Compatible One: Livrable 4.1.1
Expression des besoins et spécifications techniques pour la GED en ligne
Version 0.9

Stefane Fermigier, Thierry Delprat, Florent Guillaume,
Mathieu Guillaume, Olivier Grisel, Bogdan Stefanescu

April 22, 2011

Abstract

Ce document présente un scénario de migration de la plateforme d'ECM open source Nuxeo et des solutions associées vers une infrastructures cloud générique, incarnée par les services du IaaS (SP1) et du PaaS (SP2) de Compatible One.

Nous commençons par une présentation rapide de l'existant. Les lecçons tirées de cette première expérience amènent à proposer quelques pistes d'améliorations.

Ensuite, plusieurs étapes sont définies qui correspondent à un enchaînement logique et une prise de risque tecnologique croissante. Pour chacune, sont précisés le contexte actuel, les enjeux business et les défis techniques associés. Chaque étape doit aboutir à un démonstrateur qui permettra de juger de la pertinence de l'approche et de son potentiel d'industrialisation.

Enfin, les prerequis qu'implique le développement de ces démonstrateurs pour les autres source projets du projet Compatible One (notamment SP1 et SP2) sont développés, afin de permettre aux membres du consortium qui travaillent sur ces sous-projets d'orienter leurs travaux dans le sens souhaité.

Compte-tenu de la nature itérative du projet, ce document sera mis à jour de manière itérative, en fonction des leçons acquires et de l'avancement des autres SP.

Contents

1 Utilisation actuelle du cloud par Nuxeo

Nuxeo utilise déjà, pour ses besoins propres (sites et services web, instances d'essai ou de préproduction) ou pour héberger des applications de clients des serveurs virtualisés Amazon EC2.

1.1 Besoins

Pour que les applications Nuxeo puissent fonctionner de manière satisfaisante, il nous faut au minimum des VM capables de faire tourner l'application elle-même (application java, de préférence sous Linux, avec ses dépendances externes), ainsi qu'une base de données, un stockage persistent, et une connectivité IP.

1.2 Implémentation actuelle

Nuxeo utilise pour l'instant des instances EC2, basées sur des AMIs Ubuntu ou Debian relativement standard (celles d'Alestic).

Nous installons alors manuellement le package `.deb` Nuxeo en adaptant la configuration de façon à utiliser comme stockage un volume EBS (pour le stockage filesystem et pour la base de données PostgreSQL), avec un reverse-proxy Apache en frontal.

Une adresse IP fixe est allouée à chaque instance.

La plupart du temps, un deuxième volume EBS est utilisé pour les backups (on fait alors un snapshot de ce volume après chaque backup).

Optionnellement, le frontal HTTP et/ou la base de données peuvent être sur des instances EC2 séparées.

1.3 Faiblesses avérées de l'implémentation actuelle

Nous avons été confrontés à plusieurs faiblesses lors de l'implémentation. Certaines d'entre elles sont contournables, mais il serait mieux d'avoir ces fonctionnalités gerées directement par l'infrastructure de cloud.

- Impossibilité d'avoir plus d'une IP publique par VM (ce qui est notamment problématique pour la gestion de domaines multiples en HTTPS).
- Les machines virtuelles sont liées à un kernel et ramdisk qu'on ne peut pas modifier après la création de l'instance, ce qui limite fortement les possibilités de mises à jour.
- Pas de partage des EBS, donc pas de filesystem partagé pour des configurations cluster.
- Pas de garantie d'écriture sur le volume EBS avant un snapshot. Cela nous oblige à faire un unmount du volume de backup avant chaque snapshot.

- Pas de redimensionnement dynamique des instances (CPU, mémoire) bien que certains systèmes d'exploitation le supportent, ni de la taille des volumes EBS, la montée en capacité des applications ne peut donc se faire que de façon horizontale.

- Pas de gestion d'infrastructures multi-machines (jusqu'à AWS CloudFormation le 25/02/2011), les architectures multi-VMs ne pouvaient donc pas être gérées de façon intégrée.

- Au niveau administration et comptabilité, pas de possibilité de segmenter le compte AWS ni de déléguer certaines tâches administratives.

