Programmation Internet

November 2, 2006

Ce qui suit est un patchwork de copié-collés, principalement issus de la documentation d'Apache, du cours de Marc sur Apache (intro), et des tps de Jean-François.

Introduction

Apache est un serveur Web. Fondamentalement il délivre des fichiers par le protocole http à des clients. Les clients sont nombreux :

- des navigateurs graphiques
- des navigateurs texte
- toute sorte de programmes (par exemple des robots).

Les visites les plus nombreuses sont réalisées par des programmes et non par des navigateurs graphiques. Il est donc important que les pages soient lisibles et interprétables par ces programmes. C'est par exemple une des clefs d'une bonne indexation sur Internet.

Le serveur web se charge par exemple de :

- de la partie de contrôle, d'organisation et de gestion commune à tout type de serveurs (d'applications, de mails, etc . . .) :
  - de la liaison qui permet de définir par quels ports les échanges se font.
  - de la journalisation permettant de garder une trace des requêtes
  - de la gestion des performances (nbre de clients acceptés, temps pour garder une connexion ouverte,...)
- mais aussi de la gestion spécifique à un serveur web :
  - de la sécurité (authentification, cryptographie etc...)
  - du lancement de programmes externes au serveur pour constuire certaines réponses aux requêtes
  - de la négociation qui permet de retourner un contenu adapté à l'émetteur de la requête, selon sa langue, son type etc...
  - de la correspondance entre urls et système de fichiers
  - de la réécriture d'urls
  - de la virtualisation qui permet de faire tourner plusieurs serveurs en un seul.

Certaines de ces opérations sont propres à Apache et la liste n'est pas exhaustive. D'autres fonctionnalités peuvent dépasser cette première liste pour se rapprocher de la définition d'un Serveur d'applications comme Tomcat.

Nous allons voir plus en détail chacun de ces points.

Fichiers de configuration d'Apache

Syntaxe de base

Apache se configure en positionnant des directives en mode texte. On trouve une directive par ligne, si une directive doit être écrite sur plusieurs lignes on place en antislash en dernier caractère de la ligne pour indiquer que la directive se poursuit sur la ligne suivante.

Les lignes qui commencent par un # sont des commentaires (pas de commentaire en fin de ligne!); les lignes blanches et les espaces en début de ligne sont ignorés (ce qui permet une indentation des fichiers de configuration).

Apache n'est pas sensible à la casse pour le nom des directives, mais il l'est pour les paramètres de ces directives.

Enfin, les directives peuvent être placées à l'intérieur de *sections*. Ces sections spécifient un sous-arbre de la zone gérée par Apache et permettent d'écrire des directives spécifiques pour cette zone. Suivant les sections (`Directory`, `Location`, `VirtualHost`, ...), la zone peut correspondre à un ou des sous-arbres dans le système de fichiers de la machine physique sur laquelle tourne le serveur web, ou bien un ensemble d'URLs pour lesquelles le serveur reçoit une requête, ou bien un serveur virtuel... Il existe également des sections conditionnelles (`IfDefine`, `IfModule`, ...) qui permettent également de paramétrer finement le comportement du serveur web.

Fichiers de configuration

Généralement, le fichier principal est `httpd.conf`. Néanmoins, (et dans la plupart des distributions), son contenu est éclaté en différents fichiers de configuration. C'est la directive `Include` qui permet d'ajouter d'autres fichiers de configuration au fichier principal. Ces fichiers sont lus au démarrage du serveur, toute modification doit donc être suivie du redémarrage du serveur pour être prise en compte.

En plus de ces fichiers, le serveur lit un fichier (`mime.types`) qui contient les types des documents mime.

Remarque : les types MIME sont un moyen de décrire le contenu d'un fichier par un type majeur et un type mineur : text/html, text/css, application/ogg, image/bmp, ... Le fichier `mime.types` fait une correspondance entre un type MIME et une extension de fichier.

