www.alcove.fr

Maîtriser Apache

Benjamin Drieu

Benjamin.Drieu@alcove.fr

version 1.13

l'informatique est libre

Alcôve

Alcôve
Centre Paris Pleyel
153 bd Anatole France
93200 Saint-Denis, France

Tél. : +33 1 49 22 68 00
Fax : +33 1 49 22 68 01
E-mail : alcove@alcove.fr, Toile : www.alcove.fr

Table des matières

Les principes de base

Présentation d'Apache

Objectifs de cette section

- rappeler le fonctionnement de l'Internet
- comprendre le principe d'un serveur web
- comprendre les concepts de l'informatique libre
- avoir un apperçu de la nature d'Apache

Qu’est-ce qu’un serveur web ?

Le réseau Internet a été crée en 1973, dans un souci de distributivité et d’interconnexion globale des réseaux. C’est un projet militaire.

Souci d’hétérogénéité du logiciel et du matériel.

Protocole commun à tous les services de l’Internet : TCP/IP. Modèle client/serveur. Les protocoles de l’Internet sont ouverts et indépendants (RFC, IETF).

L’Internet n’est pas le Web. Parmi les protocoles de l’Internet : FTP (transfert de fichiers), SMTP (mail), NNTP (news), gopher (bibliothèque), HTTP, etc.

Principe de l’hypertexte : collection de données distribuées et liées les unes avec les autres. La lecture passe de linéaire à interactive.

Le protocole gopher est une implémentation de l’hypertexte. Il s’agit d’héberger une collection de documents sur un serveur.

Les limitations de gopher (pas de liens externes, formatage minimum) sont comblées par HTTP (Hyper Text Transfert Procotol), qui est le protocole du Web.

HTTP permet (entre autres) d'effectuer des liens d'une collection de documents (site) à une autre. Principe de l'URL HTTP :

```
http ://site :port/chemin
```

HTTP est un protocole dit *stateless* (sans état), ce qui implique qu'il est nécessaire d'établir une connexion à chaque requête et qu'un état ne peut pas être maintenu par le protocole.

Exemple de requête HTTP :

```
$ telnet www.apache.org 80
Connected to www.apache.org.
Escape character is '^ ]'.
GET /index.html HTTP/1.0
```

HTTP est implémenté sur NeXT par Tim Berners Lee au CERN en 1990.

En 1993, le NCSA publie Mosaic, premier navigateur web en mode graphique et indépendant de la plate-forme.

Fin 1993, il y avait 200 sites sur la planète !

Aujourd'hui, de nombreux serveurs web existent. Parmis les plus utilisés :

– Apache (Apache Group)
– IIS (Microsoft)
– Netscape Enterprise (Netscape)
– ...

Apache est aujourd'hui le serveur le plus utilisé au monde (presque 63% en juillet 2000), mais c'est surtout un **logiciel libre** .

Le logiciel libre

Le logiciel libre garantit plusieurs libertés à l'utilisateur :

– la liberté d'exécuter ;
– la liberté de redistribuer ;
– la liberté de modifier ;
– la liberté de distributer ses modifications.

L'arme légale du logiciel libre est la licence du logiciel, dont la GNU GPL, qui est la licence du projet GNU. Le principe est d'utiliser le copyright à rebours pour protéger le logiciel de l'appropriation.

Le projet GNU a démarré en 1984 à l'initiative de Richard Stallman, programmeur au MIT. Le but de ce projet est de développer un système d'exploitation complet, compatible UNIX et ne comportant que des logiciels libres.

Aujourd'hui, ce système (GNU/Linux) est pratiquement terminé et utilise Linux comme noyau.

Premier contact avec Apache, ses caractéristiques

Apache est un dérivé du serveur NCSA (le premier serveur web), qui a cessé progressivement d'être maintenu. Apache était à l'origine une collection de modifications (*patches*) appliquées au serveur NCSA puis regroupées de manière cohérente (*a « patchy » server*).

C'est aujourd'hui le serveur le plus utilisé sur l'Internet (63% en juillet 2000 selon Netcraft).

Il est développé par l'Apache Group, qui est un organisme indépendant chargé de maintenir et d'améliorer le serveur Apache et des projets connexes (mod_perl, JServ, etc.).

Parmi les avantages d'Apache :

– de bonnes performances ;

– c'est un logiciel libre ;

– le développement est actif ;

– très portable (il tourne sur la plupart des UNIX et même sur Windows NT) ;

– extensible, modulaire et configurable.

Rappels des objectifs de cette section

– rappeler le fonctionnement de l'Internet
– comprendre le principe d'un serveur web
– comprendre les concepts de l'informatique libre
– avoir un apperçu de la nature d'Apache

Mise en oeuvre du serveur

Objectifs de cette section

– comprendre les systèmes de distribution d'Apache
– comprendre les mécanismes de l'installation d'Apache
– comprendre les procédures de démarrage et d'arrêt du serveur

Son installation, sa configuration

Apache est un logiciel distribué soit sous forme de code source, soit sous forme d'archive binaire. Son architecture modulaire rend la compilation souvent non nécessaire.

Le site de référence d'Apache est `http ://www.apache.org/`. Il existe des sites miroirs un peu partout sur la planète (dont plusieurs en France).

Distribution de sources

La distribution de sources est effectuée sous forme d'archives *tar* . Par exemple `apache-1.3.12.tar.gz`.

Le décompactage est effectué avec l'outil éponyme : `tar xvfz apache-1.3.12.tar.gz`.

Les étapes suivantes sont la configuration et la compilation :

```
./configure --prefix=/usr/local

make && make install
```

Distribution de paquets

Toutes les distributions ont leur propre système de paquetage :

– Debian : `apt-get install apache`

– RedHat/Mandrake : `rpm -i apache-1.3.12_1.rpm`

– ...

Notion de modules, leur utilisation

L'architecture d'Apache permet l'intégration de modules au code, ce qui autorise l'ajout de fonctionnalités sans recompilation du noyau d'Apache.

Il est ainsi possible d'étendre les possibilités d'Apache sans intégrer ses modifications au noyau ni sous forme de *patches* .

Le chargement dynamique des modules est renseigné dans les fichiers de configuration du serveur.

Architecture du serveur

Une requête arrivant au serveur est traitée grossièrement de la manière suivante :

- transformation du chemin en nom de fichier
- authentification si nécessaire
- résolution du type MIME requis
- envoi de la réponse au client
- écriture dans les fichiers de rapport

Chacune de ces phases est gérée par un ou plusieurs *handlers* , qui se chargent de traiter cette phase. Un handler est purement et simplement une fonction déclenchée par Apache lorsqu'il l'estime pertinent. Le handler déclenché peut être positionné dans les fichiers de configuration.

Protocoles utilisés par le serveur

Le serveur Apache utilise le protocole HTTP, mais aussi le protocole HTTPS (protocole crypté, si adjonction du module mod_ssl).

Apache est de plus capable de se comporter en proxy HTTP et peut donc effectuer des requêtes HTTP sortantes.

Apache utilise aussi la CGI pour communiquer avec des processus externes.

Arrêter et redémarrer Apache

Il existe plusieurs moyens d'arrêter et de redémarrer Apache :

– exécuter le binaire `httpd` et lui envoyer des signaux pour qu'il s'arrête
– utiliser l'outil apachectl fourni avec la distribution
– utiliser les scripts de démarrage et d'arrêt dans `/etc/init.d`

Utiliser httpd

Le serveur Apache est compilé dans un binaire nommé `httpd` ou `apache` en fonction de la distribution.

Pour exécuter le serveur : `/usr/local/apache/bin/httpd`

Pour arrêter le serveur (exemple) : `kill -TERM 1426`

Pour redémarrer le serveur (exemple) : `kill -HUP 1426`

Utiliser apachectl

Apache est livré avec un script qui permet d'abstraire les tâches de démarrage et d'arrêt. Le script apachectl agit en tant que front-end et agit de manière différente en fonction des arguments qu'on lui passe.

Pour exécuter le serveur : `apachectl start`

Pour arrêter le serveur : `apachectl stop`

Pour redémarrer le serveur : `apachectl restart`

Utiliser /etc/init.d

Lorsqu'on désire exécuter des services au démarrage d'un système UNIX, l'usage est de placer des scripts de démarrage dans l'arborescence `/etc/init.d` (ce nom est dépendant du système). Ces scripts reconnaissent les arguments start, stop, restart et quelques autres.

Pour exécuter le serveur : `/etc/init.d/apache start`

Pour arrêter le serveur : `/etc/init.d/apache stop`

Pour redémarrer le serveur : `/etc/init.d/apache restart`

Échec au lancement

En cas d’erreur de syntaxe dans les directives de configuration d’Apache, un message d’erreur apparaît et Apache refuse de démarrer.

La résolution passe par l’examen des fichiers de rapport (voir plus loin) et par l’utilisation de la commande `apachectl configtest`, qui produit un diagnostic en cas d’erreur.

Rappel des objectifs de cette section

– comprendre les systèmes de distribution d'Apache
– comprendre les mécanismes de l'installation d'Apache
– comprendre les procédures de démarrage et d'arrêt du serveur

Administration et optimisation

Objectifs de cette section

- comprendre l'organisation de la distribution
- comprendre le principe de la directive
- connaître les fichiers de configuration d'Apache
- passer en revue les directives principales

Organisation de la distribution

Apache est organisé sous deux racines :

– **DocumentRoot** : c'est le répertoire sous lequel une partie les documents publiés via HTTP sont disponibles. Certains documents (les scripts CGI par exemple) sont placés à une autre position de la distribution ;

– **ServerRoot** : c'est le répertoire sous lequel sont normalement placés les fichiers de configuration, les fichiers journaux (fichiers de log) ainsi que d'autres informations.

Les distributions de GNU/Linux redéfinissent cet organisation afin de satisfaire leurs normes de développement.

Organisation de la distribution

Une distribution d'Apache est organisée sous le répertoire `/usr/local/apache` (*ServerRoot*) et contient les répertoires suivants ainsi que quelques autres :

– `bin` : le programme `httpd` et des utilitaires
– `cgi-bin` : des scripts CGI
– `conf` : les fichiers de configuration, qui contiennent des directives de configuration qui seront lues au démarrage du serveur
– `htdocs` : il s'agit de la racine *DocumentRoot* , qui contient les fichiers publics
– `logs` : les fichiers de rapport, qui contiennent des informations sur toutes les requêtes adressées au serveur

Une approche commune : principe des directives

Les *directives de configuration* sont placées dans les fichiers de configuration du répertoire `conf`. Elles sont interprétées au démarrage du serveur web et lorsque l'administrateur demande au serveur de les recharger.

Les fichiers de configuration d'Apache sont au nombre de quatre :

- `httpd.conf`
- `srm.conf`
- `access.conf`
- `mime.types`

Depuis Apache 1.3.6, ces trois fichiers ont étés regroupés dans le fichier `httpd.conf`, mais les distributions paquetées gardent généralement la compatibilité.

