11. SERVERUL WEB APACHE

11.1. Concepte web

HTTP (HyperText Transfer Protocol) este un protocol client-server de nivel aplicatie bazat pe cereri ale clientului si raspunsuri corespondente ale serverului. Traditional, HTTP foloseste pentru transportul de date protocolul TCP portul 80, sau, in cazul criptat (HTTPS), portul 443. Informatiile transferate prin HTTP pot fi in format text (text neformatat, HTML, XML) sau binar (ex: muzica, imagini, filme etc).

Serverul web este un soft ce "vorbeste" HTTP si al carui scop principal este sa publice continut – fie deja existent in sistemul sau de fisiere, fie generat pe loc ca urmare a unei cereri de client. In primul caz, continutul se gaseste in fisiere aflate dedesubtul unui director desemnat ca *radacina a paginilor web*. Pentru un server web configurat corect, clientii nu vor avea acces decat la fisierele publicate, nu la intregul sistem de fisiere al serverului; procedeul este asemanator cu file sharing-ul din Windows, unde publicam strict directoarele pe care dorim sa le partajam cu ceilalti din retea, nu intregul sistem de fisiere aflat pe statia in cauza.

Clientii HTTP sunt de obicei browserele. Ele specifica resursa dorita de la server sub forma unei adrese, denumita **URL** (Uniform Resource Locator). URL-ul este un concept larg, folosit pentru a identifica o resursa din internet si valabil pentru multe alte protocoale in afara de HTTP; vom trata insa aici numai URL-ul web.

URL-ul web este de forma *http://www.example.com:80/ex/index.html*, fiind compus din 3 parti:

- **http://** - specifica protocolul folosit. Aceasta deoarece softurile client au deseori capabilitatea de a folosi mai multe protocoale (HTTP, FTP etc) si trebuie sa stie pe care dintre acestea sa-l foloseasca pentru a comunica cu serverul
- **www.example.com:80** – desemneaza adresa serverului (fie sub forma ei numerica, fie sub forma unui nume DNS caruia trebuie sa ii corespunda in fisierul zona o inregistrare de tip A sau CNAME) si portul pe care acesta asculta
- **/ex/index.html** – reprezinta asa-numitul URL-path: calea catre resursa dorita, asa cum o vede clientul. Ceea ce clientul considera a fi / (radacina paginilor web) este de fapt continutul directorului desemnat ca atare in configurarea serverului

Nota: *URL-ul web poate contine si alte elemente, care nu sunt insa de interes in materialul de fata.*

Pentru fiecare cerere, serverul trebuie sa identifice pe baza *URL-path*-ului calea reala catre resursa solicitata. Implicit, operatia se realizeaza prefixand *URL-path*-ul cu calea catre directorul radacina a paginilor web, insa acest mod de lucru poate fi schimbat din fisierul de configurare al serverului. Un scurt exemplu: sa consideram serverul web accesibil cu numele *www.a.ro*. Daca configurarea serverului stabileste ca radacina a paginilor web directorul */var/www*, iar un client solicita resursa aflata la URL-ul *http://www.a.ro/b/c.html*, atunci serverul va prelua URL-path-ul */b/c.html*, va inlocui /-ul de la inceput cu calea reala catre radacina paginilor web (/var/www) si va obtine astfel calea reala catre resursa dorita: */var/www/b/c.html*. Punand problema invers: daca in */var/www* cream un director *poze* si in el *poza1.jpg*, aceasta va fi accesibila cu URL-ul *http://www.a.ro/poze/poza1.jpg*.

De remarcat ca acesta este modul implicit de procesare a unei cereri; serverul poate fi insa configurat sa procedeze in diverse alte moduri, prezentandu-i astfel clientului orice alta structura a resurselor detinute (webspace) care sa nu corespunda 100% celei reale de dedesubtul directorului desemnat ca radacina a paginilor web.

11.2. Serverul Apache - descriere, structura, facilitati

Apache este serverul web folosit la ora actuala in peste 60% din site-urile de pe mapamond, datorita faptului ca este un server matur, stabil, modular. Este derivat dintr-un proiect al NCSA (National Center for Supercomputing Applications din Illinois), care a fost oprit in 1994 din cauza plecarii dezvoltatorului principal (Rob McCool) si a fost preluat de catre comunitatea webmaster-ilor care in prima faza i-a adaugat numeroase patch-uri pentru a-i spori functionalitatea (de unde si numele ironic „a patchy server"). Intre timp a fost rescris si este astazi un server modular si actualizat permanent.

Serverul Apache2 este gandit de asa natura incat poate fi inclusa in executabilul serverului numai functionalitatea de baza, restul de facilitati fiind adaugabile ca module DSO (Dynamic Shared Object), care pot fi incarcate dinamic (fara a recompila serverul), facand mult mai usoara extinderea si configurarea serverului.

Printre facilitatile Apache se numara:

- **controlul accesului la resurse la nivel de fisier/director/URL-path** (atat pentru filesystem cat si pentru webspace)
  - in functie de adresa clientului sau/si serverului
  - in functie de numele domeniului din care face parte clientul
  - prin crearea de fisiere/directoare parolate
- **virtual hosting** – gazduirea mai multor site-uri pe aceeasi masina, in doua moduri:
  - IP-based – host-ul in cauza dispune de mai multe adrese IP, fiecare cu propriul nume DNS
  - name-based – masina are un singur IP, iar departajarea serverelor virtuale se face dupa numele serverului asa cum apare el in URL cerut de client
- **site-uri personale ale utilizatorilor definiti in sistem**. Facilitate utila in cazul ISP-urilor. Utilizatorii pot fi imputerniciti sa seteze unele optiuni ale site-ului personal printr-un fisier de configurare individual (eliminand necesitatea interventiei permanente a administratorului)
- **posibilitatea de generare de pagini dinamice**, prin folosirea altor programe:
  - prin intermediul CGI (Common Gateway Interface), serverul poate interactiona cu aplicatii scrise in orice limbaj de programare
  - exista interpretoare sub forma de modul DSO pentru majoritatea limbajelor de scripting importante (Perl, PHP etc)
- **posibilitatea de manevrare a webspace-ului** (ierarhia de resurse vazuta de catre client):
  - creare de alias-uri (fisiere/directoare virtuale) sau symlink-uri
  - rescrierea automatizata a URL-ului, pe baza de regex-uri, pentru redirectionarea cererilor de resurse
- **filtrare I/O** – trecerea datelor de I/O printr-un lant de module (prelucrare succesiva)
- **securizarea comunicatiei cu serverul prin intermediul modulului SSL** (Secure Sockets Layer) ce permite autentificarea si criptarea comunicatiei intre client si server pe baza de chei publice/private si certificate digitale
- **MPM (Multi Processing Modules)** – module DSO care dau posibilitatea serverului de a se adapta la caracteristicile de multi-procesare ale sistemului de operare pe care ruleaza (pana la versiunea 2, apache utiliza thread-urile POSIX, ceea ce il facea sa ruleze ineficient pe Windows sau Netware)

11.3. Fisa serverului

Iata sumarul informatiilor legate de server:

- executabil: **httpd** (in unele distributii **apache, apache2** sau **httpd2**)
- fisier de configurare: **httpd.conf**, sau uneori **apache.conf** sau **apache2.conf** (in functie de distributie). In restul materialului ne vom referi la el ca fiind **httpd.conf**. Locatia sa poate fi */etc/apache*, */etc/apache2*, */etc/httpd*, */etc/httpd/conf* etc. (in functie de distributie)
- utilitar de control: **apachectl** (in unele distributii **apache2ctl**), care poate efectua urmatoarele operatii:

- validarea fisierului de configurare: **apachectl configtest**
- pornirea serverului: **apachectl start**
- oprirea serverului
  - **apachectl stop** – opreste serverul, inchizand toate conexiunile curente cu clientii
  - **apachectl graceful-stop** – opreste serverul, lasand insa conexiunile curente deschise (clientii curent conectati vor continua sa fie serviti)
- repornirea serverului
  - **apachectl restart** – restarteaza serverul, inchizand toate conexiunile curente
  - **apachectl graceful** – restarteaza serverul, insa fara a inchide conexiunile curente ale serverului cu clientii sai
- afisarea starii serverului: **apachectl status** sau **apachectl fullstatus**

- rulare server in foreground, cu output detaliat pe ecran: **httpd -X -f /cale/catre/fisier/configurare -c 'LogLevel debug' -c 'ErrorLog "|/bin/cat"'**

11.4. Configurare de baza

11.4.1. Compozitia si sintaxa generala a fisierului de configurare

Fisierul de configurare *httpd.conf* reprezinta un ansamblu de directive de configurare. Fiecare directiva apartine de un anume modul, si poate fi folosita numai in masura in care acel modul este incarcat.

