INTRODUCTION TO

TURBO BOOST YOUR WEB SERVER!

GABRIEL CÁNEPA

Java Code Geeks
Java 2 Java Developers Resource Center

Introduction to Nginx

Contents

Preface

Nginx is an open source HTTP and reverse proxy server, as well as a mail proxy server, load balancer, and HTTP cache. The nginx project started with a strong focus on high concurrency, high performance and low memory usage. It runs on Linux, BSD variants, Mac OS X, Solaris, AIX, HP-UX, as well as on other *nix flavors. It also has a proof of concept port for Microsoft Windows. According to Netcraft nginx served or proxied 17.65% busiest sites in March 2014.

This ebook will introduce you to the magic of nginx. You will learn to install and configure nginx for a variety of software platforms and how to integrate it with Apache.

Additionally, you will get involved with more advanced concepts like Load Balancing, SSL configuration and Websockets proxying.

About the Author

Gabriel's areas of expertise and interest are Linux system administration, shell scripting, database administration (SQL Server, MySQL, Oracle 11g), object-oriented and procedural programming (Python and PHP), desktop applications (C#, Visual Basic, Excel with VBA) and web development (jQuery, HTML5, CSS3, PHP).

He has also been working as a Level-1 TSR (Technical Support Representative) supporting onsite the startup and ongoing operation of the WMS in a major multinational company, running Red Hat Enterprise Linux and Oracle 11g as RDBMS.

Chapter 1

Nginx installation on Linux

1.1 Introduction

Nginx (pronounced "engine x") is - in few words- a small, powerful, and scalable web/proxy server. According to a recent survey performed by Netcraft, Nginx powers more than 15% of the web, equating to 111,680,078 sites (Sept. 2013 Web Server Survey), including giants like *Netflix* and *Wordpress.com*.

Nginx is available under the *Simplified BSD* License, an open source license, and can be installed either from online repositories or from sources. In this article we will cover the installation of Nginx in Debian, Ubuntu, and CentOS using both methods. It is important to note that the repositories are often somewhat out-of-date. If we want the latest features and bugfixes, it's recommended to build from source or use packages directly from *nginx.org*.

1.2 Installation of Nginx in Debian Wheezy 7.2 using online repositories

Using `aptitude`, the high-level interface to the Debian GNU/Linux package manager, we can check the list of packages related to Nginx (see Fig. 1.1). However, it is advisable to run the command `aptitude update` first in order to see an updated list of available packages.

```
root@debian:~# aptitude search nginx
p   nginx                                   - small, powerful, scalable web/proxy server
p   nginx-common                            - small, powerful, scalable web/proxy server - common files
p   nginx-doc                               - small, powerful, scalable web/proxy server - documentation
p   nginx-extras                            - nginx web/proxy server (extended version)
p   nginx-extras-dbg                        - nginx web/proxy server (extended version) - debugging symbols
p   nginx-full                              - nginx web/proxy server (standard version)
p   nginx-full-dbg                          - nginx web/proxy server (standard version) - debugging symbols
p   nginx-light                             - nginx web/proxy server (basic version)
p   nginx-light-dbg                         - nginx web/proxy server (basic version) - debugging symbols
p   nginx-naxsi                             - nginx web/proxy server (version with naxsi)
p   nginx-naxsi-dbg                         - nginx web/proxy server (version with naxsi) - debugging symbols
p   nginx-naxsi-ui                          - nginx web/proxy server - naxsi configuration front-end
root@debian:~#
```


Figure 1.1: screenshot

(By the way, the letter "p" in the first column indicates that no trace of the package currently exists on the system).

If we can't decide which package to install, `aptitude search` -followed by a package name such as `aptitude search nginx`- is our friend and will help us to make up our minds as to which one is right for us. Based on the description of each package listed above, we will proceed with the installation of `nginx-full` (see Fig. 1.2). It is important to note that the

description of each package lists the additional modules that are available by default through the installation using repositories, but we'll cross that bridge when we get there - later in this tutorial.

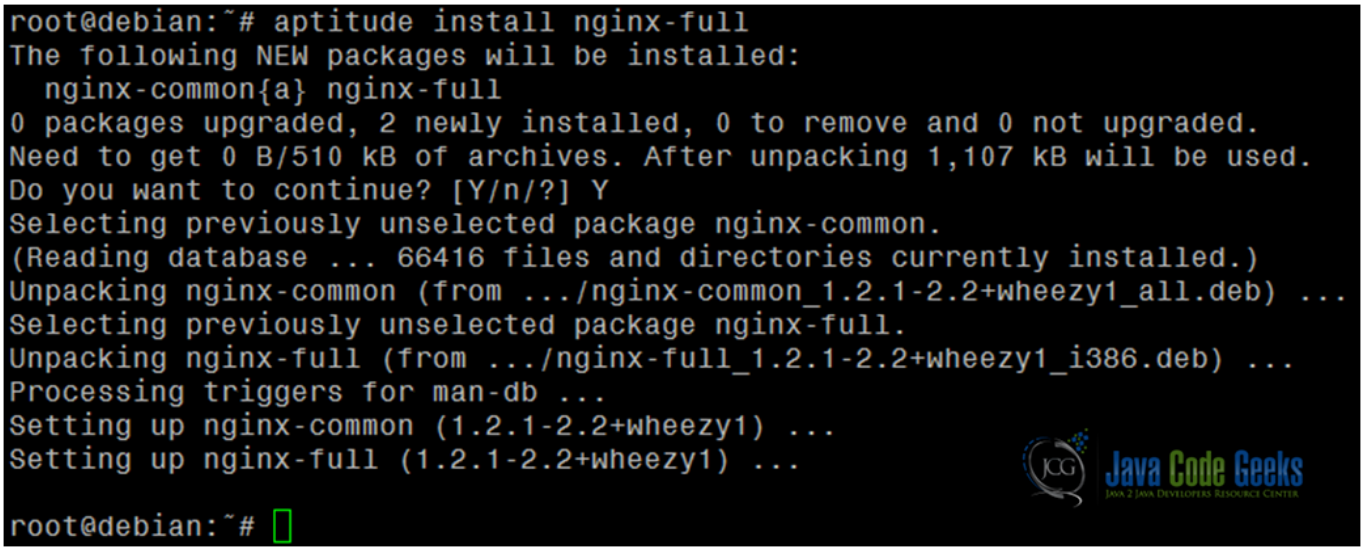

Figure 1.2: Installation of nginx-full in Debian Wheezy 7.2 using repositories

At this point, Nginx has been installed but it is not running yet. We will set things in motion with `service nginx start` and then we will be able to see its start page in a web browser (see Fig. 1.3).

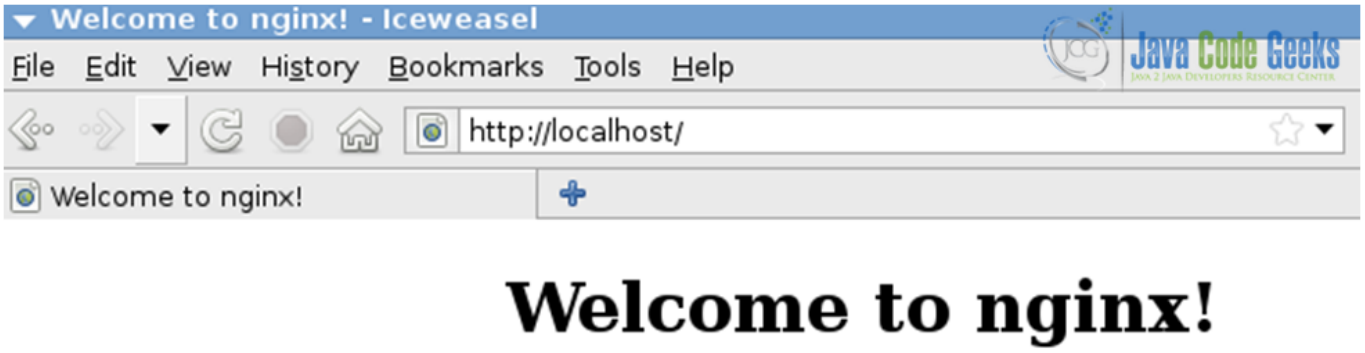

Figure 1.3: Nginx start page

Then the following command will show us the version of Nginx that we have just installed (see Fig. 1.4):

Figure 1.4: Nginx v1.2.1 installed from repositories

However, as of today the latest version of Nginx is 1.5.6, while version 1.2.1 is dated June 05, 2012 (NGINX Download Page). That goes to show that we need to install the program from sources if we want to use the latest, up-to-date version.

1.3 Installation of Nginx in Debian Wheezy 7.2 from sources

Please note that the following instructions represent the default way of building a package from scratch in Linux, so be advised that the regular installation procedure using `./configure`, `make`, and `make install` makes it harder for you to uninstall the package later if you don't want to use it anymore because there is no way for the system to keep track of all the files that were added / modified during the process.

In summary, you should have a valid reason (a very good one, actually!) for compiling a package using the method mentioned above. There are a couple of valid reasons why you may want to do so, though. The most common reason is to install a more recent version of a certain package in your system; another reason is to compile in order to add support for a particular feature.

If you build and install a `.deb` or a `.rpm` file, then the corresponding package manager (`aptitude` / `apt-get` or `yum`, respectively) will be aware of the presence of the package and it will make sure that you do not overwrite the files of a previously installed package. On the other hand, the `make install` command will overwrite anything that gets in its way. We will discuss later the other options that we have when we DO have a valid reason to compile and install a package from source.

Now that we have decided that we will not settle for less than the latest version of Nginx, we need to follow these steps to download the compressed tarball from http://nginx.org/download/ and uncompress it before proceeding with the build per se.

- Download the tarball: `wget http://nginx.org/download/nginx-1.5.6.tar.gz`
- Uncompress it: `tar xvzf nginx-1.5.6.tar.gz`
- Go to the directory that was automatically created during the last step: `cd nginx-1.5.6`

And then:

- `./configure` (add the `--help` option if you want to list all the configuration options). The output of `./configure` shows the directory where Nginx will be installed (`/usr/local/nginx`, see Fig. 1.5)

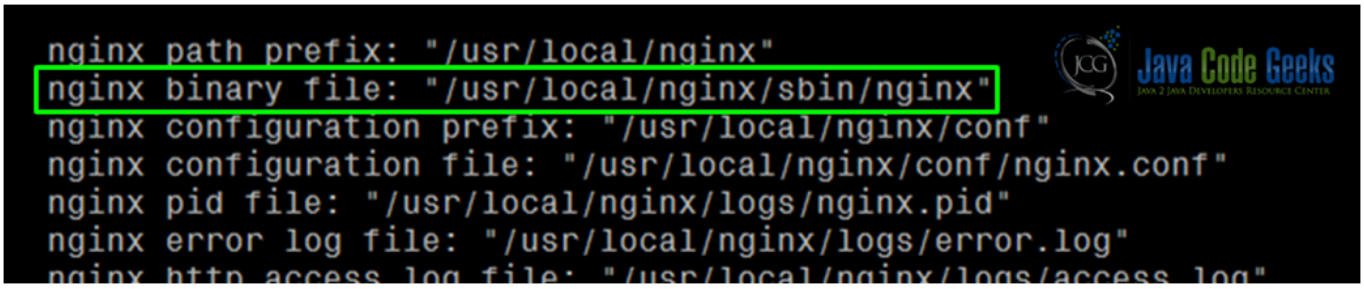

Figure 1.5: Nginx installation path

- `make`
- `make install`

Even when the installation is complete, the directory where Nginx is located has not yet been added to the PATH environment variable (see Fig. 1.6)

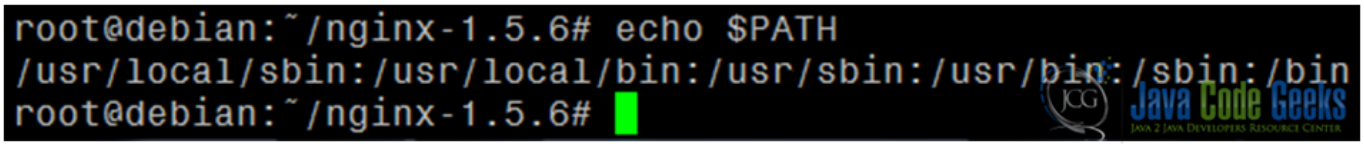

Figure 1.6: The PATH variable (before)

Now let's add the /usr/local/nginx/sbin directory to the PATH variable and let's check the version of Nginx that we have just installed from sources (see Fig. 1.7).

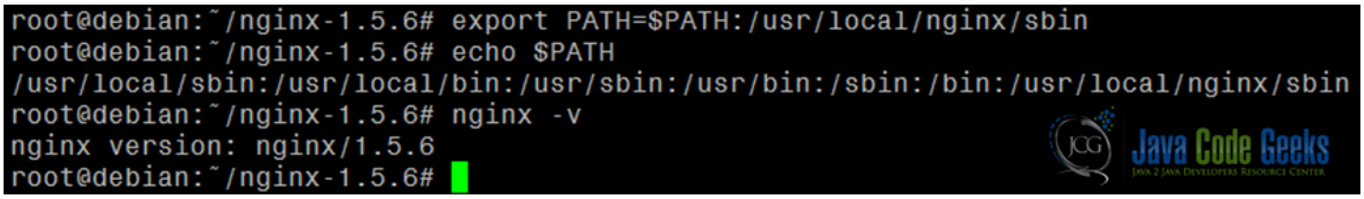

Figure 1.7: The PATH variable (after) and Nginx v1.5.6 installed from sources

NOTE: During the configure process, it is possible that the system will complain about missing libraries (see Fig. 1.8). In that case we can either installed the packages that provide such libraries (in our case, `libpcre3-dev` and `zlib1g-dev`) or ignore them during configure.

```
checking for PCRE library ... not found
checking for PCRE library in /usr/local/ ... not found
checking for PCRE library in /usr/include/pcre/ ... not found
checking for PCRE library in /usr/pkg/ ... not found
checking for PCRE library in /opt/local/ ... not found

./configure: error: the HTTP rewrite module requires the PCRE library.
You can either disable the module by using --without-http_rewrite_module
option, or install the PCRE library into the system, or build the PCRE library
statically from the source with nginx by using --with-pcre=<path> option.

root@debian:~/nginx-1.5.6#
```

Figure 1.8: Missing libraries

1.4 Installation of Nginx on Ubuntu 12.04 LTS

Even though the latest version of Ubuntu is 13.10 (codename *Saucy Salamander*, released on October 17th, 2013) we have chosen to perform the installation on Ubuntu 12.04 LTS (codename *Precise Pangolin*) due to the extended support provided by Canonical until April 2017.

We will proceed to update sources with `sudo aptitude update` and then install *nginx-full* from the distribution's online repositories. The `sudo` keyword must be added as in Ubuntu the root account is disabled by default (see Fig. 1.9). Other than that, the installation will not differ significantly than the same procedure that we just performed in Debian. The same applies to the installation from source code.

```
gacanepa@ubuntuOS:~$ sudo aptitude install nginx-full
The following NEW packages will be installed:
  nginx-common{a} nginx-full
0 packages upgraded, 2 newly installed, 0 to remove and 72 not upgraded.
Need to get 400 kB of archives. After unpacking 1,101 kB will be used.
Do you want to continue? [Y/n/?]
```

Figure 1.9: Installing package nginx-full in Ubuntu from repositories

However, we can see that the available version is even more outdated in this case (see Fig. 1.10).

```
gacanepa@ubuntuOS:~$ nginx -v
nginx version: nginx/1.1.19
gacanepa@ubuntuOS:~$
```

Figure 1.10: Nginx version installed in Ubuntu from repos

As before, we will remove (uninstall) all packages related to *nginx* before proceeding with the installation from source code (see Fig. 1.11).

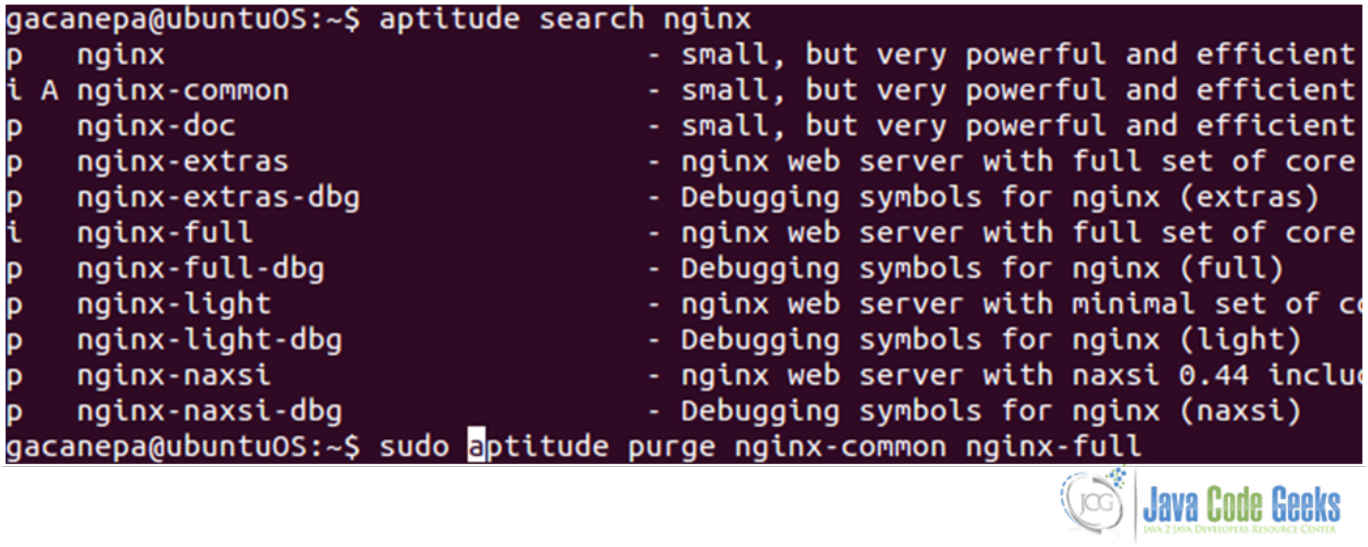

Figure 1.11: screenshot

As before, after installing Nginx from source code, we have the latest and up-to-date version of the package (see Fig. 1.12):

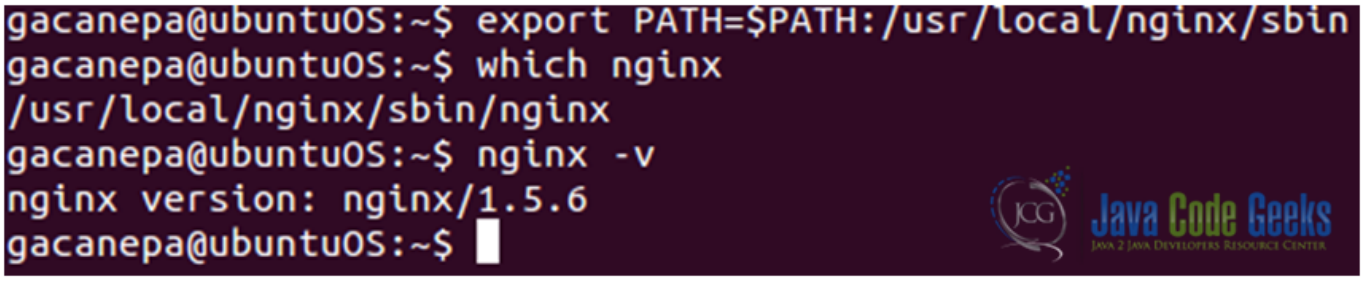

Figure 1.12: screenshot

However, when it comes to starting Nginx -and just as what would have happened had we tried to do so in Debian- we will most likely get a **nginx: unrecognized service** error message (see Fig. 1.13).

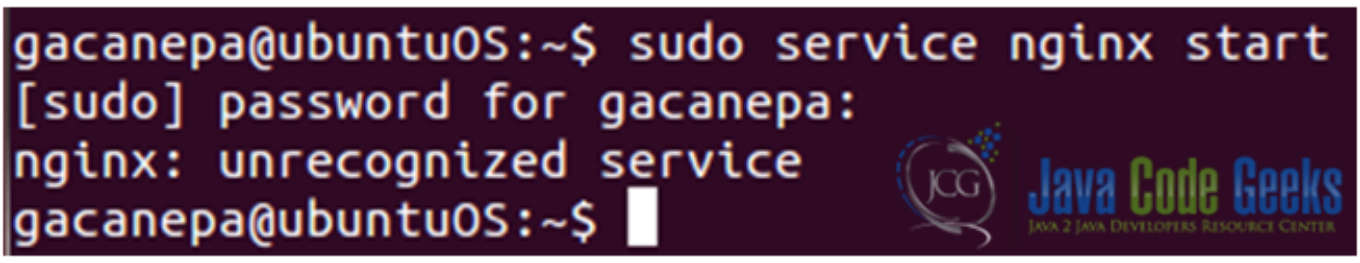

Figure 1.13: screenshot

This is due to the fact that we have installed the package from sources and therefore the startup script has not yet been put in place. In this case, we can either start the *nginx* daemon by running the main executable from the command line using its full

path (/usr/local/nginx/sbin/nginx) or by writing a script that will take care of the job for us - of course this last option represents the best choice as we want to be able to use all of the usual arguments (`start, stop, restart, reload`, and so on). Also, we can just use one of the startup scripts provided along with this tutorial (which we can also modify to better suit our needs).

