Alfresco Enterprise 3.2 and Above

Scaling your Alfresco Solutions - Best Practices

Table of Contents

Introduction

This Alfresco Field Series document describes scalability best practices, measurements, and benchmarks for large Alfresco Enterprise content repositories that typically contain millions of objects.

The benchmarking approach is empirical and based on scenarios gathered from the experience of Alfresco Support, Professional Services, and the Alfresco Partner network. It has important consequences for the business applicability of Alfresco in large organizations.

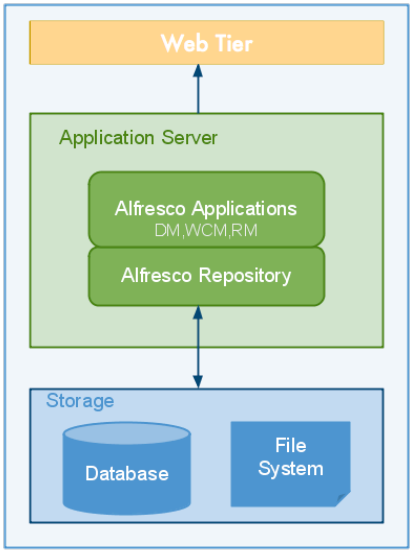
Audience

This document is for anyone who needs to customize and integrate Alfresco. It is based on Alfresco Enterprise 3.2 and above and assumes the availability of a full-featured Alfresco Enterprise instance. However, the concepts and best practices apply to the Community release as well.

The target audience includes architects and technical resources with an appropriate level of experience in Alfresco development and administration. The information in this document assumes in-depth knowledge of Alfresco concepts, design, and core architecture.

Prerequisites

To fully understand the terminology used in this document, you must have knowledge of the Alfresco content platform core architecture. The following diagram shows a high-level view of this architecture.



Alfresco recommends that you understand the following as a minimum for this document.

- Alfresco Repository Architecture
- Repository Hardware
- Repository Configuration
- Administering an Alfresco ECM Enterprise Edition 3.2 Production Environment

An understanding of Alfresco High Availability is desirable as a full explanation of clustering is outside the scope for this document.

Document scope

This document is organized as follows:

- Chapter 1 provides an overview of this document.
- Chapter 2 describes best practices for managing the architecture, design, and technical tuning of Alfresco and its components, and offers pros and cons for different architectures.
- Chapter 3 describes real life scenarios for implementing best practices on large repository deployments and provides an overview of tools for stress testing, benchmarking, and profiling deployments.
- Chapter 4 provides a list of references for background reading.

The scope of this document excludes WCM/AVM content.

Terminology

The following terminology is used for architectural, design, and technical suggestions.

Term	Definition
MUST	The specific pattern, configuration, or best practice is required to meet the current solution requirements.
MUST NOT	The specific pattern, configuration, or best practice must not be applied to the current solution requirements, as it will harm the functionality or expectations.
SHOULD	The specific pattern, configuration, or best practice is not mandatory, but suggested as a viable addition or improvement to improve the chance of meeting the full requirements set for the current solution.
SHALL/SHOULD NOT	The specific pattern, configuration, or best practice should be used only with specific reasons, acknowledging all consequences and risks of the choice for a solution that normally does not suggest that usage.
MAY	The specific pattern, configuration, or best practice is optional for the solution under analysis, but presents some collateral benefits or side effects to consider while designing the solution.
CANNOT	The specific pattern, configuration, or best practice cannot be used to fulfil the current requirement.

Glossary

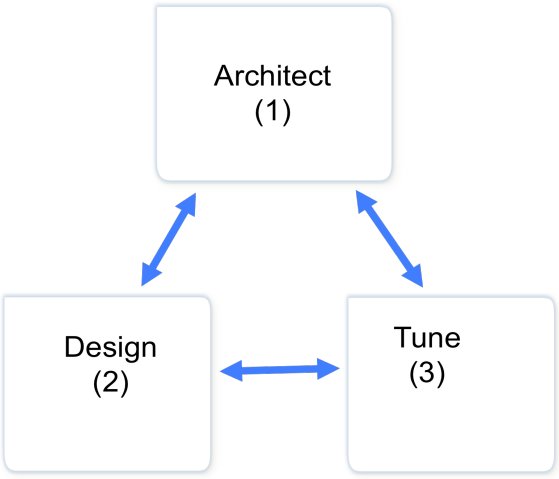
This document uses the following terms within an Alfresco context.

Term	Definition
ACL	Access Control List
CMS	Content Management System
FSR	File System Receiver, an Alfresco system component that handles deployment of Alfresco managed content to a file system
WCM	Web Content Management, specifically the Alfresco Web Content Management functionality
Web project	A special type of space within Alfresco dedicated to managing a set of web content assets usually synonymous with a single website
DM	Document Management
RM	Records Management
CIFS	Common Internet File system
ADM	Alfresco Document Management
AVM	Alfresco Versioning Model
CRUD	Create/Read/Update/Delete
POC	Proof of Concept
SMB	Small and Medium Businesses
ECM	Enterprise Content Management
JDBC	Java Database Connectivity
DOS	Denial Of Service
HA	High Availability
KPI	Key Performance Indicator
ILM	Information Lifecycle Management
OLAP	Online Analysis Processing
OLTP	Online Transaction Processing
VFS	Virtual File Systems
RAM	Random Access Memory
SPP	SharePoint Protocol

Scaling Alfresco for large content repositories

This section introduces the Alfresco architecture and defines the scope and objectives for its architectural components. It also defines a common vocabulary and set of patterns to use for improving performance at the design stage before attempting to optimize your Alfresco solution and run benchmarks on it.

Tuning Alfresco to scale for supporting large repositories comprises best practices at three levels: architectural, design, and technical tuning. These levels must be managed in a continuous feedback and optimization loop to build the appropriate solution for specific requirements.



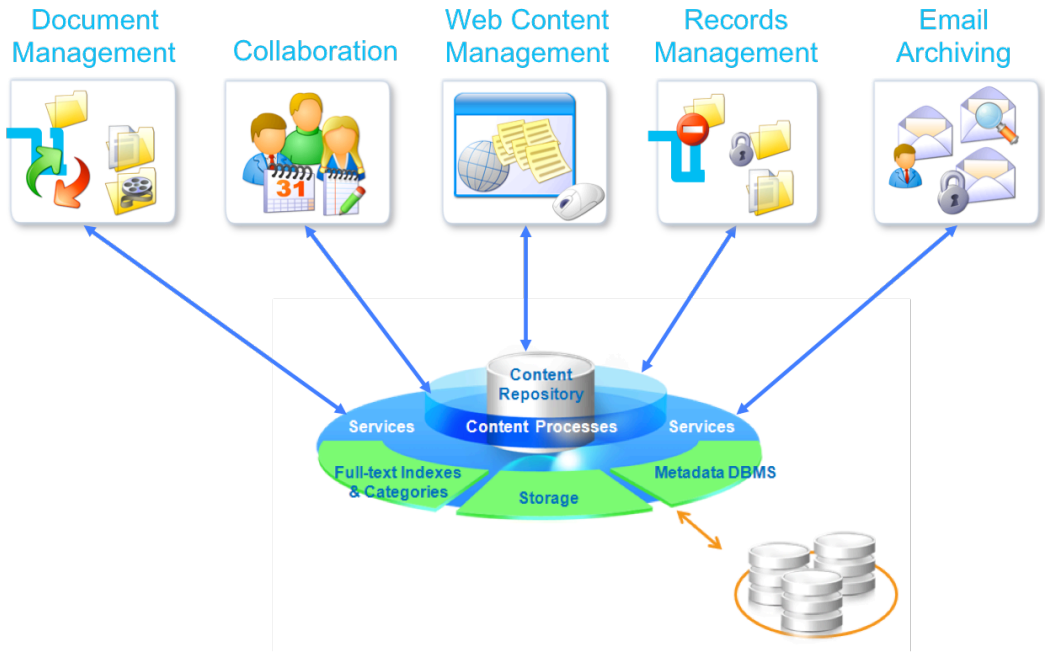
Choosing an Alfresco architecture

Alfresco delivers a complete Enterprise content platform capable of offering a reliable, scalable, and distributed content repository and delivery architecture. The concept of content platform or content application server fits perfectly in the technologies and features that Alfresco provides, as it sets itself as a comprehensive repository for almost any type of enterprise process requiring the manipulation of structured or unstructured content.

This means that the same standardized repository and access layer powers content interaction for the following:

- Document Management (DM)
- Web Content Management (WCM)
- Records Management (RM)
- Social and Enterprise Collaboration (Share)
- Email Archiving (IMAP)
- User synchronization and management

The following diagram shows an overview of the Alfresco content platform.



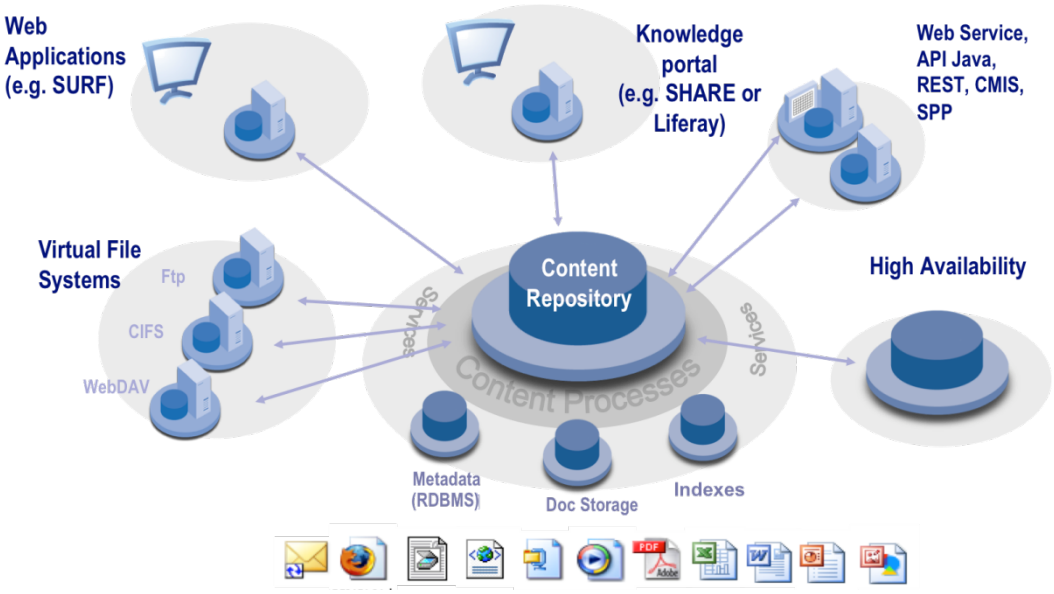
The Alfresco repository design encapsulates architectural quality, scalability, and ease of integration. This allows the developer to extend the content platform to include custom content-centric processes, through user and developer interfaces. The tools available include:

- Spring Service Layer Foundation APIs
- RESTful Public APIs
- SOAP Content Web Services
- CMIS API
- Alfresco Share
- Surf custom applications
- Virtual file systems (CIFS/FTP/WebDAV/NFS)
- SharePoint Protocol (SPP)

The Alfresco repository is suitable for any content-oriented vertical application, such as:

- Publishing
- Content delivery through an Enterprise intranet/extranet/portal
- Bulk scanning
- Migration and consolidation from legacy systems
- CEVA (Content Enabled Vertical Applications), such as:
  - ERP
  - CRM
- Any Enterprise process that manipulates content

The following diagram illustrates the possible ways Alfresco can power a content-oriented architecture.



When properly scoped in an Enterprise context, Alfresco can be populated with hundreds of millions of objects. This document provides a best practice guide for tuning Alfresco to serve the size and load of your content processes most effectively and help you to:

- Choose the best architecture for real life scenarios
- Tune your infrastructure (JVM, database, application server) for optimum Alfresco performance
- Use scalability best practices for using and developing Alfresco customizations
- Identify thresholds and indicators for scaling up and out

Out of Scope

WCM is out of scope for this document. Future versions may include a complete DM/WCM benchmark/best practices guide.

Any discussion around file server clustering, including CIFS, FTP, and WebDAV is out of scope for this document as well. Contact Alfresco Support for best practice advice before clustering these items.

Customization and integration

To realize the potential of Alfresco while meeting the needs of your business-specific logic and environment, you typically perform customization and integration while deploying the alfresco content platform. This section discusses strategies for these two practices.

Alfresco content platform customization APIs and patterns

The following typical customization use cases are discussed:

- Core services personalization
- Integration of external services
- UI customization

Alfresco offers various public APIs that define the different approaches.

Java Foundation API

Java-based APIs are available via the Spring repository service layer. Customizing Alfresco based on this approach involves packaging custom Spring beans and configurations, in the Alfresco Extensions directory, which override or add functionality together with the alfresco.war file, or creating and deploying Alfresco Module Packages (AMPs).

Pros and cons

This approach has the following advantages and disadvantages.

Pros

- Customization acts directly in the repository runtime without any external process
- You MUST use for Alfresco Explorer UI customizations or core integration with custom business logic services acting at low levels (such as persistence, import, and authentication)

Cons

- Requires development, J2EE knowledge, and development cycle expertise
- Might impact core behavior and service availability if misused
- You SHOULD NOT use for pure User Interface (UI) customization or for integrations possible through other mechanisms (such as configuration or a simple UI mashup)

JavaScript API

JavaScript APIs wrap the Alfresco repository service layer in a JavaScript object model for use in the Alfresco Rules and Actions engine or the Spring Web Script Framework to perform remote and local pieces of business logic without involving any Java code.

Pros and cons

This approach has the following advantages and disadvantages.

Pros

- Simple, powerful, and incremental functionality with every version
- You SHOULD use rules and actions for performing automated business logic on repository events
- You SHOULD use web scripts to produce custom DM remotely invoked (HTTP) functionality
- You MAY use for low-level Alfresco Explorer UI customization where JSF customization is not possible

Cons

- Does not expose all low-level internals and the pattern of exposing custom root objects
- Useful in certain cases only
- Generally, you SHOULD NOT use the rules/engine JavaScript approach to perform internal business critical/high load customizations; instead, you SHOULD handle this by directly accessing the Java Foundation Layer

Surf API

The Surf API is the framework for composing pages and the dispatching and retrieving of remote content. It is the basis of the complete Alfresco Share interface, providing a modular, scriptable, and dynamic framework for a simpler and more scalable pattern for customizing content delivery via Alfresco Share customizations, or by creating custom front-ends with Surf components/pages. You MUST use this approach for customizing Alfresco Share functionality and SHOULD consider it the best option for heavy UI customizations or for building custom Alfresco content delivery and front-ends.

Pros and cons

This approach has the following advantages and disadvantages.

Pros

- Modular, scriptable, and completely based on Spring MVC
- Growing, due to the contributions from the large, open source Spring community. SHOULD be used to build reusable front-end components and to connect to external mashup data sources
- You SHOULD use to build remote interfaces to Alfresco decoupling by distribution to the Alfresco repository and web tier

Cons

- Currently, development of completely custom Surf front-ends may require a high degree of expertise in the framework

Alfresco content platform integration APIs and patterns

Integration is a broad term in Enterprise framework deployment contexts. This section covers typical integration use cases, such as:

- Enterprise authentication and SSO
- External systems
- Custom front-ends
- SOA architectures

Alfresco local and remote APIs mandate the various patterns possible for these kinds of integrations.

Subsystems API

The internal (in process) API for the Alfresco repository (from Alfresco 3.2) modularizes Alfresco core functionality in independent subsystems. This allows various out-of-the box configurations to integrate with Enterprise authentication and external third-party applications used by Alfresco. With Alfresco Enterprise, the JMX interface can manage these configurations dynamically without requiring a server restart.

Pros and cons

This approach has the following advantages and disadvantages.

Pros

- Configurability and modularity of subsystems offer many integration hooks for Alfresco platforms and you MUST use it to integrate within an Enterprise context
- You MAY use subsystems to encapsulate new functionality in Alfresco and benefit from the high configurability and modularity of the approach and you SHOULD use to integrate external systems into the Alfresco repository functionality

Cons

- Requires in-depth knowledge of Alfresco
- Might impact core internals when not used carefully
- You MUST NOT use for UI customizations

RESTful API

The RESTful API wraps Alfresco core functionality around remotely invoked URLs via GET, POST, PUT, and DELETE HTTP methods. This is Alfresco's most complete out-of-the-box remote API and services, which among other things serves as a remotely callable API for Alfresco Share front-end services. The RESTful API is based on web scripts and SHOULD be considered the primary interface for applications to integrate with the Alfresco content platform.

Pros and cons

This approach has the following advantages and disadvantages.

