Interface de gestion d'une architecture de stockage distribué

Sébastien Riaudel

CONSERVATOIRE NATIONAL DES ARTS ET MÉTIERS

CENTRE RÉGIONAL ASSOCIÉ DE NANTES

_____

MÉMOIRE

Présenté en vue d'obtenir le

DIPLOME D'INGENIEUR C.N.A.M

En

INFORMATIQUE – Option : Système d'Information

Par

Sébastien RIAUDEL

_____

Interface de gestion d'une architecture de stockage distribué

Soutenu le 16 Décembre 2011

_____

JURY :

Présidente : Mme **Élisabeth METAIS**

Membres :

Mr **Henri BRIAND**, responsable du cycle ingénieur informatique CNAM Pays de la Loire ;

Mr **Jean-Pierre GUEDON**, enseignant chercheur à l'Ecole Polytechnique de Nantes ;

Mr **Pierre Evenou**, co-fondateur de la société Fizians ;

Mr **jean-Côme ANSQUER**, responsable Ingénierie Infrastructure chez SIGMA Informatique ;

Remerciements

Je tiens tout d'abord à remercier Monsieur JeanPierre Guédon, enseignant chercheur à l'Ecole Polytechnique de Nantes et membre de l'équipe IVC du laboratoire IRCCyN d'avoir accepté de m'encadrer et me conseiller tout au long de ce mémoire.

Je remercie par ailleurs tous les membres de l'équipe IVC de m'avoir accueilli au sein du laboratoire de recherche.

Je remercie également Monsieur Pierre Evenou de m'avoir confié le sujet du mémoire et toute l'aide qu'il a pu m'apporter au cours de sa réalisation.

Je remercie aussi toutes les personnes qui me font l'honneur de participer au jury en dehors des personnes que j'ai précédemment citées, je pense notamment à Monsieur Ansquer pour l'intérêt qu'il porte à la réalisation de mon projet, Monsieur Briand, Madame Métais qui préside le jury.

Enfin, toute ma reconnaissance et mon affection vont vers mon épouse qui m'a toujours soutenu ainsi que mes trois enfants : Théo, Anaïs et Amandine pour leur patience envers un papa si occupé.

Liste des abréviations

AMI	Amazon Machine Image
API	Application Programming Interface
ARPANET	Advanced Research Projects Agency Network
ASP	Application Services Providers
AWS	Amazon Web Services
B2B	Business to Business
CERN	Conseil Européen pour la Recherche Nucléaire
CGI	Common Gateway Interface
CIDR	Classless Inter-Domain Routing
CMS	Content Management System
CNRS	Centre National de la Recherche Scientifique
CPU	Central Processing Unit
CRM	Customer RelationShip Management
CRUD	Create Read Update Delete
CVS	Concurrent Versions System
DFS	Distributed File System
EBS	Elastic Block Store
EC2	Elastic Compute Cloud
FTP	File Transfert Protocol
GFS	Google File System
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
IP	Internet Protocol
IRCCyN	Institut de Recherche en Communications et Cybernétique de Nantes
IVC	Image Vidéo Communication
JVM	Java Virtual Machine
MVC	Model View Controller
MVT	Model Template View
NFS	Network File System
NIST	National Institute of Standards and Technology
ORM	Object Relational Mapping
OSI	Open Systems Interconnection
PaaS	Platform as a Service
PAC	Pierre Audoin Consultant
pNFS	Parallel Network File System
PVM	Python Virtual Machine
RAM	Random Access Memory
RDA	Rich Desktop Application
REST	Representational State Transfer
RIA	Rich Internet Application
RPC	Remote Procedure Call
RSS	Really Simple Syndication

S3	Simple Storage Service
SaaS	Software as a Service
SID	Security IDentifier
SOA	Service Oriented Architecture
SOAP	Simple Object Access protocol
SSH	Secure Shell
SSL	Secure Socket Layer
SYNTEC	Syndicat des Sociétés de Services et d'Ingénierie Informatique
TCO	Total Cost of Ownership
TCP	Transmission Control Protocol
TIC	Technologies de l'Information et de la Communication
UDDI	Universal Description Discovery and Integration
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VFS	Virtual File System
VM	Virtual Machine
VPC	Virtual Private Cloud
VPN	Virtual Private Network
WSDL	Web Service Description Langage
WSGI	Web Server Gateway Interface
WSOA	Web Services Oriented Architecture
XDR	External Data Representation
XHTML	Extensible HyperText Markup Language
XML	Extensible Markup language

Table des matières

Introduction

1.1 Le contexte

Ce mémoire est réalisé en participation avec JeanPierre Guédon, enseignant chercheur à l'Ecole Polytechnique de Nantes et membre de l'équipe IVC du laboratoire IRCCyN, ainsi qu'avec la collaboration de la société Fizians, porteur du projet.

1.1.1 Le laboratoire IRCCyN

L'IRRCyN est une unité mixte de recherche du CNRS, rattachée au département scientifique des Sciences et Technologies de l'Information et de l'Ingénierie, et dont les tutelles locales sont l'Ecole Centrale de Nantes, l'Université de Nantes et l'Ecole des Mines de Nantes. Le laboratoire comporte une douzaine d'équipes de recherche parmi les orientations suivantes :

- *L'automatique (équipe Commande)*
- *Le traitement du signal et des images (équipe ADTSI)*
- ***La vidéo communication (équipe IVC)***
- *La robotique (équipe Robotique)*
- *La conception mécanique assistée par ordinateur (équipe MCM)*
- *La modélisation et l'optimisation de processus de production (équipe MO2P)*
- *L'ingénierie virtuelle pour l'amélioration des performances industrielles (équipe IVGI)*
- *Les systèmes temps réel (équipe Temps Réel)*
- *La modélisation et la vérification des systèmes embarqués (équipe MoVES)*
- *Les systèmes logistiques et de production (équipe SLP)*
- *Les systèmes à évènements discrets (équipe ACSED)*
- *La psychologie cognitive et l'ergonomie (équipe PsyCoTec)*

Le laboratoire rassemble en janvier 2011 un peu plus de 260 personnes avec 103 chercheurs et enseignants-chercheurs (dont 14 du CNRS), 17 ingénieurs, techniciens et administratifs ainsi que 142 chercheurs non-permanents (dont 111 doctorants, 28 CDD et 3 PostDocs) sans compter de nombreux professeurs invités et divers stagiaires.

1.1.2 L'équipe IVC

L'équipe IVC est constituée d'une vingtaine de personnes permanentes et d'une dizaine de stagiaires. Son axe de recherche s'articule autour de la perception, la communication et la représentation de l'information et se décline en cinq thèmes de recherche :

- *Représentation et perception Modèles psychovisuels*
- *Représentation MOJETTE et géométrie discrète*
- *Représentation et communication Vidéo-Multimédia*
- *Représentation et communication réseaux*
- *Représentation et écrits et documents.*

Dans l'équipe IVC, je travaille sur l'axe Mojette qui possède trois types d'applications :

- *Imagerie médicale*
- *Transport réseaux*
- *Stockage de données*

Mon sujet de mémoire concerne la partie « *stockage de données* » dans le contexte du « *Cloud* » ou « *informatique dans les nuages* ». Le Cloud, qui place Internet au cœur de l'activité des entreprises, permet d'utiliser des ressources matérielles distantes (*serveurs, ordinateurs, etc.*) pour créer des services accessibles en ligne.
A partir de là, les entreprises peuvent développer des offres de type « *SaaS* » gérables depuis une interface Web et accessibles depuis n'importe quel terminal connecté à Internet.

1.2 Le projet de réalisation

1.2.1 Présentation du sujet

L'objectif du mémoire est de créer une application *SaaS* qui permettra au client de gérer, d'administrer et de contrôler ses machines virtuelles et de bénéficier d'un espace de stockage accessible à l'ensemble des machines virtuelles (*VMs*), à travers un Cloud Storage (*espace de stockage extensible et accessible en réseau*). Les clients pourront ainsi via une interface Web, ajouter, supprimer des machines virtuelles et augmenter selon le besoin le volume de stockage distribué.
Les machines virtuelles seront fournies par Amazon via les services Web d'Amazon *[Amazon 2010]*.
Dans les services *AWS* figure notamment *EC2* (*Amazon Elastic Computer Cloud*) qui permet d'obtenir les machines virtuelles. Amazon propose également le service *EBS* (*Amazon Elastic Block Store*) qui permet d'ajouter de l'espace de stockage propre à chaque machine virtuelle.
Puis le groupe américain met à disposition le service « *DevPay* » qui permet de gérer les comptes clients mais aussi la facturation de ses services.

L'application Web que je vais développer reposera sur le système de fichiers distribués « *Rozofs* », développé par la société Fizians, qui permet de mutualiser l'espace de stockage des machines virtuelles afin d'avoir un seul système de fichiers commun, accessible à tous.
Ainsi, l'offre B2B de Fizians permettra à l'entreprise de gérer de manière simple, fiable et efficace, les machines virtuelles et l'espace de stockage partagé, sans pour autant supporter les coûts d'infrastructures. Le client paye uniquement à l'usage.
Par ailleurs, l'application s'appuiera sur des interfaces de programmation (*API*) fournies par Amazon afin de gérer les machines virtuelles et de contrôler la facturation clientèle. Une page Web authentifiera si besoin le client et devra exploiter ces *APIs*.

1.2.2 Les thèmes abordés

Pour mener à bien ce projet, il convient d'aborder plusieurs thèmes s'articulant autour du Cloud Computing et du stockage distribué avec les différentes solutions qui existent sur le marché. Ensuite on étudiera la manière de mettre en œuvre une application Web au service d'un Cloud Storage avec tous les outils nécessaires comme les interfaces de programmation, les langages de développement orientés Web, les applications réparties, etc. Une fois cet état de l'art réalisé, on procédera à l'élaboration du projet par une phase de modélisation puis de conception.

Etat de l'art

2.1 Le Cloud Computing

2.1.1 Définition

Selon le NIST *[Mell et al. 2009],* le Cloud Computing est un modèle qui permet d'accéder à la demande, à un « *pool* » partagé de ressources informatiques configurables (*réseaux, serveurs, stockage, applications et services*) qui peuvent être provisionnées rapidement et distribuées avec un minimum de gestion ou d'interaction avec le fournisseur de services.

Pour le Syntec *[Syntec 2010]*, le Cloud Computing consiste en une interconnexion et une coopération de ressources informatiques, massivement extensibles, exploités par de multiples clients externes sous forme de services fournis via Internet. Ces ressources sont situées au sein d'une même entité ou dans diverses structures internes, externes ou mixtes.

Pour les entreprises utilisatrices, le Cloud Computing peut se définir comme une approche leur permettant de disposer d'applications, de puissance de calcul, de moyens de stockage, etc.

Wikipedia *[Wikipédia 2011]* apporte également une définition plus imagée du Cloud en le comparant à une distribution de l'énergie électrique. La puissance de calcul ou de stockage de l'information est proposée à la consommation par des compagnies spécialisées. De ce fait, les entreprises n'ont plus besoin de serveurs propres, mais confient cette ressource à une entreprise qui leur garantit une puissance de calcul et de stockage à la demande.

2.1.2 Les origines du Cloud

Le Cloud Computing n'est pas une technologie récente (*Cf. Figure 01*) puisqu'elle a été introduite dans les années 1960 avec John McCarthy, pionnier dans l'intelligence artificielle, qui énonçait déjà le concept de grille de calcul en affirmant que le calcul peut être organisé comme un service public *[Mohamed 2009]*. Tandis que Joseph Carl Robnett Licklider, responsable du développement d'ARPANET, rêvait « *d'un réseau global où chacun aurait la possibilité de s'interconnecter pour accéder aux données et aux programmes depuis n'importe où* ».

Le terme devient populaire en octobre 2007 lorsqu'IBM et Google forment une collaboration « *Blue Cloud Effort*» *[Lambel 2007]* afin d'aider les banques et diverses entreprises à distribuer leurs calculs sur un très grand nombre de machines sans posséder d'infrastructure interne. Depuis, il devient omniprésent sur Internet avec les sites comme « *Wikipédia* », « *E-bay* », etc.

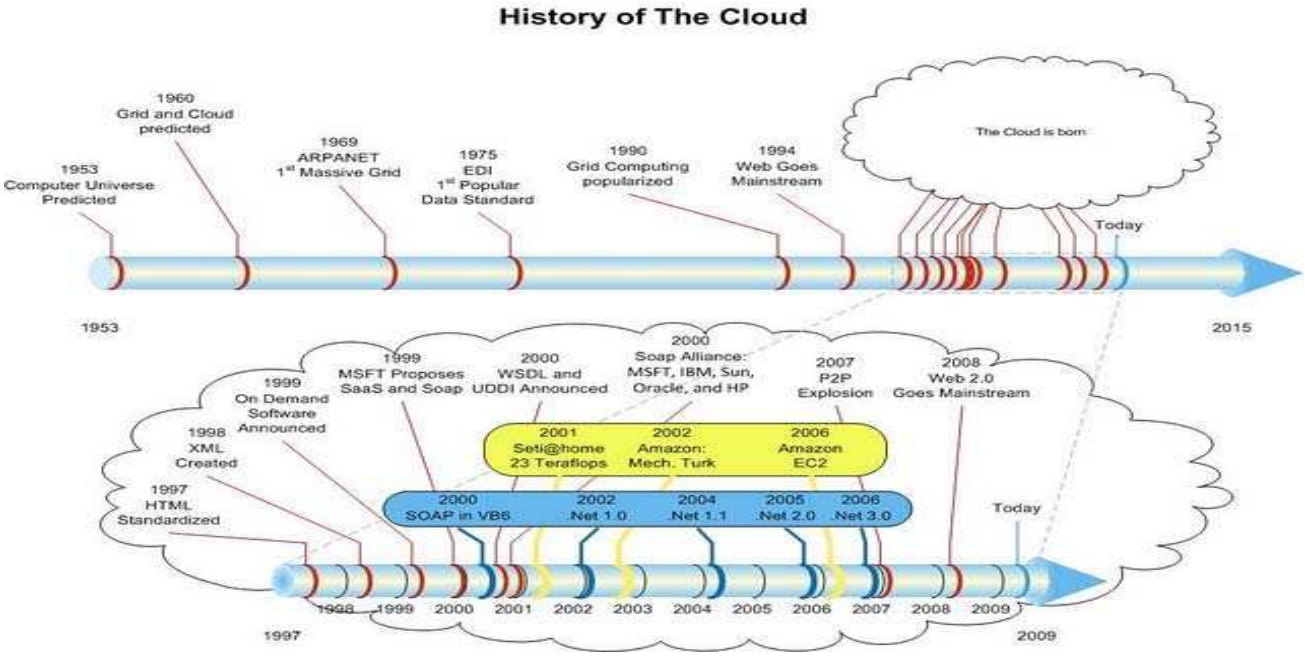


Figure 01 *- History of the Cloud [Blanchard 2009].*

2.1.3 Les types de Cloud

Différents modèles de Cloud coexistent pour répondre à des problématiques et des besoins différents *[Velte 2010]*.

2.1.3.1 Cloud public

Il est externe à l'organisation, géré par un prestataire externe propriétaire des infrastructures, avec des ressources partagées entre plusieurs sociétés. Ce type de Cloud est accessible via Internet par une interface Web. Il offre ainsi la possibilité d'allouer de la ressource au forfait d'utilisation.

2.1.3.2 Cloud privé/privatif

Le Cloud privé (*interne ou privatif*) se définit comme la mise en place d'un réseau informatique propriétaire ou un centre de données fournissant des services hébergés pour un nombre restreint d'utilisateurs. Le Cloud privé mise sur une infrastructure stable et moins étendue. Il contient des données stratégiques et reste au cœur de l'entreprise.

Concrètement, les applications virtualisées « *privées* » sont soit administrées directement par l'entreprise (*qui gère seule son infrastructure*), soit mutualisées (*un prestataire de confiance prend en charge une partie des services externalisés*). A la différence avec un Cloud public, le Cloud privé permet à une entreprise de gérer les données et les traitements en interne et évite ainsi les problèmes de bandes passantes dans le flux réseau. D'autre part, le Cloud privé est plus fiable en termes de sécurité et n'est pas confronté aux obligations légales en matière juridique.

2.1.3.3 Cloud hybride

Ici, il s'agit de la conjonction de plusieurs Cloud (*public et privé*) amenés à coopérer, à partager entre eux les applications et les données. Les intérêts d'un Cloud hybride sont les suivants :

- *L'optimisation d'un parc existant.*
- *Il est inutile de maintenir en permanence un parc de machines dimensionnées pour supporter des pics de charge temporaire.*
- *Le provisionnement à la demande.*
- *Le Cloud hybride permet d'améliorer les délais de mise à disposition de nouvelles ressources matérielles.*
- *L'établissement d'une infrastructure de secours.*
- *Mise en œuvre d'un plan de reprise d'activité.*

Le Cloud hybride permet donc de faire cohabiter un SI interne d'une organisation à un service proposé par le Cloud public afin de répondre à des objectifs, des besoins de l'entreprise.

2.1.4 Les services

Le Cloud Computing permet aux entreprises de consommer des services informatiques à la demande. Les fournisseurs Cloud déclinent donc ce concept en trois marchés : *IaaS*, *PaaS* et *SaaS*.

Figure 02 *– Représentation du Cloud [Schuller 2008].*

2.1.4.1 IaaS (*Infrastructure as a Service*)

Le premier modèle de Cloud est IaaS « *Infrastructure as a Service* », qui permet au client de louer des capacités de traitement, de stockage, de réseau et autres ressources de calcul.

L'entreprise maintient les applications, l'intégration SOA, les bases de données, le logiciel serveur. Le fournisseur Cloud maintient la virtualisation, le matériel serveur, le stockage, les réseaux.

L'acteur dominant dans ce domaine est *Amazon* avec AWS.

2.1.4.2 PaaS (*Platform as a Service*)

Le client peut déployer ses propres applications sur l'infrastructure Cloud, dans la mesure où le fournisseur supporte le langage de programmation. L'utilisateur contrôle ses applications déployées et peut aussi configurer l'environnement d'hébergement applicatif. Le fournisseur Cloud maintient l'intégration SOA, les bases de données, le logiciel serveur, la virtualisation, le matériel serveur, le stockage, les réseaux. Les principaux acteurs sont *Google* et *Microsoft*.

2.1.4.3 SaaS (*Software as a Service)*

Le SaaS consiste à fournir aux clients, des applications via Internet. Ces applications sont accessibles via différentes interfaces, clients légers, navigateur Web. Les clients n'achètent plus le logiciel, mais le consomme à la demande, en payant à l'usage réel. Le logiciel est hébergé chez le fournisseur, dans son propre Datacenter. *Salesforce.com* est un acteur majeur dans les applications SaaS.

2.1.4.4 L'intérêt du modèle de représentation

Le modèle de représentation du Cloud, décomposé en trois parties distinctes, permet de séparer les différents domaines de compétence et impose des abstractions à tous les niveaux.

Ainsi, les différentes couches n'interagissent pas entre elles et chaque système est développé en faisant abstraction des contraintes de plus haut et de plus bas niveau.

2.1.5 Le Cloud et la virtualisation

Le Cloud Computing permet de développer et d'utiliser des applications accessibles via Internet.

Les utilisateurs peuvent accéder aux données par l'intermédiaire d'applications orientées services qui fonctionnent sur des machines virtuelles. Le concept même du Cloud fait cohabiter plusieurs technologies qui ont contribué à son existence dont la virtualisation.

2.1.5.1 La virtualisation

Le principe consiste à partager les ressources matérielles d'une machine physique pour faire fonctionner et mettre à disposition plusieurs machines virtuelles. L'allocation des ressources physiques s'effectue par l'intermédiaire d'une couche logicielle entre la partie matérielle et applicative.

2.1.5.2 La Machine virtuelle

Une machine virtuelle est un environnement d'exécution qui émule une machine physique.

Elle se comporte exactement comme un ordinateur physique et contient ses propres CPU, mémoire RAM, disque dur et carte d'interface réseau virtuel. Parmi les techniques de virtualisation, on distingue la virtualisation complète de la paravirtualisation (*Cf. Figure 03*).

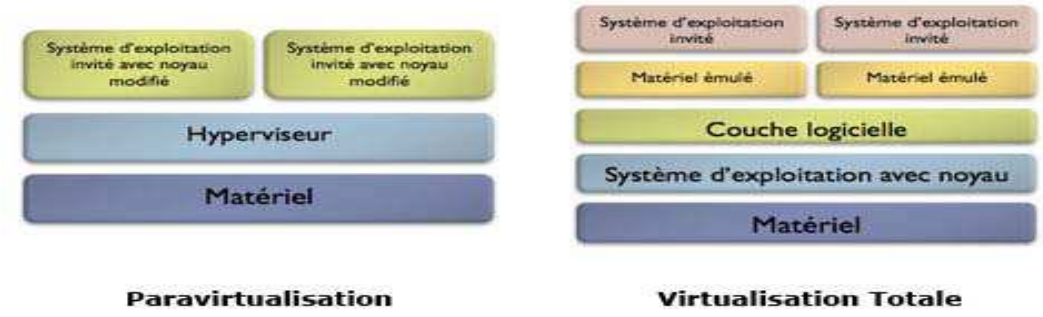


Figure 03 *– Type de virtualisation [Benkemoun 2009].*

2.1.5.3 L'hyperviseur

Le rôle de l'hyperviseur est de mettre en place une ou plusieurs machines virtuelles dans lesquelles sont exécutés des systèmes d'exploitation. Il existe deux types d'hyperviseur :

- *Hyperviseur de « type 1 »*

  L'hyperviseur s'exécute directement sur une plateforme matérielle, il n'est pas dépendant d'un système d'exploitation classique pour fonctionner. Ce type d'hyperviseur sera employé pour de la paravirtualisation.

  Exemple : Xen, VMware, KVM, etc.

- *Hyperviseur de « type 2 »*

  C'est un hyperviseur qui est géré par un système d'exploitation classique comme n'importe quel autre programme. Il utilise les services fournis par le système d'exploitation hôte pour gérer de la mémoire et l'ordonnancement des machines virtuelles. Ce type d'hyperviseur sera employé pour de la virtulisation complète.

  Exemple: Microsoft VirtualPC/VirtualServer, VMware Server/Workstation, etc.

2.1.5.4 La virtualisation complète

La virtualisation complète fournit un environnement matériel virtuel (*interface et ressources*) représentant une architecture réelle. La virtualisation complète consiste donc à émuler l'intégralité d'une machine physique. Ainsi, l'ensemble des systèmes d'exploitation virtualisés s'exécutant sur un unique ordinateur, peut fonctionner indépendamment les uns des autres et être vu comme des ordinateurs à part entière sur un réseau.

2.1.5.5 La paravirtualisation

La paravirtualisation vise à instaurer une collaboration directe entre l'hyperviseur et ses systèmes « *invités* » dans le but d'en faciliter la gestion. La paravirtualisation implique donc une modification des systèmes d'exploitation « *invités* » pour communiquer directement avec un hyperviseur au lieu de communiquer avec une machine physique. Elle apporte une meilleure fiabilité dans l'utilisation des ressources, car dans une virtualisation complète, tous les éléments sont émulés. (*Bios, lecteur...)*.

L'inconvénient majeur est qu'il est nécessaire d'adapter les systèmes d'exploitation pour chaque couche de virtualisation.

2.1.5.6 Les solutions de virtualisation

De nombreuses solutions de virtualisation *[Ozitem 2007]* existent sur le marché aussi bien dans les solutions propriétaires comme *VMware* ou *Microsoft* mais également dans les logiciels libres comme *Xen* et *KVM*.

- ***Vmware vSphère 4*** *est aujourd'hui un acteur majeur dans le monde de la virtualisation.*
  Le principe de l'offre repose sur l'utilisation d'un hyperviseur de type 1.
  Son rôle est de gérer l'accès à ces ressources grâce à un planificateur pour la mise à disposition dans les machines virtuelles. Ces dernières sont alors reliées aux ressources physiques par l'intermédiaire de la couche de virtualisation VMware qui héberge un système d'exploitation. VSphere4 apporte également des fonctionnalités de haute disponibilité, de migrations à chaud, de sauvegardes centralisées, etc.
- ***Microsoft Hyper-V****, livré avec Windows Server 2008 R2, repose sur l'utilisation d'un hyperviseur de type 1 et permet ainsi d'installer plusieurs machines virtuelles sur un serveur physique. Hyper-V apporte de la haute disponibilité avec le basculement des machines d'un serveur à l'autre qui s'appuie sur la technologie Microsoft Cluster (MSCS).*
  D'autre part, l'offre permet de migrer à chaud des serveurs grâce à Virtual Machine Manager qui permet de gérer l'ensemble des machines virtuelles, les ressources et les goulets d'étranglements.
- ***XenServer****, récemment racheté par Citrix, est une solution open source offrant plusieurs solutions de virtualisation. L'offre classique est une solution d'hypervision à laquelle s'ajoute une offre récente « Xen Cloud Platform » orientée Cloud Computing.*
  Xen Hypervision offre une solution de paravirtualisation avec un hyperviseur qui contrôle directement la partie matérielle de la machine et contrôle l'utilisation des ressources.
- ***KVM*** *est une solution open-source de virtualisation complète et s'appuie comme Xen sur les fonctions de virtualisation matérielle « AMD-V » et « INTEL-VT ».*
  De plus Kvm peut exécuter des machines virtuelles Linux ou Windows avec les ressources matérielles virtualisées et il est compatible avec les images vmx de VmWare. Aujourd'hui, KVM est intégré au noyau de base d'un système Linux.

2.1.6 Les enjeux

Le Cloud offre de nombreux avantages pour une entreprise qui cherche à investir dans une infrastructure informatique sans pour autant en assurer la gestion technique.

- ***Architecture adaptable à la demande***

  Le Cloud permet d'allouer de manière dynamique et automatique de la ressource supplémentaire, de l'ajuster par rapport au besoin de l'entreprise notamment avec les montées en charge.

- ***Ressource partagée***

  La ressource est disponible pour tous les utilisateurs abonnés. Le fournisseur du Cloud possède et gère les serveurs physiques dont les ressources sont partagées avec d'autres utilisateurs.

- ***Virtualisation***

  La virtualisation des ressources informatiques permet de maximiser, d'optimiser le hardware.

- ***Gestion souple***

  Le Cloud public se gère par une interface Web pour la gestion des ressources informatiques. Le Cloud privé ou hybride sert en particulier comme moyen de stockage interne pour assurer la virtualisation et la consolidation des ressources informatiques. Il permet à l'utilisateur de gérer lui-même les ressources informatiques qu'il désire à travers une interface Web.

- ***Avantage concurrentiel***

  Il permet à une petite entreprise de démarrer un projet avec un minimum de capital et la possibilité de le faire évoluer au fur et à mesure.

- ***Réactivité, adaptabilité***

  Le modèle du Cloud améliore l'agilité d'une entreprise en réduisant les délais de lancement d'un projet.

2.1.7 Les contraintes

Les principales critiques du Cloud Computing concernent sa stabilité et sa sécurité. L'utilisation de l'informatique dans les nuages comporte un double problème de propriété, d'une part la propriété des systèmes d'information et d'autre part, la propriété des informations de ces systèmes.

- ***Dépendance du fournisseur***

  L'externalisation dans les nuages permet à l'entreprise de se concentrer sur son cœur de métier, mais la rend également dépendante d'un tiers. La défaillance de ce tiers peut entraîner de lourdes pertes pour l'entreprise utilisatrice.

- ***Incompatibilité des applications***

  Les machines virtuelles ne supportent que les systèmes d'exploitation standards (Windows, Linux) et non les systèmes propriétaires comme HP-UX. D'autre part, certaines applications de technologies anciennes ou propriétaires ne seront pas prises en charge. Ce qui impliquera de les réécrire, refaire ou supprimer.

- ***Application temps réel non pris en charge***

  Bien que le Cloud puisse permettre d'accéder à une grande puissance de calcul sans supporter le coût du matériel, il ne permet pas, pour l'instant, l'utilisation des applications temps réel.

- ***Problème de disponibilité***

  Les services sont basés sur la politique du « Best Effort ». La connectivité sur l'Internet public met en péril l'exécution d'application critique avec des contraintes en temps réel rigoureuses.

- ***Problème de bande passante***

  Les réplications sollicitent de la bande passante. Il est nécessaire également de mettre les applications et les bases de données qui fournissent beaucoup de transactions, près des serveurs physiques. (Datacenter ou serveurs de données).

- ***Problème de sécurité des données***

  Avec un disque classique dans une infrastructure interne, il est toujours possible de récupérer les données même après un formatage. Par conséquent, on a toujours un contrôle sur la totalité du disque. En revanche, les données dans un Cloud sont stockées dans une baie de disques partagées et on perd ainsi un degré de contrôle sur la disposition des données.

  Si on efface un fichier, on n'est pas certain que la donnée soit vraiment détruite.

- ***Confidentialité de données privées***

  D'un point de vue sécurité, protection et confidentialité des données, il n'est pas prudent d'utiliser des données sensibles au sein d'un Cloud (public). En tout état de cause, il faut prendre les mêmes précautions au niveau sécurité, intégrité et sauvegardes des données que dans le cadre d'une infrastructure interne. Un bon usage serait de crypter les données avant de les envoyer vers le fournisseur et d'éviter que les données soient accessibles à tous.

2.1.8 La tendance du marché

2.1.8.1 Représentation dans le monde

Au niveau mondial, le cabinet d'études IDC *[Pac 2010]* estime que les services Cloud représentaient 5% des investissements TIC mondiaux en 2009, soit 17 milliards de dollars.

Poussé par une phénoménale croissance moyenne annuelle de 25%, le Cloud Computing capterait d'ici 2013, 10% des investissements mondiaux, soit 44 milliards de dollars.

2.1.8.2 A l'échelle de l'Europe

A l'échelle européenne, et selon une étude réalisée pour la Commission Européenne par le cabinet Pierre Audoin Consultant (PAC), le marché du Cloud Computing dans l'Europe des 27 atteignait 4 milliards d'euros en 2009 (*Cf. Figure 04*). Selon PAC, cette croissance dynamique devrait se maintenir jusqu'en 2015 pour atteindre 13% du marché des logiciels et services informatiques.

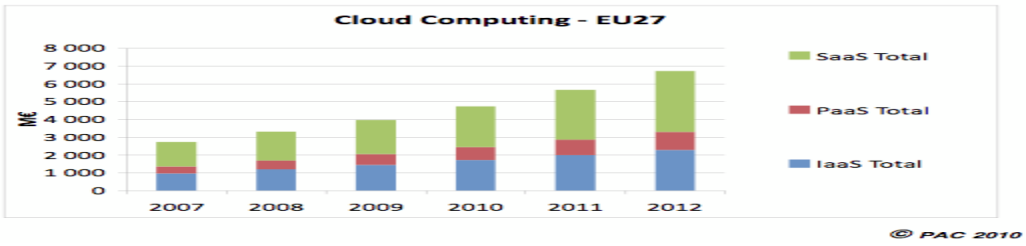


Figure 04 *– Marché du Cloud Computing en Europe [Pac 2010].*

2.1.8.3 A l'échelle de la France

Le Cloud Computing s'avère être un choix architectural majeur en 2010 et pour les années à venir, comme le prouve l'enquête réalisée en mars dernier par PAC pour EMC, Intel et Vmware *[Pac 2010]*. Ce marché est estimé à 780 millions d'euros et devrait croître de plus de 20% (*Cf. Figure 05*).

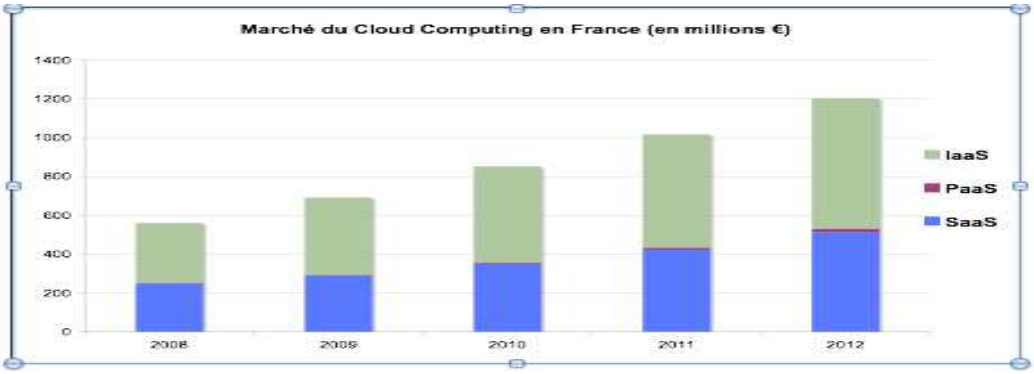


Figure 05 *– Marché du Cloud Computing en France [Pac 2010].*

Le Cloud Computing a initialement été dopé par les offres SaaS et en particulier par les offres liées à la messagerie, la collaboration, le CRM et les logiciels de gestion des Ressources Humaines. Malgré tout, ce sont les offres IaaS qui vont continuer à tirer le marché, en particulier en Europe. En effet, les directions informatiques souhaitent avant tout conserver le contrôle de leurs infrastructures en interne ou dans un Cloud Privé.

2.1.8.4 Le business modèle

Le modèle économique lié aux infrastructures réseaux et télécoms s'en trouve complètement bouleversé. L'externalisation des ressources permet aux entreprises d'alléger la gestion d'une infrastructure sous-jacente de plus en plus complexe, de mutualiser les matériels et de ne plus gérer l'administration de serveurs informatiques. L'informatique dans le nuage est plus économique grâce à son évolutivité. Ne nécessitant aucun investissement préalable (*homme ou machine*), son coût est fonction de la durée de l'utilisation du service rendu.

2.1.9 Les offres sur le marché

Les offres de Cloud sont nombreuses et dépendent du type de solution que l'on souhaite mettre en œuvre. Les offres les plus représentatives du marché sont aujourd'hui portées massivement par les fournisseurs américains.

2.1.9.1 Amazon

L'offre de Cloud d'Amazon *[Amazon 2010]* AWS est vaste et relativement complète (*Cf. Figure 06*).

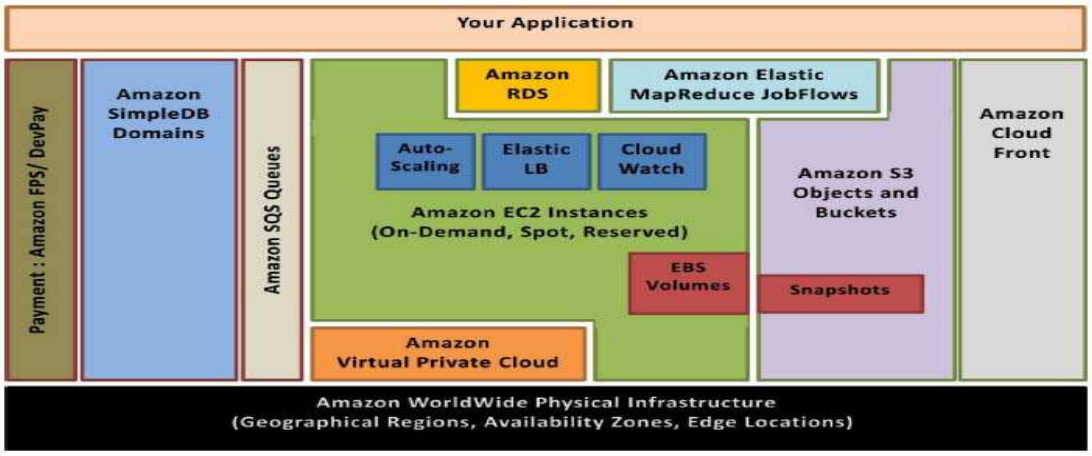


Figure 06 *– Amazon Web Services [Neoxia et al. 2011].*

- ***Amazon Elastic Compute Cloud***

  EC2 représente l'offre de base des services AWS et permet grâce à l'utilisation d'un hyperviseur Xen, de mettre en place dynamiquement des machines virtuelles.

  Une machine virtuelle est déployée à partir d'une AMI « Amazon Machine Image » constituée d'un système préconfiguré et modifiable pour l'adapter à ses propres besoins.

- ***Amazon Elastic Block Store***

  Amazon propose le service EBS qui permet d'obtenir de l'espace de stockage en mode bloc de 1Go à 1To, rattaché à une machine virtuelle. Les données contenues dans la machine sont par défaut volatiles. Il est donc impératif d'utiliser EBS afin de stocker les données dans des volumes qui persistent indépendamment de la durée de vie de la machine virtuelle.

- ***Amazon Simple Storage Service***

  Amazon S3 est un système de stockage virtuel, illimité qui intéresse particulièrement les développeurs Web. Le système stocke arbitrairement des objets allant jusqu'à 5 Go et sont organisés dans des conteneurs. Une API permet de gérer simplement le déploiement, le déplacement et la suppression des fichiers.

- ***Amazone Elastic Load Balancing***

  Amazon Elastic Load Balancing effectue un équilibrage de charge en dirigeant les requêtes de façon dynamique vers les machines virtuelles en état de les traiter. Par exemple il détecte les machines qui deviennent hors ligne.

- ***Amazon Virtual Private Cloud***

  Amazon VPC permet de faire du Cloud hybride en offrant la possibilité aux entreprises de connecter leurs infrastructures existantes à un ensemble de ressources Internet Amazon isolé par le biais d'une connexion VPN et d'étendre leurs capacités de gestion existantes.

- ***Amazon CloudWatch***

  Amazon CloudWatch permet de faire du monitoring dans l'utilisation du Cloud afin de contrôler notamment les ressources, la performance d'exploitation, l'utilisation CPU, la lecture et l'écriture du disque, le trafic sur le réseau.

Ainsi les services Web Amazon sont particulièrement adaptés dans les cas suivants :

- *Le client souhaite utiliser une tierce partie prenant en charge les softwares open-source.*
- *Le client souhaite implémenter un code existant.*
- *Le client souhaite transférer une application Web de sa machine vers un serveur.*
- *Le client veut avoir un contrôle intégral de son instance.*
- *Le client veut faire un test de grande ampleur (1000 instances).*

Aujourd'hui, Amazon est leader dans les offres IAAS du Cloud Computing.

2.1.9.2 Microsoft Azure *(IaaS)*

Microsoft Azure *[Msdn 2010]* propose également une solution IaaS avec d'autres services de niveau PaaS. L'offre initiale est composée des applications suivantes:

- *Live Services*
- *SQL Services*
- *Net Services*
- *SharePoint Services*
- *CRM Services*

La solution utilise les standards comme REST et SOAP, XML.

2.1.9.3 Google APP Engine (*PaaS*)

Google App Engine *[Google 2011]* est exclusivement PaaS.

Le langage de programmation « *APP Engine* » utilise uniquement Python et Java. Cette solution convient particulièrement pour les applications Web. En revanche, c'est une solution fermement contrôlée par Google qui ne permet aucune installation de logiciels open source.

Ainsi, il s'agit d'une solution envisageable si le client n'a aucun code préexistant.

De plus, elle permet de construire de manière simple et efficace des applications Web effectuant des échanges client serveur.

2.1.9.4 Salesforce.com et Force.com (*SaaS/PaaS*)

Salesforce.com *[IDG 2009]* est une solution SaaS qui permet de faire de la gestion de la relation client (*CRM*). L'extension « *Force.com* » est une solution PaaS qui permet aux développeurs d'utiliser le langage de programmation Apex afin d'ajouter des fonctionnalités supplémentaires dans leurs applications hébergées dans une infrastructure « *SalesForce.com* ».

En outre, Google et SaleForce.com ont créé une interface entre leurs solutions (*« App Engine » et « Force.com »*). Cependant, la solution « *Force.com* » nécessite d'être déjà client de Salesfore.com.

2.1.9.5 Gogrid(*IaaS*)

Gogrid *[IDG 2009]* est une solution IaaS qui délivre des services Web de stockage.

L'offre permet de déployer rapidement des serveurs virtuels Windows et Linux sur le Cloud avec des packages logiciels préinstallés, dont *Apache*, *PHP*, *SQL Server* de Microsoft et *MySQL*. GoGrid est l'un des principaux concurrents d'Amazon en matière de stockage et de calcul dans le Cloud.

2.1.9.6 Rackspace (*IaaS*)

Le Cloud de Rackspace *[IDG 2009]* est également connu sous le nom de « *Mosso* ».

Le fournisseur américain propose une offre IaaS en concurrence avec Amazon et Gogrid.

L'offre se compose de trois grands services:

- *Les sites « Cloud » (une plate-forme de création de sites Web).*
- *Les fichiers « Cloud » (un service de stockage).*
- *Les serveurs « Cloud » (un service similaire à EC2 d'Amazon qui fournit l'accès à des instances de serveurs virtualisés).*

2.1.10 Les solutions de Cloud en France

Quelques fournisseurs en France comme OVH ou Gandi, fournissent une solution Cloud en continuité de leurs services d'hébergement.

Ainsi, OVH permet avec son offre « *minicloud* », d'obtenir de petites configurations rapidement et à moindre coût, basées uniquement sur des systèmes linux *Debian*, *CentOS* ou *Ubuntu*.

Le fournisseur propose également une offre « *DevCloud* » en intégrant l'application *MySQL* puis une offre « *Private Cloud* » pour les entreprises.

Tandis que Gandi est un registraire de nom de domaine et un hébergeur reconnu qui s'est lancé depuis peu dans une offre de machines virtuelles avec un choix varié sur la distribution linux. Les offres sont basées sur l'hyperviseur *Xen* et proposent des solutions de gestions des ressources par une *API XML*.

2.1.11 Le Cloud dans notre quotidien

Le Cloud Computing est perçu aujourd'hui comme quelque chose d'extrêmement puissant, capable de résoudre un nombre d'opérations de calcul largement supérieur à n'importe quel ordinateur moderne. Cette puissance est fournie par des architectures distribuées constituées d'ordinateurs « *low-cost* » qui s'échangent et se distribuent le travail en permanence. Actuellement les experts sont convaincus que bientôt, nous utiliserons le Cloud Computing de la même manière que nous utilisons l'électricité, c'est-à-dire en payant uniquement ce que nous consommons sans même se soucier des aspects techniques nécessaires au bon fonctionnement du système.

2.2 Les systèmes de fichiers distribués

2.2.1 Définition

Un système de fichiers distribué *[Lebre 2002]* permet à plusieurs utilisateurs, d'accéder à une arborescence de fichiers résidant sur une ou plusieurs machines distantes (*Serveurs de fichiers*) tout en utilisant la même syntaxe que pour accéder au système de fichiers locaux.

Il s'agit d'une architecture client/serveur composée d'un système de fichiers proprement dit puis d'un service de gestion. Le premier prend en charge les opérations sur les fichiers, comme la lecture, l'écriture ou l'ajout, alors que le second crée et gère les répertoires (*ajout, destruction de fichiers, parfois appelé service de nommage*).

2.2.2 Système de fichiers

Un système de fichiers est le côté le plus visible d'un système d'exploitation.

Il a pour rôle de stocker, gérer, organiser les données dans un support physique et de définir les méthodes d'accès. Le système de fichiers est composé d'une collection de fichiers qui rassemblent les données puis d'une structure de répertoires qui organise et fournit les informations des fichiers.

2.2.2.1 Un fichier

Le système d'exploitation fournit une vue logique uniforme du stockage afin de faciliter l'utilisation du système informatique. Il fait abstraction des propriétés physiques de ses périphériques de stockage en définissant une unité logique : « *le fichier* ». Les données ne peuvent pas être écrites sur le stockage secondaire sans résider dans un fichier représenté par un programme ou des données d'un format quelconque (*numérique, binaire, texte, etc.*).

2.2.2.2 Types de fichiers

On distingue par ailleurs le fichier texte (*succession de caractères*), le fichier source (*succession de sous-programmes et de fonctions*), le fichier objet (*succession de blocs d'octets*) ou le fichier exécutable (*section de codes exécutables en mémoire*).

2.2.2.3 Attributs de fichiers

Un fichier comporte plusieurs attributs afin de le rendre compréhensible par l'utilisateur et de pouvoir s'y référer. Ainsi, on trouvera le nom, le type, l'emplacement, la taille, les permissions, l'heure et la date de création ou modification, l'identification de l'utilisateur.

2.2.2.4 Structure de fichiers

UNIX définit tous les fichiers comme un simple flux d'octet. Le système de fichiers compacte et décompacte automatiquement les octets en blocs de disques physiques (*par exemple, 512 octets par bloc*). L'espace de disque est toujours alloué en bloc, ainsi une partie du dernier bloc de chaque fichier peut être gaspillée, on parle de fragmentation interne. Plus la taille du bloc est grande, plus la fragmentation interne est importante.

2.2.2.5 Structure des répertoires

Un système d'information est découpé en partitions et chacune d'entre elles contient des informations sur les fichiers à travers une table. Sous UNIX, la structure des répertoires a une forme arborescente avec un répertoire racine. On distingue par ailleurs le chemin absolu qui commence à la racine et se poursuit jusqu'au fichier, tandis que le chemin relatif définit un chemin à partir d'un répertoire courant propre à chaque utilisateur.

2.2.2.6 I-nœuds

Un utilisateur se réfère à un fichier par un nom textuel. Or, sous UNIX, une correspondance est réalisée avec un identifiant numérique (« *i-nœud* ») qui est lui même une interprétation d'une portion d'un disque physique. L'i-nœud contient entre autre, les droits d'accès au fichier, le propriétaire, les adresses des blocs physiques contenant les données, etc. Ces abstractions permettent de cacher à l'utilisateur final les détails de l'implantation physique du fichier sur le périphérique de stockage.

2.2.3 Organisation d'un système de fichiers

Les systèmes de fichiers sont souvent structurés en couches ou modules.

- *Les niveaux inférieurs (périphériques et contrôles des entrées/sorties) s'occupent des caractéristiques physiques des périphériques de stockage.*
- *Les niveaux supérieurs (système de fichiers logiques, programme d'application) se préoccupent des noms des fichiers symboliques et des propriétés logiques des fichiers.*
- *Tandis que les niveaux intermédiaires (système de fichiers de base, module d'organisation de fichiers) établissent une correspondance entre les concepts de fichiers logiques et les caractéristiques des périphériques physiques.*

Pour chaque opération E/S, le fichier peut être recherché dans la structure de répertoires, mais il doit avant tout être ouvert. Lorsque le fichier est trouvé, ses informations sont généralement copiées dans une table en mémoire appelée « *table des fichiers ouverts* ». Son indice dans la table est renvoyé au programme utilisateur et toutes les références ultérieures seront effectuées par l'intermédiaire de l'index (« *l'i-nœud* »).

Tant que le fichier n'est pas fermé, toutes les opérations sur ce fichier sont effectuées dans la table des fichiers ouverts. Lorsque le fichier est fermé, les informations actualisées sont recopiées dans la structure de répertoires reposant sur le disque.

2.2.4 Montage d'un système de fichiers

Un système de fichiers doit être monté avant d'être disponible par les processus du système.
Le système d'exploitation reçoit le nom du périphérique et l'emplacement dans la structure de fichiers auquel il convient d'attacher le système de fichiers. Ensuite, le système vérifie que le périphérique contient un système de fichier valide, en demandant au pilote de lire son répertoire et de vérifier que le format est bien celui escompté. Puis il note dans sa structure de répertoire, qu'un système de fichiers est attaché au point de montage spécifié.

2.2.5 Généralités des systèmes de fichiers distribués

Un système de fichiers distribués (*DFS, Distributed File System*) *[Lebre 2002]* est un système de fichiers sur lequel plusieurs utilisateurs partagent des fichiers et des ressources de stockage. Un DFS devrait apparaître à ses clients comme un système de fichiers conventionnel et centralisé.
L'interface utilisateur d'un DFS ne fait pas de différence entre des fichiers locaux ou des fichiers éloignés.

2.2.5.1 Nommage et transparence

Le nommage est l'établissement d'une correspondance entre les objets logiques (*noms de fichiers*) et les objets physiques (*blocs de données*) stockés sur des pistes de disques.
Un DFS transparent facilite la mobilité des clients en apportant l'environnement de l'utilisateur partout où il se connecte. Ainsi deux niveaux de transparence ont été définis pour caractériser un DFS :

- *Transparence de l'emplacement: le nom du fichier ne comporte aucune information sur son emplacement physique.*
- *Indépendance de l'emplacement: le nom du fichier n'a pas à être modifié lorsque son emplacement physique change.*

Plusieurs approches de nommage sont utilisées dans les DFS afin d'apporter la transparence.
La plus simple consiste à nommer les fichiers par une combinaison du nom de machine et du nom local du fichier, ce qui garantie un nom unique au niveau du système. Une autre méthode, popularisée par NFS, fournit un moyen d'attacher des répertoires distants à des répertoires locaux, donnant ainsi l'apparence d'une arborescence de répertoires classique.

2.2.5.2 Méthode d'accès

Lorsqu'un utilisateur souhaite accéder à un fichier distant, celui-ci envoie une requête au serveur qui stocke le fichier. Un transfert des données répondant à la requête doit se produire. Pour réaliser ce transfert, on utilisera un mécanisme de service à distance et éventuellement une gestion d'un cache de données.

- ***Service à distance***

  Pour le service à distance, une requête d'accès peut être transmise au serveur de façon périodique, à l'ouverture ou à la fermeture d'une ressource. Le serveur effectue les accès et les résultats sont envoyés à l'utilisateur. La requête d'accès utilise les appels de procédures distantes (RPC).

- ***Cache de données***

  Cependant, il est possible d'améliorer les performances du mécanisme de service à distance en utilisant une forme de mise en cache. Dans des systèmes de fichiers locaux, le cache permet de réduire les entrées/sorties de disques. Tandis que dans les DFS, cette mémoire virtuelle permet également de diminuer le trafic réseau. Le principe consiste à réaliser une copie des données faisant l'objet d'une requête en mémoire cache, depuis le serveur vers le client. Les accès sont réalisés sur la copie cachée.

2.2.6 Modèles de DFS

Il existe de nombreux modèles de systèmes de fichiers distribués. Certains reposent sur des standards (*NFS, pNFS, VFS*) et d'autres sont des systèmes propriétaires libres ou payants (*GFS, Hadoop, Lustre, etc.*). Dans cette partie, on étudiera dans un premier temps les systèmes de fichiers standards qui sont des références en la matière. Ensuite on s'intéressera au système de fichiers Rozofs de Fizians, qui est une des briques de construction sur lequel repose notre projet.

2.2.6.1 VFS (Virtual File System)

Virtual File System *[Cochoy 2010]* est le nom commun de différents systèmes de fichiers virtuels fonctionnant sur la famille des systèmes d'exploitation de type UNIX.

Comme tous les systèmes de fichiers virtuels, il s'agit d'une couche logicielle permettant de faire coexister plusieurs systèmes de fichiers concrets de manière transparente pour les processus utilisateurs (*Cf. Figure 07*).

L'architecture de VFS est conçue pour répondre aux contraintes suivantes:

- *L'implémentation du système de fichiers est divisée en deux couches, l'une indépendante des systèmes de fichiers concrets, l'autre dépendante de ces systèmes.*
- *L'architecture doit supporter les systèmes de fichiers UNIX et non UNIX, les systèmes de fichiers avec ou sans état.*
- *Elle doit supporter le côté serveur des systèmes de fichiers distants comme NFS ;*
- *Les opérations doivent être atomiques pour éviter l'usage des verrous.*

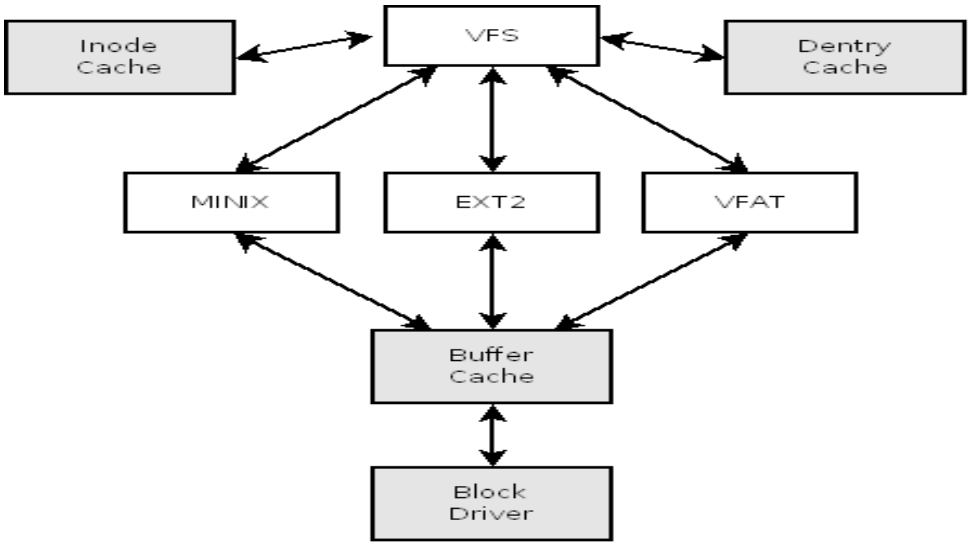


***Figure 07** – Liens entre VFS et divers systèmes de fichiers [Cochoy 2010].*

La recherche d'un nœud dans une arborescence depuis le disque est une opération qui s'avère très coûteuse en performance. Dans certains systèmes de fichiers (*ext2, minix*), il est nécessaire de lire autant de blocs de données sur le disque qu'il existe de nœuds jusqu'au fichier.