Directivele pot fi de doua tipuri:

- **directive simple** – directive de o linie, care pot fi plasate independent sau incluse in directive bloc
- **directive bloc** (numite si *container;* ex: *Directory*, *Files*, *Location*), care pot ingloba directive simple in scopul restrangerii efectului lor la un anume context. Ele se prezinta sub forma a doi delimitatori (de inceput si de sfarsit) care amintesc de tag-urile HTML: <directiva>....</directiva>. Spre exemplu, urmatoarea directiva bloc realizeaza activarea unei optiuni numai pentru fisierele livrate din directorul */var/www* din sistemul de fisiere al serverului:

```
<Directory /var/www/>
      Options +Indexes
</Directory>
```

In documentatia oficiala Apache, pentru fiecare directiva se specifica setul de contexte in care poate fi folosita. Desi lista posibila este mult mai bogata, mentionam aici doua dintre ele:

- *contextul global* – reprezinta "radacina" fisierului de configurare (in afara oricaror directive bloc). Astfel de directive au efect asupra tuturor resurselor solicitate serverului
- *contextul unei directive bloc* – dupa cum s-a vazut, directivele bloc restrang efectul directivelor de dinauntrul lor la un grup de resurse specificat de administrator (efectul nu mai este global)

11.4.2. Incarcarea dinamica a modulelor

Serverul Apache este unul modular; exista o functionalitate de baza, imbogatita prin prezenta unei multitudini de module posibile. Modularitatea in cauza se manifesta inca de la compilare: putem alege, pentru fiecare modul in parte, una din trei variante:

- sa nu fie inclus deloc in versiunea finala
- sa fie inclus in executabil (httpd), ceea ce inseamna ca nu-l vom mai putea dezactiva
- sa fie inclus ca modul separat de executabil. Aceasta solutie are avantajul ca administratorul de server poate activa sau dezactiva modulele folosind fisierul de configurare, in functie de necesitati

In general serverele pe care le instalam din pachete au optat pentru cea de-a treia varianta, deoarece este cea mai flexibila: executabilul httpd este “suplu”, continand doar functionalitatea de baza, iar majoritatea facilitatilor de server se regaseste in module aditionale.

Modulele Apache poarta in general denumirea **mod_*numeModul*** (ex: mod_autoindex, mod_rewrite etc) si iau forma unor fisiere cu numele *mod_numeModul.so*. Pentru a putea fi incarcat, un modul trebuie sa indeplineasca doua conditii:

- sa fi fost compilat ca shared (sa se regaseasca sub forma de fisier *mod_NumeModul.so* in sistemul de fisiere)
- sa NU fi fost compilat in executabilul *httpd*. Lista de module prezente in executabil se obtine cu comanda **httpd –l**.

Incarcarea unui modul se realizeaza cu directiva **LoadModule** ce are sintaxa urmatoare:

```
LoadModule          module_identifier          /cale/catre/mod_NumeModul.so
```

unde *module_identifier* poate fi gasit in documentatia Apache a fiecarui modul (prezenta pe http://httpd.apache.org/docs/current/mod/) , iar fisierul .so se gaseste in sistemul de fisiere, locatia sa diferind in functie de distributie si de natura instalarii (surse vs pachete precompilate).

Nota: *locatia modulelor de Apache poate fi determinata afland lista de pachete ce compun serverul si efectuand o listare a fisierelor pe care aceste pachete le-au instalat. Cautam fisierele de forma mod_nume.so.*

Fiecare modul inseamna functionalitate suplimentara adaugata serverului. Este firesc ca aceasta functionalitate sa fie configurabila, de aceea fiecare modul introduce si una sau mai multe directive de configurare. Din acest motiv, multe directive nu pot fi folosite decat daca modulul corespunzator a fost incarcat. Directivele apartinatoare de un modul pot fi evaluate conditionat folosind container-ul *IfModule*:

```
<IfModule modul.c>
directive specifice
</IfModule>
```

Exemplu:

```
<IfModule mod_authz_host.c>
      Require ip 10.0.0.100
</IfModule>
```

Atunci cand incercam sa folosim o directiva fara sa fi incarcat modulul care o introduce, vom fi atentionati; spre exemplu: *“Invalid command 'AuthType', perhaps misspelled or defined by a module not included in the server configuration”*.

Nota: *lista de module incarcate din fisierul de configurare poate fi obtinuta cu* httpd -M.

11.4.3. Module obligatorii

11.4.3.1. MPM-ul utilizat

Un MPM (Multi-Processing Module) reprezinta o anumita strategie a serverului Apache de a se diviza in subprocese si fire de executie. Exista urmatoarele strategii mai folosite:

- **prefork** – un MPM care creeaza cate un subproces pentru fiecare client servit

- **worker** – un MPM care utilizeaza mai multe procese, care la randul lor se pot ramifica in mai multe fire de executie
- **event** – o varianta optimizata a lui worker, care creeaza fire de executie numai pentru conexiunile active

In Apache 2.2, MPM-ul era stabilit la compilare, executabilul generat avand “incastrat” un anumit MPM, care nu mai putea fi apoi schimbat decat prin recompilare. Incepand cu Apache 2.4, MPM-urile devin module obisnuite de Apache, care pot fi activate sau dezactivate din fisierul de configurare, ceea ce inseamna o mult mai mare flexibilitate; reversul medaliei este ca administratorul serverului trebuie sa se asigure ca a fost incarcat un modul MPM – si numai unul! (nu este permisa activarea mai multor module MPM simultan):

```
# incarcare modul prefork
LoadModule mpm_prefork_module /usr/lib/apache2/modules/mod_mpm_prefork.so
```

11.4.3.2. Securizare minimala: mod_unixd

Un server reprezinta, in fond, o aplicatie, si in consecinta se executa in sistemul de operare sub forma unui ansamblu de procese, fiecare ruland cu un anumit UID si GID. In general este bine sa se evite ca UID-ul sa fie *root* deoarece, in cazul in care un atacator reuseste sa “sparga” serverul (adica sa preia controlul asupra unuia sau mai multora dintre procesele sale), el va avea drepturi depline in sistem. Pentru a evita astfel de situatii, serverele implementeaza un mecanism prin care administratorul poate configura UID/GID al proceselor serverului; chiar daca serverul este pornit ca root (lucru necesar pentru a putea deschide porturile sub 1024!) el renunta la privilegiile maximale si le preia pe cele ale userului specificat – iar acest lucru se intampla inainte ca serverul sa inceapa sa comunice cu clientii.

Serverul Apache implementeaza si el aceasta strategie cu ajutorul modulului **mod_unixd**, ce introduce, printre altele, directivele esentiale *User* si *Group*:

```
LoadModule unixd_module /usr/lib/apache2/modules/mod_unixd.so
User wwwu
Group wwwg
```

Userul si grupul specificate trebuie sa existe in sistem in momentul pornirii serverului. Ele pot fi create ca orice alt user/grup (spre exemplu, folosind comenzile *groupadd* si *useradd*).

Atentie! *Nu in toate distributiile mod_unixd este compilat ca modul; unele dintre ele includ mod_unixd direct in executabilul serverului. Putem determina in care dintre cazuri ne aflam ruland httpd -l; daca mod_unixd se regaseste in lista afisata, atunci el a fost inclus in executabil si nu mai este necesara/posibila incarcarea separata a modulului.*

In cazul utilizarii strategiei MPM *prefork* vom constata ca, dintre toate procesele pornite de server, unul ruleaza ca root, deschizand portul 80 si jucand rolul de dispecer al conexiunilor, iar celelalte ruleaza cu userul/grupul specificate; aceste procese neprivilegiate sunt de fapt cele care deservesc clientii. In acest fel un eventual cracker care ar prelua controlul unui astfel de proces ar avea posibilitati mult reduse.

11.4.3.3. Infrastructura de autorizare a cererilor

Dupa cum se va explica ulterior, serverul trebuie sa decida, la nivelul fiecarei cereri receptionate, daca o onoreaza sau nu. Exista o intreaga infrastructura de module si directive care ii permit administratorului sa autorizeze cererile in functie de o multitudine de parametri ale clientului/cererii; la baza acestei infrastructuri se afla **mod_authz_core** – modulul ce implementeaza functiile fundamentale de autorizare:

```
LoadModule authz_core_module /usr/lib/apache2/modules/mod_authz_core.so
```

Pe moment facem doar mentionarea modulului – dat fiind faptul ca este necesar pentru functionarea serverului – urmand ca intregul mecanism de control al accesului sa fie detaliat intr-una dintre sectiunile ulterioare.

