NGINX Web Server

Tommaso Sardelli
sardelli.tommaso[at]gmail.com

Corsi GNU/Linux Avanzati 2016
Politecnico Open unix Lab

11th May 2016

Today's topic

- What is a web server?
- How do I configure one?
- Security? (It's dangerous to go alone!)

Whoops

Microsoft

IIS

That's better!

NGINX

Even better!

Disclaimer

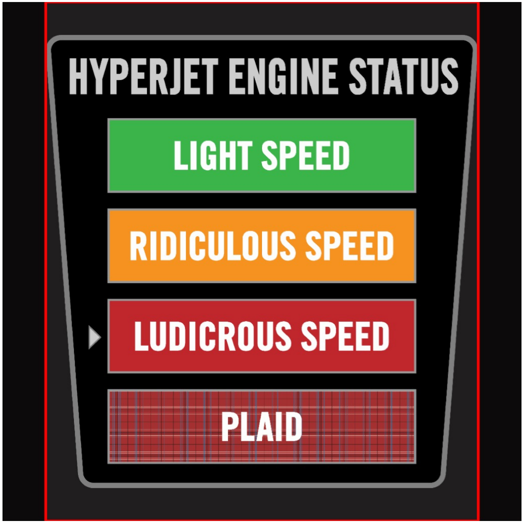

HYPERJET ENGINE STATUS
LIGHT SPEED
RIDICULOUS SPEED
LUDICROUS SPEED
PLAID

Table of Contents

Outline

What is a Web Server?

- A web server is a software that processes requests via **HTTP**.
- The primary function of a web server is to store, process and deliver web pages to ~~browsers~~ clients.
- Many generic web servers also support server-side scripting using **PHP** or **other scripting languages**.

First things first: HTTP

HTTP is the foundation of data communication for the (guess what..) **World Wide Web** (yay!).

- HTTP functions as a **request–response protocol.**
  - The client submits an HTTP **request message** to the server.
  - The server performs some functions and returns a **response message** such as HTML files or other content.
- The response contains completion **status information** about the request and may also contain requested content in its **message body**.

HTTP Request Message

An HTTP request message is composed of **three** parts:

- An HTTP **Method** and a **request URI**:
  - (GET | POST | PUT | DELETE | PATCH | ... |) /index.html HTTP/1.1
- Zero o more **Headers**:
  - Host, Connection, Cookie, Cache-Control, User-Agent, X-Forwarded-Host, many more.
- Optionally, a message **Body**:
  - Useful if you are uploading something or submitting data to an html form.

HTTP Response Message

Nothing fancy, just like a request message but instead of the HTTP method you have:

- The **Status** code (*404 not found anyone?*)
  - **1xx: Informational** - Request received, continuing process (good)
  - **2xx: Success** - The action was successfully received, understood, and accepted (good)
  - **3xx: Redirection** - Further action must be taken in order to complete the request (good)
  - **4xx: Client Error** - The request contains bad syntax or cannot be fulfilled (bad)
  - **5xx: Server Error** - The server failed to fulfill an apparently valid request (badder D:)

Enough talk, lemme see!

```
telnet www.poul.org 80
Trying 176.31.102.216...
Connected to www.poul.org.
Escape character is '^]'.
GET / HTTP/1.1
Host: www.poul.org

####################################

curl -i "https://www.poul.org"

####################################

http https://www.poul.org
(requires the httpie package)
```

Enough talk, lemme see!

```
HTTP/1.1 200 OK
Cache-Control: max-age=3, must-revalidate
Connection: keep-alive
Content-Encoding: gzip
Content-Length: 7421
Content-Type: text/html; charset=UTF-8
Date: Sun, 08 May 2016 19:21:32 GMT
Strict-Transport-Security: max-age=15768000
WP-Super-Cache: Served supercache file from PHP
X-Answer: 42
X-Fact: systemd is bloated

<html lang="it-IT">
<head [...] />
  <title>POuL Politecnico Open unix Labs</title>
```

Outline

What does 'He' have that I don't?

