NGINX Web Server

Tommaso Sardelli
sardelli.tommaso[at]gmail.com

Corsi GNU/Linux Avanzati 2015
Politecnico Open unix Lab

8 Aprile 2015

Di cosa parliamo oggi?

- Cos'è un web server
- Come configurarlo
- Come renderlo sicuro

Di cosa parliamo oggi?

Indice Generale

Introduzione

- Cos'è un Web Server
- Parliamo di NGINX

Installazione e Configurazione

- Repository e Sorgenti
- La configurazione di NGINX
- FastCGI e PHP-FPM

Sicurezza

- SSL/TLS e HTTPS
- HTTPS Hardening

Link

Outline

Introduzione
- Cos'è un Web Server
- Parliamo di NGINX

Installazione e Configurazione
- Repository e Sorgenti
- La configurazione di NGINX
- FastCGI e PHP-FPM

Sicurezza
- SSL/TLS e HTTPS
- HTTPS Hardening

Link

Cos'è un Web Server

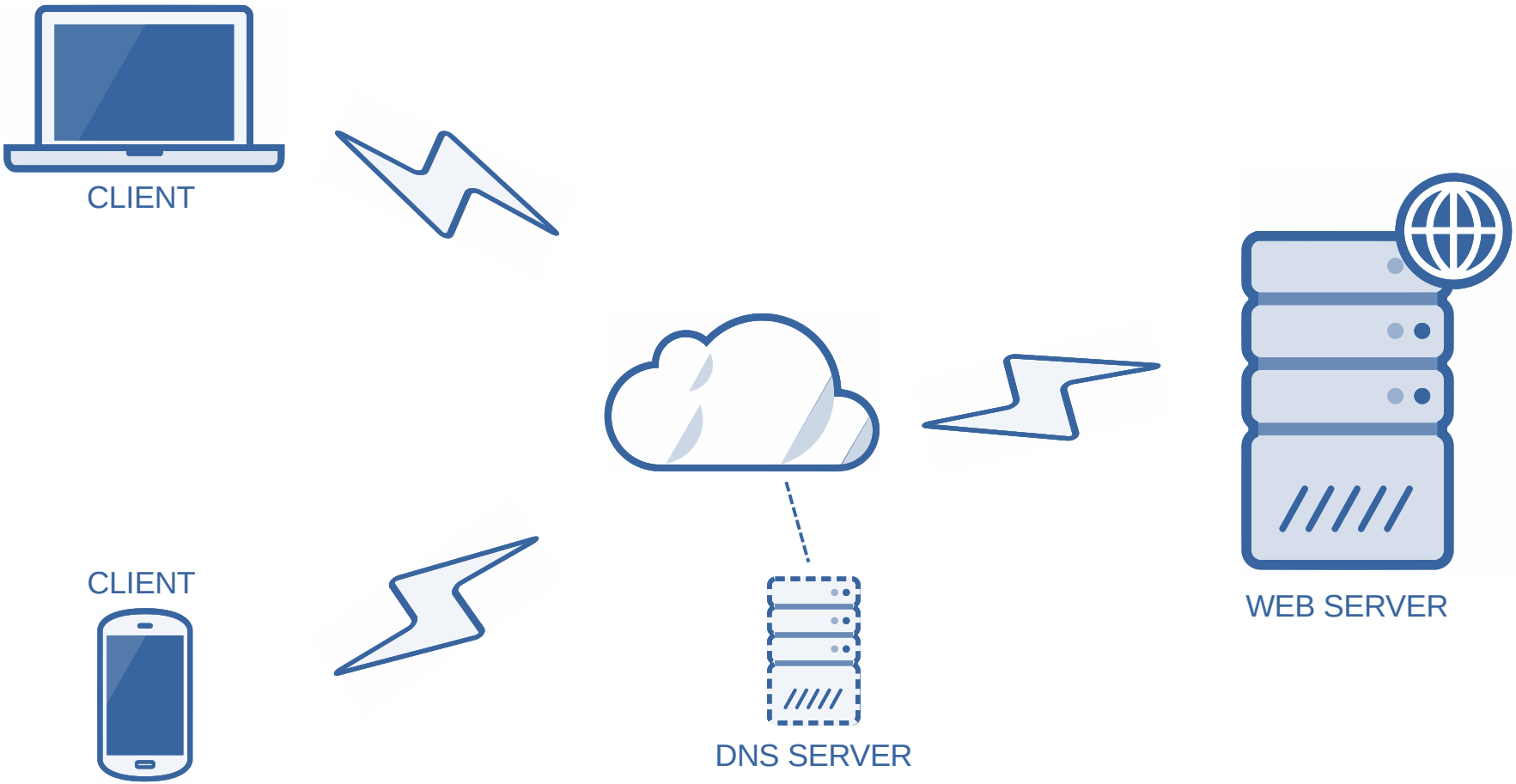
- Un web server è quel componente che **invia pagine web** (solitamente in formato html) ai browser, nel momento in cui le **richiedono**.
- Tutto questo avviene tramite il protocollo di comunicazione **HTTP.**
- Se vogliamo creare un sito internet, oppure un blog con Wordpress o ancora, caricare i nostri file su ownCloud, abbiamo bisogno di un **web server**.