`httpd.conf`

C’est le fichier principal d’Apache, c’est aussi le seul qui peut être spécifié à la ligne de commande et c’est lui qui détermine où aller chercher les deux autres fichiers de configuration.

Il contient normalement les directives spécifiant les paramètres du serveur relatifs à TCP/IP (port utilisé, serveurs virtuels, etc.), à la performance (nombre de serveurs lancés, de serveur inactifs, etc.) et d’autres directives.

`srm.conf`

C’est le fichier qui contient les directives déterminant la forme par laquelle les ressources du serveur seront acheminées vers le client (formatage des répertoires listés, option multilingue, *aliases* , etc.).

`access.conf`

C’est le fichier qui détermine les conditions d’accès aux répertoires et aux différents URLs du serveur web. C’est à cet endroit qu’on place les restrictions d’accès et les directives d’authentification.

`.htaccess`

`.htaccess` n’est pas à proprement parler un fichier de configuration mais un ensemble de fichiers dont nous reparlerons par la suite.

Lorsqu’un administrateur du site a besoin de placer des directives de configuration pour une partie du site mais n’a pas la permission de modifier la configuration du serveur, il a la possibilité de créer un fichier `.htaccess` dans un répertoire visible du serveur. Lorsqu’une requête aboutit à ce répertoire, ce fichier de configuration est lu par le serveur à chaque requête.

Les directives de ces fichiers sont dans un format standard UNIX :

```
Directive valeur1,valeur2,...
```

Certaines directives sont des *blocs* (ou conteneurs) et ont une syntaxe proche de l'HTML :

```
< Directive >
...contenu...
</ Directive >
```

La plupart des 200 directives fournies par Apache en standard peuvent être utilisées à plusieurs niveaux du fichier de configuration (notion de contexte). Notons que chaque directive fournit une valeur par défaut.

Étude des directives utilisées

ServerRoot

– Syntaxe : `ServerRoot chemin`
– Exemple : `ServerRoot /usr/local/apache`

Cette directive permet de spécifier la racine du serveur web. Tous les chemins relatifs des fichiers de configuration le seront à partir de ce chemin.

ResourceConfig

– Syntaxe : ResourceConfig fichier
– Exemple : ResourceConfig conf/srm.conf

Cette directive spécifie le fichier de configuration de ressources (`srm.conf`) chargé au démarrage du serveur.

AccessConfig

- Syntaxe : AccessConfig fichier
- Exemple : AccessConfig conf/access.conf

Cette directive spécifie le fichier de configuration de l'accès aux ressources (`access.conf`) chargé au démarrage du serveur.

PidFile

- Syntaxe : PidFile *fichier*
- Exemple : PidFile logs/httpd.pid

Cette directive spécifie le fichier dans lequel Apache va stocker le numéro de processus du serveur. Ce fichier permet d’envoyer facilement des signaux au serveur.

User

- Syntaxe : User *id*
- Exemple : User www-data

Cette directive spécifie l'utilisateur UNIX qui fera tourner les processus du serveur. Préfixé par un dièse, la valeur de la directive est un numéro d'id UNIX plutôt qu'un nom d'utilisateur.

Note : cette directive n'est applicable que lorsque le serveur est lancé en tant qu'utilisateur root.

Group

– Syntaxe : Group *id*
– Exemple : Group www-data

Cette directive spécifie le groupe UNIX qui fera tourner les processus du serveur. Préfixée par un dièse, la valeur de la directive peut-être un numéro d'id UNIX plutôt qu'un nom d'utilisateur.

Note : cette directive n'est applicable que lorsque le serveur est lancé en tant qu'utilisateur root.

ServerAdmin

– Syntaxe : ServerAdmin *email*
– Exemple : ServerAdmin foo@bar.com

Cette directive spécifie l'adresse email de l'administrateur du site web. Elle apparaît par exemple dans les pages générées en cas d'erreur.

ServerName

– Syntaxe : ServerName *nom*
– Exemple : ServerName www.alcove.fr

Cette directive spécifie le nom du serveur web, qui doit être un nom visible de la machine l'hébergeant. Il apparaît par exemple dans les redirections HTTP et le positionnement d'une valeur erronée peut causer des problèmes à la navigation.

DocumentRoot

– Syntaxe : DocumentRoot *chemin*
– Exemple : DocumentRoot /usr/local/apache/htdocs

Cette directive spécifie la racine de la partie visible du serveur web, où les documents HTML servis seront placés.

Note : il est important de ne pas placer de fichiers sensibles sous cette racine.

DefaultType

– Syntaxe : DefaultType *type-mime*
– Exemple : DefaultType text/plain

Cette directive spécifie quel type mime sera retourné en cas de document de type inconnu. Le type text/plain permet de l'afficher tel quel dans à peu près n'importe quel navigateur.

Rappel des objectifs de cette section

– comprendre l’organisation de la distribution
– comprendre le principe de la directive
– connaître les fichiers de configuration d’Apache
– passer en revue les directives principales

Architecture modulaire d’Apache

Objectifs de cette section

– comprendre le fonctionnementr modulaire d’Apache
– être capable de charger les modules du serveur

L'architecture modulaire d'Apache permet l'ajout de fonctionnalités par l'utilisation de modules, qui sont simplement des ensembles de fonctionnalités regroupées dans un binaire. L'ajout de modules ajoute de plus des directives au jeu des directives déjà reconnues par Apache.

Il y a deux façons d'utiliser les modules :
– compiler le module au sein du binaire Apache
– compiler le module et le faire charger par Apache au démarrage

LoadModule

– Syntaxe : LoadModule *module fichier*
– Exemple : LoadModule status_module modules/mod_status.so

Dans le cas d'un module chargé dynamiquement, nécessité d'utiliser la directive LoadModule, qui charge un fichier objet (ou une DLL sous MS-Windows) et l'inclut dans l'espace de nommage du binaire Apache.

AddModule

– Syntaxe : AddModule *module*
– Exemple : AddModule mod_access

AddModule permet d'activer un module chargé dans le binaire du serveur Apache mais non précédemment activé (par exemple suite à l'utilisation de la directive ClearModule, qui désactive l'ensemble des modules chargés).

<IfModule>

– Syntaxe : <IfModule [!] *module* > ... </IfModule>
– Exemple : <IfModule mod_autoindex.c> ... </IfModule>

Cette directive est un bon exemple de conteneur. Elle permet de n'interpréter qu'une partie du fichier de configuration au démarrage d'Apache.

Comme l'adjonction de modules ajoute des directives à Apache, la suppression d'un module précédemment utilisé dans le fichier de configuration implique l'annulation des directives se reposant sur ce dernier et produit donc une erreur au démarrage. L'utilisation de la directive <IfModule> permet de s'assurer que les directives utilisant un module ne seront exécutées que si celui-ci est chargé.

Rappel des objectifs de cette section

- comprendre le fonctionnementr modulaire d’Apache
- être capable de charger les modules du serveur

Gestion des droits des répertoires

Objectifs de cette section

- comprendre la problématique des droits des répertoires
- comprendre les options de paramétrage des répertoires
- mettre en place un système de pages personnelles

Gestion des droits des répertoires

Une problématique complexe :

– donner des possibilités au client mais limitées et propres à une partie du contenu
– donner des possiblités aux administrateurs mais limitées

Apache résoud ce problème en introduisant des directives de paramétrage des fonctionnalités spécifiques aux répertoires de l’arborescence.

<Directory>

– Syntaxe <Directory *répertoire* > ... </Directory>
– Exemple <Directory /usr/local/apache/htdocs> ... </Directory>

Cette directive de bloc (ou conteneur) spécifie que les directives contenues entre son début et sa fin seront applicables seulement au répertoire passé en argument. L'argument passé à la directive <Directory> est un répertoire dans lequel elle s'appliquera.

Attention, toutes les directives ne peuvent pas être utilisées dans ce contexte.

<Location>

– Syntaxe <Location répertoire> ... </Location>
– Exemple <Location /usr/local/apache/htdocs> ... </Location>

Cette directive de bloc (ou conteneur) spécifie que les directives contenues entre son début et sa fin seront applicables seulement à l'URL passée en argument. L'argument passé à la directive <Location> est une URL dans laquelle elle s'appliquera.

La différence par rapport à la directive <Directory> est que <Location> s'utilise pour une URL visible de l'extérieur du site. Elle peut donc s'appliquer à un répertoire « virtuel » (par exemple cgi-bin ou les répertoires des utilisateurs).

Options

– Syntaxe : Options [+|-]option
– Exemple : Options ExecCGI Indexes

Cette directive permet de spécifier quelles fonctionnalités seront disponibles dans le contexte où elle est utilisée (généralement <Directory> et <Location>).

Les options disponibles sont :

- All : toutes les options disponibles sauf MultiViews
- ExecCGI : exécution des scripts CGI
- FollowSymLinks : lecture des liens symboliques
- Includes : utilisation des Server-Side Includes
- IncludesNOEXEC : utilisation des Server-Side Includes mais pas d'exécution de programme
- Indexes : création d'un index des fichiers du répertoire
- MultiViews : utilisation des procédures multilingues
- SymLinksIfOwnerMatch : lecture des liens symboliques effectués sur un fichier appartenant au créateur du lien seulement.

.htaccess

Pour donner quelques droits aux utilisateurs du serveur, Apache permet l'évaluation à la volée (lors de la réponse à une requête) d'un fichier de configuration situé dans le répertoire où la requête aboutit. Ce fichier est le fichier **.htaccess** .

La plupart des directives de configuration peuvent être placées dans un fichier .htaccess, qui se comporte exactement comme une directive <Directory>.

AllowOverride

– Syntaxe : AllowOverride *option*
– Exemple : AllowOverride None

Pour des raisons de sécurité, il est possible de n'autoriser les utilisateurs qu'à effectuer un paramétrage limité sur leurs répertoires, en utilisant la directive AllowOverride. Elle peut prendre en argument une valeur parmi :

– AuthConfig : autorise l'utilisation des directives d'authentification
– FileInfo : autorise le paramétrage des documents retournés (multilinguisme, document d'erreur, etc.)
– Limit : autorise le contrôle d'accès
– Options : autorise l'utilisation de la directive Option

Gestion des index des répertoires

Gestion des index des répertoires

Si l'URL consultée ne comporte pas de fichier HTML mais uniquement un nom de répertoire, Apache adopte le comportement suivant :

– s'il existe un fichier d'index dans le répertoire, celui-ci est affiché

– sinon le serveur affiche une page d'index à la volée

Par exemple, pour la requête `http ://www.foobar.org/rep/` :

– si `/rep/index.html` existe, on affiche `http ://www.foobar.org/rep/index.html`

– sinon, on génère un index automatique

Cette page convient à la plupart des utilisations, mais il est possible de la paramétrer assez finement.

