Configuration de base

Hello, handsome.

Le onzième docteur

1 Configuration de base d'Apache

La documentation officielle, https://httpd.apache.org/docs/current deviendra vite votre meilleure amie. Que ce soit pour vérifier la syntaxe d'un élément de configuration, savoir si une modification nécessite un redémarrage ou juste un rechargement d'Apache, trouver des exemples de configuration, c'est bien là qu'il vous faudra aller voir.

Ne croyez pas que la documentation est uniquement pour les débutantes : il est plus facile de savoir où chercher que de retenir les dizaines, centaines ou milliers d'éléments de configuration différents… même les plus usuels !

1.1 Les directives

Le comportement d'Apache est contrôlé par un ensemble d'instructions appelées *directives*. Ces directives se décomposent en deux groupes : les directives *natives*, disponibles sur toute installation d'Apache, et les directives d'*extension*, liées aux extensions installées sur un serveur Apache particulier.

Une directive est généralement constituée d'un nom de directive et de paramètres séparés par des espaces, sauf en ce qui concerne les directives appelées « conteneurs » (voir plus bas). Le nom des directives est insensible à la casse (vous pouvez donc les écrire avec ou sans majuscules) mais on les écrit généralement en Camel case[1].

Exemple :

```
ServerRoot /var/www/html
```

1.2 Les conteneurs

Les conteneurs sont des directives spéciales sous forme de balises ouvrantes et fermantes qui vont permettre de limiter la portée des directives encadrées par ces balises. La balise ouvrante possédera généralement un ou des paramètres séparés par des espaces (mais peut aussi parfois ne pas en nécessiter).

Exemple :

```
<Directory /var/www/html/>
    DirectoryIndex tennant.html index.html
</Directory>
```

1.3 Organisation de la configuration

Sur certaines distributions, le fichier de configuration principal, `apache2.conf` est organisé de façon à sous-traiter les directives qu'il est sensé contenir à d'autres fichiers, contenus parfois dans des sous-répertoires du répertoire principal d'installation. L'inclusion de sous-répertoires permet de séparer le configuration en petits blocs facilement repérables, comme `conf-available/security.conf`. Ce mécanisme est utilisé afin de faciliter la maintenance d'Apache par rapport à la philosophie de ces distributions.

1. https://fr.wikipedia.org/wiki/Camel_case

Sur Debian, par exemple, il faut considérer le fichier `apache2.conf` pour la configuration principale d'Apache, les sous-répertoires `conf-available` pour des morceaux de configurations supplémentaires disponibles, et `conf-enabled` pour les morceaux de configurations supplémentaires actifs (c'est-à-dire effectivement pris en compte par Apache), les sous-répertoires `mod-available` et `mod-enabled`pour les modules d'extension disponibles et actifs et enfin les sous-répertoires `sites-available` et `sites-enabled` pour la configuration des hôtes virtuels (les sites web) disponibles et actifs.

Exemple d'organisation de la configuration d'Apache dans Debian :

```
/etc/apache2
├── apache2.conf
├── conf-available
│   ├── apache2-doc.conf
│   ├── big_files_no_cache.conf
│   ├── ...
│   └── serve-cgi-bin.conf
├── conf-enabled
│   ├── big_files_no_cache.conf -> ../conf-available/big_files_no_cache.conf
│   ├── ...
│   └── serve-cgi-bin.conf -> ../conf-available/serve-cgi-bin.conf
├── envvars
├── magic
├── mods-available
│   ├── access_compat.load
│   ├── alias.conf
│   ├── alias.load
│   ├── ...
│   ├── setenvif.conf
│   ├── setenvif.load
│   ├── ssl.conf
│   └── ssl.load
├── mods-enabled
│   ├── access_compat.load -> ../mods-available/access_compat.load
│   ├── alias.conf -> ../mods-available/alias.conf
│   ├── alias.load -> ../mods-available/alias.load
│   ├── ...
│   ├── ssl.conf -> ../mods-available/ssl.conf
│   └── ssl.load -> ../mods-available/ssl.load
├── ports.conf
├── sites-available
│   ├── 000-default.conf
│   └── exemple.org
├── sites-enabled
│   └── 000-default -> ../sites-available/000-default.conf
└── ssl.conf
```

Attention ! Pour activer un morceau de configuration, un module ou un site disponible, on ne déplace pas son fichier de `xxx-available` vers `xxx-enabled`. On se contentera de faire un lien symbolique vers le fichier, par exemple : `ln -s ../conf-available/apache2-doc.conf /etc/apache2/conf-enabled`.

