BeeCon 2016

Backing Data Silo Atack: Alfresco sharding, SOLR for non-flat objects

Alexey Vasyukov <avasyukov@itdhq.com>

BeeCon 2016

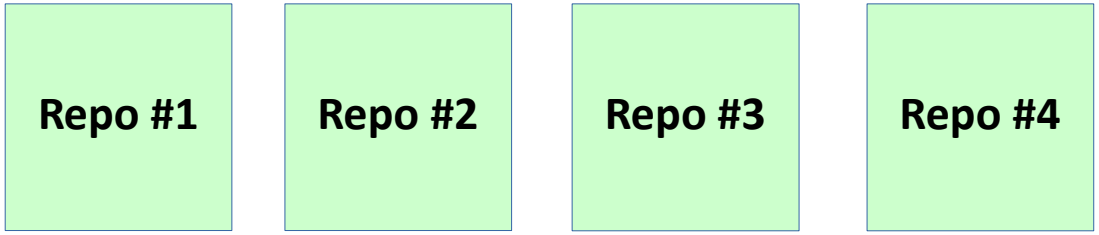
Buzzwords

Backing Data Silo Atack: Alfresco sharding, SOLR for non-flat objects

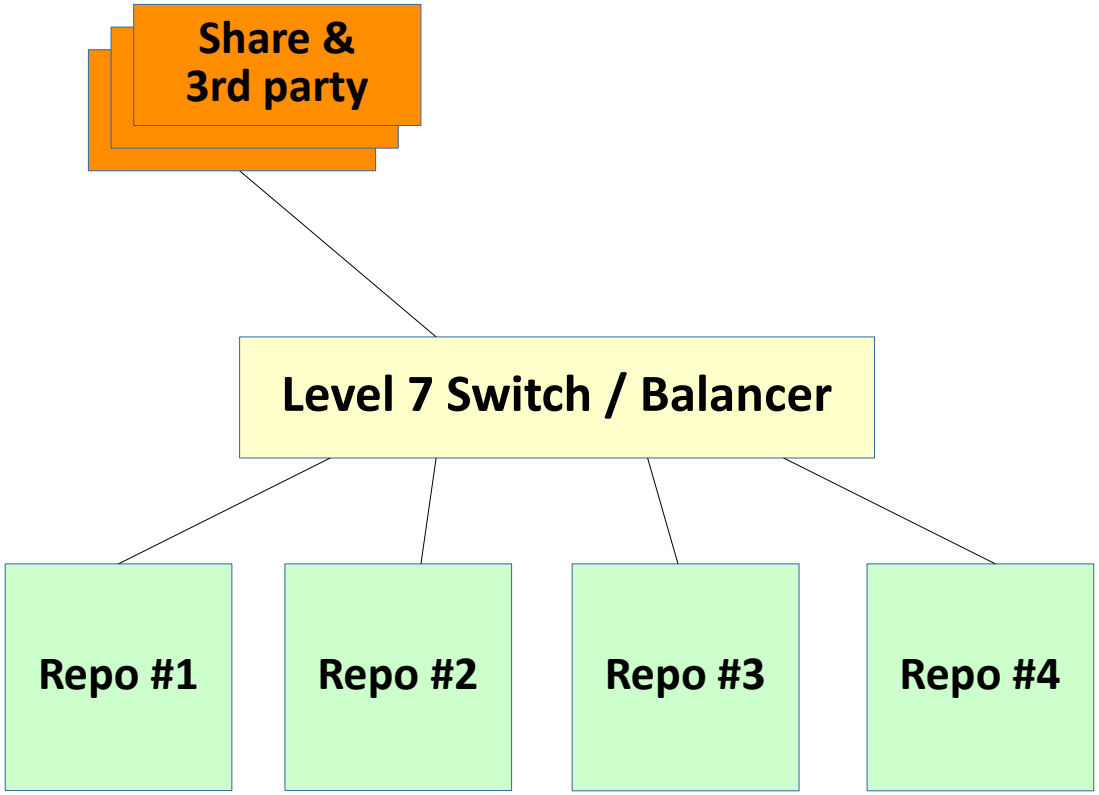
Alexey Vasyukov <avasyukov@itdhq.com>

The Scheme

- Each server is independent
- Each server stores a part of content



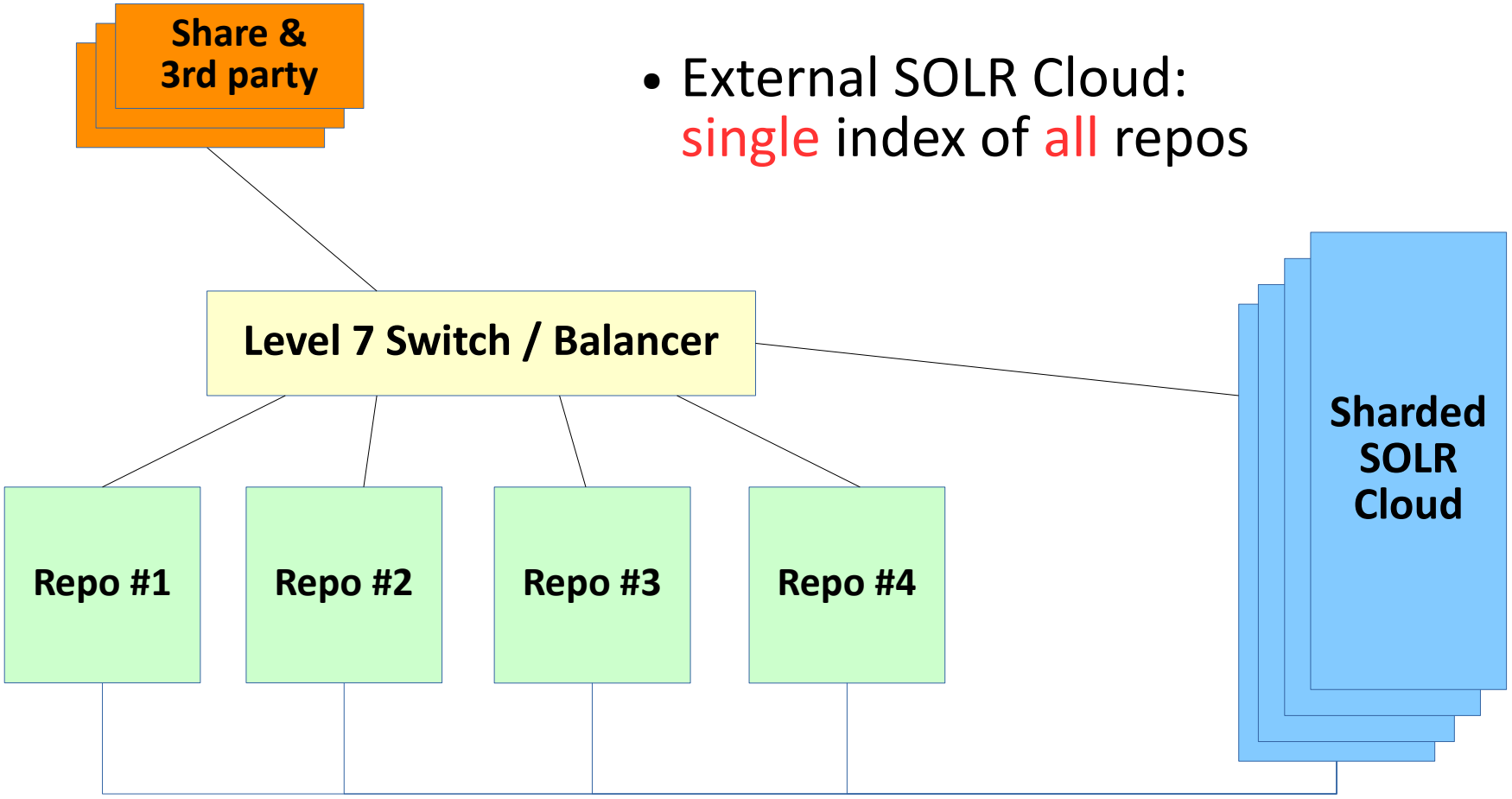
The Scheme



- Each server is independent
- Each server stores a part of content
- We can query them all with native Alfresco REST API and a single API entry point
- We can even run an unmodified Share or 3rd party app on top of federation

The Scheme

- External SOLR Cloud: single index of all repos



Questions regarding the scheme

- What the hell is going on?
- Are you guys crazy?
- Why do you need this scheme at all?
- It is a wheel reinvented, isn't it?