DirectoryIndex

– Syntaxe : DirectoryIndex *fichiers*
– Exemple : DirectoryIndex index.html index.cgi /cgi-bin/error.cgi

La directive DirectoryIndex permet de spécifier le nom des fichiers d'index, qui seront affichés lors du chargement d'un répertoire, s'ils existent. La vérification se fait de gauche à droite.

Dans l'exemple cité, on termine par un appel à une URL absolue, qui permet de charger une page par défaut pour tous les répertoires ne contenant pas d'index.

IndexOptions

– Syntaxe : IndexOptions *options*
– Exemple : IndexOptions FancyIndexing ScanHTMLTitles

Cette directive permet de spécifier les options qui seront utilisées lors de l'affichage d'un répertoire, à la manière de la directive Options. Quelques options parmi les plus utilisées :

– FancyIndexing : active l'utilisation des directives
– ScanHTMLTitles : affiche le titre des fichiers HTML à la place dans le champ description
– SuppressLastModified : n'affiche pas la date de dernière modification du fichier
– SuppressSize : n'affiche pas la taille du fichier

AddDescription

– Syntaxe : AddDescription *chaîne fichier*
– Exemple : AddDescription "Jolie image" *.jpeg

Cette directive permet de modifier le texte descriptif associé à un fichier du répertoire. Ce texte est cependant limité à 23 caractères par défaut, mais il peut comporter de l'HTML.

AddIcon

– Syntaxe : AddIcon *icône fichiers*
– Exemple : AddIcon (IMG,/icons/image.xbm) .gif .jpg .xbm

Cette directive permet l'affichage d'une icône pour un fichier (le choix de l'icône est basé sur son extension ou sur son nom, avec possiblement des méta-caractères).

L'icône est de la forme *(alt,image)* , où *alt* représente la chaîne de caractères affichée dans les navigateurs textes et en « tooltip ».

AddIconByType

– Syntaxe : AddIconByType *icône type-mime*
– Exemple : AddIconByType (IMG,/icons/image.xbm) image/*

Cette directive est très proche de la précédente, hormis qu'elle se base sur les types MIME des fichiers plutôt que sur leurs extensions.

DefaultIcon

– Syntaxe : DefaultIcon *fichier*
– Syntaxe : DefaultIcon /icon/unknown.xbm

Cette directive spécifie l'icône utilisée par défaut dans le cas où aucune ne peut être trouvée pour un fichier.

HeaderName

– Syntaxe : HeaderName *fichier*
– Exemple : HeaderName HEADER

Cette directive permet de spécifier le nom d’un fichier qui sera lu et affiché au dessus de la liste des fichiers d’un répertoire (à la manière d’un site FTP).

ReadmeName

– Syntaxe : ReadmeName fichier
– Exemple : ReadmeName fichier

Cette directive permet de spécifier le nom d’un fichier qui sera lu et affiché en dessous de la liste des fichiers d’un répertoire (à la manière d’un site FTP).

IndexIgnore

– Syntaxe : IndexIgnore *fichiers*

– Exemple : IndexIgnore .htaccess *˜ README

Cette directive permet de ne pas afficher un ensemble de fichiers dans la liste du contenu d’un répertoire. Les noms de fichier peuvent utiliser des méta-caractères.

Activation de l’index

L’index s’active par le biais de l’utilisation de la directive `Options Indexes`. Ainsi, par exemple :

```
<Location /rep>

Options Indexes

</Location>
```

Pages personnelles

Apache permet la déclaration d'arborescences personnelles spécifiques aux utilisateurs et ainsi la définition d'URL commençant par un tilde, suivi par le nom de l'utilisateur.

Généralement, les pages personnelles sont placées sous le répertoire personnel des utilisateurs UNIX, mais il est possible d'effectuer des redirections sur des sites extérieurs.

UserDir

– Syntaxe : UserDir *répertoire* | *url* |disabled [*utilisateur*]
– Exemple : UserDir public_html

Cette directive détermine le répertoire personnel des utilisateurs. La valeur **disabled** suivie d'une liste d'utilisateurs supprime l'utilisation des répertoires personnels pour ces derniers.

Si le répertoire est un chemin relatif, il le sera par rapport au répertoire personnel de l'utilisateur (généralement /home/ *utilisateur*).

Il est possible d'utiliser l'étoile comme méta-caractère et d'utiliser des URL pour effectuer une redirection HTTP.

Rappel des objectifs de cette section

– comprendre la problématique des droits des répertoires
– comprendre les options de paramétrage des répertoires
– mettre en place un système de pages personnelles

Apache multisite

Objectifs de cette section

- comprendre les problématiques de la configuration d'Apache en multisite
- connaître les différentes stratégies possibles et savoir laquelle adopter
- comprendre le principe de l'hébergement virtuel de masse

Apache multisite

Le multisite ou *virtual hosting* est une technique permettant de disposer de plusieurs serveurs webs sur une même machine.

Les gains sont évidents :
– économie de machines sur les petits sites
– administration unique et centralisée

En revanche, le virtual hosting possède des inconvénients :
– contraintes de mise en place
– problèmes de sécurité entre les différents sites

Techniques de virtual hosting

Il existe plusieurs techniques de virtual hosting :

– le virtual hosting basé sur IP

– le virtual hosting basé sur le nom

Virtual hosting basé sur IP

Le principe est de disposer de plusieurs interfaces réseau avec chacune une adresse IP. On peut définir sous Linux des interfaces réseau virtuelles rattachées à une interface réseau physique (avec une limite de 256). Par exemple :

```
ifconfig eth0 :0 192.168.1.12.
```

Le but est de faire fonctionner des serveurs Apache sur plusieurs adresses IP différentes. Deux solutions sont envisageables :

– faire fonctionner plusieurs jeux de serveurs différents, chacun sur une adresse IP
– faire fonctionner un jeu de serveurs mais sur plusieurs adresses IP

Virtual hosting basé sur IP

Dans la première solution, on dispose de plusieurs jeux de serveurs. Chaque serveur a une configuration propre et on détermine sur quel interface il va attendre (*écouter*) des connexions par l'utilisation de la directive **Listen** , positionnée dans le fichier de configuration httpd.conf.

Exemple : Listen 10.16.110.19 :80

Avantages :

– meilleure sécurité : chaque site fonctionne sur son propre serveur

– configuration propre à chaque serveur

Inconvénients :

– plus de ressources occupées car plusieurs serveurs

Virtual hosting basé sur IP

Dans la deuxième solution, on lance un seul jeu de serveurs Apache, mais qui écoute sur plusieurs adresses IP. La configuration de tous les sites est commune mais nécessite l'utilisation d'un conteneur spécial (<VirtualHost>) pour spécifier les directives de configuration propres aux serveurs virtuels.

L'utilisation de la directive Listen permet de plus de n'écouter que sur certaines adresses IP. Par défaut, Apache écoute sur toutes les interfaces réseau.

<VirtualHost>

– Syntaxe : <VirtualHost adresse_ip> ... </VirtualHost>
– Exemple : <VirtualHost 10.16.110.19> ... </VirtualHost>

Ce conteneur permet de n'appliquer les directives contenues que pour un seul serveur virtuel, basé sur l'interface réseau passée en argument

C'est par le biais de <VirtualHost> qu'on paramètre les différents serveurs virtuels. Parmi les directives les plus couramment spécifiées : DocumentRoot et ServerName.

Une directive non spécifiée dans un conteneur <VirtualHost> hérite du paramétrage du serveur.

Virtual hosting basé sur le nom

On utilise une extension du protocole HTTP version 1.1 qui permet d'envoyer avec la requête HTTP (dans le champ *Host*) le nom du serveur qu'on désire contacter. Il est ainsi possible de ne plus effectuer de distinction sur l'adresse IP de destination mais sur ce champ.

Exemple de requête HTTP/1.1 :

```
GET /index.html HTTP/1.1

Host : www.alcove.fr
```

L'avantage est qu'on n'a pas besoin d'une adresse IP par serveur (on peut donc potentiellement définir une infinité de serveurs), en revanche le client doit être compatible HTTP/1.1.

Virtual hosting basé sur le nom

La configuration est pratiquement identique à celle du virtual hosting monoserveur basé sur l'adresse IP. La manoeuvre consiste à spécifier une adresse IP sur laquelle on effectuera une « résolution » de nom et à définir autant de <VirtualHost> que désirés.

L'argument passé à <VirtualHost> est le **nom** du serveur virtuel et non pas son adresse IP.

NameVirtualHost

– Syntaxe : NameVirtualHost adresse_ip
– Exemple : NameVirtualHost 192.168.1.12

La directive NameVirtualHost permet d'activer l'utilisation du virtual hosting basé sur le nom pour l'adresse IP passée en argument. Toutes les requêtes utilisant cette adresse IP **seront** considérées comme étant des requêtes adressées à un serveur virtuel.

Faute d'utiliser cette directive, Apache résoud le nom et utilise le virtual hosting basé sur IP.

Exemple

```
NameVirtualHost 192.168.1.12
<VirtualHost 192.168.1.12>
ServerName www.foobar.org
ServerPath /foobar
DocumentRoot /htdocs/www.foobar.org
</VirtualHost>
```

Virtual hosting de masse

Le problème pour l'hébergement de sites multiples est la redondance d'informations lorsqu'ils sont pratiquement identiques. Le virtual hosting de masse permet de définir des serveurs virtuels en se basant sur des champs de la requête HTTP. Les avantages sont les suivants :

- fichiers de configuration plus petits (temps de chargement plus court et moins de mémoire utilisée) ;
- ajout et suppression de serveurs virtuels sans redémarrer ni arrêter apache

En revanche, le plus gros désavantage est l'unicité du fichier de rapport.

VirtualDocumentRoot

- Syntaxe : VirtualDocumentRoot chemin
- Exemple : VirtualDocumentRoot /www/hosts/%0/docs

Cette directive spécifie le chemin qui sera utilisé en tant que *DocumentRoot* pour toutes les requêtes sur un serveur virtuel.

La chaîne %0 est remplacée par le nom du serveur auquel la requête est adressée. Ainsi, la requête adressée à `www.foobar.org` aboutira dans `/www/hosts/www.foobar.org/docs`

VirtualScriptAlias

– Syntaxe : VirtualScriptAlias chemin
– Exemple : VirtualScriptAlias /www/hosts/%0/cgi-bin

Cette directive spécifie le chemin qui sera utilisé pour spécifier le chemin où les scripts CGI seront placés par serveur virtuel.

