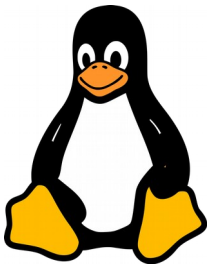
Module administration réseau GNU/Linux

Introduction au stack LAMP



Linux **A**pache **M**ariaDB / **M**ySQL **P**hp

REFERENTIEL EMPLOI ACTIVITES COMPETENCES DU TITRE PROFESSIONNEL TSSR	
Activité type	**Maintenir, exploiter et sécuriser une infrastructure centralisée**
Compétences professionnelles	Maintenir et exploiter un serveur Linux
	Maintenir et exploiter un environnement virtualisé
Activité type	**Maintenir et exploiter une infrastructure distribuée et contribuer à sa sécurisation**
Compétences professionnelles	Automatiser les tâches à l'aide de scripts
	Superviser l'infrastructure

Sommaire

Pré-requis

- Manipuler les fichiers en ligne de commande
- Manipuler les utilitaires de dépannage réseau
- Connaissance des principaux mécanismes et du vocabulaire liés au HTTP/ HTTPS/DNS

Objectifs

À la fin de cette séance, vous serez en mesure :

- de décrire le fonctionnement et de configurer un serveur HTTP/HTTPS
- Être capable de déployer des applications web reposant sur une architecture 3-tiers.

A. Généralités

LAMP est un acronyme désignant un ensemble de logiciels libres permettant de construire des serveurs de sites web. L'acronyme original se réfère aux logiciels suivants :

- **L**inux, le système d'exploitation (GNU/Linux) ;
- **A**pache, le serveur Web ;
- **M**ySQL ou MariaDB », le serveur de base de données (MariaDB est devenue le SGDB par défaut des dernières distributions GNU/Linux) ;
- À l'origine, « **P**HP », « **P**erl » ou « **P**ython », les langages de script mais qui de nos jours est principalement rattaché à **PHP**.

Même si les auteurs de chacun de ces programmes ne se sont pas coordonnés pour construire des plates-formes LAMP, cette combinaison de logiciels s'est popularisée du fait du faible coût de l'ensemble et de la présence de tous ces composants dans la plupart des distributions GNU/Linux.

Les rôles de ces quatre composants sont les suivants :

- **Linux** assure l'attribution des ressources aux autres composants (Rôle d'un Système d'exploitation ou OS pour Operating System) ;
- **Apache** est le serveur HTTP(S) « frontal » : il est « devant » tous les autres et répond directement aux requêtes du client web (navigateur) ;
- **MariaDB** (ou MySQL) est un système de gestion de bases de données (SGBD). Il permet de stocker et d'organiser des données ;
- **PHP** permet la génération de pages web dynamiques et la communication avec le serveur MySQL.

Les composants peuvent être situés :

- sur une même machine ;
- sur deux machines, généralement Apache et le langage de script d'un côté et MySQL de l'autre. On parle d’architecture 3-tiers : le client pour la couche présentation (affichage), Apache2/Php pour la partie traitement et MariaDB pour la partie accès aux données ;
- sur de nombreuses machines pour assurer la haute disponibilité (répartition de charge et/ou failover).

B. Apache2

1. Présentation

Apache 2 est le serveur HTTP le plus utilisé actuellement sur les serveurs Web (46% du marché suivi de près par Nginx à 20%. IIS est à 9%). Sa configuration et sa flexibilité en font un serveur incontournable.

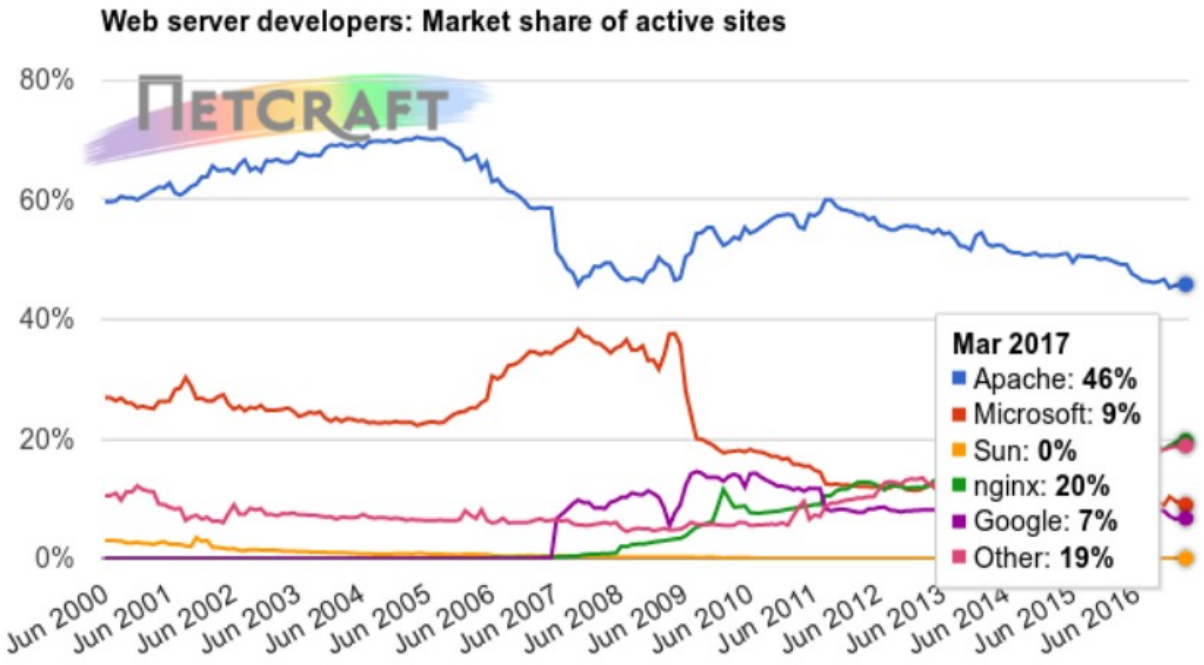


Figure 1. Statistiques NetCraft : Market Share of Active Sites

Le serveur Apache est très présent sur l'Internet, puisqu'il représente encore environ 50% des parts de marché pour l'ensemble des sites actifs.

Lorsqu'un serveur Apache reçoit des requêtes, il peut les redistribuer à des processus fils. La configuration permet de lancer des processus de manière anticipée et d'adapter dynamiquement ce nombre en fonction de la charge.

Apache est modulaire. Chaque module permet d'ajouter des fonctionnalités au serveur. Le module le plus connu est probablement celui gérant le langage PHP, « mod_php ». Chaque module s'ajoute via les fichiers de configuration, et il n'y a même pas besoin de relancer le serveur Apache : on lui donne juste l'ordre de relire sa configuration.

Apache peut gérer plusieurs sites Web en même temps, ayant chacun leur nom, à l'aide des hôtes virtuels.

Apache2 est disponible sur les distributions GNU/Linux les plus répandues mais son implémentation diffère selon celles-ci, au niveau du nom des fichiers de configuration et du nom des paquets.

Notre stack LAMP se basera sur Debian version 9.

2. Installation

L'installation d'Apache2 sous Debian se fait en une ligne de commande :

```
# apt-get update
# apt-get install apache2
```

Si vous saisissez en URL l'adresse IP de votre serveur web, vous verrez alors apparaître la page par défaut fournie avec le paquet (souvent personnalisée selon la distribution Linux utilisée).

Les fichiers de configuration se situent dans « ***/etc/apache2/*** » :

```
/etc/apache2/
|-- apache2.conf
|-- conf-available
|   |-- charset.conf
|   |-- ...
|-- conf-enabled
|   |-- charset.conf -> ../conf-available/charset.conf
|   |-- ...
|-- envvars
|-- magic
|-- mods-available
|   |-- access_compat.load
|   |-- ...
|-- mods-enabled
|   |-- access_compat.load -> ../mods-available/access_compat.load
|   |-- ...
|-- ports.conf
|-- sites-available
|   |-- 000-default.conf
|   `-- default-ssl.conf
`-- sites-enabled
    `-- 000-default.conf -> ../sites-available/000-default.conf
```

Notons que les ressources visées (les pages web,...) seront stockées dans des répertoires placés par défaut dans « **/var/www** ». La page par défaut ouverte précédemment est elle-même stockée dans « /var/www/html » (cette page est propre à Debian).

3. Les fichiers de configuration

Apache2 doit être configuré à l'aide de directives décrites dans des fichiers texte de configuration. Ces directives sont réparties entre les fichiers et dossiers suivants (NDR : **ces informations ne sont pas à retenir par cœur et cette organisation des fichiers de conf est propre à Debian**) :

- **apache2.conf** est le fichier de configuration principal. Il contient les paramètres globaux d'Apache2. Ce fichier est très rarement modifié. Par le biais d'instructions Include, ce fichier en appelle d'autres, de manière à répartir cette configuration.

- **conf-available/** : ce répertoire contient contient les éléments de configurations complémentaires ; en général des configurations globales au serveur que l'on peut modifier.

- **conf-enabled/** : conserve les liens symboliques dans /etc/apache2/conf-available. Quand un fichier de configuration est pointé, il sera activé au prochain démarrage de apache2.

- **envvars** : fichier où les variables d'environnement d'Apache2 sont définies.

- **mods-available/*.load et mods-available/*.conf** : ce répertoire contient les fichiers de configuration qui permettent de charger les modules et de les paramétrer. Certains modules peuvent ne pas avoir de fichier de configuration.

- **mods-enabled/** : contient les liens symboliques vers les fichiers de /etc/apache2/mods-available (.conf et .load). Lorsqu'un lien symbolique vers un module de configuration est créé, il sera activé au prochain redémarrage d'Apache2.

- **ports.conf** : héberge les directives déterminant les ports TCP sur lesquels Apache2 est en écoute (80 et 443).

- **sites-available/** : ce dossier contient des fichiers de configuration pour les serveurs virtuels d'Apache2. Les serveurs virtuels permettent de configurer Apache2 pour plusieurs sites ayant différentes configurations.

- **sites-enabled/** : comme mods-enabled, sites-enabled contient des liens symboliques vers le dossier /etc/apache2/sites-available. De la même manière que lorsqu'un lien symbolique vers un fichier de configuration de sites-available est créé, le site ainsi configuré sera activé au prochain redémarrage d'Apache2.

- **magic** : instructions pour déterminer le type MIME (Internet media type → https://fr.wikipedia.org/wiki/Type_MIME) à partir des premiers bytes d'un fichier.

Comme vous pouvez le constater (voir avec la commande tree) la configuration d'Apache2 est plutôt fournie. Ce cours est une introduction à Apache2 afin d'illustrer par la pratique différents concepts. Apache2 comme beaucoup de services réseaux vaut à lui tout seul une formation entière.

Nous allons donc principalement nous concentrer sur la configuration des sites, à destination d'un **intranet**, et plus précisément les **hôtes virtuels** et nous garderons la configuration globale d'Apache2 fournie par défaut.

4. Présentation des hôtes virtuels (virtual hosts)

Très souvent, un serveur matériel est surdimensionné par rapport aux besoin d'un seul site web.

Un concept simple permet d'obtenir une solution parfaitement adaptée : les hôtes virtuels.

Lorsqu'on saisit une URL d'un site dans un navigateur, une requête DNS transforme cette adresse textutelle en adresse IP. Cette adresse IP est celle de la machine où réside le serveur Apache.
Cette association FQDN/IP n'est pas nécessairement exclusive; un nombre illimité de FQDN peuvent être associés à une seule et même adresse IP.

À partir de là, lorsque votre navigateur web interroge un serveur, il précise le FQDN qu'il rechercher et le serveur HTTP - Apache en l'occurrence - est capable de desservir des données différentes selon le serveur demandé. L'hôte physique est le même, mais il s'agit virtuellement, pour les navigateurs (clients) qu s'y connectent, de serveurs différents : dans la terminologie d'Apache, on parle alors d'**hôtes virtuels (virtual host ou bien vhost)**.

On peut héberger n sites web ayant des URL différentes, mais avec une seule interface réseau et une seule adresse IP.

Par exemple, votre serveur web héberge deux sites web dont les url sont : http://dsi.tssr.lan et http://compta.tssr.lan, et a comme adresse IP: 192.168.1.205.

