

## Il server Web Apache

## Apache

- Apache: **A PA**tCHy s**E**rver (<http://httpd.apache.org>)
  - Sviluppato sulla base del server NCSA a partire dal 1994
  - Versione più recente: Apache 2.2 (ultima release: Apache 2.2.15)
- **Free** e **open-source** (disponibilità del codice sorgente)
- **Portabilità**: SO Linux, Unix, Microsoft Windows, OS/2, ...
- Architettura **modulare**
  - **Nucleo** (core) piccolo che realizza le funzionalità di base
  - Estensione delle funzionalità di base mediante **moduli** (scritti usando l'Apache module API) **compilati staticamente** nel nucleo oppure **caricati dinamicamente** a tempo di esecuzione (**DSO**)
- Buon supporto dei protocolli (conformità con HTTP/1.1)
- Efficienza e flessibilità
- Stabilità, affidabilità, robustezza
  - Processo di sviluppo open source

SD - Valeria Cardellini, A.A. 2008/09

2

## Apache: alcune caratteristiche

- Alcune funzionalità disponibili:
  - Autenticazione
  - Negoziazione dei contenuti in base alle capacità del client
  - Virtual hosting (più siti Web sullo stesso server)
  - Personalizzazione di logfile e messaggi di errore
  - Possibilità illimitata di URL rewriting, aliasing e redirecting
    - Aliasing (mod\_alias)
      - Alias per accedere ad una risorsa reale sul server
      - Trasparente per il client
    - Redirecting (mod\_alias e mod\_rewrite)
      - Redirezione della richiesta verso un'altra URL locale o remota
      - Non è trasparente per il client
    - URL rewriting (mod\_rewrite)
      - Manipolazione e riscrittura flessibile dell'URL

SD - Valeria Cardellini, A.A. 2008/09

3

## Apache: il servizio HTTP

- Il servizio HTTP è fornito dal demone **httpd**
  - Eseguito continuamente in background per gestire richieste
- I file di configurazione (il principale è **httpd.conf**) vengono letti al momento dell'avvio di httpd
- In SO Unix-based, il demone httpd può essere eseguito lanciando lo script **apachectl**, che configura alcune variabili d'ambiente dipendenti dal SO
  - Necessari i privilegi di root per il binding sulla porta 80
- Con il comando apachectl è possibile specificare quattro operazioni
  - start: avvia il server
  - stop: blocca l'esecuzione del server
  - restart: riavvia il server
  - graceful: riavvia il server senza interrompere le connessioni aperte

SD - Valeria Cardellini, A.A. 2008/09

4

## Installazione e configurazione di Apache

- Installazione per sistemi Unix-like

**Download**     \$ lynx http://httpd.apache.org/download.cgi  
**Extract**     \$ gzip -d httpd-NN.tar.gz  
                 \$ tar xvf httpd-NN.tar  
                 \$ cd httpd-NN  
**Configure**    \$ ./configure --prefix=PREFIX  
**Compile**      \$ make  
**Install**       \$ make install  
**Customize**    \$ vi PREFIX/conf/httpd.conf  
**Test**          \$ PREFIX/bin/apachectl -k start

PREFIX è il path del file system sotto il quale installare Apache  
(per default /usr/local/apache2/)

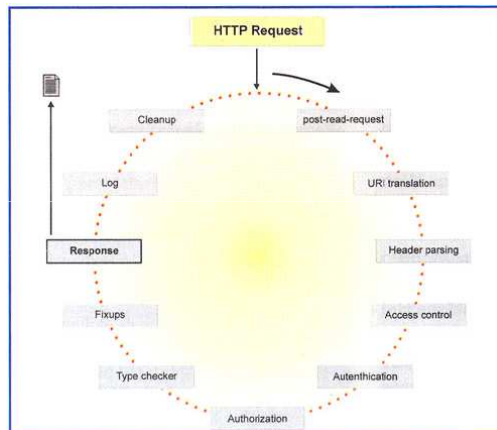
## Le directory in Apache

Apache utilizza le seguenti directory fondamentali:

- **ServerRoot**
  - Punto di origine dei file di amministrazione del server (/usr/local/apache2/)
- **bin** (/usr/local/apache2/bin/) (**usr/sbin**)
  - httpd, apachectl, ab (apache benchmark), htpasswd
- **conf** (/usr/local/apache2/conf/) (**/etc/apache2**)
- **logs** (/usr/local/apache2/logs/)
- **DocumentRoot**
  - Punto di origine dei documenti (/usr/local/apache2/htdocs/)  
(**var/www**)
- **Directory ScriptCGI**
  - Directory contenente script CGI (/usr/local/apache2/cgi-bin/)
- **Directory utente UserDir**
  - Directory contenente le pagine Web degli utenti del sistema (/home/\*/public\_html/)

## Ciclo richiesta-risposta

- Serie di fasi successive, in cui vengono prese decisioni sulla richiesta (elaborata, scartata oppure passata intatta alla fase successiva)
- Fasi gestite dal core di Apache (es., parsing di una richiesta, invio della risposta HTTP) oppure da moduli
  - Se non viene definito alcun modulo gestore per una determinata fase, Apache manda in esecuzione il gestore di default



