



12 AVRIL 2019

TRAVAUX PRATIQUES SR2

COMPTE-RENDU

ALEXIS GUYOT – ALEXANDRE LABROSSE
UNIVERSITE DE BOURGOGNE
Licence 3 Informatique

TABLE DES MATIERES

I. INTRODUCTION	3
A. CONTEXTE	3
B. ARCHITECTURE FONCTIONNELLE	3
II. MISE EN MARCHÉ DU SERVEUR	4
A. LE SYSTÈME D'EXPLOITATION	4
B. PARTITIONS DE LA MÉMOIRE	5
III. ADRESSAGE ET ROUTAGE	7
A. ARCHITECTURE PHYSIQUE	7
B. CONFIGURATION DES ADRESSES IP	7
C. IMPLEMENTATION DU NAT	9
D. ROUTAGE	11
E. IMPLEMENTATION DU SERVICE DHCP	13
F. COMMANDES DU GESTIONNAIRE DE PACKAGES À CONNAÎTRE	16
IV. INSTALLATION D'UN DNS	18
A. COMMANDES ET FICHIERS UTILES AU DNS SOUS DEBIAN	18
B. INSTALLATION DU DNS	20
C. PROTOCOLE DE VÉRIFICATION	24
V. INSTALLATION DES FONCTIONNALITÉS	25
A. SERVEUR APACHE	25
B. BASES DE DONNÉES	29
C. PHP	33
D. SAUVEGARDE AUTOMATIQUE	35
E. AUTHENTIFICATION ET PARTAGE DE RESSOURCES	37
VI. FILTRAGE	45
A. MATRICE DE FILTRAGE	45
B. SCRIPT DE FILTRAGE	46
C. DÉMARRAGE AUTOMATIQUE DU SERVICE	48

VII. CONCLUSION	50
VIII. ANNEXES	51
A. /ETC/NETWORK/INTERFACES	51
B. /ETC/DHCP/DHCPD.CONF	51
C. /ETC/DEFAULT/ISC-DHCP-SERVER	52
D. /ETC/BIND/NAMED.CONF.OPTIONS	52
E. /ETC/BIND/NAMED.CONF.LOCAL	53
F. /ETC/BIND/DB.AGENCE.TOULOUSE	53
G. /ETC/BIND/DB.AGENCE.TOULOUSE.INV	54
H. /ETC/BIND/DB.INFORMATIQUE.AGENCE.TOULOUSE	54
I. /ETC/APACHE2/APACHE2.CONF	54
J. /ETC/APACHE2/SITES-AVAILABLE/INFORMATIQUE.AGENCE.TOULOUSE	55
K. /ETC/SUDOERS	55
L. /ETC/SCRIPTS/POSTGRESBACKUP.SH	55
M. /ETC/SCRIPTS/MYSQLBACKUP.SH	56
N. /ETC/SCRIPTS/RSYNC.SH	56
O. /ETC/CRONTAB	56
P. /ETC/LDAP/SLAPD.CONF	56
Q. /ETC/SAMBA/SMB.CONF	57
R. /ETC/NSSWITCH.CONF	58
S. /ETC/SMBLDAP-TOOLS/SMBLDAP.CONF	59
T. /ETC/SMBLDAP-TOOLS/SMBLDAP_BIND.CONF	59
U. /ETC/INIT.D/FILTRE.SH	60
V. /ETC/INIT.D/ROUTE.SH	61

I. INTRODUCTION

A. CONTEXTE

Dans le cadre des trois travaux pratiques qui composent l'unité d'enseignement Systèmes et Réseaux II, il nous a été donné comme tâche de monter un réseau d'entreprise et de l'accompagner d'une infrastructure de services. Ce réseau est constitué d'un ensemble d'agences géographiquement distantes qui possèdent elles-mêmes un réseau privé interne constitué du serveur et d'un poste client DELL. Les agences ont la possibilité de communiquer entre elles grâce à un réseau d'interconnexion et sont toutes reliées à un réseau externe, celui de l'IEM. Tous ces réseaux sont connectés grâce à des serveurs qui agissent comme des routeurs. L'objectif de ce projet est donc d'installer et de configurer correctement ces serveurs afin de permettre la bonne utilisation des réseaux et des services par les différentes agences. L'ensemble des fichiers de configuration et les scripts principaux réalisés pour la mise en place des fonctionnalités mentionnées dans ce rapport seront disponibles dans la partie annexes.

B. ARCHITECTURE FONCTIONNELLE

Pour nos agences, nous avons défini plusieurs fonctionnalités intéressantes à mettre en place sur notre serveur. Tout d'abord, il faudrait qu'elle possède un site internet pour pouvoir s'en servir comme vitrine. Pour répondre à cette fonctionnalité, il faudra donc installer un serveur web, Apache par exemple, ainsi que les outils usuels qui vont avec comme une base de données et PHP. Chaque agence devra également posséder un nom de domaine propre pour qu'on puisse accéder aux services des autres agences facilement. Puisque nous travaillons dans un modèle d'entreprise, un service d'authentification via annuaire est également intéressant pour pouvoir connaître la composition de l'agence, autant au niveau de ses employés que de ses services. Un moyen d'implémenter cette fonctionnalité est le service LDAP. On peut également fournir aux postes du réseau privé de l'agence un répertoire commun pour pouvoir travailler de manière collaborative plus facilement. Il est possible d'ajouter ce service en ajoutant Samba et en le liant à l'authentification de LDAP. Pour des raisons de sécurité, il est important de fournir à chaque agence un système de sauvegardes. Une sauvegarde régulière d'un ou plusieurs dossiers sur le serveur d'une autre agence permettrait de dupliquer les données importantes en cas de panne. Les bases de données utilisées par une agence devront aussi être sauvegardées régulièrement pour permettre une option de retour en arrière en cas de problème. Le serveur web et le service d'annuaire devront pouvoir être accessibles depuis l'extérieur. Enfin, il faudra que chaque agence puisse communiquer avec les autres à travers le réseau d'interconnexion et que chaque agence soit protégée par un pare-feu.

II. MISE EN MARCHÉ DU SERVEUR

A. LE SYSTÈME D'EXPLOITATION

PRESENTATION

Le système d'exploitation mis en place pour notre serveur est un Debian Stretch 9 se basant sur une distribution GNU/Linux. La version 9 est la dernière version stable sortie de Debian, disponible depuis le 17 juin 2017. Elle contient donc un noyau linux accompagné plus de 51000 paquets de logiciels et de certains utilitaires comme gcc, apt, PHP, Python, Apache, ...

AVANTAGES

Le premier avantage du système d'exploitation Debian est qu'il est libre. Cela signifie que la communauté a accès à son code source mais surtout que celui-ci est gratuit. Et cela est bien évidemment un énorme avantage pour nous qui apprenons, mais aussi pour l'université qui n'a pas besoin de payer un très grand nombre de licences propriétaires. Du point de vue du sujet, cela est également un avantage pour une entreprise qui n'a, dans la même idée, pas besoin de dépenser de l'argent dans des licences pour mettre en place son réseau, ce qui réduit finalement le coût total de la main d'œuvre.

Son deuxième avantage est qu'il a toujours été pensé pour une utilisation sur des serveurs. De fait, dès l'installation, certains paquets très utiles à la création d'un serveur (Apache, PHP, MariaDB, ...) sont déjà présents à l'intérieur du système. On peut noter aussi que l'installation de paquets passe par APT (Advanced Packaging Tool) qui est un gestionnaire de paquets très réputé pour simplifier l'installation, la mise à jour et la suppression de paquets. Le système de paquetage de Debian, dpkg, est d'ailleurs considéré comme l'un des meilleurs du marché. Le très grand nombre de logiciels disponibles grâce aux paquets permet également d'éviter d'installer des programmes qui viennent de n'importe où.

Un autre avantage de Debian est la sécurité offerte par l'OS. En effet, celui-ci est réputé comme étant l'un des plus fiables de la distribution Linux/GNU. Cela s'explique par le fait que, comme tous les systèmes d'exploitation libres, les équipes qui travaillent sur celui-ci publient régulièrement les différentes failles de sécurité découvertes. Cela permet à la communauté et aux spécialistes de la sécurité de trouver rapidement des corrections aux différents problèmes.

Ensuite, comme nous l'avons précisé dans la présentation de l'OS, Stretch 9 est une version stable de Debian, ce qui le rend très fiable et réduit la probabilité de voir les logiciels et démons que nous utilisons planter régulièrement pour des raisons plus ou moins obscures. C'est donc un très gros avantage pour une machine comme la nôtre, qui est un serveur, et qui devra donc pouvoir tourner la plupart du temps sans avoir à la redémarrer régulièrement. Toujours concernant la stabilité, les mises à jour du système d'exploitation ont lieu tous les 18 mois en moyenne. Un tel temps permet à la fois aux développeurs de s'assurer que leurs mises à jour sont bien stables et aux entreprises de ne pas avoir à migrer d'une version à l'autre trop régulièrement.

Enfin, son dernier avantage est la présence d'une communauté très active et dynamique. En cas de problème, il est très facile de trouver une solution en effectuant une recherche sur Internet, ce qui est un plus non négligeable pour des « amateurs » comme nous qui commençons dans l'installation et la mise en place d'un réseau.

INCONVENIENTS

Malgré tous ses avantages, Debian Stretch 9 présente tout de même quelques inconvénients. Tout d'abord, puisque la distribution est libre, celle-ci reste dans cette philosophie jusqu'au bout et propose moins de logiciels

commerciaux au profit de logiciels libres. Pour certains, cet inconvénient n'en sera pas un mais si l'entreprise souhaite que les agences de son réseau utilisent un logiciel commercial, comme un de la suite office par exemple, cela ne sera pas forcément possible et il faudra envisager des alternatives (comme Libre Office pour reprendre l'exemple).

Ensuite, conséquence du fait que Debian soit libre, l'entreprise ne pourra pas bénéficier d'un interlocuteur fixe attribué par la signature d'un contrat pour un partenariat. Elle devra se contenter de la communauté présente sur Internet ou au mieux des quelques consultants certifiés par Debian. Cela peut être un problème pour certaines entreprises qui ne veulent pas engager de spécialiste pour maintenir leur réseau.

Parmi les autres défauts trouvés lors de nos recherches, il a également été mentionné plusieurs fois que la configuration de certains matériels (les imprimantes par exemple) sont plus difficiles sur un système Debian que sur d'autres comme Windows par exemple.

Cependant, malgré ces quelques défauts, on peut quand même en conclure que Debian 9 Stretch est un système d'exploitation tout à fait adapté à notre projet du fait de son orientation côté serveur, de sa politique de partage et de sa fiabilité.

B. PARTITIONS DE LA MEMOIRE

Nous avons pris la décision pour notre serveur de fractionner sa mémoire en plusieurs parties afin de spécialiser chaque partition pour un usage particulier et ainsi éviter de perdre toutes les données si l'une d'elles venait à être corrompue.

Dans un premier temps, 4 partitions ont été créées.

La première est la partition système, la racine de notre système d'exploitation (/), et elle mesure 70 Go. Celle-ci n'est pas plus grosse car notre serveur reste une vieille machine et qu'elle ne contient pas un disque dur avec une très grande capacité.

La deuxième est la partition « home ». Celle-ci contient les répertoires des utilisateurs et donc possiblement des données sensibles. Il s'agit également d'une partition qui peut vite grossir et empiéter sur le système. C'est pour cela qu'il est important de l'isoler dans une partition. Puisque la machine est un serveur, il est cependant peu utile de donner un très grand espace de stockage pour les utilisateurs, qui seront surtout des rôles pour manipuler les fonctionnalités du réseau. Nous avons donc décidé de lui attribuer 5 Go.

La troisième est la partition « swap ». Il s'agit d'une zone tampon où la mémoire vive de la machine pourra se décharger afin de ne pas saturer. Les partitions « swap » sont moins répandues aujourd'hui puisque les machines possèdent des mémoires vives de plus en plus importantes. Cependant, puisque la nôtre est un peu plus vieille, elle ne possède que 2 Go de RAM (Random Access Memory/Mémoire vive). Il est donc intéressant de faire une telle partie. Pour savoir quelle taille attribuer au swap, on considère en général que cette partie doit faire 1,5 à 2 fois la taille de la RAM, tout en ne dépassant pas 8 Go au total. De fait, nous avons attribué 4 Go à cette partition.

Enfin, la quatrième est la partition « log ». Comme son nom l'indique, elle contiendra tous les fichiers de log des différents services installés sur le serveur. Faire une partition pour cela sur un serveur est indispensable puisque ceci est typiquement un cas où les fichiers vont très vite prendre beaucoup de place. Il faut donc éviter qu'ils en prennent trop au point d'étouffer le reste du système.

Puisque les noyaux Linux n'autorisent que 4 partitions primaires, les partitions dont la description est écrite dans le MBR (Master Boot Record, première zone de la mémoire chargée au lancement de la machine),

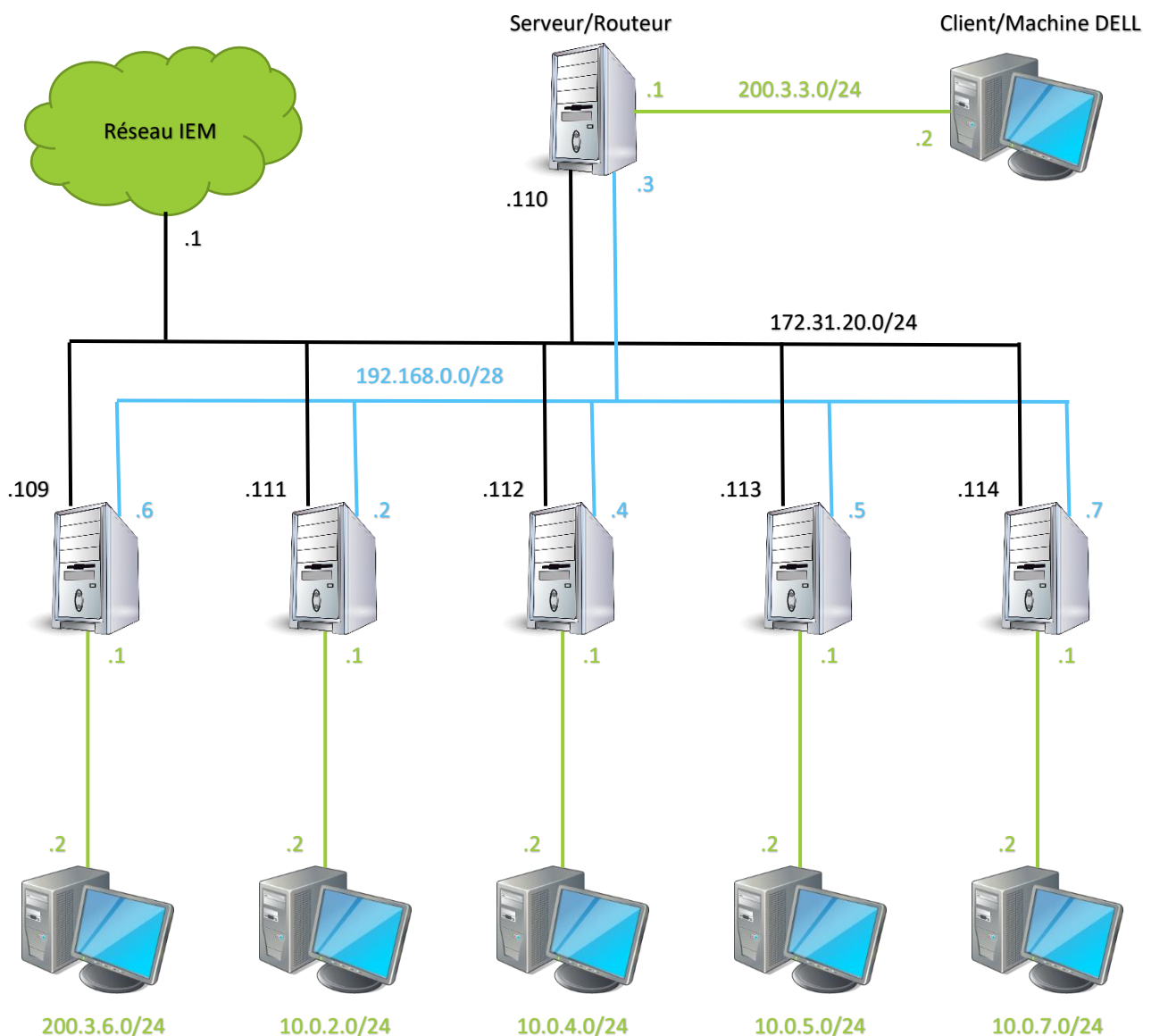
nous avons décidé de partir sur un système de partitions étendues pour pouvoir se permettre d'ajouter d'autres partitions plus tard. Ce système consiste à considérer une seule partition comme primaire et ensuite de se servir de celle-ci comme d'un container pour les autres partitions, qui sont alors des partitions secondaires (ou logiques). La partition primaire de notre machine est la partition système.

Concernant le formatage, toutes les partitions ont pour système de fichiers EXT4. Nous avons fait ce choix car il permet de gérer des fichiers d'une taille pouvant aller jusqu'à 2^{60} octets (ce qui revient dans notre cas à aucune limite de taille).

Les 4 parties décrites précédemment composent le premier découpage effectué lors de l'installation de la machine. L'idée était ainsi de se réserver le reste de la mémoire disponible pour plus tard créer d'autres partitions en fonction de nos besoins.

III. ADRESSAGE ET ROUTAGE

A. ARCHITECTURE PHYSIQUE



Sur le schéma ci-dessus, notre agence est celle représentée en haut à droite. La carte réseau associée à l'adresse 200.3.3.1/24 (réseau privé) est la carte enp1s0. Celle associée à 192.168.0.3/24 (réseau d'interconnexion) est la carte enp3s0. Celle associée à 172.31.20.110/24 (réseau IEM) est la carte enp0s25. Nous sommes à priori un des derniers groupes avec une IP de réseau privé en 200 parce que les autres ont voulu changer en cours de projet mais que nous étions trop avancés pour tout changer.

B. CONFIGURATION DES ADRESSES IP

INTERETS

Une adresse IP est un numéro d'identification codé sur 32 bits pour la version 4 (IPv4). Elle est attribuée à chaque interface connectée au réseau utilisant le protocole IP (Internet Protocol). Ces dernières sont généralement liées à du matériel informatique tel qu'un routeur, un ordinateur, une imprimante, ... Ainsi, une adresse IP permet d'identifier une interface sur le réseau.

Cette adresse peut être attribuée de deux manières différentes : individuellement pour chaque interface ou via le protocole DHCP qui assigne automatiquement une adresse IP à chacune des interfaces. Ces deux méthodes seront détaillées au sein de ce rapport.

METHODE UTILISEE

Dans un premier temps, afin de tester notre plan d'adressage, nous décidons de définir temporairement les adresses IP de nos interfaces.

Dans la nouvelle version de Debian (Debian 9), une nouvelle commande est apparue sous le nom de **ip** permettant notamment de gérer les interfaces et les routes. Dans un souci de simplicité, nous décidons d'utiliser la commande **ifconfig**, présente dans les versions précédentes de Debian. Pour cela, il nous faut donc installer le package **net-tools** via la commande suivante : **apt-get install net-tools**. Ainsi, nous pourrions définir nos adresses IP temporaires via la commande **ifconfig** et utiliser la commande **route** pour définir les routes au sein de notre réseau.

