elastic

search as
you type

search- as-you-type

```
curl -XGET '127.0.0.1:9200/movies/movie/_search?pretty' -d '
{
    "query": {
        "match_phrase_prefix": {
            "title": {
                "query": "star trek",  "slop": 10
            }
        }
    }
}'
```

index with N-grams

“star”:

unigram: [s, t, a, r]
bigram: [st, ta, ar]
trigram: [sta, tar]
4-gram: [star]

edge n-grams are built only on the beginning of each term.

indexing n-grams

Create an “autocomplete” analyzer

filter for edge n-grams
min = 1
max = 20

custom analyzer, in addition to standard lowercase filter also has autocomplete filter (n-grams)

```
curl -XPUT '127.0.0.1:9200/movies?pretty' -d '
{
    "settings": {
        "analysis": {
            "filter": {
                "autocomplete_filter": {
                    "type": "edge_ngram",
                    "min_gram": 1,
                    "max_gram": 20
                    }
                },
                "analyzer": {
                    "autocomplete": {
                        "type": "custom",  "tokenizer": "standard",
                        "filter": ["lowercase",
                        "autocomplete_filter"
                        ]
                    }
            }
        }
    }
}'
```

map your field

```
curl -XPUT '127.0.0.1:9200/movies/_mapping/movie?pretty' -d '
{
    "movie": {
        "properties" : {
            "title": {
                "type" : "string",
                "analyzer": "autocomplete"
            }
        }
    }
}'
```

title is of type "string" and uses our
custom analyzer "autocomplete

n-grams only on index

Use n-grams only on the index side or query will also get split into n-grams, and we'll get results for everything that matches 's', 't', 'a', 'st', etc.

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

analyzer = standard so don't split up what was typed into n-grams.

completion suggesters

You can also upload a list of all possible completions ahead of time using completion suggesters.

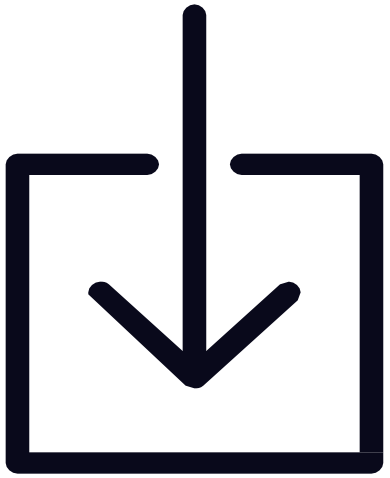
- Most customizable
- Reliable results
- Most control

lab: Query tips

- Lab 10: Pagination, sorting, filtering and fuzzy matching
- Lab 11: Prefix/wildcard and auto-completion

importing data

importing data



stand-alone scripts can submit bulk documents via REST API

logstash and beats can stream data from logs, S3, databases, and more

AWS systems can stream in data via lambda or kinesis firehose

kafka, spark, and more have Elasticsearch integration add-ons

importing

via script / json

python import

- read in data from some distributed filesystem
- transform it into JSON bulkinserts
- submit via HTTP / REST to your elasticsearch cluster

```
import csv
import re

csvfile = open('ml-latest-small/movies.csv', 'r')

reader = csv.DictReader( csvfile )
for movie in reader:
        print ("{ \"create\" : { \"_index\": \"movies\", \"_type\": \"movie\", \"_id\" : \"" , movie['movieId
        title = re.sub(" \(.*\)$", "", re.sub('"','', movie['title']))
        year = movie['title'][-5:-1]
        if (not year.isdigit()):
            year = "2016"
        genres = movie['genres'].split('|')
        print ("{ \"id\": \"", movie['movieId'], "\", \"title\": \"", title, "\", \"year\":", year, ", \"genre
        for genre in genres[:-1]:
            print("\"", genre, "\",", end='', sep='')
        print("\"", genres[-1], "\"", end = '', sep='')
        print ("] }")
```

python example:

```
import csv
import re

csvfile = open('ml-latest-small/movies.csv', 'r')

reader = csv.DictReader( csvfile )
for movie in reader:
    print ("{ \"create\" : { \"_index\": \"movies\", \"_type\": \"movie\", \"_id\" : \"" , movie['movieId'], "\" } }", sep='')
    title = re.sub(" \(.*\)$", "", re.sub('"','', movie['title']))
    year = movie['title'][-5:-1]
    if (not year.isdigit()):
      year = "2016"
    genres = movie['genres'].split('|')
    print ("{ \"id\": \"", movie['movieId'], "\", \"title\": \"", title, "\", \"year\":", year, ", \"genre\":[", end='', sep='')
    for genre in genres[:-1]:
      print("\"", genre, "\",", end='', sep='')
    print("\"", genres[-1], "\"", end = '', sep='')
    print ("] }")
```

importing

via client api’s

client libraries

free elasticsearch client libraries are available for pretty much any language.