De plus, les différents blocs ne sont pas forcément contigus sur le disque et augmentent les accès disques. L'utilisation d'un cache de données permet d'améliorer les performances notamment avec l'utilisation du « *cache d'I-nœuds* » et le « *cache de Dentries* ».

- ***Cache d'I-nœuds***

  Le cache d'i-nœuds est un espace mémoire réservé au stockage des derniers i-nœuds lus afin d'en optimiser la lecture sur le périphérique de stockage. Il s'agit d'une table de hachage unique et commune à tous les systèmes de fichiers, où chaque entrée est un pointeur vers une liste d'i-nœuds qui possèdent la même valeur de hachage.

- ***Cache de Dentries***

  Les Dentries (Directory entries) sont des structures qui contiennent l'association entre un nom de fichier et son numéro d'i-nœud correspondant. Elles sont générées par la fonction « lookup » qui prend en paramètre l'i-nœuds du répertoire parcouru et le nom du fichier/répertoire.

2.2.6.2 NFS (Network File System)

NFS est le premier système de fichiers de réseaux modernes (*construit sur le protocole IP*).

Dans un premier temps, NFS était un système de fichiers expérimental développé en interne par Sun Microsystems au début des années 1980.

Le protocole NFS permet de gérer des fichiers distribués sur plusieurs ordinateurs d'un réseau comme s'ils étaient sur un disque dur local. Ainsi, l'utilisateur n'a plus à se soucier de l'emplacement physique de ses fichiers pour y accéder. La version NFS v3 définie par la RFC 1813 permet de supporter de gros fichiers (*supérieur à 2Go*), les écritures sont en mode asynchrone. En 2000, Sun a introduit la version NFSv4 qui améliore la sécurité et permet d'effectuer des accès parallèles à travers des serveurs distribués (*extension pNFS*).

a) NFS Versus VFS

Tout d'abord, NFS s'appuie sur VFS afin d'ajouter son système de fichiers au système de base *[Jones 2010]*. En effet, dans un environnement Linux, VFS permet de supporter des systèmes de fichiers multiples concurremment avec un système de fichiers de base comme *ext3fs*.

Or, NFS est un système de fichiers à part entière qui doit permettre d'assurer une interopérabilité entre des machines, des systèmes d'exploitation, des architectures de réseaux hétérogènes.

L'indépendance des couches matérielles et logicielles est assurée grâce à des primitives RPC construites au-dessus du protocole de représentation de données externes XDR.

b) RPC et Portmapper

L'appel de fonctions distantes permet la programmation d'applications réparties entre machines distantes de technologies et de systèmes d'exploitation potentiellement différents *[Rossi 2002]*.

Le plus connu est celui développé par SUN (*SunRPC*) qui est devenu un standard.

A l'origine, la société SUN a développé les RPC afin de mettre en place le système de fichiers NFS.

Le principe du protocole RPC est le suivant :

- *Un service réseau est fourni par plusieurs programmes.*
- *Un programme élabore plusieurs fonctions (procédures).*
- *Un client appelle une procédure qui est identifiée par trois entiers (positifs) : le numéro de programmes, le numéro de version du programme et le numéro de procédure au sein du programme.*

Le protocole basé sur l'échange de message décrit en XDR se trouve au-dessus de TCP/UDP du modèle OSI. Il est géré sous Unix par le processus portmapper :

- *Le portmapper mappe les services RPC aux ports qu'il écoute.*
- *Les processus RPC informent portmap quand ils démarrent, enregistrant les ports qu'ils écoutent et les numéros de programmes RPC qu'ils s'attendent à desservir.*
- *Le système client contacte alors portmap sur le serveur avec un numéro de programme RPC particulier.*
- *Le service portmap redirige le client vers le numéro de port correct pour qu'il puisse communiquer avec le service demandé.*
- *Étant donné que les services basés sur RPC dépendent de portmap pour toutes les connexions avec les requêtes entrantes du client, portmap doit être disponible avant qu'un de ces services ne démarre.*

c) XDR

XDR qui est situé au niveau de la couche présentation du modèle OSI, donc au-dessus des RPC (*couche session*), gère la mise en forme des données. Le protocole définit les types utilisés pour l'échange de variables entre le client et le serveur qui ne fonctionne pas forcément sur la même plate-forme. XDR prend soin de convertir le type de données vers une représentation commune afin que toutes les architectures soient interopérables et puissent partager les files systèmes via NFS.

d) Protocole de Montage

Le standard NFS permet de faire un partage des systèmes de fichiers du serveur vers un ou plusieurs clients. Pour réaliser ce partage de manière transparente, il faut que le client effectue un montage NFS du répertoire distant sur un répertoire de son système de fichier local.

Lorsque le serveur reçoit une requête de montage conforme à sa liste d'exportation, il renvoie au client un descripteur de fichier qui sera le point d'accès des requêtes de fichiers du serveur.

Le descripteur est constitué d'un identificateur de système de fichiers et un « *i-nœud* » permettant d'identifier le répertoire monté du système de fichiers exporté.

e) Architecture NFS

Tout d'abord, le client déclenche une opération sur un fichier distant supposé ouvert *(Cf. Figure 08*).

Cette opération se fait par un appel système ordinaire *(« open », « read », « write », « close »)*.

Ensuite, la couche du système d'exploitation du client transforme l'appel sur le « *vnode*[1] » approprié. La couche VFS identifie le fichier comme distant et invoque la procédure NFS appropriée (*« open », »access », « create », « read », « close », »remove », etc.*).

Un appel RPC est effectué afin d'exécuter la procédure sur le serveur puis contacte le service portmap du serveur. Le service portmap fournit au client le numéro de port d'écoute de l'application NFS.

Avant d'envoyer la requête au port TCP indiqué, l'interface XDR du client prend soin de convertir le type de données vers une représentation commune. La requête est alors envoyée au serveur en parcourant la pile NFS qui va de la couche TCP/IP à l'application NFS via RPC/XDR.
Un démon NFS identifie l'arborescence du système de fichiers cible nécessaire pour la requête.
La couche VFS détermine ensuite que le fichier est local et invoque l'opération de système de fichiers appropriée. La consultation du système de fichiers distant est maintenant possible.

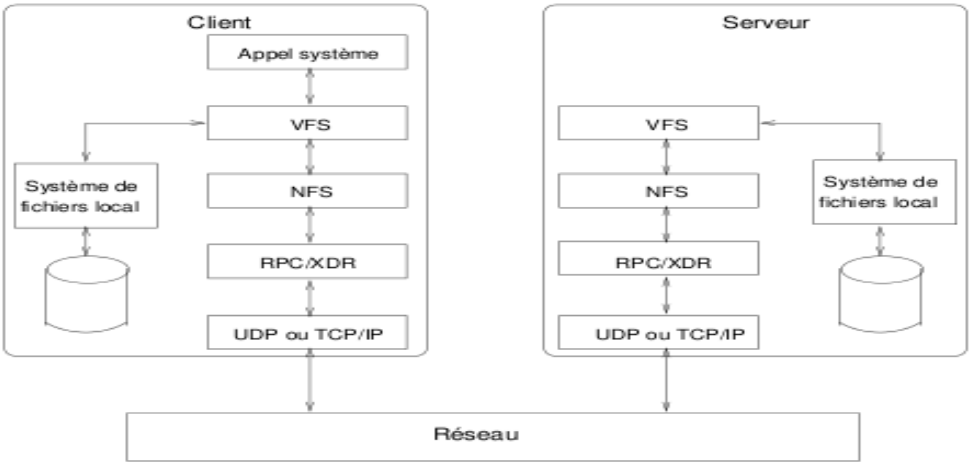


Figure 08 *– Architecture NFS [Jones 2010].*

2.2.6.3 pNFS (parallèle Network File System)

Parallel NFS *[Pnfs 2008]* est une nouvelle norme qui fait partie des spécifications du protocole NFS version 4.1. Ce standard a pour but de répondre aux problématiques de goulets d'étranglement liées à l'utilisation de serveurs uniques. En effet, l'accès aux données partagées est un facteur essentiel pour les performances des serveurs de calculs actuels qui exécutent des applications scientifiques, métiers, etc. Cependant, NFS, qui est la norme la plus utilisée pour partager l'accès aux données, n'est pas adapté pour le partage de données distribuées et encore moins aux calculs répartis. Les serveurs de fichiers se retrouvent vite saturés et freinent le transfère des données.

a) Les intérêts de pNFS

Le protocole pNFS permet aux clients d'accéder directement aux fichiers répartis sur deux serveurs de données ou plus. En accédant de manière parallèle à plusieurs serveurs de données, les clients peuvent bénéficier d'E/S bien plus rapides. D'autre part, le nombre de clients peut être étendu pour fournir plus de puissance de calcul. De la même manière, la taille du système de stockage peut s'étendre avec peu d'impact sur la configuration du client.

b) L'architecture pNFS

L'architecture pNFS s'articule autour de trois composants principaux (*Cf. Figure 09*) :

- *Le **serveur de méta-données** (MDS) est chargé de maintenir à jour les informations concernant la structure des fichiers ainsi que la distribution physique des sous-fichiers sur les différents disques. Il assure la maintenance des méta-données, lesquelles décrivent l'emplacement et le mode de stockage de chaque fichier.*
- *Les **serveurs de données** stockent les données des fichiers et répondent directement aux demandes de lecture et d'écriture de la part des clients. Les données des fichiers peuvent être réparties sur plusieurs serveurs de données.*
- *Un ou plusieurs **clients** peuvent accéder aux serveurs de données directement sur la base des informations contenues dans les méta-données.*

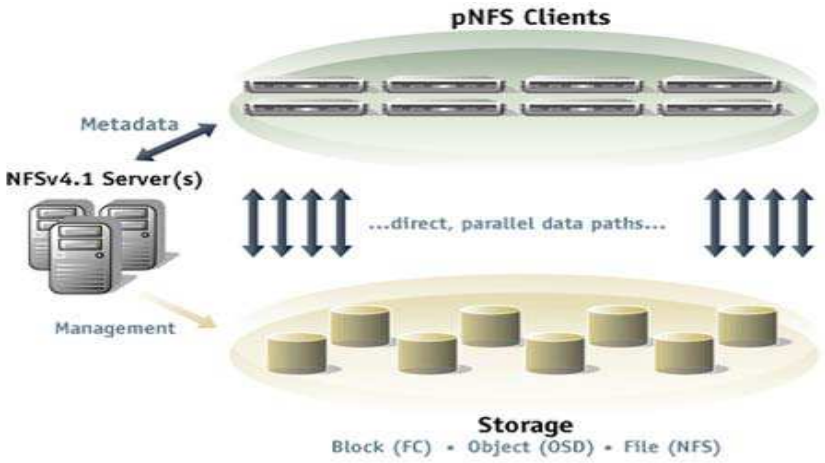


***Figure 09** – Architecture PNFS [pnfs 2008].*

c) Les protocoles pNFS

Trois protocoles sont utilisés entre les clients, le serveur de méta-données et les serveurs de données :

- *Un **protocole de contrôle** est utilisé entre le serveur de méta-données et les serveurs de données pour assurer la synchronisation.*
- *Le **protocole pNFS** est utilisé entre les clients et le serveur de méta-données. Il s'agit essentiellement du protocole NFS version 4 auquel s'ajoutent des extensions spécifiques à pNFS. Ce protocole sert à récupérer et manipuler les agencements des fichiers constituant les méta-données. Il indique l'emplacement des fichiers stockés et le protocole d'accès au stockage requis sur différents serveurs de données.*
- *Un ensemble de **protocoles d'accès au stockage** est utilisé par les clients pour accéder directement aux serveurs de données.*

d) Principe d'utilisation

Le serveur pNFS exporte les systèmes de fichiers et maintient les informations sur le serveur de méta-données *(Cf. Figure 10*). Le client monte le système de fichiers exporté et le gère comme un système de fichiers local. Contrairement à NFS, une lecture ou écriture des données avec pNFS se fait directement entre un nœud (*client*) et l'espace de stockage. Ainsi, le serveur pNFS ne prend plus en charge les transactions.

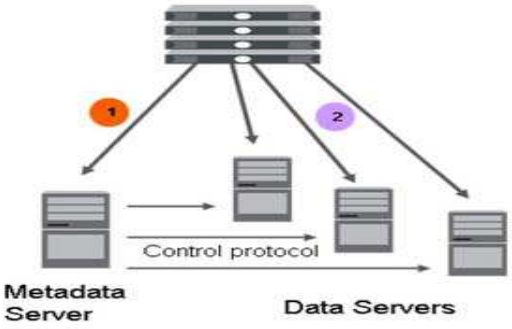


Figure 10 *– Exemple d'accès aux fichiers [Eisler 2008].*

Pour accéder à un système de fichiers dans un espace de stockage, un client contacte le serveur de méta-données. Ce dernier lui envoie, après autorisation un fichier de méta-données contenant les informations nécessaires pour déterminer l'emplacement de chaque « *morceau* » d'un fichier, la manière d'y accéder. Il précise également le protocole d'accès aux stockages devant être utilisé.

Le client utilise ces informations pour procéder directement aux E/S vers et depuis les serveurs de données, de manière parallèle, en utilisant le protocole d'accès aux stockages, sans devoir recourir à nouveau au serveur de méta-données. Si le client modifie le système de fichiers partagés, alors toutes les modifications sont validées en arrière-plan sur le serveur de méta-données. Quand le client n'a plus besoin du système de fichiers partagés, les dernières modifications sont prises en compte sur le fichier de méta-données et une copie est retournée au serveur de méta-données. Le système de fichier est alors fermé. Une opération d'écriture est similaire sauf que le client doit envoyer une requête supplémentaire pour signaler les changements du fichier au serveur pNFS avant de fermer le système de fichiers.

Les données, dans l'espace de stockage, sont agencées de manière différente selon le protocole de stockage mis en œuvre. Cette flexibilité est assurée grâce à la couche VFS qui permet d'ajouter différents systèmes de fichiers. Ainsi, les données peuvent être organisées par fichiers, par blocs ou par objets.

- *Avec un agencement par fichiers, deux clients pNFS différents peuvent accéder à la même zone logique du même fichier en lecture ou en écriture.*
- *Avec un agencement par blocs ou par objets, un seul client pNFS à la fois peut s'approprier un agencement inscriptible correspondant à une zone ou un fichier.*

2.2.6.4 RozoFS

Le système de fichiers RozoFS est un système de fichiers distribués open-source développé par la société Fizians *[Fizians 2011]* depuis 2010. L'architecture physique sur laquelle repose le système *Rozo* est constituée des trois éléments suivants (*Cf. Figure 11*) :

- *Un serveur de méta-données sur lequel fonctionne le démon « **exportd** ».*
- *Un ou des serveur(s) de stockage sur lesquels fonctionne le démon « **storaged** ».*
- *Un ou des client(s) qui utilisent « **rozofsmount** ».*

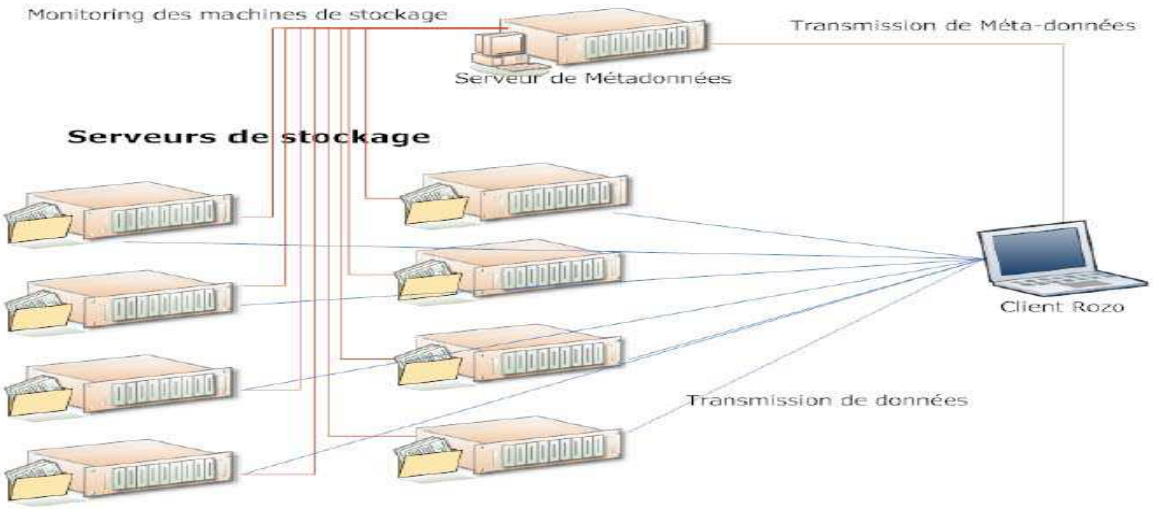


***Figure 11** – Architecture Rozo [David et al. 2011].*

Le serveur de méta-données regroupe l'ensemble des informations nécessaires pour que le client puisse accéder au fichier. Les serveurs de stockage conservent les blocs de données permettant de reconstruire le fichier. Le découpage du fichier en blocs utilise la transformée Mojette élaborée notamment par mon tuteur de stage JeanPierre Guédon.

a) La transformée Mojette

La transformée Mojette *[Guédon 2009],* de la famille des codes correcteurs a été mis au point en 1995 par Messieurs JeanPierre Guédon et Nicolas Normand, membres tous deux de l'équipe IVC du laboratoire IRRCyN. Le principe de la transformée Mojette repose sur une représentation des données sous forme de matrice (*Cf. Figure 12*) sur laquelle on effectue des projections suivant les angles souhaités. L'objectif est de reconstruire le fichier d'origine à partir d'un ensemble de projections.

5	3	4
6	2	3
1	3	6

***Figure 12** – Matrice pour la transformée Mojette.*

Les projections qui apparaissent autour de la matrice (*Cf. Figure 13*), sont représentées par des chiffres déterminés en effectuant la somme des valeurs dans la matrice selon un angle d'orientation (*matérialisé sur la figure par les lignes en pointillées*).

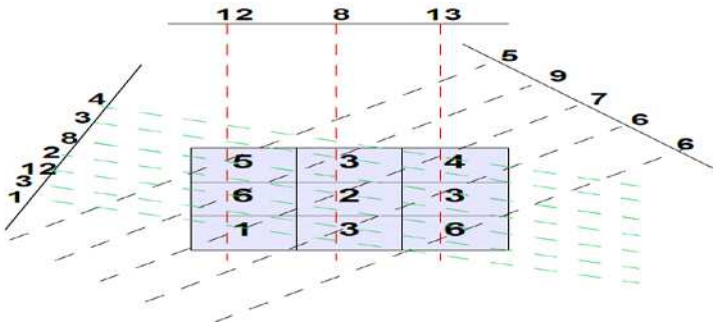


Figure 13 *– projections sur une matrice.*

Les trois angles de projections sont définis arbitrairement, il peut, par ailleurs, en avoir une infinité.

- *Pour l'angle de gauche, la somme s'effectue en ajoutant un chiffre puis celui en dessous, décalé de deux.*
- *L'angle au-dessus totalise les chiffres verticalement.*
- *Tandis que l'angle à droite, effectue la somme en diagonale.*

L'avantage de cette transformée est de fournir un espace de représentation bijectif avec l'espace standard. En effet, si on conserve un nombre suffisant de projections on est capable de reconstruire de façon unique le contenu originel. De plus il a été montré que contrairement à de nombreux codes correcteurs, il n'est pas nécessaire de récupérer les projections dans l'ordre pour reconstruire le contenu originel. Il suffit d'avoir un minimum de projections pour le reconstruire indépendamment de l'ordre. Par ailleurs, le fait d'avoir davantage de projections avec des angles différents, permet d'avoir des projections supplémentaires qui ajoutent de la redondance facilement.

De plus comme il est possible de reconstruire un fichier, indépendant de l'ordre des projections, il n'est pas nécessaire de renforcer la fiabilité sur une projection spécifique puisqu'elles sont toutes équivalentes.

b) Les projections

Les projections sont réalisées par l'application « *Rozofsmount* » via la transformée Mojette qui va découper un fichier en blocs de données binaires. Avec ces projections, il est ainsi possible de reconstituer un bloc de données et à plus large échelle, de rétablir entièrement le fichier.

L'ordre des projections pour reconstruire le fichier est aléatoire. Chaque projection fait référence à un fichier spécifique par l'intermédiaire d'un identifiant unique. Un fichier est composé d'un certain nombre de projections qui assurent le degré de redondance énoncé précédemment.

Par exemple, pour 12 projections, il en suffit de 8 pour reconstruire le fichier.

Les projections sont situées dans les serveurs de stockages qui sont déclarés dans le fichier de configuration du serveur de méta-données.

c) *Le serveur de méta-données*

Le serveur de méta-données est chargé de maintenir à jour les informations concernant la structure des fichiers (*nom complet, taille, identifiant*) ainsi que la liste des points de stockage contenant les projections nécessaires pour reconstruire le fichier.

Pour chaque fichier à reconstruire, le serveur stocke les méta-données dans un emplacement déclaré dans le fichier de configuration « *export.conf* » (*Cf. Figure 14*). Ce fichier liste également chaque serveur de stockage avec un identifiant (*SID*) ainsi que leur nom (*IP ou Hostname*) sur le réseau. L'ensemble des serveurs déclarés dans le fichier, constitue un « *cluster*» de stockage.

```
layout = 0;
volume = (
  {
    cid = 9;
    sids = (
      {
        sid = 27;
        host = "10.194.189.215";
      },
      {
        sid = 25;
        host = "10.242.201.80";
      },
      {
        sid = 26;
        host = "10.245.197.233";
      },
      {
        sid = 24;
          host = "10.242.61.108";
      } );
  } );
exports = (
  {
    eid = 0;
    root = "/srv/rozofs/export9";
  } );
```

Figure 14 *- Exemple de fichier de configuration du serveur de méta-données.*

Le degré de redondance est configurable avec le « *Layout* » de telle sorte que :

- *Si « Layout = 0 », pour un nombre de 4 « storages », il suffit d'en contacter 3 pour reconstruire le fichier.*
- *Si « Layout = 1 », pour un nombre de 8 « storages», il suffit d'en contacter 6 pour reconstruire le fichier.*
- *Si « Layout = 2 », pour un nombre de 12 « storages », il suffit d'en contacter 8 pour reconstruire le fichier.*

Cette propriété de redondance vient directement de l'exploitation de la transformée Mojette incluse dans l'application cliente de « *Rozo* » (*Rozofsmount*).

d) Les serveurs de stockages

Les serveurs contiennent une ou plusieurs projections d'un ou plusieurs fichiers.

Le serveur de méta-données sélectionne les serveurs de stockages déclarés dans le fichier de configuration. Ces serveurs sont destinés à contenir la ou les projections, en fonction de l'espace disponible qui leur reste. Chaque serveur de stockage se configure à l'aide du fichier « *storage.conf* » (*Cf. Figure 15*) qui liste un ou plusieurs points de stockage contenant les projections.

```
layout = 0;
storages = (
  {
    sid = 27;
    root = "/mnt/rozo/storage";
  } );
```

Figure 15 *- Exemple de fichier de configuration du serveur d'un serveur de stockage.*

Un « *cluster* » de stockage regroupe un ensemble de serveurs de stockage qui interagissent avec le même serveur de méta-données. Le nombre de serveurs contenu dans chaque cluster dépend du degré de redondance choisi (« *Layout* »).

e) Opération de création de fichier

Le client réalise un point de montage à l'emplacement qu'il souhaite et accède à une arborescence partagée avec le serveur de méta-données.

Or, pour permettre à un utilisateur d'effectuer un point de montage, la solution « *RozoFS* » s'appuie sur l'application « *Fuse* » qui permet à un utilisateur, un programme, sans privilèges particuliers, de monter des systèmes de fichiers.

Ainsi, l'utilisateur qui souhaite stocker un fichier, envoie une requête au serveur de méta-données pour lui demander la liste des serveurs de stockage qu'il doit utiliser. Ce dernier vérifie au passage que le fichier n'est pas déjà créé puis sélectionne les serveurs de stockage qui contiennent le plus d'espace libre. Pour cela, à intervalle régulier, le serveur de méta-données interroge chaque serveur de stockage pour s'assurer d'une part qu'il est bien présent sur le réseau et d'autre part, afin de connaitre la volumétrie d'espace disque restant. Le serveur de méta-données retourne une sélection de serveurs de stockage vers l'application cliente « *Rozofsmount* ». Cette dernière segmente alors le fichier par blocs de 8Ko qui seront transformés puis envoyés aux différents serveurs de stockage figurant dans la liste que retourne le serveur de méta-données. Lors de la transformation des blocs, une part de redondance est ajoutée pour fournir une plus grande disponibilité au système.

A chaque envoi de bloc, le client reçoit un accusé-réception de la part du serveur de stockage.

Le client informe alors le serveur de méta-données que tel numéro de bloc appartient à tel serveur de stockage. Il procède de la même manière pour tous les blocs constituant le fichier.

En parallèle, le serveur de méta-données enregistre ces informations dans une liste.

Cette liste sera nécessaire lors d'une opération de lecture afin de reconstruire le fichier chez le client.

f) Opération de lecture de fichier

Lors de l'opération de lecture, le fonctionnement inverse est effectué.

Le client, qui souhaite reconstruire un fichier à partir de son identifiant unique (*Cf. Figure 16*), interroge le serveur de méta-données pour obtenir la liste des serveurs de stockage concernés.

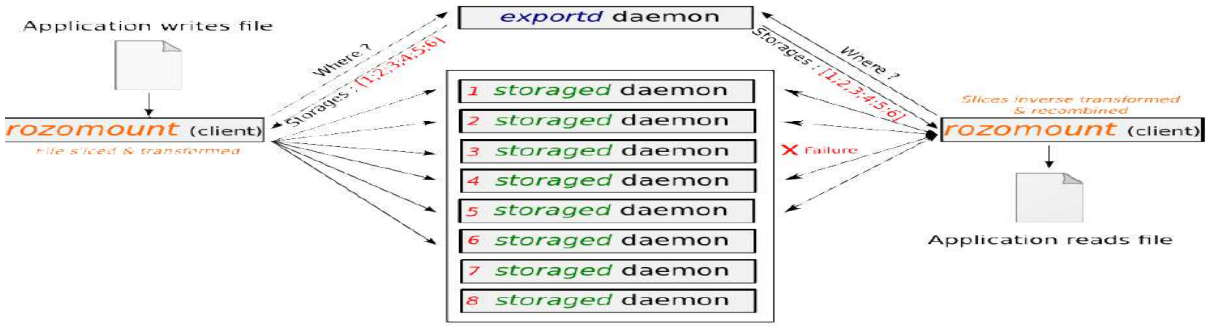


Figure 16 *– Principe d'accès à un fichier entre un client et l'architecture Rozo [David et al. 2011].*

Une fois la liste des serveurs de stockage en possession, le client communique avec chacun d'entre eux pour récupérer la projection associée au fichier demandé. L'application reconstruit alors le fichier et permet son accès en modification à l'utilisateur.

g) Les avantages de Rozofs

Le système de fichiers « *Rozofs* » permet de reconstruire un fichier avec un nombre suffisant de projections (*au minimum 2/3*). Les projections sont stockées dans des machines différentes et inexploitables individuellement, ce qui garantit plus de sécurité et de redondance dans le stockage des données. En répartissant ces projections localement et à distance, on évite de faire une réplication supplémentaire des données tout en économisant de l'espace disque en comparaison à une solution classique de stockage réparti.

2.3 Les architectures Web

Tout d'abord, avant d'aborder les technologies émergentes du Web qui ont permis de créer de nouveaux concepts comme le Cloud Computing, il convient d'en décrire les principes fondamentaux. Ainsi, cette partie s'attache à définir le Web d'aujourd'hui avec les technologies responsables de son évolution *[Atelier 2008]*. On parlera également des applications Web comme les architectures orientées services, les services Web, les outils nécessaires pour créer ce type d'infrastructure.

2.3.1 Les évolutions du Web

Le Web a très largement évolué depuis une décennie entre les premiers langages du Web 1.0 (html) et les langages plus évolués du Web 3.0 (*Web sémantique*) (*Cf. Figure 17*).

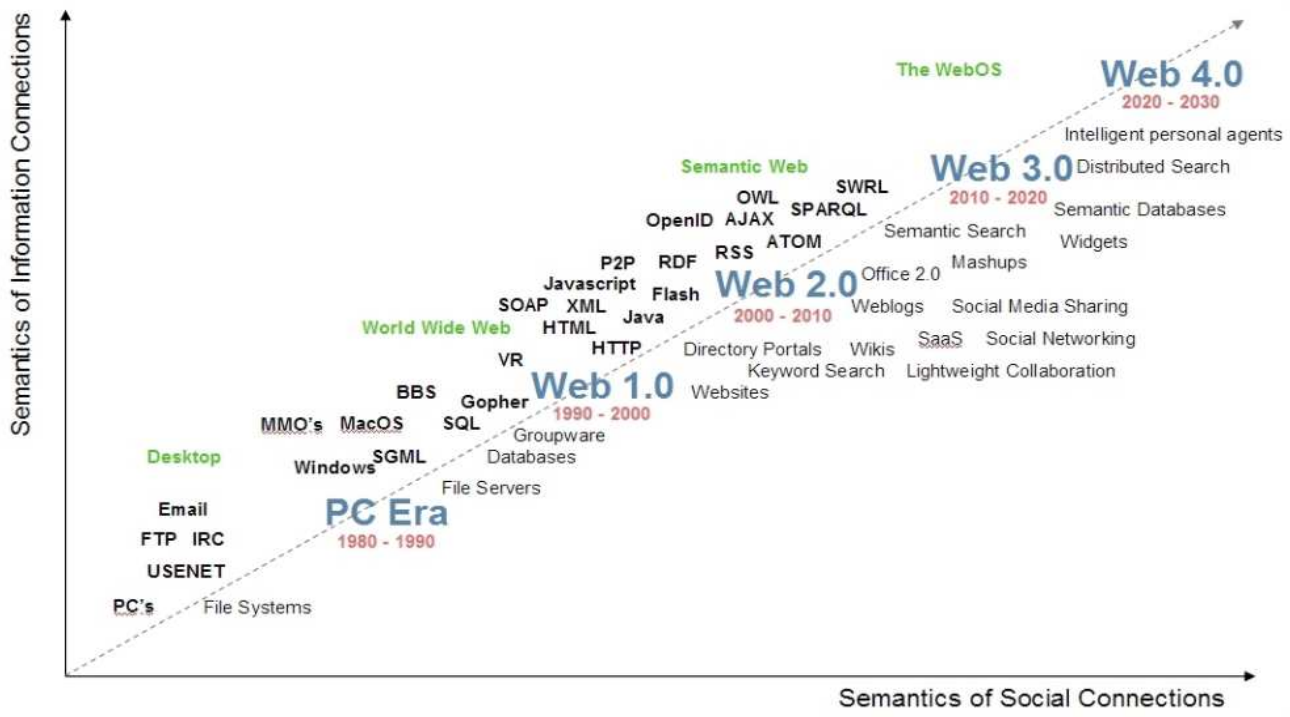


Figure 17 *- Evolution du Web [Spivack 2009].*

2.3.1.1 Le Web 1.0

Le Web 1.0 qui est caractérisé par des pages Web statiques rarement mises à jour, a évolué lentement vers un Web dynamique utilisant des systèmes de gestion de contenus. Il est considéré principalement comme un outil de diffusion et de visualisation de données. Une première évolution fut réalisée par des solutions se basant sur un Web dynamique (*parfois appelé Web 1.5*), où des systèmes de gestion de contenus alimentaient des pages Web dynamiques, créées à la volée à partir d'une base de données.

2.3.1.2 Le Web 2.0

Le concept du Web 2.0 est apparu lors d'une conférence « *brainstorming* » organisée par *O'Reilly* et *Medialive International*. Le terme « *Web 2* » est à l'initiative de Tim O'Reilly dont il en explique le sens et la portée à travers le texte fondateur « *What Is Web 2.0* » *[O'Reilly 2009]*.

Selon Tim O'Reilly, le Web 2 est similaire à une plate-forme à part entière et les applications doivent disparaître au profit de services Web équivalents et gratuits. Les applications deviennent ainsi des services nomades, accessibles partout dans le monde sur Internet.

Le Web 2.0 désigne un Web collaboratif où les utilisateurs peuvent interagir avec le contenu des pages, mais aussi entre eux. Le Web 2 donne naissance aux blogs, aux flux RSS, aux réseaux sociaux, aux partages de contenus, etc. Les technologies comme Ajax permettent de modifier le contenu des pages sans les recharger. Ainsi, avec le Web 2, les utilisateurs sont à la fois les contributeurs et les bénéficiaires. On est passé du moyen-âge du Web à une guerre de mouvement, où le contenu tend à devenir plus important que le site qui le porte.

2.3.1.3 Le Web 3.0

Le Web 3.0 qui est en train de naître, correspond à un nouveau stade de cette évolution.

On parle de Web sémantique, car on peut retrouver n'importe quelle information utile à partir de quelques fragments significatifs de celle-ci, indépendamment du site qui l'héberge.

Aujourd'hui, la définition du Web 3 est encore assez floue bien que tout le monde s'accorde à dire que c'est un Web déstructuré. Il ne sera plus nécessaire d'aller sur Internet, c'est Internet qui viendra à nous.

2.3.2 Les méthodes de développement Web

Toutes les applications Web reposent sur des langages de développement nécessaire pour leur mise en œuvre. Cependant, l'architecture Web nécessite de choisir le langage de programmation adapté. Certains font l'objet de standard et ne dépendent pas de la solution adoptée tandis que d'autres font l'objet d'une utilisation spécifique.

2.3.2.1 Le langage de développement

Un langage informatique décrit l'ensemble des actions consécutives qu'un ordinateur doit exécuter.

Le code source subit ensuite une transformation ou une évaluation dans une forme exploitable par la machine, ce qui permet d'obtenir un programme. Suivant le langage utilisé, un programme doit être interprété ou compilé en passant par un autre langage qui pourra être compris par la machine (*l'assembleur ou même en code binaire*). D'un point de vue pratique, un langage de programmation permet l'écriture de programmes de manière compréhensible par un être humain. Le programmeur n'a pas besoin de connaître le langage machine, dit « *de bas niveau* ».

2.3.2.2 Les types de langages

Les langages informatiques peuvent se classer en trois catégories: les langages interprétés, les langages compilés, les langages intermédiaires *[Gonzalez 2008]*.

a) Langage interprété

Un programme écrit dans un langage interprété a besoin d'un programme auxiliaire (*l'interpréteur*) pour traduire au fur et à mesure les instructions du programme.

Exemple : HTML, XML, PHP, ASP, Ruby, SQL, Oracle, etc.

b) Langage compilé

Un programme écrit dans un langage dit « *compilé* » va être traduit une fois pour toutes par un programme annexe *(le compilateur*) afin de générer un nouveau fichier autonome qui n'aura pas besoin de programme annexe pour s'exécuter. Bien que plus rapide à s'exécuter qu'un programme interprété, il est en revanche moins souple, car chaque modification du fichier source nécessite de recompiler le programme pour que les modifications prennent effet.

Exemple : C, C++, Cobol, Fortran, Ada, etc.

c) Langage intermédiaire

Certains langages appartiennent en quelque sorte aux deux catégories car le code source peut dans certaines conditions subir une phase de pré-compilation. Le code source est alors converti en fichier intermédiaire proche du langage machine et non exécutable. L'interpréteur se chargera ensuite de traiter le nouveau fichier d'une manière beaucoup plus rapide, car plus compréhensible pour la machine.

Exemple : Java, Python, C#.

2.3.3 Typologie et mise en œuvre des pages Web

Une page Web est un fichier textuel créé par un langage de développement Web (*HTML, XML*).
Ces langages permettent de développer des pages Web statiques.
Cependant, d'autres langages interprétés ou compilés permettent de rendre les pages dynamiques notamment pour assurer une interaction dans les échanges de données, améliorer le contenu, etc.

2.3.3.1 Serveur Web

Un serveur Web permet de fournir aux navigateurs Web, les fichiers nécessaires à l'affichage des pages Web. Le serveur Web repose sur trois standards (*HTTP, HTML et URL*) développés par Timothy John Berners-Lee en 1990 au CERN *[Wikipedia 2011]*.

- ***HTTP :***
  Il s'agit d'un protocole de communication client-serveur situé sur la couche application du modèle OSI et permet d'échanger des documents écrits en HTML.
  Ces pages Web sont identifiées par des URL et échangées avec des clients HTTP matérialisés par les navigateurs Web.
- ***URL :***
  Il s'agit d'une chaîne de caractères utilisée pour identifier de manière unique, un document sur Internet. Les URL ou adresses Web permettent aux navigateurs Web d'accéder à toutes les ressources Internet de manière uniforme.

Ainsi, le serveur Web échange des pages Web statiques ou dynamiques avec les clients.
La typologie de la page dépend du langage de développement Web employé pour la conception.

2.3.3.2 Page Web statique

Une page web statique est constituée d'un fichier texte contenant du code HTML et éventuellement des images et des liens vers d'autres documents. Cependant, elles seront toujours représentées de la même façon, ce qui engendre les inconvénients suivants :

- *Modification manuelle du contenu.*
- *Impossibilité d'adapter le contenu selon le visiteur.*
- *Pas d'interactivité possible (notamment avec des bases de données).*
- *Contenu vieillissant.*

En revanche, comme il n'y a pas de traitement du contenu par le client ou le serveur, l'accès à la page est plus rapide.

2.3.3.3 Page Web dynamique

Pour répondre aux problèmes d'interactivité entre le client et le serveur, il est possible de créer des pages Web dynamiques via des langages adaptés (*PHP, ASP, JAVA, etc.*).

Les pages dynamiques permettent de mettre à jour dynamiquement une page Web avec souvent l'utilisation d'une base de données. En revanche, l'accès à la page est plus lent à cause des traitements des requêtes et peuvent poser des problèmes d'incompatibilités avec les clients.

Il y deux façons d'échanger une page Web dynamique, soit du côté client ou bien du côté serveur.

2.3.4 Le langage de script avec Python

Un langage de script est un langage de programmation interprété qui permet de manipuler les fonctionnalités d'un système informatique configuré via un environnement de travail.

Le langage de script peut alors s'affranchir des contraintes de bas niveau.

Dans le sens le plus traditionnel, qui est celui des « *Shell scripts* », un script sert principalement à lancer et coordonner l'exécution de programmes dans un ordre défini.

Il existe aujourd'hui une infinité de langages scripts utilisés dans des applications en tout genre.

On les retrouve notamment dans la conception de pages dynamiques (*PHP, JavaScript*), dans la manipulation de fichiers ou de textes (*Perl*), dans l'automatisation de tâches (*Python*), etc.

2.3.4.1 Description du langage Python

Python est un langage portable (*Unix, Linux, Windows, Mac OS*), orientée objet, dynamique, extensible et permet une approche modulaire *[Lutz 2009]*.

Il fonctionne selon un typage dynamique fort, dispose d'une gestion automatique de la mémoire et d'un système de gestion d'exceptions.

Python a été développé en 1989 par Guido van Rossum et poursuit son évolution avec de nombreux contributeurs bénévoles. Il est libre d'utilisation et de distribution avec une forte communauté en ligne qui répond à toutes demandes des utilisateurs.

a) Types offerts par le langage

Python offre plusieurs types de données natifs avec tout d'abord les types numériques de base (*int, long, float, complex*) mais également un ensemble de types sur lesquels il est possible d'itérer :

- *List : Une liste d'objets de taille variable de type non fixe.*
  Exemple : [« a », 123]
- *Tuple : Une variante de liste non modifiable.*
  Exemple : t = (« a », 123)
- *Str : Une chaîne de caractères.*
- *Dict : Dictionnaire mettant en relation un objet (la clé) avec un autre (la valeur).*
  Exemple : dict = {« key1 » : « value1 »}

b) Modules, classes et méthodes

Pour Python, un module est un simple fichier de code contenant une ou plusieurs classes.

Les modules externes peuvent être importés de deux manières dans un script :

- *La première consiste simplement à indiquer le fichier qui contient les méthodes.*
  Exemple : ***import Module***
- *La deuxième permet d'importer directement les classes et méthodes dans l'espace de nom local.*
  Exemple : ***from Module import Classe***

En Python, tout est objet, que ce soit les classes, les instances de classes, les fonctions, etc.

De plus, Python supporte l'héritage multiple, mais ne permet pas de surcharger les opérateurs.

Une classe se définit avec le mot-clé « *class* ». Les méthodes de la classe se construisent de la même manière que des fonctions indépendantes (*Cf. Figure 18*).

```
class Personne:
    def __init__(self, age, sexe):
        self.age = age
        self.sexe = sexe

    def getAge(self):
        return self.age

Toto = Personne(36, 'homme')
print Toto.getAge()
```

Figure 18 *– Exemple de code Python [Lutz 2009].*

Le paramètre « *self* » représente l'instance de la classe. La fonction « *__init__()* » correspond au constructeur de la classe.

2.3.4.2 Les intérêts de Python

Tout d'abord, le langage Python est simple d'utilisation et particulièrement adapté à la réalisation de scripts. Par exemple, un programme Python est souvent trois à cinq fois plus court qu'un programme C ou C++, ce qui représente en général un temps de développement de cinq à dix fois plus court et une facilité de maintenance largement accrue. D'autre part, Python est portable, non seulement sur les différentes variantes d'Unix, mais aussi sur les OS propriétaires :

- *Linux et systèmes Unix.*
- *Microsoft Windows et DOS.*
- *Mac OS (OS X et Classic).*
- *BeOS, OS/2, VMS, et QNX.*
- *système temps réel tel que VxWorks.*
- *PDA exécutant Palm OS, PocketPX.*
- *Téléphone cellulaire exécutant Symbian OS et Windows Mobile.*
- *Console de jeux et iPods.*

Le langage intègre, comme Java ou les versions récentes de C++, un système d'exceptions qui permet de simplifier considérablement la gestion des erreurs.

De plus, c'est un langage interprété car il est directement exécuté sans passer par une phase de compilation comme c'est le cas pour le langage C.

Cependant, il s'exécute moins rapidement qu'un programme C car la traduction à la volée du programme ralentit l'exécution. Mais le temps d'exécution est suffisamment rapide pour la plupart des applications car bien souvent, le programme Python invoque des routines en C.

En revanche, si la vitesse est un élément important, Python offre la possibilité d'ajouter des outils de compilation (*Psyco, Pyrex, Shed Skin, etc.*).

Aussi, la bibliothèque standard de Python et les paquetages ajoutés, donnent accès à une grande variété de services comme les chaînes de caractères et les expressions régulières, les services UNIX standards (*fichiers, pipes, signaux, sockets, threads, etc.*), les protocoles Internet (*Web, News, FTP, CGI, HTML, etc.*), les bases de données, les interfaces graphiques. Un script Python peut facilement communiquer avec différentes parties d'une application grâce à une variété de mécanismes d'intégration.

Un programme Python peut également intégrer Java et des composants NET, communiquer à travers des sockets, transférer des fichiers par FTP, générer des fichiers XML, communiquer via XML-RPC, SOAP, Telnet, etc.

2.3.4.3 Le fonctionnement de Python

Lorsqu'on exécute un programme (f*ichier .py*), Python compile d'abord le code source en « *bytecode* ».

Le « *bytecode* » est un code intermédiaire plus concret (*plus proche des instructions de la machine*) que le code source, il n'est pas directement exécutable. Il est contenu dans un fichier binaire qui représente un programme, tout comme un fichier objet produit par un compilateur.

Il est nommé « *bytecode* » du fait de son format où chaque instruction est codée en binaire, ce qui permet d'être exécuté plus rapidement que le code source original.

Le fichier compilé porte l'extension « *.pyc* ». Le programme contrôle régulièrement les modifications apportées sur le fichier source et le recompile le cas échéant (*après un nouvel enregistrement*).

Le programme compilé sera ensuite envoyé vers la machine virtuelle Python (« *PVM* »).

Cette PVM qui fait partie du système Python, est le moteur qui va exécuter le code (« *Runtime* »).

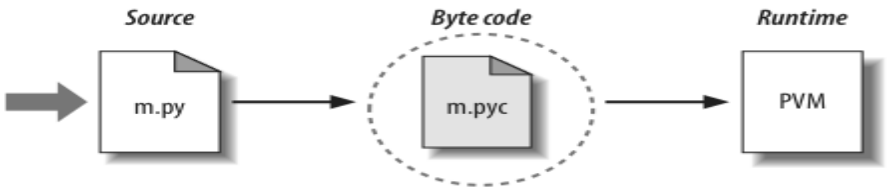


Figure 19 *- Exécution du code sous Python [Lutz 2009].*

Le compilateur est toujours associé à un moteur d'exécution et forme un unique système.

Le cycle de développement est plus rapide, il n'y a pas d'étape de pré-compilation.

2.3.4.4 Les différentes mises en œuvre de Python

Il y a trois principales implémentations de Python selon le langage de programmation souche.

On trouve dans un premier temps la mise en œuvre classique de Python intitulée « *Cpython* ».

Il s'agit de l'utilisation normale et originale du langage Python. Le « *C* » vient directement du langage *C ANSI* qui a permis de mettre en œuvre Python.

Ensuite on trouve une implémentation de Python avec le langage de programmation Java.

Cette mise en forme se prénomme « *Jython* », il s'agit d''une alternative par rapport à la version originale. « *Jython* » est constitué de classes Java qui compilent le code source Python en « *bytecode* » Java. Ce « *bytecode* » sera ensuite exécuté par la machine virtuelle Java (*JVM*).

L'objectif de « *Jython* » est de permettre au code Python de créer des applications JavaScript pour réaliser notamment des applets ou servlets Java.

Puis une troisième alternative propose d'utiliser Python dans un environnement Microsoft.

Il s'agit de « *IronPython* » qui a été développé en « *C sharp* » (*c# est un langage de programmation objet créé par Microsoft*). Le langage « *IronPython* » permet au programme Python de s'intégrer avec des applications basées sur le framework « *.NET* » de Microsoft ou Mono (*équivalent de .NET sur Linux*). Il intéresse donc les développeurs qui souhaitent intégrer Python avec les composants « *.NET* ».

2.3.5 Les cadriciels Python

2.3.5.1 Un cadriciel

L'expression « *cadriciel* » *[Wikipédia 2011]* désigne un ensemble de composants mis à la disposition du développeur. On parle plus exactement de « *boîtes à outils* », de « *cadre d'application* » ou bien de « *framework* ».

Les éléments qui se révèlent utiles et fréquemment employés sont ajoutés au fur et à mesure dans la boîte à outils. Un cadriciel désigne également, dans le domaine des interfaces graphiques, une bibliothèque de « *widgets* », c'est-à-dire de composants graphiques de conception et apparence homogène.

Le plus souvent, chaque bibliothèque est dédiée à un domaine particulier et forme un tout cohérent (*Exemple : Curl, Dom XML, J2EE, etc.*).

On parle d'API (*Application Programming Interface*) ou interface de programmation pour désigner les spécifications des composants de ces bibliothèques.

Ainsi, un framework, ou cadre d'application, fournit à travers ses API un cadre pour développer un type d'application.

Par exemple, pour les applications Web, les frameworks les plus connus se basent sur le modèle MVC (*modèle, vue, contrôleur*). Ainsi, boîtes à outils, bibliothèques et frameworks visent tous à simplifier le code des applications en réutilisant des briques existantes.

2.3.5.2 Exemples de cadriciels Python

Le langage Python peut être utilisé avec d'autres technologies pour créer des sites Web ou interfaces d'applications Web. Parmi les frameworks disponibles, on peut citer *Django, TurboGears, Zope, Web 2py, Pylons, etc.*

2.3.5.3 A propos du cadriciel Django *[Django 2011]*

Django est un framework développé en 2003 avec le langage Python par Adrian Holovaty et Simon Willison pour un petit journal local du Kansas « *Lawrence* ».

En 2005, l'équipe de développement décide de rendre le framework open source et de nommer la première version publiée « *Django* » en hommage au guitariste Jazz Django Reinhardt.

Bien que ce framework bénéficie de contributeurs à travers le monde, les développeurs originaux continuent de fournir une direction pour le développement du framework.

Comme tous les frameworks, le but est de développer rapidement des sites pouvant être complets.

Ainsi, Django permet de faciliter la mise en œuvre de sites Web complexes reliés à des bases de données. Il peut être considéré comme une boite à outils où chaque module peut fonctionner de façon indépendante. Ce framework est utilisé notamment pour la plate-forme de développement « *Google App Engine* ».

a) Django et le modèle MVC

Django s'inspire du modèle MVC qui définit un mode de développement de logiciels stipulant que le code de la définition et de l'accès aux données (*le modèle)* soit distinct de la demande de routage logique (*le contrôleur*), qui est à son tour séparée de l'interface utilisateur (*la vue*).

L'architecture MVC *[Plouin et al. 2008]* est l'architecture de référence des applications composites interactives. Elle permet d'isoler d'une part un modèle métier, d'autre part des vues interactives rendues par un moteur graphique sur de l'information en provenance du modèle et des contrôleurs de dialogues. Ce modèle d'architecture impose la séparation entre les données, la présentation et les traitements, ce qui donne trois parties fondamentales dans l'application finale : ***le modèle, la vue et le contrôleur***.

- *Le modèle :*

  Il représente le comportement de l'application. Il décrit ou contient les données manipulées par l'application. Il assure la gestion de ces données et garantit leur intégrité.

- *La vue :*

  Elle correspond à l'interface avec laquelle l'utilisateur interagit. Sa première tâche est de présenter les résultats renvoyés par le modèle. Sa seconde tâche est de recevoir toutes les actions de l'utilisateur.

- *Le contrôleur :*

  Il prend en charge la gestion des événements de synchronisation pour mettre à jour la vue ou le modèle. Il reçoit tous les événements de l'utilisateur et enclenche les actions à effectuer.

  Si une action nécessite un changement des données, le contrôleur demande la modification des données au modèle et avertit la vue de leurs changements. Cette dernière pourra se remettre à jour.

Certains événements de l'utilisateur ne concernent pas les données, mais la vue.

Dans ce cas, le contrôleur demande à la vue de se modifier.

Le contrôleur n'effectue aucun traitement, ne modifie aucune donnée, il analyse la requête du client, se contente d'appeler le modèle adéquat et de renvoyer la vue correspondant à la demande.

b) *Représentation du modèle MVC*

Tout d'abord, l'utilisateur envoie une demande d'action au *Contrôleur* via un moyen d'interaction quelconque (*clavier, souris, voix...)* géré par le moteur graphique (*Cf. Figure 20*).

Ensuite, le *Contrôleur* reçoit cette demande, l'interprète et en déduit les actions à demander au *Modèle*. Le *Modèle* est activé et doit renvoyer un objet « *résultat* ». Puis, le *Contrôleur* choisit la *Vue* qui doit afficher ce résultat à l'utilisateur final. Elle élabore ensuite le flux graphique à partir des informations de l'objet résultat, puis envoie ce flux vers le moteur graphique. Celui-ci devra interpréter ce flux pour afficher la page ou la fenêtre graphique.

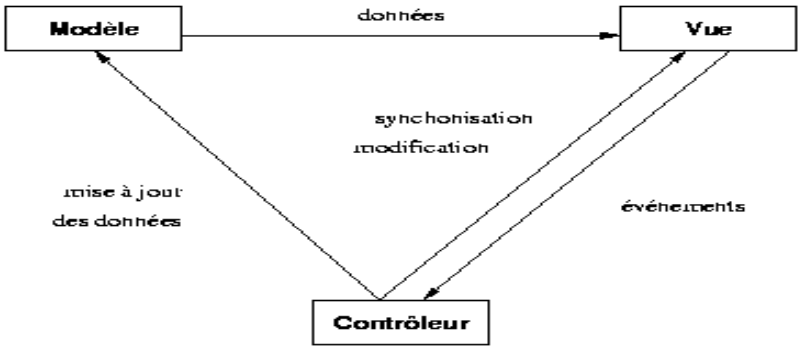


***Figure 20** - Interactions entre le modèle, la vue et le contrôleur [Carton 2005]*

c) *Le modèle MVT*

Cependant, la terminologie utilisée par Django diffère du modèle MVC avec le modèle MTV (*Modèle Template Vue*). En réalité, le *template* selon Django correspond à l'affichage tandis que la *vue* correspond au traitement des données.

