Compatible One: Livrable 29.3
Spécification du service documentaire multitenant basé sur CMIS
Version 1.0

Benjamin Jalon (Nuxeo), Stefane Fermigier (Nuxeo),
Florent Guillaume (Nuxeo), Thierry Delprat (Nuxeo)

Novembre 2011

Abstract

Ce document présente les spécifications du service de stockage et de caching.

Contents

1 Introduction

L'objectif du POC2 est de développer un service de gestion documentaire cloudifié implémentant le modèle et les API (REST) de CMIS.

Nous ne reviendrons pas sur le modèle CMIS, qui est documenté par un standard OASIS[1].

Nous nous concentrerons dans ce document sur le principal aspect spécifique à ce projet, à savoir la gestion du *multi-tenant*, autrement dit la capacité à gérer dans un cadre unique (une seule instance du serveur d'applications, ou plusieurs instances coordonnées par un système de gestion centralisé, ou encore une combinaison des deux) des domaines applicatifs spécifiques à chaque organisation utilisatrice, isolés les uns des autres afin de préserver la confidentialité des informations stockées mais aussi d'isoler les configurations spécifiques à chaque organisation.

Deux approches sont donc envisagées:

- L'approche multi-tenant au sens propre: introduire la notion de "domaine" au niveau applicatif, et faire en sorte que chaque domaine, vu des utilisateurs, se comporte comme une application indépendante.
- L'approche multi-instances: une instance différence de Nuxeo est déployé pour chaque organisation utilisatrice, en prenant soin néanmoins d'optimiser l'utilisation des resources système en partageant tout ce qui est pertinent entre les clients.

2 Multi-tenant: Approche applicative

2.1 Introduction

L'approche applicative repose sur la structure documentaire qui décrit chaque tenant. En effet un tenant correspond à un noeud sous la racine. Chacun de ces noeuds contiendra tous les documents rattachés à un tenant.

La difficulté de l'approche applicative est de fournir un système offrant une barrière hermétique entre tenants, que ce soit les documents, l'audit, les workflows, les actions, les types documentaires et les vocabulaires.

Dans un premier temps, nous allons décrire comment Nuxeo a abordé cette ségrégation entre ces différentes notions, puis dans un deuxième temps nous verrons comment en terme de code cela se traduit.

Toute la difficulté de cette approche est de tenir compte de la notion de tenant dans chacun des services Nuxeo.

[1] `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=cmis`

2.2 Description

2.2.1 Ségrégation des documents

Pour comprendre comment la ségrégation documentaire est abordée dans Nuxeo, il faut comprendre le système de sécurité.

Dans Nuxeo un document est décrit par un ensemble de méta-données et optionnellement de binaires. Chaque document porte sa visibilité par des ACLs (Access Control List).

Le système d'ACLs repose sur des liste d'ACE (Access Control Entry) ordonnés. Ces ACE sont décrit par une permission, un utilisateur ou un groupe et un booléen. Pour résoudre la permission d'un utilisateur sur un document, le système parcours cette liste d'ACE:

- si l'utilisateur et celui décrit dans l'ACE ou appartient au groupe décrit dans l'ACE,
- et que la permission à résoudre est celle décrite par l'ACE ou une sous-permission de celle décrite dans l'ACE
- Alors le si le booléen est positif (GRANT), l'utilisateur est considéré ayant la permission sinon il est considéré n'ayant pas la permission.

Enfin si aucun ACE correspond, l'utilisateur est considéré n'ayant pas cette permission sur le document.

Associé à la structure hiérarchique documentaire, Nuxeo a implémenté un système d'héritage de ces ACL d'un document à ses enfants. Nous utilisons donc ce système comme premier élément pour assurer l'étanchéité entre tenant. En effet l'élément racine du tenant porte un droit restrictif pour les utilisateurs n'appartenant pas au tenant. Les enfants de cet élément racine qui hériteront de ce droit seront ainsi également invisible des utilisateurs des autres tenants. Pour faciliter l'apposition de ce droit restrictif, il faut que les utilisateurs appartiennent à un même groupe Nuxeo.

Il faut également comprendre que dans Nuxeo, les utilisateurs et les groupes sont définis à plat. Or un utilisateur peut s'il le souhaite ajouter une ACE librement. Pour toujours assurer l'étanchéité du tenant, il faut qu'il ne puisse ajouter un ACE portant que des utilisateurs du tenant auquel il appartient, sans quoi il pourrait donner un droit positif à un utilisateur n'appartenant pas à son propre tenant. Nous avons donc dû ajouter un système de filtrage pour n'afficher que les utilisateur d'un tenant. La ségrégation des utilisateur est groupes est abordée plus loin.

