Alfresco Best Practices

Luis Cabaceira, Solutions Architect at Alfresco

Alfresco Best Practices

A Simple and Clear set of rules for success

About me

Started in Alfresco in 2013 as a Senior Consultant and have been helping customers ever since. Currently a solutions architect specialized in large scale architectures, content migrations, benchmarking and distributed deployment scenarios.

- Alfresco Certified Engineer
- Alfresco Certified Administrator
- Java Certified Programmer
- Blogger in community.alfresco.com (Be sure to check my blog post on Alfresco best Practices)
- OpenSource and community driven mindset
- Some of my projects available in github.com/cabaceira and github.com/lcabaceira

1 – The Basics

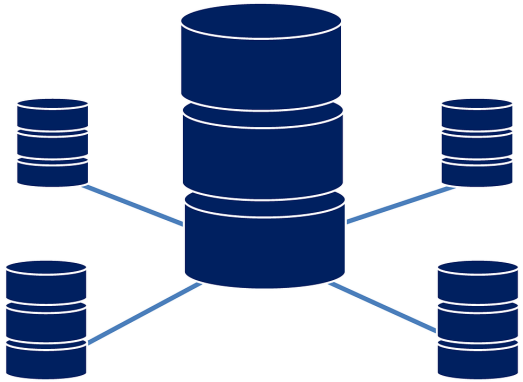
- Use only a supported Stack
- Own your code, your integrators may not be here tomorrow
- Recognize your team ! Your team is your strength ! It’s all about the people
- Maintain awareness of all Alfresco Service packs.
- Don’t be afraid to embrace changes in thinking and technology.

2 – Cherish your Database

- Alfresco is a highly transactional system, Database is key.
- Make sure your nodes round-trip communication to the database is below 2ms.
- Monitoring your database performance is very important as it can detect some possible performance problems or scaling needs.
  - **Slow Queries, query plans for the slow queries, create new indexes when applicable**
  - **Database sizing statistics (database growth, individual table sizes)**
  - **Update your database statistics regularly** or have Auto Update Statistics turned on.

3 – Plan, chose and size your Database Properly

Choose a database that can scale in size so it can serve the expected number of concurrent connections on your deployment. Specially when using a multi-node deployment.



Amazon Aurora

ORACLE RAC

PostgreSQL

MariaDB

MySQL

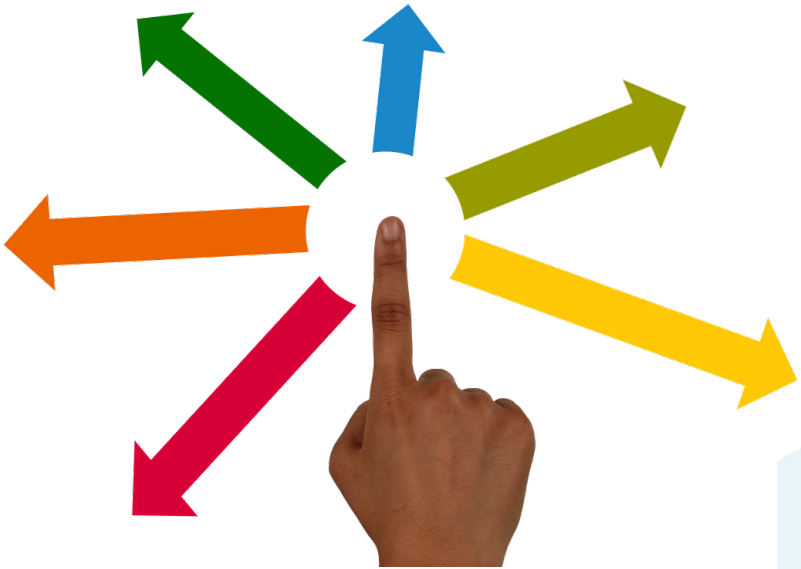
4 – Application Server

Your repository can easily get maxed out if you are setting the application server to process more connections/threads than the machine can handle.

Consider “bottlenecking/reducing” the number of application server connector’s threads in order that each call gets CPU and memory resources needed to perform properly, so that your system gets a more stable response time over peak usages (this should be tuned together with the database pool).

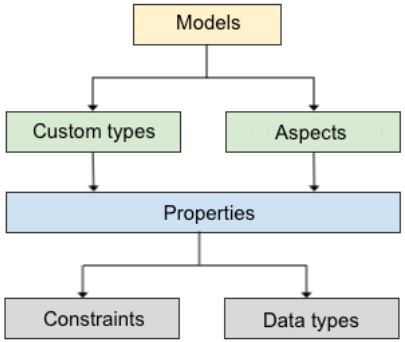
Its important having enough of RAM to minimize input/output. RAM is cheap, probably the cheaper hardware resource you can scale, but the exact amount will vary depending on your application/solution.