1.4 Fonctionnalités supplémentaires utiles pour l'avenir dans ce cadre

Suite à l'utilisation quotidienne de l'infrastructure actuelle, les éléments d'évolutions suivants ont été identifiés par l'équipe d'exploitation comme pouvant s'avérer utile, dans le cadre de l'arcitecture existante:

- Support d'architectures et protocoles réseau plus avancées (virtual switches, IPv6, IPSec...)

- Bases de données PostgreSQL en PaaS (type RDS)

- Diverses autres fonctionnalités en PaaS: reverse-proxy HTTP/HTTPS, relai SMTP vers un hôte ou une liste d'hôtes données (puisque n'impor-te quel cloud public va de toutes façons se retrouver dans les blacklists anti-spam), SMSC, AMQP...

- Conversion des VMs entre plusieurs architectures de virtualisation, pour pouvoir migrer simplement de l'une à l'autre, passer d'un fournisseur à un autre, ou d'un cloud privé à un cloud public (et inversement).

2 Expression des besoins

Cette section décrit les principaux enjeux de la cloudification d'une plateforme d'ECM comme Nuxeo et de son écosystème d'applications métiers, horizontales et verticales.

Les trois objectifs primaires fixés initialement pour les démonstrateurs du SP4.1 sont:

- Meilleure intégration dans un écosystème cloudifié : l'objectif ici est d'industrialiser le déploiement d'applications Nuxeo "stock" dans le Cloud, en se basant sur des outils et des services standard ou génériques, plutôt que sur des outils ad-hoc. Un tel développement peut se faire

par étapes, en partant des couches inférieures, et en remplaçant les services existants du Cloudware et de la plate-forme Nuxeo par des services génériques de l'IaaS et du PaaS Compatible One.

- "Nuxeo Document Storage as a Service", aka "Elastic CMIS" : si, comme nous le prévoyons, le standard CMIS connait une adoption rapide à l'horizon 2011–2012, nous pensons qu'il pourrait y avoir un intérêt pour du stockage CMIS "on demand" et infiniment élastique. Le contenu ainsi stocké et manipulé pourrait être consommé via des aplications clientes (MS-Office, applications métiers), ou en ligne via des mashups.

- Nuxeo App Store for CEVAs : la troisième étape cible serait de permettre à des éditeurs d'applications verticales orientées contenu de construire des applications par dessus le CMIS élastique, soit en tant que process indépendants, soit en tant qu'applications directement intégrées dans la base documentaire (mais en multi-tenant). Outre la réalisation des applications, le principal challenge est la mise sur le marché de ces applications, avec la vision d'un "App Store" dédié à la plate-forme.

2.1 Cloudification des applications Nuxeo

2.1.1 Contexte

Nuxeo a développé un certain nombre d'applications, horizontales et verticales, comme Nuxeo DM (Document Management), Nuxeo DAM (Digital Asset Management) et Nuxeo Courrier (Gestion de courrier et des cas métier), dédiées à la gestion documentaire.

Ces applications sont actuellement packagées sous la forme d'un serveur d'applications Java EE (ex: Tomcat, JBoss, Jetty) enrichi par un environnement "OSGi-like" et d'un ensemble de composants qui implémentent les services Nuxeo. Un ensemble de fichiers XML permet de décrire l'assemblage de ces composants et leur paramêtrage. Cet environnement d'exécution Java doit être complété par un certain nombre de services: SGBR (ex: PostgreSQL), moteur de rendu de fichier bureautique (ex: OpenOffice.org en mode "headless"), outil de tranformation d'images (ex: ImageMagik), outil de tranformation video (ex: ffmpeg), etc.

Dans un contexte cloud, l'ensemble de ces serveurs et outils sont en général packagés dans un ou des VM, afin de garantir une bonne isolation entre les applications des différents utilisateurs.

Ce mode de déploiement présente cependant plusieurs défis:

- *scaling down*: la taille minimale d'une VM contenant une instance de Nuxeo, pour avoir des performances décentes pour une utilisation non intensive par quelques dizaines d'utilisateurs est de 2 Go. Proposer

des applications pour les très petits utilisateurs (TPE) ou des versions d'essais n'est économiquement rentable que si on passe par une approche où plusieurs clients sont hébergés dans la même VM, et même dans la même JVM (i.e. le même serveur d'appli), ce qui oblige à passer par une approche multi-tenants.