Les modules

Apache est modulaire : cela signifie qu'il tourne avec un certain nombre de fonctionnalités de bases, et que les extensions sont dans des modules externes qu'il faut charger si on veut les utiliser. Néanmoins, par défaut, un certain nombre de modules sont disponibles. Selon qu'Apache a été compilé ou non avec le support dynamique des modules ou pas, les modules peuvent être simplement chargés avec la directive `LoadModule` (c'est ce que l'on appelle les objets dynamiques partagés ou Dynamic Shared Objects DSO) ou bien nécessitent une recompilation complète d'Apache avec les nouveaux modules.

Il existe des sections conditionnelles `IfModule` qui permettent de paramétrer le comportement d'Apache en fonction du chargement d'un module.

Paramétrage fin

On définit dans la configuration quelles sont les zones du système de fichiers de la machine qui sont sous la "responsabilité" du serveur. Basiquement, il s'agit du sous-arbre qui se trouve à la racine qui est indiquée dans la directive `DocumentRoot`. Mais les choses sont un peu plus compliquées (cf section sur la correspondance entre URLs et système de fichiers, ou sur les serveurs virtuels).

Les directives placées à la racine du fichier de configuration s'appliquent pour le serveur dans son ensemble. Lorsqu'on veut paramétrer le comportement pour une sous-partie du serveur, on peut placer les directives à l'intérieur d'une des sections `Directory`, `DirectoryMatch`, `Location`, `LocationMatch`, `Files` et `FilesMatch`.

Enfin, le paramétrage peut être délégué à des fichiers `.htaccess` qui sont placés directement dans les répertoires concernés. Cette possibilité étant une ouverture (et donc une prise de risques) non négligeable, certaines directives n'y sont pas autorisés. Pour les autres, il faut absolument que leur présence dans les `.htaccess` soit autorisée par une directive `AllowOverride` dans les fichiers de configuration principaux pour qu'elles puissent être prises en compte.

Ces sections peuvent d'ailleurs s'imbriquer les unes dans les autres pour raffiner encore le paramétrage.

Néanmoins, certaines directives qui contrôlent la création du processus Apache ne peuvent s'appliquer qu'au serveur principal.

Description d'une directive

Il est important de savoir rechercher les informations dans les différents fichiers de configuration d'Apache et également de savoir parcourir la documentation. L'architecture des fichiers de configuration d'Apache varient d'une distribution à l'autre, mais les concepts restent les mêmes.

Une directive est généralement présentée dans la documentation en indiquant :

- sa description
- sa syntaxe : on entend par là la liste de ses paramètres et leur type : URL, type MIME, expression régulière, chemin d'accès à un fichier, etc etc...
- sa valeur par défaut (le cas échéant)
- son contexte : la section où on peut la trouver! cela peut être dans la section de configuration principale (et uniquement là), dans une section `Directory` (valable aussi pour une section `Location` ou `Files`), dans une section de serveur virtuel, ou dans un fichier `.htaccess`. On reparle de ces deux derniers éléments plus loin.
- son module : la directive fait elle partie d'un module de base, ou bien du noyau interne d'Apache, ou bien d'un extension , ...
- et quelques autres éléments (sa compatibilité avec les différentes versions d'Apache, quelles conditions supplémentaires sont requises pour la prise en compte de la directive dans un fichier .htaccess, ...)

Liaison

(Source : `http://httpd.apache.org/docs/2.0/bind.html`)

Au moment de son démarrage, Apache se lie à un port et à une adresse sur la machine et se met en attente de requêtes entrantes. Par défaut, toutes les adresses de la machine se retrouvent à l'écoute. Dans tous les cas, Apache accepte d'écouter sur un ou plusieurs ports spécifiques, ou sur une seule ou plusieurs adresses, ou encore une combinaison des deux. Il est fréquent d'utiliser ces possibilités avec les fonctionnalités de Serveurs Virtuels, qui permettent de faire répondre Apache de manière différente en fonction de l'adresse IP, du nom ou du port.