Background information

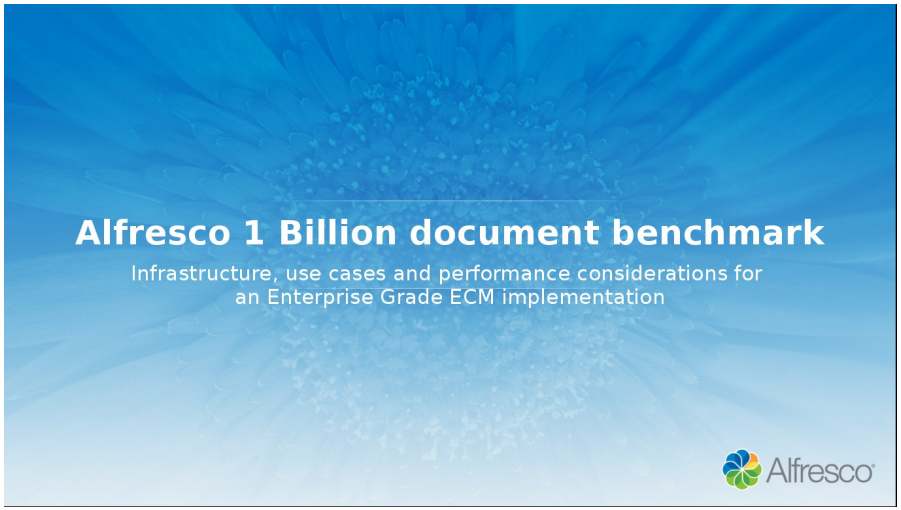
- The project started a year ago (Spring 2015)
- The need for a content platform:
  - Few billions objects in several years — nobody knows the real size of the repo in 5-10 years
  - Several thousands concurrent users
  - Few dozens external apps — nobody knows content structure in several years (but for sure it will be compex)

Background information

- Concerns regarding Alfresco (Spring 2015!)
  - uncertanty on repo size scalability (strictly unofficial rumors regarding 100M nodes as a soft limit for a single repo, internal testing confirms this)
  - no SOLR sharding (yes, the customer wants full text for billions of documents — hello, Google)

Background information

- And ... Q4 2015
  - Alfresco releases 1B benchmark
  - Alfresco supports sharded SOLR (finally!)

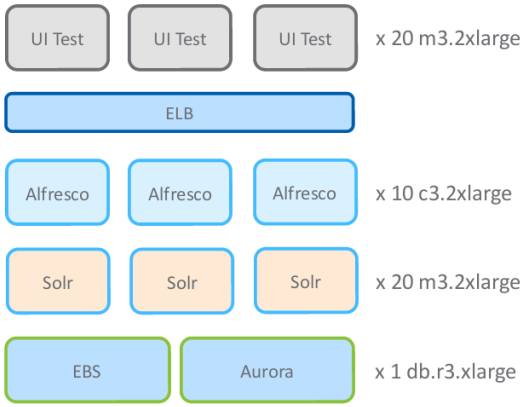


- Do we still need our wheel? Yes!