- *scaling up*: un utilisateur ayant des gros besoins devra utiliser des VM plus importantes ($>$ 10 Go de RAM, CPU multi-cores puissants), jusqu'à un point où seule une installation multi-serveurs est capable de répondre aux besoins. Dans ce cas, plusieurs VM doivent être provisionnées, selon plusieurs templates (base de données, repository, transformation, frontaux), en fonction du besoin du client. Ce "capacity planning" manuel est consommateur de resources humaines, et ne permet pas une élasticité optimale des solutions déployées: il doit être possible de faire mieux en utiliser des technologies cloud dédiées!

- *tolérance aux pannes*: un déploiement mono-serveur est sensible aux pannes matérielles ou logicielles, il convient d'introduire de la redondance et des mécanismes de fail-over si on souhaite garantir aux utilisateurs les SLAs qu'ils attendent, et non du "best effort".

Plus généralement, les besoins d'un client peuvent évoluer et c'est une des propositions de valeur du cloud computing de permettre aux utilisateurs d'adapter le dimensionnement de leur application en fonction de l'évolution de leurs besoins. L'objectif pour Nuxeo est de se donner les moyens techniques de répondre à ces attentes, qui seront de plus en plus pressantes à mesure que l'approche cloud computing va se démocratiser.

2.1.2 Objectifs

Disposer de prototypes de la plateforme et des applications Nuxeo sous une forme exploitant une partie plus ou moins importante des fonctionnalités de Compatible One (IaaS seul, IaaS + PaaS, PaaS seul), de façon à les benchmarker et pouvoir évaluer la pertinence d'en dériver ultérieurement des produits industrialisés.

2.1.3 Challenges

- Démontrer la pertinence des outils du SP1 (IaaS) pour le provisionning et la gestion de machines virtuelle, de stockage virtualisés, et le déploiement automatisé d'applications.

- Démontrer la pertinence des services la plateforme SP2 (PaaS), en particulier:

  - Le runtime OSGi.

  - Le stockage (des métadonnées et des BLOBs).
  - Le service de gestion documentaire qui sera lui-même une abstraction des services Nuxeo bas niveau (Nuxeo Core).
  - Le monitoring et les services liés à l'auto-scalabilité.

- Aussi bien au niveau du IaaS que du PaaS, un challenge important est la gestion de l'élasticité du cloud sous-jacent, gestion que l'on souhaite rendue autant que possible transverse aux points d'attention des développeurs d'applications.

2.1.4 Principales tâches

Nous listons ci-après les principales tâches que nous pensons nécessaire d'accomplir pour démontrer la pertinence du cloud Compatible One dans le cadre ainsi fixé:

- Utiliser les outils de SP1.2 (Administration de machines virtuelles) pour provisionner et administrer des machines virtuelles contenant des instances des applications Nuxeo.

- Utiliser les outils de SP1.3 (Système de stockage distribué à haute disponibilite) et/ou SP2.3 (Service de stockage et de caching) pour stocker et cacher les informations de bas niveau nécessaires aux fonctionnement du repository Nuxeo, mais aussi pour s'assurer d'un niveau correct (garanti par SLA) de sécurité des données des utilisateurs.

- Utiliser le SP 2.4 (Runtime OSGI) pour découper les applications Nuxeo (qui sont déjà modularisées sous forme de bundles OSGi) en services indépendants.

- Faire évoluer, grâce au SP2.6 (API-gestion documentaire), les logiciels Nuxeo vers le multi-tenant.

- Utiliser les outils de SP2.1 (PaaS Scheduling, Calcul Distribué) pour les calculs intensifs comme par exemple le rendering (transformation des documents en PDF) ou le text-mining sémantique.

- Utiliser les outils du SP2.5 (Comptabilisation, inscription et facturation) afin de rentre la commande et le déploiement des applications simple pour les utilisateurs et pour proposer un mode de facturation "à la demande" qui soit à la fois attractif pour l'utilisateur mais également créateur de valeur pour Nuxeo.