D'un point de vue DNS, les noms d'hôte virtuel « dsi » et « compta » seront des alias du nom d'hôte réel de la machine hébergeant les sites en question, par exemple « srvweb ».

Les requêtes pointeront donc vers « srvweb » au niveau de la couche réseau, puis, au niveau de la couche transport, redirigées par exemple sur le port 80 (ou autre port sur lequel Apache sera en écoute) et lorsque Apache2 recevra la requête au niveau de la couche applicative, il associera le vhost contenue dans l'URL aux ressources visées.

Il y a trois possibilités pour configurer les hôtes virtuels avec Apache2 :

- **Hôtes virtuels basés sur le nom**, comme nous venons de le voir ci-dessus.

- **Hôtes virtuels basés sur le port**, où les ports permettront de différencier les ressources auxquelles nous voulons accéder. Apache est en effet capable d'écouter sur plusieurs ports en même temps (déjà vu avec le port 80 et 443). Ex. : http://192.168.1.205:8282 nous dirige vers le site web de la DSI et http://192.168.1.205:8383 nous dirige vers le site web de la comptabilité.

- **Hôtes virtuels basés sur l'IP**, dans le cadre de cette méthode, le serveur est soit doté de plusieurs interfaces réseau , soit de plusieurs adresses IP associées à une seule interface réseau. Dans ce dernier cas, on parlera d'**IP aliasing**. Les systèmes Linux permettent de mettre facilement en œuvre cette fonctionnalité.

Étant donné que nous avons configuré précédemment le service DNS, nous utiliserons la configuration des hôtes virtuels basés sur le nom et avouons-le cette méthode est bien plus « user-friendly » pour les utilisateurs que d'utiliser des ports ou des IP.

5. Configuration des hôtes virtuels

5.1 Préparation de la maquette

Les sites web que nous allons mettre en place n'ont qu'une **portée locale**. Il est donc inutile de les mettre dans une **DMZ**. Cependant nous pourrions tout de même l'isoler dans un VLAN dédié. La segmentation réseau permet en effet de limiter la casse en cas de compromission. Vu que vous découvrez pour la plupart Apache2, nous allons installer notre serveur sur le LAN virtuel pour des raisons de commodité.

Le serveur web tournera sous **Debian 9 core** sur l'infrastructure virtuelle que vous utilisez depuis le module réseau et se nommera « **srvweb** ». Vous utiliserez donc vos serveurs DNS créés précédemment.

Vous êtes libre de l'adressage IP. N'oubliez pas de vous appuyer sur un tableau d'adressage IP (aussi créé lors du module précédent).

Nous allons créer deux vhosts : « **glpi.tssr.lan** » et « **rss.tssr.lan** ».

Chacun hébergera à terme une application PHP.

5.2 Fichier de configuration des hôtes virtuels

Les fichiers de configuration sont placés dans « **/etc/apache2/sites-available** ». C'est dans ce répertoire que nous créons dans un premier temps nos fichiers de configuration.

Les fichiers de configurations des hôtes virtuels actifs sont situés quant à eux dans « **/etc/apache2/sites-enabled** ». Ce répertoire contient les **liens**

symboliques vers les fichiers de « **/etc/apache2/site-available** ».

Lorsqu'un lien symbolique vers un module de configuration est créé, il sera activé au prochain redémarrage d'Apache2.

Nous allons créer deux fichiers correspondant à nos deux vhosts :

```
# touch /etc/apache2/sites-available/001-{glpi,rss}.conf
```

En saisissant la commande « ls » sur le répertoire ci-dessus, vous noterez la présence des fichiers « 000-default.conf » et « default-ssl.conf »

Le fichier nommé « 000-default.conf » n'est pas spécial du point de vue des paramètres. Le fichier est simplement nommé de cette façon afin qu'il soit inclus en premier dans la configuration d'Apache2 (« 000- » sera évalué en priorité). Lors de l'installation d'Apache2 sous Debian, ce fichier est activé par défaut.

Si quelqu'un se connecte à votre serveur en saisissant directement dans l'URL l'adresse IP, ou un nom qui n'est pas couvert par les hôtes virtuels définis (le nom d'hôte réel de le machine par exemple), cet fichier définissant donc un hôte virtuel par défaut sera utilisé pour répondre à la requête.

Pour finir, ne placez pas une application critique dans ce fichier puisque on pourrait y accéder sans même connaître l'URL.

Pourquoi ne pas définir tous les vhosts dans un seul et même fichier ?

La réponse réside dans le fait que vous ne connaissez pas l'évolution future de votre serveur HTTP. Aujourd'hui vous hébergez 2 sites et d'ici quelques années cela pourrait être une quinzaine de sites voire bien plus surtout avec la multiplication des applications dites web.

Cela aurait pour conséquence de rendre difficilement lisible votre fichier (sans parler des différentes directives qui s'ajouteraient suivant les applications web).

Mais là aussi pas d'obligation à respecter. Ça sera à vous d'analyser et de faire un choix.

Passons à la configuration du vhost « glpi.tssr.lan ». Éditez le fichier créé précédemment et mettez ceci (**faites de même pour « rss.tssr.lan**) :

```
<VirtualHost *:80>
        ServerAdmin votre-mail@tsgeri.lan
        ServerName glpi.tsgeri.lan

        DocumentRoot /var/www/glpi

        <Directory /var/www/glpi>
                Options -Indexes -FollowSymLinks -MultiViews
                AllowOverride None
                Order allow,deny
                Allow from all
        </Directory>

        ErrorLog /var/log/apache2/error.log

        # Possible values include: debug, info, notice, warn, error, crit,
        # alert, emerg.
        LogLevel warn

        CustomLog /var/log/apache2/access.log combined

</VirtualHost>
```

Quelques explications s'imposent : la configuration d'Apache2 utilise des balises « **<...>** » (ouverture du bloc) et « **</...>** » (fermeture du bloc), comme HTML. Dans notre cas nous créons un bloc « **VirtualHost** » qui contiendra les paramétrages relatifs à celui-ci.
Cela permet de délimiter la configuration de deux vhosts au sein d'un même fichier.
On verra par la pratique que ces blocs permettront de délimiter la configuration d'un même vhost mais sur deux ports différents avec la mise en place du HTTPS.

L'ouverture du bloc avec « ***:80** » correspond à l'**interface physique** et le **port**.
Si le serveur avait plusieurs adresses IP, on remplacerait alors le caractère « * » par l'adresse IP en écoute.
Si vous souhaitez mettre un autre port en écoute, 8080 par exemple, il faudra donc modifier le port du vhost mais aussi ajouter la directive « **Listen 8080** » dans le fichier « **/etc/apache2/ports.conf** » (spécifique à Debian et dérivées).

La directive **ServerName** permet d'identifier l'hôte virtuel. Elle correspond à l'adresse contenue dans l'URL à savoir dans notre « glpi.tssr.lan ».

Le paramètre **DocumentRoot** permet d'indiquer à Apache2 les ressources liés au vhost. Il est défini majoritairement dans le contexte d'un hôte virtuel (directement dans le bloc <VirtualHost></VirtualHost>).

Au sein de notre hôte virtuel, nous pouvons définir des droits selon les répertoires grâce au bloc **"<Directory /chemin/repertoire></Directory>"**. Dans ce cas, le **contexte est limité** (et donc les droits) au répertoire en question et à ses sous-répertoires.

Abordons les principales directives que vous rencontrerez au sein de ce dernier bloc (soit plus précisément, de ce contexte).

La directive « **Options** » permet de spécifier les options à appliquer quant au traitement des requêtes à destination du contexte courant. Cette directive peut se trouver directement dans le contexte de l'hôte virtuel voire dans le contexte global (configuration du serveur). Voici les principaux paramètres qu'elle peut prendre :

None	Désactive toutes les options
All	Active toutes les options sauf Multiviews.
Indexes	Si l'index du répertoire (index.html ou index.php) est manquant, cela autorise le serveur à lister le contenu du répertoire. L'utilisateur verra donc dans son navigateur la liste de tous les fichiers contenu dans le répertoire. **Option à proscrire** dans le cas d'application web (sauf si prévu par l'application en question) mais qui peut s'avérer pratique lors de partage de fichiers à sens unique (en téléchargement uniquement).
FollowSymLinks	Autorise à suivre les liens symboliques. On la désactive dans la majeure partie des cas.
Multiviews	Autorise les vues multiples suivant un contexte. Par exemple, cela permet d'afficher les pages dans la langue préférée de l'utilisateur. Il s'agit plus globalement pour le serveur de pouvoir choisir la façon la plus adaptée de présenter une contenu en fonctions des préférences du navigateur. Aujourd'hui cette fonctionnalité est gérée plus souvent par l'application web, on peut donc la désactiver.

La directive « **AllowOverride** » est liée au fichier « **.htaccess** » et n'est acceptée que dans le contexte de répertoires (notez bien le « . » devant, c'est un fichier caché).
Ce fichier permet de spécifier un contexte (prévalant sur la configuration système) qui sera limité au sein du répertoire dans lequel il est placé et les sous-répertoires.
Ce fichier est très utile dans le cas où les webmestres n'ont pas accès aux fichiers de configuration (chez les hébergeurs par exemple).
Le fichier « **.htaccess** » est aussi souvent utilisé par des applications web, ce qui évitera à l'administrateur d'alourdir la configuration et la maintenance du fichier de configuration du vhost.

La directive « **AllowOverride** » prend différentes options :
→ **None** : bloque toute modification de la configuration et désactive ce mécanisme dans son ensemble.
→ **All** : autorise tous les cas d'utilisation de ces fichiers.

Les options « **Order** », « **Allow** » et « **Deny** » permettent de définir les autorisations d'accès, selon les éléments fournis par les requêtes (IP, nom d'hôte, user-agent,...) :

→ **Allow** définit les clients autorisés.

→ **Deny** définit les clients refusés.

→ **Order** définit l'ordre dans lequel évaluer ces deux directives.

Les possibilités fournies par Allow et Deny sont complexes. La plus simple de toute est le filtrage par IP. La syntaxe sera alors :
→ Allow from Adresse_IP
→ Deny from Adresse_IP

« **All** » permet de refuser ou d'accepter tous les clients.

« **ErrorLog /var/log/apache2/error.log** » indique où seront stockés les journaux contenant les erreurs.

« **LogLevel** » : permet de définir le niveau de verbosité. Sur un serveur en production, on le placera entre « info » et « crit ». Sur un serveur ou une application web en test, on peut le mettre en niveau debug.

« **CustomLog** » : indique où seront stockés les journaux contenant l'ensemble des requêtes effectuées et le format de fichier. Le format le plus utilisé est « **combined** ».

5.3 Création des répertoires des hôtes virtuels

Après la définition des vhosts, il faut créer les répertoires sur lesquels pointeront ces derniers.

Les ressources (pages HTML, applications php,...), s'installent dans dans le répertoire « **/var/www** » (racine du serveur HTTP définie par défaut mais modifiable).

Créez donc les deux répertoires qui contiendront les ressources associées à nos deux vhosts :

```
# mkdir /var/www/{glpi,rss}
```

Mais au fait ? À qui doit appartenir ces répertoires ?

Sous Debian, quand on installe Apache2, l'utilisateur « **www-data** » est créé (ainsi que son groupe éponyme).

« ***www-data*** » et le g*roupe* « ***www-data*** », spécifient le compte anonyme utilisé par le serveur une fois qu'il est lancé. Note : ce compte est un compte uniquement utilisé par Apache2, ce n'est pas un utilisateur au sens classique. On évitera donc de tenter de s'identifier avec (même si cela est par défaut impossible, voir /etc/passwd).

Le premier processus est toutefois lancé en tant que root. En effet, pour accéder aux ports inférieurs à 1024, le serveur utilise un compte administrateur (root), ce qui présente par la suite des dangers.

Donc une fois le processus actif, les processus suivants utiliseront l'UID d'un autre compte (ici le **compte de service** « www-data ») qui eux traiteront les requêtes de clients.

Ce compte doit pouvoir accéder aux fichiers de la racine du serveur HTTP (« /var/www/ » dans notre cas). Il faudra donc veiller aux droits sur les pages web desservies.
D'autres distributions utilisent le compte « nobody » ou « apache ».

