Table of Contents

Course Overview

Chapter 0: Overview

Chapter 1: Introduction

Chapter 2: ElasticSearch Index

Chapter 3: Search

Chapter 4: Advanced Search and Mapping

Chapter 5: Logstash

Chapter 6: Filebeat

Chapter 7: Kibana

Chapter 8: Watcher

Chapter 9: Running in a Container

Chapter 10: Preparing for Production

Chapter 11: Running in Production

Chapter 12: Course Summary

Extras

Course: Elastic.co Products

by Vlad Khazin

- Course materials can be accessed online1 or online2
- Course materials can also be downloaded in pdf or pub format from Gitbook
- If you did not have a chance to fill-out pre-course survey, please do so now
- If you have not provided your public ssh key via email to configure access to your private sandbox, please do so now
    - How to generate public/private key
    - Email public key to vladimir.khazin@icssolutions.ca
- Take a moment to register at GitHub.com - you can clone the course repo and make your own comments throughout the course
- Select repository url and then 'fork' link to create a repo for yourself
- Alternatively you are welcome to use any other method to comment and to take notes
- You will be using your own laptop and sandbox virtual machine to setup, to configure and to operate Elastic.co products

Chapter 0: Overview

- Why ElasticSearch
- Intended audience
- Prerequisites
- About Instructor
- About Participants
- Event Logistics

Why ElasticSearch

- ElasticSearch became popular choice for search engine
- Started as full text distributed, scalable database
- Evolved to cover logs processing, graph database, and big data
- Rich ecosystem with in-house and 3rd party plugins
- Open source with enterprise support
- Lucene based
- Impressive scalability and performance
- Rest not Restful API
- Fast version release cycle

Course Objectives

- Get familiar with ElasticSearch as a distributed database and search engine
- Install and configure ElasticSearch 5.x
- Gain understanding and proficiency with NoSql data-modeling principals
- Learn Query Dsl (domain specific language)
- Understand difference between search and filter
- Build aggregation queries
- Run Filebeat to parse log file
- Setup Logstash to transform log lines into json documents
- Install and configure Kibana 5.x
- Leverage Kibana's discover, visualize, and dashboard functionality
- Install and explorer X-Pack
- Review production planning and operational principals

Intended Audience

- Developers interested in NoSql databases and Search Engines
- System and database administrators with solid Rdbms knowledge and little experience with Polyglot persistence

Prerequisites

- Rdbms development and basic administrations experience
- Experience with data modeling principles
- Familiarity with Json
- Recent programming and/or scripting experience
- Exposure to Linux/MacOs shell environment

About Instructor/Author

- Real-world ElasticSearch experience
- Large/Medium Enterprise and start-up environments
- Full-Stack Development
- Variety of Database Environment: Transactional Processing, Data Analytics, Reporting, and Searching
- Sees training approach differently:
    - Objective of the course is to shorten runway to get flying, fast!
    - In nowadays information is readily available using web search engine, hence overloading course materials with lot's of details is not overly helpful
    - Materials are here to provide general guidance with references & links for more details
    - Instructor is to paint a runway to self-sufficiency finding information and dealing with setbacks
    - Labs are to provide a safe way to experiment with the technology and to address common troubles

About Participants

- Your name
- Daily duties and responsibilities
- What you are looking to get out of this event?
- What excites you in your job?

Logistics

- Start at 9:30
- Morning break around 10:45 for 15 min
- Lunch break 12:00 to 13:00
- Afternoon breaks: around 14:30 and 16:00 for 15 min each
- Finish at 17:30
- Materials are available on-line during and after the course
- Materials contain external links
- Exercises are mini-hackathons, not step-by-step instructions
- Questions, open discussions, and comments are encouraged
- Keep it professional and polite
- Exercises are progressively turning from the step-by-step instructions into a general guidance
- Let's have fun!

Chapter 1: Introduction

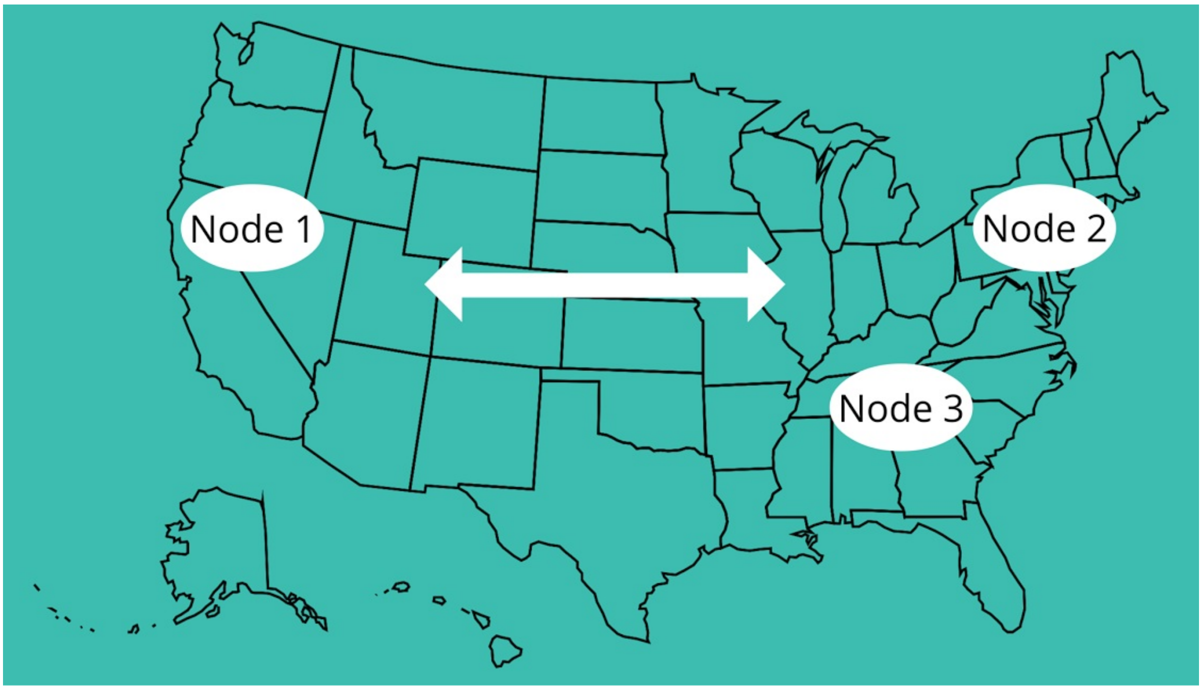
- Terminology, basic concepts, implementation, setup, and basic operations.
- Data modeling with ElasticSearch
- Overview of best practices
- What’s in a distributed database?
- Understanding ElasticSearch cluster, shards, and replicas
- Value of multiple indices, index aliases, and cross-index operations

Terminology

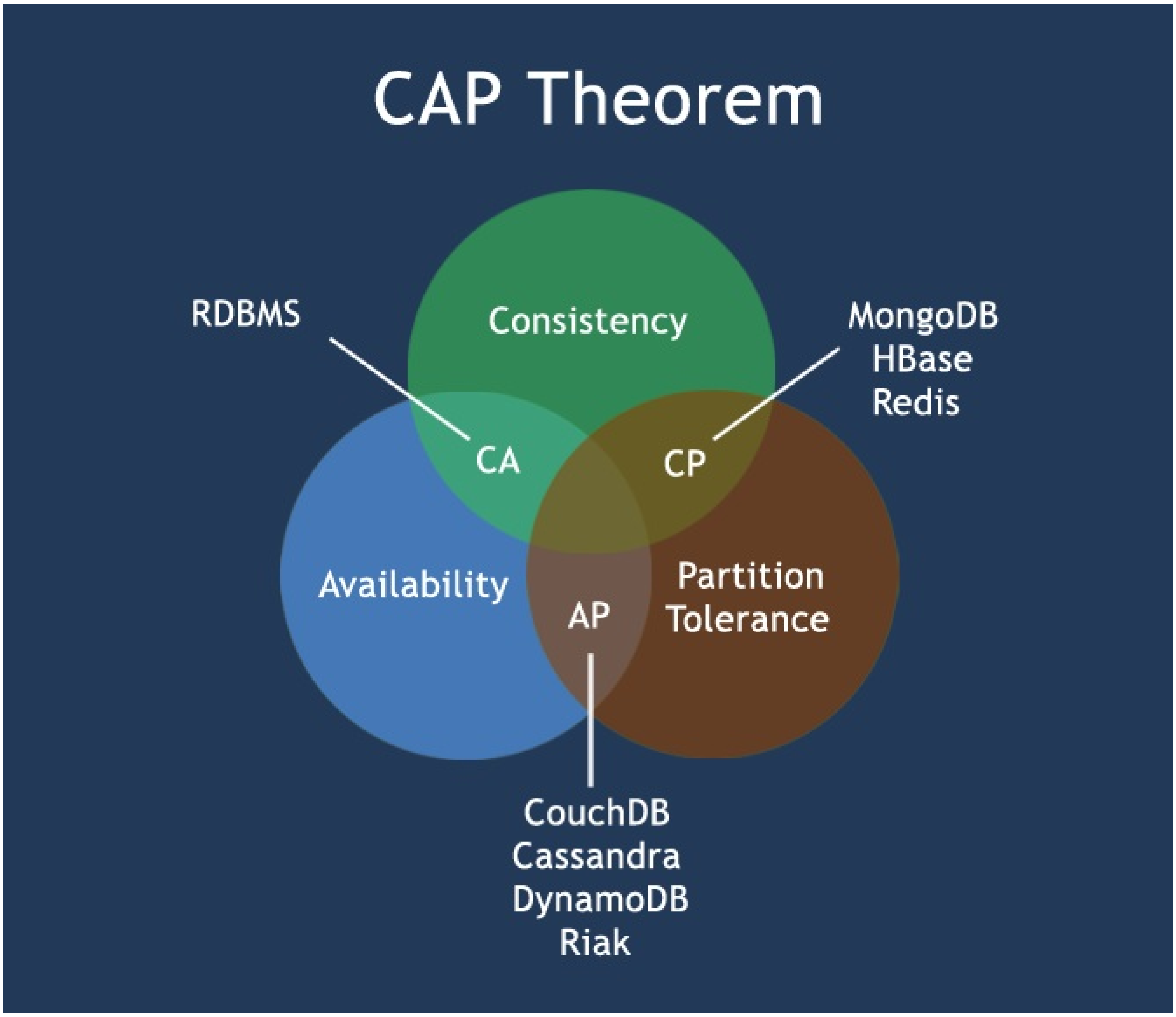
- ElasticSearch is a near real time search platform build on top of Apache Lucene™
- ElasticSearch supports ACID for a single document only
- Cluster is a collection of one or more nodes storing, indexing, and searching data
- Node is a single server in a cluster, nodes discover each other with unicast and by cluster name defined in /etc/elasticsearch/elasticsearch.yml
- Node can be configured as master-eligible, data, client, tribe, or left default: master-eligible and data node
- Index is a collection of documents and equivalent of Rdbms (Relational Database Management System) database or schema
- Type is a leftover from previous version as collection of documents of a specific type inside index, v6 supports one type per index only
- Document is a basic unit of operation for indexing, replications, and searching. An approximate equivalent of a row/record in Rdbms. Document can be searched for in an index, but when being indexed must be assigned to a type

Terminology - Cont'd

- Shards is a unit of storage and operation assigned to a node to distribute index across nodes in a cluster - to allow horizontal scalability
- Replication supports high availability as in case of a node failure and improves scalability by allowing search operations on replicas
- Clustering across multiple data centers, i.e. higher network latency and lower bandwidth - not supported



CAP Theorem


CAP Theorem
RDBMS
Consistency
MongoDB
HBase
Redis
CA
CP
Availability
AP
Partition
Tolerance
CouchDB
Cassandra
DynamoDB
Riak

CAP Theorem and Beyond

Pick two out of three

- Consistency - all nodes see all the data at the same time
- Availability - any node can execute read/write operations
- Partition tolerance - cluster continues to operate despite node(s) failure

Realistic choices

- CP - Consistency/Partition Tolerance: wait for a response from the partitioned node which could result in a timeout error
- AP - Availability/Partition Tolerance: get the most recent version of the data, possibly stale
- ElasticSearch is a search engine: consistency is at the document level

Data Modeling Best Practices

- Document is a unit of storage, indexing, search, and aggregation
- 3rd norm of normalization (or beyond) does not apply to ElasticSearch
- ElasticSearch supports no join or its equivalents
- Document indexing, searching, and aggregation uses no locks
- ElasticSearch designed for scalability
- Application side joins: a no-no in a Rdbms world and is a common practice in NoSql world
- Data de-normalization: redundant copies of data in each document removes needs for join
- Nested objects: storing parent and child data in a single document
- Parent-Child Relationship: storing child documents separately and are associated with parent document
- ElasticSearch lacks built-in mechanism for de-normalized data maintenance, so are many other NoSql

Data Examples

- Simple document

  ```
  POST http://localhost:9200/index-name/type-name/document-id {
    "message": "Hello World!"
  }
  ```

- Nested document

  ```
  POST http://localhost:9200/index-name/type-name/document-id {
    "message": "Hello World!",
    "keywords": [
        "cheerful",
        "happy"
    ]
  }
  ```

App Side Joins

- Docs stored, indexed, and searched individually - application joins the docs
- Fetch the order by id

  ```
  POST http://localhost:9200/orders/orders/1
  {
    "id": "1",
    "placedOn": "2016-10-17T13:03:30.830Z",
    "customerId": "123"
  }
  ```

- Fetch the customer by customer id from the order document

  ```
  POST http://localhost:9200/customers/customers/123
  {
    "id": "123",
    "firstName": "John",
    "lastName": "Smith"
  }
  ```

Data Examples - Parent-Child

- Docs are stored, indexed, and searched individually - children are associated with a parent in the same index

  ```
  POST http://localhost:9200/orders/order/123
  {
    "id": "123",
    "placedOn": "2018-01-01T13:35:03.034Z",
    "amount": 12.45
  }
  ```

  ```
  PUT orders/order/4?routing=123&refresh=true
  {
  "text": "This is another answer",
  "customer2customer": {
    "name": "customer",
    "parent": "123"
  }
  }
  ```

- Important: index mappings must be defined before hand! Discussed later...