Once we have added the script in the `/etc/init.d` directory (and named it ***nginx***), we need to change the *DAEMON* variable to point to `/usr/local/nginx/sbin/nginx` and include the installation directory (`/usr/local/nginx/sbin`) in the *PATH* variable (see Figs. 1.14 and 1.15):

```
13 PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
14 DAEMON=/usr/sbin/nginx
15 NAME=nginx
16 DESC=nginx
```

Figure 1.14: Nginx startup script (before)

```
13 PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/nginx/sbin
14 DAEMON=/usr/local/nginx/sbin/nginx
15 NAME=nginx
16 DESC=nginx
```

Figure 1.15: Nginx startup script (after)

Then we can run the script as follows (see Fig. 1.16):

```
gacanepa@ubuntuOS:~$ sudo netstat -npltu | grep nginx
gacanepa@ubuntuOS:~$ ps -ef | grep nginx | grep -v grep
gacanepa@ubuntuOS:~$ sudo service nginx start
Starting nginx: nginx.
gacanepa@ubuntuOS:~$ sudo netstat -npltu | grep nginx
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN      1931/nginx
gacanepa@ubuntuOS:~$ ps -ef | grep nginx | grep -v grep
root      1931     1  0 15:53 ?        00:00:00 nginx: master process /usr/local/nginx/sbin/nginx
nobody    1932  1931  0 15:53 ?        00:00:00 nginx: worker process
gacanepa@ubuntuOS:~$
```

Figure 1.16: Nginx listening on port 80 with PID 1931

Also, we need to make sure that the nginx.conf file "knows" where to find the PID of Nginx. Uncomment the following line in `nginx.conf` (see Fig. 1.18, most likely it will be found in `/usr/local/nginx/conf`) and change the path to the one that the startup script indicates (see Fig. 1.17):

```
start-stop-daemon --start --quiet --pidfile /var/run/$NAME.pid \
    --exec $DAEMON -- $DAEMON_OPTS || true
echo "$NAME."
;;
```

Figure 1.17: The file `/var/run/$NAME.pid` (where $NAME=nginx) contains the current PID of Nginx

```
pid         /var/run/nginx.pid;
```

Figure 1.18: screenshot

1.5 Use of the `checkinstall` package to keep track of all the files created or modified by an installation script

The `checkinstall` package (see Fig. 1.19) keeps track of all the files created or modified during the installation process. It also creates and installs a package (`.deb` or `.rpm`) that is compatible with your package manager (see Figs. 16 and 17), which can be used later to completely remove the package if you don't need it anymore. Check the attached man page for details on its use.

```
gacanepa@ubuntuOS:~/nginx-1.5.6$ sudo aptitude install checkinstall
[sudo] password for gacanepa:
The following NEW packages will be installed:
  build-essential{a} checkinstall dpkg-dev{a} fakeroot{a} g++{a} g++-4.6{a}
  libalgorithm-diff-perl{a} libalgorithm-diff-xs-perl{a}
  libalgorithm-merge-perl{a} libdpkg-perl{a} libstdc++6-4.6-dev{a}
0 packages upgraded, 11 newly installed, 0 to remove and 72 not upgraded.
Need to get 9,326 kB of archives. After unpacking 27.7 MB will be used.
Do you want to continue? [Y/n/?] Y
```

Figure 1.19: Installation of the checkinstall package

```
gacanepa@ubuntuOS:~/nginx-1.5.6$ sudo checkinstall -D make install

checkinstall 1.6.2, Copyright 2009 Felipe Eduardo Sanchez Diaz Duran
            This software is released under the GNU GPL.


The package documentation directory ./doc-pak does not exist.
Should I create a default set of package docs?  [y]: y

Preparing package documentation...OK
```

Figure 1.20: Creating a .deb file with checkinstall and make install (I)

```
Done. The new package has been installed and saved to

/home/gacanepa/nginx-1.5.6/nginx_1.5.6-1_i386.deb

You can remove it from your system anytime using:

     dpkg -r nginx
```

Figure 1.21: Creating a .deb file with checkinstall and make install (II)

When we run `aptitude search nginx`, the package that we just installed will appear with the comments that we added as description (see Fig. 1.22):

```
gacanepa@ubuntuOS:~/nginx-1.5.6$ aptitude search nginx
i   nginx                                       - Nginx v1.5.6
p   nginx-common                                - small, but very powerful and efficient web server (co
p   nginx-doc                                   - small, but very powerful and efficient web server (do
```

Figure 1.22: screenshot

1.6 Installing Nginx in CentOS 6.4

1.6.1 From repositories

- Download and install nginx yum configuration file from http://nginx.org/packages/centos/6. Make sure you select the appropriate architecture; "noarch" is a safe choice:
  - Download: `wget http://nginx.org/packages/centos/6/noarch/RPMS/nginx-release-centos-6-0.el6.ngx.noarch.rpm`

  - Install: `rpm -ivh nginx-release-centos-6-0.el6.ngx.noarch.rpm`
- Install nginx: `yum install nginx`
- Start nginx: `service nginx start`

1.6.2 From sources

Follow the same procedure as for Debian and Ubuntu.

1.6.3 Enabling modules

According to its wiki (NGINX Wiki Page), Nginx modules must be selected during compile as run-time selection of modules is not currently supported. A full summary of the compile-time options, including optional modules, can be found in the provided configure script by running `./configure --help`. Unfortunately, if we have installed Nginx installed and want to add a certain module, we will have to uninstall it and the recompile it with support for the desired module.

For a list of the standard HTTP modules, refer to Table 1 (NGINX Modules):

Table 1.1: Standard HTTP modules

Name	Description	Version	configure argument to disable
HTTP Core	Control ports, locations, error pages, aliases, and other essentials.		--without-http
Access	Allow/deny based on IP address.		--without-http_access_module
Auth Basic	Basic HTTP authentication.		--without-http_auth_basic_module
Auto Index	Generates automatic directory listings.		--without-http_autoindex_module
Browser	Interpret "User-Agent" string.	0.4.3	--without-http_browser_module
Charset	Recode web pages.		--without-http_charset_module
Empty GIF	Serve a 1x1 image from memory.	0.3.10	--without-http_empty_gif_module
FastCGI	FastCGI Support.		--without-http_fastcgi_module
Geo	Set config variables using key/value pairs of IP addresses.	0.1.17	--without-http_geo_module
Gzip	Gzip responses.		--without-http_gzip_module
Headers	Set arbitrary HTTP response headers.		
Index	Controls which files are to be used as index.		
Limit Requests	Limit frequency of connections from a client.	0.7.20	--without-http_limit_req_module
Limit Conn	Limit concurrent connections based on a variable.		--without-http_limit_conn_module
Log	Customize access logs.		

Table 1.1: (continued)

Map	Set config variables using arbitrary key/value pairs.	0.3.16	--without-http_map_module
Memcached	Memcached support.		--without-http_memcached_module
Proxy	Proxy to upstream servers.		--without-http_proxy_module
Referer	Filter requests based on Referer header.		--without-http_referer_module
Rewrite	Request rewriting using regular expressions.		--without-http_rewrite_module
SCGI	SCGI protocol support.	0.8.42	--without-http_scgi_module
Split Clients	Splits clients based on some conditions	0.8.37	--without-http_split_clients_module
SSI	Server-side includes.		--without-http_ssi_module
Upstream	For load-balancing.		--without-http_upstream_ip_hash_module (ip_hash directive only)
User ID	Issue identifying cookies.		--without-http_userid_module
uWSGI	uWSGI protocol support.	0.8.40	--without-http_uwsgi_module
X-Accel	X-Sendfile-like module.		

For a list of option HTTP modules, refer to Table 2:

Table 1.2: Optional HTTP modules

Name	**Description**	**Version**	**configure argument to enable**
Addition	Append text to pages.		--with-http_addition_module
Auth Request	Implements client authorization based on the result of a subrequest.	1.5.4	--with-http_auth_request_module
Degradation	Allow to return 204 or 444 code for some locations on low memory condition.	0.8.25	--with-http_degradation_module
Embedded Perl	Use Perl in Nginx config files.	0.3.21	--with-http_perl_module
FLV	Flash Streaming Video	0.4.7	--with-http_flv_module
GeoIP	Creates variables with information from the MaxMind GeoIP binary files.	0.8.6, 0.7.63	--with-http_geoip_module
Google Perftools	Google Performance Tools support.	0.6.29	--with-google_perftools_module
Gzip Precompression	Serves precompressed versions of static files.	0.6.23	--with-http_gzip_static_module
Gunzip	On-the-fly decompressing of gzipped responses.	1.3.6	--with-http_gunzip_module
Image Filter	Transform images with Libgd	0.7.54	--with-http_image_filter_module

Table 1.2: (continued)

MP4	Enables mp4 streaming with seeking ability.	1.1.3, 1.0.7	--with-http_mp4_module
Random Index	Randomize directory indexes.	0.7.15	--with-http_random_index_module
Real IP	For using nginx as backend	0.3.8	--with-http_realip_module
Secure Link	Protect pages with a secret key.	0.7.18	--with-http_secure_link_module
SSL	HTTPS/SSL support.		--with-http_ssl_module
Stub Status	View server statistics.	0.1.18	--with-http_stub_status_module
Substitution	Replace text in pages		--with-http_sub_module
WebDAV	WebDAV pass-through support.	0.3.38	--with-http_dav_module
XSLT	Post-process pages with XSLT.	0.7.8	--with-http_xslt_module

For a list of mail modules, refer to Table 3:

Table 1.3: Mail modules

Name	Description	configure argument
Mail Core	Core parameters for mail module.	--with-mail
POP3	POP3 settings.	--without-mail_pop3_module
IMAP	IMAP settings.	--without-mail_imap_module
SMTP	SMTP settings.	--without-mail_smtp_module
Auth HTTP	Use Nginx to authenticate mail services.	
Proxy	Nginx can proxy IMAP, POP3, and SMTP protocols.	
SSL	This module ensures SSL/TLS support for POP3/IMAP/SMTP.	--with-mail_ssl_module

As an example, we will recompile Nginx to include the Addition module (see Fig. 1.23):

```
gacanepa@ubuntuOS:~$ cd nginx-1.5.6
gacanepa@ubuntuOS:~/nginx-1.5.6$ sudo ./configure --with-http_addition_module
```

Figure 1.23: Recompiling Nginx with the Addition module

1.7 Adding Nginx as a system service

- **Debian / Ubuntu:** `update-rc.d -f nginx defaults` (use `sudo` on Ubuntu)

Note: If any files /etc/rcrunlevel.d/[SK]??name already exist then update-rc.d does nothing. The program was written this way so that it will never change an existing configuration, which may have been customized by the system administrator. The program will only install links if none are present, i.e., if it appears that the service has never been installed before.

- **CentOS: chkconfig nginx on**

Chapter 2

Nginx Configuration Guide

2.1 Configuration file syntax

According to Merriam-Websters online dictionary, the word syntax represents "the way in which words are put together to form phrases, clauses, or sentences". Of course that definition, as it is taken from a dictionary, is aimed at English (or actually any other language) students. So what about IT people coming from all backgrounds, languages, and countries? The change in context does not alter the meaning of the term very significantly. In particular, the syntax of a configuration file must be correct, as seen by the program that will parse it, in order for it to work efficiently.

We will learn to understand and how to write (and / or modify) a configuration file for Nginx under Ubuntu 12.04 LTS. That will be accomplished by specifying a set of values that will define the behavior of the web server.

2.2 Configuration directives

By default, the behavior of Nginx is defined by its main configuration file, which is located (as seen before and unless we have modified this setting) at `/usr/local/nginx/conf/nginx.conf`. This file is composed of a list of directives organized in a logical structure that is very easy to follow and understand (see Fig. 2.1).

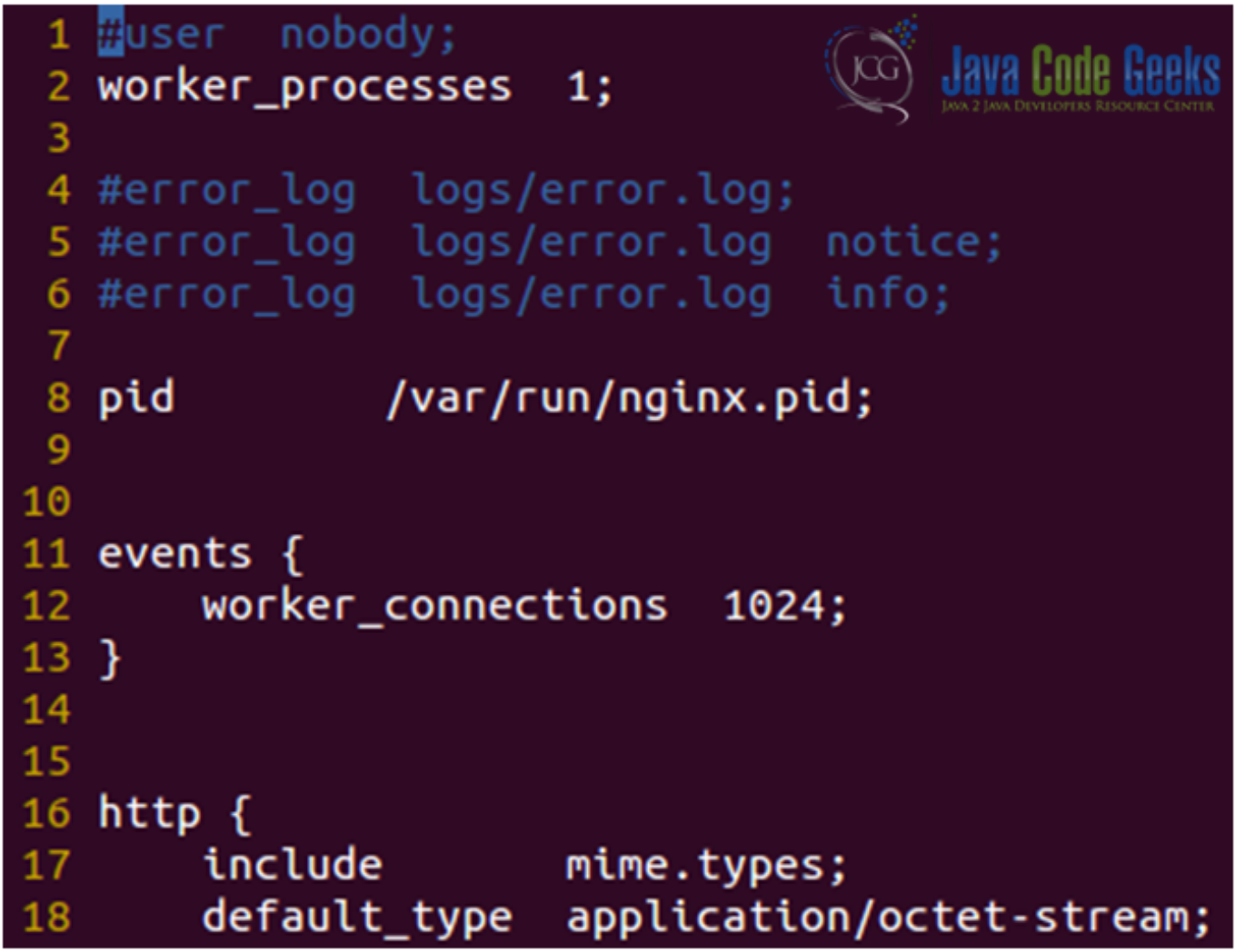

Figure 2.1: Nginx main configuration file (nginx.conf)

The lines that begin with "#" are comments, or in other words, strings of text that are not interpreted by the program during execution. You can comment out entire lines or add brief comments after a line to clarify what it is supposed to do.

The next line shows a directive, in this case `worker_processes`, which represents a setting to which we will append one or more values, depending on its syntax. Actually, the `worker_processes` directive only accepts one argument, which needs to be a numerical value. Another example of a directive is `user` (which is commented out in the previous line). The `user` directive lets us add up to 2 text strings - the first one is required and indicates the user account Nginx will run as, and the second one is optional (user group). We will need to remember, right from the start, that each directive has its own meaning and defines a particular feature of the application.

Each module that is part of Nginx has a specific set of directives. When a new module is activated, its specific set of directives becomes available, and directive blocks may also be enabled.

Directive blocks are precisely that: a block of text that lets us specify a logical construction of the configuration and allows us to use inheritance, which means that configuration found in a certain block is passed on to its children blocks as well. However, inside a children block you can still change the value of a directive that was defined in its parent block.

In Fig. 2.2 and 2.3 we see that all access to the root directory of the site (/) is saved to `/home/gacanepa/nginx-1.5.6/html/access_log` except for the access to the directory named restricted (notice the directive `access_log off`) in the corresponding directive block.

```
server {
    listen       80;
    server_name  localhost;
    root   /home/gacanepa/nginx-1.5.6/html;
        access_log   /home/gacanepa/nginx-1.5.6/html/access_log;
    location / {
        index  index.html index.htm;
    }
    location /restricted {
        index  index.html index.htm;
        access_log off;
    }
```

Figure 2.2: Nested directive blocks

```
gacanepa@ubuntuOS:~/nginx-1.5.6/html$ ls -l
total 20
-rw-r--r-- 1 gacanepa gacanepa 537 Oct  1 10:44 50x.html
-rw-r--r-- 1 root     root     157 Oct 31 15:41 access_log
-rw-rw-r-- 1 gacanepa gacanepa   9 Oct 30 12:23 add.html
-rw-r--r-- 1 gacanepa gacanepa 612 Oct  1 10:44 index.html
-rw-r--r-- 1 root     root     306 Oct 31 15:50 main_access_log
gacanepa@ubuntuOS:~/nginx-1.5.6/html$ cat access_log
127.0.0.1 - - [31/Oct/2013:15:41:59 -0300] "GET /50x.html HTTP/1.1" 404 168 "-" "Mozil
la/5.0 (X11; Ubuntu; Linux i686; rv:23.0) Gecko/20100101 Firefox/23.0"
gacanepa@ubuntuOS:~/nginx-1.5.6/html$ cat main_access_log
127.0.0.1 - - [31/Oct/2013:15:49:31 -0300] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X1
1; Ubuntu; Linux i686; rv:23.0) Gecko/20100101 Firefox/23.0"
127.0.0.1 - - [31/Oct/2013:15:50:13 -0300] "GET /index.html HTTP/1.1" 200 612 "-" "Moz
illa/5.0 (X11; Ubuntu; Linux i686; rv:23.0) Gecko/20100101 Firefox/23.0"
gacanepa@ubuntuOS:~/nginx-1.5.6/html$
```

Figure 2.3: Access logs

2.3 Organization and inclusions

Let's focus on line 17. There is a special directive there: `include`. This directive is used to insert the contents of the specified file at this exact location. The name of the file can be specified either by an absolute path or a relative path to the current directory (as it is in this case). We can see that there's a file named `mime.types` in the same directory as `nginx.conf` (see Fig. 2.4).

```
gacanepa@ubuntuOS:/usr/local/nginx/conf$ ls -l
total 36
-rw-r--r-- 1 root root 1034 Oct 30 11:35 fastcgi.conf
-rw-r--r-- 1 root root  964 Oct 30 11:35 fastcgi_params
-rw-r--r-- 1 root root 2837 Oct 30 11:35 koi-utf
-rw-r--r-- 1 root root 2223 Oct 30 11:35 koi-win
-rw-r--r-- 1 root root 3815 Oct 30 11:35 mime.types
-rw-r--r-- 1 root root 2700 Oct 31 13:43 nginx.conf
-rw-r--r-- 1 root root  596 Oct 30 11:35 scgi_params
-rw-r--r-- 1 root root  623 Oct 30 11:35 uwsgi_params
-rw-r--r-- 1 root root 3610 Oct 30 11:35 win-utf
gacanepa@ubuntuOS:/usr/local/nginx/conf$
```


Figure 2.4: screenshot

The end result is the same that would be accomplished by actually inserting the contents of `mime.types` into the `nginx.conf` file. Of course, if we do that, the main configuration file would soon become a nightmare to read. This way the include directive helps us to ensure that the `nginx.conf` file remains easy to read and understand. As a plus, it works recursively in that an included file can reference another file and so on, and it also supports `filename globbing`, which means it recognizes and expands the standard wild card characters (* and ?) and character lists in square brackets, for example. This way we can add multiple configuration files such as `20131030.conf`, `20131031.conf`, and `20131101.conf`. If we only want to include the files that begin with 201310, we must add the following line to the `nginx.conf` file (see Fig. 2.5):

```
root@debian:~# nginx -v
nginx version: nginx/1.2.1
root@debian:~#
```


Figure 2.5: Including files using wildcards

However, if we add a specific file (not by `filename globbing`) that doesn't exist, Nginx will not start properly (see Fig. 2.6). Otherwise, we will be presented with a "test is successful" message (see Fig. 2.7):

```
gacanepa@ubuntuOS:/usr/local/nginx/conf$ sudo service nginx restart
Restarting nginx: nginx: [emerg] open() "/usr/local/nginx/conf/myfile.conf" failed (2:
 No such file or directory) in /usr/local/nginx/conf/nginx.conf:18
nginx: configuration file /usr/local/nginx/conf/nginx.conf test failed
gacanepa@ubuntuOS:/usr/local/nginx/conf$
```

Figure 2.6: Nginx fails to restart due to a missing include file

```
gacanepa@ubuntuOS:/usr/local/nginx/sbin$ sudo ./nginx -t
nginx: the configuration file /usr/local/nginx/conf/nginx.conf syntax is ok
nginx: configuration file /usr/local/nginx/conf/nginx.conf test is successful
gacanepa@ubuntuOS:/usr/local/nginx/sbin$
```

Figure 2.7: The syntax test of the configuration file has completed successfully

2.4 Base modules

The base modules allow us to define the basic parameters and configuration of Nginx. They are built-in into Nginx automatically during compile time. Their directives and blocks are always available. There are three types of base modules:

- The **core** module contains essential directives and features. Most of its directives must be placed at the root (meaning the top) of the configuration file and are only used once. Table 1 shows some directives in the core module with a brief explanation and context (if no context is specified, the actual context is global and the directive must be placed at the root of the configuration file).