La commande **ifconfig** se manipule de la manière suivante :

```
ifconfig <interface> <adresse> netmask <masque>
```

Ainsi, pour chacune des interfaces du client et du routeur, nous avons défini leur adresse sur le réseau en suivant l'architecture physique définie auparavant. Par exemple, pour définir l'adresse IP de l'interface du routeur allant vers l'IEM, nous avons exécuté la commande suivante sur le routeur :

```
ifconfig enps0s25 172.31.20.110 netmask 255.255.255.0
```

PERSISTANCE DES MODIFICATIONS

Afin que l'adressage établi auparavant soit sauvegardé, nous devons spécifier ce dernier dans les fichiers de configuration de notre machine. Le fichier permettant cela est le suivant : **/etc/network/interfaces**. La syntaxe utilisée dans ce fichier pour configurer une carte réseau est la suivante :

```
allow-hotplug <nomInterface>
iface <nomInterface> inet static
address <IP>
broadcast <Broadcast>
netmask <masque>
network <addrRéseau>
gateway <passerelle>
```

L'instruction **allow-hotplug <nomInterface>** permet de spécifier que la configuration de l'interface s'effectue au démarrage seulement si cette dernière est connectée à un réseau. L'instruction **auto** aurait également pu être utilisée. La seule différence est que cette dernière configure l'interface qu'elle que soit son état : si elle n'est pas connectée au réseau, la configuration se fera quand même.

Ensuite, la deuxième ligne décrit le nom de l'interface suivi de l'instruction **inet static**. **inet** spécifie que le protocole utilisé est IPV4 et **static** permet de définir la configuration comme étant statique (**dhcp** si la

configuration est dynamique). Dans notre cas, on choisit **static** parce que les adresses IP des cartes de notre serveur ne vont pas évoluer au cours du temps.

Les lignes suivantes permettent de spécifier l'adresse IP correspondante à l'interface, son adresse de broadcast, son masque de réseau, l'adresse de réseau ainsi que la passerelle par défaut. L'adresse de broadcast et de réseau ainsi que la gateway sont facultatives.

Voici en pratique la façon dont nous avons configuré ce fichier pour l'interface enp0s25, celle correspondant à l'IEM. Nous avons placé en annexe la configuration complète de ce fichier.

```
# The primary network interface
# Carte reseau IEM
allow-hotplug enp0s25
iface enp0s25 inet static
    address 172.31.20.110
    netmask 255.255.255.0
    network 172.31.20.0
    broadcast 172.31.20.255
```

PROTOCOLE DE VERIFICATION

Afin de vérifier que les adresses IP sont bien définies de manière durable, nous avons effectué la commande **ifconfig** sans paramètre après redémarrage de notre serveur. Cette dernière permet de lister les interfaces actives avec de nombreuses informations les concernant, telles que leur nom, leur adresse IP, leur adresse de broadcast, de réseau...

De plus, nous avons utilisé la commande **ping** qui permet de vérifier que deux machines sont bien capables de dialoguer entre elles. Cette commande a donc permis de vérifier la communication entre notre routeur et le client, notre routeur et le routeur de l'IEM et notre routeur et le routeur d'un autre groupe connecté au réseau d'interconnexion. Etant donné qu'il est directement relié à ces réseaux, il est donc logique qu'il puisse communiquer, sans règles de routage encore définies, avec d'autres machines directement connectées à chacun de ces réseaux.

Si ces deux vérifications sont validées, alors notre adressage est défini correctement.

C. IMPLEMENTATION DU NAT

INTERETS

Le NAT (Network Address Translation) est un mécanisme permettant de faire correspondre à plusieurs adresses IP une ou plusieurs autres adresses IP. Son utilisation est courante dans le cas d'un réseau interne constitué de machines n'ayant pas des adresses IP routables et uniques à l'échelle d'internet. Une translation d'adresse est donc nécessaire afin que ces dernières puissent accéder à internet. Il s'agit également d'un excellent moyen pour masquer l'existence d'un réseau privé du point de vue d'Internet. Ce mécanisme fonctionne grâce à une table où sont sauvegardées les adresses IP du réseau à masquer. Quand un paquet sort du réseau par le routeur, la source est modifiée pour prendre à l'extérieur l'apparence du routeur. Quand il revient, ce dernier retrouve la machine du réseau privé qui avait envoyé le paquet d'origine grâce à sa table NAT puis lui redirige le résultat.

Dans notre cas, nous avons utilisé le NAT pour que notre réseau privé puisse communiquer avec le réseau de l'IEM (et donc Internet) ainsi qu'avec le réseau d'interconnexion. Nous allons voir par la suite comment nous avons mis en place ce mécanisme.

METHODE UTILISEE

Afin de mettre en place le mécanisme de translation d'adresse, nous décidons de créer un script contenant toutes les commandes nécessaires.

Dans un premier temps, il est nécessaire de s'assurer que le routage des paquets IP est activé (`ip_forwarding`). Pour cela, nous avons ajouté à notre script la commande suivante :

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Il aurait également été possible d'aller modifier directement le fichier de configuration **`sysctl.conf`** situé dans le dossier **`/etc`**. Pour activer de manière définitive l'`ip-forwarding`, il aurait alors suffi de décommenter la ligne **`#net.ipv4.ip_forward=1`** en enlevant le dièse.

Ensuite, afin de mettre en place le NAT pour notre routeur, nous avons utilisé la commande **`iptables`**. Cette dernière permet de configurer temporairement un ensemble de règles utiles pour le filtrage de paquets et pour le NAT. Ces règles sont supprimées à chaque redémarrage de la machine.

Voici la commande utilisée pour mettre en place le NAT :

```
iptables -t nat -A POSTROUTING -s 200.3.3.0/24 -j MASQUERADE
```

Pour commencer, il faut indiquer la table sur laquelle nous voulons agir avec **`iptables`**, en spécifiant l'option **`-t`**. Dans notre cas, il s'agit logiquement de la table NAT.

Ensuite, nous avons utilisé l'option **`-A`** afin de définir la chaîne que nous voulons utiliser. Cette chaîne va préciser à quel moment le NAT va être déclenché. Il s'agit ici de la chaîne **`POSTROUTING`**, signifiant que la translation de l'adresse source aura lieu après que le routage aura été effectué. Cette chaîne autorise seulement la modification de l'adresse source. En effet, comme le routage a été effectué vers une adresse précise, il est logique que l'on ne puisse pas la modifier avant le routage, ce dernier ayant besoin de connaître l'adresse IP exacte de la destination pour pouvoir prendre la bonne décision. Pour modifier l'adresse IP de destination (DNAT), nous aurions utilisé la chaîne **`PREROUTING`** qui autorise la modification de l'adresse de destination avant le routage. Cette chaîne est utilisée notamment pour rediriger des paquets vers une DMZ, un sous-réseau où l'on place les machines pouvant être jugées comme dangereuses pour un réseau privé car exposées sur Internet.

Nous spécifions ensuite l'adresse du réseau pour laquelle nous voulons que le mécanisme de translation d'adresse s'effectue : il s'agit ici de notre réseau privé, possédant l'adresse IP suivante : `200.3.3.0/24`.

Enfin, nous spécifions la cible, c'est-à-dire ce que la règle doit faire si elle reçoit un paquet satisfaisant la condition précédente. Ici, la cible est **`MASQUERADE`**, c'est-à-dire qu'on remplace l'adresse IP source de notre paquet pour l'adresse IP de l'interface par lequel il a été dirigé lors du routage.

Ainsi, tout paquet provenant de notre réseau privé pourra circuler vers l'IEM et le réseau d'interconnexion. Cependant, la règle NAT définie précédemment sera supprimée en cas de redémarrage du routeur. Nous allons donc voir dans la partie suivante comment rendre cette règle permanente.

PERSISTANCE DES MODIFICATIONS

Afin que le NAT ne soit plus temporaire, il est nécessaire que le script créé précédemment soit lancé à chaque démarrage de notre machine.

La commande **iptables-save** permet de sauvegarder les règles définies avec iptables dans un fichier. Nous lançons dans un premier temps le script défini dans la partie précédente afin d'ajouter la règle NAT à la table ayant le même nom. Ensuite, nous sauvegardons celle-ci dans le fichier **/etc/iptables.rules** grâce à la commande iptables-save, comme nous pouvons le voir ci-dessous.

```
iptables-save > /etc/iptables.rules
```

Il faut maintenant charger le fichier de sauvegarde à chaque démarrage de notre routeur. Pour cela, la commande **iptables-restore** permet de restituer des règles iptables grâce à un fichier fourni par l'entrée standard (STDIN). Nous ajoutons donc dans le fichier **/etc/network/interfaces** la ligne de commande suivante :

```
pre-up iptables-restore < /etc/iptables_rules.txt
```

La commande **pre-up** permet d'exécuter une commande avant l'activation d'une interface. Dans notre cas, il s'agit de l'interface correspondant au réseau privé puisque le NAT s'effectue pour ce dernier. Nous plaçons donc la commande ci-dessus au niveau de la configuration de la carte enp1s0.

Ainsi, notre NAT est défini de manière permanente. Cependant, nous verrons par la suite qu'il existe une autre méthode pour rendre le NAT persistant, et que la méthode décrite ci-dessus n'est d'ailleurs pas celle que nous avons retenue à posteriori. En effet, nous avons estimé qu'il était plus propre de ne pas inclure des fichiers à lancer au démarrage n'importe où et qu'il était plus intéressant de tout gérer cela grâce à **systemd** que nous décrirons plus tard dans ce rapport.

PROTOCOLE DE VERIFICATION

Afin, de vérifier que notre règle NAT est bien prise en compte par notre routeur, nous exécutons la commande **iptables -L -t nat** qui permet de lister les règles de la table NAT de notre routeur. Nous obtenons le résultat suivant :

```
Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
MASQUERADE all  --  200.3.3.0/24          anywhere
```

On peut ainsi voir que les paquets venant du réseau privé et à destination de n'importe où à l'extérieur du réseau de l'agence (IEM ou interconnexion) sont masqués comme prévu. D'autres vérifications plus fonctionnelles seront effectuées lorsque le routage sera mis en place.

D. ROUTAGE

INTERET

Le routage est un mécanisme qui permet d'indiquer à une machine à qui envoyer un paquet dans le cas où elle ne connaîtrait pas le destinataire (quand celui-ci est un réseau auquel elle n'est pas directement connectée). On nomme une règle de routage une route.

Plusieurs routes ont été établies sur notre routeur afin que les paquets envoyés à destination des autres sous-réseaux privés (autres agences) ou du routeur de l'IEM puissent être transmis. Nous allons voir comment ce mécanisme a été mis en place.

METHODE UTILISEE

Tout d'abord, nous avons utilisé la commande **route** afin de pouvoir ajouter nos règles de routage. Elle prend la syntaxe suivante :

```
route add -net <destination> netmask <masque> gw <passerelle>
```

Dans un premier temps, grâce au mot-clé **add**, on précise qu'on souhaite ajouter une nouvelle route (**del** pour en supprimer).

Ensuite, grâce à l'option **-net**, nous indiquons quels sont les paquets de données qui vont suivre cette route selon leur adresse IP de destination. Il s'agit d'une adresse de réseau et non pas d'une adresse de machine. On indique également le masque de cette adresse juste après avec le mot-clé **netmask**. A noter qu'il aurait également été possible de ne pas utiliser **netmask** et de directement indiquer le masque avec la notation CIDR juste après l'adresse IP (ex 200.3.6.0/24)

Pour finir, nous déclarons la passerelle après **gw** en indiquant vers quel routeur le paquet de données doit se diriger s'il doit aller sur le réseau indiqué juste avant.

Une route par défaut est également à définir pour notre routeur afin d'envoyer les paquets pour lesquels il ne possède pas de règle de routage précise. Voici la syntaxe de cette commande :

```
route add default gw <passerelle>
```

Pour notre projet, nous avons spécifié six routes au total : cinq afin d'assurer la communication avec les autres groupes (une pour chaque groupe) et une autre par défaut allant vers le routeur de l'IEM. Voici deux exemples :

```
route add -net 200.3.6.0 netmask 255.255.255.0 gw 192.168.0.6
```

Cette route permet d'assurer l'envoi de paquets à destination du réseau privé 200.3.6.0/24. On spécifie que la passerelle à utiliser est 192.168.0.6 (l'adresse IP de la carte réseau du groupe 6 sur l'interconnexion).

Nous avons répété cette route pour chacun des sous réseaux privés en changeant l'adresse de destination et la passerelle en fonction de chacun des groupes (voir annexes, script **route.sh**).

```
route add default gw 172.31.20.1
```

Ici, nous spécifions notre route par défaut vers le routeur de l'IEM qui a pour adresse IP 172.31.20.1/24.

PERSISTANCE DES MODIFICATIONS

Afin que les routes soient prises en compte lors du démarrage de la machine, il est nécessaire de créer un script comportant chacune des règles vues précédemment. Pour cela, nous avons créé un script shell nommé **route.sh** dans le répertoire **/etc/scripts**. Nous le lançons à partir du fichier **/etc/network/interfaces** qui est chargé à chaque démarrage. Nous utilisons la commande **up** permettant de charger des routes après le démarrage des cartes à partir d'un script de la manière suivante.

```
up /etc/scripts/route.sh
```

Les règles de routage sont donc définies de manière permanente puisqu'elles sont définies à chaque démarrage des interfaces réseau du routeur. Cependant, de la même manière que pour le NAT, nous n'avons finalement pas retenu cette solution, pour la même raison. Nous verrons donc une nouvelle fois dans la suite du rapport comment spécifier l'exécution d'un script au démarrage de la machine.

PROTOCOLE DE VERIFICATION

Pour vérifier que les routes définies juste avant sont bien prises en compte par notre routeur, nous avons effectué la commande suivante : **route -n**. Cette dernière affiche la table routage, soit toutes les routes actives sur notre machine. Nous avons alors juste vérifié que toutes nos routes étaient présentes dans cette table après redémarrage de la machine.

```
root@Servera:/home/alex# route -n
Table de routage IP du noyau
Destination      Passerelle      Genmask          Indic Metric Ref       Use Iface
0.0.0.0          172.31.20.1     0.0.0.0          UG      0      0        0 enp0s25
10.0.2.0         192.168.0.2     255.255.255.0    UG      0      0        0 enp3s0
10.0.4.0         192.168.0.4     255.255.255.0    UG      0      0        0 enp3s0
10.0.5.0         192.168.0.5     255.255.255.0    UG      0      0        0 enp3s0
10.0.7.0         192.168.0.7     255.255.255.0    UG      0      0        0 enp3s0
172.31.20.0      0.0.0.0         255.255.255.0    U       0      0        0 enp0s25
192.168.0.0      0.0.0.0         255.255.255.240  U       0      0        0 enp3s0
200.3.3.0        0.0.0.0         255.255.255.0    U       0      0        0 enp1s0
200.3.6.0        192.168.0.6     255.255.255.0    UG      0      0        0 enp3s0
root@Servera:/home/alex#
```

Ensuite, étant donné que notre règle NAT a été définie dans la partie précédente, notre machine du réseau privé, tout comme notre serveur, devait pouvoir communiquer avec les autres machines placées sur les réseaux privés des autres groupes (à condition que leur NAT soit fonctionnel). En plus de cela, notre réseau privé devait également pouvoir communiquer avec l'IEM et donc avec internet.

Nous avons alors effectué plusieurs tests avec la commande **ping** vers les différentes interfaces citées précédemment. Nous avons également testé d'accéder à internet via notre réseau privé.

Une fois ces vérifications faites, nous avons pu en conclure que les parties routage et déploiement du NAT sur notre serveur étaient fonctionnelles.

E. IMPLEMENTATION DU SERVICE DHCP

INTERET

Le service DHCP (Dynamic Host Configuration Protocol) est un protocole permettant à un routeur d'attribuer automatiquement des adresses IP à des machines qui lui sont reliées. Nous avons mis en place ce protocole pour configurer automatiquement l'adresse IP de notre machine située sur le réseau privé grâce à notre routeur. Nous allons voir comment ce service a été mis en place.

METHODE UTILISEE

Tout d'abord, nous avons installé le service DHCP sur notre routeur via la commande suivante :

```
apt-get install isc-dhcp-server
```

Ensuite, nous avons manipulé le fichier de configuration ***dhcpd.conf*** (présent dans la partie annexes) se trouvant dans le répertoire ***/etc/dhcp***. Dans un premier temps, nous définissons le paramétrage de notre DHCP :

```
default-lease-time 14400;  
max-lease-time 14400;
```

Le **default-lease-time** correspond à la durée (en secondes) du bail par défaut que le DHCP propose au client. Concrètement il s'agit du temps pendant lequel l'IP sera valide et attribuée à la machine qui l'a demandée. Le **max-lease-time** correspond à la durée (en secondes) maximale du bail que le DHCP peut attribuer à un client.

Dans notre cas, nous avons choisi d'utiliser une durée de bail par défaut de 4 heures. Cette valeur nous semble intéressante puisque notre client ne reste jamais connecté plus de quatre heures, la durée d'un TP, à notre routeur. Cela nous permet également d'éviter de surcharger le réseau avec les requêtes émises en broadcast par le serveur et le client au moment de l'attribution d'une adresse IP.

Ensuite, nous spécifions les différentes options que le DHCP va proposer au client à travers le bail qu'il lui transmettra (masque de réseau, adresse de broadcast, adresse de notre routeur, adresse de DNS). L'adresse de DNS fournie est celle de notre serveur DNS dont l'installation sera détaillée dans la suite de ce rapport.

```
option subnet-mask 255.255.255.0;  
option broadcast-address 200.3.3.255;  
option routers 200.3.3.1;  
option domain-name-servers 200.3.3.1;
```

Pour spécifier la plage d'adresse que nous souhaitons attribuer aux machines de notre sous-réseau privé, nous utilisons la syntaxe suivante :

```
subnet 200.3.3.0 netmask 255.255.255.0 {  
    range 200.3.3.2 200.3.3.10 ;  
}
```

Ces instructions permettent d'attribuer aléatoirement une adresse IP dans la plage d'adresse spécifiée. Dans notre cas, l'adresse de notre sous-réseau privé est 200.3.3.0 et celle de notre routeur sur ce dernier est 200.3.3.1. Nous décidons donc d'attribuer une adresse IP à nos clients entre 200.3.3.2 et 200.3.3.10 puisqu'il n'y aura jamais plus de 9 machines connectées à notre réseau privé.

En pratique, il est plus intéressant de relier l'adresse IP de la machine à son adresse MAC. En effet, cela permet d'identifier précisément que telle machine possède telle adresse IP contrairement à la méthode ci-dessus qui attribue aléatoirement une adresse IP pour chaque machine. En cas de trafic anormal de données avec internet, il sera donc aisé de trouver d'où provient celui-ci. De plus, cela permet un confort supplémentaire au client qui possédera toujours la même adresse IP. Pour tester cette fonctionnalité, nous avons attribué une adresse IP au client que nous disposions ce jour-là de la manière suivante :

```
host client1 {  
    hardware ethernet 02:03:04:05:06:07;  
    fixed-address 200.3.3.11  
}
```

Etant donné que nous changeons de client à chaque séance de TP, nous n'avons pas défini pour chaque adresse MAC une adresse IP spécifique. Nous laisserons le DHCP faire l'attribution aléatoirement. Cependant, en entreprise, il peut être intéressant de lier l'adresse IP à l'adresse MAC pour la raison évoquée précédemment.