Exercise Setup

Aws

- AWS Regions and Availability Zones
- Elastic Search cluster should be setup in one region and using multiple availability zones
- EC2 t2.medium instances with Ubuntu 16.04, could have been any Linux Distribution
- Root access via ssh using public/private key
- For Windows user please download putty
- For Mac and Linux users - ssh is available via terminal window
- If you don't have private/public key pair available follow instructions for Windows or for Mac
- Email public key to vladimir.khazin@icssolutions.ca to configure your access
- You can also use private ssh key emailed to you before the course
    - How to import private key using Putty
    - Setting permissions on Linux/MacOs: `chmod 600 /path/to/private-key/file`
- Another option is to use Secure Shell Chrome Extension to launch in-browser ssh

One Node Setup Exercise

- Login into your sandbox
- Update distro using terminal window:

  ```
  sudo apt-get update && sudo apt-get install apt-transport-https -y
  ```

- Install Java Runtime Environment using terminal window

  ```
  sudo apt-get install default-jre -y
  ```

- Download and install Public Signing Key:

  ```
  curl https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo apt-key add -
  ```

- Add repository definition:

  ```
  echo "deb https://artifacts.elastic.co/packages/6.x/apt stable main" | sudo tee -a
   /etc/apt/sources.list.d/elastic-6.x.list
  ```

- Install Elastic Search:

  ```
  sudo apt-get update && sudo apt-get install elasticsearch
  ```

- Start Elastic Search service:

  ```
  sudo service elasticsearch start
  ```

- We will be using 'curl' to troubleshoot setup and to run our first queries before we install and configure Kibana
- Give it a moment to finish the initialization and verify it is running:

  ```
  curl localhost:9200
  ```

- Expected response:

```
{
"name" : "A28UK7n",
"cluster_name" : "elasticsearch",
"cluster_uuid" : "6-nG7QniTiuSFtPtJOdJsg",
"version" : {
  "number" : "6.0.0",
  "build_hash" : "8f0685b",
  "build_date" : "2017-11-10T18:41:22.859Z",
  "build_snapshot" : false,
  "lucene_version" : "7.0.1",
  "minimum_wire_compatibility_version" : "5.6.0",
  "minimum_index_compatibility_version" : "5.0.0"
},
"tagline" : "You Know, for Search"
}
```

- Posting first document:

```
curl -XPOST 'localhost:9200/orders/orders/1?pretty=true' \
  -H 'content-type: application/json' \
  -d '{
"id": "1",
"placedOn": "2016-10-17T13:03:30.830Z",
"status": "shipped"
}'
```

- Expected Response:

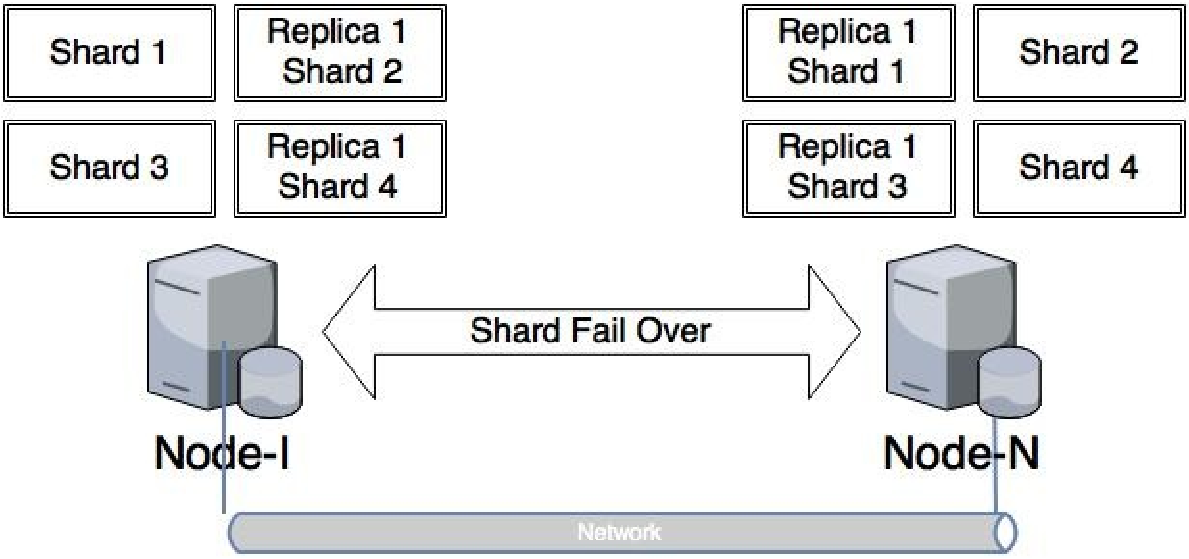
```
{
  "_index" : "orders",
  "_type" : "orders",
  "_id" : "1",
  "_version" : 2,
  "result" : "created",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 0,
  "_primary_term" : 1
}
```

- `_seq_no` : unique sequence of indexing operation
- `_primary_term` : shard id where primary copy stored
- First query:

```
curl 'localhost:9200/orders/orders/_search?pretty=true&q=id:1'
```

- Please review results - where are: doc id, document data, index name, type name, and search score?

Distributed Database



- How do I connect to multiple nodes?
- Client-Side vs. Service-Side load balancing: pros and cons

Cluster, Shards, and Replicas

- How many nodes in the cluster?
- Shards cannot be split between nodes, a shard is a complete Lucene index
- The question therefore is how many shards per cluster and then per node
- Max Jvm Heap size recommended for ElasticSearch: 32GB
- Jvm Heap size recommended at half of the RAM
- Number of shards is often based on the dataset size and many organizations mistakenly over-allocate
- How many number of replicas should I have?
- You may have guessed the answer - it depends...
- Number of replicas affects more than fault tolerance: write performance, read performance, and split-brain problem
- Fault tolerance rule is N+1, therefore is you would like you data to be stored twice - replica settings should be equal to 2
- Write performance - in extreme cases index request to cluster will time-out when number of available nodes is less than number of replicas configured for the index
- Read performance - search uses replicas as well, more replicas should result in faster searches and aggregations
- Split-brain problem - no permanent solution, designated nodes complicate cluster setup and operation, but offer more granular control.

Multi-Index Operations and Aliases

- ElasticSearch is not a traditional database, searches can be executed acorss indices with no known performance implications
- Entire cluster search:

  ```
  curl 'localhost:9200/_search?q=id:1'
  ```

- Multiple indices search:

  ```
  curl 'localhost:9200/index1,index2/_search?q=id:1'
  ```

- Wildcard search:

  ```
  curl 'localhost:9200/index*/_search?q=id:*'
  ```

- Index alias - create an alias for index or for group of indexes:

  ```
  curl -XPOST 'http://localhost:9200/_aliases' -H 'content-type: application/json' -
  d '
  {
    "actions" : [
        { "add" : { "index" : "test1", "alias" : "alias1" } },
        { "add" : { "index" : "test2", "alias" : "alias1" } }
    ]
  }'
  ```

- Why bother creating aliases? Logical data partitioning, archiving by index, access control...

Chapter Summary

- What problems ElasticSearch solves?
- Can I replace my database with ElasticSearch?
- What is distributed database?
- How would you define cluster, shard, node, index, and document?
- What are factors to take in consideration when sizing the cluster?
- How do you scale-out a shard?

Chapter 2: ElasticSearch Index

- In-depth analysis of mappings, indexing, and operations
- Discussion of transaction logs and Lucene indexing
- Understanding configuration options, mappings, APIs, and available settings

Index

- Index is broken down, stored, and processed as collection of shards
- Each shard is a complete Lucene Index and cannot be scaled-out
- Form Lucene performance perspective larger index is faster than series of smaller indices
- Shards are allocated to nodes and are searched independently
- At the completion of search individual shards results must be aggregated - too many shards may impact performance
- Index cannot store documents directly - types are used as buckets to index documents

Type

- Type is a mechanism to store different data in the same index
- Types help reducing number of indices for previously discussed performance reasons
- Searching across types within index and between shards adds no overhead
- Lucene implications - field that exists in one type will consume resources in other types
- Fields across types in an index must use consistent data types, e.g. string or number, not string and number
- Score for search results calculated at the index level
- Mapping for document's properties are defined at the type level
- Search and aggregation can be executed at type, index, alias, multi-index, or cluster levels

Index and Type Api

- How do we create an index?
- Index will be created lazily by ElasticSeach when we post a document:

  ```
  curl -XPOST localhost:9200/orders/orders/1 \
  -H 'content-type: application/json' \
  -d '{"id": "1", "placedOn": "2016-10-17T13:03:30.830Z"}'
  ```

- Type will be created lazily by the same operation
- To retrieve the document just posted:

  ```
  curl 'localhost:9200/orders/orders/_search?pretty=true'
  ```

- Expected result: ``` { "took" : 0, "timed_out" : false, "_shards" : { "total" : 5, "successful" : 5, "skipped" : 0, "failed" : 0 }, "hits" : { "total" : 1, "max_score" : 1.0, "hits" : [

  ```
  {
    "_index" : "orders",
    "_type" : "orders",
    "_id" : "1",
    "_score" : 1.0,
    "_source" : {
      "id" : "1",
      "placedOn" : "2016-10-17T13:03:30.830Z"
    }
  }
  ```

  ] } }

```
```

Transaction Log and Lucene Index

- Lucene index is organized during commit phase - relatively heavy operation
- Lucene does not have built-in transaction log capabilities
- Change made between two commit operations is lost in case of a failure
- To minimize data loss each shard uses write ahead log or transaction log
- In case of crash recent operations can be replayed back to Lucene index
- Flush is performing Lucene commit and is starting new transaction log in the background
- Flush and transaction log settings are configured in elasticsearch.yml

Index Configuration

- elasticsearch.yml defines defaults for index configuration:

  ```
  number_of_shards: 5
  number_of_replicas: 1
  ```

- Each index can be configured with desired number of shard during creation:

  ```
  curl -XPUT 'localhost:9200/orders' \
  -H 'Content-Type: application/json' \
  -d'
  {
  "settings": {
    "number_of_shards": 1,
    "number_of_replicas": 0
  }
  }'
  ```

- Do you recall discussion about multi-index search capabilities?

Index Settings

- Number of shards cannot be changed after index has been created
- Number of replicas can be updated on existing index:

  ```
  PUT /index-name/_settings
  {
    "number_of_replicas": 5
  }
  ```

- Number of replicas can be configured dynamically:

  ```
  index.auto_expand_replicas: 0-5
  ```

- Query size can be limited to conserve heap memory:

  ```
  index.max_result_window: 10000
  ```

- More index settings

Shrink Index

- Number of shards cannot be changed after index has been created, but
- New in version 5.x - index can be shrunk to smaller number of shards
- Index must be in good health and in read-only state, can be achieved with following request:

```
PUT /index-name/_settings
{
"settings": {
  "index.routing.allocation.require._name": "new-node-name",
  "index.blocks.write": true
}
}
```

- Shrink Index is a single RESTful command:

```
POST source-index/_shrink/target-index
```

- Shrink Index is brand new in version 5.x, it is ready for prime time? https://www.elastic.co/guide/en/elasticsearch/reference/current/indices-shrink-index.html

Mapping

- Defines how document and its fields are stored and indexed
- Mapping can be derived dynamically by ElasticSearch
- Handful core data types are supported
- Mapping can be added to existing index for new fields
- Existing field mapping not always possible to modify
- ElasticSearch will derive mapping for new type and for new fields:

```
curl -XPOST localhost:9200/orders/orders/1 \
-H 'Content-Type: application/json' \
-d '{"id": "1", "placedOn": "2016-10-17T13:03:30.830Z"}'
```

- Retrieving existing mappings:

```
curl 'localhost:9200/orders/orders/_mapping?pretty=true'
```

- Expected response: \`\`\` { "orders" : { "mappings" : {

```
"orders" : {
  "properties" : {
    "id" : {
      "type" : "text",
      "fields" : {
        "keyword" : {
          "type" : "keyword",
          "ignore_above" : 256
        }
      }
    },
    "placedOn" : {
      "type" : "date"
    }
  }
}
```

} } }

\`\`\`

- Rich support for date-time formats

Mapping Exercise

- Login into your sandbox
- We have not yet configured elasticsearch service to start automatically, start using terminal window:

  ```
  sudo service elasticsearch start
  ```

- Service will start but listener will take its time before responding to incoming requests:

  ```
  curl localhost:9200
  ```

- Give it few minutes before you get json response
- Post new document:

  ```
  curl -XPOST localhost:9200/orders/orders/1 \
  -H 'content-type: application/json' \
  -d '
  {
  "id": "1",
  "placedOn": "2016-10-17T13:03:30.830Z"
  }'
  ```

- Fetch mapping:

  ```
  curl 'localhost:9200/orders/orders/_mapping?pretty=true'
  ```

- Expected response:

```
{
"orders" : {
  "mappings" : {
    "orders" : {
      "properties" : {
        "id" : {
          "type" : "text",
          "fields" : {
            "keyword" : {
              "type" : "keyword",
              "ignore_above" : 256
            }
          }
        },
        "placedOn" : {
          "type" : "date"
        }
      }
    }
  }
}
}
```