- java has a client maintained by elastic.co
- python has an elasticsearch package
- elasticsearch-ruby
- several choices for scala
- elasticsearch.pm module for perl

You don't have to wrangle JSON.

```
es = elasticsearch.Elasticsearch()

es.indices.delete(index="ratings",ignore=404)
deque(helpers.parallel_bulk(es,readRatings(),index="ratings",doc_t
es.indices.refresh()
```

python full script

- function to read Movies
  - open csv
  - build dictionary
  - match movieid with title
- function to read Movies
  - open csv
  - build dictionary
  - match movieid with title

- create ES instance,
- delete ratings table
- load ratings into "ratings" index.

```
import csv
from collections import deque
import elasticsearch
from elasticsearch import helpers

def readMovies():
    csvfile = open('ml-latest-small/movies.csv', 'r')
    reader = csv.DictReader( csvfile )
    titleLookup = {}

    for movie in reader:
        titleLookup[movie['movieId']] = movie['title']
    return titleLookup

def readRatings():
    csvfile = open('ml-latest-small/ratings.csv', 'r')

    titleLookup = readMovies()

    reader = csv.DictReader( csvfile )
    for line in reader:
        rating = {}
        rating['user_id'] = int(line['userId'])
        rating['movie_id'] = int(line['movieId'])
        rating['title'] = titleLookup[line['movieId']]
        rating['rating'] = float(line['rating'])
        rating['timestamp'] = int(line['timestamp'])
        yield rating

es = elasticsearch.Elasticsearch()

es.indices.delete(index="ratings",ignore=404)
deque(helpers.parallel_bulk(es,readRatings(),index="ratings",doc_type="rating"), maxlen=0)
es.indices.refresh()
```

lab: python scripts

- Lab 12: Python scripts to import data
- We will also do the exercise on the next slide.

python - tags

exercise

write a script to import the tags.csv data from ml-latest-small into a new “tags” index.