Table 2.1: Some Events Module directives

Directive/Context	Syntax and description
`daemon`	Accepted values: on / off Syntax: daemon on; Enables or disables daemon mode (starts the program in the background or the foreground, respectively). Useful to troubleshoot issues.
`env`	Syntax: `env MY_VARIABLE=my_value;` Allows us to define or define environment variables.
`error_log`	Context: `main, http, server`, and `location` Syntax: `error_log /path/to/file level` (where level can be one of the following values: debug, info, notice, warn, error, and crit, depending on the type of errors that we want to save in the log file). To disable error logging, redirect the log output to /dev/null: `error_log /dev/null crit;`
`master_process`	Accepted values: `on /off` Default value: `on` If enabled (on) Nginx will start both the master process and worker processes. If disabled, Nginx will work with a unique process. This is used for debugging purposes only and will cause that clients won't be able to connect to the server.
`pid`	Syntax: `file path` Default value: defined at compile time. Path of the file where the PID of Nginx will be stored.

Table 2.1: (continued)

`user`	Syntax: `user username groupname; user username;` Lets you define the user account, and optionally the user group used for starting the worker processes. For security reasons, a user / group with limited privileges must be used for this.
`worker_processes`	Syntax: `numeric` or `auto`, for example: `worker_processes 4;` Defines the amount of worker processes. If a process is blocked for some reason, future requests can be served by other worker processes. If the value auto is used, Nginx selects an appropriate value (which by default it is the amount of CPU cores detected).
`worker_priority`	Syntax: `Numeric worker_priority 0;` Defines the priority, as the operating system sees it, of the worker processes from -20 (highest) to 19 (lowest).

- The **events** module allows us to configure the operation of the networking capabilities of Nginx. These directives must be placed inside the events block at the root of the configuration file (see Fig. 2.8). Table 2 shows two of the directives available in this module.

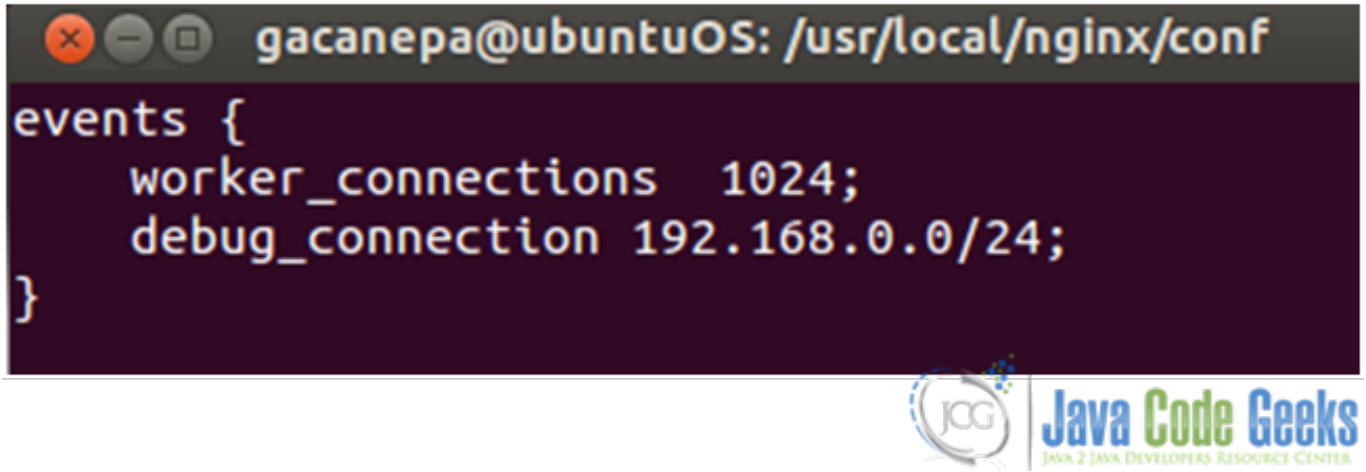

Figure 2.8: The events block

Table 2.2: Some Events Module directives

Directive/Context	**Syntax and description**
`worker_connections`	Syntax: `Numeric` Defines the amount of connections that a worker process may handle simultaneously.
`debug_connection`	Syntax: IP address or CIDR block. `debug_connection 192.168.0.100 debug_connection 192.168.0.0/24` Writes detailed log for clients matching this IP address or CIDR block. The information is saved in the file defined in the `error_log` directive (it must be enabled with the debug level, and Nginx must be compiled with the `--debug` switch in order to enable this directive).

- The **configuration** module enables file inclusions with the include directive, as discussed earlier. The directive may be placed anywhere in the configuration file and accepts one (and only one) parameter: the file's path relative to the current working directory (unless it is specified with the path all the way down from the / directory).

2.5 The HTTP Server

The web server itself is configured using the directives found in the HTTP Core module. This module is the essential component of the HTTP configuration and will allow us -among other things- to serve multiple websites, which are referred to as virtual hosts. It is organized into three major blocks (see Fig. 2.9):

- **http:** must be placed at the root of the configuration file. It is used to define directives and blocks related to the web server functionality of Nginx.
- **server:** must be inserted inside the http block and is used to declare a specific website.
- **location:** allows us to define a group of settings to be applied to certain sections of the web site. This block can either be used within a server block or nested inside another location block.

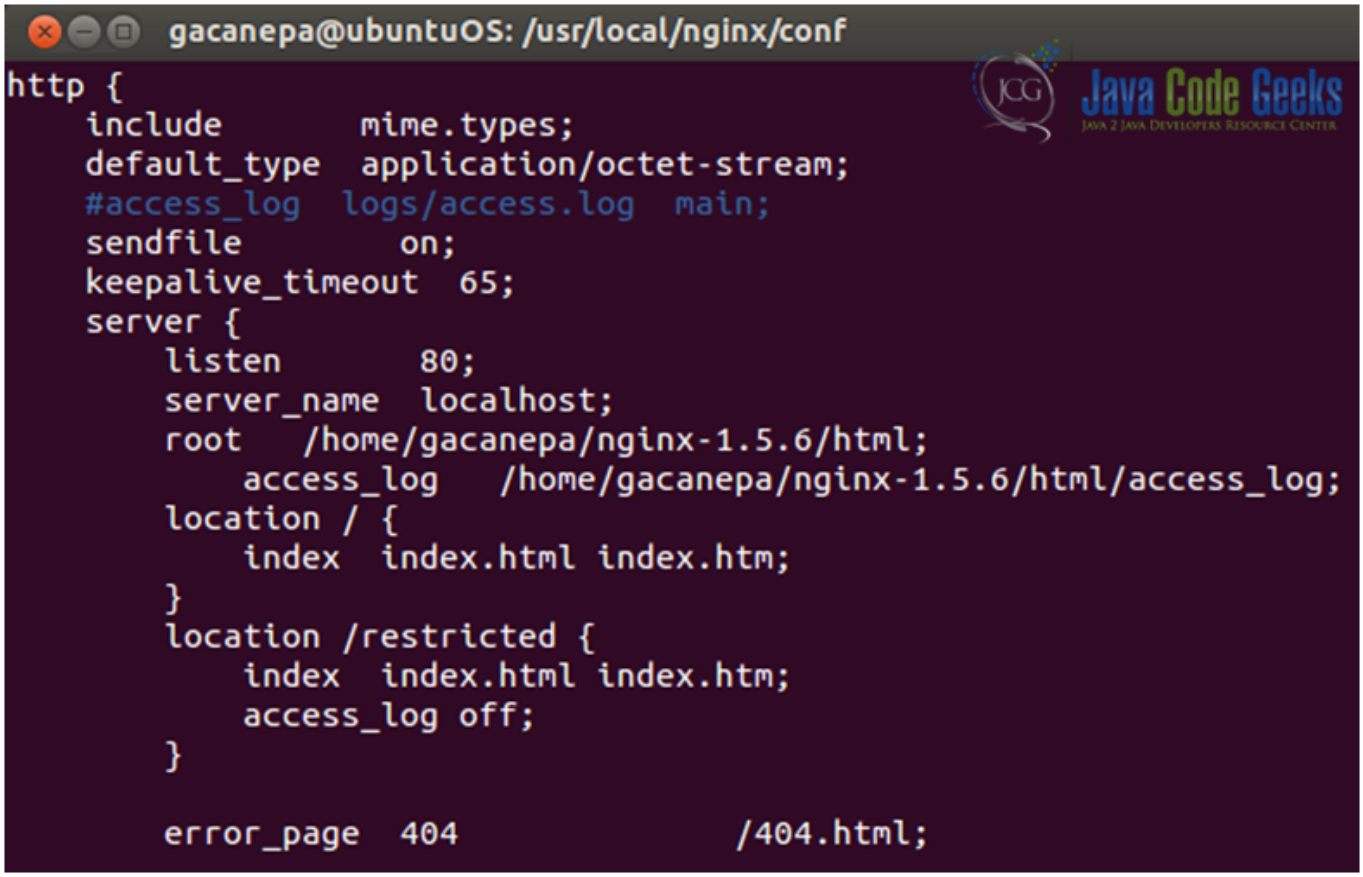

Figure 2.9: The http, server, and location blocks

2.6 Mail server proxy

To act as mail server (not enabled by default), Nginx must be compiled with the `--with-mail` (which enables mail server proxy module with support for POP3, IMAP4, and SMTP) option in `./configure`. If for some reason this will be the only use of Nginx in our system, the HTTP Core module can be disabled using the `--without-http switch`.

In Fig. 2.10 we can see a portion of the mail block, which is used to set the configuration of the mail server. The capabilities of the imap, pop3, and smtp protocols and a detailed description can be found in the IANA (Internet Assigned Numbers Authority) web site.

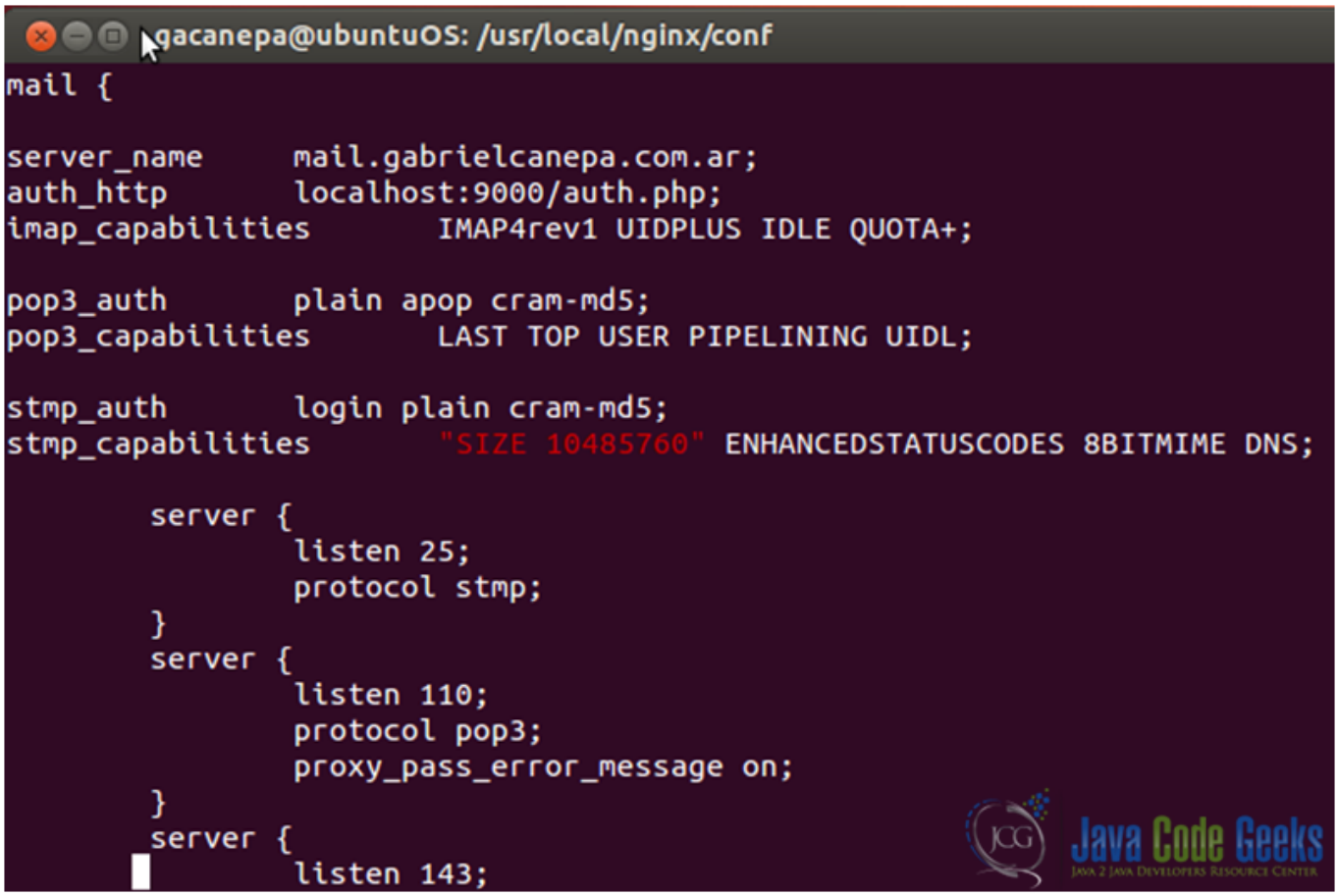

Figure 2.10: The mail block

2.7 Virtual hosts

As mentioned earlier, a virtual host is a certain website that is served by Nginx in a single server. The first step to set up virtual hosts is to create one or more server blocks in the main configuration file. Those server blocks are identified either by a hostname or through a combination or IP address and port number (see Fig. 2.11).

```
server {
    listen       80;
    server_name  nginxtest.com;
    root   /home/gacanepa/nginx-1.5.6/html;
        access_log   /home/gacanepa/nginx-1.5.6/html/access_log;
    location / {
        index  index.html index.htm;
    }
    location /restricted {
        index  index.html index.htm;
        access_log off;
    }
```

Figure 2.11: Creating a server block to serve a virtual host

Next, we need to set up the virtual host main configuration file. The default installation of Nginx provides a sample file (located in `/etc/nginx/sites-available/default`) that we will copy and name after our website:

```
sudo cp /etc/nginx/sites-available/default /etc/nginx/sites-available/nginxtest.com
```

Then the next thing to do is edit the sample file (`nginxtest.com`) with basically the same information that is found in the `nginx.conf` file (see Fig. 2.12).

```
gacanepa@ubuntuOS:~$ nginx -v
nginx version: nginx/1.1.19
gacanepa@ubuntuOS:~$
```

Figure 2.12: Virtual host configuration file

The virtual host must now be enabled by creating a symlink to this file in the `sites-enabled` directory:

```
sudo ln -s /etc/nginx/sites-available/example.com /etc/nginx/sites-enabled/nginxtest.com
```

To avoid conflicts, we can also delete the file named default in the sites-enabled directory:

```
sudo rm /etc/nginx/sites-enabled/default
```

Now let's restart Nginx and look what happens! (Fig. 2.13)

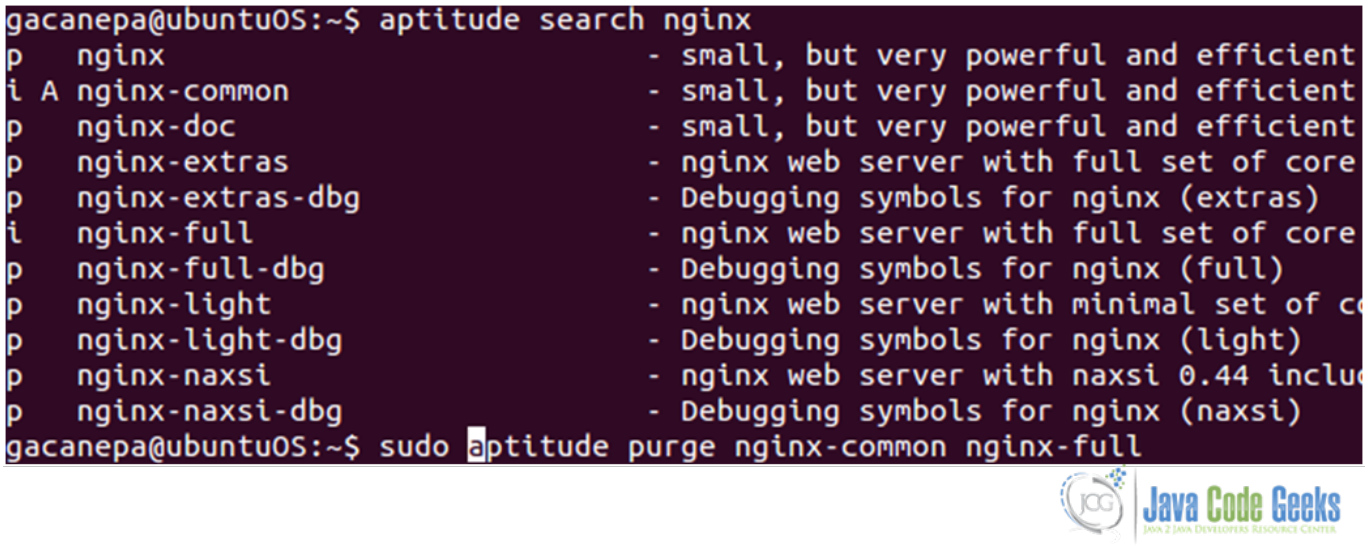

Figure 2.13: Our first virtual host is working

To add more virtual hosts, we can just repeat the process above step by step, with the caution to set up a new document root with the appropriate domain name, and then creating and activating the new virtual host file as we did with the nginxtest.com website.

Chapter 3

Nginx and Apache

3.1 Introduction

Nginx and Apache can certainly work together, not necessarily replacing each other as our web server of choice. This solution offers many advantages and solves the issues that most system administrators are familiar with, such as slowdowns and complex configurations. You can just take a look at the Apache configuration file and chances are you'll probably agree with me!

3.2 Nginx as reverse proxy

A reverse proxy is a device or service placed between a client and a server in a network infrastructure. Incoming requests are handled by the proxy, which interacts on behalf of the client with the desired server or service residing on the server1. In this case, Nginx will act as reverse proxy between the client (your computer, for example) and Apache, the backend web server (see Fig. 3.1).

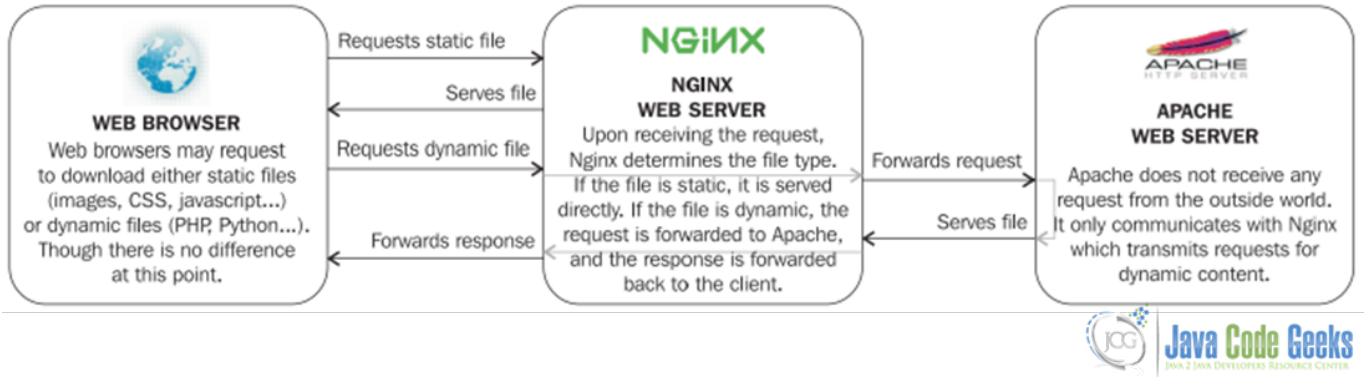

Figure 3.1: Nginx acting as reverse proxy

In this above diagram, Nginx acts as reverse proxy (or in other words, as frontend server) and receives all requests from the outside world. At this point those requests can be filtered or "delivered" to Apache (acting as HTTP client) in the backend. These two services can even be hosted in the same physical server with the caution to use different listening ports for each of them. This whole operation is handled by the proxy module of Nginx.

The main purpose of setting up Nginx as a frontend server and giving Apache a simple backend role is to improve the serving speed, given the fact that great amount of requests coming from clients are for static files (html pages, cascading style sheets, regular images, and so on), and static files are served much faster by Nginx. The overall performance sharply improves both on the client side and server side.

3.3 Nginx proxy module

Fortunately, the proxy module is enabled by default during the installation of Nginx. The main directives of the module can be seen in Table 1.

