Serveurs Web — Apache

- Rôle d'un serveur Web
- Exemples de serveurs
- Serveur Apache
  1. Principes
  2. Installation
  3. Configuration
  4. Sécurité et contrôles

Rôle d'un serveur Web

Serveur Web = serveur HTTP $\Rightarrow$

- Réponde aux requêtes des clients
- Déterminer le type de document à transmettre
- Respecter les préférences des clients
- Vérifier l'identité du client et/ou son droit d'accès
- Interpréter les instructions encodées dans les éléments HTML spéciaux (SSI, ASP)
- Exécuter des CGIs
- Simuler des hôtes virtuels
- Faciliter la maintenance (logs)
- Jouer le rôle de proxy

Exemples de serveurs

Il existe des serveurs pour toutes les architectures, pour toutes les S.E., gratuits ou payants.

Choisir de préférence un serveur sur un S.E. robuste (p.e. Unix) et largement utilisé.

Serveurs gratuits :

- Apache (Unix, Windows 95/NT)
- NCSA (ancêtre Apache)
- CERN (dépassé)
- Jigsaw - W3C (en Java)

Serveurs commerciaux :

- Netsite - Netscape (Unix, Windows NT)
- Internet Information Server - Microsoft (Windows NT)
- WebSite - O'Reilly ()
- MacHTTP (Mac)

Comparatif des serveurs Web :
http://www.netcraft.com/survey/

Serveur Apache

- Né en 1995 à partir du serveur NCSA, par développements "open source"
- Apache = "A PAtCHy Server"
- Gratuit, portable sur tous les Unix et Windows 95/NT, sources disponibles.
- Très robuste, complet, performant, facile à configurer et à étendre
- Références :
  - http://www.apache.org/ : site officiel d'Apache
  - R. Bowen, K. Coar & co. Apache Serveur Unleashed, Sams Publishing 2000. Traduction : CampusPress France, 2000

1. Principes (1/3)

Pré-fork

- Algorithme traditionnel de serveur :

```
while(1) {
    accept();
    if (!fork()) traite_la_requete();
}
```

- `fork` est une primitive système très coûteuse
- Pré-fork en Apache : plusieurs processus sont lancés dès le début ; ils écoutent le même port TCP.
  - Unix répartit les connexions sur les différents processus.
  - Chaque processus traite les requêtes sans appel à `fork`.
  - Apache auto-régule le nombre de processus en fonction de la charge.

1. Principes (2/3)

Modularité

- Architecture modulaire d'Apache :
  - "Core modules" : minimum indispensable
  - Un ensemble de modules implémentant chacun une fonctionnalité
- Avantages :
  - On peut ajouter ou enlever les modules à volonté
  - Modules standard
  - Modules contributaires
  - API très simple pour développer ses modules
- Installation/dés-installation d'un module
  - Editer le fichier `src/Configuration` pour ajouter/enlever les lignes du type

    `Module imap_module mod_imap.o`
  - Exécuter le script `src/Configure` qui génère un `Makefile`
  - Compiler : `make`

  ou utiliser directement le script `configure`.

1. Principes (3/3)

Gestionnaires (Handlers)

- Action du serveur en réponse à un type de requête ou de ressource.
- Un gestionnaire peut être :
  - de base ("built-in"), inclus par défaut :

```
default-handler    send-as-is
cgi-script         imap-file
server-info        server-status
type-map
```

  - programmé dans un module supplémentaire
  - programmé par l'utilisateur

  Dans les deux derniers cas, pour utiliser le gestionnaire il faut préciser (à la configuration) le nom de l'action et à qui elle s'applique.
  Exemple :

```
Action add-footer /cgi-bin/footer.pl
AddHandler add-footer .html
```


2. Installation

1. Obtenir Apache : http://httpd.apache.org/dist/httpd/ ou ftp://ftp.script.jussieu.fr/pub/www/apache/ ($\approx$ 2.1MB)
2. Extraire les sources dans $www (p.e. /usr/local/www)

```
cd $www; tar -zxf apache_1.3.22.tar.gz
mv apache-1.3.22 src; cd src
```

3. Configurer l'installation
   - comme serveur statique, c-à-d. les modules sont chargés en mémoire au lancement (problème de performances)

```
./configure --prefix=$www
            --enable-module=most
```

   - comme serveur dynamique, c-à-d. les modules sont chargés/déchargés en fonction de besoins (DSO, *Dynamic Shared Object*)

```
./configure --prefix=$www
            --enable-module=most
            --enable-shared=max
```