Modèle	⟷	Modèle
Vue	⟷	Template
Contrôleur	⟷	Vue

L'avantage majeur d'une telle approche est que les composants sont faiblement couplés.

Chaque composant d'une application Django peut être changé indépendamment sans affecter les autres.

d) Django et la notion de projet

Django fonctionne selon sur la notion de projets qui gèrent une ou plusieurs applications.

Un projet est une instance d'un certain nombre d'applications avec une configuration associée.

Un fichier de configuration nommé « *settings.py* » est propre pour chaque projet.

Il contient la liste des applications actives, les informations de connexion à la base de données, ainsi que le chemin d'accès pour les templates. Le projet met aussi en place le routage des URLs par l'intermédiaire du fichier « *urls.py* ».

Une application, qui est un ensemble de fonctionnalités Django portables (*exploitables dans un autre projet*), est composée de modèles et de vues.

e) Django et la notion de modèle

Un modèle Django *[Django 2011]* se base sur le « *Mapping Objet Relationnel* » (*ORM – Object Relational Mapping*) qui permet de faire correspondre une base de données relationnelle vers une base de données orientée objet avec les équivalences suivantes :

Modèle Django	SGBDR
Classe	Table
Objet	Attribut
Champ	n-uplets

A partir d'un modèle, Django crée le schéma de la base de données et offre la possibilité au développeur, d'utiliser des fonctions de haut niveau permettant de manipuler les objets sans utiliser le langage SQL (*Cf. Figure 21*).

```
from django.db import models

class Personne(models.Model):
    nom = models.CharField(maxlength=30)
    prenom = models.CharField(maxlength=30)
```

Figure 21 *– Exemple de modèle Django (models.py) [Borer et al. 2007]*

- *Exemple d'utilisation avec l'interpréteur Python:*

Personne.objects.filter(nom ="toto") ➔ select * from Personne where (nom ="toto")

Requête avec Django **Requête avec SQL**

La commande Python « *manage.py syncdb* » permet de créer le modèle dans la base de données.

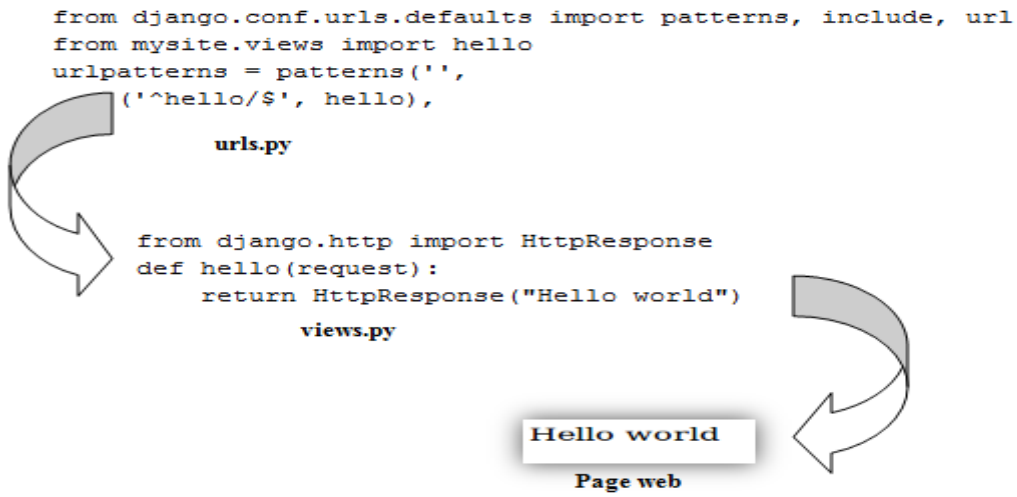
Un code SQL est généré automatiquement. Le modèle est stocké dans un fichier dénommé « *models.py* » qui est propre à chaque application.

f) Django et la notion de vue

Une vue est un fichier « *view.py* » contenant des fonctions Python d'une application Django qui permettent de gérer les données stockées dans une base et de les renvoyer vers un template de présentation spécifique.

g) Django et la gestion des URLs

Les URLs qui sont définies dans le fichier « *urls.py* » permettent d'appeler une vue qui va exécuter des fonctions Python et renvoyer le résultat vers une page Web (*le template*) grâce à des méthodes comme « *HttpRequest* » ou « *HttpResponse* ». Ces objets permettent de charger la vue appropriée via l'URL. Dans l'exemple ci-dessous (*Cf. Figure 22*), lorsque l'adresse « */hello/* » est sélectionnée, la vue « *hello* » est requise, afin d'afficher directement « *Hello world* » sur l'écran.



(

Figure 22 *– Exemple d'utilisation d'une vue [Holovaty et al. 2009]*

h) Django et la notion de template

Les templates sont intimement liés aux vues et s'occupent uniquement de la représentation des données dans une page Web pour l'utilisateur.

Un template est un simple fichier texte qui permet de générer n'importe quel format *(Html, Xml, Cvs, etc.*). Il contient des variables qui prennent une valeur lors de l'évaluation du template et des tags qui contrôlent la logique du template.

h.1) Les variables

Une variable est référencée dans une page comme « {{ ***variable*** }} ».

Lorsque le moteur du template rencontre une variable, il l'évalue et la remplace par sa valeur.

Ainsi si une variable est déclarée comme « {{ ***personne.nom*** }} », son évaluation aura pour effet de remplacer la variable par l'objet « ***nom*** » contenu dans la classe « ***personne*** ».

Si la variable n'existe pas, le système de template lui attribue une valeur par défaut (*chaîne de caractère vide*).

h.2) *Les tags*

Un tag de la forme « {% tag %} », permet de manipuler les objets de la base de données dans un template en effectuant par exemple des boucles pour en afficher tous les champs.

Dans l'exemple suivant (*Cf. Figure 23*), on teste si la classe « ***athlete_list*** » existe puis on récupère l'objet « ***athlete*** » et on affiche le champ « ***name*** » dans la page Web. Si la classe n'existe pas, on affiche un message.

```
{% if athlete_list %}
    {% for athlete in athlete_list %}
        <p>{{ athlete.name }}</p>
    {% endfor %}
{% else %}
    <p>There are no athletes. Only computer programmers.</p>
{% endif %}
```

Figure 23 – *Exemple de template [Holovaty et al. 2009]*

i) Exemple d'utilisation de Django avec la vue et le template

Dans l'exemple suivant, lorsque l'URL « *http://120.0.0.1/time* » est saisi, la vue « ***current_datetime*** » est appelée. Dans cette vue, une fonction Python permet de récupérer la date du jour et de l'affecter à une variable « ***current_date*** » déclarée également dans le template sous le format {{ ***current_date*** }}. Au moment de lire le template, le moteur remplacera la variable par sa valeur.

A l'écran on aura une page Web avec la mention « « ***It is now 2011 …*** » d'afficher.

```
from django.conf.urls.defaults import patterns, include, url
from mysite.views import current_datetime
urlpatterns = patterns('',
    ('^time/$', current_datetime),
```

urls.py

```
from django.shortcuts import render_to_response
import datetime

def current_datetime(request):
now = datetime.datetime.now()
return render_to_response('current_datetime.html', {'current_date': now})
```

views.py

```
<html>
    <body>
    It is now {{ current_date }}.
    </body>
</html>
```

Template : current_datetime.html

It is now 2011-06-28 17:16:53.033808.

Page web

Figure 24 – *Exemple d'utilisation de Django avec une vue et un template [Holovaty et al. 2009].*

j) Philosophie de Django

Django est écrit entièrement en Python et nécessite donc que le moteur Python soit installé.

Django offre les caractéristiques suivantes :

- *Développement rapide :*

  Simplification dans la construction d'une application Web.
- *Couplage faible :*

  Les composants logiciels développés par Django ont un faible couplage afin d'être interchangeables. En effet, si deux morceaux de code sont faiblement couplés, alors les changements effectués à l'un des morceaux n'auront que peu, voir pas, d'effet sur l'autre.
- *Code concis :*

  L'application ne nécessite que très peu de code.
- *Gestion des URLs :*

  Le framework simplifie la gestion des URLs et permet d'afficher un chemin clair.

  De plus, plusieurs URLs peuvent utiliser la même vue.
- *Création de templates :*

  Django propose son propre langage de template permettant de mettre en forme les données sans taper de code Python.

D'autre part, le framework fournit un ensemble d'applications réutilisables dans les projets :

- *Interface d'administration :*

  Cette interface est limitée aux administrateurs authentifiés du site et permet l'ajout, l'édition et la suppression du contenu d'un site.
- *Système d'authentification :*

  Le système d'identification des utilisateurs sous Django gère les comptes utilisateurs, les groupes, les permissions et les sessions utilisateurs basées sur les cookies.
- *Créateur de flux RSS :*

  Django est fourni avec une API de génération de syndication de contenu de haut niveau qui permet de créer facilement des flux RSS et Atom.
- *Mise en cache :*

  Django est fourni avec un système de cache qui permet d'enregistrer les pages dynamiques et de les fournir à la demande aux utilisateurs.
- *Middleware :*

  Django permet également d'ajouter la fonctionnalité de middleware afin d'effectuer des actions directement sur les requêtes et les réponses avant que celles-ci n'arrivent à la vue.
- *Possibilité d'inclure du JavaScript*

Django est un framework puissant, offrant un grand nombre de fonctionnalités tout en restant relativement simple d'accès. De plus, il fournit un grand nombre d'outils facilitant le développement. En se basant sur le langage Python, lui aussi très puissant tout en restant simple d'accès, Django permet de développer rapidement des applications Web en se concentrant uniquement sur la logique du métier.

2.3.5.4 A propos du cadriciel Pylon *[Pylon 2010]*

Pylon est un framework léger soulignant la flexibilité, la facilité et le développement rapide des applications Web en combinant les meilleures idées des langages *Ruby*, *Python* et *Perl*.

Il figure parmi les premiers projets qui utilisent la spécification « *WSGI* » (*Web Server Gateway Interface*) qui vise à améliorer la flexibilité d'un site Web avec Python.

WSGI qui est issue de la communauté Python, unifie la connexion de composants Python avec les différents serveurs Web existants et permet de construire et étendre des applications Web.

Ainsi, une application développée selon *WSGI*, peut facilement être élaborée dans un autre framework de développement respectant les mêmes mécanismes. D'ailleurs, la plupart des frameworks Web actuels basés sur Python s'appuient sur *WSGI*.

De plus, Pylon est basé sur le modèle MVC comme *Rails* ou *Django*.

Toutes les applications sont encapsulées dans des paquets facilement distribuables désignés sous le terme anglais « *eggs* ».

2.3.5.5 A propos du cadriciel TurboGears *[TurboGears 2010]*

TurboGears, développé en 2005 par Kevin Dangoor est un framework écrit en Python qui utilise également l'architecture MVC. Il se caractérise par une architecture modulaire dont la plupart des composants peuvent être remplacés par d'autres. Il s'agit d'un « *méga-framework* » qui repose sur des composants existants issus du meilleur code source disponible. Une très large communauté permet de faire évoluer le framework sans jamais réécrire le code. Cependant, l'utilisation du framework nécessite comme Django, la présence du moteur Python sur le système.

TurboGears intègre de nombreux projets Python comme « *SQLObject* », « *Cherrypy* », « *Kid* », et « *Mochikit* ». Par défaut TurboGears utilise « *SQLObject » ou « SQLAlchemy »* pour le *Modèle*, « *CherryPy »* pour le *Contrôleur* et « *Kid »* pour la *Vue*.

Les applications « *SQLObject* » et « *SQLAlchemy »* sont des solutions d'ORM servant à implémenter le *Modèle* dans les applications TurboGears. Elles permettent de manipuler la base de données par le biais d'objets Python, sans avoir à écrire de requêtes SQL.

D'autre part, « *CherryPy »* écoute les requêtes et associe les objets responsables des traitements à effectuer aux URLs correspondantes avant de renvoyer une réponse HTTP aux requêtes de l'utilisateur.

Le framework TurboGears utilise également « *Kid* », un moteur de templates XHTML qui se charge d'afficher les résultats. Il utilise également l'API « *Mochokit* » pour intégrer du *JavaScript* dans les applications Web.

TurboGears 2.x est une modification de TurboGears 1.x au-dessus de l'API Pylons avec un jeu de composants standards. Le framework profite de la robustesse de l'implémentation de la norme *WSGI* sur Pylon en améliorant la prise en main et en facilitant le développement des applications.

Le framework offre également une interface d'administration destinée aux développeurs.

Elle leur permet d'avoir accès à divers outils dont un éditeur de contenu et un gestionnaire de base de données basiques.

2.3.5.6 A propos du cadriciel Zope *[Zope 2010]*

Zope est un serveur d'application Web orienté objet, élaboré par Jim Filton, développeur pour Digital Creation qui deviendra Zope Corporation en 2001.

Ce framework, gérable à partir d'une interface Web, est devenu libre en 1998 et entièrement écrit avec le langage de programmation Python. Zope est désormais maintenu par la fondation « *Zope* » qui est composée de membres de la communauté de développeurs.

Zope offre la possibilité d'administrer le site à travers une interface Web (*création de pages, programmer des composants, créer de nouvelles classes, etc.*). Il s'agit avant tout d'un serveur Web dynamique qui inclut une base de données orientée objet. Les objets Python peuvent donc être créés et stockés au sein de cette base de données spécifique, qui prend en charge leur gestion, leur modification, ainsi que leur présence, ou non, au sein d'un cache.

L'un des principaux reproches à l'égard de Zope est sa courbe d'apprentissage.

D'autre part, Zope souffre encore aujourd'hui d'un manque de notoriété malgré une forte communauté en Europe et tout particulièrement en France.

2.4 Architecture Orientée Services (SOA)

SOA (*Service Oriented Architecture : Architecture Orientée Services*) est une architecture qui rassemble les grandes applications de l'entreprise (dites « *applications composites* ») en services interopérables et réutilisables *[Lhérault 2011]*. SOA est d'abord et avant tout, une démarche architecturale et organisationnelle, le choix des technologies et des outils reste secondaire.

2.4.1 Généralités sur les services

Un « *service* » au sens SOA, met à disposition aux acteurs intervenant dans des processus métiers, un accès vers une ou plusieurs fonctions métiers (*Cf. Figure 25*). Il concrétise le lien entre la couche métier (*consommateur*) et les implémentations dans le système d'information (*fournisseur*) en prenant à sa charge un contrat (*réalisé par le pourvoyeur*). Le service, qui doit être simple d'emploi et réutilisable, peut être consommé par un utilisateur (*via une application composite interactive*), un système traditionnel, un processus métier ou bien un autre service SOA.

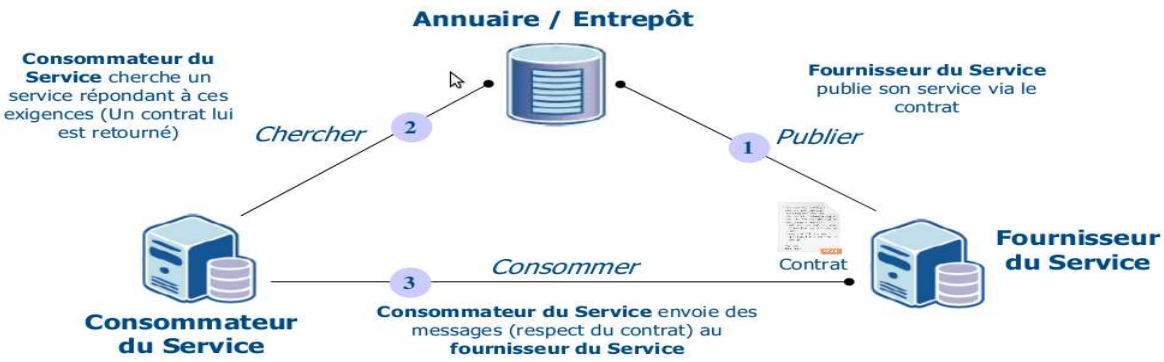


Figure 25 *- Le concept SOA [Baron 2011].*

2.4.1.1 Caractéristiques d'un service SOA

Voici les aspects fondamentaux qui caractérisent un service *[Erl 2007] :*

- *Contrat standardisé :*

  L'ensemble des services d'un même système technique sont exposés au travers de contrats respectant les mêmes règles de standardisation.

- *Couplage lâche :*

  Le contrat d'un service doit imposer un couplage lâche de ses clients.

  Les services sont connectés aux clients et autres services via des standards. Ces standards assurent le découplage, c'est-à-dire la réduction des dépendances. Ces standards sont des documents XML comme dans les Web Services.

- *Abstraction :*

  Le contrat d'un service ne doit contenir que les informations essentielles à son invocation.

  Seules ces informations doivent être publiées.
- *Réutilisable :*

  Un service exprime une logique agnostique et peut ainsi être positionné comme une ressource réutilisable.
- *Autonome :*

  Un service doit exercer un contrôle fort sur son environnement d'exécution sous-jacent.

  Plus ce contrôle est fort, plus l'exécution d'un service est prédictible.
- *Sans état :*

  Un service doit minimiser la consommation de ressources en déléguant la gestion des informations d'état quand cela est nécessaire.
- *Découverte :*

  Un service est complété par un ensemble de méta-données de communication au travers desquelles il peut être découvert et interprété de façon effective.
- *Modulable :*

  Un service doit être conçu de façon à participer à des compositions de services.

2.4.1.2 Principes d'un service SOA

L'ouvrage publié par Guillaume Plouin *[Plouin et al. 2008]* nous permet de comprendre le mécanisme du service au sein d'une architecture SOA.

Le service effectue une ***opération*** (*couple : requête, réponse*) qui consiste à renvoyer une ***réponse*** au client dès que ce dernier a émis une ***requête*** de demande d'un service.

La requête transmise par le client ainsi que la réponse du serveur sont envoyées sous forme de **message** transitant vers un bus de message SOA sur le réseau. Les messages décrivent chaque opération avec une signature de l'opération (*triplet : nom de l'opération, message de requête, message de réponse*). Un message doit être transporté dans une enveloppe normalisée qui précise dans un en-tête les éléments suivants :

- *L'adresse du service.*
- *Une demande d'accusé de réception.*
- *L'encodage ou la compression du contenu transporté.*
- *Des informations de sécurité.*

L'indépendance vis-à-vis des protocoles de transport conduit à privilégier l'usage naturel d'une structuration de l'enveloppe et du contenu en XML. Le W3C recommande depuis 2003, SOAP 1.2 comme protocole de messagerie standardisé pour les services Web.

2.4.1.3 Le contrat de services

L'ouvrage *[Plouin et al. 2008]* nous donne des indications sur la notion de contrat de service.
Le contrat est constitué d'une liste d'une ou plusieurs opérations.
Pour obtenir ou offrir un service, les parties (*consommateur et pourvoyeur*) doivent respecter le contrat qui décrit le type de protocole de communication et définis les messages entrants/sortants échangés.
Le consommateur doit non seulement connaître l'offre de service (*contrat de base*) mais aussi les garanties offertes par le pourvoyeur qui peuvent proposer plusieurs niveaux de qualités selon le service disponible. Le pourvoyeur offrant des garanties, maximise l'utilisation de son service et le rentabilise.

2.4.1.4 Les services Web

Une architecture SOA est une architecture logicielle qui s'appuie sur un ensemble de services simples.
Lorsque l'architecture SOA s'appuie sur des Web services, on parle alors de WSOA (*Web Services Oriented Architecture*). De nombreuses définitions de « *services Web* » existent, mais sont généralement trop vagues ou trop précises. Tout d'abord, le consortium W3C apporte une définition plutôt orientée protocoles *[W3C 2002]* :
« *Un service Web est une application logicielle, identifiée par un URI, dont les interfaces et les liaisons sont définies, décrites et découvertes par des méthodes basées sur XML. Le service Web doit également supporter les interactions directes avec d'autres applications logicielles en utilisant des messages en format XML, des protocoles Internet standard* ».

Tandis qu'un expert de chez IBM, Mark Colan, fournit une définition plutôt axée sur les principes *[Colan 2004]* :
« *Les services Web sont des applications modulaires, auto-contenues et auto-descriptives qui peuvent être publiées, localisées et invoquées depuis le Web. Les services Web effectuent des actions allant de simples requêtes à des processus métiers complexes. Une fois qu'un service Web est déployé, d'autres applications peuvent le découvrir et l'invoquer) ».*

En fait d'après l'article *[3ie 2003],* les services *Web* facilitent l'invocation de certains traitements depuis Internet. Le modèle logiciel proposé contient à la fois un nouveau mode de développement et une nouvelle méthode de fourniture de services.

2.4.1.5 Les composants d'un service Web

Afin de mettre en œuvre un service Web, trois composants sont nécessaires (*Cf. Figure 26*) :

- *Un protocole de transport (SOAP, XML-RPC, REST).*
- *Un protocole pour décrire le service (WSDL) qui permet entre autres d'énumérer les méthodes disponibles, ainsi que leurs signatures.*
- *Un protocole pour écrire les messages (XML, HTML).*

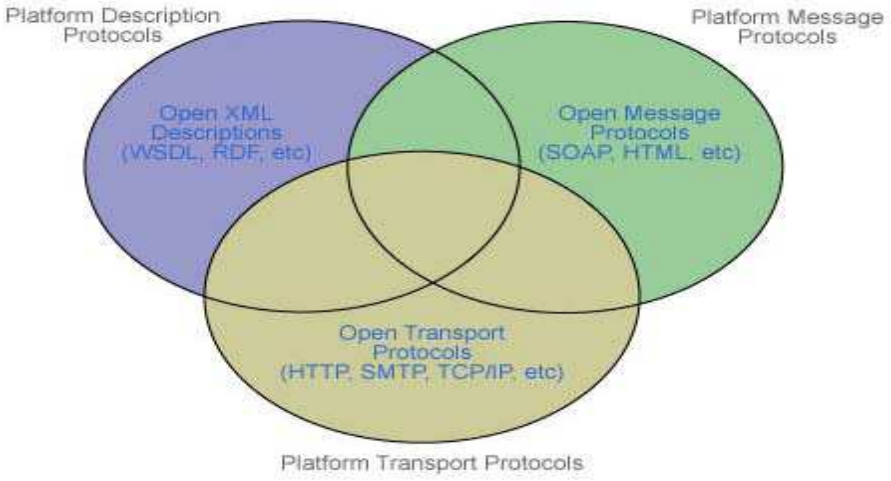


Figure 26 *- Protocoles d'une application orientée services [IBM 2002].*

2.4.1.6 Les protocoles de communication

Il existe trois standards qui permettent à des applications de communiquer directement entre elles sans se préoccuper des technologies sur lesquelles elles sont implémentées *[Plouin et al. 2008]*.

Parmi ces trois méthodes nous avons d'un côté XML-RPC et SOAP qui sont basés sur des technologies XML et qui utilisent le protocole HTTP, mais qui peuvent s'appuyer sur d'autres protocoles de transport. Le standard REST à l'inverse est le mode natif du protocole HTTP.

a) Le protocole XML-RPC

Le protocole XML-RPC, qui date de 1995, est l'ancêtre du protocole SOAP. Il permet d'invoquer des services à distance via les standards XML et HTTP, indépendamment du langage de programmation de l'application logicielle. Les processus d'invocation de service à distance utilisent le protocole HTTP pour le transport des données et le langage XML pour leur codage.

Il est toutefois possible de faire du XML-RPC sur un autre protocole que HTTP.

La technologie ne propose pas de notion de description de l'interface du service.

En effet, les deux parties impliquées dans l'invocation (*consommateur et fournisseur*) doivent se concerter pour se mettre d'accord sur l'adresse du service, ses méthodes, ses paramètres, etc.

D'autre part, il n'y a pas non plus de normalisation des erreurs XML-RPC.

Cependant, la technologie est très simple à utiliser, mais inadaptée dans des architectures distribuées complexes.

b) Le protocole SOAP

SOAP a été conçu en 1998 afin d'étendre les possibilités de XML-RPC.

Ce standard a été pensé d'une part pour être indépendant de l'implémentation technique du service, d'autre part pour passer à travers des pare-feux et ainsi permettre l'invocation de services situés dans des systèmes d'information partenaires. SOAP gère l'échange de messages XML et s'appuie sur différents protocoles de transport pour des échanges synchrones ou asynchrones :

- *HTTP (échange synchrone).*
- *Java RMI (échange synchrone).*
- *.Net Remoting (échange synchrone).*
- *SMTP (échange asynchrone).*
- *FTP (échange asynchrone).*
- *Java JMS (échange asynchrone).*
- *.net MSMQ (échange asynchrone).*

SOAP permet de créer des applications de type distribuées, en suivant le modèle.

Il ne définit pas le modèle de programmation sous jacent (*appel synchrone ou asynchrone*) mais décrit uniquement la structure des messages échangés.

b.1) Les avantages de SOAP

SOAP propose une manière simple d'interconnecter les services en faisant abstraction du langage d'écriture du service et de la plate-forme d'exécution et en s'intéressant uniquement aux échanges entre les services. Le protocole permet la sérialisation et la désérialisation de tout objet métier, sous une forme XML. Il est utilisable quelle que soit la technologie d'implémentation et normalise la gestion des erreurs.

b.2) Les contraintes de SOAP :

Le poids d'un message SOAP (*codé en mode texte*) est beaucoup plus important qu'un message codé en binaire et augmente son temps d'acheminement.

De plus, la syntaxe du protocole SOAP est relativement complexe comparée au protocole XM-RPC ou REST. SOAP est la spécification la plus récente, la plus complète et la plus complexe des trois méthodes de communication. C'est également la méthode la plus utilisée dans les architectures SOA.

c) Le protocole REST

Il s'agit d'un modèle d'architecture proche des origines du Web.

Le standard REST est beaucoup plus simple que SOAP, car il utilise les différentes méthodes du protocole HTTP pour consulter ou mettre à jour des données distantes.

REST n'est pas à proprement parlé un protocole d'invocation à distance, il s'agit essentiellement d'un protocole destiné à des services CRUD (« *Create, Read, Update, Delete* ») utilisant des messages XML pour l'accès à des ressources distantes.

Les services CRUD sont des services de base qui permettent de créer, mettre à jour, supprimer et rechercher des données au sein du système d'information (*via une base de données*).

REST affecte une adresse URI *(« Uniform Resource Identifier » : format d'adressage sur le Web*) à chacune des ressources à accéder.

Ainsi, les URLs sont particulièrement lisibles et les requêtes sont fortement simplifiées.

En outre, invoquer un service Web de type REST revient à utiliser une simple URL en HTTP.

De plus, REST est largement utilisé pour les intégrations au niveau de l'interface utilisateur dans des services sur Internet. Il est davantage destiné à des développeurs de sites Web et ne peut être utilisé pour des compositions de services au sein d'applications métiers complexes.

Les systèmes qui suivent les principes REST sont souvent appelés « *RESTful* ».

2.4.1.7 Les protocoles de contrats de services et de découverte

Les services Web reposent sur des standards de communication pour permettre leurs utilisations au sein d'Internet. Il existe également un standard WSDL qui permet de décrire le mode de fonctionnement d'un service Web puis le standard UDDI qui permet de découvrir les services Web dans un réseau et offre la possibilité de s'y connecter *[Plouin et al. 2008], [Kadima et al. 2003]*.

a) WSDL (Web Service Description Langage)

WSDL est un langage XML normalisé pour décrire le mode de fonctionnement d'un service Web.

Il permet ainsi de décrire les modalités d'invocation distante d'un service Web :

- *Les opérations possibles au sein du service.*
- *Les paramètres d'entrées/sorties de ces opérations.*
- *Le typage de ces paramètres.*
- *Les points d'entrée (URL) des opérations.*

b) UDDI (Universal Description, Discovery and Integration)

UDDI a été conçu pour permettre la découverte et l'intégration automatique d'un service.

La spécification UDDI est à l'origine d'une norme du W3C écrite en 2000.

Le standard repose sur le protocole de transport SOAP et assure que les requêtes et les réponses sont des objets UDDI envoyés sous forme de message SOAP (*Cf. Figure 27*).

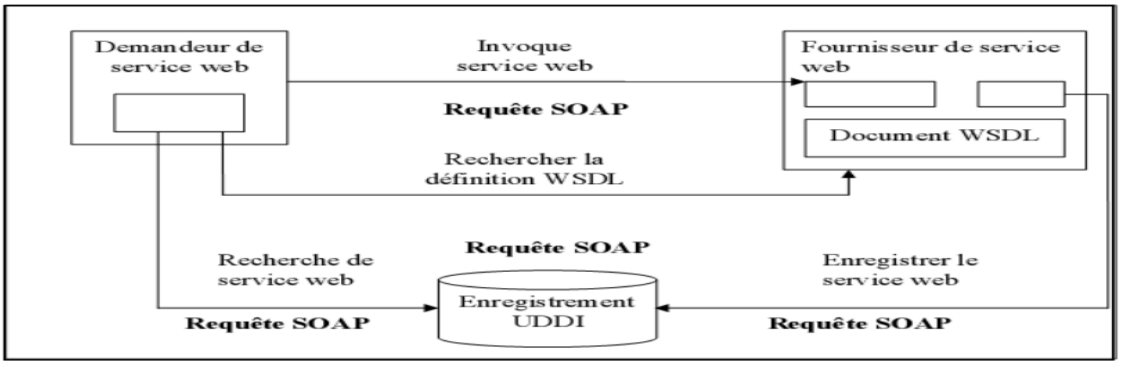


Figure 27 *- Principe de l'annuaire UDDI [Kadima et al. 2003].*

Un annuaire UDDI est construit sur la base des normes WSDL et SOAP, les services y sont décrits en WSDL et on accède à l'annuaire via des requêtes SOAP. L'annuaire est constitué de trois types d'informations :

- *Les pages blanches qui décrivent les fournisseurs de services.*
- *Les pages jaunes qui décrivent les catégories de services.*
- *Les pages vertes qui décrivent les contrats WSDL.*

UDDI et WSDL sont complémentaires dans l'implémentation des services Web.
UDDI fournit une méthode de publication et de recherche de description de service.
Les entités de données UDDI fournissent un support pour définir aussi bien les informations métiers et celles de service. UDDI permet de prendre en charge n'importe quel langage de description de service Web, grâce à son modèle de données extensible. Dans le monde des entreprises, l'annuaire est souvent utilisé pour mettre en œuvre un registre interne des services constituant un référentiel d'entreprise.

2.4.1.8 Les enjeux d'une architecture SOA

Une publication du journal du net *[Journal du net 2011]* explique les enjeux d'une architecture orientée services au sein d'une entreprise. Ainsi par son caractère standard, l'approche SOA contribue à améliorer la rapidité ainsi que la productivité des développements.
Un composant exposé sous forme de Web Services peut être réutilisable par d'autres applications.
Elle offre une modularité qui permet de remplacer un service par un autre et de meilleures possibilités d'évolutions. Les services Web sont des composants flexibles qui offrent une plus grande tolérance aux pannes. D'autre part, avec l'utilisation d'un annuaire de service, il est possible de mettre en place une solution de déploiement à la demande.

2.4.1.9 Les obstacles d'une architecture SOA

L'article indique également les principaux freins aux développements d'une architecture SOA.
Certains anciens systèmes demeurent difficilement compatibles avec les Web Services et ne peuvent pas s'inscrire dans une telle architecture.
De plus, même si les standards des services Web (SOAP/WSDL) commencent à se généraliser, les solutions d'intégration doivent encore trop souvent proposer des langages complémentaires pour la gestion des transactions ou de la sécurité.
D'autre part, il existe encore aujourd'hui assez peu de méthodes couvrant l'élaboration et le déploiement d'une architecture SOA, peut être par manque de retour d'expérience.

2.5 SaaS (Software as a Service)

Le « *Software as a Service* », autrement dit, « *logiciel en tant que service* » est un modèle de déploiement d'application dans lequel un fournisseur loue une application clé en main à ses clients en tant que service à la demande au lieu de leur facturer des licences.

De cette façon, l'utilisateur final n'a plus besoin d'installer tous les logiciels existants sur sa machine de travail. Contrairement à une architecture SOA qui s'utilise dans une infrastructure interne d'une entreprise à des fins stratégiques de modélisation des processus métiers, une architecture SaaS reprend les principes du SOA en les adaptant à un contexte plus large qui est celui d'Internet.

La plupart des informations relatives au SaaS proviennent de l'ouvrage de référence *[Plouin 2009]*.

2.5.1 Les origines du SaaS

Selon Wikipédia *[Wikipédia 2011]*, le SaaS correspondait à « ASP » (*Application services providers*) du début des années 2000. Le terme « ASP » (*en français : Fournisseur d'applications hébergées*) désigne la fourniture par un prestataire d'une application utilisable à travers les réseaux.

Il s'agit donc de l'hébergement d'une application ou d'un service en ligne avec un système d'abonnement. L'ASP peut être considéré comme descendant indirect du « *service bureau* » des années 1960 et 1970. Leur objectif est de permettre à des clients de pratiquer l'externalisation sur des applications spécifiques et ainsi de se recentrer sur le cœur du métier. Il existe plusieurs alternatives dans l'utilisation des applications hébergées :

- *Utilisation d'une interface Web :*

  L'application logicielle installée sur le système informatique du vendeur est accessible par l'utilisateur à travers un navigateur Web.
- *Utilisation d'une interface client/serveur :*

  L'application logicielle, qui doit être installée sur chaque poste utilisateur, fait office d'interface avec le serveur en utilisant des APIs et/ou un langage de communication généralement basé sur XML. L'application ne fonctionne plus en mode hébergée et n'est pas réalisable pour un grand nombre d'utilisateur.
- *Utilisation d'un client riche RIA (Rich Internet Application) :*

  L'application offre un supplément d'ergonomie aux pages Web et permet d'utiliser des interfaces sophistiquées. Le RIA est basé sur un environnement d'exécution intégré au navigateur Web. Il offre une solution purement Web sans problématique de déploiement, tout en bénéficiant d'une architecture client/serveur décentralisée. En revanche, il n'y a pas de gestion en mode déconnecté en cas de perte de connexion.

- *Utilisation d'un client riche RDA (Rich Desktop Application :*

  Les applications sont téléchargées depuis un navigateur sur le système d'exploitation et s'exécutent en dehors du navigateur. En outre, le logiciel charge toujours les versions les plus récentes des applications, et permet ainsi d'éviter les procédures complexes d'installation ou de mise à niveau.

Une application SaaS repose sur le modèle RIA dans une architecture multi-locataire.

Le modèle SaaS est fortement inspiré des principes qui constituent le Web 2.0 avec des applications omniprésentes et accessibles via Internet. Il utilise les principes d'une architecture SOA en utilisant les applications en tant que services. En outre, les architectures ASP et SaaS sont identiques dans leurs intentions (*vendre un service plutôt qu'un logiciel qui s'installe*), mais différentes dans leur réalisation technique, car seul le SaaS reproduit des architectures Web (*SOAP, XML, etc.)*.

2.5.2 Les enjeux du SaaS

Le modèle SaaS permet la réduction du coût total de possession (*TCO*).

En effet, la consommation par abonnement est souvent moins coûteuse que la somme de l'achat de licences dites « *perpétuelles* », de l'achat de support, de l'achat de mises à jour et des frais d'exploitation. Par ailleurs, les SaaS permettent des modèles de commercialisation variés avec notamment les offres « *freemium* », l'abonnement mensuel ou annuel, le paiement à l'usage.

Il permet en outre de déporter vers des sociétés spécialisées, les problématiques d'exploitation, ainsi, les entreprises peuvent se recentrer sur leur cœur de métier.

2.5.3 Les domaines d'applications

L'essentiel des applications concernées se trouvent dans le domaine du collaboratif et de la messagerie électronique avec notamment les outils de gestion de la relation client, la vidéo-conférence, la gestion de documents, la gestion de contenu (*CMS : content management system*), les courriels, les ressources humaines, la comptabilité (*Cf. Figure 28*).

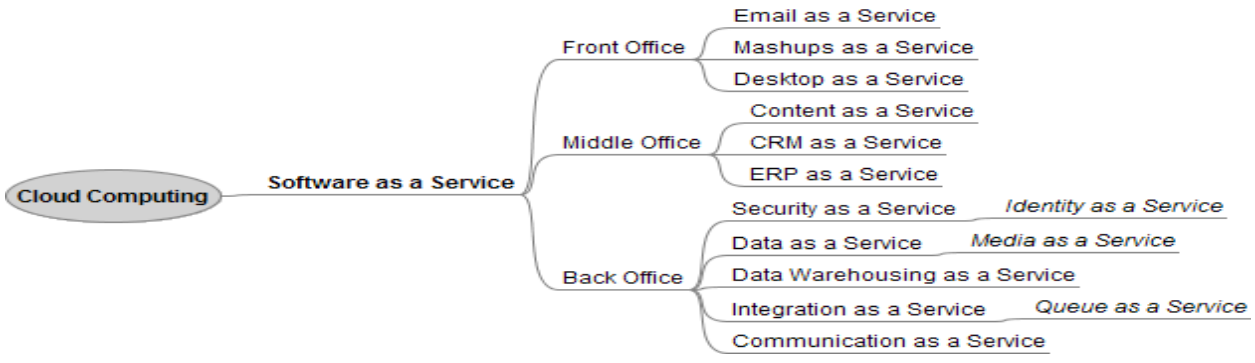


Figure 28 *- Les applications en tant que services [Plouin 2009].*

2.5.4 Les bénéfices et les risques du modèle SaaS

Les acteurs du SaaS fournissent des services ou des logiciels informatiques par le biais du Web et non plus sous la forme d'une application de bureau ou en mode client/serveur. Ce mode de consommation du logiciel a des avantages, mais aussi des inconvénients *[Traumat 2008], [Marcombe 2010]*.

2.5.4.1 Les bénéfices

Le SaaS est devenu un modèle crédible et plébiscité sur le marché pour ses différentes vertus.
Voici les avantages les plus courants que l'on peut citer :

- *Les SaaS proposent des **logiciels opérationnels**, prêts à l'emploi, sans passer par une étape d'installation et sans aucune tâche de maintenance.*
- *Il n'est pas nécessaire de disposer d'un **fond propre** important pour la mise en œuvre et la maintenance d'une infrastructure.*
- *Les **mises à jour** sont automatiquement mises en œuvre sur tous les clients dès qu'elles sont disponibles.*
- *La **maintenance** n'est plus à la charge des clients, mais aux fournisseurs du service. Le client s'attend selon les contrats de service à une disponibilité quasi permanente avec un délai maximal de non disponibilité.*
- *Possibilité d'**adapter les ressources** en fonction de la montée en charge.*
- *L'entreprise peut **se concentrer sur son cœur de métier**.*
- ***L'informatique devient un service** comme l'électricité, on consomme ce dont on a besoin sans investir dans les infrastructures.*
- *Les éditeurs de logiciels peuvent désormais bâtir de nouvelles solutions en utilisant les **APIs proposés par les éditeurs SaaS**.*
- *Les fournisseurs rassemblent les logiciels à partir de plusieurs sources et construisent un **service complet**.*
- *Le modèle SaaS permet d'**augmenter les ventes** en proposant de nouveaux services comme l'hébergement d'applications logicielles ou la mise à disposition des outils (CRM, CMS, etc.).*
- *Il permet également d'avoir des **revenus réguliers** en proposant aux clients des contrats de services.*

2.5.4.2 Les risques

Malgré ces avantages, il existe des inconvénients qui freinent l'essor du SaaS dans les entreprises :

- ***La confidentialité des données :***

  SaaS augmente le risque de perdre des données critiques ou de les exposer à des tiers.

  Bien que beaucoup d'entreprises utilisent les services de prestataires d'hébergement pour leurs applications Web et leurs extranets clients.
- ***Applications métiers non externalisable :***

  Les applications métiers sont parfois impossibles à externaliser en raison d'un socle d'exécution spécifique. De même, les applications métiers représentent le savoir-faire de l'entreprise que l'on souhaite conserver dans l'enceinte de l'établissement.

  D'une manière générale, les données sont confidentielles et difficiles à externaliser.
- ***Dépendance du fournisseur SaaS :***

  La disponibilité, la fiabilité de la solution dépendent de la technologie utilisée par le fournisseur du SaaS.
- ***La conformité réglementaire :***

  Les entreprises sont méfiantes vis-à-vis des applications SaaS hébergées dans d'autres pays où les réglementations peuvent être différentes.
- ***Rejet de la part du client :***

  Une réaction de rejet liée à une méconnaissance du modèle ou bien en raison de la politique de sécurité du client. Ainsi, d'un point de vue déontologique, il est essentiel d'informer les clients et de leur faire valider le recours au modèle SaaS.

En fait, le principal problème est la confiance que l'on peut accorder à l'opérateur SaaS sur la confidentialité de données. Cette confiance repose sur la réputation et les engagements contractuels du fournisseur de service. Il y a notamment des règles de confidentialité qui les obligent à ne pas divulguer les données de leurs clients.

2.5.5 Les acteurs du modèle SaaS

On peut regrouper les acteurs du SaaS selon leurs origines en trois groupes :

- *Les acteurs pour lesquels le SaaS est l'activité principale (SalesForce.com, Netsuite).*
- *Les éditeurs de logiciels traditionnels qui proposent des services SaaS (Microsoft, IBM, SAP, Oracle).*
- *Les acteurs du Web comme Google.*

2.5.6 La tendance du marché SaaS

Le marché du SaaS était en hausse de 15,7 % en 2010, avec un chiffre d'affaires mondial de 9,2 milliards de dollars *[Barathon 2010]*. En 2011, selon le cabinet d'études américain Gartner, ce marché devrait passer à 10,7 milliards de dollars, en progression de 16,2 %. La gestion des contenus et le collaboratif arrivent en tête, avec un CA de 2,9 milliards de dollars en 2010, suivi du CRM avec 2,6 milliards de dollars.

2.5.7 L'architecture SaaS

2.5.7.1 Les niveaux de maturité

Microsoft *[Chong et al. 2006]* a identifié quatre niveaux de maturité du modèle SaaS combinant les quatre attributs suivants : « *Configuration* », « *Multi-location* », « *Efficacité* » et « *Evolutivité* » (*Cf. Figure 29*).

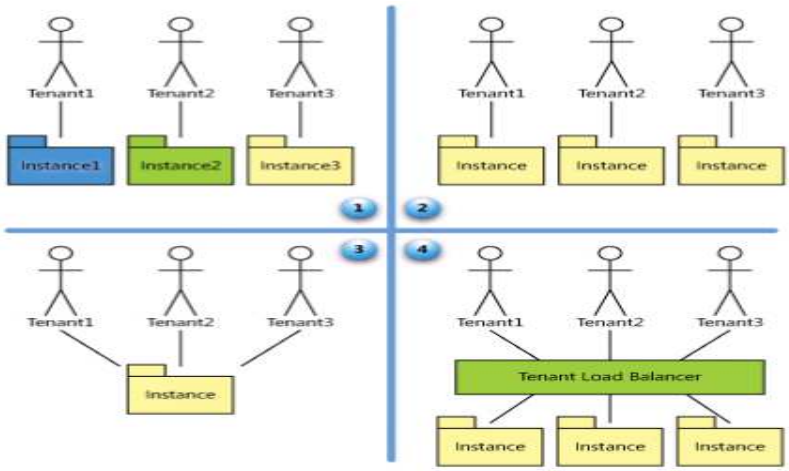


Figure 29 *- Les quatre niveaux du modèle de maturité SaaS [Chong et al. 2006].*

- ***Niveau 1 – Standard :***

  Le premier niveau se rapproche du modèle ASP qui consiste à fournir à chaque client, une application personnelle qui s'exécute dans une instance propre à l'utilisateur. La plupart des applications client-serveur peuvent migrer facilement vers ce premier niveau de maturité.

- ***Niveau 2 – Configuration :***

  Au deuxième niveau, une instance est séparée de l'application pour chaque client.

  Toutes les instances utilisent le même code d'implémentation, mais sont isolées les unes des autres. La migration vers ce niveau de maturité implique au préalable d'avoir le niveau de base. D'autre part, le fournisseur doit fournir suffisamment de ressources matérielles et de stockages pour supporter plusieurs instances exécutant une même application en parallèle.

- ***Niveau 3 - Configuration, Multi-location et Efficacité :***

  A ce niveau, le fournisseur de service fournit une seule instance pour tous les clients.

  Un serveur de méta-données sert d'intermédiaire en collectant les informations de chaque utilisateur. Des politiques d'authentification et de sécurité s'assurent que chaque donnée d'un client soit protégée des autres clients. L'instance peut également être partagée avec l'ensemble des utilisateurs pour former un espace commun de partage.

- ***Niveau 4 - Evolution, Configuration, Multi-location et Efficacité :***

  Ce dernier niveau permet d'ajouter un équilibrage de charge au niveau deux.

  Un serveur de méta-données sert d'intermédiaire en collectant les informations de chaque utilisateur. Il est possible d'augmenter ou de diminuer le nombre de serveur ou d'instance en fonction de la demande sans pour autant refaire l'architecture.

2.5.7.2 La multi-location

Un directeur d'une société de service *C2S* du groupe Bouygues, explique bien le terme de *« multi-location »* (« *multi-tenant* ») *[Laské 2011]***.**

Selon lui, une application est dite « *multi-tenant*e» lorsque son exécution est partagée de manière transparente et étanche entre plusieurs entreprises.

Autrement dit, si l'entreprise A et l'entreprise B décident de faire l'acquisition de l'application X en SaaS, c'est le même serveur qui va être utilisé par les deux entreprises (*Cf. Figure 30*).

Néanmoins, les utilisateurs de l'entreprise A ne verront jamais les données de l'entreprise B dans l'application X et celle-ci pourra supporter un paramétrage différent pour l'entreprise A et pour l'entreprise B.

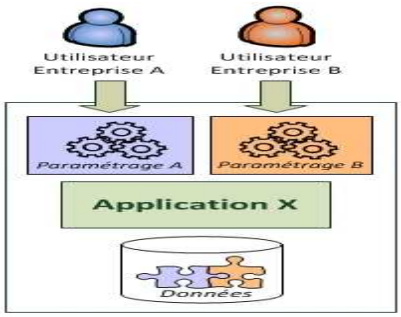


Figure 30 *- Multi-tenant [Laské 2011].*

A l'opposée, une application qui ne serait pas multi-tenante imposerait de séparer l'installation de l'application X pour l'entreprise A et l'installation de l'application X pour l'entreprise B sur deux serveurs différents. On parlerait alors plutôt de « *multi-instance* ».

L'intérêt d'une application « *multi-tenante* » est de lisser la charge entre les différentes entreprises et donc de mutualiser l'infrastructure et les coûts d'utilisation. D'autre part, cela permet au fournisseur de ne gérer qu'une seule version de l'application pour tous ses utilisateurs et donc de limiter les coûts de maintenance.

2.5.7.3 Le modèle SaaS

Les applications SaaS reprennent le modèle SOA avec des services sous forme d'API.

Elles privilégient les architectures de services simplifiées suivant le style REST comme c'est le cas pour les offres de services Amazon. Le standard SOAP est cependant universel et compatible avec toutes les offres de Cloud du marché. Un ouvrage de référence en matière de Cloud Computing *[Furht 2010]* offre un panorama complet sur l'architecture de base du Cloud Computing qui est composée des trois niveaux « IaaS », « PaaS » et « SaaS ».

Le modèle SaaS est la partie interface utilisateur qui permet de piloter une application via le Web.

Cette interface est gérée par la couche « *frontend* » (*Cf. Figure 31).*

Elle fournit les outils nécessaires pour traduire une demande utilisateur en requête système, à travers une interface Web. La couche « *frontend* » est constituée de deux parties avec un côté serveur qui implémente la gestion des ressources et communique les problèmes, tandis que le côté client léger fournit les mécanismes et les outils pour authentifier, accéder et interagir avec le Cloud.

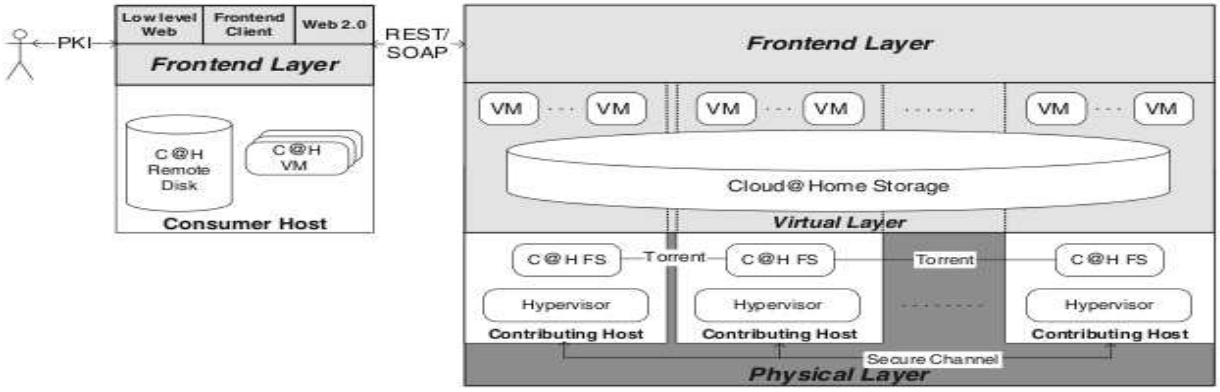


Figure 31 *- Architecture du Cloud Computing [Furht 2010].*

L'architecture suivante (*Cf. Figure 32*), décrite dans le même ouvrage *[Furht 2010]*, est celle adoptée par Google dans son offre « *Google App Engine* » qui permet de développer puis d'héberger en SaaS des sites dynamiques. La première version de 2008 utilisait le langage Python avec le framework Django. Les dernières versions toujours développées en Python supportent le langage Java.

Pour développer une application, Google fournit pour chaque langage un ensemble d'API permettant d'accéder à différents services.

Figure 32 *- Architecture du modèle Saas de Google (Google App Engine) [Furht 2010]*

Le modèle SaaS s'appuie donc sur des normes et des standards comme XML, les services Web, car un système d'information ouvert doit s'appuyer sur ces technologies.

Conception

3.1 Introduction

Durant la première partie de ce mémoire, nous avons fait un tour d'horizon sur les différentes technologies qui sont les fondements même de la mise en place d'un *Cloud Storage* à travers une interface web orientée service. Tout d'abord, les concepts du *Cloud* ont mis en évidence des stratégies financières notamment avec une diminution du coût de possession et une centralisation de l'entreprise à son cœur de métier. L'informatique devient un service facturé à l'usage allant de l'utilisation d'une infrastructure informatique à l'utilisation d'une application web selon le modèle « *PaaS* », « *IaaS* » ou « *SaaS* » choisi. L'offre du maître d'ouvrage Fizians se situe dans le modèle « *SaaS* » (« *Storage as a Service* » plutôt que « *Software as a Service* »).

Ensuite nous avons examiné plusieurs systèmes de fichiers distribués qui sont les éléments de base pour élaborer des ressources partagées de manière distante. L'application *Rozo* avec notamment le système de fichiers Rozofs (*Cf. §2.2.6.4*) repose sur le système de base VFS (*Cf. §2.2.6.1*).

La plupart de ces systèmes distribués sont basés sur le même principe avec une architecture constituée d'un serveur de méta-données, plusieurs serveurs de données et des clients qui les interrogent afin d'accéder à l'information. La différence de *Rozofs* avec les autres systèmes de fichiers se trouve dans l'utilisation d'un algorithme qui segmente les données et récupère des projections nécessaires pour reconstituer le fichier. Il s'agit de la « *transformée Mojette* », élaborée notamment par JeanPierre Guédon *[Guédon 2009]*.

D'autre part, nous avons abordé les architectures web et les langages de développement qui constituent le puzzle pour mettre en place notre interface web de gestion d'un *Cloud Storage*.

On retiendra notamment l'intérêt du langage Python (*Cf. §2.3.4*) dans la portabilité des applications puis l'utilisation du cadriciel *Django* (*Cf. §2.3.5*) avec son architecture *MVC* qui assure l'interdépendance des applications développées. Les données (« *Model* ») sont séparées des fonctions qui les traitent (« *Controller* ») puis de l'interface utilisateur (« *View* ») qui les affiche.

L'échange des informations passe par le biais d'un protocole de communication de type *REST* ou *SOAP* (*Cf. §2.4.1.6*) avec une tendance générale à utiliser *REST* dans les interfaces web grâce à sa simplicité de mise en œuvre.