## Ciclo richiesta-risposta (2)

- **Post-Read-Request**

Analisi dei principali header presenti nella richiesta HTTP e inizializzazione delle strutture dati utilizzate successivamente dai moduli che implementano le fasi di gestione
- **URI Translation**

L'URI richiesto può riferirsi a:

  - un file fisico
  - una risorsa dinamica prodotta da uno script esterno
  - una risorsa generata da un modulo interno

Il server deve sapere come individuare la risorsa, prima di poter effettuare decisioni successive: necessaria la conversione da URI a risorsa presente sul server

Direttive standard *Alias*, *ScriptAlias* e *DocumentRoot*: permettono di tradurre l'URI nel nome di un file presente nell'albero dei documenti

Moduli esterni come *mod\_rewrite* possono assumere il controllo di questa fase ed effettuare traduzioni più sofisticate

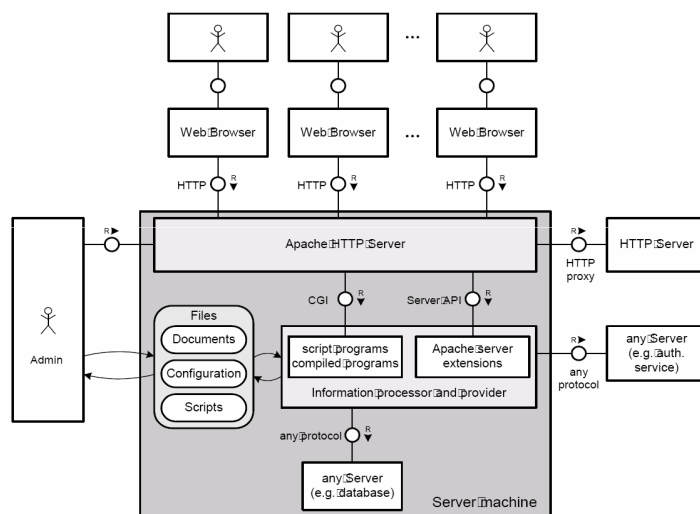
## Ciclo richiesta-risposta (3)

- **Header Parsing**  
Analisi dell'header della richiesta HTTP, al fine di estrarre informazioni riguardanti il client
- **Access control**  
Identificazione della locazione di provenienza della richiesta
- **Authentication**  
Richiesta di autenticazione del client
- **Authorization**  
Controllo dell'autenticazione
- **Mime type checking**  
Individuazione del tipo MIME della risorsa richiesta  
Il server deve sapere il tipo della modalità di elaborazione richiesta prima di poter preparare la risposta  
Noto il tipo di risorsa, Apache individua il gestore opportuno per la fase di risposta

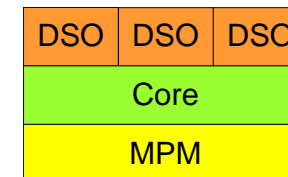
## Ciclo richiesta-risposta (4)

- **Fixup**  
Fase introdotta per permettere l'esecuzione di un qualunque tipo di operazione prima della fase di risposta (ad es. impostare un cookie)
- **Response**  
Le informazioni riguardanti la risorsa sono passate al gestore opportuno (*content handler*), che costruisce l'header della risposta HTTP e lo invia al client  
Successivamente, generazione o lettura del contenuto ed invio al client (o errore)
- **Logging**  
Scrittura su logfile dell'esito delle operazioni effettuate
- **Cleanup**  
Operazioni di chiusura, con cui si rilasciano le risorse allocate per la gestione della richiesta (ad es., liberare memoria principale, chiudere file)

## Architettura del Server



## Architettura modulare



- **Apache core**
  - Funzionalità di base (**fasi**)
  - Le fasi possono essere prese in consegna da specifici moduli DSO
  - Apache si “interfaccia” al SO sottostante tramite i moduli MPM
- **Apache MPM**
  - Fornisce un layer intermedio tra Apache ed il SO sottostante
  - Scopo: fornire un'interfaccia comune verso tutti i SO su cui Apache può girare
- **Apache DSO**

## I moduli di Apache

- L'architettura modulare di Apache permette di aggiungere o eliminare funzionalità semplicemente attivando o disattivando moduli SW
- All'avvio del server, è possibile scegliere i moduli che devono essere caricati, indicandoli nel file di configurazione
  - Alcuni moduli inclusi di default nel core server (*moduli standard*)
- I moduli possono essere compilati
  - Staticamente nel binario httpd
  - Dinamicamente (shared object: .so, .dll) sfruttando il meccanismo detto *Dynamic Shared Objects* (DSO)
    - DSO permette di costruire un pezzo di codice di programma in un formato speciale e di caricarlo a run-time nello spazio di indirizzamento del programma eseguibile
- I moduli sono scritti in linguaggio C o PERL