4. Compiler et installer

```
make; make install
```

5. Configurer le fonctionnement : éditer $www/conf/httpd.conf

6. Démarer (versions $\geq$ 1.3)

   `$www/bin/apachectl start`

   Autres options :

   - `configtest` : test du fichier de configuration
   - `restart` : redémarrage, équivalent à

     `$www/bin/httpd -HUP 'cat $www/logs/httpd.pid'`

   - `graceful` : redémarrage en gardant les connexions en cours

7. Tester l'exécution de httpd

   `ps -aux | grep httpd`

8. Arrêter le serveur

   `$www/bin/apachectl stop`

   ou

   `$www/bin/httpd -HUP` *pid*

9. Nettoyer le répertoire des sources `src` ($\approx$ 10MB).

A faire en TP !

Arborescence du répertoire $www

- $www/bin/ : exécutables (httpd, apachectl, etc.)
- $www/cgi-bin/ : contient les CGIs
- $www/conf/ : fichiers de configuration
- $www/htdocs/ : racine des documents Web
- $www/htdocs/index.html : page de garde du serveur
- $www/icons/ : racine pour les images
- $www/include/ : API Apache
- $www/libexec/ : modules Apache
- $www/logs/ : fichiers rapport
- $www/man/ : manuel du serveur
- $www/proxy/ : racine du proxy

3. Configuration d'Apache

= ensemble de *directives* placées dans des fichiers texte.

Syntaxe des fichiers de configuration :

- une directive par ligne
- # en début de ligne = commentaire

Syntaxe des directives :

- simple : *directive valeur1 ... valeurN*
- *section* : <*directive par1 ... parN*> ... </*directive*>

Porté des directives :

- globale (hors sections)
- partielle :
  `<Directory>` `<DirectoryMatch>`
  `<Files>` `<FilesMatch>`
  `<Location>` `<LocationMatch>`
  `<Proxy>` `<ProxyMatch>` `<VirtualHost>`
- conditionnelle par rapport à un module ou une définition :
  `<IfModule>` `<IfDefine>`

Contextes d'utilisation des directives :

server config, *virtual host*, *directory*, *.htaccess*

3.1 Fichiers de configuration

Fichier principal : `httpd.conf`

- définit la configuration globale du serveur principal et des hôtes virtuels
- lu au lancement du serveur
- peut inclure d'autres fichiers de configuration (directive `Include`)

Fichier pour les types MIME : `mime.types` (par défaut) ou nom précisé par la directive `TypesConfig`

- correspondance extension de fichier et types MIME
- lu au lancement du serveur

Fichiers de configuration locaux : `.htaccess` (par défaut) ou nom précisé par la directive `AccessFileName`

- directives applicables qu'au répertoire où il apparaît et à ses sous-répertoires
- limitation des directives qui peuvent y être utilisées (directive `AllowOverride`)
- lu à chaque requête sur le répertoire ou ses sous-répertoires

3.2 Sections de configuration (1/2)

Types :

- évalués au (re)lancement du serveur : `IfDefine`, `IfModule`

```
<IfDefine RedirectOpt> # httpd -DRedirectOpt
   Redirect / http://otherserver/
</IfDefine>
<IfModule mod_iup.c>
   SetHandler send-as-is
</IfModule>
```

- évalués à chaque requête, contenu applicable si correspondance : `Directory*, Files*, Location*, Proxy*, VirtualHost`

```
<Directory /usr/local/www/dir1>
  Options +Indexes
</Directory>
```

Sections “système de fichiers” versus “locations” :

- `Directory*, Files*` configurent les parties du système de fichiers de la machine
- `Location*` configurent les ressources disponibles sur le serveur HTTP (peut ne pas correspondre à un fichier !).

3.2 Sections de configuration (2/2)

Ordre d’évaluation :

1. `<Directory>` et `.htaccess`
2. `<DirectoryMatch>`
3. `<Files>, <FilesMatch>`
4. `<Location>, <LocationMatch>`

Exemple :

```
<Location>
  D
</Location>
<Files f.html>
  C
</Files>
<DirectoryMatch "^.*b$">
  B
</DirectoryMatch>
<Directory /a/b>
  A
</Direcory>
```

Ordre d’application des directives : A, B, C, D !

3.3 Configuration globale du serveur

Dans `httpd.conf`, quelques exemples ...