Table 3.1: Main directives of the proxy module

Directive/Context	Description
`proxy_buffers` Context: `http, server, location`	The request is sent to the backend server by specifying its location. Syntax: TCP sockets: `proxy_pass` http://hostname:port; UNIX domain sockets: `proxy_pass` http://unix:/path/to/file.socket; (https can be used instead of http for secure traffic) Examples: `proxy_pass` http://localhost:8080; `proxy_pass` http://127.0.0.1:8080; `proxy_pass` http://unix:/tmp/nginx.sock;
`proxy_method`	Allows overriding the HTTP method of the request to be forwarded to the backend server. Syntax: `proxy_method method`; Example: `proxy_method POST;`
`proxy_hide_header` Context: `http, server, location`	By default, as Nginx prepares the response received from the backend server to be forwarded back to the client, it ignores some of the http headers4. With this directive, you can specify an additional header line to be hidden from the client. This directive can be inserted multiple times with one header name for each. Syntax: `proxy_hide_header header_name;` Example: `proxy_hide_header Cache-Control;`
`proxy_pass_header` Context: `http, server, location`	Forces some of the ignored headers to be passed on to the client. Syntax: `proxy_pass_header header_name;` Example: `proxy_pass_header Date;`
`proxy_pass_request_body` `proxy_pass_request_headers` Context: `http, server, location`	Defines whether or not, respectively, the request body and extra request headers should be passed on to the backend server. Syntax: `on` or `off`;
`proxy_redirect` Context: `http, server, proxy_cachelocation`	Allows you to rewrite the URL appearing in the Location HTTP header on redirections triggered by the backend server. Syntax: `off`, `default`, or the `URL` of your choice `off`: Redirections are forwarded as it is. `default`: The value of the `proxy_pass` directive is used as the hostname and the current path of the document is appended. Note that the `proxy_redirect` directive must be inserted after the `proxy_pass` directive as the configuration is parsed sequentially. `URL`: Replace a part of the URL by another. Additionally, you may use variables in the rewritten URL. Examples: `proxy_redirect off;` `proxy_redirect default;` `proxy_redirect` http://localhost:8080/ http://mysite.com/;

The best scenario is to limit to the extent possible the number of requests that are forwarded to the backend server. To that end, the proxy module comes with a group of directives that will help us build a caching system as well as control buffering options and the way Nginx deals with temporary files (see Table 2 for more information on most of these directives).

Table 3.2: Some caching / buffering directives

Directive/Context	Description
`proxy_pass` Context: `location, if`	Sets the amount and size of buffers that will be used for reading the response data from the backend server. If the buffers aren't large enough the data will be saved to disk before being served to the user. Syntax: `proxy_buffers amount size;` Default: 8 buffers, 4k or 8k each depending on platform Example: `proxy_buffers 8 4k;`
`proxy_method`	Sets the size of the buffer for reading the beginning of the response from the backend server, which usually contains simple header data. Syntax: Numeric value (size) Example: `proxy_buffer_size 4k;`
`proxy_cache_key` Context: `http, server, location`	This directive defines the cache key, in other words, it differentiates one cache entry from another. Syntax: `proxy_cache_key key;` Example: `proxy_cache_key "$scheme$host$request_uri $cookie_user";` Note: strings beginning with "$" (dollar sign) are variables. The proxy module offers 4 built-in variables; others can be created at the user's will.
`proxy_cache` Context:`http, server, location`	Defines a shared memory zone used for caching. The same zone can be used in several places. The off parameter disables caching inherited from the previous configuration level. Syntax: `proxy_cache zone off;` Default: `proxy_cache off;`
`proxy_cache_path` Context: `http`	Sets the path and other parameters of a cache. Cache data are stored in files. Both the key and file name in a cache are a result of applying the MD5 function to the proxied URL. The levels parameter defines hierarchy levels of a cache. Syntax: `proxy_cache_path path [levels= levels] keys_zone=name:size [inactive= time] [max_size=size] [loader_files= number] [loader_sleep=time] [loader_threshold=time];` [Optional parameters are indicated inside square brackets] Example: `proxy_cache_path /tmp/nginx_cache levels=1:2 zone=zone1:10m inactive=10m max_size=200M;`
`proxy_cache_min_uses` Context: `http, server, location`	Defines the minimum amount of hits before a request is eligible for caching. By default, the response of a request is cached after one hit (next requests with the same cache key will receive the cached response). Syntax: Numeric value Example: `proxy_cache_min_uses 1;`

There are even more directives that let you define the behavior of Nginx in the case of timeouts or other limitations regarding communications with the backend server (see Table 3):

Table 3.3: Some directives regarding communications with the backend server (Apache)

Directive/Context	Description

Table 3.3: (continued)

`proxy_connect_timeout` Context: `http, server, location`	Defines the backend server connection timeout. This is different from the read/send timeout. If Nginx is already connected to the backend server, the `proxy_connect_timeout` is not applicable. It should be noted that this timeout cannot usually exceed 75 seconds. Syntax: Time value (in seconds) Example: `proxy_connect_timeout 15;`
`proxy_read_timeout` Context: `http, server, location`	Defines a timeout for reading a response from the proxied server. A timeout is set only between two successive read operations, not for the transmission of the whole response. If a proxied server does not transmit anything within this time, a connection is closed. Syntax: Time value (in seconds) Default value: 60 Example: `proxy_read_timeout 60;`
`proxy_send_timeout` Context: `http, server, location`	This timeout is for sending data to the backend server. The timeout isn't applied to the entire response delay but between two write operations instead. Syntax: Time value (in seconds) Default value: 60 Example: `proxy_send_timeout 60;`
`proxy_ignore_client_abort` Context: `http, server, location`	Determines whether the connection with a proxied server should be closed when a client closes a connection without waiting for a response. If set to on, Nginx will continue processing the proxy request, even if the client aborts its request. In the other case (off), when the client aborts its request, Nginx also aborts its request to the backend server. Default value: `off`

3.4 A note on variables

The proxy module comes with the following variables that can be used as arguments for the directives listed above:

- `$proxy_host`: the hostname of the backend server.
- `$proxy_port`: the port of the backend server.
- `$proxy_add_x_forwarded_for`: Contains client request-header "X-Forwarded-For" with separated by comma $remote_addr. If there is no X-Forwarded-For request-header, than $proxy_add_x_forwarded_for is equal to $remote_addr.
- `$proxy_internal_body_length`: Length of the request body (set with the `proxy_set_body` directive or 0).

3.5 Configuring Apache

By default, web servers are configured to listen on tcp port 80. So the first thing that we need to do is to change the settings of Apache in order to avoid conflicts with Nginx (which will be running as the frontend server).

In Ubuntu 12.04, the main configuration file for Apache is located in `/etc/apache2` under the name `ports.conf` (see Fig. 3.2).

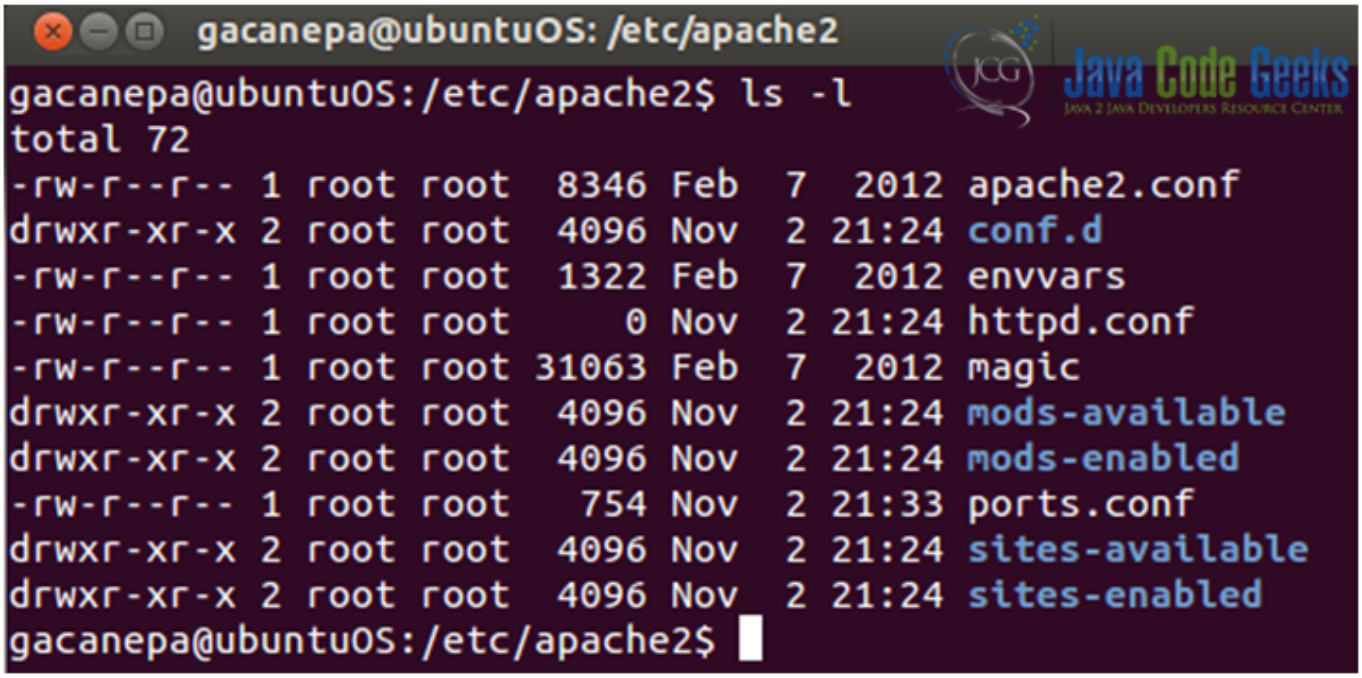

Figure 3.2: The Apache configuration files

There are 3 main elements that need to be replaced in our Apache configuration (see Fig. 3.3 and 3.4):

1) The Listen directive must be changed to a port other than 80, such as 8080. 2) The following configuration directive is present in the main configuration file:

```
NameVirtualHost A.B.C.D:8080
```

where A.B.C.D is the IP address of the main network interface on which server communications (between the frontend and the backend servers) go through. In this case, we use the loopback interface and its IP address since both Apache and Nginx are installed in the same physical server. If you do not host Apache on the same server, you will need to specify the IP address of the network interface that can communicate with the server hosting Nginx. 3) The port that was just selected must be reported in all our virtual hosts configuration sections (in `/etc/apache2/sites-available/default`).

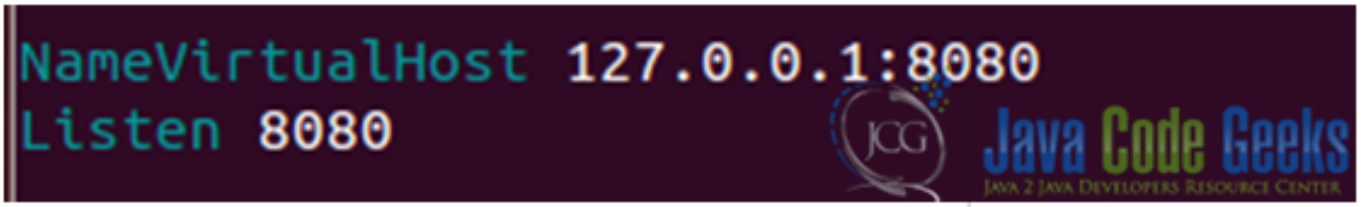

Figure 3.3: screenshot

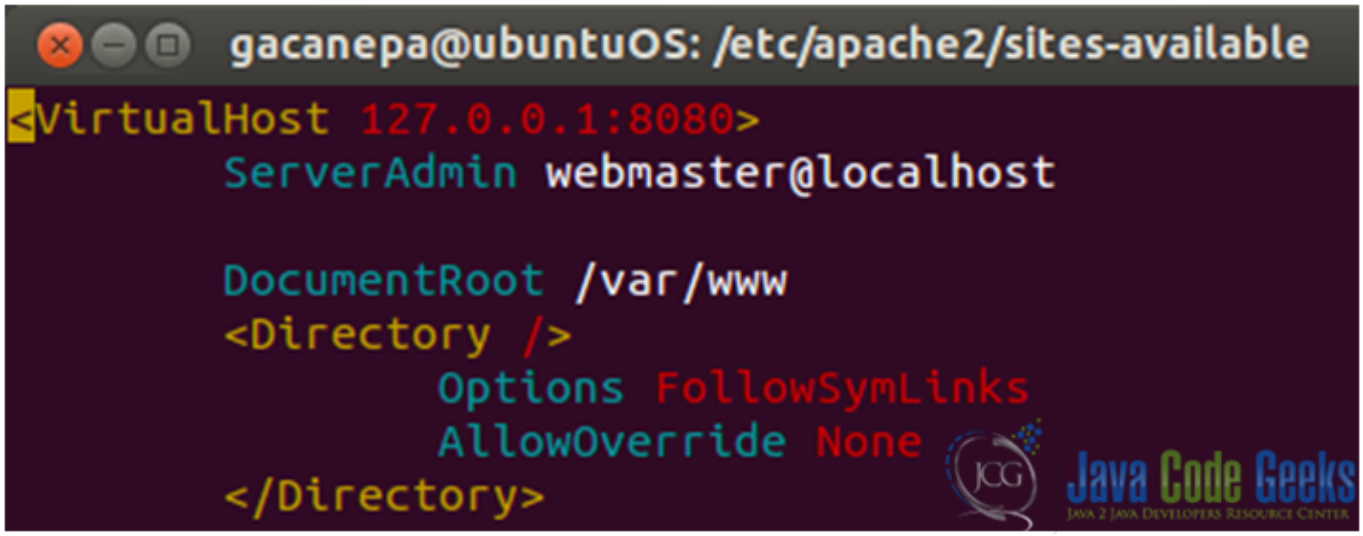

Figure 3.4: screenshot

After restarting Apache, we can open a web browser and confirm that it is listening on port 8080 (see Fig. 3.5):

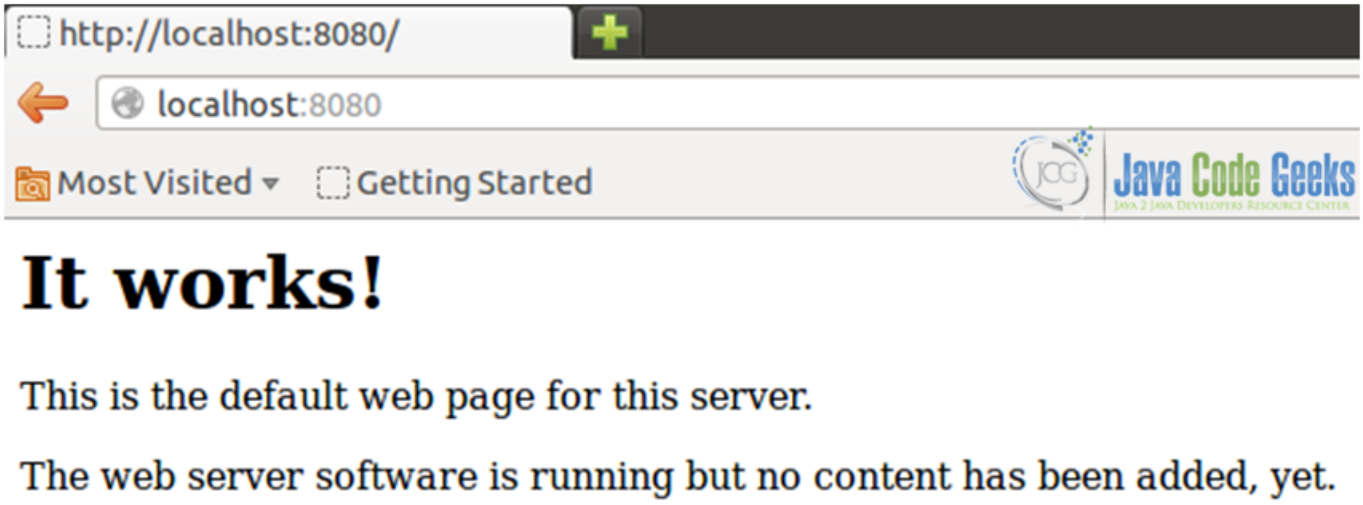

Figure 3.5: Apache is listening on port 8080

As a extra security measure, we can tell Apache to only serve requests coming from the frontend server. This can be performed in 2 ways: 1) system wide or by 2) establishing per-virtual-host restrictions.

1) As discussed earlier, the Listen directive of Apache lets you specify a port, but also an IP address. However, by default, no IP address is selected which results in communications coming from all interfaces. All you have to do is replace the `Listen *:8080` directive by `Listen 127.0.0.1:8080`, Apache should then only listen on the local IP address.

2) Using the allow and deny Apache directives we can define which IP addresses will be able to access each virtual host. Once the changes are made, Apache must be restarted (or its configuration reloaded) in order to reflect the changes that we have just made.

3.6 Configuring Nginx

The first directive that we will use in the process of enabling proxy options is `proxy_pass`. Since it can't be used at the http or server level, we will include it in every single place that we want to be forwarded. As a preliminary example, we will have all requests made to the restricted folder be forwarded to the Apache web directory (/var/www). See Figs. 3.6 and 3.7, 3.8 and 3.9:

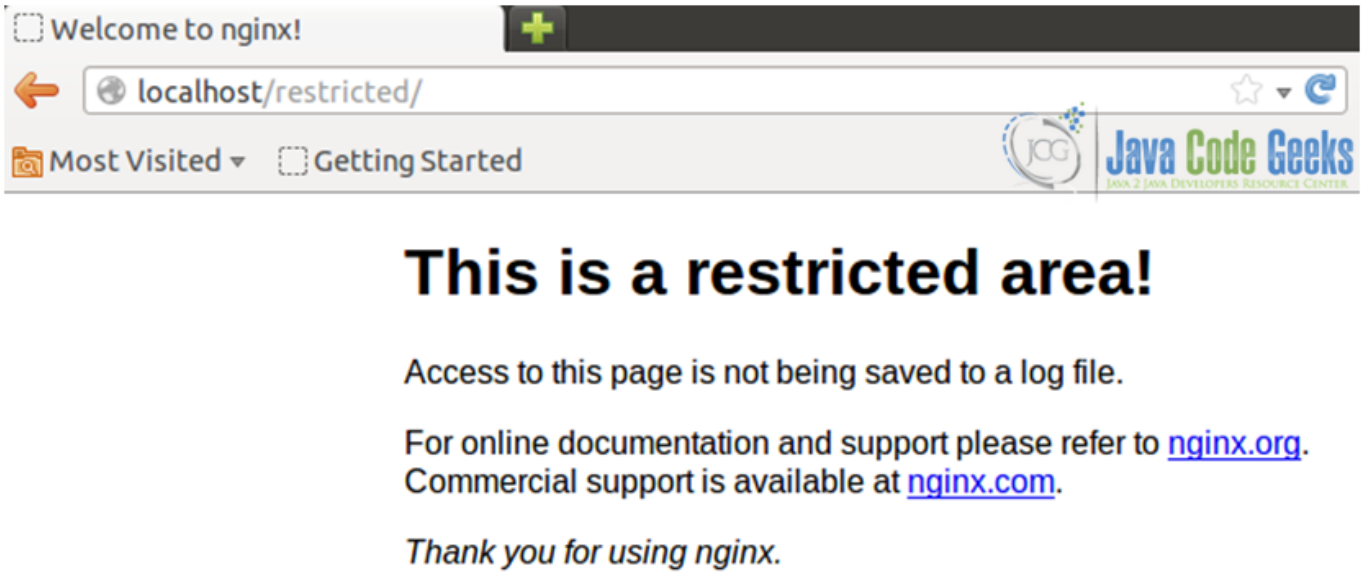

Figure 3.6: The restricted folder shows a simple notice (BEFORE)

```
server {
    listen       80;
    server_name  nginxtest.com;
    root   /home/gacanepa/nginx-1.5.6/html;
        access_log   /home/gacanepa/nginx-1.5.6/html/access_log;
    location / {
        index  index.html index.htm;
    }
    location /restricted {
        index  index.html index.htm;
        access_log off;
    }
```

Figure 3.7: Nginx main configuration file (BEFORE)

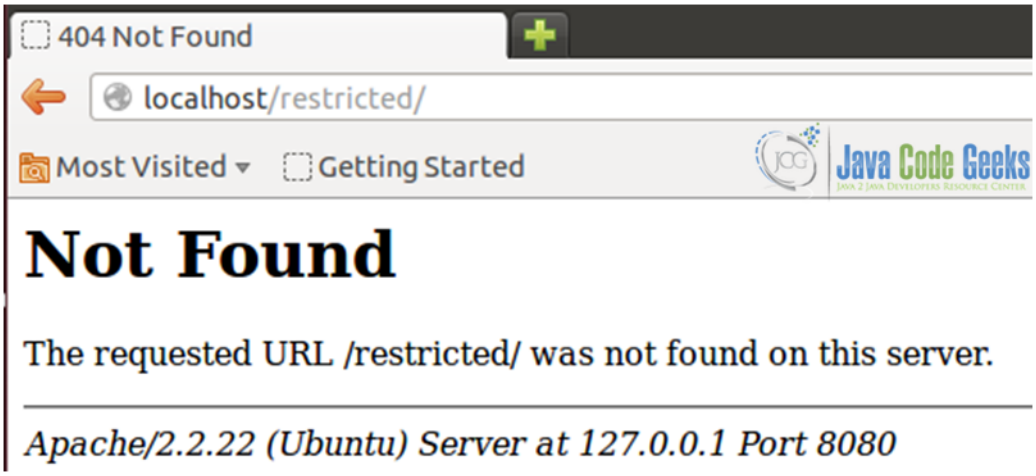

Figure 3.8: A request to view the restricted directory shows a `Not Found` message since there is no such page in Apache's root directory

```
server {
    listen       80;
    server_name  nginxtest.com;
    root   /home/gacanepa/nginx-1.5.6/html;
    access_log   /home/gacanepa/nginx-1.5.6/html/access_log;
    location / {
        index  index.html index.htm;
    }
    location /restricted {
        proxy_pass http://127.0.0.1:8080;
    }
```

Figure 3.9: Nginx main configuration file (AFTER)

3.7 Separating content

In order to take better advantage of this Nginx-Apache setting, we can separate the content that each one will deliver upon request.