Enfin, il est nécessaire de spécifier l'interface d'écoute de notre DHCP. Nous souhaitons attribuer nos adresses IP uniquement aux machines de notre sous-réseau privé connectées à l'interface **enp1s0**. Nous devons donc préciser cette information dans le fichier **/etc/default/isc-dhcp-server** de la manière suivante :

```
INTERFACESv4="enp1s0"
```

Maintenant que notre DHCP est en place, il était intéressant de mettre en place un fichier de log spécifique au DHCP. Sans aucune configuration, les logs du DHCP sont envoyés dans le fichier **/var/log/syslog**, avec les logs de plein d'autres services. Séparer ceux du DHCP des autres permettrait de remarquer plus facilement qui a interagi avec ce dernier, quand et pour quel résultat. Pour cela, nous spécifions dans le fichier de configuration du DHCP, **/etc/dhcp/dhcp.conf**, l'instruction **log-facility local7**. Cela permet d'envoyer les logs du DHCP via le protocole **syslog**. Sous Unix, les logs sont rangés dans plusieurs catégories appelées **facilities**, qui vont permettre de filtrer les logs (mail, user, console, local...). Nous utilisons la catégorie « local » puisqu'il s'agit de logs générés localement dans notre réseau. Un niveau de priorité défini entre 0 et 7 est également présent. Plus celui-ci est faible, plus le message a une priorité importante. Dans notre cas, il s'agit d'un simple message de débogage, nous utilisons donc le numéro 7.

Ensuite, nous spécifions dans le fichier de configuration du protocole **syslog (/etc/rsyslog.conf)**, que les logs provenant de local7 sont stockés dans le fichier **/var/log/dhcpd.log** que nous avons créé précédemment avec l'instruction :

```
local7.*                /var/log/dhcpd.log
```

Afin d'éviter de continuer de loguer le DHCP dans **/etc/log/syslog**, nous modifions l'instruction suivante dans le fichier de configuration de syslog (**/etc/rsyslog**) :


```
*.*;auth,authpriv.none -/var/log/syslog
```

Cette instruction permet de spécifier quelle catégorie de log ne vont pas s'inscrire dans **/var/log/syslog**. Nous ajoutons donc local7 à la liste déjà existante comme ci-dessous :

```
*.*;auth,authpriv.none,local7.none -/var/log/syslog
```

Enfin, il est nécessaire de redémarrer les services **rsyslog** et **isc-dhcp-server** pour que notre DHCP et que le système de log soient fonctionnels. Nous utilisons les commandes suivantes :

```
service rsyslog restart
service isc-dhcp-server restart
```

PROTOCOLE DE VERIFICATION

Afin de vérifier que le service DHCP fonctionne, nous avons simplement redémarré le client. L'adresse IP de ce dernier n'étant pas fixée de manière permanente, si l'adresse IP est à nouveau présente au redémarrage sans intervention de notre part, c'est que le DHCP a bien fonctionné. Si ce n'est pas le cas, une erreur s'est produite. Nous pouvons savoir de quel type d'erreur il s'agit grâce au fichier de log mis en place précédemment.

F. COMMANDES DU GESTIONNAIRE DE PACKAGES A CONNAITRE

Un gestionnaire de paquets est un outil qui permet d'automatiser le processus d'installation, de désinstallation et de mise à jour de logiciel informatique. Il permet donc de mettre à disposition facilement de nombreux paquets pour une installation standard. Sur une distribution Debian, on distingue quatre outils principaux pour gérer les paquets : **dpkg**, **apt**, **apt-get** et **aptitude** que nous détaillons ci-dessous.

La commande **dpkg** est un outil de bas niveau si nous le comparons à **apt** ou **aptitude**. Il permet, entre autres, d'installer un logiciel sur une machine à partir d'un package. Les fichiers de packages sont, dans ce cas, des fichiers téléchargés auparavant qui ont pour format **.deb**. Voici un résumé des principales utilisations de la commande **dpkg** :

- **dpkg -i <nomPaquet>** : permet d'installer le paquet précisé en paramètre.
- **dpkg -r <logiciel>** : désinstalle le logiciel dont on a précisé le nom.
- **dpkg-reconfigure <paquet>** : reconfigure un paquet déjà installé.

La commande **apt** simplifie la gestion des paquets. En effet, elle permet notamment l'installation et la désinstallation de paquets directement à partir de leur nom. Le téléchargement préalable du paquet qu'il était nécessaire d'effectuer avec **dpkg** n'est plus à faire. Voici une synthèse de son utilisation :

- **apt install <nomPaquet>** : installe sur la machine le paquet souhaité.
- **apt search <Texte>** : liste les paquets ayant dans leur nom le texte spécifié.
- **apt show <NomPaquet>** : affiche les détails du paquet spécifié.
- **apt remove <NomPaquet>** : désinstalle le paquet voulu.
- **apt update** : met à jour la listes des paquets. Le dépôt se trouve dans le fichier **/etc/apt/source.list**.
- **apt upgrade** : met à jour le système en installant et en mettant à jour les paquets.

La commande **apt-get** est très proche de l'instruction **apt**. Elle offre cependant plus de fonctionnalités de bas niveau que cette dernière. Voici une liste de ces utilisations :

- **apt-get install <nomPaquet>** : installe le paquet souhaité.
- **apt-get remove <nomPaquet>** : désinstalle le paquet spécifié.
- **apt-get autoremove <nomPaquet>** : désinstalle le paquet ainsi que toutes les dépendances liées à ce dernier.
- **apt-get autoremove --purge <nomPaquet>** : désinstalle le paquet, ses dépendances ainsi que ses fichiers de configuration.
- **apt-get clean** : **apt** conserve une copie de chaque paquet sur notre disque dur. Cette commande permet de supprimer ces paquets pour libérer de l'espace mémoire.
- **apt-get update** : met à jour la liste des paquets de la même manière que apt update.
- **apt-get upgrade** : met à jour le système de la même manière que apt upgrade.

La commande **aptitude** présente des fonctionnalités équivalentes à apt-get. Nous n'allons donc pas détailler son fonctionnement puisque ses options sont les mêmes que la commande précédente. Cependant, elle présente une interface utilisateur permettant de gérer les paquets comme nous pouvons le voir ci-dessous.

```
C-T: Menu ? : Help q: Quit u: Update g: Preview/Download/Install/Remove Pkgs
aptitude 0.8.7 @ Servera DL: 0 B/115 kB
--- Security Updates (11)
--- Paquets pouvant être mis à jour (28)
--- Paquets installés (664)
--- Paquets non installés (50171)
--- Paquets virtuels (5914)
--- Tâches (216)

Security updates for these packages are available from security.debian.org (or mirrors).
Ce groupe contient 11 paquets.
```

IV. INSTALLATION D'UN DNS

A. COMMANDES ET FICHIERS UTILES AU DNS SOUS DEBIAN

Un DNS, ou Domain Name System, est un service informatique basé sur un système d'arborescence qui permet d'effectuer une correspondance entre un nom de domaine et une adresse IP. Un nom de domaine est tout simplement un nom plus simple à retenir donné à une adresse IP. Par exemple, on utilise quotidiennement le nom de domaine www.google.fr plutôt que l'adresse 8.8.8.8. Sous Debian, il existe trois commandes permettant d'interroger un serveur de nom : **host**, **dig** et **nslookup**.

La commande **host** permet d'obtenir un certain nombre d'informations concernant une machine reliée à Internet. Par défaut, la commande affiche l'adresse IP de la machine quand on lui passe un nom de domaine et inversement. Si des informations complémentaires sur la machine sont disponibles dans la base de données du serveur de domaine, il est possible d'en prendre connaissance en utilisant les options **-t** ou **-a** avant l'IP/le nom de domaine.

```
alexis@GUYOT-PC:~$ host www.debian.org
www.debian.org has address 5.153.231.4
www.debian.org has address 130.89.148.14
www.debian.org has IPv6 address 2001:67c:2564:a119::148:14
www.debian.org has IPv6 address 2001:41c8:1000:21::21:4
```

Ici, on peut voir grâce à la commande **host** que le nom de domaine www.debian.org est associé à 4 adresses IP, 2 IPv4 et 2 IPv6. Le fait qu'il y ait plusieurs adresses IP nous indique que la machine possède plusieurs interfaces réseau. Il est possible de connaître le nom des serveurs DNS de Debian en utilisant la commande :

```
➤ host -t ns www.debian.org
```

```
alexis@GUYOT-PC:~$ host -t ns www.debian.org
www.debian.org name server geo2.debian.org.
www.debian.org name server geo3.debian.org.
www.debian.org name server geo1.debian.org.
```

La commande **dig** est un utilitaire s'installant à travers le paquet **dnsutils** et qui permet lui aussi d'interroger des serveurs DNS. La réponse à une requête **dig** est cependant bien plus précise et développée que celle offerte par **host**. Par défaut, **dig** ne fournit les informations que sur les adresses IPv4.

```

alexis@GUYOT-PC:~$ dig www.debian.org

; <<>> DiG 9.11.3-1ubuntu1.1-Ubuntu <<>> www.debian.org
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 60004
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 3, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.debian.org.                IN      A

;; ANSWER SECTION:
www.debian.org.                20      IN      A      5.153.231.4
www.debian.org.                20      IN      A      130.89.148.14

;; AUTHORITY SECTION:
www.debian.org.                2783    IN      NS      geo1.debian.org.
www.debian.org.                2783    IN      NS      geo3.debian.org.
www.debian.org.                2783    IN      NS      geo2.debian.org.

;; Query time: 26 msec
;; SERVER: 89.2.0.1#53(89.2.0.1)
;; WHEN: Tue Apr 02 18:39:01 DST 2019
;; MSG SIZE rcvd: 132

```

Dans cette réponse, les champs importants sont :

- ➔ **Status**, dans la partie ->>HEADER<<- : Il contient une valeur qui indique s'il est possible de résoudre le nom de domaine passé en paramètre. Le champ peut valoir entre autres NOERROR en cas de réussite, NXDOMAIN si le nom de domaine n'existe pas ou SERVFAIL si le serveur DNS est mal configuré.
- ➔ **Question section** : Cette section nous indique le type d'IP associée au nom de domaine www.debian.org. Ici, A signifie que l'IP est en IPv4.
- ➔ **Answer section** : Affiche les interfaces réseau associées au nom de domaine dans le serveur DNS.
- ➔ **Authority section** : Cette partie indique la liste des DNS de référence pour le nom de domaine fourni. Le NS signifie Name Server et indique que ce qui suit est le nom du serveur DNS de référence. Ici, on peut voir que Debian possède trois serveurs DNS, sur les machines geo1, geo2 et geo3. Cette catégorie n'apparaît pas forcément au retour de la commande. Il est cependant possible de forcer son apparition en précisant l'option **ns** après le nom de domaine.

```
➤ dig www.debian.org ns
```

- ➔ La dernière partie contient des statistiques sur la requête, comme son temps d'exécution par exemple.

Enfin, **nslookup** est la dernière commande permettant de rechercher des informations dans un serveur DNS. Elle n'est actuellement plus maintenue par UNIX et est dépréciée au profit des deux dernières. Cependant, elle reste quand même très utilisée par les utilisateurs. Nslookup permet pour un nom de domaine de connaître son adresse IP, le nom de la machine sur laquelle se trouve le domaine et éventuellement les alias (CNAME) mis en place. Un CNAME est une façon de dire que deux noms de domaine couvrent la même adresse et que l'un est un alias de l'autre. Il est également possible de passer une adresse IP en paramètre, la réponse s'adapte alors pour fournir le nom de domaine à la place de l'adresse IP. Il existe un mode interactif de nslookup qui s'active quand l'adresse ou le domaine n'est pas passé en premier argument. Le mode interactif permet à l'utilisateur

d'interroger les serveurs de noms pour obtenir des informations à propos de nombreux autres serveurs ou domaines. Le mode non-interactif affiche juste les informations pour le domaine demandé.

```
alexis@GUYOT-PC:~$ nslookup www.debian.org
Server:      89.2.0.1
Address:     89.2.0.1#53

Non-authoritative answer:
Name:   www.debian.org
Address: 5.153.231.4
Name:   www.debian.org
Address: 130.89.148.14
Name:   www.debian.org
Address: 2001:67c:2564:a119::148:14
Name:   www.debian.org
Address: 2001:41c8:1000:21::21:4
```

Les deux premiers champs indiquent le serveur DNS qui a répondu à la requête **nslookup**. Il est indiqué que la réponse n'a pas été fournie par les machines d'autorité de Debian, geo1, geo2 et geo3 que nous avons vues précédemment. Cependant la machine nous retourne quand même les résultats que nous attendions, à savoir les 2 adresses IPv4 et les 2 IPv6 du nom de domaine.

Sur une machine Debian, l'adresse du serveur DNS à qui demander de résoudre un nom de domaine inconnu est inscrit dans le fichier **resolv.conf** du dossier **/etc**. On indique dans celui-ci le nom de domaine géré par le serveur identifié avec le mot clé **nameserver** grâce aux mots-clés **domain** et **search**.

```
domain agence.toulouse
search agence.toulouse
nameserver 200.3.3.1
#nameserver 172.31.21.36
```

Il est également possible de « simuler localement » un serveur DNS en indiquant dans le fichier **/etc/hosts** des correspondances entre adresses IP et noms de domaine.

B. INSTALLATION DU DNS

FONCTIONNEMENT GENERAL

Pour notre serveur, nous avons utilisé le paquet **bind9** pour mettre en place le DNS, que nous avons configuré en tant que « maître principal » et « cache local ». Nous aurions également pu utiliser l'application **unbound** qui fournit globalement les mêmes fonctionnalités que **bind9**. Cette dernière est bien plus récente (2006 contre 1984 pour la première version de **bind**) et est très performante pour la technologie de serveurs modernes. Cependant, puisque nous utilisons une vieille machine comme serveur, il est plus intéressant de rester sur **bind**. Il s'installe grâce à la commande

```
apt-get install bind9
```

« Maître principal » est l'une des trois façons de configurer un serveur DNS. Dans cette configuration, on définit un ensemble d'enregistrements DNS pour un nom de domaine. Ceux-ci sont définis dans ce qu'on appelle une zone. Les deux autres configurations possibles sont le « cache local » et le « maître secondaire ». Le « cache local » revient à faire en sorte que **bind9** se souvienne des requêtes DNS et des réponses en les plaçant dans un cache pour ne pas avoir à refaire plusieurs fois la même requête. Cette méthode est particulièrement utile pour diminuer l'utilisation de la bande passante et le temps de latence. Notre serveur DNS implémente également cette configuration puisqu'elle est mise en place par défaut par **bind9**. Enfin, le « serveur esclave » est une configuration utilisée pour un serveur DNS secondaire utilisé en complément à un autre serveur maître principal. Le serveur secondaire est donc une copie des zones configurées sur le serveur principal. L'utilité de ce type de configuration est de pouvoir assurer la disponibilité du serveur DNS même si le premier est hors-ligne. Cependant dans notre cas, cela ne sera pas utile puisque nous ne disposons pas d'autre serveur DNS qui pourrait faire office de « serveur maître ».

Les informations fondamentales du DNS sont stockées dans des enregistrements qu'on appelle des RR (Ressources Records). Ces derniers se trouvent dans des zones, que nous devons par la suite configurer pour mettre en place notre serveur DNS. Il existe plusieurs types d'enregistrements :

- **SOA** : Start Of Authority Record, pour indiquer les informations générales de la zone qui contient les enregistrements comme son serveur principal, le mail de contact, quelques durées, comme celle d'expiration par exemple, ainsi que le numéro de série de la zone.
- **NS** : Name Server Record, pour définir le serveur DNS de référence d'un domaine.
- **A** : Adresse Record ou Enregistrement d'Hôte en français, fait correspondre un nom d'hôte, de domaine ou de sous-domaine à une adresse IPv4.
- **PTR** : Pointer Record, fait l'inverse d'un enregistrement A, à savoir associer une adresse IP à un nom de domaine.
- **MX** : Mail Exchange Record, définit les serveurs de mail pour un domaine.
- **CNAME** : Canonical Name Record, permet de définir un alias pour un autre domaine.

Une fois le concept d'enregistrement compris, il suffit d'aller modifier quelques fichiers de configuration qui se trouvent dans le dossier **/etc/bind**. Ces scripts complets sont disponibles dans les annexes.

FICHIERS DE CONFIGURATION

FICHIER *NAMED.CONF.OPTIONS*

Le fichier **named.conf.options** contient les options communes à toutes les zones du serveur BIND.

```
options {
    directory "/var/cache/bind";

    // If there is a firewall between you and nameservers you want
    // to talk to, you may need to fix the firewall to allow multiple
    // ports to talk. See http://www.kb.cert.org/vuls/id/800113

    // If your ISP provided one or more IP addresses for stable
    // nameservers, you probably want to use them as forwarders.
    // Uncomment the following block, and insert the addresses replacing
    // the all-0's placeholder.

    forwarders {
        192.168.0.6; 192.168.0.2; 192.168.0.4; 192.168.0.5; 192.168.0.7; 172.31.21.35;
    };
}
```

Le champ qui nous intéresse dans la mise en place de notre serveur DNS est l'option **forwarders**. Elle permet d'indiquer une liste de serveurs DNS à joindre si celui du serveur n'a pas réussi à résoudre un nom de domaine. Dans cette liste, on place alors les adresses des serveurs DNS des autres groupes pour pouvoir utiliser leurs noms de domaine et on finit par l'adresse du DNS de l'IEM qu'il faut questionner en dernier recours.

FICHER NAMED.CONF.LOCAL

C'est dans ce fichier qu'on ajoute les différents noms de domaine qu'on souhaite ajouter à notre serveur DNS, avec pour chaque la création d'une zone. Une zone est composée de la sorte :

```
zone « nom » { options } ;
```

```
zone "agence.toulouse" {
    type master;
    file "/etc/bind/db.agence.toulouse";
};

zone "informatique.agence.toulouse"{
    type master;
    file "/etc/bind/db.informatique.agence.toulouse";
};

zone "3.3.200.in-addr.arpa" {
    type master;
    file "/etc/bind/db.agence.toulouse.inv";
};
```

L'option **type** permet d'indiquer si le serveur est esclave ou maître. Comme nous l'avons vu précédemment, ici notre serveur est maître. Le deuxième paramètre, **file**, permet d'indiquer où se trouvent les informations sur la zone. Deux noms de domaine sont définis dans notre serveur, pour des raisons qui seront expliquées ultérieurement dans ce rapport. Plutôt que de créer deux zones, il est important de noter qu'il aurait également été possible de passer par un CNAME, « agence.toulouse » et « informatique.agence.toulouse » correspondant à la même adresse IP. Cependant, les deux méthodes fonctionnent et nous avons utilisé cette implémentation simplement parce que nous avons découvert l'existence des CNAME qu'après coup. Il est également nécessaire de définir une zone inverse, qui couvrira le cas où l'utilisateur demandera le nom de domaine en passant une adresse IP. Une zone inverse fonctionne de la même manière qu'une zone classique.

Seul le nom est un peu spécifique car on le compose du net-id du réseau sur lequel se trouve le serveur DNS, ici notre réseau privé en 200.3.3.0/24, écrit à l'envers et suivi de « in-addr.arpa ».