Toutes ces informations nous permettent de mieux appréhender la conception d'une interface web dans le but de gérer un ensemble de machines virtuelles hébergées chez Amazon puis de mettre en œuvre la solution *Rozo* de Fizians. Tout cela afin de rendre accessible un espace de stockage situé dans le « *nuage* ».

3.2 Description du projet

La société Fizians souhaite proposer à ses clients un service de stockage à la demande dans un environnement de *Cloud public*. Le modèle économique de la prestation est basée sur le paiement à l'usage (« *Pay as you go* »). Ainsi, ce service de stockage offre aux clients la possibilité d'obtenir un ou plusieurs espaces de stockage répartis sur des machines virtuelles hébergées chez le fournisseur *Amazon*. Chaque machine virtuelle de configuration identique contient le système de fichiers *Rozofs* qui assure ce mode de gestion des données. Un ensemble de machines virtuelles forme un espace de stockage unifié, accessible à un utilisateur. L'espace de stockage est calculé selon un nombre fini de machines virtuelles, équivalentes à des conteneurs. Ce nombre dépend de la volumétrie que le client souhaite pour stocker ces données. Par ailleurs, le service permet d'adapter la volumétrie de stockage par l'ajout ou la suppression de machines virtuelles. De plus, il apporte de la redondance et de la répartition des données, car elles sont partagées sur plusieurs machines virtuelles.

3.2.1 L'objectif

Pour assurer une gestion souple et dynamique du système « *Rozofs* » à travers des machines virtuelles Amazon, la société Fizians souhaite réaliser une application web qui permettra à un client de gérer son environnement de stockage au sein d'un *Cloud public*.

Ce *Cloud* est basé sur l'offre de service *AWS* (*Amazon Web Service*) (*Cf. §2.1.9.1*) notamment avec le service *EC2* (*Elastic Compute Cloud*) qui permet de déployer et d'exécuter des machines virtuelles de manière dynamique, à la demande. Ces machines virtuelles sont accessibles depuis n'importe où, via le réseau public. Le client utilisera l'interface web pour gérer son espace de stockage à travers un navigateur web. Il aura besoin également d'un poste de travail configuré dans un environnement de type « *Gnu/Linux* » pour accéder aux données. Ce poste de travail devra contenir l'application cliente de *Rozo* (« *Rozofsmount* ») qui permet de monter un espace de stockage distant dans un répertoire local afin de gérer le système de fichiers comme un répertoire classique.

3.2.2 Les usages

L'offre de service est adaptée à toutes les applications métiers qui ont besoin d'un espace de stockage unifié (*CRM, SAP, Workflow, Serveur Web, Messagerie, etc.*).

Elle concerne plus précisément toutes personnes ou applications qui effectuent les actions suivantes :

- *Stockage de fichiers dans un volume qui peut s'étendre à volonté.*
- *Sauvegardes de données dans un espace de stockage fiable et flexible.*
- *Partage de fichiers accessibles depuis n'importe où via un accès par internet.*
- *Centralisation des données dans un espace de stockage commun.*

3.3 Organisation du projet

Le projet doit suivre une méthode de travail afin de garantir sa réussite qui se mesure par rapport à la satisfaction du client et la qualité du résultat (*conformité du produit*), livré dans le respect du délai imparti et du budget alloué.

3.3.1 Le cycle en V

L'organisation de notre projet suit une approche classique du modèle en V (*Cf. Figure 33*).

La première phase consiste à décrire les besoins du maître d'ouvrage et ses exigences en matière de conception. Ensuite, lors de la spécification, on élabore un cahier des charges que l'on devra faire valider auprès du client. On entre ensuite dans une phase de conception afin de déterminer le choix technologique, de cadrer le projet dans son environnement puis de définir le modèle architecturale et fonctionnel du produit.

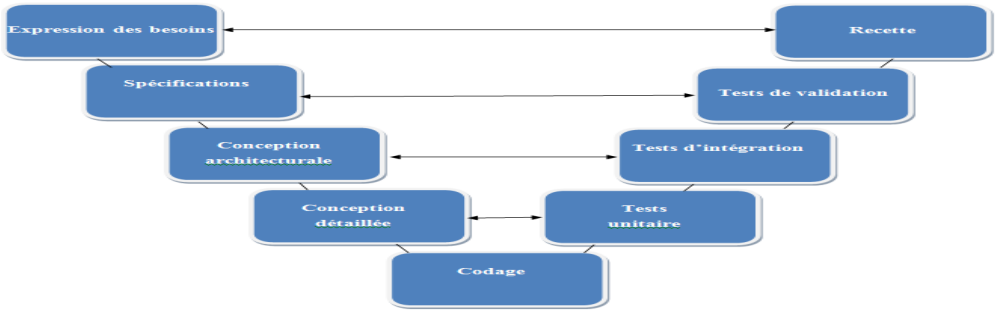


Figure 33 – *Cycle en V.*

Une fois le modèle conceptuel élaboré, on aborde la phase de réalisation du projet avec le développement de l'interface web de gestion. Ensuite, on vérifie que l'interface est fonctionnelle et bien conforme aux exigences du client.

3.3.2 Découpage en sous-projets

L'interface web de gestion est décomposée en trois sous-projets *(Cf. Figure 34)* réalisés successivement.

Figure 34 – *Décomposition de l'interface.*

Chaque sous-projet conduit à livrer une application web qui fonctionne de manière indépendante.

Ainsi le maître d'ouvrage pourra bénéficier au fur et à mesure de l'avancement du projet, d'une partie des fonctionnalités attendues. Il pourra ainsi tester l'application et suggérer des améliorations. Finalement, le produit final consiste à réunir l'ensemble des applications pour en former une seule lors de la phase d'intégration.

3.3.2.1 Réalisation de l'interface Amazon sous Django

Dans un premier temps, on réalisera l'interface « *Amazon* » (*Cf. Figure 35*) qui consiste à intégrer les *APIs* dans une application web sous *Django*. Cette interface doit permettre de fournir plusieurs machines virtuelles nécessaires pour installer la solution de *Cloud Storage* avec les applications de Fizians.

Figure 35 *- Entrées/Sorties de l'interface « Amazon ».*

3.3.2.2 Réalisation de l'interface Rozo sous Django

Ensuite, on réalisera l'interface « *Rozo* » *(Cf. Figure 36*) qui consiste à intégrer les *APIs* dans une application web avec *Django*. Cette interface suppose que l'environnement de travail soit opérationnel avec au moins un serveur de méta-données puis quatre serveurs de stockages.

Cet environnement de travail doit être accessible à partir de l'application web.

De plus, les fonctions *Rozo* (*APIs*) développées en Python doivent être fournies en entrée afin que l'application web Django puisse les intégrer pour offrir un outil de gestion des services *Rozo*.

Figure 36 *- Entrées/Sorties de l'interface « Rozo ».*

3.3.2.3 Réalisation de l'interface Primaire sous Django

Puis on entamera la réalisation de l'interface « *Primaire* » (*Cf. Figure 37*) qui servira de passerelle entre les interfaces « *Amazon* » et « *Rozo* ». Cette interface doit permettre l'authentification d'un utilisateur puis l'accès à une page d'accueil. En entrée, l'utilisateur saisit ses identifiants pour se connecter à l'interface web. Lors de la première utilisation, il a la possibilité de créer un nouveau compte et d'enregistrer les paramètres dans une base de données. En sortie, l'utilisateur authentifié doit pouvoir accéder à une page d'accueil.

Figure 37 *- Entrées/Sorties de l'interface « Primaire ».*

3.3.2.4 Réalisation de l'interface finale

On procédera ensuite à une phase d'intégration qui consiste à élaborer une interface unique à partir des trois interfaces précédemment décrites (*Cf. Figure 38*).

Figure 38 *- Entrées/Sorties de l'interface finale.*

3.3.3 Planification du projet

L'organisation d'un projet nécessite également d'être planifiée afin de pouvoir livrer le produit à temps. La planification permet de découper le projet en tâches et de les ordonnancer en respectant un planning.

3.3.3.1 Planification prévisionnelle

Une première planification est initialisée au début du projet et liste chronologiquement une série de tâches nécessaires pour aboutir à la réalisation du produit final (*Cf. Tableau 1*).

Tableau I – Ordonnancement prévisionnel des tâches.

N°	Remarques	Nom de la tâche	Durée	Début	Fin	Prédécesseurs
1	Etude	Recherche bibliographique autour du Cloud Computing	20 jours	Mar 01/02/11	Lun 28/02/11	
2	-	Recherche bibliographique sur les systèmes de fichiers distribués	24 jours	Mar 01/03/11	Ven 01/04/11	
3	-	Recherche bibliographique sur les architectures web	20 jours	Lun 04/04/11	Ven 29/04/11	
4	Faisabilité	Préparation env KVM (Rozo + Python)	5 jours	Lun 02/05/11	Ven 06/05/11	
5	-	Mise en œuvre Rozofs sur KVM	4 jours	Lun 09/05/11	Jeu 12/05/11	4
6	-	Utilisation APIs Rozo sous Python	6 jours	Ven 13/05/11	Ven 20/05/11	5
7	-	Developpement de fonctions EC2 Amazon sous Python (APIs fournies)	15 jours	Lun 23/05/11	Ven 10/06/11	4
8	Spécification	Cahier des charges	18 jours	Lun 13/06/11	Mer 06/07/11	6;7
9		Validation cahier des charges	2 jours	Jeu 07/07/11	Ven 08/07/11	8
10	Conception sous-projet 1	Développement interface Django pour l'utilisation des services EC2 d' Amazon	33 jours	Lun 11/07/11	Mer 24/08/11	9
11		Test/validation interface	2 jours	Jeu 25/08/11	Ven 26/08/11	10
12	Conception sous-projet 2	Développement interface Django pour la gestion des services Rozo	13 jours	Lun 29/08/11	Mer 14/09/11	9
13		Test/validation interface	2 jours	Jeu 15/09/11	Ven 16/09/11	12
14		Congé	15 jours	Lun 01/08/11	Ven 19/08/11	
15	Conception sous-projet 3	Développement interface Django pour l'authentification des utilisateurs	8 jours	Lun 19/09/11	Mer 28/09/11	9
16		Test/validation interface	2 jours	Jeu 29/09/11	Ven 30/09/11	15
17	Intégration des sous-projets	Intégration interface1 à 3	8 jours	Lun 03/10/11	Mer 12/10/11	11;13;16
18		Test et validation	2 jours	Jeu 13/10/11	Ven 14/10/11	17
19	Livraison	Recette	2 jours	Lun 17/10/11	Mar 18/10/11	18
20		Livrables	3 jours	Mer 19/10/11	Ven 21/10/11	19
21	Assistance utilisateurs	Transfert de compétence	3 jours	Lun 24/10/11	Mer 26/10/11	20

Description des tâches :

- *Etude*

Une partie du projet consiste à effectuer un état de l'art autour des technologies permettant de mieux appréhender la conception de notre interface web de gestion.

- *Faisabilité*

Ensuite, une phase de préparation s'amorce afin d'étudier la faisabilité de réaliser l'interface à partir des éléments fournis. Il s'agira dans un premier temps de préparer une plateforme de test et d'exploiter les fonctions Rozo (APIs) et celles d'Amazon. Cette plateforme est constituée de cinq machines virtuelles déployées avec l'outil de virtualisation KVM (Kernel Virtual Machine) qui s'installe dans un serveur local mis à disposition. On reconstituera ainsi l'architecture Rozo avec un serveur de méta-données et quatre serveurs de stockage.

- *Spécification*

A partir de là, nous aurons une meilleure connaissance des interactions que doit avoir l'application avec son environnement extérieur. On pourra donc procéder à la rédaction d'un cahier des charges qui permettra de définir le périmètre de notre projet dans le respect des exigences du maître d'ouvrage (Fizians).

- *Conception*

Puis on procèdera à la conception de l'interface web de gestion en trois sous-projets que nous avons décrit précédemment.

- *Intégration*

La phase d'intégration permet ensuite de regrouper les sous-projets pour former le projet final qui fera l'objet d'une série de tests avant d'être validé.

- *Recette et livraison*

La recette est la dernière étape avant de livrer le produit et permet de s'assurer de la conformité du produit vis-à-vis du client.

- *Assistance utilisateurs*

Afin que le client puisse utiliser le produit et y apporter des améliorations, il est important de fournir de la documentation (PV de recette, manuel utilisateurs, manuel d'exploitation) puis de procéder à une phase d'accompagnement.

3.3.3.2 Gantt prévisionnel

Le diagramme de Gantt prévisionnel (*Cf. Figure 39*) nous permet de visualiser dans le temps les diverses tâches qui constituent une première ébauche du projet. Le projet est réparti sur 9 mois avec 1/3 d'études, 1/3 développement et 1/3 partagé avec les phases de faisabilité, de spécification puis d'intégration.

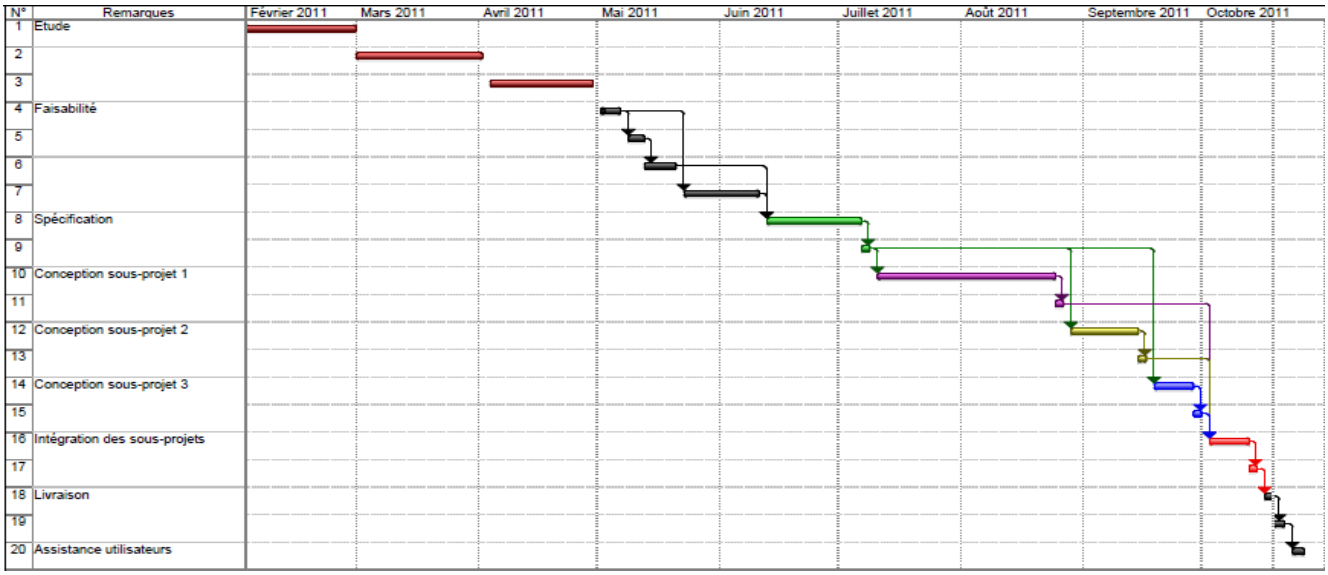


Figure 39 *- Diagramme de Gantt prévisionnel.*

3.3.3.3 Planification réelle

La planification est initialisée au début du projet et mise à jour pendant toute sa durée de vie.

Au fur et à mesure de l'avancé du projet, on optimise la méthode de conception en ajoutant des tâches intermédiaires. La condition est de respecter la date de livraison du produit final.

En comparaison avec le découpage précédent lors de la planification prévisionnelle, certaines tâches ont été ajoutées notamment dans la partie « ***Etude*** », « ***Spécification*** », « ***Conception générale*** » puis au niveau « ***Intégration*** » et « ***Déploiement*** ».

Tableau II – Ordonnancement réel des tâches.

N°	Remarques	Nom de la tâche	Durée	Début	Fin
1	Etude	Recherche bibliographique autour du Cloud Computing	20 jours	Mar 01/02/11	Lun 28/02/11
2	-	Exploitation des Services Web Amazon	4 jours	Mar 01/03/11	Ven 04/03/11
3	-	Recherche bibliographique sur les systèmes de fichiers distribués	15 jours	Lun 07/03/11	Ven 25/03/11
4	-	Etude de l' architecture Rozo	5 jours	Lun 28/03/11	Ven 01/04/11
5	-	Recherche bibliographique sur les architectures web	20 jours	Lun 04/04/11	Ven 29/04/11
6	-	Etude du langage Python	5 jours	Lun 02/05/11	Ven 06/05/11
7	-	Etude du cadriciel Django	15 jours	Lun 09/05/11	Ven 27/05/11
8	Faisabilité	Préparation environnement avec KVM (Python+ Rozo installé)	2 jours	Lun 30/05/11	Mar 31/05/11
9	-	Exploitation APIs Rozo sur l'environnement de travail	3 jours	Mer 01/06/11	Ven 03/06/11
10	-	Exploitation des APIs Amazon sur l'environnement de travail	5 jours	Lun 06/06/11	Ven 10/06/11
11	Spécification	Elaboration du cahier des charges	8 jours	Lun 13/06/11	Mer 22/06/11
12	-	Validation du cahier des charges	2 jours	Jeu 23/06/11	Ven 24/06/11
13	-	Modelisation interface web	5 jours	Lun 27/06/11	Ven 01/07/11
14	-	Elaboration MCD	2 jours	Lun 04/07/11	Mar 05/07/11
15	Conception générale	Création d'un environnement de travail avec Django, Apache et MySQL	1 jour	Mer 06/07/11	Mer 06/07/11
16	-	Création schéma de données	1 jour	Jeu 07/07/11	Jeu 07/07/11
17	-	Configuration Apache-SSL	1 jour	Ven 08/07/11	Ven 08/07/11
18	Conception sous-projet 1	Développement interface Django pour l'utilisation des services EC2 d'Amazon	36 jours	Lun 11/07/11	Lun 29/08/11
19	-	Test/validation interface	2 jours	Mar 30/08/11	Mer 31/08/11
20	Conception sous-projet 2	Développement interface Django pour la gestion des services Rozo	15 jours	Jeu 01/09/11	Mer 21/09/11
21	-	Test/validation interface	2 jours	Jeu 22/09/11	Ven 23/09/11
22	Conception sous-projet 3	Développement interface Django pour l'authentification des utilisateurs	4 jours	Lun 26/09/11	Jeu 29/09/11
23	-	Test/validation interface	1 jour	Ven 30/09/11	Ven 30/09/11
24	Intégration des sous-projets	Intégration interface1 à 3	3 jours	Lun 03/10/11	Mer 05/10/11
25	-	CSS - Menus de navigation	3 jours	Jeu 06/10/11	Lun 10/10/11
26	-	Test et validation	1 jour	Mar 11/10/11	Mar 11/10/11
27	Préparation produit final	Déploiement chez Amazon	1 jour	Mer 12/10/11	Mer 12/10/11
28	-	Test et Validation	1 jour	Jeu 13/10/11	Jeu 13/10/11
29	Livraison	Recette	2 jours	Ven 14/10/11	Lun 17/10/11
30	-	Livrables	5 jours	Mar 18/10/11	Lun 24/10/11
31	Assistance utilisateurs	Transfert de compétence	2 jours	Mar 25/10/11	Mer 26/10/11

Description des évolutions :

- *Etude*

*Dans la partie « **Etude** », nous avons réservé un peu de temps pour la compréhension de l'architecture Rozofs juste après la phase de recherche sur les systèmes de fichiers distribués.*
Le choix technologique pour la réalisation de l'interface impose de consacrer une partie de l'étude dans la compréhension et la manipulation du langage Python et de son cadriciel Django juste après la phase de recherche sur les architectures web.

- *Spécification*

*En ce qui concerne la « **Spécification** », nous avons ajouté une tâche pour la modélisation de l'interface web puis une autre pour l'élaboration d'un modèle conceptuel des données.*
La modélisation de l'interface consistera à réaliser des diagrammes UML afin de s'assurer que l'on est conforme dans les fonctions attendues du maître d'ouvrage. Tandis que le modèle conceptuel de données servira pour créer le schéma de la base de données nécessaire pour l'application Django.

- *Conception générale*

Cette partie qui ne figure pas dans l'ordonnancement prévisionnel des tâches, est consacrée sur la mise en œuvre et la configuration d'un environnement de travail en local avec les applications Python, Django, Apache, Mysql, OpenSSL et Rozofs. Cet environnement est indispensable pour faire fonctionner les trois applications Django (Cf. §212). On trouve aussi dans cette partie une tâche pour la création de la base de données avec l'application Django qui fait l'objet d'une attention particulière. Puis il y a également une autre tâche consacrée à la configuration du serveur web Apache avec Django en y apportant une couche de sécurité fournie par SSL avec les certificats d'authentification.

- *Intégration*

Dans la partie intégration, nous avons ajouté une tâche pour l'ajout de feuilles de styles afin d'apporter une amélioration visuelle et un certain confort dans l'utilisation de l'interface web de gestion.

- *Produit final*

Cette partie regroupe une tâche prévue pour le déploiement de notre interface web de gestion d'un serveur local vers une machine virtuelle hébergée chez Amazon.
Une image de la machine virtuelle pourra être réalisée ensuite et constituera le produit à l'état final.

3.3.3.4 Gantt réel

Le diagramme de Gantt (*Cf. Figure 40*) montre l'ordonnancement des tâches effectuées pour réaliser le projet dans les délais impartis. En comparaison avec la planification prévisionnelle, la partie « ***Etude*** » occupe quatre mois au lieu de deux, tandis que la partie « ***Faisabilité*** » occupe un demi mois contre un mois et demi initialement prévu. D'autre part, la partie « ***Spécification*** » est légèrement réduite pour ajouter la nouvelle phase intitulée « ***Conception générale*** ». Le temps alloué à la conception des trois interfaces reste quasiment inchangé par rapport au planning prévisionnel. En revanche, la partie « ***Intégration*** » est également écourtée pour ajouter la nouvelle étape intitulée « ***Préparation produit final*** » qui est destinée à déployer l'interface web de gestion chez Amazon.

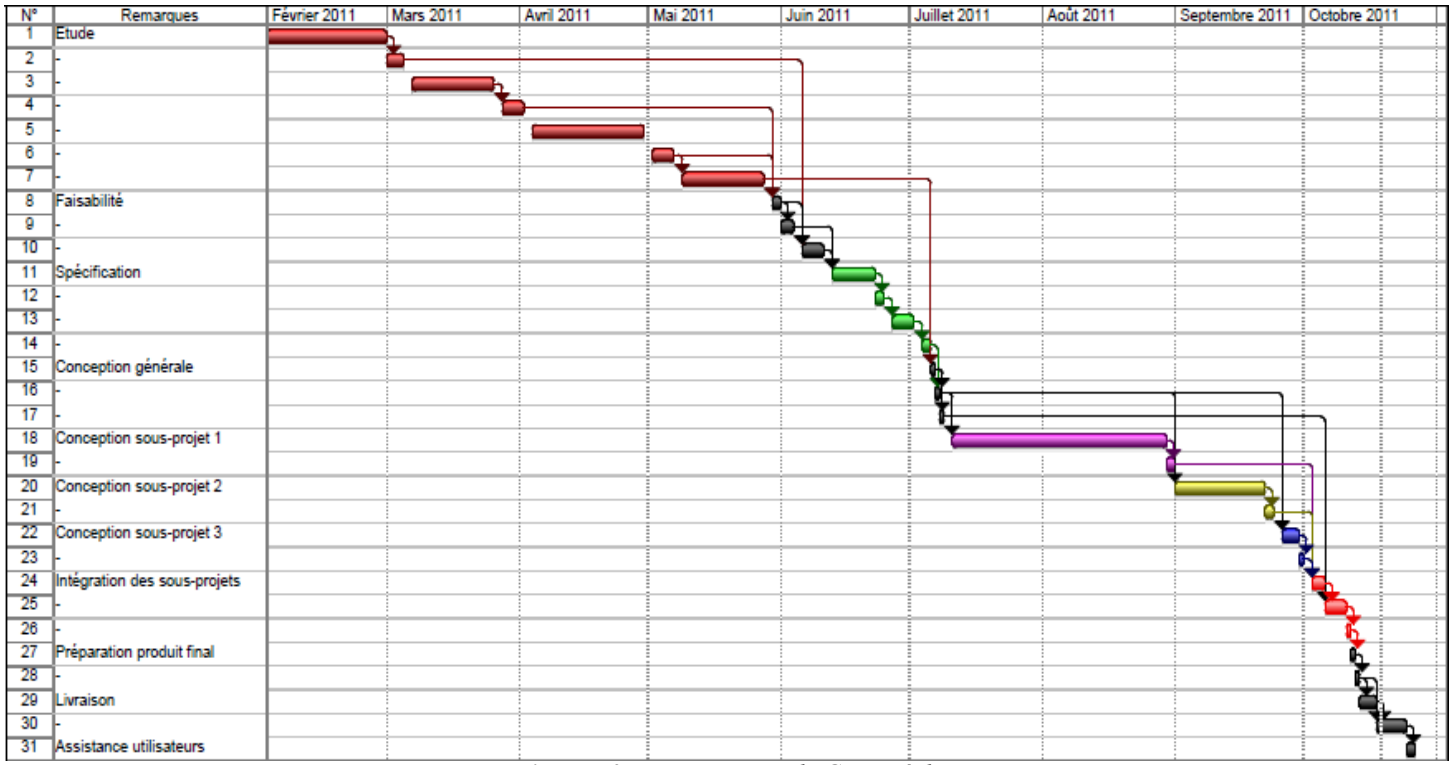


Figure 40 - *Diagramme de Gantt réel.*

3.4 Spécification du projet

Une fois que les différentes étapes ont été définies et planifiées, il faut tenir compte des exigences du maître d'ouvrage, des contraintes liées à l'environnement afin de déterminer les choix techniques et définir le cahier des charges.

3.4.1 Positionnement de l'interface dans son environnement global

L'interface web de gestion d'un ***Cloud Storage*** qui constitue l'objectif de réalisation, se situe en tête de l'architecture globale du système ***Rozofs*** que Fizians souhaite proposer à ses clients (*Cf. Figure 41*). Elle pilote à la fois les interfaces de programmation (***APIs***) d'Amazon pour l'utilisation des services AWS puis celles de Rozo pour la gestion du stockage.

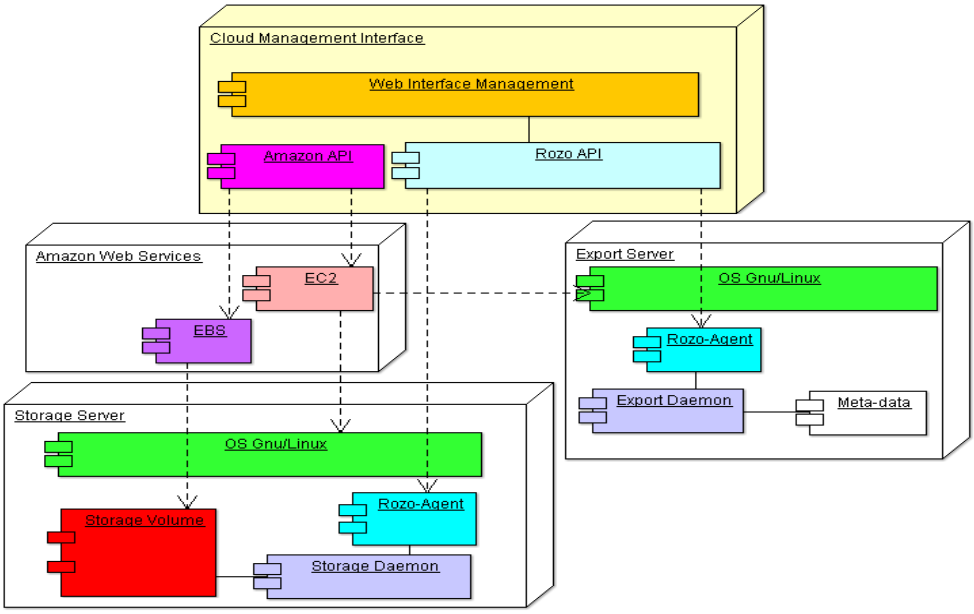


Figure 41 *- Architecture globale de gestion du Cloud Storage.*

3.4.2 Spécification technique de l'interface

Le choix technique est une des étapes les plus importantes du projet, car il en va de sa réussite.

En effet, la structure du produit dépend des technologies que l'on utilise. Il ne faut pas choisir une technologie trop ancienne au risque d'élaborer un produit rapidement désuet.

Cependant, il faut choisir une technologie en mesure d'intégrer les *APIs* d'Amazon pour la gestion des machines virtuelles et celles de *Rozo* pour la gestion du stockage. Une fois ce choix déterminé, la construction découlera d'elle-même à la manière d'un puzzle.

3.4.2.1 Choix de l'architecture

L'objectif du projet est de concevoir une interface web de gestion qui utilise les services *EC2* et *EBS* d'Amazon puis les services *Rozo*. Cette interface fonctionne dans une machine virtuelle hébergée chez Amazon pour le compte de Fizians. La machine virtuelle est un serveur qui est constitué d'un système d'exploitation et de plusieurs applications formant ainsi une architecture 3-tiers (*Cf. Figure 42*).

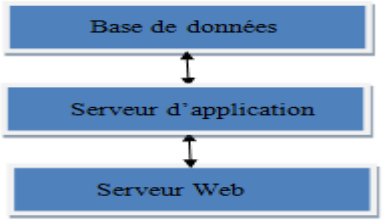


Figure 42 *- Architecture 3-tiers.*

Les clients pourront accéder à l'interface via un navigateur web, mais ils devront posséder au préalable un compte chez Amazon pour utiliser les services. L'interface sera constituée d'un serveur web sécurisé et accessible depuis le web public puis une base de données et un interpréteur de langage de programmation web. Les fonctions d'Amazon et de Rozo seront intégrées directement dans l'interface web et prises en charge par l'interpréteur (*Cf. Figure 43*).

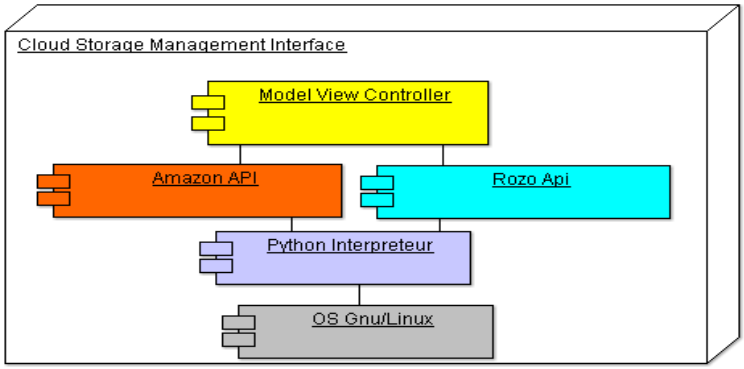


Figure 43 *- Architecture de l'interface web de gestion.*

3.4.2.2 Choix du serveur d'application

Le serveur d'application a pour rôle de déclencher les opérations nécessaires pour satisfaire les demandes utilisateurs formulées via le serveur web. Il doit en l'occurrence piloter les services Rozo dont les fonctions sont développées en langage Python. D'ailleurs, ces fonctions sont disponibles sous forme de paquetages logiciels compatibles avec le système d'exploitation Linux Debian via le gestionnaire de paquets (*Format .deb*).

Ainsi le choix du système d'exploitation et celui du langage de développement web sont clairement définis de part les contraintes d'utilisation des services Rozo.

Finalement, le serveur d'application devra être constitué d'un interpréteur Python et être installé sur le système d'exploitation Debian. On choisira à cet effet le cadriciel Django (*Cf. §2.3.5.3*) qui repose sur le modèle MVC et permet d'élaborer une application web en langage python.

Ce choix de langage impose d'utiliser des *APIs* Amazon développées en Python.

Or, Amazon fournit au sein de sa communauté, des fonctions écrites dans plusieurs langages comme *Java*, *Php*, *Ruby*, *Windows.NET* et *Python*. Ces fonctions sont ouvertes aux développeurs afin d'intégrer des offres de services Amazon dans les applications web.

Django est disponible en open-source sous la licence BSD qui autorise son intégration dans un logiciel libre ou propriétaire avec l'obligation de conserver le copyright dans toutes publicités ou documents fournis.

3.4.2.3 Choix du serveur web

Le serveur web doit interagir avec le serveur d'application Django dans un système d'exploitation Linux Debian avec le langage Python. On utilisera pour cela le serveur Apache qui est sans aucun doute le serveur web le plus populaire dans un environnement Gnu/Linux.

- *Apache dispose d'une librairie Python (mod_python) pour interpréter le langage.*
- *Il peut intégrer le protocole SSL afin de sécuriser l'échange de données par l'intermédiaire de certificats.*
- *Il s'interface très bien avec le serveur d'application Django.*
- *La licence Apache est une licence de logiciel libre et open-source compatible GPL qui fixe les conditions légales de distributions des logiciels libres du projet GNU.*

3.4.2.4 Choix du serveur de base de données

L'intérêt de disposer d'une base de données est de pouvoir enregistrer, de mettre à jour toutes les données que l'interface web utilisera pour afficher l'information. Les données sont conservées dans le temps et actualisées pour rester cohérentes avec le contexte de l'utilisateur.

Ceci est d'autant plus vrai que Django est conçu pour élaborer des applications web reposant sur une base de données. Il permet de communiquer avec la base de données via le langage Python.

On utilisera le serveur de base de données MySQL qui s'interface bien avec le serveur web Apache et le serveur d'application Django.

3.4.2.5 Bilan du choix technologique

Le choix technologique pour l'élaboration de l'interface web de gestion est restreint par l'utilisation des services *Rozo* notamment avec le langage Python. Or, le fait de s'obliger à utiliser une technologie particulière conditionne le choix des autres pour assurer ainsi une cohésion dans le fonctionnement global du système.

Ainsi à travers une analyse des besoins et une description des contraintes techniques, on aboutit à l'organisation logicielle (*Cf. Figure 44*) de notre interface.

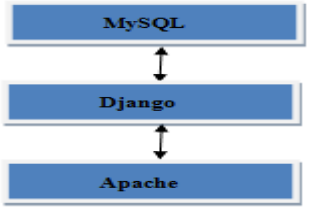


Figure 44 *– Applications de l'interface web de gestion.*

En suivant ce choix technique, l'interface doit être en mesure de communiquer avec les services web d'Amazon afin de déployer l'ensemble des machines virtuelles qui constituera l'architecture du *Cloud Storage*. De la même manière, elle doit intégrer facilement les services *Rozo* pour configurer l'espace de stockage au sein de l'environnement précédemment créé.

Tout cela est rendu possible grâce à l'utilisation du langage Python qui est commun dans l'usage des différents services (*Amazon et Rozo*). Cependant, il existe d'autres contraintes extérieures au système dont il faut tenir compte afin de garantir un bon fonctionnement. Il s'agit notamment des contraintes liées à l'environnement Amazon puis celles issues de l'architecture Rozo.

3.4.3 Spécification des services AWS d'Amazon

L'interface web de gestion doit intégrer les *APIs* d'Amazon et plus précisément celles qui permettent de déployer des machines virtuelles chez l'hébergeur. Pour prendre en charge ces fonctions, il est nécessaire de bien comprendre l'offre Amazon et les contraintes dans l'utilisation des services.

3.4.3.1 Périmètre d'usage des services web d'Amazon

Amazon propose de nombreux services web (*Cf. §2.1.9.1*) accessibles via le port *HTTP* par l'intermédiaire du protocole de communication *REST* ou *SOAP* et facturés en fonction de l'utilisation. L'intérêt est d'utiliser ces services dans une application web afin de fournir à l'utilisateur un haut niveau d'abstraction et de simplifier les usages. Parmi l'ensemble des services proposés, notre projet s'intéressera au déploiement et la configuration de machines virtuelles (*Service EC2*) puis l'ajout de volumes de stockage (*Service EBS*).

3.4.3.2 La bibliothèque AWS d'Amazon sous Python

Amazon fournit des APIs disponibles sous le langage Python au sein de sa communauté d'AWS.

Pour Python, il s'agit la bibliothèque « *Boto* » *[Garnaat 2010]* qui prend en charge la plupart des services dont *EC2* et *EBS*. Bien que Python possède une grande bibliothèque standard, il est possible de l'étendre par l'ajout de modules spécifiques développés en C ou en Python.

La bibliothèque « *Boto* » est installée et utilisable avec l'interpréteur Python par l'appel des fonctions.

3.4.3.3 Le déploiement d'un environnement Rozo chez Amazon

L'interface web de gestion doit déployer des machines virtuelles qui contiennent toutes les applications nécessaires pour mettre en œuvre la solution complète de Rozo *(Rozo-export, Rozo-storage et Rozofsmount*). Toutes ces machines virtuelles proviennent de la même image (*Template*) désignée chez Amazon par le sigle *AMI* (*« Amazon Machine Image »*).

L'image contient le système d'exploitation *Debian* et le système de fichier *Rozofs*.

Tant que l'image est à disposition d'un utilisateur, ce dernier peut instancier autant de machines virtuelles qu'il souhaite. Une fois que la machine virtuelle est exécutée, il peut ensuite ajouter un volume de stockage (*EBS*) qui permet de conserver les données utilisateurs.

En effet, les machines virtuelles par défaut ne disposent pas d'espace de stockage permanent.

Ainsi lorsque la machine redémarre, les données utilisateurs sont perdues.

3.4.3.4 Les contraintes des services web Amazon

Les services Amazon engendrent un certain nombre de contraintes qu'il convient de connaitre pour pouvoir ensuite les intégrer dans l'application web de gestion du *Cloud Storage*.

a) Console AWS d'Amazon

Pour utiliser les services AWS, il est indispensable de créer un compte chez le fournisseur de services. Les frais sont occasionnés uniquement à l'utilisation des services.

Pour créer un compte, utiliser les services web ou récupérer des informations, Amazon met à disposition une console de gestion accessible sur internet avec un navigateur web (*Cf. Figure 45*).

Figure 45- *Console de gestion AWS accessible sur internet via un navigateur.*

b) Console AWS d'Amazon

Pour assurer une cohérence avec la base de données de l'interface web de gestion, l'utilisateur se dispensera d'utiliser la console AWS pour mettre en œuvre ou gérer les machines virtuelles d'une architecture Rozo. Cette console est utilisable uniquement pour gérer tout autres environnements que celui de Rozo. Elle permet également de récupérer des informations sur le compte d'un utilisateur, d'obtenir des informations sur ses machines virtuelles, les services à disposition, etc.

c) Les contrôles d'accès des services web Amazon

Dans notre application, la console AWS est indispensable pour créer un compte Amazon et obtenir les identifiants afin d'utiliser les services web d'Amazon via les *APIs*.

Ces informations sont des justificatifs d'identité qui renforcent l'accès aux applications d'Amazon. Pour récupérer ces informations, il faut se connecter à la console de gestion avec les paramètres de son compte utilisateur (*Cf. Figure 46*).

Figure 46 *- Les différents identifiants nécessaires pour utiliser les services AWS.*

Ces justificatifs sont au nombre de trois dont la clé d'accès, le certificat X.509 puis une paire de clé SSH.

- *La clé d'accès concerne la sécurisation des requêtes de protocole REST pour l'utilisation des services API d'Amazon. Cette clé est formée d'un identifiant de clé d'accès ainsi qu'un identifiant de clé d'accès secrète.*
- *Le certificat X.509 concerne la sécurisation des requêtes de protocole SOAP.*
- *La paire de clé SSH est indispensable pour utiliser une machine virtuelle. L'utilisateur devra utiliser la clé privée pour se connecter à la machine virtuelle qui disposera de la clé publique.*

En ce qui nous concerne, les requêtes se feront avec le protocole *REST*, on n'utilisera donc pas les certificats *X.509* avec le protocole *SOAP*. En revanche, on disposera de la clé d'accès pour s'authentifier chez Amazon puis de la clé *SSH* pour se connecter aux machines virtuelles.

d) Configuration réseau

La configuration réseau qui est automatique et non modifiable est exclusivement gérée par Amazon. Le fournisseur associe à la machine virtuelle, des adresses *DNS* et *IP* publiques ainsi qu'une paire de clés SSH afin que l'utilisateur puisse se connecter à la machine après le déploiement de *l'AMI*. Amazon fournit également des adresses *DNS* et *IP* privées pour accéder à la machine depuis le réseau Amazon (*via une machine virtuelle hébergée chez Amazon*).

La clé SSH privée qui est stockée chez le client, est nécessaire pour accéder à la machine virtuelle.
La commande suivante permet de se connecter à une machine virtuelle depuis l'extérieur de l'environnement Amazon via une *adresse DNS publique*.

"ssh -i saas.pem ubuntu@ *ec2-50-16-160-110.compute1.amazonaws.com*"

Cependant, au redémarrage de la machine, l'adresse *DNS* publique (*tout comme l'adresse DNS privée*) est modifiée. Il faut donc récupérer la nouvelle adresse à partir de la console de gestion Amazon (*Cf. Figure 47*).

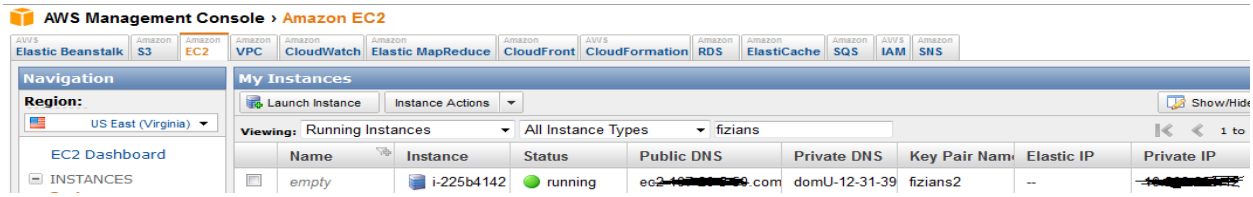


Figure 47 *– Console de gestion Amazon EC2.*

Toutefois, pour plus de souplesse, il est possible d'utiliser une adresse *IP* fixe avec le service « *Elastic IP Address* **».** Cette *adresse IP publique* permet de se connecter à une machine depuis l'extérieur même après son redémarrage. L'accès à la machine depuis l'extérieur se fera alors toujours d'une manière identique :

"ssh -i saas.pem ubuntu@*174.129.200.116*"

En revanche, les adresses *IP* et *DNS* privées auront changées et il faudra obligatoirement se connecter à la console de gestion pour connaitre les nouvelles adresses.

e) Tarification des services Amazon

La tarification est basée sur l'utilisation des différents services *AWS*.
Par exemple, le service *EC2* est basé sur le nombre d'heures que la machine fonctionne et la somme de données transférées (*émission ou réception*). Il n'y a pas de tarif pour transférer les données depuis et vers les services AWS dans une même région. La tarification EBS est basée sur la taille des volumes et les requêtes entrées/sorties.

f) Tarification d'un produit tiers utilisant les services d'Amazon

Amazon peut prendre en charge la tarification d'une application qui utilise ses services web.

En l'occurrence, l'interface web de gestion se place dans cette position et il serait intéressant d'en faire facturer l'usage comme un service quelconque d'Amazon. Pour cela, Amazon propose le service « *DevPay* » qui offre la possibilité de créer un plan de paiement personnalisé en fournissant les honoraires sur l'utilisation de l'application développée par une entreprise ainsi que l'usage des services AWS d'Amazon. Cependant, l'utilisation de « *DevPay* » nécessite pour le fournisseur de l'application, d'avoir un compte bancaire aux États-Unis. Cette solution trop contraignante oblige à trouver une méthode alternative pour assurer cette facturation.

3.4.3.5 Interactions entre l'interface web de gestion et AWS

Après avoir identifié les contraintes dans l'utilisation des services web d'Amazon et la façon de les utiliser avec les *APIs Boto* sous *Python*, on peut décrire les interactions entre l'interface web de gestion et les services *AWS* d'Amazon. La finalité consiste à mettre en œuvre un environnement composé au minimum d'un serveur de méta-données et de quatre serveurs de stockage qui disposent en plus d'un volume de disque EBS pour contenir les données.

Le diagramme de séquence (*Cf. Figure 48*) décrit les différentes étapes pour créer cet environnement.

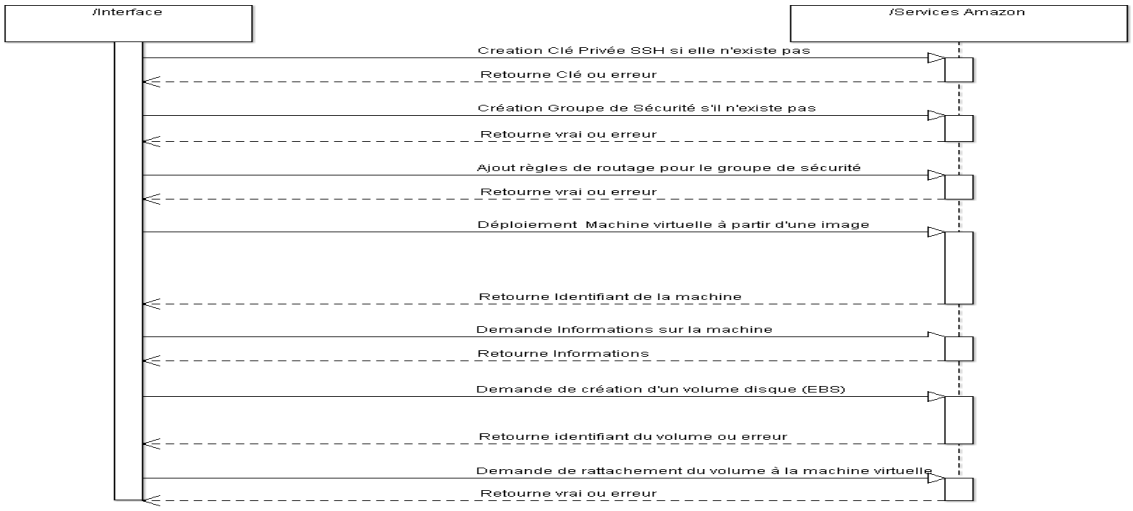


Figure 48 *- Diagramme de séquence entre l'interface et les services AWS d'Amazon.*

Chacune des requêtes émises de la part de l'interface par l'intermédiaire du client aura besoin comme arguments les identifiants de la clé d'accès (*clé d'accès et clé privée d'accès*) pour pouvoir être exploitable (*Cf. §*3.4.3.4***).***

Nous allons maintenant décrire le diagramme de séquence pour analyser les différentes étapes qui permettent la mise en œuvre complète d'un environnement Rozo.

a) Création d'une clé SSH

La première étape consiste à créer une paire de clé *SSH* en récupérant la clé privée. Cette clé est nécessaire pour se connecter à la machine virtuelle. La clé publique sera rattachée à la machine lors de son déploiement. Cette étape s'effectue la première fois lorsque l'utilisateur ne dispose d'aucune clé. Ensuite, chaque nouvelle machine utilisera la même clé publique. De la même manière, on utilisera la même clé privée pour se connecter à chaque machine.

b) Création d'un groupe de sécurité

Ensuite il faut créer un groupe de sécurité et configurer les règles d'accès. En effet, le groupe de sécurité agit comme un pare-feu qui par défaut bloque tous les protocoles *TCP* et *UDP*.

Pour pouvoir se connecter à la machine virtuelle, il faut autoriser au minimum une connexion *SSH*. Par convention, on autorisera en plus le protocole *HTTP* et *HTTPS*.

c) Déploiement de l'AMI

L'utilisateur dispose d'une image préconfigurée fournie par Fizians. Il pourra ainsi déployer les machines virtuelles nécessaires pour mettre en place son *Cloud Storage*.

Cette architecture est constituée au minimum d'un serveur de méta-données et de quatre serveurs de stockage. Chaque machine virtuelle provient de la même image, le processus de déploiement est donc identique et en série. Lorsqu'une machine est déployée, cette dernière est immédiatement opérationnelle et accessible en *SSH* par la clé privée que l'utilisateur devra avoir conservé.

Il faudra également être vigilant sur l'utilisation des adresses dite publiques ou privées :

- *Les adresses publiques sont utilisables lorsqu'une machine ou application extérieure souhaite communiquer avec une machine ou application hébergée chez Amazon.*
- *Les adresses privées concernent l'interconnexion des machines ou applications hébergées chez Amazon.*

Cependant, les informations relatives à la machine virtuelle comme l'adresse *IP* ou l'adresse *DNS*, ne sont pas disponibles avant que la machine virtuelle soit opérationnelle, ce qui peut prendre un certain délai qu'il faut prendre en compte. On dispose simplement de l'identifiant de la machine virtuelle.

Il faut donc envoyer une nouvelle requête, pour récupérer les paramètres de connexion de la nouvelle machine.

d) Création d'un volume disque EBS

Cette étape concerne uniquement les serveurs de stockage qui ont besoin d'un espace de données permanent et d'une taille déterminée. Une fois que le volume est créé, on récupère en retour son identifiant. Le volume crée n'est rattaché à aucune machine virtuelle, il convient donc d'envoyer une requête en spécifiant la référence du volume et celle de la machine virtuelle pour procéder au rattachement.

e) Attachement du volume à la machine virtuelle

Lors de la demande de rattachement du volume à la machine virtuelle, il faut également indiquer l'endroit où le volume sera rattaché à la machine (*par exemple : /dev/sda*).

Le volume doit ensuite être formaté puis monté dans un répertoire pour pouvoir être utilisé.

Le point de montage sera identique pour chaque serveur de stockage.

3.4.3.6 Bilan sur les services AWS d'Amazon

Après une phase d'étude et d'exploitation des services AWS avec les APIs *Boto* (*Cf. §3.4.3.2*) sous Python, nous avons décomposé les différentes étapes à réaliser pour élaborer l'architecture du *Cloud Storage*. Ces différentes étapes constituent des fonctions que l'on devra développer et intégrer dans notre environnement *Django*. Une fois que l'environnement *Rozo* est créé, il faut mettre en œuvre l'espace de stockage réparti entre les différentes machines virtuelles sous le contrôle du serveur de méta-données. Pour cela, l'interface web de gestion devra intégrer les *APIs Rozo* pour pouvoir configurer les différents serveurs et permettre à un client ou un programme d'accéder à son espace de stockage.

3.4.4 Spécification des services Rozo

La gestion de stockage est assurée par l'application « *Rozo* » présente dans chaque serveur virtuel. *Rozo* est un logiciel libre implémentant un système de fichiers distribués capable de stocker facilement une grande quantité de données (*Cf. §2.2.6.4*). L'architecture physique sur laquelle repose le système *Rozo* est constituée des trois éléments suivants :

- *Un serveur de méta-données sur lequel fonctionne le démon « **exportd** » ;*
- *Un ou des serveur(s) de stockage sur lesquels fonctionne le démon « **storaged** » ;*
- *Un ou des client(s) qui utilisent « **rozofsmount** ».*

3.4.4.1 Les démons Rozo

Les démons Rozo permettent de traiter les informations transmises par l'interface web de gestion puis de déclencher une action au sein de la machine virtuelle dont ils font partie.

Ainsi, on distingue le démon « *exportd* » qui tourne en tâche de fond sur le serveur de méta-données, du démon « *storaged* » disponible sur chaque serveur de stockage. Le démon « *exportd* » permet d'effectuer des tâches au sein du serveur de méta-données comme mettre à jour le fichier de configuration, relancer les services, etc. Tandis que le démon « *storaged* » gère les opérations de mises à jour du fichier de configuration, l'ajout d'un volume de stockage, l'arrêt ou le démarrage des services. Plus exactement, ce n'est pas les démons qui traitent les informations, mais plutôt un programme « *Rozo-agent* » qui fonctionne sur chaque serveur constitué de l'application « *Rozo* ».

3.4.4.2 Le rôle du serveur d'application

Toutes les machines de l'architecture *Rozo* sont identiques puisqu'elles proviennent de la même image (*AMI*), ainsi même pour le serveur de méta-données, le démon « *storaged* » fonctionne alors que cela n'est pas nécessaire. De la même manière, le démon « *exportd* » fonctionne inutilement sur chaque serveur de stockage. Cependant, notre serveur d'application pourra faire la distinction grâce aux informations stockées dans la base de données. Il sera en mesure d'identifier le serveur de méta-données et l'ensemble des serveurs de stockages associés. Ainsi le serveur d'application fera appel à l'API *Rozo* pour envoyer une requête vers l'agent *Rozo* d'un serveur particulier.

Ainsi, suivant le format de la requête *(« rozo-export » ou « rozo-storage »),* l'agent *Rozo* pilotera le démon approprié (*« export » si commande « rozo-export » ; « storage » si commande « rozo-storage »)* pour invoquer le service demandé. L'interface web (*Cf. Figure 49*) agira directement avec chaque agent par l'intermédiaire *d'APIs Rozo* en leur fournissant les paramètres nécessaires pour l'exécution des commandes.