6 – Distribute your Load (Devide and Conquer)



One of the secrets of a successful architecture is to know exactly what, when and where is the “action” occurring and what resources are influenced by the running processes. Having this information brings the administrator the power to “Divide and Conquer”. Overall, don’t abuse of one layer of your architecture. It’s all about “BALANCE”

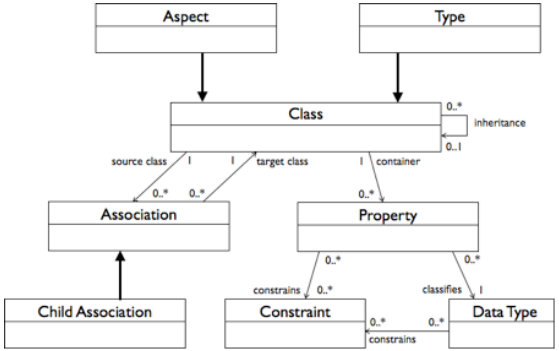
07 – Content Modeling Best Practices

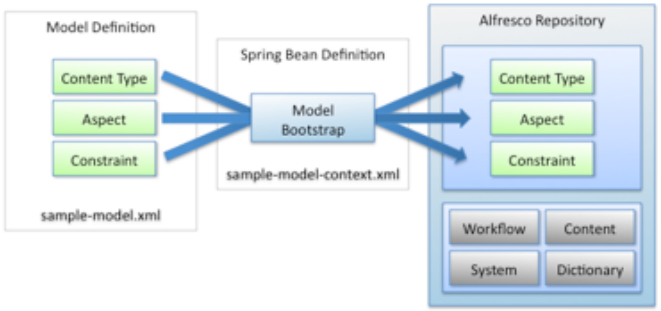


A effective tuned design of your content model can contribute an overall performance increase. Index **just the bare minimum** number of properties that are required for querying, also taking into account that the transactional metadata query mechanism doesn't require that the properties present on those queries to be marked as indexed.

Use an Aspect oriented model. Common properties in various content types should be included on aspects. Having properties on aspects gives you the flexibility to add the properties only to certain types and you can make aspects mandatory.

07 – Content Modeling Best Practices

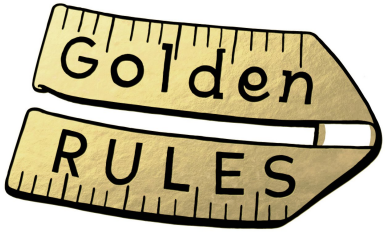




Don’t repeat the same properties on the content model - You should not define for different types on your content model the same property (in practical terms: same meaning, same constraints, etc.). Leverage the capabilities of aspects and type inheritance for just defining each property only once.

Use root types on your model - This will make it easier to separate your solution model content and folder types from others in the repository, what can be important for many reasons: policies, searches, etc.

08 – Content Modeling Golden Rules



- Leverage but do not change alfresco out-of-the-box content model
- In the beginning add only the properties that you really need as it is hard to remove properties later.
- Use aspects when you can, they are really flexible and provide lots of extra possibilities.
- Create multiple content models to keep them in order
- Implement one java class that corresponds to your custom model
- Remember Solr eventual consistency

09 – Repository Modeling and Design

Limit Groups hierarchy to 5 - Acl checks are known to slow down performance when the maximum group hierarchy depth exceeds 5 levels

Limit the maximum number of nodes in a single folder to 2000

Limit the folder hierarchy depth - The depth of the folder hierarchy also has an impact when browsing and performing document actions under a certain folder.

Keep a low ratio on user/groups membership

10 – Repository Management

Remove orphaned content - Alfresco by default won't delete any binary content of your repository. All content finally deleted from trash will be moved to the deleted content store folder on the file system and kept there till you delete it. Make sure you delete regularly old orphaned content in order not to occupy more file system disk space than needed.

Cron Expressions and Scheduled tasks- Review your scheduled jobs cron expressions, so that you confirm you are really scheduling your process as intended. On a cluster environment make sure you avoid triggering duplicated jobs from each Alfresco node, this should in general be done by implementing your custom scheduled jobs in a cluster aware mode.

11 – Repository Performance Best Practices

Disable un-used features - Disabling features that are not being used can release important resources allowing them to be used for active tasks, contributing for increased performance. For example :

**system.thumbnail.generate=false - ** Alfresco creates document thumbnails by default as part of any upload. If you are not making any use of thumbnails, disabling their creation can increase upload times, as no transformers will be involved.