## Alcuni moduli

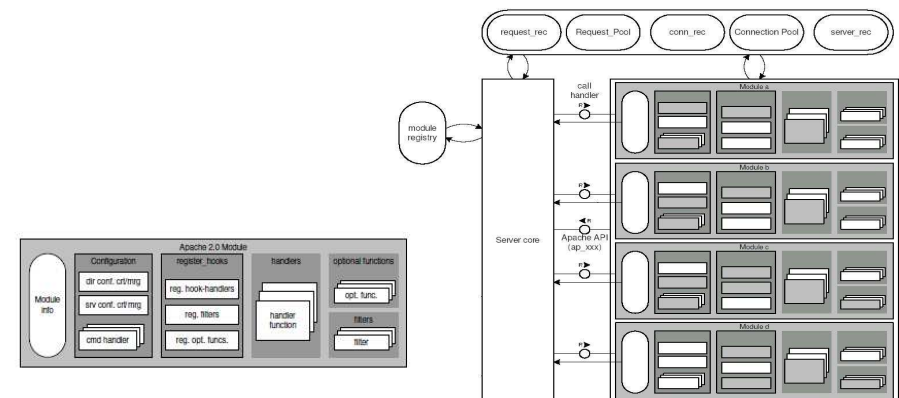
- <http://httpd.apache.org/docs/2.2/mod/> per la lista di tutti i moduli inclusi nella distribuzione
- mod\_rewrite (fase di URI translation)
  - Motore di riscrittura basato su regole per riscrivere dinamicamente l'URL richiesta
- mod\_access (fase di accesso)
- mod\_auth (fase di autenticazione)
- mod\_expires
  - Generazione degli header HTTP Expires e Age secondo criteri stabiliti dall'amministratore
- mod\_proxy
  - Proxy/gateway per Apache; altri moduli di supporto a mod\_proxy
- mod\_ssl
  - Supporto crittografico
- mod\_log\_config (fase di logging)
- mod\_status
  - Informazioni sull'attività e le prestazioni di Apache
- mod\_perl e mod\_php

## Apache: API

- Apache fornisce un'API per la programmazione di nuovi moduli
  - Supporto per C, C++ e Perl
- L'API permette allo sviluppatore del modulo di disinteressarsi dei dettagli implementativi legati al protocollo HTTP o alla gestione delle risorse del sistema
- L'API è costituita da un insieme di strutture e funzioni da utilizzare per creare i moduli aggiuntivi
- Quasi 500 moduli sviluppati
  - <http://modules.apache.org/>

## Architettura dei Moduli

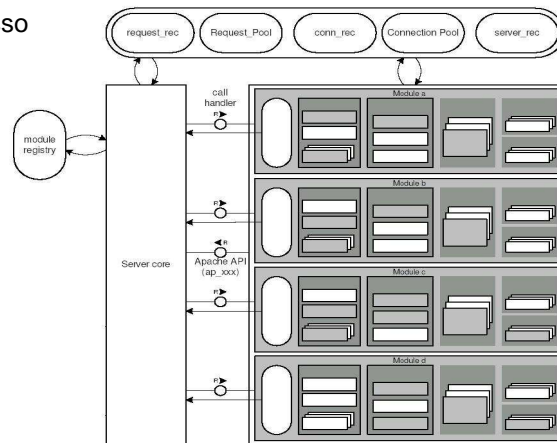
- Moduli registrano gestori (handlers) per hook del core o di altri moduli
- Il core di Apache invoca gli handler registrati all'attivazione dell'hook



## Architettura dei Moduli

- Tutti i processi server (master e child), contengono lo stesso codice eseguibile:

- Core
- Moduli caricati staticamente
- Moduli caricati dinamicamente

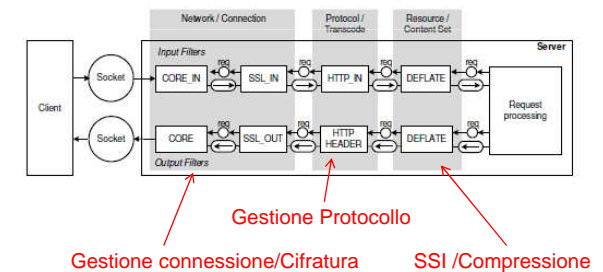


SD - Valeria Cardellini, A.A. 2008/09

17

## Content Handler - Filtri

- Il Content Handler è il componente che genera la risposta
- In Apache 2.0 la richiesta/risposta viene manipolata da filtri in cascata (filter chain)
  - Soluzione efficiente



SD - Valeria Cardellini, A.A. 2008/09

18

## Content Handler - Filtri

- Usando la tecnica del "bucket brigade"



Figure 3.7: Apache Filters: A Brigade contains a series of buckets

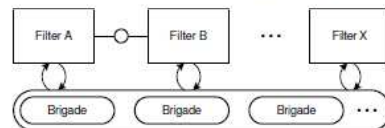


Figure 3.8: Apache Filters: A Filter chain using Brigades for data transport

SD - Valeria Cardellini, A.A. 2008/09

19

## Multi-Processing Module (MPM)

- A partire da Apache 2.0
  - Apache è stato progettato per essere flessibile su ogni tipo di piattaforma e con ogni configurazione d'ambiente
- Moduli MPM responsabili per binding su porta, accettare connessioni, gestire le richieste tramite processi child/thread
- Un modulo MPM deve essere scelto durante la configurazione e compilato nel server; permette
  - A SO differenti di fornire moduli appropriati per una maggiore efficienza
  - All'amministratore di applicare politiche di gestione diverse in base alle proprie esigenze
- Essendo specifici per il SO, solo un modulo MPM alla volta può essere caricato nel server
- Alcuni MPM:
  - prefork (massima stabilità), worker (massime prestazioni), event (variante sperimentale di worker), mpm\_winnt (default per Windows, basato su thread),