En cas de compromissions du serveur web, l'attaquant aura les privilèges non pas de « root » mais de « www-data » l'empêchant ainsi d'avoir accès à tout le système. Des configurations plus poussées permettent d'isoler encore plus notamment en utilisant un compte de service différent pour chaque vhost (et donc chaque processus associé Apache2) au sein du même serveur, ou bien en utilisant par exemple la conteneurisation (Docker ou LXC). Cela est un autre (vaste) sujet.

Revenons à nos deux vhosts. Une fois les répertoires créés, nous devons donc les associer au bon propriétaire :

```
# chown www-data: /var/www/{glpi,rss}
```

Avant de passer à l'activation des hôtes, on va placer un fichier « index.html » permettant de voir si nous sommes bien dans le bon répertoire lors des futures requêtes.

Créez le fichier « **index.html** » dans « **/var/www/glpi** » et saisissez ceci :

```
<!DOCTYPE html>
<HTML>
   <HEAD>
      <TITLE>Bienvenue sur GLPI</TITLE>
   </HEAD>
   <BODY>
      <P>Bonjour tout le monde et bienvenue sur GLPI !</P>
   </BODY>
</HTML>
```

Faites de même (en adaptant) dans « **/var/www/rss** ». N'oubliez pas les droits.

Un « **Doctype** » est une ligne de code servant a indiquer le Type de votre document. Le Doctype précise les normes définis que vous allez donc utilisé dans ce document. Lorsque vous codez une page web en HTML, vous devez définir son Doctype.

5.4 Activation des vhosts

Pour activer/désactiver un site il suffit de saisir :

```
# a2ensite nom_du_fichier_vhost (pensez à la touche TAB)
```

Cette commande permet de créer le lien symbolique dans « ***/etc/apache2/sites-enable/*** ».

Soit pour les vhosts « glpi » et « rss » :

```
# a2ensite 001-glpi.conf && a2ensite 001-rss.conf
```

Pour désactiver un vhost, il suffira d'utiliser la commande :
« **a2dissite *nom_fichier_vhost.conf*** »

Un moyen sûr, surtout quand on est en production, est de tester la configuration avant de relancer Apache2. Pour cela on va utiliser la commande « **apachectl** » :

```
# apachectl configtest ou bien apachectl -t
Syntax OK

Si NOK, Apache2 vous indiquera quel est le premier fichier (si
plusieurs) et quelle ligne de ce fichier le met en erreur. Donc test à
relancer après correction.
```

Si OK, il faut maintenant redémarrer Apache2 pour prendre en compte la nouvelle configuration :

```
# systemctl reload apache2.service (ou bien service apache2 reload)
```

Pourquoi pas « restart » au lieu de « reload ». Tout simplement en cas de serveur en production, hébergeant d'autres applications, cela évite de fermer toutes les connexions.

Il faut vérifier les logs système d'Apache2 (journalctl).

Il ne vous reste plus qu'à tester l'accès aux deux sites web depuis un navigateur.

6. Mise en place du HTTPS

6.1 Introduction

Avertissement : *La configuration de HTTPS avec Apache2 (comme avec ses concurrents) a clairement évolué et s'est complexifiée. La configuration qui sera présentée ici est surtout pour illustrer les grands principes du chiffrement utilisés par HTTPS. Tout-e administrateur-trice souhaitant mettre en production un serveur web Apache2, et qui plus est accessible depuis l'externe, devra d'abord se documenter minutieusement auprès de* ***supports récents****.*

HTTPS signifie « HTTP over SSL ». **SSL** (qui s'appelle **TLS** depuis 2001, et qui la version standardisée de SSL) est la couche cryptographique qui se place entre la couche transport (TCP dans ce cas) et la couche application (HTTP).

Pour rappel, le protocole **TLS** n'est pas lié à HTTP puisque nous pouvons le rencontrer avec d'autres protocoles applicatifs tels que IMAP, SMTP ou bien FTP.

Même si un serveur est sur un intranet et ne dessert que lui, il est indispensable aujourd'hui de mettre en place **HTTPS** pour assurer la sécurité des transmissions des données. En effet la frontière du périmètre de sécurité d'un SI n'est plus aussi claire qu'avant de par la mobilité des machines clientes.

Un appareil mobile a pu très bien être contaminé à l'extérieur par un logiciel espion qui pourrait « sniffer » le trafic une fois connecté sur le réseau interne. En résumé, il faut se méfier aussi bien du réseau interne que du réseau externe.

Afin d'assurer la sécurité de la communication, TLS s'appuie sur une suite cryptographique (cipher suite en anglais). Lors du **handshake TLS (voir annexe)**, le client envoie au serveur les suites cryptographiques qu'il supporte (ClientHello).
Le serveur choisit parmi cette liste la **suite cryptographique** qu'il souhaite utiliser, et la renvoie au client (ServeurHello).

6.2 Explication simplifiée de la suite cryptographique

Pour cela je vais partir de la fin. Cette partie est une approche globale qui nous sera utile pour comprendre certains paramètres lors de la configuration de HTTPS sur Apache2 (mais qui pourra aussi servir pour ses concurrents).

Dans une **communication sécurisée**, un navigateur et un serveur utilisent un **clé partagée** employée dans un algorithme de **chiffrement symétrique** (fonction mathématique prenant en valeur entrante la clé partagée et le message dont le résultat est un message chiffré) afin de garantir la **confidentialité**.

Quelques algorithmes utilisés à cette étape : AES, Camellia,...

La clé partagée est aussi utilisée dans un algorithme de **code d'authentification de message** (dit **MAC** pour Message Authentication Code / autre fonction mathématique prenant en valeur entrante la clé partagée, le message en clair et une fonction de hachage, comme *sha256* dont le résultat est une **empreinte** ou « **hach** ») afin d'assurer l'**intégrité** et **l'authentification** (en effet la clé partagée utilisée n'est connue que du client et du serveur) de chaque bloc composant le flux de données.
Le « **hach** » calculé est concaténé aux données non chiffrées, le tout étant ensuite chiffré en utilisant l'algorithme de chiffrement symétrique comme indiqué plus haut.

Quelques algorithmes utilisés à cette étape : HMAC, CMAC,...

Mais pour arriver aux étapes ci-dessus, il faut pouvoir **échanger la clé partagée** sur un canal sécurisé afin que personne ne puisse l'intercepter. Pour atteindre cet objectif, on fait appel à un **algorithme d'échange de clé**.

Deux grande familles d'algorithmes existent pour cette étape :

- **Chiffrement asymétrique :** Le client choisit la clé puis l'envoie chiffrée avec la clé publique du serveur. **RSA** est l'algorithme le plus utilisé dans ce cas.
- **Les algorithmes de type "key-agreement" :**Dans ce cas, le client et le serveur participent tous les deux à la génération de la clé. Les principaux algorithmes de type "key-agreement" **DHE** (Ephemeral Diffie-Hellman), et sa variante **ECDHE** (Ephemeral Elliptic curve Diffie–Hellman) qui est donnée pour être nettement plus performante et est recommandé.

Au lieu de transmettre la clef de session (clé partagée) et de chiffrer celle-ci en utilisant la partie publique de la clef RSA du serveur SSL (fourni par le certificat SSL de ce dernier), **les algorithmes de la famille DH** (Diffie-Hellman) permettent au client et au serveur de négocier une clé sans que celle-ci ne soit **JAMAIS** transmise sur le réseau.

Les algorithmes **DHE** et **ECDHE** assurent ce que l'on appelle la **confidentialité persistante** (*Perfect Forward Secrecy*, dit **PFS**) **mais** ne permettent pas de vérifier l'identité des deux parties. Ils doivent donc être complété par une **étape d'authentification**.
En effet le seul «problème» avec ces algorithmes DH, DHE ou ECDHE c'est qu'ils sont sujets aux attaques de « Man-In-the-Middle » (*« attaque de l'homme du milieu »*).
C'est pourquoi le **certificat SSL** du serveur SSL, dûment vérifier auprès d'une autorité de certification, reste utile afin que le client puisse vérifier qu'il est bien en contact avec le bon serveur et pas un usurpateur lors de la phase de négociation de la clef via DH, DHE ou ECDHE.

Quelques algorithmes utilisés à cette étape : RSA, ECDSA, DSS.

Retour sur la confidentialité persistante / PFS

Le problème principal avec le chiffrement asymétrique est que la clef privée est à la fois utilisée pour l'**authentification** du serveur et pour le **chiffrement** du **secret partagé**. Alors que l'authentification n'est réellement importante que pendant le laps de temps où la communication entre les deux parties est établie, le chiffrement quant à lui doit être capable de protéger un secret pendant plusieurs années.
Supposons qu'un attaquant enregistre tous les échanges entre le serveur et ses nombreux clients pendant plusieurs mois. Deux ans plus tard, le serveur est décommissionné et jeté à la benne. L'attaquant en profite pour subtiliser le disque et y trouve la clef privée. Il est désormais capable de déchiffrer toutes les communications qu'il a pu intercepter. Cela lui permet de récupérer, par exemple, des mots de passe encore valides aujourd'hui.
La propriété de **PFS** permet donc de protéger la sécurité des communications contre les écoutes à postériori d'une compromission des clefs SSL d'un serveur grâce au principe est que la clef de session (chiffrant la connexion) n'est plus envoyée sur le réseau que ce soit chiffrée ou pas!

Ce que vous devez retenir

*→ **HTTP couplé à TLS** permet de garantir la sécurité des connexions (piliers sécu info sans la dispo) et de nos jours devrait être systématique notamment avec l'application du **RGDP** (Règlement général sur la protection des données).*

En partant de l'idée que le serveur web est compromis :
*→ **Avec le PFS**, seules les communications futures ou « à venir » pourront être compromisses*
*→ **Sans le PFS**, ce sont non-seulement les communications à venir qui pourront être compromises mais c'est surtout toutes les communications passées/anciennes qui pourront être déchiffrées…*

→ L'écosystème de TLS change souvent, notamment avec l'évolution des algorithmes et des attaques. Il faut se maintenir à jour et ne pas considérer la configuration de HTTPS comme acquise.

6.3 Création du certificat X.509

a. Certificat et certification

Un **certificat électronique** (aussi appelé **certificat numérique** ou **certificat de clé publique**) peut être vu comme une carte d'identité numérique. Il est utilisé principalement pour identifier et authentifier une personne physique ou morale, mais aussi pour chiffrer des échanges.

Il est **signé** par un **tiers de confiance** (aussi appelé **autorités de certification**, **AC** ou **CA** pour **Certificate Authority** en anglais) qui atteste du lien entre l'identité physique et l'entité numérique.

Les **autorités de certification** sont des organismes enregistrés et certifiés auprès d'autorités publiques et/ou de gouvernance de l'Internet qui établissent leur viabilité comme intermédiaire fiable.

Ces organismes diffusent leurs propres clés publiques et étant certifiées fiables, ces autorités sont en contact direct avec les principaux producteurs de navigateurs web (tels que Mozilla Firefox, Google Chrome, Internet Explorer, etc.) qui incluent nativement les listes de clés des autorités de certification.

C'est cette relation qui est à la base de la « **chaîne de confiance** ». Ces clés sont appelées **clés publiques racines** ou **certificats racines** et sont utilisées pour identifier les clés publiques d'autres organismes.

Cependant, les **CA** doivent répondre à des critères de sécurité très stricts, notamment pour garantir que leurs propres certificats ne sont pas compromis, ce qui entraînerait la corruption de tous les certificats émis sous leur responsabilité.

C’est pourquoi la clé privée de leurs **certificats racines** est mise à l’abri, et n’est pas utilisée pour signer des certificats SSL, mais pour signer des **certificats intermédiaires**, qui à leur tour signent les **certificats SSL finaux**. Ainsi même si la clé privée d'un de ces certificats intermédiaires était compromise, les certificats SSL dépendant d'un autre certificat intermédiaire ne seraient pas affectés. On retrouve ici le principe de la segmentation.

C’est le terme « **chaîne de confiance** » qui désigne ceci.

En effet, la certification peut s’effectuer en cascade. Un certificat peut permettre d’authentifier d’autres certificats jusqu’au certificat qui sera utilisé pour la communication.