Pour chaque zone ajoutée dans ce fichier de configuration (même les zones inverses), il faut créer un autre fichier (celui passé en paramètre de l'option file), où on va pouvoir préciser les enregistrements de la zone. Ceux-ci sont de la forme suivante : la durée de validité de l'enregistrement, le « Time To Live » (TTL), suivie des enregistrements en eux-mêmes. Voici pour illustrer le fichier de configuration de la zone « agence.toulouse ».

```
;
; BIND data file for agence.toulouse interface
;
$TTL      604800
@         IN      SOA      agence.toulouse. root.agence.toulouse. (
                        2      ; Serial
                        604800 ; Refresh
                        86400  ; Retry
                        2419200 ; Expire
                        604800 ) ; Negative Cache TTL
;
@         IN      NS       agence.toulouse.
@         IN      A        200.3.3.1
servera   IN      A        200.3.3.1
```

Ici, la zone possède 4 enregistrements. Mettre un @ en début de ligne permet d'indiquer que les informations qui suivent sont relatives à la zone. On voit alors qu'on dispose d'un enregistrement SOA, d'un NS et deux A. Je vous renvoie à la partie précédente pour voir ce à quoi correspond chaque type d'enregistrement.

Il est important de noter qu'un seul SOA est autorisé par zone. Le premier paramètre de ce dernier correspond au nom du serveur maître de la zone, ici donc la zone en elle-même. Le deuxième est l'adresse mail du responsable de la zone. Le point entre root et agence.toulouse correspond à l'arobase. A l'intérieur du SOA :

- « serial » est un numéro de série/de version pour identifier la zone.
- « refresh » est l'intervalle de temps au bout duquel les serveurs secondaires associés à cette zone vont devoir rafraîchir leurs informations.
- « retry » est l'intervalle de temps au bout duquel les serveurs devront réessayer d'obtenir les informations sur la zone en cas d'échec la première fois.
- « expire » est le temps au bout duquel les serveurs devront se débarrasser des informations s'ils n'arrivent plus à questionner la zone.
- « negative cache TTL » est le temps au bout duquel les réponses négatives stockées dans le cache doivent être supprimées.

Avec la seconde ligne, on indique avec le @ que le nom de domaine agence.toulouse est une adresse Internet (IN) qui a pour serveur de nom de référence lui-même. Les lignes 3 et 4 enfin permettent d'associer l'adresse 200.3.3.1 au nom de domaine et à notre machine (qui s'appelle servera). Ainsi notre machine se retrouve liée au nom de domaine agence.toulouse par association.

Pour la zone inverse, la syntaxe est tout à fait similaire, mis à part le fait que comme expliqué précédemment dans la présentation des différents types d'enregistrements, il faut en utiliser un de type PTR pour associer la machine 200.3.3.1 (le net-id 200.3.3 est indiqué dans le nom de la zone) avec le nom de domaine.


```

$TTL 604800
@ IN SOA agence.toulouse. root.agence.toulouse. (
    1 ; Serial
    604800 ; Refresh
    86400 ; Retry
    2419200 ; Expire
    604800 ) ; Negative Cache TTL
;
1 IN NS servera.agence.toulouse.
1 IN PTR servera.agence.toulouse.

```

C. PROTOCOLE DE VERIFICATION

Afin de vérifier la bonne implémentation de notre DNS, nous avons effectué une série de tests à l'aide de la commande **nslookup**. Depuis notre serveur, nous avons d'abord effectué les commandes suivantes

```

➤ nslookup agence.toulouse
➤ nslookup 200.3.3.1

```

Pour valider ces tests, il suffisait alors que les commandes retournent respectivement « 200.3.3.1 » et « servera.agence.toulouse ». Ensuite, nous avons réeffectué les mêmes commandes sur le client de notre réseau privé, avec toujours les mêmes attentes. Enfin, nous avons essayé une fois Apache installé (voir partie suivante) d'accéder à notre site Internet directement en tapant notre nom de domaine dans la barre de recherche. Ce dernier s'affichant correctement, nous en avons conclu que notre installation était correcte.

V. INSTALLATION DES FONCTIONNALITES

A. SERVEUR APACHE

INTERET

Apache est un serveur HTTP libre. Il permet à un serveur WEB de communiquer avec des clients (navigateurs) en utilisant les protocoles HTTP et HTTPS. Apache est souvent couplé avec d'autres logiciels permettant de gérer une base de données par exemple. Dans notre cas, nous utiliserons MySQL et PostgreSQL. Nous serons également amenés à manipuler PHP pour tester la connexion à ces dernières.

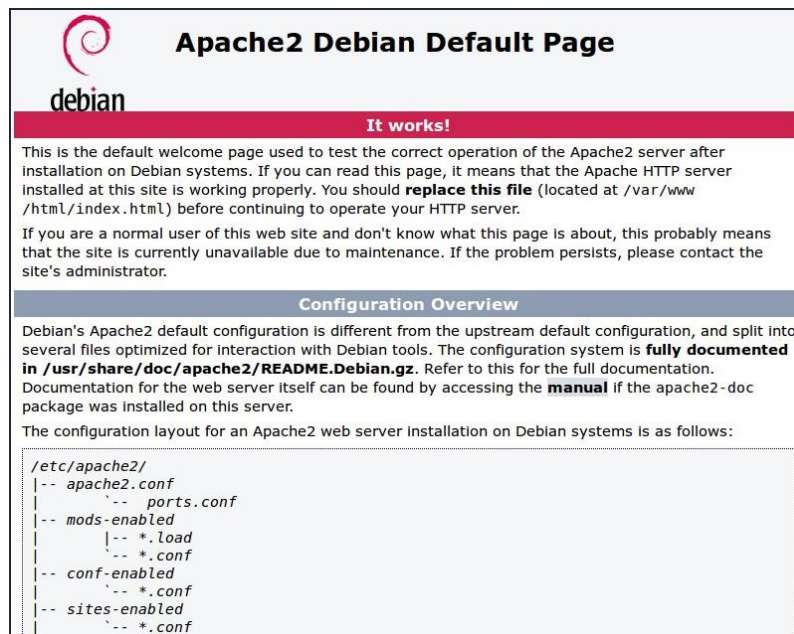
INSTALLATION ET VERIFICATION

Comme pour la plupart des paquets, nous installons apache via le gestionnaire de paquet mis en à disposition sur les machines Debian. Nous exécutons donc la commande

```
apt-get install apache2
```

Pour vérifier que notre installation s'est bien déroulée, nous testons d'accéder depuis le client à la page web par défaut de notre serveur. Cette information est précisée dans le fichier **000-default.conf** situé dans le répertoire **/etc/apache2/sites-available**. Il s'agit du fichier de configuration de la page par défaut de notre serveur web. L'instruction **DocumentRoot** indique le chemin absolu du répertoire du site web. Il s'agit ici de **/var/www/html**. Par défaut, le serveur va donc chercher un fichier dans ce dossier s'intitulant **index.html**. Nous verrons plus en détail par la suite la syntaxe d'un fichier de configuration de site web hébergé sur un serveur apache.

Pour accéder à la page Web par défaut de notre serveur, on saisit donc l'adresse IP de ce dernier (ou le nom de domaine **agence.toulouse** puisque notre DNS est fonctionnel) dans la barre de recherche d'un navigateur sur notre client. La page ci-après s'affiche alors dans notre navigateur. Il s'agit de la page par défaut que propose Apache via le fichier **index.html**.



Apache2 Debian Default Page

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Debian systems. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

Debian's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Debian tools. The configuration system is **fully documented** in **/usr/share/doc/apache2/README.Debian.gz**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the `apache2-doc` package was installed on this server.

The configuration layout for an Apache2 web server installation on Debian systems is as follows:

```
/etc/apache2/
|-- apache2.conf
|   |-- ports.conf
|-- mods-enabled
|   |-- *.load
|   |-- *.conf
|-- conf-enabled
|   |-- *.conf
|-- sites-enabled
|   |-- *.conf
```

CREATION D'UN COMPTE UTILISATEUR POUR UN DEVELOPPEUR WEB

Le développeur devra être en mesure de créer un site web qui sera stocké dans le répertoire **/srv/login/www** situé sur le serveur Web. Depuis un navigateur, son site devra être accessible via l'URL suivant : **http://agence.toulouse/login**.

Dans un premier temps, nous avons créé l'utilisateur sur notre serveur grâce à la commande **adduser <identifiant>**. Notre utilisateur se nomme **webdev**. Plusieurs informations sont demandées par le système à la création de cet utilisateur (numéro de téléphone, nom complet...). Nous ne les mentionnons pas puisqu'elles ne sont pas importantes.

Ensuite, sachant que le site web doit être stocké dans **/srv/webdev/www**, nous devons créer l'arborescence correspondante. Nous mettons donc en place le répertoire **webdev** dans **/srv** et le répertoire **www** dans **/srv/webdev** à l'aide de la commande **mkdir**.

L'arborescence créée précédemment est destinée à l'utilisateur **webdev**. Les répertoires **webdev** ainsi que **www** doivent donc avoir comme seul propriétaire celui-ci. Nous exécutons pour cela la commande suivante :

```
chown -R webdev /srv/webdev
```

La commande **chown** permet définir un propriétaire sur un répertoire ou sur un fichier. Ici, nous spécifions **webdev** comme étant le propriétaire du répertoire **/srv/webdev**. L'option **-R** permet d'exécuter cette commande dans les sous-répertoires de **/srv/webdev**, donc pour le répertoire **www**.

Pour que le répertoire créé précédemment soit pris en compte par Apache, nous devons également le spécifier dans le fichier de configuration Apache : **/etc/apache2/apache2.conf** de la manière suivante :

```
Alias "/webdev" "/srv/webdev/www"
<Directory /srv/webdev/www>
    Options Indexes FollowSymLinks
    AllowOverride None
    Require all granted
</Directory>
```

L'instruction **<Directory>** permet de regrouper un ensemble de directives ne s'appliquant qu'au répertoire spécifié, à ses sous-répertoires et à leur contenu. Dans notre cas, il s'agit logiquement du répertoire du développeur créé précédemment : **/srv/webdev/www**. Les directives à appliquer à ce chemin sont les suivantes :

- ➔ **Options Indexes** : indique qu'Apache contrôle l'indexation des répertoires (Apache affiche la liste des dossiers dans le cas où aucun index.html n'est trouvé mais qu'il existe des sous-répertoires).
- ➔ **Options FollowSumLinks** : le serveur suit les liens symboliques dans le répertoire concerné.
- ➔ **AllowOverride None** : le serveur ne prend pas en compte d'autres configurations définies via un fichier **.htaccess**. Comme nous n'avons jamais utilisé ces fichiers, nous décidons de désactiver cette option pour des raisons de sécurité.
- ➔ **Require all granted** : le serveur autorise tous les utilisateurs à accéder au répertoire.

Aucune spécification n'a été faite sur les utilisateurs ayant le droit d'accéder au site web du développeur : nous avons donc autorisé chaque personne ayant accès au réseau d'y accéder. L'option **Indexes**

est intéressante puisqu'elle permet au développeur de pouvoir visualiser les dossiers créés dans son répertoire de développement. Il pourra ainsi créer facilement plusieurs site web dans son répertoire.

Enfin, pour permettre au développeur d'accéder à son site via l'URL : **http://agence.toulouse/webdev**, nous devons spécifier un alias. Ce dernier va permettre de desservir aux utilisateurs connectés au serveur des documents situés en dehors du répertoire de stockage par défaut de site web du serveur (**/var/www/html**). Ainsi, nous spécifions que l'alias **/webdev** correspondra au contenu du répertoire **/srv/webdev/www** qui est situé en dehors du répertoire par défaut. Le développeur pourra alors facilement visualiser son travail via l'URL <http://agence.toulouse/webdev>.

CREATION D'UN SECOND COMPTE UTILISATEUR

Nous souhaitons désormais créer un utilisateur qui aura son site web accessible à partir de l'adresse suivante : **http://informatique.agence.toulouse/**.

Dans un premier temps, nous définissons un nouvel utilisateur appelé **userapache** de la même manière que précédemment. Nous lui créons un répertoire à son nom dans **/srv** et un dossier **www** à l'intérieur de ce dernier pour lui permettre de publier son site web.

Ensuite, afin de lier le nom de domaine du site : **informatique.agence.toulouse** et l'emplacement du site web en lui-même sur le serveur, il est nécessaire de définir un **VirtualHost**. Ce dernier est spécifique au site de l'utilisateur **userapache** et sera stocké dans un fichier appelé **informatique.agence.toulouse** situé dans le répertoire **/etc/apache2/sites-available**. Voici le contenu de ce fichier :

```
<VirtualHost *:80>
    DocumentRoot "/srv/userapache/www"
    ServerName informatique.agence.toulouse
    <Directory /srv/userapache/www>
        Options Indexes FollowSymLinks
        AllowOverride None
        Require all granted
    </Directory>
</VirtualHost>
```

Dans un premier temps, nous déclarons l'hôte virtuel (**VirtualHost**) en spécifiant qu'il doit répondre aux requêtes qui s'adressent à toutes les adresses IP de notre serveur (réseau privé, Interconnexion et IEM) sur le port 80 (HTTP). Nous aurions pu spécifier seulement l'adresse IP de l'interface du sous réseau privé si nous aurions voulu que notre site soit accessible seulement pour les machines de ce réseau. Ensuite, nous spécifions les directives suivantes :

- ➔ **DocumentRoot** pour spécifier dans quel répertoire est stocké le site web.
- ➔ **ServerName** pour indiquer le nom de domaine auquel le VirtualHost va répondre.

Ensuite, tout comme précédemment, nous spécifions les différentes options à appliquer sur le répertoire. Ces dernières sont expliquées dans la partie précédente.

Enfin, il ne reste plus qu'à indiquer à Apache que notre site est plus que disponible, qu'il est carrément activé. Pour cela, il suffit de créer un lien symbolique appelé **informatique.agence.toulouse.conf** dans le répertoire **/etc/apache2/sites-enabled** de la manière suivante :

```
ln -s /etc/apache2.conf/sites-available/informatique.agence.dijon
informatique.agence.dijon.conf
```

Il est alors nécessaire de redémarrer Apache avec la commande suivante pour que les modifications soient prises en comptes.

```
service apache2 restart
```

MISE EN PLACE D'UN PUBLIC_HTML POUR LES UTILISATEURS OCCASIONNELS

Afin que les utilisateurs occasionnels puissent développer leur site web personnel sur le serveur, nous souhaitons mettre en place un répertoire **public_html** auquel ils pourront accéder via l'adresse : **http://agence.toulouse/~login**. Pour cela, nous avons activé le module **userdir** via la commande suivante :

```
a2enmod userdir
```

La commande **a2enmod** permet d'activer un module au sein de la configuration d'apache. Le module **userdir** permet l'accès aux répertoires propres à un utilisateur. Nous avons ensuite redémarré apache pour que les modifications effectuées soient prises en compte.

Nous avons ensuite créé un répertoire **public_html** pour l'utilisateur **alex** ainsi qu'une page html à l'intérieur de celui-ci, **index.html**. Enfin, nous avons tenté d'accéder depuis notre client à cette page avec l'adresse suivante : **http://agence.toulouse/~alex**. La page s'affichant, nous avons donc validé cette fonctionnalité.

Afin de valider notre travail de manière générale sur Apache, nous avons, pour finir, créé des pages web de test dans chacun des répertoires devant être accessibles des différentes manières évoquées tout au long de cette partie. Puisque que nous avons réussi à accéder à toutes ces dernières, nous avons pu en conclure que l'installation d'Apache était réussie sur notre serveur.

AUTORISER UN UTILISATEUR A MODIFIER LA CONFIGURATION D'APACHE

Actuellement, seul l'utilisateur root a le droit de modifier la configuration d'apache et de relancer le service. Nous décidons d'autoriser un utilisateur **webmaster** (créé auparavant grâce à la commande **adduser**) à faire ces manipulations. Cela peut être utile dans le cas où cet utilisateur serait le responsable développement d'une entreprise. Il est important de ne pas autoriser n'importe quel utilisateur à pouvoir configurer apache pour des raisons de sécurité.

Pour effectuer cela, nous allons manipuler la configuration de l'utilitaire **sudo**. Cet outil permet à un utilisateur d'effectuer des tâches en se faisant passer pour un autre utilisateur.

Le fichier de configuration **/etc/sudoers** permet notamment de spécifier les droits d'exécution de certaines tâches pour les utilisateurs. Nous indiquons donc dans ce fichier que l'utilisateur **webmaster** a le droit de configurer apache et de redémarrer ce service de la manière suivante :

```
webmaster      ALL=(ALL:ALL)      /bin/nano /etc/apache2/  
apache2.conf   /bin/systemctl restart apache2 /usr/bin/vi  
/etc/apache2/apache2.conf /usr/sbin/service apache2 restart
```

- ➔ Nous spécifions dans un premier temps le nom de l'utilisateur auquel nous souhaitons donner les droits : **webmaster**.
- ➔ Ensuite, l'instruction **ALL=(ALL:ALL)** suit la syntaxe suivante : **host:(user:group)**. Nous autorisons tous les utilisateurs se nommant **webmaster** appartenant à n'importe quel groupe et connectés depuis n'importe quel hôte à exécuter les commandes placées juste après.
- ➔ Nous autorisons l'utilisateur en question à éditer le fichier de configuration d'apache via les éditeurs de texte **nano** et **vi**. Nous lui autorisons également de redémarrer le service apache2 via les commandes **systemctl** et **service**.

Ainsi, l'utilisateur **webmaster** pourra éditer le fichier **/etc/apache2/apache2.conf** et redémarrer le service apache. Nous avons placé le fichier **/etc/sudoers** en annexe.

B. BASES DE DONNEES

POSTGRESQL

Nous allons désormais installer le Système de Gestion de Base de Données (SGBD) PostgreSQL. Il s'agit d'un logiciel libre faisant face à d'autres SGBD tel que MariaDB, MySQL ou encore Oracle. Son développement est basé sur une communauté mondiale de développeurs. C'est un SGBD relationnel, c'est-à-dire que l'information est stockée sous forme de tableaux à deux dimensions appelés relations ou tables. Il fonctionne sous la plupart des distributions Unix et Linux tel que Solaris, SunOS ou encore Debian. Nous allons donc voir comment nous avons procédé pour mettre en place son installation sur notre serveur.

Pour installer PostgreSQL, nous avons utilisé le gestionnaire de paquet en saisissant la commande **apt-get install postgresql** tout en étant connecté en root. Par défaut, PostgreSQL crée un utilisateur nommé **postgres** avec lequel nous allons pouvoir configurer notre SGBD. Nous utilisons la commande suivante pour se connecter sous cet utilisateur :

```
su postgres
```

Nous avons ensuite créé notre base de données, appelée **dbres**, avec l'instruction :

```
createdb dbres
```

Une fois la base de données créée, nous avons défini un utilisateur qui pourrait s'y connecter. En effet, à ce moment de l'installation, seul l'utilisateur **postgres** le pouvait. Pour remédier à cela, il a fallu dans un premier temps créer un utilisateur sur notre serveur, que nous avons appelé **marinette**, à l'aide de la commande

```
sudo useradd marinette
```

Puis créer un utilisateur dans le SGBD avec la commande **createuser <username>**. Par défaut, la commande cherche à lier le compte en cours de création avec un compte identique présent dans le système d'exploitation. Une fois la commande exécutée, le compte marinette a donc été automatiquement lié à notre utilisateur système.

```
createuser marinette
```

Enfin, nous avons modifié le mot de passe de notre utilisateur en utilisant la requête SQL suivante en étant connecté à PostgreSQL avec l'utilisateur **postgres** (commande **psql**) :

```
alter user marinette with password 'marinette'
```

La commande **alter user** permet de modifier un utilisateur de la base de données. Elle prend en paramètre l'option **with password** qui permet de définir un mot de passe pour un utilisateur donné.