Apache will serve dynamic files, that is, files that require some sort of processing before being sent to the client, such as php files, Python scripts, and so on. Nginx will serve static files - all other content that does not require additional processing (html pages, cascading style sheets, images, media, and so on).

To do this, add the following blocks in the nginx.conf file (see Fig. 3.10):

```
# proxy the PHP scripts to Apache listening on 127.0.0.1:8080

location ~ \.php$ {
    proxy_pass   http://127.0.0.1:8080;
}

# Serve directly: *.html, *.css, *.ico, *.png, *.jpeg,
# *.jpg, *.gif, *.swf, *.txt, *.pdf
location ~* \.(html|css|ico|png|jpeg|jpg|gif|swf|txt|pdf)$ {
        root /home/gacanepa/nginx1-5.6/html;
        access_log /home/gacanepa/nginx-1.5.6/html/access_log;
 }
```

Figure 3.10: Separating content served by the frontend and backend servers

When we restart Nginx, we may run into the following issue (see Fig. 3.11):

```
gacanepa@ubuntuOS:/usr/local/nginx/conf$ sudo service nginx restart
[sudo] password for gacanepa:
Restarting nginx: nginx: [emerg] using regex "\.php$" requires PCRE library in /
usr/local/nginx/conf/nginx.conf:45
nginx: configuration file /usr/local/nginx/conf/nginx.conf test failed
```

Figure 3.11: Missing library

We will go ahead and install the PCRE library that is available in the `libpcre3-dev` package (refer to tutorial 1: Nginx installation on Linux). See Fig. 3.12 for details on this package. Once installed, we will have to recompile Nginx.

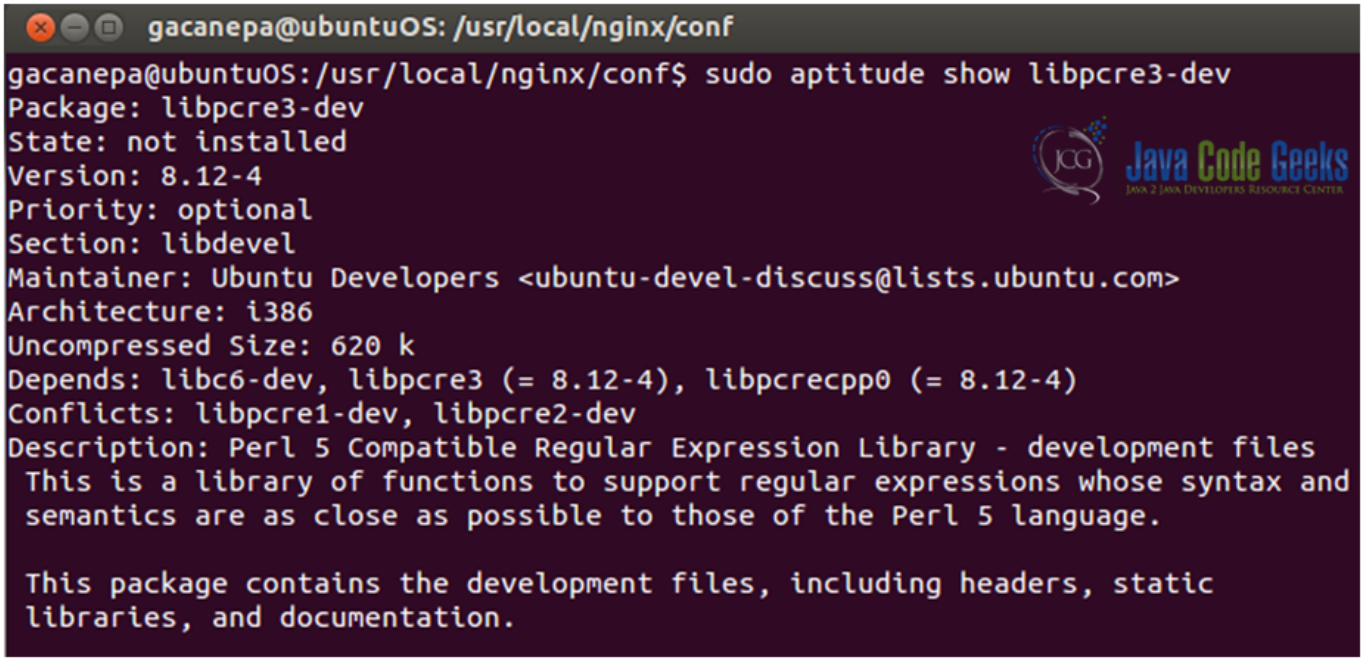

Figure 3.12: The PCRE library comes with the libpcre3-dev package

Let's create a sample php file in `/var/www` (see Fig. 3.13):

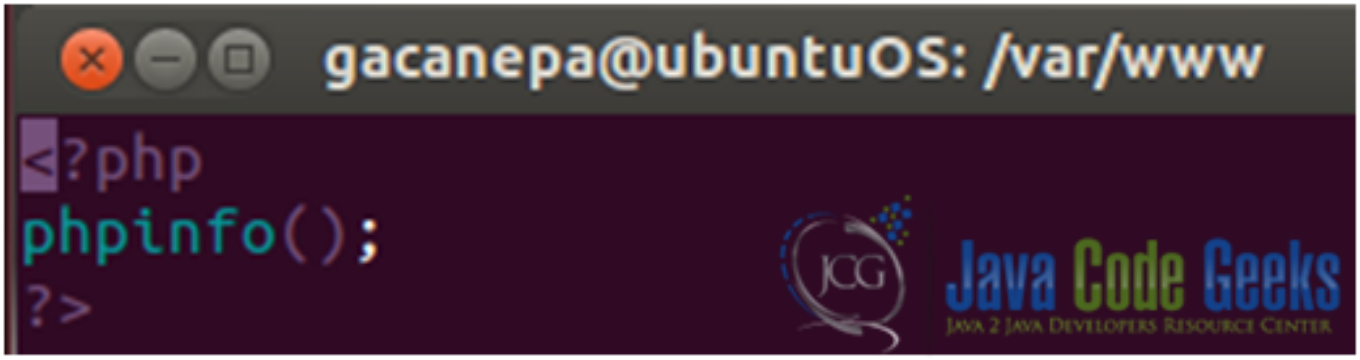

Figure 3.13: Sample php file

Now we will point our web browser to `http://localhost/test.php`. Please note that localhost per se points to the frontend server, so when it receives a request for a php file, it will forward the request to Apache (see Fig. 3.14)

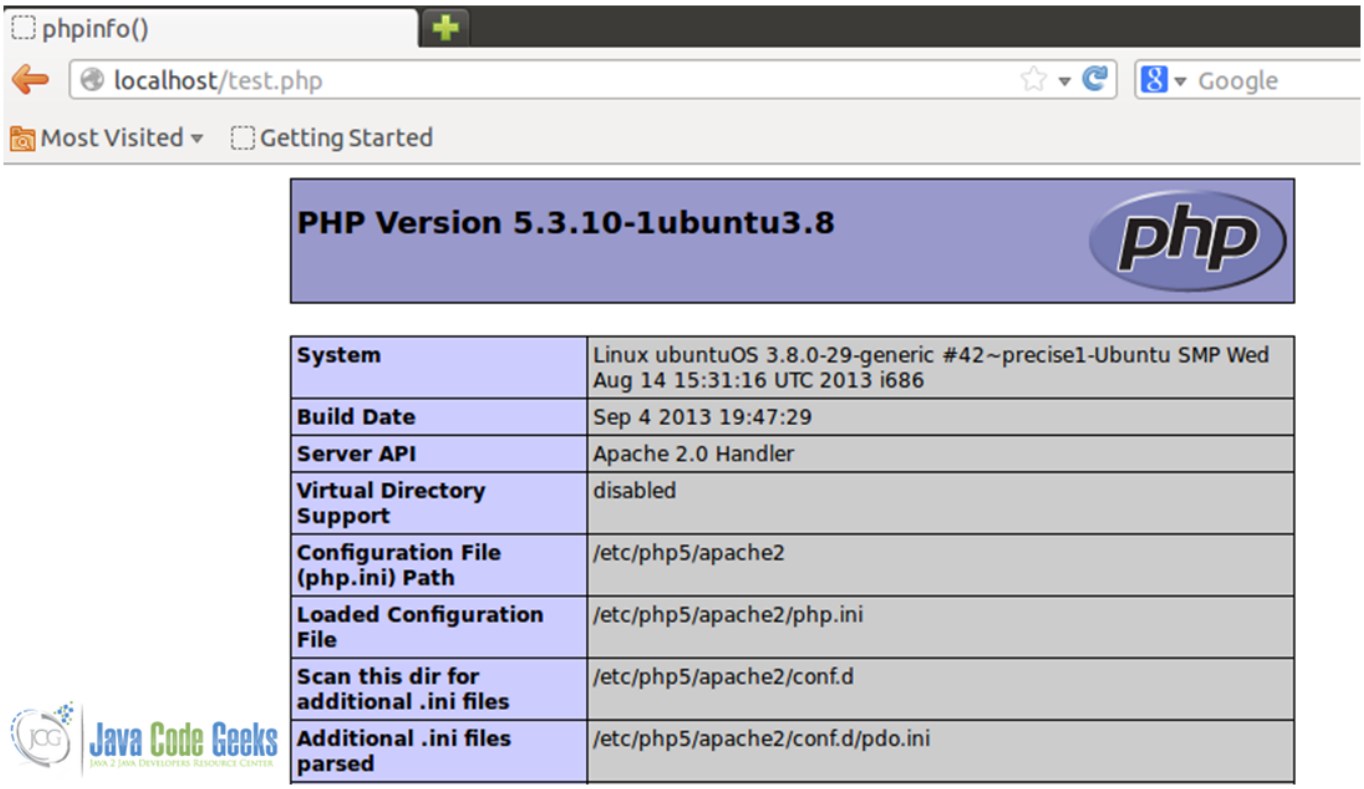

Figure 3.14: Request for php files are forwarded to the backend server

3.8 Download the configuration files

Here you can download the configuration files used in this tutorial: Config_files.zip

Chapter 4

Load balancing with Nginx

4.1 Introduction - The need for load balancing

Load balancing is a networking method for distributing workloads across multiple computing resources, such as servers, a server cluster (a group of servers that work together in such a way that they can be viewed as a single system), network links, CPUs, or other hardware components (http://en.wikipedia.org/wiki/Load_balancing_(computing) target=[Wikipedia entry]). This technique is aimed at increasing both the capacity (supporting a large number of concurrent users) and the reliability of the backend applications by decreasing (or balancing, to be more accurate) the burden on individual servers or nodes. For that same reason, load balancing also provides redundancy and disaster recovery capabilities. Using Nginx, we will set up a layer-7 load balancer (which will distribute requests based upon data found in application layer protocols such as HTTP and FTP).

4.2 Necessary modules

In order for us to set up a load balancer, we will need 2 modules: the `proxy` module and the `upstream` module. Both are built into Nginx by default.

As before, we will work with the nginx.conf file, which is located in the `/usr/local/nginx/conf` directory.

4.2.1 upstream module

Insert an `upstream` directive block inside http { } (see Fig. 4.1). You can name this block as you like (app_rack in the example below).

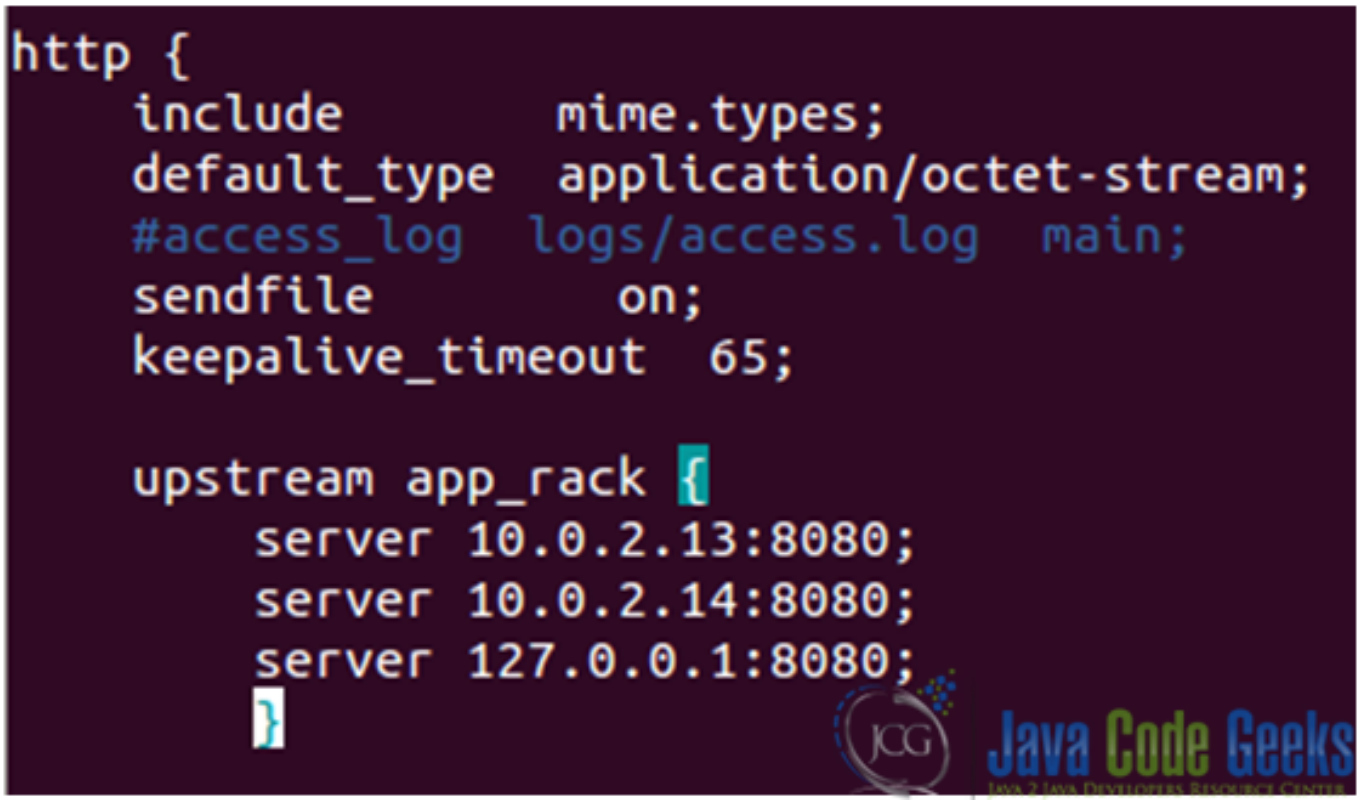

Figure 4.1: An upstream block

The server directive assigns the name and the parameters of server. “Name” can be a domain name, an IP address, port or unix socket. If domain name resolves to several addresses, then all are used. There are several extra parameters available, but we will explore them later.

- **10.0.2.13** is the IP address of a Debian Wheezy 7.2 server where Apache is listening on port 8080.
- **10.0.2.14** is the IP address of a CentOS 6.4 server where Apache is listening on port 8080.
- Finally, **127.0.0.1** is the IP address of the loopback interface, where Apache is listening on port 8080 in the same physical server where Nginx is running on port 80.

This way we have three servers - all of them running a simple PHP script that can be served by an instance of Apache (see Fig. 4.2).

As we mentioned earlier, the server directive that appears within `upstream` blocks accepts several parameters that influence the backend selection by Nginx:

- **weight=n:** if this parameter is placed after a server name, that server will be selected “n-times” more often.
- **max_fails=n:** This defines the number of timed-out connections that should occur (in the time frame specified with the fail_timeout parameter below) before Nginx considers the server inoperative.
- **fail_timeout=n:** If Nginx fails to communicate with the backend server max_fails times over fail_timeout seconds, the server is considered inoperative.
- **down:** this server is no longer used. This only applies when the ip_hash directive is enabled.
- **backup:** if a backend server is marked as backup, Nginx will not make use of the server until all other servers (servers not marked as backup) are down or inoperative.

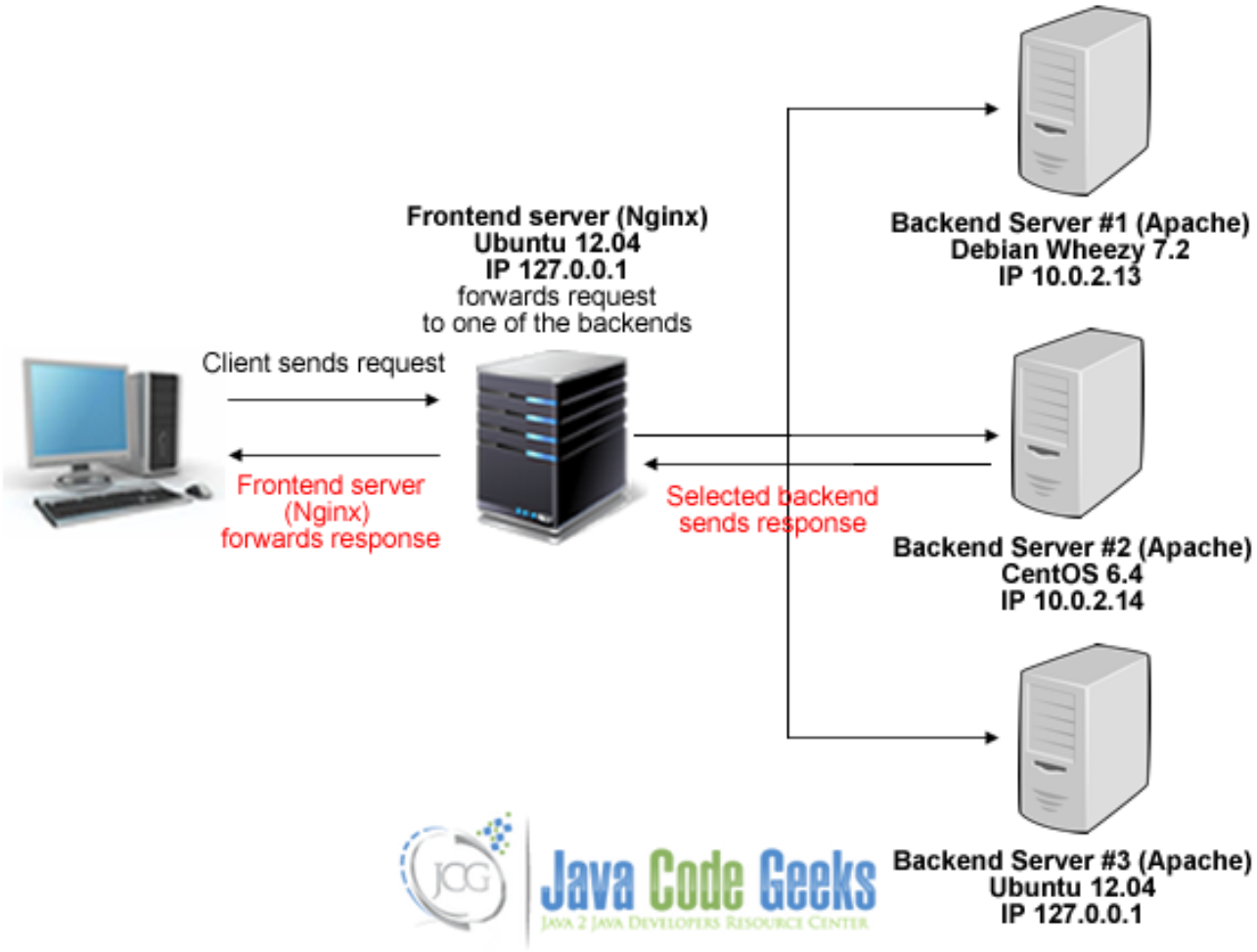

Figure 4.2: Nginx is connected to multiple backend servers

Note: There are several other parameters to use with the upstream module. For a more comprehensive list, refer to the online documentation of the upstream module.

That being said, if we want to ensure that all requests from the same visitor always get processed by the same backend server, we need to enable the `ip_hash` option when declaring the upstream block.

Now it is time to "play around" with the parameters discussed above. Let's take a look at our modified nginx.conf file (see Figs. 4.3 and 4.4).

```
upstream app_rack {
    ip_hash;
    server 10.0.2.13:8080 max_fails=3 fail_timeout=30s;
    server 10.0.2.14:8080 weight=3;
    # backup can only be used if ip_hash is not present
    server 127.0.0.1:8080 down;
    }
```

Figure 4.3: The `upstream` block in the nginx.conf file

```
# proxy the PHP scripts to Apache
# listening on each server listed
# in the upstream block above.

location ~ \.php$ {
    proxy_pass   http://app_rack;
}
```

Figure 4.4: Redirecting requests to the servers listed in the `upstream` block

With the above settings, we can expect that the CentOS 6.4 server (IP 10.0.2.14) will get twice as much traffic as the other two backend servers. The presence of the `ip_hash` option will ensure that each request from a certain visitor -actually, from a certain IP- will be served by the same backend server.