SD - Valeria Cardellini, A.A. 2008/09

20

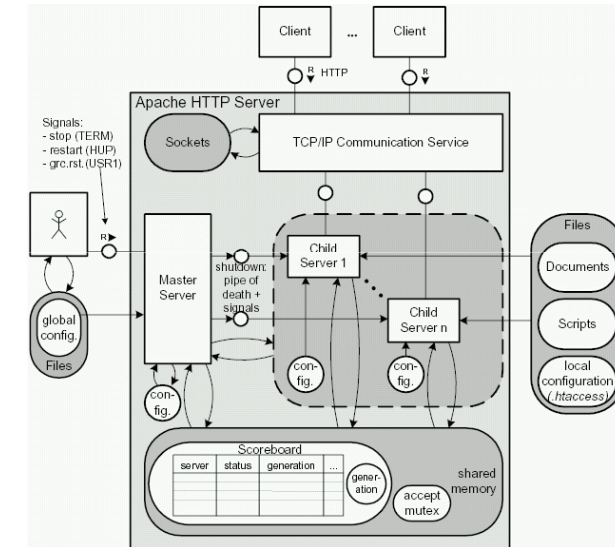
## Server prefork per il servizio di richieste HTTP

- Apache 1.3 ed Apache 2.2 (con **MPM prefork**)
- Architettura multi-process: esistono un processo principale (*padre*) ed alcuni processi ausiliari (*figli*) per il servizio delle richieste
  - Il padre manda in esecuzione i child (*pre-forking* dei child)
  - I child attendono le connessioni e le servono quando arrivano
  - Preforking basato sullo schema leader-follower già esaminato
- Vantaggi
  - Child creati una sola volta e poi riutilizzati (no overhead per fork())
  - Maggiore semplicità, stabilità e portabilità rispetto ad un server multi-threaded puro
- Svantaggi
  - Gestione del numero di child (presenza di processi child *idle* in grado di gestire una nuova connessione)
  - Come gestire dinamicamente il pool di processi child?

SD - Valeria Cardellini, A.A. 2008/09

21

## Architettura dell'MPM prefork

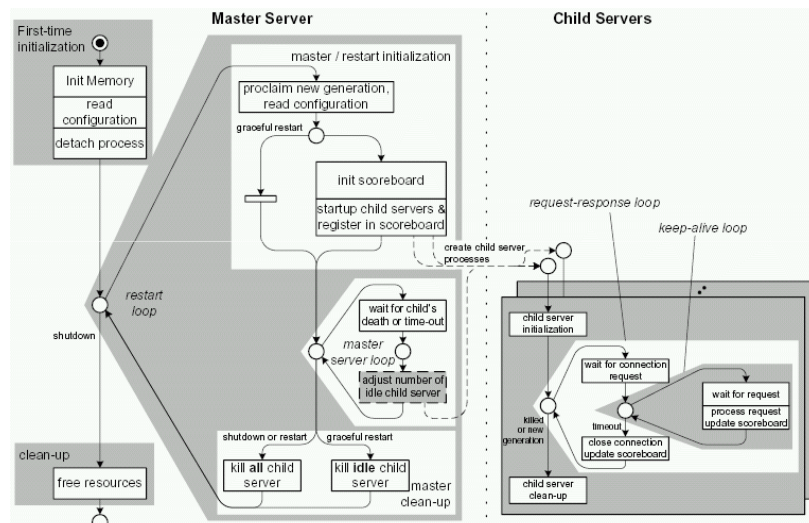


Tratto da <http://www.fmc-modeling.org/projects/apache>

SD - Valeria Cardellini, A.A. 2008/09

22

## Attività dell'MPM prefork



Tratto da <http://www.fmc-modeling.org/projects/apache>

SD - Valeria Cardellini, A.A. 2008/09

23

## Direttive per MPM prefork

- Direttive per la gestione dei processi child in Apache 1.3 ed Apache 2.2 (con **MPM prefork**)
  - **StartServers** (default 5): preforking dei processi child
  - **MaxClients** (default 256): limite sul numero di processi child
    - Numero massimo di richieste servite contemporaneamente
  - **MinSpareServers** (default 5) e **MaxSpareServers** (default 10): limite sul numero minimo e massimo di processi child idle
    - Per gestire dinamicamente il pool di processi child
  - **MaxRequestsPerChild** (default 10000): numero massimo di richieste HTTP servite da ciascun processo child
    - Allo scadere del numero di richieste il processo child termina
    - Può essere impostato a 0: il processo child non termina (problemi accidentali di memory leak)

