# Running & Scaling Large Elasticsearch Clusters

Fred de Villamil, Director of Infrastructure

@fdevillamil

October 2017

synthesio

# Background

- Fred de Villamil, 39 ans, Team Coffee @Synthesio,
- Linux / (Free)BSD user since 1996,
- Open Source contributor since 1998,
- Loves tennis, photography, cute otters, inappropriate humor and Elasticsearch clusters of unusual size.

Writes about ES at https://thoughts.t37.net

synthesio

# ELASTICSEARCH @SYNTHESIO

- 8 production clusters, 600 hosts, 1.7PB storage, 37.5TB RAM,  average 15k writes / s, 800 search /s, some inputs > 200MB.

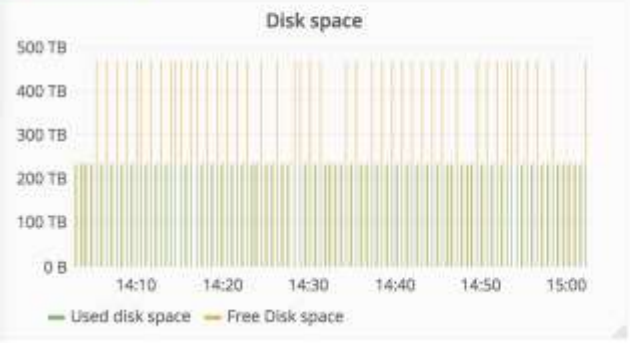
- Data nodes: 6 core Xeon E5v3, 64GB RAM, 4*800GB SSD RAID0. Sometimes bi Xeon E5-2687Wv4 12 core (160 watts!!!).

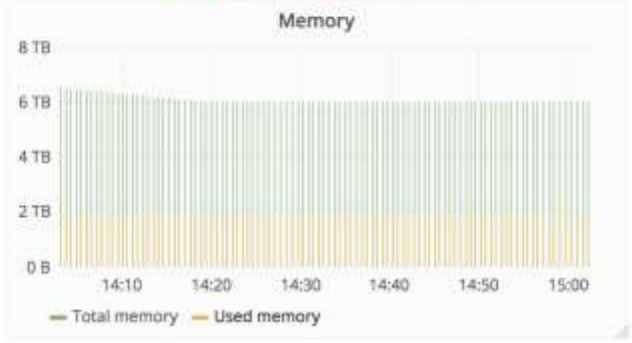
- We agregate data from various cold storage and make them searchable in a giffy.

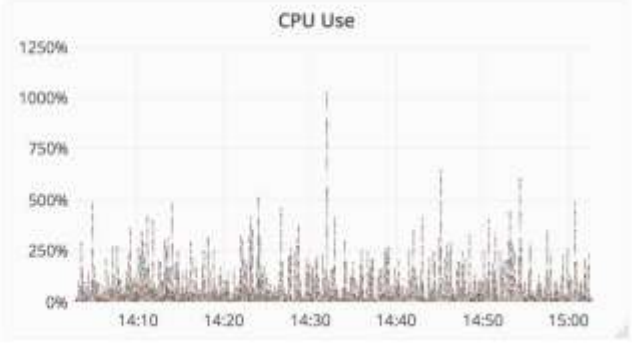
# An Elasticsearch cluster of unusual size

| Status | Cluster size | Indices | Shards | Documents | Data | Storage | Heap |
|---|---|---|---|---|---|---|---|
| OK | 186 nodes | 1138 | 39900 | 50.0 Bil | 209 TiB | 429 TiB | 6.05 TB |

50 billion unique docs * 4




synthesio

# Ensuring High Availability

# Never Gonna Give You Up

- Never gonna let you down,
- Never gonna run around and desert you,
- Never gonna make you cry,
- Never gonna say goodbye,
- Never gonna tell a lie & hurt you.

