id": "1924", "title" : "Plan 9 from Outer Space", "year":1959 , "genre":["Horror", "Sci-Fi"] } \
```

# updating documents

# versions

Every document has a `_version` field

Elasticsearch documents are immutable.

When you update an existing document:

a new document is created with an incremented `_version`

the old document is marked for deletion

# partial update api

Lab:

- Look at document for Interstellar
- Run curl command to output Interstellar document data

```
curl -XGET 127.0.0.1:9200/movies/movie/109487?pretty
```

```
{
  "_index": "movies",
  "_type": "movie",
  "_id": "109487",
  "_version": 1,
  "found": true,
  "_source": {
    "id": "109487",
    "title": "Interstellar",
    "year": 2014,
    "genre": [
      "Sci-Fi",
      "IMAX"
    ]
  }
}
```



INNOVATION  
SOFTWARE  
LEARN. EMPOWER. INNOVATE.

# partial update api

```
curl -XPOST 127.0.0.1:9200/movies/movie/109487/_update -d '  
{'  
  "doc": {  
    "title": "Outerstellar"  
  }  
}'
```

Send data to REST API using `POST` verb

Update title for movie with id 109487

New version of document created

Old version deleted (eventually)

# █ deleting documents █

it couldn't be easier.

Just use the DELETEmethod:

```
curl -XDELETE 127.0.0.1:9200/movies/movie/12345
```



INNOVATION  
SOFTWARE  
LEARN. EMPOWER. INNOVATE.

# lab: delete document

Now let's delete the Dark Knight

First: Find out movie ID

```
curl -s -XGET 127.0.0.1:9200/movies/_search?q=Dark
```

Second: Delete it!

```
curl -XDELETE 127.0.0.1:9200/movies/movie/58559?pretty
```

Third: Confirm it was deleted

```
curl -s -XGET 127.0.0.1:9200/movies/_search?q=Dark
```



INNOVATION  
SOFTWARE  
LEARN. EMPOWER. INNOVATE.

# elasticsearch

## exercise

insert, update, and then delete a movie of your choice into the movies index!



INNOVATION  
SOFTWARE  
LEARN. EMPOWER. INNOVATE.

└ dealing with  
concurrency ┘

# the problem



But it should be 12!

# optimistic concurrency control



Use `retry_on_conflicts=Y` to automatically retry.

# labs

- Lab4: Import Movie data
- Lab 5: Update Documents
- Lab6: Versions & Conflict resolution

# controlling **full-text search**

# using analyzers

sometimes text fields should be exact-match

- use **keyword** mapping type to suppress analyzing (exact match only)
- Use **text** type to allow analyzing

search on analyzed fields will return anything remotely relevant

- depending on the analyzer, results will be case-insensitive, stemmed, stopwords removed, synonyms applied, etc.
- searches with multiple terms need not match them all

# Changing mappings

- Can not change mapping on existing index
- Have to delete index and start over

```
curl -XDELETE 127.0.0.1:9200/movies
```

- New mapping of "keyword" for "genre"
- New analyzer of "english" for "title"

```
curl -XPUT 127.0.0.1:9200/movies -d '  
{  
  "mappings": {  
    "movie": {  
      "properties": {  
        "id": {"type": "integer"},  
        "year": {"type": "date"},  
        "genre": {"type": "keyword"},  
        "title": {"type": "text", "analyzer": "english"}  
      }  
    }  
  }'
```



I N

LEARN. EMPOWER. INNOVATE.

# data modeling

# strategies for relational data

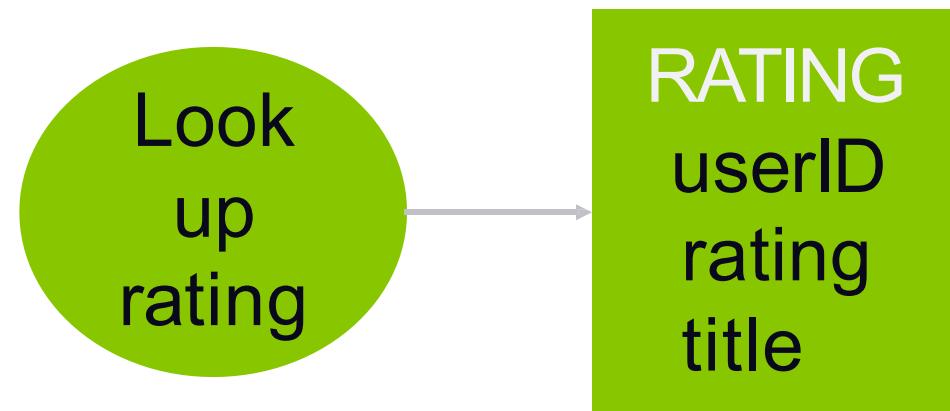
normalized data



Minimizes storage space, makes it easy to change titles  
But requires two queries, and storage is cheap!

# strategies for relational data

denormalized data



titles are duplicated, but only one query

# strategies for relational data

Parent / Child Relationship

Star Wars

A New Hope

Empire  
Strikes Back

Return of the  
Jedi

The Force  
Awakens



INNOVATION  
SOFTWARE  
LEARN. EMPOWER. INNOVATE.

# strategies for relational data

Series index created with mapping for relationships

Star Wars