Analisi di una richiesta HTTP

Il protocollo HTTP è usato come principale sistema per la trasmissione d'informazioni sul web: il **client esegue una richiesta e il server restituisce la risposta.**

- Method (**GET**, POST, PUT, DELETE, etc.)
- URI (generalmente è la risorsa richiesta)
- Headers
- Body

Come funziona



Demo

Outline

Introduzione
- Cos'è un Web Server
- Parliamo di NGINX

Installazione e Configurazione
- Repository e Sorgenti
- La configurazione di NGINX
- FastCGI e PHP-FPM

Sicurezza
- SSL/TLS e HTTPS
- HTTPS Hardening

Link

Caratteristiche Principali

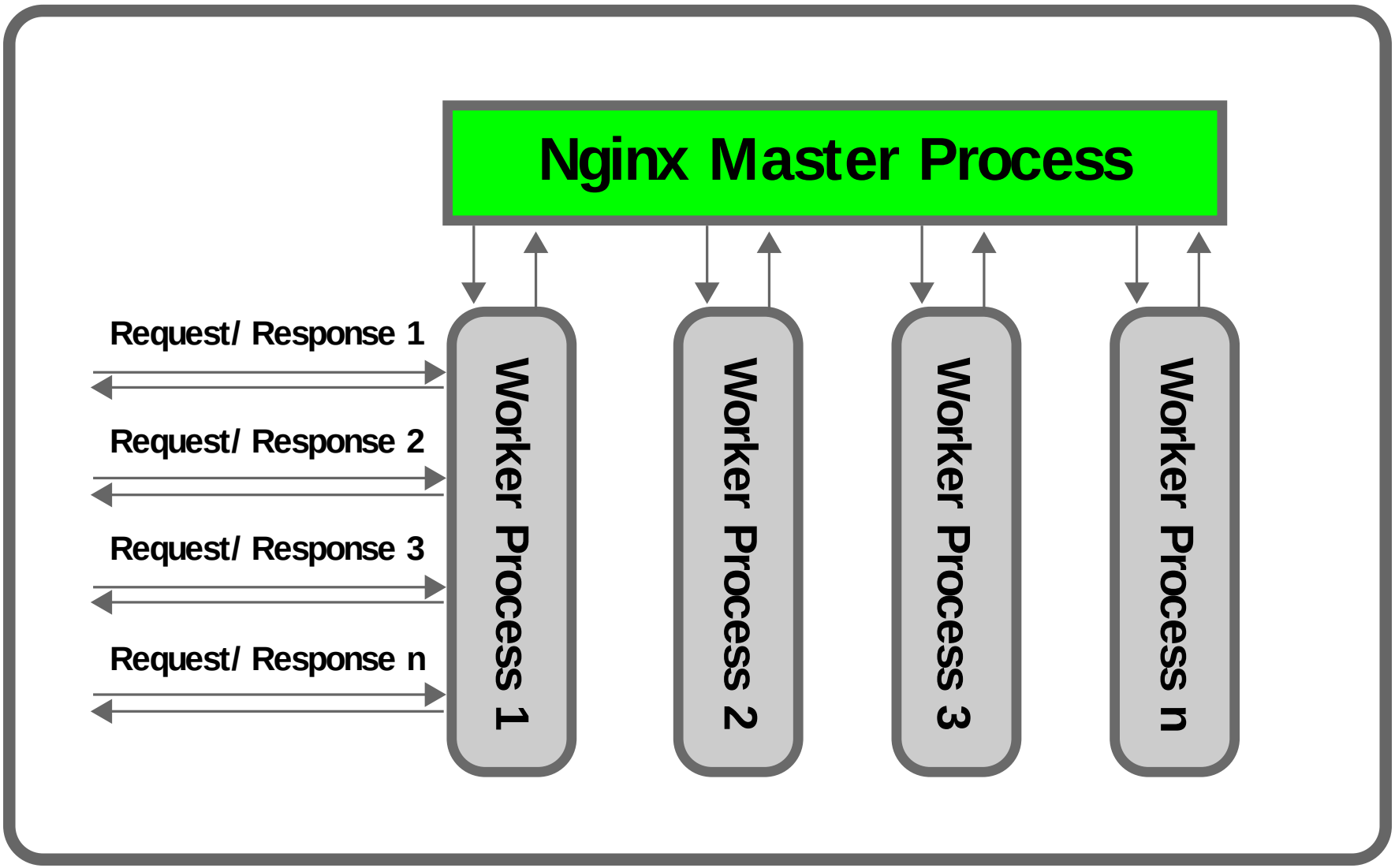
- Programma **leggerissimo** sia nel consumo di memoria che nell'utilizzo della CPU.
- **Performance elevate** grazie a una mirata gestione dei processi/thread.
- Perfetto per il **load balancing**.
- **Altamente configurabile** anche a basso livello (numero di file aperti per ogni processo, utilizzo dei core della CPU, ...).
- **Modulare**, offre infatti la possibilità di attivare o disattivare funzionalità in fase di compilazione.