BeeCon 2016

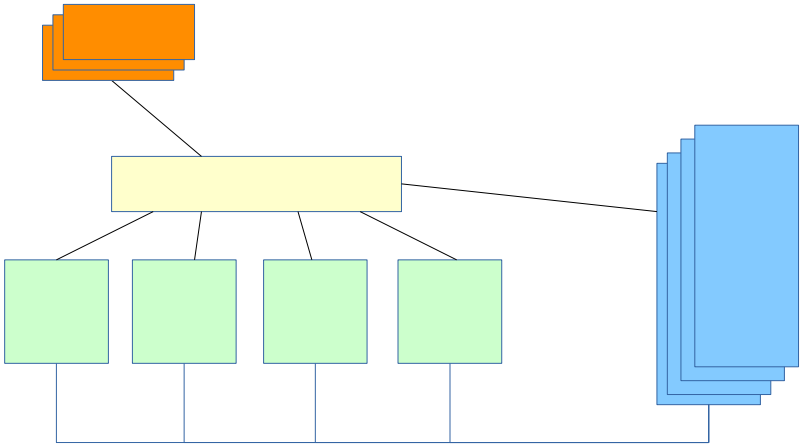
Different architectures & use cases

1B benchmark



- Powered by repo clustering
- Single DB & content store
- All features

Sharded scheme



- Powered by L7 switch / balancer
- Separate DBs & content stores
- Limited features
  - No native assocs between repos

Key Decision Point: Storage!

- Not addressed in 1B benchmark

- *Content Store I/O*
  - File were spoofed, so not on the filesystem (bm-dataload allows to store them)

Source: Alfresco 1B Benchmark

Files were spoofed, so not on the filesystem. Alfresco server will generate a consistent text document each time the content is requested by an API, SOLR or Share.

Source: Alfresco Wiki

Section	Property	Description
File Spoofing	**Spoof File Creation**	When **false**, the usual single file upload via CMIS will be used, giving a realistic CMIS upload simulation. In order to have the server generate plain text document, enable spoofing by setting this to **true**.
	Force Binary Stroage	By default the Alfresco server will generate a consistent text document each time the content is requested by an API, SOLR or Share. If this setting is **true**, then the generated text documents will be written through to the content store and stored in the usual manner. Apart from stressing the disk IO, storage or backup mechanisms, there is no compelling reason to change this setting.
	Files per Transaction	For each folder that needs to be loaded, a single call is made to Alfresco. The files and related metadata are generated on the server and committed in batches.

Key Decision Point: Storage!

- Not addressed in 1B benchmark
  - Reasonable for load testing repo & SOLR
  - Not acceptable for infrastructure planning

Key Decision Point: Storage!

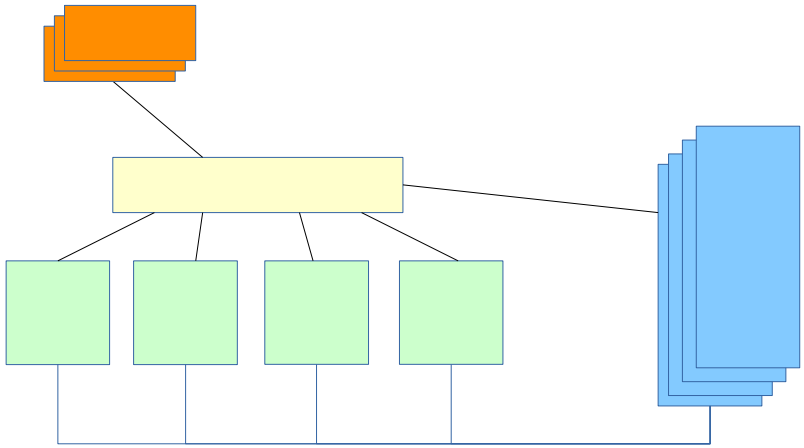
- Not addressed in 1B benchmark
  - Reasonable for load testing repo & SOLR
  - Not acceptable for infrastructure planning
- 1B files x 100 kb x 1 version = 100 Tb
  1B files x 100 kb x 10 versions = 1 Pb
  - Clustering requires shared storage
  - Terribly expensive, complex in operation

Key Decision Point: Storage!