Concrètement…

Les navigateurs web modernes intègrent nativement une liste de **certificats** provenant de différentes **Autorités de Certification** choisies selon des règles internes définies par les développeurs du navigateur.

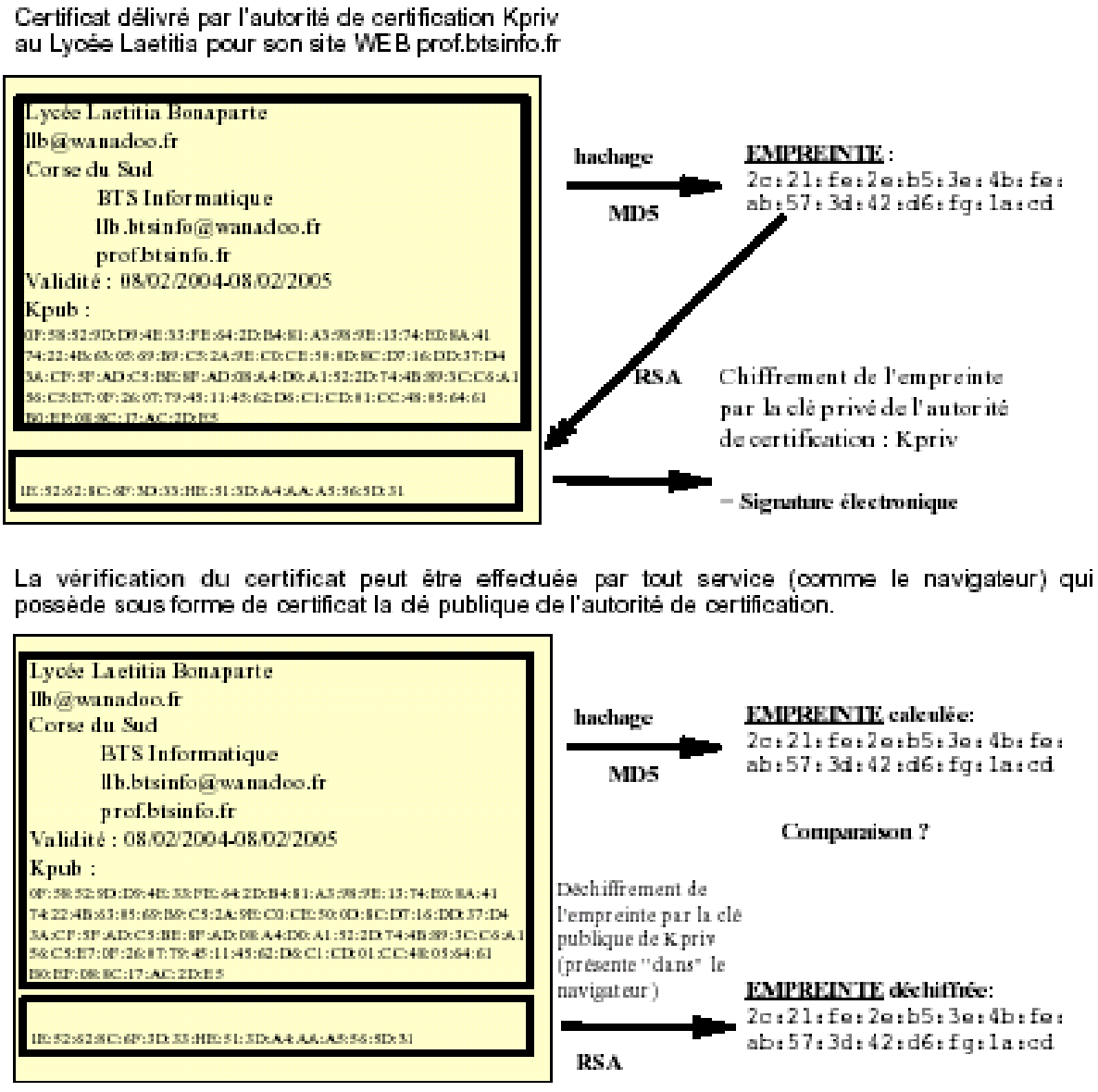
Lorsqu'une personne physique ou morale souhaite mettre en place un serveur web utilisant une communication HTTPS sécurisée par TLS, elle génère une clé publique, une clé privée (clé privée qu’elle ne communique à personne) puis envoie à une Autorité de Certification une **demande de signature de certificat** (en anglais **CSR** : Certificate Signing Request) contenant sa clé publique ainsi que des informations sur son identité (coordonnées postales, téléphoniques, email...).

Après vérification de l'identité du demandeur du certificat par une a**utorité d'enregistrement** (RA), l'**Autorité de Certification** signe le **CSR** grâce à sa propre clé privée qui devient alors un certificat puis le transmet en retour à la personne qui en a fait la demande.

Le certificat ainsi retourné sous forme de fichier informatique est intégré dans le serveur web du demandeur. Lorsqu'un utilisateur se connecte à ce serveur web, celui-ci lui transmet à son tour le certificat fourni précédemment par l'Autorité de Certification (et éventuellement le certificat de l'AC intermédiaire si le certificat du site a été signé par une autorité intermédiaire).

Le navigateur web du client authentifie le certificat du serveur grâce au certificat de l'Autorité de Certification (intégré nativement dans le navigateur, cf. ci-dessus) qui l'a signé précédemment. L'identité du serveur est ainsi confirmée à l'utilisateur par l'Autorité de Certification.

La figure suivante illustre un exemple de certificat émis pour un serveur WEB :



La vérification du certificat peut être effectuée par tout service (comme le navigateur) qui possède sous forme de certificat la clé publique de l'autorité de certification.

Structure d'un certificat x509

La partie authentifiée contient les champs suivants :

- Version
- Numéro de série
- Algorithme de signature du certificat
- DN (*Distinguished Name*) du délivreur (autorité de certification)
- Validité (dates limites)
    - Pas avant
    - Pas après
- DN de l'objet du certificat
- Informations sur la clé publique
    - Algorithme de la clé publique
    - Clé publique proprement dite
- Identifiant unique du signataire (optionnel, X.509v2)
- Identifiant unique du détenteur du certificat (optionnel, X.509v2)
- Extensions (optionnel, à partir de X.509v3)
    - Liste des extensions
- Signature des informations ci-dessus par l'autorité de certification

Pour finir…

L'Autorité de certification est un modèle de **PKI** (***Public Key Infrastructure*** ou bien ***IGC, infrastructure de gestion de clés*** ou encore ***infrastructure à clés publiques***).

Nous ne verrons pas la mise en place d'une PKI, celle-ci étant plutôt réservée aux architectes SI, mais c'est une infrastructure que vous pourriez rencontrer en interne dans une entreprise.

Une PKI est une organisation centralisée afin d'instaurer la confiance dans les échanges de données, principalement en permettant l'échange de clés publiques et l'identification des ordinateurs et des individus.

Composants d'une PKI :

- **Les certificats** : le certificat associe une clé publique et des données d'identités, le tout signé par un CA. L'identité d'un utilisateur est généralement son email. L'identité d'un serveur est son adresse DNS complète (FQDN).

- Les AC en charge de gérer les certificats (signature, cycle de vie,…). On trouve deux types d'AC :

- les AC privées : AC internes d'une société limitées à un périmètre restreint. Permettent de créer et de gérer en interne une PKI.
- Les AC publiques : rayonnement mondial permettant de vérifier l'identité des serveurs sur Internet.

Cycle de vie des certificats :

- Vous avez pu noter dans la structure du certificat x509 le champs « Validité » contenant lui-même deux champs « Pas avant » et « Pas après ». Lors de la création du CSR, vous pouvez déterminer une durée de validité. La durée maximale supportée par une AC publique est de 39 mois. Lorsque le certificat arrive à expiration et deviendra donc inutilisable, il faudra réaliser une demande de renouvellement de certificat.
- Il existe une phase optionnelle entre l'émission du certificat et l'expiration de ce dernier : la révocation.

La révocation

Après que l'identité d'un serveur ait été confirmée à l'utilisateur par l'Autorité de Certification, le navigateur web contacte ensuite l'Autorité de Certification concernée pour savoir si le certificat du serveur n'a pas été révoqué (= invalidé) depuis qu'il a été émis par l'Autorité de Certification via une demande **OCSP** (Online Certificate Status Protocol).

Auparavant, les navigateurs téléchargeaient régulièrement des listes de révocation (**CRL : Certificate Revocation List**) de la part des Autorités de Certification au lieu de contacter directement celles-ci par des demandes **OCSP**. Ce processus a été abandonné depuis car utilisant inutilement beaucoup de bande passante.

La révocation se fait au sein du CA dont l'étape suivante consistera en la diffusion de ces révocations.

Plus d'infos : https://fr.wikipedia.org/wiki/Online_Certificate_Status_Protocol

b. Création de l'AC et du certificat serveur

L'objectif est d'accéder de manière sécurisée à nos vhosts « glpi » et « rss » en utilisant un certificat auto-signé. Ce type de certificat est utilisé en interne dont le périmètre est restreint (ne vise que les clients internes).

Pour ce faire il nous faut créer une AC afin de signer le certificat serveur. Le certificat de l'AC sera plus tard importé dans le magasin des certificats des navigateurs web afin d'éviter le message d'erreur lié aux certificats auto-signées.

Création de l'AC :

On génère la clé privée de l'AC qui sera chiffrée en AES256 et qui aura comme longueur 4096 bits (vous devrez aussi définir une passphrase lors de la génération de la clé / Placez vous dans le répertoire de root) :

```
# openssl genrsa -aes256 4096 > ca.key
# chmod 600 ca.key
```

La clef de notre CA étant générée, nous allons créer un certificat "auto-signé" valable pour 365 jours pour notre AC :

```
# openssl req -new -x509 -days 365 -key ca.key > ca.crt
```

Il faut saisir la passphrase puisqu'on utilise “**ca.key**” que l'on a protégé. Et on donne les renseignements concernant l'autorité de certification.

```
Enter pass phrase for ca.key:
You are about to be asked to enter information that will be
incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name
or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:FR
State or Province Name (full name) [Some-State]:France
Locality Name (eg, city) []:Toulouse
Organization Name (eg, company) [Internet Widgits Pty Ltd]:TSSR
Organizational Unit Name (eg, section) []:HOMELAB
Common Name (e.g. server FQDN or YOUR name) []:cert_CA
Email Address []:admin@tssr.lan
```

On génère le certificat à destination des clients web (celui qu’on importera dans le magasin des certificats de Firefox par exemple) :

```
# openssl x509 -in ca.crt -outform DER -out ca.der
```

Création du certificat serveur :

On génère la clé privée :

```
# openssl genrsa 4096 > tssr.lan.key
# chmod 600 tssr.lan.key
```

On génère le CSR :

```
# openssl req -days 365 -new -key tssr.lan.key \
> -out tssr.lan.csr
```

Saisissez les renseignements suivants en précisant bien « ***.tssr.lan** » (nous verrons plus loin pourquoi) :

```
You are about to be asked to enter information that will be
Incorporated into your certificate request.
What you are about to enter is what is called a Distinguished Name
or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:FR
State or Province Name (full name) [Some-State]:France
Locality Name (eg, city) []:Toulouse
Organization Name (eg, company) [Internet Widgits Pty Ltd]:TSSR
Organizational Unit Name (eg, section) []:HOMELAB
Common Name (e.g. server FQDN or YOUR name) []:*.tssr.lan
Email Address []:admin@tssr.lan
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []: laisser vide
An optional company name []: laisser vide
```

On signe le CSR avec notre AC :

```
# openssl x509 -req -days 360 -in tssr.lan.csr \
> -CA ca.crt \
> -CAkey ca.key \
> -CAcreateserial -out tssr.lan.crt -sha256

Ce qui renvoie :

Signature ok
subject=/C=FR/ST=France/L=Toulouse/O=TSSR/OU=HOMELAB/CN=*.tssr.lan/
emailAddress=olivier@ldnr.fr
Getting CA Private Key
Enter pass phrase for ca.key: Entrez votre passphrase
```