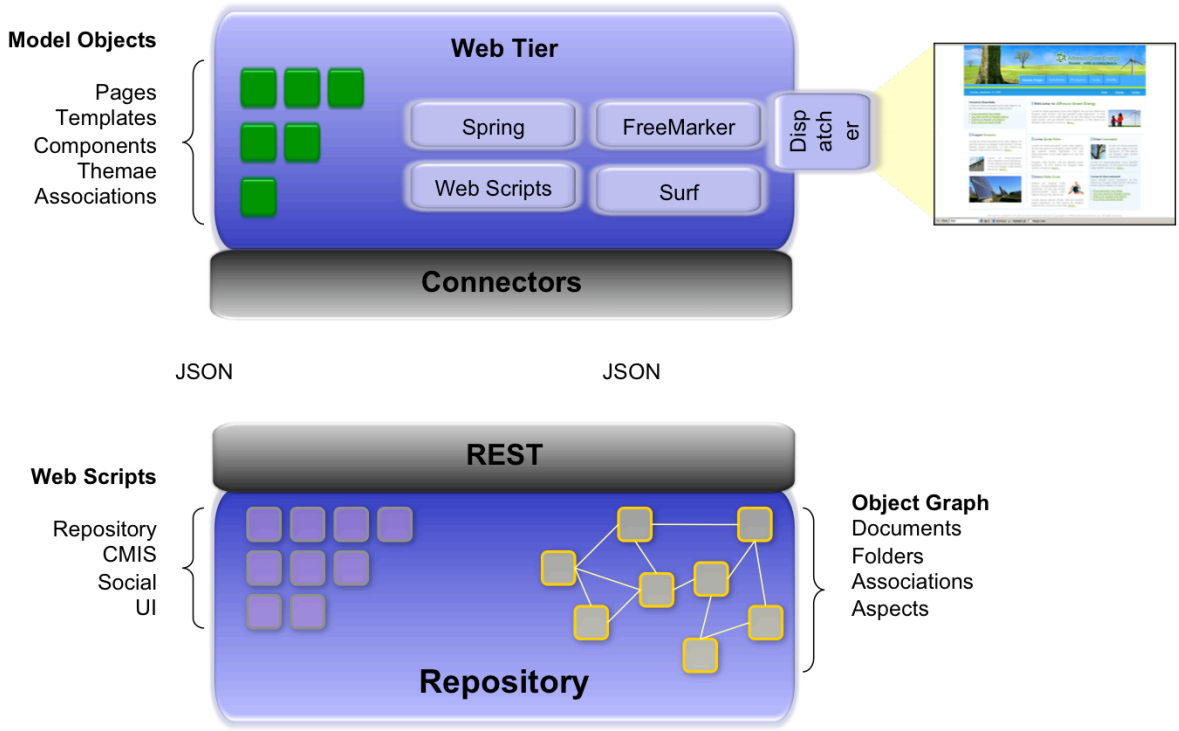
Pros

- REST paradigm offers a high performance, lightweight, and remote integration pattern
- The scriptable, extensible nature of the underlying Web Script Framework allows easy customization and extension of API functionality
- You SHOULD use the REST paradigm for any kind of remote invocation/integration with the Alfresco platform; you MUST use web scripts to extend Alfresco Share with new content-oriented functionality that require extending the RESTful API on the repository side
- You SHOULD use web scripts/REST to extend the RESTful API on Alfresco Share and the repository

Cons

- Does not yet support full set of Alfresco capabilities
- Non-standard and Alfresco proprietary when compared against the CMIS API

The following diagram highlights how a pure web-tier application (like Share) can use its own scripts and templates, then use JSON to talk to the REST APIs as the access point to the repository.



SOAP API

Alfresco Content Management Web Services offer an Alfresco-proprietary SOAP interface to expose the repository functionality to external consumers via WSDL exposed services. A Web Service client is offered with every Alfresco release to support embedding in consumer applications and remote interaction with the Alfresco repository.

Pros and cons

This approach has the following advantages and disadvantages.

Pros

- Compatible with SOA infrastructures
- Provides simplicity of use via the Web Services client
- You SHOULD use in SOA enterprise consolidated infrastructure and for fast integration with the Alfresco content platform

Cons

- Less complete than the RESTful API
- Non-standard and Alfresco proprietary when compared against the CMIS API
- Impacts performances as regards the RESTful API due to the SOAP marshalling/unmarshalling overhead
- You SHOULD NOT use when developing custom UI functionality (SHOULD use RESTful API), or when standard content-oriented interaction is required (SHOULD use CMIS Web Services API)

CMIS API

The CMIS standard is the solution for a protocol independent, product agnostic, and complete API for content management. It supports both REST and SOAP bindings to provide a common model and set of services for content management interoperability.

Backed by major ECM vendors, CMIS is expected to become the standard for all systems that require content-oriented interaction. This includes delivery front-ends, vertical solutions, or repository federations. Client libraries are under development by multiple vendors, and the open source community is concentrating implementation efforts in Apache Chemistry, specifically in the recently contributed OpenCMIS.

CMIS 1.0 (Content Management Interoperability Services) was declared an OASIS standard on May 2nd 2010): CMIS 1.0 server support is available in Enterprise versions from Alfresco 3.3 onwards. Nevertheless, the Alfresco CMIS implementation was the first and most complete CMIS repository server and is constantly evolving with the latest version of the specification.

Pros and cons

This approach has the following advantages and disadvantages.

Pros

- Standard, cross protocol, and vendor independent
- Avoids one-off integrations and promotes reuse of integrations
- You SHOULD use for any type of content-oriented type of CRUD application and integration, especially for typical cases of integrating different ECM systems
- It MAY be useful in content migration between different systems

Cons

- You SHOULD NOT use for BPM, WCM, and RM unless custom extensions/mapping code are developed

  Note: CMIS 1.0 scope is limited to common content management functionality. However, the scope of CMIS 2.0 may include BPM, WCM, and RM.

Recommendations for a sustainable build and customization strategy

This section provides recommendations to improve the sustainability and speed of your development and the overall reliability and longevity of the applications you build around Alfresco. When developing a custom Alfresco application, you:

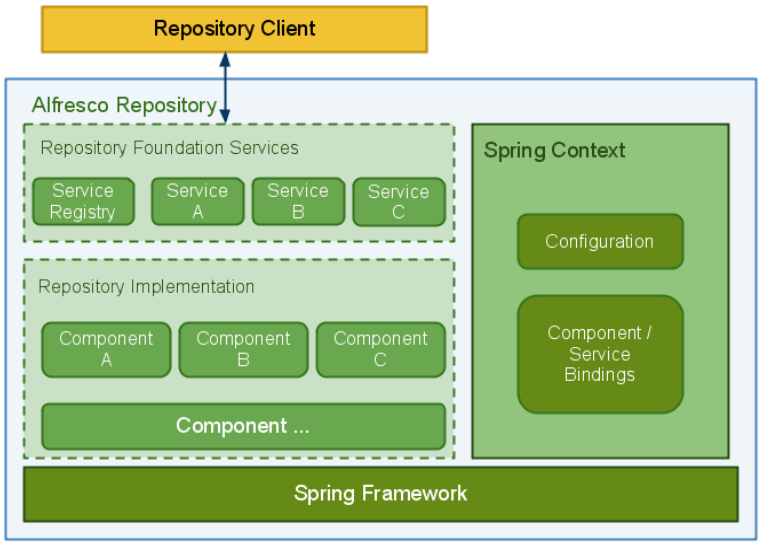
- You SHOULD design a development process that keeps the Alfresco provided binary software and customizations/configuration source code separate, merging them at build time to provide a runtime customized application. This facilitates Alfresco Support and simplifies the upgrades to newer Alfresco versions and ensures that what is running as the core has been fully Quad's by Alfresco.
- You MUST NOT overwrite classes or Alfresco configuration.
- You MUST override Alfresco default files and hook in to Alfresco provided extension points.

Introduction to Alfresco content platform architecture

This section describes the architecture (or *deployment models*) for the Alfresco content platform, analyzes the pros and cons for each architecture, and explains common integration/personalization technologies to meet specific solution requirements.

The architectural level is the first and best way to design Alfresco content platform solutions that are properly scalable and sized. Choosing the appropriate deployment model is a prerequisite for all other best practices presented in this document.

The following Alfresco repository overview architecture provides an overview and serves as reference for the following sections.

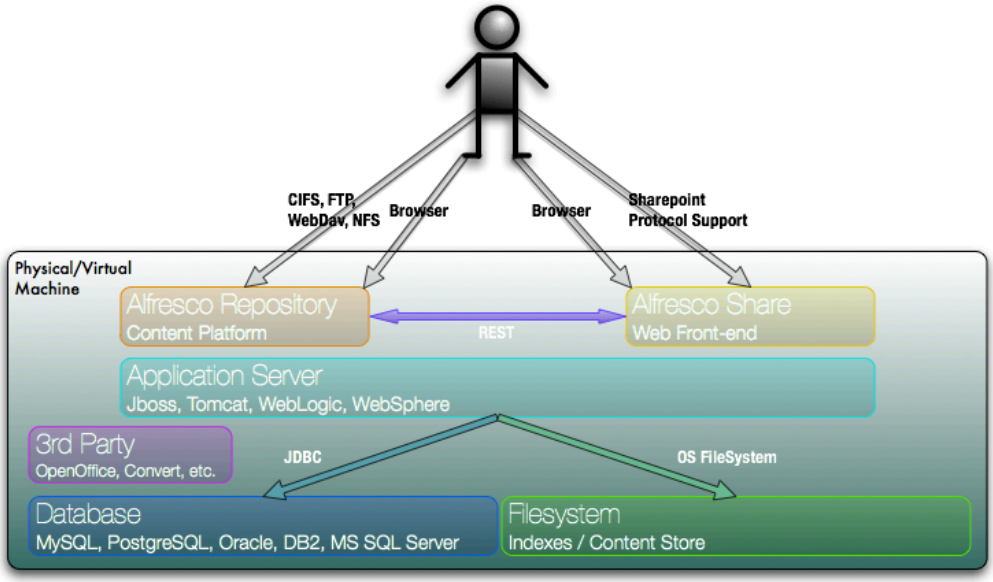


Basic deployment model

In this basic deployment model of the Alfresco content platform, a single host server (physical or virtual) runs the entire stack necessary for a working Alfresco system: Application Server, database, file system, and ancillary 3rd party executables. Alfresco installers can provide tested and supported versions of all the components required to run Alfresco using this deployment model.

Architecture description

The following figure illustrates this basic deployment model with the different components of the Alfresco content platform and support infrastructure.



These components are:

- Application server selected from the supported stack running:
  - Alfresco repository (content platform)
  - Alfresco Share (web front-end)
- Third-party Alfresco applications:
  - Open Office for document transformation
  - ImageMagick (convert) for image transformation
  - Pdftools for PDF conversion
- Database of choice
- File system on local machine for content store and indexes

Users access Alfresco Share or the repository through a web browser or a supported access protocol, (such as CIFS, FTP, NFS, WebDAV, SPP), while the RESTful API ensures communication between Alfresco Share and the repository.

Pros and cons

The basic deployment model is simple and easy to maintain, but presents some scalability challenges for large enterprise deployments.

Pros

- Simple maintenance and backup procedures
- Reduced network traffic for Repository →Database and Repository → Share communications

Cons

- Pure front-end load (such as mostly high concurrent Reads) impacts drastically on overall platform performances as there is no back-end/front-end distribution
- Database on the same machine might reduce and speed up transaction I/O, but will become a scalability issue when the number of concurrent users or the size of the content increases
- Sizing and HW selection becomes difficult as RDMBS and the content platform are on the same machine and have different requirements
- Disaster recovery and service continuity are harder to achieve unless some virtualization OS level functionality is used
- Content growth on a local FS might require resizing of physical storage or mandating use of the Content Store Selector
- Low security due to exposure of low level services (for example, database) on application server machine

When to use this deployment model

A basic deployment model MAY be a valid option for POCs and small departmental installations. Considering the inherent scalability of Alfresco, this model MAY be used for powering a full SMB installation when properly supported by backup/restore procedures at the OS level.

Large and mission critical contexts SHALL NOT use this basic model due to its obvious limitations.

Simple distributed deployment model

This deployment model introduces the standard practice of architectural distribution for any Enterprise production environment. For example, it separates the content platform tier where

Alfresco products are deployed from the storage tier where content (data + metadata) is actually stored.

This architecture only requires simple alfresco-global.properties configuration to set up.

Note: Alfresco Installers have an Advanced Mode, which allow you to configure a remote database running on the storage tier during an Alfresco installation.

Architecture description

The architecture allows deployment of the content platform on a separate physical/virtual environment from the data storage.

Typically, these are deployed as follows:

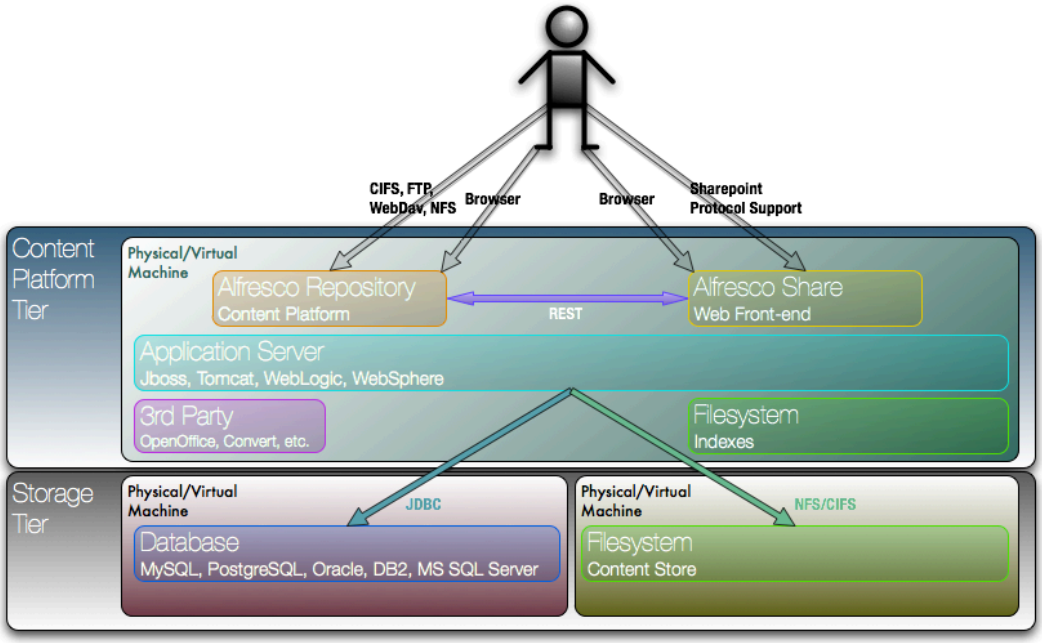
Content platform

- Application server of choice running
  - Alfresco repository (content platform)
  - Alfresco Share (web front-end)
- Local indexes storage
- Third-party Alfresco applications
  - Open Office for document transformation
  - ImageMagick (convert) for image transformation
  - Pdftools for PDF conversion

Storage platform

- Dedicated database machine of choice accessed via JDBC
- File system on SAN/NAS using CIFS/NFS or storage specific access protocols

From the user perspective, access does not change from a front-end full content platform. However, the inherent scalability and load distribution across different machines allows this deployment model to scale better than a basic deployment.



Pros and cons

A simple distributed architecture is a better option for Enterprise deployments where:

- More scalability and load is required
- Different resources, such as database and SAN/NAS, are shared and can be reused virtually cost-free

Pros

The main advantages for choosing this deployment model include:

- Simple configuration by editing core alfresco-global.properties
- Manage content growth externally without impacting the content platform service
- Can size the database and the content platform independently
- Improved security due to distribution of the storage layer
- Low network latency between Alfresco Share and the repository (loopback interface)
- Local indexes ensure fast I/O retrieval during searches

Cons

This architecture presents some obvious scalability or Enterprise-readiness disadvantages:

- Front-end load impacts the overall platform performances as no back-end/front-end distribution is performed
- Local indexes might increase in cases of large content stores and require infrastructural changes (or virtual machine reconfiguration)
- Front-end/back-end security can be an issue in controlled environments as there is no machine separation between front-end services (user accessed) and the content platform tier
- To scale out better, consider separating the actual application server instance in two instances running the Alfresco repository and Alfresco Share, respectively

When to use this deployment model

Medium-sized installations SHOULD use the two-tier deployment model and MAY use it for enterprise-wide deployments. Used with clustering, this deployment model SHOULD be considered for HA environments and requirements.

This deployment model SHALL NOT be used where front-end load is expected to grow. Instead, consider a front-end/back-end separation. For security reasons, this deployment model SHALL NOT be used when the platform is facing the WWW as it might expose the repository to intrusion or DOS attacks.

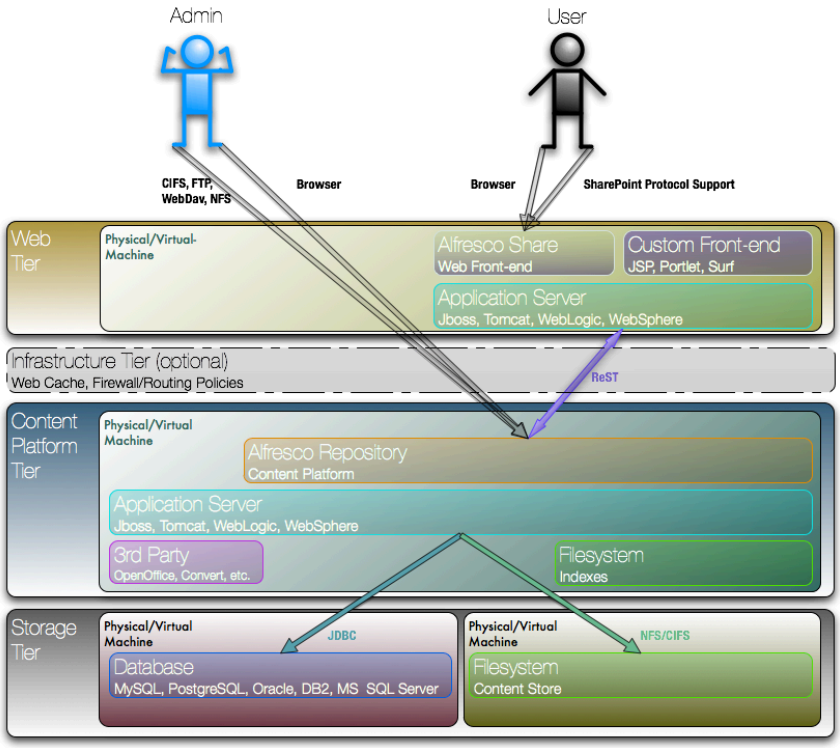
Distributed content platform deployment model

Note: This architecture is available from Alfresco 3.x and allows deployment of a separate web tier, such as Alfresco Share.