```
curl -XPUT 127.0.0.1:9200/series -d '  
{'  
  "mappings" : {  
    "movie": {  
      "properties": {  
        "film_to_franchise": {"type": "join", ""relations": {"franchise" : "film"}}  
      }  
    }  
  }'  
'
```

join type field

Relationship: Parent/Child

# strategies for relational data

- ES6: Parents/Children all in same shard
- Forcing everything to be indexed to shard 1.
- Use relationships minimally.

```
{ "create" : { "_index" : "series", "_type" : "movie", "_id" : "1", "routing" : 1 }

{ "id": "1", "film_to_franchise": {"name": "franchise"}, "title" : "Star Wars" }

{ "create" : { "_index" : "series", "_type" : "movie", "_id" : "260", "routing" : 1 }

{ "id": "260", "film_to_franchise": {"name": "film", "parent": "1"}, "title" : "Star Wars: Episode IV – A New Hope", "year": "1977" , "genre": ["Action", "Adventure", "Sci-Fi"] }
```

- Parent created: Franchise
- Fields for child created

# lab: Analyzers & Relational data

- Lab 7: Create new index mapping
- Lab 8: Create index for relational data

query-lite  
search

# “query lite”

```
curl -XGET 127.0.0.1:9200/movies/movie/_search?pretty -d '  
{  
  "query": {  
    "match": {  
      "title": "Star Trek"  
    }  
  }  
}'
```

Proper JSON query

Query lite

```
curl -XGET 127.0.0.1:9200/movies/movie/_search?q=title:star
```

```
curl -XGET 127.0.0.1:9200/movies/movie/_search?q=+year:>2010+title:trek
```



LEARN. EMPOWER. INNOVATE.

# it's not always simpler.

spaces etc. need to be URLencoded.

```
curl -XGET 127.0.0.1:9200/movies/movie/_search?q=-+year:>2010+title:trek
```



```
curl -XGET 127.0.0.1:9200/movies/movie/_search?q=%2B-yea%3A%3E2010+%2Btitle%3Atrek
```

and it can be  
dangerous.

- **cryptic** and tough to debug
- can be a **security issue** if exposed to end users
- **fragile** – one wrong character and you're hosed.

But it's handy for quick experimenting.

## Parameters



The parameters allowed in the URI are:

Name	Description
<code>q</code>	The query string (maps to the <code>query_string</code> query, see <a href="#">Query String Query</a> for more details).
<code>df</code>	The default field to use when no field prefix is defined within the query.
<code>analyzer</code>	The analyzer name to be used when analyzing the query string.
<code>analyzeWildcard</code>	Should wildcard and prefix queries be analyzed or not. Defaults to <code>false</code> .
<code>batchedReduceSize</code>	The number of shard results that should be reduced at once on the coordinating node. This value should be used as a protection mechanism to reduce the memory overhead per search request if the potential number of shards in the request can be large.
<code>defaultOperator</code>	The default operator to be used, can be <code>AND</code> or <code>OR</code> . Defaults to <code>OR</code> .
<code>lenient</code>	If set to true will cause format based failures (like providing text to a numeric field) to be ignored. Defaults to false.
<code>explain</code>	For each hit, contain an explanation of how scoring of the hits was computed.
<code>_source</code>	Set to <code>false</code> to disable retrieval of the <code>_source</code> field. You can also retrieve part of the document by using <code>_source_include</code> & <code>_source_exclude</code> (see the <a href="#">request body</a> documentation for more details)
<code>storedFields</code>	The selective stored fields of the document to return for each hit, comma delimited. Not specifying any value will cause no fields to return.

learn more.

this is formally called “URI Search”. Search for that on the Elasticsearch documentation.

it’s really quite powerful, but again is only appropriate for quick “curl tests”.

request body  
search

# request body search

how you're supposed to do it

query DSL is in the request body as JSON  
(yes, a GET request can have a body!)