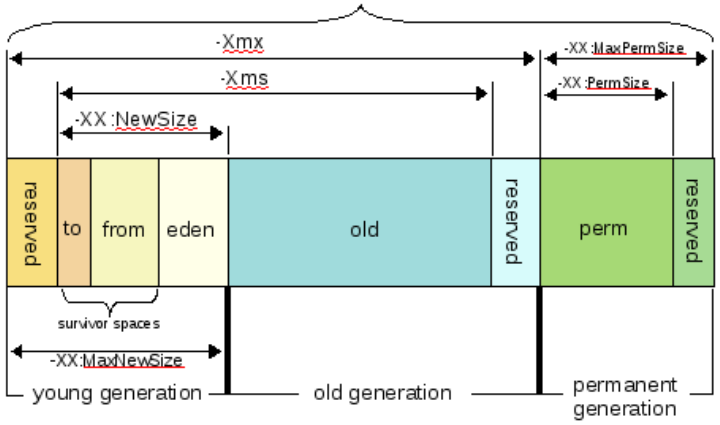
Disable user quotas - Checking for user quotas can add some overload to alfresco. If you are not using user quotas, this feature can also be disabled with (**system.usages.enabled=false and system.usages.clearBatchSize=0)**

Disable activities feed – When not needed, disabling the activities feed will prevent regular checks to the activities and will again save on system resources.

12 – Memory and Garbage collection

A regular analysis to the garbage collection logs is also a known best practice. The health of the Garbage collection engine is normally related with the overall effectiveness of memory usage across the system. This is valid for Alfresco, Solr and any possible client that is part of the deployment. Make sure you enable your garbage collection logs on your application server and you analyze those on a regular basis.

13 – JVM Configuration



Make your heap explicit : -Xms<heap size> -Xmx<heap size>

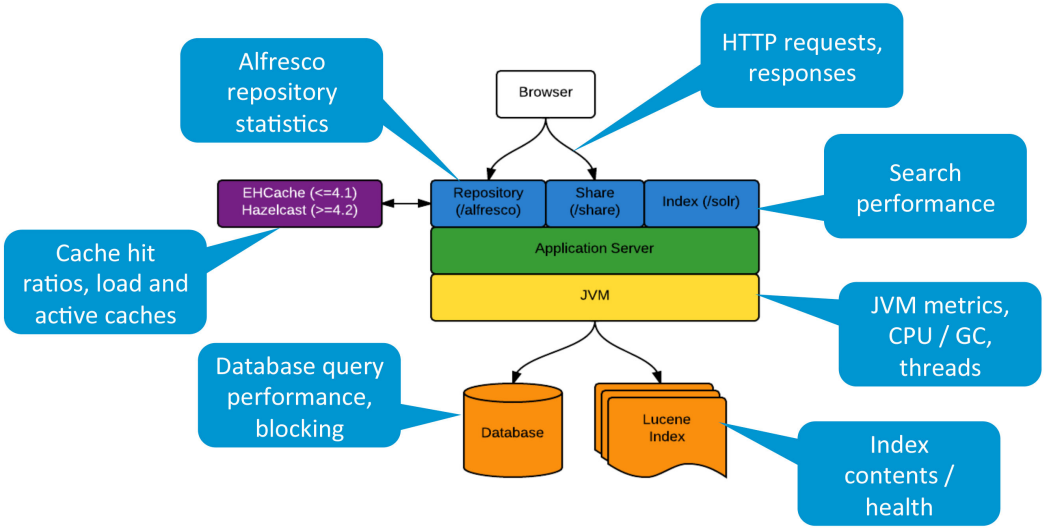
To avoid dynamic heap resizing and lags, you can explicitly specify minimal and maximal heap size. This way the VM will spend time only once to commit on the entire heap.

Choose a appropriate Garbage collector : CMS, G1, …

Dump on OutOfMemory -XX:+HeapDumpOnOutOfMemoryError

14 – Have a compreensive monitoring plan

The Monitoring schema is key, it helps you to know and understand your environment.



Identify your key performance indicators and build your own monitoring schema. Some samples are shown in the image above.

15 – Solr Best Practices

Use local and fast disks for the indexes (SSD/Raid).

Tune the mergeFactor, **25** is ideal for indexing, while 2 is ideal for search.

Avoid path search queries, those are know to be slow.Avoid using sort, you can sort your results on the client side using js or any client side framework of your choice. Avoid * search, avoid ALL search

Tune the transformations that occur on the repository side, set a transformation timeout.

Closely monitor Solr JVM (especially GC and Heap usage). Use a dedicated tracking alfresco instance, several architecture options

Use Sharding from 40 M onwards. It will have performance benefits and will result on a faster re-indexing process.