SD - Valeria Cardellini, A.A. 2008/09

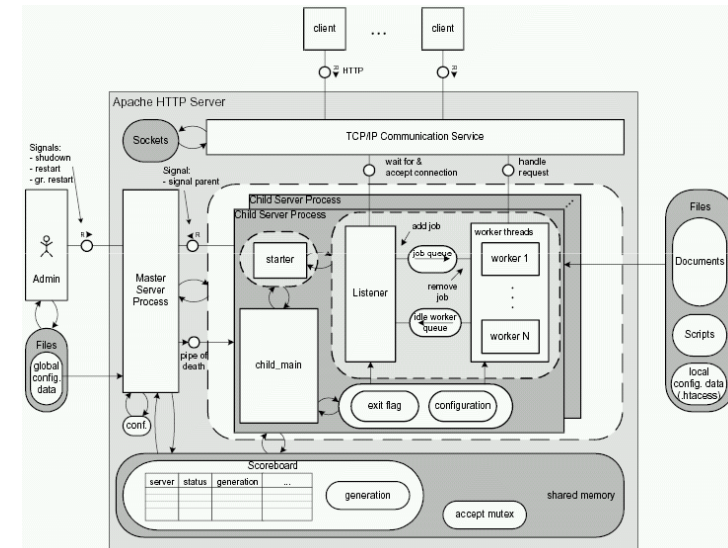
24



## Server multi-process multi-threaded per il servizio di richieste HTTP

- Apache 2.2 (con **MPM worker**)
- Architettura di server **ibrida**: un processo padre, molteplici processi figli, ciascuno dei quali genera multipli thread di esecuzione
  - Il padre manda in esecuzione i child
  - Ciascun child crea un numero fissato di thread server ed un thread listener
  - Quando arriva una richiesta, il listener la assegna ad un thread worker che la gestisce
- Vantaggi
  - Maggiore scalabilità e minor consumo di risorse del sistema
  - Stabilità simile (comunque inferiore) ad un server multi-process puro
- Svantaggi
  - Maggiore complessità del codice del server (gestione dei thread)
  - Supporto del multi-threading da parte del SO

## Architettura dell'MPM worker



Tratto da <http://www.fmc-modeling.org/projects/apache>

## Direttive per MPM worker

- Direttive per la gestione dei processi child e dei thread in Apache 2.2 (con **MPM worker**)
  - **StartServers** (default 3): preforking dei processi child
  - **ThreadsPerChild** (default 25): numero di thread creati da ciascun processo child
  - **MinSpareThreads** (default 75) e **MaxSpareThreads** (default 250): limite sul numero minimo e massimo di thread idle (complessivo per tutti i processi)
  - **MaxClients** (default  $\text{ServerLimit} \times \text{ThreadsPerChild}$ ): limite sul numero totale di thread
    - Numero massimo di richieste servite contemporaneamente
  - **ServerLimit** (default 16): limite sul numero di processi child attivi
    - $\text{ServerLimit} \geq \text{MaxClients} / \text{ThreadsPerChild}$
  - **ThreadLimit** (default 64): limite sul numero di thread creati da ogni processo child
    - $\text{ThreadLimit} \geq \text{ThreadsPerChilds}$

## Virtual hosting in Apache

- Virtual hosting (o multi-homing): più siti Web ospitati su di una singola macchina
  - Due architetture possibili: molteplici demoni httpd, un singolo demone httpd (Apache)
- Virtual hosting di due tipi:
  - Basato sul nome di dominio (**name-based**)
    - Al singolo indirizzo IP sono associati più nomi di dominio a livello di DNS
    - Una o più NIC a cui sono associati uno o più nomi logici (usando l'alias CNAME a livello di DNS)
    - E' necessario che il client supporti HTTP/1.1 (header Host)
  - Basato sull'indirizzo IP (**IP-based**)
    - Il server è dotato di uno o più indirizzi IP (reali o virtuali)
    - Una o più NIC a cui sono associati uno o più indirizzi IP (usando il comando `ifconfig alias`)

## Name Based Virtual Host: Esempio

```
# httpd.conf file
ServerName main.server.com
Port 80
ServerAdmin mainguy@server.com
DocumentRoot "/www/main/htdocs"

ScriptAlias /cgi-bin/
"/www/main/cgi-bin/"
Alias /images/
"/www/main/htdocs/images/"

<VirtualHost 192.168.1.100>
    ServerName vhost1.server.com
    ServerAdmin
    vhost1_guy@vhost1.server.com
    DocumentRoot
    "/www/vhost1/htdocs"
    ScriptAlias /cgi-bin/
    "/www/vhost1/cgi-bin/"
</VirtualHost>

...
<VirtualHost 192.168.1.110>
    ServerName vhost2.server.com
    ServerAdmin vhost2_guy@vhost2.server.com
    DocumentRoot "/www/vhost2/htdocs"
    ScriptAlias /cgi-bin/ "/www/vhost2/"
    Alias /images/
    "/www/vhost2/htdocs/images/"
</VirtualHost>
```

SD - Valeria Cardellini, A.A. 2008/09

29

## IP Based Virtual Host: Esempio

```
...
<VirtualHost 192.168.1.1>
    ServerName vhost1.server.com
    # Other directives go here
</VirtualHost>

<VirtualHost 192.168.1.2>
    ServerName vhost2.server.com
    # Other directives go here
</VirtualHost>

<VirtualHost 192.168.1.3>
    ServerName vhost3.server.com
    # Other directives go here
</VirtualHost>
...
```