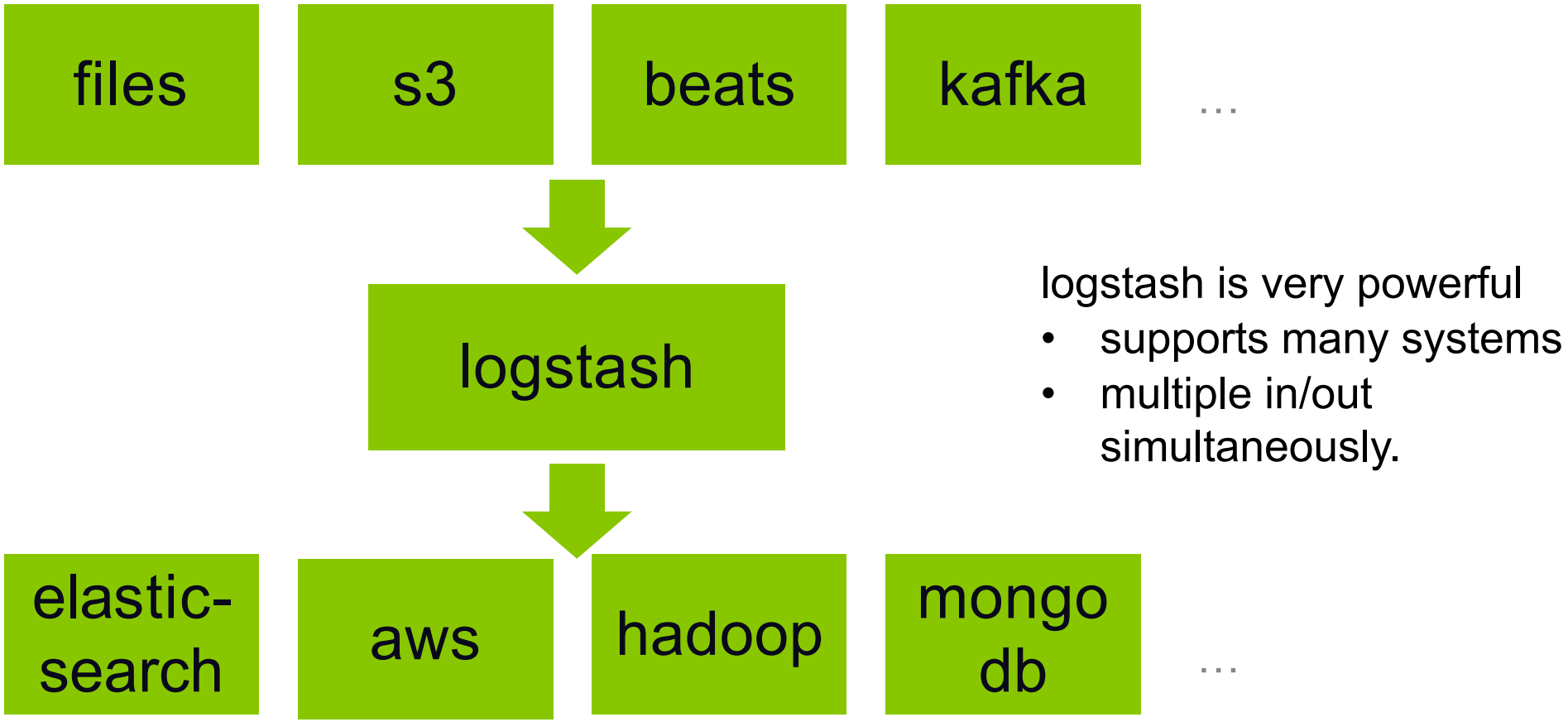
one solution

- tag = line ['tag'] instead of float line (no longer a number)

```
import csv
from collections import deque
import elasticsearch
from elasticsearch import helpers

def readMovies():
  csvfile = open('ml-latest-small/movies.csv', 'r')
  reader = csv.DictReader( csvfile )
  titleLookup = {}

  for movie in reader:
      titleLookup[movie['movieId']] = movie['title']
  return titleLookup

def readTags():
  csvfile = open('ml-latest-small/tags.csv', 'r')

  titleLookup = readMovies()

  reader = csv.DictReader( csvfile )
  for line in reader:
      tag = {}
      tag['user_id'] = int(line['userId'])
      tag['movie_id'] = int(line['movieId'])
      tag['title'] = titleLookup[line['movieId']]
      tag['tag'] = line['tag']
      tag['timestamp'] = int(line['timestamp'])
      yield tag

es = elasticsearch.Elasticsearch()

es.indices.delete(index="tags",ignore=404)
deque(helpers.parallel_bulk(es,readTags(),index="tags",doc_type="tag"), maxlen=0)
es.indices.refresh()
```

introducing logstash

logstash



logstash is very powerful

- supports many systems
- multiple in/out simultaneously.

Powerful features

- logstash parses, transforms, and filters data as it passes through.
- it can derive structure from unstructured data
- it can anonymize personal data or exclude it entirely
- it can do geo-location lookups
- it can scale across many nodes
- it guarantees at-least-once delivery
- it absorbs throughput from load spikes

See https://www.elastic.co/guide/en/logstash/current/filter-plugins.html for the huge list of filter plugins.

- Can serve as buffer between source and destination

logstash - input sources

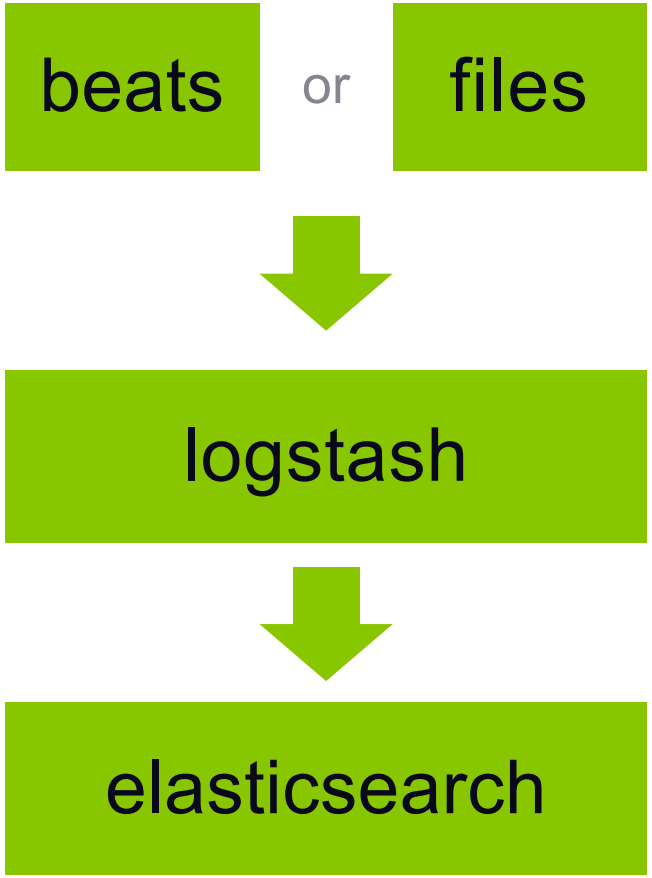
elastic beats – cloudwatch – couchdb – drupal – elasticsearch – windows event log – shell output – local files – ganglia – gelf – gemfire – random generator – github – google pubsub – graphite – heartbeats – heroku – http – imap – irc – jdbc – jmx – kafka – lumberjack – meetup – command pipes – puppet – rabbitmq – rackspace cloud queue – redis – relp – rss – s3 – salesforce – snmp – sqlite – sqs – stdin – stomp – syslog – tcp – twitter – udp – unix sockets – varnish log – websocket – wmi – xmpp – zenoss – zeromq

logstash - output

boundary – circonus – cloudwatch – csv – datadoghq – elasticsearch – email – exec – local file – ganglia – gelf – bigquery – google cloud storage – graphite – graphtastic – hipchat – http – influxdb – irc – jira – juggernaut – kafka – librato – loggly – lumberjack – metriccatcher – mongodb – nagios – new relic insights – opentsdb – pagerduty – pipe to stdin – rabbitmq – rackspace cloud queue – redis – redmine – riak – riemann – s3 – sns – solr – sqs – statsd – stdout – stomp – syslog – tcp – udp – webhdfs – websocket – xmpp – zabbix - zeromq

typical usage

Web logs



Parse into structured fields, geolocate

indexed and analyzed

installing logstash

installing logstash

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

configuring logstash

input = log location
start_position
ignore_older

Set timestamp format being used in access_log

Send to Elasticsearch & standard out

sudo vi /etc/logstash/conf.d/logstash.conf

```
input {
    file {
        path => "/home/<user>/access_log“  start_position => "beginning"  ignore_older => 0
    }
}

filter {
    grok {
        match => { "message" => "%{COMBINEDAPACHELOG}" }
}
    date {
        match => [ "timestamp", "dd/MMM/yyyy:HH:mm:ss Z" ]
    }
}

output {
    elasticsearch {
        hosts => ["localhost:9200"]
}
    stdout {
        codec => rubydebug
    }
}
```