15 – Solr Best Practices

Analyze your Indexing process - During the **indexing process**, plug in a profiler tool (YourKit) to check the repository health during the indexing. Sometimes, during indexing, the repository layer executes heavy and IO/CPU/Memory intensive operations like transformation of content to text in order to send it to Solr for indexing. This can become a bottleneck when for example the transformations are not working properly or the GC cycles are taking a lot of time.

Have a large disk cache - For index updates, Solr relies on fast bulk reads and writes. One way to satisfy these requirements is to ensure that a **large disk cache** is available. Ideally have enough memory available in the OS disk cache so that the important parts of your index, or ideally your entire index, will fit into the cache.

Try to keep your index small - Housekeeping and tuning your indexes is an important part of application maintenance. You can blacklist un-used types.

16 – Security Best Practices

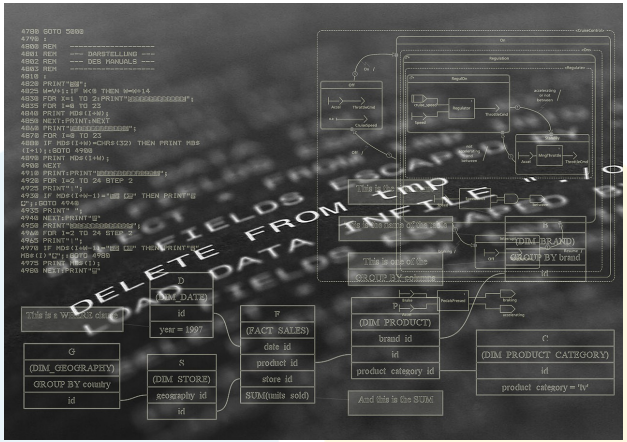
Perimeter Security - Make sure you set up firewalls properly. If you want your repository to be unavailable for direct end user access, remember that Share has a proxy that gives end users access to repository Rest API execution.

Default passwords - Apart from changing the default admin password, remember to change the DB and JMX access passwords.

Don’t run alfresco as root - Running Alfresco as a non root user will prevent the system from a compromise application or a failure on it, or even have a greater control of operating system resource consumption. It is important to configure all ports beyond port 1024 because of the security restriction for a non-root user running linux-like systems.

Use SSL for all Alfresco services, Disable the guest user, Limit the number of users in the repository, Set timeout as your convenience for Webdav, Share or CIFS

17 – Development Best Practices



- **Use version control, Use branches**
- **Make Unit Tests mandatory –** A crucial part of your development cycle the creation and execution of unit tests that test your solution individual pieces of code and also functional tests that will test broader scopes of your solution functionality.
- **Close your streams and resultsets, Use upper-case bean versions.**
- **Use transactions property –** Leverage RetryingTransationHelper.
- **Leverage SDK 3.0 (Finally Hot Reloading again !!!!.)**

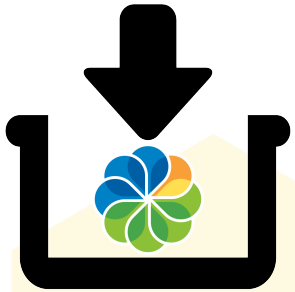
18 – Caching Best Practices

Alfresco now uses Hazelecast clustering and caching. The database is now used as the central place of discovery for cluster nodes. 3 types of cache strategies possible : **fully-distributed, local and invalidating**

Fully-distributed - This is the normal/default value for a hazelcast cache. Cache values (key value pairs) are evenly distributed across cluster members. Leads to more remote lookups when a get request is issued and that value is present in other node (remote).
Local - Some caches you may not actually want them to be clustered at all (or distributed), so this option works as a unclustered cache.
invalidating - This is a local (cluster aware) cache that sets up a messenger that sends invalidation messages to the remaining cluster nodes if you updated an item in the cache

To perform a cache tuning exercise we need to analyze 3 relevant factors : **type of data**, **how often it changes**, **number of gets compared to the number of writes**

19 – Content Ingestion Best Practices



Fastest way for ingesting content into alfresco is via bulk imports. Use the latest version of the bulk-import tool in https://github.com/pmonks/alfresco-bulk-import as it contains bug fixes and enhancements.

- Source disk, target disk and the database are key factors, try to maximize their I/O. Do in-place imports whenever possible.
- Database statistics and Index statistics will suffer, so break the imports into chunks. Make sure to update statistics after each chunk, this will be the key to performance and high ingestion rates, not doing so may result in full table scans and poor database performance.

Speaker contacts

luis.cabaceira@alfresco.com