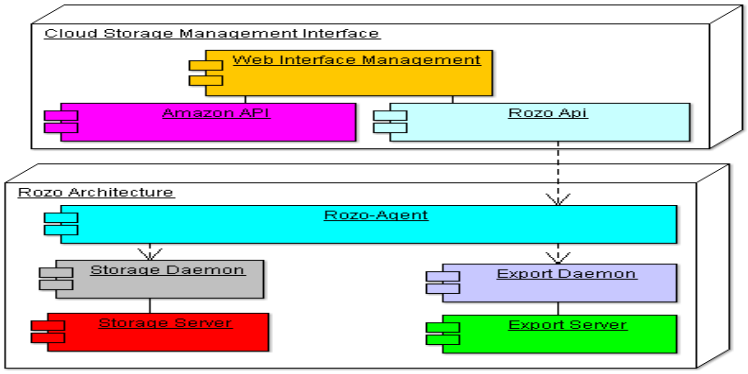


Figure 49 *– L'interface web de gestion avec l'architecture Rozo.*

3.4.4.3 Configuration de l'architecture Rozo

L'application *Rozo* permet d'élaborer un espace de stockage à partir d'un ensemble de serveurs de stockage et un serveur de méta-données. Avant qu'un utilisateur puisse accéder à son espace de stockage, il convient de suivre différentes étapes notamment dans la configuration de l'application *Rozo*. Cette configuration passe notamment dans la modification du fichier « *storage.conf* » des serveurs de stockages (« *Storage Server* ») ainsi que le fichier « *export.conf* » du serveur de méta-données (« *Export Server* ») (*Cf. Figure 50*).

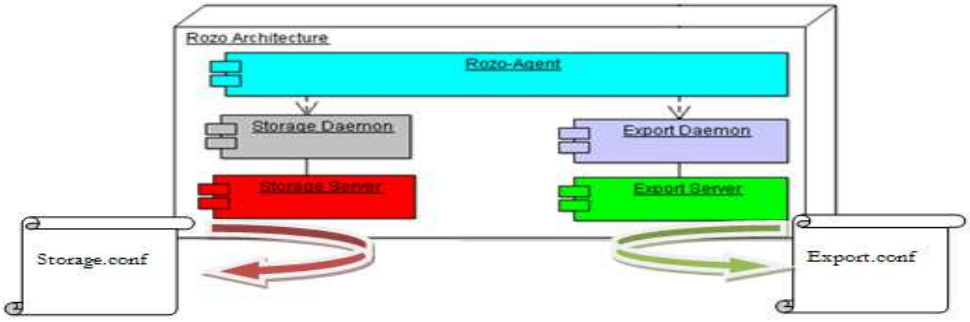


Figure 50 *- Les fichiers de configurations de l'architecture Rozo.*

Une fois que les machines virtuelles sont déployées, elles sont toutes dans le même état avec les démons « *exportd* » et « *storaged* » qui tournent en tâche de fond puis « *l'agent-rozo* » qui écoute en attendant de traiter les requêtes émises de la part de l'interface web par l'intermédiaire des APIs *Rozo*. Les fichiers de configurations sont dans un état initial (*Cf. Figure 51*).

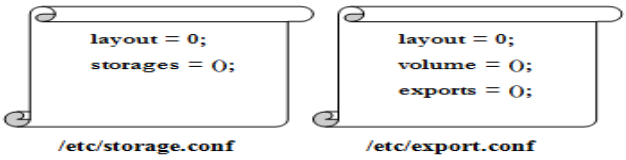


Figure 51 *– Etat des fichiers de configuration de l'architecture Rozo.*

Les principales étapes consistent à configurer les fichiers « *storage.conf* » et « *export.conf* » dans un ordre d'opération qui importe peu. On peut commencer par configurer les serveurs de stockage avant le serveur de méta-données et réciproquement. Seul le redémarrage des démons « *exportd* » et « *storaged* » finalise la procédure.

3.4.4.4 Interactions entre l'interface web de gestion et le serveur de méta-données

L'interface web de gestion devra interagir avec l'agent *Rozo* afin de préparer le serveur de méta-données et de le rendre opérationnel. Le serveur de méta-données est chargé de maintenir à jour les informations concernant la structure des fichiers (*nom complet, taille, identifiant*) ainsi que la liste des points de stockage contenant les projections nécessaires pour reconstruire le fichier.

Pour chaque fichier à reconstruire, le serveur stocke les méta-données dans un emplacement déclaré dans le fichier de configuration « *export.conf* ». Le fichier de configuration « *export.conf* » liste également chaque serveur de stockage avec un identifiant (*SID*) ainsi que son nom (*IP ou Hostname*) sur le réseau. Ainsi, l'objectif de l'interface web de gestion via l'agent Rozo est de paramétrer le fichier « *export.conf* », de créer un répertoire qui contiendra les méta-données des fichiers utilisateurs puis de prendre en compte les modifications en redémarrant le démon « *exportd* » pour forcer la relecture du fichier de configuration. Le diagramme de séquence suivant (*Cf. Figure 52*) décrit les différentes étapes pour configurer le serveur de méta-données.

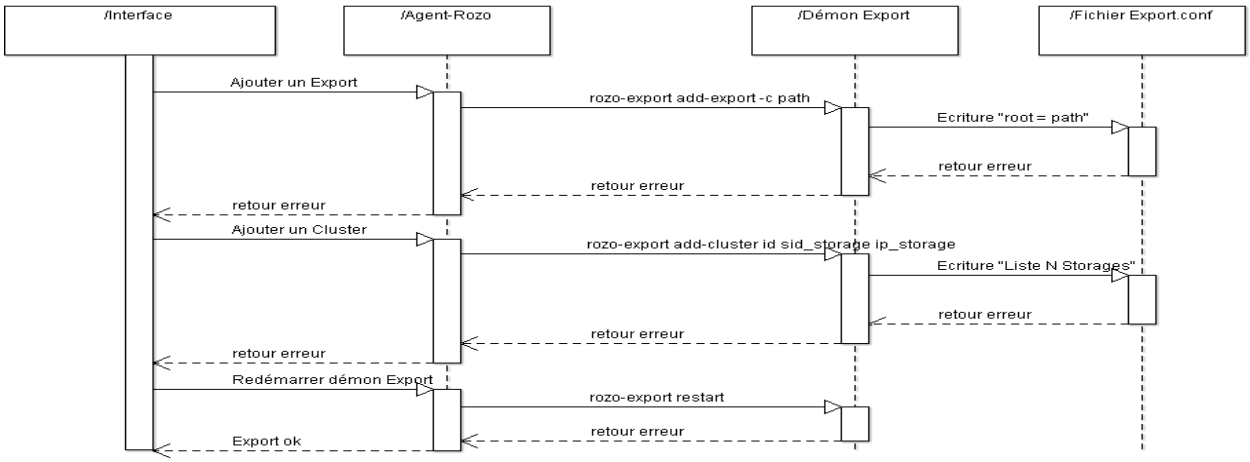


Figure 52 *- Diagramme de séquence entre l'interface et le serveur de méta-données.*

a) Informations transmises par l'interface

L'interface web de gestion fait appel à l'API *Rozo* pour transmettre des requêtes vers l'agent Rozo du serveur de méta-données. La requête est constituée d'une commande spécifique *([« rozo-export » ...])* mais également des informations émises par l'interface comme le nom et l'emplacement du répertoire de méta-données, les paramètres de connexion de l'ensemble des serveurs (*l'adresse IP privée et l'emplacement de la clé privée SSH*).

b) Emplacement des méta-données

A partir des requêtes émises par l'interface, l'agent « *Rozo* » donne des instructions au démon « *exportd* » afin de créer le répertoire contenant les méta-données puis de déclarer son emplacement dans le fichier « *export.conf* ».

« rozo-export add-export –c [emplacement des méta-données] »

c) Déclaration des serveurs de stockage (Cluster)

L'agent fournit ensuite une liste de serveurs de stockage également appelé « *cluster* » au démon « *exportd* » qui va les déclarer dans le fichier de configuration.

« rozo-export add-cluster [identifiant cluster]
[Identifiant server de stockage 1 : IP server de stockage 1]
[Identifiant server de stockage n : IP server de stockage n] »

d) Redémarrage du démon export

Pour que ces opérations soient effectives, l'agent « *Rozo* » donne l'ordre de redémarrer le démon « *exportd* ». Le serveur de méta-données est maintenant opérationnel, il faut maintenant configurer les serveurs de stockage.

3.4.4.5 Interactions entre l'interface web de gestion et les serveurs de stockage

Les serveurs de stockage contiennent une ou plusieurs projections d'un fichier utilisateur.

Les projections d'un fichier sont réparties dans plusieurs serveurs de stockage à un emplacement particulier et identique pour chaque serveur. Cet emplacement correspond à un répertoire créé dans un volume *EBS* attaché à la machine virtuelle.

Ainsi, à la demande d'un client, le serveur de méta-données sélectionne les serveurs de stockage déclarés dans le fichier de configuration « *export.conf* ». Ces serveurs sont destinés à contenir les projections, en fonction de l'espace disponible qui leur reste. Le client récupère la liste des serveurs de stockage puis les interroge afin de récupérer les projections nécessaires pour reconstruire le fichier et effectuer des opérations de lectures ou d'écritures.

Chaque serveur de stockage se configure à l'aide du fichier « *storage.conf* » qui liste l'emplacement des répertoires contenant les projections. L'emplacement du répertoire est indiqué dans le fichier « */etc/storage.conf* ».

Ainsi, l'objectif de l'interface web de gestion via l'agent *Rozo* est de paramétrer le fichier « *storage.conf* », de créer un répertoire qui contiendra les projections d'un fichier utilisateur puis de prendre en compte les modifications en redémarrant le démon « *storaged* » pour forcer la relecture du fichier de configuration. Le processus est itératif pour l'ensemble des serveurs de stockage.

Le diagramme de séquence suivant (*Cf. Figure 53*) décrit les différentes étapes pour configurer un serveur de stockage.

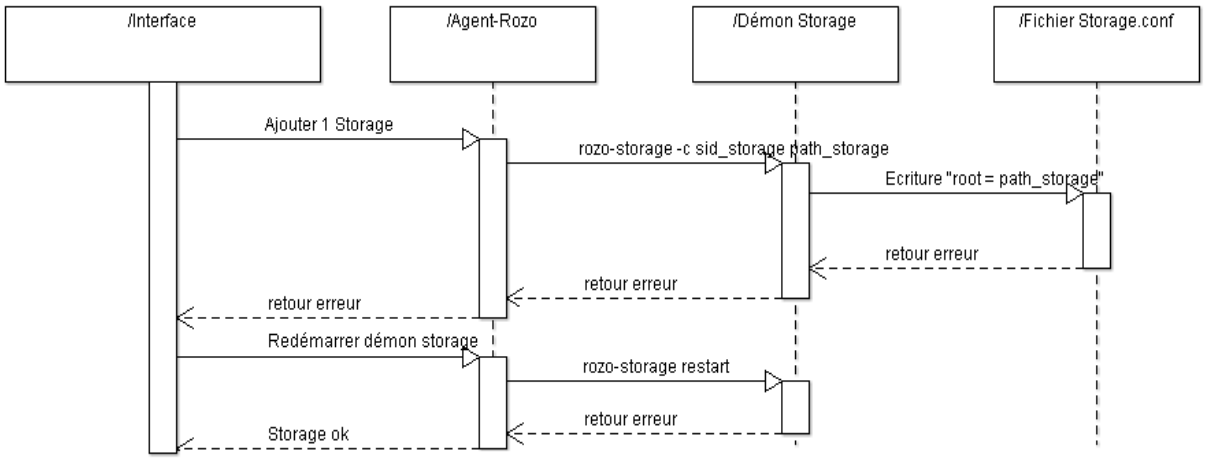


Figure 53 *- Diagramme de séquence entre l'interface et un serveur de stockage.*

a) Informations transmises par l'interface

L'interface web de gestion fait appel à *l'API Rozo* pour transmettre des requêtes vers l'agent *Rozo* de chaque serveur de stockage.

La requête est constituée d'une commande spécifique *([« rozo-storage »…])* mais également des informations émises par l'interface comme le nom et l'emplacement du répertoire contenant les projections, les paramètres de connexion de l'ensemble des serveurs *(l'adresse IP privée et l'emplacement de la clé privée SSH*).

b) Emplacement de stockage des projections

L'agent *Rozo* donne alors des instructions au démon « *storaged* » afin de créer le répertoire qui contiendra les projections puis de déclarer son emplacement dans le fichier « *storage.conf* ».

« rozo-storage –c [identifiant server de stockage] [répertoire de stockage] »

c) Redémarrage du démon Storage

Pour que ces opérations soient effectives, l'agent *Rozo* donne l'ordre de redémarrer le démon « *storaged* ».

3.4.4.6 Interaction d'un client avec l'architecture Rozo

Une fois que l'ensemble des serveurs de stockage est configuré, l'utilisateur final peut procéder au montage de l'espace de stockage puis créer, modifier des fichiers.

Pour cela, il utilisera l'application cliente de *Rozo* « *rozofsmount* ».

Le client Rozo permet de monter le système de fichier *Rozofs* dans un répertoire local en utilisant la librairie « *FUSE* » (*File system in UserSpaceE*). Ce logiciel libre permet à un utilisateur, sans privilège particulier, de créer un VFS (*Cf. §2.2.6.1*). Une fois que le système de fichier *Rozofs* est monté, l'utilisation est ensuite transparente pour l'utilisateur qui accède d'une manière classique à son répertoire de stockage.

a) Opération de création d'un fichier

Lors de l'état de l'art, nous avons décrit cette opération de création de fichier (*Cf. §*2.2.6.4).

Le diagramme (*Cf. Figure 54*) illustre les flux de données entre un client et l'architecture Rozo pour créer un fichier dans le volume de stockage.

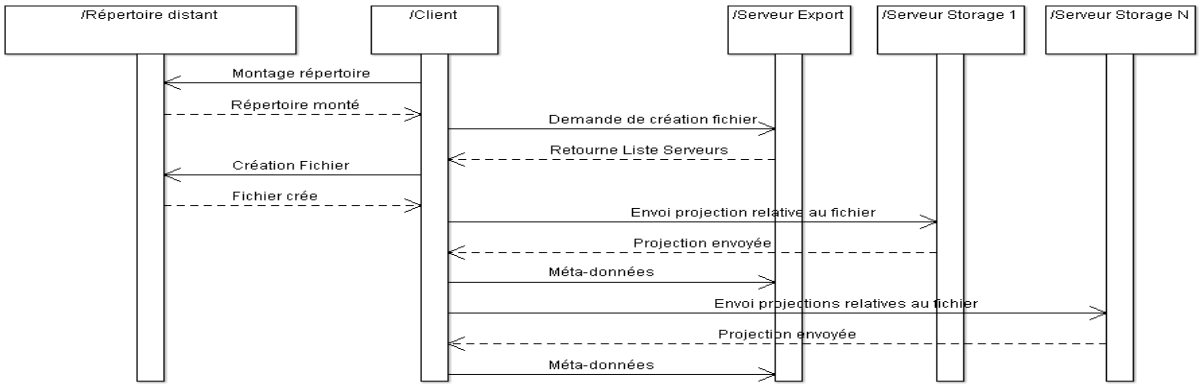


Figure 54 *- Diagramme de séquence d'une opération de création de fichier dans le volume de stockage.*

b) Opération de lecture ou modification d'un fichier

Lors de l'état de l'art, nous avons analysé cette opération de lecture de fichier (*Cf. §*2.2.6.4).

Le diagramme (*Cf. Figure 55*) illustre les flux de données entre un client et l'architecture *Rozo* pour lire ou modifier un fichier situé dans le volume de stockage.

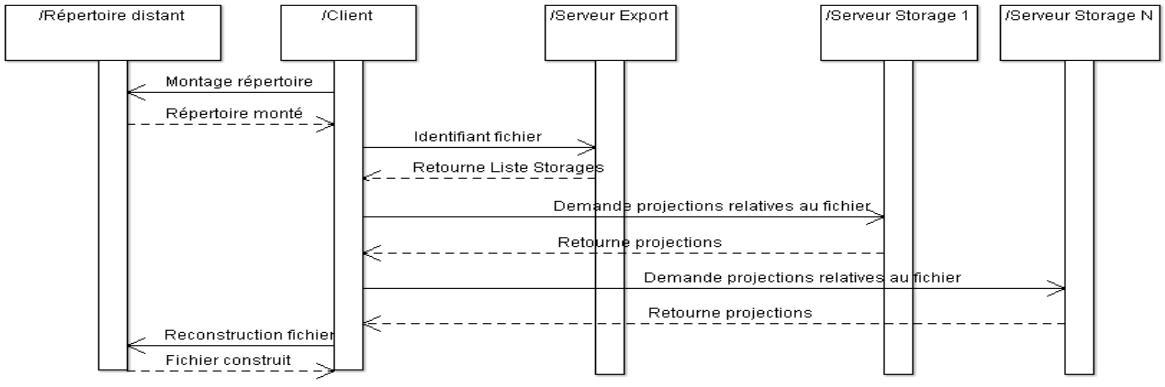


Figure 55 *- Diagramme de séquence d'une opération d'accès au fichier dans le volume de stockage.*

3.4.4.7 Bilan sur les services Rozo

Après une phase d'étude et d'exploitation des services *Rozo* sous *Python*, nous avons décomposé les différentes étapes pour créer un espace de stockage à travers plusieurs machines virtuelles.

Ces différentes étapes constituent des fonctions que l'on devra développer et intégrer dans notre environnement Django.

3.4.5 Bilan sur la spécification du projet

Cette partie nous a permis de décrire le besoin fonctionnel et technique afin d'élaborer notre interface web de gestion constituée d'un serveur d'application sous Django, un serveur web Apache et un serveur de base de données MySQL.

Dans cette architecture 3-tiers figure également une bibliothèque *Boto* pour gérer les services *EC2* et *EBS* d'Amazon et une bibliothèque *Rozo* pour gérer les démons des serveurs de stockage et du serveur de méta-données. Ces bibliothèques sont gérées par l'interpréteur Python et donc intégrables dans une application Django.

Après avoir élaboré l'architecture logicielle, nous avons décrit les interactions entre l'interface et son environnement extérieur. Nous avons commencé par la mise en œuvre de l'architecture *Rozo* via les services *AWS* d'Amazon. Cette architecture est constituée au minimum de cinq machines virtuelles dont un serveur de méta-données et quatre serveurs de stockage.

Ensuite, nous avons utilisé les services Rozo afin de configurer les différents serveurs pour leur attribuer un rôle (*serveur de stockage ou serveur de méta-données*).

Une fois que l'architecture est opérationnelle, un utilisateur qui dispose d'une station de travail avec l'application cliente de *Rozo* peut accéder à son espace de stockage disponible.

A cette étape, nous avons une forte compréhension du projet et l'on peut déjà proposer une solution.

La prochaine partie est consacrée à la modélisation de l'interface web pour en décrire les fonctionnalités.

3.5 Conception de l'interface web de gestion

Avant de se lancer dans la réalisation de l'interface web de gestion, quelques étapes sont importantes pour être en mesure de prouver que notre produit répond bien aux exigences du maître d'ouvrage.

Pour cela, nous allons utiliser les principes de modélisation afin de représenter la structure fonctionnelle et technique de l'interface web de gestion.

Dans un premier temps, nous exposerons des cas d'utilisation pour montrer l'interaction d'un utilisateur avec l'interface.

Puis, dans un deuxième temps, nous nous intéresserons à l'élaboration d'un modèle conceptuel de données. En effet la conception de l'interface web de gestion repose sur une base de données qui est gérée par le serveur d'application *Django*. Ce serveur créé le schéma de la base de données à partir d'un modèle que l'on doit réaliser au préalable.

Ensuite dans un troisième temps, on fera un récapitulatif des fonctionnalités de l'interface afin de définir les priorités dans la réalisation des fonctions.

3.5.1 Cas d'utilisation

On distingue deux cas d'utilisations, celui de l'utilisateur qui souhaite déployer la solution et celui de l'administrateur chargé de gérer à la fois le système d'information, les utilisateurs ainsi que les mises à jour des applications.

3.5.1.1 Usage de l'interface en tant qu'utilisateur

A travers le cas d'utilisation (*Cf. Figure 56*), nous illustrons le comportement fonctionnel de l'interface web de gestion avec un acteur qui interagit avec le système. Le rôle de l'acteur est d'utiliser l'interface web de gestion pour mettre en œuvre un Cloud Storage.

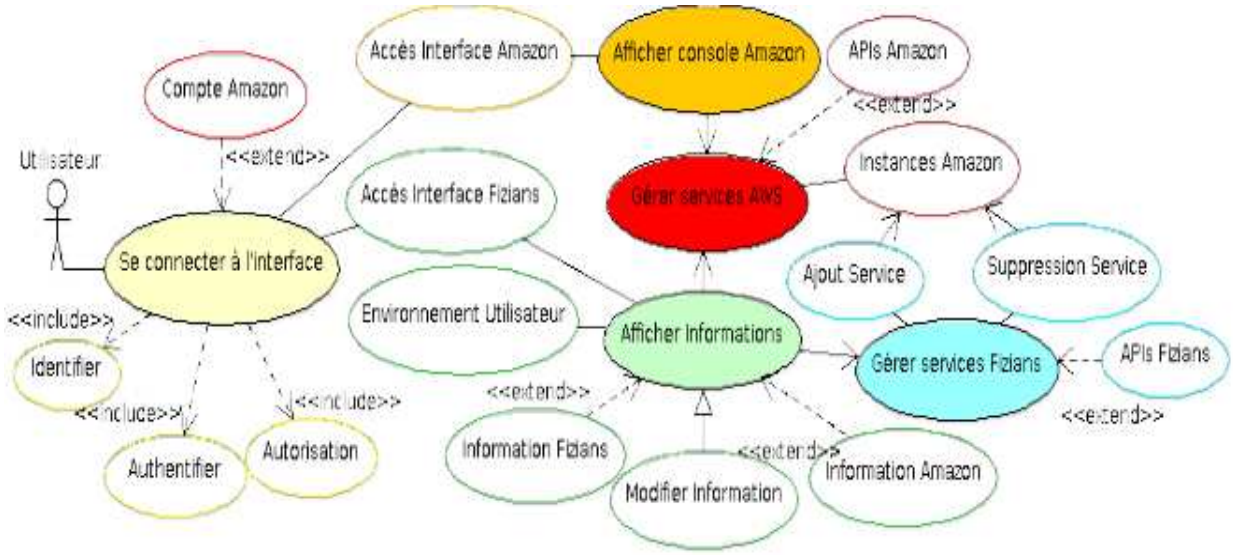


Figure 56 – *Cas d'utilisation de l'interface en tant qu'utilisateur.*

Tout passe par une interface commune qui permet à l'utilisateur de gérer son environnement avec tous les services disponibles. L'interface est le point d'accès commun à tous les utilisateurs.

Le client utilise son compte Amazon pour se connecter à l'interface web de gestion.

Il peut utiliser les services AWS d'Amazon pour gérer son environnement et les services *Rozo* pour gérer le stockage. Les machines déployées appartiennent au client et Amazon en facture l'usage tant qu'elles fonctionnent.

Au niveau de la gestion des services AWS, l'utilisateur peut démarrer ou arrêter une machine virtuelle, affecter une nouvelle adresse IP statique, ajouter un volume EBS, etc.

En fait, toutes ces opérations lui sont accessibles de la même manière que lorsqu'il se connecte à la console de gestion Amazon. L'intérêt est de s'affranchir de cette console et de pouvoir gérer entièrement un environnement de stockage.

Au niveau des services *Rozo* que propose Fizians, l'utilisateur doit avoir la possibilité d'ajouter ou de supprimer un espace de stockage, arrêter ou démarrer un service (*Export/Storage*), ajouter ou supprimer un emplacement de méta-données. La liste des fonctionnalités n'est pas exhaustive et doit pouvoir s'étendre au fur et à mesure de l'évolution des applications *Rozo*.

3.5.1.2 Usage de l'interface en tant qu'administrateur

A travers le cas d'utilisation (*Cf. Figure 57*), nous illustrons le comportement fonctionnel de l'interface web de gestion avec un acteur qui interagit avec le système. Le rôle de l'acteur est d'administrer et de gérer l'interface web de gestion pour notamment assurer une continuité de services, faire évoluer les applications.

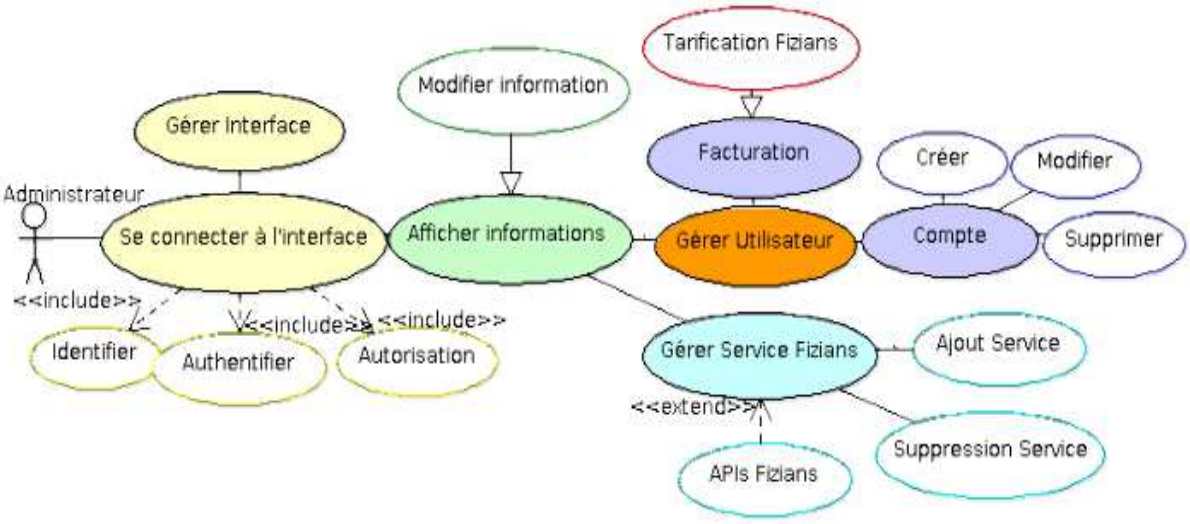


Figure 57 *– Cas d'utilisation de l'interface en tant qu'administrateur.*

L'administrateur se connecte à l'interface principale via un compte administrateur.

Il est chargé de maintenir l'interface web et d'en assurer la disponibilité. Il veille également aux mises à jour de la distribution *Rozo* puis de la mise en œuvre de nouveaux services qui pourront être accessibles aux utilisateurs. Il peut également gérer la facturation de la solution *Rozo* selon l'espace de stockage consommé. Les informations seront stockées dans une base de données afin de faciliter la gestion des informations relatives aux utilisateurs, aux versions d'applications, etc.

3.5.1.3 Bilan sur les cas d'utilisations

Ces cas d'utilisation permettent de structurer les besoins fonctionnels de l'interface web de gestion.

On s'assure ainsi que notre produit répond bien aux objectifs que l'on doit atteindre conformément aux exigences du maître d'ouvrage. Après cette étape de modélisation des besoins, la prochaine étape consiste à modéliser la base de données qui permet de gérer les données d'une manière structurée.

Ces données sont utilisées dans notre interface web par le serveur d'application Django afin d'afficher les informations dans des pages web. Ainsi l'utilisateur peut à tout moment être informé sur l'état de son environnement de Cloud Storage. Mais avant d'évoquer les concepts de notre base de données, nous allons expliquer l'intérêt d'utiliser Django dans une application web.

3.5.2 Django et la base de données

Dans la partie spécification de projet (*Cf. §3.4.5*), nous avons défini un choix technologique pour réaliser l'interface web de gestion avec l'utilisation du langage Python puis une architecture logicielle composée d'un serveur web *Apache*, d'une base de données *MySQL* puis du cadriciel *Django*.
L'intérêt de Django est de construire une application web avec le langage Python et d'utiliser les données stockées dans la base pour les afficher ensuite dans une page web.
C'est la raison pour laquelle on décrit Django comme un serveur d'application web.
D'autant plus que les applications sous Django sont indépendantes les unes des autres, car elles disposent de leur propre « *modèle* », « *template* » et « *vue* ».

3.5.2.1 Particularités de Django

Il s'agit ici de décrire le fonctionnement de *Django* notamment dans l'élaboration du schéma de base de données de notre interface web de gestion.
Nous savons que Django repose sur une architecture *MTV* (*Modèle Template Vue*) avec un « *modèle* » qui correspond au stockage de l'information dans une base de données relationnelle.
Puis le « *template* » correspond à l'affichage des données souvent extraites de la base de données par des requêtes spécifiques à Django. Tandis que la « *vue* » correspond au traitement des données et fait la transition entre le modèle et son affichage.
En outre, ce qui nous intéresse dans Django pour réaliser l'interface web de gestion en dehors du fait qu'il utilise le langage Python, est de tirer profit des fonctionnalités qu'il offre.
Or, parmi les fonctionnalités principales, Django permet de faire du « *Mapping Relationnel Objet* ».
Il dispose pour cela d'une API dynamique pour accéder directement aux bases de données.
D'autre part, Django met à disposition une interface d'administration qui permet à la fois de gérer le contenu des applications web mais également l'authentification des utilisateurs.
Cette interface d'administration servira également pour l'administration de notre application web.
Mais dans ce paragraphe, seule l'interaction de Django avec la base de données nous intéresse afin par la suite, de réaliser le schéma de la base de données de notre application web.

3.5.2.2 Le modèle Django

Un modèle Django est une description des données dans la base représentée sous la forme de code Python. Une fois le modèle créé, Django fournit automatiquement une API Python de haut niveau pour travailler avec les attributs perçus comme des objets appartenant à des classes.

3.5.2.3 Django au cœur de l'interface web de gestion

Ainsi notre interface web de gestion reposera sur Django pour la réalisation de l'application web, pour l'utilisation de l'interface d'administration dans la gestion des utilisateurs et pour la construction de la base de données. La prochaine étape consiste à créer un modèle conceptuel de données.
Ensuite il suffira de l'adapter dans un format que Django pourra interpréter afin qu'il puisse créer le schéma de la base de données.

3.5.3 Le modèle conceptuel de données

L'interface web de gestion repose sur une base de données qui enregistre toutes les informations relatives à l'environnement du système. Cette base de données est nécessaire pour le serveur Django afin d'afficher les informations utiles dans les pages web.
L'utilisateur disposera ainsi de toutes les informations pour qu'il puisse gérer son environnement Rozo mais également celui d'Amazon. Pour cela, il faut construire un modèle conceptuel de données qui décrit la structure des données de l'instance de la base.

3.5.3.1 Définition des tables de l'interface web de gestion

Toute d'abord, on définit chacune des tables qui figureront dans le modèle avec une description des différents attributs qu'elle comporte (*Cf. Annexe A*). La conception des tables est réalisée suite à l'utilisation des APIs Amazon et Rozo que notre application web devra utiliser.
Ces APIs ont besoin d'arguments qu'il faut récupérer à partir d'une source de données.
Par exemple, pour déployer une machine virtuelle, il faut fournir les identifiants de son compte Amazon, la référence d'une image, le nom de la clé SSH précédemment créée ainsi qu'un groupe de sécurité.
L'interface devra donc récupérer ces informations avant d'appeler la fonction Python en mesure d'effectuer l'opération en question.

Un autre exemple consiste à obtenir les informations du Cloud Storage.
De la même manière, l'interface devra récupérer les renseignements sur le serveur de méta-données (*Adresse IP, emplacement des méta-données, statut, etc.*) puis de chaque serveur de stockage (*Adresse IP, taille du volume EBS, emplacement des projections, statut, etc.*).
Ainsi, les tables doivent être élaborées en fonction des données que l'interface doit manipuler, des informations utiles pour l'utilisateur et celles qui sont transmises par les services web.

3.5.3.2 Modélisation de la base de données Django

Le modèle conceptuel de données permet d'avoir une représentation précise de chaque élément constituant une base de données (*données, tables et relations entre tables*).

Comme on peut le voir (*Cf. Figure 58*), la table « *account* » est le noyau central qui interagit sur les autres tables. En pratique, cela oblige l'utilisateur à être enregistré dans la base avec un compte pour se connecter à l'interface. Dans ce compte figurent les identifiants d'accès des services Amazon nécessaires à l'utilisation des *APIs* AWS. Ainsi pour utiliser tel ou tel service, l'interface aura seulement besoin de l'identifiant de l'utilisateur pour ensuite accéder à toutes les informations.

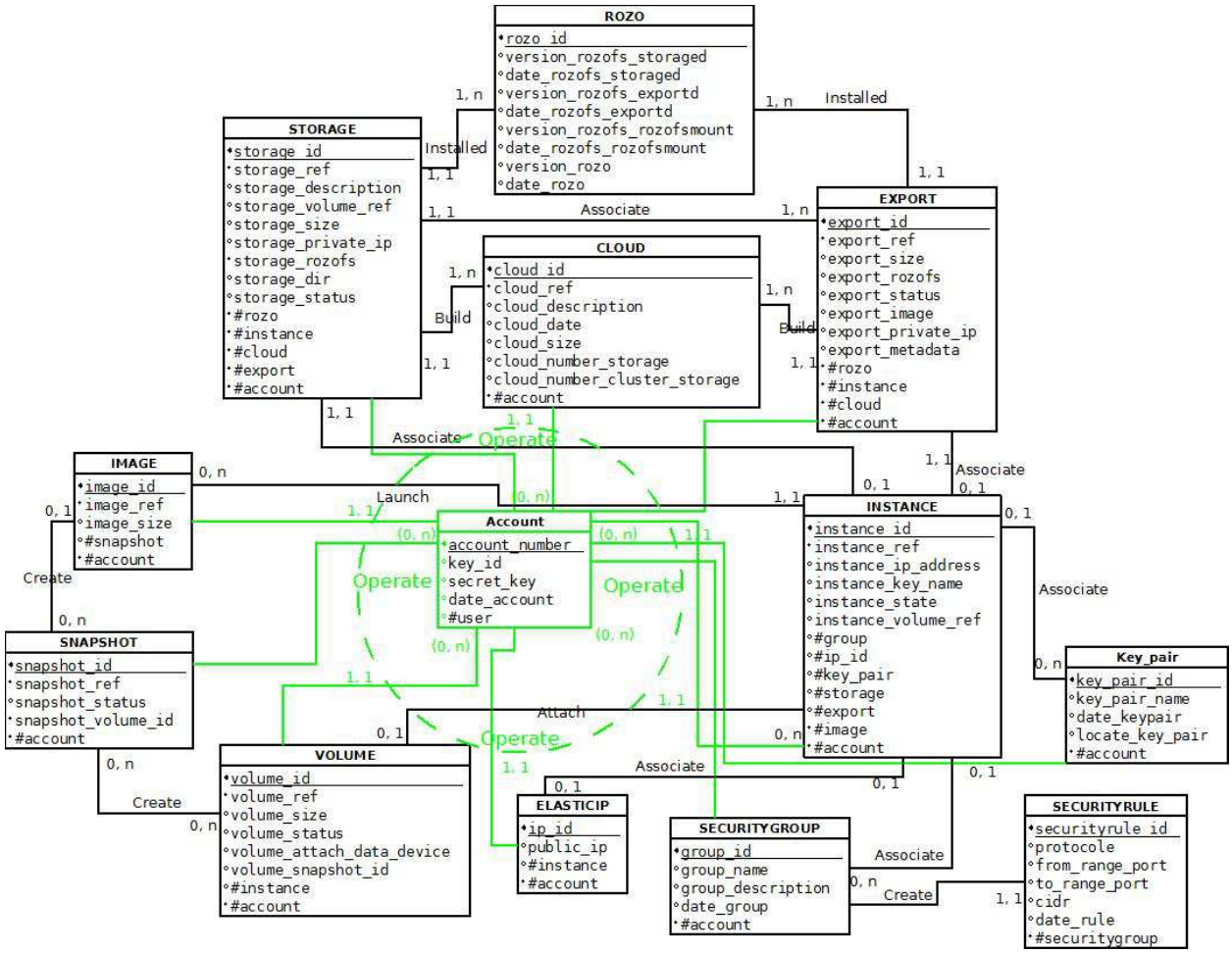


Figure 58 – *Modèle conceptuel de données de l'interface web de gestion.*

Avant de procéder à la construction de la base de données avec Django, il faut définir le modèle au format Python dans le fichier « *models.py* » (*Cf. Annexe B*).

En fait, le formalisme consiste à convertir les tables en classes Python qui contiennent des objets que Django manipulera pour gérer les données. Ces objets correspondent aux attributs des tables du modèle conceptuel. Par ailleurs, dans le modèle Python, il ne faut pas déclarer les clés primaires, car Django les attribue automatiquement à chacune des classes.

3.5.4 Fonctionnalités prise en charge par l'interface web de gestion

Le tableau (*Cf. Tableau III*) indique plusieurs fonctionnalités que doit réaliser l'interface avec un niveau de priorité. Dans un premier temps, on privilégiera les priorités « *Importante* » et « *Elevée* » qui permettent d'assurer le fonctionnement de l'interface conformément aux exigences du maître d'ouvrage.

Tableau III– Liste non exhaustive des fonctionnalités le l'interface.

Domaine concerné	**Tâches**	**priorité**
Stockage	Gérer les services Rozo	Élevée
Stockage	Déterminer le nombre de serveurs de stockage	Importante
Stockage	Choisir le nombre de projections	Moyenne
Stockage	Informations du cloud storage	Importante
Stockage	Tarification sur l'espace de stockage utilisé	Importante
Système de fichiers	Ajouter, supprimer un espace de nommage	Moyenne
Système de fichiers	Choisir, modifier un point de montage	Moyenne
Système de fichiers	Informations (quantités, descriptions, etc.)	Moyenne
Environnement	Informations (instances, performance, volumétrie, etc.).	Moyenne
Environnement	Contrôle (État, arrêter, démarrer machines)	Importante
Amazon	Gérer les services AWS	Élevée
Utilisateur	Gestion des erreurs, support, journalisation	Moyenne

3.6 Bilan sur la partie Conception

Dans cette partie de conception, nous avons suivi plusieurs étapes pour aboutir à la réalisation de l'interface web de gestion de Cloud Storage. Pour cela, nous avons adopté une méthode d'organisation de projet inspirée du modèle en V.
Tout d'abord, nous avons rédigé un cahier des charges qui défini les besoins exprimés du maître d'ouvrage. A partir du cahier des charges qui énumère également les exigences fonctionnelles, nous avons réalisé un organigramme des tâches puis effectuer une planification des opérations avec le diagramme de Gantt. Sur la définition des exigences du maître d'ouvrage et en fonction des contraintes de l'environnement extérieur, nous avons procédé à un choix technologique dans l'élaboration de notre interface. Ce choix technologique repose sur une architecture 3-tiers composée d'un serveur d'application *Django*, d'une base de données *MySQL* puis d'un serveur *Apache*.
Cette architecture est en mesure d'utiliser les bibliothèques Python fournit par Amazon pour la gestion des services AWS puis celles de Fizians pour la gestion du stockage répartie.

Ensuite, nous avons exploité ces différentes APIs qui engendrent des contraintes d'utilisation que notre application web devra gérer. Ainsi, pour la gestion des machines virtuelles avec les services Amazon, le fait d'utiliser les services web impose d'avoir un compte utilisateur puis de récupérer plusieurs éléments de sécurité qui seront exigés. De même lorsqu'une machine virtuelle redémarre, elle retrouve une nouvelle configuration réseau. Tandis que pour les services *Rozo*, la seule contrainte imposée est d'utiliser le langage Python pour développer l'application web, car les APIs *Rozo* sont exploitables uniquement avec ce langage. Il faut également disposer d'une image Amazon qui est préalablement configurée avec le système de fichier *Rozofs*.

Après avoir bien avancé dans le projet, nous avons entamé la modélisation de l'interface web de gestion. La première étape concerne la réalisation de cas d'utilisations dans le but de clarifier les besoins du maître d'ouvrage. Ils permettent également d'étudier le fonctionnement de l'interface web sous le rôle d'un utilisateur ou d'un administrateur chargé de maintenir l'application en état.
Cette phase de conception passe également par une définition d'un modèle conceptuel de données. Ce modèle servira à construire la base de données *MySQL* sur laquelle le serveur d'application *Django* s'appuiera. En effet *Django* utilise ces données dans les pages web d'une manière dynamique grâce à des fonctions Python de haut niveau capable d'interagir directement avec la base de données. Les attributs sont perçus comme des objets appartenant à des classes.

L'utilisateur doit obtenir toutes les informations nécessaires afin qu'il puisse gérer son environnement *Rozo* mais également celui d'Amazon. Cette gestion consiste d'une part à déployer un environnement de machines virtuelles identiques chez Amazon à partir d'une image.

Une fois cet environnement opérationnel, il faut attribuer un rôle pour chacune des machines de sorte qu'une machine virtuelle soit définie comme serveur de méta-données et le reste comme serveurs de stockage. L'interface web de gestion doit être en mesure d'effectuer ces opérations à partir de renseignements fournis par l'utilisateur notamment sur la volumétrie souhaitée en matière d'espace de stockage, l'emplacement des projections, etc.

Le travail de conception prend fin, nous abordons maintenant la phase de réalisation dont la finalité consiste à fournir un espace de stockage réparti entre les différents serveurs de stockage.

Réalisation

4.1 Introduction

Dans cette partie, nous allons mettre en œuvre l'interface Web de gestion selon le modèle architectural et fonctionnel défini lors de la partie conception.

L'interface Web de gestion est une application Web créée par le cadriciel Django avec le langage de développement Python. Django utilise une base de données pour extraire l'information et l'afficher dans les pages Web. Dans la partie conception, nous avons élaboré un modèle au format Python qui décrit le schéma de notre base de données *(Cf. §3.5.3.2*). A partir du modèle, Django construit la base et fournit des fonctions Python capables de manipuler les objets de la base (*Attribut d'une table*) sans utiliser de manière explicite le langage SQL. Puis, Django offre l'avantage d'utiliser toutes les librairies du langage Python dont les *APIs* d'Amazon et de Fizians. L'interface Web utilise ces fonctions Python pour gérer d'une part l'environnement Amazon avec la mise à disposition de machines virtuelles, puis d'autre part, la mise en œuvre d'un espace de stockage réparti au cœur de l'environnement.

De plus, la réalisation de l'interface Web de gestion correspond au développement et à l'intégration de trois applications Web. Tout d'abord, la première application Web concerne la gestion des services AWS d'Amazon, la deuxième se consacre à la gestion du stockage à partir d'un environnement existant, puis la troisième permet de gérer l'authentification des utilisateurs et de les rediriger vers le service qu'ils souhaitent utiliser. La dernière étape consiste à intégrer ces interfaces pour réaliser une seule et unique application Web en mesure d'effectuer toutes les opérations de déploiement d'un Cloud Storage.

Ainsi, l'objectif de la partie réalisation est d'expliquer le fonctionnement et l'utilisation de l'interface Web de gestion dans sa globalité regroupant ainsi les trois applications.

4.2 Préparation de l'environnement

Dans cette partie, nous allons décrire la préparation de notre environnement de travail qui va servir à réaliser l'interface Web de gestion.

4.2.1 Installation des logiciels

Pour préparer notre environnement de travail, nous avons besoin d'une station de travail dotée d'un système d'exploitation (*Debian version 6.0.2*). Ensuite, il faut installer le logiciel Python (*Version 2.6.6*), l'application Django (*Version 1.3.1*), le serveur Mysql (*Version 5.1.49*), le serveur Apache (*Version 2.2.16*). D'autre part, l'application Web nécessite les librairies Rozo (*Version 0.3.13-1*) fournies par Fizians sous forme de paquets Debian ainsi que celles de Boto (*Version 2.0*) pour gérer les services AWS d'Amazon *[Garnaat 2010]*.

Toutes les étapes pour préparer notre environnement de travail sont décrites en annexe *(Cf. Annexe C)*.

Une partie importante dans cette préparation concerne la création d'une instance dans la base de données MySQL. Cette instance sera utilisée par Django pour créer les objets de l'application Web. Une fois l'instance créée, il faut également déclarer un utilisateur avec tous les privilèges sur l'instance en question.

4.2.2 Création d'un projet

Lorsque l'environnement est opérationnel, il faut concevoir un projet Web avec Django *(Cf. §2.3.5.3)*. Il s'agit d'un répertoire contenant tout le code de développement de notre interface.

Pour cela on utilise la commande Django suivante :

« *django-admin.py startproject cloud* »

Cette commande permet de créer un répertoire contenant les fichiers suivants :

cloud/

__init.py

manage.py

settings.py

urls.py

- *Le fichier « **manage.py** » est un script Python qui permet notamment de vérifier et créer le schéma de la base de données.*
- *Le fichier « **settings.py** » permet de configurer Django.*
- *Le fichier « **urls.py** » permet de déclarer les URLs.*

4.2.3 Configuration de Django

Afin que Django puisse interagir avec la base de données, il faut indiquer dans le fichier « ***settings.py*** » le nom de l'instance précédemment créé puis les identifiants d'un utilisateur *MySQL* qui possède tous les privilèges sur l'instance (*Cf. Figure 59*).

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql', # Add 'postgresql_psycopg2', 'postgresql', 'mysql', 'sqlite3' or 'oracle'.
        'NAME': 'cloud',                      # Or path to database file if using sqlite3.
        'USER': 'fizians',                    # Not used with sqlite3.
        'PASSWORD': 'xxxx',                   # Not used with sqlite3.
        'HOST': '',                      # Set to empty string for localhost. Not used with sqlite3.
        'PORT': '',                      # Set to empty string for default. Not used with sqlite3.
    }
}
```

Figure 59 – *Configuration de la base de données dans le fichier settings.py.*

Aussi, il faut préciser dans le fichier de configuration l'emplacement des templates de notre application (« *template_dirs* »). La variable *« root_urlconf »* est configurée lors de la création de l'application avec « *startproject* » et indique à Django l'emplacement du fichier « ***urls.py*** » (*Cf. Figure 60*).

```
ROOT_URLCONF = 'cloud.urls'

TEMPLATE_DIRS = (
    # Put strings here, like "/home/html/django_templates" or "C:/www/django/templates".
    # Always use forward slashes, even on Windows.
    # Don't forget to use absolute paths, not relative paths.
   '/var/www/cloud/templates',
)
```

Figure 60 – *Déclaration des templates dans le fichier settings.py.*

Dans le répertoire « *templates* » que l'on doit élaborer au préalable, on peut classer les fichiers HTML des différentes applications Web que l'on va créer dans des répertoires distincts, Django ira directement les chercher.

template/

amazon

authentification

rozofs

D'autre part, toutes les applications Django de notre interface (*« authentification », « amazon » et « rozofs »*) doivent également être mentionnées dans le fichier « ***settings.py*** » (*Cf. Figure 61*).

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    # Uncomment the next line to enable the admin:
     'django.contrib.admin',
    # Uncomment the next line to enable admin documentation:
    # 'django.contrib.admindocs',
    'authentification',
    'amazon',
    'rozofs',

)
```

Figure 61 *–Liste des applications Django dans le fichier settings.py.*

On peut noter au passage l'application par défaut « *django.contrib.auth* » qui permet de gérer l'authentification des utilisateurs qui seront en mesure d'utiliser les applications du projet.

Par ailleurs, pour réaliser les opérations d'administration de notre système, on utilisera l'interface d'administration Django. Pour cela, il suffit de décommenter la ligne « *django.contrib.admin* ».

4.2.4 Création des applications

Après avoir configuré l'environnement Django, nous pouvons créer les trois applications avec la commande suivante :

« python manage startapp amazon »

L'utilitaire génère alors automatiquement la structure de répertoires de base d'une application.
On procède de la même manière pour l'application « *rozofs* » puis « *authentification* » et on se retrouve avec l'arborescence suivante.

```
cloud/
    amazon/                rozofs/              authentification/
        __init__.py            __init__.py          __init__.py
        views.py               views.py             views.py
        models.py
```

Chaque application est composée de trois fichiers, mais il n'est pas nécessaire de tous les utiliser.
C'est le cas du fichier « *models.py* » dans les applications « *rozofs* » et « *authentification* ».
Ces fichiers sont initialement vides et il convient de les remplir avec des fonctions Python (*views.py*) et des classes (*models.py*).

4.2.5 Création des modèles

Lors de l'état de l'art, nous avons présenté Django et le principe des vues et modèles (*Cf. §2.3.5.3*).
Nous savons que le fichier « *models.py* » permet à Django de construire le schéma de la base de données. Ce fichier contient les classes Python qui sont équivalentes aux tables de notre instance.
De même, dans la partie conception, après avoir élaboré le modèle conceptuel de données, nous avons décrit l'ensemble des tables de notre schéma de base de données (*Cf. §Annexe A*).
Puis, nous avons créé le modèle au format Python à partir du fichier « *models.py* » (*Cf. Annexe B*).
Ce fichier se trouve uniquement dans l'application « *amazon* » car la plupart des classes concernent principalement cette application, mais certaines sont également nécessaire pour l'application « *rozofs* ».
Après avoir configuré Django avec les paramètres de la base de données puis créé le fichier « *models.py* », on réalise les opérations suivantes :

- *Vérification d'erreur avec la commande suivante : « **python manage.py validate** »*
- *Vérification de la syntaxe SQL : « **python manage.py sqlall « nom_projet** »*
- *Construction du schéma de la base de données : « **python manage.py syncdb** »*

Au moment de la création de la base, Django demande d'indiquer un compte administrateur qui sera autorisé à utiliser l'interface d'administration Django.

4.3 L'interface d'administration Django

La génération d'une interface d'administration par une équipe de développement afin d'ajouter, de modifier ou supprimer le contenu d'un site Web, est un travail pénible et répétitif.

C'est pour cette raison que Django automatise entièrement la création des interfaces d'administration. La raison provient s'en doute des origines de Django et du fait de sa conception dans un environnement éditorial avec une très nette séparation entre les *« éditeurs de contenu »* et le site « *public* ». Ainsi cette interface est destinée aux gestionnaires du site afin de gérer les utilisateurs, d'ajouter ou supprimer de l'information, etc. L'interface d'administration de notre application Web est donc déjà mise en œuvre dès le début de la phase de réalisation.

4.3.1 Serveur de développement

Pour afficher le site sans pour autant disposer d'un serveur Web Apache, Django intègre un serveur Web pour le développement. Ce serveur est à proscrire dans un environnement de production, car il est primordial d'utiliser un serveur Web adapté, en mesure d'offrir un certain niveau de sécurité et capable de gérer plusieurs connexions en même temps.

Pour exécuter le serveur de développement, il suffit d'effectuer la commande suivante dans l'interpréteur Python :

« *python manage.py runserver* »

Ensuite le lien « *http://127.0.0.1:8000* » apparait, puis après avoir cliqué dessus, le navigateur Web se lance et affiche la page d'index du site Web (*Cf. Figure 62*).

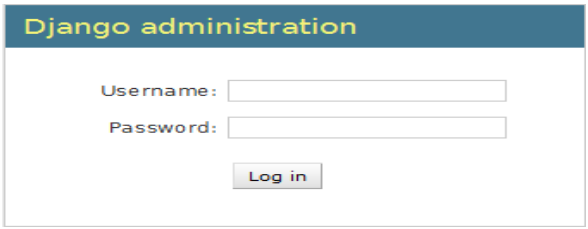


Figure 62 *– L'interface d'administration Django.*

Et pour afficher l'interface, il suffit d'ajouter « */admin/* » à l'url (« *http://127.0.0.1:8000/admin/*»).

Dans notre application, on utilise directement le serveur Apache avec l'interface d'administration qui est accessible via l'url « *https://@ip_public/admin* » avec une adresse IP fixe allouée à notre interface Web, hébergée chez Amazon.

4.3.2 Présentation de l'interface d'administration Django

Une fois connecté à l'interface d'administration, il est possible de gérer chacune des tables de la base de données en ajoutant ou en supprimant les valeurs des attributs. Ainsi, si on sélectionne le lien « *Users* », il est possible d'ajouter, de modifier ou de supprimer un utilisateur (*Cf. Figure 63*).