Type du serveur (inetd ou daemon) :

```
ServerType standalone
```

Répertoire d'installation ($www) :

```
ServerRoot /usr/local/www
```

Nom du fichier contenant le pid du démon "maître" :

```
PidFile logs/httpd.pid
```

Port d'écoute :

```
Port 8080
```

Nom utilisateur et/ou groupe sur lequel tournera le serveur.
Ne jamais mettre root !

```
User sighirea
Group ens
```

Régulation des connexions :

```
KeepAlive on
KeepAliveTimeout 15
MaxKeepAliveRequest 1000
MaxRequestPerChild 256
```

Régulation du nombre de processus :

```
StartServers 5
MinSpareServers 5
MaxSpareServers 10
MaxClients 256
```


3.4 Configuration de l'accès aux ressources

Il faut préciser :

- la ressource en utilisant les directives "section" (fichier, répertoire, location, proxy)
- la méthode d'accès par la directive `Limit`

  Exemple :

  ```
  <Directory /www/htdocs/protected>
     protections1
     <Limit DELETE>
        protections2
     </Limit>
     protections3
  </Directory>
  ```

  Les `protections2` sont appliquées que pour la méthode `DELETE`.
- le contrôle appliqué : IP ou non-IP

Contrôle par l'adresse IP de la machine ou du réseaux

Exemple :

```
<Directory /www/htdocs/protected>
   order deny,allow
   deny from all
   <Limit DELETE>
      allow from 193.51.192.27
   </Limit>
   allow from 10.10.64 .apache.org
</Directory>
```

Algorithme d'évaluation parametré par `order` qui contrôle l'état initial et l'application des directives...

Dans l'exemple ci-dessus :

1. Etat initial : `allow from all` (càd `from all` de la deuxième valeur d'`order`).
2. Evaluation de toutes les directives `deny`.
3. Evaluation de toutes les directives `allow`.

Contrôle par authentification (login, mot de passe)

Exemple :

```
<Directory /www/htdocs/secret>
   AuthName "Secret Zone"
   AuthType Basic
   AuthUserFile /www/passwd/.htpasswd-secret
   require valid-user
</Directory>
```

Directives :

- `AuthName` définit le domaine
- `AuthType` peut être `Basic` ou `Digest`
- `AuthUserFile` spécifie le fichier passwd généré par :
  ```
  htpasswd -c .htpasswd-secret dupont
  ```
- `require` précise une liste d'utilisateurs ou groupes.
  Autres exemples :
  ```
  require user dupont martin
  AuthGroupFile /www/passwd/groups
  require group ens
  ```

  et `/www/passwd/groups` contient une ligne qui definit le groupe `ens`.

3.5 Configuration de l'exécution des scripts

Configurer les SSI :

1. Inclure `mod_include` à la compilation.
2. Définir le gestionnaire de SSI :
```
AddType text/html .shtml
AddHandler server-parsed .shtml
```
3. Donner le droits d'exécuter des SSI
   (a) dans un répertoire :
```
<Directory /www/htdocs>
   Options +Includes
</Directory>
```
   (b) pour les fichier HTML avec droits d'exécution
```
XBitHack on
```


Configurer les CGI :

1. Inclure `mod_cgi` à la compilation.
2. Définir le gestionnaire de CGI :
```
AddHandler cgi-script .cgi .pl
```
3. Donner le droits d'exécuter des CGI
   (a) globalement :
```
ScriptAlias /cgi-bin/ "/www/cgi-bin/"
```
   (b) dans un répertoire :
```
<Directory /www/htdocs/local/cgi-bin>
   Options +ExecCGI
</Directory>
```

3.6 Permettre une configuration locale

.. par des fichiers `.htaccess`.

Par défaut, la configuration locale désactivée :

```
<Directory />
   ...
   AllowOverride None
</Directory>
```

L'activation de la configuration locale doit préciser :

- le répertoire (section `Directory*`)
- les classes de directives qui peuvent apparaître

Attention : la configuration locale est très coûteuse, car chargement dynamique avec recherche itérative dans le chemin.