Architettura di NGINX

NGINX è suddiviso in due processi:

- **Master process**: viene lanciato da root ed è il processo principale. Ha il compito di leggere i file di configurazione e di aprire il socket che gli permetterà di comunicare con i worker processes.
- **Worker processes**: sono una serie di processi lanciati come utente non privilegiato (ad esempio www-data) che hanno il compito effettivo di servire le richieste HTTP.

Architettura di NGINX



Outline

Introduzione
- Cos'è un Web Server
- Parliamo di NGINX

Installazione e Configurazione
- Repository e Sorgenti
- La configurazione di NGINX
- FastCGI e PHP-FPM

Sicurezza
- SSL/TLS e HTTPS
- HTTPS Hardening

Link

Installazione su Debian

A partire da **Debian 7 Wheezy** vengono forniti 4 diversi pacchetti di NGINX.

- nginx-light: contiene soltanto un minimo numero di core modules.
- **nginx-full**: contiene tutti i core modules (è quello che viene installato quando si installa il metapacchetto **nginx**).
- nginx-extras: contiene tutti i core modules più alcuni moduli extra.
- nginx-naxsi: contiene il modulo per naxsi Web Application Firewall.

`https://wiki.debian.org/Nginx`

Installazione da sorgenti

Quando compilare NGINX da sorgenti:

- Si vogliono abilitare dei moduli che non sono presenti nella versione fornita dalla distribuzione in uso.
- Si vogliono disabilitare moduli che sappiamo non verrebbero usati.
- La versione precompilata fornita dalla distribuzione è **troppo datata**.