running logstash

```
cd /usr/share/logstash/

sudo bin/logstash -f /etc/logstash/conf.d/logstash.conf
```

logstash with mysql

jdbc driver

get a mysql connector from https://dev.mysql.com/downloads/connector/j/

wget https://dev.mysql.com/get/Downloads/Connector-J/mysql-connector-java-5.1.42.zip

unzip mysql-connector-java-5.1.42.zip

configure logstash

```
input {
    jdbc {
        jdbc_connection_string => "jdbc:mysql://localhost:3306/movielens"
        jdbc_user => "root"
        jdbc_password => “password"
        jdbc_driver_library => "/home/<user>/mysql-connector-java-5.1.46/mysql-connector-java-5.1.46-bin.jar"
        jdbc_driver_class => "com.mysql.jdbc.Driver"
        statement => "SELECT * FROM movies"
    }
}
```

```
output {
    stdout { codec => json_lines }
    elasticsearch {
        "hosts" => "localhost:9200"
        "index" => "movielens-sql"
        "document_type" => "data"
    }
}
```

lab: logstash & mysql

- Lab 13: Install Logstash and integrate with MySQL

logstash with s3

what is s3

amazon web services' simple storage service

cloud-based distributed storage system

integration is easy

```
input {
    s3{
        bucket => "learning-es"
        access_key_id => "AKIAIS****C26Y***Q"
        secret_access_key => "d*****FENOXcCuNC4iTbSLbibA*****eyn****"

    }
}
```

logstash with kafka

what is kafka

- apache kafka
- open-source stream processing platform
- high throughput, low latency
- publish/subscribe
- process streams
- store streams

has a lot in common with logstash, really.

integration is easy

```
input {
    kafka{
        bootstrap_servers => "127.0.0.1:9200"
        topics = ["kafka-logs"]
    }
}
```

⌜ aggregations ⌟

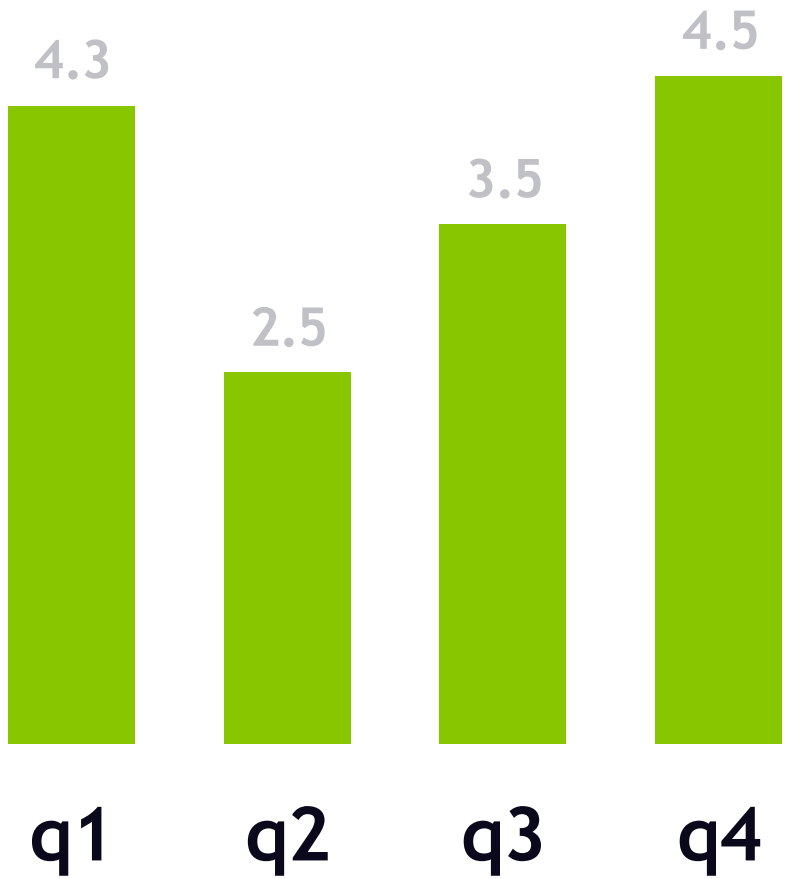
not just for search anymore

metrics

average, stats, min/max, percentiles, etc.

buckets

histograms, ranges, distances, significant terms, etc.



pipelines

moving average, average bucket, cumulative sum, etc.

matrix

matrix stats

aggregations are amazing

elasticsearch aggregations can sometimes take the place of hadoop / spark / etc – and return results instantly!

it gets better

you can even nest aggregations together!

Pair aggregations with search queries

let’s learn by example

bucket by rating value:

```
curl -XGET
'127.0.0.1:9200/ratings/rating/_search?size=0&pretty' -d '
{
    "aggs": {
        "ratings": {
            "terms": {
                "field": "rating"
            }
        }
    }
}'
```

Sum up all documents that have a "rating" value.

let’s learn by example

- query narrows down movies to only 5.0 rating
- aggregates sum up all those values.

count only 5-star ratings:

```
curl -XGET
'127.0.0.1:9200/ratings/rating/_search?size=0&pretty' -d ‘
{
    "query": {
        "match": {
            "rating": 5.0
        }
    },
    "aggs" : {
        "ratings": {
            "terms": {
                "field" : "rating"
            }
        }
    }
}'
```

let's learn by example

- Find Star Wars Episode IV
- Return average rating for that specific movie.