```
curl -XGET 127.0.0.1:9200/movies/movie/_search?pretty -d '  
{'  
    "query": {  
        "match": {  
            "title": "star"  
        }  
    }  
}'
```

# queries and filters

**filters** ask a yes/no question of your data  
**queries** return data in terms of relevance

use filters when you can – they are faster and cacheable.

# example: boolean query with a filter

```
curl -XGET 127.0.0.1:9200/movies/movie/_search?pretty -d'  
{  
  "query":{  
    "bool": {  
      "must": {"term": {"title": "trek"}},  
      "filter": {"range": {"year": {"gte": 2010}}}  
    }  
  }  
}'
```

boolean (combines multiple searches)

must (search for title, only return results newer or equal to 2010)



INNOVATION  
SOFTWARE  
LEARN. EMPOWER. INNOVATE.

# some types of filters

**term:** filter by exact values

```
{"term": {"year": 2014}}
```

**terms:** match if any exact values in a list match

```
{"terms": {"genre": ["Sci-Fi", "Adventure"]}}
```

**range:** Find numbers or dates in a given range (gt, gte, lt, lte)

```
{"range": {"year": {"gte": 2010}}}
```

**exists:** Find documents where a field exists

```
{"exists": {"field": "tags"}}
```

**missing:** Find documents where a field is missing

```
{"missing": {"field": "tags"}}
```

**bool:** Combine filters with Boolean logic (must, must\_not, should)

# some types of queries

**match\_all:** returns all documents and is the default. Normally used with a filter.

```
{"match_all": {}}
```

**match:** searches analyzed results, such as full text search.

```
{"match": {"title": "star"}}
```

**multi\_match:** run the same query on multiple fields.

```
{"multi_match": {"query": "star", "fields": ["title", "synopsis"]}}
```

**bool:** Works like a bool filter, but results are scored by relevance.

# boosting results

When searching across multiple fields, we may want to boost the scores in a certain field. In the example below, we boost scores from the summary field by a factor of 3 in order to increase the importance of the summary field.

```
POST /shakespeare/_search
{
  "query": {
    "multi_match" : {
      "query" : "lear",
      "fields": ["title", "summary^3"]
    }
  },
  "_source": ["title", "summary", "publish_date"]
}
```

# syntax reminder

queries are wrapped in a “query”: { } block,  
filters are wrapped in a “filter”: { } block.

you can combine filters inside queries, or queries inside filters too.

```
curl -XGET 127.0.0.1:9200/movies/movie/_search?pretty -d'
{
  "query":{
    "bool": {
      "must": {"term": {"title": "trek"}},
      "filter": {"range": {"year": {"gte": 2010}}}
    }
  }
}'
```



phrase  
search

# phrase matching

must find all terms, in the right order.

```
curl -XGET 127.0.0.1:9200/movies/movie/_search?pretty -d'  
{  
  "query":{  
    "match_phrase": {  
      "title": "star wars"  
    }  
  }  
}'
```

# slop

order matters, but you're OK with some words being in between the terms:

```
curl -XGET 127.0.0.1:9200/movies/movie/_search?pretty -d '  
{'  
  "query": {  
    "match_phrase": {  
      "title": {"query": "star beyond", "slop": 1}  
    }  
  }  
}'
```

the **slop** represents how far you're willing to let a term move to satisfy a phrase (in either direction!)

another example: “quick brown fox” would match “quick fox” with a slop of 1.

# proximity queries

remember this is a query – results are sorted by relevance.

just use a really high slop if you want to get any documents that contain the words in your phrase, but want documents that have the words closer together scored higher.

