nginx on Slackware ARM

Nginx in brief

Created by Igor Sysoev and first publicly released in 2004, Nginx is a high-performance, highly scalable, highly available web server, reverse proxy server, and web accelerator (combining the features of an HTTP load balancer, content cache, and more). Nginx offers a highly scalable architecture that is very different from that of Apache (and many other open source and commercial products in the same category). Nginx has a modular, event-driven, asynchronous, single-threaded architecture that scales extremely well on generic server hardware and across multi-processor systems. Nginx uses all of the underlying power of modern operating systems such as Linux to optimize the usage of memory, CPU, and network, and extract the maximum performance out of a physical or virtual server. According to the Nginx FAQ, the end result is that Nginx can often serve **at least** 10x more (and often 100–1000x more) requests per server compared to Apache – that means more connected users per server, better bandwidth utilization, less CPU and RAM consumed, and a greener environment too!

Nginx has gained support and popularity over the years and [as of February 2020] 31.7% of all known web servers running on the Internet use Nginx, compared to 40.5% for Apache, and 7.8% for Microsoft-IIS. See: https://w3techs.com/technologies/comparison/ws-apache,ws-microsoftiis,ws-nginx

Notes

This page will hopefully allow you how run nginx on Slackware ARM successfully, from scratch. There's no reason what-so-ever that the same procedure(s) shouldn't work on Slackware x86_64 [or Slackware ARM 14.2], but for the purpose of testing it has been carried out under Slackware ARM -current [23 February - 05 March 2020].

Slackware ARM -current was running [kernel 5.4.22] on a Raspberry Pi 4 and used to build the package, install and configure nginx. Any ARM device running Slackware may be used for this purpose. The -current (development) version was chosen as it includes the latest software, etc.

Requirements

Please expect and acknowledge the following before going ahead;

- You'll use the CLI in a shell for the entire process.
- You'll edit config files using a text editor of your choosing; elvis, vim, nano, etc.
- Apache (httpd) will be disabled/removed from your system in order for nginx to run. [i.e. they both use port:80]

Building the nginx package

In order to build the nginx package, first you need to download the SlackBuilds.org nginx files and put

them in a folder on your system. We will create and be using '/tmp/nginx' directory for this purpose, as 'root' user.

```
~# mkdir -p /tmp/nginx
~# cd /tmp/nginx
~# wget https://slackbuilds.org/slackbuilds/14.2/network/nginx/README
~# wget https://slackbuilds.org/slackbuilds/14.2/network/nginx/doinst.sh
~# wget
https://slackbuilds.org/slackbuilds/14.2/network/nginx/nginx.SlackBuild
~# wget https://slackbuilds.org/slackbuilds/14.2/network/nginx/nginx.info
~# wget
https://slackbuilds.org/slackbuilds/14.2/network/nginx/nginx.logrotate
~# wget https://slackbuilds.org/slackbuilds/14.2/network/nginx/rc.nginx
~# wget https://slackbuilds.org/slackbuilds/14.2/network/nginx/slack-desc
```

Now make the 'nginx.SlackBuild' build script exectuable.

```
~# chmod +x nginx.SlackBuild
```

You also need to download the latest nginx mainline version and put it in the same directory as the SlackBuild files. At the time of writing this guide the latest version was nginx-1.17.9 but it's best to check if an updated version exists.

```
~# wget http://nginx.org/download/nginx-1.17.9.tar.gz
```

After downloading the latest nginx version you need to edit the 'nginx.SlackBuild' script to tell it which nginx version package you want to create. We'll be using 'nano' for this purpose.

```
~# nano nginx.Slackbuild
```

Find the following line:

```
VERSION=${VERSION:-1.12.2}
```

... and change it to the version of nginx you have downloaded. For 'nginx-1.17.9' it would be like this:

```
VERSION=${VERSION:-1.17.9}
```

The 'nginx.SlackBuild' script by default is set to build a *.tgz package and we prefer to build *.txz packages. If you want to do the same then edit the last line of the script to look like this:

```
/sbin/makepkg -l y -c n $OUTPUT/$PRGNAM-$VERSION-$ARCH-
$BUILD$TAG.${PKGTYPE:-txz}
```

Note here that the only change in this line of code is the '${PKGTYPE:-tgz}' to '${PKGTYPE:-txz}' variable [NB: xz gives a better compression ratio than gzip, which is why we prefer it]. Nothing more needs changing.