- Try modifying existing mapping:

```
curl -XPUT 'localhost:9200/orders/_mapping?pretty=true' \
-H 'content-type: application/json' \
-d '
{
"orders" : {
"mappings" : {
  "orders" : {
    "properties" : {
      "id" : {
        "type" : "text",
        "fields" : {
          "keyword" : {
            "type" : "keyword",
            "ignore_above" : 256
          }
        }
      },
      "placedOn" : {
        "type" : "date",
        "format" : "strict_date_optional_time||epoch_millis"
      }
    }
  }
}
}
}'
```

- What's the outcome? And why?
- Try modifying existing type mapping:

```
curl -XPUT 'localhost:9200/orders/orders/_mapping?pretty=mapping' \
-H 'content-type: application/json' \
-d '{
"orders" : {
  "mappings" : {
    "orders" : {
      "properties" : {
        "id" : {
          "type" : "text",
          "fields" : {
            "keyword" : {
              "type" : "keyword",
              "ignore_above" : 256
            }
          }
        },
        "placedOn" : {
          "type" : "date",
          "format" : "strict_date_optional_time||epoch_millis"
        }
      }
    }
  }
}
}'
```

- What now? Why?
- Let us try again:

```
curl -XPUT 'localhost:9200/orders/orders/_mapping?pretty=mapping' \
-H 'content-type: application/json' \
-d '
{
"orders" : {
  "properties" : {
    "id" : {
      "type" : "text",
      "fields" : {
        "keyword" : {
          "type" : "keyword",
          "ignore_above" : 256
        }
      }
    },
    "placedOn" : {
      "type" : "date",
      "format" : "strict_date_optional_time||epoch_millis"
    }
  }
}
}'
```

- Did it work? What's the difference?
- Let's modify data type for existing field

```
curl -XPUT 'localhost:9200/orders/orders/_mapping?pretty=true' \
-H 'content-type: application/json'
-d '
{
"orders" : {
  "properties" : {
    "id" : {
      "type" : "double"
    },
    "placedOn" : {
      "type" : "date",
      "format" : "strict_date_optional_time||epoch_millis"
    }
  }
}
}'
```

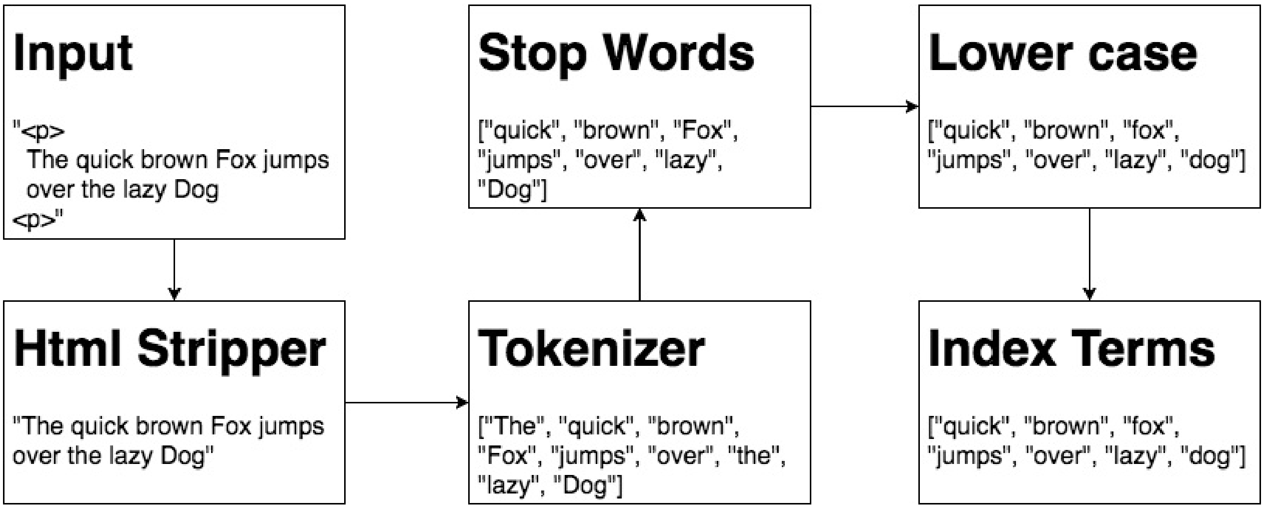
- Seriously, what now? Why? So much for the dynamic data mapping?
- What if we need to change data type after we have indexed the data?
- There is no (more) option to delete the mapping - delete and recreate index is the only option :-(

```
curl -XDELETE localhost:9200/orders
curl -XPUT localhost:9200/orders
```

- User json data from the previous step to define the mapping
- Watch out! There is no warning or confirmation using curl!
- Psst: look out for the proper response too:

```
{
"acknowledged": true
}
```

Analyzer


Input
"<p>
The quick brown Fox jumps
over the lazy Dog
<p>"
Html Stripper
"The quick brown Fox jumps
over the lazy Dog"
Tokenizer
["The", "quick", "brown",
"Fox", "jumps", "over", "the",
"lazy", "Dog"]
Stop Words
["quick", "brown", "Fox",
"jumps", "over", "lazy",
"Dog"]
Lower case
["quick", "brown", "fox",
"jumps", "over", "lazy", "dog"]
Index Terms
["quick", "brown", "fox",
"jumps", "over", "lazy", "dog"]

Not Analyzed & Multi-Fields

- Lucene and hence ElasticSearch break strings into terms using built-in or custom tokenizers
- Some strings don't make sense to tokenize e.g. uuid or guid often used as equivalent of a primary key and/or unique identifier
- not_analyzed: ElasticSearch mapping option to suppress tokenization:

```
curl -XPUT 'localhost:9200/orders/orders/_mapping?pretty=true' \
 -H 'Content-Type: application/json' \
 -d '
{
 "orders": {
    "properties": {
       "id": {
          "type": "text",
          "index": false
       }
    }
 }
}'
```

- What you think difference will be searching or aggregating tokenized uuid/guid vs. non-tokenized uuid/guid property?
- What if I need both tokenized and non-tokenized option for the same field?
- Multi-Fields Mapping allows double indexing the same data:

```
curl -XPUT 'localhost:9200/ordering/orders/_mapping?pretty=true' -d '
{
 "orders":{
    "properties": {
       "streetName": {
          "type":"text",
          "fields": {
             "notparsed": {
                "type":"keyword",
                "index":"not_analyzed"
             }
          }
       }
    }
 }
}'
```

Not Analyzed & Multi-Fields Exercise

- Login into you virtual box
- Delete previously created index and its mapping:

```
curl -XDELETE 'localhost:9200/orders?pretty=true'
```

- Post new document:

```
curl -XPOST 'localhost:9200/orders/orders/1?pretty=true'
-H 'content-type: application/json' \
-d '{"id": "1", "placedOn": "2016-10-17T13:03:30.830Z"}'
```

- Fetch mapping:

```
curl 'localhost:9200/orders/orders/_mapping?pretty=true'
```

- Expected response:

```
{
"orders" : {
  "mappings" : {
    "orders" : {
      "properties" : {
        "id" : {
          "type" : "text",
          "fields" : {
            "keyword" : {
              "type" : "keyword",
              "ignore_above" : 256
            }
          }
        },
        "placedOn" : {
          "type" : "date"
        }
      }
    }
  }
}
}
```

- Add mapping for a new field

```
curl -XPUT 'localhost:9200/orders/orders/_mapping?pretty=true' \
-H 'content-type: application/json' \
-d '
{
"orders" : {
  "properties" : {
    "id" : {
      "type" : "text"
    },
    "placedOn" : {
      "type" : "date",
      "format" : "strict_date_optional_time||epoch_millis"
    },
    "trackingId" : {
      "type" : "keyword"
    }
  }
}
}'
```

- Expected response:

```
{"acknowledged":true}
```

- Populate new order with spaces in id and trackingId fields/properties:

```
curl -XPOST 'localhost:9200/orders/orders/1?pretty=true' \
-H 'content-type: application/json' \
-d '
{
"id": "orderId with spaces",
"placedOn": "2016-10-17T13:03:30.830Z",
"trackingId": "trackingId with spaces"
}'
```

- Let's run first search:

```
curl 'localhost:9200/ordering/orders/_search?pretty=true&q=id:orderId'
```

- Did you get any results?
- Let's run second search:

```
curl 'localhost:9200/ordering/orders/_search?pretty=true&q=trackingId:trackingId'
```

- Did you get any results?
- What's the difference in behaviour and why?
- Adding mapping for multi-field:

```
curl -XPUT localhost:9200/orders/orders/_mapping \
-H 'Content-Type: application/json' \
-d '
{
"orders" : {
  "properties":{
     "streetName":{
        "type":"text",
        "fields":{
           "notparsed":{
              "type":"keyword"
           }
        }
     }
  }
}
}'
```

- Re-populate the data:

```
curl -XPOST 'localhost:9200/orders/orders/1?pretty=true' \
-H 'Content-Type: application/json' \
-d '
{
"id": "string with spaces",
"placedOn": "2016-10-17T13:03:30.830Z",
"streetName": "name with spaces"
}'
```

- Let's search for the street name:

```
curl 'localhost:9200/ordering/orders/_search?pretty=true&q=streetName:name'
```

- Let's search for the street name on not-parsed field:

```
curl 'localhost:9200/orders/orders/_search?pretty=true&q=streetName.notparsed:name
'
```

- Let's search for the street name on not-parsed field again:

```
curl 'localhost:9200/ordering/orders/_search?pretty=true&q=streetName.notparsed:na
me%20with%20spaces'
```

- What are results in the search #1, #2, and #3; and what is the reason for these results?

Summary

- How can we create an index?
- What's type and what's it role?
- How would you define mapping?
- Do we define mapping at index or at type level
- Can we change mapping for existing fields?
- How would you define transactions in Lucene?
- What advanced mapping options we have covered?

Chapter 3: Search

- Understanding search Query DSL
- In-depth understanding of search components: aggregations, search types, highlighting and other options.
- Overview of Filter DSL compared to Query DSL

Query Dsl

- Json defined query:

```
curl -XPOST 'http://localhost:9200/orders/orders/_search?pretty=true' \
      -H 'content-type:application/json' \
      -d '
{
 "query": {
    "bool": {
       "must": {
       "query_string" : {
          "query": "id:1"
          }
       }
    }
 }
}'
```

Query Dsl Cont'd

- Leaf query clause: an equivalent of where clause in Sql statement for a particular field value
- `query_string` is using Lucene syntax: `id:1`
- Compound query clause: an equivalent of `and` , `or` , `not` , and etc. operators in sql statement
- `bool` represents boolean combinations of other queries, e.g.: `must` and `filter`
- `must` represents a condition that must be matched and will be used for computing score or relevancy of the search result
- `filter` act as a `must` condition, but will not contribute to the computation of score or relevancy of document found
- Additional options for bool are: should and must_not

Query Dsl Leaf Clause

- Field(s) level `where` clause
- `match_all` the most simplistic query

  ```
  curl -XPOST 'localhost:9200/_search?pretty=true' \
  -H 'content-type:application/json' \
  -d '{
   "query": {
      "match_all": {}
   }
  }'
  ```

- Simple match query:

  ```
  curl -XPOST 'localhost:9200/_search?pretty=true' \
  -H 'content-type:application/json' \
  -d '{
  "query": {
    "match": {
      "streetName": "name"
    }
  }
  }'
  ```

Query Dsl Term

- Frequently used to match on exact value
- Equivalent of sql statement:

  ```
  select * from orders where id = "1"
  ```

- Example

  ```
  curl -XPOST 'localhost:9200/orders/orders/_search?pretty=true' \
  -H 'content-type:application/json' \
  -d '{
  "query" : {
    "term": {
      "id": "1"
    }
  }
  }'
  ```

- Note that 'from orders' part of the statement is part of the url rather than body
- Capable of handling numbers, booleans, dates, and text.
- Often used as a filter rather than for scoring, commonly used with `constant_score` :

  ```
  curl -XPOST 'localhost:9200/orders/orders/_search?pretty=true' \
  -H 'content-type:application/json' \
  -d '{
  "query" : {
    "constant_score" : {
      "filter" : {
        "term": {
          "id": "1"
        }
      }
    }
  }
  }'
  ```

Query Dsl Terms

- Used to match on any of the values
- Equivalent of sql statement:

  ```
  select * from orders where id = "1" or id = "2"
  ```

- Example

  ```
  curl -XPOST 'localhost:9200/orders/orders/_search?pretty=true' \
  -H 'content-type:application/json' \
  -d '{
  "query" : {
    "terms": {
      "id": [ "1", "2"]
    }
  }
  }'
  ```

- Can be extended with lookup - where list of values is a reference to a doc:

  ```
  curl 'localhost:9200/orders/orders/_search?pretty=true' \
  -H 'content-type:application/json' \
  -d '{
  "query" : {
    "terms": {
      "id": {
        "index" : "lateorders",
        "type" : "lateorders",
        "id" : "2018-01-01",
        "path" : "ids"
      }
    }
  }
  }'
  ```

Query Dsl Range

- Used to match on any range of values
- Equivalent of sql statement:

  ```
  select * from orders where id >= 1 and id <= 2
  ```

- Example

  ```
  curl -XPOST 'localhost:9200/orders/orders/_search?pretty=true' \
  -H 'content-type:application/json' \
  -d '{
   "query" : {
      "range" : {
        "id" : {
        "gte" : 1,
        "lte" : 2
        }
      }
    }
  }'
  ```

- More leaf query options...

