Alfresco Repository

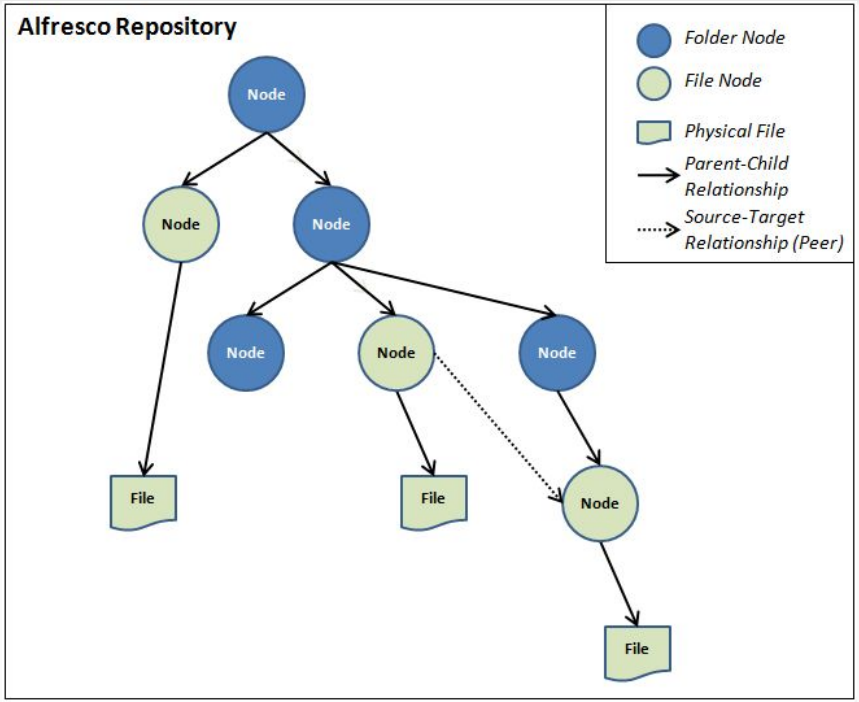
Logical Internal Structure - PART 2

Key Concepts

- **Content files** are stored on the **disk** / file system
- **Metadata** is information about the content files
- **Metadata** for content files are stored in the **database**
- **Indexes** are stored on the **disk** / file system
- Sometimes we talk about a **space**, it **is** the same as a **folder**
- The **top folder** in Alfresco is referred to as **Company Home**
- **Folders and content files** in Alfresco are stored in a repository and called **nodes**