11.4.4. Adrese si porturi pe care asculta serverul

Dintre elementele fundamentale ale configurarii Apache, fara de care serverul nu va porni, face parte si setul de adrese si porturi pe care asculta serverul; configurarea sa se realizeaza cu ajutorul directivei **Listen**. Alte servere asculta automat pe toate adresele definite in sistem la pornire, fara interventie din partea administratorului; spre deosebire de acestea, in Apache este necesara specificarea a cel putin o directiva *Listen*, in caz contrar serverul nedeschizand niciun port!

```
# adresele/porturile pe care asculta serverul. Directiva poate avea diverse forme:
Listen 80                # serverul asculta pe toate adresele definite in sistem, portul 80
Listen *:8080            # serverul asculta pe toate adresele definite in sistem, portul 8080
Listen 10.0.0.100:8000   # serverul asculta doar pe adresa 10.0.0.100 portul 8000
# Atentie! Adresa trebuie sa fie una dintre cele ale interfetelor serverului!
```

Pot fi folosite simultan mai multe directive *Listen*. Spre exemplu, pentru un server web care ofera atat serviciul HTTP cat si HTTPS, vom dori sa deschidem porturile 80 si 443 folosind doua directive *Listen* diferite.

11.4.5. Radacina paginilor web

Radacina paginilor web (asa-numitul "**document root**") este un alt element de configurare obligatoriu si reprezinta directorul ce contine resursele publicate de server; la o prima aproximare, putem considera ca este locul in care trebuie plasate fisierele site-ului (definitia va rafinata ulterior). In cazul in care serverul gazduieste mai multe site-uri, fiecare va avea propriul document root, dupa cum se va vedea mai tarziu.

```
# directorul ce constituie radacina paginilor web
DocumentRoot /var/www
```

Document root-ul este folosit ca prefix la fiecare cerere de client. Spre exemplu, daca utilizatorul scrie in browser *http://www.exemplu.ro/produse/lista.html*, serverului i se solicita resursa */produse/lista/html;* cunoscand faptul ca ceea ce clientul numeste "/" reprezinta de fapt document root-ul, serverul prefixeaza calea primita in cerere cu cea catre document root pentru a forma calea completa, reala, catre resursa solicitata din sistemul sau de fisiere – in cazul nostru */var/www/produse/lista.html*.

11.4.6. Logul de erori

Principala modalitate de diagnosticare a serverului o reprezinta analiza logurilor pe care acesta le genereaza. Exista multiple tipuri de loguri pe care Apache le poate mentine, insa cel mai important dintre ele in etapa de invatare/diagnosticare de server este logul de erori. Locatia acestuia poate fi stabilita cu ajutorul directivei **ErrorLog**, iar gradul de detaliu al informatiilor consemnate se configureaza cu ajutorul directivei **LogLevel**. Aceasta din urma primeste ca parametru una dintre valorile *emerg* (cel mai scazut grad de detaliu), *alert, crit, error, warn, notice, info, debug*, *trace1...trace8* (trace8 fiind cel mai detaliat).

```
ErrorLog /var/log/apache.err
LogLevel debug
```

11.4.7. Procesarea cererilor adresate directoarelor

Indexul unui director este ceea ce serverul livreaza clientului ca raspuns la un URL caruia ii corespunde pe server un director. Indexul poate proveni din:

- unul dintre fisierele specificate cu directiva *DirectoryIndex*, daca **mod_dir** este incarcat. Fisierele vor fi cautate in ordinea in care apar in directiva. Nume tipice: *index.htm*, *index.html*, *index.php* etc., desi nu este obligatoriu ca fisierul sa poarte numele *index*
- generat automat (dinamic), daca **mod_autoindex** este incarcat; pentru activare intr-un director, este necesar ca in containerul <*Directory*> corespunzator sa fie specificat *Options +Indexes*.

In cazul in care **mod_dir** si **mod_autoindex** actioneaza simultan, in director va fi cautat mai intai unul dintre fisierele index si abia apoi se incearca generarea automata.

Nota: *atunci cand userul scrie in browser simplul nume al serverului, fara URL path, serverului i se va solicita / , adica un director! De aceea serverul web trebuie sa aiba DirectoryIndex configurat corect.*

Sa consideram urmatorul exemplu: in */var/www* se gaseste fisierul *unu.html* si directorul *doi* ce contine fisierul *trei.html*. Daca directiva *DirectoryIndex* are valoarea *unu.html*, iar numele serverului este www.exemplu.ro, atunci:

- scriind in browser http://www.exemplu.ro (fara URL-path) obtinem automat continutul fisierului *unu.html*, insa neformatat – simplul cod sursa HTML
- scriind in browser http://www.exemplu.ro/doi obtinem un listing autogenerat al directorului *doi*, deoarece serverul nu gaseste in interiorul lui un fisier numit *unu.html*

Modulul **mod_dir** adauga o functie suplimentara serverului: URL-urile care refera directoare vor primi automat un sufix ”/”, fara de care cererea clientului ar returna o eroare. Serverul trimite o redirectionare catre client, in care include noul URL (ce contine si sufixul). Atentie insa! URL-ul este construit folosind numele serverului declarat in directiva *ServerName*. *Daca aceasta nu apare in httpd.conf, va fi folosit automat numele statiei sau 127.0.0.1!*

11.4.8. Corecta determinare si transmitere a tipului de continut

Continutul primit de catre browser de la serverul web ca urmare a unei cereri consta intr-o insiruire de octeti. Browserul nu are de unde sa banuiasca cum trebuie tratati acei octeti – ca HTML? PDF? MP3? - si de aceea este datoria serverului sa determine tipul de continut si sa i-l comunice clientului. Tipul de continut (asa-numitul ***tip MIME***) ii este indicat clientului prin intermediul headerului HTTP *Content-type;* in functie de valoarea acestuia, browserul poate deschide fisierul, poate folosi plug-in-uri sau poate apela o alta aplicatie.

Atentie! *In majoritatea cazurilor, browserul asculta "orbeste" de server! Daca serverul este prost configurat si trimite un fisier HTML, indicand insa un continut de tip text, browserul se va conforma si va afisa sursa HTML, fara a o formata!*

Pentru ca serverul Apache sa poata determina corect natura informatiei livrate clientului este necesara incarcarea modulului **mod_mime**, care “invata” serverul sa determine tipul de continut pe baza extensiei de fisier. Determinarea se realizeaza cu ajutorul unui fisier de corespondente extensie ↔ tip, numit implicit *mime.types*. Dupa ce serverul primeste o cerere de la client si determina calea absoluta catre fisierul ce trebuie livrat acestuia, el cauta extensia fisierului in *mime.types* si extrage tipul MIME, trimitandu-i apoi clientului continutul solicitat insotit de headerul *Content-type* ce-i indica tipul. In acest fel clientul stie cum sa trateze continutul receptionat.

Serverul Apache include un fisier *mime.types* prepopulat, aflat in subdirectorul *conf* al instalarii de Apache (la instalarea din surse) sau in interiorul directorului */etc* la instalarea din pachete (*/etc*, */etc/apache2* etc. in functie de distributie). Continutul acestuia poate fi customizat de catre administrator dupa necesitati, iar locatia sa se poate configura cu ajutorul directivei *TypesConfig*:

```
TypesConfig /etc/apache2/mime.types
```

11.5. Particularizarea configurarii per grup de resurse

In Apache, efectul multora dintre directive se poate aplica global sau per grup de resurse. Resursele pot fi grupuri de fisiere sau anumite parti ale webspace-ului. Pentru specificarea grupurilor de resurse carora li se aplica setari particularizate se pot folosi urmatoarele directive bloc:

- **<Directory>** - efectueaza particularizarea setarilor la nivel de director din sistemul de fisiere al serverului:

  ```
  <Directory /cale/in/sistemul/de/fisiere/al/serverului>
        ...restrictii...
  </Directory>
  ```

  Restrictiile specificate se vor aplica automat in toata adancimea directorului precizat. Exceptie face cazul in care exista o alta directiva *<Directory>* care i se aplica unuia dintre subdirectoare, si care astfel va suprascrie setarile facute in directiva *<Directory>* corespunzatoare directorului parinte

  Cand folosim *<Directory>*, restrictiile specificate in interior li se aplica resurselor livrate din directorul specificat indiferent de URL-ul folosit pentru a se ajunge la ele. Altfel spus, conteaza locatia finala, nu modul de accesare.