2.2.2 Ségrégation typologique, recherches, thème (Configuration Locale)

Une grande partie de la différenciation dans Nuxeo pour implémenter la notion multi-tenant est produite par la notion de Configuration Locale.

Une configuration locale est un service qui permet de rajouter dynamiquement une structure de méta-données à un noeud du repository documentaire. Avec ce nouveau service, il est possible de rajouter dynamiquement, une méta-donnée de type chaine de caractères, ou des structures beaucoup plus complexes.

La ségrégation typologique correspond au fait de laisser pouvoir utiliser un ensemble de types documentaires spécifiques à chacun des tenants tout en gardant un ensemble de types documentaires communs.

Par défaut dans Nuxeo, les types pouvant être créés dans un conteneur sont décrits dans la définition même de ce conteneur (au niveau UI). Une configuration locale a été ajoutée pour décrire les types documentaires qu'un noeud acceptera ainsi que ces enfants.

Ainsi ajouter un type documentaire spécifique à un tenant revient tout simplement à le créer - comme un type documentaire standard - mais en ajoutant ce nouveau type dans la configuration locale pour le tenant qui est intéressé par ce nouveau type.

L'interface de sélection de choix des types documentaires filtrés est totalement gérée par l'interface dans l'onglet "Manage/Local Configuration", dans la section "Document type configuration".

Que ce soit pour la configuration de la liste des filtres proposés dans le "Faceted Search", le formulaire de recherche avancé ou encore le thème utilisé pour chaque tenant, le mode de fonctionnement décrit au-dessus est le même. En effet il suffit de créer sa liste de filtres spécifique, son formulaire de recherche avancée spécifique, son thème spécifique puis de configurer le tenant dans l'onglet "Manage/Local Configuration", respectivement dans la section "Faceted Search Configuration", dans la section "Search Configuration", dans la section "Theme Configuration".

2.3 Ségrégation des vocabulaires (Directory)

Nuxeo propose une service centralisé pour tous les systèmes de stockage à plat des données externes au repository documentaire qui s'appelle Directory. Ce service est fort utile car permet de créer facilement des vocabulaires fermés (utilisé pour des champs à possibilités de valeurs restreintes) ou de décrire le stockage externe des utilisateurs et groupes. Plusieurs implémentation de ce service existe: pour un stockage dans une base de données de type SQL, dans une base de données LDAP ou encore une agrégation de plusieurs type de stockage (ce que nous appelons le multi- directory).

Ce service propose :

- une interface unifiée de définition de structure
- ainsi qu'une interface unifiée de requêtage

La ségrégation des vocabulaires a été traité au niveau de ce service. En effet, pour chaque tenant et chaque vocabulaire, un directory est créé. La différencenciation de ces directory est porté par leur nom. Le nom est formé par le nom du directory et un suffixe.

Le suffixe utilisé pour un tenant donné et toujours le même, quelque soit le directory. La définition de ce suffixe est porté par une configuration locale sur le noeud racine du tenant (la notion de configuration locale est décrite dans la section précédente).

Nous avons également ajouté un "fallback": si le directory spécifique à un tenant pour un vocabulaire donné n'existait pas (le nom du vocabulaire avec le suffixe du tenant), alors le directory utilisé est celui sans suffixe.

2.3.1 Ségrégation des utilisateurs et groupes

Comme il a été dit dans la section Ségrégation de sdocuments, l'exposition des utilisateurs dans Nuxeo n'est pas structuré. Une API a été ajouté au niveau du service des gestion des utilisateurs afin de permettre de filtrer les utilisateurs et les groupes dans le contexte d'un tenant.

Dans le cadre d'une instance multi-tenant, chaque tenant a son propre directory pour les utilisateurs et la logique suivie est la même que celle décrite pour les vocabulaires. Le directory utilisé est le nom du directory utilisateur avec le suffixe spécifique du tenant.

Pour différentes raisons techniques, il n'est pas possible de faire la même chose pour les groupes. Il faut impérativement que tous les groupes soient tous dans un unique directory de type simple (pas multi-directory). Pour différencier à quel tenant appartient chacun des groupes, l'ID de chaque groupe est suffixé par le suffixe du tenant. L'utilisateur de Nuxeo et le développeur Nuxeo n'a pas à gérer ce suffixe, le service de gestion des utilisateurs gère automatiquement le suffixage.