Aggiungendo entry per ogni indirizzo nel file di configurazione del DNS server

**DNS records**

**; Address Records**

vhost1.server.com. IN A 192.168.1.1

vhost2.server.com. IN A 192.168.1.2

vhost3.server.com. IN A 192.168.1.3

Ed impostando gli indirizzi sulle interfacce Ethernet

/sbin/ifconfig eth0 192.168.1.1 up

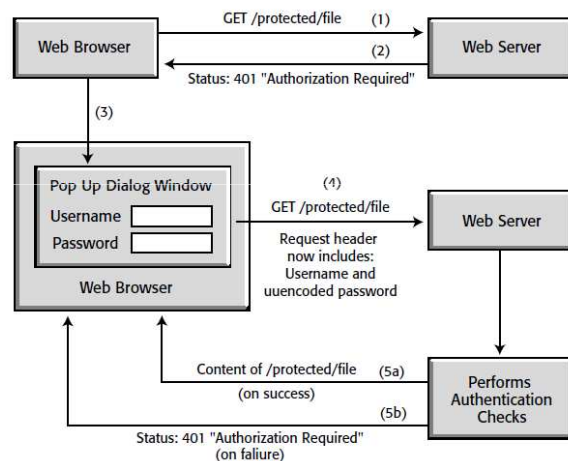
/sbin/ifconfig eth0:0 192.168.1.2 up

/sbin/ifconfig eth0:1 192.168.1.3 up

SD - Valeria Cardellini, A.A. 2008/09

30

## Autenticazione



SD - Valeria Cardellini, A.A. 2008/09

31

## Autenticazione: Esempio

- Aggiungere autenticazione per accedere alle pagine di un sito



**It works!**

This is the default web page for this server.

The web server software is running but no content has been added, yet.