- **<Location>** - particularizeaza setarile la nivel de “director web”, adica pentru toate URL-urile al caror URL-path incepe cu un anumit prefix:

  ```
  <Location /URL-path>
        ...restrictii...
  </Location>
  ```

  De data aceasta conteaza modul de accesare a resurselor, indiferent unde se afla ele in sistemul de fisiere. Exemplu: dorim ca *www.a.ro/admin* sa fie accesibil numai cu parola; nu ne intereseaza unde anume sunt stocate fisierele corespunzatoare pe hard-disk.

- **<Files>** sau **<FilesMatch>** - permit particularizarea setarilor la nivel de nume de fisier. Ele primesc ca parametru numele de fisiere ce fac subiectul restrictionarii, diferenta fiind ca *<FilesMatch>* lucreaza implicit cu regular expressions.

```
# restrictionarea accesului la fisierele din /stats (accesibile clientului cu
# www.server.com/stats/fisier), indiferent unde se afla ele in sistemul de fisiere al serverului
<Location /stats>
      Require ip 10.0.0.0/24
</Location>

# restrictionarea accesului la fisierele .jpg
<Files ~ \.jpg$>    # ceea ce urmeaza dupa ~ este un regex ce descrie numele de fisier (nu calea!)
      Require ip 10.0.0.0/24
</Files>
```

Aceste directive au aplicabilitate generala, insa in acest material vor fi folosite in special pentru controlul accesului la resurse.

11.6. Controlul accesului la resurse

11.6.1. Principii. Modalitati de control al accesului

Nota: *materialul de fata prezinta modul de control al accesului incepand cu Apache 2.4; pentru versiunea 2.2 consultati ANEXA 1 a materialului. Din motive de compatibilitate/usurare a migrarii, vechile directive de control al accesului (Allow, Deny si Order) pot fi inca utilizate in Apache 2.4, cu conditia incarcarii modulului* ***mod_access_compat*** *in care au fost mutate.*

Pentru fiecare cerere formulata de un client, serverul trebuie sa decida daca ii permite sau ii interzice clientului accesul la resursa dorita; actiunea poarta numele de **autorizare** a cererii**.** Pentru a lua decizia, serverul se foloseste de informatii primite de la client, care pot fi impartite in urmatoarele categorii:

- **caracteristici directe ale clientului** – adresa sau portul de pe care se conecteaza, browser folosit, tipul de cerere HTTP etc. Aceasta nu implica o actiune speciala din partea clientului, ci doar o “filtrare” a clientilor de catre server in functie de informatiile disponibile lui inca din momentul conectarii clientului la server. Observam ca in acest mod nu putem filtra utilizatori, ci doar statii, grupuri de statii, browsere etc (sa ne amintim ca in spatele unei singure adrese IP se poate afla o intreaga retea!)
- **identitatea utilizatorului** – daca dorim sa autorizam utilizatori, atunci trebuie sa-i obligam sa-si arate (si dovedeasca!) identitatea; actiunea poarta numele de **autentificare** si presupune operatii suplimentare din partea clientului. Aceasta se poate realiza in diferite moduri – varianta uzuala fiind combinatia user/parola, dar existand diverse altele (ex: certificate digitale). In acest fel putem crea, de exemplu, zone restrictionate ale unui site, in care accesul este permis numai unui grup controlat de utilizatori pe baza de user/parola.

Cele enumerate mai sus (IP, port, browser, username etc) reprezinta criterii elementare de autorizare – caracteristici ale cererii pe care serverul le poate lua in considerare pentru stabilirea deciziei; astfel de criterii poarta denumirea de *authorization providers* si provin din module Apache. Serverul poate decide sa autorizeze sau nu o cerere pe baza unui singur provider sau a unei unei combinatii de astfel de provideri (ex: cererea va fi servita doar daca clientul are adresa din subnet-ul 10.0.0.0/24, browserul este Firefox si cererea HTTP este de tip GET). Lista de module disponibile va fi prezentata mai jos.

Practic, la receptionarea unei cereri, serverul isi pune doua intrebari:

- voi livra resursa ceruta? Raspunsul la aceasta intrebare este unul binar (da sau nu) iar procesul decizional reprezinta insasi autorizarea
- ce informatie despre client folosesc pentru a putea raspunde la intrebarea anterioara? Raspunsul la aceasta intrebare stabileste authorization providerul: daca dorim sa filtram clientii in functie de IP vom folosi un modul, daca vrem sa-i filtram in functie de numele DNS alt modul etc.

11.6.2. Autorizare

11.6.2.1. Directive si module de autorizare

Directiva principala folosita pentru autorizare este **Require** (care inlocuieste vechile *Allow* si *Deny* din Apache 2.2), iar sintaxa ei este urmatoarea:

```
Require nume_provider parametri
# sau, cu negare:
Require not nume_provider parametri
```

Atentie! Directiva nu poate fi folosita in contextul global, ci doar in directive bloc, deoarece este necesara specificarea setului de resurse la care ea restrictioneaza accesul!

Primul parametru al directivei *Require* reprezinta authorization providerul - criteriul in functie de care se ia decizia de autorizare. Restul de parametri sunt datele suplimentare necesare acelui criteriu (adrese IP sau subnet-uri pentru providerul *ip*, nume de statii pentru providerul *host* etc).

Fiecare provider este valabil numai daca a fost incarcat modulul Apache corespunzator. Iata in continuare principalele module Apache responsabile cu autorizarea si criteriile introduse de acestea:

- **mod_authz_core** – un modul obligatoriu pentru functionarea autorizarii (este cel care introduce directiva *Require*!). Acest modul ofera urmatoarele criterii de autorizare:
  - ***all*** – criteriu generic care se refera la toti clientii, folosit pentru a scrie reguli de autorizare globala, echivalente cu vechile *Allow from all*/*Deny from all*
  - ***method*** – permite autorizarea in functie de tipul de cerere HTTP (GET, POST etc)
  - ***env*** – permite autorizarea in functie de prezenta unei variabile Apache (detalii in cursul Linux AS&S)
- **mod_authz_host** – modul ce implementeaza autorizarea in functie de IP sau nume DNS al clientului. Criterii introduse:
  - ***ip*** – permite autorizarea in functie de adresa IP sursa a clientului sau de subnet-ul din care aceasta face parte (vezi exemplele de mai jos)
  - ***host*** – autorizare in functie de numele DNS al clientului, obtinut prin rezolutie DNS inversa
  - ***local*** – un shortcut care desemneaza clientii aflati pe aceeasi masina cu serverul (cei a caror adresa IP este identica cu una dintre adresele serverului, incluzand aici adresa de loopback)
- **mod_authz_user** – efectueaza autorizarea in functie de identitatea clientului, functionand la fel ca in Apache 2.2. Modulul introduce doua valori posibile pentru primul parametru al directivei *Require*:
  - ***valid-user*** – este permis accesul oricarui user autentificat corect, conform mecanismului de autentificare definit in acel context (vezi mai jos sectiunea dedicata autentificarii)
  - ***user*** - permite specificarea unei liste discrete de useri autentificati care au acces la resursele din acel context
- **mod_authz_groupfile** – permite definirea de grupuri de utilizatori si gestionarea accesului pe baza apartenentei la grup, nu a username-ului individual. Grupurile sunt definite in fisiere text. Singurul authorization provider introdus este ***group***
- **mod_authz_owner** – permite accesul la resurse pe baza ownership-ului la nivel de sistem de fisiere al resursei solicitate. Criterii introduse:
  - ***file-owner*** – permite accesul doar daca userul (deja autentificat de catre serverul web!) este ownerul fisierului pe care il acceseaza
  - ***file-group*** – analog, pentru grupul proprietar
- **mod_authz_dbd** – permite filtrarea accesului pe baza de useri/grupuri memorate in baze de date SQL

Exemple:

```
#   autorizare in functie de IP exact client (10.0.0.2 sau 10.0.0.5)
Require ip 10.0.0.2 10.0.0.5
#   autorizare in functie de subnet-ul din care face parte IP-ul sursa al clientului
Require ip 10.0.0.2/27
#   autorizare in functie de numele DNS al clientului, obtinut prin rezolutie DNS inversa
Require host c1.infoacademy.net a.b.ro
#   autorizare in functie de tipul de cerere HTTP (asa-numitul “request method”)
Require method GET POST
```

```
# acces permis pentru userii user1 si user2 (autentificati de catre serverul web)
Require user user1 user2
# acces permis daca userul autentificat de catre serverul web este owner pe resursa ceruta
Require file-owner
# acces permis pentru clientii aflati pe aceeasi masina cu serverul
Require local
```

Atentie! Serverul nu functioneaza corect daca nu are incarcat cel putin mod_authz_core! *Eroarea emisa va fi de forma:* AH00025: configuration error: couldn't check user: /

11.6.2.2. Combinarea mai multor criterii de autorizare

Dupa cum s-a spus mai sus, directivele de autorizare trebuie plasate in contexte mai restranse decat cel global, cum ar fi <Directory>, <Location> etc. Un astfel de context poate contine mai multe directive *Require*; vom explica in continuare cum se configureaza efectul combinat al unui astfel de grup de directive.