Le serveur utilise la directive `Listen` pour n'accepter que des requêtes provenant de ports spécifiques ou d'une combinaison adresse IP + port passés en argument. Dans le cas où seul un port est spécifié avec la directive `Listen`, le serveur se met à l'écoute sur le port spécifié, sur toutes les interfaces et adresses de la machine. Si une adresse IP est précisée en plus du port, le serveur n'écoute que sur l'adresse et le port spécifiés. Il est possible de configurer plusieurs directives `Listen`, afin qu'Apache écoute sur plusieurs adresses et ports. Dans ce cas, le serveur répondra aux requêtes faites sur tous les adresses et ports énumérés.

Par exemple, pour que le serveur accepte les connexions à la fois sur les ports 80 et 8000, spécifiez :

```
Listen 80
Listen 8000
```

Pour qu'Apache accepte les connexions sur deux combinaisons adresses + ports, spécifiez :

```
Listen 192.170.2.1:80
Listen 192.170.2.5:8000
```

La négociation de contenus

L'idée est que pour une même requête http, plusieurs documents peuvent être retournés. Par exemple, pour une requête `http://www.mon-asso.fr`, le serveur web va me retourner un document qui est à la racine du site web. Mais quel document ? L'index en français ou en anglais ? un .html ou un .txt ?

Le choix va pouvoir être effectué sur le serveur en fonction d'éléments supplémentaires dans l'entête de la requête du client. Ces éléments peuvent concerner aussi bien les langues acceptées (et celles qui sont préférées), que les types de fichiers. Voici un exemple d'entête de requête :

```
Accept-Language: fr; q=1.0, en; q=0.5
Accept: text/html; q=1.0, text/*; q=0.8, image/gif; q=0.6,
        image/jpeg; q=0.6, image/*; q=0.5, */*; q=0.1
```

Ici, le client déclare préférer le français mais accepte l'anglais, pour les types de documents, sa préférence va aux document html mais il accepte tout autre type text, enfin pour les images il accepte autant les gif que les jpeg. Néanmoins, il ne refuse pas tout autre type de média, le cas échéant.

On appelle ressource une entité conceptuelle qui correspond à une URI, et variantes les différentes représentations possibles pour cette ressource. Le travail du serveur http est ici de fournir les caractéristiques des différentes représentations d'une ressource. S'il existe plusieurs variantes, alors la ressource est négociable.

Pour fournir une représentation de toutes les variantes possibles, il y a deux possibilités pour Apache :

- soit il possède un fichier associé à la ressource qui contient une représentation pour chacune des variantes. Voilà ce qu'on pourrait trouver dans un fichier `foo.var` (on appelle ce genre de fichier une table de types)

  ```
  URI: foo

  URI: foo.en.html
  Content-type: text/html
  Content-language: en

  URI: foo.fr.de.html
  Content-type: text/html;charset=iso-8859-2
  Content-language: fr, de
  ```

  (Source de l'exemple : `http://httpd.apache.org/docs/2.0/content-negotiation.html`)

- soit c'est le serveur lui-même qui construit sa table de types à partir des extensions des fichiers disponibles pour la ressource et en fonction de `mod_mime`, le module de gestion des types mime, qui associe à une extension de fichier ses caractéristiques (type mime, encodage, langue, etc...). Pour que le serveur puisse effectuer cette table, il faut que l'`Option Multiviews` ait été posée sur la partie du site où se trouve la ressource. On peut donc résumer en disant que l'option `Multiviews` permet de faire correspondre la ressource `index.html` à l'une des variantes (au choix) `index.html.fr`, `index.html.en`, `index.html.gz`, ...

Le choix final de la variante renvoyée au client s'effectue soit sur le serveur http via l'algorithme de résolution de négociation d'Apache, soit, à la demande du client, est délégué au client lui-même.

La journalisation

Le répertoire `<serverRoot>/logs` contient différents journaux d'activités mis à jour par le serveur Apache. Parmi les fichiers d'historique se trouvent par défaut :

- `error_log` dans lequel le serveur `httpd` enregistre des informations concernant tous les problèmes rencontrés : les problèmes systèmes (problèmes de démarrage, d'arrêt, ...) et les problèmes concernant les documents publiés (document non trouvé, non accessible, ...).