- Utiliser les outils du SP3 pour s'assurer de la qualité de service, de la sécurité, etc.

2.1.5 Roadmap

Afin de procéder de manière itérative et de maximiser les opportunités de collaboration avec les autres SP, notamment SP1 et SP2, le plan est de réaliser la tâche SP4.1 par étapes, en commençant par les modifications les moins intrusives pour les logiciels Nuxeo existants, et les plus en phase avec l'état de l'art actuel du cloud computing (e.g. Amazon AWS).

Phase 1: Nuxeo sur le IaaS Compatible One Dans cette phase, les logiciels Nuxeo sont déployés dans des VM grâce aux API et aux services du SP1.

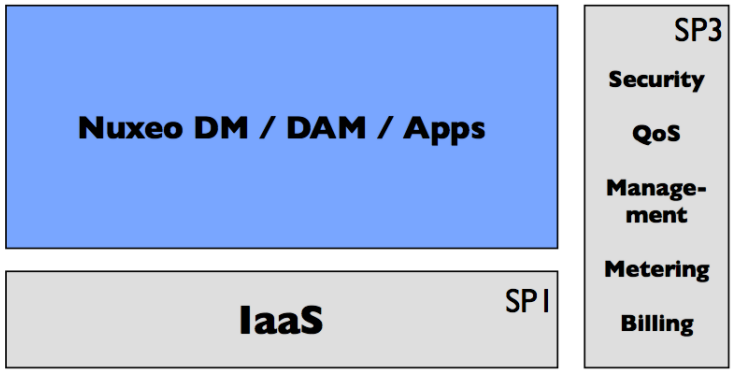

Figure 1: Nuxeo sur le IaaS

Par extension, il est aussi possible que l'on expérimente l'utilisation des services du PaaS (par exemple, le stockage des blobs et le stockage relationnel) sans remettre en cause cependant le modèle de déploiement.

Il s'agit d'une phase relativement peu risquée, car les services IaaS se présentent à l'heure actuelle sous une forme relativement stabilisée (e.g. Amazon EC2), il sera donc possible de conduire des expérimentations sur le cloud Amazon (ou Rackspace, Gandi, etc.) sans avoir besoin de déployer un cloud Compatible One pour les tests.

Phase 2: Nuxeo sur le PaaS de Compatible One Cette deuxième phase est plus intrusive pour la base de code Nuxeo, et représente également un challenge plus important pour la collaboration au sein du projet, du fait qu'elle suppose d'arriver à faire tourner les logiciels Nuxeo dans un "container" cloudifié, typiquement un environnement OSGi.

Cette phase sera également l'occasion d'approfondir l'utilisation des autres services PaaS, ainsi que des fonctionnalités d'auto-scalabilité du cloud Compatible One.

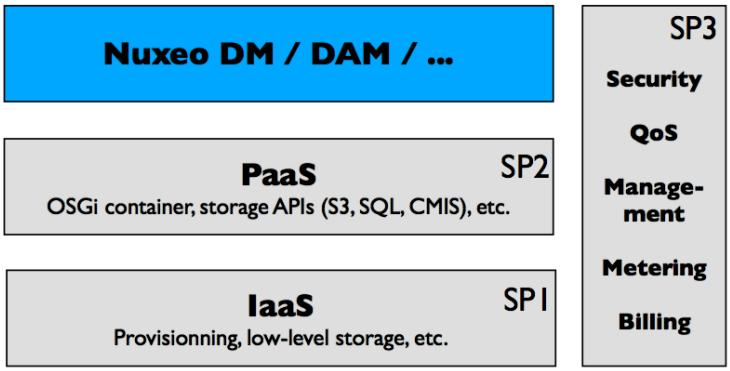

Figure 2: Nuxeo sur le PaaS

Une partie des expérimentations pourra se faire "en local", une partie nécessitera la mise à disposition d'une véritable instance du cloud Compatible One, notamment pour ce qui concerne l'auto-scalabilité.