The distributed content platform model is the most scalable deployment model for the Alfresco content platform as it exploits the most of its load distribution capacity and out-of-the-box features. It is also inherently ready for clustering and high availability policies as it presents logical and physical separation between the web and content platform tiers. This architecture enables independent growth for different platform components, and can be integrated in Enterprise infrastructures where security and optimization are key performance indicators.

Architecture description

The overall architecture for the distributed content platform or three-tier deployment model is shown in the following diagram.



The three-tier deployment model architecture introduces an important scalability factor by distributing the web tier. There are three logical/physical tiers:

Web tier

- Application server of choice running
  - Alfresco Share (web front-end)
  - (Optionally or alternatively to Alfresco Share) Any custom front-end

Content platform tier

- Application server of choice running
- Alfresco repository (content platform)
- Local indexes storage
- Third-party Alfresco applications
  - Open Office for document transformation
  - ImageMagick (convert) for image transformation
  - Pdftools for PDF conversion

Storage tier

- Dedicated database machine of choice accessed via JDBC
- File system on SAN/NAS using CIFS/NTFS or storage specific access protocols

An optional infrastructure tier provides an additional benefit of this infrastructure — the possibility of adding an intermediate layer between the web and the content platform tiers to support:

- Enforcement of security policies via firewall or routing rules
- Web caching and load balancing layers that MAY be considered for improved scalability, reliability, or performance

User access has two distinct roles: a normal user and an admin. Both roles have different access rights (typically internal and external) and access different user interfaces. Certain admin functions are still performed in Alfresco Explorer, while users only access the Alfresco Share web-facing interface. It is important to understand that this is just one typical option for role deployment. You can implement different security models by configuring the Alfresco content platform.

Pros and cons

This model offers the following advantages and disadvantages.

Pros

The nature of the distributed content platform deployment model offers the most scalability and functionality. Other advantages for choosing this architecture are:

- Completely distributed web tier allows scaling up and out with high front-end load, allowing user requests to hit the content platform only when needed
- Independent sizing and clustering for the web and platform tiers as they live in separate environments and application servers
- REST integration between Alfresco Share and the repository offers a performing and scriptable remote integration paradigm
- Intermediate caching layer allows improved scalability and performance
- Alfresco Share can undergo a completely different lifecycle (frequent UI changes) from the content platform (corporate information model)
- Multiple content-centric applications can coexist and run against the same content platform

Cons

There are few disadvantages, mostly concerning the complexity of the infrastructure:

- Remote REST communication might introduce a small latency degree in repository/Share interaction
- Infrastructure can grow in costs and maintenance complexity

When to use this deployment model

Enterprise-wide deployment MUST consider the distributed content platform, and it SHOULD be considered an extended enterprise-wide intranet/extranet, particularly around collaboration.

This deployment model SHOULD be considered when building a high load, mainly read, content delivery application that MAY benefit from the web-caching layer and load balancing.

When Alfresco Share functionality does not match requirements, development of a custom front-end application SHOULD be considered. However, you MUST consider customization of this interface with Surf components, given Alfresco Share configurability and flexibility. When building a custom front-end, using Spring Surf SHOULD be the easiest way to build content-centric applications.

This deployment model SHOULD also be considered when the custom front-end is built on a JSR-168/286 portlet container, either deploying and exposing Alfresco Share on the portal container (using 3.2r ProxyPortlet features), or developing the custom portlet using Spring Surf or any remote integration API.

Do not use this deployment model for small departmental installations and you MUST avoid it in cases where the degree of Alfresco experience and availability of Alfresco skilled resources is low, since it will require configuration and administration knowledge.

Alfresco Share and the repository MUST NOT reside in different geographical locations for this deployment model as this could introduce high latency.

Distributed content platform as reference architecture

This document uses the distributed content platform architecture as reference to discuss scalability for addressing large Enterprise repository deployments.

Scale out and scale up

This section discusses how Alfresco can scale out and scale up deployment infrastructure to support large content repositories.

While the notion of scaling up becomes clear in the context of "growing the CPU/memory/storage resources allocated to machines hosting the Alfresco content platform components", it is worth mentioning which steps scaling out can take in your context, and which of these steps are specific to Alfresco.

Scaling up

Typically, the decision to scale up the Alfresco content platform is driven by additional performance requirements in user numbers, transactions to handle, or content size.

Scaling up comprises multiple possible steps but generally comes down to allocating:

- More CPU to a component
  This can mean either running a more powerful processor or increasing the number of processing units. Note that choosing the wrong way to increase your CPU power might degrade performance (for example, where you want to increase parallel computation capabilities).
- More memory to a component
- More storage to a component (where applicable)

Referring to the three-tier distributed content platform architecture, scaling up the Alfresco platform can be achieved by allocating:

- **More CPU to the content platform tier**
  Increasing the number of processing units SHOULD be considered when the content platform is under high concurrent load, composed mainly of Writes (mainly Reads SHOULD be handled in the web tier). Running a more powerful processor SHOULD be considered when there is a requirement for frequent CPU intensive operations (for example, document transformations or long serial operations, such as creating complete hierarchical folder structures from templates).
- **More memory to the content platform tier**
  You SHOULD consider for heavy concurrent load, which is composed mainly of Writes (mainly Reads SHOULD be handled in the web tier). It CAN frequently be used to support extensive use of protocols, such as CIFS, which allow bulk content load directly to the Alfresco repository or to support heavy batch import processes (in general, any process involving large indexing jobs).
- **More CPU to the web tier**
  Increasing CPU capacity on the web tier SHOULD be considered as the primary response to high (Read or write) concurrent front-end load and CAN be used with caching. Increasing both the number of processing units and running a more powerful processor are viable options, but the former SHOULD provide more native parallelism, where custom front-end mashups (fat web tier) require the latter option, to minimize context switches.
- **More memory to the web tier**
  This practice SHOULD be considered when managing concurrent, long lasting front-end sessions that could potentially overload the web tier. In addition, this CAN be considered when the web tier is processing bulk content uploads (500+ MB).

- **More storage to the content platform tier**
  You MUST use this action to cope with index growth, which is typically directly proportional to the content growth. You must have double the index size free at all times.
- **More storage to the storage tier**
  You MUST take this action to cope with content growth at the infrastructure level. It typically requires service interruption. Alfresco Enterprise 3.2 and above supports dynamic content growth and advanced storage policies management. See *Content Store Selector* for more information.
- **More CPU/memory to the DBMS in the storage tier**
  You SHOULD consider this option (together with proper tuning of the database) to improve transactional capabilities of the content platform with heavy concurrent load. For clarity, this is not normally a bottleneck in a non-clustered architecture where transactional capabilities of RDMBS exceed content platform capabilities in a 1-1 relationship.
- **Improved network speed or latency or simplified network topology**
  A full discussion on network optimization is outside the scope of this document, but a reliable and predictable network is a prerequisite for a large repository deployment. You SHOULD always consider this for improvement, particularly in cases of extensive web tier use (to improve REST communication), and when storage and RDMBS physical backing storage are accessed via a network.

When to scale up

Scaling up is not the definitive answer to growth in requirements or to an originally undersized production system, but MAY, nevertheless, be used as a primary response to achieve a specific performance requirement, especially in virtualized environment where scaling does not involve new infrastructure deployment.

Here are some tuning practices to consider before deciding to scale up:

- Tune the existing system first - your installation might require simple optimization for example, JVM parameters, tuning of Alfresco specific parameters, or RDMBS configuration optimization.
- Caching MAY support performance improvements.
- Scaling out MUST be used for HA requirements and SHOULD be preferred for high concurrent load scenarios.

Scaling out the Alfresco content platform

To scale out an Alfresco platform means taking measures that raise the number of physical or virtual processing nodes. Only some of these measures are related directly to Alfresco, (particularly in the web and content platform tiers), while others involve scaling out lower level infrastructure (particularly in the storage tier).

Deciding when and how to scale out is one of the key tasks of an Alfresco Administrator. This may be mandated by design requirements (such as full HA or disaster recovery), or by expected load (high number of concurrent users). Sometimes scaling out must be taken as a measure to respond to increased/unexpected load on any of the Alfresco content platform components and support infrastructure.

Alfresco offers two large integration and scalability elements:

- Any component of the application layer delivered within the Alfresco content platform offers dynamic and self-contained clustering:
  - Dynamic in that you can configure either Alfresco Share or the repository to run and split the load among different physical/virtual nodes.

  - Self-contained as Alfresco clustering does not require any application server level clustering, such as session sharing, working by default with HW/SW load balancers, but still supports it when mandated by additional requirements.
- Alfresco has always applied the integrate over impose enterprise deployment approach, so while not mandating any specific infrastructural component for Enterprise deployments, it strongly integrates and delegates as much as possible to native underlying HA enterprise capabilities.
- Typical examples of key HA functionality that Alfresco delegates at lower layers comprise:
  - Database clustering and replication
  - File system (SAN/NAS) backup/restore and replication
  - Serviceability and performance for the Open Office Transformation suite
  - Application server session sharing and replication

The appropriate strategy to scale out your Alfresco content platform is to correctly orchestrate the growth of the different (internal and external) components to avoid bottlenecks, and to support a reliable level of service for each of them.

The next sections refer to official resources and best practices for Alfresco content platform clustering, as well as details for scaling out and tuning the support infrastructure.

Scaling out the application layer

Typically, scaling out the application layer is achieved by increasing the number of nodes powering application servers in the content platform or web tiers.

Considering that Alfresco Share (or any Surf/custom front-end) is typically a quasi-stateless front-end using the RESTful API, the content platform tier is the first component that might require scaling out. It requires the greatest attention as it holds the entire content access layer.

There are different solution requirements for choosing one or another measure. However, those requirements are generally classified in three main horizontal classes:

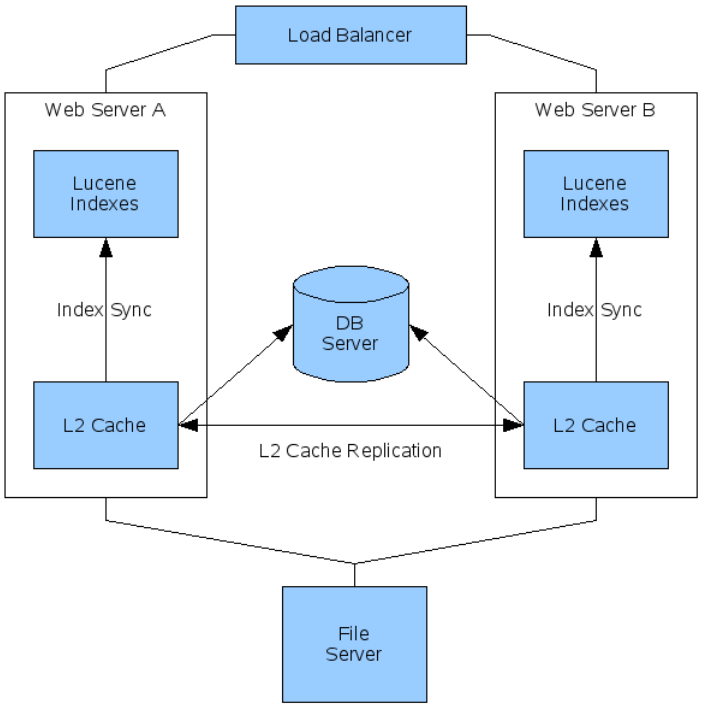
- **High availability of service**
  Requirements above 99.9% of service availability (with no maintenance windows) typically require scaling out to avoid a single point of failure. This usually involves creating multiple up-to-date active or passive sites (disaster recovery) that ensure service. This can involve clustering for either or both content platform and web tiers.
- **High load balancing/splitting**
  When the Alfresco content platform experiences excessive processing load, it may be an option to balance it through multiple instances. Profiling user group usage may allow you to optimize performance and improve user experience by splitting load using more heuristic policies (such as Read/Write split or batch/interactive split). This can include clustering for either or both content platform and web tiers.
- **Strict information serviceability preservation/replication**
  In specific cases, (government, mission critical) information must have its integrity preserved (at the content level in the platform, such as database, content store, and indexes in sync). As a result, replication and preservation policies cannot be delegated to the lower infrastructure layer as it takes time to restore integrity at the content level. This usually involves clustering at only the content platform tier.

Given these types of requirements, the following section discusses content platform tier clustering followed by web tier and Alfresco Share clustering, as well as the classes of requirements typically fulfilled for each case.

Simple repository clustering

The simple repository-clustering configuration is used in 95% of Alfresco clustered installations, offering high availability features while keeping architectural complexity and duplication of information low.

The following diagram illustrates the simple repository clustering configuration:



Simple repository clustering comprises:

- Two+ active nodes allocated for the content platform tier application server
- One shared database between the two nodes
- One shared file system between the two nodes (typically a SAN/NAS via NFS)
- Local indexes kept for each node
- A front load balancer using sticky sessions facing either end users directly accessing the Alfresco repository or web tier applications. This can be implemented using HW load balancers or SW load balancing (such as using Apache or ISS together with mod_jk). This covers only HTTP protocols.

The only Alfresco configuration required for this clustering configuration to work with standard Alfresco deployments is the L2 Cache replication that, for each node's transaction, will notify other cluster nodes of a remote user transaction and trigger invalidation and regeneration of impacted cache entries.

You SHOULD consider this configuration when trying to fulfill classes of requirements one and two in *Scaling out the application layer*. This configuration offers:

- **High availability**
  Two+ active nodes can be configured for reliability and to allow complex maintenance policies without service disruption (for example, backup/restore and application updates). In addition, you can configure passive nodes to support disaster recovery, that keep their indexes constantly up to date by participating to the cluster, but are actually only pointed to on demand by the load balancer in case of service disruption of active nodes.

- **High load balancing/splitting**
  An overall higher throughput of the whole Alfresco content platform tier can be achieved by having two or more active nodes using a simple round robin process with sticky sessions. The throughput growth is actually sub linear. For every node added, more network traffic is generated and additional in-transaction L2 cache synchronization. From Alfresco 3.2 onwards, you can also achieve load splitting using Alfresco clustered job locking features, which allows high load jobs to be performed once, only between all cluster nodes.

This configuration SHALL NOT be used to fulfill requirement three from *Scaling out the application layer* as no content level replication is provided at the application level. To achieve/mimic content level replication in this configuration, you SHOULD delegate typical solutions to the lower layer. A possible approach to obtain warm content replication CAN be:

- Use snapshot-enabled storage or Rsync/other FS sync for content store
- Use database clustering and replication (such as Oracle RAC)
- Use custom scripts to sync indexes (from backup copy)
- However, you still need to start up (and possibly run a small re-index) before the replicated instance is up and running

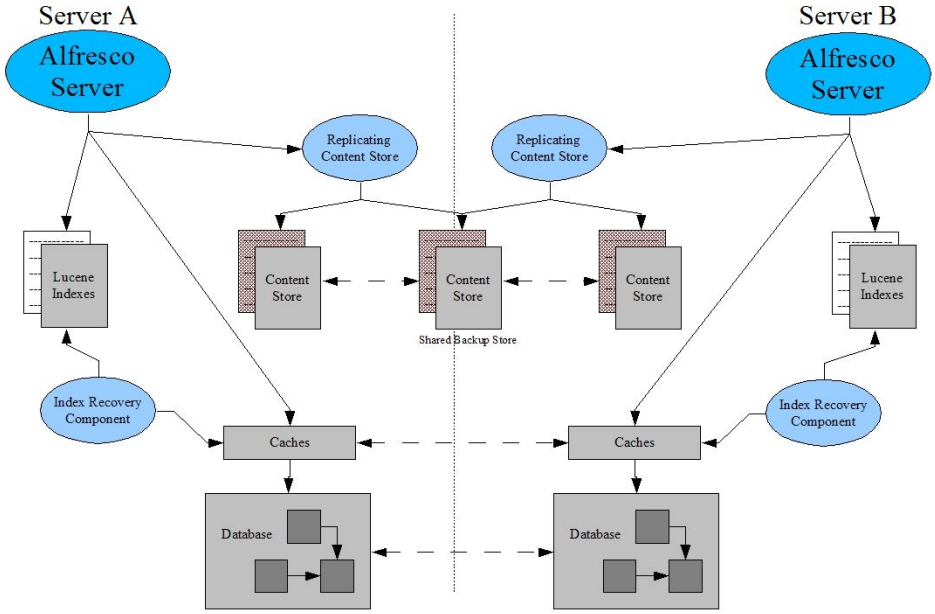
This configuration is completely dynamic. For example:

- New nodes can be hot deployed into this architecture by using Alfresco cluster nodes (using JGroups for Alfresco 3.1 onwards), auto-configuration, and discovery capabilities. When properly configured, you can add new nodes and after an initial re-indexing time directly proportional to the content size, they will be ready to serve as a new active node in the cluster. To speed up the re-index process, you can restore the backup-lucene-indexes folder from another running instance to the lucene-indexes folder of the newly added node. Here, the re-indexing process is limited to the content created since the last index backup (default configuration provides daily backup at 3AM).
- The level of service is almost completely configurable in the load balancer (HW or SW), which can be configured to provide a pure active-active load balanced architecture (or active-active-passive in the case of three nodes with a disaster recovery site).
- A JMX configuration interface is available (from Alfresco 3.2) as an Enterprise feature for the Subsystems API. This enables dynamic configuration of behavior and core parameters of the Alfresco platform. The JMX interface is cluster-aware, so that changes in one cluster node propagate to all cluster nodes seamlessly.