La chaîne %0 est remplacée par le nom du serveur auquel la requête est adressée. Ainsi, la requête adressée à `www.foobar.org` aboutira dans `/www/hosts/www.foobar.org/cgi-bin`

Rappel des objectifs de cette section

– comprendre les problématiques de la configuration d'Apache en multisite
– connaître les différentes stratégies possibles et savoir laquelle adopter
– comprendre le principe de l'hébergement virtuel de masse

Utilisation des fichiers de rapport

Objectifs de cette section

– connaître le principe des fichiers de rapport
– comprendre l'intérêt des fichiers de rapport
– savoir tirer parti des fichiers de rapport correctement
– savoir utiliser les outils d'administration de fichiers de rapport

Utilisation des fichiers de rapport

Le serveur Apache conserve une trace de chaque requête HTTP dans un fichier de rapport (ou fichier de *log* ou encore fichier de journal). L'intérêt de disposer de fichiers de rapport est multiple :

- conservation d'une trace des requêtes HTTP :
  - correction des erreurs 404 (documents non trouvés) ;
  - trace d'une éventuelle intrusion ;
  - analyse du parcours des visiteurs.
- mise en place d'outils de statistiques.

Les fichiers dans lesquels les rapports sont stockés sont dans le répertoire logs, sous DocumentRoot.

Formats des fichiers de rapport

Les fichiers de rapport sont par défaut dans un format commun, le Common Log Format ou CLF. Le format est composé de champs séparés par des espaces. Si un des champs n'a pas de valeur, il est remplacé par un tiret. Les champs du format CLF sont :

– l'hôte client (une adresse IP ou un nom)
– une vérification d'identité par identd si elle est activée (IdentityCheck)
– le nom d'utilisateur si disponible
– la date et l'heure de la requête
– la requête HTTP
– le statut de la réponse (code HTTP)
– le nombre d'octets renvoyés par le serveur (sans les en-têtes)

LogFormat

– Syntaxe : LogFormat *format* [*nom*]
– Exemple : LogFormat "%h %l %u %t \" %r \" %s %b" custom

Cette directive permet de spécifier un format de rapport et éventuellement de le nommer (utilisation du nom en deuxième argument de la directive). Le format est composé de caractères et de chaînes spéciales qui seront remplacées par des valeurs propres à la requête. Parmis les plus utilisés :

– %B : nombre d’octets envoyés dans la requête HTTP sans les en-têtes
– %h : hôte distant
– %{foobar} : le contenu de l’en-tête foobar dans la requête HTTP
– %P : le numéro de processus qui a répondu à la requête (utile pour corriger les bugs)

– %r : la première ligne de la requête HTTP
– %s : le code HTTP retourné
– %t : la date et l'heure de la requête
– %T : le temps en secondes nécessaire à la réponse à la requête
– %u : le nom utilisé en cas d'authentification (peut être faux)

Notons que chacun de ces champs peut être préfixé d'une série de conditions, qui sont des codes de retour HTTP validant ou invalidant (utilisation du point d'exclamation) l'utilisation d'un champ. Par exemple, pour afficher le nom d'utilisateur dans le cas d'une authentification **réussie** : `%!401u`.

CustomLog

– Syntaxe : CustomLog *fichier* | *programme format* / nom [env= *valeur*]
– Exemple : CustomLog log/gif.log custom env=image-gif

Cette directive permet de spécifier qu'un fichier de rapport utilisera tel ou tel format de rapport spécifié en argument (qui peut être un nom de format ou un directement un format).

Le dernier argument (facultatif) permet de spécifier un test sur une variable d'environnement. Voir la documentation du module **mod_setenvif** . Par exemple, pour stocker une trace des requêtes sur les fichiers d'images GIF :

```
SetEnvIf Request_URI gif$ image-gif

CustomLog log/gif.log common env=image-gif
```

Plutôt que de stocker les rapports dans un fichier, il est possible de les faire passer à un programme externe par le biais d'un *pipe* . Ce programme est lancé par Apache au démarrage et il lira les requêtes sur l'entrée standard.

Ce programme pourra effectuer tous les traitements possibles et imaginables (par exemple, stocker les rapports dans une base de données, les afficher en temps réel sur un support quelconque, etc.).

Par exemple : `CustomLog "|xrootconsole /dev/stdin"`

Rotation de rapports

Un des problèmes avec les sites à gros traffic est la place utilisée par les fichiers de rapport. Une politique qui prônerait l'effacement régulier des fichiers de rapport est mauvaise car elle ferait perdre des informations récentes et potentiellement importantes.

La solution est de placer les rapports dans des fichiers d'archivage au fur et à mesure de leur obsolescence. On peut par exemple utiliser un nouveau fichier de rapport chaque jour et compresser les anciens fichiers.

Une des contraintes et le moment de la rotation. Sur un site web possédant un fort traffic, l'écriture dans les fichiers de rapport est continue et la perte de requêtes est possible.

rotatelog

La solution est de ne pas écrire directement dans un fichier mais de faire appel à un programme externe qui se chargera de l'écriture dans les fichiers de rapport.

L'outil rotatelog est livré en standard avec Apache et il permet d'effectuer des rotations de rapports de manière transparente. Il lit des rapports sur l'entrée standard, se charge de les écrire dans un fichier et change de fichier au bout d'un nombre donné de secondes. La syntaxe du programme rotatelog est `rotatelog fichier temps`.

Le paramètre *fichier* est le nom de la base des fichiers de rapport

Le paramètre *temps* est le temps en secondes entre deux rotations de fichiers de rapport (par exemple 86400 pour une journée).

Ainsi, pour écrire dans des fichiers de rapports commençant par `/usr/local/apache/log/access.log` et effectuer une rotation tous les jours, il suffit de placer dans les fichiers de configuration d'Apache la ligne suivante (sans saut à la ligne) :

```
TransferLog "|rotatelogs
/usr/local/apache/log/access.log 86400"
```

Résolution de noms d’hôtes

Les clients HTTP n’ont pas l’obligation de fournir au serveur leur nom d’hôte, en revanche le serveur connaît leur adresse IP.

Une adresse IP n’est pas parlante, lors de l’analyse des fichiers de rapport, elles n’apportent pas d’informations supplémentaires (par exemple de quel pays se connectent les visiteurs), ce qui n’est pas le cas des noms d’hôtes.

La procédure de résolution d’adresse IP en nom d’hôte est lente et coûteuse, mais il est possible de l’activer dynamiquement au moment d’écrire dans les fichiers les rapports.

HostNameLookups

– Syntaxe : HostNameLookups on|off|double
– Exemple : HostNameLookups off

Cette directive permet d'activer la résolution de noms dans les fichiers de rapport. C'est généralement une mauvaise idée car cela augmente le temps de réponse à la requête.

Parmi les valeurs possibles :
– on : active la résolution
– off : désactive la résolution
– double : active la résolution et enclenche une résolution inverse pour chaque requête

logresolve

L'outil logresolve est fourni en standard avec Apache. Il permet de résoudre les adresses IP fournies par l'entrée standard et les affiche par la sortie standard.

Le principe est de résoudre les adresse IP :
- à l'écriture dans le fichier de rapport
- plus tard, par exemple à la rotation

L'outil logresolve utilise la syntaxe suivante :

```
logresolve [-s fichier] [-c] < entrée > sortie
```

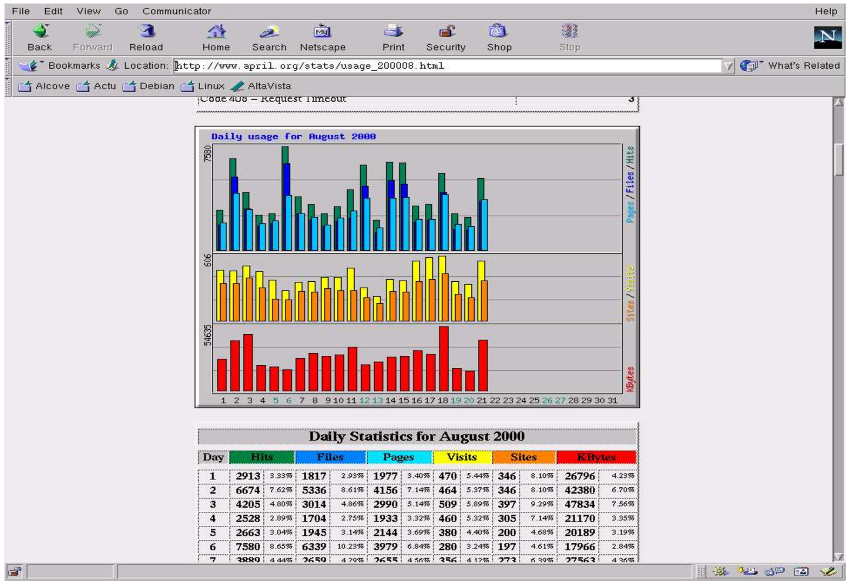
`-c` permet d'utiliser une résolution inverse pour plus de sécurité.

`-s fichier` spécifie le fichier où écrire les adresses résolues.

Analyse des fichiers de log

Une pléthore d'outils d'analyse de rapports sont disponibles sur l'Internet. Parmis les plus utilisés : webalizer, analog, http_analyze, ...

La plupart permettent de générer des rapports d'analyse en HTML, avec des graphiques montrant l'évolution du traffic sur le serveur.



Rappel des objectifs de cette section

– connaître le principe des fichiers de rapport
– comprendre l'intérêt des fichiers de rapport
– savoir tirer parti des fichiers de rapport correctement
– savoir utiliser les outils d'administration de fichiers de rapport

Optimisation des performances

Objectifs de cette section

– Connaître les mécanismes de gestion des connexions du serveur Apache
– Savoir paramétrer ces mécanismes
– Savoir choisir le réglage approprié

Gestion des connexions

L'architecture d'Apache est basée sur la création d'un jeu de serveurs chargés de répondre aux requêtes HTTP des clients. Ce jeu de serveurs est créé par un processus « père » qui se charge de créer plus de serveurs lorsque ceux existants sont surchargés ou d'en supprimer en période de faible activité.

L'administrateur est capable par le biais de quelques directives de paramétrer les bornes utilisées par le serveur Apache afin d'influer sur la politique de création et de suppression de serveurs.

KeepAlive

– Syntaxe : KeepAlive on|off
– Exemple : KeepAlive on

Cette directive permet d'utiliser ou non les extensions KeepAlive, définies dans le protocole HTTP/1.1 et qui éliminent quelques défauts du protocole HTTP. Le principe est de transmettre plusieurs requêtes HTTP au sein d'une seule connexion.

On appelle cette possibilité les connexions HTTP persistantes.

Timeout

– Syntaxe : Timeout *secondes*
– Exemple : Timeout 30

Cette directive permet de spécifier le temps maximum que le serveur Apache attendra entre l'initiation de la connexion et la fin de la réponse à la requête HTTP.

KeepAliveTimeout

– Syntaxe : KeepAliveTimeout secondes
– Exemple : KeepAliveTimeout 15

Cette directive spécifie le nombre de secondes maximum que le serveur Apache va attendre avant de fermer une connexion HTTP persistente. Elle permet d'éviter des attaques de type *Denial Of Service* (*DoS*).