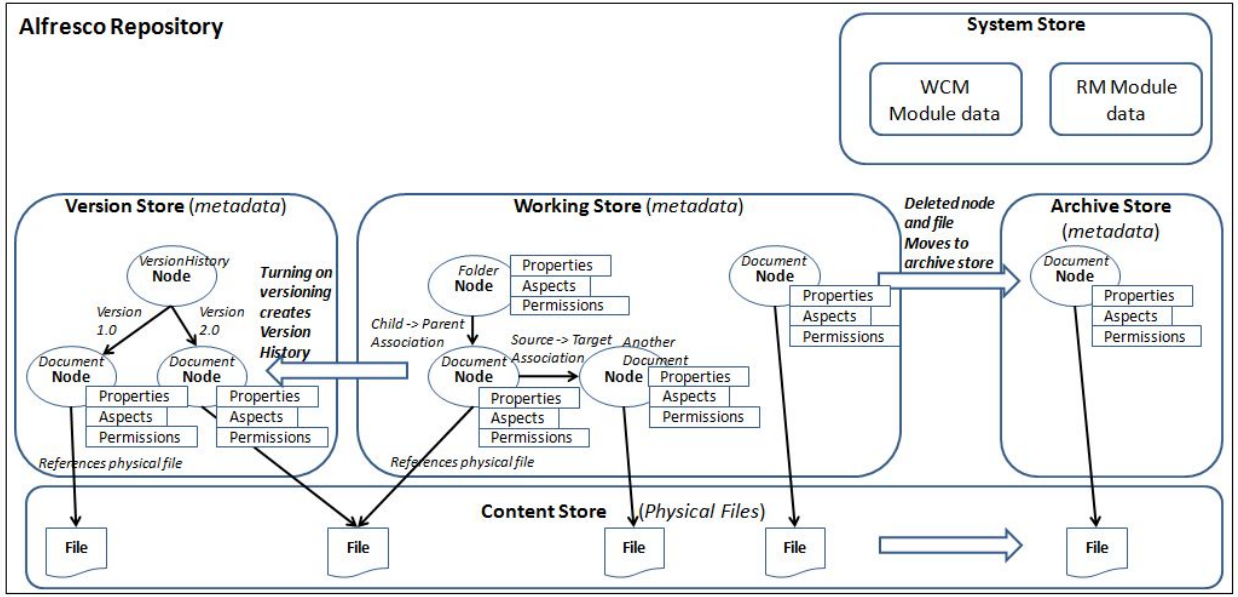
Logical Structure

- Everything is a node
- Some nodes can contain other nodes
- Some nodes can be associated with other nodes
- Root node at the top
- All nodes live in a store



Stores

- All nodes live in a store, a node is of a type, has properties, and can have aspects applied
- Access permissions are set per node



Working Store (Live content)

- The **Working Store** (`workspace://SpacesStore`) is the main store used in day to day work
  - Metadata is stored in the database
  - Files are located in the `alfresco\alf_data\contentstore` directory
- Is indexed in its own core
  - Files are located in the `alfresco\alf_data\solr4\index\workspace` directory
  - Solr6: `alfresco\solr6\solrhome\alfresco`

Archive Store (Soft Deleted content)

- When something is deleted it ends up in the **Archive Store** (`archive://SpacesStore`)
  - Metadata is stored in the database
  - Files are still located in the alfresco\alf_data\contentstore directory
- When files are deleted via **My Profile | Trashcan,** then they are later (14 days) moved, by the Content Store Cleaner job, to the `alfresco\alf_data\contentstore.deleted` directory
  - They stay in the contentstore.deleted directory forever
- Is indexed in its own core
  - Files are located in the `alfresco\alf_data\solr4\index\archive` directory
  - Solr6: `alfresco\solr6\solrhome\archive`

Version Store

- When versioning is applied to a node a version history is created in the **Version Store** (`workspace://lightWeightVersionStore`)
  - Metadata is stored in the database
  - Files are located in the alfresco\alf_data\contentstore directory
  - Versioning is not applicable to folder/space nodes
- Is indexed in Solr 6 and you can then search in the version history
  - Only via API, no UI

System Store

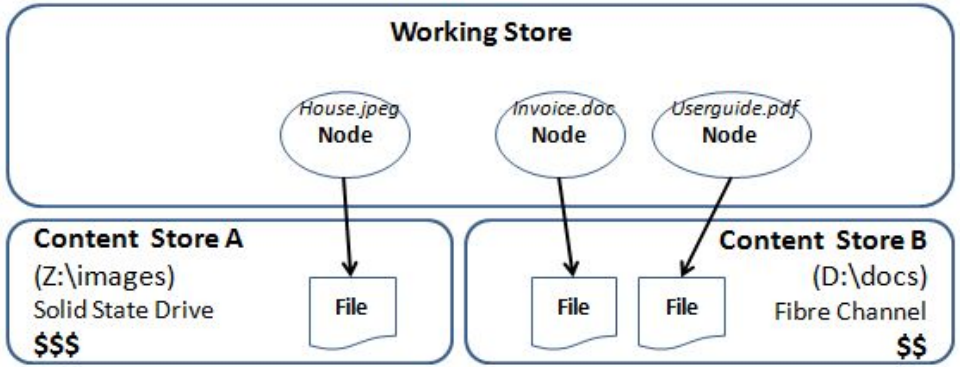
- The System Store is used to save information about installed AMP/JAR modules
- Not indexed