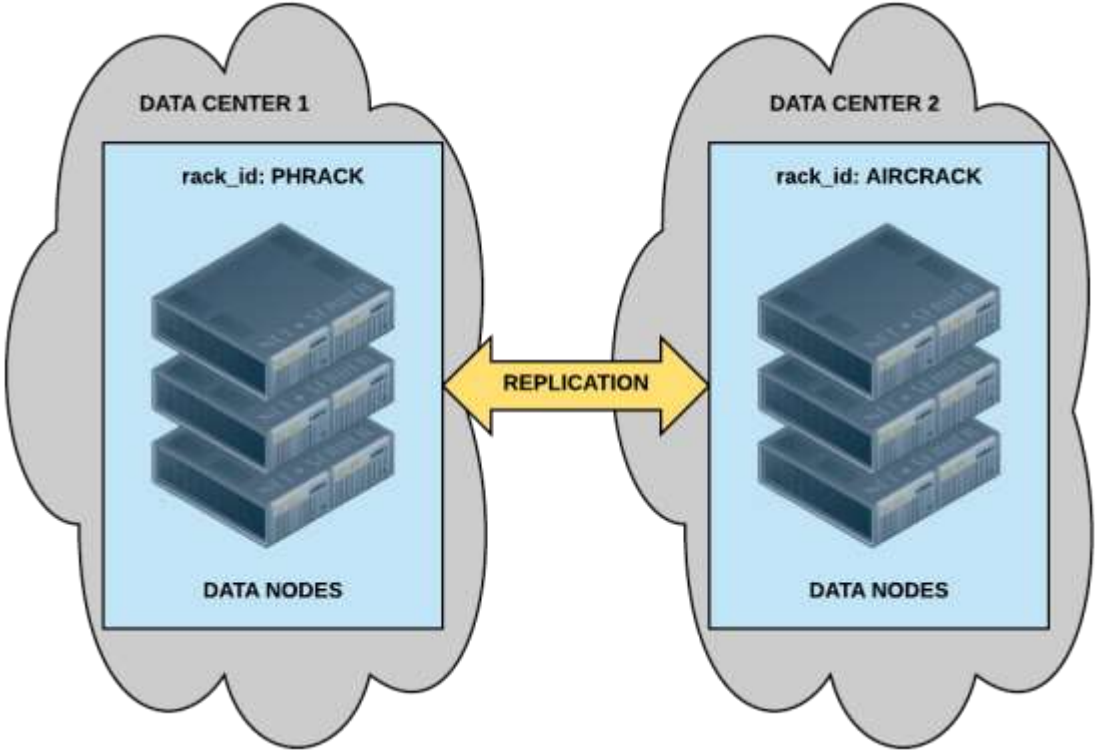
# Avoiding Downtime & Split Brains

- Run at least 3 master nodes into 3 different locations.
- NEVER run bulk queries on the master nodes.
- Actually NEVER run anything but administrative tasks on the master nodes.
- Spread your data into 2 different location with at least a replication factor of 1 (1 primary, 1 replica).

# Rack Awareness

Allocate a rack_id to the data nodes for even replication.