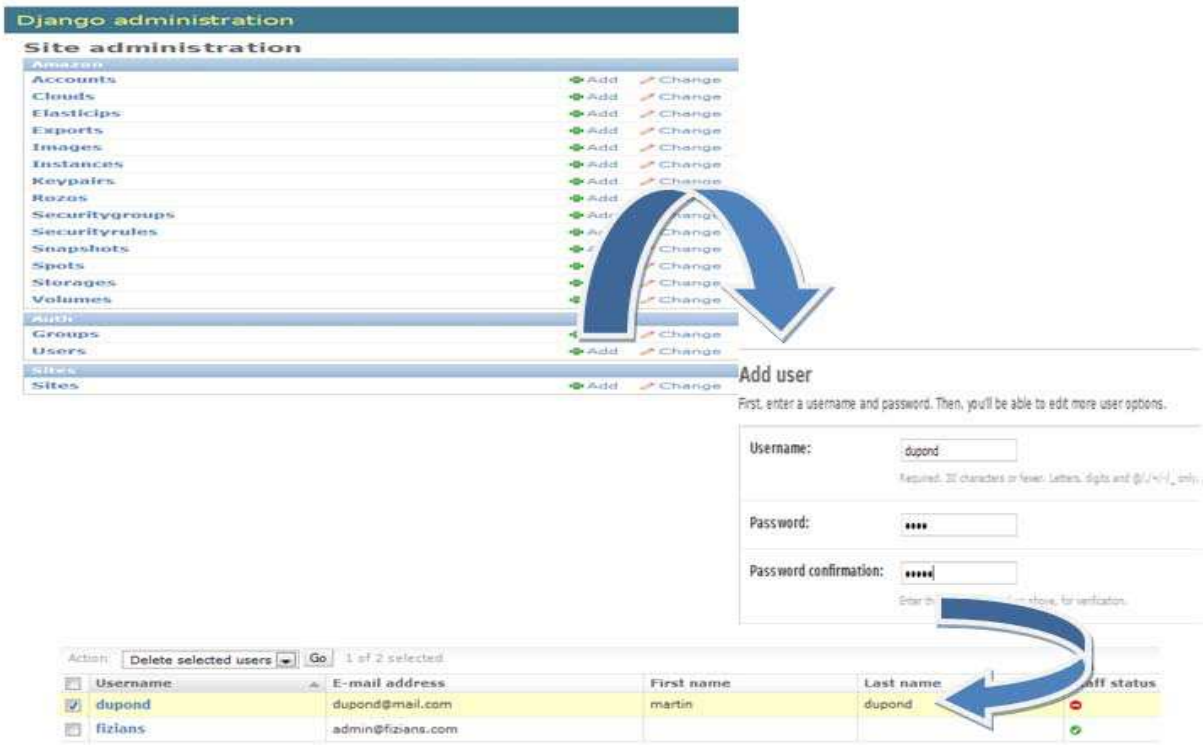


Figure 63 *– Gestion des utilisateurs dans l'interface d'administration Django.*

Par défaut, un utilisateur a le droit de se connecter sur une application Web mais non sur l'interface d'administration.

Une liste de permissions peut ajuster les autorisations d'un utilisateur (*Cf. Figure 64*).

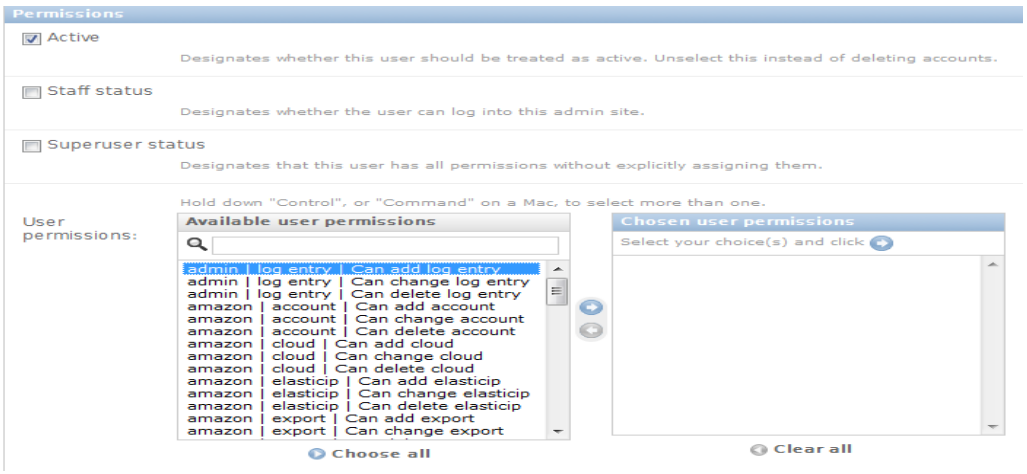


Figure 64 *– Gestion des permissions dans l'interface d'administration Django.*

L'interface Web de gestion doit donc être en mesure de créer directement les utilisateurs avec les permissions par défaut. Pour cela, elle n'utilisera pas l'interface d'administration, mais enregistrera directement les informations (*identifiant de compte*) dans la base de données.

On remarque également que dans l'interface d'administration, si on sélectionne le lien « *Accounts* », on aperçoit les attributs de la table (*Cf. Figure 65*).

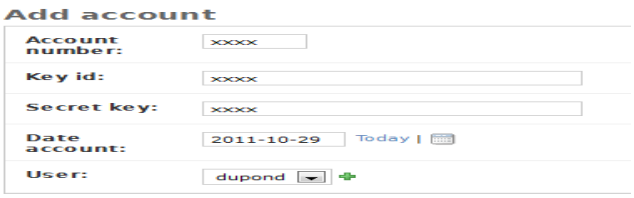


Figure 65 *– Classe « Account » de l'interface d'administration Django.*

En effet, l'interface d'administration ne fait que représenter graphiquement la structure de la base de données. On peut ainsi modifier les valeurs de toutes les tables du schéma.

4.4 Réalisation de l'application Web Amazon

L'interface Web de gestion doit en premier lieu, interagir avec les services Web d'Amazon afin de déployer un environnement de machines virtuelles. Bien que toutes ces opérations puissent être effectuées avec la console de gestion Amazon accessible sur le net, l'utilisateur aurait alors deux interfaces à gérer pour mettre sa solution de Cloud storage.

Ainsi, l'idée est d'offrir au client une solution globale de gestion de stockage répartie dans le nuage à travers une unique interface

4.4.1 Pré-requis

Comme nous l'avons expliqué dans la partie conception (*Cf. §3.4.3.4*), avant d'utiliser les services Web d'Amazon, il faut tout d'abord créer un compte en se connectant à la console de gestion (*Cf. Figure 66*). Cette console est accessible à l'adresse suivante :

« https://aws-portal.amazon.com/gp/aws/developer/registration/index.html »

Figure 66 – Console de gestion AWS d'Amazon.

L'idée est donc de s'inspirer de cette console pour réaliser l'application Web « *amazon* » avec l'utilisation de la librairie Boto.

4.4.2 APIs Amazon avec la librairie Boto

Boto est une librairie Python qui permet d'utiliser la plupart des services Amazon :

- *Simple Storage Service (S3)*
- *Simple Queue Service (SQS)*
- *Elastic Compute Cloud (EC2)*
- *SimpleDB*
- *Virtual Private Cloud (VPC)*
- ...

Cette librairie a été créée par la communauté d'Amazon, notamment par Mitch Garnaat qui fait partie également de l'équipe d'ingénierie d'Eucalyptus (*Solution open-source de cloud computing*).
Elle est constituée d'un ensemble d'objets et de méthodes que l'on peut intégrer dans des fonctions Python *[Garnaat 2010]*. Le code de développement de notre interface Web de gestion contient plusieurs fonctions qui sont utilisées notamment pour le déploiement ou la suppression de machines virtuelles.

4.4.2.1Réalisation d'un script Python avec la librairie Boto

Tout d'abord, avant de réaliser les applications Django de notre interface Web, nous avons développé un script Python qui regroupe toutes les fonctions Boto que l'interface utilisera pour gérer les services Web Amazon.
Dans le cadre de ce mémoire, l'objectif n'est pas d'expliquer toutes ces fonctions dont le principe de mise en œuvre est similaire, mais plutôt d'en illustrer quelques-unes à travers un exemple de script.
Ce script permet de créer un groupe de sécurité « *test* » puis de déployer une machine virtuelle chez Amazon à partir de la référence d'une AMI publique (« *ami-0ce41865* ») avant de procéder à sa destruction. Le script Python est constitué de cinq fonctions qui sont exécutées successivement.
Nous allons décrire ces cinq fonctions avant d'en exploiter le résultat. Le script complet figure en annexe (*Cf. Annexe D*).

a) Fonction « connection »

Une première fonction intitulée « *connection* » utilise la méthode « *EC2Connection* » avec en paramètre, les identifiants de clé d'accès pour s'authentifier chez Amazon.
La fonction retourne une valeur qui contient l'adresse DNS Publique du portail AWS (« *ec2.amazonaws.com* »). Cette adresse est passée en argument pour toutes les autres fonctions du script.

```
def connection():
    print '--- running EC2Connection tests ---'
    c = EC2Connection(key, secret)
    return c
```

b) *Fonction « add_group »*

La deuxième fonction intitulée « *add_group* » permet de créer un groupe de sécurité « *test* » et utilise pour cela la méthode « *create_security_group* » avec les paramètres « *nom* » et « *description*».

```
def add_group(c, group_name):
    group_desc = 'This is a security group created by Fizians'
    c.create_security_group(group_name, group_desc)
```

c) *Fonction « add_perm »*

La troisième fonction intitulée « *add_perm* » permet d'ajouter une règle de permission au groupe de sécurité « *test* » en autorisant le port SSH. Pour cela, elle utilise la méthode Boto « *get_all_security_groups* » afin de récupérer une liste de tous les noms des groupes de sécurité à rechercher. Cependant, puisque nous avons exécuté cette commande avec un filtre sur le nom du groupe souhaité, la liste ne contient qu'un seul enregistrement que l'on place dans la variable « *group* ». Ensuite, nous utilisons une deuxième méthode « *authorize* » qui va transmettre à l'objet « *group* », le numéro de port TCP, l'adresse de début, l'adresse de fin et le « *CIDR* » pour préfixer les adresses IPv4. Ainsi nous pourrons effectuer une connexion en SSH sur la machine virtuelle Amazon configurée avec le groupe de sécurité en question puisque le protocole est autorisé. Par défaut, un groupe de sécurité sans règles de permission, bloque tous les ports TCP/UDP.

```
def add_perm(c, group_name, port, numfrom, numto, cidr):
    rs = c.get_all_security_groups([group_name])
    group = rs[0]
    perm = group.authorize (port, numfrom, numto, cidr)
```

d) *Fonction « add_instance »*

La quatrième fonction intitulée « *instance* » permet de déployer une machine virtuelle à partir d'une image. Pour cela, elle utilise la méthode Boto « *run_instances* » avec plusieurs arguments comme la référence de l'image *« image_id* », le type de configuration de la machine « *instance_type* », le nom d'une clé SSH « *key_name* » et le nom du groupe de sécurité « *security_groups* ».

Cette méthode récupère un identifiant de réservation fourni par Amazon que nous pouvons utiliser pour obtenir toutes les informations sur la machine virtuelle en cours de déploiement.

Nous pouvons ainsi connaître le statut de la machine virtuelle.

Lorsque celui-ci passe à l'état « *running* », nous pouvons récupérer les attributs « *identifiant* », «*adresse IP privé* » et « *nom de clé* » de l'objet « *instance* ».

```
def instance (c, image_id, instance_type, key, group):
    reservation = c.run_instances(image_id, instance_type = str(instance_type), key_name = str(key), security_groups = [group])
    instance = reservation.instances[0]
    while instance.state != 'running':
        time.sleep(5)
        instance.update()
    instance_id = str(instance.id)
    print "Instance id: "+instance_id
    print "IP Private: "+instance.private_ip_address
    print "Key Pairs: "+instance.key_name
    return instance_id
```

e) *Fonction « add_instance »*

La dernière fonction intitulée « *del_instance* » consiste à supprimer la machine virtuelle précédemment déployée. Pour cela, elle utilise la méthode « *terminate_instances* » qui utilise comme argument, l'identifiant de l'instance retourné par la fonction précédente (« *instance_id* »).

```
def del_instance(c, instance_id):
    c.terminate_instances(instance_id)
    print '---  Fin Test EC2Connection  ---'
```

4.4.2.2 Exploitation du script Python

Nous utilisons un poste de travail, configuré avec Python et la librairie Boto, qui peut accéder sur le réseau Internet. Le système d'exploitation est « *Debian 6.0* » et avec le terminal du système d'exploitation, nous exécutons le script Python. Le résultat du script (*Cf. Figure 67*) indique que nous récupérons bien l'identifiant de la machine virtuelle déployée, son adresse IP privée puis le nom de la clé SSH nécessaire pour se connecter à la machine.

```
riaudel@cloud:~$ python test.py
--- running EC2Connection tests ---
Instance id: i-251db846
IP Private: 10.249.43.53
Key Pairs: essai
---  Fin Test EC2Connection  ---
```

Figure 67 *– Exploitation d'un script Python avec la librairie Boto.*

D'autre part, nous pouvons vérifier que cette machine a bien été déployée chez Amazon en nous connectant sur la console AWS (*Cf. Figure 68*).

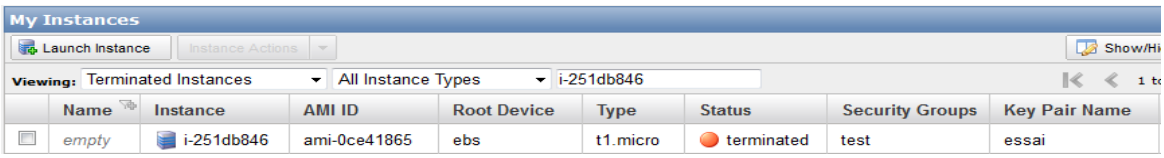


Figure 68 *– Résultat d'un script Python sur la console AWS d'Amazon.*

Cette phase de manipulation nous a permis d'utiliser les services Web d'Amazon à travers des fonctions Python qui peuvent être intégrer dans le fichier « *views.py* » de l'application Django (*Cf. §4.2.4*).

4.4.2.3 Intégration des fonctions Amazon dans l'interface Web de gestion

L'interface Web utilise les fonctions Amazon que nous avons développé avec la librairie Boto.

Nous en avons déjà expliqué quelques-unes (*Cf. §4.4.2.1*) comme le déploiement d'une machine virtuelle, nous ne détaillerons pas les autres dans le cadre de ce mémoire. En effet, la mise en place d'un serveur dans le Cloud Amazon, constitue l'opération élémentaire pour élaborer un environnement de Cloud Storage.

En outre, la procédure pour la plupart des méthodes Boto est identique avec, tout d'abord, la récupération des arguments nécessaires, puis l'appel de la fonction, le passage des arguments et ensuite la récupération des données fournies en retour par Amazon. D'ailleurs, l'utilisation des APIs Rozo repose également sur le même principe, car nous utilisons toujours le langage de développement. Python qui est orienté objet. En ce qui concerne notre interface, la récupération des arguments passe soit par l'intermédiaire de formulaires lorsque les informations dépendent du choix de l'utilisateur (*Par exemple, la taille d'un volume*), soit par une extraction dans la base de données lorsque les informations sont permanentes (*Paramètres d'authentification, nom de la clé SSH ou du groupe de sécurité, etc.*). Aussi, toutes les fonctions Amazon et Rozo sont intégrées réciproquement dans les applications Django « *amazon* » et « *rozofs* » (*Cf. §4.2.4*).

Tout d'abord, le fichier « *views.py* » de chaque application, contient des fonctions qui vont permettre de gérer les données provenant des formulaires ou de la base de données, d'appeler les méthodes Boto ou Rozofs, de récupérer les informations en retour afin de les transmettre pour l'affichage vers un template ou les enregistrer dans la base de données. Tandis que les « *templates* » contiendront des formulaires pour récupérer les données de l'utilisateur ou des tableaux pour ordonner et afficher l'information. L'utilisateur pourra naviguer entre les différents services par le mécanisme des URLs référencées dans le fichier « *urls.py* » (*Cf. 2.3.5.3*) qui permet de faire le lien entre le traitement des données (« *views.py* ») et leurs représentations (« *template* »).

4.4.3 Description de l'application Web Amazon

L'application Web Amazon intitulée « *amazon* » permet de gérer les services Web notamment dans la gestion des machines virtuelles « *instances* », des images « *AMI* », des volumes, etc.

Dans la page principale de l'application Django « *amazon* », figure un menu de navigation nécessaire pour accéder aux services Web souhaités (*Cf. Figure 69*).

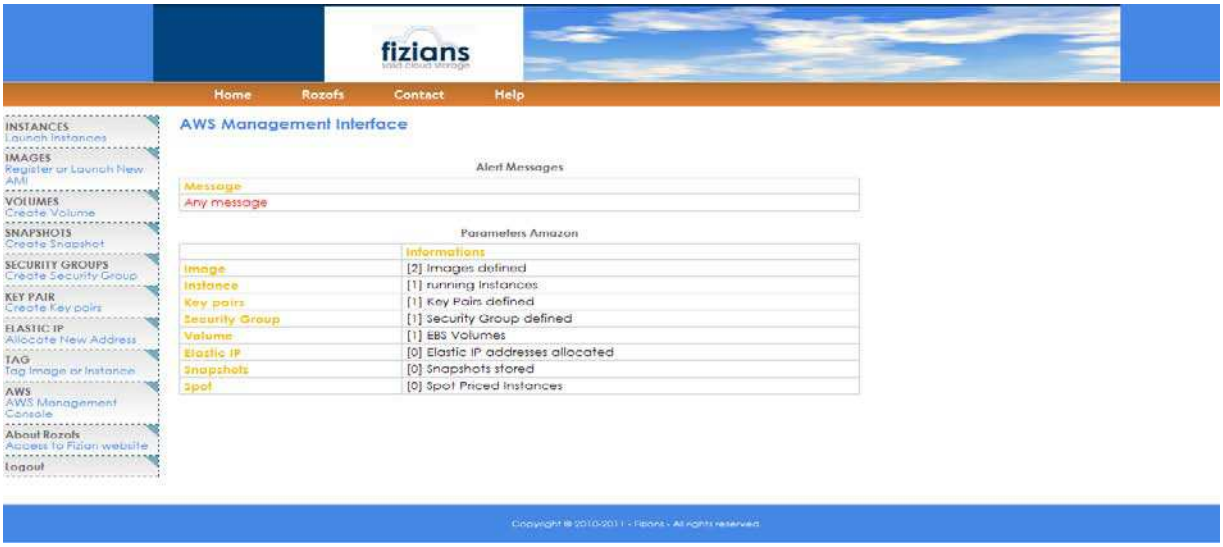


Figure 69 *– Page principale de l'application Django « amazon ».*

4.3.3.1 La gestion des machines virtuelles

L'utilisateur accède via le lien « */amazon/instance* » à la page *« instance.html »* qui permet de gérer les machines virtuelles Amazon (*Cf. Figure 70*).

Cette page comporte tout d'abord un formulaire « *Instance Action* » prévu pour gérer les instances ou créer une image à partir d'une instance, puis un deuxième formulaire « *Launch Instance* » qui permet de déployer une instance en indiquant la référence d'une AMI.

a) Présentation des fonctionnalités

A partir du formulaire « *Instance Actions* », l'utilisateur peut contrôler ses machines virtuelles avec la possibilité de les supprimer, de les stopper ou de les redémarrer.

Tandis qu'avec le formulaire « *Launch Instance* », il peut créer une machine virtuelle en indiquant la référence d'une image Amazon (*Amazon Machine Image*).

La liste déroulante « *Instance type* » permet de choisir la configuration de la machine virtuelle afin d'ajuster de la puissance de calcul. Par défaut, il s'agit d'une « *small instance* ».

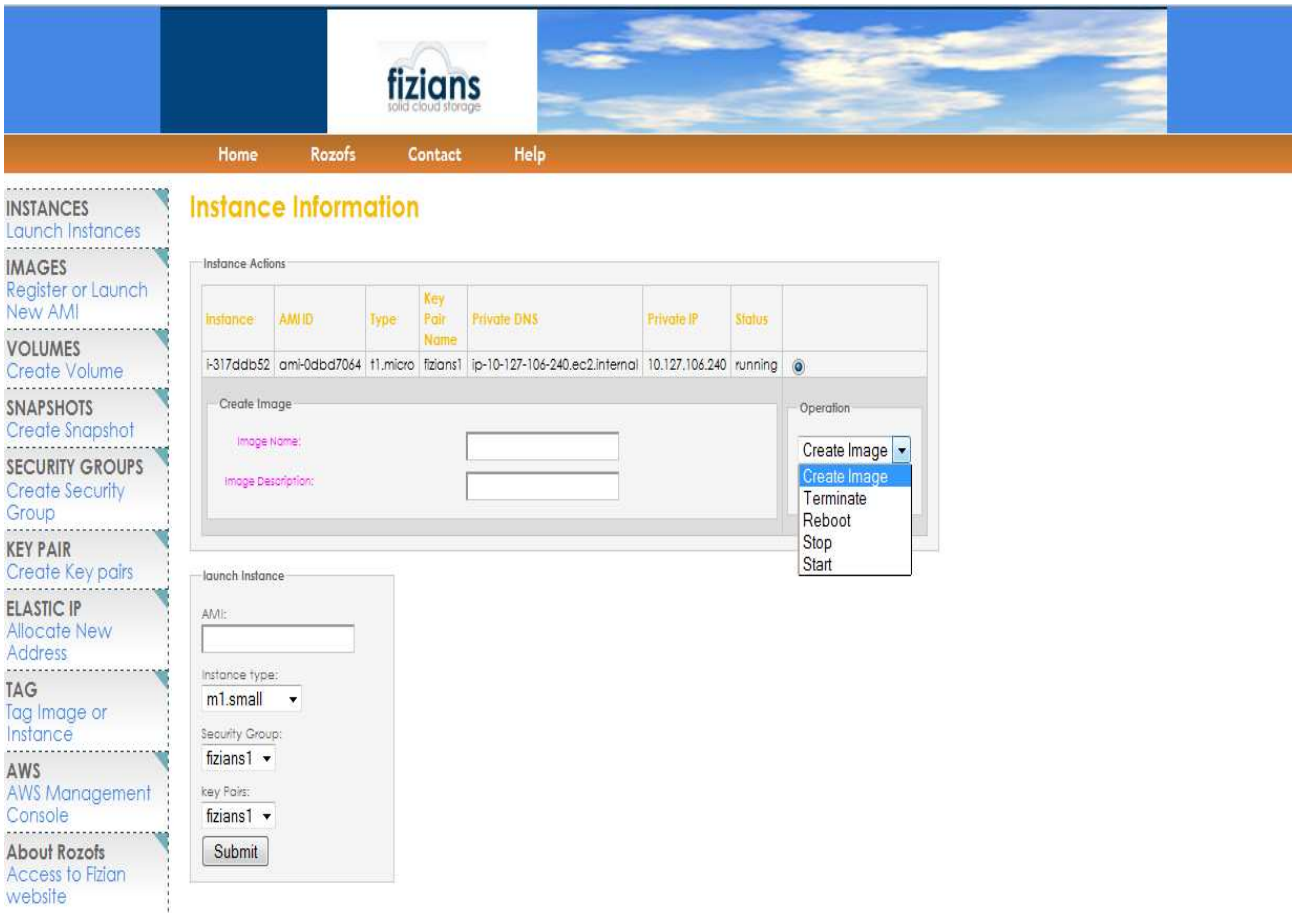


Figure 70 *– Page « instance » de l'application Django « amazon ».*

b) Description des fonctionnalités au niveau interface

Nous allons expliquer une partie du fonctionnement de l'application Web « *amazon* » à travers un diagramme d'activité (*Cf. Figure 71*).

Tout d'abord, l'utilisateur accède à la page principale « *AWS* » (*Cf. Figure 69*) qui propose un menu permettant de choisir le service Web désiré. Cette page affiche des informations sur le nombre d'éléments que dispose l'utilisateur. Pour cela, une fonction Python va interroger la base de données pour connaître le nombre d'occurrences de l'objet en question et va retourner ce nombre dans le template pour en afficher la valeur. Chaque création d'un objet Amazon (*groupe de sécurité, clé privée, etc.)* engendre un enregistrement dans la table concernée, de même que chaque suppression entraîne une modification.

Ensuite, l'utilisateur accède à la page « *instance* » qui affiche également des informations sur les machines virtuelles uniquement s'il y a des enregistrements les concernant dans la base de données (*table « instance »)* (*Cf. Annexe A*).

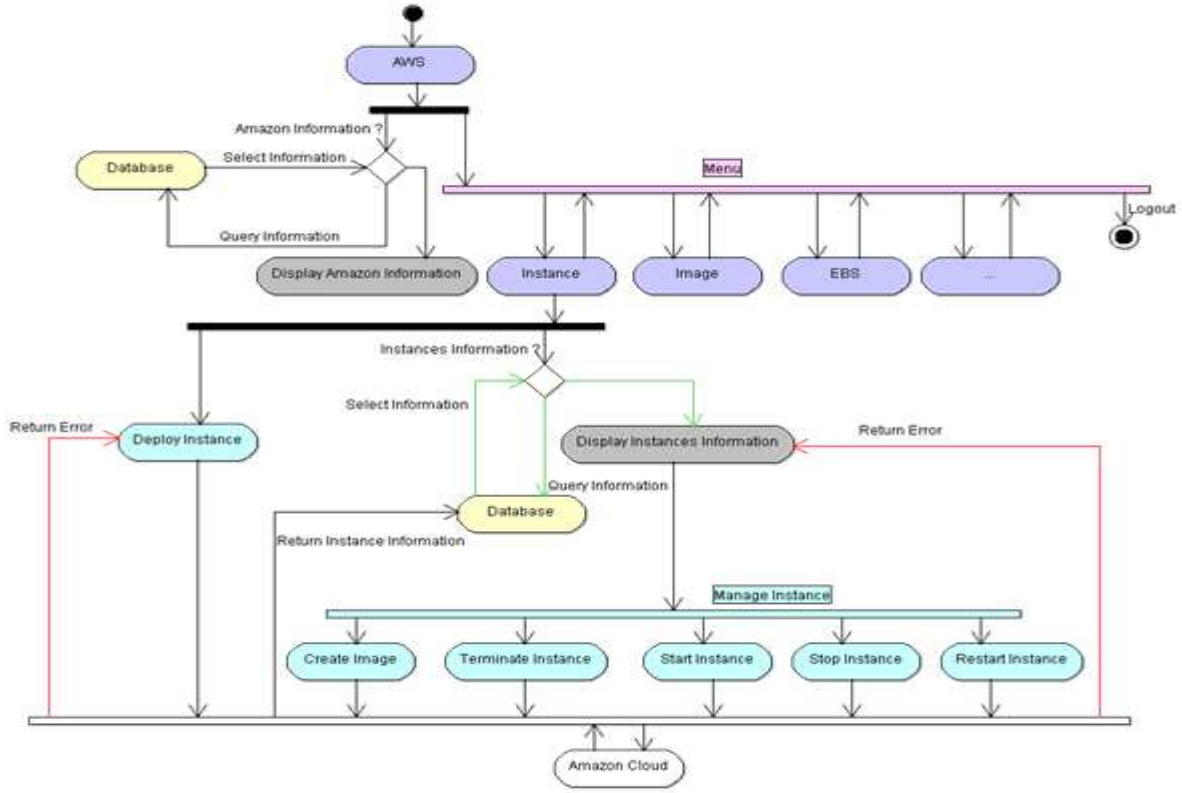


Figure 71 *– Diagramme d'activité de l'application Amazon*

Lorsque la page ne comporte pas d'informations, l'utilisateur peut uniquement utiliser la fonction de déploiement d'instance (*Cf. §4.4.2.1).* En revanche, lorsque ces informations sont présentes dans la base de données, elles sont affichées dans un tableau dont chaque ligne représente un enregistrement de la table « *instance* ». Des cases à cocher permettent de sélectionner l'enregistrement souhaité et d'en récupérer la clé primaire qui identifie de manière unique la machine virtuelle.

Ensuite, l'utilisateur sélectionne le service Web qu'il désire, via une liste déroulante qui propose les options « *create image* », « *start* », « *stop* », « *reboot* » ou « *terminate* ». En validant ainsi le formulaire, le service Web correspondant à l'option choisie sera invoqué.

Ainsi, la gestion des services Amazon consiste simplement à utiliser des fonctions Python qui exploitent l'API Boto, en leur fournissant les paramètres nécessaires pour fonctionner.

Cette gestion consiste également à mettre à jour la base de données avec les informations retournées par l'API. Ainsi, la base de données conserve des informations cohérentes sur l'environnement Amazon.

c) Description des fonctionnalités au niveau développement

Nous avons expliqué le fonctionnement de l'application Web « *amazon* » au niveau interface Web, nous allons reprendre l'explication en couche basse, au niveau échange de données entre les différents fichiers Django (« *views.py* », « *models.py* » et le « *template* ») (*Cf. Figure 72*).

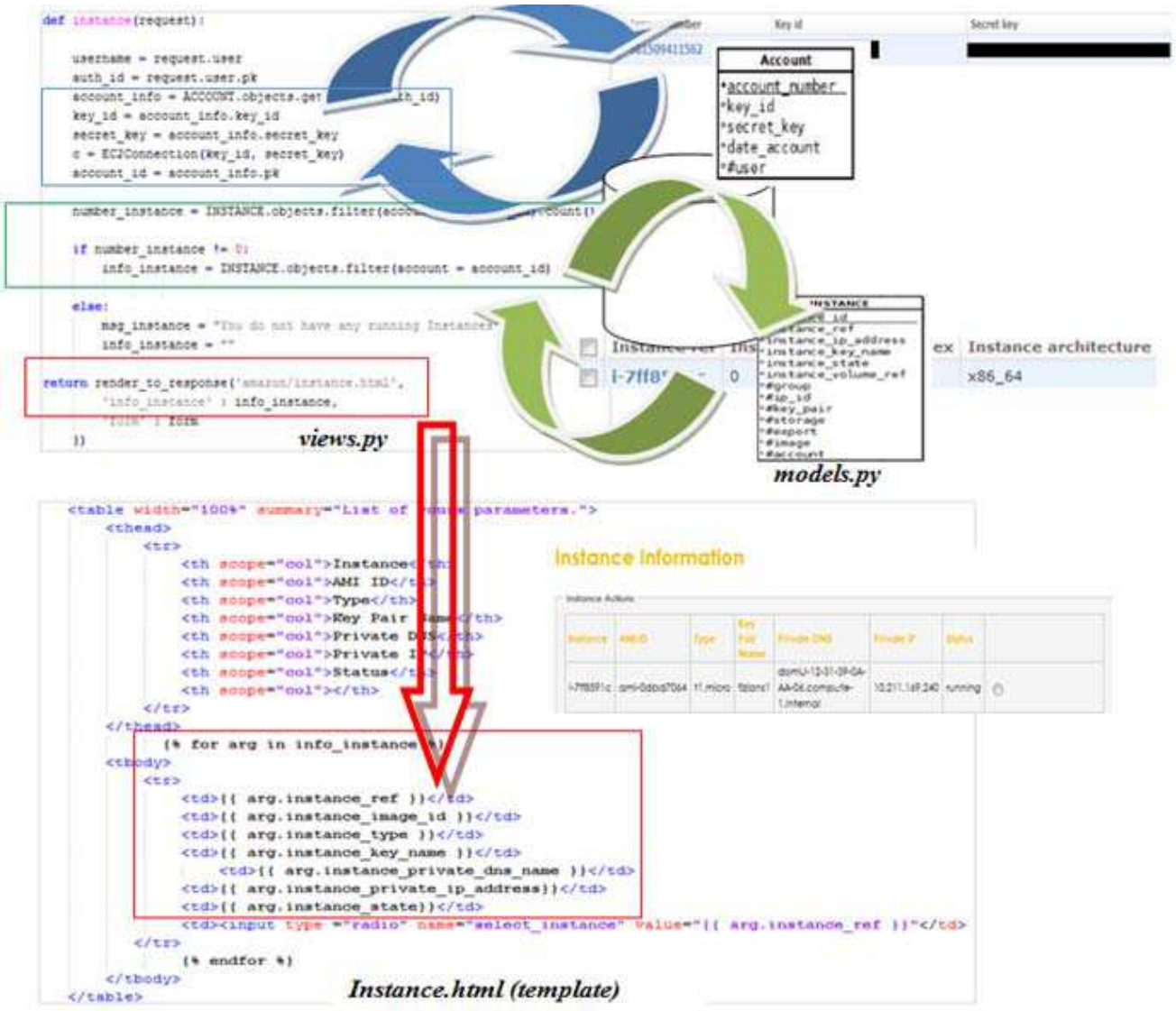


Figure 72 *– Interaction des fichiers Django de l'application « amazon ».*

Tout commence par une fonction « *instance* » qui est définie dans le fichier *views.py* de l'application « *amazon* » (Cf. §4.2.4).

Tout d'abord, la fonction récupère un identifiant de l'utilisateur qui a ouvert une session dans l'interface Web grâce à la commande « *auth_id = request.user.pk* ».
Elle interroge la table *« auth_user »* (*Cf. Annexe A)* et récupère la clé primaire.
En effet, des requêtes Django peuvent récupérer les informations d'un compte utilisateur lorsque sa session est ouverte, car les données sont stockées temporairement dans la base de données.
Or, à partir de cette clé primaire et par une contrainte référentielle avec la table « *Account* », la fonction « *instance* » récupère la clé primaire de la table « *Account* » (*account_id*) et les identifiants Amazon propres à l'utilisateur. Ces identifiants de clé d'accès et de clé d'accès privé sont affectés réciproquement dans les variables « *key_id* » et « *secret_key* ».
La fonction appelle ensuite la méthode « *EC2Connection* » (Cf. §4.4.2.1) et lui passe les variables d'identification en argument.
Il est ainsi possible d'utiliser tous les services Web Amazon que l'utilisateur désire.

Ensuite, la fonction « *instance* » va compter le nombre d'enregistrement dans la table « *instance* » de l'utilisateur à partir de la clé primaire « *account_id* » récupérée précédemment.
Nous procédons d'abord à un comptage avant de récupérer les valeurs des champs, car si la valeur est nulle, alors nous envoyons un message au template qui indique que l'utilisateur ne possède aucune machine virtuelle. En revanche, si cette valeur est non nulle alors la fonction récupère tous les enregistrements de la table « *instance* » propre à l'utilisateur et stocke les données dans une liste *« info_instance »* (*info_instance = INSTANCE.objects.filter(account = account_id))*.

La fonction « *instance* » finit ensuite par transmettre la liste *« info_instance »* à la template « *instance.html* » par la méthode « *render_to_response* » propre à Django.
Dans cette template, on récupère la liste sous forme de tag (*Cf. 2.3.5.3*).
Avec une boucle « *for* », on récupère tous les objets de la liste que l'on ordonne dans un tableau au format Html.

4.4.4 Bilan sur l'application Amazon

Tout d'abord, les fonctions Amazon sont facilement intégrables dans une application Django puisqu'il suffit simplement de les insérer dans le fichier Python « *views.py* ».
Ensuite, il est facile de manipuler les informations d'une base de données, car pour Python, tout est objet et donc, manipuler les champs d'une table est aussi simple que de gérer des variables dans des fonctions. Les informations d'une base de données sont exploitables par les fonctions Python afin d'argumenter les méthodes Amazon mais également pour les transférer vers les templates en vue de représenter les données de manière lisible et compréhensible par l'utilisateur.
Nous n'avons pas mentionné les formulaires qui permettent de récupérer les données émises par l'utilisateur, mais le fonctionnement est relativement simple avec l'utilisation de la méthode « *POST* ». Le principe est d'envoyer les données après validation, vers une fonction Python qui va récupérer tous les champs des formulaires identifiés par des labels.
Ainsi, dans le fichier « *views.py* », la fonction « *instance* » récupère notamment les valeurs du formulaire pour argumenter les APIs Amazon. C'est par exemple le cas pour créer une image à partir d'une instance avec l'obligation de fournir un nom et une description.

Maintenant que nous avons expliqué le fonctionnement de l'application « *amazon* » notamment dans le déploiement d'une machine virtuelle, nous allons nous intéresser à la mise en œuvre du Cloud Storage avec les services Web Rozofs.

4.5 Réalisation de l'application Web Rozo

L'application Web Rozofs permet de mettre en œuvre un Cloud Storage constitué d'un serveur de méta-données et d'un ensemble de serveurs de stockage destiné à contenir des projections stockées dans les volumes de données (*EBS*). L'espace de stockage de l'utilisateur est ainsi réparti entre les serveurs de stockage qui contiennent des projections. *(Cf. 2.2.6.4*). Pour réaliser cette opération, l'interface Web procède en trois étapes successives :

- *La première étape consiste à déployer les machines virtuelles Amazon.*
- *La deuxième étape permet de configurer les volumes EBS attachés aux serveurs de stockage.*
- *La troisième étape est chargée de configurer l'ensemble des serveurs avec l'API Rozofs.*

Une fois que l'architecture est opérationnelle, un utilisateur qui dispose d'une station de travail avec l'application cliente de *Rozo* peut accéder à son espace de stockage disponible.

Dans cette partie, nous allons surtout décrire la troisième étape qui consiste à exploiter les APIs Rozofs dans l'application Django sur le même principe que pour les APIs d' Amazon.
En effet, nous avons précédemment analysé la première étape avec l'application « *amazon* » (*Cf. 4.4*).
Tandis que la deuxième étape consiste simplement à préparer les volumes EBS afin que les serveurs de stockage puissent stocker les projections des fichiers appartenant à l'utilisateur. Il s'agit en fait de formater chaque volume avec le système de fichiers *ext4* puis de monter le volume dans un répertoire créé préalablement. Ce répertoire est destiné à contenir certaines projections des fichiers de l'utilisateur selon le choix arbitraire du serveur de méta-données et en fonction de l'espace de stockage disponible sur chaque serveur. Celui qui offre le moins d'espace de stockage est en dernière position.

4.5.1 Présentation de l'interface Web Rozofs

Dans la page principale de l'application Django « *rozofs* », figure un menu de navigation nécessaire pour accéder aux services Web souhaités (*Cf. Figure 73*).

- *Tout d'abord, la page « Add Cloud Storage » permet de déployer les machines virtuelles Amazon, il s'agit de la première étape dans le processus de configuration.*
- *Ensuite, la page « Mount EBS in Storage Server » permet de formater et monter les volumes EBS sur chaque serveur de stockage, il s'agit de la deuxième étape.*
- *Puis la page « Rozofs Services » permet dans un premier temps de configurer les serveurs afin que l'environnement Rozofs soit opérationnel. L'utilisateur pourra aussi gérer les démons Rozofs (« storaged » et «exportd ») pour notamment les arrêter ou bien les redémarrer.*

Figure 73 *– Page principale de l'application « Rozofs » avant la configuration des serveurs.*

Lorsque l'utilisateur n'a pas de Cloud Storage, un message est indiqué sur la page Web.

En revanche, lorsque l'environnement est déployé, un tableau résume les paramètres du Cloud Storage avec notamment sa taille globale, le nombre de « *cluster* » ainsi que le nombre de serveurs de stockage (*Cf. Figure 74*). Un cluster représente le nombre d'environnements Rozofs avec au minimum pour chacun d'entre eux, un serveur de méta-données et quatre serveurs de stockage.

D'autre part, lorsque l'utilisateur sélectionne le lien dans la case « *Cluster* », un nouveau tableau affiche les informations des machines virtuelles avec notamment leur référence (« *i-xxx* »), leur adresse IP privée, la référence et la taille des volumes EBS attachés aux serveurs de stockage.

Figure 74 *– Page principale de l'application « Rozofs » après la configuration des serveurs.*

4.5.1.1 Page « *Add Cloud Storage* »

Cette page permet à l'utilisateur de déployer un certain nombre de machines virtuelles à partir d'une image fournie par Fizians (*Cf. Figure 75*).

- *Les deux premiers champs du formulaire indiquent la même référence d'une image.*
  En effet l'image désignée permet de déployer une machine virtuelle avec les applications Rozo installées.
- *Les deux autres champs indiquent respectivement la liste des groupes de sécurité puis la liste des clés SSH que l'utilisateur dispose. Un groupe de sécurité permet de configurer l'autorisation des ports TCP/UDP en entrée et sortie des machines virtuelles. Une clé SSH permet de se connecter aux machines virtuelles.*

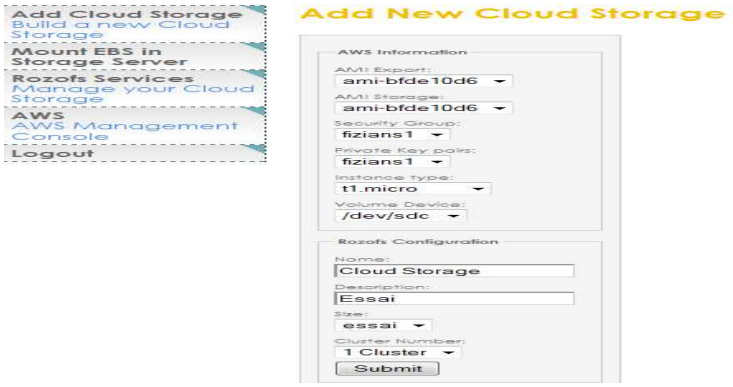


Figure 75 *– Page « Add Cloud Storage » de l'application Web Rozofs.*

Comme nous l'avons décrit pour l'application « *amazon* » en ce qui concerne le déploiement d'une machine virtuelle (*Cf. §4.3.3.1*), l'utilisateur saisit ou choisit les paramètres dans le formulaire puis le valide. Les données sont alors récupérées par une fonction Python et vont servir à argumenter la méthode Boto nécessaire pour déployer une machine virtuelle (*Cf. §4.4.2.1*).

4.5.1.2 Page « *Mount EBS in Storage Server* »

Dans cette page figure un tableau qui liste les serveurs de stockage avec pour chacun d'eux des informations fournies par Amazon et stockées ensuite dans la base de données (*Cf. Figure 76*).

Lorsque l'utilisateur clique sur un bouton « *select* », un script est automatiquement généré puis exécuté. Une fois terminé, le statut du volume passe de l'état « *EBS Not Ready* » à « *EBS Ready* ». L'utilisateur n'a plus qu'à sélectionner le volume suivant dans la liste.

Figure 76 *– Page « Mount EBS in Storage Server » de l'application Web Rozofs.*

4.5.1.3 Page « *Rozofs Services* »

Afin que l'utilisateur puisse accéder à son espace de stockage, il faut configurer les démons des serveurs de stockage et celui du serveur de méta-données. Ces opérations de même que la gestion des services Rozo se réalisent sur la page « *Rozofs Services* » (*Cf. Figure 77*). Cette page comporte deux tableaux et deux formulaires associés afin de gérer le serveur de méta-données (*tableau du haut*), ainsi que chaque serveur de stockage (*tableau du bas*).

Tout d'abord, le tableau du haut liste les informations du serveur de méta-données :

- *La mention « Not Build » de la colonne « Rozofs » indique que la configuration de l'environnement Rozo n'a pas été effectuée.*
- *La mention « Not defined » indique que le serveur de méta-données ne dispose pas d'emplacement pour contenir les méta-données.*
- *La mention « None » de la colonne « Status » indique que le démon « exportd » est absent.*

D'autre part, le formulaire associé au tableau, permet d'effectuer plusieurs opérations :

- *Afficher les serveurs de stockage associés au serveur de méta-données avec l'option « display storage »,*
- *Configurer le Cloud Storage avec « build »,*
- *Arrêter ou démarrer le démon « exportd » du serveur de méta-données avec les options « stop », « start » et « restart ».*

En sélectionnant « *display storage* », l'utilisateur obtiendra la liste de tous les serveurs de stockage associés au serveur de méta-données sélectionné (*tableau du bas*). Une liste déroulante « *Opération Storage* » permet d'arrêter ou de démarrer le démon « *storaged* » d'un serveur de stockage après l'avoir sélectionné en cochant la case. Ainsi, pour configurer l'environnement Rozofs, l'utilisateur doit sélectionner le serveur de méta-données puis sélectionner l'option « *build* » dans la liste déroulante. Ensuite, il doit indiquer le répertoire qui contiendra les projections.

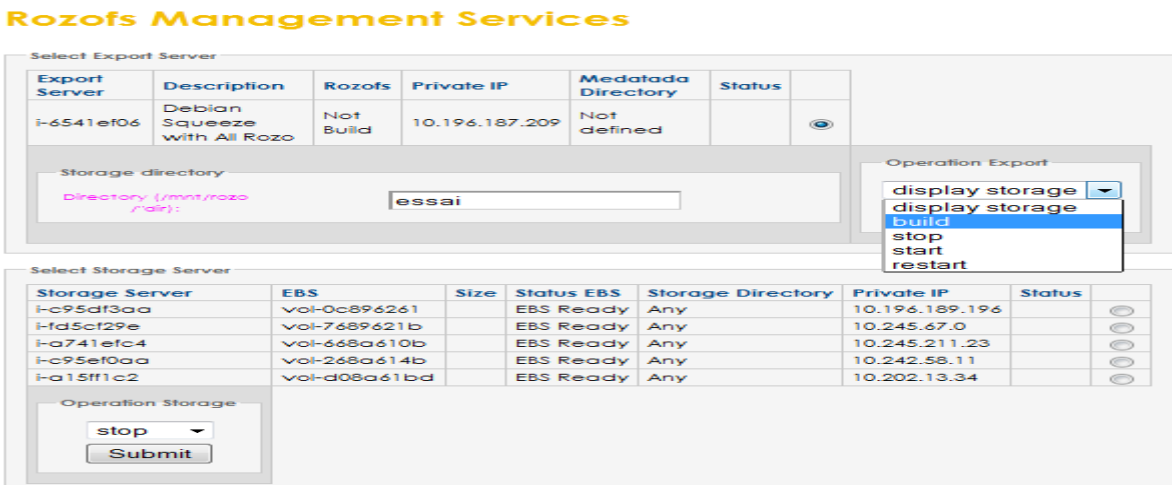


Figure 77 *– Page du module « Rozofs Services » avant la configuration des serveurs.*

Lorsque la configuration est terminée, les informations concernant les serveurs de l'environnement Rozofs se mettent à jour avec notamment la mention « *Build* » pour signaler que les opérations de configuration Rozofs ont été correctement réalisées (*Cf. Figure 78*).

Le tableau affiche également l'emplacement par défaut des méta-données et indique le statut « *running* » qui signifie que le démon « *exportd* » fonctionne.

En ce qui concerne le tableau du bas, la colonne « *Storage Directory* » renseigne l'emplacement du répertoire de stockage pour chaque serveur. Le statut « *running* » indique que le démon « *storaged* » fonctionne.

Figure 78 *– Page du module « Rozofs Services » après la configuration des serveurs.*

4.5.1.4 L'accès à l'espace de stockage

L'environnement Rozofs est maintenant opérationnel, l'utilisateur peut accéder à son espace de stockage avec une station de travail ou une machine virtuelle dont l'application « *rozofs-rozofsmount* » est installée. Pour cela, après avoir créé un répertoire, il procédera au point de montage de l'espace de stockage sur le répertoire avec la commande suivante :

L'utilisateur peut maintenant accéder à son point de montage et écrire ou lire ses fichiers comme dans un répertoire standard. Les projections sont réparties dans quatre serveurs de stockage, le cinquième est un serveur de remplacement. En effet, si au cours d'une transaction en écriture d'un fichier, un des serveurs susceptible de contenir la projection associée tombe en panne, alors le serveur de remplacement prendra le relais. D'autre part, il suffit de trois serveurs sur les cinq pour que l'utilisateur puisse utiliser son fichier. Ainsi l'avantage de cette solution est de continuer à accéder aux fichiers même si un ou plusieurs serveurs de stockage deviennent inopérants.

4.5.2 Récapitulatif des fonctionnalités de l'interface Web « Rozofs »

Le diagramme d'activité (*Cf. Figure 79*) schématise les étapes de fonctionnement pour mettre en œuvre un Cloud Storage. Comme nous pouvons le voir sur ce diagramme, lorsque l'utilisateur accède à la page principale avec « *Rozofs* », une vérification est effectuée dans la base de données pour savoir notamment si la table « *cloud* » comporte des enregistrements liés au compte utilisateur. Si c'est le cas alors ces informations sont affichées dans un tableau (*Cf. Figure 74*).

L'utilisateur a également la possibilité de supprimer complètement son environnement avec un bouton « *Delete* ». Les informations sur l'environnement seront alors supprimées de la base de données.

En outre, l'utilisateur peut accéder à un menu lui proposant d'ajouter un Cloud Storage (*Cf. §4.5.1.1*), de configurer les volumes *EBS* après que l'environnement Amazon soit déployé *(Cf. §4.5.1.2*) et de configurer et gérer les services Rozofs (*Cf. §4.5.1.3*).

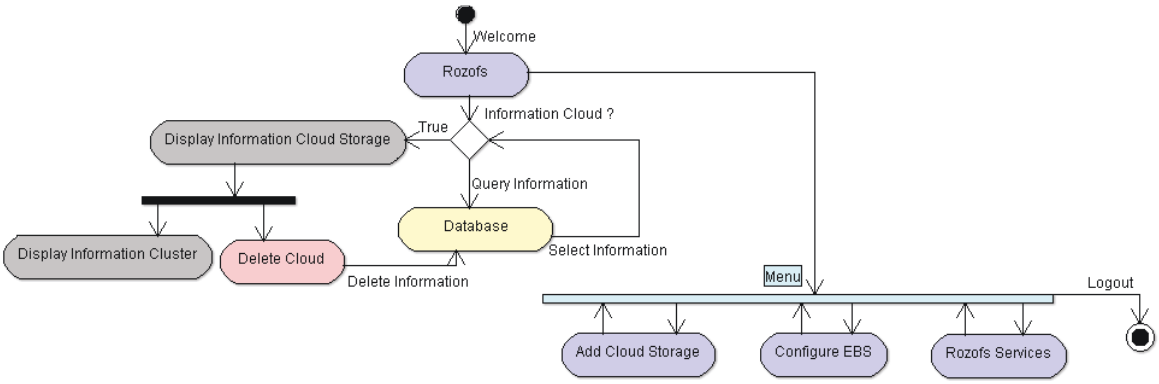


Figure 79 *– Diagramme d'activité de l'application Rozofs.*

4.5.2.1 Détail de fonctionnement de l'application « *Rozofs Services* »

Le diagramme d'activité (*Cf. Figure 80)* indique les différentes interactions pour utiliser les services Rozofs accessible à la page « *services.html* » avec le lien « */rozofs/services/* » depuis la page principale « *rozofs.html* ». On remarque sur le diagramme que les services Rozofs ne sont accessibles que si l'utilisateur possède un environnement.

Les informations associées aux serveurs de méta-données sont stockées dans la table « *export* », tandis que celles concernant les serveurs de stockage figurent dans la table « *storage* » (*Cf. §Annexe A*).

L'application Django récupère donc les informations de la table « *export* » et les affiche dans le tableau en haut de la page « *services.html* », alors que les informations de la table « *storage* » figure dans le tableau du bas de la page (*Cf. §4.5.1.3*).

Ensuite l'application propose trois fonctionnalités *« Configure Rozofs Environnement* », « *Manage Export daemon* » et « *Manage Storage Daemon* ».

Lorsque l'utilisateur sélectionne le service qu'il désire, l'application Django récupère les informations dans la base (*Tables « export » et « storage »)* pour vérifier l'état des démons Rozofs.

Par exemple, lorsque l'utilisateur souhaite configurer son environnement Rozofs, l'application vérifie si l'attribut « *export_rozofs* » comporte une valeur nulle. Si c'est le cas alors l'application Django utilise les APIs Rozofs pour démarrer le processus de configuration de l'environnement, sinon un message avertit l'utilisateur que l'environnement Rozo a déjà été configuré.

Il s'agit du même processus pour gérer le démon « *exportd* » ou chaque démon « *storaged* ».

Lorsque l'utilisateur souhaite par exemple arrêter un démon « *storaged* », l'application Django récupère la valeur de l'attribut « *storage_status* » de la table « *storage* » qui doit être différente de « *stopped* ». Si c'est le cas, alors l'application Django utilise l'API Rozo pour arrêter le processus du serveur de stockage concerné.