6.4. Configuration d'Apache2 et des vhosts

a. Mise en place des fichiers

```
# mkdir /etc/apache2/ssl
# chown root: /etc/apache2/ssl/
# chmod 640 /etc/apache2/ssl
# mv tssr.lan.{key,crt} /etc/apache2/ssl/
```

b. Activation de modules

```
# a2enmod ssl rewrite headers
```

c. Définition des paramètres SSL pour le contexte des vhosts

Modification de « 001-glpi.conf » (faire de même avec « 001-rss.conf ») :

```
<VirtualHost *:80>
        ServerName glpi.tssr.lan
        Redirect / https://glpi.tssr.lan
</VirtualHost>

<VirtualHost *:443>
        ServerAdmin votre-mail@tssr.lan
        ServerName glpi.tssr.lan

        DocumentRoot /var/www/glpi

        <Directory /var/www/glpi>
                Options -Indexes -FollowSymLinks -MultiViews
                AllowOverride None
                Order allow,deny
                Allow from all
        </Directory>

        ErrorLog /var/log/apache2/error.log

        # Possible values include: debug, info, notice, warn, error, crit,
        # alert, emerg.
        LogLevel warn

        CustomLog /var/log/apache2/access.log combined

        # Cette directive permet d'activer/désactiver le moteur du protocole
        # SSL/TLS.
        SSLEngine on
        # Fichier de données contenant le certificat X.509 du serveur codé en
        # PEM.
        SSLCertificateFile /etc/apache2/ssl/tssr.lan.crt
        # Fichier contenant la clé privée du serveur codée en PEM
        SSLCertificateKeyFile  /etc/apache2/ssl/tssr.lan.key

        # Strict Transport Security
        # HSTS (mod_headers is required) (15768000 seconds = 6 months)
        Header always set Strict-Transport-Security "max-age=15768000"

</VirtualHost>
```

d. Définition des paramètres SSL dans le contexte global d'Apache

Éditez le fichier « ssl.conf » :

```
# vim /etc/apache2/mods-enabled/ssl.conf
```

Modifiez les lignes de « ssl.conf »suivantes avec ceci (*pensez à faire un backup du fichier avant de le modifier*) :

À modifier / ajouter :

```
# Cette directive permet de définir quelles versions du
# protocole SSL/TLS seront acceptées lors de l'initialisation
# d'une nouvelle connexion. Ici on accepte les versions TLS
# mais pas le SSLv3.
SSLProtocol             all -SSLv3

# Algorithmes de chiffrement disponibles pour la négociation
# au cours de l'initialisation de la connexion SSL
# LA LIGNE SUIVANTE EST SUR UNE SEULE LIGNE (pas de retour et
# pas d'espace, sauf entre la directive et la liste)
SSLCipherSuite          ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-
RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-
AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-
AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-
GCM-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-
SHA256:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-
RSA-AES128-SHA:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES256-
SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-
SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES256-SHA:ECDHE-ECDSA-DES-
CBC3-SHA:ECDHE-RSA-DES-CBC3-SHA:EDH-RSA-DES-CBC3-SHA:AES128-
GCM-SHA256:AES256-GCM-SHA384:AES128-SHA256:AES256-
SHA256:AES128-SHA:AES256-SHA:DES-CBC3-SHA:!DSS

# Option permettant de classer les algorithmes de chiffrement
# du serveur par ordre de préférence. Si cette directive est
# activée, ce sont les préférences du serveur qui seront
# prises en compte à la place.
SSLHonorCipherOrder     on

# Compression SSL désactivée car sensible à l'attaque CRIME
SSLCompression          off

# Désactivation des tickets de session TLS car représente un
# risque de compromission de la confidentialité.
SSLSessionTickets       off
```

e. Vérification et relance d'Apache2

Avant de relancer Apache, nous allons configurer le vhost par défaut. Ainsi si on essai d'atteindre le serveur web avec son IP ou son nom d'hôte (srvweb dans notre cas), la requête sera redirigée vers ce vhost qui lui-même sera associé à une page web indiquant de saisir une URL existante.

Éditez le fichier etc/apache2/sites-enabled/000-default.conf et configurez le ainsi :

```
<VirtualHost _default_:443>
            DocumentRoot /var/www/default
            SSLEngine on
            # Fichier de données contenant le certificat X.509 du serveur codé en
            # PEM.
            SSLCertificateFile /etc/apache2/ssl/tssr.lan.crt
            # Fichier contenant la clé privée du serveur codée en PEM
            SSLCertificateKeyFile  /etc/apache2/ssl/tssr.lan.key
</VirtualHost>

<VirtualHost _default_:80>
            DocumentRoot /var/www/default
</VirtualHost>
```

Créez le répertoire « default » indiqué dans la directive DocumentRoot, puis créez le fichier « index.html » dans ce répertoire :

```
<!DOCTYPE html>
<HTML>
   <HEAD>
      <TITLE>URL invalide</TITLE>
   </HEAD>
   <BODY>
      <P>Merci de saisir une adresse de site existante.</P>
   </BODY>
</HTML>
```

Vérifiez et relancez Apache :

```
# apachectl configtest ou bien apachectl -t
Syntax OK

Si NOK, Apache2 vous indiquera quel est le premier fichier (si
plusieurs) et quelle ligne de ce fichier le met en erreur. Donc test
à relancer après correction.

# systemctl reload apache2.service (ou bien service apache2 reload)
```

Importez le fichier « **ca.der** » dans le magasin des certificats de Firefox.

Connectez-vous en https sur vos vhosts. Gardez en parallèle une fenêtre affichant les logs en direct (« journalctl -f », « tail -f » sur les fichiers de logs de vos vhsost).

Si tout se passe bien, vous n'aurez pas de message d'alerte de Firefox (comme c'est le cas avec PFSense) concernant le certificat.

7. Conclusion

Nous venons de mettre en place la deuxième couche de notre stack LAMP.

Savoir mettre en place un serveur web interne est aujourd'hui indispensable car il permet d'utiliser des clients légers à travers les navigateurs et donc permet de gagner du temps dans le déploiement d'applications de type métier.

Mettre en place un serveur web sécurisé demande surtout un travail de recherche et de veille constante surtout si les sites hébergés doivent être accessibles depuis Internet.

À moins de passer par un prestataire il n'existe pas de solutions clé en main (que ce soit sous Apache, IIS, Nginx,...).

En tant que solution interne, on consacrera la partie sécurité sur le chiffrement des transmissions et la disponibilité des données (sauvegarde et redondance).

Notez qu'un serveur web est très dépendant du service DNS. Si celui-ci tombe en panne, toutes les applications deviennent alors inaccessibles.

Au fait pourquoi avoir créer un certificat « *.tssr.lan » et non deux certificats, un pour GLPI et l'autre pour RSS ? C'est ce qu'on appelle la **portée des certificats** :

→ Un seul domaine : suffisant pour un site web, mais pas pour les autres services (mail, webmail, ftp, etc.) exemple : www.ldnr.fr

→ Domaine + sous-domaine (l'étoire « * » dite wildcard) : idéal pour un ou plusieurs sites web et les autres services. exemple : *.ldnr.fr (www..ldnr.fr, smtp.ldnr.fr, ...)

→ Multi-domaine : généralement réservé aux grandes structures, agences web ou hébergeurs. Avec un seul certificat on valide des domaines différents.
Exemple: www.ldnr.fr, pf4.campus-numerique.eu, ...

C. MySQL / MariaDB

1. Présentation

Dans cette partie, nous verrons uniquement l'installation et la configuration de base du **SGBDR** (**Système de Gestion de Bases de Données Relationnelles**).

Un **SGBD** (Système de Gestion de Bases de Données) est un logiciel qui stocke des données de façon organisées et cohérentes. Un **SGBDR** (Système de Gestion de Bases de Données Relationnelles) est le type particulier de SGBD qui fera l'objet de ce cours. Il vous sera décrit dans un prochain cours ce qui fait qu'une bases de données est relationnelle.

Pourquoi MySQL / MariaDB ?

Je vous laisse lire ces deux pages qui résument bien la raison et la situation actuelle :

- https://fr.wikipedia.org/wiki/MariaDB.
- https://www.scriptol.fr/programmation/mysql.php

Les distribution GNU/Linux récentes ne proposent plus **MySQL** mais **MariaDB.** MySQL n'est plus proposée par Debian depuis la version 9. La version 8 de Debian étant toujours maintenue, il se peut que vous rencontriez encore MySQL. MySQL et MariaDB sont très proches. MariaDB ajoute des capacités qu'Oracle ne fournit que sur la version commerciale de MySQL. Malgré les différences, on peut installer directement MariaDB en remplacement de MySQL, sans changement dans les bases. Sous Linux avec juste une commande.

Il existe d'autres SGBDR. Voici l'état du marché (source : https://db-engines.com/en/) :

Rank			DBMS	Database Model	Score		
Feb 2018	Jan 2018	Feb 2017			Feb 2018	Jan 2018	Feb 2017
1.	1.	1.	Oracle	Relational DBMS	1303.28	-38.66	-100.55
2.	2.	2.	MySQL	Relational DBMS	1252.47	-47.24	-127.83
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1122.04	-26.03	-81.42
4.	4.	4.	PostgreSQL	Relational DBMS	388.38	+2.19	+34.70
5.	5.	5.	DB2	Relational DBMS	189.97	-0.30	+2.07
6.	6.	6.	Microsoft Access	Relational DBMS	130.07	+3.37	-3.32
7.	7.	7.	SQLite	Relational DBMS	117.27	+3.02	+1.96
8.	8.	8.	Teradata	Relational DBMS	72.99	+0.36	-2.60
9.	9.	9.	SAP Adaptive Server	Relational DBMS	63.49	-1.98	-8.25
10.	10.	↑ 13.	MariaDB	Relational DBMS	61.65	+3.35	+16.30

Notez la présence de **PostgreSQL** est un concurrent libre et gratuit à Mysql/MariaDB.

Travaillant sous Debian 9, nous utiliserons donc MariaDB. Les applications web que nous installerons, utiliseront donc ce SGBDR afin de stocker et traiter les données (comptes utilisateurs, ...).

2. Installation

```
# apt-get install mariadb-server mariadb-client
```

Voilà MariaDB est opérationnel. Par défaut MariaDB écoute sur le port « 3306 » sur l'interface « localhost ». L'accès est donc très restreint, ce qui est normal car on évitera de rendre la base de données directement accessible depuis l'externe.

Vérifiez que MariaDB fonctionne.

3. Sécurisation

MariaDB n'utilise pas les comptes Unix mais gère ses propres utilisateurs et mots de passe.

Par défaut, lors de l'installation, le compte root (au sens MariaDB) est créé sans mot de passe.

Il est donc, au minimum, nécessaire de le modifier rapidement (surtout si votre machine est accessible sur un réseau).

La solution la plus complète, consiste à utiliser le script de sécurisation fourni avec MariaDB.

```
# mysql_secure_installation
```

Il vous sera proposé de

- **Change the root password?** => modifier le mot de passe de l'administrateur (ne pas mettre le même mot de passe que celui du « root système »)
- **Remove anonymous users?** => supprimer les utilisateurs anonymes
- **Disallow root login remotely?** => interdire les connexions distantes de l'administrateur
- **Remove test database and access to it?** => supprimer la base de test et les droits associés
- **Reload privilege tables now?** => recharger les privilèges pour prise en compte immédiate

En répondant Y (oui) à toutes ces question, on obtient (*théoriquement*) un serveur sécurisé.

Le compte « root » dans ce cas est le compte créé avec tous les privilèges sur les bases.

Jusqu'à Debian 8, lorsque l'on souhaitait se connecter à MariaDB/Mysql en tant que « root » (celui du SGBDR), il fallait saisir le mot de passe défini lors de l'installation (ou bien depuis mysql_secure_installation) :

```
# mysql -u root -p
Enter password: saisie du mot de passe root du SGBDR
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection...
```

Avec Debian 9, si l'on est identifié en tant que root sur le système, on peut se connecter en console au SGBDR sans saisir le mot de passe pourtant demandé :

```
# mysql -u root -p
Enter password: mot de passe vide
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection...
```

Par contre si l'on se connecte en tant qu'utilisateur ordinaire, il faut que l'utilisateur en question soit membre du groupe « sudo » et lancera la connexion de cette manière :

```
superadm@srvweb:/$ sudo mysql -u root -p
Enter password: saisie du mot de passe root du SGBDR
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection...
```

Ce changement a bouleversé nombre d'administrateurs et semble spécifique à Debian 9. Une des conséquence c'est que l'on doit créer un utilisateur avec les mêmes privilèges que l'utilisateur « root » du SGBDR.