Info sulla versione installata

- Per visualizzare la **versione installata:**

```
sudo nginx -v
```

Info sulla versione installata

- Per visualizzare la **versione installata:**

```
sudo nginx -v
```

- Per avere la lista completa dei **moduli abilitati:**

```
sudo nginx -V
```

Info sulla versione installata

- Per visualizzare la **versione installata:**

```
sudo nginx -v
```

- Per avere la lista completa dei **moduli abilitati:**

```
sudo nginx -V
```

- Per **cercare un modulo** in particolare tra quelli abilitati:

```
sudo nginx -V 2>&1 | grep --color nome_modulo
```

Outline

Introduzione
- Cos'è un Web Server
- Parliamo di NGINX

Installazione e Configurazione
- Repository e Sorgenti
- La configurazione di NGINX
- FastCGI e PHP-FPM

Sicurezza
- SSL/TLS e HTTPS
- HTTPS Hardening

Link

nginx.conf

Come abbiamo detto nell'introduzione, la semplicità nella configurazione è stato da sempre uno degli obiettivi di NGINX.

nginx.conf

Come abbiamo detto nell'introduzione, la semplicità nella configurazione è stato da sempre uno degli obiettivi di NGINX.

- Interamente configurabile mediante un unico file (**/etc/nginx/nginx.conf** su Debian).
- È comunque possibile **spezzare il file** in tanti file diversi richiamabili con "include".
- Al termine della configurazione diamo **sudo nginx -t && sudo service nginx reload** per controllare errori sintattici e attivare le modifiche.

Context

- Il file di configurazione di NGINX è diviso in **context** che sarebbero blocchi di direttive racchiusi tra parentesi graffe.
- Esistono 5 context e sono: **main, events, http, server e location.**
- Nel file di configurazione possiamo avere **un solo blocco** events e http ma **molteplici** blocchi server e location.
- I context sono **ereditari,** cioè ogni context eredita le impostazioni del context in cui viene inserito.
- I context **location** si trovano sempre all'interno di context **server** che a loro volta risiedono nel context **http.**

Struttura di nginx.conf

```
user www-data;
worker_processes 1;
pid /run/nginx.pid;

events {
    worker_connections 128;
}

http {

    server {

        location {
         }
    }
}
```

main e events

- **user www-data:** specifichiamo da quale utente saranno lanciati i worker processes (es. www-data).
- **worker_processes:** con questa direttiva decidiamo quanti worker processes avviare, solitamente si consiglia **uno per core.**
- **pid /run/nginx.pid:** diciamo a NGINX in quale file salvare il proprio PID.
- **worker_connections 128:** Impostiamo il numero di connessioni che può gestire ciascun worker process (dobbiamo trovare il valore ideale facendo dei test ad esempio con httperf).

Blocco http

- È il blocco di configurazione **globale**, le direttive inserite in questo blocco hanno effetto su tutti i siti serviti da NGINX.
- Generalmente le impostazioni di default non pregiudicano il funzionamento di NGINX ma è sempre bene **modificarle in base alle nostre necessità.**

Blocchi server

- È il blocco usato per configuare i **virtual domains** (VirtualHosts per chi è pratico di Apache).
- Avremo un blocco server per ogni sito (dominio o sottodominio) hostato sul server.
- Insieme al blocco location è quello che andremo a configurare manualmente ogni volta che vogliamo far girare qualcosa su NGINX.

VirtualHosts

```
http{
    server {
        listen 80;
        server_name example.org www.example.org;
        ...
    }

    server {
        listen 80;
        server_name poul.org www.poul.org;
        ...
    }

    server {
        listen 80;
        server_name aranzulla.it www.aranzulla.it;
        ...
    }
}
```

Blocchi location

- In questi blocchi troviamo le direttive che operano su file e cartelle
- Queste direttive ci permettono di decidere come NGINX debba comportarsi quando viene richiesta una specifica risorsa
- È possibile utilizzare un percorso specifico per indicare una risorsa
  - In questo caso useremo il prefisso “=”
- In alternativa possiamo eseguire il matching utilizzando le **regex**
  - Prefisso **"~"** se vogliamo che il matching sia case sensitive
  - Prefisso **"~*"** se vogliamo che il matching sia case insensitive

```
Syntax: location [ = | ~ | ~* | ^~ ] uri { ... }
```

Alcuni esempi

```
location ~ /download {
        autoindex on;
}
```

Alcuni esempi

```
location ~ /download {
        autoindex on;
}
```

- Impedire l'accesso a script eseguibili all'interno delle cartelle con permessi di scrittura

```
location ~* /(images|cache|media|logs|tmp)/.*.(php|pl|py)$ {
     return 403;
     error_page 403 /403_error.html;
}
```