Local and remote content store

The local and remote content store is a more advanced and complex clustering deployment model that takes advantage of the high configurability of the Alfresco content store to provide application level replication of binary content. It also comprises out-scaling the storage layer. For the specific configuration, see *Scaling out content storage*.

The following diagram shows an overview of one possible replicated content store configuration (local and remote) that extends the simple repository clustering:



- Each cluster node writes content in a local content store and replicates it to a remote content store. In this configuration, this is shared between the two servers.
- Relying on underlying RBDMS co-location and replication features to completely duplicate the infrastructure, removing any single point of failure. Replicated content store will only replicate binaries and no metadata.
- All other configurations (for example, load balancer, local indexes, and L2 cache replication) stay the same as for simple repository clustering.

Configuring a replicating content store that can synchronously replicate content to multiple stores in a SAN or LAN enables content binary replication. However, this is not designed for geographically distributed sites.

For out-scaling requirements, you SHOULD consider this clustering configuration as improved support for Content Level Preservation as content binaries are replicated to separate storage automatically (optionally, in transaction). You SHOULD also consider content replication for automatic binaries backup to lower speed/cost storage, such as tape or Amazon S3. To enable higher levels of availability and disaster recovery features, you SHOULD consider combining this with database scale-out and replication features.

This content replication approach SHOULD only be considered as a solution for very specific requirements because of the following limitations and constraints:

- Replicating the content store was not designed for content level geographical distribution and must be corroborated by lower layer features. Alfresco provides repository level content transfer solutions as of the 3.3 releases (see Transfer Service API), and Alfresco 3.4 is expected to offer Master Slave replication services which can work on geographically distributed sites.

- This configuration requires a more expensive infrastructure setup than a simple repository, particularly for large repositories, as configuration for two active nodes will require:
  - Three times – shared storage space (two local + one shared)
  - Two times – indexes spaces (two local)
  - Two times – database for DB replication

In addition, certain cases will have added costs for database replication licenses.

- Additional load is delegated to the application layer. For synchronous replication, this might turn into lower perceived interactive performances for users and in a heavily sub linear throughput growth when adding cluster nodes.
- Typically in an Enterprise context, snapshot and backup/revert enabled storage systems are given to projects as commodities, so their responsibilities SHOULD be exploited and maximized offloading application layers from them.
- There is a higher complexity of infrastructure to maintain.

In summary, you SHOULD use simple repository clustering to local and remote content store for easier maintainability and performances. To address more advanced content distribution policies (ILM), Alfresco 3.2 introduces the Content Store Selector, an Enterprise-only feature.

Web tier clustering

A separate quasi-stateless web tier enables scaling of this tier independently from the content platform, allowing for different scalability configurations. This can be useful if corroborated by proper caching and balancing policies to:

- Distribute high load across multiple (lower cost) web tier node instances
- Allow arbitrarily complex user request routing to the repository and, if properly designed, to hit a cache layer instead of the actual repository

Characteristics to consider when designing out scaling of Alfresco Share (or any Surf web tier application) are:

- The Alfresco web tier does not include any kind of client session replication, so this must be delegated to the web tier application server. The default option is to use a front facing load balancer that will use sticky sessions to maintain client sessions attached to the same Alfresco Share instance.
- All UI custom interactions and mash-up integrations (external or web services) can be concentrated and out scaled in the web tier independently from the content platform, provided there is no interaction with the repository.

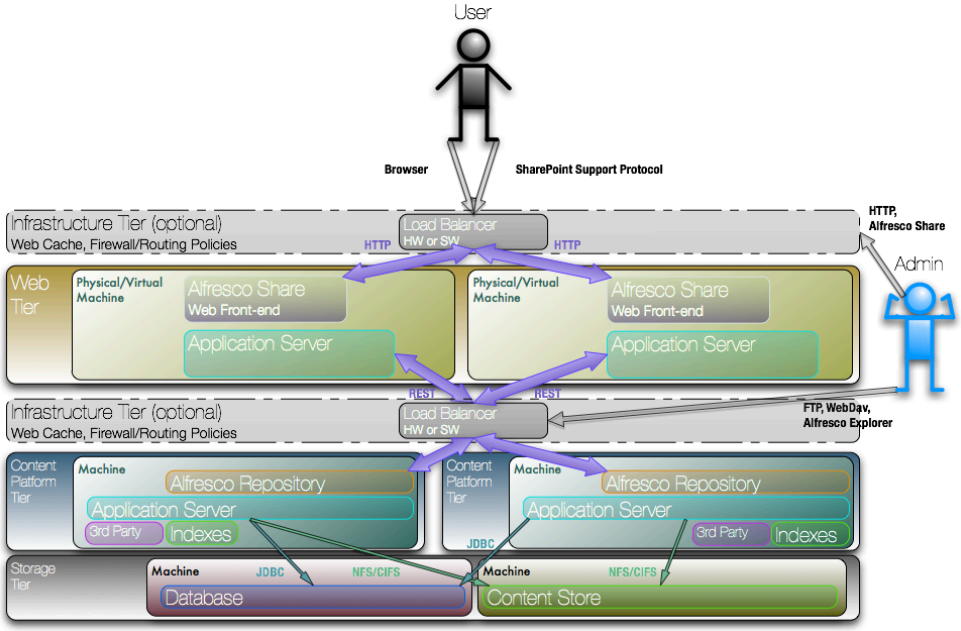
Given these considerations, and using the simple repository clustering for the content as reference, there are at least two main architectural patterns you can use while deploying the web tier in clustering:

- Transparent repository web tier scale-out model
- N-swimlanes deployment model

Transparent repository web tier scale-out model

This involves (in addition to the front facing load balancer/client session replication) a sticky session-based load-balancing layer between the web and the content platform tiers. This layer masks the Alfresco repository instance accessed by a specific user session, allowing complex load distribution/splitting to be performed at multiple levels.

This way, web and content platform tiers can grow independently from each other, allowing a different number of Alfresco Share and repository instances. Admin user access (directly to the Alfresco repository layer) can also be balanced and allowed via this intermediate load-balancing layer or drill down to single instance accesses where needed, as shown in the following diagram.



An additional benefit is that the configuration and lifecycle of web tier applications is simplified because they all refer to the same logical content platform instance. The main drawback to this approach is having balancing layers as a single point of failure for the architecture. However, this is often an acceptable condition since it is resolvable at the infrastructure level.

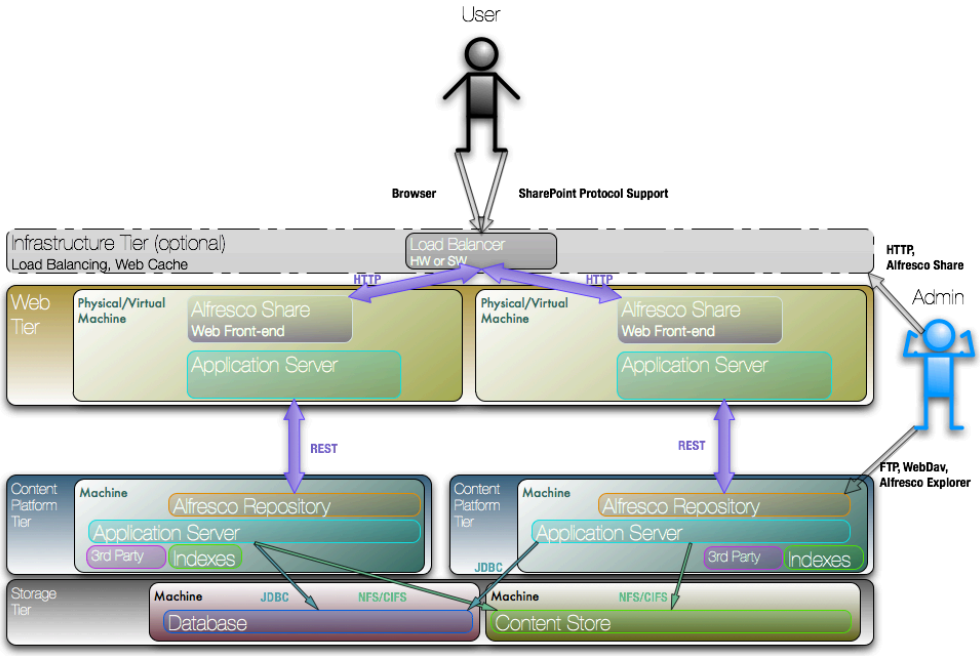
N-swimlanes deployment model

This model involves configuring every web tier instance to directly point to an instance of the repository in the content platform tier. In this deployment configuration, there is no (or optional) intermediate balancing layer – no single point of failure – while Alfresco Share instances MUST have physical knowledge of Alfresco repository instances location creating virtually independent processing pipelines composed by couples (Share instance, repository instance).

This deployment model ensures that a specific node of the content platform tier serves fewer or the same amount of user requests as the connected web tier instance, so that the load distribution policy SHOULD only be controlled by the front facing load balancer. In this deployment model, admin access to the content platform tier is offered to specific instances of the Alfresco repository. Despite removing the load balancer single point of failure, the result is a much less scalable deployment model than the transparent one due to:

- **Maintainability** as every Alfresco Share instance must have separate configuration

- **Reliability** as the disruption of one single instance (of either Alfresco Share or repository) results in the disruption of a complete pipeline due to the inter-tier heavy coupling (as opposed to a transparent out-scaling model)



The remainder of this document uses the transparent content platform out-scaling model for Alfresco components using the classic two-node simple repository clustering.

Scaling out the support infrastructure

Support infrastructure is fundamental to a proper out scaling of the platform. The following practices SHOULD be considered to manage large repositories:

Scaling-out the database

In HA or a mission critical context database, it MAY be required to use co-located RDMBS solutions such as replication and failover or clustering using database specific technologies, open source or proprietary, such as:

- Oracle RAC or Oracle Failsafe (with the latter being a much simpler solution based on Windows clustering)
- MySQL Cluster
- Postures HA

Note: The basic requirement for a distributed DMBS deployment with Alfresco is to have READ-COMMITTED isolation level with foreign key constraint enforcement.

As a general out scaling best practice, you SHOULD share a logically replicated or clustered database instead of using an active-active configuration between different database used by different Alfresco repository cluster nodes, as these configurations MAY introduce useless latency.

For MySQL and high load and/or highly concurrent requirements, a solution for having one logical instance splitting load to multiple processing database nodes (raising maximum throughput) is to point all Alfresco repository instances to a MySQL Proxy instance, providing any possible load distribution or splitting policy to several MySQL worker servers.

Scaling out content storage

Although a detailed discussion of scalable storage media is out of scope for this document, there are common use cases and best practices gathered from the currently installed Alfresco user base. The following are some approaches taken from open source and proprietary solutions:

- **Amazon S3**
  Being Alfresco cloud-ready, one cost effective option to host large repositories could be using the Amazon Web Services suite to deploy Alfresco in the cloud and exploit the Amazon S3 (Simple Storage Service). Another option is to deploy on premises using S3 with a special content store implementation.
- **Content Store Selector**
  Alfresco Enterprise 3.2 enables you to implement complex, dynamic Information Lifecycle Management (ILM) policies to cope with large repository content growth and cost effective storage management, effectively providing a software alternative to HW storage management. See *Content Store Selector* for more information.

Scale-out Open Office

Where the transformation and text extraction engine is under high load (such as with collaboration), you SHOULD consider scaling out the Open Office computational nodes to improve stability. You MUST consider this when transformations are mission critical.

As of Alfresco 3.2, a new applicative reliability and availability layer based on JODConverter is modeled as a subsystem. You can configure this to pool different instances of Open Office and automatically handle Open Office instances automated restart. You SHOULD always consider this to scale up/out transformation capabilities. In addition, you SHOULD consider running Open Office on a separate physical/virtual machine to separate transformation engine load from Alfresco platform operations.

When to scale out

This section provides some general guidelines about clustering use. You MUST NOT over use scaling out as it can make the overall system more unstable and complex to debug.

Use proper mix of scaling up (with multiple processing units per node) and scaling out (2/3 nodes) to fulfill the vast majority of high concurrent load and large repository requirements. Since much of Alfresco scalability is due to its linear design and modern technologies, Alfresco is inherently more scalable than older platforms.

Consider the following possible aspects when scaling out:

- You SHOULD avoid adding too many cluster nodes to an Alfresco repository, as it will increase the time spent in specific transaction indexes propagation and present higher network traffic. Most Alfresco Enterprise installations have two cluster nodes (+ one passive); scaling up CPU/memory is how to manage further growth. For a large repository target, there is no Alfresco theoretical limit for cluster nodes, but common sense is to deploy up to four Alfresco instances in a cluster.
- Alfresco clustering is not designed for geographical distribution so cluster instances must be co-located.
- You SHOULD use scaling up when:
  - There are no HA strict requirements
  - There are long and CPU/memory intensive operations to perform (out scaling will not help)

The following section completes the architectural analysis on how to scale large alfresco repositories balancing and caching, which can turn out to be crucial for performance and sustainability of the platform.

Balancing and caching

You SHOULD use load balancing and caching to provide an additional level of scalability to high load installations. They MAY be used to improve performance and throughput and typically are arbitrarily specific to the usage profile or your Alfresco platform with policies that improve over time. You MUST NOT use these two architectural tools to cover other design or performances issues.

In general, it is best practice to first try to tune or scale up your current infrastructure. You SHOULD carry out good application profiling both from a functional (load peaks and frequent operations) and technical point of view (resource consumption and service delays) before implementing these strategies. A better understanding of the application corresponds to a better balancing or caching strategy.

Load balancing

Load balancing can occur at many levels in your architecture, and implementing context profiled load distribution policies might be vital for user experience and overall load of the systems. You CAN implement the load balancing strategies that follow with common features from both HW and SW (such as Apache Httpd) load balancers.

A detailed discussion about load balancing strategies is out of the scope for this document, but at any level (front facing or between content platform and web tiers), some common configurations are:

- **Round Robin (w/fallback)**
  Load balancer distributes the load equally to all underlying (Alfresco Share or repository) nodes using sticky sessions to ensure user session continuity. Load on each of the N nodes will be 1/N of the total load. You SHOULD configure a fallback node in HA contexts, which is invoked if an active node is down or not responding.
- **Read/Write split**
  With a proper profiling of application functionality and technical bottlenecks; it is possible to split Read-only user requests from actual repository intensive operations. You CAN use this at the Alfresco Share or repository level to ensure high retrieval performance (typically to a specific class of users) while other nodes are busy processing CPU and memory intensive content elaborations. This MAY be used to ensure the best QoS in an enterprise-wide content platform shared for both collaboration and content delivery processes.
- **Scheduled node exclusion**
  Another typical load balancing policy is to have each node excluded from the execution pool in turn, optionally taking care not to kill user sessions. This is typically used for operational maintenance of the production application (unless using some application server webapp cluster distribution features) to distribute a new release over the cluster.

Caching

You SHOULD consider caching when approaching the design and deployment of heavy load content-centric websites. Caching can happen at many levels, inside or outside the alfresco content platform products.

The typical caches that can come in handy to support out scaling of your Alfresco platform are:

- **L2 Cache (Ehcache)**
  Ehcache is the L2 cache representing Hibernate objects and is distributed over the cluster. You can tweak the default configuration heavily depending on the specific usage/solution for which Alfresco is being used. Configuration best practices for Alfresco Ehcache internals are described in Ehcache tuning, even if this is not normally required.
- **Web scripts cache**
  Web scripts on Alfresco Share and repository instances can be configured to cache responses and implement complex and tailor-made caching policies. Web scripts comply with HTTP cache directives.
- **External web cache**
  Web caching involves adding web server caches (such as Apache Httpd, IIS), external proprietary, or open source (Squid) proxies to support massive front-end loads and offload the web tier/content platform instance from most of the read load. Use this cache carefully as the result might make debugging and real problems more challenging. However, you SHOULD consider it an almost infinitely scalable solution for high load front-facing web tiers, particularly if backed by non-volatile repositories. You can use this with the web script cache to provide a high level of caching. Corroborate this approach with a proper profiling of your applications to allow a more specific and effective caching profile and not hamper proper data refresh rates.

Application server clustering

Alfresco does not require any application server level clustering by default. Nevertheless, in certain contexts, supporting Alfresco native clustering capabilities with application server level clustering features MAY drive improved overall HA.

While a full coverage of application servers clustering techniques is out of scope for this document, the following summarizes typical application server clustering use cases:

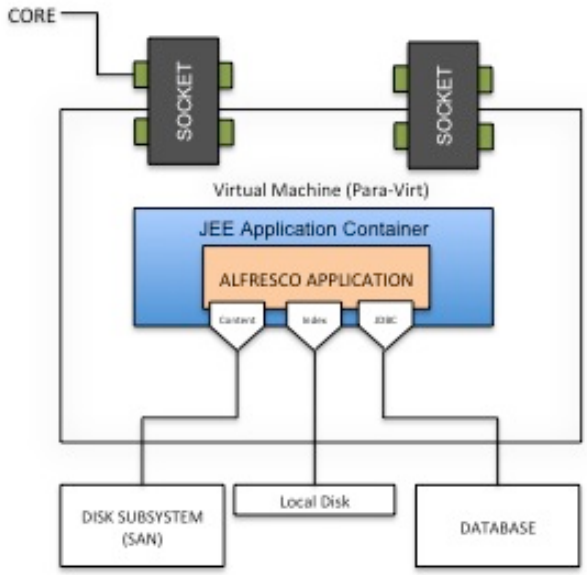
- **Active-Passive** configurations can increase system availability providing failover (and in certain application servers preserve the user session)
- **Active-Active** configurations (for example, JBoss) MAY also be considered for session replication to provide service continuity in case of failure of one instance. As this is a rarely needed configuration, check with Alfresco Support before you put this kind of configuration in production.