Save and exit the file.

It's now time to create the Slackware ARM package. So, without further ado, do it like this:

```
~# time ./nginx.SlackBuild
```

Alternatively, if you plan on using Maxmind's GeoIP database with 'nginx' then GeoIP support is available as an option. All that's required is the GeoIP package to have been installed first. If you wish to enable GeoIP support in 'nginx' then pass the GEOIP variable to the slackbuild command, like this:

```
~# time GEOIP=yes ./nginx.SlackBuild
```

After a few short minutes, you should see the resulting package in your '/tmp' directory. [NB: if you didn't modify the '${PKGTYPE:-tgz}' variable then the file extension will be '*.tgz' instead of '*.txz'.] Check it like this:

Q. Why do we use 'time' in combination with running the build script?
A. We like to know how long the process takes. That's all. You don't have to include it in order to be successful.

```
~# cd /tmp
~# ls -lah *.txz
-rw-r--r-- 1 root root 442K Feb 26 06:01 nginx-1.17.9-arm-1_SBo.txz
```

So, you now have the Slackware ARM nginx package which can easily be installed.

Installing the nginx package

Before installing the Slackware ARM nginx package it is prudent to deal with Apache (httpd). Apache and nginx cannot run together on port:80. So, in order to use nginx as your web server software, Apache must be disabled completely.

First make sure Apache (httpd) is not running. If it is then stop the daemon, like this:

```
~# apachectl stop
```

Then prevent the 'httpd' daemon from starting up on (re)boot by making the initialisation script non-executable, and check the file permissions, like this:

```
~# chmod -x /etc/rc.d/rc.httpd
~# ls -lah /etc/rc.d/rc.httpd
-rw-r--r-- 1 root root 1.1K Jan 13 19:26 /etc/rc.d/rc.httpd
```

Now you can install the nginx package:

```
~# installpkg nginx-1.17.9-arm-1_SBo.txz
```

Configuring nginx to run on Slackware ARM

First of all, after installing the Slackware ARM nginx package, we're going to make the rc.nginx AND rc.php-fpm [system initialization] files executable. We do this now because we intend to serve .php, as well as .html, files from our web server file directories. Plus, we'll be configuring the php FastCGI implementation as well, all at the same time. 😉

```
~# chmod +x /etc/rc.d/rc.php-fpm
~# chmod +x /etc/rc.d/rc.nginx
```

In order to start these system initialization scripts on (re)boot, add an entry to the rc.local file. Just add 2 lines to to the end of the file, like this:

```
~# nano -w /etc/rc.d/rc.local

# Add these two lines below
/etc/rc.d/rc.php-fpm start
/etc/rc.d/rc.nginx start
```

Save and exit the file. Now, whenever you boot or restart your system, 'nginx' and 'php-fpm' will be executed automatically.

Next, we'll edit the '/etc/nginx/nginx.conf' file. The settings we used are for a test setup and as a "*proof of concept*" on Slackware ARM. Your requirements may be different but for the purpose of this guide, and getting a feel of nginx, this is how we did it.

```
~# nano -w /etc/nginx/nginx.conf
```

Things to note:

- Nginx will run under the '**nobody**' user and with those user privileges.
- Nginx PID location will be: **/var/run/nginx.pid**
- Our web directory in which the html files will be located is '**/var/www/html**'. Yours may be in a different location.
- PHP root location is the same as the web directory: '**/var/www/html**'
- Nginx access and error logs will be located in '**/var/log/nginx/**' directory.
- Log format has been configured as '**vcombined**' because we intend to use GoAccess for further tesing on more than 1 site on our nginx web server.
- The location index starts with '**index.php**' followed by 'index.html index.htm'.
- PHP fastcgi_index is: **index.php**
- FastCGI server configuration is included. See section: '# pass the PHP scripts to FastCGI server ...'
- FastCGI 'SCRIPT_FILENAME' is: **$document_root$fastcgi_script_name**