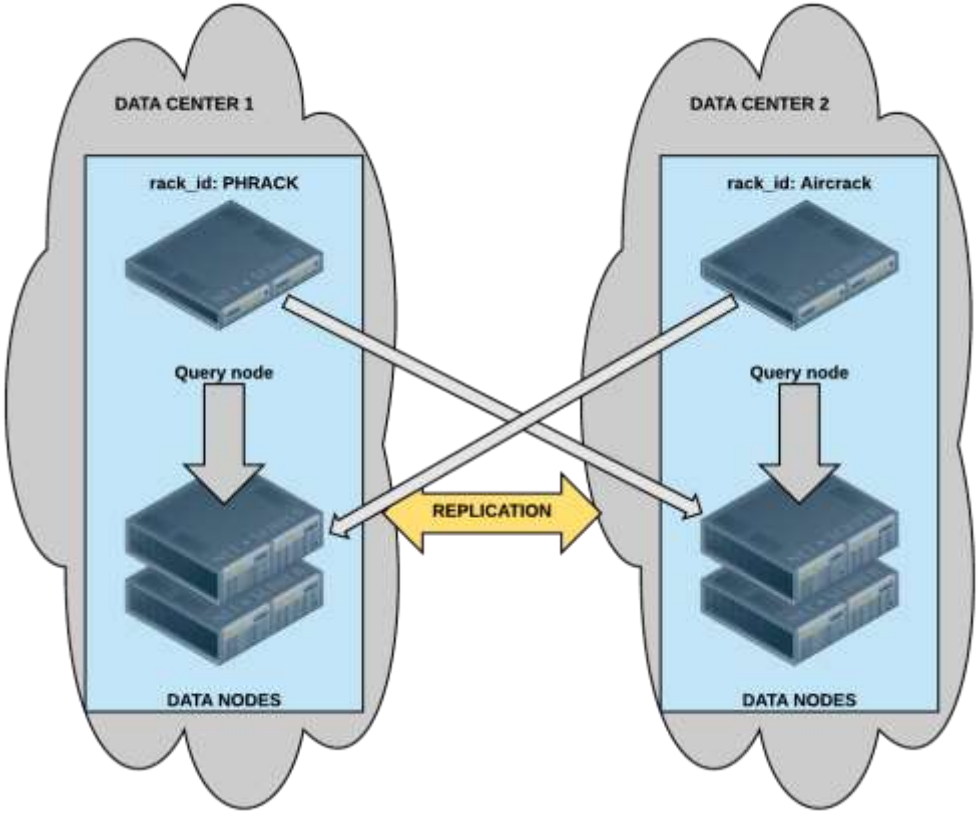
Restart a whole data center @once without downtime.



# Rack Awareness + Query Nodes == Magic

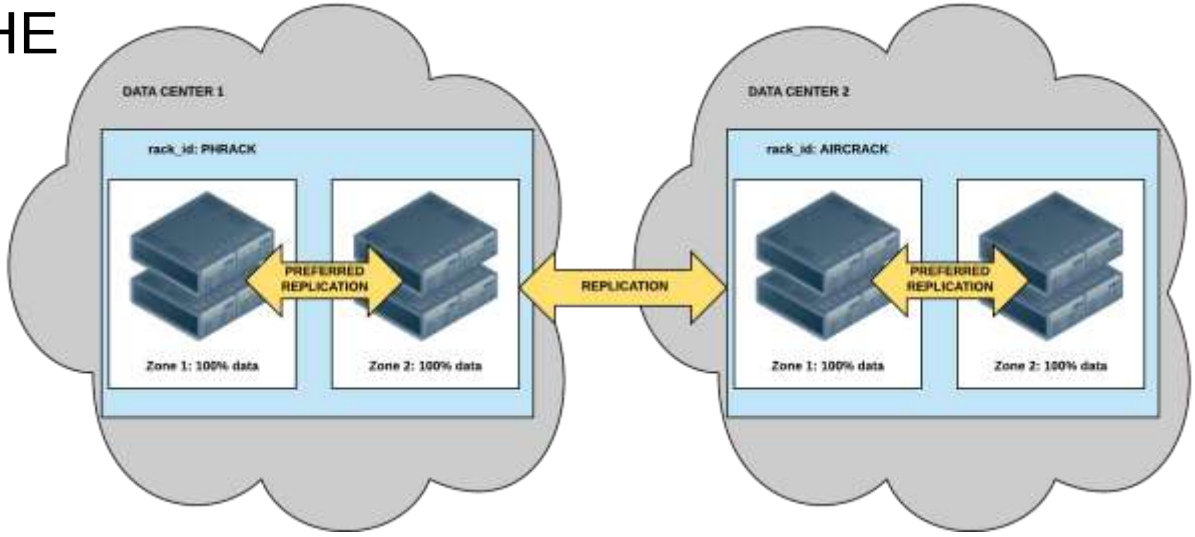
ES privileges the data nodes with the same rack_id as the query.

Reduces latency and balances the load.