Oops! An error occurred (see Fig. 4.5)

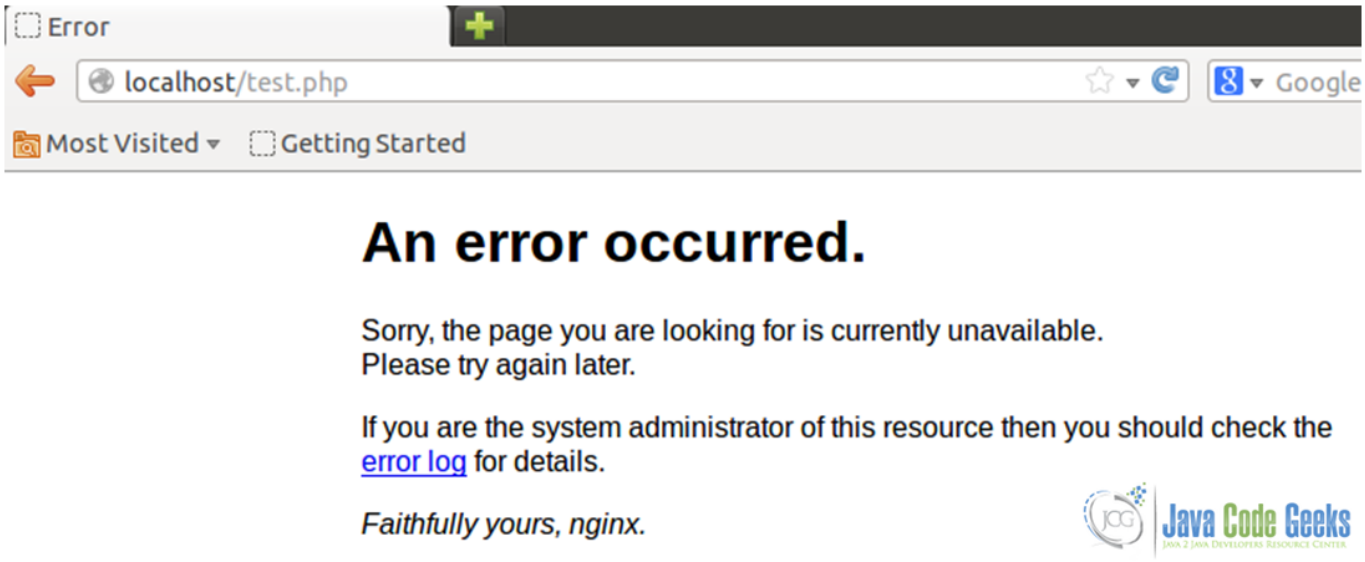

Figure 4.5: Nginx returns an error page while trying to forward the request to a backend server

Since we are instructed to take a look at the error log, that's what we'll do (see Fig. 4.6)

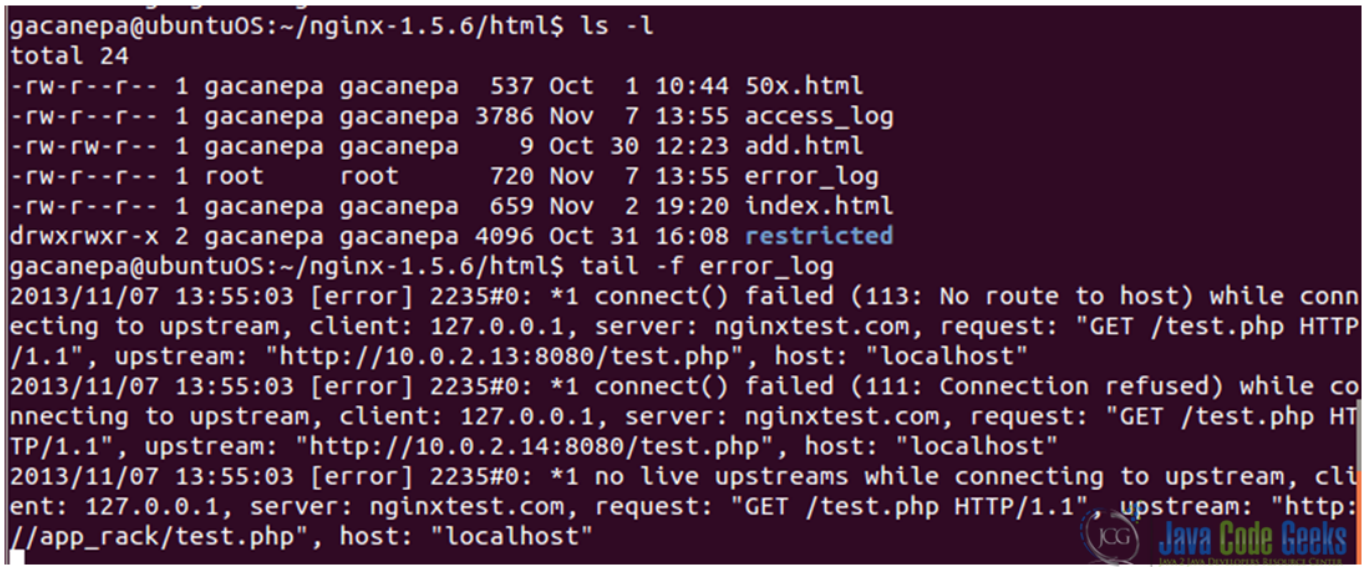

Figure 4.6: screenshot

It seems as though all of the upstream servers are down!

The backend server #2 is pingable. So what else could be wrong? After running

```
netstat -npltu | grep :8080
```

in the CentOS 6.4 server we find out that Apache is not running. We can start it with service httpd start.

Let's see what happens now (see Fig. 4.7)

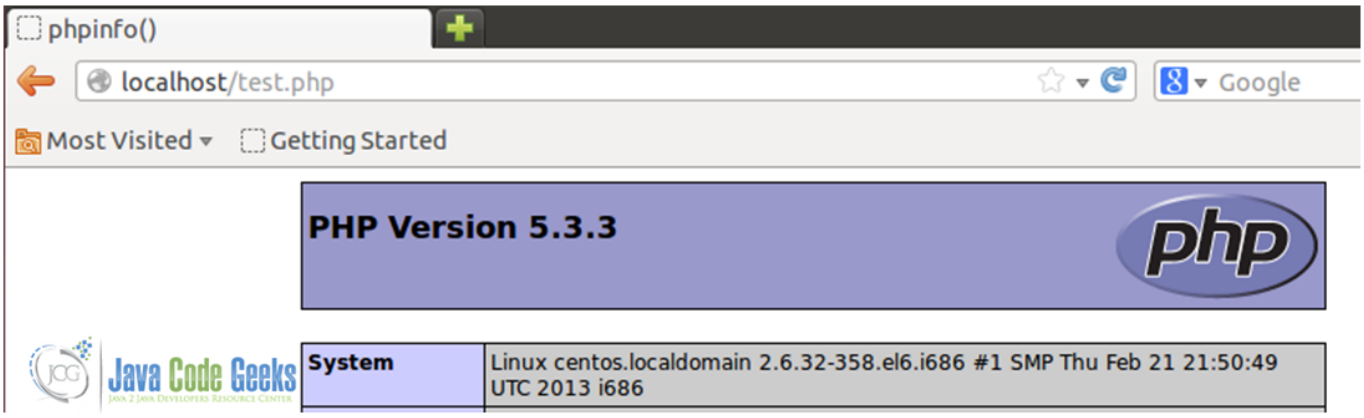

Figure 4.7: screenshot

The test.php file was served by the CentOS 6.4 server.

We will go ahead and edit the `nginx.conf` file again (see Figs. 4.8 and 4.9).

```
upstream app_rack {
    #ip_hash;
    server 10.0.2.13:8080 max_fails=3 fail_timeout=30s;
    server 10.0.2.14:8080 weight=3;
    # backup can only be used if ip_hash is not present
    server 127.0.0.1:8080 down;
    }
```

Figure 4.8: The requests will be served by either the Debian or the CentOS server

```
upstream app_rack {
    #ip_hash;
    server 10.0.2.13:8080 down; # max_fails=3 fail_timeout=30s;
    server 10.0.2.14:8080 weight=2 down;
    # backup can only be used if ip_hash is not present
    server 127.0.0.1:8080 backup;
    }
```

Figure 4.9: Since the Debian and CentOS servers have been marked as down, all the requests will be forwarded to the local Ubuntu 12.04 server

With the `upstream` block show in Fig. 4.9, the php file will be served by the Ubuntu server (see Fig. 4.10)

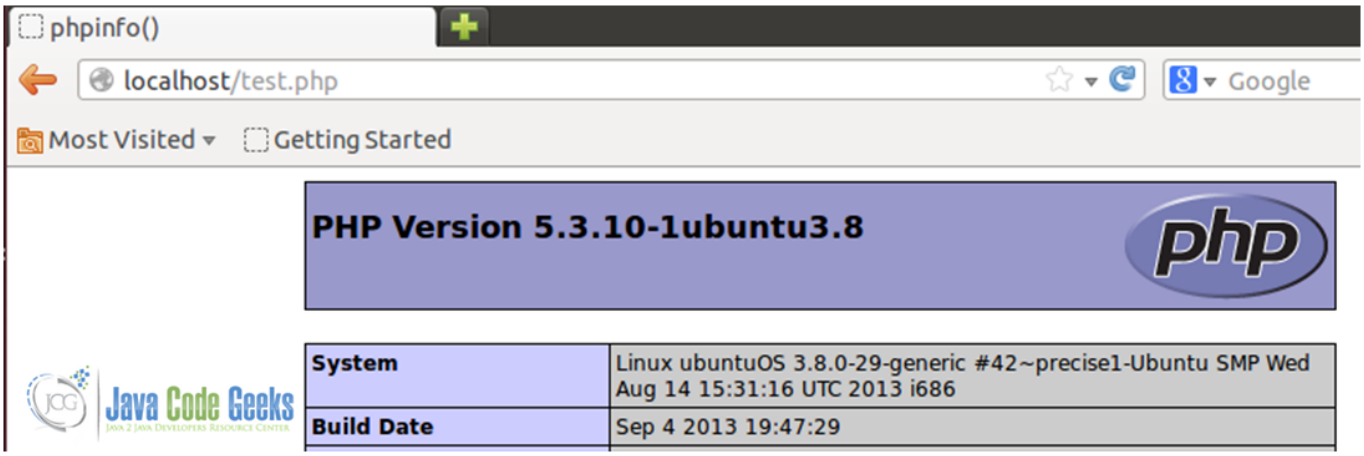

Figure 4.10: The php file is served by the Ubuntu server, which is marked as backup

Now, say that we have an application (running on the 3 servers) that is drawing high traffic. How can we know which server is receiving each request from the frontend server? That is where the embedded variable `$upstream_addr` -which is built-in into the upstream module- and the `log_format` directive come into play.

- Define a custom `log_format` (`MyLogFormat` in the example below) at http level in the main configuration file (see Fig. 4.11)

```
http {
    include       mime.types;
    default_type  application/octet-stream;
    #access_log  logs/access.log  main;
    sendfile        on;
    keepalive_timeout  65;

    log_format MyLogFormat '$upstream_addr [$time_local] - $http_referer';

    upstream app_rack {
```

Figure 4.11: Creating a custom log format

- Use the log format created earlier with one of the logs (see Fig. 4.12)

```
server {
    listen       80;
    server_name  nginxtest.com;
    root   /home/gacanepa/nginx-1.5.6/html;
    access_log  /home/gacanepa/nginx-1.5.6/html/access_log MyLogFormat;
    error_log   /home/gacanepa/nginx-1.5.6/html/error_log;
    location / {
        index  index.html index.htm;
    }
```

Figure 4.12: Adding a custom log format to the access log

Now let's modify the upstream block once again (see Fig. 4.13) in order to forward traffic to the 3 servers.

```
upstream app_rack {
    #ip_hash;
    server 10.0.2.13:8080; # max_fails=3 fail_timeout=30s;
    server 10.0.2.14:8080 weight=2;
    # backup can only be used if ip_hash is not present
    server 127.0.0.1:8080;
    }
```

Figure 4.13: Modifying the upstream block in order to forward requests to the three backend servers

Now restart Nginx and, finally, let's see what the access log shows (see Fig. 4.14) after refreshing the browser's windows a couple of times:

```
gacanepa@ubuntuOS: ~/nginx-1.5.6/html
gacanepa@ubuntuOS:~/nginx-1.5.6/html$ tail -f access_log
10.0.2.14:8080 [08/Nov/2013:11:16:50 -0300] - -
127.0.0.1:8080 [08/Nov/2013:11:16:50 -0300] - http://localhost/test.php
10.0.2.13:8080 [08/Nov/2013:11:16:50 -0300] - http://localhost/test.php
10.0.2.14:8080 [08/Nov/2013:11:17:02 -0300] - -
10.0.2.14:8080 [08/Nov/2013:11:17:02 -0300] - http://localhost/test.php
10.0.2.13:8080 [08/Nov/2013:11:17:02 -0300] - http://localhost/test.php
127.0.0.1:8080 [08/Nov/2013:11:17:07 -0300] - -
10.0.2.14:8080 [08/Nov/2013:11:17:07 -0300] - http://localhost/test.php
10.0.2.14:8080 [08/Nov/2013:11:17:07 -0300] - http://localhost/test.php
10.0.2.13:8080 [08/Nov/2013:11:17:07 -0300] - http://localhost/test.php
```

Figure 4.14: The access log

4.3 Download the configuration file

You can download the configuration file of this tutorial: LoadBalance_conf.zip

Chapter 5

Nginx SSL configuration guide

5.1 Introduction

The SSL (Secure Socket Layer) protocol was created by Netscape to ensure secure transactions between web servers and browsers (using secure pages often identified with https://). The protocol uses a third party, a Certificate Authority (CA), to identify one end or both ends of the transactions (The Linux Documentation Project).

5.2 Adding support for SSL to Nginx

Since The Linux Documentation Project website offers a comprehensive explanation of how this procedure is performed, we will limit this tutorial to show how to set up SSL with Nginx on Ubuntu 12.04 LTS. We will need to compile Nginx with SSL support (see Fig. 5.1).

```
gacanepa@ubuntuOS:~/nginx-1.5.6$ sudo ./configure --without-http_gzip_module --with-http_ssl_modu
le
```

Figure 5.1: Compiling Nginx with SSL support

However, the ssl module requires the OpenSSL library (see Fig. 5.2) to be installed on the system beforehand. We will install the `libssl-dev` package, which includes the said library (see Fig. 5.3).

```
./configure: error: SSL modules require the OpenSSL library.
You can either do not enable the modules, or install the OpenSSL library
into the system, or build the OpenSSL library statically from the source
with nginx by using --with-openssl=<path> option.
```

Figure 5.2: The ssl module needs the OpenSSL library to be installed

```
gacanepa@ubuntuOS:~$ sudo aptitude install libssl-dev
[sudo] password for gacanepa:
The following NEW packages will be installed:
  libssl-dev libssl-doc{a} zlib1g-dev{a}
0 packages upgraded, 3 newly installed, 0 to remove and 150 not upgraded.
Need to get 2,620 kB of archives. After unpacking 6,773 kB will be used.
Do you want to continue? [Y/n/?] Y
WARNING: untrusted versions of the following packages will be installed!

Untrusted packages could compromise your system's security.
You should only proceed with the installation if you are certain that
this is what you want to do.

  libssl-dev libssl-doc zlib1g-dev

Do you want to ignore this warning and proceed anyway?
To continue, enter "Yes"; to abort, enter "No": Yes
```

Figure 5.3: Installing the libssl-dev package, which provides the OpenSSL library

5.3 Creating, signing, and using a certificate

- We will create a directory to store our public key and ssl certificate (see Fig. 5.4).

```
gacanepa@ubuntuOS:~$ sudo mkdir /usr/local/nginx/ssl
[sudo] password for gacanepa:
gacanepa@ubuntuOS:~$
```

Figure 5.4: Creating a directory to store the public key and ssl certificate

- Now let's generate the server private key (see Fig. 5.5)

```
gacanepa@ubuntuOS:/usr/local/nginx/ssl$ sudo openssl genrsa -out server.key 1024
Generating RSA private key, 1024 bit long modulus
.............................++++++
..............++++++
e is 65537 (0x10001)
gacanepa@ubuntuOS:/usr/local/nginx/ssl$ ls -l
total 4
-rw-r--r-- 1 root root 887 Nov  8 15:50 server.key
gacanepa@ubuntuOS:/usr/local/nginx/ssl$
```

Figure 5.5: screenshot

- Create a certificate signing request (see Fig. 5.6)

```
gacanepa@ubuntuOS:/usr/local/nginx/ssl$ sudo openssl req -new -key server.key -out server.csr -config /etc
/ssl/openssl.cnf
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AR]:
State or Province Name (full name) [Some-State]:San Luis
Locality Name (eg, city) []:Villa Mercedes
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Gabriel A. Canepa
Organizational Unit Name (eg, section) []:IT
Common Name (e.g. server FQDN or YOUR name) []:Gabriel A. Canepa
Email Address []:myemail@mydomain.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
gacanepa@ubuntuOS:/usr/local/nginx/ssl$
```

Figure 5.6: screenshot

- Sign your certificate (see Fig. 5.7). Please note that this certificate will only last one day (you can modify this setting by changing the argument to the - `days` option).

```
gacanepa@ubuntuOS:/usr/local/nginx/ssl$ sudo openssl x509 -req -days 1 -in server.csr -signkey server.key
-out server.crt -extensions v3_req -extfile /etc/ssl/openssl.cnf
Signature ok
subject=/C=AR/ST=San Luis/L=Villa Mercedes/O=Gabriel A. Canepa/OU=IT/CN=Gabriel A. Canepa/emailAddress=mye
mail@mydomain.com
Getting Private key
gacanepa@ubuntuOS:/usr/local/nginx/ssl$
```

Figure 5.7: Signing your ssl certificate

- Set up the certificate. Edit the nginx.conf file (see Fig. 5.8). Note that even though there is a ssl directive `(ssl on | off)` available, it is recommended to use the ssl parameter of the listen directive instead of this directive.

```
server {
    listen       443 ssl;
    server_name  nginxtest.com;
    root   /home/gacanepa/nginx-1.5.6/html;
    access_log  /home/gacanepa/nginx-1.5.6/html/access_log MyLogFormat;
    error_log   /home/gacanepa/nginx-1.5.6/html/error_log;
    ssl_certificate /usr/local/nginx/ssl/server.crt;
    ssl_certificate_key /usr/local/nginx/ssl/server.key;
    location / {
        index  index.html index.htm;
    }
```

Figure 5.8: screenshot

It should be kept in mind that due to the HTTPS protocol limitations virtual servers should listen on different IP addresses (see Fig. 5.9). Refer to the nginx.conf file to perform the following modifications if needed:

```
server {
    listen          192.168.1.1:443;
    server_name     one.example.com;
    ssl_certificate /usr/local/nginx/conf/one.example.com.cert;
    ...
}

server {
    listen          192.168.1.2:443;
    server_name     two.example.com;
    ssl_certificate /usr/local/nginx/conf/two.example.com.cert;
    ...
}
```

Figure 5.9: screenshot

Otherwise the first server's certificate will be issued for the second site.

In order to allow to share a single IP address between multiple HTTPS servers is to use a certificate with a wildcard name, for example, `*.example.org`. A wildcard certificate secures all subdomains of the specified domain, but only on one level. This certificate matches `www.example.org`, but does not match example.org and `www.sub.example.org`. These two methods can also be combined. A certificate may contain exact and wildcard names in the `SubjectAltName` field, for example, `example.org` and `*.example.org`.

If we want to allow Subject Alternative Names (SANs) for our certificates we need to enable the following options in the file openssl.cnf file (located in /etc/ssl/openssl/):

1. Include the X509 Version 3 (RFC 2459) extension to allow an SSL certificate to specify multiple names that the certificate should match. We need the `[ req ]` section to read as follows (see Fig. 5.10). This tells openssl to include the `v3_req` section while generating certificate requests:

```
[ req ]
req_extensions = v3_req # The extensions to add to a certificate request
```

Figure 5.10: screenshot

1. Edit the `[ v3_req ]` section as follows (see Fig. 5.11):

```
[ v3_req ]
# Extensions to add to a certificate request
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName = @alt_names

[ alt_names ]
DNS.1 = nginxtest.com
DNS.2 = www.nginxtest.com
```

Figure 5.11: screenshot

Please note that whatever we put in the file openssl.cnf will appear on all certificate requests generated from this point on: if at a later date you want to generate a CSR with different SANs, you'll need to edit this file by hand and change the DNS.x entries.