MaxKeepAliveRequests

– Syntaxe : MaxKeepAliveRequests *nombre*
– Exemple : MaxKeepAliveRequests 100

Cette directive spécifie le nombre maximum de requêtes HTTP qui transiteront au sein d’une connexion. La valeur 0 signifie un nombre illimité.

Un grand nombre augmente la performance, mais afin d’éviter des attaques de type DoS, il est recommandé de ne pas y mettre de nombre trop important. Le nombre 100 est un bon compromis.

MinSpareServers

– Syntaxe : MinSpareServers *nombre*
– Exemple : MinSpareServers 5

Apache essaye dans la mesure du possible d'avoir des serveurs libres (*idle*) afin de répondre plus vite aux requêtes et d'éviter le coûteux démarrage d'un serveur.

La directive MinSpareServers permet de spécifier le nombre minimum de serveurs libres. Si le nombre de serveurs libres est subitement plus petit que ce nombre, Apache en démarre un par seconde jusqu'à attendre MinSpareServers.

La modification de cette valeur dépend du traffic du serveur.

MaxSpareServers

– Syntaxe : MaxSpareServers *nombre*
– Exemple : MaxSpareServers 5

À l'inverse de la directive précédente, MaxSpareServers définit le nombre **maximum** de serveurs libres.

En effet, un trop grand nombre de serveurs libres (inactifs) risque de consommer trop de ressources (mémoire) et de compromettre la performance du serveur.

StartServers

- Syntaxe : StartServers *nombre*
- Exemple : StartServers 5

Cette directive spécifie le nombre de serveurs qui sont exécutés au démarrage d'Apache. Ce nombre est ensuite ajusté en fonction de la charge de la machine.

Ce paramètre n'a besoin d'être modifié que dans le cas de grosses charges.

MaxClients

– Syntaxe : MaxClients *nombre*
– Exemple : MaxClients 256

Cette directive spécifie le nombre maximum de serveurs simultanés. En cas d'un nombre de requêtes simultanées supérieur à ce nombre, les requêtes restantes sont placées dans une queue en attendant une réponse.

La valeur maximale normale est de 256, pour augmenter ce plafond, il est nécessaire d'éditer les sources d'Apache.

MaxRequestsPerChild

- Syntaxe : MaxRequestsPerChild *nombre*
- Exemple : MaxRequestsPerChild 1000

Cette directive permet de spécifier le nombre maximum de requêtes auxquelles un processus Apache répondra avant de s’arrêter. La motivation d’un tel mécanisme est de garantir que le serveur ne remplira pas la totalité de la mémoire de la machine suite à une fuite mémoire.

Profilage

Une fois que le serveur Apache a été paramétré, il reste à déterminer l'impact réel de sa configuration sur la performance. L'outil `ab` (pour *Apache Bench*) permet de tester la rapidité du serveur en générant un nombre important de requêtes simultanées afin de le tester en cas de montée de charge.

Le but du jeu est d'envoyer un nombre maximum de requêtes à la seconde et de mesurer le temps de réponse minimum, moyen et maximum. Des études montrent qu'au delà d'un temps de réponse de deux secondes, l'utilisateur à tendance à se lasser d'attendre et renonce souvent à consulter le document.

ab

```
Usage : ab [options] http ://hostname :port/chemin
```

`ab` prend en argument des options suivies d'une URL, qu'il se chargera de consulter. Parmi les options possibles :

– -n *nombre* : nombre de requêtes à effectuer
– -c *nombre* : nombre maximum de requêtes concurrentes
– -k : utilise les connexions HTTP/1.1 persistentes
– -A *utilisateur* , *mot-de-passe* : effectue une authentification HTTP

Rappel des objectifs de cette section

– Connaître les mécanismes de gestion des connexions du serveur Apache
– Savoir paramétrer ces mécanismes
– Savoir choisir le réglage approprié

Utilisation de la CGI, contenu dynamique

Objectifs de cette section

- Connaître le principe et le fonctionnement de la CGI
- Connaître ses avantages et désavantages
- Savoir mettre en oeuvre la CGI

Utilisation de la CGI : Shellscript, Perl

Le contenu dynamique permet de produire un contenu plus fréquemment mis à jour et de s'abstraire des processus de mise à jour d'un simple fichier. Il permet aussi d'interagir plus finement avec le client.

La solution standard est l'utilisation de la CGI (Common Gateway Interface), qui consiste en l'exécution d'un programme externe au serveur web et en la communication des deux processus.

Le CGI s'exécute sur son propre plan d'adressage et renvoie ses résultats au serveur Apache via un protocole simple. Des paramètres peuvent êtres passés au programme.

Avantages de la CGI

- indépendance du serveur
  - pas de corruption de la mémoire du serveur
  - en cas de crash, seul le CGI s'arrête
  - le serveur n'a pas à savoir ce que fait le CGI
- indépendance du langage

Inconvénients de la CGI

- temps de démarrage du programme très lent
- interaction avec Apache limitée
- communication inter-processus nécessaire
- sécurité

Principe d'un script CGI

Un script CGI est simplement un programme qui communique avec le serveur apache par un protocole de communication élémentaire.

Plutôt que d'ouvrir un fichier et de le lire, le serveur exécute un processus et communique avec lui via l'entrée et la sortie standards. Le serveur Apache transmet des informations au CGI via des variables d'environnement et en retour, le script CGI construit la réponse (y compris les en-têtes HTTP, par exemple `Content-type`).

Exemple de script shell CGI

```
#!/bin/sh

echo "Content-type : text/html"

echo

echo "Bonjour, je suis un script CGI.<p>"

echo -n "La date est "

date
```

Mise en place des CGI

Deux méthodes pour mettre en place des CGI :

- placer les cgi dans un répertoire
- définir une extension de fichier propre au CGI

ScriptAlias

– Syntaxe : ScriptAlias *url répertoire*
– Exemple : ScriptAlias /cgi-bin /usr/local/apache/cgi-bin

Cette directive permet de définir un répertoire dans lequel on placera les scripts CGI. Tous les fichiers qui s'y trouvent seront considérés comme des scripts CGI et exécutés comme tels.

Ne pas oublier de les rendre exécutables au sens UNIX du terme, ni de positionner l'option **ExecCGI** dans le répertoire où ils se trouvent.

SetHandler

– Syntaxe : SetHandler *handler*
– Exemple : SetHandler server-status

Cette directive se place dans un conteneur <Directory>, <Location> ou dans un fichier `.htaccess`. Elle spécifie que tous les fichiers contenus dans cet espace seront traités avec le handler *handler* passé en argument.

AddHandler

– Syntaxe : AddHandler *handler extension*
– Exemple : AddHandler cgi-script cgi

Une des puissances d'Apache est l'utilisation de *handlers* . Il s'agit d'une suite d'instructions utilisées pour traiter une requête. Le handler normal est d'ouvrir un fichier et de retourner son contenu.

Une autre est d'ouvrir un fichier et de l'exécuter. C'est le handler *cgi-script* , celui utilisé pour exécuter des scripts CGI.

Comparaison des deux méthodes

L'utilisation de la deuxième méthode :
- est plus flexible (répartition des CGI)
- introduit des problèmes de sécurité

Les problèmes de sécurité peuvent être contournés en utilisant judicieusement les directives Options.

Rappel des objectifs de cette section

– Connaître le principe et le fonctionnement de la CGI
– Connaître ses avantages et désavantages
– Savoir mettre en oeuvre la CGI

Sécurité et aspects avancés

Éléments de sécurité

Objectifs de cette section

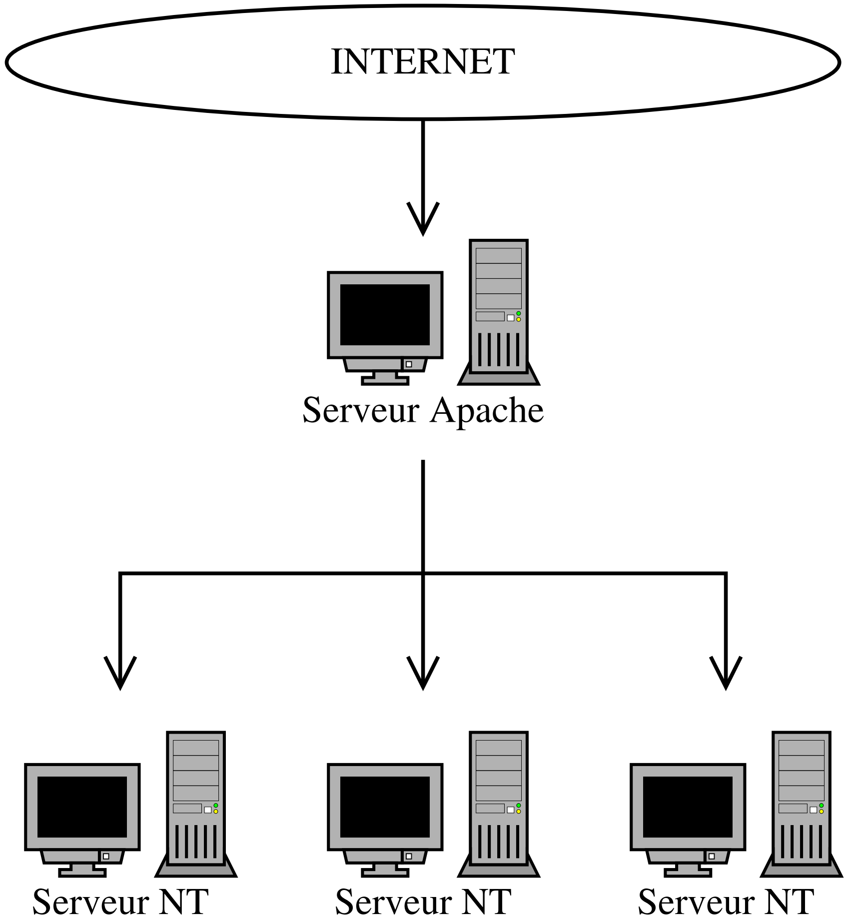
– Comprendre les problématiques de sécurité soulevées par Apache
– Savoir configurer Apache en tant que proxy/cache

Configuration d'Apache en tant que proxy/cache

En plus de répondre aux requêtes HTTP entrantes, Apache peut se comporter en tant que client HTTP et ainsi servir de proxy, remplissant les fonctionnalités de logiciels comme Squid.

Exemples d'utilisation de cette technique :
- frontal d'accès à une grappe de serveurs
- redondance de serveurs et/ou répartition de charge
- filtrage de contenu

Schéma typique d'un intranet protégé par Apache :



ProxyRequests

- Syntaxe : ProxyRequests on|off
- Exemple : ProxyRequests off

Cette directive active ou désactive l'utilisation d'Apache en tant que proxy.