Il est à noter que le requêtage du directory prend en compte toute ces problématiques de multi-tenant automatiquement.

Maintenant que la ségrégation entre les utilisateurs et les groupes est obtenur, il faut aborder la connexion d'un utilisateur. En effet, quand un utilisateur se connecte Nuxeo a aucun moyen de connaître à quel tenant il est sensé se connecter et donc à quel directory celui-ci doit se référer. Donc au moment de la connexion, il faut présenter un directory qui agrège tous les utilisateurs de tous les tenants, afin que le service d'authentification puisse résoudre l'utilisateur. Avec Nuxeo, il faut tout simplement créer un multi-directory qui agrège tous les directory utilisateurs - suffixé avec le suffixe des tenants - dans un directory qui porte le nom du directory utilisateur sans suffixe. Pour ce qui est des groupes étant donnée que ceux-ci sont dans un même directory, il n'y a pas cette problématique.

Remarque: Il est évident qu'il faut assurer de manière organisationnel l'unicité d'un login utilisateur entre tenant. En effet si un utilisateur d'un

tenant A a le même utilisateur d'un tenant B, un seul pourra se connecter. Il faut donc, par exemple ajouter un suffixe au niveau du login, comme `@tenanta` ou `@tenantb`.

2.4 Implémentation d'un prototype

Nous avons vu dans la section précédente de manière macroscopique la stratégie abordée par Nuxeo pour implémenter la notion de multi-tenant et pour assurer l'étanchéité entre les données entre chacun de ces tenants. Nous allons maintenant voir comment cela se traduit de manière explicite en terme de code.

2.4.1 Définition d'un tenant

Pour définir un tenant, il faut tout d'abord créer le type documentaire qui correspondra à la racine de ce tenant. La contribution correspondant à cette définition donne le XML suivant:

```
<doctype name="Tenant" extends="Domain">
<facet name="SuperSpace"/>
<facet name=''DirectoryLocalConfiguration''/>
<facet name=''SimpleConfiguration''/>
</doctype>
```

La facette "DirectoryLocalConfiguration" est nécessaire pour le filtrage des directory, c'est à dire des vocabulaires, des utilisateurs et des groupes. La facette "SimpleConfiguration" est utilisée pour filtrer les opérations et les listeners.

Ainsi la création d'une instance de tenant devra être traitée de manière programmatique de la façon suivante:

```
CoreSession session = ...
DocumentModel tenant = session.createDocumentModel(‘‘Tenant’’);
tenant.setPathInfo(‘‘/’’, ‘‘customer1’’);
tenant.setPropertyValue(‘‘dc:title’’, ‘‘Customer 1’’);

... all properties you want ...

ACP acp = doc.getACP();
ACL localACL = acp.getOrCreateACL();
localACL.add(new ACE("mytenant_reader", READ, true));
localACL.add(new ACE("mytenant_contributor", WRITE, true));
acp.addACL(localACL);
doc.setACP(acp, true);
session.createDocument(tenant);
session.save();
```

Ici est créé donc un tenant qui port le nom de “Customer 1”. Les droits d'écriture sont donnés au groupe “mytenant_contributor” et lecture au groupe “mytenant_reader”.

Ici nous exposons la solution programmatique, mais il est également possible de créer ce tenant au moyen du service Content Template qu'il est possible de trouver dans la documentation Nuxeo qui se trouve à l'adresse suivante: doc.nuxeo.com

2.4.2 Ségrégation des directories

Nous allons exposer ici comment offrir un vocabulaire spécifique pour chaque tenant. Nous avons vu que les vocabulaires sont portés par des directory et chaque tenant porte une configuration local “DirectoryLocalConfiguration” de type chaine de caractères qui ajouté en suffixe utilisé pour trouver le directory associé au tenant.

Prenons par exemple le directory “topic” qui stocke un plan de classement que nous voulons spécifique pour chaque tenant. Pour chaque tenant, il est nécessaire de créer un directory avec un suffixe qui doit lui être propre. Disons que nous avons 2 tenants qui utiliseront respectivement le suffixe “tenanta” et le suffixe “tenantb”.