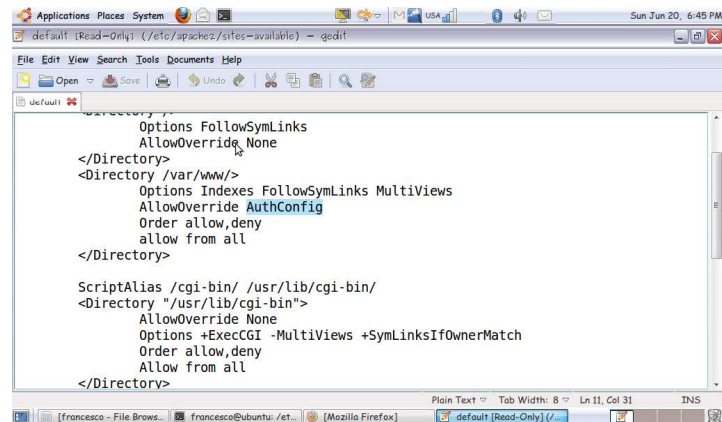
SD - Valeria Cardellini, A.A. 2008/09

32



## Autenticazione: Esempio

- Modificare la direttiva AllowOvveride

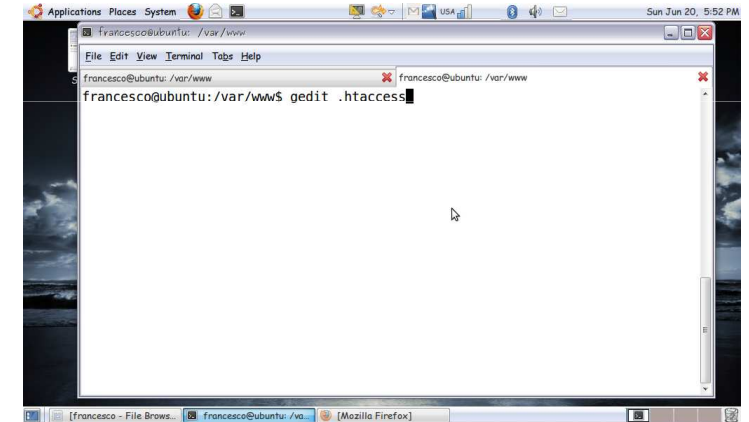


SD - Valeria Cardellini, A.A. 2008/09

33

## Autenticazione: Esempio

- Creare file .htaccess nella directory radice

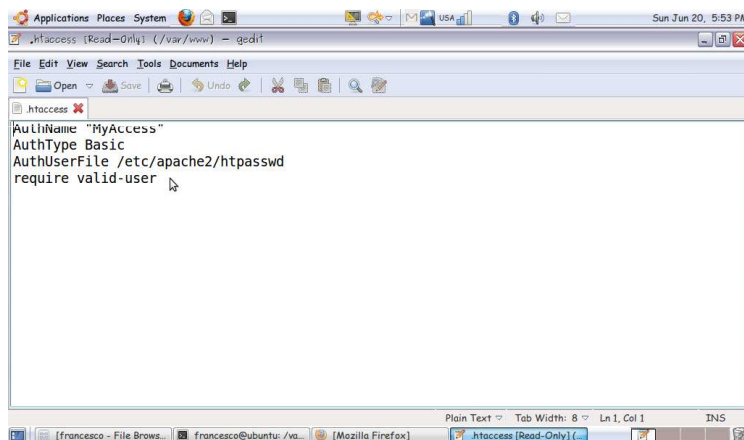


SD - Valeria Cardellini, A.A. 2008/09

34

## Autenticazione: Esempio

- Creare file .htaccess nella directory radice

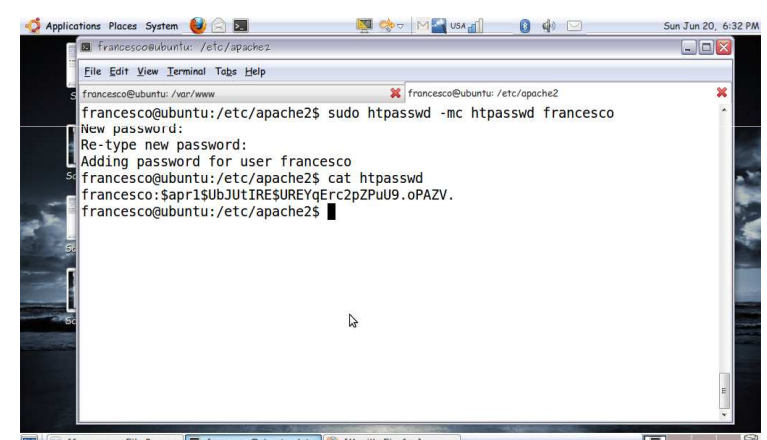


SD - Valeria Cardellini, A.A. 2008/09

35

## Autenticazione: Esempio

- Generare file di password ed inserire utenti

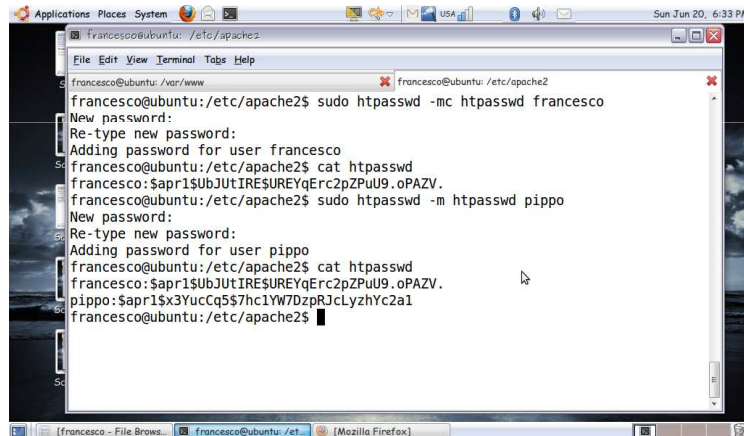


SD - Valeria Cardellini, A.A. 2008/09

36

## Autenticazione: Esempio

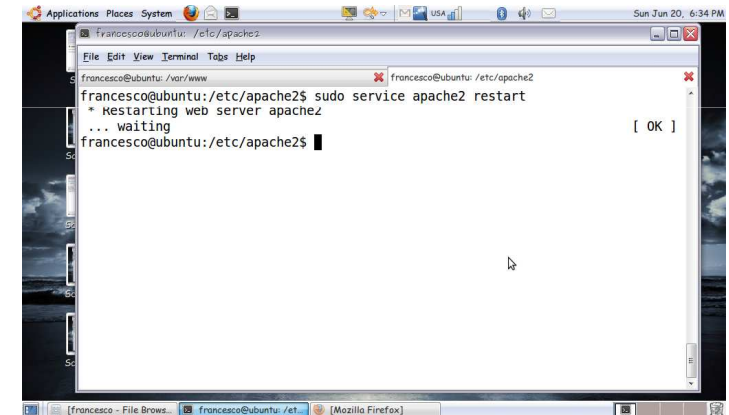
- Generare file di password ed inserire utenti



```
francesco@ubuntu: /etc/apache2$ sudo htpasswd -mc htpasswd francesco
New password:
Re-type new password:
Adding password for user francesco
francesco@ubuntu: /etc/apache2$ cat htpasswd
francesco:$apr1$SubJUTIRE$UREYqErc2pZPuU9.oPAZV.
francesco@ubuntu: /etc/apache2$ sudo htpasswd -m htpasswd pippo
New password:
Re-type new password:
Adding password for user pippo
francesco@ubuntu: /etc/apache2$ cat htpasswd
francesco:$apr1$SubJUTIRE$UREYqErc2pZPuU9.oPAZV.
pippo:$apr1$X3YucQq5$7hc1YW7DzpRjClyzhYc2a1
francesco@ubuntu: /etc/apache2$
```

## Autenticazione: Esempio

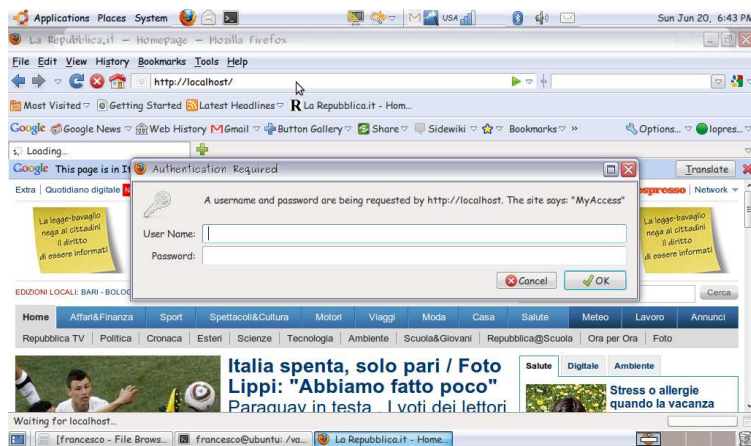
- Far ripartire il server



```
francesco@ubuntu: /etc/apache2$ sudo service apache2 restart
* Restarting web server apache2
... waiting
francesco@ubuntu: /etc/apache2$
```

## Autenticazione: Esempio

- Accedere di nuovo alla pagina...