```
curl -XGET 127.0.0.1:9200/movies/movie/_search?pretty -d '  
 {  
   "query": {  
     "match_phrase": {  
       "title": {"query": "star beyond", "slop": 100}  
     }  
   }  
 }'
```

# lab: Phrase & Slop

- Lab 9: URI, JSON, Phrase and Slop searches

└ pagination ┘

# specify “from” and “size”

result 1  
result 2  
result 3  
result 4  
result 5  
result 6  
result 7  
result 8



from = 0, size=3



from = 3, size=3



**INNOVATION**  
**SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

# pagination syntax

## URI Search

```
curl -XGET '127.0.0.1:9200/movies/movie/_search?size=2&from=2&pretty'
```

## JSON body

```
curl -XGET 127.0.0.1:9200/movies/movie/_search?pretty -d '
{
  "from": 2,
  "size": 2,
  "query": {"match": {"genre": "Sci-Fi"}}
}'
```



# beware

deep pagination can **kill performance**.

every result must be **retrieved, collected, and sorted**.

enforce an **upper bound** on how many results you'll return to users.

sort

sorting your results is usually quite simple.

```
curl -XGET '127.0.0.1:9200/movies/movie/_search?sort=year&pretty'
```

# unless you're dealing with strings.

A **text** field that is **analyzed** for full-text search can't be used to sort documents

This is because it exists in the inverted index as individual terms, not as the entire string.

# unanalyzed copy using the keyword type.

To sort analyzed field you must make a copy using keyword type and sort by that.

```
curl -XPUT 127.0.0.1:9200/movies/ -d '  
{  
  "mappings": {  
    "movie": { "properties" : {  
      "title": {  
        "type" : "text", "fields": {  
          "raw": {  
            "type": "keyword",  
          }  
        }  
      }  
    }  
  }  
}'
```



# raw keyword field

Now you can sort on the unanalyzed raw field.

```
curl -XGET '127.0.0.1:9200/movies/movie/_search?sort=title.raw&pretty'
```

sadly, you cannot change the mapping on an existing index.

you'd have to delete it, set up a new mapping, and re-index it.

like the number of shards, this is something you should think about **before** importing data into your index.

more with  
filters

# complex filtered query

Science fiction movies without term "trek" in the title, released between the years of 2010 and 2015

```
curl -XGET 127.0.0.1:9200/movies/_search?pretty -d'
{
  "query": {
    "bool": {
      "must": {"match": {"genre": "Sci-Fi"}},
      "must_not": {"match": {"title": "trek"}},
      "filter": {"range": {"year": {"gte": 2010, "lt": 2015}}}
    }
  }
}'
```

# elasticsearch

exercise

search for science fiction movies  
before 1960, sorted by title.



INNOVATION  
SOFTWARE  
LEARN. EMPOWER. INNOVATE.

[ fuzziness ]

# fuzzy matches

a way to account for typos and misspellings

the **levenshtein distance** accounts for:

- **substitutions** of characters (interstellar ->intersteller)
- **insertions** of characters (interstellar ->insterstellar)
- **deletion** of characters (interstellar ->interstelar)

all of the above have an edit distance of **1**.

# the fuzziness parameter

Example of Interstellar being misspelled by 2 characters.

- fuzziness = 2, so we can tolerate 2 errors.

```
curl -XGET 127.0.0.1:9200/movies/movie/_search?pretty -d '  
{  
    "query": {  
        "fuzzy": {  
            "title": {"value": "intrsteller", "fuzziness": 2}  
        }  
    }  
}'
```

# AUTO fuzziness

fuzziness: AUTO

- 0 for 1-2 character strings
- 1 for 3-5 character strings
- 2 for anything else

partial  
matching

# prefix queries on strings

If we remapped `year` field to be a string, we could do a simple query as below.

```
curl -XGET 127.0.0.1:9200/movies/movie/_search?pretty -d '  
{  
  "query": {  
    "prefix": {  
      "year": "201"  
    }  
  }  
}'
```

# wildcard queries

```
curl -XGET 127.0.0.1:9200/movies/movie/_search?pretty -d '  
{'  
  "query": {  
    "wildcard": {  
      "year": "1*"  
    }  
  }'  
}'
```

# regexp queries

```
curl -H 'Content-Type: application/json' -XGET  
'10.0.2.15:9200/shakespeare/_search?pretty' -d '  
{  
  "query" : {  
    "regexp" : {  
      "play_name" : "H[a-z]*t"  
    }  
  },  
  "_source": ["play_name"]  
}  
'
```

clustering

Quantity? How  
much data.

Producers?  
Who is writing  
and what is the  
periodicity

Consumers?  
Apps, Kibana,  
... questions

Velocity? Queries  
per second, minute, ...

key questions

Indices? How  
many?



INNOVATION  
SOFTWARE  
LEARN. EMPOWER. INNOVATE.