Pentru fiecare cerere primita sunt evaluate directivele *Require* ale contextului din care este servita acea cerere. Fiecare directiva *Require* poate produce unul din trei rezultate:

- *admis* – directiva *Require* in cauza autorizeaza clientul sa primeasca resursa solicitata
- *respins* – directiva *Require* ii interzice clientului sa primeasca rezultatul cererii sale
- *neutru* – directiva *Require* nu autorizeaza clientul dar nici nu ii interzice accesul

Pentru ca o cerere sa fie autorizata, este necesar ca rezultatul ansamblului de directive *Require* ale contextului din care este servita sa fie *admis* – ori asta presupune ca cel putin una dintre directive sa produca rezultatul *admis*.

Sa consideram fragmentul de configurare de mai jos:

```
<Directory /var/www/test>
      Require ip 10.0.0.0/24
      Require not ip 10.0.0.100
</Directory>
```

Pentru clientul cu adresa 10.0.0.22, prima directiva *Require* produce rezultatul *admis* iar cea de-a doua rezultat neutru. In consecinta, clientul va avea accesul permis. (vezi mai jos detalii despre combinarea directivelor *Require*)

Atentie! *O directiva Require cu negare nu poate produce niciodata rezultatul* admis *si de aceea nu poate autoriza de una singura o cerere! Iata un exemplu:*

```
# dorim sa autorizam toti clientii in afara de 10.0.0.100

# abordare gresita: clientii cu adrese diferite de 10.0.0.100 NU vor avea accesul permis!
<Directory /var/www/test>
      Require not ip 10.0.0.100
</Directory>
```

Putem configura felul in care serverul compune efectul mai multor directive *Require* plasate in acelasi context. In acest scop au fost introduse in Apache 2.4 trei directive bloc: **<RequireAll>**, **<RequireAny>** si **<RequireNone>**. Acestea sunt folosite pentru a grupa directive *Require* multiple si a produce un rezultat combinat; fiecare dintre ele poate produce aceleasi rezultate ca si o directiva Require – admis, respins sau neutru, in functie de caz. O cerere este servita dintr-un astfel de context complex numai daca rezultatul autorizarii este *admis*, dupa cum urmeaza:

- **<RequireAll>** - rezultatul de ansamblu al blocului este *admis* numai daca NICIO directiva *Require* componenta nu produce *respins* si CEL PUTIN UNA produce *admis;* daca vreuna dintre directive produce *respins*, rezultatul blocului este de asemenea *respins*. Daca toate directivele din bloc produc rezultat neutru, rezultatul final al lui *<RequireAll>* este de asemenea neutru, ceea ce este insuficient pentru a autoriza cererea.
- **<RequireAny>** - daca CEL PUTIN UNA dintre directivele *Require* continute produce *admis*, rezultatul de ansamblu al blocului este *admis*. Daca toate directivele produc rezultat neutru, rezultatul blocului *<RequireAny>* va fi la randul sau unul neutru. In celelalte cazuri rezultatul este *respins*. ***Atentie!*** *Intr-un astfel de bloc nu sunt permise directive Require negate, deoarece ele nu pot produce* admis *si deci nu pot influenta rezultatul blocului*
- **<RequireNone>** - NICIUNA dintre directivele *Require* continute nu trebuie sa produca *admis*. Daca vreuna produce *admis*, rezultatul blocului este *respins*. Nici in acest bloc nu sunt permise directive Require negate, din acelasi motiv ca in cazul lui <RequireAny>. ***Atentie!*** *Rezultatul blocului nu poate fi niciodata* admis, *ci doar* respins *sau* neutru, *in consecinta un astfel de bloc neputand fi folosit de unul singur pentru a acorda acces la o resursa!*

Constatari:

- nu toate blocurile pot produce toate tipurile de rezultate
- directivele *Require* negate nu pot fi folosite in *<RequireNone>* sau *<RequireAny>* deoarece ele nu pot produce *admis* si in consecinta nu pot influenta rezultatul ansamblului. Singurul context permis pentru ele este asadar *<RequireAll>*, unde eventualul lor esec ar produce rezultatul *respins* pentru intregul ansamblu.
- blocul *<RequireNone>* nu poate produce admis si deci nu poate autoriza de unul singur o cerere; in consecinta, i se aplica aceleasi restrictii ca directivelor *Require* negate
- *<RequireAll>* este echivalentul unui SI logic intre conditiile componente, pe cand *<RequireAny>* este echivalentul unui SAU. *<RequireNone>* are efectul unui SI logic intre negarile conditiilor componente.

Atentie! Atunci cand un grup de directive Require nu sunt incluse explicit intr-un bloc dintre cele enumerate mai sus, ele sunt considerate a face parte automat dintr-un bloc RequireAny!

```
# va fi servit din directorul /var/www/test orice client in afara de cel cu adresa 10.0.0.2
<Directory /var/www/test1>
      <RequireAll>
            Require all granted
            Require not ip 10.0.0.2
      </RequireAll>
</Directory>

# vor fi serviti din acest director numai “posesorii” celor doua adrese IP specificate;
# automat cele doua directive Require sunt tratate ca si cum ar fi cuprinse intr-un RequireAny!
<Directory /var/www/test2>
      Require ip 10.0.0.100
      Require ip 192.168.1.100
</Directory>
# exemplu complex: vor fi livrate resurse din directorul /var/www/test numai pentru clientii care
#  se autentifica cu un username ce face parte din grupul crowd dar nu si din grupul weird si care
#  nu se conecteaza de pe 10.0.0.100, si in plus respecta una dintre conditiile urmatoare: fie fac
#  parte din grupul gspecial, fie se conecteaza din reteaua 10.0.0.0/24, fie nu sunt autentificati
#  cu userul keepout si in acelasi timp cererea este de tip GET

<Directory /var/www/test>
      <RequireAll>
            Require group crowd
            <RequireAny>
                  Require group gspecial
                  Require ip 10.0.0.0/24
                  <RequireAll>
                        Require not user keepout
                        Require method GET
```

```
                </RequireAll>
            </RequireAny>
            <RequireNone>
                Require group weird
                Require ip 10.0.0.100
            </RequireNone>
      </RequireAll>
</Directory>
```

11.6.3. Autentificare. Autorizare pe baza identitatii clientului

11.6.3.1. Principii. Mecanisme de autentificare

Daca dorim sa autorizam nu statia sau browserul client, ci utilizatorul din spatele lor, este necesar sa-i solicitam acestuia sa se autentifice – adica sa-si arate si dovedeasca identitatea. Acest lucru poate fi realizat in diferite moduri; in fond, si in viata reala dovedirea identitatii se poate realiza printr-o multitudine de procedee: prezentarea unui act de identitate, verificarea amprentei, scanare retinala/de iris etc. Unele dintre aceste procedee presupun transmiterea ca atare a informatiei (numele figureaza in clar pe buletin), altele dovedesc identitatea fara ca ea sa fie rostita/aratata. In acelasi fel, autentificarea din domeniul digital se poate realiza in diferite moduri, numite **mecanisme de autentificare**.

Un astfel de mecanism reprezinta modalitatea efectiva in care serverul si clientul schimba informatie pentru ca unul sa-si dovedeasca identitatea fata de celalalt. Identitatea poate fi prezentata si dovedita sub forma combinatiei user/parola, a unui certificat digital, a unei combinatii user+parola criptata etc. Dintre toate mecanismele disponibile, serverul Apache suporta din oficiu doua, fiecare dintre ele fiind implementat cu ajutorul cate unui modul:

- mecanismul **basic** – acesta presupune transmiterea in clar a unei combinatii user/parola de la client catre server. Este considerat un mecanism nesigur deoarece exista riscul interceptarii informatiei in tranzit; este de preferat ca el sa fie folosit numai in cazul conexiunilor criptate (HTTPS). Modulul aferent este *mod_auth_basic*
- mecanismul **digest** – un mecanism gandit la origini sa fie mai sigur decat cel basic, deoarece clientul nu transmite parola in clar, ci transformata cu ajutorul unei functii de hashing. In ziua de astazi algoritmul este considerat depasit, avand breşe importante de securitate, si in locul sau sunt preferate conexiunile criptate HTTPS. Modulul aferent este *mod_auth_digest*

Oricare ar fi mecanismul ales, serverul trebuie sa dispuna de niste baze de date cu identitati/caracteristici de clienti care sa-i permita verificarea informatiilor transmise de catre client. Aceste baze de date poarta denumirea de *back-end*-uri de autentificare sau – in terminologia Apache – **authentication providers**. Fiecare mecanism poate avea provideri multipli; spre exemplu, putem configura serverul sa utilizeze mecanismul basic, preluand listele de useri atat din baza de date de sistem (passwd/shadow) cat si dintr-o tabela SQL. Toate acestea au in spate o intreaga infrastructura de module Apache, ce va fi prezentata mai jos.