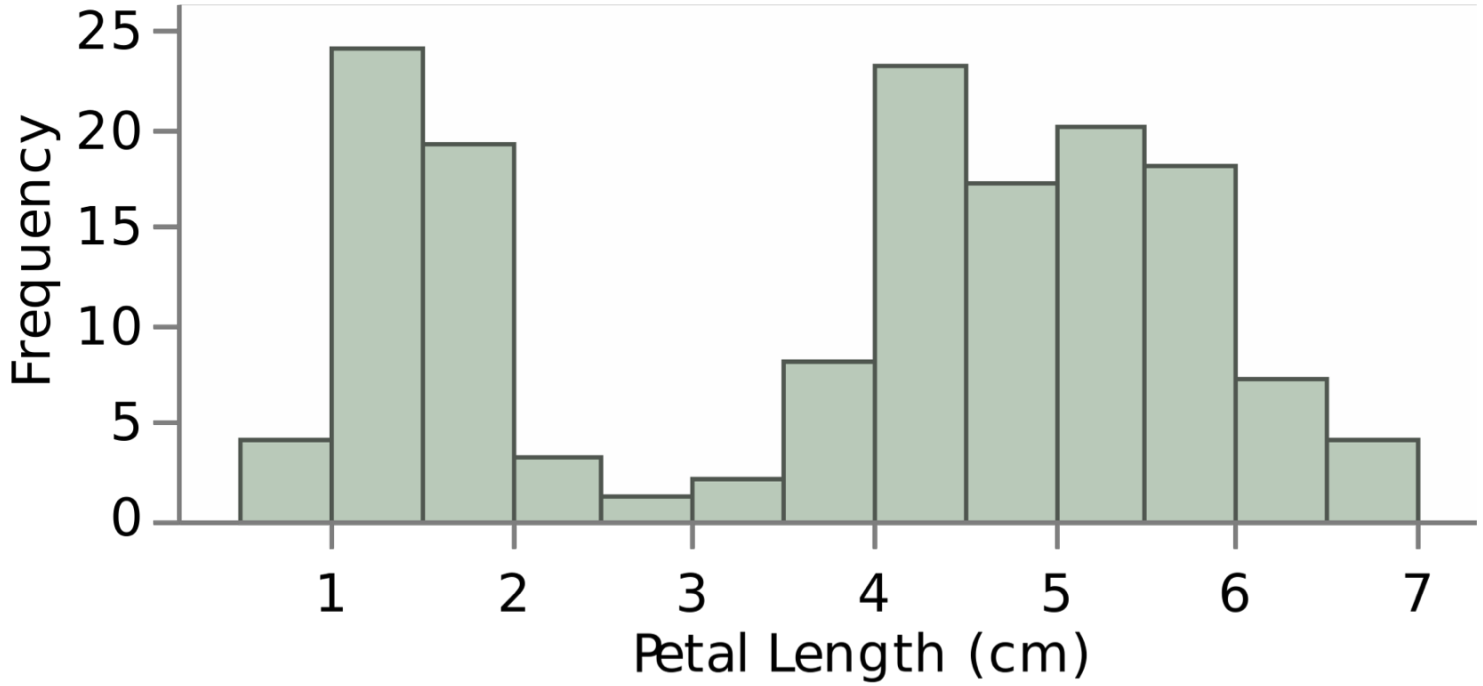
average rating for Star Wars:

```
curl -XGET
'127.0.0.1:9200/ratings/rating/_search?size=0&pretty' -d '
{
    "query": {
        "match_phrase": {
            "title": "Star Wars Episode IV"
        }
    },
    "aggs" : {
        "avg_rating": {
            "avg": {
                "field" : "rating"
            }
        }
    }
}'
```

histograms

what is a histogram

display totals of documents bucketed by some interval range



rating intervals

Group ratings together by whole number.

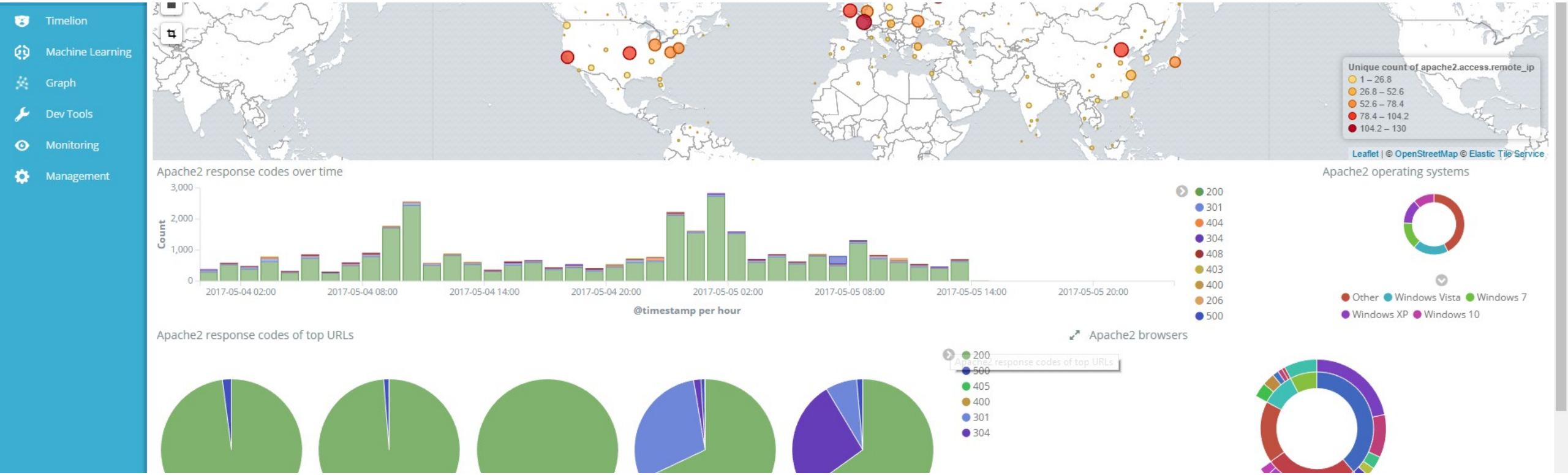
```
curl -XGET
'127.0.0.1:9200/ratings/rating/_search?size=0&pretty' -d '
{
    "aggs" : {
        "whole_ratings": {
            "histogram": {
                "field": "rating",
                "interval": 1.0
            }
        }
    }
}'
```