Query Dsl Compound Clause

- Combing leaf query clauses
- bool frequently used and is simple to use:

```
curl 'localhost:9200/orders/orders/_search?pretty=true' \
-H 'content-type:application/json' \
-d '{
"query" : {
  "bool" : {
    "must" : {
      "term" : { "id" : "1" }
    },
    "should": {
      "query_string" : {
          "query" : "trackingId:*"
      }
    }
  }
}
}'
```

Query Pagination

- Query results are limited to page size of 10 by default
- Query pagination and page number controlled by From/Size parameters:

```
curl -XPOST 'localhost:9200/orders/orders/_search?pretty=true' \
-H 'content-type: application/json' \
-d '
{
  "from":0,
  "size":2,
  "query": {
    "match_all":{}
  }
}'
```

- Expected result (partially presented):

```
{
...
"hits" : {
  "total" : 1,
  "max_score" : 1.0,
  "hits" : [
    {
      "_index" : "orders",
      "_type" : "orders",
      "_id" : "1",
      "_score" : 1.0,
      "_source" : {
        "id" : "1",
        "placedOn" : "2016-10-17T13:03:30.830Z"
      }
    },
    ...
  ]
}
}
```

- Note hits.total field, what it stands for?
- How results are sorted?

Query Uri

- In addition to json dsl there is URI Search
- Support is more limited than json Dsl, but Kibana seems to be just fine with it
- **q** parameter allows to specify query in lucene formatted query:

  ```
  curl 'localhost:9200/orders/orders/_search?q=placedOn:*&pretty=true'
  ```

- Expected result:

  ```
  {
  ...
  "hits" : {
    "total" : 1,
    "max_score" : 0.30685282,
    "hits" : [
      {
        "_index" : "ordering",
        "_type" : "order",
        "_id" : "3",
        "_score" : 0.30685282,
        "_source" : {
          "id" : "3",
          "placedOn" : "2016-10-01T00:00:00Z",
          "status" : "shipped"
        }
      },
      ...
    ]
  }
  }
  ```

Aggregation Query

- First aggregation query:

```
curl -XPOST 'http://localhost:9200/orders/orders/_search?pretty=true' \
-H 'content-type: application/json' \
-d '
{
"size": 0,
"aggregations": {
  "order-status": {
    "terms": {
      "field": "status.keyword"
    }
  }
}
}'
```

- "size": 0 - suppress query results to fetch aggregations results only
- "aggregations" or "aggs" - part of ElasticSearch Dsl
- "order-status" - an arbitrary name for aggregation
- "terms" - type of aggregation to use
- "status.keyword" - multi-field mapping for text fields

Query Exercise

- Login into your ElasticSearch sandbox
- Make sure elastic search is running:

  ```
  sudo service elasticsearch restart
  ```

- Populate few orders:

  ```
  curl -XPOST localhost:9200/orders/orders/2 \
  -H 'content-type: application/json' \
  -d '
  {
  "id": "2",
  "placedOn": "2017-01-01T00:00:00Z",
  "status": "pending"
  }'
  ```

  ```
  curl -XPOST localhost:9200/orders/orders/3 \
  -H 'content-type: application/json' \
  -d '
  {
  "id": "3",
  "placedOn":
  "2016-10-01T00:00:00Z",
  "status": "shipped"
  }'
  ```

  ```
  curl -XPOST localhost:9200/orders/orders/4 \
  -H 'content-type: application/json' \
  -d '
  {
  "id": "4",
  "placedOn": "2016-01-01T00:00:00Z",
  "status": "received"
  }'
  ```

- Confirm there are some records to search on:

  ```
  curl 'localhost:9200/orders/orders/_search?pretty=true'
  ```

- How many documents did you find?
- How do you know whether got all the documents or just first page of records?

- How do you find all orders that were shipped?
- What is the order of results?
- How do you sort result using an arbitrary field?
- Pick couple of options and run your sort query
- Reformat your query to use query uri instead of query json
- You will be asked to present your sort findings to others...

Highlighting

- Due to the elastic search text analyzers it is not always obvious why document was a match
- Highlights search result on one or more document fields:

```
curl -XPOST 'localhost:9200/orders/orders/_search?pretty=true' \
  -H 'content-type: application/json' \
  -d '
{
  "query" : {
      "match": { "id": "1" }
  },
  "highlight" : {
      "fields" : {
          "id" : {}
      }
  }
}'
```

- Expected result:

```
{
...
  "hits" : {
      "total" : 1,
      "max_score" : 0.71231794,
      "hits" : [ {
        "_index" : "orders",
        "_type" : "orders",
        "_id" : "1",
        "_score" : 0.71231794,
        "_source" : {
          "id" : "1",
          "placedOn" : "2016-10-17T13:03:30.830Z"
        },
        "highlight" : {
          "id" : [ "<em>1</em>" ]
        }
    } ]
  }
}
```

Filter

- Leaf and Compound components of the search Dsl can be used in query and in filter context
- So far we have been using Search Dsl in the query context
- Filtering context is a 'non-scoring' or 'filtering' query - yes/no answer, no score is computed
- Typical use of query is to find a best matching document, similar to Google Search
- Scoring calculates how relevant each document is, relative to the search criteria
- Using Search Dsl in filtering context is filtering documents out

Filter Example

- Query search:

  ```
  curl -XPOST 'localhost:9200/collisions/collisions/_search?pretty=true' \
  -H 'content-type:application/json' \
  -d '{
  "query": {
    "term": { "COUNTY_NAME": "worcester" }
  }
  }'
  ```

- Filter search:

  ```
  curl -XPOST 'localhost:9200/collisions/collisions/_search?pretty=true' \
  -H 'content-type:application/json' \
  -d '
  {
  "query": {
    "bool": {
      "filter": {
        "term": { "COUNTY_NAME": "worcester" }
      }
    }
  }
  }'
  ```

- Psst: convert the term value to lower case, or face an empty hits.hits response

Filter Exercise

- Login into your ElasticSearch sandbox
- Make sure elastic search is running:

```
sudo service elasticsearch restart
```

- Populate few sample data borrowed from data.gov:

```
curl https://elasticsearch-courseware.icssolutions.ca/examples/data-sets/collisions.txt -o collisions.txt
curl -XPOST 'localhost:9200/_bulk' -H 'content-type:application/json' --data-binary "@collisions.txt"
```

- Confirm there are some records to search on:

```
curl 'localhost:9200/collisions/collisions/_search?pretty=true'
```

- How many documents did you find?
- Pick couple of options to search on: different leaf and compound clauses, different field types
- Rewrite your queries into filtered query
- You will be asked to present your findings to others

Chapter 3 - Summary

- What are the components of Query Dsl?
- What's the difference between leaf and compound clause
- How search results are sorted?
- What's score?
- What's the difference between a search and a filter context?
- How to paginate through search results?
- Should I be using ElasticSearch for all my queries?

Chapter 4: Advanced Search and Mapping

- Introduction to data aggregations and nested document relations
- Understanding nested objects and parent-child relationships
- Aggregation queries

Data Aggregation

- Main entity includes or aggregates related data in it
- Movie data - hierarchical and verbose
- Directors, crew, producers, images, and etc... How to index it all?
- Example of a movie data:

```
{
"AssetId" : "86c1bba8-d18f-4bbc-9cb4-a90a4220f59c",
"Title" : "My First Mister",
"ShortSynopsis" : "Desperate to escape the world of her infuriatingly cheery mothe
r and mindless classmates, punk-rebel Jennifer impulsively applies for a job at Ra
ndall's conservative clothing store where an unlikely friendship blooms.",
"RunTimeSec" : 6552,
"StarRating" : 4,
"AvailableDate" : "2016-03-01T08:00:00",
"ExpirationDate" : "2018-02-27T09:00:00",
"GeoRestriction" : "Canada Only",
"EndCreditsTimeMarkerSec" : 6285,
"Directors" : [ "Christine Lahti" ],
"Starring" : [ {
    "CastCrewName" : "Leelee Sobieski",
    "CastCrewRole" : "Cast",
    "Weight" : 100,
    "Order" : 0
  }, {
    "CastCrewName" : "Albert Brooks",
    "CastCrewRole" : "Cast",
    "Weight" : 99,
    "Order" : 1
  }, {
    "CastCrewName" : "Mary Kay Place",
    "CastCrewRole" : "Cast",
    "Weight" : 98,
    "Order" : 2
  }
]
}
```

Data Aggregation Cont'd

- Aggregated data is not likely to be stored in the same shape as it is in a Rdbms
- Rdbms more likely to store data in related tables using foreign key relationship, whether enforced or not
- Aggregating data in ElasticSearch from Rdbms is likely to result in data de-normalization
- Rdbms likely data model:

- Movies

AssetId	Title	ShortSynopsis	...
86c1bba8-d18f-4bbc-9cb4-a90a4220f59c	My First Mister	Desperate to escape the..	...

- Cast Members

MemberId	Name	...
b42e5484-e7e1-4eaf-b9a2-b8f25487533b	Leelee Sobieski	...
1290de91-2bc3-4b72-9a0d-6fa0e04b44ab	Albert Brooks	...

- Movie Cast Members

AssetId	MemberId	Role
86c1bba8-d18f-4bbc-9cb4-a90a4220f59c	b42e5484-e7e1-4eaf-b9a2-b8f25487533b	Cast
86c1bba8-d18f-4bbc-9cb4-a90a4220f59c	1290de91-2bc3-4b72-9a0d-6fa0e04b44ab	Cast

- Document databases such as Couchbase and MongoDb may store data aggregated neatly already

Nested Datatype

- Lucene has no concept of inner objects and hence data is flattened when being indexed with not quite as expected behavior.
- Data posted

  ```
  {
  "group" : "fans",
  "user" : [{
      "first" : "John", "last" :  "Smith"
    },{
      "first" : "Alice", "last" :  "White"
    }
  ]
  }
  ```

- Indexed by Lucene as:

  ```
  {
  "group" :        "fans",
  "user.first" : [ "alice", "john" ],
  "user.last" :  [ "smith", "white" ]
  }
  ```

- What you think would be a consequence of such indexing?
- Should you be able to find person `user.first:john AND user.last:white` ?
- Nested is a specialized version of the 'object' datatype that allows querying arrays of objects independently from each other addressing less than perfect search results

Nested Datatype Mapping

- To maintain granularity of 'user' array indexing - 'nested' datatype can be used to index each object as an independent, hidden document:

  ```
  {
     "mappings": {
       "groups": {
         "properties": {
           "user": {
             "type": "nested"
           }
         }
       }
     }
  }
  ```

- With above mapping user `user.first:john AND user.last:white` won't be found, with special query syntax

Nested Datatype Querying

- The query against nested objects is executed as if the nested documents were indexed independently from the parent document using special query Dsl syntax:

```
{
"nested" : {
  "path" : "user",
  "query" : {
    "bool" : {
      "must" : [
        {
            "match" : {"user.first" : "John"}
        },
        {
            "match" : {"user.last" : "Smith"}
        }
      ]
    }
  }
}
}
```

- Additional options available for nested datatype mapping, querying, and aggregation

Parent-Child Relationship

- Similar in nature to the nested model: both allow you to associate one entity with another
- Parent-Child linked documents will be stored in the same shard, possible trouble with data distribution
- With nested datatype all data lives within the scope of parent document
- With parent-child: parent and each child are indexed separately
- Children can be queried, updated, and deleted separately from the parent and from each other
- With large number of child documents parent-child indexing is more effective
- Child documents can be incorporated into query criteria and into the query results
- Parent-Child maps are stored in so called doc values quick in-memory processing and scalable split on disk
- There are still performance concerns voiced in Elastic documentation

Parent-Child Mapping

- Requires mapping at the time of index creation or using update-mapping
- Creating new index with two types for parent-child relationship:

```
curl -XPUT 'localhost:9200/politics?pretty=true' -d '
{
 "mappings": {
   "party": {},
   "supporter": {
     "_parent": {
       "type": "party"
     }
   }
 }
}'
```

Parent-Child Indexing

- Index parent first:

```
curl -XPOST 'localhost:9200/politics/party/1' -H 'content-type: application/json' -d '
{
 "id": 1,
 "name": "The Heartless"
}'
```

```
curl -XPOST 'localhost:9200/politics/party/2' -H 'content-type: application/json' -d '
{
 "id": 2,
 "name": "The Brainless"
}'
```

- Index child/children second:

```
curl -XPUT 'localhost:9200/politics/supporter/101?parent=1&pretty=true' -H 'content-type: application/json' -d '
{
"id": "101",
"name":  "Jane Smith",
"dob":   "1970-10-24"
}'
```

```
curl -XPUT 'localhost:9200/politics/supporter/201?parent=2&pretty=true' -H 'content-type: application/json' -d '
{
"id": "201",
"name":  "John Smith",
"dob":   "1970-01-13"
}'
```

Parent-Child Searching

- Searching for parent by child:

```
curl 'localhost:9200/politics/party/_search?pretty=true' -d '{
"query": {
  "has_child": {
    "type": "supporter",
    "query": {
      "term": {
        "dob": "1970-10-24"
      }
    }
  }
}
}'
```

- Search for children by parent:

```
curl 'localhost:9200/politics/supporter/_search?pretty=true' -d '{
"query": {
  "has_parent": {
    "type": "party",
    "query": {
      "match": {
        "name": "The Heartless"
      }
  }
}
}'
```

Search Exercise

- Log-in into your ElasticSearch sandbox
- Make sure elastic search is running:

  ```
  sudo service elasticsearch restart
  ```

- We have covered following data-model approaches: aggregation, nested objects, and parent-child relationship
- There is likely not enough time to cover all options in a single exercise
- Pick a use case for data to map and discuss what's the best way to define mapping: simple, nested, or parent-child
- Create index using create index api:

  ```
  curl -XPUT 'localhost:9200/<index-name>?pretty=true' \
  -H 'content-type:application:json' \
  -d '
  {
  "mappings": {
    ...
  }
  }'
  ```

- Populate test data
- Come up with queries to run
- Execute and troubleshoot queries using syntax specific to your use-case: simple, nested, parent-child
- If there is time - consider alternative data-modelling and discuss pros/cons between the two
- You will be asked to present your considerations and findings

Terms Aggregation

- Recall first aggregation query:

```
curl -XPOST 'localhost:9200/orders/orders/_search?pretty=true' \
-H 'content-type: application/json' \
-d '
{
"size": 0,
"aggregations": {
  "order-status": {
    "terms": {
      "field": "status.keyword"
    }
  }
}
}'
```

- size": 0 - suppress raw query results to return aggregations only
- "aggregations" or "aggs" - part of ElasticSearch Dsl
- "order-status" - an arbitrary name for aggregation
- "terms" - type of aggregation to use
- Why the field is "status.keyword" rather than "status"?