Impact of virtualization

This section discusses building an Alfresco production infrastructure in virtual environments as there are a number of specific considerations.

Virtualized deployment architecture

The following figure illustrates how you can define a traditional Alfresco setup using a virtual machine:



In this scenario, there are two sockets on the bare metal machine: each socket has four cores, allowing the virtual machine to virtualize up to eight hardware equivalent compute units. When configuring your virtual machine, you can choose how many virtual CPUs (hardware equivalent compute units) you want it to use.

Virtualized deployments recommendations

Some recommendations for virtualized deployments are:

Bare metal machine

- Max out the configuration for disk speed, bus speed, CPUs, and memory
- Choose CPUs that are x86_64

Host operating system

- Run a 64-bit operating system
- Use a para-virtualized kernel
- Ensure you are using the 64-bit version of your virtualization software

Virtual operating system

- If possible, run a para-virtualized environment over a fully virtualized operating system
- Run a 64-bit operating system
- Ensure the JDK is 64-bit (retrieving via yum or apt-get should work if using a 64-bit Linux operating system)
- Ensure the indexes are local to the virtual disk and not over the wire on the SAN/NAS device. The default measure for index space is about 10-20% of content size, but could be as large as 75% of the total content stored in Alfresco.
- Configure multiple NICs (one inbound/two outbound). Route the traffic accordingly between the NAS/SAN and database.

Alfresco optimizations

- JVM tuning parameters

```
-Xcomp -Xbatch -Xss1M –Xms3G -Xmx3G
-XX:+UseConcMarkSweepGC
-XX:+CMSIncrementalMode -XX:NewSize=1G
-XX:MaxPermSize=128M
-X:CMSInitiatingOccupancyFraction=80
```

- Change contentModel.xml to set the atomic=false parameter. This is currently set to true for cm:content and cm:folder.
- Do NOT use virtualization for Alfresco WCM or Alfresco ASRs.

Storage considerations

When executing the Alfresco content platform tier in the VMWare virtual machine, you MUST NOT use logical VM disks for indexes. Use a mounted (via CIFS or NFS) physical disk for the indexes instead. On Sun Solaris, Solaris Zones are used without issues.

Timing sync

While deploying Alfresco content platform clusters in a virtualized environment, you MUST ensure proper clock synchronization between all the involved nodes in a cluster. You MUST put this in place before you configure any Alfresco L2 Cache synchronization because transactions are propagated and merged in remote caches based on their timestamp. This is why improper timing sync results in unpredictable cache distribution.

Note: Improperly configured machine clocks result in indexing issues for single node deployments as well because the very same time-based sorting is used for tracking transactions and building the Lucene indexes accordingly.

You CAN achieve proper timing configuration by configuring your VM instances to start an NTP client as a service so each instance can be synchronized with an external time reference. In the case of physical host machine overload, VM clocks can start delaying and fall into the same out-of-sync issue. To cope with overall load requirements, properly dimension your physical host machine.

I/O

I/O might become a bottleneck while using virtualized environments that share the same physical machine and thus I/O access and resources. Alfresco can be quite I/O intensive, particularly in contexts such as collaboration, transformation, and a high percentage of full text indexing. Therefore, make sure that you dimension your VM configuration and host machines to support a high ratio of physical storage access. One typical sign of overloaded OS I/O is reaching the OS open file handles maximum number (1024 in Unix) of the host machine.

JGroups unicast TCP

VMs are known to have issues with multicast. For this reason, you SHOULD use unicast and TCP when configuring clustering via JGroups or standard Ehcache (this is also true for cloud environments).

Alfresco design best practices

This section discusses the second level of guidelines to consider when deploying Alfresco for large repositories. This second level is actually the proper design and configuration of the ECM platform regarding all the decisions to take during the design phase and while configuring the Alfresco content platform during the development process and in production.

While deploying the proper architecture for your requirements remains a prerequisite, proper configuration of the ECM platform and design of the custom business logic processes to deploy on it, based on platform experience and ECM wisdom, might drastically impact the performances and overall maintainability of the production Alfresco content platform.

The following sections analyze how to choose the proper Alfresco functionality and make the best use of it, referring to *Scaling Alfresco solutions* for real use cases where you can apply these best practices.

Taxonomy design best practices

One of the first steps to take when translating functional requirements into technical design for an ECM system is to organize content in taxonomies. In this sense, it is important to know that Alfresco allows multi-dimensional content taxonomies providing primitives for content organization, such as:

Spaces

The classic folder-based content organization is a tree, implementing a 1-N relationship model with two-way parent-child visibility. This is designed in Alfresco as a child-association.

- You MUST use Spaces for taxonomy dimensions that involve specific access rights and/or explicit business logic (via rules/actions) for a specific content categorization, as spaces are the smart folders that hold the concepts of ACLs and content rules.
- A typical example is to use spaces to map an organizational structure that involves different departments with different access rights. Multi-categorization is supported via shortcuts in the repository.

Categories

Alfresco categories are a more loosely coupled type of categorization, which allows the implementation of an N-N relationship model with a direct child-parent and derived parent-child visibilities. These are designed in Alfresco as NodeRef multi-valued properties.

A dedicated Categories browser allows semantic and native multi-category (a document can belong to one or more categories) support for document taxonomies. Categories do not hold any business logic rules and/or access rights and can be used at any level in search operations to retrieve specific content.

Tags (Alfresco Share only)

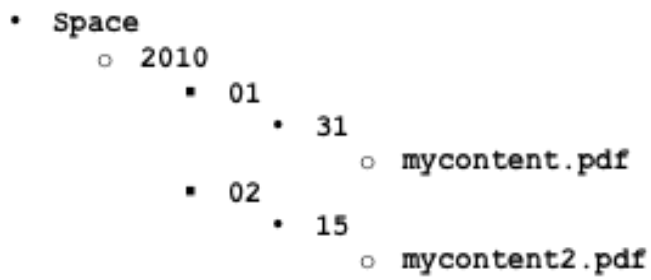
Tags are the Alfresco Share equivalent for Alfresco categories and make use of Alfresco categories under the hood. The main difference from Alfresco categories is that tags are user-created while categories are admin-created, top-down organizations. Their purpose is clear in a collaboration environment where a number of labels are provided for each content item. Alfresco Share then allows search and easy retrieval of specific tagged content.

Custom metadata and constraints

Defining a custom mandatory metadata field in the content model and introducing a multi-valued constraint for this field establishes another categorization dimension in your content base. By configuring the search forms, you can then use this metadata field to explore specific partitions of your repository. A typical example is a custom “Privacy” field representing the privacy level of a document, where values are specific to each company and normally used to identify public or confidential documents in searches.

Given these basics, here are some guidelines you SHOULD use when deciding how to design your taxonomy:

- Maximize usage of the out-of-the-box Alfresco primitives, using them in combination with each other to offer a maintainable and low load content taxonomy. This mixes and matches the use of spaces and categories/tags to possibly avoid the following:
  - Duplication of content space hierarchies MAY make sense in enterprise-wide department-oriented deployments, but CAN indicate improper usage of categorization primitives and abuse of space usage.
  - Frequent admin (IT) interaction MAY make sense in a highly volatile context, but you SHOULD probably have preferred tags to categories to allow users to auto-organize their content.
  - Proliferation of logical custom groups/rules - a complex environment MAY require this, but too many ad-hoc created user groups and rules/actions (to enforce custom permissions on documents) may indicate that spaces should be used.
- Use a content tree that is balanced as much as possible. Specifically, do not store more than 1000/2000 nodes (spaces/contents) under one single Alfresco space. This principle applies to most content storage systems, but particularly for Alfresco. Designing an inherently scalable content taxonomy prevents Alfresco from degrading in retrieval performance when content grows. This best practice applies to all Alfresco installations but MUST be considered in high injection and large repositories contexts.
- Organize the space based on the content creation date to overcome the creation of content clusters under a specific space. This approach, which is used by the Alfresco content store on the file system, results in a space hierarchy that under a specific node of the tree might look like the following:



- Do not try to map completely independent and parallel organizations into top-level spaces. You SHOULD consider multi-tenancy when there is no access required between the different organizations because of the added indexing and content isolation benefits.
- Do not use spaces to model lifecycle and document related information. One common mistake in using Alfresco taxonomies is to overuse the spaces to express almost any kind of document categorization and workflow information (for example, document status as a separate folder). Unless you have very basic workflow requirements, you SHOULD gear your processes using the

several other Alfresco features for content lifecycle modeling, such as Advanced Workflow with custom metadata/aspects, and the powerful Alfresco search engine features.

This section showed how to best map your process onto Alfresco categorization primitives. The following section explores the complementary side of the coin—optimization and best practices for modeling your content in Alfresco.

Content model design best practices

Content modeling is a core ECM practice, so this section assumes good ECM knowledge. Model design also requires experience with the specific ECM platform-modeling primitives. This is where Alfresco allows a lot of flexibility into modeling your domain by providing the following:

Inheritable content and space types

Alfresco allows your custom type to inherit from cm:content or cm:folder. To maximize reuse and factor out common configurations/behaviors/properties, you SHOULD leverage ever-longer custom chains of model inheritance for very structured content models.

Alfresco aspects

Alfresco is based on AOP and offers Aspect Oriented Design capabilities for your model. Aspects are powerful tools for the information architect to design reusable sets of information and behavior, which can then be applied to specific types (via mandatory-aspects), or to specific content instances (via the UI). You SHOULD leverage aspects to model crosscutting concerns that cannot be easily modeled or reused by inheritance, and to provide an important primitive to reduce the number of properties of types in the repository (and the related indexing effort). In addition, aspects CAN be used to mark content lifecycle state, reducing the overall complexity of the spaces taxonomy.

Using out-of-the-box aspects, you CAN optimize Alfresco as follows:

- cm:versionable
  This aspect marks content as versionable so that every time content is updated a new version is created. This can be applied via the UI or content model. The suggestion here is to properly identify content that requires versioning, so applying this aspect to the complete cm:content base SHOULD be avoided unless there are strict requirements for it. It MAY result in unnecessary versions being created, such as for content renditions or for Alfresco Share user-generated content like comments or forum posts. In addition, advanced configuration is possible as of Alfresco 3.2 to prevent property-only versioning, and this might result in a more optimized version management.
- Where the numbers of versions increase to an unnecessary high value, contact Alfresco Support for a script to clean old versions. Where numbers of versions increase to an unnecessarily high value, create a script to clean out old versions, based on your requirements. You may want to contact Alfresco Support or Alfresco Professional Services for advice.

Indexing options

Indexing is one of the most stressed Alfresco capabilities in large repository deployments. Therefore, it is important to tune your indexing requirements from the design phase, trying to balance performance and maintainability, and actually tuning how, when, and which content and properties must be stored in the index.

One important analysis that CAN be turned into Alfresco configuration is which properties must be indexed per content type and whether they have to be indexed in transaction or not. This configuration SHOULD be tuned to avoid uncontrolled indexes growth and performance degradation due to overload of synchronous indexing operation. Where full text binary indexing is not needed for

certain content types or instances, Alfresco Solution Engineering provides a custom aspect called DoNotIndex to apply for disabling cm:content indexing. This CAN be used to perform additional control on binary indexing. Additional tuning can be done for the Lucene search engine. See *Lucene indexing tuning* for more low level tuning best practices.

Quotas and usages

For Enterprise-wide deployments and collaboration environments (decentralized content management), you SHOULD consider limiting the content a specific user/space can have in the repository by using Alfresco quotas. Although this MAY introduce some additional processing delay (due to quota evaluation per every new operation), you SHOULD consider it as the main option to limit uncontrolled repository growth in decentralized environments.

Content Store Selector

The Alfresco Content Store Selector feature can be used at design time to support information flowing through different content storage media in its lifecycle. This means designing a system that uses the cm:storeSelector aspect to implement arbitrarily complex ILM policies. It MAY be a solution to manage large repository content loads and large migrations from legacy systems, especially allowing the storage of content deemed not as critical into low cost storages, while using high cost/performance storage for “live” content.

Multi-store repository

Alfresco manages its ADM repository storage in multiple logical stores that are identified by a specific access protocol (workspace or user) and a storeId (SpacesStore or Version2Store). These logical stores are in a default Alfresco installation as follows:

- **workspace://SpacesStore** - the default working store and logically represents the entire space structure and content
- **archive://SpacesStore** - holds deleted content; basically, Trash Can content
- **workspace://Version2Store** - contains versions for objects with the cm:versionable aspect
- **user://alfrescoUserStore** - contains authorities (user and groups) related information
- **system://system** - contains system information
- **avm://sitestore** - provides AVM versioning time machine-like functionality

Alfresco stores MUST NOT be confused with the physical content storage locations mentioned in Quotas and usages. While the former are logical wrappers of access to low-level information, the latter are only different physical storage locations and will be mapped onto the Alfresco stores. The workspace://SpacesStore is usually the most used store, and liable to complexity growth when deploying large repositories. In addition, the accompanying complexity of computing and managing a large index for the store may result in degraded performances and Read/search throughput.

The idea behind partitioning your content into separate logical stores for very large repositories (50M + documents) is to simplify index and content store management. A multi-store repository strategy that uses RMI exported Foundation APIs SHOULD be considered when managing content using Alfresco as a headless backing repository, accessed by custom applications using either the Foundation API or the RESTful API (both supporting the notion of a store), and not leveraging any of the Alfresco user interfaces.

Managing user-created and configured stores is not natively supported by any Alfresco user interfaces and requires custom code. The impact of custom approaches is discussed in *Custom multi-content store impacts*. However, there is one native way to partition your repository in smaller and more manageable chunks using an out-of-the-box Alfresco feature called multi-tenancy.

Multi-Tenancy

The Multi-Tenancy (MT) feature allows you to host multiple logical repositories on the same physical Alfresco instance. MT enables multiple tenants that are independent content repositories. They inherit the same content model but store content separately. In this sense, each tenant gets its own copies of the six Alfresco stores, technically partitioning the full repository in virtually separate content storages.

Pros and cons

There are a number of advantages and disadvantages to consider when approaching MT because it presents some limitations:

Pros

- Content isolation (each tenant has a separate backup/restore procedure)
- Effective reduction of the content (and index) store per tenant
- Alfresco Share (3.2 and above) supports MT, so Alfresco Share sites might be deployed over different tenants to scale
- MT works in clustering

Cons

- CIFS is not supported
- Each tenant has its user realm (for example, users log in as username@tenant) and one tenant user cannot access another tenant
- Only “alfresco” authentication is supported, so no easy integration with external or SSO methods is provided
- You SHOULD consider MT for hosted (SaaS) deployments of Alfresco where multiple separate organizations and user realms can coexist on the same machine. In addition, it CAN be considered a way to deploy large repositories if the following requirements relaxations are accepted:
  - No out-of-the-box integration with Enterprise authentication system
  - No need for inter-tenant access

Custom multi-content store impacts

If MT is not an option because of specific requirements, you CAN consider a completely custom solution. However, you SHOULD be aware that there are risks, which might lead to invalidating support or data inconsistency. For this, contact Alfresco Support before implementing any multi-content store strategy.

Consider this approach when using Alfresco as a headless remote backing repository by using multiple stores that can be queried and treated completely independently by custom code, using any of the customization APIs (both Foundation and RESTful API are multi-store aware).

However, consider the following:

- Custom code must be written to create and manage this custom store
- Alfresco search does not support multi-store searches at the API level, so custom code might be written to support aggregation of separate stores
- No Alfresco UI is compatible out-of-the-box with multiple stores, so you might require custom code for Alfresco Share to work properly against one or more custom stores

Properly designing your custom code

This section provides some guidelines on how best to design your Alfresco integration/customizations, which MAY have crucial results in the overall performance of the platform.

Tuning your batches

Multi-document batches

In cases of massive content injection/deletion and the use of Foundation APIs for integration and data import in Alfresco, it is important to avoid injecting/deleting one document per transaction into Alfresco. You can tune this in your custom code by creating batches, which you MUST write into Alfresco in a single transaction using Alfresco RetryingTransactionHelper.

For very high load contexts, the batch size CAN be arbitrary values, but this depends on the size of the content and properties being injected (it MAY generate out of memory errors). Alfresco Solutions Engineering provides a SampleInjector to test, tune, and benchmark your injection rates and batches size. Best results using this injector have been achieved using 150 documents per batch for document creations and 50 documents per batch for document deletions.

No multi-injection capabilities are currently available in the RESTful Public API, while in cases of web scripts integration you can extend the functionality of the default Alfresco Upload content web scripts to accept multiple content and perform a multi-document batch writing in the repository. This approach MAY require additional memory and sizing tuning. You SHOULD use Foundation APIs for low level, high load data imports.

Multi-threaded batches

For parallelization of data sets (when the import of a node or of a node-batch is independent to the import of other nodes or node-batches), use a multi-threaded import strategy, which can load multiple nodes or batches in parallel, increasing the overall bulk import throughput substantially.