- Not addressed in 1B benchmark
  - Reasonable for load testing repo & SOLR
  - Not acceptable for infrastructure planning
- 1B files x 100 kb x 1 version = 100 Tb
  1B files x 100 kb x 10 versions = 1 Pb
  - Clustering requires shared storage
  - Terribly expensive, complex in operation
- Answer? Sharding allows to use local disks!
  - Fast, cheap, simple, vendor neutral
  - Hello, Google

External SOLR Cloud

- Single SOLR collection for all federated repos
  - Native sorting, paging, etc
  - No heavy logic and overhead in L7 switch



External SOLR Cloud

- Complex queries on associated objects
- Common approach — policy to copy properties between related objects:
  - Overhead in DB and index
  - Your clear content structure goes crazy
  - Tons on policies to handle assoc creation / deletion, properties update for all objects
  - What about multiple assocs with queries inside child nodes?
  - What about full text, touching more than one object?

External SOLR Cloud

- Complex queries on associated objects
- Alternative approach — nested SOLR objects:
  - No redundancy in DB tables / SOLR index
  - Metadata and full-text in a single query

External SOLR Cloud

- Complex queries on associated objects:
- Alternative approach — nested SOLR objects:
  - No redundancy in DB tables / SOLR index
  - Metadata and full-text in a single query
  - Not a silver bullet, not an only option
  - Requires custom indexer implementation

L7 Switch Internals

- Separate Spring App
  - Yes, we implemented the switch in Java
- Handles requests, communicates with backend repos and external SOLR Cloud

L7 Switch Internals

- Separate Spring App
  - Yes, we implemented the switch in Java
- Handles requests, communicates with backend repos and external SOLR Cloud
- Stateless, can be scaled out with zero changes

L7 Switch Internals

- Separate Spring App
  - Yes, we implemented the switch in Java
- Handles requests, communicates with backend repos and external SOLR Cloud
- Stateless, can be scaled out with zero changes
- Common Alfresco REST API
  - Native API for Share and 3rd party apps
  - No heavy logic in the switch

L7 Switch Internals: Deeper Look

- Public APIs from L7 switch point of view:
  - Distributed (all repos)
  - Replicated (write to all repos, read from any repo)
  - Single-node (single repo, content based routing)
  - SOLR Cloud calls

L7 Switch Internals: Real Life

- Mapping all repo APIs to correct switch controllers is hard and time consuming:
  - Test, test, test
  - (+ upstream changes track, track, track)

- However, you do not need all APIs for your project in real life
  - Still test, test, test

Demo

BeeCon 2016

L7 Switch Benchmark

- Concentrates on L7 switch benchmark
  - Backend repos and SOLR are «fast enough» in the test

See benchmark details in backup slides (Appendix A)

L7 Switch Benchmark

- Concentrates on L7 switch benchmark
  - Backend repos and SOLR are «fast enough» in the test
- 15k users with 30s think time
  - 125 browse req/s (federated)
  - 125 search req/s (SOLR)
  - 250 node access req/s (single repo access)

See benchmark details in backup slides (Appendix A)

L7 Switch Benchmark

- Concentrates on L7 switch benchmark
  - Backend repos and SOLR are «fast enough» in the test
- 15k users with 30s think time
  - 125 browse req/s (federated)
  - 125 search req/s (SOLR)
  - 250 node access req/s (single repo access)
- Switch HW: 8 cores, 8 Gb RAM

See benchmark details in backup slides (Appendix A)

L7 Switch Benchmark

- Concentrates on L7 switch benchmark
  - Backend repos and SOLR are «fast enough» in the test
- 15k users with 30s think time
  - 125 browse req/s (federated)
  - 125 search req/s (SOLR)
  - 250 node access req/s (single repo access)
- Switch HW: 8 cores, 8 Gb RAM
- Results (user wait time):
  - Browse: < 3.9s
  - Node access: < 1.2s
  - Search: < 9.7s

See benchmark details in backup slides (Appendix A)

Distributed system in production

- Docker + Ansible: nice but not enough
- JGroups for auto-discovery
- Pushing configs from the central location
- Reconfigure running nodes without restart when a member joins / leaves federation
- Basic safety checks against human mistakes (like adding repos with incompatible code versions into the same federation)