Nested Aggregations

- Example of nested aggregation query, terms and histogram:

```
curl -XPOST 'localhost:9200/orders/orders/_search?pretty=true' \
-H 'content-type: application/json' \
-d '
{
"size": 0,
"aggs": {
  "order-status": {
    "terms": {
    "field": "status.keyword"
    },
    "aggs": {
      "placedOnMonth": {
          "date_histogram" : {
            "field": "placedOn",
            "interval": "month",
            "format": "YYYY-MM"
           }
      }
    }
  }
}
}
'
```

Nested Aggregations Results

- Expected result (snippet):

  ```
  {
  ...
  "aggregations" : {
      "order-status" : {
        "doc_count_error_upper_bound" : 0,
        "sum_other_doc_count" : 0,
        "buckets" : [ {
          "key" : "pending",
          "doc_count" : 1,
          "placedOnMonth" : {
            "buckets" : [ {
              "key_as_string" : "2017-01",
              "key" : 1483228800000,
              "doc_count" : 1
            } ]
          }
        },
        ...
  ```

- doc_count_error_upper_bound: documents count are approximate as every shard executes its own query
- sum_other_doc_count: count of documents that were not part of the top N buckets returned

Date Histogram Aggregation

- Aggregation that can be applied on date/time field values extracted from the documents
- Builds fixed size buckets based on the interval dynamically:

```
curl -XPOST 'localhost:9200/orders/orders/_search?pretty=true' \
-H 'content-type: application/json' \
-d '
{
"size": 0,
"aggs": {
  "placedOnMonth": {
    "date_histogram" : {
      "field": "placedOn",
      "interval": "month",
      "format": "YYYY-MM"
     }
  }
}
}
'
```

- Results:

```
{
...
"aggregations" : {
  "placedOnMonth" : {
    "buckets" : [ {
      "key_as_string" : "2016-01",
      "key" : 1451606400000,
      "doc_count" : 1
    }, {
      "key_as_string" : "2016-02",
      "key" : 1454284800000,
      "doc_count" : 0
    }, {
      "key_as_string" : "2016-03",
      "key" : 1456790400000,
      "doc_count" : 0
    }, {
...
```

Range Aggregation

- Statically defines buckets for aggregation using value ranges, 0-50, 51-100, 100-*:

```
curl -XPOST 'localhost:9200/orders/orders/_search?pretty=true' \
-H 'content-type: application/json' \
-d '{
"aggs" : {
  "order-amount" : {
    "range" : {
      "field" : "order_amount",
      "ranges" : [
          { "to" : 50 },
          { "from" : 50, "to" : 100 },
          { "from" : 100 }
      ]
    }
  }
}
}'
```

- Result:

```
{
...
"aggregations": {
  "order-amount" : {
    "buckets": [
      {
          "to": 50,
          "doc_count": 2
      },
      {
          "from": 50,
          "to": 100,
          "doc_count": 4
      },
      {
          "from": 100,
          "doc_count": 4
      }
    ]
  }
}
}
```

Aggregation Exercise

- Log-in into your ElasticSearch sandbox
- Make sure elastic search is running:

```
sudo service elasticsearch restart
```

- Populate few sample movie data:

```
curl https://elasticsearch-courseware.icssolutions.ca/examples/data-sets/movies.txt -o movies.txt
curl -XPOST 'localhost:9200/_bulk' -H 'content-type: application/json' --data-binary "@movies.txt"
```

- Confirm there are some records to search on:

```
curl 'localhost:9200/movies/movies/_search?pretty=true'
```

- Let's aggregate on actor name

```
curl -XPOST 'localhost:9200/movies/movies/_search?pretty=true' \
-H 'content-type:application/json' \
-d '
{
"size": 0,
"aggs": {
  "actor_name": {
    "terms": {
      "field": "Starring.CastCrewName.keyword"
    }
  }
}
}'
```

- What buckets did you get?
- How do I get more than 10 buckets? Check StackOverflow posting!
- What is 'sum_other_doc_count' field?
- Let's find out average movie rating for the actor:

Aggregation Exercise

```
curl -XPOST 'localhost:9200/movies/movies/_search?pretty=true' \
-H 'content-type:application/json' \
-d '
{
"size": 0,
"aggs": {
  "actor_name": {
    "terms": {
      "field": "Starring.CastCrewName.keyword"
    },
    "aggs": {
      "rating": {
        "avg": {
          "field": "StarRating"
        }
      }
    }
  }
}
}
'
```

- Take a moment to understand the variety of aggregation types
- Pick an aggregation we did not cover in the slides and make it work

Chapter 4: Summary

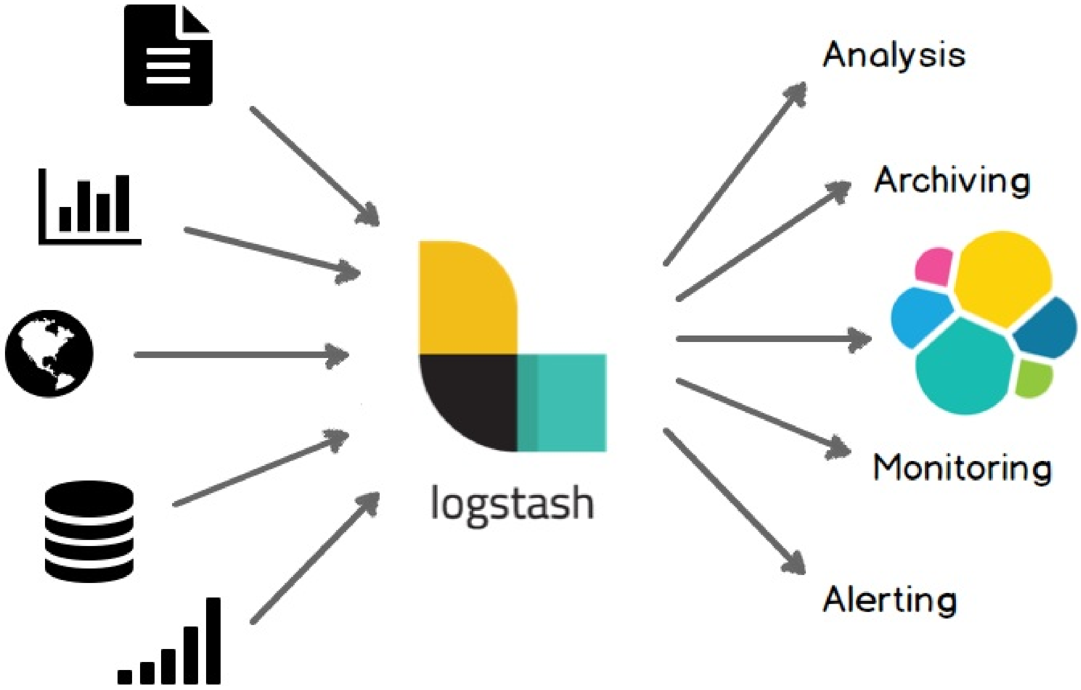
- What's the difference between data aggregation and nested-documents?
- How to establish parent-child relationships?
- How to index child document?
- How to move child document to another parent?
- What happens to child documents when parent is deleted?
- What aggregation types/queries do you recall and when to use them?

Chapter 5: Logstash

- Overview
- Installation, configuration, and usage
- Data analysis and procesing

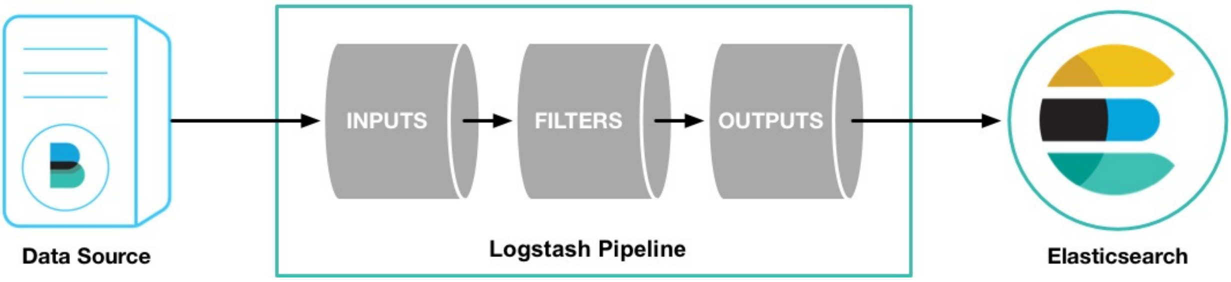
Logstash Overview

- Open source data collection engine
- Built-in pipeline capabilities
- Originally designated for logs collection
- Diverse input/output streams: logs, http requests, webhooks, jdbc, nosql, kafka and more.
- Filters to derive structure out of unstructured data
- Key-value pairs and csv data normalization
- Local lookups and Elasticsearch queries for data enrichment
- Simplifies ingest and data analysis processes



Setup

- Logstash requires Java 8 Oracle or OpenJDK, Java 9 is not supported
- Linux distribution repository is available similary to other Elastic.co products
- A Docker image has been made available by Elastic.co, more about docker later...
- If installing manually - do not install into a directory path that contains colon (:) characters
- Pipeline configuration files define the default Logstash processing pipeline: ALL files in /etc/logstash/conf.d folder
  NOTE: Logstash 6.0 and higher has support for running multiple pipelines which defined in pipelines.yml file.
- Settings files specify options for Logstash startup and execution:
  - /etc/logstash/logstash.yml - configuration flags
  - /etc/logstash/jvm.options - contains JVM configuration flags
  - /etc/logstash/startup.options - used by system-install to create logstash service or services
- Basic pipeline with Logstash



Setup Excercise

- Download and install the Public Signing Key:

  ```
  wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo apt-key add -
  ```

- Add repository definition:

  ```
  sudo apt-get install apt-transport-https
  echo "deb https://artifacts.elastic.co/packages/6.x/apt stable main" | sudo tee -a
   /etc/apt/sources.list.d/elastic-6.x.list
  ```

- Install logstash from repository:

  ```
  sudo apt-get update && sudo apt-get install logstash
  ```

- Create a manual pipeline for Logstash

  ```
    cd /usr/share/logstash
    sudo bin/logstash -e 'input { stdin { } } output { stdout {} }'
  ```

- -e: enable specifying configuration directly from the command line
- stdin: file handle that process reads to get information from you, human
- stdout: process writes log information to this file handle
- After pipeline has started

  ```
  The stdin plugin is now waiting for input:
  ```

  type:

  ```
  Hello World!
  ```

  And press enter
- Expected response:

  ```
    2017-06-20T01:22:14.405+0000 ubuntu Hello World!
  ```

- Logstash adds a timestamp and hostname to the message we sent
- To exit Logstash use keyboard combination ctrl-d

Logs Processing - Input

- Typical pipeline is a bit more than hello-world example: one or more input, filter, and output plugins
- We will configure Logstash to pickup log files and to send data to Elasticsearch
- Log file: /var/log/cloud-init.log file, snippet of data:

```
2017-06-21 22:18:25,270 - helpers.py[DEBUG]: config-keys-to-console already ran (freq=once-per-instance)
2017-06-21 22:18:25,266 - helpers.py[DEBUG]: config-mcollective already ran (freq=once-per-instance)
2017-06-21 22:18:25,276 - util.py[DEBUG]: Reading from /proc/uptime (quiet=False)
2017-06-21 22:18:25,276 - util.py[DEBUG]: Read 10 bytes from /proc/uptime
2017-06-21 22:18:25,276 - util.py[DEBUG]: cloud-init mode 'modules' took 0.102 seconds (0.10)
2017-06-21 22:18:25,276 - handlers.py[DEBUG]: finish: modules-final: SUCCESS: running modules for final
```

- Input configuration:

```
input {
  file {
    path => "/var/log/cloud-init.log"
    start_position => "beginning"
    type => "logs"
  }
}
```

Logs Processing - Grok Filter

- `grok` filter provides parsing of the unstructured log data into something query-able
- Log data:

```
2017-06-21 22:18:25,276 - util.py[DEBUG]: Reading from /proc/uptime (quiet=False)
```

- Predefined match patterns
- Custom parse patterns added:

```
For module: (?<= - )(.+)(?=\[)
For loglevel (?<=\[)(.+)(?=\])
```

- Capture into a field: `%{TIMESTAMP_ISO8601:datetime}` where `datetime` is a field name
- Full capture pattern:

```
%{TIMESTAMP_ISO8601:datetime}%{SPACE}%{SPACE}-%{SPACE} (?<module>(?<= - )(.+)(?=\[
))(\[)(?<loglevel>(.+)(?=\]))(\]: )%{GREEDYDATA:message}
```