ProxyPass

– Syntaxe : ProxyPass *chemin url*
– Exemple : ProxyPass /mirror/foo http ://foobar.org/

Cette directive permet d’inclure des serveurs distants dans l’espace de nommage du site local. Le serveur local transmet les requêtes reçues vers un serveur distant et apparaît comme un miroir du site distant.

Exemple

Si le serveur local s'appelle `http ://www.foo.org/` et que la ligne suivante est insérée dans le fichier de configuration :

```
ProxyPass /bar http ://www.bar.org/
```

Alors, une requête à l'URL `http ://www.foo.org/bar/test.html` est convertie en `http ://www.bar.org/test.html` de manière interne.

ProxyPassReverse

– Syntaxe : ProxyPassReverse *chemin url*
– Exemple : ProxyPassReverse /mirror/foo http ://foobar.org/

Cette directive agit de concert avec ProxyPass. ProxyPassReverse permet de plus de modifier certains en-têtes HTTP afin d'éviter de passer à travers le proxy en cas de réécriture d'URL (par exemple dans le cas de redirections lors d'une requête sur un répertoire en omettant le slash final).

ProxyDomain

– Syntaxe : ProxyDomain *domaine*
– Exemple : ProxyDomain .foobar.org

Cette directive est très utile dans le cas d'intranets, où il est possible de spécifier un nom de machine sans lui accoler le nom de domaine. Lorsqu'Apache rencontre ce cas de figure, il accole au nom de machine l'argument passé à la directive ProxyDomain.

ProxyRemote

- Syntaxe : ProxyRemote *filtre url*
- Exemple : ProxyRemote * http ://proxy :3128/

Cette directive permet de rediriger les requêtes correspondant au filtre (qui peut utiliser le méta-caractère étoile) vers le proxy spécifié en deuxième argument.

Le filtre peut aussi être un protocole (ftp, https, ...). Par exemple :

```
ProxyRemote ftp http ://ftpproxy :3128/
```

NoProxy

- Syntaxe : NoProxy *domaine* | *sous-réseau* | *adresse_ip* | *nom*
- Exemple : NoProxy .foobar.org 192.168.112.0/21

NoProxy permet de ne pas utiliser un proxy spécifié par ProxyRemote pour un ensemble de domaines, de sous-réseaux, d'adresses IP ou de noms d'hôtes donnés.

Répartition de charge

Le module mod_backhand (`http ://www.backhand.org/`) permet de répartir la charge entre différents serveurs web en fonction de critères arbitraires (charges de machines, occupation mémoire, ...).

Il fonctionne de manière similaire au module mod_proxy, c'est à dire en effectuant lui-même une requête HTTP au serveur de destination.

Rappel des objectifs de cette section

– Comprendre les problématiques de sécurité soulevées par Apache
– Savoir configurer Apache en tant que proxy/cache

L'authentification

Objectifs de cette section

- Connaître les problématiques de l'authentification HTTP
- Savoir mettre en place une authentification HTTP
- Pouvoir utiliser indépendamment différents annuaires d'utilisateurs

L'authentification

L'authentification permet de restreindre tout ou partie du site à une ou plusieurs personnes qui seront munies du mot de passe correspondant et devront au préalable s'authentifier.
L'authentification est définie dans le protocole HTTP et la quasi totalité des navigateurs web comprennent cette authentification.

Pour Apache, il existe deux procédures d'authentification :
– l'authentification individuelle, avec stockage des mots de passe dans un fichier ou une base de données
– l'authentification anonyme, similaire à l'accès FTP anonyme, où l'utilisateur peut taper son adresse email à la place de son mot de passe

Principe de l'authentification HTTP

Le scénario typique d'une authentification HTTP est le suivant :

– le serveur reçoit une requête sur un répertoire protégé par mot de passe
– il renvoie un message d'erreur HTTP (erreur 401) signifiant que le client ne s'est pas correctement authentifié, ainsi qu'un champ (WWW-Authenticate) contenant le nom de la collection de documents à laquelle le client a essayé d'accéder
– le client demande un mot de passe à l'utilisateur
– le client envoie au serveur la même requête avec un champ de la requête (Authorization) contenant le mot de passe

Principe de l'authentification HTTP

Afin de permettre la définition de plusieurs zones différentes protégées par mot de passe sur le même serveur, le protocole HTTP permet le nommage de chacun de ces zones par un nom, appelé *realm* .

Dans le cas où plusieurs zones partageraient le même *realm* , le client n'a aucun moyen de les distinguer et envoie les mêmes informations d'authentification.

AuthType

– Syntaxe : AuthType basic|digest
– Exemple : AuthType basic

Cette directive permet de choisir entre les deux méthodes de communication de mot de passe entre le client et le serveur :

– basic : le mot de passe transite encodé, mais pas encrypté
– digest : le mot de passe est encrypté, mais pas le message. De plus, tous les navigateurs ne connaissent pas cette méthode

AuthName

- Syntaxe : AuthName nom
- Exemple : AuthName "Secure area"

Cette directive permet de spécifier le *realm* envoyé au client pour identifier la collection protégée par mot de passe, ce qui permet au client de garder le mot de passe en mémoire et de ne pas le redemander à l'utilisateur à la prochaine consultation d'un document de ce même realm.

require

– Syntaxe : require user|group|valid-user *valeurs*
– Exemple : require user admin

Cette directive détermine les conditions de succès d'une authentification. Elle peut s'utiliser de trois manières différentes :

– `require user` *`liste`* : demande que l'utilisateur authentifié fasse partie de la liste d'utilisateurs passée en argument
– `require group` *`liste`* : demande que l'utilisateur authentifié fasse partie d'un des groupes d'utilisateurs de la liste passée en argument
– `require valid-user` : demande que l'utilisateur soit tout simplement correctement authentifié

Stockage des utilisateurs

Il existe plusieurs moyens de stocker les utilisateurs du contenu du serveur web :

– un fichier texte
– un fichier db
– un fichier dbm
– une base de données SQL
– un annuaire LDAP
– un module PAM

AuthUserFile

– Syntaxe : AuthUserFile *fichier*
– Exemple : AuthUserFile /usr/local/apache/users.txt

Cette directive précise le nom du fichier contenant la base des utilisateurs. Le format de la base est proche de celui du fichier `/etc/passwd`. Par exemple :

```
admin :yjVdcWdyNbZgI
```

Par sécurité, éviter de placer ce fichier dans un répertoire accessible via HTTP.

AuthGroupFile

- Syntaxe : AuthGroupFile *fichier*
- Exemple : AuthGroupFile /usr/local/apache/groups.txt

Cette directive précise le nom du fichier contenant la base des groupes. Le format de la base est proche de celui du fichier `/etc/groups`. Par exemple :

```
admin : joe foo bar
```

Chaque utilisateur fait partie d'un ou plusieurs groupes à la manière d'UNIX.

htpasswd

htpasswd est un outil fourni en standard avec Apache qui permet d'ajouter des utilisateurs à la base. Il prend en argument un fichier de mots de passe et un nom d'utilisateur et demande le mot de passe sur l'entrée standard. Ainsi, pour ajouter l'utilisateur bob à la base des utilisateurs, on utilisera la commande :

```
$ htpasswd /usr/local/apache/users.txt bob
```

La commande htpasswd reconnaît (entre autres) les arguments suivants :

– `-f` *`fichier`* : crée un nouveau fichier ou vide le fichier s'il existe
– `-b` *`password`* : lit le mot de passe sur la ligne de commande plutôt que sur l'entrée standard

Stockage des utilisateurs dans une base db/dbm

Le gros problème de la solution précédente est le temps d'accès à l'information : il est nécessaire de parcourir la base séquentiellement.

L'utilisation d'une base de données dbm permet de supprimer ce problème en mettant en place une table de hashage contenant des données sous la forme d'un couple clef/mot de passe.

DB et DBM sont deux formats de fichiers traditionnels dans le monde UNIX et aux caractéristiques équivalentes. Les directives d'utilisation de ces deux formats sont pratiquement identiques, nous ne traiterons que de l'authentification DBM.

AuthDBMAuthoritative

– Syntaxe : AuthDBMAuthoritative on|off
– Exemple : AuthDBMAuthoritative on

Cette directive indique explicitement que l'authentification se basera sur une base de données DBM et non pas sur un fichier texte.

AuthDBMUserFile

– Syntaxe : AuthDBMUserFile *fichier*
– Exemple : AuthDBMUserFile /usr/local/apache/users.dbm

Cette directive précise le nom du fichier contenant la base des utilisateurs. Il est au format DBM, ce qui signifie qu'on fait correspondre à une clef (le nom de l'utilisateur) une donnée (mot de passe crypté).

De la même manière qu'avec une base texte, éviter de le placer dans un répertoire accessible via le web.

AuthDBMGroupFile

- Syntaxe : AuthDBMGroupFile *fichier*
- Syntaxe : AuthDBMGroupFile /usr/local/apache/groups.dbm

Cette directive précise le nom du fichier contenant la base des groupes. Chaque enregistrement est composé d'une clef (le nom de l'utilisateur, tel que dans le fichier positionné par AuthDBMUserFile) et d'une donnée (une liste de groupes, séparés par des virgules).

Rappel des objectifs de cette section

- Connaître les problématiques de l'authentification HTTP
- Savoir mettre en place une authentification HTTP
- Pouvoir utiliser indépendamment différents annuaires d'utilisateurs

Utilisation de PAM

Objectifs de cette section

– Comprendre le fonctionnement de PAM
– Connaître l'architecture utilisée par PAM
– Comprendre le principe d'un annuaire
– Savoir utiliser PAM dans Apache

Utilisation de PAM

PAM, pour *Pluggable Authentification Module* , est un mécanisme flexible pour authentifier les utilisateurs. Le principe est d'ajouter une couche d'abstraction entre l'application requêtant une authentification et la source supportant la base des utilisateurs.

Avantages de PAM

- abstraction totale du média supportant la base des utilisateurs
- portage d'un mécanisme à un autre transparent pour l'utilisateur
- de nombreuses applications supportent PAM
- pas besoin de modifier les applications à chaque nouvelle source

Exemples de sources reconnues par PAM

– /etc/passwd
– cryptocard
– radius
– SQL
– LDAP

Exemples d’applications utilisant PAM

- Python
- Perl
- Squid
- Qpopper
- IMAP
- ProFTPd
- les services UNIX classiques
- ... et bien sûr Apache

Fonctionnement de LDAP

Le répertoire `/etc/pam.d` contient un fichier par protocole utilisant PAM. Ces fichiers décrivent :

– le module PAM utilisé (source)
– les conditions à remplir lors d'une authentification

Exemple de fichier de configuration de PAM

Cet exemple permet à l'authentification HTTP de se baser sur les fichiers de mots de passe UNIX `/etc/passwd`.

```
#%PAM-1.0

# Configuration file for the 'httpd' service

auth required pam_pwdb.so md5

account required pam_pwdb.so md5
```

L’exemple de LDAP

Nous allons prendre pour exemple la source LDAP.