Closing your results set

To maintain a manageable number of Lucene indexes directories, Alfresco customizations that use the Lucene Search API MUST always close ResultSets after querying Alfresco. Otherwise, due to the Lucene indexing mechanism, the number of physical folders in the index will grow uncontrolled.

To achieve this, wrap all your customizations using the searchService in a final clause such as:

```
try {
 ResultSet resultSet = null;

 resultSet = searchService.query(Repository.getStoreRef(),
SearchService.LANGUAGE_LUCENE, query.toString());
 List<NodeRef> nodes = resultSet.getNodeRefs();
 for (int index=0; index<nodes.size(); index++) {
  //Do whatever you want here
  }
catch (Throwable err) {
 (…)
 }
 finally {
   if (resultSet != null)
    {
          resultSet.close();
    }
 }
```

You MUST do this for every Java customization. JavaScript and FreeMarker Search APIs handle the resultSet closing in the wrapping layer.

Alfresco tuning best practices

This section discusses the most complex and low level of tuning—tuning your Alfresco and support infrastructure.

Supported hardware selection

The first and most important step when choosing an Alfresco infrastructure is to accurately select supported stack components from the Alfresco Supported Stacks.

While Alfresco will function correctly on virtually all modern 32-bit and 64-bit CPUs, for production use, Alfresco recommends a clock speed greater than 2.5 GHz to ensure reasonable response times to the end user. This is true at any level of the stack. Other typical considerations while choosing the proper hardware are:

- Select an application server processor based on the usage profile of your system. Do not choose the CPU based solely on total throughput ratio, but on an accurate matching to the way you use it. Use fully tested stacks over supported platforms where possible.
- As a rule, particularly for high read contexts, you SHOULD consider using a 64-bit processor for fast access to memory and to improve caching and overall performances. In addition, a 32-bit machine will set a restrictive upper bound of 1.5G to the maximum possible JVM heap size, which might not fit high load and concurrency requirements.

Sun Sparc T series warning notice

Alfresco installations have had issues with Sparc T2000/T5200 machines, which are highly parallel computational machines (up to 64 parallel processing threads using a technology called coolthreads), but not suited for long serial repository intensive operations (single thread throughput is lower), such as long batch space/document hierarchy creation.

CPU clock speed is of particular concern for the Sun UltraSPARC architecture, as some current UltraSPARC based servers ship with CPUs that have clock speeds as low as 900Mhz, well below what is required for adequate Alfresco performance. If you intend to use Sun servers for hosting Alfresco, ensure all CPUs have a clock speed of at least 2.5Ghz. At the time of writing, this implies that:

- An X or M class Sun server is required with careful CPU selection to ensure 2.5Ghz (or better) clock speed.
- You SHOULD NOT use T class servers as they do not support CPUs faster than approximately 2Ghz

Alfresco cannot provide specific guidance on Sun server classes, models, or configurations.

JVM tuning

Alfresco JVM tuning can optimize the performances of your system and SHOULD be considered to overcome limitations of the standard JVM configurations, particularly for large repositories.

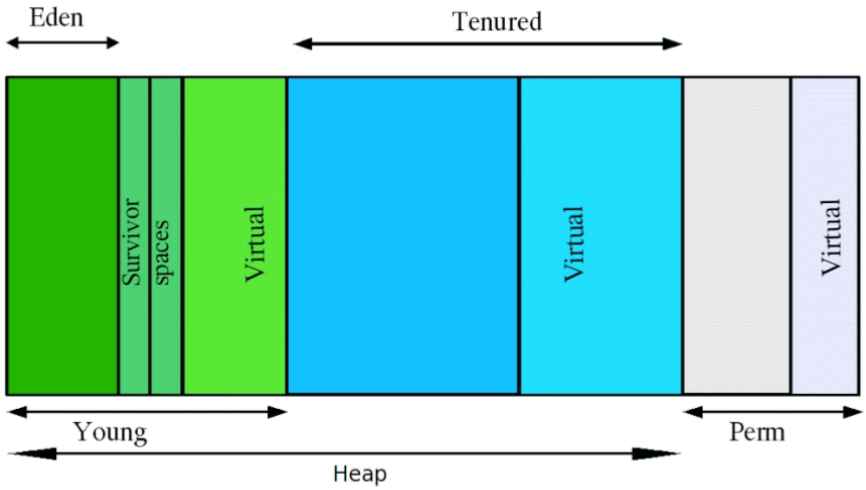
However, aggressive JVM tuning MAY result in an unstable production system. Ensure you properly stress test and stage the JVM tuning before production deployment.

Refer to *Administering an Alfresco ECM Enterprise Edition 3.2 Production Environment* for more detailed information on JVM tuning.

The following sections provide generic guidelines for the JVM high performance tuning. These refer to specific use cases in *Benchmarking Alfresco* with sample configurations.

Sizing

The various memory zones in the JVM are shown in the following figure.



JVM memory map

The first parameter to configure is the maximum amount of system memory that can be allocated to the JVM Heap (-Xmx). Make sure that for each of the various memory portions of the VM, the minimum value is set the same size as the maximum value: the heap (-Xms and -Xmx parameters), the young generation (-XX:NewSize and -XX:MaxNewSize, or by setting -XX:NewRatio, which defines the old gen/new gen ratio), and the PermGen (-XX:PermSize and -XX:MaxPermSize).

- This avoids using the memory resizing algorithms that are not necessary when proper values have been chosen for production/preproduction. It also improves predictability. This may not apply when explicitly using GC ergonomics introduced in Java 5 with throughput/time goals, where it might be desirable to resize the heap dynamically.

  Note: It is not possible, or required to set memory values for the old generation as it is computed from the heap size minus the new generation size.

The New Gen sizing is important to choose properly, particularly for high activity/batch processing jobs where the object allocation rate is much higher than casual/low activity UI use. Allocating one third to one half of the heap, depending on the use case to the new size, (using -XX:NewRatio=2 for one third and -XX:NewRatio=1 for one half), is a good starting point.

- It not necessary to allocate more or much more than half the heap to the new generation. It can lead to excessive major collections if the object promotion for the new generation cannot be accommodated in the tenured generation. It can also break the young generation guarantee when using the serial collector. This would mean that almost only major collections would occur, severely degrading performance.
- You MUST use the -server option activating the JVM's server HotSpot compiler. This parameter is a factor in determining the survivor/eden ratio. In certain hardware configurations (or “server class”), the server option is turned on, but it is recommended to always make it explicit.
- The -XX:+AggressiveHeap option attempts to reserve the maximum of system memory for the JVM. It does not take -Xms and -Xmx into account, which can cause unpredictable behavior, particularly when several JVMs run on the same machine. This option is not recommended.

- Barring unusual cases, do not set very large values for the heap; there is a risk of a large GC penalty. Adding more memory is rarely the solution to out of memory errors; there is usually more to it than just a lack of space.
- If the spill over rate from the survivor spaces to the tenured generation is too high after minor collections, collect information about the generational distribution of the objects held in the survivor spaces via the -XX:+PrintTenuringDistribution option. The log generated with this option will show the cumulative size of all objects of that age retained in the survivor space for each generation (or "age"). It enables you to identify two possible issues:
  1. If the survivor spaces are too small, relative to the Eden, the copy from the Eden space to the first survivor space (also called "from", the other one being called "to") will spill directly into the old generation space. This can be adjusted with the -XX:SurvivorRatio=n option where 'n' defines the eden/survivor ratio. One survivor zone occupies 1/(n+2) of the total allocated to the new generation since there are two survivor spaces. You can observe this in the log if the JVM adjusts the TenuringThreshold to a value that is sub-optimal because only a small number of object generations will fit in the space allocated to the survivor spaces. Lowering the SurvivorRatio will improve performance if more objects can stay in the survivor spaces, as they will have a longer time to die before promotion. If you observe that the survivor spaces are underused, performance can be slightly improved by raising the SurvivorRatio so that the Eden has more space, delaying the minor collections a little.
  2. The information provided by the logs also enables you to deduce if objects coming from Eden and stored in the Survivor spaces are copied from one survivor space to the other more times than they should be. To identify this, observe the log over long periods. It is usual to see the size occupied by each age decrease, as each time, minor collections collect more objects from that age that might have died. If there is repeatedly an age (or several) where the occupied size of these objects is bigger than the previous age, it often indicates that this particular generation contains objects with a long life span.

     To improve that, you can set the -XX:MaxTenuringThreshold=n option to that age minus one (the low point), where 'n' defines the number of copies between the Survivor spaces before an object is willingly promoted to the old generation (hence the maximum age before promotion). If you use this option, do not set a value past the maximum age allowed by the JVM. The maximum is 15 for JVMs >= 1.5.0_06 and 31 before this version (the object header has lost one bit to store the object age, thus the division by 2).
- The -XX:+PrintCommandLineFlags option allows the used options to display on JVM startup, including some implicit options that arise from the choices made with the explicit options.

Garbage Collection (GC)

Your choice of GC strategy depends on the hardware and the use case. A few are described in this section.

- The 'ParNew' GC (-XX:+UseParNewGC) focuses on the new generation, and is relevant for 2+ CPUs/core machines. It can be used along with the CMS whereas the previous implementation (-XX:+UseParallelGC) is similar, but a little less efficient and lacks the synchronization logic necessary to be compatible with the CMS. The use of ParNew alone is recommended for non-interactive work, such as batch processing and /or bulk loading via the Foundation API.
- The CMS GC (ConcMarkSweep), also called Low Pause Collector, focuses on Old Generation and allows certain phases of the collection to share processor resources with application threads to minimize the stop the world state.

- Alongside the -XX:+UseConcMarkSweepGC option, if you have more than two CPUs, investigate activatingXX:+CMSParallelRemarkEnabled. There are two mutually exclusive behaviors that you can select depending on the context:
  - Use the -XX:CMSInitiatingOccupancyFraction=n option that defines at what percentage of old generation capacity a major collection will be triggered. Here, 'n' has a value between 0 and 100. By default, in Java6, it is 92. Lowering this value to around 80, anticipating a major GC collection, will lower the overall GC impact. This option is recommended with more than two CPUs.
  - If the machine does not have many CPUs /cores and you want to maximize the time allocated to application threads, the incremental mode can be used. This periodically yields back processor time to application threads during the concurrent mark phase of the collection with the options -XX+CMSIncrementalMode -XX:+CMSIncrementalPacing, thus minimizing the impact on the application's interactivity and UI responsiveness. However, it can have a detrimental effect on performance when doing load testing/batch processing as the allocation rate is usually much higher than normal in these scenarios and the disconnected nature of this collection's different phases might make it difficult to cope with the live changes during the collection.

    Note: The concurrent CMS GC threads/concurrent application threads are about ¼ for a number of CPUs (or cores) less than 16. The actual expression used by the JVM is as follows:

    ```
    concurrent CMS GC threads = ( ParallelGCThreads + 3 ) / 4 with
    ParallelGCThreads  = number of CPUs <= 8 ? number of CPUs : 3 +
    number of CPUs * 5/8
    ```

    This ratio can be directly overridden by the +XX:ParallelCMSThreads=n option or indirectly by +XX:ParallelGCThreads=n, if necessary. However, these have influence over non-STW phases so it is rarely useful to change the default values.
- You can also set target goals (throughput, pause times, and so on) using GC Ergonomics introduced in Java 5 and improved in Java 6, using the -XX:+UseAdaptiveSizePolicy and specifying desired goals. Examples would be 99% application time using -XX:GCTimeRatio=99. In this scenario, you should only specify a maximum heap size and let the JVM decide the rest to try to meet the goals you set. Note that you only register intent, with no guarantees that your goals will be met. You should closely monitor application performance and GC behavior, reverting back to the "manual" configuration if necessary. Be aware that it might take some time for ergonomics to settle into the chosen values, so application performance and GC overhead will probably not be optimal from the start.

System interaction

Here are a few recommendations about JVM interaction with the underlying OS:

- If the application is using many temp files (such as image conversions), it can be useful to point the JVM's temp directory to a tmpfs partition via the VM option -Djava.io.tmpdir=/path/to/tmpfs. For GNU/Linux machines, there is often a tmpfs partition ready to be used mounted on /dev/shm. On Solaris, /tmp is the default temporary directory for the VM and tmpfs by default. The IO benefit is larger when the application is managing large temp files.
- Swapping: A Linux kernel parameter defines a percentage of RAM utilisation beyond which the kernel will start to pre-emptively swap out pages. For a server, it is recommended to place this parameter to maximize RAM usage and swap as a last resort. The values range from 0 (never swap until the RAM is full) to 100 (swap as soon as possible, at every opportunity). A value of 10 (do not swap until RAM is 90% full) is a good starting point. This parameter is included here as it

is important for JVM performance to avoid swapping. This is equally important for the MySQL Query Planner, which can choose poor query plans if its queries are swapped out without its knowledge. To have this configuration persistent across reboots, add vm.swappiness = 10 to /etc/sysctl.conf. To take this configuration immediately into account after modifying the file, run systctl -p

JVM profiling

The GC logs can be activated with these JVM options:

```
-Xloggc:/tmp/gc.log -verbose:gc -XX:+PrintGCTimeStamps -XX:+PrintGCDetails
```

The logs can then be parsed for analysis by tools such as GC Histo, HP JMeter, or equivalents. One of the most important factors to monitor is the GC overall overhead. Consider that an overhead lower than 1% (or even 2%) of overall application time is acceptable. There should be as few major collections as possible in production - ideally none or a few per day. A running JVM behavior can also be observed without restart using tools such as VisualGC (standalone, or as a jvisualvm plugin), Yourkit Java Profiler, or standard tools that come with the JVM. The in memory object instance distribution can be viewed and analyzed with tools such as jmap (-heap, -histo, -permstat), which is bundled with Sun's JDK.

You can analyze thread dumps with tools such as TDA, Samurai, or equivalents. Under Linux, thread dumps can be obtained by sending the SIGQUIT signal (kill -3) to the JVM.

System tools such as iostat, sar, vmstat and equivalents, and the profilers such as Yourkit Java Profiler can diagnose hardware/software/configuration problems to further improve performance.

Database tuning

Discussing an optimized configuration for each of the supported Alfresco databases is out of scope for this document. However, a general suggestion is that default database configurations are normally not suitable for large repository deployments and MAY result in:

- Wrong or improper support for ACID transaction properties
- I/O bottlenecks in the database throughput
- Excessive queue for transactions due to overload of connections
- In active-active cluster configurations, excessive latency

The following sections provide some general hints to consider when stressing the database component during Alfresco operations, for example, high concurrent load.

MySQL suggested configurations

For a sample database configuration, this example uses MySQL, the most popular database for Alfresco deployment. A my.cnf configuration file is provided to optimize performances of Alfresco when using this database.

For improved reliability and performances, upgrade the MySQL JDBC driver to the latest version (use the latest JDBC driver for MySQL).

Database sizing

One Alfresco node typically stores from 1 to 5 kilobytes of information in the database (excluding any database specific information and depending on the specific content model), so you should use this metric to size the database backing disks.

Database connection pooling

Thread pool configuration

A default Alfresco instance is configured to use up to a maximum of 40 database connections. Because all operations in Alfresco require a database connection, this places a hard upper limit on the amount of concurrent requests a single Alfresco instance can service (40), from all protocols.

Most Java application servers have higher default settings for concurrent access, and this, coupled with other threads in Alfresco (non-HTTP protocol threads, background jobs, etc.) can result in excessive contention for database connections within Alfresco and poor user performance.

If you are using Alfresco in anything other than a single user evaluation mode, you SHOULD increase the maximum size of the database connection pool to at least [number of application server worker threads] + 75.

For the Tomcat 6 default HTTP worker thread configuration and with all Alfresco thread pools left at the defaults, this means this setting should be at least 275.

Add this db.pool.max setting to your alfresco-global.properties. It is recommended that you place this setting with the other db-related settings in the properties file.

Note: After increasing the size of the Alfresco database connection pool, you must also increase the number of concurrent connections your database can handle to at least the size of the Alfresco connection pool. Alfresco recommends configuring at least ten more connections to the database than is configured into the Alfresco connection pool to ensure that you can still connect to the database even if Alfresco saturates its connection pool. Remember to factor in connections required by other applications using the same database server as Alfresco. See *MySQL suggested configurations* for database configuration.

The precise mechanism for reconfiguring your database's connection limit depends on the relational database product you are using.

DBCP 1.2.1 patch

In cases of high load and short lasting transactions, a DBCP 1.2.1 (Database Connection Pool Apache library shipped with Alfresco) issue might result in your Alfresco transaction wasting unnecessary time in setting auto-commit state of the transaction in the database. Patching DBCP CAN be an option after profiling database accesses and finding a high number of auto-commit related statements.

Note: Check with Alfresco Support before applying this patch to Alfresco.

Hibernate tuning

Hibernate is the ORM framework used by Alfresco to manage RBDMS persistency and, in conjunction with Spring, to manage the transactional layer of Alfresco. Hibernate can be configured/tuned to match non-standard Alfresco usage profiles. Properly use transactions to match high load or concurrency requirements.

Hibernate optimizations

There are no specific Hibernate optimizations to mention at the time of this writing. Please refer to official Hibernate optimization guides to improve possible bottlenecks at the Hibernate level. Note that Alfresco is moving out of Hibernate in the next versions and will leverage iBatis more (Alfresco version 3.4 is the target release for the complete Hibernate removal, apart from JBPM).