# Rack Awareness + Query Nodes + Zone == Magic + Fun

Add zones into the same rack for even replication with higher factor.

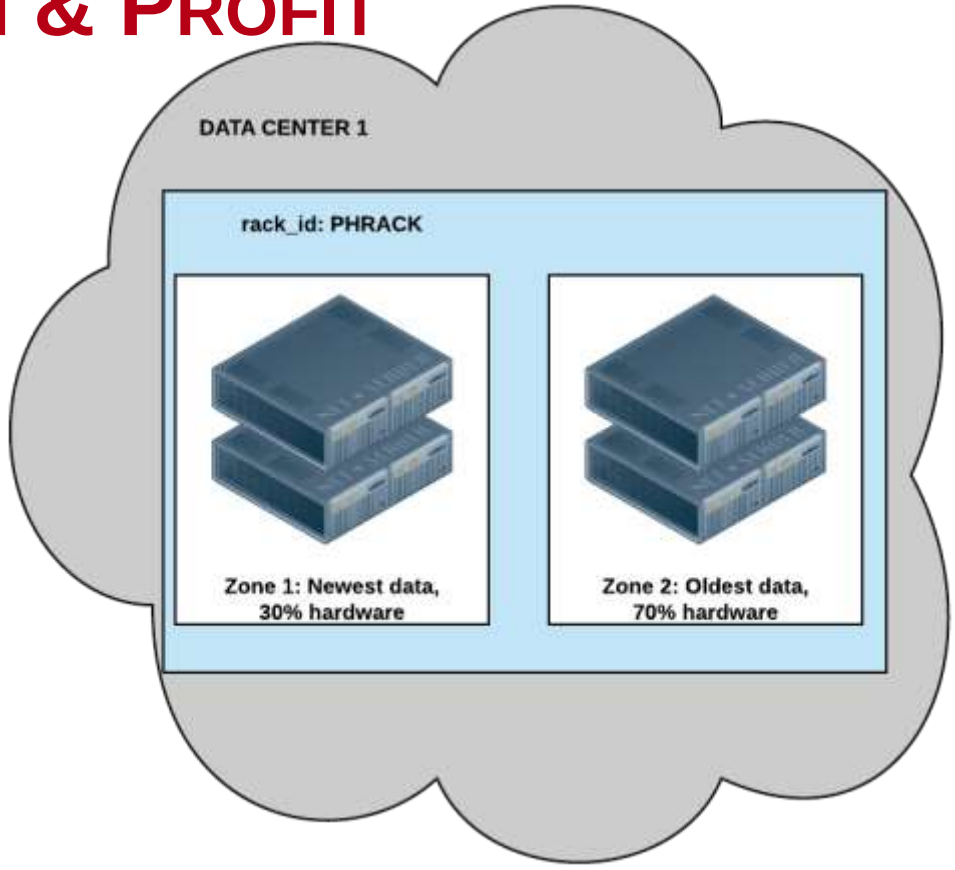


# Using Zones for Fun & Profit

Allowing even replication with a higher factor within the same rack.

Allowing more resources to the most frequently accessed indexes.

…



# Avoiding memory Nightmare

# How ElasticSearch Uses the Memory

- Starts with allocating memory for Java heap.
- The Java heap contains all Elasticsearch buffers and caches + a few other things.
- Each Java thread maps a system thread: +128kB off heap.
- Elected master uses 250kB to store each shard information inside the cluster.

# Allocating memory

- Never allocate more than 31GB heap to avoid the compressed pointers issue.
- Use 1/2 of your memory up to 31GB.
- Feed your master and query nodes, the more the better (including CPU).

# Memory lock

- Use `memory_lock: true` at startup.
- Requires `ulimit -l unlimited`.
- Allocates the whole heap at once.
- Uses contiguous memory regions.
- Avoids swapping (you should disable swap anyway).

# Chosing the right garbage collector

- ES runs with CMS as a default garbage collector.
- CMS was designed for heaps < 4GB.
- Stop the world garbage collection last too long & blocks the cluster.
- Solution: switching to G1GC (default in Java9, unsupported).