LDAP est un annuaire, permettant :
– de stocker des informations de manière arborescente
– de stocker des informations dans des classes
– de distribuer l’information sur le réseau
– de donner des droits limités sur l’information

LDAP

Les données d'un annuaire LDAP sont organisées en feuilles et en noeud, chacun étant un **enregistrement** , possédant :

– une classe

– des attributs

Chaque entrée est repérée par un chemin (`dn`) qui permet de la retrouver dans l'arborescence. Sous chaque noeud, on trouve une liste de feuilles étant chacune repérée par un attribut unique (par exemple le nom). Le noeud « racine » étant appelé le **dc** (par exemple, `dc=alcove-int`).

OpenLDAP

L'implémentation LDAP la plus utilisée sous les systèmes libres est OpenLDAP. OpenLDAP est un projet de développement d'un annuaire LDAP de qualité et visant à supporter les derniers standards.

Depuis août 2000, OpenLDAP supporte LDAPv3.

PAM et OpenLDAP

L'authentification PAM/OpenLDAP fonctionne de la manière suivante :

– lecture du fichier de configuration de PAM/LDAP (`/etc/ldap.conf` ou `/etc/pam_ldap.conf`)
– recherche dans l'annuaire d'un enregistrement possédant un attribut (`uid`) égal au nom d'utilisateur entré par le visiteur
– vérification de l'égalité du mot de passe (crypté) avec celui transmis par le visiteur

Possibilité de restreindre la recherche à une partie de l'annuaire et à une classe d'enregistrements.

Apache et PAM

Apache peut utiliser PAM pour effectuer une authentification, à condition de lui faire utiliser le module *mod_auth_pam* . Une fois ce module installé, il reste :

- à installer le support LDAP pour PAM
- à configurer le module PAM/LDAP (spécifier le `dc`)
- à placer un fichier `http` dans le répertoire `/etc/pam.d` spécifiant que l'authentification sera à base de LDAP
- à spécifier à Apache d'utiliser PAM

AuthPAM_Enabled

– Syntaxe : AuthPAM_Enabled on|off
– Exemple : AuthPAM_Enabled on

Cette directive permet d'activer l'utilisation de PAM pour une authentification donnée.

Rappel des objectifs de cette section

– Comprendre le fonctionnement de PAM
– Connaître l'architecture utilisée par PAM
– Comprendre le principe d'un annuaire
– Savoir utiliser PAM dans Apache

Contrôle d'accès

Objectifs de cette section

- Comprendre les problématiques du contrôle d’accès
- Savoir définir plusieurs politiques d’accès et de rejet au site

Contrôle d'accès

Le contrôle d'accès est utile pour cacher certaines parties du site aux visiteurs et les réserver au personnel autorisé.

Le contrôle d'accès sur la provenance est basé sur l'analyse de l'adresse IP du client et sur l'autorisation ou non de l'accès. Ainsi, il est par exemple possible d'effectuer un contrôle sur :

- l'adresse IP d'une machine
- le sous-réseau d'une machine
- le nom de domaine d'une machine

En cas d'impossibilité d'accéder à la page, un message d'erreur explicite est affiché.

allow

- Syntaxe : allow from *machine* | *réseau machine* | *réseau* ...
- Exemple : allow from 127.0.0.1 localhost .foobar.org

La directive allow permet d'autoriser l'accès à un répertoire particulier. Le contrôle se fait sur l'adresse IP de la provenance.

Les arguments de cette directive sont de la forme suivante :

- `all` : toutes les machines sont autorisées
- un nom de domaine (partiel) : toutes les machines de ce domaine sont autorisées. Exemple : .foobar.org
- une adresse ip complète : cette machine est autorisée
- une adresse ip partielle (avec 1, 2 ou 3 octets) : les adresses ip de ces classes sont autorisées. Exemple : 10.1
- un réseau et un masque de réseau : seul les adresses correspondant à cette paire sont autorisées. Exemple :

10.1.0.0/255.255.0.0
– un masque CIDR : un réseau suivi du nombre de bits significatifs. Exemple : 10.1.0.0/16

Ainsi, pour autoriser les machines d'un réseau local (domaine .foobar.org et de réseau 10.1.0.0), on pourrait utiliser les techniques suivantes :
– allow from 10.1
– allow from .foobar.org
– allow from 10.1.0.0/255.255.0.0
– allow from 10.1.0.0/16

deny

– Syntaxe : deny from *machine* | *réseau machine* | *réseau* ...
– Exemple : deny from 127.0.0.1 localhost .foobar.org

La directive deny permet d'interdire l'accès à un répertoire particulier. Le contrôle se fait sur l'adresse IP de la provenance.

Les arguments de cette directive sont de la forme suivante :
– `all` : toutes les machines sont bloquées
– un nom de domaine (partiel) : toutes les machines de ce domaine sont bloquées. Exemple : .foobar.org
– une adresse ip complète : cette machine est bloquée
– une adresse ip partielle (avec 1, 2 ou 3 octets) : les adresses ip de ces classes sont bloquées. Exemple : 10.1
– un réseau et un masque de réseau : seul les adresses correspondant à cette paire sont bloquées. Exemple :

10.1.0.0/255.255.0.0

– un masque CIDR : un réseau suivi du nombre de bits significatifs. Exemple : 10.1.0.0/16

Ainsi, pour interdire l'accès à une page à un concurrent on pourrait utiliser :

– deny from .concurrent.com

order

– Syntaxe : order *ordre*
– Syntaxe : order deny,allow

Cette directive permet de spécifier dans quel ordre appliquer les directives allow et deny. Elle prend en argument :

– allow,deny : on commence par lire les règles allow, puis deny. Si aucune directive ne correspond, alors on rejette l'accès.
– deny,allow : on commence par lire les règles deny, puis allow. Si aucune directive ne correspond, alors on autorise l'accès.
– mutual-failure : seuls les machines présentes dans la liste des machines autorisées et pour lesquelles aucune règle d'interdiction ne s'applique sont autorisée. Si ce n'est pas le cas, elles sont rejetées.

deny from env=

- Syntaxe : deny from env= *variable*
- Exemple : deny env=bad_browser

Cette directive permet d'interdire l'accès à un répertoire en cas de présence d'une variable d'environnement. Apache possède plusieurs variables d'environnement et il est possible d'en définir plusieurs en fonction de certains critères (par exemple avec BrowserMatch).

allow from env=

– Syntaxe : allow from env= *variable*
– Exemple : allow env=bad_browser

Cette directive permet d’autoriser l’accès à un répertoire en cas de présence d’une variable d’environnement. Apache possède plusieurs variables d’environnement et il est possible d’un définir plusieurs en fonction de certains critères (par exemple avec BrowserMatch).

BrowserMatch

– Syntaxe *regexp attribut attribut* ...
– Exemple MSIE !javascript

Cette directive permet de donner une valeur à une variable d'environnement d'Apache en fonction du navigateur présent. Une comparaison entre l'expression régulière *regexp* et le nom du navigateur sera effectuée et en cas de réussite, on effectuera des modifications sur l'environnement du serveur. Ces modifications dépendent de la syntaxe de chaque attribut :

– *nom* : la variable d'environnement *nom* sera positionnée à 1
– *nom* = *valeur* : la variable d'environnement *nom* sera positionnée à la valeur *valeur*
– ! *nom* : la variable d'environnement *nom* sera supprimée si elle était déjà définie

Rappel des objectifs de cette section

– Comprendre les problématiques du contrôle d’accès
– Savoir définir plusieurs politiques d’accès et de rejet au site

Les connexions sécurisées, SSL

Objectifs de cette section

- Comprendre le principe de l'encryption
- Comprendre les mécanismes de SSL
- Comprendre son intéraction avec le serveur Apache

Les connexions sécurisées, SSL

Un des plus gros problèmes d'HTTP est sa sécurité :

– aucune encryption de contenu n'est prévue

– l'encryption du mot de passe est peu supportée

N'importe qui peut « écouter » les données qui transitent par le réseau (par exemple un proxy HTTP).

L'utilisation de SSL permet d'ajouter une sécurité supplémentaire sans modifier le protocole HTTP.

SSL

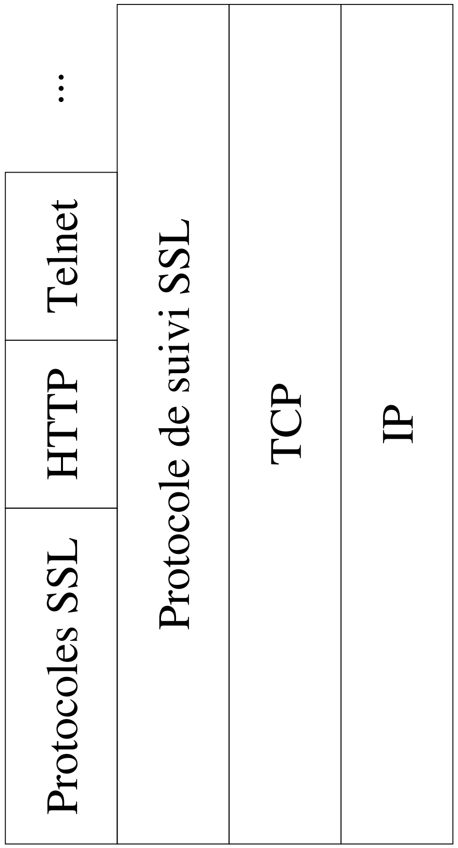
SSL, pour *Secure Socket Layer* , est une couche d'encryption qui peut se placer sous tous les protocoles possibles sans modifier leur comportement, à condition que les deux parties permettent l'utilisation de SSL.

Deux approches de la cryptographie :
– l'encryption conventionnelle
– l'encryption à clef publique

Pour garantir une sécurité accrue, SSL utilise non seulement la deuxième solution mais aussi un mécanisme de **signatures** et de **certificats** .

SSL permet de plus un grand choix parmi les algorithmes d'encryption et de signature possibles.

Pile des protocoles SSL



Protocoles SSL
HTTP
Telnet
...
Protocole de suivi SSL
TCP
IP

Interface avec Apache

Apache peut utiliser le protocole SSL de deux manières différentes :
- utiliser Apache-SSL
- utiliser mod_ssl, un module Apache

Les deux solutions se basent sur OpenSSL, qui est une implémentation libre de SSL.

OpenSSL

OpenSSL est un projet d'implémentation libre des protocoles SSLv2, SSLv3 et TLSv1. Il est basé sur l'implémentation SSLeay.

OpenSSL fournit une bibliothèque de fonctions manipulant des connexions SSL ainsi qu'une suite d'outils permettant de manipuler les clefs et certificats utilisés.