Demo

Outline

Introduzione
- Cos'è un Web Server
- Parliamo di NGINX

Installazione e Configurazione
- Repository e Sorgenti
- La configurazione di NGINX
- FastCGI e PHP-FPM

Sicurezza
- SSL/TLS e HTTPS
- HTTPS Hardening

Link

CGI

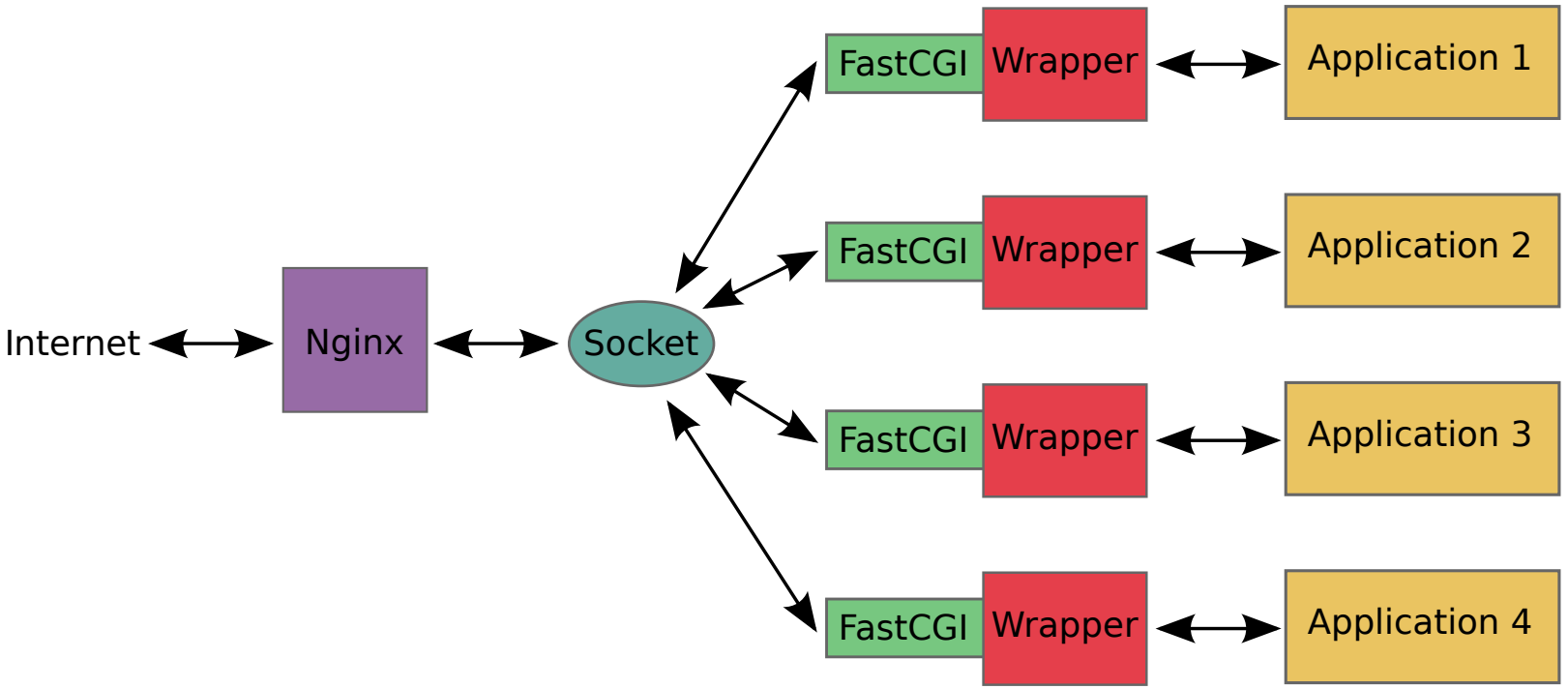
- CGI o Common Gateway Interface, è un meccanismo ideato per permettere di realizzare e far girare applicazioni lato server, in **qualunque linguaggio** (C, PHP, Python, Perl, ...)
- Le richieste generalmente vengono servite in questo modo:
  - Il web server riceve una richiesta per servire una certa pagina (relativa all'applicazione).
  - **Lancia un processo** tramite CGI per eseguire il programma e gli passa come parametri da linea di comando le varie variabili passate dalla GET/POST.
  - Il programma esegue il codice e restituisce l'output al web server che inoltra la risposta al client.