Notre utilisateur était alors défini, ce qui nous a permis de nous connecter à la base donnée avec l'utilisateur marinette. Pour faire cela, il faut dans un premier temps se connecter sur le serveur sous l'utilisateur **marinette**. Ensuite, grâce à la commande **psql -d <nomBD>**, nous pouvons nous authentifier. Les deux commandes ont été résumées ci-dessous :

```
su marinette  
psql -d dbres
```

➔ L'option **-d** permet de spécifier la base de données à laquelle nous souhaitons nous connecter.

MySQL

De la même manière que pour PostgreSQL, nous installons MySQL avec la commande **apt-get install mysql-server**.

Par défaut, l'utilisateur **root** de MySQL est authentifié par son compte système sur notre serveur. En étant connecté en tant que root sur notre serveur, nous pouvons donc nous authentifier à la base de données avec la simple commande **mysql**.

Une fois connecté à MySQL, nous utilisons la requête SQL suivante pour créer notre base de données que nous nommons **dbres** une nouvelle fois.

```
CREATE DATABASE dbres ;
```

Nous pouvons vérifier sa création avec la commande suivante qui affiche les bases de données MySQL existantes :

```
SHOW DATABASES;
```

Résultat :

Database
dbres
information_schema
mysql
performance_schema

Nous pouvons donc remarquer que la base de données a bien été créée.

Ensuite, nous créons un utilisateur que nous nommons **eric** et qui aura accès à MySQL. Nous utilisons pour cela la commande **CREATE USER <username>**

```
CREATE USER 'eric' IDENTIFIED BY 'eric';
```

➔ Nous utilisons la clause **IDENTIFIED BY** pour spécifier le mot de passe de l'utilisateur.

L'utilisateur **eric** pourra se connecter à MySQL grâce à la commande suivante :

```
mysql -u eric -D dbres -p
```

- ➔ L'option **-u** permet de spécifier le nom de l'utilisateur avec lequel nous voulons nous connecter à la base de données.
- ➔ L'option **-D** définit à quelle base de données on souhaite se connecter.
- ➔ L'option **-p** indique qu'un mot de passe doit être saisi pour la connexion. Le système le demandera juste après que la commande soit exécutée.

MISE EN PLACE DE DONNEES

Afin de tester la connexion aux bases de données **dbres** définies dans MySQL et PostgreSQL, nous avons créé une même table appelée **Utilisateur** dans ces deux dernières. Nous décidons d'insérer des données différentes dans chacune des deux tables. Voici la table relationnelle **Utilisateur** pour chacune d'entre elles :

Pour dbres de MySQL :

Nom	Prenom
LECLERQ	Eric
SAVONNET	Marinette

Pour dbres de PostgreSQL :

Nom	Prenom
GUYOT	Alexis
LABROSSE	Alexandre

Tout d'abord, nous avons créé la table Utilisateur au sein des deux bases de données en utilisant l'instruction SQL **CREATE TABLE**, comme nous pouvons le voir ci-dessous :

```
CREATE TABLE Utilisateur (nom VARCHAR(50), prenom VARCHAR(50));
```

- ➔ Nous spécifions en premier le nom de la table : **Utilisateur**.
- ➔ Les éléments entre parenthèses correspondent aux attributs de nos tables. Nous avons donc **nom** et **prenom**.
- ➔ Nous indiquons le type de chacun des attributs après leur déclaration : nous décidons de mettre une chaîne de caractères ayant pour taille 50 pour nos deux attributs.

Ensuite, nous insérons les données décrites dans les deux tables **Utilisateur** vues précédemment. Pour cela, nous utilisons la commande SQL **INSERT INTO** de la manière suivante :

```
INSERT INTO Utilisateur(nom,prenom) VALUES ( 'nom', 'prenom' );
```

- ➔ Nous indiquons en premier le nom de la table dans laquelle nous allons insérer les données, suivie de ses attributs.
- ➔ Ensuite, la clause **VALUES** permet de spécifier les valeurs que nous souhaitons insérer.

SAUVEGARDE

Nous souhaitons enfin mettre en place une sauvegarde quotidienne de nos bases de données MySQL et PostgreSQL. Nous allons dans un premier temps voir en détail comment nous avons procédé pour mettre en place la sauvegarde de la base de données PostgreSQL. Pour cela, nous avons écrit un script shell qui sauvegardera nos données dans un répertoire prédéfini. Celui-ci sera alors déclenché tous les jours à une certaine heure grâce au programme **cron**. Nous verrons comment celui-ci fonctionne plus loin dans le rapport. En attendant, regardons à quoi ressemble notre script de sauvegarde.

```
#!/bin/bash

DATE=`date -I`

# Dossier où sauvegarder les backups

BACKUP_DIR="/backup/postgres"

# Sauvegarde

mkdir $BACKUP_DIR/$DATE
sudo -u postgres pg_dumpall -c > $BACKUP_DIR/$DATE/backup.sql
```

Tout d'abord, nous avons défini deux variables :

- ➔ **DATE** qui correspond à la date du jour sous le format **YYYY-MM-DD**. Nous la récupérons grâce à la commande shell **date -I** qui récupère la date système au format ISO 8601.
- ➔ **BACKUP_DIR** qui correspond au chemin d'enregistrement vers nos répertoires de sauvegarde. Nous décidons de les enregistrer dans **/backup/postgres**.

A noter que pour que cela fonctionne nous sommes allés créer le répertoire de sauvegarde **/backup/postgres/date_du_jour** avec la commande **mkdir**. Enfin, nous procédons à la sauvegarde de nos bases de données PostgreSQL à l'aide la commande **pg_dumpall**. Seul l'utilisateur postgres a le droit d'effectuer cette commande puisqu'il s'agit de l'administrateur de PostgreSQL. Nous devons donc lancer **pg_dumpall** en se substituant à lui, grâce à la commande **sudo -u <username>**. L'option **-c** permet d'inclure les commandes SQL dans le fichier de sauvegarde permettant de nettoyer les bases de données avant de les recréer. Nous redirigeons le fichier dans le répertoire créé précédemment et nous lui donnons comme nom **postgresbackup.sql**.

Pour la sauvegarde des bases de données MySQL, seule la commande de sauvegarde change. Nous avons donc dupliqué le script précédent en le renommant **mysqlbackup.sh** puis nous avons remplacé la commande de sauvegarde par la commande **mysqldump** de la manière suivante :

```
mysqldump -u root -C --all-databases > $BACKUP_DIR/$DATE/backup.sql.tgz
```

- ➔ L'option **-u** permet d'indiquer l'utilisateur MySQL avec lequel la sauvegarde va s'exécuter.

- ➔ Nous décidons de compresser le dossier de sauvegarde en utilisant l'option **-C**.
- ➔ On spécifie que l'on souhaite sauvegarder toutes les bases de données grâce à l'indication **--all-databases**.
- ➔ Pour finir, nous redirigeons le fichier de sauvegarde dans le répertoire créé précédemment et nous lui donnons comme nom **backup.sql.tgz**.

C. PHP

PHP est un langage de programmation web libre. Il est utilisé pour la création de pages web dynamiques à travers un serveur http, comme Apache que nous avons configuré précédemment. Il permet entre autres de se connecter assez facilement à une base de données et d'en extraire les informations demandées par l'utilisateur. Nous allons donc tester la connexion aux deux bases de données que nous avons créées précédemment. Nous avons préalablement installé PHP sur notre serveur en exécutant la commande : **apt-get install php php-mysql php-pgsql**. Les paquets **php-mysql** et **php-pgsql** permettent respectivement de pouvoir utiliser PHP avec MySQL et PostgreSQL.

CONNEXION A LA BASE DE DONNEES MYSQL

Voici le script PHP que nous avons produit pour nous connecter à la base de données. Ce dernier affiche le contenu de la table **Utilisateur**.

```
<?php
    $mysqli = new mysqli("localhost","eric","eric","dbres");
    if($mysqli->connect_errno){
        echo "Echec lors de la connexion à MySQL : (".$mysqli->connect_errno.") ".$mysqli->connect_error;
    }

    $mysqli->real_query("select nom, prenom from utilisateur");
    $res = $mysqli->use_result();
    echo "Contenu de la BD mysql : <br/>";
    while($row = $res->fetch_assoc()){
        echo "Nom : ".$row['nom']." Prenom : ".$row['prenom']."<br/>";
    }
?>
```

Comme vous pouvez le voir, nous utilisons la classe **mysqli**, qui permet de se connecter à une base de données MySQL. Nous spécifions en paramètre le nom d'hôte (localhost car la base de données est hébergée sur le serveur en lui-même), le nom d'utilisateur (eric) et le mot de passe (eric) d'un utilisateur MySQL avec lequel nous nous connectons. Nous ajoutons en dernier paramètre le nom de la base de données (**dbres**).

Ensuite, si aucune erreur ne s'est produite, nous exécutons une requête SELECT sur la table Utilisateur pour récupérer les données stockées dedans avec la méthode **real_query(<requête SQL>)** de l'objet **mysqli** créé précédemment.

Enfin, nous parcourons le résultat de la requête ligne par ligne en l'affichant. Chaque ligne est renvoyée sous la forme d'un tableau associatif par la méthode **fetch_assoc**, avec comme clé le nom de la colonne et comme valeur l'information correspondante.

Ainsi, nous plaçons ce script (**mysql.php**) dans le répertoire **www** de l'utilisateur **userapache** stocké dans **/srv/userapache/www**. Nous pouvons donc y accéder grâce à l'adresse : **http://informatique.agence.toulouse/mysql.php**.

CONNEXION A LA BASE DE DONNEES POSTGRESQL

Voici le script PHP que nous avons écrit pour nous connecter à la base de données PostgreSQL. Ce dernier affiche le contenu de la table **Utilisateur**.

```
<?php
print "Contenu de la BD postgresQL :";
echo "<br/>";
$dbh = pg_connect("host=localhost dbname=dbres user=marinette password=marinette");
if($dbh){
    $result = pg_query($dbh,"select * from utilisateur");
    if($result){
        while($row = pg_fetch_row($result)){
            echo "Nom : $row[0] Prenom : $row[1]<br/>";
        }
    }else echo "Aucun résultat";
}else echo "Echec de connexion à la BD";
?>
```

Dans un premier temps, nous nous connectons à la base de données grâce à la fonction **pg_connect**. Nous indiquons en paramètre, comme pour MySQL, le nom de l'hôte (**localhost**), la base de données (**dbres**), le nom de l'utilisateur (**marinette**) et son mot de passe (**marinette**).

Si la connexion est réussie, nous lançons la requête SELECT sur Utilisateur via la fonction **pg_query**. Ensuite, tout comme pour MySQL, nous parcourons les lignes sélectionnées par la requête puis nous les affichons. La fonction **pg_fetch_row** renvoie une ligne du résultat sous la forme d'un tableau simple. L'index du tableau représente le numéro (en commençant par 0) de la colonne de la table.

Enfin, nous plaçons ce script (**postgres.php**) dans le répertoire **www** de l'utilisateur **userapache**, accessible via le chemin suivant : **/srv/userapache/www**. Nous pouvons donc y accéder grâce à l'adresse : <http://informatique.agence.toulouse/postres.php>.

CONNEXION AUX BASES DE DONNEES VIA PDO

PDO (PHP Data Object) est une extension qui fournit des services d'accès à de nombreux types de bases de données tels que MySQL, Oracle et PostgreSQL. Grâce à ce système, il est très simple de changer de base de

```
<?php
print("<br/>Contenu de la BD PostgreSQL :<br/>");
try{
    $dbh = new PDO('pgsql:host=localhost;dbname=dbres;user=marinette;password=marinette');
    foreach($dbh->query('SELECT * FROM UTILISATEUR') as $row){
        echo "Nom : ".$row[0]. " Prenom : ".$row[1]. "<br/>";
    }
    $dbh=null;
}
catch(PDOException $e){
    print("ERREUR !!!: " . $e->getMessage() . "<br/>");
    die();
}

print("<br/>Contenu de la BD MySQL :<br/>");
try{
    $dbh = new PDO('mysql:dbname=dbres;user=eric;password=eric');
    foreach($dbh->query('SELECT * FROM UTILISATEUR') as $row){
        echo "Nom : ".$row[0]. " Prenom : ".$row[1]. "<br/>";
    }
    $dbh=null;
}
catch(PDOException $e){
    print("ERREUR !!!: " . $e->getMessage() . "<br/>");
    die();
}
?>
```

données pour un même script PHP. En effet, les fonctions permettant la connexion à la base de données et la récupération d'informations auprès de ces dernières sont les mêmes pour tous les types de BD. Voici comment nous avons utilisé PDO.

Tout d'abord, nous définissons un nouvel objet de type PDO. Nous lui spécifions en paramètre le type de la base de données à laquelle nous souhaitons nous connecter (mysql ou pgsql). Ensuite, nous définissons l'hôte (localhost), le nom d'utilisateur (marinette ou eric) puis le mot de passe (marinette ou eric). Nous exécutons la requête via la méthode **query(<requête>)**. Cette dernière renvoie un tableau à deux dimensions correspondant à chaque ligne sélectionnée par la requête. Nous parcourons chacune de ces lignes et nous les affichons.

Ainsi, nous plaçons ce script (**index.php**) dans le répertoire **www** de l'utilisateur **userapache** stocké dans **/srv/userapache/www**. Nous pouvons donc y accéder grâce à l'adresse : <http://informatique.agence.toulouse/>.

D. SAUVEGARDE AUTOMATIQUE

Rsync (Remote Synchronization) est un logiciel de synchronisation unidirectionnelle, c'est-à-dire qu'il copie des fichiers depuis une machine source vers une machine cible. Rsync est donc utilisé pour réaliser des sauvegardes (incrémentielles ou différentielles) ou pour diffuser un répertoire de référence. Ce programme est libre et fonctionne sur de nombreux systèmes d'exploitation, tels que Windows, Mac OS et Linux.

Il est possible d'utiliser Rsync en tant que service. Cela peut être utile pour l'administrateur système s'il souhaite mieux contrôler les synchronisations des données lorsque ces dernières sont multiples. Les commandes seront également plus simples à établir pour le client puisque le répertoire de destination où sera stocké les données est spécifié dans un fichier de configuration. Cependant, étant donné que nous souhaitons seulement établir la sauvegarde d'un répertoire d'un de nos utilisateurs, nous n'optons pas pour cette méthode qui aurait été intéressante dans le cas où nous aurions de nombreuses sauvegardes à gérer. Nous décidons donc d'utiliser simplement la commande **rsync** pour réaliser notre synchronisation.

La sauvegarde devra avoir lieu quotidiennement à 15 heures. Nous décidons de sauvegarder le répertoire de l'utilisateur **alex** sur le serveur du groupe composé de Arnaud François et de Théophile Wallerich. Leur adresse IP sur le réseau d'interconnexion est 192.168.0.2.

MISE EN PLACE DE L'AUTHENTIFICATION SSH VIA DES CLES RSA

Il est nécessaire d'établir une connexion **ssh** afin de se connecter à la machine distante à laquelle nous voulons envoyer nos données. Par défaut, la connexion ssh demande le mot de passe de l'utilisateur auquel nous nous connectons. Nous souhaitons éviter cela, puisque nous voulons mettre en place la sauvegarde de façon automatique, quotidiennement à une heure précise.

Nous décidons donc de définir un système d'authentification client/serveur via la génération de clés RSA privées et publiques. Le principe est simple, le client génère une clé privée ainsi que la clé publique correspondante. La clé privée permettra à ce dernier de prouver son identité auprès du serveur auquel nous avons transféré la clé publique auparavant.

Dans un premier temps, nous générons les clés RSA grâce à la commande suivante :

```
ssh-keygen
```

Deux fichiers sont alors créés (un pour chaque clé) appelés **id_rsa.pub** et **id_rsa** dans le répertoire **/root/.ssh/**. Nous suivons la méthode expliquée précédemment en copiant le fichier correspondant à la clé publique vers le serveur cible, celui de nos camarades. Pour cela, nous utilisons la commande **ssh-copy-id** de la manière suivante :

```
ssh-copy-id /root/.ssh/id_rsa.pub etudiant@192.168.0.2
```

- ➔ Nous indiquons en premier le chemin du fichier contenant la clé publique.
- ➔ Ensuite, nous spécifions l'utilisateur auquel nous souhaitons envoyer la clé suivie de l'adresse IP de l'hôte (sur l'interconnexion).

La clé est désormais envoyée à l'utilisateur **etudiant**. Nous pourrions alors nous y connecter en ssh sans mot de passe.

MISE EN PLACE DE RSYNC

Nous avons établi le script ci-dessous pour lancer la sauvegarde des données de l'utilisateur alex. Celui-ci sera lancé de manière quotidienne grâce au service **cron**. Voici le script établi :

```
#!/bin/bash

date=`date +%Y-%m-%d`
rsync -ae ssh /home/alex etudiant@192.168.0.2:/home/alexrsync/$date
```

- ➔ Nous déclarons une variable **date** ayant le format YY-MM-DD grâce à la commande **date** que nous avons déjà pu utiliser auparavant.
- ➔ Nous utilisons la commande **rsync** en spécifiant **-e ssh** pour indiquer que nous utilisons le protocole **ssh** pour se connecter à la machine. L'option **-a** permet d'indiquer que la sauvegarde s'effectue de manière récursive, c'est-à-dire que tous les sous dossier (et sous sous dossier...) du répertoire indiqué dans la commande seront copiés. On spécifie ensuite le nom de l'utilisateur auquel nous voulons nous connecter, suivi de l'adresse IP du serveur distant. Enfin, nous indiquons le répertoire (sur le serveur de l'autre groupe) dans lequel nous souhaitons stocker nos données. Un dossier pour chaque jour sera donc créé.

Ainsi, en exécutant le script, le répertoire de notre utilisateur alex sera sauvegardé sur le serveur de l'autre groupe. Nous allons maintenant détailler la méthode utilisée pour lancer ce script chaque jour de manière automatique.

AUTOMATISATION DE CERTAINES TACHES

Nous allons évoquer à plusieurs reprises dans ce rapport l'utilisation du programme **cron** pour lancer certaines tâches automatiquement. C'est notamment le cas pour la sauvegarde des bases de données MySQL et PostgreSQL que nous verrons plus tard dans ce rapport ainsi que pour l'enregistrement du répertoire personnel de l'utilisateur **alex**.