11.6.3.2. Structura de module si directive

Modulele implicate in procesul de autentificare se impart in urmatoarele mari categorii:

- module ce asigura infrastructura de autentificare
  - **mod_authn_core** – asigura elementele comune tuturor mecanismelor si providerilor. Directive introduse:
    - ***AuthName*** *nume* – folosita pentru a specifica denumirea data zonei parolate. Informatia ii este vizibila clientului in fereastra de autentificare deschisa de browser si, in plus, este utilizata intern de catre browser, care efectueaza caching-ul informatiilor de autentificare per zona parolata
    - ***AuthType*** *mecanism* – configureaza mecanismul de autentificare folosit (basic/digest)

- mecanisme de autentificare
  - **mod_auth_basic** – implementeaza mecanismul HTTP Basic Authentication. Directive relevante:
    - ***AuthBasicProvider*** *prov1 prov2 …* – stabileste providerul/lista de provideri asociati mecanismului basic. Pentru fiecare provider trebuie sa fie incarcat modulul corespunzator! Valoarea implicita este *file* (autentificare din fisiere locale in format text)
  - **mod_auth_digest** – analog, pentrtu mecanismul digest. Directiva relevanta:
    - ***AuthDigestProvider*** *p1 p2 ...* – stabileste lista de provideri pentru mecanismul digest
- provideri de autentificare
  - **mod_authn_file** – permite stocarea bazelor de date cu useri/parola in fisiere text, in format asemanator cu */etc/passwd*. Directive relevante:
    - ***AuthUserFile*** *cale* – precizeaza calea catre fisierul cu useri/parole. Administrarea acestuia se realizeaza cu utilitarul *htpasswd* sau *htdigest* (vezi mai jos)
  - **mod_authn_dbm** – permite stocarea bazelor de date in fisiere DBM. Formatul DBM este unul binar, optimizat, superior ca performante variantei clear-text si deci mai potrivit pentru informatii de anvergura. Directive relevante:
    - ***AuthDBMType*** *tip* – precizeaza tipul de baza de date DBM utilizat (exista mai multe: SDBM, GDBM etc)
    - ***AuthDBMUserFile*** *cale* – stabileste calea care baza de date cu useri/parole
  - **mod_authn_dbd** – permite stocarea informatiilor de autentificare in baze de date SQL. Utilizarea sa presupune incarcarea unui alt modul – *mod_dbd* – ce asigura functiile de baza pentru interactiunea cu servere de baze de date. Directive relevante:
    - ***AuthDBDUserPWQuery*** *interogare* – stabileste interogarea SQL folosita pentru a extrage parola unui anumit user

Nota: pentru a despovara providerul in cazul unui numar mare de cereri autentificate pe secunda exista modulul *mod_authn_socache* care memoreaza temporar informatiile de autentificare, reducand astfel numarul de cereri adresate direct authentication provider-ului.

Atunci cand intr-un context sunt listati mai multi provideri de autentificare, username-ul este cautat pe rand in fiecare dintre ei pana cand este gasit. Acel provider va autentifica userul si, daca parola nu este corecta, NU vor fi incercati mai departe restul de provideri din lista! Altfel spus, trecerea la urmatorul provider din lista este determinata de absenta username-ului, dar nu si de incorectitudinea parolei.

Sectiunea urmatoare prezinta modul de “asamblare” a modulelor si directivelor de mai sus pentru a realiza autorizarea bazata pe identitatea userului.

11.6.3.3. Modalitatea de implementare a autentificarii

Pentru a pune ordine in cele expuse pana acum, sa inventariem mai intai elementele necesare protejarii unui set de resurse prin username/parola. Vom alege cazul mecanismului *basic* ce utilizeaza o baza de date memorata intr-un fisier text. Iata pasii necesari si rationamentul din spatele lor:

- serverul trebuie sa poata autoriza cererile in functie de identitatea clientului. In acest scop trebuie sa incarcam modulele *mod_authz_core* si *mod_authz_user*
- fiind necesara autentificarea userului, trebuie incarcat *mod_authn_core* si modulul corespunzator providerului ales – incazul nostru, *mod_authn_file*
- scenariul va folosi mecanismul de autentificare basic, in consecinta vom incarca modulul aferent: *mod_auth_basic*
- descriem cu ajutorul unei directive bloc setul de resurse ce va fi protejat prin user/parola. In interiorul acesteia trebuie precizate urmatoarele:
  - numele zonei parolate
  - mecanismul de autentificare folosit – in cazul nostru, basic

- providerii folositi – in cazul nostru, *file*
- calea catre baza de date cu useri/parole
- directivele de autorizare ce gestioneaza accesul la resursele dorite in functie de username-ul furnizat de client

- generam baza de date cu useri/parole folosind utilitarul *htpasswd*

Rezultatul se prezinta astfel:

```
# module de autorizare
LoadModule authz_core_module /usr/lib64/httpd/modules/mod_authz_core.so
LoadModule authz_user_module /usr/lib64/httpd/modules/mod_authz_user.so

# module de autentificare
LoadModule authn_core_module /usr/lib64/httpd/modules/mod_authn_core.so
LoadModule authn_file_module /usr/lib64/httpd/modules/mod_authn_file.so
LoadModule auth_basic_module /usr/lib64/httpd/modules/mod_auth_basic.so

<Directory /var/www/restricted>
      AuthType Basic
      AuthName "Zona restrictionata"
      AuthBasicProvider file
      AuthUserFile /apache/useri
      Require valid-user
</Directory>
```

Crearea fisierului cu utilizatori se face cu comanda **htpasswd** si optiunea **–c** (creare). Adaugarea oricarui alt utilizator in acelasi fisier se va face ulterior cu aceeasi comanda dar omitand optiunea -c:

```
# crearea fisierului si adaugarea unui prim user
root@server# htpasswd –c /apache/useri user1
# adaugarea unui alt user – nu mai trebuie folosit -c !
root@server# htpasswd /apache/useri user2
```

Daca username-ul specificat nu exista in fisier, el va fi adaugat; daca exista deja, i se va schimba parola cu cea specificata.

Atentie! *Folositi optiunea -c numai la crearea fisierului! Aplicata pe un fisier deja populat, ea cauzeaza golirea acestuia!*

Important! Fisierul cu useri trebuie plasat in afara DocumentRoot-ului, pentru a nu putea fi accesat direct de catre clienti.

Parolarea se poate face fie la nivel de sistem de fisiere (directiva Directory) fie la nivel de webspace (directiva Location). *Atentie! Alegerea gresita a tipului de container poate duce la compromiterea securitatii!*

11.7. Imbogatirea si reorganizarea webspace-ului prin alias-uri

Cu elementele de configurare expuse pana in acest moment, clientii nu pot “vedea” decat resursele aflate fizic dedesubtul directorului desemnat ca DocumentRoot. Exista insa multiple situatii de viata reala in care este nevoie de a publica si resurse suplimentare, aflate in directoare exterioare document root-ului. DocumentRoot-ul este unic per site, asadar nu avem solutia definirii de DocumentRoot-uri multiple; in schimb putem crea **alias**-uri.

Un alias este un URL-path virtual, care nu corespunde unei resurse prezente fizic sub document root, ci trimite catre directoare/fisiere aflate in exteriorul acestuia. Acest lucru este posibil deoarece serverul web are acces si la

resurse din afara document root-ului si deci poate face artificiul tehnic necesar pentru a le prezenta acea informatie clientilor. In acest fel, arborele de resurse publicate de catre serverul web (vazut de catre client ca un tot unitar) poate fi compus din resurse distribuite in sistemul de fisiere al serverului, analog cu modul in care sistemul de fisiere Unix se compune prin montarea diferitelor partitii pe care el rezida.