sort by decade

```
curl -XGET
'127.0.0.1:9200/movies/movie/_search?size=0&pretty' -d '
{
    "aggs" : {
        "release": {
            "histogram": {
                "field": "year",
                "interval": 10
            }
        }
    }
}
```

using
kibana

what is kibana



installing kibana

sudo apt-get install kibana
sudo vi /etc/kibana/kibana.yml
change server.host to 0.0.0.0

sudo /bin/systemctl daemon-reload
sudo /bin/systemctl enable kibana.service
sudo /bin/systemctl start kibana.service

kibana is now available on port 5601

playing with
kibana

let’s analyze the works of william shakespeare…

because we can.

lab: aggs & kibana

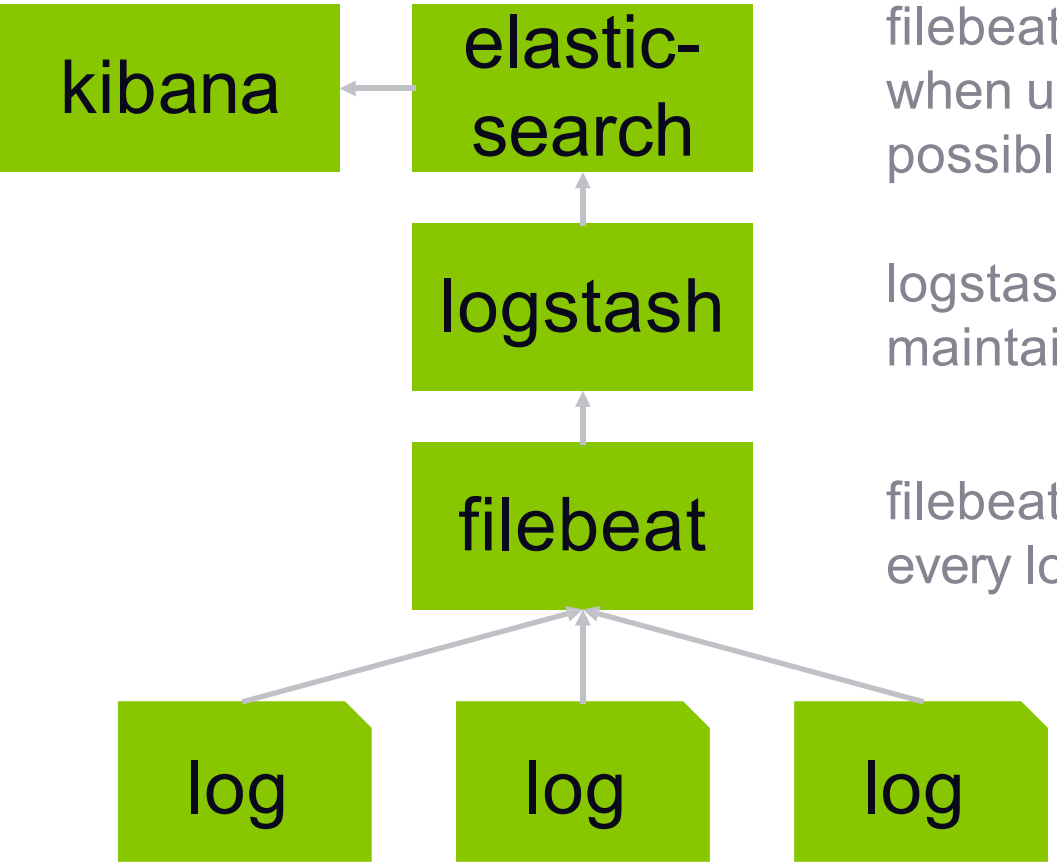
- Lab 14: Using aggregates and histograms
- Lab 15: Installing and using Kibana

exercise

find the longest shakespeare plays - create a vertical bar chart that aggregates the count of documents by play name in descending order.

using
filebeat

filebeat: lightweight shipper



filebeat can optionally talk directly to elasticsearch.
when using logstash, elasticsearch is just one of many possible destinations!