2.1.6 Use cases et contraintes pour les autres SP

Nous listons ci-après les prérequis sur les autres SP induits par le use case 4.1:

- SP1.2: TODO tiry.
- SP1.3 et SP2.3: doivent fournir un système de stockage de binaires (BLOBs), élastique autant que possible, à défaut simple à provisionner et à rendre auto-scalable. Ce système peut être utilisé soit directement depuis les API systèmes de l'OS (i.e. en tant que filesystem), soit en tant que service (HTTP ou autre, de manière similaire à Amazon S3).
- SP2.4: doit fournir un environnement d'exécution de services OSGi scalable, managé, etc. Il doit pouvoir être possible de:
  - déployer simplement des applications à partir d'un "bundle repository" et de descripteurs de ces applications.
  - définir, avec ces descripteurs ou mieux grâce à des règles métiers, les schémas de déploiement de ces services sur les noeuds de la plateforme OSGi (i.e. les instances de JVM), en fonction des règles de SLA.
  - redéployer ou migrer très rapidement (sans downtime) les applications d'une configuration à une autre.

Ce service pourrait être comparable à ce que proposent Paremus Service Fabric (`http://www.paremus.com`) ou bien le PaaS d'Intalio (`http://www.intalio.com/osgi`).

- SP2.6: doit fournir les services "core" de Nuxeo sous une forme totalement multitenant (isolation des repositories, des utilisateurs et des customisations) dans un même environnement d'exécution OSG (SP2.4). Plusieurs approches sont d'ailleurs possibles, dépendant du niveau de "multitenancy" souhaité.

- SP2.5: doit permettre la comptabilisation des ressources IaaS et PaaS consommées par les applications Nuxeo afin d'envisager de proposer des schémas de facturation à la demande.

- SP2.1: permettre l'exécution de jobs asynchrone de type MapReduce et GridGain, afin notamment de proposer des services de transformation de contenu: vignettage de videos, redimensionnement et conversion en batch de photos, rendu de documents de formats bureautiques ou autres vers PDF ou autres.

2.2 Cloudification d'application verticales tierces

2.2.1 Contexte

Un objectif stratégique pour Nuxeo est d'encourager le développement d'appli-cations verticales (aussi appelées CEVAs pour "Content Enabled Vertical Applications" ou CCA pour "Composite Content Applications) au-dessus de sa plateforme, par des développeurs tiers.

Afin de rendre cette approche attractive pour les développeurs, Nuxeo souhaite proposer une approche intégrée, depuis le développement jusqu'au déploiement des applications ("build to run"), en passant par la facturation et le paiement par les utilisateurs, avec partage des revenus entre Nuxeo et les développeurs tiers, similaire au modèle de l'App Store d'Apple.

Il est à noter qu'une telle "Nuxeo Marketplace" existe déjà depuis fin 2010, avec pour l'instant des fonctions beaucoup plus limitées, et n'est notamment pas intégrée à l'offre cloud de Nuxeo.

2.2.2 Objectifs

- Proposer aux développeurs tiers (ISVs) un environnement de build, de test et de run qui leur permette de se concentrer sur leur offre métier.

- Proposer un espace de promotion de ces applications, ainsi que la possibilité pour les clients de commander directement une application selon un mode de facturation qui peut être déterminé

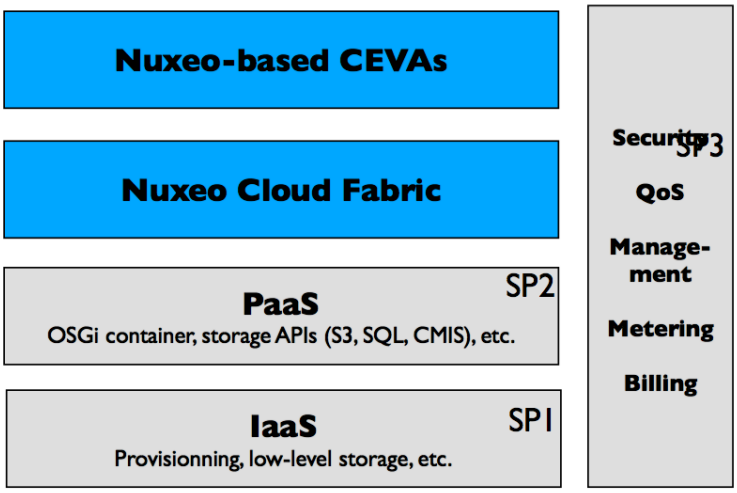

Figure 3: Nuxeo pour les CEVAs

2.2.3 Principales tâches

- Permettre l'intégration de l'outil de développement Nuxeo Studio avec le cloud (marketplace et déploiement).
- Intégrer le modèle de partage de revenu dans le système de comptabilité et de facturation.