## Apache: i file di configurazione

- Contengono le opzioni di configurazione (dette *direttive*) del server stesso e dei moduli usati
- Le direttive sono analizzate in sequenza
  - Attenzione all'ordine con cui sono scritte!
- httpd.conf
  - È il file di configurazione principale: configura il demone (numero di porta, utente, ecc.) e le sue funzionalità
- mime.types
  - Definizione tipi MIME
- File .htaccess
  - Consentono di modificare la configurazione per ciascuna directory del Web tree

## Apache: httpd.conf

- Le direttive sono raggruppate in 3 sezioni principali
- Direttive che controllano le **impostazioni globali**
  - ServerRoot, connessioni persistenti (KeepAlive, MaxKeepAliveRequests, KeepAliveTimeout), Listen, LoadModule, impostazioni su processi e thread (in base al modulo MPM scelto), ...
- Direttive che controllano le **impostazioni del server principale**
  - ServerName, DocumentRoot, <Directory>, AllowOverride, Allow, Deny, UserDir, Logging, ...
- Parametri di configurazione del **virtual hosting**

## Esempio di httpd.conf

```
#####  
# Section 1: Global Environment  
# Many of the values are default values, so the directives could be omitted.  
ServerType standalone  
ServerRoot "/etc/httpd"  
Listen 80  
Listen 8080  
Timeout 300  
KeepAlive On  
MaxKeepAliveRequests 100  
KeepAliveTimeout 15  
MinSpareServers 5  
MaxSpareServers 10  
StartServers 5  
MaxClients 150  
MaxRequestsPerChild 10000
```

## Esempio di httpd.conf (2)

```
#####  
# Section 2: "Main" server configuration  
ServerAdmin webmaster@foo.org  
ServerName www.foo.org  
DocumentRoot "/var/www/html"  
# a very restrictive default for all directories  
# .htaccess files are completely ignored  
<Directory />  
Options FollowSymLinks  
AllowOverride None  
</Directory>  
<Directory "/var/www/html">  
Options Indexes FollowSymLinks MultiViews  
AllowOverride None  
Order allow,deny  
Allow from all  
</Directory>
```

## Esempio di httpd.conf (3)

```
#####  
# Section 3: virtual hosts  
<VirtualHost www.foo.com:80>  
# all hosts in the www.ce.uniroma2.it domain are allowed access;  
# all other hosts are denied access  
<Directory />  
Order Deny,Allow  
Deny from all  
Allow from www.ce.uniroma2.it  
</Directory>  
# "Location" directive will only be processed if mod_status is loaded  
# To enable status reports only for browsers from foo.com domain  
<IfModule mod_status.c>  
<Location /server-status>  
SetHandler server-status  
Order Deny,Allow  
Deny from all  
Allow from .foo.com  
</Location>  
</IfModule>  
</VirtualHost>
```

## Utilità dei logfile di un server Web

- Monitorare gli accessi ad un server Web e lo stato del server
  - Le informazioni memorizzabili nel logfile sono quelle che viaggiano negli header HTTP di messaggi di richiesta e risposta
  - Generalmente, i server Web permettono di definire quali campi dei messaggi devono essere memorizzati (**logfile custom**)
- Capacity planning
- Billing
  - Esempio: banner pubblicitari
- Attack detection

## Informazioni estraibili dai logfile

- Le informazioni che maggiormente si vogliono trarre da un logfile riguardano:
  - numero di utenti del sito e loro provenienza geografica
  - browser utilizzati
  - giorni ed orari di maggior affluenza
  - pagine più popolari
  - errori verificatisi per determinare la presenza di link sbagliati all'interno delle pagine del sito
  - siti che fanno riferimento al proprio sito
- Attenzione: i proxy possono falsare i risultati!

## Common Logfile Format

- Il W3C ha definito uno standard per il logfile, denominato Common Logfile Format (CLF)
- Nel Common Logfile, ogni riga rappresenta una richiesta e si compone di più campi, separati da uno spazio
  - se il campo non assume alcun valore, viene indicato con il simbolo '-'
  - la riga termina con CRLF
- Formato: **host ident authuser date request status bytes**
  - Esempio:  
`213.45.176.42 - - [12/Dec/2007:20:53:54 +0100] "GET /courses/iw08/ HTTP/1.1" 200 32096`

## Logfile custom in Apache

- La definizione in Apache di un logfile custom avviene mediante l'uso della direttiva LogFormat in httpd.conf:  
**LogFormat string**  
`LogFormat "%h %l %u %t \"%r\" %>s %b" common`  
`LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\"" combined`
  - Gli elementi nel campo stringa possono essere:
    - %h remote host
    - %l remote logname
    - %u remote user
    - %t timestamp della richiesta
    - \"%r\" prima riga della richiesta
    - %>s codice di stato della risposta
    - %b byte trasmessi
    - %{Header}i header nella richiesta
    - %{Header}o header nella risposta