FastCGI - l'evoluzione

FastCGI è la diretta evoluzione di CGI, riprende alcune delle sue funzionalità migliorando però due aspetti importanti:

- **Performance:** I processi lanciati da FastCGI sono persistenti e vengono riutilizzati per servire diverse richieste. Viene creato un socket per scambiare le richieste tra web server e applicazione. Si occupa poi l'applicazione di generare eventualmente altri processi.
- **Scalabiltà:** FastCGI infatti offre la possibilità di eseguire le applicazioni da remoto permettendo la distribuzione del carico di lavoro su server differenti.

Architettura di FastCGI



php-fpm.conf

L'implementazione più famosa di fastCGI per PHP è **PHP-FPM** ed è quella utilizzeremo per NGINX. È presente nei repo di tutte le principali distribuzioni.

php-fpm.conf

L'implementazione più famosa di fastCGI per PHP è **PHP-FPM** ed è quella utilizzeremo per NGINX. È presente nei repo di tutte le principali distribuzioni.

- Su Debian lo troviamo nel pacchetto **php5-fpm**
- Possiamo configurarlo con il file **"/etc/php5/fpm/pool.d/www.conf"**
- All'interno di questo file ci interessa trovare il percorso del socket con cui far comunicare NGINX e PHP-FPM
  - "**listen = /var/run/php5-fpm.sock**"

Passiamo gli script PHP a FastCGI

Per abilitare l'esecuzione di codice PHP all'interno di un sito è sufficiente creare un apposito **blocco location** all'interno del context server interessato.

```
location ~ \.php$ {
    try_files $uri =404;
    fastcgi_pass unix:/var/run/php5-fpm.sock;
    include fastcgi.conf;
    fastcgi_index index.php;
}
```

Analizziamo il blocco

- **“location ~ \.php$”:** Cerca tutti i file con estensione .php
- **“try_files $uri =404”:** Controlla che il file esista veramente per evitare l’esecuziona arbitraria di codice, se così non fosse restituisce un errore 404.
- **“fastcgi_pass unix:/var/run/php5-fpm.sock”:** Indichiamo il full path del socket di PHP-FPM.
- **“include fastcgi.conf“:** Include il file contenente le impostazioni predefinite di FastCGI.

Demo

Outline

Introduzione
- Cos'è un Web Server
- Parliamo di NGINX

Installazione e Configurazione
- Repository e Sorgenti
- La configurazione di NGINX
- FastCGI e PHP-FPM

Sicurezza
- SSL/TLS e HTTPS
- HTTPS Hardening

Link

It's a conspiracy!

Cos'è HTTPS?

- HyperText Transfer Protocol over Secure Socket Layer.
- È il risultato dell'applicazione di un protocollo di **crittografia asimmetrica e simmetrica** al protocollo di trasferimento di ipertesti HTTP.
- Viene utilizzato per garantire trasferimenti riservati di dati nel web, in modo da impedire intercettazioni dei contenuti che potrebbero essere effettuati tramite tecniche di attacco **man in the middle**.
  - Password
  - Dati sensibili
  - Furti di identità

Come funziona?

- Aggiunge un livello di sicurezza all'HTTP grazie al protocollo di crittografia noto come **Transport Layer Security (TLS),** o il suo predecessore Secure Sockets Layer (SSL) **(da evitare)**.
- Crea quindi un **canale di comunicazione** criptato tra il client e il server attraverso uno scambio sicuro di **chiavi di cifratura.**
- una volta stabilito questo canale sicuro, al suo interno viene utilizzato il protocollo **HTTP** per la comunicazione tra le parti.
- Impiega come porta di default la **443** e non la porta 80 come per l'HTTP non criptato.

Bello...Lo voglio!