Cron utilise un fichier nommé **crontab** stocké directement dans le répertoire **/etc**. Ce dernier permet d'organiser les tâches en spécifiant à quel moment chacune d'entre elles doit s'exécuter. Voici la syntaxe du fichier :

```
minute heure jourDansLeMois mois jourDansLaSemaine tache
```

Voici les instructions que nous avons ajoutées dans le fichier **crontab** :

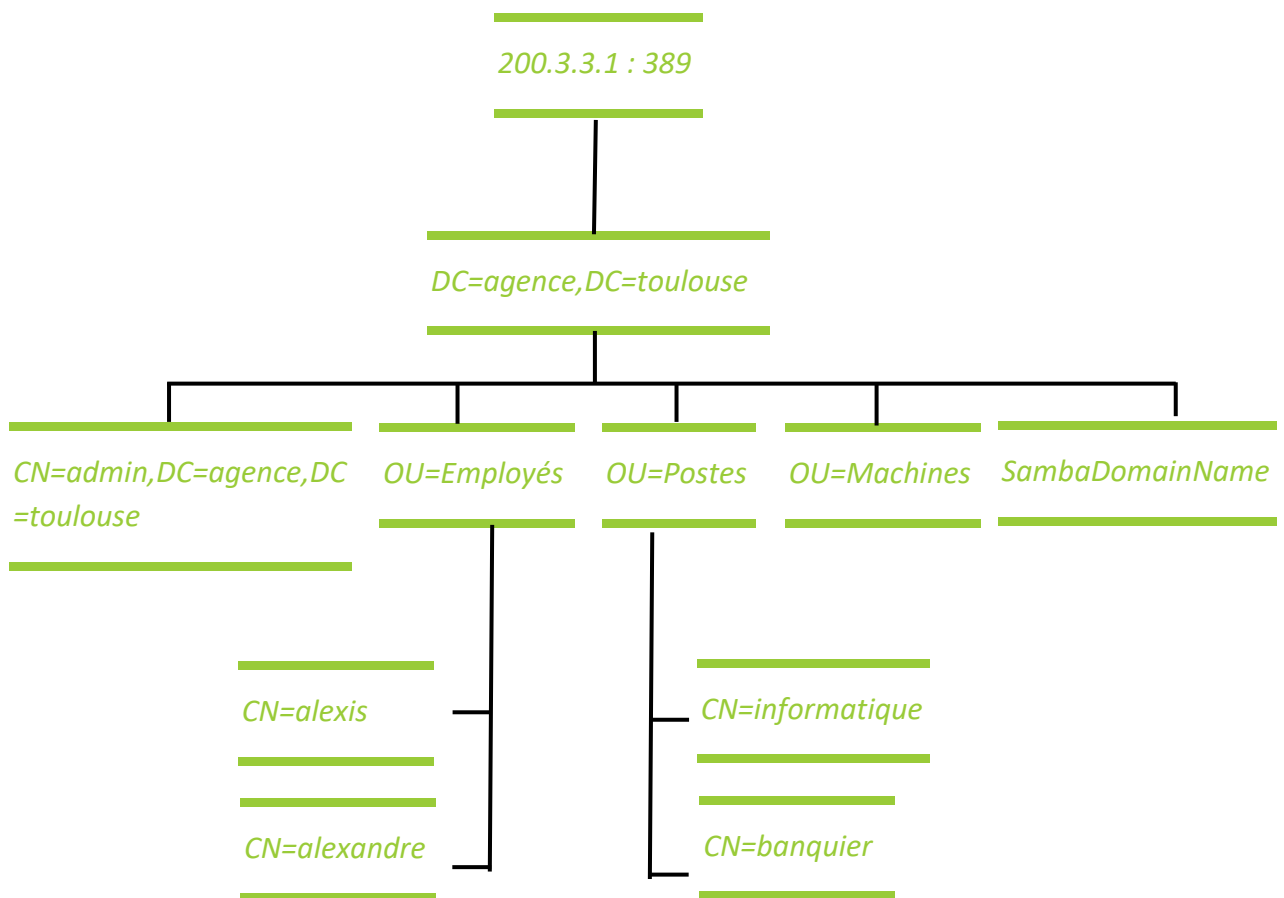
```
0 15 * * * root /etc/scripts/mysqlbackup.sh
0 15 * * * root /etc/scripts/postgresbackup.sh
0 15 * * * root /etc/scripts/rsync.sh
```

Nous pouvons donc voir que les trois scripts de sauvegardes (le répertoire et les BD) sont lancés à partir du fichier **crontab**. Nous spécifions dans un premier temps que nous exécuterons chacun des scripts à 15 heures. Ensuite, le caractère « * » utilisé pour le jour dans le mois, le mois et le jour dans la semaine permet d'indiquer que les tâches seront lancées pour chacune de ces unités. Nous indiquons également l'utilisateur se chargeant de l'exécution des scripts. Dans notre cas, il s'agira de **root**. Nous spécifions enfin le chemin absolu des différents scripts à exécuter.

E. AUTHENTIFICATION ET PARTAGE DE RESSOURCES

QU'EST-CE QUE LDAP ?

LDAP est un système d'annuaire d'entreprise basé sur TCP/IP et développé depuis le début des années 90. Il s'agit d'une norme, c'est-à-dire qu'il est constitué d'un protocole du même nom mais aussi d'un ensemble de modèles pour faciliter sa mise en place, notamment pour la sécurisation, le stockage et le nommage des données. De fait, la plupart des attributs utilisables par LDAP (nom de la personne, login, mot de passe, mail, ...) sont déjà définis dans des schémas (CommonName, Uid, ...). Les données sont organisées dans une structure arborescence appelée le DIT (Directory Information Tree), qui propose une organisation qui se veut proche du modèle de l'entreprise. Dans ce DIT sont stockées un certain nombre d'informations, respectant une nomenclature très précise. Les éléments mis à disposition de LDAP pour construire notre DIT sont de manière non-exhaustive les éléments DC pour Domain Components, pour définir le nom de domaine, CN pour Common Name, les acteurs de l'entreprise, et OU pour Organization Unit, les services de l'entreprise. Pour notre agence, nous avons obtenu le DIT suivant :



A noter que nous avons utilisé les éléments DC car notre agence a un rayonnement à l'échelle régionale. Pour une plus grande entreprise qui se déploie à l'internationale, il aurait fallu utiliser l'élément C pour Country. On peut consulter un annuaire LDAP grâce au navigateur **JxPplorer**, que nous avons donc installé sur notre client pour l'occasion.

IMPLEMENTATION DE LDAP

Pour implémenter la norme LDAP sur notre serveur, nous utilisons l'utilitaire Openldap. Celui-ci a pour avantage d'être une implémentation libre de la norme et d'offrir la possibilité de modifier la configuration de LDAP à chaud sans avoir à redémarrer le service à chaque modification. Dans notre cas, ce dernier avantage ne nous servira pas vraiment puisqu'il faut pour ce faire utiliser des fichiers LDIF que nous n'avons pas vus dans un souci de temps consacré aux TP de Réseau. Cependant, dans un contexte d'entreprise cela peut s'avérer très utile pour l'administrateur réseau qui n'a pas besoin de couper et de relancer le service s'il le modifie. Les paquets à installer pour la mise en place de Openldap sont **slapd** et **ldap-utils**.

```
apt-get install slapd
apt-get install ldap-utils
```

Une fois cela fait, nous avons récupéré le schéma pour l'installation de SAMBA, que nous verrons ultérieurement, et nous l'avons placé dans le dossier « schema » du répertoire **/etc/ldap**. Nous avons ensuite

configuré le fichier `/etc/ldap/slapd.conf` de la façon suivante (le script est disponible dans sa version complète dans les annexes).

```
# Schema and objectClass definitions
include      /etc/ldap/schema/core.schema
include      /etc/ldap/schema/cosine.schema
include      /etc/ldap/schema/nis.schema
include      /etc/ldap/schema/inetorgperson.schema
include      /etc/ldap/schema/samba.schema
```

Dans cette partie du fichier de configuration, nous avons rajouté la dernière directive `include` vers le schéma de samba. C'est dans les schémas inclus à cet endroit du fichier de configuration que LDAP va aller chercher la définition des attributs utilisés.

```
# The base of your directory in database #1
suffix       "dc=agence,dc=toulouse"
checkpoint 512 60

# rootdn directive for specifying a superuser on the database. This is needed
# for syncrepl.
# rootpw      a crypter en dessous
rootdn       "cn=admin,dc=agence,dc=toulouse"
rootpw       {SSHA}8iEY0HAaq37Y2f/w9dyVr7UX6/EZiktr

# Where the database file are physically stored for database #1
directory    "/ldap"
```

Plusieurs informations importantes sont indiquées dans cette partie du fichier de configuration. À l'aide de la directive « `suffix` », on indique le point de départ de notre DIT, nommé de sorte à respecter notre nom de domaine. La directive « `rootdn` » permet d'indiquer le Distinguished Name de l'administrateur de l'annuaire, son super-utilisateur. Comme indiqué sur notre DIT, il s'agit de l'utilisateur `admin`. Dans la ligne « `rootpw` », on indique le mot de passe de l'administrateur, crypté à l'aide de la commande `slappasswd`. Enfin, on précise le répertoire où sera stockée la base de données contenant l'annuaire. Ceci se fait grâce à la directive « `directory` ». Puisque c'est l'utilisateur **ldap** (créé automatiquement à l'installation du service) qui exécute les commandes liées au protocole, il est important d'aller changer le propriétaire du dossier `/ldap` en mettant l'utilisateur **ldap** à la place de **root**. Cela se fait grâce à la commande `chown` que nous avons déjà vue à de multiples reprises dans ce rapport.

Enfin pour ce fichier de configuration, on édite à la fin les droits des différents utilisateurs sur l'annuaire.

```
# Sécurisation du DIT
## Par défaut l'admin peut tout faire et on peut lire le LDAP
access to *
    by dn="cn=admin,dc=agence,dc=toulouse" write
    by * read
```

Avec ces trois lignes, il est dit que pour l'ensemble de l'annuaire (« `access to *` »), l'administrateur (« `by dn="cn=admin ...` ») a le droit d'écriture (`write`) et que lui et tous les autres utilisateurs (« `by *` ») ont le droit de lecture. Les droits du DIT dans LDAP sont nommés les ACL pour Access Control Lists (les listes de contrôle des accès).

Ensuite, il faut préciser dans le fichier **/etc/nsswitch.conf**, le fichier de configuration de tous les services de noms, identification comprise, qu'on souhaite, en plus de l'identification système une identification en utilisant ldap.

```
passwd:                                compat ldap
group:                                compat ldap
```

Pour cela, on édite les lignes ci-dessus du fichier en rajoutant **ldap** après le mot-clé **compat**. Ce dernier correspond à l'authentification système et lui indique qu'il doit utiliser le fichier **/etc/passwd** pour l'effectuer. En ajoutant **ldap** après **compat**, on indique alors au système qu'il doit faire une authentification **ldap** après celle système. L'ordre a son importance ici et nous avons pris la décision de mettre l'authentification ldap dans un second temps pour éviter d'effectuer une requête LDAP inutilement si l'utilisateur est incapable de s'authentifier sur le système d'exploitation.

Enfin, pour finaliser l'implémentation de LDAP, on installe le paquet **libnss-ldap** et on suit l'installation en insérant les différents paramètres au fur et à mesure qu'ils sont demandés par l'installateur. Une fois cela fait, il est important de relancer slapd et de s'assurer qu'il démarre bien. Quand nous avons pu observer cela, nous avons conclu que l'installation de LDAP était bien effectuée et que nous pouvions passer à la partie SAMBA.

IMPLEMENTATION DE SAMBA ET LIEN AVEC LDAP

Un serveur SAMBA est un logiciel utilisé pour partager sur un réseau des répertoires UNIX, en accordant aux utilisateurs un ensemble de privilèges pour s'en servir. Il peut être utilisé en combinaison avec LDAP. La configuration de samba se fait à travers le fichier **/etc/samba/smb.conf**, que nous allons décrire dans la suite de cette partie.

```
#----- Global Settings -----
[global]
# Authentication Ldap
passdb backend = ldapsam:ldap://200.3.3.1
ldap suffix = dc=agence,dc=toulouse
ldap machine suffix = ou=Machines
ldap user suffix = ou=Personnels
ldap group suffix = ou=Fonctions
ldap admin dn = "cn=admin,dc=agence,dc=toulouse"
ldap ssl = off
```

Dans cette première section, introduite par le mot-clé **[global]**, nous avons décrit les paramètres généraux à tous les partages avec samba. Le paramètre « **passdb backend** » permet de spécifier l'adresse IP de notre serveur ldap avec qui samba va être associé. Si on regarde notre DIT fait dans la partie précédente, on voit que cette adresse est celle de notre serveur sur le réseau privé, soit 200.3.3.1. Comme dans le fichier **slapd.conf** édité précédemment, nous précisons le suffixe de notre DIT, à savoir nos deux domain contents. Les paramètres « **ldap machine/user/group suffix** » permettent d'indiquer à quel organization unit va être associé chaque type d'utilisateur LDAP. Avec « **ldap admin dn** », on indique l'administrateur de l'annuaire et enfin on indique à samba si LDAP utilise SSL pour se sécuriser ou non.

```
# workgroup = NT-Domain-Name or Workgroup-Name
workgroup = agencetoulouse

# server string is the equivalent of the NT Description field
server string = Samba AGENCETOULOUSE

# Nom serveur et chemin des pass
interfaces = 200.3.3.1
netbios name = servera
```

Pour continuer, on indique un nom de workgroup, en français un groupe de travail, dans lequel on pourra ajouter les différentes machines de notre réseau. Le server string est une description donnée du serveur de partage. Pour éviter de partager nos répertoires avec le réseau d'interconnexion ou avec l'IEM, on précise dans « interfaces » qu'on souhaite faire le partage uniquement avec le réseau privé (sur l'interface 200.3.3.1). Le « netbios name » est le nom de la machine sur le réseau de partage, ici on lui donne le même nom que notre machine. Enfin, grâce à la ligne suivante, on indique à la configuration que seules les machines du réseau privé (et le localhost) peuvent se connecter au service SAMBA du serveur.

```
hosts allow = 200.3.3. 127.0.0.1
```

Une fois les configurations précédentes effectuées, le paramétrage de samba est terminé pour le moment (on reviendra dans ce fichier à la fin de la partie pour éditer une dernière zone). Il est nécessaire à ce moment-là de faire le lien entre SAMBA et LDAP grâce aux outils fournis par le paquet **smbldap-tools**. Cet utilitaire crée deux fichiers dans le dossier **/etc/smbldap-tools/**, **smbldap.conf** et **smbldap_bind.conf**. Nous avons dans un premier temps configuré le fichier **smbldap.conf** de la façon suivante.

Tout d'abord, on obtient le SID du serveur grâce à la commande « net getlocalsid ». Nous avons à ce moment-là rencontré pas mal de problèmes parce que samba était mal configuré. Mais finalement après avoir retravaillé le script précédent nous avons réussi à relancer samba et ainsi pouvoir exécuter cette commande pour obtenir notre SID. Pour le « sambaDomain », il faut indiquer le groupe de travail de notre réseau.

```
# Put your own SID. To obtain this number do: "net getlocalsid".
# If not defined, parameter is taking from "net getlocalsid" return
SID="S-1-5-21-3990874614-3818723452-870321906"

# Domain name the Samba server is in charged.
# If not defined, parameter is taking from smb.conf configuration file
# Ex: sambaDomain="IDEALX-NT"
sambaDomain="agencetoulouse"
```

Ensuite, on continue dans le paramétrage de ce fichier de configuration en indiquant sur quel port et sur quelle adresse IP le serveur LDAP est installé.

```
# If not defined, parameter is set to "200.3.3.1"
masterLDAP="200.3.3.1"

# Master LDAP port
# If not defined, parameter is set to "389"
masterPort="389"
```

Pour finir avec ce fichier, on indique les chemins vers les différents groupes de notre serveur LDAP.

```
suffix="dc=agence,dc=toulouse"

# Where are stored Users
# Ex: usersdn="ou=Users,dc=IDEALX,dc=ORG"
# Warning: if 'suffix' is not set here, you must set the full dn for usersdn
usersdn="ou=Personnels,${suffix}"

# Where are stored Computers
# Ex: computersdn="ou=Computers,dc=IDEALX,dc=ORG"
# Warning: if 'suffix' is not set here, you must set the full dn for computersdn
computersdn="ou=Machines,${suffix}"

# Where are stored Groups
# Ex: groupsdn="ou=Groups,dc=IDEALX,dc=ORG"
# Warning: if 'suffix' is not set here, you must set the full dn for groupsdn
groupsdn="ou=Fonctions,${suffix}"
```

Pour finir la configuration, il ne nous reste donc plus que le fichier `smbldap_bind.conf`, un fichier très court à paramétrer puisqu'on indique à l'intérieur de celui-ci les identifiants de l'administrateur de LDAP. Puisque le mot de passe est affiché en clair, il est important de modifier les droits pour que seul l'utilisateur propriétaire du fichier, à savoir `root`, puisse le lire et écrire à l'intérieur. On utilise pour cela la commande suivante qui retire tous les droits aux autres utilisateurs que `root`.

```
chmod 600 smbldap_bind.conf
```

```
slaveDN="cn=admin,dc=agence,dc=toulouse"
slavePw="ldap"
masterDN="cn=admin,dc=agence,dc=toulouse"
masterPw="ldap"
```

REPLISSAGE DE L'ANNUAIRE

Le remplissage de l'annuaire se fait en deux temps mais est largement simplifié par l'utilitaire **smbldap**. Pour commencer, il faut utiliser la commande **smbldap-populate** qui va créer à partir de la configuration effectuée de notre LDAP la base de données qui va contenir l'annuaire. Une fois cela fait, il ne nous reste plus qu'à ajouter les utilisateurs et les groupes de notre choix à l'aide des commandes **smbldap-useradd** et **smbldap-groupadd**. A l'ajout, il est possible de fournir un ensemble de paramètres pour commencer à donner des valeurs aux attributs de chaque utilisateur. Cependant, nous avons décidé de faire les choses en deux temps et de plutôt utiliser les commandes **smbldap-passwd** et **smbldap-usermod** à la place, pour être sûrs de ne pas faire d'erreurs avec les nombreux paramètres disponibles. Nous avons ajouté les deux utilisateurs « alexis » et « alexandre » et les deux groupes « informatique » et « banquier » comme nous l'avons spécifié sur le DIT de notre annuaire.

RESSOURCES A PARTAGER

Pour partager un dossier avec Samba, nous sommes retournés dans le script **smb.conf** dans rajouter une zone à la fin.

```
[partage]

comment = "Partage Agence"
path = /home/partage
browseable = yes
writable = yes
```

Pour chaque répertoire à partager sur le serveur, il faut ajouter une section, débutée par le mot clé [nom du répertoire]. On peut ensuite préciser un ensemble de paramètres pour la section, à savoir une phrase de commentaire avec « comment » et le chemin vers le dossier en local avec « path » (nous avons bien sûr créé ce dossier au préalable). Avec les attributs « browseable » et « writable », on indique si les utilisateurs peuvent respectivement lire le répertoire et le modifier. Une fois cela fait, notre répertoire **partage** du dossier **/home** sera partagé avec les utilisateurs de Samba/LDAP.