Modulul necesar pentru definirea de alias-uri este **mod_alias .** Iata cateva dintre directivele introduse de acesta:

- **Alias** – creeaza un URL-path virtual, care trimite catre o resursa aflata in afara document root-ului. Primul parametru este directorul virtual (URL-path-ul aparent) iar cel de-al doilea parametru reprezinta calea reala din sistemul de fisiere al serverului de unde vor fi livrate resursele din acest URL path virtual:

```
Alias  /docs  /apache/documentatii
```

  Din acest moment, URL-path-ul */docs* se va comporta ca un "director web" in toata regula. Sa consideram urmatorul exemplu: serverul are DocumentRoot-ul in */var/www*, iar clientul foloseste adresa *http://www.example.com/docs/intro.htm.* Serverul va primi cererea *GET /docs/intro.htm.* In mod normal (in absenta unui alias), serverul ar inlocui / cu */var/www* rezultand calea */var/www/docs/intro.htm*, ceea ce ar presupune prezenta fizica a directorului *docs* sub document root; in schimb el va detecta faptul ca inceputul URL-path-ului (*/docs*) este numele unui alias si va inlocui intregul sir */docs* cu corespondentul sau, */apache/documentatii*, rezultand astfel calea corecta catre fisierul dorit: */apache/documentatii/intro.htm*.

- **Redirect** – instiinteaza explicit clientul ca trebuie sa ceara un alt URL; ca urmare, clientul va depune o noua cerere, adresata URL-ului indicat de server. Redirectionarea este prioritara fata de aliasuri.

```
Redirect     /poze  http://www.altserver.ro/imagini
```

Nota: *un alias poate trimite si catre un fisier, nu neaparat catre directoare; in acest fel, este posibil ca pentru orice fisier cerut dintr-un anume director sa fie apelat un executabil.*

11.8. Servere virtuale (virtual hosts)

11.8.1. Principii

Un server web poate gazdui mai multe site-uri web, corespunzatoare mai multor nume DNS. Potentiala problema este efectuarea distinctiei intre site-urile solicitate de clienti: atunci cand un server gazduieste *site1.ro* si *site2.ro*, iar doi clienti solicita URL-urile *www.site1.ro/index.htm* si *www.site2.ro/index.htm*, in ambele cazuri URL-path-ul cerut serverului este */index.htm*. Asadar este nevoie de informatii suplimentare care sa-i clarifice serverului despre ce site este vorba in fiecare dintre cazuri.

Distinctia intre site-urile gazduite se poate face:

- **name-based** – serverul web are o adresa IP catre care pointeaza mai multe nume DNS. Serverul decide ce site doreste clientul in functie de numele pe care clientul l-a folosit pentru a accesa serverul. Numele in cauza este transmis de la client catre server prin intermediul headerului HTTP *Host*, care este obligatoriu incepand cu HTTP 1.1
- **IP-based** – serverul web are mai multe adrese IP, fiecare dintre ele avand propriul nume DNS. Serverul decide ce site doreste clientul in functie de adresa IP (a serverului!) pe care a sosit cererea

Pentru fiecare site gazduit de catre Apache este necesara folosirea unei directive bloc **<VirtualHost>**, care actioneaza ca un fel de server virtual – in interiorul ei este posibila utilizarea majoritatii directivelor utilizate in restul fisierului de configurare, cu diferenta ca directivele continute intr-un *<VirtualHost>* i se vor aplica numai serverului virtual in cauza.

11.8.2. Servere virtuale IP-based

Atunci cand dorim ca serverul sa deduca site-ul dorit de client in functie de adresa serverului pe care a sosit cererea, vom specifica adresa fiecarui site in cadrul directivei *<VirtualHost>* corespunzatoare. Iata exemplul unui server ce gazduieste *site1.ro* si *site2.ro* si care dispune de doua adrese IP – 1.2.3.4 si 5.6.7.8:

```
# www.site1.ro este livrat din /var/ww/site1
<VirtualHost 1.2.3.4:80>
      DocumentRoot /var/www/site1
</VirtualHost>

# www.site2.ro este livrat din /var/ww/site2
<VirtualHost 5.6.7.8:80>
      DocumentRoot /var/www/site2
</VirtualHost>
```

11.8.3. Servere virtuale name-based

Pentru a configura serverul sa deduca site-ul dorit de client pe baza numelui cu care clientul a accesat serverul, este suficient sa definim mai multe blocuri **<VirtualHost>** cu aceeasi adresa si port, ce contin urmatoarele directive:

1. **ServerName** – obligatoriu. Este folosit pentru diferentierea intre site-urile gazduite: acest nume va fi comparat cu cel folosit de client pentru a accesa serverul (provenit din headerul HTTP Host) pentru a decide daca cererea trebuie servita din VirtualHost-ul in cauza
2. **DocumentRoot** – indica radacina paginilor web pentru hostul virtual repsectiv (pentru a servi pagini diferite in functie de numele de site cerut de client)
3. (optional) **ServerAlias** – indica serverului nume DNS alternative ale hostului virtual si care trebuie deci servite din acelasi document root (in conditiile in care exista mai multe nume DNS care trebuie sa trimita catre aceeasi pagina web)
4. majoritatea directivelor prezente si in alte contexte (de verificat Context-ul in documentatia directivei respective). Directivele din *<VirtualHost>* au prioritate fata de cele din radacina fisierului de configurare.

```
<VirtualHost *:80>
      DocumentRoot /var/www/site1
      ServerName   www.site1.ro
      ServerAlias  site1.ro
</VirtualHost>
<VirtualHost *:80>
      DocumentRoot /var/www/site2
      ServerName   www.site2.ro
</VirtualHost>
```

Nota: *pentru a nu avea probleme cu DNS, se recomanda folosirea de adrese IP in loc de nume in VirtualHost*

11.8.4. Logica de alegere a virtual host-ului

Cele doua modalitati de a implementa virtual hosting (name based si IP based) pot coexista. In aceste conditii, o cerere venita de la client este analizata de server conform urmatoarei logici:

- se verifica mai intai daca IP-ul destinatie al cererii se regaseste in mai multe directive <*VirtualHost*>
  - daca nu, atunci serverul va cauta in continuare VH-uri al caror IP se potriveste cu cel al cererii sau, in ultima instanta, VH-uri de forma <*VirtualHost _default_:port*>. Daca nu e gasit un astfel de VH, cererea e servita de catre serverul principal (configuratia din *httpd.conf* aflata in afara directivelor VH, in radacina fisierului)
  - daca da, atunci numele prezent in headerul Host se compara pe rand cu cel din directiva ServerName a fiecarui VH din grupul identificat. **Daca nu se potriveste nici un VH (sau headerul Host nu exista) cererea va fi tratata din primul VH name-based!**

Exemplu: fie un server care are 4 adrese: 1.1.1.1, 2.2.2.2, 3.3.3.3 si 4.4.4.4, si configurarea VH de mai jos:

```
DocumentRoot /var/www/site0
ServerName www.site0.ro

# servere virtuale IP-based
<VirtualHost 2.2.2.2>
DocumentRoot /var/www/site2
ServerName www.site2.ro
</VirtualHost>

<VirtualHost 3.3.3.3>
DocumentRoot /var/www/site3
ServerName www.site3.ro
</VirtualHost>

# servere virtuale name-based
<VirtualHost 1.1.1.1>
DocumentRoot /var/www/site11
ServerName www.site11.ro
</VirtualHost>

<VirtualHost 1.1.1.1>
DocumentRoot /var/www/site12
ServerName www.site12.ro
</VirtualHost>

<VirtualHost 1.1.1.1>
DocumentRoot /var/www/site13
ServerName www.site13.ro
</VirtualHost>
```

Iata mai jos ce site este livrat pentru fiecare caz posibil:

Adresa serverului	Numele cu care este accesat serverul	Site livrat clientului	Explicatie
2.2.2.2	oricare	site2	este gasita o directiva bloc VH a carei adresa corespunde cu cea pe care a venit cererea; nu exista alt VH cu acelasi IP, deci suntem in cazul IP-based
3.3.3.3	oricare	site3	idem
4.4.4.4	oricare	site0	nu exista directive bloc VH (nici name-based, nici IP-based) care sa corespunda acelei adrese a serverului, deci cererea este servita din contextul global
1.1.1.1	www.site11.ro	site11	IP-ul destinatie se regaseste in mai multe VH-uri (deci suntem in cazul name-based) iar numele cu care a fost accesat serverul se regaseste intr-unul dintre VH-uri
1.1.1.1	www.site12.ro	site12	idem
1.1.1.1	www.site13.ro	site13	idem
1.1.1.1	www.site4.ro	site11	IP-ul destinatie se regaseste in mai multe VH-uri, dar numele cu care este accesat serverul nu corespunde nici unuia dintre ServerName-urile VH-urilor in cauza; in consecinta cererea e servita de primul dintre ele
1.1.1.1	orice alt nume in afara de www.site{1,2,3}.ro	site11	idem cazul anterior