2.3 Nuxeo Document Storage as a Service

Cette partie du use case 4.1 correspond au SP 2.6 ("CMIS as a Service"), nous en reprenons ici seulement les grandes lignes en renvoyant au livrable 2.6.1 pour les détails des spécifications.

2.3.1 Contexte

Les plateformes de PaaS actuelles dont s'inspire Compatible One (ex: Amazon S3 ou Google Storage) se focalisent sur le stockage de BLOBs ou de fichiers. Il ne proposent pas les services que l'on attend d'un système de gestion documentaire et qui ont été, pour partie, formalisés dans le modèle documentaire du standard CMIS: gestion des gestion des droits hiérarchique, indexation multi-critères, gestion du cycle de vie des documents, check-in check-out, versionning, locking, etc.

Il s'agit donc ici de proposer ces services sous la forme d'une API du PaaS compatible (API Java exposée sous forme de services OSGi, ou API web REST) afin de permettre à des développeurs de réaliser des applications documentaires sans nécessairement adopter tous les paradigmes de la plateforme Nuxeo, notamment son interface utilisateur.

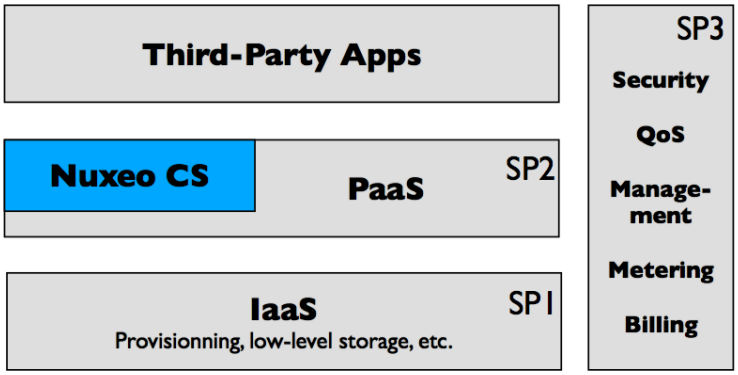

Figure 4: Nuxeo comme service PaaS

2.3.2 Objectifs

- Exposer via CMIS (+ d'éventuelles extensions) et des API Java les services fondamentaux de la GED, sous une forme consommable par des développeurs tiers, et facturer cette consommation à la demande selon le modèle Amazon ou Google.
- Valider que ces API sont bien pertinentes et que le modèle a été découpé aux bons endroits.

2.3.3 Challenges

- Implémenter le service après découpage adéquat des composants serveurs Nuxeo.
- Démontrer la pertinence des API par un ensemble de démonstrateurs
- Valider la montée en charge par des benchmarks

2.3.4 Principales tâches

- Implémenter les API serveur.
- Benchmark des performances.
- Réalisation d'un démonstrateur d'application Web (mashup navigateur) basé sur l'API CMIS browser binding exposée par le démonstrateur.
- Réalisation d'un démonstrateur d'application de synchronisation desktop pour Windows, Linux et Mac OS, exploitant les API REST côté serveur.

- Réalisation d'un démonstrateur d'application mobile orientée contenu pour iPhone / iPad, Android et HTML5, exploitant les API REST côté serveur.

3 Requirements

NB: pour des raisons historiques, cette partie a été écrite en anglais.

3.1 High level requirements

3.1.1 Integrated Configuration and test environment in the Cloud

Nuxeo Connect users can already use the configuration services provided by Nuxeo Studio to configure their local Nuxeo instance.

In the future, we want to leverage the Cloud infrastructure to let the user directly test his customization on a Cloud instance.

This basically means:

- create the target VM
- deploy the target Nuxeo distribution
- deploy the Studio configuration

Since we are currently integrating Studio and Eclipse IDE, the next step will be to provide the same Cloud services so that developers working on Nuxeo Platform can use the Could infrastructure to provision test instances and test their custom applications.