En effet lors de l'installation d'applications web, la plupart propose un assistant graphique pour créer leur base de données associée (ainsi qu'un utilisateur ayant tous les privilèges sur cette dernière). Nous utilisions jusqu'à présent le compte « root » du SGBDR mais cela ne fonctionne plus. D'où le besoin de créer un utilisateur avec les mêmes privilèges que le compte « root » du SGBDR, partie que l'on verra un peu plus loin.

Une solution existe pour revenir sur l'ancien mode de fonctionnement : https://www.citizenz.info/mariadb-mysql-connexion-root-avec-phpmyadmin-sous-ubuntu-16-04

4. Fichiers de configuration

Dans un souci d'interopérabilité, Debian utilise les anciens chemins et les commandes de MySQL.

Les fichiers de configuration se situent donc dans « **/etc/mysql** ».

Le fichier de configuration principal du serveur est : « **/etc/mysql/mariadb.conf.d/50-server.cnf** ».

On peut notamment y configurer le port et l'interface d'écoute.

5. Introduction au langage SQL

Le SQL, Structured Query Language, est un langage Standard permettant à un client de communiquer des instructions à la base de données.

- Le **DDL** (Data definition language) comporte les instructions qui permettent de définir la façon dont les données sont représentées. (création, suppression, modification de bases de données)
- le **DML** (Data manipulation language) permet d'écrire dans la base et donc de modifier les données. (création, suppression, modification et sélection des données)
- le **DCL** (Data control language), qui permet de contrôler l'accès aux données.

Comme expliqué plus haut, la plupart des applications web permettent d'utiliser un assistant pour créer leur base de données sur notre SGBDR et l'utilisateur qui se connectera à celle-ci. Cependant il leur faut utiliser un compte ayant les privilèges. Il est possible que vous ne compreniez pas complètement la finalité mais ne vous inquiétez pas vous la comprendrez très vite lors de l'installation de notre première application. On se connecte à MariaDB :

```
# mysql -u root -p
Enter password: saisie du mot de passe root du SGBDR ou laissez vide
Welcome to the MariaDB monitor…

MariaDB [(none)]> GRANT ALL PRIVILEGES ON *.* (faire Entrée)
    -> TO 'NOM_USER'@'localhost' (faire Entrée)
    -> IDENTIFIED BY 'supermotdepasse'
    -> WITH GRANT OPTION; (faire Entrée)

MariaDB [(none)]> FLUSH PRIVILEGES;(faire Entrée)
```

On vérifie maintenant que l'utilisateur a bien été créé avec les privilèges :

```
MariaDB [(none)]> SHOW GRANTS FOR 'NOM_USER'@'localhost';
+-----------------------------------------------------------------------+
| Grants for NOM_USER@localhost|
+-----------------------------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'NOM_USER'@'localhost' IDENTIFIED BY
PASSWORD 'Mot de passe chiffré' WITH GRANT OPTION |
+-----------------------------------------------------------------------+
```

Et voilà ! Vous venez de pratiquer vos **premières requêtes DCL** ! Nous avons un utilisateur équivalent au compte « root » du SGDBR que l'on pourra utiliser avec les assistants d'installation de nos applications web (Vous aurez bien évidemment penser à noter le nom du compte et son mot de passe).

Détaillons la commande « ***GRANT ALL PRIVILEGES***... » :

- **GRANT ALL PRIVILEGES** : Donne tous les privilèges (création, suppression, ...)

- ***ON *.**** *:* sur toutes les bases de la BDD et les données (compte à conserver clairement à l'abri des regards). Le premier « * » correspond aux bases et le second « * » aux tables. On peut donc spécifier une base et affiner sur une table. Ex : « ON mabdd.* » ou bien « ON mabdd.tableuser »

- ***TO*** '***NOM_USER***'@'***localhost***' *:* à l'utilisateur « NOM_USER » depuis **localhost**. Depuis localhost indique que les requêtes qui seront acceptées devront être réalisées depuis la machine locale hébergeant MariaDB. Si l'on veut se connecter à la BDD depuis une autre machine hôte, il faudra remplacer localhost par l'IP de la machine ou bien par le caractère joker « % ».

- *IDENTIFIED BY* '*supermotdepasse*' : cela se passe de commentaires :)

- ***WITH GRANT OPTION*** *:* celui qui reçoit ce droit *peut affecter des droits aux autres.*

La partie création de la BDD et de son utilisateur associé pour une application web est souvent laissée aux administrateurs.

Donc selon les applications, on devra se connecter à MariaDB pour créer l'ulisateur :

```
# mysql -u root -p
Enter password: saisie du mot de passe root du SGBDR ou laissez vide
Welcome to the MariaDB monitor…

MariaDB [(none)]> CREATE DATABASE bddapp; (faire Entrée)

MariaDB [(none)]> SHOW DATABASES; (pour vérifier que la base a bien
                                   été créée)

MariaDB [(none)]> GRANT ALL PRIVILEGES ON bddapp.* (faire Entrée)
    → TO 'APP_NOM_USER'@'localhost' (faire Entrée)
    → IDENTIFIED BY 'supermotdepasse'; (faire Entrée)

MariaDB [(none)]> FLUSH PRIVILEGES;(faire Entrée)
```

La **différence** avec la requête précédente, c'est que nous avons créé la base de données (qui accueillera les données de l'application) et nous avons créé et dédié un utilisateur pour celle-ci. Mieux vaut éviter en effet d'utiliser un utilisateur ayant accès à toutes les bases de données (encore de la segmentation)

IMPORTANT : l'utilisateur que nous créons dans le SGBDR n'est pas un utilisateur avec lequel on se connectera sur l'application web. C'est un utilisateur dont se servira l'application en question (paramétré lors de l'installation de celle-ci) afin de manipuler et d'accéder à des listes de données organisées par tables dans la BDD.

6. Conclusion

Vous venez d'avoir un aperçu du SGBDR MariaDB et du langage SQL. En ce qui concerne l'architecture 3-Tier, nous y reviendrons dans un chapitre de ce cours lui étant dédié.

D. PHP

1. Présentation

« **PHP: Hypertext Preprocessor4** », plus connu sous son sigle **PHP** (acronyme récursif), est un langage de programmation libre, principalement utilisé pour produire des **pages Web dynamiques** (formulaires,...) via un serveur HTTP, mais pouvant également fonctionner comme n'importe quel langage interprété de façon locale. PHP est un langage impératif orienté objet.

PHP a permis de créer un grand nombre de sites web célèbres, comme Facebook, Wikipédia, LDNR, etc.

Il est considéré comme une des bases de la création de sites web dits dynamiques mais également des **applications web**. Ça tombe bien, c'est sur ce quoi nous allons travailler.

Un exemple type : un internaute veut afficher son profil sur un site internet. Au moyen de son navigateur il effectue une requête sur un serveur web en appelant une URL sur le domaine du serveur. **Apache** lance alors automatiquement le **module PHP**.

PHP va interpréter un script : pour répondre à la requête du navigateur il va à son tour envoyer une requête au serveur **MySQL** en lui demandant les informations de l'utilisateur. Une fois ces informations récupérées, toujours en suivant le script, **PHP** mettra en forme le résultat sous forme de page web (HTML). **Apache** pourra alors retourner cette page au navigateur qui l'affichera.

2. Installation

On installera la version 7 de PHP.

Il existe différente manière d'installer PHP. Nous utiliserons la plus connue et la plus simple :

```
# apt install php libapache2-mod-php php-mysql
```

Et voilà !

Paquets installés :

- Le paquet **php** installe le logiciel permettant d'interpreter le langage PHP.
- Le paquet **libapache2-mod-php** permet d'utiliser PHP en tant que module d'Apache (on peut aussi utiliser PHP en ligne de commande ou indépendamment en FastCGI, ce qui le rend plus performant mais plus compliqué à mettre en place).
- Le paquet **php-mysql** installe le module permettant d'utiliser MySQL ou MariaDB avec PHP.

On relance Apache2 et on vérifie tout de même si le module PHP est bien chargé :

```
# apachectl -t -D DUMP_MODULES
Loaded Modules:
(…)
php7_module (shared)
(…)
```

Si ce n'était pas le cas :

```
# a2enmod php7.0
# systemctl restart apache2
# apachectl -t -D DUMP_MODULES
Loaded Modules:
(…)
php7_module (shared)
(…)
```

3. Fichiers de configuration

Le fichier de configuration est « **/etc/php/7.0/apache2/php.ini** ». Le fichier fait quasiment 2000 lignes d'où l'intérêt de savoir utiliser correctement un éditeur de texte.

Si vous éditez le fichier avec votre éditeur préféré (vim) vous comprendrez que nous n'aborderons pas toutes les options de configuration. Les applications web correctement documentées vous indiqueront les paramétrages à faire si besoin.

Cependant, il me paraît intéressant d'aborder certaines options :

- `max_execution_time = 30` : Permet de définir en secondes le temps d'exécution maximal d'un script . S' il vaut 0 alors il n'y a plus de limite. Sur des petites configurations systèmes il peut être utile d'augmenter ce temps.
- `memory_limit = 8M` : Permet de définir la mémoire disponible pour l'exécution de vos scripts.
- `display_errors = Off` : Permet de na pas afficher les messages d'erreurs (pratique pour les serveurs de production). À mettre sur On lors de la phase de test d'une application php.
- `log_errors = On` : Permet de journaliser les erreurs, utile si vous avez mis à off le paramètre précédent
- `upload_max_filesize = 2M` : Permet de limiter la taille des fichiers uploadés.
- post_max_size = 8M : Définit la taille maximale des données reçues par la méthode POST. Cette option affecte également les fichiers chargés. Pour charger de gros fichiers, cette valeur doit être plus grande que la valeur de upload_max_filesize.

4. Modules PHP

La plupart des scripts PHP (LMS, ERP, CMS, forums, et applications web en tout genre) utilisent des modules de PHP pour bénéficier de certaines fonctionnalités.

Voici comment installer les modules les plus courants :

```
# apt install php-curl php-gd php-intl php-json php-mbstring php-mcrypt
php-xml php-zip
```

Les applications web correctement documentées vous indiqueront quels sont les modules à ajouter. Si l'application est très bien documentée, elle vous donnera le nom des paquets correspondants à votre distribution (le noms varient entre par exemple un distribution basée sur Redhat et une autre sur Debian). Il faudra donc utiliser la fonction de recherche des paquets avec le gestionnaire de paquet de la distribution en question et parfois passer par de la recherche sur les forums et moteurs de recherche.

5. Test

Dans le doute on redémarre Apache avant d'écrire un script PHP afin de tester le bon fonctionnement de notre stack.

Créez le fichier «**test.php** » (sans oublier de mettre les droits adéquates dessus) dans « **/var/www/rss** » et saisissez ceci dedans :

```
<?php
        phpinfo();
?>
```

RDV sur « **https://rss.tssr.lan/test.php** ».

IMPORTANT : ce fichier ne doit pas se trouver sur un vhost en production, sécurité oblige.

6. Conclusion

Nous venons de mettre en place la dernière couche de notre pile LAMP. « P » ne veut pas dire forcément PHP, on peut notamment mettre en place des applications écrites en Python.

E. Architecture n-Tier

Lorsque nous mettons en place un service réseau, nous l'implantons dans une logique **client-serveur (2-Tier)**. L'architecture client-serveur possède toutefois des inconvénients. Ce sont ces inconvénients qui poussent les entreprises à utiliser d'autres technologies.

Les **deux inconvénients** principaux sont la difficulté à gérer correctement les questions de **sécurité** et le **coût** du déploiement.

Note : « Tier » nom provient de l'anglais, signifiant étage ou niveau.

1. Cas d'un SGBDR en architecture client/serveur

La sécurité d'un système en architecture client-serveur est gérée au niveau du SGBDR. Celui-ci contrôle l'accès aux données en attribuant des autorisations d'accès aux différents utilisateurs du système.