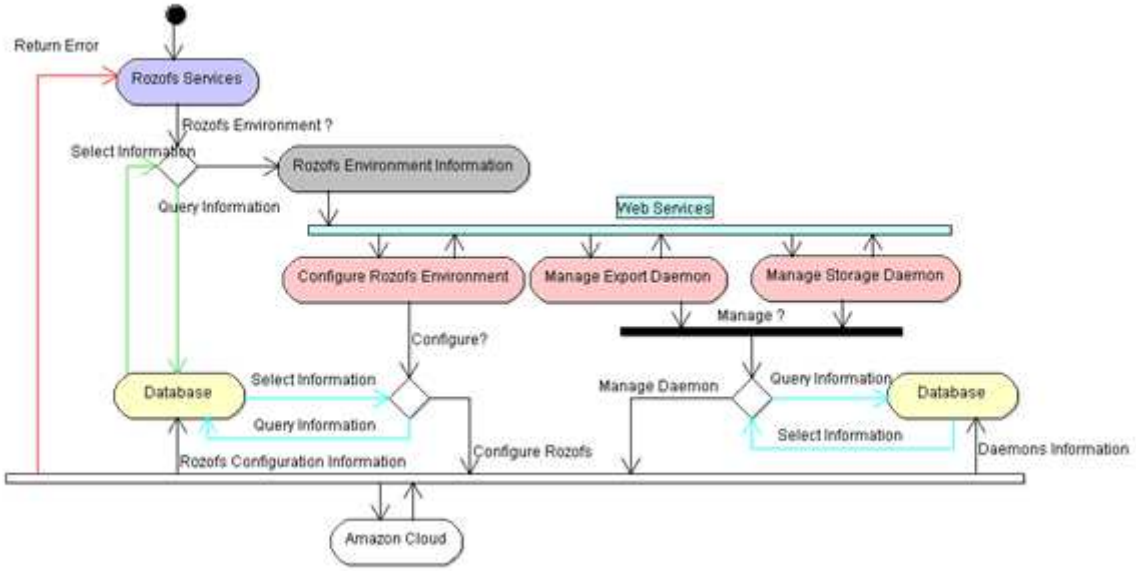


Figure 80 *– Diagramme d'activité du module « Rozofs Services ».*

4.5.2.2 Description de l'application Rozofs au niveau développement

Comme pour l'application « *amazon* », l'interface Web Rozofs est une application Django indépendante avec son propre fichier « *views.py* » qui est constitué de fonctions Python et d'un template nécessaire pour afficher les informations au format Html. Cependant, l'application Rozofs utilise le fichier « *models.py* » de l'application « *amazon* », car les tables « *cloud* », « *export* » et « *storage* » ont une contrainte référentielle avec la table « *account* » et « *instance* » (*Cf. Annexe A*).

Or avec Django, il est plus commode de gérer des tables qui ont des relations communes lorsqu'elles sont définies dans un même fichier. C'est pourquoi dans le fichier « *views.py* » figure toutes les fonctions nécessaires pour mettre en œuvre le Cloud Storage à partir du déploiement des machines virtuelles jusqu'à la configuration des démons Rozo.

Dans ce paragraphe, nous allons détailler la fonction Python du fichier « *views.py* » qui permet de configurer le serveur de méta-données et l'ensemble des serveurs de stockages. Cette fonction qui s'intitule « *services* » intègre l'API Rozofs et renvoie les informations vers le template « *services.html* ».

Nous allons décrire cette fonction en deux phases avec tout d'abord une première partie qui illustre la définition des variables puis l'extraction des données de la base.

Puis une seconde partie qui explique l'utilisation des APIs Rozofs et la mise à jour des données dans la base (*Cf. Figure 81).*

```
def services(request):
----//
if request.method == 'POST
----//
if 'export' in request.POST:
    export_services = param['export_services']
    select_export = request.POST['select_export']
    export = EXPORT.objects.get(export_ref = select_export)
    ip = export.export_private_ip
    status = export.export_rozofs
    export_id = export.pk
    if export_services == 'build':
        mount_dir = request.POST['stor_dir']
        if mount_dir == "":
            msg_action = "you have to indicate directory ([storage]-> /mnt/rozo/[storage])"
        elif status != 'ok':
            directory = '/mnt/rozo/'+str(mount_dir)
            path_export = '/srv/rozofs/export'+str(export_id)
            info_storage = STORAGE.objects.filter(export = export_id)
            number_storage = STORAGE.objects.filter(export = export_id).count()
            os.system('rozo-export -H '+ip+' add-export -c '+path_export+'')
            for item in info_storage:
                res = [
                        item.storage_private_ip,
                        item.pk ]
                os.system('rozo-storage -H '+str(res[0])+' add-storage -c '+str(res[1])+' '+directory+'')
                os.system('rozo-storage -H '+str(res[0])+' stop')
                os.system('rozo-storage -H '+str(res[0])+' start')
                STORAGE.objects.filter(id = str(res[1])).update(storage_status = 'running')
                STORAGE.objects.filter(id = str(res[1])).update(storage_dir = directory)
                List1 = (str(res[1])+':'+str(res[0]))
                List2.append(List1)
            val = " ".join(List2)
            os.system('rozo-export -H '+str(ip)+' add-cluster '+str(export_id)+' '+str(val)+'')
            os.system('rozo-export -H '+ip+' stop')
            os.system('rozo-export -H '+ip+' start')
            EXPORT.objects.filter(export_ref = select_export).update(export_rozofs = 'ok')
            EXPORT.objects.filter(export_ref = select_export).update(export_metadata = path_export)
            EXPORT.objects.filter(export_ref = select_export).update(export_status = 'running')
```

Figure 81 *- Extrait de code Python de la fonction « services » du fichier views.py de l'application « rozofs ».*

a) Phase préliminaire de configuration Rozofs

Tout d'abord, la fonction « *services* » extrait dans un premier temps tous les champs de la table « *export* » qui concerne le compte utilisateur puis les transfère vers le template « *services.html* » (*Cf. Figure 82*). L'utilisateur souhaite alors exécuter le processus de configuration de l'environnement Rozofs. Il choisit pour cela l'option « *build* » (*Cf. §4.5.1.3*) puis clique sur une case à cocher pour sélectionner le serveur de méta-données qu'il souhaite.

En effet, comme un environnement Rozofs est constitué d'un serveur de méta-données qui est associé à plusieurs serveurs de stockage, le fait de désigner le serveur revient à sélectionner l'environnement complet. D'autre part, derrière la case à cocher, se trouve une valeur extraite de l'attribut « *export_ref* » de la table « *export* », il s'agit de l'identifiant Amazon de la machine virtuelle.

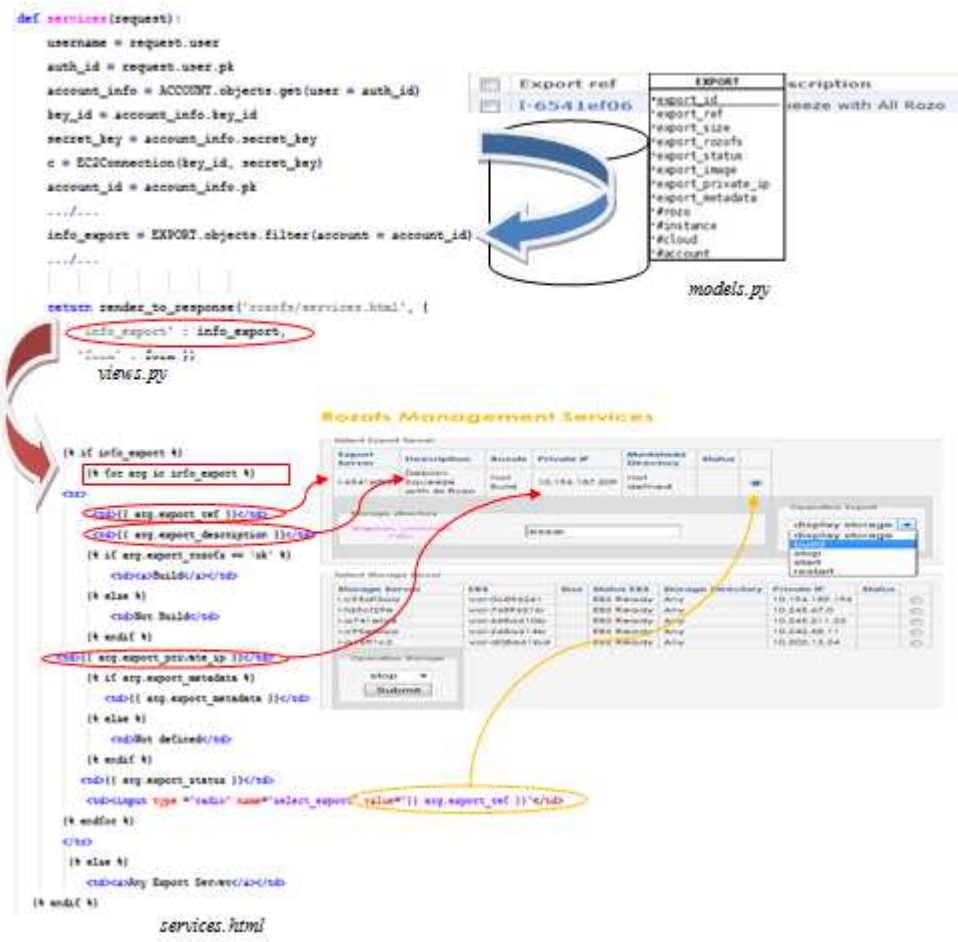


Figure 82 – *Interaction des fichiers Django de l'application « Rozofs ».*

La première partie de la fonction « *services* » consiste tout d'abord à récupérer l'identifiant du serveur de méta-données via la variable « *select_export* » par la méthode « POST » (*Cf. Figure 83*).
Puis elle stocke l'enregistrement de la table « *export* » correspondant à l'identifiant dans une liste d'objets. A partir de cette liste, la fonction récupère l'adresse IP privée et l'affecte dans la variable « *IP* ». Elle procède de la même manière pour la variable « *status* » dont la valeur provient de l'attribut « *export_rozofs* », indique si l'environnement Rozofs est configuré ou non *(Cf. 4.5.2.1)* ; tandis que la variable « *export_id* » contient la clé primaire de l'enregistrement.

Ensuite la fonction vérifie si l'utilisateur sélectionne l'option « *build* » dans le formulaire, indiquée par la variable « *export_services* ». Cette variable contient toutes les options de la liste déroulante du formulaire de gestion du serveur de méta-données (*Cf. §4.5.1.3*). Si ce test est vrai alors la fonction récupère la valeur du champ texte « *stor_dir* » du formulaire, qui sert à définir le nom du répertoire de stockage contenant les projections. Le chemin de l'emplacement (« *directory* ») est prédéfini comme étant « */mnt/rozo/nom_repertoire* ».

La fonction contrôle également la variable « *status* » pour s'assurer que l'environnement Rozofs n'a pas déjà été configuré. Elle définit l'emplacement des méta-données avec la variable « *path_export* ». Cet emplacement est unique par environnement Rozofs puisque la terminaison du nom du répertoire « *export* » est la clé primaire de l'enregistrement de la table « *export* » que l'on a extrait précédemment (« */srv/rozoffs/export[clé_primaire_export]*»).

La première partie de la fonction « *services* » finit par récupérer dans la base tous les serveurs de stockage qui sont associés au serveur de méta-données grâce à la clé primaire « *export_id* » qui l'identifie. Ces enregistrements sont stockés dans la liste « *info_storage* ».

De plus, la fonction affecte à la variable « *number_storage* », le nombre de serveurs de stockage que l'environnement Rozofs comporte.

```
if 'export' in request.POST:
    export_services = param['export_services']
    select_export = request.POST['select_export']
    export = EXPORT.objects.get(export_ref = select_export)
    ip = export.export_private_ip
    status = export.export_rozofs
    export_id = export.pk
    if export_services == 'build':
        mount_dir = request.POST['stor_dir']
        if mount_dir == "":
            msg_action = "you have to indicate directory ([storage]-> /mnt/rozo/[storage])"
        elif status != 'ok':
            directory = '/mnt/rozo/'+str(mount_dir)
            path_export = '/srv/rozofs/export'+str(export_id)
            info_storage = STORAGE.objects.filter(export = export_id)
            number_storage = STORAGE.objects.filter(export = export_id).count()
```

Figure 83 *– Première partie de la fonction « services » du fichier views.py de l'application « rozofs ».*

b) Phase d'exécution de la configuration Rozofs

La fonction « *services* » entre maintenant dans le processus de configuration de l'environnement Rozofs. Tout d'abord, elle effectue un premier appel système pour exécuter la méthode « *rozo-export* » de l'API Rozofs directement à partir du serveur Web.

[rozo-export –H « @ip serveur_méta-données » add-export –c « emplacement des méta-donées »]

Cette commande va créer le répertoire qui contiendra les méta-données puis va déclarer son emplacement dans le fichier « export.conf » (Cf. 3.4.4.4).

Ensuite, la fonction va parcourir la liste « *info_storage* » qui contient plusieurs enregistrements de la table « *storage* » dans une boucle récursive parcourue par la variable « *item* » (*Cf. Figure 84*).

```
def services(request):
----//
if request.method == 'POST
----//
        os.system('rozo-export -H '+ip+' add-export -c '+path_export+'')
        for item in info_storage:
            res = [
                    item.storage_private_ip,
                    item.pk ]
            os.system('rozo-storage -H '+str(res[0])+' add-storage -c '+str(res[1])+' '+directory+'')
            os.system('rozo-storage -H '+str(res[0])+' stop')
            os.system('rozo-storage -H '+str(res[0])+' start')
            STORAGE.objects.filter(id = str(res[1])).update(storage_status = 'running')
            STORAGE.objects.filter(id = str(res[1])).update(storage_dir = directory)
            List1 = (str(res[1])+':'+str(res[0]))
            List2.append(List1)
        val = " ".join(List2)
        os.system('rozo-export -H '+str(ip)+' add-cluster '+str(export_id)+' '+str(val)+'')
        os.system('rozo-export -H '+ip+' stop')
        os.system('rozo-export -H '+ip+' start')
        EXPORT.objects.filter(export_ref = select_export).update(export_rozofs = 'ok')
        EXPORT.objects.filter(export_ref = select_export).update(export_metadata = path_export)
        EXPORT.objects.filter(export_ref = select_export).update(export_status = 'running')
```

Figure 84 *– Seconde partie de la fonction « services » du fichier views.py de l'application « rozofs ».*

Cette boucle récursive, qui permet de configurer l'environnement Rozofs, peut être décomposée en plusieurs étapes (*Cf. Figure 85*).

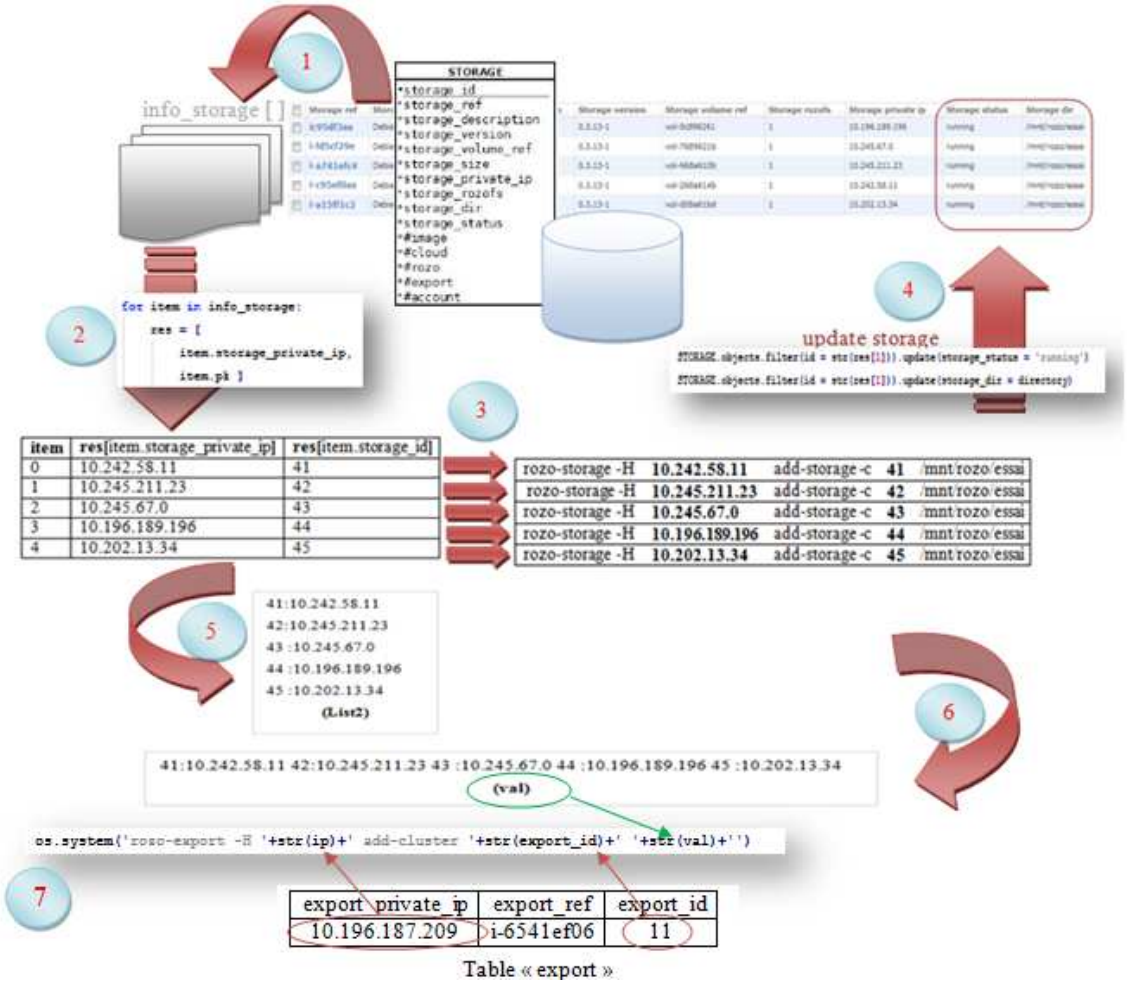


Figure 85 *– Description de la boucle récursive pour configurer chacun des démons « storaged »*

Dans la première étape, la fonction récupère tous les enregistrements de la table « *storage* » associé au serveur de méta-données puis les stocke dans une liste « *info_storage* ».

Puis dans la deuxième étape, avec une boucle for, la fonction « *services* » va parcourir la liste, enregistrement par enregistrement en sélectionnant uniquement les champs « *storage_id* » puis «*storage_private_ip* ». Ainsi lorsque la fonction parcourt le premier enregistrement de la table « *storage* », elle récupère les valeurs des champs sélectionnés, c'est-à-dire la clé primaire et l'adresse IP privée.

Avec ces arguments, la fonction utilise en troisième étape, la méthode « *rozo-storage* » pour ajouter un stockage dans le serveur :

rozo-storage –H « @ip serveur_stockage » add-storage –c « emplacement de stockage»

Cette commande va créer le répertoire « /mnt/rozo/essai » dans le serveur de stockage puis elle va déclarer l'emplacement du répertoire dans le fichier « storage.conf » du serveur en question.

La fonction utilise également la méthode « *rozo-storage* » pour arrêter et redémarrer le démon « *storaged* » : **rozo-storage –H « @ip serveur_stockage » stop**

Ensuite en quatrième étape, la fonction va mettre à jour la table « *storage* » en modifiant l'enregistrement associé à la clé primaire «*storage_id* ». Pour cela, elle attribue la valeur « *running* » à l'attribut « *storage_status* » puis indique le chemin du répertoire de stockage dans l'attribut « *storage_dir* ».

A la cinquième étape, la fonction enregistre dans une nouvelle liste « *List2* », une chaîne composée de l'adresse IP du serveur de stockage et de la clé primaire de l'enregistrement associé sous le format « *@ip_serveur : clé_primaire* ».

La boucle passe aux enregistrements suivants et répète les étapes deux à cinq.

Après avoir parcouru l'ensemble des valeurs dans la liste « *info_storage* », la boucle se termine.

En sixième étape, la fonction « *services* » convertit la liste « *liste2* » en une chaîne de caractères qu'elle affecte à la variable « *val* ».

Enfin à la septième étape, elle utilise la méthode « *rozo-export* » pour finir de configurer le serveur de méta-données : **rozo-export –H « @ip serveur_méta-données » add-cluster « export_id » « val »**

Cette commande permet de déclarer la liste des serveurs de stockages (variable « val ») dans le fichier « export.conf » du serveur de méta-données.

La fonction procède alors à l'arrêt et le redémarrage du démon « *exportd* » du serveur de méta-données avec la méthode « rozo-export »: **rozo-export –H « @ip serveur_méta-données » stop**

Puis elle se termine par une mise à jour dans la table « *export* » de l'enregistrement associé au serveur de méta-données (*clé primaire*) en ajoutant la valeur « *ok* » à l'attribut « *export_rozofs* » pour signaler que la configuration Rozofs est effectuée.

Elle ajoute également une chaine de caractère correspondant à l'emplacement du répertoire des méta-données, dans l'attribut « *export_metadata* ». Et la fonction enregistre la valeur « *running* » à l'attribut « *export*_status » qui indique que le démon « *exportd* » est en cours d'exécution.

4.5.3 Bilan sur l'application Rozofs

Le processus de déploiement d'un Cloud Storage comporte peu d'étapes pour l'utilisateur qui a seulement besoin de déterminer le nombre de serveurs de stockage et de choisir la configuration Amazon qui lui convient (*groupe de sécurité, type d'instance, etc.)*. Ensuite toute la configuration de l'environnement Rozofs est automatisée. Lorsque l'opération est terminée, l'utilisateur récupère toutes les indications nécessaires pour monter l'espace de stockage sur son poste avec l'application « *rozofs-rozofsmount* ». Cette dernière étape s'apparente d'ailleurs à un simple montage NFS avec tous les avantages du mécanisme des projections. En effet, l'utilisateur peut accéder à l'intégralité de ses fichiers même si un des serveurs de stockage tombe en panne. D'autre part, comme le fichier est découpé en blocs de données répartis sur plusieurs serveurs de stockage, personne ne peut lire le fichier sans avoir récupéré un nombre de projections nécessaires. La solution est donc fiable et assure un bon niveau de sécurité notamment contre le piratage d'informations.

La réalisation de l'interface Web de gestion du cloud storage englobe l'application « *amazon* » et celle de « *rozofs* ». Il reste une dernière application qui permet de gérer la connexion des utilisateurs puis de les orienter vers l'interface « *amazon* » ou « *rozofs* ».

4.6 Réalisation de l'application Web de gestion des utilisateurs

L'application Web de gestion des utilisateurs correspond à la première étape de l'interface Web de gestion du Cloud Storage. En effet, elle permet d'autoriser ou non un utilisateur qui souhaite exploiter les fonctionnalités de l'interface. En outre, l'application offre la possibilité de créer un nouveau compte ou de modifier le mot de passe. Une fois que l'utilisateur est authentifié tant sur le serveur Django que chez Amazon, celui-ci accède à la page principale de l'interface.

A partir de là, il peut utiliser à sa guise l'application « *amazon* » (*Cf. §4.4*) ou bien l'application « *rozofs* » (*Cf. §4.5*). A tout instant, il peut fermer sa session en se déconnectant de l'interface.

4.6.1 Présentation de l'application Web de gestion des utilisateurs

Tout d'abord, l'utilisateur se connecte à une page Web qui correspond à l'index de l'application Web. Pour cela, il utilise l'URL *« https://@ip_application_Web »* et accède directement à un formulaire qui lui permet de saisir les identifiants pour se connecter à l'interface Web (*Cf. Figure 86*).

A partir de cette page, l'utilisateur qui dispose déjà d'un compte référencé dans la base de données, peut accéder à la page principale de l'interface Web de gestion qui affiche un message d'accueil.

L'utilisateur peut utiliser soit l'application « *rozofs* » en sélectionnant le lien « *Rozofs* », ou bien se servir de l'application « *amazon* » en cliquant sur le lien « *AWS* » du menu de navigation.

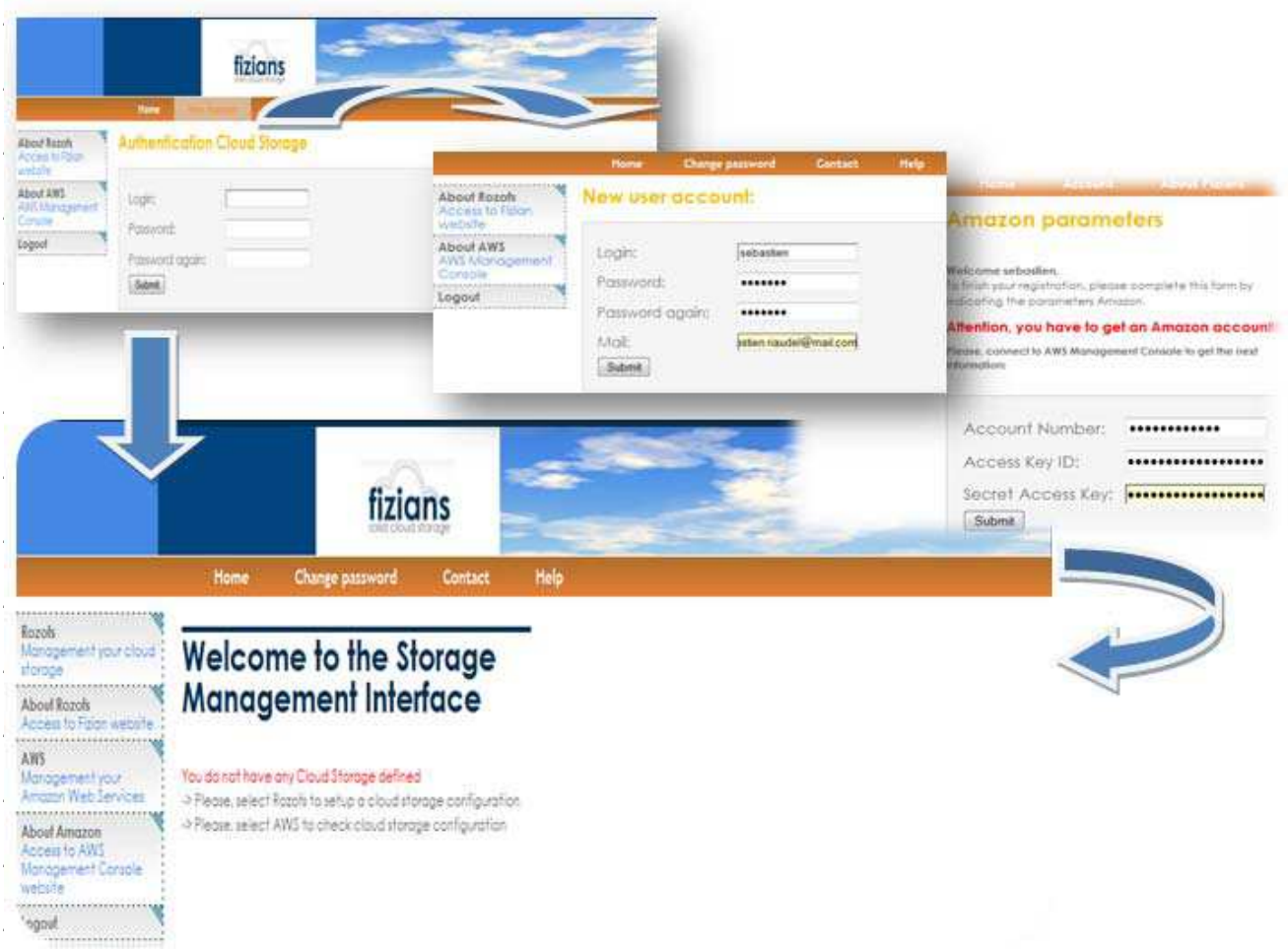


Figure 86 *- Présentation de l'interface Web de gestion des utilisateurs.*

En revanche, si l'utilisateur ne possède pas de compte, il peut en créer un avec le lien « *New Account* » figurant en haut de la page d'authentification. Alors, il devra saisir un identifiant et un mot de passe qui seront enregistrés dans la table « *auth_user* » (*Cf. Annexe A*) de la base de données. Par mesure de sécurité, Django enregistre plutôt la signature du mot de passe dans la base de données, en utilisant l'algorithme de hachage SHA. Ensuite, l'utilisateur est redirigé vers une nouvelle page qui lui permet de saisir les identifiants Amazon pour s'authentifier et pouvoir ainsi utiliser les services AWS.

Lorsqu'il valide le nouveau formulaire, une requête est envoyée chez Amazon pour vérifier si les identifiants de l'utilisateur sont valables. En cas d'erreur retournée par Amazon, l'utilisateur sera redirigé vers une autre page qui lui permettra de rectifier le numéro de compte Amazon ou bien l'identifiant de clé d'accès et la clé secrète associée. Les valeurs concernant les identifiants Amazon, sont stockées dans la table « *Account* » de la base de données Django (*Cf. Figure 87*).

Figure 87 *- Identifiants Amazon enregistrés dans la table « Account ».*

Ensuite, après avoir enregistré les paramètres dans la base de données et vérifié qu'ils sont corrects, l'utilisateur accède à la page principale de l'interface Web.

4.6.2 Description de l'application Web de gestion des utilisateurs

Après avoir présenté l'application Web de gestion des utilisateurs, nous allons décrire le déroulement des opérations à l'aide d'un diagramme d'activité (*Cf. Figure 88*).

Ainsi, lorsque l'utilisateur s'authentifie sur l'interface, un contrôle de ses identifiants Amazon est automatiquement effectué. L'application Web vérifie dans la base de données, si le compte utilisateur est associé à un enregistrement dans la table « *Account* ». Ainsi, tant que l'utilisateur n'aura pas référencé des identifiants Amazon « *valides* » dans la base de données, il ne pourra pas utiliser l'interface Web. Après avoir testé que les identifiants de l'utilisateur sont valides auprès d'Amazon, l'interface Web contrôle dans la base de données, les tables « *keypair* » et « *securitygroup* » pour savoir si l'utilisateur dispose d'une clé SSH et d'un groupe de sécurité (*Cf. §3.4.3.5*).

En effet, il est indispensable de créer une clé SSH chez Amazon avant de déployer une machine virtuelle sinon, l'utilisateur ne pourra pas s'y connecter.

De même, il est recommandé de créer un nouveau groupe de sécurité et d'ajouter les règles nécessaires pour utiliser la machine virtuelle dans de bonnes conditions. Amazon met à disposition un groupe de sécurité « *par défaut* » qui bloque tous les ports TCP/UDP. Avec ce groupe par défaut, l'environnement Rozofs ne peut pas être mis en œuvre.

Par ailleurs, il n'est pas prudent d'ajouter de nouvelles règles de permissions sur ce groupe qui peuvent être utilisées par d'autres machines virtuelles qui n'ont rien à voir avec l'environnement Rozofs.

Ainsi, l'interface Web va créer une clé SSH en fournissant la clé publique chez Amazon.

Cette clé publique sera présente dans la machine virtuelle lors de son déploiement.

L'utilisateur disposera de sa clé privée pour se connecter à la machine virtuelle via le protocole SSH.

De même, l'interface Web ajoute un groupe de sécurité avec les règles de permissions nécessaires pour être en mesure de déployer et d'utiliser l'environnement Rozofs.

Pour créer la clé SSH et le groupe de sécurité, l'interface Web utilise les méthodes Boto (*Cf.4.4.2*).

Ces méthodes retournent des informations qui seront stockées dans la base de données.

Puis, une fois que l'utilisateur possède une clé SSH et un groupe de sécurité, il peut accéder à la page principale de l'interface Web. Il sera en mesure de déployer son environnement Rozofs ou bien de gérer son environnement Amazon. Lors d'une prochaine connexion, l'utilisateur aura simplement besoin de saisir les identifiants de son compte Web, l'interface retrouvera automatiquement les identifiants Amazon associés.

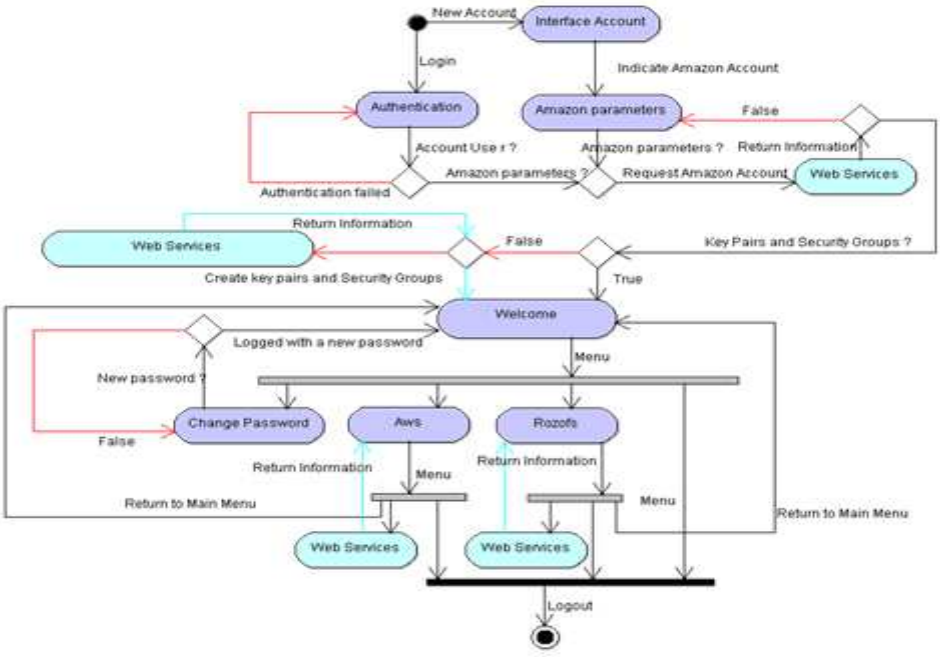


Figure 88 *– Description de l'application Web de gestion utilisateurs.*

4.6.3 Bilan de l'application Web de gestion des utilisateurs

La réalisation de cette application repose sur le mécanisme d'authentification Django grâce à l'interface d'administration. La table « *auth_user* » qui référence les utilisateurs de l'interface Web est générée lors de la mise en œuvre de l'interface administration Django (*Cf. §4.3.2*).

Ainsi, pour réaliser notre application de gestion utilisateur, il a simplement fallu construire une nouvelle table « *account* » liée par une contrainte référentielle à la table « *auth_user* ».

Cette table enregistre les identifiants Amazon et évite ainsi à l'utilisateur de les saisir pour utiliser les services AWS.

De même, avec l'interface d'administration, la gestion des utilisateurs est facilement réalisable car il est possible d'ajouter des droits ou de supprimer un compte. La suppression d'un compte aura pour conséquence, de vider tous les enregistrements associés à l'utilisateur. En effet, toutes les tables que l'interface Web de gestion utilise, dépendent de la table « *account* ».

4.7 Bilan sur la partie réalisation

La mise en œuvre de l'interface Web de gestion d'un Cloud Storage repose essentiellement sur l'exploitation de Django et du langage Python. Bien qu'une base de données soit utilisée pour gérer les données de l'environnement Rozofs, il n'est pas nécessaire de connaître le langage SQL, car Django utilise son propre formalisme pour manipuler les objets figurant dans la base de données.

D'autre part, l'interface Web utilise en majeure partie les services Web AWS, car l'architecture même de l'environnement Rozofs est fournie par Amazon. Cependant, comme les applications Django sont interopérables, rien n'empêche de prévoir une autre application qui utilisera les services Web d'autres fournisseurs tels que « *Gogrid* » ou « *Rackspace* » (*Cf. §2.1.9)* afin d'avoir un espace de stockage qui ne dépend pas d'un fournisseur particulier.

Toutefois, quelques fonctionnalités doivent être améliorées afin d'assurer tout d'abord une disponibilité permanente de l'interface Web et de garantir un certain confort d'utilisation.

Par exemple, il faudrait créer une deuxième interface Web qui viendrait épauler la première pour garantir d'une part la montée en charge puis d'autre part, pour assurer le fonctionnement permanent si la première interface venait à dysfonctionner.

D'autre part, si le serveur physique qui contient l'interface Web redémarre, alors l'adresse IP privée et le nom de la machine sont modifiés automatiquement. Sans doute une volonté d'Amazon d'obliger les clients à laisser leurs machines en fonctionnement, car le paiement est à l'usage.

Mais, cela oblige l'administrateur du serveur à modifier la configuration Apache notamment le paramètre « *NameVirtualHost* » qui pointe sur l'adresse IP privée et permet à l'utilisateur d'accéder à la page Web d'accueil. En effet, bien que le client accède à l'interface avec une adresse IP publique qui ne change pas, le serveur Apache a besoin de l'adresse IP privée pour que l'application Web soit accessible.

La dernière étape qui consiste à monter le système de fichiers pourrait être automatisée par l'application Web afin d'assurer une certaine souplesse dans l'utilisation. L'application pourrait également démonter le système de fichiers, faire un inventaire des fichiers situés dans l'espace de stockage, etc.

Bref, l'application Web de gestion du Cloud Storage n'a pas fini d'être améliorée et optimisée tout comme les applications Rozofs de Fizians et les services Web d'Amazon.

Conclusion

5.1 Rappel des objectifs

Tout d'abord, l'objectif initial consiste à réaliser une interface Web de type « *saas* » *(software as a services*) pour gérer une solution de Cloud Storage hébergée chez Amazon avec les applications Rozofs de Fizians. Le Cloud Storage est une architecture Web qui met à disposition un espace de stockage évolutif et élastique en tant que service. Pour cela, Amazon fournit l'environnement qui permet de composer l'espace de stockage. Cet environnement est constitué d'un ensemble de machines virtuelles et de volumes de données qui s'adaptent suivant le besoin en capacité de traitement de stockage. Puis, la solution de Cloud Storage s'appuie sur les propriétés du système de fichiers distribués « *Rozofs* » présent sur chaque machine virtuelle. En effet « *Rozofs* » utilise un code correcteur (« *la transformée Mojette* ») qui permet de découper un fichier en blocs de données sur lesquels seront effectués des projections. Ces projections sont stockées dans des volumes attachés aux serveurs de stockage. Et l'ensemble de ces volumes forme un espace de stockage global monté dans un système de fichiers du poste client. L'avantage du Cloud Storage avec Rozofs est d'assurer l'accès aux données même si un ou deux serveurs deviennent inopérants, ceci par l'intermédiaire d'un nombre de projections suffisant pour reconstruire le fichier. Ensuite, les données transformées en projections sont inexploitables sans l'application cliente de Rozo, ce qui garantit une solution de confidentialité.

De plus, l'ordre de récupération des projections pour reconstruire le fichier est aléatoire.

En outre, le transfert des projections entre les serveurs de stockage et le poste client est optimisé, car d'une part, la taille des données envoyées est faible (8 Ko) et d'autre part, il y a une meilleure répartition de la charge puisque les projections sont réparties sur plusieurs serveurs.

5.2 Bilan du projet

Ainsi, en combinant un environnement Amazon, qui est basé sur une offre de services à la demande, avec une solution de stockage fiable et élastique, nous réalisons une architecture de stockage dans le nuage plutôt performante. L'interface Web de gestion joue donc un élément central qui permet de mettre en œuvre un environnement de stockage d'une manière transparente et relativement souple pour l'utilisateur. Ce dernier n'a pas à se soucier de créer les machines virtuelles, d'installer et de configurer les applications, l'interface le fait pour lui. De plus, l'utilisateur peut construire son propre environnement Amazon d'une manière indépendante de l'environnement de stockage. La connexion à l'interface est sécurisée avec un premier système d'authentification pour se connecter à l'interface Web puis un second pour utiliser les services Web AWS.

Cette interface est réalisée en langage Python avec le cadriciel Django qui permet de réaliser d'une manière relativement simple, des applications portables, modulaires, interopérables et dynamiques avec l'utilisation d'une base de données. A partir d'un modèle de classes, Django construit le schéma de la base de données puis implémente des fonctions Python capables de manipuler les objets de la base (*Attribut d'une table*) sans utiliser de manière explicite le langage SQL. Cette base de données est nécessaire pour le serveur Django afin d'afficher les informations utiles dans les pages Web. D'ailleurs, toutes les informations de l'environnement Rozofs et d'Amazon proviennent de la base de données.

D'autre part, avec Django, on dispose d'une interface d'administration qui permet de gérer les comptes utilisateurs, les enregistrements de la base de données et le contenu des pages Web.

Par conséquent, l'objectif initial du projet est bien rempli, car l'application Web gère complètement l'environnement Amazon et permet de configurer automatiquement l'environnement Rozofs. L'utilisateur final n'a plus qu'à monter le système de fichiers distant avec l'application client de Rozo installée sur son poste.

5.3 Perspective d'évolution

Une première évolution de l'interface Web serait d'intégrer des librairies Python pour utiliser les services Web d'autres fournisseurs de Cloud tels que « *Google* », « *Gogrid* », « *Rackspace* » afin de ne pas dépendre d'un fournisseur en particulier. On pourrait également s'intéresser aux solutions open-source avec notamment « *Eucalyptus* » ou « *Openstack* ». Ensuite, il faudrait améliorer l'architecture de l'interface Web afin de mettre en place un système d'équilibrage de charges.

En effet, comme l'interface est utilisée par tous les utilisateurs, il faut prévoir une roue de secours en cas de dysfonctionnement de l'application Web.

D'autre part, pour améliorer l'efficacité de l'interface Web, il faudrait ajouter une solution de monitoring sur les serveurs afin de connaitre en permanence l'espace disque restant, la remontée des erreurs, etc.

De plus, l'utilisateur pourrait, à partir de l'interface Web, effectuer un montage automatique de l'espace de stockage vers son poste client. L'inverse serait également possible avec le démontage du système de fichiers.

Par ailleurs, il faudrait mettre en place un système de tarification qui n'est pas implémenté sur l'interface car le mode de facturation n'est pas déterminé. En effet, la facturation peut être basée sur un forfait d'utilisation de l'interface Web, sur un pourcentage d'allocation de l'espace de stockage ou bien, sur un système de licences des applications Rozo.

Comme on peut le constater, il y a encore beaucoup de choses à faire pour améliorer l'interface Web de gestion dont l'objectif principal est de fournir un service de stockage fiable à la demande.

5.4 Bilan personnel

La réalisation de ce mémoire a été pour moi l'occasion de travailler dans un laboratoire de recherche.
J'ai pu ainsi me consacrer à plein temps sur l'étude, l'expérimentation et la réalisation du projet.
De plus, le stage m'a permis de côtoyer des gens formidables qui n'hésitent pas à donner de leur temps pour expliquer et faite avancer les choses d'une manière pédagogique. Aussi, le laboratoire de recherche offre un mélange de cultures en regroupant des chercheurs de nationalités différentes et le fait de travailler dans un contexte cosmopolite est socialement enrichissant.
Ensuite, le sujet proposé par Monsieur Evenou, responsable de la société Fizians, en collaboration avec mon tuteur JeanPierre Guédon, correspondait à ce que j'espérais. En effet, le Cloud Computing et plus particulièrement le Cloud Storage, reposent sur la virtualisation qui est l'un de mes domaines de prédilection avec tout ce qui se rapproche de l'infrastructure des systèmes (*Serveurs, Stockages, etc.)*.
D'autre part, la conception d'un environnement de stockage hébergé, exige d'avoir des connaissances pluridisciplinaires (*réseau, développement, système, base de données, etc.*).
En l'occurrence, le projet de réalisation du mémoire m'a permis de découvrir le langage Python qui permet de faire des applications, des scripts d'administration sur des systèmes hétérogènes.
Ainsi s'achève ce mémoire qui marque la fin d'une longue péripétie commencée en 2004 avec la première unité de valeur du Cycle B (*réseaux et télécommunications*) en cours du soir.
Le courage et la ténacité sont sans aucun doute les qualités requises pour parvenir au bout de mon parcours pour devenir ingénieur au Cnam.

.

Table des annexes

Annexe A

a) Description de la table ACCOUNT

La table contient toutes les données utiles pour utiliser les services AWS d'Amazon et ceux de Rozo. Sans ces données, l'utilisateur ne pourra pas utiliser l'interface web de gestion.

Le tableau indique la description des attributs avec notamment ceux qui permettent d'enregistrer les identifiants de connexion Amazon.

Table ACCOUNT						
Attribut	**Type de données**	**Taille maximum**	**Clé primaire**	**Clé étrangère**	**Obligatoire**	**Description**
account_id	Smallint		Auto_increment		Oui	Identifiant unique d'un n-uplet
account_number	BigInt				Oui	Numéro du compte utilisateur Amazon
Key_id	Char	50			Oui	Identifiant de la clé d'accès
Secret_key	Char	50			Oui	Identifiant de la clé d'accès secrète
Date_account	Date				Oui	Commentaire
user	Smallint			Oui	Oui	Identifiant utilisateur

L'attribut « *user* » contient la clé primaire d'une table créée par Django pour authentifier les utilisateurs à se connecter sur le site. Il s'agit de la table « *auth_user* »

Attribut	**Type de données**	**Taille maximum**	**Clé primaire**	**Obligatoire**
id	Smallint		Auto_increment	Oui
username	VarChar	30		Oui
firstname	VarChar	30		Oui
last_name	VarChar	30		Oui
email	VarChar	75		Oui
password	VarChar	128		Oui
Is_staff	Tinyint			Oui
Is_active	Tinyint			Oui
Is_superuser	Tinyint			Oui
Last_login	Date			Oui
date_joined	Date			Oui

Une fois cette table construite, on peut effectuer une projection sur un jeu de données.

```
+----+--------------+----------------+--------------------------------+-----------+-------+
| id | account_number | key_id         | secret_key                     | date_account | user_id |
+----+--------------+----------------+--------------------------------+-----------+-------+
| 2 |  981509411562  | Axxxxxxxxxx | kN1Rxxxxxxxxxxxxxxxxxx | 2011-10-20  |     2 |
+----+--------------+----------------+--------------------------------+-----------+-------+
```

b) Description de la table KEYPAIR

Cette table contient toutes les données sur les clés SSH qui permettent de se connecter à la machine virtuelle avec un terminal. Chaque n-uplet dépend d'un compte Amazon.

Table KEYPAIR						
Attribut	**Type de données**	**Taille maximum**	**Clé primaire**	**Clé étrangère**	**Obligatoire**	**Description**
keypair_id	Smallint		Auto_increment		Oui	Identifiant unique d'un n-uplet
keypair_name	Char	30			Oui	Nom de la clé SSH
date_keypair	Date				Oui	Identifiant de la clé d'accès
locate_keypair	Char	100			Non	Emplacement de la clé SSH privée
account	Smallint			Oui	Oui	Contrainte référentielle

Une fois cette table construite, on peut effectuer une projection sur un jeu de données.

```
+----+-----------+-----------+-----------------------------------+----------+
| id | keypair_name | date_keypair | locate_keypair                    | account_id |
+----+-----------+-----------+-----------------------------------+----------+
| 1 | fizians2    | 2011-10-20   | /home/users/981509411562/.ssh/fizians2.pem |       2 |
+----+-----------+-----------+-----------------------------------+----------+
```

c) *Description de la table SECURITYGROUP*

Cette table contient le nom et la description de chaque groupe de sécurité qui permet de définir les ports TCP/UDP autorisés en entrée/sortie. Chaque n-uplet dépend d'un compte Amazon.

Table SECURITYGROUP						
Attribut	**Type de données**	**Taille maximum**	**Clé primaire**	**Clé étrangère**	**Obligatoire**	**Description**
securitygroup_id	Smallint		Auto_increment		Oui	Identifiant unique d'un n-uplet
group_name	Char	50			Oui	Nom du groupe de sécurité
group_description	Char	200			Non	Description du groupe
date_group	Date				Oui	Date de création
account	Smallint			Oui	Oui	Contrainte référentielle

Une fois cette table construite, on peut effectuer une projection sur un jeu de données.

```
+----+----------+-----------------------------------------------------------------+----------+----------+
| id | group_name | group_description                                               | date_group | account_id |
+----+----------+-----------------------------------------------------------------+----------+----------+
| 2 | fizians2   | This is a security group created by Fizians: ssh, http and https access are autorised | 2011-10-20 |       2 |
+----+----------+-----------------------------------------------------------------+----------+----------+
```

d) *Description de la table SECURITYRULE*

Comme il est possible de définir plusieurs règles pour un groupe de sécurité, nous avons créé la table « *securityrule* » pour les enregistrer. Chaque n-uplet dépend d'un groupe de sécurité.

Table SECURITYRULE						
Attribut	**Type de données**	**Taille maximum**	**Clé primaire**	**Clé étrangère**	**Obligatoire**	**Description**
securityrule_id	Smallint		Auto_increment		Oui	Identifiant unique d'un n-uplet
protocol	Char	30			Oui	Protocol IP (tcp, udp)
from_range_port	Date				Oui	Début de la plage de ports autorisés
to_range_port	Char	100			Oui	Fin de la plage de ports autorisés
cidr	Smallint				Oui	Sous-réseau d'application
date_rule						Date de création de la règle
securitygroup				Oui	Oui	Contrainte référentielle

Une fois cette table construite, on peut effectuer une projection sur un jeu de données.

```
+----+--------+-------------+-----------+---------+-----------+---------------+
| id | protocol | from_range_port | to_range_port | cidr    | date_rule  | securitygroup_id |
+----+--------+-------------+-----------+---------+-----------+---------------+
| 6 | tcp    |         443 |       443 | 0.0.0.0/0 | 2011-10-20 |           2 |
| 5 | tcp    |          80 |        80 | 0.0.0.0/0 | 2011-10-20 |           2 |
| 4 | tcp    |          22 |        22 | 0.0.0.0/0 | 2011-10-20 |           2 |
+----+--------+-------------+-----------+---------+-----------+---------------+
```

e) Description de la table IMAGE

Cette table contient toutes les informations sur l'image (*AMI*) dont l'utilisateur dispose. Cette image est enregistrée dans un conteneur de données (*Amazon S3*) et peut être créée à partir d'un snapshot (*Instantané*) ou fournie par Amazon (*Image publique*).

Chaque n-uplet dépend d'un compte Amazon et éventuellement d'un snapshot particulier.

Table IMAGE						
Attribut	**Type de données**	**Taille maximum**	**Clé primaire**	**Clé étrangère**	**Obligatoire**	**Description**
Image_id	Smallint		Auto_increment		Oui	Identifiant unique d'un n-uplet
image_ref	Char	50			Non	Référence de l'image
Image_architecture	Char	50			Non	Architecure du cpu (32 ou 64 bits)
image_block_device_mapping	Char	200			Non	Informations volume EBS (si l'image en dispose)
image_description	Char	200			Non	Commentaire
image_hypervisor	Char	30			Non	Type d'hypeviseur (ovm ou xen)
image_is_public	Char	30			Non	Si l'image est publique
image_kernel_id	Char	30			Non	Version du noyau système
image_location	Char	120			Non	Emplacement de l'image
image_name	Char	100			Non	Nom de l'image
image_owner_alias	Char	30			Non	Alias Amazon ou compte utilisateur
image_owner_id	Char	30			Non	Identifiant du compte utilisateur
image_platform	Char	30			Non	Type de l'OS (Linux ou Windows)
image_root_device_name	Char	30			Non	Emplacement du disque racine
image_root_device_type	Char	30			Non	Type de disque utilisé (EBS ou instance-store)
image_state	Char	30			Non	Etat de l'image (disponible ou indisponible)
image_type	Char	50			Non	Type d'image (machine, kernel, ramdisk)
image_virtualization_type	Char	50			Non	Type de virtualisation (paravirtuel ou hvm)
image_size	Char	30			Non	Taille du volume EBS attaché à l'image
image_key	Char	50			Non	Nom d'une étiquette de l'image
image_value	Char	50			Non	Valeur de l'étiquette de l'image

Une fois cette table construite, on peut effectuer une projection sur un jeu de données.

```
+------------+---------------------------+------------------+---------+---------+---------------+---------+
| image_ref  | image_description         | image_root_device_type | image_size | date_image | image_snapshot_id | account_id |
+------------+---------------------------+------------------+---------+---------+---------------+---------+
| ami-7d68aa14 | Debian amd-64 squeeze with all rozo | ebs              | 1 Go    | 2011-10-20 |               |       2 |
+------------+---------------------------+------------------+---------+---------+---------------+---------+
```

f) Description de la table INSTANCE

Cette table contient toutes les informations sur la machine virtuelle déployée à partir d'une image. Parmi les informations importantes figurent notamment l'identifiant de la machine (*instance_id*), les adresses *IP* et *DNS* publiques et privées, le nom de la clé *SSH* pour se connecter à la machine, le type de volume disque (*EBS ou Instance_store*), la taille du volume puis l'état de la machine.

La table dispose de nombreuses contraintes référentielles avec les tables « *account* », « *image* », « *keypair* », « *securitygroup* », « *elasticip* », « *export* » puis « *storage* ». Parmi elles, seules « *account* » et « *image* » ont un caractère obligatoire. En effet, la machine virtuelle (*Instance*) est forcément associée à un compte utilisateur et provient obligatoirement d'une image.

En revanche, bien que cela soit recommandé, elle n'est pas obligée d'avoir une clé SSH (« *keypair* ») et un groupe de sécurité crée par l'utilisateur (*autre que le groupe de sécurité par défaut fourni par Amazon*). Cependant, sans clé *SSH*, la machine virtuelle est inutilisable.