3.1.2 One click production environment

For clients using the Cloud infrastructure to customize and develop their Nuxeo based application, the obvious next step is to be able to run the production instances directly on the Cloud too.

3.1.3 Scaling out

Nuxeo also wants to leverage the Could infrastructure to provide an easy scaling solution.

This includes:

- using Cloud based filesystem for document storage
- using Cloud ready Database systems for meta-data storage
- provide true multi-tenant support
- deploy Nuxeo components on multiple VMs (Spanning or clustering)

3.1.4 ECM as a service

Nuxeo ECM platform provides several technology neutral API to access and manage content:

- CMIS connector: standard compliant interoperability protocols
- Content Automation: Extensible and Business friendly REST API

One of the goals of Nuxeo's Cloud approach is to be able to provide ECM features as a service via CMIS and Content Automation.

3.2 Technical requirements

3.2.1 Nuxeo component deployment and OSGi

Nuxeo platform component model being based on an OSGi, we want to be able to deploy the required bundles according to a global configuration.

OSGi-based distribution Currently Nuxeo distributions are a selection of Maven Artifacts (Code or Configuration resources) that is generated using a dedicated maven plugin.

The selection of the required artifact is based on maven dependencies and the assembly is done at packaging time.

Our goal is to be able to have distributions based on OSGi rather than maven.

This means having a descriptor listing the required bundles and letting the OSGi deployment and provisionning system pull and activate the needed bundles dynamically.

This kind of features is for example available using some forked versions of P2 director with Equinox.

At the end this high level descriptor should be enough to create a specific instance of Nuxeo on the Cloud:

- the descriptor contains the list of needed features
- the OSGi dependency and provisioning system know how to find the bundles
- the PAAS knows how to deploy and run a set of OSGi bundles as an application
- OSGi know how to assemble (fragments), start and activate the bundles

We need to define a standard descriptor model for this as well has the tools and infrastructure to run it.

For illustration purpose, the target command line could look something like:

```
> compatible --deploy <software-descriptor-url> <instance-descriptor-url>
```

where:

- `software-descriptor-url`: is an url pointing to a file listing the top level required bundles
- `instance-descriptor-url`: is an url pointing to a file containing VM deployment configuration

The expected gains for this switch from maven to an OSGi distribution model are:

- better integration with the low level provisioning systems
- be able to dynamically add new features by adding new bundles (no restart, just additional activation)
- be able to dynamically add or refresh configuration bundles

These features are very important for running a useful ECM platform on the Cloud:

- we want to be able to easily refresh configuration
  ⇒ let end users change some configuration in Nuxeo Studio and directly see the result in his production system
- we want to let able to add or remove services on a running application instance
  ⇒ let the end user subscribe to additional services and features (without recreating a new instance or restarting the existing one)

Stay compatible Even is we build a very cool Cloud platform, we will still need to stay as much as possible compatible with other deployment infrastructure.

And, at least, we must provide bridges so that applications that run inside the Cloud, can also be run on premises.

So, even if we must leverage the Cloud infrastructure, we should still be able to run part of the software services on standard internal IT infrastructure.

It provides a smooth migration to the cloud model and allows for a mixed model:

- use Cloud for evaluation or development environment, but deploy on premises for security reasons

- Manage dev and testing in-house but use the Cloud for production hosting
- Start with in-house deployment and scales out to the Cloud when needed

This basically means that there should be a strong decoupling between the application layer and the cloud infrastructure and deployment model.

For that matter, OGSi model provides part of the solution since it standardize several aspects of the application life-cycle.

For on premises deployments we need to be able:

- to propose alternative for the Cloud based features (like standard DB/FS storage)
- to run as a standard JEE application

This second point may seem like a problem, but:

- this problem can be not addressed
- this should not be a big constraint

In fact several application servers have OSGi support, and Nuxeo EP currently deploys OSGi and Nuxeo Components in JBoss 5 and Tomcat 6.