Exemple : si la configuration locale est activée dans le répertoire `/`, l'accès à `http://my.net/docs/file.html` provoque recherche et chargement des fichiers `.htdocs` dans :

```
/www/.htaccess
/www/htdocs/.htaccess
/www/htdocs/docs/.htaccess
```


Classes de directives à utiliser dans `AllowOverride` :

- `AuthConfig` : directives d'autorisation d'accès, p.e.
  `AuthType AuthName AuthUserFile AuthGroupFile`
  `<Limit> require`

- `FileInfo` : directives de définition de type de document, p.e.
  `AddEncoding AddHandler AddIcon AddLanguage`
  `AddType RedirectPermanent RedirectTemp`

- `Indexes` : directives de contrôle des indexes de répertoires générés automatiquement, p.e.
  `IndexIgnore IndexOptions AddAlt`
  `AddDescription DirectoryIndex`

- `Limit` : directives de contrôle d'accès par site, p.e.
  `allow, deny, order`

- `Options` : directives d'accès spécifique aux répertoires, p.e.
  `XBitHack, Options ExecCGI,`
  `Options SymLinksIfOwnerMatch,`
  `Options MultiViews`
  `Options Includes, Options Indexes`

Exemple :

Si dans le `httpd.conf` on a

```
<Directory /www/htdocs/dir>
   ...
   AllowOverride AuthConfig Limit Options
</Directory>
```

alors dans le `.htaccess` du répertoire `/www/htdocs/dir` on peut avoir :

```
order deny,allow
deny from all
allow from 193.51.192.27
Options +Indexes
```

mais pas

```
AddType text/iup3 .iup3
```


3.7 Configurer la négociation de contenu

Activer la négociation de contenu pour une ressource (`Options All` ne l'inclut pas) :

```
Options MultiViews
```

Deux façons de faire pour le serveur Apache :

- *explicite* : en utilisant un *fichier de variantes* (`.var`) qui enumère les différentes variantes explicitement
- *implicite* : en utilisant un recherche de variantes pour la ressource donnée.

Méthode explicite :

1. ajouter à la configuration le type fichier de variantes

   ```
   AddHandler type-map var
   ```

2. définir le fichier de variantes

   ```
   URI: english.html
   Content-Language: en;qs=0.6
   URI: francais.html
   Content-Language: fr;qs=0.4
   ```

3. la requête doit s'effectuer sur le fichier .var

Méthode implicite :

1. ajouter à la configuration les variantes supportées

```
AddLanguage en .eng
AddLanguage fr .fra
AddType application/pdf pdf
```

2. créer les variantes, p.e. `file.eng` et `file.fra`

3. la requête peut s'effectuer
   - sur le nom du fichier sans l'extension discriminante, p.e. `file`
   - sur l'une des variantes

3.8 Hôtes virtuels

Exemple de configuration générique (2 hôtes) :

```
<VirtualHost Addr1>
    ServerAdmin webmaster@addr1.domain.com
    DocumentRoot /www/htdocs/server1
    ServerName addr1.domain.com
    ErrorLog /www/logs/server1/error_log
    CustomLog /www/logs/server1/access_log
</VirtualHost>
<VirtualHost Addr2>
    ServerAdmin webmaster@addr2.domain.com
    DocumentRoot /www/htdocs/server2
    ServerName addr2.domain.com
    ErrorLog /www/logs/server2/error_log
    CustomLog /www/logs/server2/access_log
</VirtualHost>
```

- Hôtes virtuels IP : Addr1 et Addr2 sont deux adresses IP différentes
- Hôtes virtuels IP basé sur un port : Addr1 et Addr2 spécifient en plus un port défini par

  ```
  Listen Addr1:port1
  Listen Addr2:port2
  ```

- Hôte virtuel nommés : Addr1 et Addr2 sont deux adresses IP identiques, spécifiées par

  ```
  NameVirtualHost Addr1
  ```

3.9 Configuration de l'administration

càd, des informations stockées dans les fichiers de log et de leur accès.

Fichiers de log :

- logs/access.log : une ligne par requête reçue
  `<machine> <date> <requête> <code-retour> <taille>`
- logs/errors.log : description des erreurs
  - Démarrages/redémarrages du serveur
  - Documents non trouvés
  - Requêtes incorrectes
  - CGIs terminés anormalement
- autres logs obtenus en chargeant les modules ad-hoc :
  - agent.log : identification de navigateur
  - referer.log : en-tête HTTP `Referer:`

Accès aux informations d'adminstration :

- Handler Info :
  - Module `mod_info`, URL /server-info
  - Configuration

```
<Location /server-info>
   SetHandler server-info
   AuthType Basic
   AuthUserFile /etc/.htpasswd-info
   Require valid-user
</Location>
```

  - Exemple de rapport :

```
Server Version: Apache/1.3.12 (Unix)
Server Built: Oct 24 2000 19:57:46
Run Mode: standalone
```