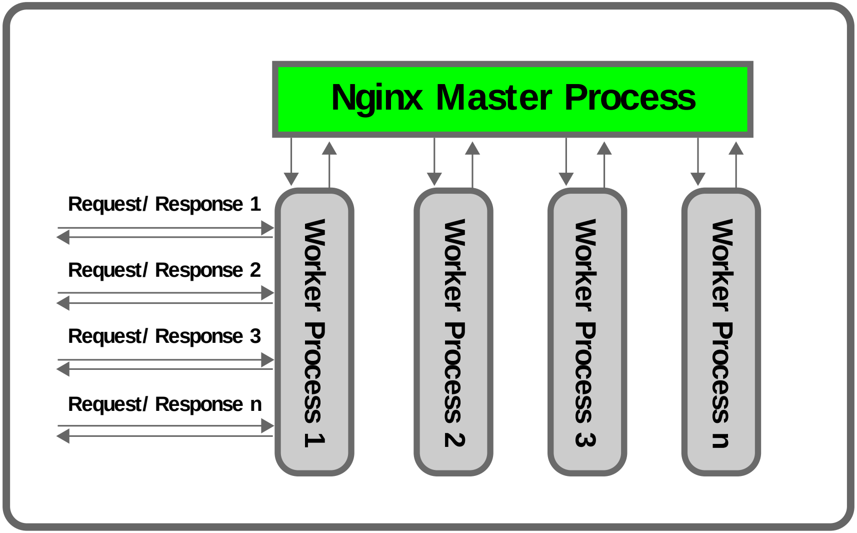
- Lightweight
- Fast
- A pleasure to configure (shame on you Apache)
- Versatile (reverse proxy, load balancer)
- (Modular)

NGINX Architecture

There are two different kinds of process:

- **Master process**: It's the main process, it runs as root and fulfills two main tasks
  - Read the configuration files.
  - Open the socket used to communicate with the worker processes.
  - (Slack off for the rest of the time).
- **Worker processes**: one or more processes run as unprivileged user (www-data on Debian)
  - They do the real hard work managing all the HTTP requests coming from thousands of clients.
  - ~~(They work out in their free time)~~ They don't have free time.

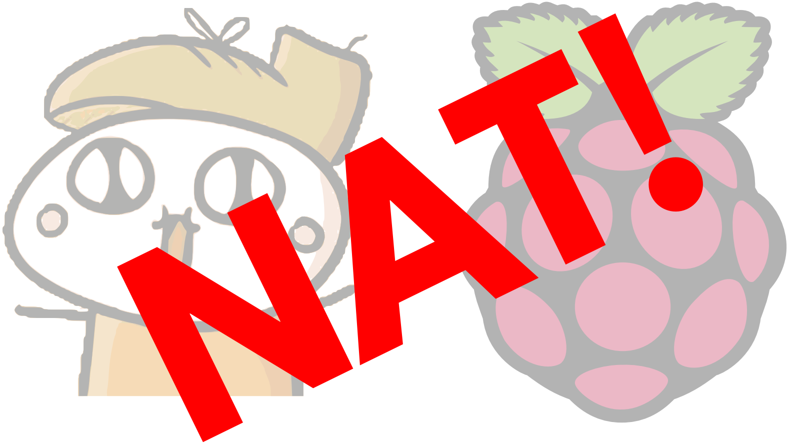
NGINX Architecture



Outline

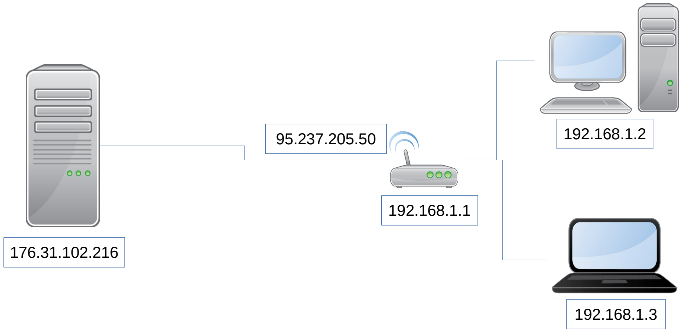
Can I try this at home?

NAT!



DDNS

One image to rule them all



To sum up

- NAT (port forwading/virtual server)
- DDNS (Duck DNS, Afraid, no-ip)
- Happy Googling :D

Outline

apt-get install

Debian Stable (Jessie at the time of writing) provides **three different flavours** of nginx (it's modular, remember?)

- nginx-light: just a small subset of core modules
- **nginx** (nginx-full): he is your man!
- nginx-extras: bloated edition

If you want to see the full comparison...
If you want the (almost) latest version, install it from the **Backports**!