See more details in backup slides (Appendix B)

Summary

- Yes, you can have sharded Alfresco
- Sharding in an option to solve storage problem for really large repository

Summary

- Yes, you can have sharded Alfresco
- Sharding in an option to solve storage problem for really large repository
- Sharding is not a silver bullet, there are limitations, one size does not fit all

Summary

- Yes, you can have sharded Alfresco
- Sharding in an option to solve storage problem for really large repository
- Sharding is not a silver bullet, there are limitations, one size does not fit all
- Sharded Alfresco (Community 5.1.e) on commodity hardware can scale up to 15k concurrent users

Summary

- Yes, you can have sharded Alfresco
- Sharding in an option to solve storage problem for really large repository
- Sharding is not a silver bullet, there are limitations, one size does not fit all
- Sharded Alfresco (Community 5.1.e) on commodity hardware can scale up to 15k concurrent users
- We would like to share and re-implement the switch and indexer from scratch as separate open source projects, if the community is interested

BeeCon 2016

Thank you for your time and attention!

Alexey Vasyukov <avasyukov@itdhq.com>

BeeCon 2016

Appendix A: Benchmark Details

L7 Switch Benchmark: Infra

- 3 repositories
  - 1 separate PostgreSQL database server per repo
  - sharded content is not indexed by internal Alfresco SOLR, only by external SOLR Cloud
- 3 SOLR Cloud shards
  - + 1 ZooKeeper node
- 1 custom indexer server
- 1 custom L7 switch server

L7 Switch Benchmark: Limits

- Synthetic load with just 'ab' tool
  - Share servers exist in real setup, but they are not included into benchmark
- Quite simple SOLR queries
  - Not a real nested objects benchmark yet
- Backends benchmark is out of scope
  - They are «fast enough» to make L7 switch a bottleneck
  - Real system performance data is the customer property
  - Real system has a lot of deep repo customizations, this data is not relevant for any other use case

L7 Switch Benchmark: Load

- 15k users:
  - 25% browse (federated)
  - 25% search (SOLR)
  - 50% access data (single repo)
- Think time 30 seconds:
  - 125 browse req/s
  - 125 search req/s
  - 250 node access req/s
- Load lasts for 1 hour

L7 Switch Benchmark: HW

- L7 switch hardware
  - 8 cores
  - 8 Gb RAM

L7 Switch Benchmark: Results

- Browse (ms):
  - 50% 969
  - 66% 1181
  - 75% 1345
  - 80% 1447
  - 90% 2074
  - 95% 2624
  - 98% 2953
  - 99% 3319
  - 100% 3895 (longest request)

L7 Switch Benchmark: Results

- Node access (ms):
  - 50% 423
  - 66% 515
  - 75% 592
  - 80% 662
  - 90% 821
  - 95% 895
  - 98% 962
  - 99% 1065
  - 100% 1165 (longest request)

L7 Switch Benchmark: Results

- Search (ms):
  - 50% 3180
  - 66% 3708
  - 75% 4059
  - 80% 4291
  - 90% 5134
  - 95% 5659
  - 98% 6674
  - 99% 7571
  - 100% 9638 (longest request)

BeeCon 2016

Appendix B: Production Consideration

Distributed system in production

- We do use Docker + Ansible
- It's not a silver bullet and it's not enough
  - Components should detect and handle restarts and failures of each other
  - Adding new nodes into running federation — too many interconnections and relations, too many configs to edit on each server, existing nodes should be reconfigured
  - Handling human mistakes — starting new system while the old one is still running, starting new repos with the new application version while old repos are running the old version

Distributed system in production

- Auto-discovery with JGroups
  - Plus safety-check against multiple federations on the same subnet
- Pushing configs from L7 switch
  - Single config for sysadmin to rule the federation
  - On-the-fly reconfiguration of existing nodes (without restart) when federation topology changes
- Code version check on joining federation
  - Protection against running incompatible code versions in the same federation