11.9. BIBLIOGRAFIE

- Documentatia oficiala Apache: http://httpd.apache.org/docs/2.4/
- Cartea Apache Security (editura O'Reilly)
- Organizarea fisierelor de configurare in distributiile bazate pe Debian: http://www.control-escape.com/web/configuring-apache2-debian.html

11.10. ANEXA 1: Controlul accesului in Apache 2.2

11.10.1.1. Controlul accesului la resurse in functie de caracteristicile clientului

Accesul la resursele serverului poate fi controlat in functie de adresa sau numele statiei client. In acest scop este necesar ca modulul ce ofera aceasta facilitate sa fie incarcat:

- in Apache 2.0, numele modului este **mod_access**
- in Apache 2.2 el se numeste **mod_authz_host**

Nota: *controlul accesului la resurse se mai poate face si in alte moduri: spre exmplu,* ***mod_setenvif*** *permite setarea de variabile interne in functie de diferite caracteristici ale clientului si apoi autorizarea accesului in functie de valorile variabilelor.*

Directivele introduse de modulele *mod_access/mod_authz_host* sunt **Allow**, **Deny** si **Order**, care pot fi folosite in containere precum *Directory*, *File* sau *Location*. Fiecare dintre directivele **Allow** si **Deny** poate primi ca parametru o adresa IP, un nume DNS de statie, o subretea (un lot de IP-uri), un domeniu DNS sau cuvantul cheie all (care desemneaza toti clientii). Iata un exemplu de utilizare:

```
<Directory /var/www/admin
      Allow from 10.0.0.0/24
      Deny from 192.168
      Allow from infoacademy.ro
      Order allow,deny
</Directory>
```

Atentie! *Atunci cand se efectueaza filtrarea dupa numele DNS al clientului, trebuie tinut cont de faptul ca acela este numele obtinut prin rezolutie DNS inversa! La conectarea unui nou client, serverul web nu dispune decat de adresa IP a acestuia.*

In interiorul aceleiasi directive bloc se pot afla mai multe directive Allow si Deny. Atunci cand, in cazul unui client, apar suprapuneri de Allow si Deny, modalitatea de rezolvare a situatiei depinde de valoarea directivei **Order**:

- **Order Allow,Deny** – vor fi evaluate mai intai toate directivele Allow si abia apoi toate cele Deny (indiferent de ordinea in care apar in fisier). In caz de suprapunere, directivele Deny „au ultimul cuvant”. O cerere care nu isi gaseste corespondent in nici unul din cele doua seturi de reguli este respinsa (Deny din oficiu)
- **Order Deny,Allow** – prin analogie, Allow va fi evaluat ultimul si deci va avea prioritate. Cererile care nu au corespondent vor fi automat acceptate (Allow din oficiu)

Exemplu: interzicerea accesului la fisierele dintr-un director pentru clientul cu adresa IP 10.0.0.5:

```
<Directory /var/www/statistici>
      Allow from all
      Deny from 10.0.0.5
      Order allow,deny
```

```
</Directory>
```

Atentie! *In cadrul directivei Order, valorile Allow si Deny trebuie separate printr-o simpla virgula, fara spatii!*

Iata in continuare un exemplu explicat. Fie urmatoarea directiva bloc:

```
<Directory /var/www>
      Order allow,deny
      Allow from 10.0.0.0/24
      Deny from 10.0.0.100
</Directory>
```

Tabelul de mai jos listeaza ce se intampla in fiecare dintre cazurile posibile: cand clientul este 10.0.0.100, cand clientul are o alta adresa din reteaua 10.0.0.0/24, si in sfarsit cand clientul are o adresa din afara retelei 10.0.0.0/24:

Client	Acces	Comentarii
10.0.0.2	da	Se potriveste numai directiva Allow from 10.0.0.0/24
10.0.0.100	nu	Suprapunere intre Allow si Deny; ramane deny
192.168.0.4	nu	Nu se potriveste nici una dintre reguli; implicit Deny
Orice statie din reteaua 10.0.0.0/24 in afara de 10.0.0.100	da	Se potriveste numai directiva Allow from 10.0.0.0/24
Orice statie care nu face parte din reteaua 10.0.0.0/24	nu	Nu se potriveste niciuna dintre reguli si deci se aplica implicit Deny deoarece este ultimul din Order

11.10.1.2. Controlul accesului la resurse pe baza de autentificare

Atunci cand se doreste accesul la resurse pe baza de autentificare trebuie luate in considerare mecanismul de autentificare, backend-ul de autentificare si modul in care se face autorizarea. Iata-le detaliate:

- mecanismul de autentificare – specifica modul in care dialogheaza serverul si clientul pentru ca acesta din urma sa se autentifice. In Apache exista doua mecanisme disponibile:
  - **basic** – presupune transmiterea informatiilor de autentificare in clar intre server si client. Nu este recomandabil decat, cel mult, in cazul unei conexiuni criptate. Este implementat in Apache prin intermediul modulului:
    - **mod_auth_basic** (Apache 2.2)
    - **mod_auth** (Apache 2.0)
  - **digest** – un mecanism suportat de browserele mai noi, gandit de asa natura incat informatiile de autentificare nu mai circula in clar. In Apache 2.2 ii corespunde modulul **mod_auth_digest**
- modalitatea de stocare a informatiilor de autentificare. Autentificarea utilizatorilor se poate face cu baze de date stocate in diverse forme (fisier text, fisier binar, server MySQL, LDAP directory etc). In functie de aceasta trebuie incarcate modulele corespunzatoare. Vom prezenta in continuare modalitatea de parolare a unui director atunci cand baza de date cu utilizatori este memorata intr-un fisier text, ce presupune incarcarea modulului **mod_authn_file** in Apache 2.2
- felul in care se face autorizarea odata ce identitatea a fost verificata. Pentru a restrictiona accesul in functie de username-ul cu care a fost efectuata autentificarea, in Apache 2.2 este necesara incarcarea modulului **mod_authz_user**

Iata un exemplu explicat de acces autentificat la fisierele dintr-un director al serverului:

```
<Directory /var/www/restricted>
      AuthType Basic
      AuthName "Zona restrictionata"
      AuthUserFile /apache/useri
      Require valid-user
</Directory>
```

Explicatie:

- *AuthType mecanism* – stabileste mecanismul de autentificare dorit (basic sau digest). Petru a functiona, trebuie ca modulul corespunzator sa fi fost incarcat.
- *AuthName Nume* – stabileste numele zonei parolate asa cum va aparea el in fereastra de autentificare deschisa de browserul clientului. Este de asemenea folosit de client pentru a decide ce parola trebuie sa trimita serverului in cazul in care clientul mentine o lista de parole pentru diversele zone restrictionate accesate in acea sesiune
- *AuthUserFile fisier* – stabileste locatia fisierului cu utilizatori (vezi mai jos detalii in privinta crearii)
- *Require valid-user* – este directiva ce autorizeaza accesul la resursele directorului numai pentru utilizatorii autentificati corect conform fisierului specificat cu AuthUserFile. Alternativ se mai pot specifica aici:
  - o lista discreta de utilizatori: *Require user1 user2 ...* - util atunci cand dorim sa acordam accesul numai pentru o parte a utilizatorilor din AuthUserFile
  - un grup de utilizatori: *Require group g1 g2...* (grupul trebuie sa fi fost definit anterior; pentru aceasta exista directiva AuthGroupFile)

Crearea fisierului cu utilizatori se face cu comanda **htpasswd** si optiunea **–c** (creare). Adaugarea oricarui alt utilizator in acelasi fisier se va face ulterior cu aceeasi comanda dar omitand optiunea -c:

```
# crearea fisierului si adaugarea unui prim user
htpasswd –c /apache/useri user1

# adaugarea unui alt user
htpasswd /apache/useri user2
```

Atentie! *Folositi optiunea -c numai la crearea fisierului! Aplicata pe un fisier deja populat, ea cauzeaza golirea acestuia!*

Este recomandat ca fisierul cu useri sa fie plasat in afara DocumentRoot-ului, pentru a nu putea fi accesat direct de catre clienti.

Parolarea se poate face fie la nivel de sistem de fisiere (directiva Directory) fie la nivel de webspace (directiva Location). *Atentie! Alegerea gresita a tipului de container poate duce la compromiterea securitatii!*

```
<Location /products>
      Order Allow,Deny
      Allow from all
      Deny from 10.0.0.15
</Location>
```

Atunci cand intr-un context exista mai multe mecanisme de autorizare, efectul lor combinat este stabilit de directiva **Satisfy**, care are urmatoarele doua valori posibile:

- all – cererea trebuie sa fie autorizata de catre toate mecanismele de autorizare pentru a fi servita
- any – este suficient ca unul dintre mecanisme sa autorizeze cererea pentru ca aceasta sa fie servita