Pour des raisons légales, ce logiciel a été développé principalement en Suisse et en Allemagne, où l'encryption est totalement légale.

mod_ssl

mod_ssl est un module Apache et en tant que tel permet :
– d'ajouter des fonctionnalités et des directives au serveur
– de configurer finement l'utilisation de SSL pour une partie du serveur (répertoire, virtual host, ...)

Quelques directives ajoutées :
– SSLEngine
– SSLCertificateKeyPath
– SSLCertificatePath

SSLEngine

– Syntaxe : SSLEngine on|off
– Syntaxe : SSLEngine off

Cette directive permet d'activer ou non l'utilisation du protocole SSL. Par défaut, cette directive est positionnée à *off* .

SSLCertificatePath

- Syntaxe : SSLCertificatePath *fichier*
- Exemple : SSLCertificatePath /usr/local/apache/ssl

Cette directive spécifie le chemin du fichier contenant le certificat du serveur. Ce même fichier peut contenir la clef privée du serveur.

Si le certificat est encrypté, le serveur demandera un mot de passe au démarrage.

SSLCertificateKeyPath

– Syntaxe : SSLCertificateKeyPath *fichier*
– Exemple : SSLCertificateKeyPath /usr/local/apache/ssl

Cette directive spécifie le chemin du fichier contenant la clef privée du serveur.

Si la clef est encryptée, le serveur demandera un mot de passe au démarrage.

Mise en place

La procédure est la suivante :

– installation d'OpenSSL
– installation d'Apache
– installation de mod_ssl
– génération de certificats et de clefs privées
– contact d'un serveur de certificats
– installation du certificat généré

Rappel des objectifs de cette section

– Comprendre le principe de l'encryption
– Comprendre les mécanismes de SSL
– Comprendre son intéraction avec le serveur Apache

Utilisation avancée

Objectifs de cette section

- Comprendre la réécriture d'URL
- Savoir mettre en place redirection et alias

Réécriture dynamique d'URL

La réécriture d'URL permet de définir des alias d'une URL à l'autre. L'utilisation basique est de rediriger l'utilisateur d'une page obsolète à son remplacement ou de créer des chemins virtuels dans l'espace de nommage du serveur.

On définit deux types de réécriture d'URL :
- la redirection
- l'alias

Alias

- Syntaxe : Alias *url chemin*
- Exemple : /icons /usr/local/apache/icons

Cette directive permet de réécrire une URL en faisant correspondre un répertoire à une URL du serveur. On peut voir cette réécriture comme un lien symbolique.

L'alias aide à l'administration du site en permettant de placer des données en dehors du répertoire DocumentRoot.

Redirect

– Syntaxe : Redirect [*statut*] *origine destination*
– Exemple : Redirect permanent /service http ://www.foobar.org/service

Cette directive permet de rediriger le navigateur sur une autre page en se servant du protocole HTTP, qui permet la redirection.

Le premier argument de cette directive est un chemin pour lequel toutes les requêtes seront redirigées vers l'url en deuxième argument.

mod_rewrite

Ces stratégies ne sont pas les plus puissantes car elles ne sont pas dynamiques. Le module mod_rewrite permet de réécrire les URL de manière beaucoup plus fine.

Le module mod_rewrite permet :
– de rediriger le client vers des URL (redirect)
– de proxyfier des URL

Le module mod_rewrite est puissant et flexible (le nombre d'utilisations possibles est infini), mais très complexe.

Rappel des objectifs de cette section

– Comprendre la réécriture d’URL
– Savoir mettre en place redirection et alias

Les Server Side Include (SSI) : une alternative au CGI

Objectifs de cette section

- Comprendre le principe des SSI
- Comprendre les avantages et inconvénients des SSI
- Savoir comment permettre les SSI sans compromettre la sécurité du site

Les SSI : une alternative au CGI

Les server-side include (ou SSI) sont une manière de faire exécuter des traitements simples au serveur web. Il s'agit d'incorporer des instructions dans une page HTML et de la faire lire par un handler appelé *server-parsed* .

Ce handler permet analyse la page, exécute les inistructions qui lui sont destinées et retourne le résultat. Dans le cas d'une page HTML normale, l'utilisation de server-parsed n'a aucune incidence.

Les SSI : une alternative au CGI

Les SSI permettent principalement :

- d'inclure un fichier dans une page HTML
- d'exécuter un programme et d'insérer son résultat dans un fichier HTML
- d'afficher des variables d'environnement dans un fichier HTML
- et plein d'autres choses

Syntaxe des SSI

Les SSI sont des instructions placées dans des commentaires, de telle sorte que même sans l'intervention du handler server-parsed, l'apparence de la page HTML n'est pas modifiée.

Leur syntaxe est de la forme :

```
<!--#element attribut="valeur" ... -->
```

<!--#config-->

config permet de modifier le comportement des autres SSI. Elle accepte les attributs suivants :

– errmsg : le message d'erreur affiché en cas d'erreur dans une SSI
– sizefmt : l'unité utilisée pour afficher la taille d'un fichier (`bytes` pour des octers, `abbrev` pour des kilo-octets et des méga-octets).
– timefmt : le format utilisé pour afficher une date. Voir la page de manuel de strftime(3) pour un exemple.

Exemple : `<!--#config errmsg="oups..."-->`

< ! - - #echo - - >

La commande echo permet d'afficher une variable d'environnement dans le fichier HTML. Elle accepte l'attribut `var` et affiche la variable indiquée en valeur.

Ainsi, pour afficher la variable `REMOTE_ADDR` :

```
<!--#echo var="REMOTE_ADDR"-->
```

< ! - - #exec - - >

La commande exec permet d'exécuter une commande ou un script CGI et de l'insérer à la place de la balise. Elle accepte les attributs suivants :

– cgi : exécute le CGI dont l'URL est indiquée en valeur. S'il ne s'agit pas d'un chemin absolu, le CGI est relatif au répertoire du document

– cmd : exécute la chaîne de caractères en valeur en utilisant `/bin/sh`

Attention aux problèmes de sécurité d'une telle commande.

< ! - - #fsize - - >

La commande fsize affiche la taille du fichier spécifié par un attribut.
Les attributs suivants sont acceptés :

– file : la valeur représente un fichier relatif au répertoire du fichier
– virtual : la valeur représente une URL, relative au document

< ! - - #flastmod - - >

La commande flastmod affiche la date de dernière modification du fichier spécifié par un attribut. Les attributs sont de la même forme que pour la commande fsize.

<!-- #include -->

La commande include permet d'insérer le contenu d'un fichier dans le document. De la même manière que pour fsize et flastmod, elle accepte les attributs `file` et `virtual`, ce qui permet par exemple d'insérer le contenu d'un script CGI dans ce fichier.

l'option IncludeNOEXEC permet d'interdire l'exécution d'un programme mais de permettre l'inclusion d'un fichier.

<!--#printenv-->

La commande printenv affiche les variables d'environnement courantes (à la manière de la commande shell `printenv`). Elle n'accepte pas d'attributs.

<!--#set-->

La commande set permet de donner une valeur à une variable d'environnement. Elle accepte les attributs suivants :

– var : comme pour la commande `echo`, la valeur de l'attribut `var` spécifie le nom de la variable sur laquelle on opère

– value : la valeur de cet attribut est la valeur à donner à la variable

Par exemple : `<!--#set var="categorie" value="help"-->`

Interpolation

Comme dans tout langage de programmation, il est possible d’effectuer une interpolation. La syntaxe d’une variable est la même que dans le langage shell :

– $variable

– ${variable}

Il est possible de ne pas effectuer d’interpolation sur une variable en préfixant le dollar par un anti-slash.

Contrôle de flux

Un des gros avantages des SSI est de permettre le contrôle de flux facilement. Le contrôle de flux permet d'afficher certaines parties du document selon la validité ou non d'un test.

Les commandes if, elif, else et endif permettent le contrôle de flux. La syntaxe générale est :

```
<!--#if expr=" expression "-->

<!--#elif expr=" expression "-->

<!--#else-->

<!--#endif-->
```

Contrôle de flux

Les expressions utilisées dans un test sont de la forme :

– *chaîne* : vrai si la chaîne est non nulle
– *chaîne1 = chaîne2* : vrai si les deux chaînes sont égales
– *chaîne1 != chaîne2* : vrai si les deux chaînes sont différentes
– et ainsi de suite...

Mise en place de SSI

La problématique est d'activer les SSI, mais pas sur les pages HTML statiques (pour des raisons de performances).

Solutions :

– définir une nouvelle extension et lui appliquer le handler server-parsed
– utiliser la directive XBitHack

Définir un nouveau type

On utilise normalement l'extension `.shtml` et AddHandler pour lui spécifier le traitement à effectuer.

Mais `.shtml` n'est pas de type `text/html` et ne sera donc pas formaté par le navigateur. L'utilisation de AddType permettra de résoudre le problème.

AddType

- Syntaxe : AddType *type mime extension*
- Exemple : AddType text/html .shtml

Cette directive permet de spécifier le type MIME associé à une extension de fichier. Dans notre exemple précédent, nous pouvons utiliser :

```
AddType text/html .shtml

AddHandler server-parsed .shtml
```

Rappel des objectifs de cette section

- Comprendre le principe des SSI
- Comprendre les avantages et inconvénients des SSI
- Savoir comment permettre les SSI sans compromettre la sécurité du site

Interface de programmation

Objectifs de cette section

– Comprendre l'aspect visible de l'interface de programmation d'Apache

Interface de programmation

L'interface de programmation d'Apache est prévue pour faciliter l'inclusion de nouvelles fonctionnalités par le biais des *modules* .

Dans la majorité des cas, il s'agit d'ajouter des *handlers* qui se déclencheront sous certaines conditions. Ces handlers peuvent accéder à la totalité de la requête et de l'environnement dans lequel elle s'exécute.

Il est aussi possible de définir de nouvelles directives et d'exécuter des instructions au démarrage en fonction de celles-ci.

Rappel des objectifs de cette section

– Comprendre l'aspect visible de l'interface de programmation d'Apache

Conclusion

Objectifs de cette section

- Connaître les axes de développement du projet Apache
- Faire le point sur son futur

Projets relatifs à Apache

- langages
  - mod_perl
  - Java-Apache
  - php
- interfaces graphiques
  - comanche
  - tkApache
- WebDAV
- XML
- kHTTPd

Le futur d'Apache

Apache est aujourd'hui développé par une communauté active et enthousiaste. De nombreux industriels supportent le développement du serveur par le biais de la fondation Apache.

La préocuppation actuelle est la sortie de la versions 2.0 d'Apache, qui apportera de nombreuses améliorations :
– meilleures performance
– support des threads
– meilleure intégration sous les différentes plates-formes supportées

Références

- http ://www.apache.org/
- http ://www.gnu.org/
- http ://www.w3c.org/
- http ://www.alcove.fr/

Rappel des objectifs de cette section

- Connaître les axes de développement du projet Apache
- Faire le point sur son futur