La définition des 2 directories devra être quelque chose comme suit:

```
<extension target="org.nuxeo.ecm.directory.sql.SQLDirectoryFactory"
 point="directories">
   <directory name="topic_tenanta">
   <schema>xvocabulary</schema>
   <dataFile>directories/topic_tenanta.csv</dataFile>
   ...
   </directory>
</extension>
```

Et voici le code pour attacher le suffixe “tenanta” au document qui sera associé à ce tenant:

```
DocumentModel doc = session.getDocument(new PathRef("/tenant-a"));
doc.setPropertyValue("dirconf:suffix", "local");
doc = session.saveDocument(doc);
session.save();
```

Il est à noté que cette configuration locale est hérité pour tous les documents enfants de ce document. Le vocabulaire utilisé pour les topics pour tous les document enfant et le document lui-même “/tenant-a” utiliseront le vocabulaire portant le nom topic_tenanta.

Pour récupérer le JSON de ce directory pour une utilisation externe, il suffit de créer l'opération suivante:

```
// docIntoTenantA is a document located into the Tenant A
OperationContext ctx = new OperationContext(session);
ctx.setInput(docIntoTenantA);


OperationChain chain = new OperationChain("myChain");
chain.add(FetchContextDocument.ID);
Map<String, Object> params = new HashMap<String, Object>();
params.put("directoryName", "country");
params.put("lang", "fr");
params.put("translateLabels", "true");
OperationParameters oparams = new OperationParameters(
GetDirectoryEntries.ID, params);
chain.add(oparams);

StringBlob result = (StringBlob) service.run(ctx, chain);
// result contains an export of the directory country_domaina with the label
// translated in french
```

L'interface des directory cache l'implémentation de la sélection du directory avec suffixe. Il n'est donc pas utile de récupérer la localConfiguration. Il suffit tout simplement d'effectuer ceci:

```
DirectoryService ds = Framework.getService(DirectoryService.class);
Session session = ds.openSession("topic", currentDocument);
// this will try to find the local configuration
// "Directory" of currentDocument.
// this will use the value of this configuration
// to fetch the "typeofcontract_valueOfTheLocalConfiguration''
```

Nuxeo se chargera de récupérer la configuration locale en utilisant le document courant pour obtenir le suffixe. Nuxeo vérifiera si le directory `topic` avec le suffixe trouvé existe sinon le directory sans suffixe est récupérer.

2.4.3 Ségrégation des utilisateurs et groupes

Comme nous l'avons vu dans le chapitre précédent, les utilisateurs et groupes d'un tenant sont chacun stockés dans un directory qui lui est propre. Nuxeo agrège ces directory seulement au moment de la connexion de l'utilisateur.

Ainsi, il faut donc créer :

- pour chaque tenant un directory (pour les utilisateurs)
- un directory pour l'agrégation des directory utilisateurs de tous les tenant

- et un directory pour l'agrégation des directory groupes.

Cela donne pour le directory utilisateur d'un tenant la configuration suivante:

```
<extension  target="org.nuxeo.ecm.directory.ldap.LDAPDirectoryFactory"
    point="directories">
    <directory  name="userDirectory_tenanta">
      <server>default</server>
      <schema>user</schema>
      <idField>username</idField>
      <passwordField>password</passwordField>
      <searchBaseDn>ou=people,dc=tenanta,dc=com</searchBaseDn>
      <searchClass>person</searchClass>
      <searchScope>onelevel</searchScope>
      etc....
      <references>
        <inverseReference  field="groups"  directory="groupDirectory"
          dualReferenceField="members"  />
      </references>
    </directory>
</extension>
```

Nous voyons le nom du directory contient le suffixe qui sera utilisé pour l'identifier. Ainsi chaque tenant définira son propre directory pour les utilisateurs.

Il est important de noter que l'intersection entre directory utilisateurs deux à deux doit être vide.

Par contre vous aurez remarqué que le directory du groupe "groupDirecotry" qui est défini dans la référence inverse n'a pas de suffixe. En effet tous les groupes se trouvent dans le même directory - pour des raisons techniques. Par contre le nommage des directory seront contrôlé pour assurer une unicité par tenant et avoir une possibilité de filtrage efficace.

Le service de gestion des utilisateurs s'occupera de maintenir le nom des groupes cohérent.

Mais au moment de l'authentification, il faut que la base contenant la totalité des utilisateurs référencés soit utilisés, il faut donc fusionner ces bases. Nuxeo propose cette notion à travers les multi-directory. Cela donne la définition suivante:

```
<extension
    target="org.nuxeo.ecm.directory.multi.MultiDirectoryFactory"
    point="directories">
  <directory name="userDirectory">
    <schema>user</schema>
```

```
    <idField>username</idField>
    <passwordField>password</passwordField>
    <source name="Customers" creation="true">
      <subDirectory name="userDirectory_tenanta">
      </subDirectory>
      <subDirectory name="userDirectory_tenantb">
      </subDirectory>
        etc...
    </source>
  </directory>
</extension>
```

En terme programmatique, le service s'utilise comme habituellement.