Lucene indexing tuning

A full discussion about Lucene indexing best practices is out of scope for this document. For Lucene specific performance enhancements, you SHOULD consider official Lucene documentation.

It is definitely possible, nevertheless, to tune the search and indexing configuration to properly match requirements, especially memory and CPU resources allocated for the Alfresco platform.

While most of the defaults for the indexing engine configuration defined in repository.properties are typically valid for standard Alfresco deployments, large repository deployments or high concurrency instances SHOULD require specific tuning Alfresco Lucene related configuration in a custom alfresco-global.properties.

The full description of the typical parameters to tweak appears in the Alfresco Wiki while more detail on their use is in *Benchmarking Alfresco*. The following sections describe other general-purpose best practices related to indexing.

Lucene index Merge Factor

The main way to tune Lucene indexing performance is by using the following standard Lucene parameters:

`lucene.indexer.mergeFactor`

`lucene.indexer.maxMergeDocs`

Refer to the Lucene documentation for more detailed information.

Disabling in-transaction indexing

For systems that make use of the injection of large numbers of documents, and for which the documents do not need to be immediately searchable, disable in-transaction indexing to improve performance. This defers update of the local index to a later stage, committing the transaction to the database and content store beforehand. Be aware that indexing is asynchronous for almost all content that undergoes transformations before being indexed anyway (see

Lower text transformation time), so this approach SHOULD especially be used for large text and metadata imports.

- This can be simply achieved by setting the following parameter in alfresco-global.properties:

```
index.tracking.disableInTransactionIndexing=true
```

Limiting the number of indexed tokens

If full text indexing is not needed, consider limiting the maximum number of tokens indexed by Lucene to off-load the content platform as much as possible.

Note: Do not use this approach when large metadata fields and content binaries are indexed, as it will strongly impact the search results consistency and perceived correctness.

To achieve this, tune the following property in alfresco-global.properties:

```
# The number of terms from a document that will be indexed
lucene.indexer.maxFieldLength=50
```

The value is provided as an example only. You MUST tune it based on your specific content model and indexed fields.

Lower text transformation time

For dynamic and accelerated performance of the indexing of documents that require prior transformation to text, you CAN lower the following parameter in alfresco-global.properties:

```
# Millisecond threshold for text transformations
# Slower transformers will force the text extraction to be # asynchronous

lucene.maxAtomicTransformationTime=20
```

This causes all documents that require prior transformation to text and that have a higher transformation time than 20ms to be transformed and indexed out of transaction.

Note: If you want to use this approach, contact Alfresco Support for further validation. In most situations, the defaults will meet your needs.

Lucene index tracking in cluster

Index tracking is the process of keeping Lucene indexes up-to-date with executed transactions in timestamp order and in a consistent state with what is on the content store.

Tracking occurs for the following:

- Server restart
- Upon detection of new transactions between nodes in the cluster

The Defaults meet most requirements. However, you MAY want to tune the following properties in alfresco-global.properties:

```
index.recovery.maximumPoolSize=5

index.tracking.maxTransactionsPerLuceneCommit=100
```

File system for storing indexes considerations

To improve indexing performances for a Linux deployment server, you can store Lucene indexes on a tmpfs file system. This can provide up to a 30% gain in the document indexing speed. This file system type is temporary, so you must set a hot backup strategy for the Lucene indexes, using persistent storage as target. For this approach:

- Use the Alfresco LuceneIndexBackupJob
- Perform the backup during low activity timeframes
- The time between backups of the indexes must be shorter than the retention period specified for the tmpfs. The backup frequency MUST always be greater than the tmpfs configured retention periodicity for consistent indexes. The database backup must be more recent than the index.

It is recommended that you use the fastest physically attached disk to store indexes. SCSI disks are preferred over IDE disks.

Ehcache tuning

Alfresco L2-cache (Ehcache) tuning can significantly improve the performance of an Alfresco repository, particularly for Read operations. Write operations may also benefit from a faster retrieval of parent nodes.

The settings used for the caches depend on your particular use case and your Alfresco server memory capacity. However, the default ehcache-default.xml file meets the needs for most systems. It is currently setup for approximately 512MB of cache heap memory - this is the recommended default for a Java heap size of 1GB.

Adjusting the L2 cache size

To adjust the values of L2-caches (using ehcache-default.xml and ehcache-custom.xml), multiply the number of maximum element entries in the L2-cache by the same factor used to increase the JVM Heap size (-Xmx). By default, Alfresco is configured for 512 MB cache size (half of 1GB total - Xmx1GB).

You SHOULD also consider using a 64-bit environment, as caches strongly depend on a quick evaluation of cache element access rights.

Disabling L2 cache

EHCache is used as the L2 cache and is distributed across the cluster.
Sometimes disabling Ehcache CAN be required and MAY be considered for debugging purposes.

Set the following parameter in alfresco-global.properties:

```
hibernate.cache.use_second_level_cache=false
```

It may improve performance and stability when using post 3.2 WCM editorial site clustering, depending on the Read/Write load and data size.

Tuning unused Alfresco features

It is possible to configure Alfresco to reduce overhead from unused features. Use this technique sparingly since some configurations may hamper data consistency.

Important: Do not apply any of the recommendations in this section without consulting Alfresco Professional Services or Alfresco Support.

Disabling subsystems

The Alfresco Subsystems API is designed to offer independence of functionality and can dynamically be started and stopped via JMX. In certain cases, this might include features like:

- IMAP
- File servers
- Third party applications

Note: Authentication is mandatory.

Reducing the Virtual File Server (VFS) worker thread pool

In case of low Alfresco VFS (Virtual File Systems) usage, you MAY reduce the number of Alfresco VFS worker threads (these threads provide CIFS, FTP, and NFS services for a repository).

By default, Alfresco will start 25 VFS worker threads, and this pool may increase to a total of 50 worker threads.

To tune these settings:

1. Navigate to the following configuration file:

   ```
   ${ALFRESCO_HOME}/tomcat/shared/classes/alfresco/extension/subsystems/fileservers/default/default/file-servers-custom-context.xml
   ```

2. Add the following <bean> override:

```
<bean id="fileServerConfiguration"
class="org.alfresco.filesys.config.ServerConfigurationBean"
parent="fileServerConfigurationBase">
      <property name="cifsConfigBean">
         <ref bean="cifsServerConfig" />
      </property>
      <property name="ftpConfigBean">
         <ref bean="ftpServerConfig" />
      </property>
      <property name="nfsConfigBean">
         <ref bean="nfsServerConfig" />
      </property>
      <property name="filesystemContexts">
         <ref bean="filesystemContexts" />
      </property>
      <property name="securityConfigBean">
         <ref bean="fileSecurityConfig" />
      </property>
      <property name="coreServerConfigBean">
         <bean
class="org.alfresco.filesys.config.CoreServerConfigBean">
            <property name="threadPoolInit">
                <value>25</value>
            </property>
            <property name="threadPoolMax">
                <value>50</value>
            </property>
        </bean>
      </property>
   </bean>
```

3. Reconfigure the initial (“init”) and maximum (“max”) thread pool size as appropriate for your intended usage.

ACLs and permission checks

During specific operations, such as during bulk upload operations, you can completely disable permission checks in the repository service layer by:

- Running as the “system” user to prevent permissions from being checked. Running searches as an “admin” user significantly reduces the time spent in evaluating permissions.
- Using lower case beans (for example, nodeService, contentService, and so on) for your Foundation API integration custom code. These are the original Alfresco service layer beans prior to security (and transaction) interceptor wrapping.
- Tuning the time and number of ACLs checked via configuration by modifying the following parameters in alfresco-global.properties:

```
# Properties to limit resources spent on individual searches
# The maximum time spent pruning results
system.acl.maxPermissionCheckTimeMillis=10000
# The maximum number of results to perform permission checks
# against
system.acl.maxPermissionChecks=1000
```

This configuration MAY impact the large search results for non-system users as a too low ACL dedicated time or number of results to check against prevents the repository from scanning through all results for a consistent permission filtered output.

Aspects and rules

Certain aspects, such as Versioning, can be disabled (not applied to content types) while running specific repository intensive operations. In addition, rules can be temporarily disabled programmatically when using the Foundation Integration API by:

```
serviceRegistry.getRuleService().disableRules()
```

Quotas

Quotas in 3.2 are enabled by default. Users who have just upgraded to this version MAY perceive this as performance degradation.

To disable quotas and usages, configure the following parameter in alfresco-global.properties:

```
system.usages.enabled=false
```

Audit

Audit in Alfresco 3.2 and 3.2r supports RM features. It is disabled by default to avoid audit interception overhead. In contexts where audit is enabled but only required on specific operations, audit SHOULD be profiled to only intercept specific operations. Where auditing must coexist with large import/injection procedures, which might not need to be audited, it SHOULD be considered to use a proper load balancing policy that isolates the node that is supposed to perform the import, and then configures this node to have auditing completely disabled.

Set the following parameter in alfresco-global.properties:

```
audit.enabled=false
```

Handling ML manually

In cases of high document injection, particularly where multilingual features are not needed, you can disable the cost intensive Alfresco-managed multilingual features by preventing the multilingual Spring interceptor from acting on content.

When using the Foundation API for high load imports, you SHOULD handle MLProperties by disabling the following interceptor:

```
MLPropertyInterceptor.setMLAware(true)
```

Then create ML-aware content as:

```
Map<QName,Serializable> titledProps = new HashMap<QName,Serializable>();
titledProps.put(ContentModel.PROP_TITLE, new MLText(Locale.ENGLISH,name));
nodeService.addAspect(content, ContentModel.ASPECT_TITLED, titledProps);
```

Disabling Trashcan

Alfresco offers soft delete functionality. By default, content is not actually deleted on delete operations (both via the UI or programmatically).

It MAY then be relevant (in bulk replacement such as injection + deletion) context to disable the Trashcan behavior. This can be achieved at a few different levels:

- Per content instance, you can programmatically (or via a rule) apply the sys:temp aspect: this aspect marks content as transitional, instructing the repository not to keep references (for example, the Trash Bin) to the document upon deletion.
- Per content type, you could configure <archive>false</archive> flag in the specific type definition (or in a mandatory-aspect of that type).
- Globally, you could disable the Archive store by overriding the configuration of the Store map.

Fine tuning LDAP synchronization in a cluster

To optimize performance, particularly for an initial bulk import of LDAP users in an Alfresco clustered installation, remember that cluster Ehcache tracking will be generating additional computational cost to the import. Alfresco recommends you disable additional cluster nodes when performing the initial import in this situation.

Scaling Alfresco solutions

This section focuses on real life scenarios, specifically large repository deployments, and provides specific best practices on deploying a scalable, high performing, and maintainable Alfresco instance. This section also discusses tools you can use to stress test, benchmark, and profile the suggested deployments.

Benchmarking Alfresco

This section provides all known Alfresco benchmark results as of Version 3.2.

Alfresco benchmark tools

This section provides an overview of the various tools and approaches to stress test Alfresco performances and functionality.

Alfresco server benchmarks

Alfresco provides a package called alfresco-bm.jar. This simple client accesses Alfresco internal services via remote RMI loading. This tool is recommended to test performances and throughput of an Alfresco server under high concurrency conditions (an arbitrary number of threads can be started) and under massive injection conditions (process can be single-threaded and load an arbitrary number of documents).

JMeter

Apache JMeter is designed to load test functional behavior and measure performance. It can be used to ramp-up threads and stress HTTP interactions with the server, as well as run JUnit tests in stress-test mode.

You SHOULD consider JMeter, especially in the case of a custom injector application that can then be stress tested using the framework. Alternatively, you CAN use JMeter to directly testRead/Write performances on HTTP-based integration, directly stress testing the RESTful or CMIS APIs in Alfresco.

If you need a sample injector application to be ramped up with JMeter, refer to the following section.

Sample Injector

Alfresco Solutions Engineering have created a modification of the standard Alfresco sample FirstFoundationClient to offer a simple injection/content loading suite for testing Alfresco ingestion performance. The SampleInjector is an Eclipse project comprising a few configuration files and a sample injector class.

This injector is based on the Alfresco Foundation API and must be deployed as an Alfresco repository extension. You SHOULD consider using the injector when approaching architecture validation and tuning in the context of massive document injection, as it implements most of the design and tuning best practices previously discussed.

Contact Alfresco Professional Services for further details.

CIFS benchmarking

CIFS is possibly the easiest protocol to be tested, as stress testing the Alfresco JLan CIFS interface comprises:

- Mounting the Alfresco CIFS share on a local machine
- Writing a simple shell/Java/etc script that creates files on the local file system
- Ramping it up either using JMeter or directly using Java multi-threading capabilities to test high concurrency scenarios, while increasing file number/size to test massive injection scenarios

Refer to *Administering an Alfresco ECM Enterprise Edition Production Environment* associated with the version you are using for more detailed information.

Profiling Alfresco

In normal operation or, particularly, during a benchmark phase, it is vital to collect profiling information you cause to tune your application performance. The following provides an overview of profiling tools at different levels.

JMX monitoring

With Alfresco 3.2 Enterprise and later, you can use any standard JSR-160 JMX console to connect to Alfresco for monitoring functional and system parameters.

For details on how to use the different capabilities of JMX monitoring, refer to *Administering an Alfresco ECM Enterprise Edition Production Environment* associated with the version you are using.

AuditSurf

AuditSurf is a powerful, user-friendly, and open source remote monitoring tool for Alfresco. It is a separate web application, based on Spring Surf, that allows you to monitor an Alfresco instance at all levels (functional and system resources).

Alfresco Nagios integration

Nagios (and its recent fork Icinga) are leading open source enterprise monitoring tools. A plugin lets you monitor and check Alfresco-specific information aggregated in the standard monitoring console offered by these products.

JVM profiling

See *JVM profiling* for JVM profiling tools.

Logging and debugging

A complete description of the logging classes for specific Alfresco functionality is provided in *Administering an Alfresco ECM Enterprise Edition 3.2 Production Environment*.

Remote debugging

When all other methods (debugging and profiling) fail to identify bottlenecks and debug the flow of operations internal to Alfresco, remote debugging CAN be helpful. This comprises running your JVM with jpda debugging enabled, which will publish all debugging information on a TCP port that can then be managed using any IDE with remote debugging capabilities.

Alfresco scalable solutions

Alfresco is typically customized and integrated into the business infrastructure and processes. Three example scenarios that require optimization are included in this section.

Massive content injection

At some point in most projects, there will be a need for rapid content injection. This section focuses on tuning your system for optimum performance.

Solution description

Alfresco is often used as an archive system for storing millions of scanned documents. The typical scenario here is an external application, such as a scanning solution, rapidly injecting a large number of documents

- Content store size, long term storage, and injection rate are the most important parameters for this solution. Features of this solution are:
  - Document Reads are done by a small number of users.
  - Content is directly accessed via NodeId by an external application or a search on the metadata.
  - Documents are not full-text indexed due to the large volume of documents
  - Documents will only have one version and are Read-only
  - Although documents might be accessed multiple times for processing, the emphasis is on fast injection.
  - The folder structure design is fixed and the folder structure will grow (linearly) with the number of documents over time

Solution functional requirements

The functional requirements are typically:

- 100 – 1,000 users (Read-only access)
- 10,000 documents injected each night
- 20 million+ documents per year
- Document type: PDF, TIFF, JPG
- Read-only, no full text indexing, basic content model
- Few to no rules and actions
- No content transformations
- Centralized content management
- Interface: external application, Foundation API, web scripts, CMIS. No CIFS/WebDAV/FTP or web client

Non-functional requirements

This scenario has the following throughput and document lifecycle requirements.

Throughput
Main throughput focus should be on injection and transaction optimization.

Document lifecycle
Typically, a document is written only once (one version) and could be read never or several times over a long period (years). The write constraints are mostly determining the sizing of the system due to large volumes in a short period of time (hours).

Suggested integration and customization approach

Referring to customization and integration APIs described in *Customization and integration*, implement the bulk injection using the Java Foundation API on Alfresco repository to benefit from all of the optimizations of an in-process interaction.

Suggested architecture and scaling policy

This configuration is typically CPU and memory (depending on document content) intensive, so scaling up the different components of the platform SHOULD be preferred over scaling out. Specifically:

- Processors should be scaled up in power of the CPU rather than number of CPUs
- Database can be scaled up

With reference to architectures described in *Introduction to the Alfresco architecture* and to the up and out scaling models described in *Scale out and scale up*, two different options are suggested, depending on the nature of the injection.

- If a single bulk import can be partially or completely logically divided in independent batches (for example, independent data sources or copy from different geographical sites), the client import application could actually benefit from parallelism in document ingestion.
- A simple distributed deployment model as no web tier is required, with a simple repository scaling out model, as any replication will introduce high cost for data storage. For load balancing strategies, a round robin strategy can increase the overall injection rate.
- If the bulk import is atomic and serial, you SHOULD scale up the infrastructure to scaling out as it many introduce overhead during injection due to cache synchronization. In this case, a pure, simple distributed deployment model is recommended on a single non-clustered instance of content platform. In this case, you MUST scale up in CPU and memory.

Suggested design best practices

With reference to best practices discussed in *Alfresco design best practices*, those that SHOULD be considered are:

- As much as possible, use a balanced content tree to speed up content access. See *Taxonomy design best practices* for more information.
- If your system is performing batch data replacements (delete and recreate objects), you MAY want to disable the Trash Can using cm:temporary. See*Content model design best practices* for more information.
- You CAN use multi-store configurations to increase index manageability (see *Multi-store repository*). Specifically, you SHOULD use multi-tenancy when there are isolated information silos. You CAN use custom multi-store Foundation API customizations but with all the

limitations described in *Custom multi-content store impacts* (for example, no usage of Alfresco standard user interfaces).