Le problème vient du fait que cette attribution de droit ne peut pas tenir compte des spécificités du logiciel réalisé. Pour pouvoir gérer les droits d'accès de cette façon il faut accorder à chaque utilisateur des droits sur un grand nombre de tables, ce qui devient vite laborieux.

En pratique il est souvent plus rapide de ne créer qu'un utilisateur sur le SGBDR avec lequel tous les utilisateurs se connectent. Cette approche ne permet aucune gestion de la sécurité. Il suffit de connaître le login et le mot de passe d'accès à la base pour accéder à l'ensemble des données sans aucune restriction.

L'autre problème est souvent considéré comme beaucoup plus important par les entreprises car il est beaucoup plus visible. Il s'agit des durées et coûts de déploiement des logiciels. En effet un logiciel classique, développé en architecture client-serveur, nécessite une installation et une éventuelle configuration sur chaque poste utilisateur. Le déplacement d'un technicien coûte déjà très cher aux entreprises. Mais ce qui reste le plus laborieux est la nécessité de mettre à jour régulièrement le logiciel. Dans une architecture client-serveur, chaque mise à jour du logiciel nécessite un nouveau déploiement accompagné de nombreux coûts.

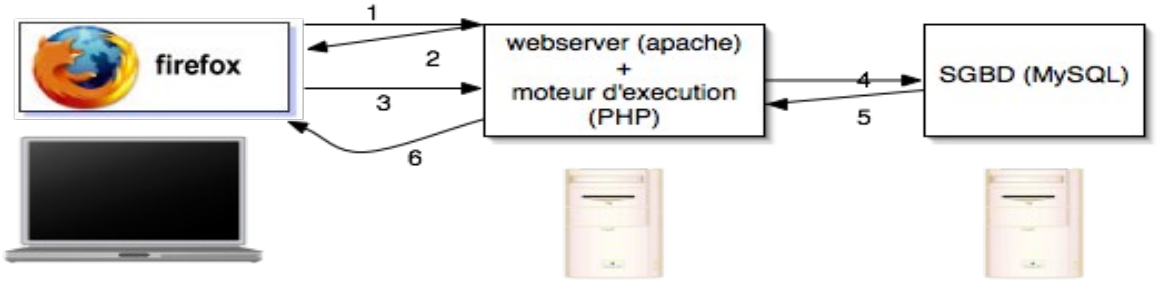
2. Architecture 3-Tier

L'architecture **3-Tier** est un modèle logique d'architecture applicative qui vise à modéliser une application comme un empilement de trois couches logicielles (ou niveaux, étages, « tier » en anglais) dont le rôle est clairement défini :

- la **présentation des données** : correspond à l'affichage, la restitution sur le poste de travail, le dialogue avec l'utilisateur ;
- La **logique applicative** (dite aussi **couche métier**): constitue les traitements nécessaires sur l'information afin de la rendre exploitable par chaque utilisateur. Note : le mot « métier » est ici à prendre au sens originel du terme, c'est à dire qui rend un service.
- l'**accès aux données persistantes** : correspond aux données qui sont destinées à être conservées sur la durée, voire de manière définitive.

Les deux **avantages principaux** sont :

- **Facilité de déploiement** : L'application en elle même n'est déployée que sur la partie serveur (serveur applicatif et serveur de base de données). Le client ne nécessite qu'une installation et une configuration minime. En effet il suffit d'installer un navigateur web compatible avec l'appication pour que le client puisse accéder à l'application, ce navigateur étant par ailleurs souvent installé par défaut sur toutes les machines. Cette facilité de déploiement aura pour conséquence non seulement de réduire le coût de déploiement mais aussi de permettre une évolution régulière du système. Cette évolution ne nécessitera que la mise à jour de l'application sur le serveur applicatif.
- **Amélioration de la sécurité** : Dans un système client-serveur tous les clients accédaient à la base de données ce qui la rendait vulnérable. Avec une architecture multi-tiers l'accès à la base n'est effectué que par le serveur applicatif. Ce serveur est le seul à connaitre la façon de se connecter à cette base. Il ne partage aucune des informations permettant l'accès aux données, en particulier le login et le password de la base. Il est alors possible de gérer la sécurité au niveau de ce serveur applicatif, par exemple en maintenant la liste des utilisateurs avec leurs mots de passe ainsi que leurs droits d'accès aux fonctions du système. On peut même améliorer encore la sécurité par la mise en place d'une architecture réseau interdisant totalement l'accès au serveur de base de données pour les utilisateurs finaux.



Exemple architecture 3 tier

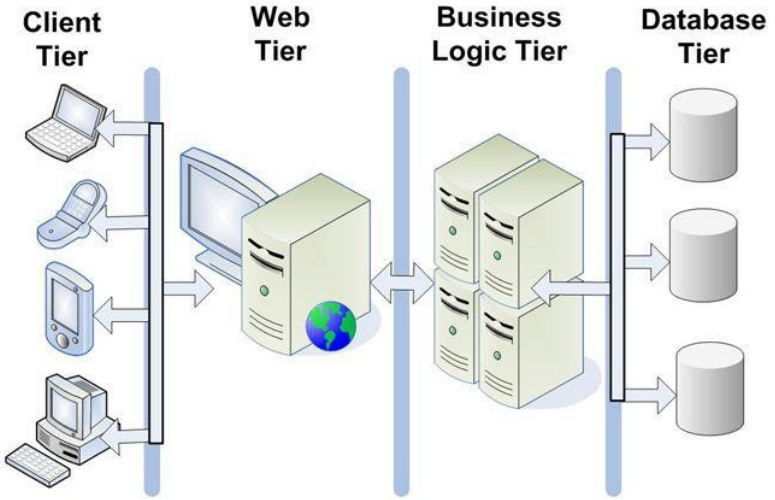
3. Architecture multi-Tier (dit aussi n tier)

Nous avons vu ce qu'était une architecture trois-tiers, une architecture multi-tiers va plus loin dans le découpage de l'application sur différents serveurs. Une **architecture multi-tiers est également appelé architecture distribuée** du fait de la distribution des traitements et des données sur différents serveurs. Le découpage de base du système reste toujours le même: une partie gestion de données, une partie gestion de la logique applicative et bien entendu le client assurant la présentation. Toutefois **les deux parties développées coté serveur vont pouvoir être déployées chacune sur plusieurs serveurs**. L'objectif général de ce type d'architecture est de permettre l'évolutivité du système sous plusieurs aspects: la quantité de données stockée, la disponibilité du serveur, le nombre d'utilisateurs,... Il faut également bien comprendre qu'un système conçu en architecture multi-tiers n'est pas forcément déployé sur plusieurs machines dès le départ. Toutefois son mode de programmation doit permettre de modifier le déploiement du système en cours d'exploitation par un administrateur. Le développeur doit donc rendre le système indépendant du serveur sur lequel il s'exécute. Il existe **deux types de répartition possibles** dans une **architecture distribuée**. Il est possible de répartir les données et de répartir la logique applicative. Chacune de ces deux répartitions permet de résoudre des problèmes de natures différentes. Elles peuvent donc être mises en place soit séparément soit en parallèle sur le même système.

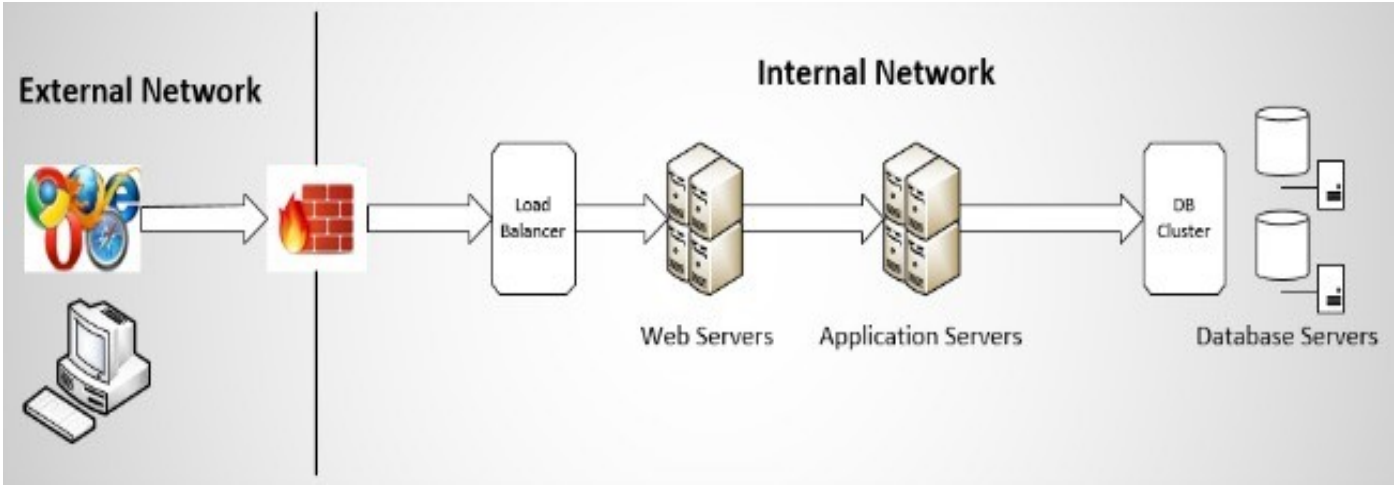
Répartition des données sur différents serveurs
Il s'agit d'utiliser plusieurs sources de données dans une même application. Chaque source possède sa propre structure de données. Chaque serveur de données peut être géré de façon très différente. Toutefois une interface de programmation commune à toutes les sources doit pouvoir exister. Il peut exister des relations entre les données des différents serveurs, il sera alors nécessaire de gérer des transactions distribuées entre des différents serveurs de données. Cette répartition de données correspond à la notion de base de données distribués. Les bases de données distribuées permettent de résoudre deux types de problèmes. La première est la performance du système: la répartition des données permet d'augmenter la disponibilité des données. Toutefois il est nécessaire de bien penser cette répartition afin de ne pas démultiplier le nombre de transactions distribuées nécessaires. Ceci aurait pour effet de diminuer la performance plus que de l'augmenter. Le deuxième type de problèmes est la réutilisation des systèmes existants. En effet de nombreux systèmes informatiques stockent actuellement une grande quantité de données. Toutefois ces systèmes ne sont pas toujours bien coordonnés entre eux. Ceci risque d'entraîner des redondances dans les données et des incohérences entre plusieurs sources de données. L'objectif est donc de réutiliser ces systèmes dans une même application afin de restructurer le système d'information sans pour autant repartir de zéro (ce qui est non seulement un non sens mais est aussi impossible). Les systèmes réutilisés sont bien souvent hétéroclites (mainframe, SGBDR,...) et nécessitent d'être réunis en utilisant diverses technologies (moniteurs transactionnels, serveurs d'applications,...).

Répartition de la logique applicative

La répartition de la logique applicative permet de distribuer les traitements sur différentes machines. Cette distribution est facilitée par l'utilisation de la programmation orientée objet et plus particulièrement de ce qu'on appelle les composants. Un composant possède entre autre la caractéristique d'être accessible à travers le réseau. Un composant peut ainsi être instancié puis utilisé au travers du réseau. Il est également possible de trouver un serveur permettant l'utilisation d'un composant, ce qui permet une forte évolutivité ainsi qu'une résistance aux pannes importantes (le service sera toujours disponible sur un serveur même si une machine tombe en panne). La réalisation de ce type de répartition nécessite l'utilisation de technologies spécifiques.



On associe à chaque couche des machines dédiés :
- Les clients gérent la présentation
- Le serveur WEB a en charge la gestion des requêtes HTTP
- La logique des traitements est assurée par une serveur d'application
- La gestion des données est assurée par un SGBD



Exemples d'architectures n tier

F. Installation de FreshRSS

Cette partie va vous permettre de faire une approche en douceur sur la mise en place d'une application web.

FreshRSS est une application libre et gratuite permettant de s'abonner à des flux RSS et de pouvoir les consulter depuis n'importe quel client. Pratique pour la veille tech.