It is better to place a certificate file with several names and its private key file at the http level of configuration to inherit their single memory copy in all servers (see Fig. 5.12).

```
    ssl_certificate /usr/local/nginx/ssl/server.crt;
    ssl_certificate_key /usr/local/nginx/ssl/server.key;

server {
    listen       443 ssl;
    server_name  nginxtest.com;
    root   /home/gacanepa/nginx-1.5.6/html;
    access_log  /home/gacanepa/nginx-1.5.6/html/access_log MyLogFormat;
    error_log   /home/gacanepa/nginx-1.5.6/html/error_log;
    location / {
        index  index.html index.htm;
    }
}
server {
    listen       443 ssl;
    server_name  www.nginxtest.com;
    root   /home/gacanepa/nginx-1.5.6/html;
    access_log  /home/gacanepa/nginx-1.5.6/html/access_log MyLogFormat;
    error_log   /home/gacanepa/nginx-1.5.6/html/error_log;
    location / {
        index  index.html index.htm;
    }
}
```

Figure 5.12: Using a certificate in two server blocks (nginxtest.com y www.nginxtest.com)

1. Now we can either browse to https://nginxtest.com or https://www.nginxtest.com and we'll see the warning of a self-signed security certificate (see Fig. 5.13):

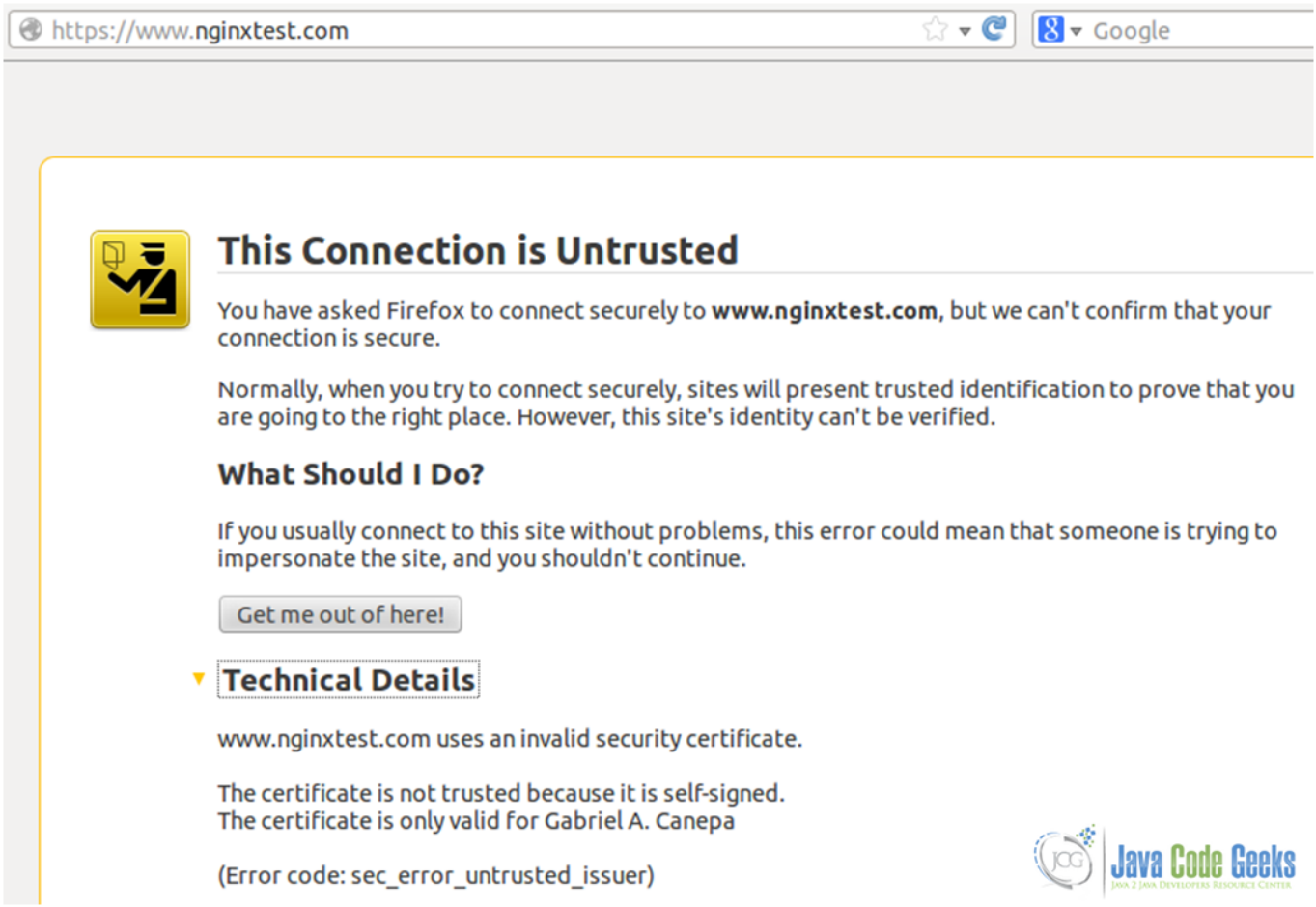

Figure 5.13: A self-signed certificate works but presents this warning screen

1. Click on “Add Exception” (see Fig. 5.14) and then on “Confirm Security Exception”:

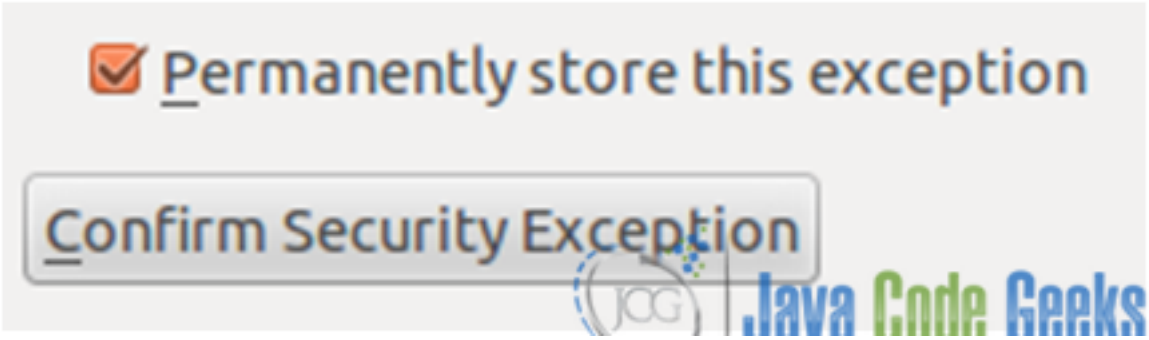

Figure 5.14: Confirming a Security Exception

If you get a **Wrong Site message** (see Fig. 5.15), it means that your certificate belongs to a different site, or that the necessary SANs have not been specified.

Figure 5.15: screenshot

On the other hand, when the SANs have been defined correctly, they can be viewed using the browser's built-in certificate viewer (see Fig. 5.16).

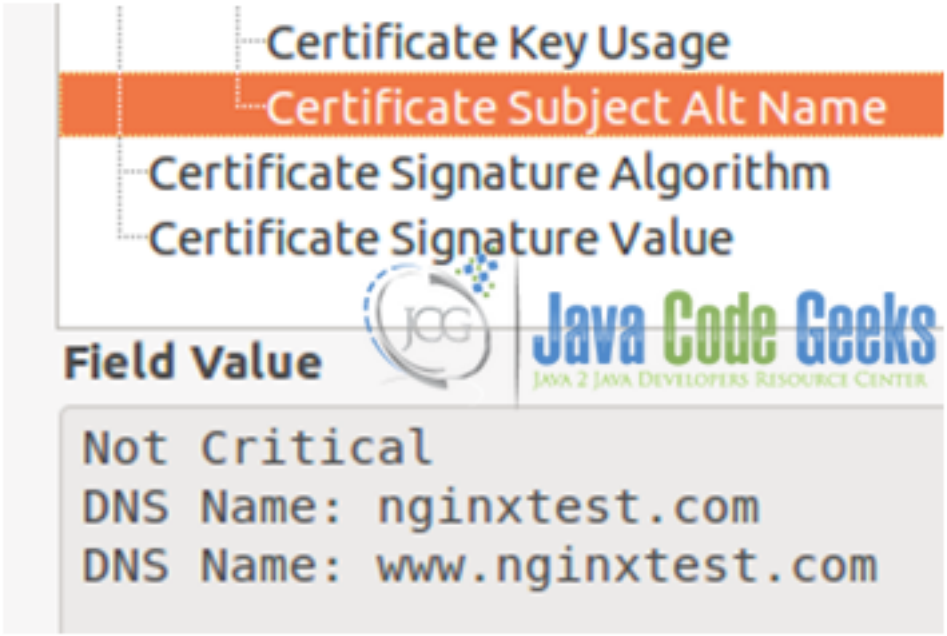

Figure 5.16: Subject Alternative Names

or using (see Fig. 5.17)

```
openssl req -text -noout -in server.csr from the command line
```

```
gacanepa@ubuntuOS:/usr/local/nginx/ssl$ sudo openssl req -text -noout -in server.csr
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: C=AR, ST=San Luis, L=Villa Mercedes, O=Gabriel A. Canepa, OU=IT, CN=Gabriel A. Canepa/ema
ilAddress=myemail@mydomain.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (1024 bit)
                Modulus:
                    00:d7:59:a6:3d:18:1f:17:64:c2:72:a7:81:5c:67:
                    dc:06:cd:55:56:d0:30:06:1c:fd:8e:b4:db:c9:e4:
                    16:44:fa:ac:fa:b2:94:0f:9d:f9:11:f6:97:6e:ea:
                    25:84:ec:58:fd:39:15:79:be:4f:38:3d:b5:83:34:
                    7f:88:ac:f8:de:66:d0:ee:70:15:ae:e7:10:8e:bd:
                    5b:7c:d7:d6:8b:db:68:27:4d:c3:af:06:53:8a:a1:
                    96:2f:c6:ce:85:4c:3f:4a:cd:f0:34:ae:fa:40:7c:
                    78:23:26:e0:95:e2:c2:a8:d6:46:91:39:83:7c:32:
                    3a:1b:7a:65:62:36:53:bb:e9
                Exponent: 65537 (0x10001)
        Attributes:
        Requested Extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Key Usage:
                Digital Signature, Non Repudiation, Key Encipherment
            X509v3 Subject Alternative Name:
                DNS:nginxtest.com, DNS:www.nginxtest.com
    Signature Algorithm: sha1WithRSAEncryption
```


Figure 5.17: screenshot

As we mentioned earlier, a certificate will only be valid within the time frame that was specified when it was first created (1 day, in our case, refer to Figs. 5.18, 5.19, and 5.20)

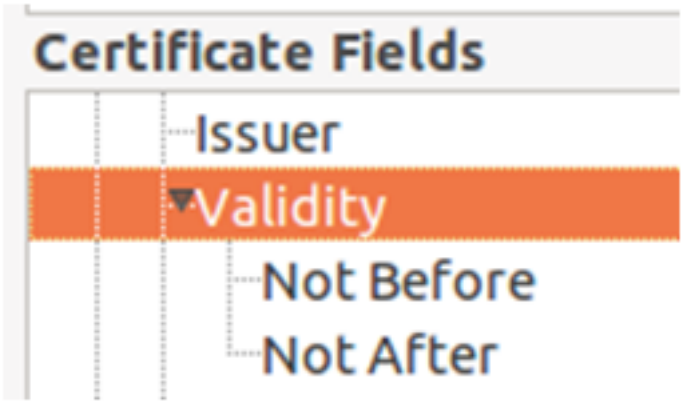

Figure 5.18: screenshot

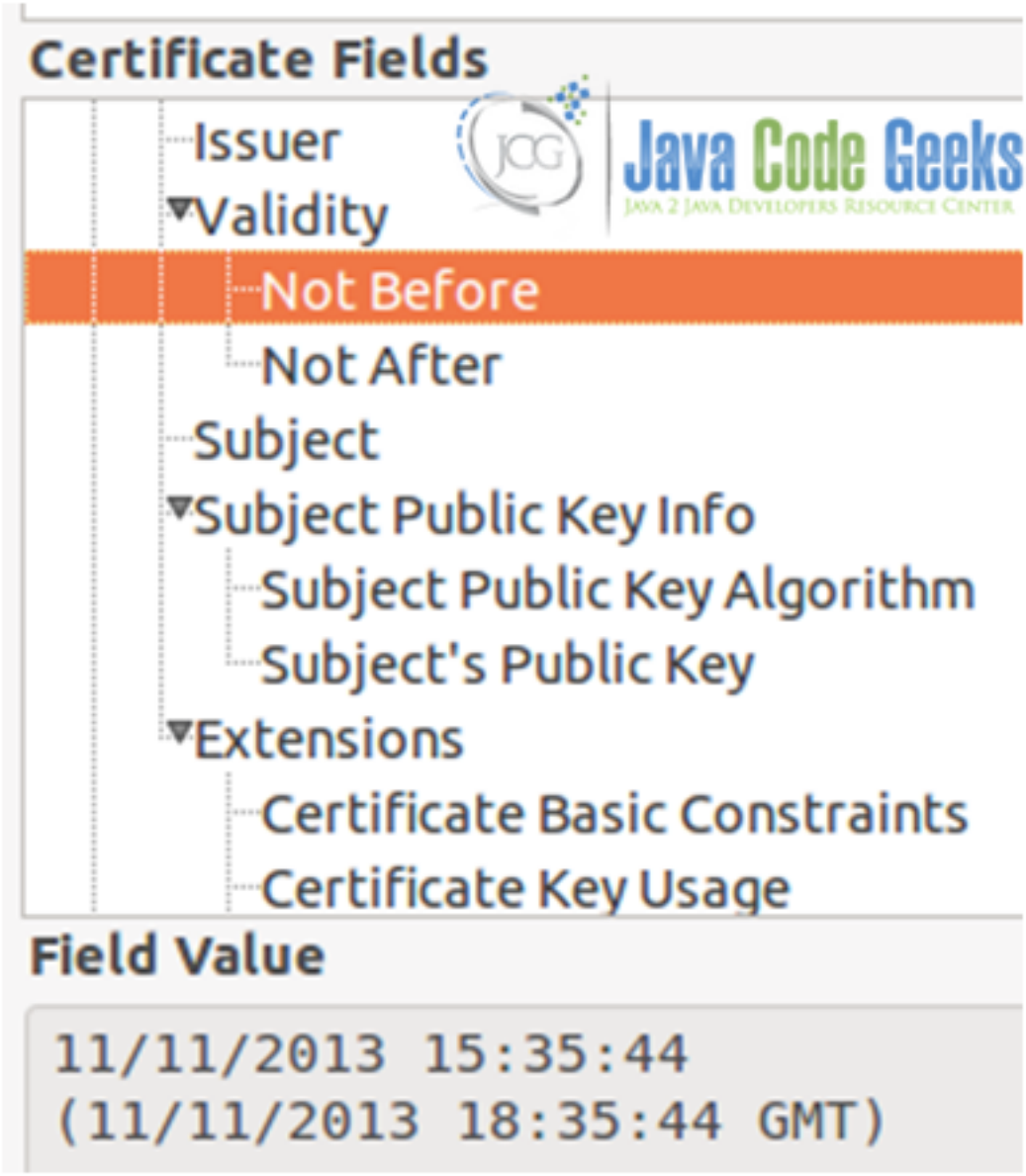

Figure 5.19: screenshot

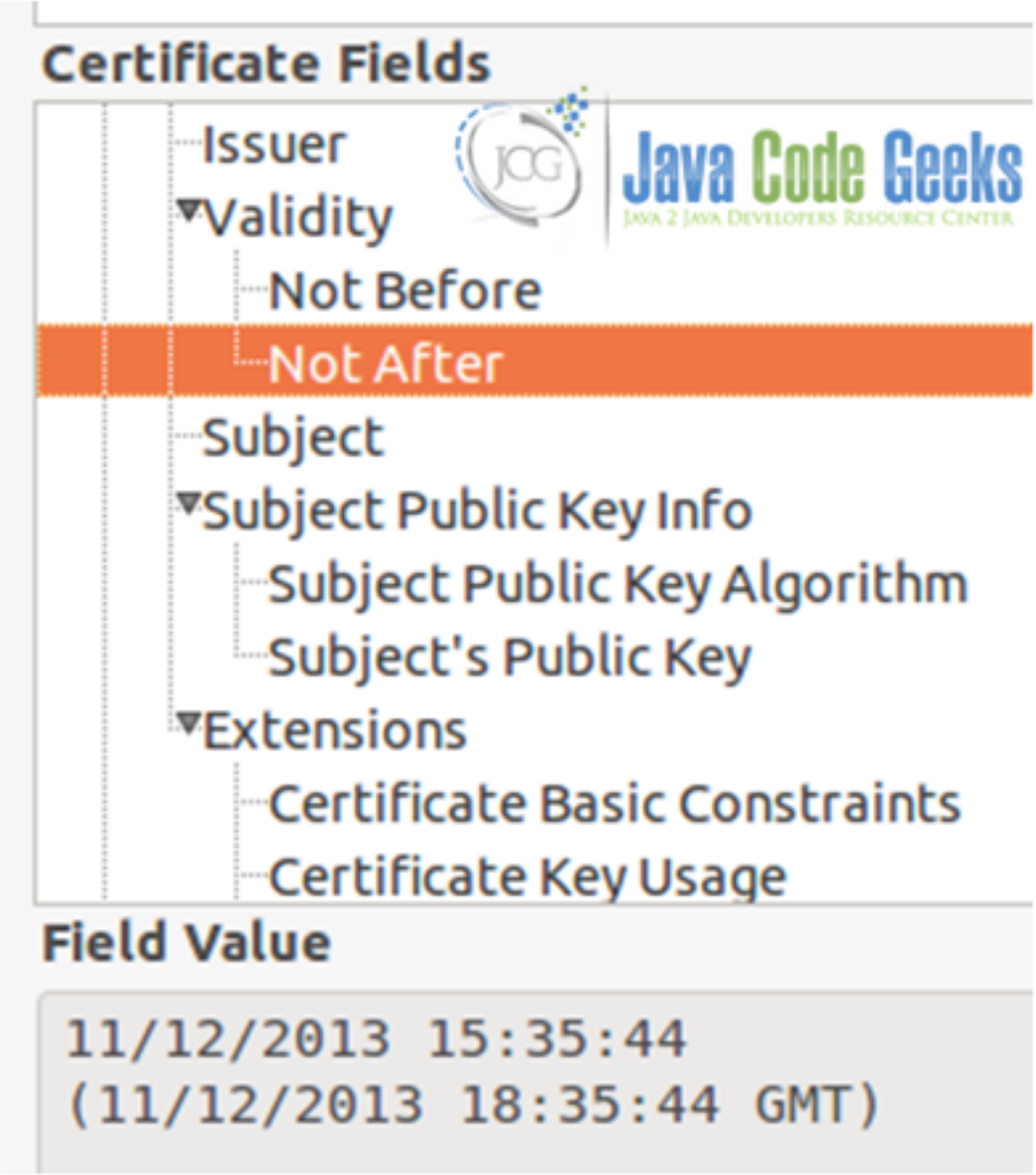

Figure 5.20: screenshot

Once that period of time is over, the certificate will expire (see Fig. 5.21).

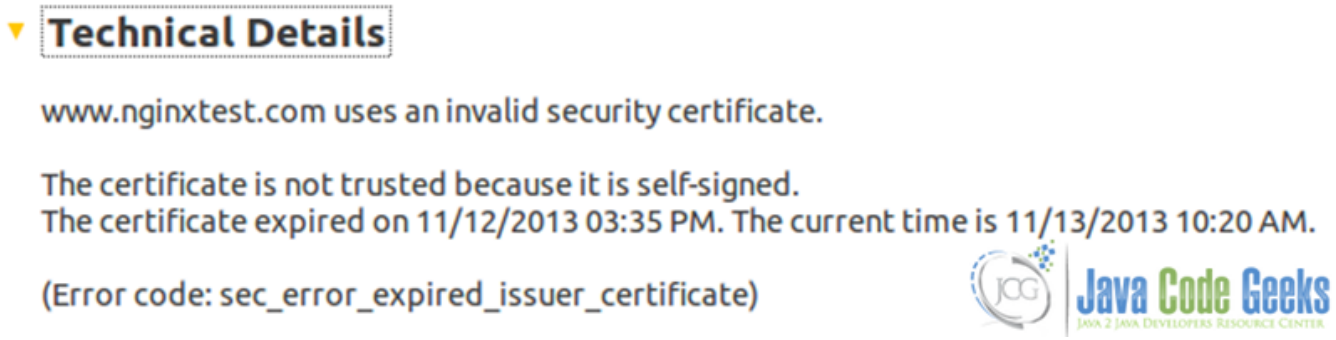

Figure 5.21: The SSL certificate expired

Unless you have a certificate from a trusted third party, your users will get the same warning as above in Fig. 5.13. Aside from the fact that it looks unprofessional, it's also a real risk due to the fact that in a large organization, it is not likely that all users will know the difference between a legitimate key generated by your IT department, and keys generated by a malicious third party. In this case, you need to buy a certificate from a trusted third party, such as GeoTrust (used by Google), Digicert, Comodo, Thawte, or VeriSign (used by Facebook, for example). For other cases, especially when you want to use https for your own use (i.e. securing your admin panel), a self-signed certificate will do just fine.

5.4 Download the files

Here you can download the configuration files used in this tutorial: ConfigFile.zip

Also you can download a useful pdf file: Openssl.pdf

Chapter 6

Nginx Websockets proxying guide

6.1 Introduction

According to the RFC (Request For Comments) #64551, issued by the IETF (Internet Engineering Task Force), WebSocket is a protocol providing full-duplex communications channels over a single TCP connection and is designed to be implemented in web browsers and servers, but it can be used by any client or server application.

Like TCP, WebSocket makes full-duplex communication possible in a low latency connection, but it differs from TCP in that it enables a stream of messages instead of a stream of bytes.

In other words, there is a persistent connection between client and server, and any of them can start sending data at any time. This way you will think about using WebSockets whenever you need a near real-time connection between the client and the server, whether it is in a web environment or not.

Please note that Nginx supports Websockets starting in version 1.3.13 (released on February 2013) and is accessible within the core product. Older versions DO NOT support this protocol.

In this tutorial we will also configure `Upstart`, a modern replacement for init.d written by the Ubuntu developers, to make sure that our websocket application (written in Node.js) automatically restarts if it crashes, and starts up when our server boots up.