Pour la recherche d'utilisateur:

```
DocumentModelList users;
users = userManager.searchUsersWithContext("%%", tenantADocumentModel);
// This query returns all users from tenant A,
// you can specify instead of ‘‘%%’’ a query of your choice
```

Pour créer un utilisateur:

```
UserManager  um =  Framework.getService(UserManager.class);
// create the user as usual with their values.
DocumentModel newUser = um.getBareGroupModel();
newUser.setPropertyValue("username", "bjalon");
// Set values as usual to create a user
// Nuxeo will manage the multi-tenant infrastructure
// the tenantADocumentModel value can be the Tenant root documentModel
// or any descendant of the Tenant.Nuxeo will add into groupname
newUser = um.createUserWithContext(newUser, tenantADocumentModel);
users = userManager.searchUsersWithContext("%%", tenantBDocumentModel);
// users will not contains bjalon as I specify the tenant B and bjalon have
// been created into the Tenant A context.
```

2.4.4 Listener spécifique à un tenant

La création d'un listener attaché à un unique tenant se fait en 3 étapes.

Premièrement, il faut attacher une configuration locale simple qui permettra d'identifier les listener à traiter et sur quel évènement:

```
DocumentModel tenantA = ...
DocumentModel tenantB = ...
SimpleConfiguration simpleConfiguration =
  localConfigurationService.getConfiguration(
```

```
    SimpleConfiguration.class, ‘‘SimpleConfiguration’’, tenantA);
 simpleConfiguration.put(‘‘documentCreated’’, ‘‘action1,action2’’);
 simpleConfiguration.save(session);


 simpleConfiguration = localConfigurationService.getConfiguration(
 SimpleConfiguration.class, ‘‘SimpleConfiguration’’, tenantB);
 simpleConfiguration.put(‘‘documentCreated’’, ‘‘action2’’);
 simpleConfiguration.put(‘‘documentModified’’, ‘‘action3’’);
 simpleConfiguration.save(session);
```

ici nous déclarons 3 actions:

- action1 qui sera exécutée pour le tenantA pour des évènements de création
- action2 qui sera exécutée pour le tenantA et le tenantB pour des évènements de création
- action3 qui sera exécutée pour le tenantB pour des évènements avant création

Une fois que cette première étape est traitée, il faut créer un listener qui s'occupe de dispatacher chaque évènement vers la bonne action à effectuer. Cela donne la déclaration du listener suivant:

```
<extension target="org.nuxeo.ecm.core.event.EventServiceComponent"
    point="listener">
  <listener name="eventTenantDispatcher" async="false" postCommit="false"
     class="org.nuxeo.ecm.platform.multitenant.EventDispatcher">
  </listener>
```

Et le code du listener:

```
public class EventDispatcher implements EventListener {
 public void handleEvent(Event event) throws ClientException {
 if (! event.getContext() instanceof DocumentEventContext) {
   return;
 }
 DocumentEventContext docCtx = (DocumentEventContext) event.getContext();
 DocumentModel doc = docCtx.getSourceDocument();
 String eventId = event.getName();
 SimpleConfiguration simpleConfiguration =
   localConfigurationService.getConfiguration(
     SimpleConfiguration.class, ‘‘SimpleConfiguration’’, doc);
```

```
 List<String> actions = simpleConfiguration.get(eventId)).toto.split(",");
 for (int i=0; i < actions.length; i++) {
   OperationContext ctx = new OperationContext(session);
   ctx.setInput(doc);
   DocumentModel out = (DocumentModel) service.run(ctx, actions[i]);
 }
}
```

2.5 Conclusion

Cette approche est efficace car facile a mettre en oeuvre pour les administrateurs de Nuxeo, mais elle reste limitée. En effet chacun des tenants partage une même base de données et tous les services n'implémentent pas tous la notion de multi-tenant.

Pour une utilisation avec ségrégation forte, cette méthode décrite dans ce chapitre n'est pas appropriée. En effet, il n'est pas possible de restaurer simplement des données d'un seul tenant, certains service ne sont pas paramétrables. C'est pourquoi Nuxeo offre une solution plus complète pour la gestion multi-tenant. Cette solution a pour base le multi-tenant basé sur du multi-instance de Nuxeo.

3 Approche multi-instances

TODO

4 Conclusion