  La directive `ErrorLog` permet d'indiquer le fichier qui va contenir cette historique (`<serverRoot>/logs/error_l` généralement).

  La directive `LogLevel` permet de fixer le type des problèmes à enregistrer en indiquant un des niveaux suivants (du type le plus spécifique au type le plus large) : `emerg`, `alert`, `crit`, `error`, `warn`, `notice`, `info`, `debug`. En indiquant emerg, seuls les problèmes de la plus haute importance (ceux rendant le système inutilisable) seront reportés. À l'autre bout de la chaîne, le niveau `debug` permet de reporter des problèmes qui ne sont pas vraiment des problèmes mais des messages utiles à un deboguage.

- `acces_log` contient des informations concernant toutes les requêtes traitées par le serveur.

  La directive `CustomLog` permet d'indiquer le fichier utilisé (`<serverRoot>/logs/access_log` par défaut) et le format des informations enregistrées. La syntaxe de cette directive est :

```
CustomLog désignationDuFichierLog formatUtilisé
```

Un format est défini par la directive `LogFormat` qui prend en paramètre une chaîne de caractères entourée de '' spécifiant ce que l'on veut afficher et le nom du format. Dans la chaîne spécifiant le format peut se trouver `%h` pour afficher l'adresse IP du client, `%u` pour afficher le userId du client, `%r` pour afficher la requête entière du client, `%m` pour afficher la méthode de la requête, `%U` pour afficher le chemin du document, `%q` pour afficher la chaîne paramètre de la requête, `%H` pour afficher le protocole de la requête, `%t` pour afficher le moment auquel la requête a été complètement traitée, `%>s` pour afficher le status retourné par le serveur, `%b` pour afficher la taille du contenu retourné au client, `%unNomi` pour afficher la valeur du champ d'en-tête portant le nom *unNom*,...

Remarque : dans le cas où le serveur ne peut fournir l'information demandée, un - est affiché. Par exemple, nous pouvons définir le format *monFormat* qui permettrait d'afficher uniquement l'adresse IP du client par :

```
LogFormat ''Adresse IP du client -> \%h'' monFormat
```

Serveurs virtuels

L'idée d'un serveur virtuel est le principe d'accéder à plusieurs serveurs web virtuels qui fonctionnent en fait sur une même machine physique. Par exemple, proposer un accès à la machine `www.mon-asso.fr` et un autre à la machine `www.mon-autre-asso.org`. On peut avoir des serveurs virtuels basés sur IP, i.e. chaque machine virtuelle possède une adresse IP différente, ou bien des serveurs virtuels basés sur le nom, i.e. une même IP correspond à plusieurs noms de machines (peut-être aussi appelé basé sur un hôte ou non basé sur IP).

En tout état de cause, le fait que ces serveurs virtuels tournent en définitive sur la même machine physique est complètement transparent pour l'utilisateur final.

Avec des serveurs virtuels

La directive `Listen` n'implémente aucun Serveur Virtuel. Elle sert simplement à indiquer au serveur principal sur quels adresses et ports écouter. Dans le cas où aucune section `<VirtualHost>` n'est utilisée, le serveur répondra de la même manière pour toutes les requêtes qu'il recevra. Des sections `<VirtualHost>` peuvent être utilisées pour qu'Apache réagisse différemment selon que la requête est destinée à telle adresse ou à tel port. Avant d'implémenter un Serveur Virtuel au moyen de la directive `VirtualHost`, la directive Listen doit être configurée pour que le serveur écoute sur l'adresse ou le port utilisé. Ensuite, une section `<VirtualHost>` devrait être utilisée pour qu'Apache réagisse différemment selon l'adresse ou le port. À noter que si un Serveur Virtuel `<VirtualHost>` est configuré sur une adresse et un port sur lesquels le serveur n'est pas à l'écoute, le Serveur Virtuel ne sera pas accessible.

Une section `VirtualHost` peut comprendre simplement un ensemble de directives (négociation de contenu, fichiers de journalisation, ...), mais il peut aussi contenir toutes les sections `Directory`, `Location`, `Files`, ... pour paramétrer le comportement du serveur virtuel.