Content Store Implementation

- Content store files are located on the disk, why is that?
- Why are they not stored in the DB as BLOBs?
- There are several reasons for that:
  - Random access to files to support CIFS
  - Real-time streaming for direct streaming of files to browser
  - Standard database access would be difficult when using BLOBs
  - Faster access

Content Store Selectors

- Storing content files on different types of disks can be achieved by defining **content store policies**
- We might use a very fast tier 1 Solid-State Drives (SSD) for our most important content files, and based on business policies that we control, gradually move the data to cheaper tier 2 drives such as Fiber Channel (FC) drives or Serial ATA drives as it becomes less important
- In this way, we can manage the storage of content more cost effectively.



Store Reference

- A store is accessed (referred to) via its store reference, for example `workspace://SpacesStore`
- The first part (i.e. `workspace`) is the protocol (what store you are interested in) and the second part is the type of store (i.e. `SpacesStore`)

Node Information

- A node usually represents a folder or a file
- A store contains a top level root node (`sys:aspect_root`)
- The root node can have one or more child nodes
- Each node has the following metadata:
  - **Type**: a node is of one type
  - **Aspects** (secondary types): a node can have many aspects applied
  - **Properties**: both types and aspects define properties, one of the properties points to the physical file in the content store
  - **Permissions**: access control settings for the node
  - **Associations**: relationships to other nodes (peer or child)

Node Reference

- A node is uniquely identified in the repository via a **node reference**
- A node reference points to a store and a node in that store
- A node reference has the following format:
  - `workspace://SpacesStore/986570b5-4a1b-11dd-823c-f5095e006c11`
  - The first part is the store reference
  - And the second part is a Universally Unique Identifier (UUID) in that store
- Node references are used a lot in the *Foundation Services Java API*

Node Properties

- Properties contain all information about a node, including where the physical content is stored (if not a folder, tag, category etc), and is called **metadata**
- `The cm:content` property points to the physical content
- Properties are either contained in a type or in an aspect
- When a node is created some properties are automatically set by the system and cannot easily be changed, they are called **audit properties** (`cm:auditable`):
  - Created Date, Creator
  - Modified Date, Modifier
  - Accessed

Metadata/Property Extractors

- Some node properties are set automatically when content is uploaded to the repo
- This is handled via so called **metadata extractors**
- What properties that are set depend on the content's MIME type
- Out-of-the-box you have metadata extractors for Office document types, PDF, Email, HTML, DWG, JPG etc
- Each metadata extractor implementation have a mapping between the properties it can extract from the content file and what properties should be set as node metadata

Node Associations

- There are two types of associations:
  - **Parent -> Child** associations (e.g. folder -> content file)
    - Cascading delete
  - **Peer -> Peer** (e.g. Article -> Image)
    - Deletes does not affect related nodes
    - Referred to as source -> target

QName

- Properties live in a **namespace**
- A property called `description`, it's so called local name, can be part of many namespaces
- To uniquely identify what `description` property we are talking about a fully qualified name, or a **QName**, is used
- A QName has the following format:
  - `{http://www.alfresco.org/model/content/1.0}description`
- The first part in curly braces is the namespace identifier, which also has a prefix associated with it, such as `cm`
- The second part is the local name of the property (i.e. `description`)

QName continued

- A QName is actually used for everything in the repository – such as:
  - **Types**: cm:folder, cm:content
  - **Aspects**: cm:emailed, cm:versioned
  - **Properties**: cm:name, cm:titled, cm:description
  - **Associations**: cm:contains
  - **Constraints**: cm:filename
  - And so on...

Permissions

- Permissions are set up per node
- A node can inherit permissions from a parent node
- A Role/Group based access control is used
- But node permissions can also be set for an individual user
- Groups and Users can be synchronized with an external directory

Note. everyone has read access to the repository in a new installation!