The difficulty will be to disturb as less as possible the people deploying on premises application:

⇒ this must be as simple and copying a WAR or an EAR

⇒ it does not means we need to deploy a real WAR or EAR (we can rely on deployment hooks and embedded OSGi runtime)

3.2.2 Multi-tenancy

We have use cases for using Nuxeo DM as a multi-tenant aware platform.

The typical requirements are:

- be able to have per-tenant data
  ⇒ Users from company A can not access the documents from company B.
  ⇒ Company B can ask for a data restore without impacting company A
- be able to have per-tenant configuration
  ⇒ The InvoiceA document Type is only available for people from company A.

- be able to have per-tenant feature list (available services)
  ⇒ CompanyA has access to a Signature service, but not CompanyB
- be able to do hot-reconfiguration/redeployment on a per-tenant basis
  ⇒ CompanyA can do an upgrade without impacting CompanyB

Doing Multi-tenant in the Could Traditional approaches for managing multi-tenant is to change the application so you have business rules to tell the software how to use shared the hardware resources between clients.

This approach has several drawbacks in our case:

- Making multi-tenant aware all software component is hard job
- possibly a lot of work
- must be addressed for each new feature
- Since application level rules may change, the admin work (ex: backup and restore) can become a nightmare

A brute force approach would be to say that since of the Cloud the resources are virtually unlimited, we can simply clone a new instance for each client.

But, we can probably do better than that to leverage as much as possible the overhead of adding a 'logical' (client) instance to a running instance.

OSGi and multi-tenant If we include the Multi-tenant management system directly inside the OSGi deployment infrastructure, we should be able to avoid most of the application level complexity.

From a high level point of view, we want to be able to run several OSGi distributions at the same time while sharing the common bundles and configuration.

So for example, if user's Principal contains information about a Tenant:

- we should be able to make available some services or not
- we should be able to take into account one configuration bundle or an other

My understanding is that there are currently discussion somehow related to these problematics in the OSGi/P2/Virgo community.

Infrastructure requirements Low level infrastructure is also impacted by the multi-tenant use case:

- impact on backup/restore systems
- impact on admin and deployment tools

Nuxeo level requirements Even if we have a fully multi-tenant aware infrastructure, there will still be impacts at Nuxeo level.

- OSGi alignments for services
- Improve extension points registries (or align to Equinox system)
- Improve fragment system (or align on OSGi partial standard)

3.2.3 VM provisioning

In order to automate instance creation, we need a vendor neutral API to provision cloud instances that will be used to deploy Nuxeo instances.

Ideally, we want it to be integrated with the OSGi distribution system so that we can deploy in one command: Software distribution and Runtime environment.

3.2.4 Distributed storage

Today the Nuxeo content repository uses a simple file system to store the binary files, however files are stored in an intelligent manner in order to minimize duplication and avoid locking and reference counting.

In the future we want to leverage distributed storage services like S3. This will allow storage of binary files (which are the biggest consumers of the storage space used by a document management system) in a totally scalable manner, with the intrinsic redundancy offered by such cloud-based systems.

This will be a new plugin for the Nuxeo Binary Manager:

- using a local filesystem cache (for performance reasons),
- using as much as possible a neutral distributed file system API (as provided for instance by the jclouds BlobStore project).

3.2.5 NoSQL storage

Even if we have no immediate requirement for it, we want to explore the possibility of using a NoSQL-based storage for our content repository.

The target is to be able to have a theoretically infinite-scale storage for the Nuxeo repository metadata.

This task is significantly more complex that simply addressing binaries storage since we need to be able to:

- manage transactions,
- manage locking,

- have an efficient query system, with dynamically created queries in some use cases,
- provide full-text indexing.

To be fair, this does not need to be a real NoSQL storage. Having a large scale RDBMS clustering and sharding system, would be good as well.

3.2.6 Distributed Caching

Nuxeo supports clustering at repository level.

The clustering model requires a cache system that can be synched between the Cluster nodes.

This is currently done:

- using separated in memory "session caches":
- there is no intra-VM shared cache
- there is no multi-VM shared cache
- multi-VM sync go through the database

We plan to work on this subject to provide a truly distributed caching system that will can use in Cluster mode on the Cloud.

This improvement will probably be needed if we want to scale out efficiently on multiple Cloud nodes.