Installation Complete

- Check the **installed version:**

```
sudo nginx -v
```

Installation Complete

- Check the **installed version:**

```
sudo nginx -v
```

- Get the full list of the **enabled modules:**

```
sudo nginx -V
```

Installation Complete

- Check the **installed version:**

```
sudo nginx -v
```

- Get the full list of the **enabled modules:**

```
sudo nginx -V
```

- Look for a **specific module**:

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

Outline

nginx.conf

- The good ol' days of httpd.conf and .htaccess have ended
- nginx.conf kicks in
- And your days get even better
- (Maybe) ^^'

Contexts

- nginx.conf is divided into ~~five~~ **four** ~~contexts~~ **blocks**: (main), events(1), http(1), server(n) and location(n).
- There is a **hierarchy**: the **events** block is standalone, **http** contains **server** blocks, a server block contains **location** blocks.
- Directives defined in an higher block (like http) affect all the underlying blocks (server and location).
- **Pro Tip**: apply directives in the highest context available.

nginx.conf stripped-down

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

events {
    worker_connections 128;
}

http {

    server {

        location {
         }
    }
}
```

Every time you do this, a kitten dies

```
http {
    . . .

    server {
        . . .

        location / {
            root /var/www/html;
            . . .
        }

        location /another {
            root /var/www/html;
            . . .
        }
    }
}
```

VirtualHosts :P