Permissions continued

- Some groups are created automatically during installation:
  - `EVERYONE` – all users in the system
  - `ALFRESCO_ADMINISTRATORS` – administrators with full system access
  - `ALFRESCO_MODEL_ADMINISTRATORS` - can manage content models
  - `ALFRESCO_SEARCH_ADMINISTRATORS` - can manage search facets
  - `SITE_ADMINISTRATORS` - Can manage sites
  - `E-MAIL_CONTRIBUTORS` – users that can send email with content into Alfresco

Permissions continued

- Permission roles are defined here: `tomcat/webapps/alfresco/WEB-INF/classes/alfresco/model/permissionDefinitions.xml`

Permissions continued

- Permission settings involve three entities:

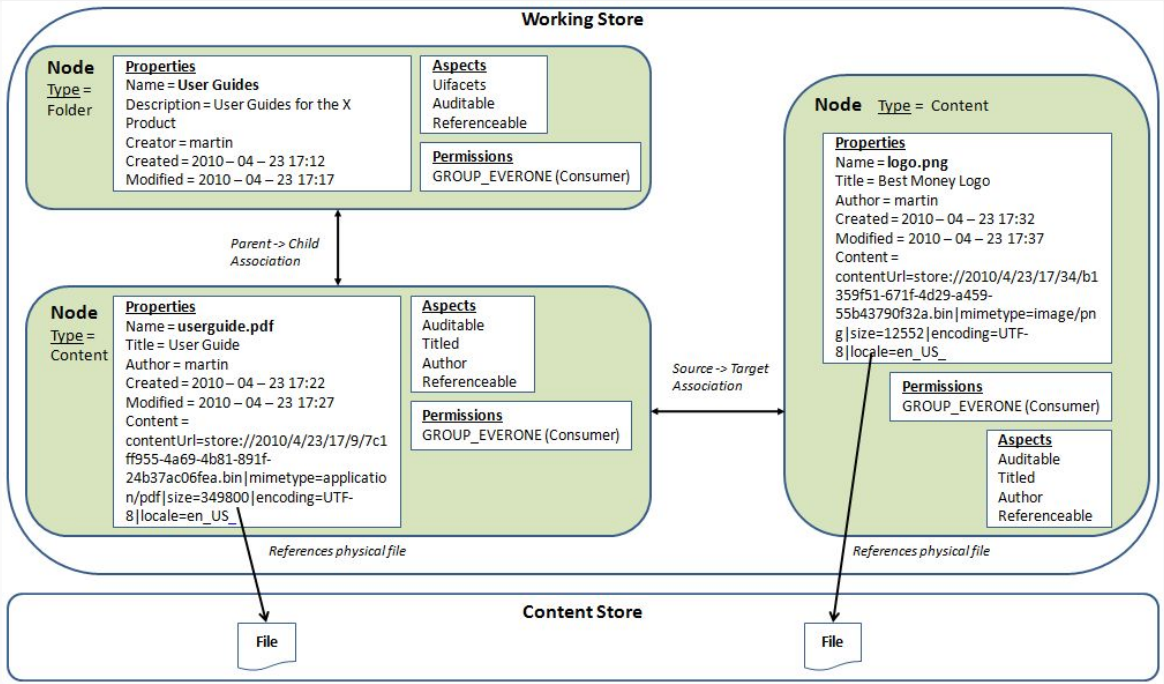
- Out-of-the-box roles:
  - Consumer, Contributor, Editor, Collaborator, Coordinator
- In the repository groups are prefixed with `GROUP_` and roles with `ROLE_`

Permissions continued: Owner

- A special authority is called **owner**
- Whoever creates a node in the repository is called the owner of the node
- Owner status overrides any other permission setting
- As owner you can do any operation on the node (coordinator/admin)
- Anyone with Coordinator or Admin status can take ownership of a node (`cm:ownable` is then applied)

Node Info

- A typical folder node with a child file node looks like this:



Node Forms

- **Forms** are used to layout the properties in the in Alfresco Share UI
- Form Definitions are associated with a type or an aspect
- When defining custom types and aspects corresponding form definitions are usually created as well

Node Lifecycle

- The node lifecycle can seem simple at first – create, update, and delete right?
- It's more to it:
- ***Files are never removed from disk!***