# CMS VS G1GC

- CMS: shared CPU time with the application. "Stops the world" when too many memory to clean until it sends an OutOfMemoryError.
- G1GC: short, more frequent, pauses. Won't stop a node until it leaves the cluster.
- Elastic says: **DON'T USE G1GC** for reasons, so read the doc.

# G1GC Options

+UseG1GC: activates G1GC

MaxGCPauseMillis: target for Max GC pause time.

GCPauseIntervalMillis: target for collection time space

InitiatingHeapOccupancyPercent: when to start collecting?

# Cho0sing the right storage

- `MMAPFS` : maps Lucene files on the virtual memory using `MMAP`. Needs as much memory as the file being mapped to avoid issues.

- `NIOFS` : applies a shared lock on Lucene files and relies on VFS cache.

# Buffers and Caches

- ElasticSearch has multiple caches & buffers, with default values, know them!
- Buffers + cache must be < total Java heap (obvious but…).
- Automated eviction on the cache, but forcing it can save your life with a small overhead.
- If you have OOM issues, disable the caches!
- From a user POV, circuit breakers are a no go!

# MANAGING LARGE INDEXES

# Index design

- Version your index by mapping: `1_*`, `2_*` etc.
- The more shards, the better elasticity, but the more CPU and memory used on the masters.
- Provisionning 10GB per shards allows a faster recovery & reallocation.

# Replication tricks

- Number of replicas must be 0 or odd.
  Consistency Quorum: `int( (primary + number_of_replicas) / 2 ) + 1`.

- Raise the replication factor to scale for reading
  up to 100% of the data / data node.

# Aliases

- Access multiple indices at once.
- Read multiple, write only one.

Example on timestamped indices:

```
"18_20171020": { "aliases": { "2017": {}, "201710": {}, "20171020": {} } }

"18_20171021": { "aliases": { "2017": {}, "201710": {}, "20171021": {} } }
```

```
Queries:

/2017/_search

/201710/_search
```

After a mapping change & reindex, change the alias:

# Rollover

- Create a new index when too old or too big.
- Support date math: daily index creation.
- Use aliases to query all rollovered indexes.

```
PUT "logs-000001" { "aliases": { "logs": {} } }

POST /logs/_rollover { "conditions": { "max_docs":  10000000 } }
```

- API only!!! You need a cron job to

# DAILY OPERATIONS

# Configuration changes

- Prefer configuration file updates to API call for permanent changes.
- Version your configuration changes so you can rollback, ES requires lots of fine tuning.
- When using _settings API, prefer transient to persistent, they're easier to get rid of.

# Reconfiguring the whole cluster

## Lock shard reallocation & recovery:

```
"cluster.routing.allocation.enable" : "none"
```

## Optimize for recovery:

```
"cluster.routing.allocation.node_initial_primaries_recoveries": 50

"indices.recovery.max_bytes_per_sec": "2048mb"
```

## Restart a full rack, wait for nodes to come back:

# The reindex API

- In cluster and cluster to cluster reindex API.
- Allows cross version indexing: 1.7 to 5.1…
- Sliced scrolls only available starting 6.0.
- Accept ES queries to filter the data to reindex.
- Merge multiple indexes into 1.

# Bulk indexing tricks

## Limit rebalance:

```
"cluster.routing.allocation.cluster_concurrent_rebalance": 1

"cluster.routing.allocation.balance.shard": "0.15f"

"cluster.routing.allocation.balance.threshold": "10.0f"
```

## Disable refresh:

```
"index.refresh_interval:" "0"
```

## No replica:

```
"index.number_of_replicas:" "0" // having replica index n times in Lucene, adding one just "rsync" the data.
```

## Allocate on dedicated hardware:

# Optimizing for Space & Performances

- Lucene segments are immutable, the more you write, the more segments you get.
- Deleting documents does copy on write so no real delete.

```
index.merge.scheduler.max_thread_count: default CPU/2 with min 4

POST /_force_merge?only_expunge_deletes: faster, only merge segments with deleted

POST /_force_merge?max_num_segments: don't use on indexes you write on!
```

Warning: `_force_merge` has a cost in CPU and I/Os.