Cette manière de faire permet de conserver les fichiers de configuration disponibles pour les activer et les désactiver à loisir, sans risquer de les supprimer définitivement. À noter que Debian fournit des utilitaires simplifiant l'activation et la désactivation des fichiers de configuration disponibles :

- `a2enconf` pour activer un morceau de configuration disponible, `a2disconf` pour le désactiver ;
- `a2enmod` et `a2dismod` pour activer et désactiver des modules ;
- `a2ensite` et `a2dissite` pour activer et désactiver des sites.

Avec l'exemple précédent, cela donnerait : `a2enconf apache2-doc`.

1.4 Portée et contexte des directives

Chaque directive agit dans un contexte[2] particulier, c'est à dire que son action est limitée à ce contexte. Il est important de connaître et de comprendre ces contextes pour mesurer la *portée* des directives qui y sont définies.

Les quatre contextes existants sous Apache sont :

- *Le contexte de configuration globale* du serveur. Ce contexte contient des directives qui ne peuvent être présentes qu'à cet endroit (comme par exemple `PidFile`, qui définit le fichier où Apache écrira son PID[3]), ainsi que d'autres directives pouvant être également définies dans d'autres contextes. Pour ces dernières directives, une définition dans le contexte de configuration globale du serveur permet de fixer une valeur par défaut à ces directives dans le cas où elles ne sont pas redéfinies dans d'autres contextes.
- *Le contexte d'hôte virtuel*, défini par le conteneur `<VirtualHost>`.
- *Le contexte répertoire*. Ce contexte inclut les directives inclues dans un des 5 conteneurs suivant : `<Directory>` (relatif au répertoire utilisé pour répondre à la requête), `<Files>` (relatif au fichier supposé être renvoyé), `<Location>` (relatif à l'URI demandée), `<If>` et `<Proxy>`.
- *Le contexte `.htaccess`*. Un fichier `.htaccess` est un fichier de configuration complémentaire pouvant être placé dans les répertoires contenant les fichiers servis par Apache. Le but de ces fichiers est de décentraliser les directives liées à Apache, en permettant par exemple à un utilisateur « lambda » (c'est-à-dire non privilégié) de configurer les requêtes liées aux données qui le concerne sans avoir à modifier le fichier `apache2.conf` (ce qu'il ne peut pas faire normalement de toute façon).

Attention : certaines directives peuvent être utilisée dans tous les contextes, d'autres uniquement dans un ou plusieurs contextes particuliers. La documentation officielle vous permettra de savoir dans quel contexte vous pouvez utiliser la directive souhaitée.

1.5 Configuration du contexte de configuration globale du serveur

Voici quelques directives utilisables dans le contexte de configuration globale. Certaines lui sont propres (et donc ne peuvent être utilisées dans un autre contexte) et d'autres non (définissant ainsi les valeurs par défaut de ces directives comme vu plus haut).

- `Servername` : contient le nom d'hôte (nom de domaine) et le port que le serveur utilise pour s'authentifier lui-même. Placée dans un contexte d'*hôte virtuel*, cette directive définit généralement le nom de domaine auquel répondra le site configuré.
- `ServerRoot` : correspond au répertoire d'installation d'Apache, contenant en particulier ses fichiers de configuration (`/etc/apache2` sur Debian). Il est possible de changer au démarrage d'Apache la valeur donnée par cette directive grâce à l'option `-d`. C'est à partir de ce répertoire que seront calculés les chemins relatifs contenus dans les autres directives (comme `Include` par exemple).
- `Include` : charge la configuration contenue dans les fichiers ou dossiers spécifiés (exemple : `Include ports.conf`)
- `IncludeOptional` : charge la configuration contenue dans les fichiers ou dossiers spécifiés mais ne génère pas d'erreur si ceux-ci n'existent pas (exemple : `IncludeOptional conf-enabled/*.conf`)
- `DocumentRoot` : permet de définir la racine des documents distribués par Apache (le répertoire où chercher les fichiers). Ce chemin est ainsi utilisé comme base pour le calcul des noms de fichiers correspondant aux requêtes reçues par le serveur.
- `ErrorDocument` : permet de changer le comportement par défaut d'Apache lorsqu'une erreur est rencontrée. Cette directive permet une redéfinition par code d'erreur, en affichant un message différent ou en redirigeant la requête vers un document traitant l'erreur en question. Cela permettra par exemple d'afficher une page 404 personnalisée.
- `User` et `Group` : permettent de définir sous quelle identité s'exécuteront les fils du processus principal Apache.