- Transformed data (1st log record):

```
{
  "datetime": "2017-06-21 22:18:25,276",
  "module": "util.py",
  "loglevel": "DEBUG",
  "message": "2017-06-21 22:18:25,276 - util.py[DEBUG]: Reading from /proc/uptime
(quiet=False)"
}
```

- Configuration settings:

```
filter {
  grok {
    match=> {
    "message"=>"%{TIMESTAMP_ISO8601:datetime}%{SPACE}%{SPACE}-%{SPACE} (?<module>(
?<= - )(.+)(?=\[))(\[)(?<loglevel>(.+)(?=\]))(\]: )%{GREEDYDATA:message}"
  }
}
```

- Invaluable tools:
  - grok constructor
  - grok debugger
  - regex tester

  - Kibana Grok Debugger (as part of X-Pack)

Logs Processing - Output

- Last phase in the logstash pipeline specifies where the parsed logs should go
- Configuration:

```
output {
elasticsearch {
 hosts => ["localhost:9200"]
 index => "cloud-init"
}
}
```

Log Processing Exercise

- Log-in into your Elasticsearch sandbox
- Make sure elasticsearch is running:

```
curl localhost:9200/_cluster/health?pretty
```

- Create configuration file:

```
sudo nano /etc/logstash/conf.d/cloud-init.conf
```

- Copy-paste settings we have reviewed during previous slide:

```
input {
  file {
    path => "/var/log/cloud-init.log"
    start_position => "beginning"
    type => "logs"
  }
}

filter {
  grok {
    match=> {
      "message"=>"%{TIMESTAMP_ISO8601:datetime}%{SPACE}%{SPACE}-%{SPACE} (?<module>(?<= - )(.+)(?=\[))(\[)(?<loglevel>(.+)(?=\]))(\]: )%{GREEDYDATA:message}"
    }
  }
}

output {
  elasticsearch {
    hosts => ["localhost:9200"]
    index => "cloud-init"
  }
}
```

- Logstash loads all files in the /etc/logstash/conf.d directory, store no extra files there
- Restart logstash service and monitor messages:

```
sudo service logstash start && sudo tail -f -n 100 /var/log/logstash/logstash-plain.log &
```

- Check the data has been populated into elastic search:

```
curl localhost:9200/cloud-init/_search?pretty=true
```

- Expected results is a long list of parsed log events
- New lines added to the log file will be posted into the index

Chapter 5: Summary

- What is the purpose of Logstash product?
- What are the components of Logstash pipeline?
- What filter plug-in we've used to extract data from log file?

Chapter 6: Filebeat

- Introduction to Beats
- Setup of Filebeat
- Filebeat Exercise

Beats

- Lightweight Data Shippers
- Installed as Agents to send data at high speed to Logstash or to ElasticSearch
- Filebeat is offering a lightweight way to forward and centralize logs and files
- Metricbeat collects metrics from systems and services using specific modules
- Packetbeat is a lightweight network packet analyzer that sends data
- Winlogbeat live streams any Windows Event log channel
- Heartbeat asks the simple question from set of defined ICMP, TCP, and HTTP endpoints: "Are you alive?"

Filebeat: Overview

- New in version 5.x
- Moves log processing from external process such as Logstash into the ElasticSearch itself
- Can send the processed logs into ElasticSearch, Logstash, Kafka, Redis and more
- Supports pipeline of data processing, removing fields, adding fields, filtering events, and etc.

Filebeat Setup Exercise

- Download public key for the repository:

```
curl -O https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo apt-key add -
```

- Add repository to the list:

```
echo "deb https://artifacts.elastic.co/packages/5.x/apt stable main" | sudo tee -a
 /etc/apt/sources.list.d/elastic-5.x.list
```

- Install Filebeat from the repository:

```
sudo apt-get update && sudo apt-get install filebeat
```

- Configure the logs location:

```
sudo nano /etc/filebeat/filebeat.yml
```

- Important settings:

```
- input_type: log
  paths:
    - /var/log/*.log

output.elasticsearch:
  hosts: ["localhost:9200"]
  template.enabled: true
  template.path: "/etc/filebeat/filebeat.template.json"
  template.overwrite: false
  index: "filebeat"
```

- Start Filebeat service and check the status:

```
sudo service filebeat start && sudo service filebeat status
```

- Query ElasticSearch using curl to confirm new index has been created: 'filebeat':

```
curl 'localhost:9200/_cat/indices?format=json'
```

- Query the data inside the newly created index:

```
curl 'localhost:9200/filebeat/_search?pretty=true'
```

Chapter 6: Summary

- What are the beats?
- What problem beats are looking to solve?
- Can you enumerate beats and their purpose?
- How to configure Filebeat to pick a specific file or folder?

Chapter 7: Kibana

- Setup and configuration
- Discover
- Visualize
- Dashboard
- Timelion

Setup and Configuration

- Kibana is an open source product installed separately from Elastic Search
- Can be downloaded from Elastic.co or installed using Linux Repositories
- Configuration file: /etc/kibana/kibana.yml
- Important settings:

  ```
  server.port: 5601
  server.host: "0.0.0.0"
  elasticsearch.url: "http://localhost:9200"
  ```

- Additional settings are detailed here
- Kibana can be installed on the same nodes where Elastic Search is running, or on separate nodes
- Kibana stores its configuration data in ElasticSearch
- Accessing Kibana from browser: http://domain-name:5601

Sample Data Setup

- For discovery and visualization we will need some data
- Logs collected by logstash "schema":

  ```
  {
  "@timestamp" : "2015-05-18T12:20:35.324Z",
  "ip" : "250.252.55.241",
  "extension" : "jpg",
  "response" : "200",
  "geo" : {
    "coordinates" : {
      "lat" : 42.10690806,
      "lon" : -111.9125389
    },
    "src" : "CN",
    "dest" : "BR",
    "srcdest" : "CN:BR"
  },
  "@tags" : [ "success", "security" ],
   ...more fields as logs are messy
  }
  ```

Exercise: Setup Kibana

- Log-in into your sand-box
- May need to start your elastic search service:

  ```
  sudo service elasticsearch start
  ```

- From terminal download and install Public Signing Key:

  ```
  curl https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo apt-key add -
  ```

- From terminal add repository definitions:

  ```
  echo "deb https://artifacts.elastic.co/packages/5.x/apt stable main" | sudo tee -a
   /etc/apt/sources.list.d/elastic-5.x.list
  ```

- Update repositories info and install Kibana:

  ```
  sudo apt-get update && sudo apt-get install kibana
  ```

- Start Kibana service:

  ```
  sudo service kibana start
  ```

- Populate sample data:

  ```
  curl https://elasticsearch-courseware.icssolutions.ca/examples/data-sets/logs.json
   -o logs.json
  curl -XPOST 'localhost:9200/_bulk' -H 'content-type: application/json' --data-bina
  ry "@logs.json"
  ```

- By default Kibana listens to localhost and it won't be really helpful in most environments
- Edit kibana.yml to set server.host: 0.0.0.0:

  ```
  sudo nano /etc/kibana/kibana.yml
  ```

- Restart Kibana service:

  ```
  sudo service kibana restart
  ```

- Open browser to http://domain-name:5601, where domain name is the same as for ssh connection

- Kibana requires configuration before it display data: index name pattern is required
- To find out what indices we have in the cluster:

  ```
  curl 'localhost:9200/_cat/indices?format=json'
  ```

- After typing index name pattern with star as a wildcard tab out from the field to get the fields refreshed
- Kibana (by default) requires a date-time field to filter data on
- By default Kibana displays data for the last 15 minutes and in a simulated environment it is often an empty result set
- Look at the top-right corner to adjust the timeframe
- You should be able to see some data now, if not, common troubles are index pattern configuration and a timeframe selection
- There is a star icon at the top of the page to preserve default settings
- Head to the 'Discover' link at the top
- Adjust time-frame in the top-right corner
- Use search box to locate some record
- Select 'add' link next to few fields to present selected fields on the results pane
- Select any record and switch between text and json views
- Save search using link on top of the page
- We will look into other links a bit later...

Discover

- Overview of the data with date-time filter implied
- Search for data using Lucene syntax
- Select available fields for display in tabular format
- Histogram for document count by time interval
- Save settings
- Share settings
- Settings icon next to 'Available Fields' to select fields to add to table
- Individual document json link
- Common pitfalls: date-time filter and limited nested data support

Visualize

- Framework to present your data in aggregated or graphical fashion
- Data table - aggregated data presentation, not mechanism for a data dump
- There is no built-in mechanism to dump data of a search
- Third party plugin for data import/export: knapsack
- Data Table - by default presents count of document found by search
- Can use new or previously saved Discover search
- Two components: metrics and aggregation
- Metrics: a loose equivalent of measures in a fact table from BI (Business Intelligence) world
- Aggregation: a loose equivalent of dimension in OLAP (online analytical processing) world

Data Table Exercise

- On discover tab construct some search with results
- For the visualization we will need combination of numeric, date time, and string data types
- Save the search definitions to reuse in the Visualization
- Switch to Visualize tab
- Select Data Table
- Select saved search or start over with a new search
- Select metrics
- Select aggregations
- Experiment with different options
- Save your definitions - you may want to comeback to them later.

Metric

- Presents a single number based on a search
- Count
- Average
- Max
- Min
- Assuming you have a single important metric to monitor
- Example: max memory utilization or 95% response time percentile

Metric Exercise

- Access Kibana user interface with browser: http://domain-name:5601/
- On discover tab construct some search with results
- For the visualization we will need numeric data type
- Save the search definitions to reuse in the Visualization
- Switch to Visualize tab
- Select Metric
- Select saved search or start over with a new search
- Select aggregation type
- Add couple more metrics with different aggregation types
- Save your definitions - you may want to comeback to them later

Vertical Bar Chart

- Commonly used to present high density data such as logs and a network traffic
- As description on the Kiabana user interface suggests: "if you are not sure which chart you need, you could do worse than to start here"
- I wish it was listed as first option on the Visualize tab
- Metrics are similar to Data Table and Metric visualization - numeric value to present
- Buckets are similar to aggregation in Data Table - a way to partition the data
- Two options to consider: Split Bars or Split Chart
- Not extremely intuitive user interface and/or configuration options

Vertical Bar Chart Exercise

- Access Kibana user interface with browser: http://domain-name:5601/
- On discover tab construct some search with results
- For the visualization we will need numeric data type and some other field for aggregation
- Save the search definitions to reuse in the Visualization
- Switch to Visualize tab
- Select Vertical Chart Bar
- Select saved search or start over with a new search
- Add metrics or modify one that comes by default
- Select X-Axis and pick some field(s) for grouping
- Experiment with Split Bars and with Split Charts
- Save your definitions - you may want to comeback to them later

Tag Cloud

- New for version 5.x, previously a 3rd party kibana plugin
- Collection of words, terms or small phrases, laid out all adjacent to each other
- Descending order used to select most frequent metric
- Ascending order can be used to detect abnormalities
- The size of the tags corresponds to their importance based on selected metric, e.g. count
- Aggregation available: terms
- Number of tags to display is configurable
- Options tab provides few more choices to customize the display

Coordinate Map

- Previously 'Tile Map'
- Possibly the most interesting visualization
- Requires mappings to be configured with geo_point type
- For previously populated log data cannot modify mapping for coordinates
- We will have to delete the indices created, re-create indices with mapping, and to re-populate data
- Shell script prepared for you right-click and save
- Or download the script to the sandbox by running following command:

  ```
  curl -o geoMappings.sh https://elasticsearch-courseware-2d.icssolutions.ca/examples/data-sets/geoMappings.sh
  ```

- `chmod +x` will be required to grant execution permission
- Execute the script:

  ```
  ./geoMappings.sh
  ```

- Go back to Visualize tab and select 'Coordinate map'
- Here it is a bit tricky - may need to go back to settings and refresh the fields
- Select 'Geo Coordinates' and select 'geo.coordinates' field
- Select 'green arrow' on top to apply changes and explore zoom in/out
- Switch to advanced options and check-out wms
- There are terms of service for the maps service apparently separate from the elastic search product, go figure...

Dashboard

- Dashboard combine previously created (and saved) searches and visualizations
- Panels can be resized and re-arranged on the dashboard
- Searches, Visualization, and Dashboards can be shared via link and embedded in an IFrame
- There is an option to build a custom visualization
- There is an impressive list of ready-to-use plugins
- Some inspiration for your exercise.

Timelion

- New for Kibana version 5.x, pronounced "Timeline"
- Timelion is an Elastic {Re}search project into time series
- {Re}search projects are launched by Elastic engineers that want to grab an idea, run with it, and share it
- Brings together totally independent data sources into a single interface
- Driven by a simple, one-line expression language
- Combined data retrieval, time series combination and transformation, plus visualization
- Pulling data from ElasticSearch: .es(*)
- Can pull data from from external source World Bank

Timelion Exercise

- Access Kibana user interface with browser: http://domain-name:5601/
- Select Timelion on the left hand navigation bar
- Switch to full screen using the icon on the chart
- In the formula area type: `.es(*)`
- Select time range of last 5 years
- Select play button to update the presentation to present a typical time-series chart
- Zoom-in to more active time period of the chart before next steps
- In the formula area extend the expression to `.es(*).derivative()`
- Select '1h' for the time interval from drop-down list
- And select the play button again - variation for number of events on hourly basis will be presented
- Modify the formula: `.es(*), .es(*,offset=-1h)` to compare count of events to previous hour and select play button
- Add some custom color to the time series: `.es(*), .es(*,offset=-1h).color(yellow)` and select play button
- Convert line to bars: `.es(*), .es(*,offset=-1h).bars().color(yellow)` and select play button again
- Add secondary data source to correlate number of events in log files to population of Israel: `.es(*).bars(), .wbi(country=ISR).divide(100000)`
- Adding another metric to the chart: `.es(), .es(metric=max:memory).divide(100)`
- You are getting the rough idea of a potential...