- Handler Status
  - Module `mod_status`, URL /server-status
  - Configuration : similaire
  - Exemple de rapport :

```
Current Time: Mon Nov 12 10:07:26 2001
Restart Time: Fri Nov 9 11:06:14 2001
Server uptime: 2 days 23 hours 1 minute 12 seconds
Total accesses: 11362 - Total Traffic: 93.6 MB
```

3.10 Configuration proxy et cache

- Apache peut être serveur proxy pour HTTP,...
- Apache 1.3.XX ne peut pas cacher les requêtes HTTP/1.1, utiliser Squid (voir http://squid.nlanr.net/)
- Apache 2.0essaye de palier à cette limitation.
- Pour mettre en place un serveur proxy Apache :
  - Ajouter `mod_proxy`
  - Recompiler le serveur
  - Configurer le proxy :
    - `ProxyRequests` : active/désactive le proxy
    - `ProxyRemote` : établie l'hiérarchie de proxy
    - `ProxyBlock` : empêche l'accès
    - `NoProxy` : accès direct, sans proxy
  - Configurer le cache :
    - `CacheRoot` : activation et répertoire racine
    - `CacheSize` : kB d'espace réservé
    - `CacheMaxExpire` : temps max de stockage sans vérification
    - `NoCache` : accès/requêtes sans cache

Exemple de configuration `httpd.conf` :

```
ServerName proxy.a.org
User nobody
Group nobody
Port 8080
ServerType standalone
ServerRoot /etc/hhtpd
ProxyRequests On
NoProxy .a.org
ProxyDomain .a.org
CustomLog logs/proxy_log common
<Location />
   Order Deny,Allow
   Deny from all
</Location>
<Directory proxy:*>
   Order Deny,Allow
   Allow from .a.org 10
</Directory>
```

7. Sécurité et contrôles

Un serveur Web est :

- un programme qui tourne sur une machine
- un serveur d'informations statiques ou dynamiques

Donc il faut penser aux différents niveaux de protection :

- Protection de la machine serveur :
  - appliquer les patches de sécurité
  - éviter les dépendances externes
  - enlever, si possible, les utilisateurs
  - enlever tous les services inutiles
- Protection vis-à-vis du `httpd` :
  - choisir un programme largement répandu sur une architecture connue
  - préférer les serveurs en code source
  - suivre les patches
- Protection des informations :
  - choisir des bons droits (read-only)
  - limiter l'accès aux informations pour certains sites
  - vérifier régulièrement les accès faits
  - limiter/vérifier les exécutions de CGI

Protection du système de fichiers

- Utilisateur $\neq$ root
- Fichiers en lecture seule
- Limiter les liens symboliques : `Options`
- Limiter les informations données par les indexes de répertoires : `Options Indexes`, `FancyIndexing`

Protection des URLs

Ce qu'on peut vérifier :

- ce qu'on a
- ce qu'on sait
- ce qu'on est

Donc on a des contrôles :

- d'accès discrétionnaires : "ce qu'on a ou sait"
  - authentification par mots de passe Basic ou Digest
  - limitation des méthodes `<Limit>`
- obligatoires : "ce qu'on est"
  - limitation par adresse IP, navigateur : `allow`

Diagnostique (1/3)

- Symptôme : le serveur ne rend plus d'informations ("Deny of Service")
  - Attaque :
    - Surcharge de la machine
      - avec des robots-*crunchers*
      - avec CGI
    - Explosion de l'accès réseau
  - Défense :
    - Réplication et répartition de la charge
      - avec DNS
      - avec un firewall
      - avec un accélérateur HTTP (Squid)
    - Protection de la pile IP avec un firewall

Diagnostique (2/3)

- Symptôme : le serveur rend des informations falsifiées
  - Résultat d'une machine crackée
  - Attaque :
    - bogue du serveur
    - CGI mal écrits
    - gestion laxiste des CGI
    - utilisation SSI/PHP
    - édition (PUT) mal gérée
  - Défense :
    - données read-only
    - identité du serveur $\neq$ root
    - utilisation de librairies protégées pour les CGI
    - pas de SSI/PHP
    - séparation serveur d'édition / serveur de consultation

Diagnostique (3/3)

- Symptôme : le serveur rend trop d'informations
  - Accès à des informations protégées
  - Mauvaise protection à la configuration
  - Fissure de protections :
    - Authentification Basic
    - Authentification IP/nom
      - Facile à mettre en oeuvre
      - Problème de proxy
      - Attaque par nom et IP détournés
    - Authentification Digest
    - Cryptage
      - pour protéger le contenu
      - pour conserver l'intégrité