PROTOCOLE DE VERIFICATION

Pour vérifier l'ensemble des modifications effectuées dans cette partie, nous avons utilisé un ensemble de protocoles. Pour vérifier que nos fichiers de configuration étaient correctement remplis, nous avons simplement relancé les services. Plusieurs fois ce redémarrage a échoué et nous avons étudié les logs grâce à la commande **journalctl -xe** pour savoir d'où venaient les problèmes. Dans un même temps, nous avons utilisé la commande ci-dessous pour pouvoir observer plus facilement l'ensemble des attributs du fichier smb.conf.

```
testparm
```

```
# Global parameters
[global]
    interfaces = 200.3.3.1
    server string = Samba AGENCETOULOUSE
    unix charset = ISO8859-15
    workgroup = AGENCETOULOUSE
    domain master = Yes
    os level = 64
    preferred master = Yes
    ldap admin dn = "cn=admin,dc=agence,dc=toulouse"
    ldap group suffix = ou=Fonctions
    ldap machine suffix = ou=Machines
    ldap passwd sync = yes
    ldap ssl = no
    ldap suffix = dc=agence,dc=toulouse
    ldap user suffix = ou=Personnels
    log file = /var/log/samba/log.%m
    max log size = 1000
    add user script = /usr/local/sbin/smbldap-useradd.pl -w %u
    domain logons = Yes
    logon drive = z:
    printcap name = cups
    name resolve order = wins lmhosts bcast
    time server = Yes
    passdb backend = ldapsam:ldap://200.3.3.1
    passwd chat = *new*password* %n\n *new*password* %n\n *successfully*
    passwd program = /usr/local/sbin/smbldap-passwd.pl -o %u
    security = USER
    username map = /etc/samba/smbusers
    socket options = TCP_NODELAY SO_RCVBUF=8192 SO_SNDBUF=8192
    dns proxy = No
    wins support = Yes
    idmap config * : backend = tdb
    hosts allow = 200.3.3. 127.0.0.1

[partage]
    comment = "Partage Agence"
    path = /home/partage
    read only = No
```

Enfin, nous avons vérifié le fonctionnement par la pratique en utilisant le navigateur JxPlorer et en nous connectant à notre service LDAP. En faisant cela, nous réussissons à accéder à notre arborescence où on retrouve notre DIT, à savoir les OU et à l'intérieur les utilisateurs et groupes.

VI. FILTRAGE

A. MATRICE DE FILTRAGE

Le filtrage est un concept utilisé dans l'administration d'un réseau afin d'en assurer la sécurité en filtrant certains protocoles à travers les ports qu'ils utilisent et en n'en laissant passer que quelques-uns. Afin de préparer la mise en place des règles sur le routeur/filtreur, on construit une matrice de filtrage. Et pour ce faire, il est important de discerner de quelles fonctionnalités on va avoir besoin et dans quels contextes. Dans notre cas, voici les règles que nous devons respecter :

- Le client et le serveur peuvent communiquer librement sur le réseau privé.
- Le réseau privé doit pouvoir envoyer des requêtes DHCP au serveur, et le serveur lui répondre.
- Le serveur doit pouvoir utiliser le port ssh (22) pour communiquer avec un autre groupe de l'interco afin d'implémenter rsync.
- Le serveur et le client doivent pouvoir utiliser Internet, ils ont donc besoin des protocoles http (80), https (443) et domain (53) pour la résolution de noms de domaine.
- Le serveur et le client doivent pouvoir utiliser la commande ping. Il faut donc autoriser le protocole icmp, en entrée et en sortie (pour que les autres groupes puissent aussi faire la commande).
- Depuis l'extérieur (IEM et Interco), on doit pouvoir charger le site Apache. Il faut donc autoriser l'entrée des ports http, https et domain sur le serveur.
- Depuis l'extérieur, on doit pouvoir consulter l'annuaire d'une agence grâce à ldap (389).

Grâce à ces observations, on peut construire la matrice de filtrage suivante :

Vers →	IEM	Interco	Réseau privé	Serveur
IEM			http/s domain icmp	http/s domain ssh icmp ldap
Interco			http/s domain icmp	http/s domain ssh icmp ldap
Réseau privé	http/s domain icmp	http/s domain icmp		DHCP all
Serveur	http/s domain Ssh Icmp ldap	http/s domain Ssh Icmp ldap	DHCP all	

http : Requêtes et réponses – **dhcp** : requête – **dhcp** : réponse

DHCP n'est en réalité pas une fonctionnalité gérée par iptables mais il est intéressant de placer le service dans cette matrice puisque son fonctionnement se base quand même sur une communication entre le serveur et le client. Il n'y aura juste aucune règle dans le script que nous allons décrire dans la partie suivante le concernant. En plus de cela, il faudra veiller à respecter les règles suivantes, imposées par le sujet.

- Supprimer les règles préexistantes (nettoyer la table de filtrage), pour repartir à chaque fois sur de bonnes bases.

- Autoriser l'interface loopback.
- Définir une politique par défaut, pour indiquer ce qu'on fait d'un paquet pour lequel il n'existe aucune règle précise disant ce qu'il faut en faire.
- Bloquer l'accès d'un autre groupe à notre serveur.

B. SCRIPT DE FILTRAGE

Pour implémenter un système de filtrage sous Debian 9, on utilise le module **netfilter** et notamment sa commande **iptables**. Il existe en tout 5 listes différentes de règles, qui permettent de gérer trois tables : la table de filtrage, la table NAT et la table Mangle. Nous avons vu précédemment le remplissage de la table NAT, à l'aide des listes PREROUTING et POSTROUTING, nous allons dans cette partie voir le remplissage de la table de filtrage, avec cette fois-ci les listes INPUT, OUTPUT et FORWARD qui représentent trois contextes différents :

- INPUT correspond aux paquets qui sont à destination du serveur/routeur.
- OUTPUT correspond aux paquets qui sont émis depuis le serveur/routeur.
- FORWARD correspond aux paquets qui sont à destination du réseau privé.

Nous allons maintenant décrire l'implémentation du script « filtre.sh » permettant d'éditer ces différentes tables. Ce script sera disponible dans la partie annexe de ce rapport.

Pour commencer, il est important de penser à activer (comme pour le routage), l'ip-forwarding grâce à la ligne suivante. La raison a été évoquée précédemment lors de l'explication de la mise en place du NAT.

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Ensuite et avant de commencer à manipuler les différentes tables, nous allons vider celles en cours afin de repartir sur des bases propres. Pour cela, nous utilisons les lignes suivantes.

```
#Supprimer les règles pré-existantes
iptables -X
iptables -F
```

L'option -F sans argument derrière permet d'effacer toutes les règles **iptables** existantes une par une. L'option -X sans argument derrière fait globalement la même chose mais conserve les règles prédéfinies.

```
#Autoriser l'interface loopback
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -i lo -j ACCEPT
```

Les lignes précédentes permettent de satisfaire la règle énoncée précédemment consistant à autoriser toutes les communications concernant l'interface loopback (localhost). Pour cela, on utilise les commandes **iptables** ci-dessus en lui spécifiant un certain nombre de paramètres :

- -A : Spécifie la liste qui doit être éditée (entre les 5 citées précédemment). Ici, puisqu'on travaille sur l'interface loopback, elle est incluse au serveur/routeur et les règles s'appliquent donc aux paquets qui entrent et qui sortent de celui-ci (listes INPUT et OUTPUT).
- -i : Spécifie l'interface de laquelle viennent les paquets concernés par la règle. « lo » est le nom de l'interface loopback sur le serveur.
- -j : Spécifie l'action qui doit être effectuée par le filtrage, à savoir accepter ou refuser (ou masquer l'adresse dans le cas de l'implémentation du NAT).

```
#Définir la politique par défaut
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP
```

Avec ces trois lignes, on peut satisfaire une nouvelle règle que nous nous étions fixées, à savoir définir la politique par défaut de notre filtrage. L'option -P permet d'indiquer que la règle qui suit fait office de règle/politique par défaut. Ce qu'on indique avec ces trois commandes, c'est que par défaut aucun paquet ne rentre, ne sort ou ne passe à travers notre réseau. Nous allons par la suite autoriser au cas par cas qui a le droit de venir à partir de la matrice de filtrage construite précédemment.

```
#Bloquer l'accès d'un autre groupe à notre serveur
iptables -A INPUT -s 192.168.0.6 -j DROP
```

Cette ligne de script permet d'interdire l'accès d'un groupe de l'interco à notre serveur comme il l'était demandé dans le sujet. Grâce à l'option -s, on peut isoler les paquets qui viennent d'une source particulière (ici le groupe possédant l'ip interco 192.168.0.6).

```
#Sécuriser le serveur en ouvrant les ports strictement nécessaires
iptables -A INPUT -p tcp --dport 80 -j ACCEPT
iptables -A INPUT -p tcp --dport 443 -j ACCEPT
iptables -A INPUT -p tcp --dport 53 -j ACCEPT
iptables -A INPUT -p udp --dport 53 -j ACCEPT
iptables -A INPUT -p tcp --dport 22 -j ACCEPT
iptables -A INPUT -p tcp --dport 389 -j ACCEPT
iptables -A INPUT -p icmp -j ACCEPT
iptables -A OUTPUT -p tcp --dport 80 -j ACCEPT
iptables -A OUTPUT -p tcp --dport 443 -j ACCEPT
iptables -A OUTPUT -p tcp --dport 53 -j ACCEPT
iptables -A OUTPUT -p udp --dport 53 -j ACCEPT
iptables -A OUTPUT -p tcp --dport 22 -j ACCEPT
iptables -A OUTPUT -p tcp --dport 389 -j ACCEPT
iptables -A OUTPUT -p icmp -j ACCEPT
```

Avec les lignes précédentes, nous avons commencé à définir les règles présentes dans notre matrice de filtrage et qui concernent la communication entre notre serveur et l'extérieur. Toutes les règles qui commencent par « iptables -A INPUT » concernent les règles qui vont de l'IEM ou de l'interco vers le serveur et celles qui commencent par « iptables -A OUTPUT » sont les règles qui vont du serveur vers l'IEM ou l'interco. Deux nouveaux paramètres apparaissent avec ces commandes : --dport qui permet d'indiquer le port des paquets concernés par la règle et -p qui indique le type du protocole qui utilise ce port (tcp, udp ou icmp). Ici, on retrouve

les protocoles http (80), https (443), domain (53), ssh (22) et ldap (389) énumérés dans la matrice de filtrage ainsi que l'autorisation du ping à travers le protocole icmp.

```
#Permettre le dialogue entre le client et le serveur
iptables -A INPUT -s 200.3.3.0/24 -j ACCEPT
iptables -A OUTPUT -d 200.3.3.0/24 -j ACCEPT
```

Grâce à ces lignes, on indique que tous les paquets émis par le serveur et à destination du réseau interne (OUTPUT -d) peuvent sortir et que tous les paquets à destination du serveur et émis par une machine du réseau privé (INPUT -s) sont autorisés. Il s'agit de l'interprétation que nous avons faite de la consigne « permettre le dialogue entre le client et le serveur » du sujet mais selon nous ce n'est pas très prudent de faire cela dans la réalité. En effet, de telles règles font qu'il n'y a concrètement aucun filtrage au niveau des communications entre le réseau privé et le serveur. Si une personne mal intentionnée vient se connecter à ce réseau il aura alors accès au serveur sans avoir à se confronter au pare-feu que constituent les règles de filtrage.

```
#Sécuriser le client en ouvrant les ports strictement nécessaires
iptables -A FORWARD -p tcp --dport 80 -j ACCEPT
iptables -A FORWARD -p tcp --dport 443 -j ACCEPT
iptables -A FORWARD -p tcp --dport 53 -j ACCEPT
iptables -A FORWARD -p udp --dport 53 -j ACCEPT
iptables -A FORWARD -p icmp -j ACCEPT
```

Ces règles correspondent cette fois-ci à celles qui traversent le routeur (FORWARD) pour accéder à une machine du serveur privé ou qui viennent de ce dernier et sortent vers l'interco ou vers l'IEM.

```
#On accepte les réponses
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A OUTPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

C'est avec les règles ci-dessus que nous autorisons le passage des réponses aux différentes requêtes émises en INPUT, en OUTPUT ou en FORWARD. Nous précisons cela grâce aux options « -m state --state ESTABLISHED,RELATED ».

```
#NAT
iptables -t nat -A POSTROUTING -s 200.3.3.0/24 -j MASQUERADE
```

Enfin, nous concluons notre script par la commande permettant d'implémenter le NAT que nous avons vu dans une partie précédente.

C. DEMARRAGE AUTOMATIQUE DU SERVICE

Par défaut, la table de filtrage, la table NAT et la table Mangle sont vidées à chaque redémarrage de la machine. Afin de charger notre script à chaque démarrage de la machine et afin de pouvoir vider ces tables et relancer notre remplissage facilement, nous allons utiliser systemd, un démon conçu pour la gestion des services dans les noyaux Linux. Systemd étant un ajout récent aux systèmes UNIX, il offre toujours de nos jours une

simulation du précédent système qu'était l'utilisation du dossier « init.d ». Nous avons donc utilisé la méthode suivante afin de transformer notre script précédent en service grâce à la simulation de « init.d ».

La première étape est de construire un nouveau script dans le dossier **/etc/init.d** en respectant la même syntaxe que ceux déjà présents nativement. Cette syntaxe est la suivante :

```
#!/bin/sh
### BEGIN INIT INFO
# Provides:          script.sh
### END INIT INFO

if [ -f /bin/setupcon ]; then
  case "$1" in
    stop|status)
      #Actions à effectuer quand on arrête le service
      ;;
    start|force-reload|restart|reload)
      #Actions à effectuer quand on démarre le service
      ;;
    *)
      echo 'Usage: /etc/init.d/script.sh {start|reload|restart|force-reload|stop|status}'
      exit 3
      ;;
  esac
fi
```

Niveau algorithmie, il s'agit tout simplement d'un « case » classique. Quand le premier argument passé au script est « stop » ou « status », on effectue les actions de la première partie, quand il s'agit de « start », « force-reload », « restart » ou « reload » on effectue les actions de la seconde partie et finalement dans tous les autres cas on affiche dans la sortie standard la syntaxe correcte de la commande.

Ainsi, dans la deuxième zone nous avons placé toutes les lignes de script vues précédemment. Ensuite, nous avons écrit de nouvelles lignes de code afin de vider complètement les différentes tables et changer la politique par défaut en « tout accepter ». De cette façon, lorsqu'on appelle le script pour l'arrêter, cela revient à désactiver le pare-feu. Cela donne le script suivant :

```
case "$1" in
  stop|status)
    iptables -X
    iptables -F

    iptables -P INPUT ACCEPT
    iptables -P OUTPUT ACCEPT
    iptables -P FORWARD ACCEPT
  ;;
```

Le script complet est disponible en annexe.

Ce script étant terminé, il faut ensuite se diriger dans le dossier **/etc/rc3.d**. Ce dossier contient des liens symboliques vers tous les services présents dans **/etc/init.d/** qui doivent se lancer lorsque le serveur démarre en mode 3, le mode de lancement par défaut. Dans celui-ci on rajoute donc un lien symbolique vers notre script en utilisant la commande

```
ln -s /etc/init.d/filtre.sh filtre
```

Avec cette commande, on indique qu'on crée un lien symbolique (-s) vers le fichier /etc/init.d/filtre.sh en lui donnant le nom filtre. Une fois cela fait, notre table de filtrage se lance donc à chaque démarrage et il est possible d'arrêter ou de redémarrer le service grâce aux commandes suivantes :

```
➤ systemctl [start|restart|stop] filtre  
➤ service filtre [start|restart|stop]  
➤ /etc/init.d/filtre.sh [start|restart|stop]
```

Comme nous l'avions annoncé dans la partie sur le routage, nous avons fait de même pour que les règles de routage soient relancées à chaque démarrage de la machine. Le fonctionnement est le même, seul le code à l'intérieur du script route.sh change. Ce dernier script est d'ailleurs disponible dans la partie annexes.

VII. CONCLUSION

Pour conclure, nous avons pu voir au cours de ces TP les bases de comment un administrateur système met en place un réseau pour une entreprise ou un groupe d'entreprises. Nous avons pu manipuler un ensemble de concepts relatifs aux communications entre les machines comme le NAT, le routage et la sécurisation du tout avec les règles de filtrage. Nous avons également pu voir un ensemble de services importants pour les entreprises comme les serveurs web avec Apache, mais également les bases de données, les sauvegardes, l'authentification et le partage de ressources. Nous avons également pu mettre en place notre propre serveur DNS, ce qui nous a permis de mieux comprendre son fonctionnement et la façon dont il intervient dans les communications entre les machines.

Au niveau des difficultés rencontrées, le plus difficile a été de manipuler des fichiers de configuration avec une syntaxe parfois assez difficile à comprendre sans forcément avoir eu le cours auparavant. Quand nous étions seuls entre deux TP pour travailler sur un service, il était parfois compliqué de trouver ce qui était juste ou faux et de trouver la bonne méthode. Concernant les fonctionnalités, LDAP et Samba sont les deux services qui nous ont le plus posé de problèmes. A part cela le reste nous a parfois pris du temps mais sans forcément nous poser de gros problèmes.

A. /ETC/NETWORK/INTERFACES

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
# Carte reseau IEM
allow-hotplug enp0s25
iface enp0s25 inet static
    address 172.31.20.110
    netmask 255.255.255.0
    network 172.31.20.0
    broadcast 172.31.20.255

# Carte reseau privé
allow-hotplug enpls0
iface enpls0 inet static
    address 200.3.3.1
    netmask 255.255.255.0
    network 200.3.3.0
    broadcast 200.3.3.255

# Carte reseau interco
allow-hotplug enp3s0
iface enp3s0 inet static
    address 192.168.0.3
    netmask 255.255.255.240
    network 192.168.0.0
    broadcast 192.168.0.15
```

B. /ETC/DHCP/DHCPD.CONF

Dans un souci de gain de place nous n'incluons pas ici les parties du script qui sont en commentaires (et donc par conséquent pas importantes dans cette partie puisque nous ne les avons pas modifiées).

```
# dhcpd.conf
#
# Sample configuration file for ISC dhcpd
#
default-lease-time 14400;
max-lease-time 14400;
option subnet-mask 255.255.255.0;
option broadcast-address 200.3.3.255;
option routers 200.3.3.1;
option domain-name-servers 200.3.3.1;
log-facility local7;
subnet 200.3.3.0 netmask 255.255.255.0 {
    range 200.3.3.2 200.3.3.10;
}

host client{
    hardware ethernet 00:1a:a0:56:ec:bc;
    fixed-address 200.3.3.10;
    option host-name "poste-client";
}
```

C. /ETC/DEFAULT/ISC-DHCP-SERVER

```
Defaults for isc-dhcp-server (sourced by /etc/init.d/isc-dhcp-server)

# Path to dhcpd's config file (default: /etc/dhcp/dhcpd.conf).
#DHCPDv4_CONF=/etc/dhcp/dhcpd.conf
#DHCPDv6_CONF=/etc/dhcp/dhcpd6.conf

# Path to dhcpd's PID file (default: /var/run/dhcpd.pid).
#DHCPDv4_PID=/var/run/dhcpd.pid
#DHCPDv6_PID=/var/run/dhcpd6.pid

# Additional options to start dhcpd with.
# Don't use options -cf or -pf here; use DHCPD_CONF/ DHCPD_PID instead
#OPTIONS=""

# On what interfaces should the DHCP server (dhcpd) serve DHCP requests?
# Separate multiple interfaces with spaces, e.g. "eth0 eth1".
INTERFACESv4="enpl0"
INTERFACESv6=""
```

D. /ETC/BIND/NAMED.CONF.OPTIONS

```

options {
    directory "/var/cache/bind";

    // If there is a firewall between you and nameservers you want
    // to talk to, you may need to fix the firewall to allow multiple
    // ports to talk. See http://www.kb.cert.org/vuls/id/800113

    // If your ISP provided one or more IP addresses for stable
    // nameservers, you probably want to use them as forwarders.
    // Uncomment the following block, and insert the addresses replacing
    // the all-0's placeholder.

    forwarders {
        192.168.0.6; 192.168.0.2; 192.168.0.4; 192.168.0.5; 192.168.0.7; 172.31.21.35;
    };

    //=====
    // If BIND logs error messages about the root key being expired,
    // you will need to update your keys. See https://www.isc.org/bind-keys
    //=====
    dnssec-validation no;

    auth-nxdomain no;      # conform to RFC1035
    listen-on-v6 { any; };
};

```

E. /ETC/BIND/NAMED.CONF.LOCAL

```

//
// Do any local configuration here
//

// Consider adding the 1918 zones here, if they are not used in your
// organization
//include "/etc/bind/zones.rfc1918";

zone "agence.toulouse" {
    type master;
    file "/etc/bind/db.agence.toulouse";
};

zone "informatique.agence.toulouse"{
    type master;
    file "/etc/bind/db.informatique.agence.toulouse";
};

zone "3.3.200.in-addr.arpa" {
    type master;
    file "/etc/bind/db.agence.toulouse.inv";
};

```

F. /ETC/BIND/DB.AGENCE.TOULOUSE

```

;
; BIND data file for agence.toulouse interface
;
$TTL      604800
@         IN      SOA      agence.toulouse. root.agence.toulouse. (
                                2          ; Serial
                                604800     ; Refresh
                                86400      ; Retry
                                2419200    ; Expire
                                604800 )   ; Negative Cache TTL
;
@         IN      NS       agence.toulouse.
@         IN      A        200.3.3.1
servera  IN      A        200.3.3.1

```

G. /ETC/BIND/DB.AGENCE.TOULOUSE.INV

```

;
; BIND reverse data file for local loopback interface
;
$TTL      604800
@         IN      SOA      agence.toulouse. root.agence.toulouse. (
                                1          ; Serial
                                604800     ; Refresh
                                86400      ; Retry
                                2419200    ; Expire
                                604800 )   ; Negative Cache TTL
;
;         IN      NS       servera.agence.toulouse.
1         IN      PTR      servera.agence.toulouse.
;1        IN      PTR      informatique.agence.toulouse.