Console

- IntelliSense for Elastic Search Dsl
- A move up from curl to write and test ElasticSearch queries
- Sometime ago was a Kibana Plug-in
- Before that known as Sense Chrome Extension
- Elastic blog reflects on Sense history
- Has a stand-alone twin

Chapter 7: Summary

- What's Kibana?
- How to install Kibana?
- How many servers to allocate to Kibana?
- What are the components in Kibana
- What are the common pitfalls using searches?

Chapter 8: Watcher

- One of the X-Pack feature
- Encourages integration and automation for a wide range of use-cases:
  - Monitor your infrastructure
  - Track network activity
  - Monitor health of Elasticsearch cluster/node/index
- Gives you the power of the Elasticsearch DSL to identify changes in your data
- Create notifications when:
  - The same user logins from 4 disperse geographical locations in 10 min
  - Frequency of request for a single ip address spikes 1,000% in last hour
  - Elasticsearch cluster is experiencing increased exceptions rate in the logs

X-Pack Exercise

1. X-Pack is installed as a part of Kibana and ElasticSearch 6.4.2
2. Navigate to http://ip-address:5601
3. Select 'Monitoring' link and activate monitoring
4. Browse through ElasticSearch cluster, nodes, and indices charts available
5. Execute some queries against ElasticSearch to see the charts update
6. Browse through Kibana monitoring and notice charts available
7. Feel advan

Watcher Components

- Simple User interface accessible from Kibana to create, view, and manage alerts
- Alert history is stored inside Elasticsearch for Kibana visualization and historical analysis
- Watcher is definition of an automated action to be taken when a certain condition is met
- Action can be a webhook to an external system, an email message, a slack message, and more...

Watcher Definitions

- Trigger - determines when the watch is checked
- Input - data to be loaded into the context, if not defined empty payload will be loaded into the context:
    - Simple - loads static payload
    - Search - loads elastic search results
    - Http - loads result of Http Request
    - Chain - uses series of inputs to load data
- Conditions - when a watch is triggered:
    - Always - actions are always executed
    - Never - actions are never executed
    - Compare - compare against values
    - Array Compare - compare an array of values
    - Script - use a script to evaluate condition
- Transform - process watch payload before passing it on to watch actions
- Actions - what happens when conditions are met

Watcher Exercise

- Double-check Elasticsearch, Filebat and Kibana are running:

```
sudo systemctl start elasticsearch && sudo systemctl start filebeat && sudo systemctl start kibana
```

- Navigate to http://ip-address:5601/
- Login with `elastic` user credentials
- Navigate to Management -> Elasticsearch -> Watcher
- Select 'Create new watch' -> 'Advanced Watch' to create new Watch
- First is trigger, let's set interval to 30s
- Second is a query to execute, duplicate browser tab and use Dev Tools for composing query, e.g.:

```
{
"size": 10,
"query": {
  "bool": {
    "filter": {
      "query_string": {
        "query": "@timestamp:[now-1h TO now]"
      }
    }
  }
}
}
```

- Execute the query to make sure it produces results
- Replace `input.search.request.body` portion of the watcher configuration with the query tested in the console
- Modify `indices` portion of the input to list: `filebeat*`
- Review condition portion of the json configuration
- Type-in new watch id and name
- Simulate the new watch to review results
- And save the new watch
- Give it 30 secs to fire
- Now how do we make use of the results?
- First find what index stores the data
- Then define a new index pattern using the Management link on the left
- Use discover to explore the data
- Proceed to Visualization tab to present results

- Please share your findings and visualization selection with others
- Explore other actions available in the watcher

Chapter 8: Summary

- What features of X-Pack can you name?
- What are the functions of Watcher?
- What are components of Watcher configuration?
- What actions type are available in Watcher?

Chapter 9: Running in a Container

- High-level Introduction to Docker
- Composing Docker image for Elasticsearch
- Running Docker container
- Container Specific Considerations

Introduction to Docker

- Eliminates 'works on my machine' by escaping matrix dependencies
- Run applications side-by-side with predictable and portable deployments
- Achieves higher compute density - unlike VMs only libraries and settings are required
- Maintain immutable and tamper-proof environments, and docker hub is scanning the images
- Built-in orchestration to scale up to tens of thousands of nodes in seconds using Docker Swarm

Docker Setup Exercise

- Install using Ubuntu repositories:

  ```
  sudo apt install docker.io -y
  ```

- Alternatively, by downloading from download.docker.com
- Checking that docker cli (command line interface) is installed:

  ```
  docker --version
  ```

- Checking that docker daemon is running

  ```
  sudo docker images
  ```

- Expected output is an empty list as we have not created/pulled any images

- Running your first container:

  ```
  sudo docker run hello-world
  ```

- Expected output:

  ```
  78445dd45222: Pull complete
  Digest: sha256:c5515758d4c5e1e838e9cd307f6c6a0d620b5e07e6f927b07d05f6d12a1ac8d7
  Status: Downloaded newer image for hello-world:latest
  Hello from Docker!
  ... more text comes here...
  ```

- List your container:

  ```
  sudo docker ps -a
  ```

- Expected output is a list with one item on it

Creating a Docker Image & Running a Container

- Docker images are build using Dockerfile definitions
- E.g. hello-world image:

  ```
  FROM scratch
  COPY hello /
  CMD ["/hello"]
  ```

- Where 'scratch' is name:tag of base image, in this case most basic image 'scratch:latest'
- 'COPY' command copies executable file 'hello' to the root of file system
- 'CMD' is command executed when container is running
- Official Elastic Search Dockerfile is not as simple
- To run official image:

  ```
  sudo docker run -d -p 9201:9200 -e "http.host=0.0.0.0" -e "transport.host=127.0.0.
  1" docker.elastic.co/elasticsearch/elasticsearch:5.4.1
  ```

- -d: run in a detached mode
- -e: define listener ip for API and for cluster communication
- -p: map host port to container port

ElasticSearch Container Exercise

- Login into your sandbox
- Shutdown host services to release memory:

```
sudo service elasticsearch stop
sudo service kibana stop
sudo service filebeat stop
sudo service logstash stop
```

- Since previously installed Elastic is running, we will use a different port: 9201 for api

```
sudo docker run -d -p 9201:9200 -e "http.host=0.0.0.0" -e "transport.host=127.0.0.1" docker.elastic.co/elasticsearch/elasticsearch:5.4.1
```

- -d: run detached not to take over command prompt
- -e: parameters accepted/expected by the image
- Give it few moments to start and then test:

```
curl elastic:changeme@localhost:9201
```

- Expected output:

```
{
  "name" : "3kMnHFQ",
  "cluster_name" : "docker-cluster",
  "cluster_uuid" : "E7Ag9a_1SX2FVPsMQGUCng",
  "version" : {
    "number" : "5.4.1",
    "build_hash" : "2cfe0df",
    "build_date" : "2017-05-29T16:05:51.443Z",
    "build_snapshot" : false,
    "lucene_version" : "6.5.1"
  },
  "tagline" : "You Know, for Search"
}
```

- Using docker cli to list containers:

```
sudo docker ps -a
```

- Expected output is list with two containers: hello-world and elastic
- To stop running container, copy container id to replace in the command below:

```
sudo docker stop 8ed07e81b59c
```

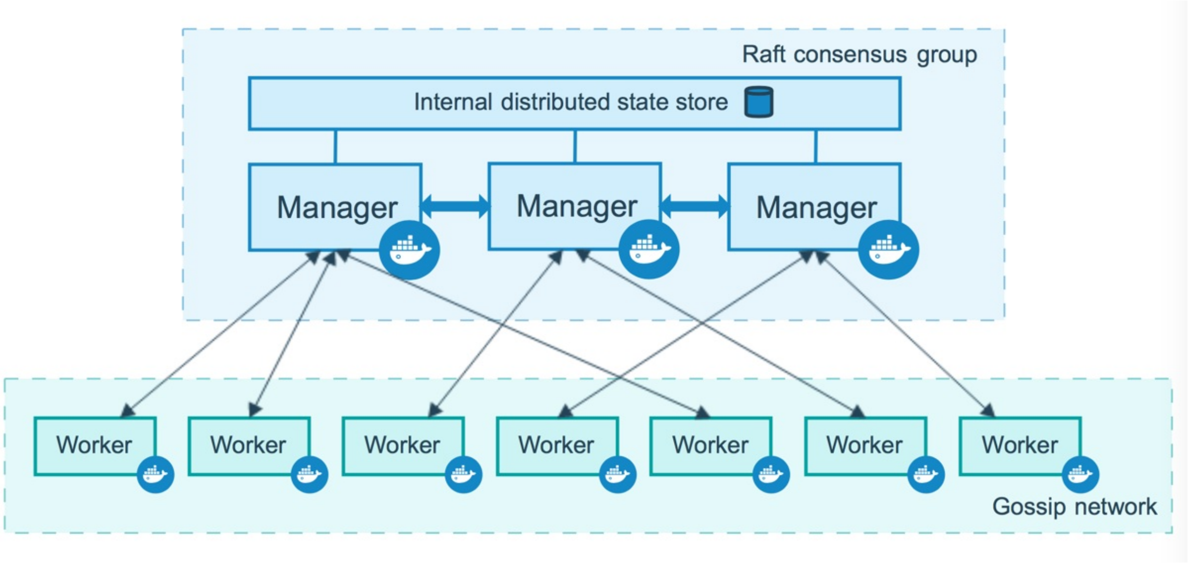
- Using docker cli to list containers to confirm none are running:

  ```
  sudo docker ps -a
  ```

- Expected output is a list of two containers previously launched: hello-world and elasticsearch

Docker Cluster

- In our exercise we have used single docker host
- For production we rather use multiple hosts to run our containers
- Docker Swarm provides orchestration up-to 1,000 nodes and 30,000 containers



- docker-compose orchestrates launching multiple containers at once using docker-compose.yml
- In addition to Docker Swarm there is also Google Kubernetes, Apache Mesos, and likely others
- Comparison from techgenix
- In addition to self-hosted version of docker there are Aws Cluster Service, Google Container Engine, and recently Docker Cloud

Chapter 9: Summary

- What's Docker?
- How can you create a Docker image?
- How do you run a Docker image?
- How do you scale from one Docker host to many?
- What are the cloud hosting options for Docker Images?

Chapter 10: Preparing for Production

- Discussion on capacity planning and data population
- Performance tuning and monitoring
- Hosted Elastic Search

Capacity Planning

- How many nodes?
- The basic starting point is two nodes with 2 cores and 4GB of memory on each node
- For fault tolerance perspective three nodes is more appropriate for any cluster
- What's better more nodes or bigger nodes?
- More nodes equals IO, Memory, and GC (garbage collector) distributed processing
- Common pitfall with distributed databases - stressing common storage e.g. SAN (system attached storage)
- Bigger nodes means more processing can be performed on a node with fast access to in-memory data and faster local IO
- Resizing node in production is likely more challenging than adding a new node to the cluster
- Elasticsearch is built for scaling out on commodity hardware, not up on single massive machine
- How high can it go? Pretty high
- So which one it is going to be: more smaller nodes or less larger nodes?

Memory Planning

- If I plan to store X documents with Y KB in size - what's the formula for memory requirements?
- Search engines including ElasticSearch perform the best when data and index are cached in memory
- How many aggregate queries vs. filtered queries vs. scan queries would you run?
- How many request per second of each type are you planning to support?
- How often will you be taking full snapshots and incremental snapshots?
- How much memory is enough memory?
- ElasticSearch is not a database per ce, hence it 'does not look before it leaps...'
- Approach recommended by Elastic: "start out with more memory than you need and scale down to find the sweet spot"
- Do not enable memory disk swap - it is deadly for ElasticSearch performance
- Swap file might need to be disabled for other NoSql as well e.g.:Cassandra

CPU Planning

- Indexing takes CPU
- Querying takes CPU
- Aggregation takes CPU
- Badly constructed queries take more CPU and that's from personal experience
- RegEx queries burn more CPU
- Experiment before you conclude is probably the accurate answer...