6.2 Installing Node.js

First off, let's create a directory inside our home to download the source code for Node.js (see Fig. 6.1). Note that for the sake of clarity, this directory is located at the same level that the one where we downloaded the source code for Nginx as explained in Tutorial #1. We will then use the same directory to extract the contents of the tarball and to install Node.js using the regular procedure:

(We are following the usual build process here because the checkinstall method produced an error while building the deb package.)

```
sudo ./configure, sudo make, sudo make install
```

At the time of this writing, the latest version of Node.js is v0.10.21, which can be downloaded (32-bit or 64-bit versions) from http://nodejs.org/download/ using the following command (see Fig. 6.2, where the proxy server being used and its IP address have been blurred for privacy reasons):

```
wget http://nodejs.org/dist/v0.10.21/node-v0.10.21.tar.gz
```

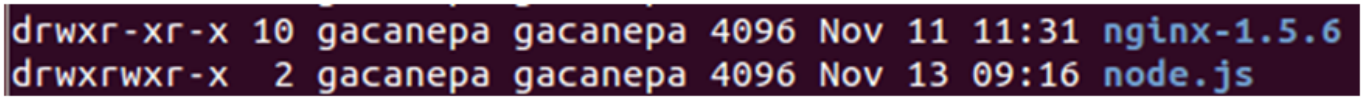

Figure 6.1: screenshot

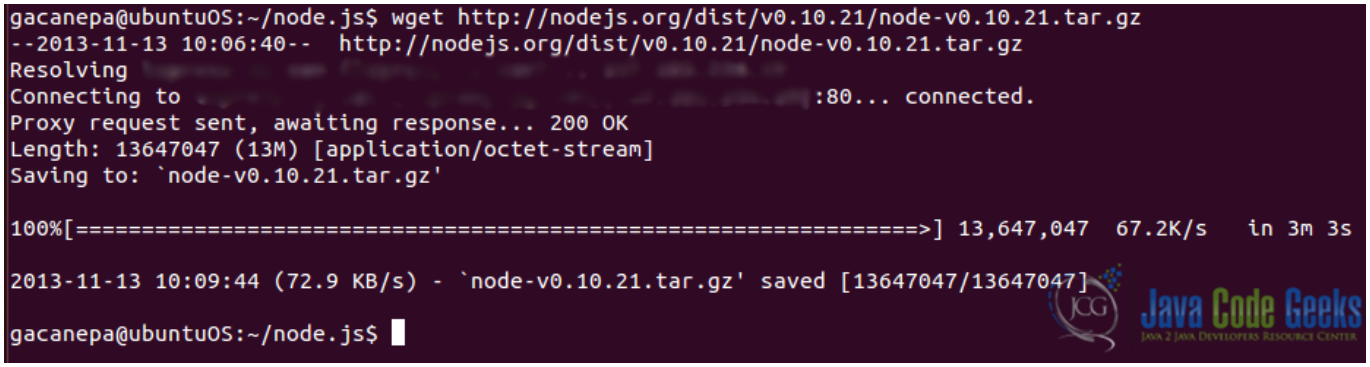

Figure 6.2: screenshot

6.3 Installing additional libraries

Next, we will install `socket.io`, which is a JavaScript library for real-time web applications. It has two parts:

- a client-side library that runs in the browser and
- a server-side library for node.js (Socket.io, Wikipedia) using npm (the official package manager for node.js, `sudo npm install socket.io`, see Fig. 6.3), and express, a web application framework for node.js.

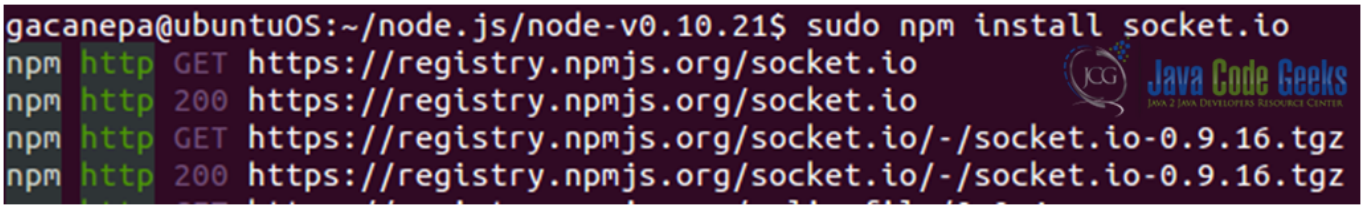

Figure 6.3: Installing the socket.io library

To download the express framework we need to define a couple of settings in a .json file (package.json) located in a directory created for our application - ~/chatroom in this example- (see Fig. 6.4). Then we proceed to download and install it with:

```
npm install -d
```

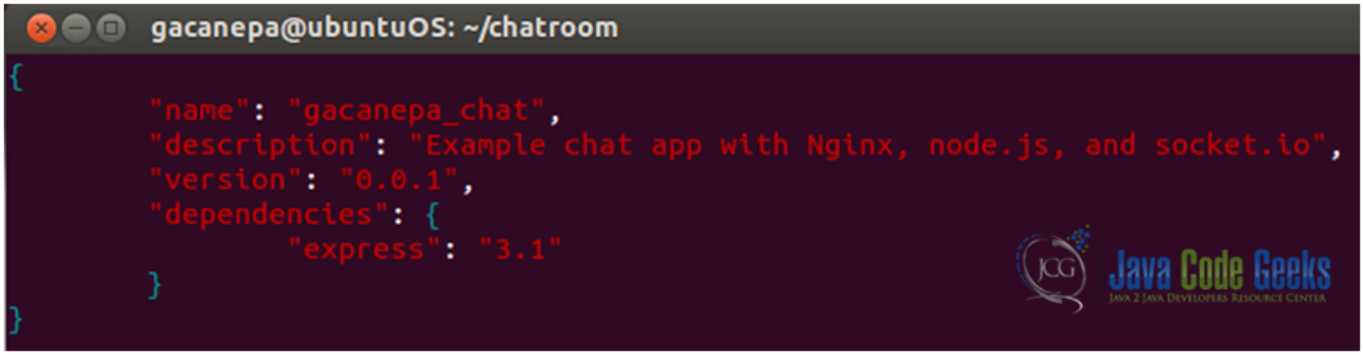

Figure 6.4: The package.json file

Since in this tutorial we are focusing on how Nginx works with websockets instead of programming with node.js, we will not develop a web app from scratch, but we will use an existing one which is available via GitHub (Chatroom example). We need to download the files index.html and app.js (see Fig. 6.5) using the following command:

```
wget https://github.com/mmukhin/psitsmike_example_1/archive/master.zip
```

Figure 6.5: Downloading the chat room web app from GitHub

Then we'll unzip the files mentioned earlier and start the app with node app.js (see Fig. 6.6). We may have to stop Apache if it is running on port 8080.

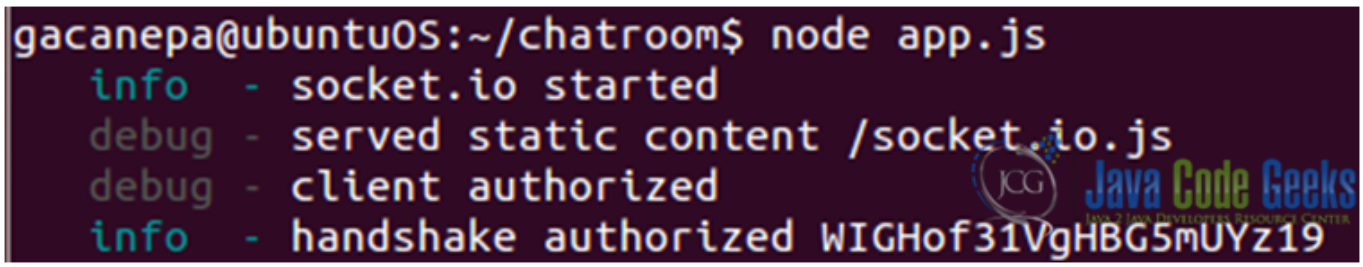

Figure 6.6: Starting the web application

The result (see Fig. 6.7 and 6.8) is not using Nginx yet and we haven't mentioned why we want Nginx with websockets yet, but we're half-way there though. So far we have a chat room-like web application that displays sent and received messages in real-time, along with server responses as well.

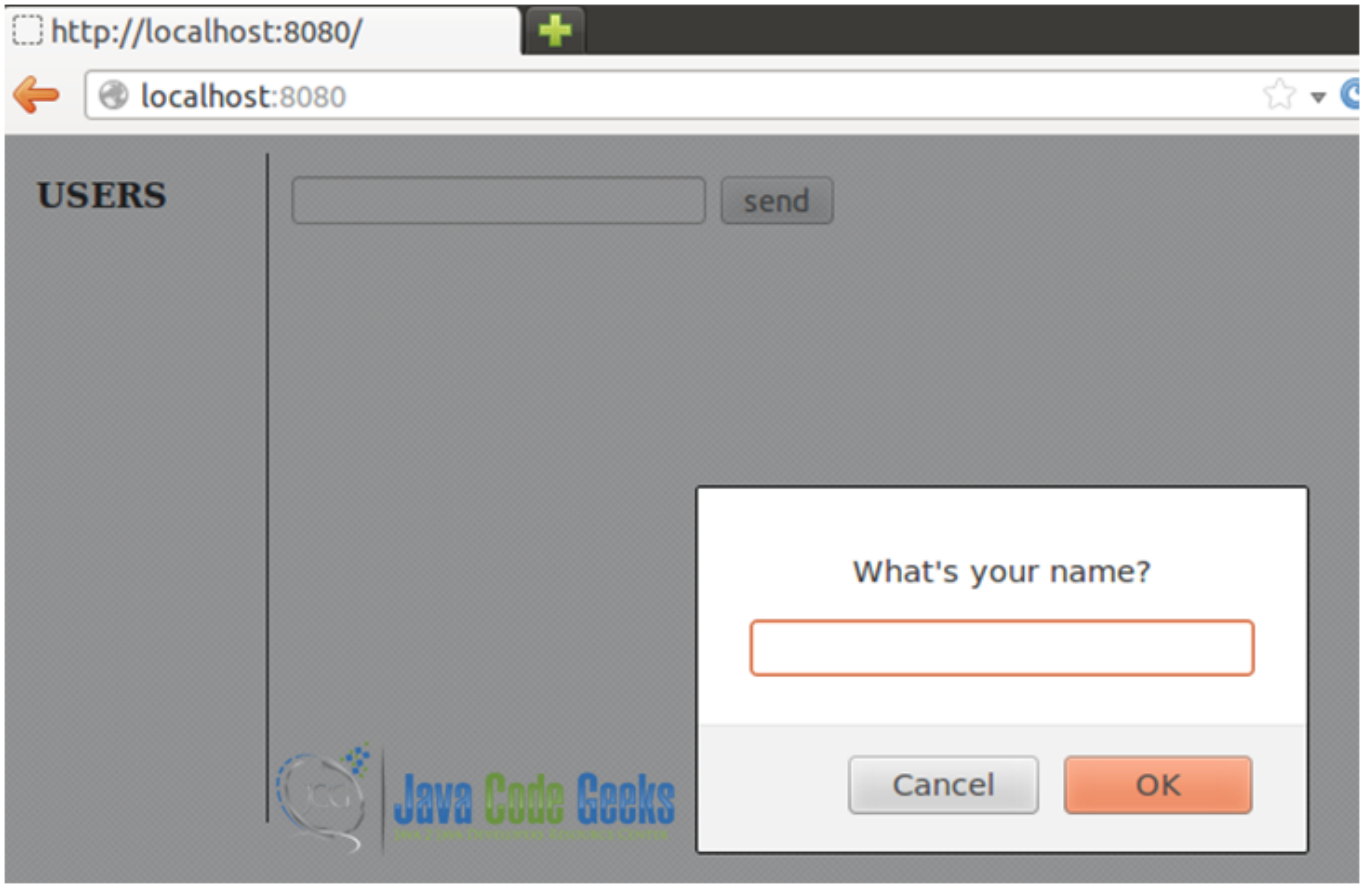

Figure 6.7: screenshot

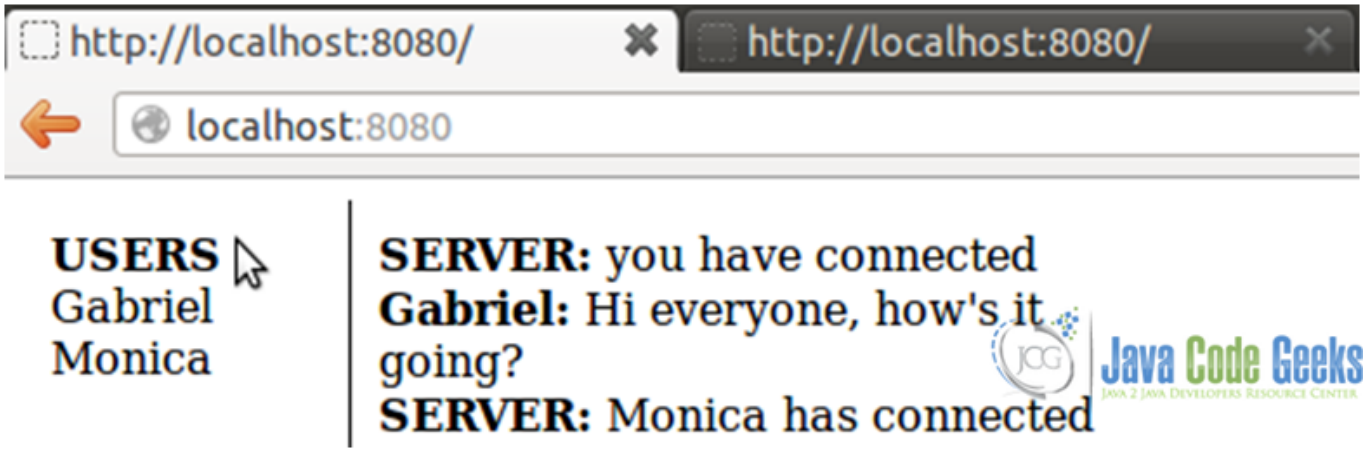

Figure 6.8: screenshot

6.4 So... what does Nginx has to do with all of this?

If we need to use this application on a live (production) environment, we will probably want it to listen on port 80 (most enterprise-level firewalls allow communications through that port). But Nginx is already listening on that port. What do we do

now? As before, our robust Nginx web server has the answer. We will simply forward incoming requests on port 80 (external connections) to another port (8080 in this case for internal connections). This way we are using Nginx as a reverse proxy and the outside world cannot talk to the chat room application directly, but through Nginx, which acts as a frontend server. This scenario will also allow us to use SSL certificates to encrypt traffic.

We will go ahead and edit the nginx.conf file (see Fig. 6.9) adding a few directives from the `proxy` module.

```
server {
     listen       80;
     server_name  nginxtest.com;
     root   /home/gacanepa/chatroom;
     access_log  /home/gacanepa/nginx-1.5.6/html/access_log MyLogFormat;
     error_log   /home/gacanepa/nginx-1.5.6/html/error_log;
     location / {
         proxy_pass http://localhost:8080;
         proxy_http_version 1.1;
         proxy_set_header Upgrade $http_upgrade;
         proxy_set_header Connection "upgrade";
         proxy_set_header Host $host;
         index  index.html index.htm;
     }
 }
```

Figure 6.9: The main configuration file, nginx.conf

We will discuss each directive in detail:

- **proxy_pass** http://localhost:8080 : enables reverse proxying to a backend server by specifying its location (in this case, the same host, port 8080).
- **proxy_http_version 1.1:** sets the HTTP version to be used for communicating with the proxy backend. HTTP 1.0 is the default value, but if we need to enable keepalive connections, it's best to set this directive to 1.1.
- **proxy_set_header**: This directive allows you to redefine header values to be transferred to the backend server. As we can see, it can be declared multiple times.
- **proxy_set_header Host $host**: The Host HTTP header in the request forwarded to the backend server defaults to the proxy hostname, as specified in the configuration file. This setting lets Nginx use the original Host from the client request instead. Refer to tutorial #2 for a complete list of http headers.

Since we want the app would start automatically when the server booted up, we need to manage the Node process with an init script or an upstart supervisor. We will choose the second option in this tutorial. First, we will install `forever`, a very useful tool for running and monitoring Node.js processes (see Fig. 6.10).

```
gacanepa@ubuntuOS: ~/chatroom
gacanepa@ubuntuOS:~/chatroom$ sudo npm install forever
npm WARN package.json gacanepa_chat@0.0.1 No repository field.
npm WARN package.json gacanepa_chat@0.0.1 No README data
```

Figure 6.10: Installing forever

We also need an init script (`app.js.conf` - found in the attached zip file, adapted from ExRatione) in the /etc/init directory to start (see Fig. 6.11) / stop / restart our process (app.js) and if need be, display its status. This script is a generic `upstart` file where we define certain environment variables that are necessary for the script to run (see Fig. 6.12).

```
gacanepa@ubuntuOS:~/chatroom$ sudo service app.js start
app.js start/running, process 3388
gacanepa@ubuntuOS:~/chatroom$ tail -f app.js.log
info: socket.io started
debug: client authorized
info: handshake authorized 19MeBhz6eqqLklU5kbwK
debug: served static content /socket.io.js
debug: client authorized
info: handshake authorized ezjJPhCjByQllFtZkbwL
debug: setting request GET /socket.io/1/websocket/ezjJPhCjByQllFtZkbwL
debug: set heartbeat interval for client ezjJPhCjByQllFtZkbwL
debug: client authorized for
debug: websocket writing 1::
debug: websocket writing 5:::{"name":"updatechat","args":["SERVER","you have connected"]}
debug: broadcasting packet
debug: websocket writing 5:::{"name":"updateusers","args":[{"Gabriel":"Gabriel"}]}
debug: websocket writing 5:::{"name":"updatechat","args":["Gabriel","this is a test message"]}
debug: emitting heartbeat for client ezjJPhCjByQllFtZkbwL
```

Figure 6.11: Our web application is started

```
# The following environment variables must be set so as to define
# where Node.js and Forever binaries and the Node.js source code
# can be found.

# The full path to the directory containing the node and forever binaries.
env NODE_BIN_DIR="/usr/local/bin"
# Set the NODE_PATH to the Node.js main node_modules directory.
env NODE_PATH="/home/gacanepa/chatroom/node_modules"
# The directory containing the application Javascript file.
env APPLICATION_DIRECTORY="/home/gacanepa/chatroom"
# The application start Javascript filename.
env APPLICATION_START="app.js"
# Log file path.
env LOG="/home/gacanepa/chatroom/app.js.log"
```

Figure 6.12: Environment variables defined by the upstart script

In the meanwhile, what we write in our chat room (see Fig. 6.13) is saved in the log of our application and we can see it in real-time, both in the web interface and from the command line using the command:

```
tail -f app.js.log
```

Notice that we are accessing our web application using Nginx as our frontend server (as opposed to the case shown in Figs. 6a and 6b, where the web application is run by node.js exclusively running on port 8080).

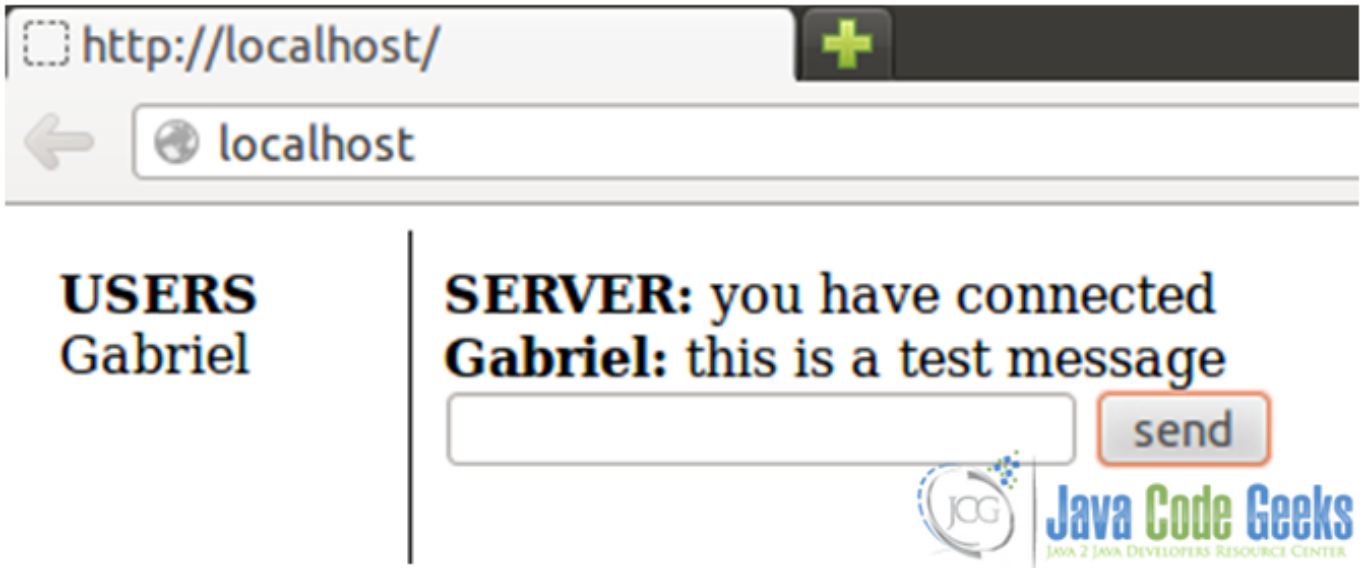

Figure 6.13: screenshot

6.5 Download source files

You can download the source files of this tutorial: WebsocketsExample.zip