```

H. /ETC/BIND/DB.INFORMATIQUE.AGENCE.TOULOUSE

```

;
; BIND data file for local loopback interface
;
$TTL      604800
@         IN      SOA      informatique.agence.toulouse. root.informatique.agence.toulouse. (
                                2          ; Serial
                                604800     ; Refresh
                                86400      ; Retry
                                2419200    ; Expire
                                604800 )   ; Negative Cache TTL
;
@         IN      NS       informatique.agence.toulouse.
@         IN      A        200.3.3.1

```

I. /ETC/APACHE2/APACHE2.CONF

Dans un souci de gain de place nous n'incluons pas ici les parties du script qui sont en commentaires ainsi que les parties que nous n'avons pas modifiées.

```
# Sets the default security model of the Apache2 HTTPD server. It does
# not allow access to the root filesystem outside of /usr/share and /var/www.
# The former is used by web applications packaged in Debian,
# the latter may be used for local directories served by the web server. If
# your system is serving content from a sub-directory in /srv you must allow
# access here, or in any related virtual host.
<Directory />
    Options FollowSymLinks
    AllowOverride None
    Require all denied
</Directory>

<Directory /usr/share>
    AllowOverride None
    Require all granted
</Directory>

Alias "/webdev" "/srv/webdev/www"
<Directory /srv/webdev/www>
    Options Indexes FollowSymLinks
    AllowOverride None
    Require all granted
</Directory>
```

J. /ETC/APACHE2/SITES-AVAILABLE/INFORMATIQUE.AGENCE.TOULOUSE

```
<VirtualHost *:80>
    DocumentRoot "/srv/userapache/www"
    ServerName informatique.agence.toulouse
    <Directory /srv/userapache/www>
        Options Indexes FollowSymLinks
        AllowOverride None
        Require all granted
    </Directory>
</VirtualHost>
```

K. /ETC/SUDOERS

```
# This file MUST be edited with the 'visudo' command as root.
#
# Please consider adding local content in /etc/sudoers.d/ instead of
# directly modifying this file.
#
# See the man page for details on how to write a sudoers file.
#
Defaults        env_reset
Defaults        mail_badpass
Defaults        secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"

# Host alias specification

# User alias specification

# Cmnd alias specification

# User privilege specification
root    ALL=(ALL:ALL) ALL
webmaster    ALL=(ALL:ALL) /bin/nano /etc/apache2/apache2.conf,/bin/systemctl restart apache2,/usr/bin/vi /etc/apache2/apache2.conf,/usr/sbin/service apache2 restart

# Allow members of group sudo to execute any command
%sudo    ALL=(ALL:ALL) ALL

# See sudoers(5) for more information on "include" directives:

#includedir /etc/sudoers.d
```

L. /ETC/SCRIPTS/POSTGRESBACKUP.SH


```
#!/bin/bash

DATE=`date -I`

# Dossier où sauvegarder les backups

BACKUP_DIR="/backup/postgres"

# Sauvegarde

mkdir $BACKUP_DIR/$DATE
sudo -u postgres pg_dumpall -c > $BACKUP_DIR/$DATE/backup.sql
```

M. /ETC/SCRIPTS/MYSQLBACKUP.SH

```
#!/bin/bash

# Configuration de base : datestamp e.g. YYYYMMDD

DATE=`date -I`

# Dossier où sauvegarder les backups

BACKUP_DIR="/backup/mysql"

# Sauvegarde

mkdir $BACKUP_DIR/$DATE
mysqldump -u root -C --all-databases > $BACKUP_DIR/$DATE/backup.sql.tgz
```

N. /ETC/SCRIPTS/RSYNC.SH

```
#!/bin/bash

date=`date +%Y-%m-%d`
rsync -ae ssh /home/alex etudiant@192.168.0.2:/home/alexrsync/$date
```

O. /ETC/CRONTAB

```
#!/etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab`
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user  command
17 * * * * root    cd / && run-parts --report /etc/cron.hourly
25 6 * * * root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6 * * 7 root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6 1 * * root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
0 15 * * * root    /etc/scripts/mysqlbackup.sh
0 15 * * * root    /etc/scripts/postgresbackup.sh
0 15 * * * root    /etc/scripts/rsync.sh
#
```

P. /ETC/LDAP/SLAPD.CONF

Dans un souci de gain de place nous n'incluons pas ici les parties du script qui sont en commentaires ainsi que les parties que nous n'avons pas modifiées.

```
##### slapd type #####
# Global Directives:

# Schema and objectClass definitions
include /etc/ldap/schema/core.schema
include /etc/ldap/schema/cosine.schema
include /etc/ldap/schema/nis.schema
include /etc/ldap/schema/inetorgperson.schema
include /etc/ldap/schema/samba.schema

# The base of your directory in database #1
suffix "dc=agence,dc=toulouse"
checkpoint 512 60

# rootdn directive for specifying a superuser on the database. This is needed
# for syncrepl.
# rootpw a crypter en dessous
rootdn "cn=admin,dc=agence,dc=toulouse"
rootpw {SSHA}8iEY0HAaq37Y2f/w9dyVr7UX6/EZiktr

# Where the database file are physically stored for database #1
directory "/ldap"

# Optimisation par creation d index sur attributs
index objectClass,uidNumber,gidNumber eq
index cn,sn,uid,displayName pres,sub,eq
index memberUid,mail,givenname eq,subinitial
# index sambaSID,sambaPrimaryGroupSID,sambaDomainName eq

# Save the time that the entry gets modified, for database #1
lastmod off

# Where to store the replica logs for database #1
# relogfile /var/lib/ldap/repllog

# Autoriser l'accès à l'annuaire
access to dn.base="" by * read

# Sécurisation du DIT
## Par défaut l'admin peut tout faire et on peut lire le LDAP
access to *
    by dn="cn=admin,dc=agence,dc=toulouse" write
    by * read
```

Q. /ETC/SAMBA/SMB.CONF

Dans un souci de gain de place nous n'incluons pas ici les parties du script qui sont en commentaires ainsi que les parties que nous n'avons pas modifiées.

```
#
#===== Global Settings =====
[global]
# Authentification Ldap
passdb backend = ldapsam:ldap://200.3.3.1
ldap suffix = dc=agence,dc=toulouse
ldap machine suffix = ou=Machines
ldap user suffix = ou=Personnels
ldap group suffix = ou=Fonctions
ldap admin dn = "cn=admin,dc=agence,dc=toulouse"
ldap ssl = off

#synchro auto de tous les pass d'un utilisateur
ldap passwd sync = Yes
enable privileges = Yes

# Table d'encodage des caractères Idem Windows
Unix Charset = ISO8859-15

# workgroup = NT-Domain-Name or Workgroup-Name
workgroup = agencetoulouse

# server string is the equivalent of the NT Description field
server string = Samba AGENCETOULOUSE

# Nom serveur et chemin des pass
interfaces = 200.3.3.1
netbios name = servera
username map = /etc/samba/smbusers
# add user script = /usr/sbin/useradd -n -g machines -d /dev/null -s /bin/false %m$
# add machine script = cpu useradd -o %m$;sleep 20;smbpasswd -m -a %m$;sleep 20
# Procedure d'ajout des machines via smbldap-tools
add user script = /usr/local/sbin/smbldap-useradd.pl -w %u
# domain admin group = " @\"Domain Admins\" "
```

```
# This option is important for security. It allows you to restrict
# connections to machines which are on your local network. The
# following example restricts access to two C class networks and
# the "loopback" interface. For more examples of the syntax see
# the smb.conf man page
hosts allow = 200.3.3. 127.0.0.1
```

```
#===== Share Definitions =====
#[home]
# comment = Repertoire utilisateur
# path = /home
# browseable = yes
# writable = yes
# directory mode = 707
# create mode = 706

[partage]

comment = "Partage Agence"
path = /home/partage
browseable = yes
writable = yes
```

R. /ETC/NSSWITCH.CONF

```

# /etc/nsswitch.conf
#
# Example configuration of GNU Name Service Switch functionality.
# If you have the `glibc-doc-reference' and `info' packages installed, try:
# `info libc "Name Service Switch"' for information about this file.

passwd:         compat ldap
group:          compat ldap
shadow:         compat
gshadow:        files

hosts:          files mdns4_minimal [NOTFOUND=return] dns
networks:       files

protocols:      db files
services:       db files
ethers:         db files
rpc:            db files

netgroup:       nis

```

S. /ETC/SMBLDAP-TOOLS/SMBLDAP.CONF

Dans un souci de gain de place nous n'incluons pas ici les parties du script qui sont en commentaires ainsi que les parties que nous n'avons pas modifiées.

```

#####
#
# General Configuration
#
#####

# Put your own SID. To obtain this number do: "net getlocalsid".
# If not defined, parameter is taking from "net getlocalsid" return
SID="S-1-5-21-3990874614-3818723452-870321906"

# Domain name the Samba server is in charged.
# If not defined, parameter is taking from smb.conf configuration file
# Ex: sambaDomain="IDEALX-NT"
sambaDomain="agencetoulouse"
#####

```

```

# Master LDAP server: needed for write operations
# Ex: masterLDAP=127.0.0.1
# If not defined, parameter is set to "127.0.0.1"
masterLDAP="200.3.3.1"

# Master LDAP port
# If not defined, parameter is set to "389"
masterPort="389"

```

```

# LDAP Suffix
# Ex: suffix=dc=IDEALX,dc=ORG
suffix="dc=agence,dc=toulouse"

# Where are stored Users
# Ex: usersdn="ou=Users,dc=IDEALX,dc=ORG"
# Warning: if 'suffix' is not set here, you must set the full dn for usersdn
usersdn="ou=Personnels,${suffix}"

# Where are stored Computers
# Ex: computersdn="ou=Computers,dc=IDEALX,dc=ORG"
# Warning: if 'suffix' is not set here, you must set the full dn for computersdn
computersdn="ou=Machines,${suffix}"

# Where are stored Groups
# Ex: groupsdn="ou=Groups,dc=IDEALX,dc=ORG"
# Warning: if 'suffix' is not set here, you must set the full dn for groupsdn
groupsdn="ou=Fonctions,${suffix}"

```

T. /ETC/SMBLDAP-TOOLS/SMBLDAP_BIND.CONF

```
#####
# Credential Configuration #
#####
# Notes: you can specify two differents configuration if you use a
# master ldap for writing access and a slave ldap server for reading access
# By default, we will use the same DN (so it will work for standard Samba
# release)
# Donne les pass pour modifier le Ldap avec les smbtools
slaveDN="cn=admin,dc=agence,dc=toulouse"
slavePw="ldap"
masterDN="cn=admin,dc=agence,dc=toulouse"
masterPw="ldap"
```

U. /ETC/INIT.D/FILTRE.SH

```
#!/bin/sh
### BEGIN INIT INFO
# Provides:          route.sh
### END INIT INFO

if [ -f /bin/setupcon ]; then
    case "$1" in
        stop|status)
            iptables -X
            iptables -F

            iptables -P INPUT ACCEPT
            iptables -P OUTPUT ACCEPT
            iptables -P FORWARD ACCEPT

            ;;
        start|force-reload|restart|reload)
            echo 1 > /proc/sys/net/ipv4/ip_forward

            #Supprimer les règles pré-existantes
            iptables -X
            iptables -F

            #Autoriser l'interface loopback
            iptables -A INPUT -i lo -j ACCEPT
            iptables -A OUTPUT -i lo -j ACCEPT

            #Définir la politique par défaut
            iptables -P INPUT DROP
            iptables -P OUTPUT DROP
            iptables -P FORWARD DROP

            #Bloquer l'accès d un autre groupe à notre serveur
            iptables -A INPUT -s 192.168.0.2 -j DROP

            #Sécuriser le serveur en ouvrant les ports strictement nécessaires
            iptables -A INPUT -p tcp --dport 80 -j ACCEPT
            iptables -A INPUT -p tcp --dport 443 -j ACCEPT
            iptables -A INPUT -p tcp --dport 53 -j ACCEPT
            iptables -A INPUT -p udp --dport 53 -j ACCEPT
            iptables -A INPUT -p tcp --dport 22 -j ACCEPT
            iptables -A INPUT -p tcp --dport 389 -j ACCEPT
            iptables -A INPUT -p icmp -j ACCEPT
            iptables -A OUTPUT -p tcp --dport 80 -j ACCEPT
            iptables -A OUTPUT -p tcp --dport 443 -j ACCEPT
            iptables -A OUTPUT -p tcp --dport 53 -j ACCEPT
            iptables -A OUTPUT -p udp --dport 53 -j ACCEPT
            iptables -A OUTPUT -p tcp --dport 22 -j ACCEPT
            iptables -A OUTPUT -p tcp --dport 389 -j ACCEPT
            iptables -A OUTPUT -p icmp -j ACCEPT

            #Permettre le dialogue entre le client et le serveur
            iptables -A INPUT -s 200.3.3.0/24 -j ACCEPT
            iptables -A OUTPUT -d 200.3.3.0/24 -j ACCEPT

            #Sécuriser le client en ouvrant les ports strictement nécessaires
            iptables -A FORWARD -p tcp --dport 80 -j ACCEPT
            iptables -A FORWARD -p tcp --dport 443 -j ACCEPT
            iptables -A FORWARD -p tcp --dport 53 -j ACCEPT
```

```

#Bloquer l'accès d un autre groupe à notre serveur
iptables -A INPUT -s 192.168.0.6 -j DROP

#Sécuriser le serveur en ouvrant les ports strictement nécessaires
iptables -A INPUT -p tcp --dport 80 -j ACCEPT
iptables -A INPUT -p tcp --dport 443 -j ACCEPT
iptables -A INPUT -p tcp --dport 53 -j ACCEPT
iptables -A INPUT -p udp --dport 53 -j ACCEPT
iptables -A INPUT -p tcp --dport 22 -j ACCEPT
iptables -A INPUT -p tcp --dport 389 -j ACCEPT
iptables -A INPUT -p icmp -j ACCEPT
iptables -A OUTPUT -p tcp --dport 80 -j ACCEPT
iptables -A OUTPUT -p tcp --dport 443 -j ACCEPT
iptables -A OUTPUT -p tcp --dport 53 -j ACCEPT
iptables -A OUTPUT -p udp --dport 53 -j ACCEPT
iptables -A OUTPUT -p tcp --dport 22 -j ACCEPT
iptables -A OUTPUT -p tcp --dport 389 -j ACCEPT
iptables -A OUTPUT -p icmp -j ACCEPT

#Permettre le dialogue entre le client et le serveur
iptables -A INPUT -s 200.3.3.0/24 -j ACCEPT
iptables -A OUTPUT -d 200.3.3.0/24 -j ACCEPT

#Sécuriser le client en ouvrant les ports strictement nécessaires
iptables -A FORWARD -p tcp --dport 80 -j ACCEPT
iptables -A FORWARD -p tcp --dport 443 -j ACCEPT
iptables -A FORWARD -p tcp --dport 53 -j ACCEPT
iptables -A FORWARD -p udp --dport 53 -j ACCEPT
iptables -A FORWARD -p icmp -j ACCEPT

#On accepte les réponses
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A OUTPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

#NAT
iptables -t nat -A POSTROUTING -s 200.3.3.0/24 -j MASQUERADE

;;
*)
    echo 'Usage: /etc/init.d/route.sh {start|reload|restart|force-reload|stop|status}'
    exit 3
;;
esac
fi

```

V. /ETC/INIT.D/ROUTE.SH

```

#!/bin/sh
### BEGIN INIT INFO
# Provides:          route.sh
### END INIT INFO

if [ -f /bin/setupcon ]; then
    case "$1" in
        stop|status)
            route del -net 200.3.6.0 netmask 255.255.255.0 gw 192.168.0.6
            route del -net 10.0.2.0 netmask 255.255.255.0 gw 192.168.0.2
            route del -net 10.0.4.0 netmask 255.255.255.0 gw 192.168.0.4
            route del -net 10.0.5.0 netmask 255.255.255.0 gw 192.168.0.5
            route del -net 10.0.7.0 netmask 255.255.255.0 gw 192.168.0.7
            route del default gw 172.31.20.1

            ;;
        start|force-reload|restart|reload)
            echo 1 > /proc/sys/net/ipv4/ip_forward

            route add -net 200.3.6.0 netmask 255.255.255.0 gw 192.168.0.6
            route add -net 10.0.2.0 netmask 255.255.255.0 gw 192.168.0.2
            route add -net 10.0.4.0 netmask 255.255.255.0 gw 192.168.0.4
            route add -net 10.0.5.0 netmask 255.255.255.0 gw 192.168.0.5
            route add -net 10.0.7.0 netmask 255.255.255.0 gw 192.168.0.7
            route add default gw 172.31.20.1

            ;;
        *)
            echo 'Usage: /etc/init.d/route.sh {start|reload|restart|force-reload|stop|status}'
            exit 3
            ;;
    esac
fi

```