Cluster State

- With ElasticSearch it is easy to create a lot of indices, nodes, and lots and lots of shards
- Cluster State contains all of the mapping for every index and every field in the cluster
- Loggly.com shared a story where they have a cluster with 900MB of cluster state
- Cluster State is stored and replicated between every node in the cluster
- Since ElasticSearch 2.0, cluster state changes have been sent as differences rather than complete state

Network Capacity

- I have had a great experience of taking non-prod environment down at shomi.com
- By running a lot of elastic search and index requests
- Some people still grimace when they see me
- It was not the bandwidth
- Number of sockets opened on the server, on the client, on the network appliance
- There was a bug in firmware that would crash network appliance under specific type of load
- There were subsequent tests where client reached tens of thousands of open tcp/ip sockets pending
- Server was alright, client running heavy batch job was not

Performance Testing

- ElasticSearch exposes RESTfull Api and any suitable tool can be used
- Postman Chrome Extension and Mac X versions available
- Github repository for ElasticSearch stress testing
- Apache Software Foundation JMeter great tool for more than just Https testing. A bit difficult to start with
- For Mac you can install JMeter using brew
- For Windows you can install JMeter using Chocolatey
- Otherwise download JMeter to your desktop
- Follow (patiently) instructor's setup of the test

Performance Monitoring

- A variety of APIs let you manage and monitor the cluster
- Can access a large number of statistics through the API
- In the terminal run:

```
curl localhost:9200/_cat
```

- Expected response:

```
=^.^=
/_cat/allocation
/_cat/shards
/_cat/shards/{index}
/_cat/master
/_cat/nodes
/_cat/indices
/_cat/indices/{index}
/_cat/segments
/_cat/segments/{index}
/_cat/count
/_cat/count/{index}
/_cat/recovery
/_cat/recovery/{index}
/_cat/health
/_cat/pending_tasks
/_cat/aliases
/_cat/aliases/{alias}
/_cat/thread_pool
/_cat/plugins
/_cat/fielddata
/_cat/fielddata/{fields}
/_cat/nodeattrs
/_cat/repositories
/_cat/snapshots/{repository}
```

Monitoring Plug-ins

- ElasticSearch Cluster and Index RESFull Api allowed a lineup of plug-ins
- X-Pack: monitoring and cluster management dashboard by Elastic
- Allows to monitor multiple clusters health, indices, and individual nodes
- Head another commonly mentioned monitoring and management plugin
- Head's user interface is a bit dated but it appears to require no license
- BigDesk comes with more appealing user interface
- BigDesk doesn't appear to require a license and similar to head is open source
- ElasticHQ runs in browser and is connecting to ElasticSearch Api, requires adding cors support to /etc/elasticsearch/elasticsearch.yml:

  ```
  http.cors.allow-origin: "*"
  http.cors.enabled: true
  ```

- Keep in mind 3rd party plugins are usually ElasticSearch and Kibana version specific

Performance Tuning

- Start with configuration of the cluster and its nodes: data, master, and client nodes
- Guesstimate usage pattern: lots of searches, vs. lots of indexing, vs. lots of aggregation
- OS Configuration: number of open file descriptors shall be set to at least 32K-64K
- OS Configuration: disable memory swapping (not always possible in virtual environment)
- Mapping and data-modeling selection: _all, _source, multi-fields
- Search vs. Filter - do you need to compute score based on the 'status:active' in your query?
- Use _bulk Api for indexing whenever possible
- Number of replicas and number of shards - reliability and scaling-out

Indices vs. Types

- ElasticSearch allows multiple types in the same index
- Shards are allocated per index not per type
- Mapping must not conflict within an Index not within a Type
- Smaller shards are easier to distribute evenly between nodes
- Aliases and multi-index search facilitates in searching across multiple indexes
- Replicas help improving query performance by distributing data retrieval

Query Choices

- Avoid wild-card and regex queries
- Using full-text and non-analyzed searches on the same field - use multi-fields
- Return fields required only rather than all fields:

```
curl 'localhost:9200/ordering/order/_search?pretty=true' -d '
{
  "fields": ["planedOn"]
}'
```

- Use filter whenever possibly to suppress scoring:

```
curl -XPOST 'localhost:9200/ordering/order/_search?pretty=true' -H 'content-type: application/json' -d '
{
 "query": {
    "bool": {
       "filter": {
          "term": { "status": "pending" }
       }
    }
 }
}'
```

Tuning References

- Microsoft Azure specific but with lots of useful tips
- Loggly.com performance lessons - a bit logs specific, but volumes and throughput perspective is very valuable
- Codecentric - a bit dated, may points are still relevant
- O'Reilly article - very systematic and detailed.

Data Population

- ElasticSearch is best as a secondary store
- Bulk Api is a must for most applications
- Payload for bulk load is a bit less than self-explanatory
- It is json but it is not proper json, first line is meta-data and second line is Json data
- The json on each line must be on one line, between two lines '\n' character is expected
- '\n' character is expected after last line as well
- Example using curl requires creating a file with not-so-json payload: 'request' first:

```
echo '{ "index" : { "_index" : "ordering", "_type" : "order", "_id" : "1" }}
{ "id" : "1", "placedOn": "2016-10-17T13:03:30.830Z" }
' > request
```

- Posting request data requires --data-binary switch to instruct curl to post the data as is

```
curl -s -XPOST localhost:9200/_bulk -H 'content-type: application/json' --data-bin
ary "@request"
```

Data Synchronization

- Previous versions of ElasticSearch advocated concept of rivers
- Rivers would be installed as plug-in into ElasticSearch
- River would pull data automatically from the source database
- Article published by ElasticSearch suggests that rivers are now deprecated
- Some NoSql vendors implemented plug-ins for data population and synchronization e.g.: Couchbase
- Some third parties implemented plug-ins for Cassandra
- Client specific Sdk tend to support bulk api natively: Node.js, Java, Python, and etc.

Data Re-Indexing

- Every so often you may run into changes required in mapping for existing fields
- As per our previous attempts updating mapping for existing fields is not possible
- The only solution I am aware of: create new index with new mapping and move data over
- To move data reliably in ETL (extract, transform, and load) like fashion - scan API is useful
- Initiating a scroll:

  ```
  curl 'localhost:9200/logstash-2015.05.20/log/_search?scroll=1m'
  ```

- To continue with scroll until no records left:

  ```
  curl -XGET  'localhost:9200/_search/scroll'  -d '
  {
    "scroll" : "1m",
    "scroll_id" : "c2Nhbjs2OzM0NDg1ODpzRlBLc0FXNlNyNm5JWUc1"
  }
  '
  ```

- What do you do with the data you receive?
- How do you plan for re-indexing possibility without affecting ElasticSearch clients?

Hosted ElasticSearch

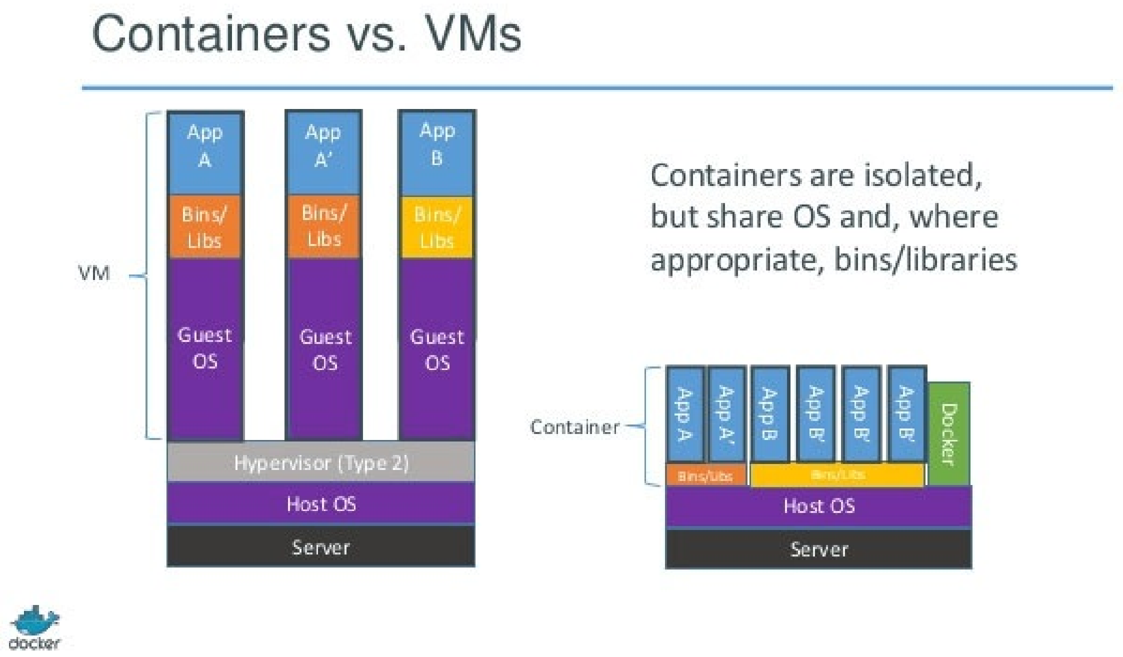
- Number of hosted providers available is growing, maybe reflective of interest
- Elastic.co acquired Found.io and now offers Elastic stack as a service
- QBox offers ElasticSearch across multiple cloud providers
- Bonsai.io offers data encrypted at rest for ElasticSearch
- http://www.searchly.com/ - I use it for ElasticSearche proof-of-concepts as it offers free tier
- Amazon ElasticSearch Service requires AWS subscription, a tad tedious to setup and to connect

Hosted vs. Self-Managed

- Using a self-managed cluster appears cheaper as far as monthly quoted costs go
- Self-managed clusters allows greater control over configuration settings
- Greater control over access to data
- Costs for self-managed option should take operations effort in consideration
- Hosted ElasticSearch is likely operated by ElasticSearch specialists than enterprise operation generalists
- Ideally hosted option eliminates the need for over-planning
- Aws hosted ElasticSearch asks for number and type of ec2 instances to run on
- Bonsai.io pre-sale engineers asking for data volumes and usage patterns instead
- Sample quote from Bonsai.io: $2,950/mo for 1.5TB of raw data coming at 50kbps for index-mostly usage pattern
- Compare that to cost of self-management: hardware and ongoing operation cost

VM vs. Container

- The VM model blends an application, a full guest OS, and disk emulation
- Container model uses just the application's dependencies and runs them directly on a host OS



Containers vs. VMs

Containers are isolated, but share OS and, where appropriate, bins/libraries

- Container model reduces the overhead associated with starting and running instances
- The same host can host 10-15 VM's compared to dozens-hunders containers
- The isolation from OS kernel provided by containers is less robust than that of virtual machines
- Some advocate leaving the docker containers
- Some provide detailed walk-through on how to embrace ElasticSearch in containers
- For me, I would test the two approaches for a specific deployment as reality is never what it seems

Optional Data Population Exercise

- Background of participants is quite different
- Data population most valuable when you have data you would like to test-drive
- It is up-to you whether you would like to engage in such an open ended exercise

Chapter 10: Summary

- What are the considerations for capacity planning?
- What are pressures on memory?
- What are requirements for storage?
- What are anticipated impacts on Cpu?
- What is Cluster State and what impact it may have on performance?

Chapter 11: Running in Production

- Installation, configuration, and hardware
- Monitoring and alerts

Installation

- Deb/Rpm repositories seems like a logical choice
- 'yum/apt-get upgrade' will update version of ElasticSearch
- Great when you don't have plug-ins installed
- Plug-ins have a version specified in their configuration
- After 'apt-get upgrade' you may end-up with plug-ins disabled

Configuration

- Review node types, number of shards, and number of replicas
- With heavy loads master nodes, client nodes, and data nodes maybe best separate
- Log location vs. data location
- Data volumes - encrypted or not
- ElasticSearch defaults to anonymous read/write access
- Maybe you don't want your cluster on this list
- Network protection anyone?
- Don't trust humans - automate configuration and deployment

Hardware

- ElasticSearch built to scale out on commodity hardware not to scale up on monster hardware
- SAN: all nodes will read and write using the same hardware, single point of failure
- SSD vs. HDD: same as with many I/O intensive systems - SSD is faster, not safer
- Have not heard of specialty ElasticSearch hardware yet
- There is no specific hardware mentioned in ElasticSearch documentation

Monitoring

- Plug-ins from Elastic.co and from third parties: Marvel, ElasticSearchHQ, BigDesk, others
- Since everything in ElasticSearch is RESTFul Api: may choose to build own monitoring
- Aws offers dashboards to monitor EC2 instances and Paas/Daas
- Azure offers monitoring tools
- DataDog is $15/node monitoring
- OpsView monitoring for on/off premises instances

Alerting

- ElasticStack 5.0 offers license for X-Pack that includes Alerts
- ElastAlert is a simple framework for alerting on anomalies, spikes, and more
- Aws CloudWatch comes with alerts for EC2 instances and PaaS/DaaS services
- Cabot - monitor and alert
- Can built our own thanks to RESTFull Api
- Automated remediation instead of alerting

Chapter 11: Summary

- What installation option would you prefer?
- What logic you are odd to use determining desired cluster configuration?
- What hardware is preferred for ElasticSearch cluster?
- What monitoring and alerting options available for ElasticSearch?

Chapter 12: What have we learned?

- What are the design differences between NoSql and Rdbms?
- What is the principal difference between database and search engine?
- How would you compare data modeling between Rdbms and NoSql and SearchEngine?
- What data modeling options we have discussed to replace Rdbms joins?
- What are components of ElasticSearch cluster?
- How do we query ElasticSearch?
- What's the different between a search and a filter?
- What aggregation types do you recall?
- What data ingest options available from Elastic.co?
- What are three stages of data ingest with Logstash?
- What types of Beat do you recall?
- What user interface product available for ElasticSearch?
- What are the roles of Discovery, Visualization, and Dashboards?
- What are the consideration for capacity planning?

ви благодариме
nirringrazzjak
شكرا
təşəkkür
köszönöm
gràcies
спасибо
grazas
eskerrik asko
gratias agimus tibi
谢谢
kiitos
paldies
cảm ơn bạn
di ou mèsi
dank je
tack
dankon
danke
gracias
dankie
aitäh
þakka þér
ありがとう
thank you
dank u
mochchakkeram
teşekkür ederim
σας ευχαριστώ
ngiyabonga
grazie
merci
благодаря
תודה
dziękuję
дзякуй
go raibh maith agat
děkuji
감사합니다
asante
hvala
kop khun krap
ačiū
धन्यवाद
terima kasih
takk
хвала
sukriya
ขอขอบคุณคุณ
buíochas a ghabháil leat
salamat
d'akujem
diolch i chi
mulțumesc
obrigado
falemnderit

Dynamic Templates

- In-between option between dynamic mapping and strict mapping
- Example:

```
"mapping": {
  "audit-events": {
      "dynamic_templates": [
          {
              "id": {
                  "match":   "*ZZIDZZ",
                  "mapping": {
                      "type": "string",
                      "index": "not_analyzed"
                  }
              }
          },
 ...
}
```

- Detailed Documentation