D'autre part, la machine ne dispose pas par défaut d'une adresse *IP* fixe (« *elasticip* ») et n'a pas nécessairement le rôle de serveur de méta-données (« *export* ») ou serveur de stockage (« *storage* »).

Table INSTANCE						
Attribut	**Type de données**	**Taille maximum**	**Clé primaire**	**Clé étrangère**	**Obligatoire**	**Description**
Instance_id	Smallint		Auto_increment		Oui	Identifiant unique d'un n-uplet
instance_ref	Char	50			Oui	Référence de l'instance
instance_architecture	Char	50			Non	Architecure du cpu (32 ou 64 bits)
instance_block_device_mapping	Char	200			Non	Informations volume EBS (si l'instance en dispose)
instance_dns_name	Char	50			Non	Adresse DNS publique
instance_hypervisor	Char	50			Non	Type d'hyperviseur (ovm ou xen)
instance_image_id	Char	50			Non	Référence de l'image
instance_ip_address	Char	50			Non	Adresse IP Publique
instance_kernel	Char	50			Non	Version du noyau système
instance_key_name	Char	30			Non	Nom de la clé SSH
instance_launch_time	Char	50			Non	Date et heure de démarrage de l'instance
instance_placement	Char	50			Non	Région où est hébergée l'instance
instance_private_dns_name	Char	120			Non	Adresse DNS privée
instance_private_ip_address	Char	120			Non	Adresse IP privée
instance_public_dns_name	Char	120			Non	Adresse DNS publique
instance_root_device_name	Char	50			Non	Emplacement du disque racine
instance_root_device_type	Char	50			Non	Type de disque utilisé (EBS ou instance-store)
instance_state	Char	50			Non	Etat de l'instance (running, stopped)
instance_type	Char	50			Non	Type de l'instance (micro, small)
instance_virtualizationType	Char	50			Non	Type de virtualisation (paravirtuel ou hvm)
instance_volume_ref	Char	50			Non	Référence du volume EBS attaché à l'instance
instance_volume_status	Char	50			Non	Statut du volume EBS (attached, detached)
instance_key	Char	50			Non	Nom d'une étiquette de l'instance
instance_value	Char	50			Non	Valeur de l'étiquette de l'instance
date_instance	Date				Oui	Date de déploiement
account	Smallint			Oui	Oui	Contrainte référentielle
image	Smalint			Oui	Oui	Contrainte référentielle
keypair	Smallint			Oui	Non	Contrainte référentielle
securitygroup	Smallint			Oui	Non	Contrainte référentielle
elasticip	Smallint			Oui	Non	Contrainte référentielle
export	Smallint			Oui	Non	Contrainte référentielle
storage	Smallint			Oui	Non	Contrainte référentielle

Une fois cette table construite, on peut effectuer une projection sur un jeu de données

```
+-------------+-----------------+------------------+-----------------+--------------------+----------------+----------+---------------+----------+----------+
| instance_ref | instance_image_id | instance_ip_address | instance_key_name | instance_volume_ref | instance_state | image_id | securitygroup_id |
keypair_id | account_id |
+-------------+-----------------+------------------+-----------------+--------------------+----------------+----------+---------------+----------+----------+
| i-225b4142  | ami-7d68aa14    | 107[redacted]    | fizians2        | vol-4554562f       | running        |    1 |             2 |        1 |        2 |
+-------------+-----------------+------------------+-----------------+--------------------+----------------+----------+---------------+----------+----------+
```

g) Description de la table ELASTICIP

Dans cette table figure les informations sur les adresses *IP* fixes publiques (*Elastic IP*). Elle dispose d'une contrainte référentielle obligatoire sur la table « *account* ».

En revanche, la contrainte référentielle sur la table « *instance* » n'est pas une obligation, car une adresse *IP* fixe peut être dissociée de l'instance (*d'où le terme « elastic-ip»).*

Table ELASTICIP						
Attribut	**Type de données**	**Taille maximum**	**Clé primaire**	**Clé étrangère**	**Obligatoire**	**Description**
elasticip_id	Smallint		Auto_increment		Oui	Identifiant unique d'un n-uplet
public_ip	Char	30			Oui	Adresse IP publique fixe
date_ip	Date				Oui	Date d'enregistrement
Instance_name	Char	100			Non	Référence de l'instance cible
instance	Smallint			Oui	Non	Contrainte référentielle
account	Smallint			Oui	Oui	Contrainte référentielle

Une fois cette table construite, on peut effectuer une projection sur un jeu de données.

```
+----+--------------+----------+---------------+------------+-------------+
| id | public_ip    | date_ip  | instance_name | account_id | instance_id |
+----+--------------+----------+---------------+------------+-------------+
|  1 | 107.22.216/162 | 2011-10-25 | i-225b4142  |          2 |          27 |
+----+--------------+----------+---------------+------------+-------------+
```

h) Description de la table VOLUME

Dans cette table (*Cf. tableau XI*) figure les informations relatives aux volumes *EBS* qui contiennent des données persistantes. Comme pour l'adresse *IP* fixe, un volume *EBS* peut être détaché d'une instance d'où le caractère non obligatoire de la contrainte référentielle avec la table « *instance* ». Tandis que celle-ci dispose d'une contrainte référentielle obligatoire sur la table « *account* ».

Table VOLUME						
Attribut	**Type de données**	**Taille maximum**	**Clé primaire**	**Clé étrangère**	**Obligatoire**	**Description**
volume_id	Smallint		Auto_increment		Oui	Identifiant unique d'un n-uplet
volume_ref	Char	30			Oui	Référence du volume EBS
volume_attach_data_device	Char	50			Non	Emplacement du volume
volume_attach_data_instance	Char	50			Non	Référence de l'instance cible
volume_attach_data_status	Char	50			Non	Etat du volume EBS (attached, detached)
volume_create_time	Char	50			Non	Date de création
volume_snapshot_id	Char	50			Non	Référence du Snapshot
volume_size	Int	30			Non	Taille du volume EBS
volume_status	Char	50			Non	Statut du volume (available, in-use, error)

Table VOLUME (suite)						
Attribut	**Type de données**	**Taille maximum**	**Clé primaire**	**Clé étrangère**	**Obligatoire**	**Description**
volume_zone	Char	50			Non	Région où est situé le volume
instance	Smallint			Oui	Non	Contrainte référentielle
account	Smallint			Oui	Oui	Contrainte référentielle

Une fois cette table construite, on peut effectuer une projection sur un jeu de données.

```
+----+------------+---------------------+----------------------+----------------+----------+-------------+-----------+------------+------------+
| id | volume_ref | volume_attach_data_device | volume_attach_data_instance | volume_snapshot_id | volume_size | volume_status | volume_zone | instance_id | account_id |
+----+------------+---------------------+----------------------+----------------+----------+-------------+-----------+------------+------------+
| 49 | vol-fd545697 | /dev/sdc | i-ae5b41ce | snap-f0728293 | 1 | in-use | us-east-1a | 28 | 2 |
+----+------------+---------------------+----------------------+----------------+----------+-------------+-----------+------------+------------+
```

i) Description de la table SNAPSHOT

Cette table permet d'obtenir des informations sur les snapshots créés à partir d'un volume ou lors du déploiement d'une image pour obtenir une machine virtuelle.

Avec un snapshot, l'utilisateur peut également créer un autre snapshot, un volume *EBS* ou bien une image. L'idée est de faire partager les données à un autre utilisateur ou bien de procéder à des points de sauvegarde.

Un snapshot est tout simplement une photo à un instant précis de l'image ou du volume EBS que l'on souhaite conserver par précaution.

Cette table et celle intitulée « *volume* » font l'objet d'une double relation multiple. En effet, avec un volume, il est possible de créer plusieurs snapshots et avec un snapshot, on peut créer plusieurs volumes.

Normalement, la règle de passage dans ce type de cardinalité serait de créer une troisième table contenant les clés primaires de la table « *volume* » et « *snapshot* ». Mais pour économiser une jointure, on affecte la référence du volume dans la table « *snapshot* » et la référence du snapshot dans la table « *volume* ». La table « *snapshot* » dispose malgré tout d'une contrainte référentielle obligatoire avec la table « *account* ».

Table SNAPSHOT						
Attribut	**Type de données**	**Taille maximum**	**Clé primaire**	**Clé étrangère**	**Obligatoire**	**Description**
snapshot_id	Smallint		Auto_increment		Oui	Identifiant unique d'un n-uplet
snapshot_ref	Char	30			Oui	Référence du snapshot
snapshot_description	Char	200			Non	description
snapshot_status	Char	50			Non	Status du snapshot (completed, error)
snapshot_start_time	Char	50			Oui	Date d'initialisation
snapshot_volume_id	Char	30			Non	Référence du volume EBS
snapshot_volume_size	Int	30			Non	Taille du volume EBS
date_snapshot	Date				Oui	Date d'enregistrement
account	Smallint			Oui	Oui	Contrainte référentielle

Une fois cette table construite, on peut effectuer une projection sur un jeu de données.

```
+----+-------------+-------------+----------------+-----------------+----------+
| id | snapshot_ref | snapshot_status | snapshot_volume_id | snapshot_volume_size | account_id |
+----+-------------+-------------+----------------+-----------------+----------+
| 1 | snap-eb4cab84 | completed     | vol-264b5e4e     |              8 |       2 |
+----+-------------+-------------+----------------+-----------------+----------+
```

j) Description de la table ROZO

Cette table permet d'enregistrer les versions des différentes applications *Rozo*.

Ces informations sont utiles pour l'administrateur et pour l'utilisateur notamment pour les mises à jour.

Table ROZO						
Attribut	**Type de données**	**Taille maximum**	**Clé primaire**	**Clé étrangère**	**Obligatoire**	**Description**
rozo_id	Smallint		Auto_increment		Oui	Identifiant unique d'un n-uplet
Version_rozofs_storaged	Char	50			Oui	Version de l'application
date_rozofs_storaged	Date				Oui	Date de la version
Version_rozofs_exportd	Char	50			Oui	Version de l'application
date_rozofs_exportd	Date				Oui	Date de la version
Version_rozofs_rozofsmount	Char	50			Oui	Version de l'application
date_rozofs_rozofsmount	Date				Oui	Date de la version
Version_rozo	Char	50			Oui	Version de l'application
date_rozo	Date				Oui	Date de la version

Une fois cette table construite, on peut effectuer une projection sur un jeu de données.

```
+----+-------------------+-----------------+-----------------+----------------+--------------------+-------------------+----------+----------+
| id | Version_rozofs_storaged | date_rozofs_storaged | Version_rozofs_exportd | date_rozofs_exportd | Version_rozofs_rozofsmount |
date_rozofs_rozofsmount | Version_rozo | date_rozo |
+----+-------------------+-----------------+-----------------+----------------+--------------------+-------------------+----------+----------+
| 1 | 0.3.13-1          | 2011-04-10      | 0.3.13-1        | 2011-04-10     | 0.3.13-1           | 2011-04-10        | 0.0.1-1  | 2011-04-10 |
+----+-------------------+-----------------+-----------------+----------------+--------------------+-------------------+----------+----------+
```

k) Description de la table CLOUD

La table permet d'informer l'utilisateur sur son environnement Rozo constitué d'un ou plusieurs clusters. Un cluster comporte un serveur de méta-données ainsi que plusieurs serveurs de stockage dont le nombre est fonction de l'espace de stockage désiré. La table « *Cloud* » dispose d'une contrainte référentielle obligatoire avec la table « *account* ».

Table CLOUD						
Attribut	**Type de données**	**Taille maximum**	**Clé primaire**	**Clé étrangère**	**Obligatoire**	**Description**
cloud_id	Smallint		Auto_increment		Oui	Identifiant unique d'un n-uplet
cloud_ref	Char	30			Oui	Référence du Cloud
cloud_description	Char	200			Non	Description
cloud_date	Char	50			Oui	Date de création
cloud_size	Char	50			Non	Volume du Cloud storage

Table CLOUD (suite)						
Attribut	**Type de données**	**Taille maximum**	**Clé primaire**	**Clé étrangère**	**Obligatoire**	**Description**
cloud_number_storage	Char	30			Non	Nombre de serveurs de stockage
cloud_number_cluster_storage	Int	30			Non	Nombre d'environnement Rozo
account	Smallint			Oui	Oui	Contrainte référentielle

Une fois cette table construite, on peut effectuer une projection sur un jeu de données.

```
+----+---------+-----------------+----------+----------+-----------------+-------------------------+----------+
| id | cloud_ref | cloud_description | cloud_date | cloud_size | cloud_number_storage | cloud_number_cluster_storage | account_id |
+----+---------+-----------------+----------+----------+-----------------+-------------------------+----------+
| 7 | mycloud   | Information cloud storage | 2011-10-21 |       1 |               4 |                       1 |       2 |
+----+---------+-----------------+----------+----------+-----------------+-------------------------+----------+
```

l) Description de la table EXPORT

Cette table comporte toutes les informations des serveurs de méta-données.

Elle comporte plusieurs contraintes référentielles obligatoires avec les tables « *instance* », « *cloud* », « *rozo* » et « *account* ». En effet, un serveur de méta-données est constitué d'une version de l'application « *rozofs-exportd* ». De plus, le serveur de méta-données est nécessairement une instance qui est rattachée à un *Cloud*. L'inverse n'est pas vrai puisqu'une instance peut être un serveur de stockage ou un autre serveur. Tandis qu'un *Cloud* peut être constitué de plusieurs serveurs de méta-données s'il y a plusieurs clusters. L'attribut « *export_rozofs* » indique si le serveur de méta-données et les serveurs de stockages sont opérationnels (« *build* »).

Table EXPORT						
Attribut	**Type de données**	**Taille maximum**	**Clé primaire**	**Clé étrangère**	**Obligatoire**	**Description**
export_id	Smallint		Auto_increment		Oui	Identifiant unique d'un n-uplet
export_ref	Char	30			Oui	Référence du serveur de méta-données
export_description	Char	200			Non	Description
export_size	Int	30			Non	Taille du disque
export_rozofs	Char	30			Non	Etat de configuration de Rozo (build ou None)
export_status	Char	30			Non	Etat du serveur (running, stopped)
export_private_ip	Char	120			Non	Adresse IP privée du serveur
export_metadata	Char	50			Non	Emplacement des méta-données
instance	Smallint			Oui	Oui	Contrainte référentielle
cloud	Smallint			Oui	Oui	Contrainte référentielle
rozo	Smallint			Oui	Oui	Contrainte référentielle
account	Smallint			Oui	Oui	Contrainte référentielle

Une fois cette table construite, on peut effectuer une projection sur un jeu de données.

```
+----+---------+----------+----------+----------+--------------+---------------+---------------+-------------+-------+----------+
| id | export_ref | export_size | export_rozofs | export_status | export_private_ip | export_metadata | export_instance_id | export_cloud_id | rozo_id |
account_id |
+----+---------+----------+----------+----------+--------------+---------------+---------------+-------------+-------+----------+
| 7 | i-225b4142 |        1 | ok       | running  | 10.208.251.12 | /srv/rozofs/export7 |            27 |           7 |     1 |        2 |
+----+---------+----------+----------+----------+--------------+---------------+---------------+-------------+-------+----------+
```

m) Description de la table STORAGE

Cette table récupère les informations de chaque serveur de stockage.

Elle comporte plusieurs contraintes référentielles obligatoires avec les tables « *instance* », « *cloud* », « *rozo* » et « *account* » pour les mêmes raisons que la table précédente.

Elle a également une contrainte référentielle obligatoire avec la table « *export* » car un serveur de stockage est obligatoirement associé à un serveur de méta-données. Tandis qu'un serveur de méta-données dispose de plusieurs serveurs de stockage.

Au niveau information, l'attribut « *storage_rozofs* » indique si le volume *EBS* qui est rattaché au serveur est opérationnel (« *ready* »). Pour cela, le volume doit être formaté puis monté à un système de fichier dans lequel figure un répertoire contenant les projections. L'attribut « *storage_dir* » précise l'emplacement du répertoire en question.

Table STORAGE						
Attribut	**Type de données**	**Taille maximum**	**Clé primaire**	**Clé étrangère**	**Obligatoire**	**Description**
storage_id	Smallint		Auto_increment		Oui	Identifiant unique d'un n-uplet
storage_ref	Char	30			Oui	Référence du serveur de stockage
storage_description	Char	200			Non	Description
storage_volume_ref	Char	50				Référence du volume EBS
storage_size	Int	30			Non	Taille du volume EBS
storage_rozofs	Char	30			Non	Etat du volume EBS (ready, not ready)
storage_status	Char	30			Non	Etat du serveur (running, stopped)
storage_private_ip	Char	120			Non	Adresse IP privée du serveur
storage_dir	Char	50				Emplacement des projections
instance	Smallint			Oui	Non	Contrainte référentielle
cloud	Smallint			Oui	Oui	Contrainte référentielle
export	Smallint			Oui	Oui	Contrainte référentielle
rozo	Smallint			Oui	Oui	Contrainte référentielle
account	Smallint			Oui	Oui	Contrainte référentielle

Une fois cette table construite, on peut effectuer une projection sur un jeu de données.

```
+----+----------+---------------+----------+---------------+------------+------------+-------------+--------+-------+---------+
| id | storage_ref | storage_volume_ref | storage_size | storage_private_ip | storage_status | storage_rozofs | storage_dir | storage_cloud_id | export_id | rozo_id | account_id |
+----+----------+---------------+----------+---------------+------------+------------+-------------+--------+-------+---------+
| 22 | i-ea58428a | vol-675a580d | 1 | 10.202.71.199 | running | 1 | /mnt/rozo/essai | 7 | 7 | 1 | 2 |
+----+----------+---------------+----------+---------------+------------+------------+-------------+--------+-------+---------+
```

Annexe B

```
from django.db import models
from django.contrib.auth.models import User

class ACCOUNT(models.Model):
    account_number = models.BigIntegerField()
    key_id = models.CharField(max_length=50)
    secret_key = models.CharField(max_length=50)
    date_account = models.DateField()
    user = models.ForeignKey(User)
    #def __unicode__(self):
        #return u'%s' % self.id
    def __unicode__(self):
        return '%s %s %s %s' % (self.account_number, self.key_id, self.secret_key, self.date_account)
    #def __unicode__(self):
        #return u'%s %s2 %s3' % (
                                        #self.id,
                                        #self.account_number,
                                        #self.key_id,
                                        #self.secret_key)

class IMAGE(models.Model):
    image_ref = models.CharField(max_length=50)
    image_architecture = models.CharField(max_length=50, blank=True, null=True)
    image_block_device_mapping = models.CharField(max_length=200, blank=True, null=True)
    image_description = models.CharField(max_length=200, blank=True, null=True)
    image_hypervisor = models.CharField(max_length=30, blank=True, null=True)
    image_instance_lifecycle = models.CharField(max_length=30, blank=True, null=True)
    image_is_public = models.CharField(max_length=30, blank=True, null=True)
    image_kernel_id = models.CharField(max_length=30, blank=True, null=True)
    image_location = models.CharField(max_length=120, blank=True, null=True)
    image_name = models.CharField(max_length=100, blank=True, null=True)
    image_owner_alias = models.CharField(max_length=30, blank=True, null=True)
    image_owner_id = models.CharField(max_length=30, blank=True, null=True)
    image_platform = models.CharField(max_length=30, blank=True, null=True)
    image_ramdisk_id = models.CharField(max_length=30, blank=True, null=True)
    image_root_device_name = models.CharField(max_length=30, blank=True, null=True)
    image_root_device_type = models.CharField(max_length=30, blank=True, null=True)
    image_state = models.CharField(max_length=30, blank=True, null=True)
    image_type = models.CharField(max_length=50, blank=True, null=True)
    image_virtualization_type = models.CharField(max_length=50, blank=True, null=True)
    image_size = models.CharField(max_length=30, blank=True, null=True)
    image_key = models.CharField(max_length=50, blank=True, null=True)
    image_value = models.CharField(max_length=50, blank=True, null=True)
    image_snapshot_id = models.CharField(max_length=50, blank=True, null=True)
    date_image = models.DateField()
    account = models.ForeignKey(ACCOUNT)
    def __unicode__(self):
        return u'%s %s' % (self.id, self.date_image)
```

```
class KEYPAIR(models.Model):
    keypair_name = models.CharField(max_length=30)
    date_keypair = models.DateField()
    locate_keypair = models.CharField(max_length=100, blank=True, null=True)
    account = models.ForeignKey(ACCOUNT)
    def __unicode__(self):
        return u'%s %s %s' % (self.keypair_name, self.date_keypair, self.locate_keypair)

class SECURITYGROUP(models.Model):
    group_name = models.CharField(max_length=50)
    group_description = models.CharField(max_length=200,blank=True, null=True)
    date_group = models.DateField()
    account = models.ForeignKey(ACCOUNT)
    def __unicode__(self):
        return u'%s %s %s' % (self.id, self.date_group, self.group_name)

class SECURITYRULE(models.Model):
    protocol = models.CharField(max_length=10, blank=True, null=True)
    from_range_port = models.IntegerField(max_length=10, blank=True, null=True)
    to_range_port = models.IntegerField(max_length=10, blank=True, null=True)
    cidr = models.CharField(max_length=30, blank=True, null=True)
    date_rule = models.DateField()
    securitygroup = models.ForeignKey(SECURITYGROUP)
    def __unicode__(self):
        return u'%s' % self.id
    #def __unicode__(self):
     #   return '%s %s' % (self.region_name, self.zone_name)

class INSTANCE(models.Model):
    instance_ref = models.CharField(max_length=50)
    instance_ami_launch_index = models.IntegerField(max_length=30, blank=True, null=True)
    instance_architecture = models.CharField(max_length=50, blank=True, null=True)
    instance_block_device_mapping = models.CharField(max_length=200, blank=True, null=True)
    instance_connection = models.CharField(max_length=50, blank=True, null=True)
    instance_dns_name = models.CharField(max_length=50, blank=True, null=True)
    instance_hypervisor = models.CharField(max_length=50, blank=True, null=True)
    instance_image_id = models.CharField(max_length=50, blank=True, null=True)
    instance_ip_address = models.CharField(max_length=50, blank=True, null=True)
    instance_kernel = models.CharField(max_length=50, blank=True, null=True)
    instance_key_name = models.CharField(max_length=30, blank=True, null=True)
    instance_launch_time = models.CharField(max_length=50, blank=True, null=True)
    instance_monitored = models.CharField(max_length=50, blank=True, null=True)
    instance_placement = models.CharField(max_length=50, blank=True, null=True)
    instance_private_dns_name = models.CharField(max_length=120, blank=True, null=True)
    instance_private_ip_address = models.CharField(max_length=120, blank=True, null=True)
    instance_public_dns_name = models.CharField(max_length=120, blank=True, null=True)
    instance_region = models.CharField(max_length=50, blank=True, null=True)
    instance_root_device_name = models.CharField(max_length=50, blank=True, null=True)
    instance_root_device_type = models.CharField(max_length=50, blank=True, null=True)
    instance_state = models.CharField(max_length=50, blank=True, null=True)
    instance_state_code = models.IntegerField(max_length=10, blank=True, null=True)
    instance_type = models.CharField(max_length=50, blank=True, null=True)
    instance_virtualizationType = models.CharField(max_length=50, blank=True, null=True)
    instance_volume_ref = models.CharField(max_length=50, blank=True, null=True)
```

```
    instance_volume_status = models.CharField(max_length=50, blank=True, null=True)
    instance_key = models.CharField(max_length=50, blank=True, null=True)
    instance_value = models.CharField(max_length=50, blank=True, null=True)
    date_instance = models.DateField()
    account = models.ForeignKey(ACCOUNT)
    image = models.ForeignKey(IMAGE)
    keypair = models.ForeignKey(KEYPAIR, blank=True, null=True, on_delete=models.SET_NULL)
    securitygroup = models.ForeignKey(SECURITYGROUP, blank=True, null=True, on_delete=models.SET_NULL)
    ip = models.ForeignKey("ELASTICIP", blank=True, null=True, on_delete=models.SET_NULL)
    instance_storage = models.ForeignKey("STORAGE", blank=True, null=True, on_delete=models.SET_NULL)
    instance_export = models.ForeignKey("EXPORT", blank=True, null=True, on_delete=models.SET_NULL)
    def __unicode__(self):
        return u'%s %s' % (self.instance_ref, self.date_instance)
    class Meta:    #def __unicode__(self):
     #   return '%s %s' % (self.region_name, self.zone_name)
        ordering = ['instance_ref']

class ELASTICIP(models.Model):
    public_ip = models.CharField(max_length=30)
    date_ip = models.DateField()
    instance_name = models.CharField(max_length=50, blank=True, null=True)
    account = models.ForeignKey(ACCOUNT)
    host = models.ForeignKey("INSTANCE",blank=True, null=True, on_delete=models.SET_NULL)
    def __unicode__(self):
        return u'%s %s' % (self.id, self.date_ip)
    class Meta:
        ordering = ['public_ip']

class VOLUME(models.Model):
    volume_ref = models.CharField(max_length=30)
    volume_attach_data_device = models.CharField(max_length=50, blank=True, null=True)
    volume_attach_data_status = models.CharField(max_length=50, blank=True, null=True)
    volume_attach_data_time = models.CharField(max_length=50, blank=True, null=True)
    volume_attach_data_instance = models.CharField(max_length=50, blank=True, null=True)
    volume_create_time = models.CharField(max_length=50, blank=True, null=True)
    volume_snapshot_id = models.CharField(max_length=50, blank=True, null=True)
    volume_size = models.IntegerField(max_length=30, blank=True, null=True)
    volume_status = models.CharField(max_length=50, blank=True, null=True)
    volume_zone = models.CharField(max_length=50, blank=True, null=True)
    date_volume = models.DateField()
    account = models.ForeignKey(ACCOUNT)
    instance = models.ForeignKey(INSTANCE,blank=True, null=True)
    def __unicode__(self):
        return u'%s %s' % (self.id, self.date_volume)
    class Meta:
        ordering = ['volume_ref']
```

```
class SNAPSHOT(models.Model):
    snapshot_ref = models.CharField(max_length=30)
    snapshot_description = models.CharField(max_length=200, blank=True, null=True)
    snapshot_progress = models.CharField(max_length=50, blank=True, null=True)
    snapshot_status = models.CharField(max_length=50, blank=True, null=True)
    snapshot_start_time = models.CharField(max_length=50, blank=True, null=True)
    snapshot_volume_id = models.CharField(max_length=30, blank=True, null=True)
    snapshot_volume_size = models.IntegerField(max_length=30, blank=True, null=True)
    date_snapshot = models.DateField()
    account = models.ForeignKey(ACCOUNT)
    def __unicode__(self):
        return u'%s %s' % (self.id, self.date_snapshot)
    class Meta:
        ordering = ['snapshot_ref']

class ROZO(models.Model):
    Version_rozofs_storaged = models.CharField(max_length=50, blank=True, null=True)
    date_rozofs_storaged = models.DateField()
    Version_rozofs_exportd = models.CharField(max_length=50, blank=True, null=True)
    date_rozofs_exportd = models.DateField()
    Version_rozofs_rozofsmount = models.CharField(max_length=50, blank=True, null=True)
    date_rozofs_rozofsmount = models.DateField()
    Version_rozo = models.CharField(max_length=50, blank=True, null=True)
    date_rozo = models.DateField()

class CLOUD(models.Model):
    cloud_ref = models.CharField(max_length=30)
    cloud_description = models.CharField(max_length=200, blank=True, null=True)
    cloud_date = models.DateField()
    cloud_size = models.IntegerField(max_length=30, blank=True, null=True)
    cloud_number_storage = models.IntegerField(max_length=30, blank=True, null=True)
    cloud_number_cluster_storage = models.IntegerField(max_length=30, blank=True, null=True)
    account = models.ForeignKey(ACCOUNT)


class EXPORT(models.Model):
    export_ref = models.CharField(max_length=30)
    export_description = models.CharField(max_length=200, blank=True, null=True)
    export_size = models.IntegerField(max_length=30, blank=True, null=True)
    export_rozofs = models.CharField(max_length=30, blank=True, null=True)
    export_status = models.CharField(max_length=30, blank=True, null=True)
    export_private_ip = models.CharField(max_length=120, blank=True, null=True)
    export_metadata = models.CharField(max_length=50, blank=True, null=True)
    export_instance = models.ForeignKey(INSTANCE,blank=True, null=True)
    export_cloud = models.ForeignKey(CLOUD,blank=True, null=True)
    account = models.ForeignKey(ACCOUNT)
    rozo = models.ForeignKey(ROZO,blank=True, null=True)
    def __unicode__(self):
        return u'%s' % (self.id)
```

```
class STORAGE(models.Model):
    storage_ref = models.CharField(max_length=30)
    storage_description = models.CharField(max_length=200, blank=True, null=True)
    storage_volume_ref = models.CharField(max_length=50, blank=True, null=True)
    storage_size = models.IntegerField(max_length=30, blank=True, null=True)
    storage_private_ip = models.CharField(max_length=120, blank=True, null=True)
    storage_status = models.CharField(max_length=30, blank=True, null=True)
            storage_dir = models.CharField(max_length=50, blank=True, null=True)
    storage_cloud = models.ForeignKey(CLOUD,blank=True, null=True)
    storage_rozofs = models.IntegerField(max_length=30, blank=True, null=True)
    export = models.ForeignKey(EXPORT,blank=True, null=True)
    account = models.ForeignKey(ACCOUNT)
    rozo = models.ForeignKey(ROZO,blank=True, null=True)
    def __unicode__(self):
        return u'%s' % (self.id)
```

Annexe C

Paramètres réseau de l'interface:

Hostname: **ip-xx-xx-xx-xx**
Elastic IP: xx.xx.xx.xx
Public DNS: ec2-xx-xx-xx-xx.compute-1.amazonaws.com
Private IP: **xx.xx.xx.xx**
Key pair: rozofstest.pem
Security Group: security_admin
Rules: permit 23, 80 and 443

Attention, ces paramètres changent lorsque l'instance redémarre!!!

1-Installation Python :

```
apt-get install python
```

2-Installation apache et mysql-server-5.1 et phpmyadmin:

```
apt-get install apache2 libapache2-mod-python
apt-get install mysql-server python-mysqldb
apt-get install phpmyadmin
```

3-Installation de Django:

On récupère la version django 1.3.1 à l'adresse suivante : https://www.djangoproject.com/download/

```
python setup.py install
```

4-Installation API "Boto" (APIs AWS d'Amazon)

```
wget http://boto.googlecode.com/files/boto-2.0.tar.gz
python setup.py install
```

5-Installation de l'application rozo

Avec le dépôt apt fizians on ajoute la ligne suivante dans le fichier « /etc/apt/sources.list » en spécifiant l'adresse DNS privé et non public :

```
deb http://ip-xx-xx-xx-xx.ec2.internal/debian experimental main contrib non-free
apt-get install rozo
```

6-Création d'une table « Cloud » et d'un utilisateur « fizians » dans mysql

```
CREATE DATABASE cloud CHARACTER SET utf8 COLLATE utf8_general_ci;
CREATE USER'fizians'@'localhost'IDENTIFIED BY'xxxxx';
GRANT ALL ON cloud.* TO 'fizians'@' localhost';
flush privileges;
```

7-Création de l'application Django "cloud"

```
cd /var/www
django-admin.py startproject cloud
```

8-Copie des fichiers et répertoires suivants dans le répertoire "/var/www/cloud":

```
amazon
authentification
django.wsgi
rozofs
settings.py
static_media
templates
urls.py
welcome
```

9-Création du site et de l'interface admin de Django

```
python manage.py validate
python manage.py syncdb
```

10-Création d'un lien symbolique avec le répertoire admin django:

```
ln -s /usr/local/lib/python2.6/dist-packages/django/contrib/admin/media/
chmod –R 755 /var/www/cloud
```

11-Mise en place de SSL et création d'un certificat auto-signé

```
apt-get install openssl
cd etc/apache2/mods-available
a2enmod ssl
/etc/init.d/apache2 reload
openssl req -x509 -nodes -days 365 -newkey rsa:1024 -out /etc/apache2/server.crt -keyout
/etc/apache2/server.key
chmod 440 /etc/apache2/server.key
```

12-Django avec Apache

/etc/apache2/sites-available/cloud

```
NameVirtualHost xx.xx.xx.xx:443
<virtualhost xx.xx.xx.xx:443>
DocumentRoot /var/www/cloud
ServerName ip-xx-xx-xx-xx
SSLEngine On
SSLCertificateFile /etc/apache2/server.crt
SSLCertificateKeyFile /etc/apache2/server.key

<Location "/">
SetHandler python-program
#SetHandler mod_python
PythonHandler django.core.handlers.modpython
SetEnv DJANGO_SETTINGS_MODULE cloud.settings
PythonOption django.root /cloud
PythonPath "['/var/www/', '/var/www/cloud', '/var/www/cloud/amazon', '/var/www/cloud/authentification']
+ sys.path"
PythonDebug On
</Location>
#WSGIScriptAlias / /var/www/cloud/django.wsgi

Alias /media "/var/www/cloud/media"
<location "/media/">
SetHandler None
</location>
<locationmatch "\.(jpg|gif|png)$">
SetHandler None
</locationmatch>

</virtualhost>
```

/etc/apache2/ports.conf

```
#NameVirtualHost *:80
#Listen 80

<IfModule mod_ssl.c>
    # If you add NameVirtualHost *:443 here, you will also have to change
    # the VirtualHost statement in /etc/apache2/sites-available/default-ssl
    # to <VirtualHost *:443>
    # Server Name Indication for SSL named virtual hosts is currently not
    # supported by MSIE on Windows XP.
    Listen 443
</IfModule>

<IfModule mod_gnutls.c>
    Listen 443
</IfModule>
```

Activation de l'interface Web :

```
a2ensite cloud
/etc/init.d/apache2 stop
/etc/init.d/apache2 start
```

Créer le répertoire contenant les clés SSH

```
mkdir /home/users/.ssh
chmod –R 777 /home/users/.ssh
```

Annexe D

```
from boto.ec2.connection import EC2Connection
import sys
import os
import time

key = 'XXXXXXXXXXX'
secret ='XXXXXXXXXXXXXXXX'

def connection():
    print '--- running EC2Connection tests ---'
    c = EC2Connection(key, secret)
    return c

def add_group(c, group_name):
    group_desc = 'This is a security group created by Fizians'
    c.create_security_group(group_name, group_desc)

def add_perm(c, group_name, port, numfrom, numto, cidr):
    rs = c.get_all_security_groups([group_name])
    group = rs[0]
    perm = group.authorize (port, numfrom, numto, cidr)

def instance (c, image_id, instance_type, key, group):
    reservation = c.run_instances(image_id, instance_type = str(instance_type), key_name = str(key), security_groups = [group])
    instance = reservation.instances[0]
    while instance.state != 'running':
        time.sleep(5)
        instance.update()
    instance_id = str(instance.id)
    print "Instance id: "+instance_id
    print "IP Private: "+instance.private_ip_address
    print "Key Pairs: "+instance.key_name
    return instance_id


def del_instance(c, instance_id):
    c.terminate_instances(instance_id)
    print '---  Fin Test EC2Connection  ---'

auth = connection()
group_security = add_group (auth, 'test')
add_perm (auth, 'test', 'tcp','22','22', '0.0.0.0/0')
instance_id = instance (auth, 'ami-0ce41865', 't1.micro', 'essai', 'test')
del_instance (auth, instance_id)
```

Bibliographie

[Barr 2010] J. Barr, 2010, Host Your Web Site in the Cloud: Amazon Web Services Made Easy, SitePoint Pty, USA, 354 pages

[Clever Age 2006] Clever Age, novembre 2006, livre blanc : étude comparative des principaux frameworks Ajax, 35 pages

[Crane et al. 2006] D. Crane, E. Pascarello, Ajax in Action, Manning Publication, Greenwich, 680 pages

[David et al. 2011] Sylvain David, Boris Lucas, juin 2011, Rozo, manuel d'installation, Fizians, 23 pages

[Erl 2007] T. Erl, Juillet 2007, SOA : Principles of Service Design, Prenctice Hall, USA, 608 Pages

[Furht 2010] B. Furht et A. Escalante, Septembre 2010, Handbook of cloud computing, Springer-Verlag, New York, 634 pages

[Ghemawat 2003] S. Ghemawat et al., 5 Décembre 2003, The Google File System, ACM, New-york, 15 pages

[Guédon 2009] Jean-Pierre Guédon, janvier 2009, The Mojette Transform : theory and application, ISTE-WILEY, Europe, 274 pages

[Holdener 2008] A. T. Holdener, janvier 2008, Ajax : The Definitive Guide, O'Reilly Media, New-york, 992 pages

[Holovaty et al. 2009] Adrian Holovaty, Jacob Kaplan Moss, 2009, The Definitive Guide to Django, APress, New York, 499 pages

[Kadima et al. 2003] H. Kadima, V. Montfort, mars 2003, Les Web services – Techniques, démarches et outils, Dunod, Paris, 432 pages

[Lutz 2009] M. Lutz, septembre 2009, Learning Python, O'Reilly Media, USA, 1216 pages

[Plasse 2006] M. Plasse, septembre 2006, Développez en Ajax, Eyrolles, Paris, 314 pages

[Plouin 2009] G. Plouin, mars 2009, Cloud Computing et SaaS : Une rupture décisive pour l'informatique d'entreprise, Dunod, Paris, 249 pages

[Plouin et al. 2008] G. Plouin**,** X. Fournier-Morel et al**.**, Mars 2008, SOA, le guide de l'architecture du SI, Dunod, Paris, 350 pages

[Rosenberg 2011] J. Rosenberg, A. Mateos, 2011, The Cloud as your service,Manning Publications., Greenwich, 223 pages

[Silberschatz 2001] A**.** Silberschatz, 2001, Principes appliqués des systèmes d'exploitation, Vuibert, Paris, 832 pages

[Velte 2010] A. Velte, R. Elsenpeter, 2010, Cloud Computing: A practical approach, Mc Graw Hill, USA, 352 pages

Webographie

[Amazon 2010] Amazon, 2010, Amazon Web Services – caractéristiques
http://aws.amazon.com/fr/ec2/#features

[Atelier 2008] Atelier Informatique, septembre 2008, Web 1.0 – Web 2.0 – Web 3.0
http://www.atelier-informatique.org/Internet/evolution-Web -10-Web -20-Web -30/358/

[Barathon 2010] D. Barathon, décembre 2010, Le marché du SaaS dans le monde en 2010
http://www.lemondeinformatique.fr/actualites/lire-article-32427.html

[Baron 2011] M. Baron, février 2011, Introduction aux architectures Orientées Services (SOA)
http://mbaron.developpez.com/soa/intro/

[Benkemoun 2009] A. benkemoun, 2009, La virtualisation
http://www.antoinebenkemoun.fr/virtualisation/

[Blanchard 2009] B. Blanchard, Juin 2009, Understanding the Cloud
http://www.devrevival.com/2009/06/understanding-cloud.html

[Borer et al. 2007] R. Borer, M. Savary, Mai 2007, Django : Un python sur la toile
http://www.borer.name/files/eivd/**django**

[Carton 2005] O. Carton, août 2005, Architecture Modèle, Vue, Contrôleur
http://www.liafa.jussieu.fr/~carton/Enseignement/InterfacesGraphiques/MasterInfo/Cours/Swing/mvc.html

[Chagnon 2009] G. Chagnon, septembre 2009, Qu'est ce que le DOM ?
http://www.licence.elec.upmc.fr/S_tec/coursEnLigne/dhtml3/introdom.html#koikce

[Chong et al. 2006] F. Chong, G. Carraro, avril 2006, Architecture Strategies for Catching The Long Tail
http://msdn.microsoft.com/en-us/library/aa479069%28printer%29.aspx

[Cochoy 2010] J. Cochoy, 2010, Introduction to Linux Virtual File System , Vuibert
http://zenol.fr/site/2009/12/28/introduction-to-linux-virtual-file-system-ch1-ch2/

[Colan 2004] M. Colan, avril 2004, Service-Oriented Architecture expands the vision of Web Services
http://www.ibm.com/developerworks/library/ws-soaintro.html

[Django 2011] Django Software Foundation, 2011, Django v1.3
http://www.djangoproject.com

[DSF 2011] Django Software Foundation, 2011, Django, the Web framework for perfectionists with deadlines,
https://www.djangoproject.com/

[Dojo 2011] Dojo, 2011, Dojo v1.6
http://dojotoolkit.org/

[Eisler 2008] M. Eisler, 2008, Accélérer l'accès aux données partagées pour les clusters de calcul
http://www.netapp.com/fr/communities/tech-ontap/pnfs-0109-fr.html

[EMC 2010] EMC, 2010, Le Cloud Privé et ses avantages métiers, livre blanc
http://france.emc.com/campaign/global/private-cloud/ppc-index.htm#tab3

[Fizians 2011] Fizians, 2011, Fizians: Solid Storage for Big Data
http://polytech.univ-nantes.fr/fizians

[Garnaat 2010] Mitch Garnaat 2010, Boto: A Python interface to Amazon Web Services
http://boto.cloudhackers.com/en/latest/index.html

[Garret 2005] J.J. Garret, février 2005, Ajax a new approach to Web applications
http://www.adaptivepath.com/ideas/e000385
[Gilli 2005] J. Gilli, janvier 2005, Les servlets
http://java.developpez.com/cours/servlets/
[Gonzalez 2008] D. Gonzalez, 2008, Généralités sur les langages informatiques
http://www.grappa.univ-lille3.fr/polys/reseaux-2004/reseaux011.html
[Google 2011] Google, 2011, What Is Google App Engine?
http://code.google.com/intl/fr/appengine/docs/whatisgoogleappengine.html
[IBM 2002] IBM, octobre 2002, Best pactices for Web services,
http://www.ibm.com/developerworks/Web services/library/ws-best1/?dwzone=Web services#figure1
[IDG 2009] IDG News Service, Mai 2009, 7 entreprises du « Cloud » à suivre de près
http://www.reseaux-telecoms.net/actualites/imprimer-7-entreprises-du--cloud--a-suivre-de-pres-20118.html
[Jones 2010] T. Jones, novembre 2010, Network file systems and Linux, IBM
http://www.ibm.com/developerworks/linux/library/l-network-filesystems/?ca=drs-
[Journal du net 2011] Journal du net, mars 2011, SOA : définition, conseils, retours d'expérience
http://www.journaldunet.com/solutions/dsi/soa/
[JQuery 2010] JQuery, 2010, JQuery v1.5.2
http://jquery.com/
[Képéklian 2000] G. Képéklian, mars 2000, L'historique des langages de l'Internet, de SGML à XHTML
http://xmlfr.org/documentations/articles/000321-0001
[Korolczuk 2008] M. Korolczuk, décembre 2008, Le php : langage interprété ou langage compilé ?
http://unearaigneeauplafond.fr/php-langage-interprete-compile
[Lambel 2007] F. Lambel, Novembre 2007, IBM lance une grille sous le nom de Blue Cloud
http://www.lemondeinformatique.fr/actualites/lire-ibm-lance-une-grille-sous-le-nom-de-blue-cloud-24591.html
[Laské 2011] L. Laské, mars 2011, Voyage au coeur du nuage : SaaS
http://www.7avoir.net/article-voyage-au-coeur-du-nuage-3-3-saas-69006998.html
[Lebre 2002] adrien lebre, juin 2002, Composition de service de données et de méta-données dans un système de fichiers distribué
http://www-id.imag.fr/~lebre/DEA/NFSp/ReportDEA.pdf
[Lhérault 2011], Y. Lhérault, SOA: Service Oriented Architecture
http://www.guideinformatique.com/fiche-soa_service_oriented_architecture-821.htm
[Makhmara et al. 2002], H. Makhmara, F. Nino, J.L. Boichard, juin 2002, Le Web et ses interfaces
http://medias.obs-mip.fr/www/Activite/Formation/Informatique/sgbd/cours003.html
[Marcombe 2010] M. Marcombe, mai 2010, LA vague SaaS : me nouvel atout pour les éditeurs de logiciel
http://www.techniques-ingenieur.fr/actualite/informatique-electronique-telecoms-thematique_193/la-vague-saas-le-nouvel-atout-pour-les-editeurs-de-logiciel-article_7207/
[Mell et al. 2009] P.Mell, T.Grance, 10 Juillet 2009, The NIST Definition of Cloud Computing
http://www.nist.gov/itl/cloud/index.cfm
[Mohamed 2009] A. Mohamed, mars 2009, a history of cloud
http://www.computerweekly.com/Articles/2009/06/10/235429/A-history-of-cloud-computing.htm
[MooseFS 2011] *MooseFS, 2011*
http://www.moosefs.org/

MooTools 2011] MooTools, 2011, MooTools v 1.3.1
http://mootools.net/download
[Msdn 2010] Msdn, 2010, About Windows Azure
http://msdn.microsoft.com/library/dd179442.aspx
[Munz 2001] S. Munz, 2001, CGI: Common Gateway Interface
http://actuel.fr.selfhtml.org/archives/docu/7.0/tbbc.htm
[Munz 2002] S. Munz, 2002, ActiveX et HTML
http://fr.selfhtml.org/introduction/technologies/activex.htm
[Neoxia et al. 2011] Neoxia, French AWS User Group, 2011, Architecture pour le Cloud:
http://www.aws-ug.fr/wp-content/uploads/Bonnes-pratiques-AWS.pdf
[O'Reilly 2009] T. O'Reilly, octobre 2009, what is Web 2.0?
http://oreilly.com/Web 2/archive/what-is-Web -20.html
[Ozitem 2007] Ozitem, 2007, Virtualisation
http://www.ozitem.com/virtualisation-g0-solutions_virtu.html
[Pac 2010] Pierre Audoin Consultant, Février 2010, 2010, l'année du Cloud Computing
https://www.pac-online.com/pac/pac/live/pac_france/global/presse/communique_de_presse/index.html?lenya.usecase=show-rapport&document=pac_sitsi_reports/press_release/fr_pr_100222_cloud&xsl=press_release
[Pac 2010] Pierre Audoin Consultant, 2010, Livre blanc : Le Cloud Computing en France
http://www.google.fr/url?sa=t&source=Web&cd=1&ved=0CBgQFjAA&url=http%3A%2F%2Ffrance.emc.com%2Fcollateral%2Fhardware%2Fwhite-papers%2Fcloud-etude-pac.pdf&rct=j&q=march%C3%A9%20du%20cloud%20en%20france%20%2B%20emc&ei=yI1iTbHVN86u8QPnkJDxCA&usg=AFQjCNFxJGXWch5gHmI7HDXNTvl75nC3Ww&cad=rja
[Pnfs 2008] Pnfs, 2008, what is pnfs?
http://www.pnfs.com/
[Prototype 2010] Prototype, novembre 2010, Prototype 1.7
http://www.prototypejs.org/
[Pylon 2010] Pylon, 2010, v1.0
http://pylonshq.com/
[Robie 1998] J. Robie, 1998, Qu'est ce que le Modèle Objet de Document ?
http://xmlfr.org/w3c/TR/REC-DOM-Level-1/introduction.html
[Rossi 2002] F. Rossi, Systèmes répartis : les Remote Procedure Calls
http://APIacoa.org/publications/teaching/distributed/rmi.pdf
[Rousseau 2011] M. Rousseau, mars 2011, Introduction à Java
http://subaru2.univ-lemans.fr/enseignements/physique/02/java/java5.html
[Sahnine 2009] *K. Sahnine, octobre 2009, Inovia conseil, Hadoop*
http://blog.inovia-conseil.fr/
[Schuller 2008] S. Schuller, Décembre 2008, Demystifying the Cloud,
http://www.saasblogs.com/2008/12/01/demystifying-the-cloud-where-do-saas-paas-and-other-acronyms-fit-in/
[Serries 2011] G. Serries, 3 Janvier 2011, Le Cloud Computing, l'informatique de demain ?, Journal du net
http://www.journaldunet.com/solutions/systemes-reseaux/analyse/le-cloud-computing-l-informatique-de-demain.shtml

[Spivack 2009] N Spivack, décembre 2009, the evolution of the Web: past, present, future
http://www.novaspivack.com
[Streicher 2008] M. Streicher, novembre 2008, Scale your file system with Parrallel NFS
http://www.ibm.com/developerworks/linux/library/l-pnfs/
[Syntec 2010] Syntec informatique, 2010, Le livre blanc du Cloud Computing
http://www.syntec-numerique.fr/actualites/liste-actualites/publication-du-livre-blanc-cloud-computing-de-syntec-informatique
[Traumat 2008] S. Traumat, juin 2008, Software as a Service
http://www.slideshare.net/straumat/prsentation-de-saas-jet-spn-du-26-juin-2008
[W3C 2002] W3C, juin 2002, Web Services Architectures
http://www.w3.org/2002/ws/arch/2/08/wd-wsa-arch-20020821.html#Web service
[Wikipédia 2010] Wikipédia, Décembre 2010, Posix,
http://fr.wikipedia.org/wiki/POSIX
[Wikipédia 2011] Wikipédia, avril 2011, Hypertext Transfer Protocol
http://fr.wikipedia.org/wiki/Hypertext_Transfer_Protocol
[Wikipédia 2011] Wikipédia, avril 2011, Langage de programmation
http://fr.wikipedia.org/wiki/Langage_de_programmation
[Wikipédia 2011] Wikipédia, avril 2011, Logiciel en tant que service
http://fr.wikipedia.org/wiki/Software_as_a_service
[Wikipédia 2011] Wikipédia, février 2011, Active Server Page
http://fr.wikipedia.org/wiki/Hypertext_Transfer_Protocol
[Wikipédia 2011] Wikipédia, 21 Février 2011, Cloud Computing,
http://fr.wikipedia.org/wiki/Cloud_computing
[Wikipédia 2011] Wikipédia, janvier 2011, Application composite
http://fr.wikipedia.org/wiki/Application_composite
[Wikipédia 2011] Wikipédia, mars 2011, Framework
http://fr.wikipedia.org/wiki/Framework
[Yahoo! 2011] Yahoo!, 2011, Yahoo User Interface Library v 3.3.0
http://developer.yahoo.com/yui/3/
[Zope 2010] Zope Corporation, 2010, Zope v2.11.4
http://www.zope.org
[3ie 2003] 3ie, 2003, Web Services
http://www.3ie.fr/nouvelles_technologies/fiches.php?techno_id=74

Liste des figures

Liste des tableaux

Résumé

Le « *Cloud Storage* » est une architecture Web qui met à disposition un espace de stockage évolutif et élastique en tant que service au client. Dans ce contexte, la société Fizians a élaborée un système de fichiers distribués « *Rozofs* » qui permet de mettre en œuvre un espace de stockage constitué d'un ensemble de machines virtuelles installées et configurées. Cependant, le processus de mise en œuvre n'est pas automatisé et il faudrait trouver une solution pour permettre à un utilisateur de créer son « *Cloud Storage* » avec une méthode conviviale et interactive.

Ainsi, l'objectif de ce mémoire consiste à réaliser une interface Web pour gérer une solution de « *Cloud Storage* » avec les services Web d'Amazon puis les applications « *Rozofs* » de Fizians.

Afin d'organiser l'infrastructure nécessaire pour préparer l'environnement de stockage, l'interface Web utilisera les APIs d'Amazon pour gérer des machines virtuelles constituées de volumes de données. Ces machines virtuelles sont préalablement configurées avec le système de fichiers distribués « *Rozofs* ». D'autre part, l'interface utilisera les APIs « *Rozo* » développées par Fizians pour automatiser la création de l'espace de données dont la volumétrie dépend du nombre de serveurs de stockage (*machines virtuelles qui disposent d'un volume de données de taille prédéfinie*).

Mots clés: *Cloud Storage, systèmes de fichiers distribués, stockage, machines virtuelles, interface Web, services Web, APIs, serveurs de stockage.*

Abstract

« *Cloud Storage* » is a web architecture which gives an elastic and scalable storage space as a service for customer. In this context, the company Fizians has designed a distributed file system « *RozoFS* » in order to implement a storage space with several installed and configured virtual machines.

The method of implementation is not yet automated and it would be necessary to find solution which will allow, for any user, to create « *Cloud Storage* » with a userfriendly and interactive method.

Thus, the purpose of this report is to develop a web interface in order to manage a « *Cloud Storage* » solution with Amazon web services and the « *RozoFS* » applications of Fizians.

In order to setup the environment of storage, the web interface will use to Amazon APIs to manage virtual machines with raw block devices attached to them. These machines are already configured with the distributed file system « *RozoFS* ». On the other hand, the interface will use Rozos' APIs developed by Fizians to automate the implementation of the storage space which depends on the number of storage servers (*virtual machines which have a predefined volume of data*).

Key words: *Cloud Storage, distributed file system, storage, virtual machines, web interface, web services, APIs, storage servers.*