Plusieurs serveurs

Une autre possibilité est de faire tourner plusieurs démons httpd (un pour chaque serveur virtuel). Cette solution est la plus sûre, dans la mesure où on souhaite que les différents serveurs ne puissent rien voir l'un de l'autre, mais c'est bien sûr aussi une solution plus coûteuse en terme de performances.

La correspondance des URLs et du système de fichiers

La directive `DocumentRoot` détermine la racine du site web. C'est ici qu'on trouve le fichier qui est retourné lorsque la requête s'adresse à la racine du site `http://www.mon-asso.org`

Exemple:

```
DocumentRoot /usr/web
```

L'accès à `http://www.mon-asso.org/index.html` correspond à `/usr/web/index.html`

Basiquement, le processus de correspondance consiste à prendre le chemin d'accès au document racine, et à ajouter au bout ce qui se trouve derrière `mon-asso.org`. On obtient ainsi l'adresse du document à retourner dans le système de fichiers. Mais il peut y avoir beaucoup de raisons pour que les documents publiés sur le site web ne se trouvent pas réellement dans le sous-arbre de racine `DocumentRoot`. Il faut alors des directives supplémentaires pour que le serveur puisse faire la correspondance.

Serveurs virtuels : on peut définir un document racine spécifique pour chaque serveur virtuel.

Directive `Alias` : un alias permet de définir comment on réécrit un chemin dans une URL en un autre chemin dans le système de fichiers.

Exemple :

```
Alias /docs /var/web
```

Ainsi, l'URL `http://www.example.com/docs/dir/file.html` correspondra au fichier `/var/web/dir/file.html` (modulo ce qu'on a vu sur la négociation de contenus).

La directive `ScriptAlias` fonctionne de la même manière, sauf que les document positionnés à l'endroit spécifié seront vus comme des scripts CGI.

Pour avoir plus de souplesse dans l'écriture des alias, on peut utiliser les directives `AliasMatch` et `ScriptAliasMatch` qui fonctionnent avec des expression régulières en paramètre.

Liens symboliques Un moyen simple est de suivre les liens symboliques spécifiés dans les répertoires... Cela nécessite que l'`Option FollowSymbolicLinks` soit posée à partir du sous-arbre qui contient les liens symboliques. Cette directive laisse pas mal de possibilités au propriétaire du sous-arbre, et délègue donc un peu de la sécurité du système...

Les répertoires des utilisateurs C'est la directive `UserDir` qui permet d'associer chemin d'accès à un répertoire dédié aux documents publics à l'intérieur du home directory d'un utilisateur. Ainsi, généralement,

```
UserDir public_html
```

permet de faire correspondre `http://www.mon-asso.org/ parrain` à `parrain/public_html/` .

Mais aussi :

- la redirection d'URLs (en cas de déménagement des documents), la directive `Redirect`
- la réécriture d'URLS...

Gestion des performances

Apache peut contrôler les ressources qu'il alloue.

Les directives `LimitRequestBody`, `LimitRequestFields`, `LimitRequestFieldSize`, `LimitRequestLine` permettent de limiter les ressources que va utiliser Apache pour lire une requête d'un client. Cela permet de contrer partiellement les attaques de déni de service (attaques qui consistent à saturer un serveur de requêtes, par exemple un simple envoi de requêtes `ping`).

Les directives `RLimitCPU`, `RLimitMEM` et `RLimitNPROC` permettent de limiter les ressources alloués aux process fils d'apache comme les scripts CGI ou les SSI.

Les directives `MaxClient`

La sécurité

Elle se situe à plusieurs niveaux :

- le contrôle des accès aux documents dans les fichiers de configuration principaux
- et via les fichiers `.htaccess`
- l'accès par authentification, restriction sur les domaines ou adresses des machines clientes;
- l'utilisation de connexions sécurisées, la gestion des certificats, etc... via une interface vers la librairie OpenSSL.