2. https://httpd.apache.org/docs/current/mod/directive-dict.html#Context
3. https://fr.wikipedia.org/wiki/Identifiant_de_processus

- `DirectoryIndex` : définit dans l'ordre la liste des fichiers qui doivent être recherchés lorsqu'une requête concerne un répertoire (par exemple `DirectoryIndex index.html index.php index.htm`). Le premier fichier présent est renvoyé.
- `Options` : permet de contrôler certaines fonctionnalités du serveur dans un répertoire particulier. Cette directive peut être associée aux valeurs suivantes (un « + » en préfixe d'une valeur force l'activation de cette option — action par défaut — et un « - » en force la désactivation. Notez que toutes les valeurs doivent être préfixées, ou aucune : il n'est pas possible de mixer des valeurs avec ou sans préfixe) :
  - `ExecCGI` : autorise l'exécution de script CGI
  - `FollowSymLinks` : le serveur peut suivre les liens symboliques de ce répertoire, mais cela ne change pas la correspondance établie avec les sections `<Directory>`.
  - `Includes` : autorise les SSI[4] (*Server-Side Includes*, un langage de programmation interprété par les serveurs web)
  - `IncludesNOEXEC` : autorise les SSI, mais leurs commandes `#exec` et `#include` sont désactivées
  - `Indexes` : renvoie une liste formatée de ce répertoire si une requête concernant ce répertoire est établie et qu'il n'existe pas de fichier correspondant à une des valeurs de la directive `DirectoryIndex`.
  - `Multiviews` : autorise les multiviews à contenu négocié (par exemple pour fournir `foo.fr.html` ou `foo.en.html` selon la préférence du navigateur quand la ressource `foo` est demandée)
  - `SymLinksIfOwnerMatch` : suit les liens symboliques dont le fichier ou le répertoire appartient au même utilisateur que le lien.
  - `All` : inclut toutes les options, sauf `Multiviews`. C'est la valeur par défaut.
  - `None` : aucune de ces options.
- `IndexOptions` (avec la valeur `FancyIndexing`) : fournit un listage du contenu des répertoires plus agréable lorsque l'option `Indexes` est activée.

2 Configuration de base de Nginx

La documentation officielle, https://nginx.org/en/docs/ vous sera, ici aussi, très précieuse. Tout particulièrement les pages https://nginx.org/en/docs/dirindex.html et https://nginx.org/en/docs/varindex.html. Créez-vous des marque-pages dans votre navigateur !

Comme pour Apache, on retrouvera généralement un découpage de la configuration en plusieurs fichiers répartis dans différents répertoires.

Sur Debian :

- `nginx.conf` : fichier de configuration générale, qui contient des inclusions d'autres fichiers de configuration ;
- `modules-available` et `modules-enabled` : dossiers contenant les fichiers permettant l'activation des modules de Nginx ;
- `conf.d` : contient des fichiers de configuration générale pour le serveur web (Nginx peut aussi faire office de proxy mail mais la configuration de cette fonctionnalité ne se fera pas dans `conf.d`) ;
- `sites-available` et `sites-enabled` : dossiers contenant la configuration des hôtes virtuels.

Le dossier `snippets` est particulier : il ne fait pas l'objet d'inclusions dans la configuration de base. Il est là pour vous permettre de ranger des bouts de configuration que vous pourrez alors vous-même inclure dans vos configurations d'hôtes virtuels. L'intérêt est de mutualiser la configuration (à quoi bon écrire 20 fois le même bout de configuration alors qu'une inclusion est plus rapide ?).

2.1 Directives et contextes

Les contextes Nginx n'ont rien à voir avec les contextes d'Apache.

4. https://fr.wikipedia.org/wiki/Server_Side_Includes

— une directive est constituée d'un nom de directive, de paramètres séparés par des espaces et se terminant par un point-virgule :
`listen 80;`
— les contextes ont une structure similaire à ceci près qu'ils se terminent par des accolades encadrant des directives ou des contextes supplémentaires :

```
http {
    server {
        server_name example.org;
    }
}
```

2.2 Portée des directives

Les directives peuvent être positionnées dans zéro, un ou plusieurs contextes. Lorsqu'elles ne peuvent être positionnées dans aucun contexte, on parlera du contexte `main`, correspondant plus ou moins [5] au contexte de configuration globale d'Apache vu précédemment.

Les contextes possibles pour une directive peuvent varier selon cette directive (tout comme pour Apache). Ainsi, si `internal` ne peut être placé que dans le contexte `location`, la directive `ignore_invalid_headers` peut être placée dans les contextes `http` et `server`.

Pour trouver dans quel contexte doit se situer une directive, reportez-vous à la documentation officielle. La liste alphabétique des directives et des contextes est disponible sur https://nginx.org/en/docs/dirindex.html.