Possiamo facilmente creare dei cosiddetti **self-signed certificates** per proteggere il traffico da e verso i nostri siti.

```
sudo mkdir -p /etc/nginx/ssl && cd /etc/nginx/ssl

# Creiamo la chiave privata
sudo openssl genrsa -out dominio.key 2048
sudo chmod 600 dominio.key

# Creiamo la certificate signing request (csr)
sudo openssl req -nodes -sha256 -new -key dominio.key \
-out dominio.csr

# Creiamo il certificato
sudo openssl x509 -req -days 365 -in dominio.csr \
-signkey dominio.key -out dominio.crt
```

Abilitiamolo in nginx.conf

Reindirizziamo tutte le richieste HTTP sul protocollo HTTPS in questo modo:

```
server {
    listen 80;
    server_name dominio.com;
    return 301 https://$server_name$request_uri;
}


server {
    listen   443 ssl default_server;
    server_name dominio.com;

    ssl_certificate      /etc/nginx/ssl/dominio.crt;
    ssl_certificate_key /etc/nginx/ssl/dominio.key;
    ssl_protocols        TLSv1 TLSv1.1 TLSv1.2;
}
```

Un piccolo inconveniente

Il certificato di sicurezza del sito non è affidabile!

Hai tentato di accedere a **bmk.cippaciong.tk**, ma il server ha presentato un certificato emesso da un'autorità ritenuta non attendibile dal sistema operativo del computer. Questo potrebbe significare che il server ha generato le proprie credenziali di sicurezza, che Chrome non ritiene attendibili ai fini del riconoscimento dell'identità, o che un utente malintenzionato sta cercando di intercettare le tue comunicazioni.

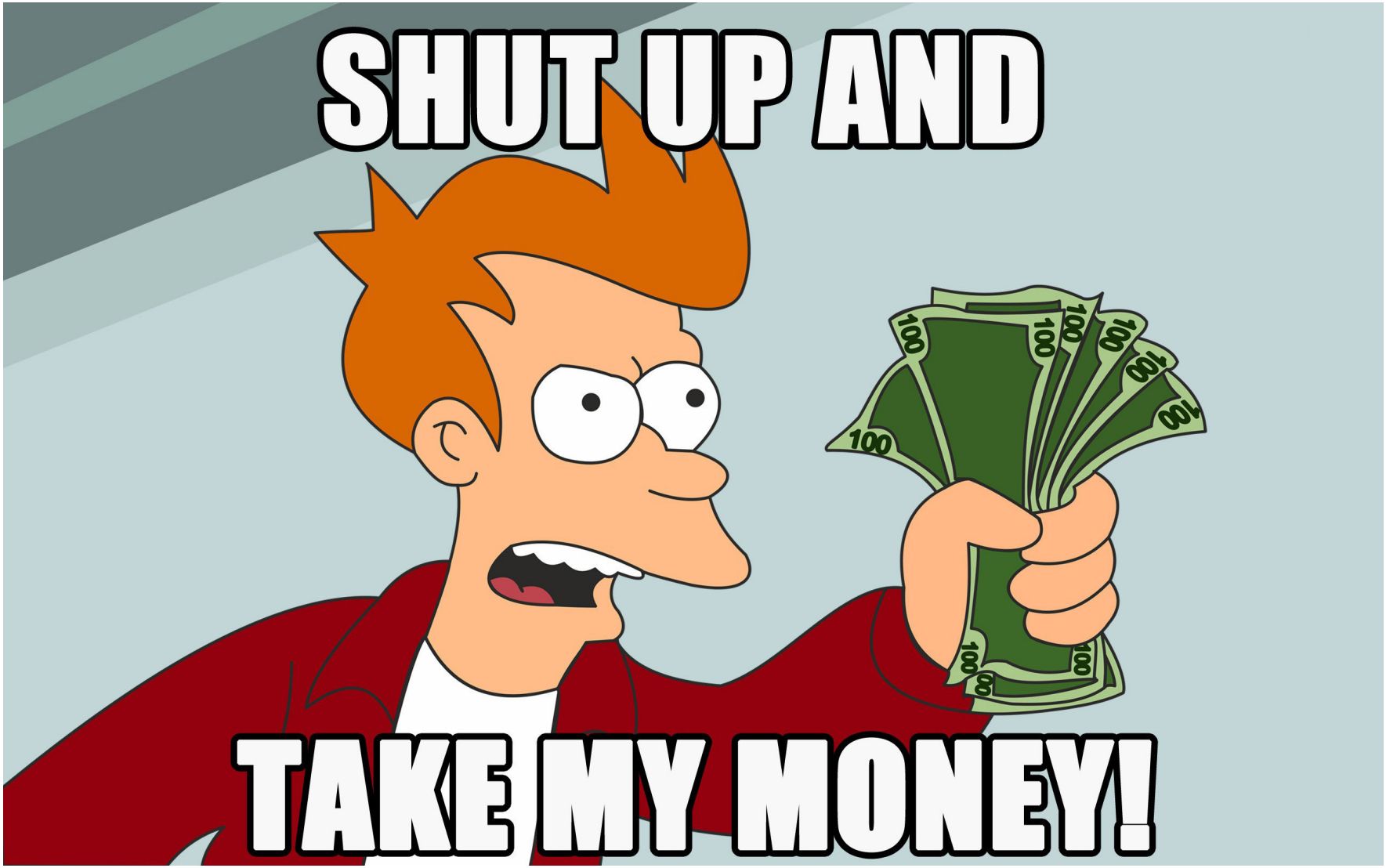
Non procedere oltre, **soprattutto** se questo avviso non è mai stato visualizzato per questo sito.