logstash and filebeat can communicate to maintain “backpressure” when things back up

filebeat maintains a read pointer on the logs.
every log line acts like a queue.

logs can be from apache, nginx, auditd, or mysql

this is called the elastic stack

prior to beats, you'd hear about the "ELK stack" – elasticsearch, logstash, kibana.

filebeat vs logstash

- Wrong question: Use them together.
- it won’t let you overload your pipeline.
- you get more flexibility on scaling your cluster.

installing filebeat

lab: kibana & filebeats

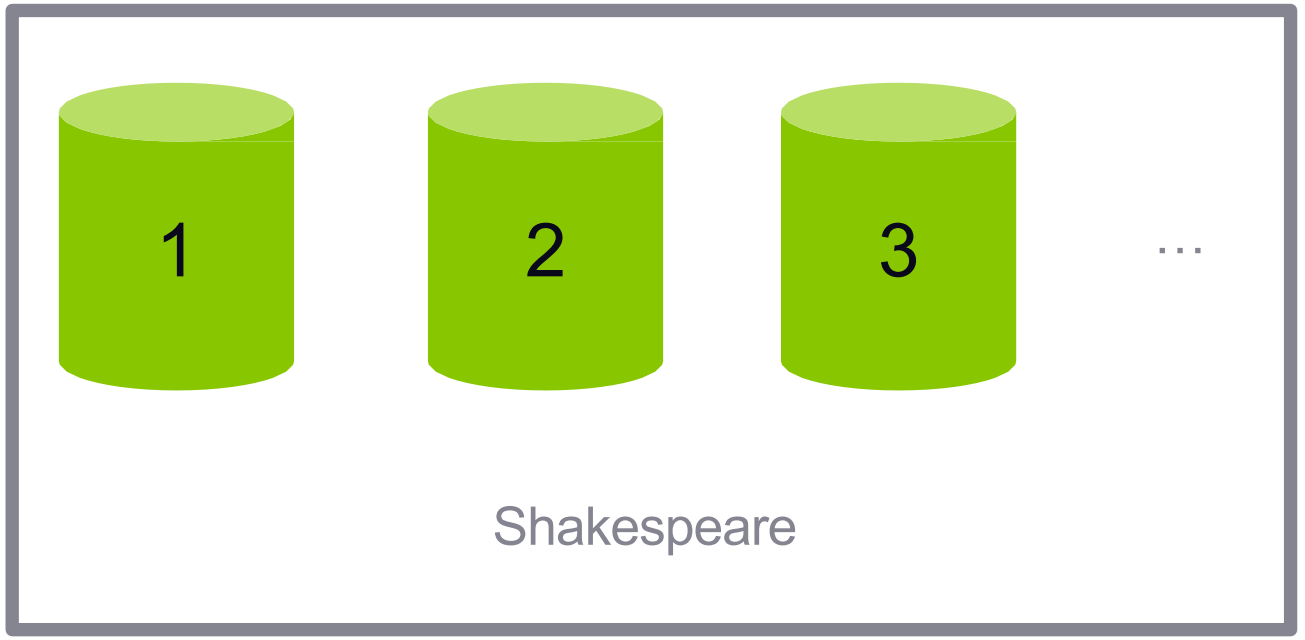
- Lab 16: Install filebeat and configure with Kibana

elasticsearch operations

choosing your shards

an index is split into shards.

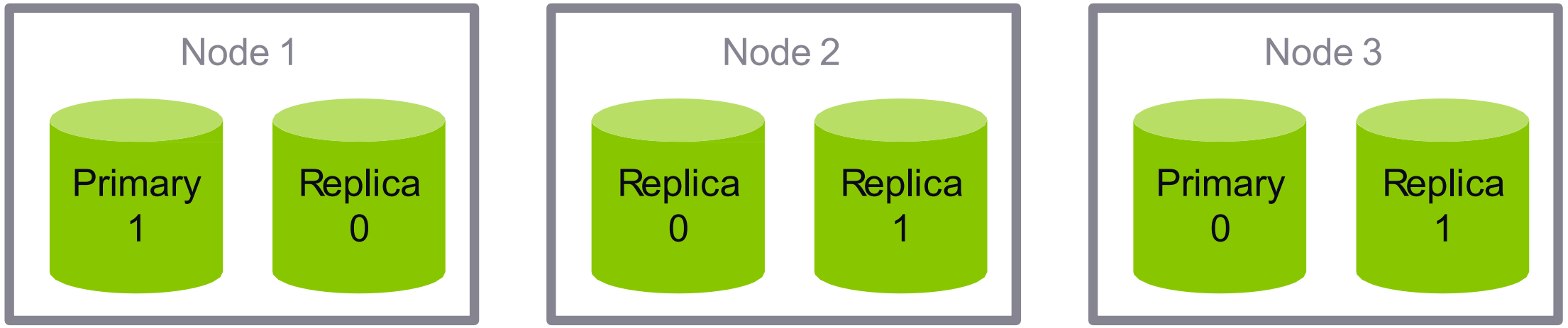
Documents are hashed to a particular shard.



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

primary and replica shards

This index has two primary shards and two replicas.
Your application should round-robin requests amongst nodes.



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

how many shards do i need?