```
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 fluffykittens.it www.fluffykittens.it;
        ...
    }
}
```

location blocks

- When we are in a location context we are usually dealing with files and folders.
- Location diectives allow us to tell NGINX what to do when a **specific resource** is requested.
- Such resource can be targeted using an **exact path**
  - In this case we will prefix the uri with **“=”**
- Alternatively we can rely on **regex matching**
  - **"~"** prefix for **case sensitive** matching
  - **"~*"** prefix for **case insensitive** matching

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

Examples or GTFO!

- When I visit http://example.org/downloads I want a list of all the files in that folder

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

Examples or GTFO!

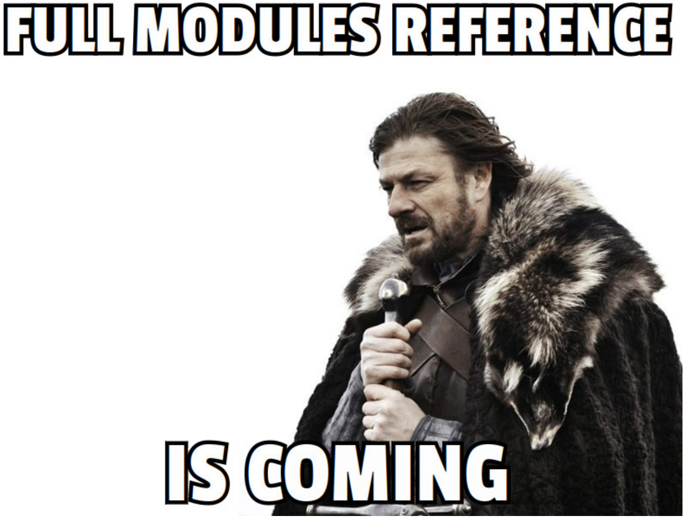
- When I visit http://example.org/downloads I want a list of all the files in that folder

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

- We don't want our users to execute scripts from write accessbile folders, RIGHT?

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

Make yourself comfortable

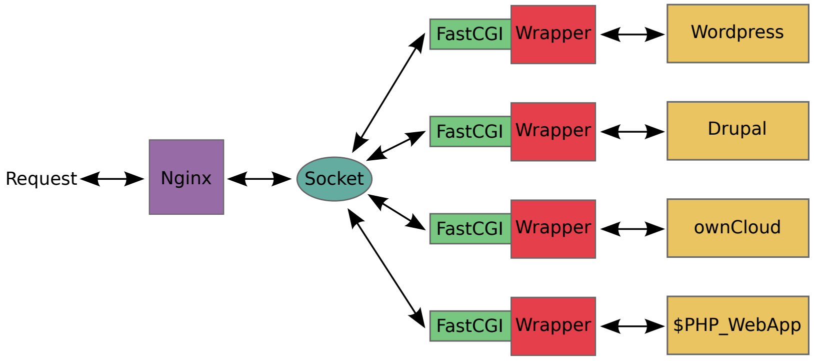


Just Kidding

Demo

Outline

PHP-FPM Architecture



php-fpm.conf

- sudo apt-get install **php5-fpm**
- sudo vim/emacs/atom/(flame?) **/etc/php5/fpm/pool.d/www.conf**
- **listen = /var/run/php5-fpm.sock**
- sudo service php5-fpm restart

No way! I want PHP 7

- sudo apt-get install **php7.0-fpm** (https://packages.sury.org/php/)
- sudo vim (we have a winner) **/etc/php/7.0/fpm/pool.d/www.conf**
- **listen = /run/php/php7.0-fpm.sock**
- sudo service php7.0-fpm restart

Demo

Demo

Outline

“To infinity… and beyond!”

- Ruby: Rails/Sinatra/Puma
- Python: Flask/Tornado/Django (the D is silent)
- JavaScript: Node.js/Ghost
- Anything: Transmission/Syncthing/ympd/...

Always the same pattern

- A service running behind some port (8000, 8080, 8384, 9091, etc.)
- You want to access it without opening all those ports in your firewall
- You want advanced settings:
  - Authentication
  - SSL/TLS

Demo

Demo

Outline

It's a conspiracy!

HTTPS? Oh yes, the green lock! :|

SSL/TLS in a nutshell:

- Choose a fast symmetric cipher (like AES). This is called, well, the **cipher**.
- Choose a **random key** for that cipher. This is called the session key.
- Encrypt that key using **RSA** (public key crypto) and send it to the person you're communicating with.
- Then you both have the same AES key, and can **encrypt all your communications back and forth after that**.
- The NSA is sad :(

Alice is suspicious

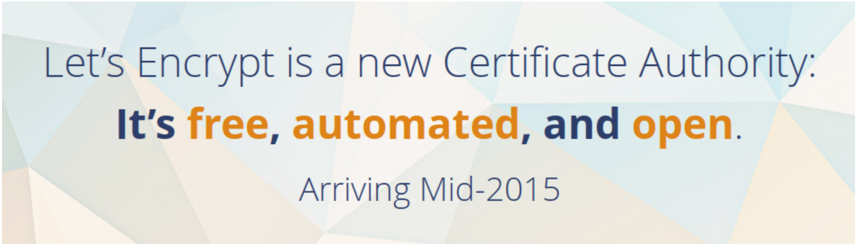
Everything is encrypted, awesome, but is Bob... well, Bob?

- A **digital certificate** is an electronic document used to **prove ownership of a public key**.
- The certificate includes information about the **key**, its **owner's identity**, and the **digital signature of a Certification Authority.**
- A Certification Authority(CA) is an entity that **issues digital certificates** and verifies that the certificate's content is correct.

Arya

Outline

Let's Encrypt will be released any second now...

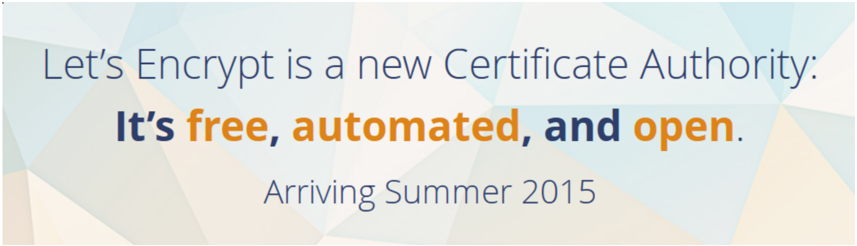


Let's Encrypt is a new Certificate Authority:

It's free, automated, and open.

Arriving Mid-2015

...any second now...

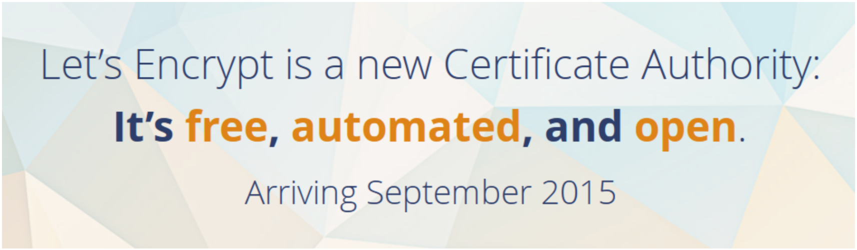


Let's Encrypt is a new Certificate Authority:

It's free, automated, and open.

Arriving Summer 2015

...aaaaany second now...

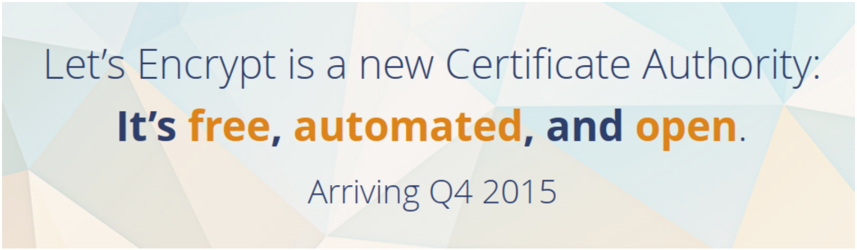


Let's Encrypt is a new Certificate Authority:

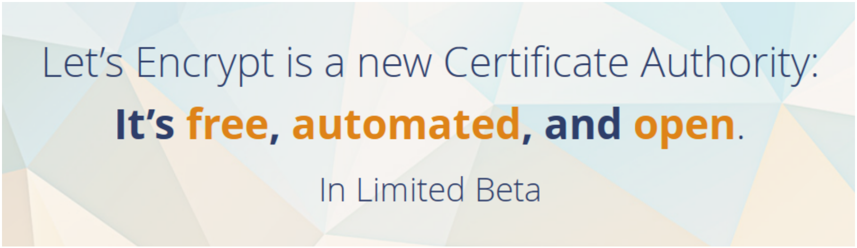
It's free, automated, and open.

Arriving September 2015

…now…


Let's Encrypt is a new Certificate Authority:
It's free, automated, and open.
Arriving Q4 2015

...

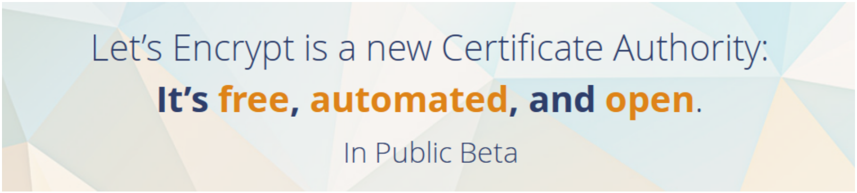


Let's Encrypt is a new Certificate Authority:

It's free, automated, and open.

In Limited Beta

...

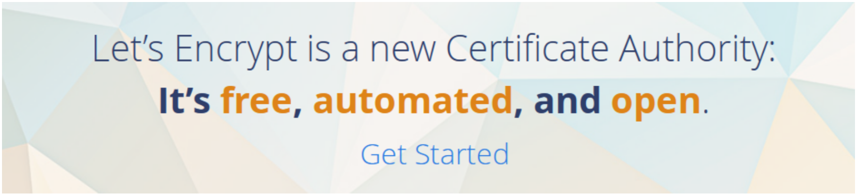


Let's Encrypt is a new Certificate Authority:

It's free, automated, and open.

In Public Beta

Thank you...



Let's Encrypt is a new Certificate Authority:

It's free, automated, and open.

So, we were saying, Let's Encrypt

Let's Encrypt

Features - sounds good!

- Free
- Automatic
- Secure
- Transparent
- Open
- Cooperative

Under the hood

When you run the letsencrypt client a few tasks are performed
https://letsencrypt.org/how-it-works/

- Domain Validation (DNS or HTTP).
  - Provisioning a DNS record under example.com
  - Provisioning an HTTP resource under a well-known URI on https://example.com/
- Certificate Issuance.
- Repeat every 2/3 months (yes, a script would be helpful)

Plugins

https://letsencrypt.readthedocs.io/en/latest/using.html

- **apache**: Automates obtaining and installing a cert with Apache 2.4 on Debian-based distributions.
- **webroot**: Obtains a cert by writing to the webroot directory of an already running webserver.
- **standalone**: Uses a “standalone” webserver to obtain a cert. Requires port 80 or 443 to be available.
- **manual**: Helps you obtain a cert by giving you instructions to perform domain validation yourself.
- **nginx**: Very experimental and not included in letsencrypt-auto. **D:**

Demo

Demo

Outline

Meet your new best friend (SSL Test)

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

HTTP Strict Transport Security (HSTS)

- It is an **HTTP header** sent from the server to the client.
- Such header informs the client that HTTPS is availbale for the requested website.
- The "**max-age**" parameters sets the validity of this information (in seconds).

Enable HSTS in NGINX

Enabling HSTS is as simple as adding a common HTTP header:

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

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

Perfect Forward Secrecy (PFS)

- Let's say someone **intercepts and stores** all our encrypted communications. I know, who would *ever* do that? (*cough*)
- If the private key is compromised/deciphered all the previous communications could be unencrypted and read.
- Solution:
  - Use a **new key for each session**!
  - Call that key **"ephimeral"**.

PFS? Pretty please... with sugar on top.

Just use the right cipher

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

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

NGINX Links

- **Getting Started**

NGINX Pitfalls
NGINX Admin Guide
NGINX Primer
NGINXTIPS
NGINX Doc and Modules Reference
Understanding Nginx Server and Location Block Selection Algorithms
Understanding the Nginx Configuration File Structure and Configuration Contexts

HTTPS Links

- **HTTPS**

BetterCrypto
Mozilla Config Generator
Cipherli.st
Why You Should Always Use HTTPS
Hardening NGINX SSL/TSL Configuration
Strong SSL Security on nginx

License

/media/Dati/Syncthing/slide_nginx/2015

https://www.poul.org/