Autorisation ou interdiction d'accès

Il est possible d'autoriser ou d'interdire l'accès de certaines pages Web (des pages d'un répertoire par exemple) à des adresses IP, à des noms d'hôtes, à des groupes d'adresses IP ou bien encore des groupes de noms d'hôtes. Pour cela, on utilise les commandes `allow from ...` (pour autoriser) et `deny from ...` (pour interdire). Par exemple :

- `allow from all` : autorise l'accès à tout le monde,
- `allow from 12.23` : autorise l'accès aux adresses IP commençant par 12.23,
- `deny from .fr` : interdit l'accès à tous les noms d'hôtes du domaine .fr,
- `deny from belgique.iut-lens.univ-artois.fr` : interdit l'accès à l'hôte belgique.iut-lens.univ-artois.fr.

La directive `Order` permet de spécifier quelle est la propriété la plus prioritaire entre `allow` et `deny` :

```
Order deny, allow
```

spécifiera que `allow` est prioritaire (il a le dernier mot) par rapport à `deny`.

```
Order allow, deny
```

spécifiera que `deny` est prioritaire par rapport à `allow`.

Ainsi, les directives suivantes :

```
<Directory rep1>
  Order allow, deny
  Allow from all
  Deny from 123.34.56.45
</Directory>

<Directory rep2>
  Order deny, allow
  Allow from all
  Deny from 123.34.56.45
</Directory>
```

font en sorte que `rep1` est accessible à tout le monde sauf au client ayant pour adresse IP `123.34.56.45`, tandis que `rep2` est accessible à tout le monde.

Authentification par mot de passe

Il est possible de protéger l'accès de fichiers par mot de passe. Pour cela il faut utiliser les directives `AuthType` pour indiquer le type d'accès utilisé (`Basic` ou `Digest`), `AuthName` pour indiquer le domaine dans lequel les noms et les mots de passe doivent être valides. `AuthUserFile` indique le fichier contenant les noms d'utilisateurs et leur mots de passe, `AuthGroupFile` indique un fichier dans lequel est défini un ensemble de groupes d'utilisateurs. `Require` permet d'indiquer quels sont les utilisateurs qui pourront accéder aux documents du répertoire après authentification, on peut spécifier des noms d'utilisateurs, des noms de groupes d'utilisateurs ou bien encore `valid-user` pour indiquer tous les utilisateurs du fichier de mots de passe. Ainsi, la directive suivante :

```
<Directory rep>
  AuthType Basic
  AuthName secret
  AuthUserFile /home/jean/.htpasswd
  Require valid-user
</Directory>
```

spécifie que les documents du répertoire `rep` sont accessibles uniquement pour les utilisateurs spécifiés dans le fichier `/home/jean/.htpasswd` après une authentification par mot de passe de type `Basic`.

L'appel de programmes externes

Essentiellement des scripts CGI et des directives SSI, mais aussi interprétation de scripts PHP... Nous verrons cela en détail dans les TPs de manipulation.

Les scripts CGI

La directive `ScriptAlias` permet d'indiquer au serveur où se trouvent les scripts CGI. La syntaxe est

```
ScriptAlias cheminCGI répertoireCGI
```

où `cheminCGI` indique le chemin utilisé dans l'url pour atteindre les scripts CGI et `répertoireCGI` indique le répertoire où se trouvent les scripts CGI. Ainsi, la requête `http://serveur/cheminCGI/script` fera s'exécuter le script `répertoireCGI/script`.

L'option `ExecCGI` permet d'autoriser l'exécution de scripts CGI.

Les server-side scripts

Il est possible de mettre du code dans les différents documents publiés qui sera interprété par le serveur. Ce code doit être filtré par des interpréteurs avant que le serveur ne le retourne au client Il peut-être également filtré lorsqu'il est envoyé par le client (requête `POST` ou `PUT`) et reçu par le serveur. On peut placer dans une section `Files` ces filtres-interpréteurs. Par exemple, la directive suivante :

```
<Files *.php>
SetOutputFilter PHP
SetInputFilter PHP
</Files>
```

permet d'indiquer que les fichiers portant l'extension `.php` doivent être filtrés par l'interpréte PHP.

L'option `Includes` permet l'exécution du code côté serveur.