FreshRSS devra être installé dans « **/*var*/www/rss** » et devra être accessible depuis l'URL « **https://rss.tssr.lan** ».

Prenez le temps de vous documenter sur l'installation et de préparer le terrain, notamment la partie SQL.

En effet l'application vous demandera un nom d'utilisateur, un mot de passe (ceux pour se connecter à la BDD) et le nom de la base de données. Il faudra donc créer avant une base et avec l'utilisateur privilégié associé. Pour cela :

- **Créez une conteneur sous Debian 9** qui s'appellera « **sql.tssr.lan** »
- Installez MariaDB
- Créez un utilisateur dédié à l'application avec des privilèges pouvant se connecter depuis une autre machine.

Évidemment pensez aux snapshots.

Lien : https://freshrss.org/

G. Installation de GLPI

Cette partie va vous permettre de faire une approche sur une application à la fois dite collaborative et métier dans une architecture 3-tier.

Solution libre et gratuite de gestion de parc informatique et de servicedesk, GLPI est une application Full Web pour gérer l'ensemble de vos problématiques de gestion de parc informatique : de la gestion de l'inventaire des composantes matérielles ou logicielles d'un parc informatique à la gestion de l'assistance aux utilisateurs.

Parfait, l'outil qui vous manquait !:)

GLPI devra être installé dans « **/*var*/www/glpi** » et devra être accessible depuis l'URL « **https://glpi.tssr.lan** ».

Lors de l'installation de GLPI, vous connecterez l'application avec un utilisateur dédié à la base de données qui devra tournée sur « sql.tssr.lan ».

Prenez là aussi le temps pour vous documenter sur l'installation et préparez le terrain.

Évidemment pensez aux snapshots.

Lien : http://glpi-project.org/

H. Annexes

1. TLS Handshake Protocol

La poignée de mains ! Ce protocole a pour objectif de permettre au serveur et au client de s'authentifier l'un et l'autre puis de négocier un algorithme de chiffrement et une clé cryptographique avant que l'application ne transmette son premier octet.

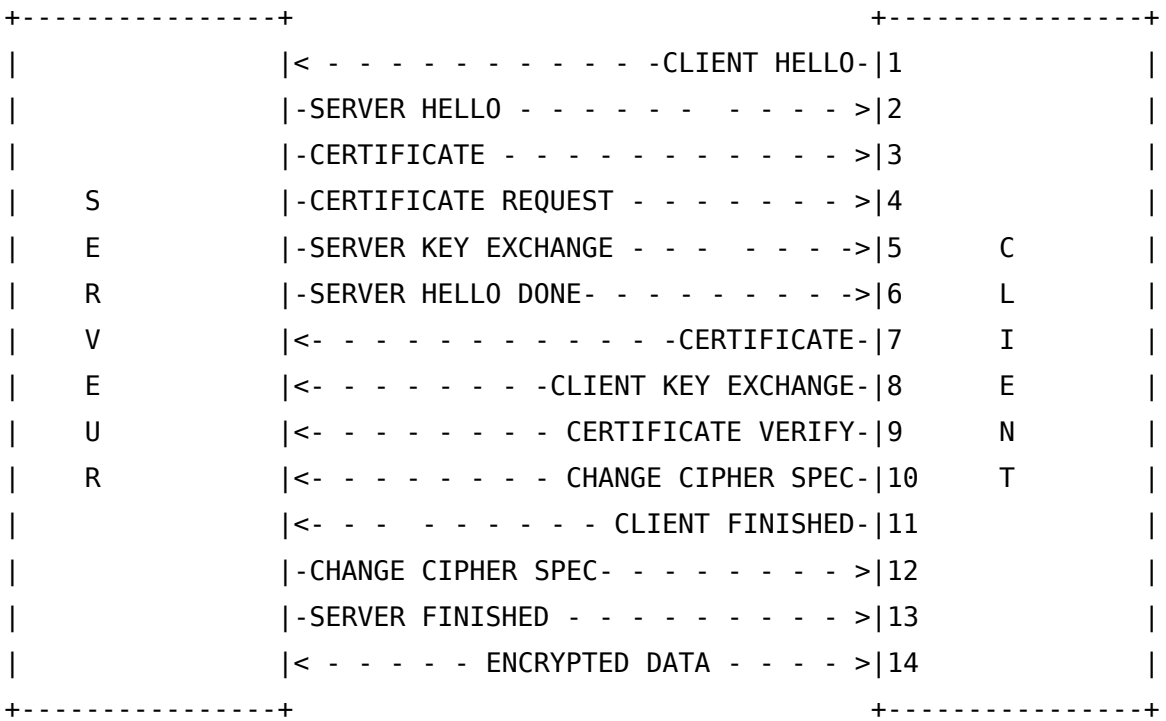
Ce protocole possède trois propriétés importantes:

1. les participants sont authentifiés en utilisant un algorithme asymétrique
2. la négociation de la clé symétrique est sécurisée. Ceci pour prévenir des attaques de types man-in-the-middle.
3. la négociation est sûre. un intrus ne peut pas modifier les informations de négociation sans être repéré par une des parties.

Nous pouvons donc, grâce à ce protocole, établir des communications privées.

Mais pour celà, les messages constituant une sessions de handshake TLS doivent être présentés dans un ordre bien précis; dans le cas contraire il en résulte une erreur fatale d'un des deux systèmes.

Voici comment se passe le handshake, dans l'ordre chronologique de haut en bas:



*** CLIENT HELLO**

envoi de la version maximale supportée (TLS = 1.3 au moment où j'écris ces lignes), de la suite d'algorithmes supportés (par ordre de préférence décroissant) et une valeur aléatoire de 32 octets.

*** SERVER HELLO**

choix de la version, de la suite d'algorithmes (Cipher Suite) et d'une valeur aléatoire.

*** CERTIFICATE (optionnel)**

envoi d'une chaine de certificats par le serveur. Le premier certificat est celui du serveur, le dernier est celui de l'autorité de certification (voir chapitre 3)

*** CERTIFICATE REQUEST (optionnel)**

demande un certificat au client pour l'authentifier

*** SERVER KEY EXCHANGE (optionnel)**

message complémentaire pour l'échange des clés. Ce message contient la clé publique du serveur utilisée par le client pour chiffrer les informations de clé de session

*** SERVER HELLO DONE**

fin des émissions du serveur

*** CERTIFICATE (optionnel)**

certificat éventuel du client si le serveur demande une authentification

*** CLIENT KEY EXCHANGE**

le client produit un secret pré-maître (encrypted pre-master key) et le crypte avec la clé publique du certificat du serveur. Ces informations sont chiffrées une deuxième fois avec la clé publique du serveur (et non la clé publique du certificat du serveur) reçue dans le message SERVER KEY EXCHANGE.

*** CERTIFICATE VERIFY (optionnel)**

message contenant une empreinte (hash) signée numériquement et créé à partir des informations de clé et de tous les messages précédents. Ce message permet de confirmer au serveur que le client possède bien la clé privée correspondant au certificat client (message 7)

*** CHANGE CIPHER SPEC**

passage du client en mode chiffrée avec la clé master comme clé symétrique

*** CLIENT FINISHED**

fin des émissions du client, ce message est chiffré à l'aide des paramètres de la suite de chiffrement.

*** CHANGE CIPHER SPEC**

passage du serveur en mode chiffrée avec la clé master

*** SERVER FINISHED**

confirmation au client du passage en mode chiffré. Ce message est chiffré à l'aide des paramètres de la suite de chiffrement

*** ENCRYPTED DATA**

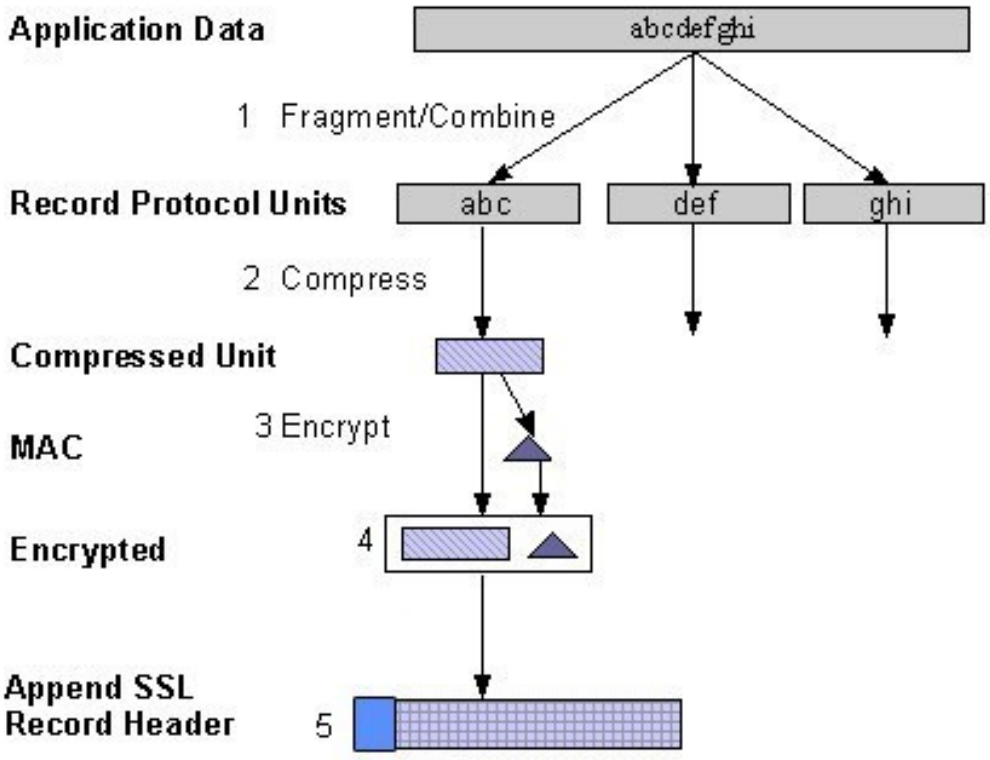
le tunnel TLS est établit, c'est maintenant le TLS Record Protocol qui prend le relais pour chiffrer les données.

2. TLS Record Protocol

Ce protocole a pour buts :

- **Encapsulation** - permet aux données SSL / TLS d'être transmises et reconnues sous une forme homogène ;
- **Confidentialité** - assure que le contenu du message ne peut pas être lu par un tiers : les données sont chiffrées en utilisant les clés produites lors de la négociation ;
- **Intégrité et Identité** - permet de vérifier la validité des données transmises, grâce aux signatures HMAC : cette signature est elle aussi générée à l'aide des clés produites lors de la négociation.

Voici en détail comment se déroule le record protocole :



1. Segmentation - les données sont découpées en blocs de taille inférieure à 16 384 octets ;
2. Compression - les données sont compressées en utilisant l'algorithme choisi lors de la négociation. A noter : à partir de SSL 3.0, il n'y a plus de compression. Et SSLv3 n'est plus utilisé.

3. Signature HMAC (0, 16 ou 20 octets) - une signature des données est générée à l'aide de la clé MAC.
4. Chiffrement - le paquet obtenu est chiffré à l'aide de la fonction récupérée lors de la négociation ;
5. Un en-tête de 5 octets est ajouté. Le champ "Type" de cet en-tête définit le type du protocole de niveau supérieur au Record Protocol. Les types sont les suivants:

```
Content-Type (1 octet) – Indique le type de paquet SSL et TLS contenu
dans l'enregistrement :
0x20 – Paquet de type Change Cipher Spec
0x21 – Paquet de type Alert
0x22 – Paquet de type Handshake
0x23 – Paquet de type Application Data : ce type correspond aux
données effectives de la transaction SSL.
```

À la réception des paquets, le destinataire effectue les opérations suivantes :

1. Vérification de l'en-tête SSL/TLS
2. Déchiffrage du paquet
3. Vérification du champ HMAC (en appliquant la même fonction que celle décrite ci-dessus, aux données déchiffrées puis en comparant le résultat au HMAC reçu)
4. Ré-assemblage des parties

Si ça se passe mal au cours de ces vérifications, alors une alarme est générée.

Je vous renvoie vers cet article récent qui donne un bon état des lieux sur SSL/TLS : https://openweb.eu.org/articles/https-de-ssl-a-tls-1-3