# Minor version upgrades

- Check your plugins compatibility, plugins must be compiled for your minor version.
- Start upgrading the master nodes.
- Upgrade the data nodes on a whole rack at once.

# OS level upgrades

- Ensure the whole cluster runs the same Java version.
- When upgrading Java, check if you don't have to upgrade the kernel.
- Per node Java / kernel version available in the `_stats` API.

# MONITORING

# Captain obvious, you're my only hope!

- Good monitoring is business oriented monitoring.
- Good alerting is actionable alerting.
- Don't monitore the cluster only, but the whole processing chain.
- Useless metrics are useless.
- Losing a datacenter: OK. losing data: not OK!

# Monitoring tooling

- Elasticsearch x-pack,
- Grafana…

# LIFE, DEATH & _CLUSTER/HEALTH

- A RED CLUSTER MEANS AT LEAST 1 INDEX HAS MISSING DATA. DON'T PANIC!
- USING `LEVEL={INDEX,SHARD}` AND AN INDEX ID PROVIDES SPECIFIC INFORMATION.
- LOTS OF PENDING TASKS MEANS YOUR CLUSTER IS UNDER HEAVY LOAD AND SOME NODES CAN'T PROCESS THEM FAST ENOUGH.
- LONG WAITING TASKS MEANS YOU HAVE A CAPACITY PLANNING PROBLEM.


```
{
    "cluster_name": "blackhole",
    "status": "green",
    "timed_out": false,
    "number_of_nodes": 78,
    "number_of_data_nodes": 68,
    "active_primary_shards": 13790,
    "active_shards": 27580,
    "relocating_shards": 18,
    "initializing_shards": 0,
    "unassigned_shards": 0,
    "delayed_unassigned_shards": 0,
    "number_of_pending_tasks": 2,
    "number_of_in_flight_fetch": 0,
    "task_max_waiting_in_queue_millis": 6519,
    "active_shards_percent_as_number": 100
}
```

# Use the _cat API

- Provides general information about your nodes, shards, indices and thread pools.
- Hit the whole cluster, when it timeouts you've probably having a node stuck in garbage collection.

# Monitoring at the cluster level

- Use the `_stats` API for precise information.
- Monitore the shards reallocation, too many means a design problem.
- Monitore the writes and cluster wide, if they fall to 0 and it's unusual, a node is stuck in GC.

# Monitoring at the node level

- Use the `_nodes/{node}`, `_nodes/{node}/stats` and `_cat/thread_pool` API.
- The garbage collection duration & frequency is a good metric of your node health.
- Cache and buffers are monitored on a node level.
- Monitoring I/Os, space, open files & CPU is critical.

# Monitoring at the index level

- Use the `{index}/_stats` API.
- Monitore the documents / shard ratio.
- Monitore the merges, query time.
- Too many evictions means you have a cache configuration level.

# TROUBLESHOOTING

# What's really going on in your cluster?

- The `_nodes/{node}/hot_threads` API tells what happens on the host.
- The elected master nodes tells you most thing you need to know.
- Enable the slow logs to understand your bottleneck & optimize the queries. Disable the slow logs when you're done!!!
- When not enough, memory profiling is your friend.

# Memory profiling

- Live memory or hprof file after a crash.
- Allows you to to know what is / was in your buffers and caches.
- Yourkit Java Profiler as a tool.

# Tracing

- Know what's really happening in your JVM.
- Linux 4.x provides great perf tools, linux 4.9 even better:
  - linux-perf,
  - Java perf map.
- Vector by netflix (not Vektor the trash metal band).

# QUESTIONS ?

@FDEVILLAMIL

@SYNTHESIO

SLIDES: HTTP://BIT.DO/ELASTICSEARCH-SYSADMIN-201

synthesio