- You MUST tune import code to use a proper batch size when either creating or deleting content in the repository, as per *Tuning your batches*.
- You MAY want to consider applying the DoNotIndex aspect on imported content.

Suggested tuning best practices

With reference to best practices discussed in *Alfresco tuning best practices*, those that SHOULD be considered are:

- Fast disks (preferably local disks) for both indexes and database, as high IO throughput is vital. Alternatively, the high growth of content can become unmanageable on a local disk, so you MAY use content storage policies (see *Quotas and usages*) to control the storage media. An example of this is implementing a policy for which nightly imports write their content in a local high speed disk (to achieve maximum injection throughput) while after the import content is actually moved to SAN/NAS allowing for more scalability but lower I/O speed.
- MySQL and MSSQL server are the database that have shown better performances in these kinds of batch imports. For MySQL, see *MySQL suggested configurations* for optimized configuration.
- After proper database connection profiling if autocommit related statements are perceived to be a bottleneck, see *DBCP 1.2.1 patch* for a DBCP patch that CAN solve the issue.
- Especially in cases of deep content hierarchies created by the import, limiting Hibernate session flushing CAN provide non-trivial performance improvements. See *Preventing Hibernate over-flushing* for more information.

A number of optimizations regarding the default indexing process CAN and SHOULD be applied to simplify the import process:

- Regarding batch non-interactive imports, it MAY be very useful to disable in-transaction indexing so that indexing is asynchronous and batch import transactions can be closed without waiting for indexing. See *Disabling in-transaction indexing* for more information.
- Alternatively, it is possible to have less aggressive optimizations by raising the Lucene merge factor (see *Lucene index Merge Factor*) to allow fewerflush-to-disk operations, or by limiting indexing time for texts/number of indexed words. See*Limiting the number of indexed tokens* and *Lower text transformation time*, respectively, for more information.
- L2-cache maintenance (check, retrieval, invalidation) can represent an overhead during long import processes (which are typically invalidating the cache continuously), so you CAN consider disabling the L2-cache for additional performance improvement. See *Disabling L2 cache* for more information.

  Note: This configuration MAY require server restart (for a single node) to disable/re-enable the cache before/after the import.

At the Alfresco level, optimizations that might significantly increase the overall ingestion throughput include the following:

- You MUST consider Disabling Audit and quotas before starting an import, as performance will be drastically improved. See *Audit* and *Quotas* for more information.
- As per *Disabling subsystems*, stopping subsystems, such as file servers (not used in this use case) and IMAP, MAY be useful to provide more resources to the actual import process.
- You SHOULD consider running the import as system user or lowering/disabling permission checks as it might improve throughput drastically. See *ACLs and permission checks* for more information.

- You CAN disable rules if they are not needed for import business logic. See *Aspects and rules* for more information.
- Handling multilingual properties manually CAN provide additional performance boost. See *Handling ML manually* for more information.

Sample JVM configuration

A sample JVM configuration for batch processing via Foundation API follows, and assumes dedicated 8GB RAM machines.

Note: This is an example and SHOULD be tuned on the specific CPU speed and number of cores. In addition, it MUST be reviewed with Alfresco Support or Solutions Engineering for further optimization.

A JVM command line with GC logs activated could look like:

```
-server –Xms6G –Xmx6G -XX:PermSize=256m
-XX:MaxPermSize=256m -XX:NewRatio=2 -XX:+UseParNewGC
-Xloggc:/tmp/gc.log -verbose:gc -XX:+PrintGCTimeStamps
-XX:+PrintGCDetails
```

Highly concurrent usage

This section describes a scenario with high concurrent usage.

Solution description

Alfresco is often used to publish content on a web application, a public website, or Intranet. A very high number of users (tens of thousands) will access a relatively small number of objects (several thousand). Content is Read-only for these users and could be updated periodically (weekly or monthly). The content is accessed via custom build web scripts or Surf custom applications. Mostly users access content anonymously. The folder structure is managed centrally and relatively consistent/frozen.

Solution functional requirements

This is typically:

- 1,000 – 100,000 users (Read-only)
- 10 - 50 authors (write)
- 1,000 - 10,000 assets published
- 100 asset updates per day
- Document types: PDF, images, HTML, XML
- (Optional) Full text indexing of all content
- Few to no rules and actions
- Content transformation only on authoring side
- Centralized content management
- Interface:
  - End users: Custom web applications/web scripts application/Surf application
  - Authors: Alfresco web client, WebDAV

Non-functional requirements

This scenario has the following high concurrency and high availability requirements.

High concurrency
You want to offer a sub-second response time for any front-end user request.

High Availability
As those systems are mission critical (public facing), HA is required for both the authoring process and content delivery process.

Suggested architecture and scaling policy

Consider the following while designing the solution:

- HA SHOULD be assured for both authoring and delivery.
- Read load is between 2 and 4 orders of magnitude higher than write load.
- Scaling out is required for the content platform to ensure HA and adequate performances in both the authoring and the delivery tier. You CAN consider scaling up (multiple CPUs) the web tier's underlying infrastructure to improve service for end users.
- For a custom, possibly Surf, front-end (referring to architectures in *Introduction* to Alfresco content platform architecture) the suggested configuration is to have a distributed content platform deployment that can separate the load on the front-end tier independently from the platform. See Distributed content platform deployment model for more information.

The suggestions based on this architecture, referring to the architectural scaling models, are:

- A three-node simple repository clustering for the content platform (plus optionally one node in hot standby for HA > 99.95, meaning in cluster but not pointed to by the load balancer) that can allow enough performance for both the authoring and delivery processes. See *Simple repository clustering* for more information.
- A transparent content platform scaling out model for the web tier, so that the web tier can be scaled independently from the platform. See *Web tier clustering* for more information.
- For strong profiling of the application in two very different processes with different requirements, consider splitting Reads/Writes using a load balancer. The suggested approach, by default, is to point to two out of three servers for the content read operations (custom front-end→ content platform), with only one out of three dedicated to content write operations (Alfresco UIs → content platform). See *Load balancing* for more information.
- The advantage of having three active nodes is that access can be dynamically reconfigured via a load balancer. For HA, if a Read-only node stops running, the Write-only node also can be pointed to for read-operations (while ensuring a high QoS for external users), and vice versa. If the Write-only node goes down, one of the Read-only nodes can be configured to serve the authoring process. For additional security and stricter HA requirements, it is possible to have a fourth node in hot standby and avoid these fallback policies.
- Caching is fundamental at all levels, and MUST be considered to offload the Alfresco content platform, particularly considering the medium/low ratio of content volatility. It is recommended you set up a caching infrastructure as follows:
  - Front-end caching using any HW/SW web cache. Using Apache Httpd, it is possible to scale indefinitely (by scaling out Apache instances) in concurrent users served. The front-end cache will web-cache pages of the custom front-end web-tier application using standard HTTP cache validation/invalidation methods.
  - Caching between the web- and content-platforms MAY also be considered (for the Read-only instances) to be able to cache at a more fine grained level (typically web tier will

aggregate multiple atomic Alfresco content, cached in this case into a more complex page, which will be cached by the front-end cache.

- Web script caching optimization is key, as it must be configured in orchestration with the external web caches.

When this type of deployment is geographically distributed, there are few possible solutions that can be used to reduce latency and match the sub-second response time requirement:

- Develop a Custom CMIS/RESTful/Web Services-based sync
- Distribute content using caching. For example, deploy multiple web tiers and intermediate caches locally to the different access locations and serve as much content as possible from there. Each cache CAN also be preloaded with typically accessed content to improve performances and reduce real hits on the Alfresco platform.

Suggested design best practices

With reference to best practices discussed in *Alfresco design best practices*, those that SHOULD definitely be considered in high concurrency contexts include the following:

- In a centralized (top-down) content management organization, it is vital to impose a balanced content hierarchy.
- Fine tune property indexing (disable indexing of properties you are not searching on) in the content model to make indexes lighter and easier to search.
- You MUST close any Lucene result set that is searched through the foundation APIs. See *Closing your results set* for more information.
- As front-end access is unauthenticated and provided security policies allow it, retrieve Alfresco content from the custom front-end using a system user. This disables ACLs checking and allows effective search retrievals.

Suggested tuning best practices

With reference to best practices discussed in *Alfresco tuning best practices*, those that SHOULD definitely be considered for this context include the following:

- Database tuning is fundamental and scaling out the database MAY be an option in case, while profiling the application, the database proves to be the bottleneck limiting connection processing. See *Database tuning* for more information.
- Database connection pool size CAN be raised to support high concurrency (even if most user requests SHOULD not hit the content platform directly, and will normally end up hitting the intermediate cache layer and retrieving on-volatile content). See *Thread pool configuration* for more information.
- The Ehcache number of cached objects can be increased proportionally with the RAM assigned to the alfresco repository process. Assuming that for these requirements 8GB or 16GB RAM per node is a valid sizing, you can multiply the number of objects in the cache by 8 or 16 times respectively. See *Rule of thumb for L2 cache size* for more information.
- As content updates are infrequent (with respect to the high Read-only load), it MAY be possible to fine tune (decrease in this case) the Lucene merge factor to obtain a more interactive index flush to disk and avoid filling RAM with large indexes to be flushed. In addition, this CAN keep the number of Lucene segments low on disk, speeding up query performances. See *Lucene index Merge Factor* for more information.
- A general tuning of Lucene parameters is suggested in the initial prototyping phase. This must be performed running stress tests (using alfresco-bm or JMeter) in similar conditions as

production requirements and profiling/monitoring with the Alfresco JMX console. See *Lucene indexing tuning* for more information.

Sample JVM configuration

The sample JVM configuration for high volume that follows assumes dedicated high-volume clustered 64-bit, dual 2.6GHz Xeon/dual-core per CPU, 8GB RAM machines.

Note: This is an example and SHOULD be tuned on the specific CPU speed and number of cores (this configuration assumes at least two CPU/cores per node). In addition, it MUST be reviewed with Alfresco Support or Solutions Engineering for further optimization.

JVM command line with GC logs activated, JMX remote management, and temp files on tmpfs could look like:

```
-server -Xcomp -Xbatch -Xss1M -Xms3G -Xmx3G
-XX:+UseConcMarkSweepGC -XX:+CMSIncrementalMode
-XX:NewSize=1G -XX:MaxPermSize=128M
-XX:CMSInitiatingOccupancyFraction=80
```

Enterprise collaboration platform

This section describes an Enterprise collaboration scenario.

Solution description

Alfresco is often used for collaboration and document management. Typically, this involves a medium size group of users (several hundreds, thousands) with distinctive use case profiles, creating multiple types of documents (such as Office, drawings, PDF, images, photos) and document versions during office hours (could be 24 x 6 for global systems).

A typical system will hold a few thousand to a few hundred thousand documents. If it is also used as an archive, this could grow to several million documents. In the case of collaboration, other actions will be performed (such as search, folder browse, check-out/in, set properties, and workflow) that could result in new content, like discussions, blogs, links, and appointments.

Normally, there is a wide variety of user profiles and a single user can use a different profile over a period, for example, to upload a timesheet once a week, collaborate with team members on a large project for several months, or download/upload a group of documents for personal or group use before traveling.

Users who move from a file server to Alfresco may use CIFS to migrate documents from the file server and/or local disk to Alfresco, if this has not been done centrally. This will significantly increase the load on the system and local networks.

Solution functional requirements

Due to the large variety of profiles and document types, system usage is very difficult to predict.

This is typically:

- 100 - 10,000 users (Read and Write)
- 10,000 - 1,000,000 documents (10,000,000 if also archived)
- 1,000 document updates per day
- Multiple versions per document
- Document types: Office, drawing, images
- Full text indexing on all documents

- Multiple content models
- Multiple (advanced) rules and actions
- Heavy use of content transformations (PDF, Flash, thumbnail)
- Dynamic ACL (many changes over time)
- Decentralized content management
- Interfaces: All
  - For DM
  - For Collaboration - mainly Alfresco Share and SharePoint Protocol Support
  - Web scripts for custom build applications/portlets

Non-functional requirements

This scenario has the following non-functional requirements.

CIFS/WebDAV/FTP high interaction
CIFS, WebDAV, and FTP are very powerful interfaces for users to do batch transactions. If not used and managed correctly, this could cripple any system. In addition, CIFS is known to be a chatty protocol sensitive to longer latencies and, therefore, should only be used if the distance between client and server is small (same building, block, campus). For WAN connections, use WebDAV and FTP.

Global system
No maintenance windows allowed.

Suggested architecture and scaling policy

This solution presents a very heterogeneous usage and load profile, so you must build a modular architecture, allowing dynamic configurations and independent scalability for the different components.

Considering the specified requirements (the heavy use of file systems and no maintenance windows allowed) and referring to architectures in the *Distributed Content Platform Model*, you SHOULD consider deploying:

- A distributed content platform deployment model with two active Alfresco repository nodes in simple repository clustering. See *Simple repository clustering* for more information.
- A two-node Share cluster in the web tier that can scale independently from the platform using the transparent content platform-scaling model. See *Web tier clustering* for more information.
- Front and intermediate load balancers implementing round-robin policies both for Alfresco Share and the repository, and load balancers to implement scheduled exclusion for operational maintenance (see *Load balancing*) without maintenance windows.
- (Optional) One passive node in hot standby for Alfresco Share and one for the repository to be used for disaster recovery and to support additional flexibility during operational maintenance and/or to be made active if additional scalability is required.
- Being a transformation engine under heavy load, Open Office SHOULD be deployed on a separate server with the two-fold advantage of having resources completely dedicated to transformations and off-loading the Alfresco repository, thus preventing service interruption due to CPU transformation peaks. As of Alfresco 3.2r, the OOOJodConverter subsystem should be used to scale out an Open Office instance at the applicative level (with all the benefits of Alfresco JMX subsystems configuration). See *Scaling out the support infrastructure* for more information.

Both scaling out (allowing potentially more concurrent connections) and scaling up infrastructure (for supporting large batch CIFS/WebDAV operations) are valid approaches to increase performances/user experience in this context.

Suggested design best practices

Referring to design best practices discussed in *Alfresco design best practices*, those that SHOULD be considered in this context are:

- A balanced content tree is fundamental, even if difficult to impose, as this is decentralized content management. Try to use a meet in the middle strategy that provides top-level organization (main folders, common tags taxonomy), but leaves freedom to organize collaboration independent environments.
- Be very careful using versioning as this MAY easily end up flooding your repository with unnecessary content versions. Considering the high number of renditions created, the Versioning aspect SHOULD NOT be applied to cm:content, but should be profiled and applied to custom types that definitely require it. See *Content model design best practices* for more information.
- Given the typical model complexity of collaboration and DM context, you SHOULD leverage the modeling power of Alfresco. For example, maximize the number of aspects used to share properties between types to simplify the indexing process and overall index size.
- In a heterogeneous and decentralized content management context, using quotas avoids uncontrolled content growth and over-stress of the transformation engine (and virtual file systems) due to overly large files added to the content lifecycle in the repository. See *Quotas and usages* for more information.

Suggested tuning best practices

With reference to Alfresco tuning best practices, those that SHOULD be considered for this context include:

- Database tuning is fundamental and scaling out the database MAY be an option in case, while profiling the application, database proves to be the bottleneck limiting connections processing. See *Database tuning* for more information.
- Database connection pool size CAN be raised to support potentially highly concurrent scenarios. See *Thread pool configuration* for more information.
- The Ehcache number of cached objects can be increased proportionally with the RAM assigned to the Alfresco repository process. Assuming for these requirements that 8GB RAM per node is a valid sizing, you can multiply the number of objects in the cache by eight. See *Adjusting for L2 cache size* for more information.
- With the requirement of full text indexing for all content and metadata, you might want to tune Lucene to disable in-transaction indexing to allow transactions to complete before text indexing is completed. See *Disabling in-transaction indexing* for more information.
- Storing application server temp files and indexes (provided a proper backup strategy is in place) on a temporary file system might improve indexing and transformation performances (using temp files). See *File system for storing indexes considerations* for more information.
- Tuning the number of threads assigned to the VFS subsystem can be an option to increase/decrease the supported concurrency on the CIFS/WebDAV interfaces. See *Reducing Virtual File Server (VFS) worker thread pool* for more information.
- As ACLs are dynamic and typically quite complex in these contexts, it is important to fine tune the time spent and/or search results on which ACLs are checked (see *ACLs and permission checks*). As this MAY impact search user experience, you CAN consider preloading ACL

permission check results cache with typical searches/and per every user group by running automated scripts on a scheduled basis or upon content updates.

Sample JVM configuration

The following is a sample JVM configuration for batch processing via the Foundation API, and assumes dedicated 8GB RAM machines.

Note: This is an example and SHOULD be tuned on the specific CPU speed and number of cores (this configuration assumes at least two CPU/cores per node). In addition, it MUST be reviewed with Alfresco Support or Solutions Engineering for further optimization.

A JVM command line with GC logs activated, JMX remote management and temp files on tmpfs could look like:

```
-server –Xms6G –Xmx6G -XX:PermSize=192m
-XX:MaxPermSize=192m -XX:NewRatio=2 -XX:+UseParNewGC
-XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled

-XX:CMSInitiatingOccupancyFraction=80
-Dsun.rmi.dgc.client.gcInterval=3600000

-Dsun.rmi.dgc.server.gcInterval=3600000
-Dcom.sun.management.jmxremote=true

-Djava.io.tmpdir=/dev/shm
```