3 Différences et comparaison des contextes et directives Apache et Nginx

3.1 Les contextes

!!! ATTENTION !!! Ne pas confondre les contextes Apache et Nginx, j'utilise le même terme mais ils ne désignent pas la même chose.

Les contextes Nginx sont plutôt similaires aux conteneurs d'Apache : un élément encadrant une ou des directives, ce qui en limite la portée. Par exemple, pour une directive n'agissant que pour le chemin `/private/` (ex : `https://exemple.org/private/`), on aura pour Apache :

```
<Location /private/>
    Require all denied
</Location>
```

Et pour Nginx :

```
location /private/ {
    deny all;
}
```

Contexte de configuration globale du serveur

Si le contexte `main` ressemble au contexte de configuration globale d'Apache, il existe toutefois une différence de taille. En effet, Nginx n'est pas qu'un serveur web, il peut aussi faire office de proxy mail.

5. voir la section suivante

Dès lors, les directives du contexte de configuration globale d'Apache relatives au programme lui-même (`User` et `Group`) auront bien généralement des directives équivalentes (`user`, qui permet aussi de définir le groupe auquel appartient le processus), tandis que celles qui ne sont utiles que dans le cadre d'un serveur web, comme `ErrorDocument`, auront une directive équivalente à placer dans le contexte `http`.

Et d'autres, bien qu'on puisse penser qu'elles pourraient se placer dans le contexte `http` puisqu'étant dans le contexte de configuration globale d'Apache, ne pourront se placer que dans `server` (par exemple).

Contexte d'hôte virtuel

Le contexte d'hôte virtuel d'Apache est comparable au contexte `server` de Nginx (équivalent du conteneur `<Virtualhost>` d'Apache).

Contexte répertoire

Le contexte répertoire d'Apache n'a pas vraiment d'équivalent strict, tant sont diverses les combinaisons entre directives et contextes dans lesquels on peut les placer.

On pourra obtenir des équivalences sur certaines configurations, comme entre le conteneur Apache `<Location>` et le contexte Nginx `location {}`, alors que d'autres configurations seront tout simplement impossibles à transcrire de façon similaire pour les mêmes buts.

Contexte `.htaccess`

Le contexte `.htaccess` n'existe tout simplement pas avec Nginx. Ceci possède des avantages et des inconvénients :

- on ne peut laisser la possibilité à un utilisateur « lambda » de modifier le comportement du serveur dans les répertoires le concernant : on risque donc plus d'avoir des demandes d'utilisateur. De plus, de nombreux projets proposent des fichiers `.htaccess` pour adapter le serveur aux besoins particuliers du projet : il faudra traduire ces besoins en configuration Nginx (ce sont généralement des réécritures d'URL, un vrai bonheur à traduire).
- par contre, cela évite les erreurs de configuration des utilisateurs, sur lesquelles on s'arrache parfois les cheveux à cause d'un `.htaccess` mal configuré.

3.2 Équivalents Nginx des directives de configuration du contexte de configuration globale d'Apache

Certaines directives pourront prendre place dans le contexte Nginx `main`, d'autres dans `http` et d'autres encore dans d'autres contextes.

- `Servername` : `server_name` (contexte : `server`).
- `ServerRoot` : pas de directive équivalente. Correspond à l'option `--prefix` utilisée lors de la compilation.
- `Include` : `include` (contextes : tous)
- `IncludeOptional` : pas de directive équivalente.
- `DocumentRoot` : `root` (contextes : `http`, `server`, `location`, `if` in `location`).
- `ErrorDocument` : `error_page` (contextes : `http`, `server`, `location`, `if` in `location`), mais ne permet pas de spécifier de message. Uniquement une page ou une redirection.
- `User` et `Group` : `user` (contexte :`main`).
- `DirectoryIndex` : `index` (contextes : `http`, `server`, `location`).
- `Options` : pas d'équivalent, mais on trouve parfois des directives équivalentes pour les options.
  - `ExecCGI` : pas d'équivalent.
  - `FollowSymLinks` : `disable_symlinks` (contextes : `http`, `server`, `location`).
  - `Includes` : `ssi` (contextes : `http`, `server`, `location`, `if` in `location`).

    - `IncludesNOEXEC` : pas d'équivalent.
    - `Indexes` : `autoindex` (contextes : `http`, `server`, `location`).
    - `Multiviews` : pas d'équivalent.
    - `SymLinksIfOwnerMatch` : voir `disable_symlinks`.
    - `All` et `None` : pas d'équivalent.
- `IndexOptions` : pas d'équivalent.