Procedi comunque | Torna nell'area protetta

Due Soluzioni

StartSSL

Due Soluzioni



Tre in realtà

Let's Encrypt

Arriving Mid-2015

mozilla

Akamai

CISCO

Demo

Outline

Introduzione
- Cos'è un Web Server
- Parliamo di NGINX

Installazione e Configurazione
- Repository e Sorgenti
- La configurazione di NGINX
- FastCGI e PHP-FPM

Sicurezza
- SSL/TLS e HTTPS
- HTTPS Hardening

Link

SSL Test

`https://www.ssllabs.com/ssltest/index.html`

HTTP Strict Transport Security (HSTS)

- È un **header HTTP** inviato dal server al client.
- Serve ad informare il client che l'accesso al sito richiesto deve essere fatto su connessione HTTPS.
- Il parametro "**max-age**" indica per quanto tempo (in secondi) è valida questa regola.

Abilitiamolo in NGINX

Abilitiamo l'header HSTS così:

```
server {
    listen   443 ssl;
    ...
    ...

    # Force HSTS
    add_header Strict-Transport-Security max-age=15768000;
}
```

Perfect Forward Secrecy (PFS)

- Impedisce di decifrare la comunicazione anche se la chiave privata del server venisse compromessa.
- La cifratura avviene per mezzo di una chiave che viene **eliminata al termine della sessione**.
- Per questo motivo la chiave viene detta **effimera**.

Abilitiamolo in NGINX

Abilitiamo PFS così:

```
server {
    listen   443 ssl;
    ...
    ...

    ssl_prefer_server_ciphers on;
    ssl_ciphers 'ECDHE-RSA-AES128-GCM-SHA256:...'
}
```

https://wiki.mozilla.org/Security/Server_Side_TLS
https://mozilla.github.io/server-side-tls/ssl-config-generator/

Link

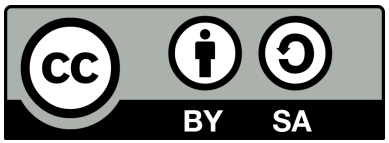
- **Getting Started**

NGINX Primer
NGINX Admin Guide
NGINXTIPS

- **HTTPS**

BetterCrypto
Why You Should Always Use HTTPS
Let's Encrypt
StartSSL Free Certificates
Hardening NGINX SSL/TSL Configuration
Strong SSL Security on nginx

License



Quest'opera è rilasciata sotto la licenza Creative Commons Attribution-Share Alike 4.0 International License. Per visualizzare una copia di questa licenza, visitare http://creativecommons.org/licenses/by-sa/4.0/ o inviare una lettera a Creative Commons, 444 Castro Street, Suite 900, Mountain View, California, 94041, USA

`https://www.poul.org/`