- you can’t add more shards later without re-indexing
- but shards aren’t free – you can't just make 1,000 of them and stick them on one node at first.
- you want to overallocate, but not too much
- consider scaling out in phases, so you have time to re-index before you hit the next phase

shard planning

- the “right” number of shards depends on your data and your application. there’s no secret formula.
- start with a single server using the same hardware you use in production, with one shard and no replication.
- fill it with real documents and hit it with real queries.
- push it until it breaks – now you know the capacity of a single shard.

remember replica shards can be added

- read-heavy applications can add more replica shards without re-indexing.
- note this only helps if you put the new replicas on extra hardware!

adding an index

creating a new index

- Remember this is an important step for production clusters!

```
PUT /new_index
{
        “settings”: {
                “number_of_shards”:     10,
                “number_of_replicas”:   1
        }
}
```

You can use *index templates* to automatically apply mappings, analyzers, aliases, etc.

scaling strategies

- multiple-indices:
  - make a new index to hold new data
  - search both indices
  - use *index aliases* to make this easy to do

scaling strategies

- multiple-indices:
  - with time-based data, you can have one index per time

  frame
  - common strategy for log data where you usually just want current data, but don't want to delete old data either
  - again you can use index aliases, ie "logs_current", "last_3_months", to point to specific indices as they rotate

alias rotation example

```
POST /_aliases
{
        “actions”: [
                { “add”:        { “alias”: “logs_current”, “index”: “logs_2017_06” }},
                { “remove”:     { “alias”: “logs_current”, “index”: “logs_2017_05” }},
                { “add”:        { “alias”: “logs_last_3_months”, “index”: “logs_2017_06” }},
                { “remove”:     { “alias”: “logs_last_3_months”, “index”: “logs_2017_03” }}
        ]
}

optionally….
DELETE /logs_2017_03
```

choosing your hardware

RAM is likely your bottleneck

64GB per machine is the sweet spot (32GB to elasticsearch, 32GB to the OS / disk cache for lucene)

under 8GB not recommended

other hardware considerations

- fast disks are better – SSD's if possible
- use RAID0 – your cluster is already redundant
- cpu not that important
- need a fast network
- don't use NAS
- use medium to large configurations; too big is bad, and too many small boxes is bad too.

heap sizing

your heap size is wrong

the default heap size is only 1GB!

half or less of your physical memory should be allocated to elasticsearch

- the other half can be used by lucene for caching
- if you're not aggregating on analyzed string fields, consider using less than half for elasticsearch
- smaller heaps result in faster garbage collection and more memory for caching

```
export ES_HEAP_SIZE=10g
```

or

```
ES_JAVA_OPTS="-Xms10g -Xmx10g" ./bin/elasticsearch
```

don't cross 32GB! pointers blow up then.

monitoring with x-pack

what is x-pack?

- an elastic stack extension
- security, monitoring, alerting, reporting, graph, and machine learning
- formerly shield / watcher / marvel
- only parts can be had for free – requires a paid license or trial otherwise

install x-pack

```
cd /usr/share/elasticsearch
sudo bin/elasticsearch-plugin install x-pack

sudo vi /etc/elasticsearch/elasticsearch.yml
(Add xpack.security.enabled:false)

sudo /bin/systemctl stop elasticsearch.service

sudo /bin/systemctl start elasticsearch.service

cd /usr/share/kibana/
sudo -u kibana bin/kibana-plugin install x-pack
sudo /bin/systemctl stop kibana.service

sudo /bin/systemctl start kibana.service
```

failover in action

in this activity, we'll...

- Set up a second elasticsearch node on our virtual machine
- Observe how elasticsearch automatically expands across this new node
- Stop our original node, and observe everything move to the new one
- Restart our original node, and observe everything going back to normal… automatically!

using snapshots

back up your indices

store backups to NAS, Amazon S3, HDFS,Azure

smart enough to only store changes since last snapshot

create a repository

add it into elasticsearch.yml:

path.repo: ["/home/<user>/backups"]

```
PUT _snapshot/backup-repo
{
 "type": "fs",
 "settings": {
   "location": "/home/<user>/backups/backup-repo"
 }
}
```

using snapshots

snapshot all open indices:

```
PUT _snapshot/backup-repo/snapshot-1
```

get information about a snapshot:

```
GET _snapshot/backup-repo/snapshot-1
```

monitor snapshot progress:

```
GET _snapshot/backup-repo/snapshot-1/_status
```

restore a snapshot of all indices:

```
POST /_all/_close
POST _snapshot/backup-repo/snapshot-1/_restore
```

rolling
restarts

restarting your cluster

sometimes you have to… OS updates, elasticsearch version updates, etc.

to make this go quickly and smoothly, you want to disable index reallocation while doing this.

rolling restart procedure

1. stop indexing new data if possible
2. disable shard allocation
3. shut down one node
4. perform your maintenance on it and restart, confirm it joins the cluster.
5. re-enable shard allocation
6. wait for the cluster to return to green status
7. repeat steps 2-6 for all other nodes
8. resume indexing new data

cheat sheet

```
PUT_cluster/settings
{
 "transient": {
   "cluster.routing.allocation.enable": "none"
 }
}
```

Disable shard allocation

```
sudo /bin/systemctl stop elasticsearch.service
```

Stop elasticsearch safely

```
PUT_cluster/settings
{
 "transient": {
   "cluster.routing.allocation.enable": "all"
 }
}
```

Enable shard allocation

let’s
practice

wrapping up