The nginx.conf file content should look similar to the following:

```
user  nobody;
worker_processes  auto;
pid /var/run/nginx.pid;
```

```
# if you wish to load nginx modules create the dir and uncomment the line
below
#include /etc/nginx/modules-enabled/*.conf;

events {
    worker_connections  1024;
    #multi_accept on;
}

http {
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    keepalive_timeout  65;
    types_hash_max_size 2048;

    include       mime.types;
    default_type  application/octet-stream;
    ##
    # Logging Settings
    ##

    log_format vcombined '$host:$server_port '
           '$remote_addr - $remote_user [$time_local] '
           '"$request" $status $body_bytes_sent '
           '"$http_referer" "$http_user_agent"';

    access_log /var/log/nginx/access.log vcombined;
    error_log /var/log/nginx/error.log;

    ##
    # Gzip Settings
    ##
    gzip on;

    ##
    # Web Server Settings
    ##

    server {
        listen       80 default_server;
        server_name  127.0.0.1 localhost;

        location / {
            root   /var/www/html;
            index index.php index.html index.htm;
        }

        # redirect server error pages to the static page /50x.html
        #
        error_page   500 502 503 504  /50x.html;
```

```
        location = /50x.html {
            root   /var/www/html;
        }

        # pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000
        #
        location ~ \.php$ {
            root           /var/www/html;
            fastcgi_pass   127.0.0.1:9000;
            fastcgi_index  index.php;
            fastcgi_param  SCRIPT_FILENAME
$document_root$fastcgi_script_name;
            include        fastcgi_params;
        }

        # deny access to .htaccess files, if Apache's document root
        # concurs with nginx's one
        #
        location ~ /\.ht {
            deny  all;
        }
    }

    include /etc/nginx/conf.d/*.conf;

}
```

Check that your settings in the nginx.conf file are correct. It's very important. Save and exit the file when you're done.

Configuring php-fpm.conf

To configure 'php-fpm' we'll copy the '/etc/php-fpm.d/www.conf.default' file and then edit the copy. Like this:

```
~# cp -av /etc/php-fpm.d/www.conf.default /etc/php-fpm.d/www.conf
~# nano -w /etc/php-fpm.d/www.conf
```

The default Unix user/group in this file is 'apache' but you need to change these settings. Find the section shown below and edit the 'user' and 'group' to match the following code:

```
; Unix user/group of processes
; Note: The user is mandatory. If the group is not set, the default user's
group
;       will be used.
user = nobody
group = nogroup
```

By default 'php-fpm' does not log anything other than startup and shutdown [see: '/var/log/php-

fpm.log' after the daemon has been (re)started]. In order to log any relevant error messages we need to make one more change to the file. Find the 'catch_workers_output“ section below and uncomment the setting, like this:

```
; Redirect worker stdout and stderr into main error log. If not set, stdout
and
; stderr will be redirected to /dev/null according to FastCGI specs.
; Note: on highloaded environement, this can cause some delay in the page
; process time (several ms).
; Default Value: no
catch_workers_output = yes
```

Save and exit the file.

Create a test index.php

Create an 'index.php' file to test the nginx web server and make sure it's working. This file will be located in the web directory where all the other .php and .html files reside [i.e. the PATH specified in the '/etc/nginx/nginx.conf' file].

```
~# nano -w /var/www/html/index.php
```

Enter the following code into the index.php file:

```
<?php
  phpinfo();
?>
```

Save and exit this file. This 'index.php' file will display the standard 'phpinfo' output which is great for testing purposes such as ours.

- There will be an 'index.html' file in the same folder as you have saved the 'index.php' [put there when nginx was installed] and this can also be used for testing.

Starting nginx and php-fpm

Now we can start 'php-fpm' and 'nginx'. Use these commands to do so:

```
~# /etc/rc.d/rc.php-fpm start
~# /etc/rc.d/rc.nginx start
```

Important Notes To Remember

When you have modified ANY nginx configuration file(s) or setting(s) you **MUST** test the nginx configuration and restart the daemon. Running 'nginx -h' will output a list of command options.

To test the the nginx configuration, do this:

```
~# nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

To restart the nginx daemon do this:

```
~# /etc/rc.d/rc.nginx restart
Shutdown Nginx gracefully...
Starting Nginx server daemon...
```

Perform the above each time you change or modify the nginx configuration.

Testing the nginx web server

If all has gone according to plan, you'll now be able to access the web-content in your '/var/www/html' directory. To test this you can use the following command:

```
~# links http://localhost
```

You should be looking at the output of the index.php page you created earlier. The 'index.php' displays instead of 'index.html' because it was specified in the index list first. If you don't want this arrangement then adjust the '/etc/nginx/nginx.conf' file with your own preferences.

To quit 'links' press the Q key and Enter.

You can view the 'index.html' page which was created by nginx, like this:

```
~# links http://localhost/index.html
```

If you can see both these pages then the nginx web server is working as expected.

Stopping nginx and php-fpm

Should you need to disable the 'rc.nginx' and 'rc.php-fpm' daemons, use the following commands:

```
/etc/rc.d/rc.php-fpm stop
/etc/rc.d/rc.nginx stop
```

To permanently disable the daemons, remove or comment the line(s) in your '/etc/rc.d/rc.local' file.

Obviously this is not in any way a guide on how to fully take advantage of what nginx offers. The result of this guide should be confirmation that Nginx works "*like a boss*" on Slackware ARM. From there the possibilities are endless. Nginx can be used with multiple domains, as a load-balancer, and for many other purposes. In order to explore the many ways in which nginx can be utilised take a look at the main Nginx.org

website.

GoAccess configuration

We created this guide with GoAccess in mind. So here's why...

"GoAccess was designed to be a fast, terminal-based log analyzer. Its core idea is to quickly analyze and view web server statistics in real time without needing to use your browser (great if you want to do a quick analysis of your access log via SSH, or if you simply love working in the terminal)." See: https://github.com/allinurl/goaccess

If you want to take advantage of open source web server analysis software then it doesn't come much better than GoAccess. Fortunately, SlackBuilds.org has a repository for GoAccess and it can be built and installed with relative ease. If you're familiar with the SlackBuilds process then creating and installing a goaccess package is basically the same as it is for nginx.

Once installed, to configure GoAccess, edit the '/etc/goaccess.conf' file and uncomment the lines below; time-format, date-format, log-format.

```
# The following time format works with any of the
# Apache/NGINX's log formats below.
#
time-format %H:%M:%S

# The following date format works with any of the
# Apache/NGINX's log formats below.
#
date-format %d/%b/%Y

# NCSA Combined Log Format with Virtual Host
log-format %v:%^ %h %^[%d:%t %^] "%r" %s %b "%R" "%u"
```

Then save and exit the file.

Now we should be able to view [instantly-updated] LIVE stats on our nginx web server activity by using GoAccess and the following command:

```
~# goaccess -f /var/log/nginx/access.log --log-format=VCOMBINED
```

To exit GoAccess press the Q key.

We can create a static html page with GoAccess using the following command:

```
~# goaccess -f /var/log/nginx/access.log -a -o /var/www/html/static-log.html
--log-format=VCOMBINED
```

We saved the resulting page in our web directory. So, now we can view that static html page using 'links', like this:

```
~# links http://127.0.0.1/static-log.html
```

GoAccess is very versatile and configurable, and open source. It can parse a great many log formats [logs can be rotated] and can also create static html pages as well as being fully functional in a shell.

Adding additional sites to nginx

In this guide we've only used one web directory ['/var/www/html'] but in order to expand our single site we're going to add another two sites. This is achieved by adding *.conf files to the '/etc/nginx/conf.d/' directory. You can name them anything you like as long as the file extension is ".conf". For the purposes of this guide we'll configure 'test.local' and 'extra.local' along with our existing 'localhost' site.

First we create the '/var/www/test.local' and '/var/www/extra.local' directories where our web content will reside:

```
~# mkdir /var/www/test.local
~# mkdir /var/www/extra.local
```

Because we want to use 'test.local' and 'extra.local' as domain name [server_name] aliases we will add the following to our '/etc/hosts' file:

```
# For loopbacking.
127.0.0.1               localhost
127.0.0.1               myhost.name myhost
127.0.0.1               test.local
127.0.0.1               extra.local

# End of hosts.
```

The above settings will take effect as soon as the file is saved. However, there's nothing there to see yet - because we haven't created it! 

So, then we create and edit the 'test.local.conf' file:

```
~# nano -w /etc/nginx/conf.d/test.local.conf
```

In this file we need to instruct 'nginx' which server requests it should listen for on a specific port. We need to specify the path/to/the web directory files where this site will be located and we'll also configure the FastCGI server. So we'll enter the following text into this file:

```
server {
    listen 80;
    server_name test.local;

    # this root path is to the webdir
    root /var/www/test.local;
    # index pages to load in this order
    index index.php index.html index.htm;
```

```
    #This will be needed for GoAccess ** make sure vcombined map is defined
in nginx.conf
    access_log /var/log/nginx/access.log vcombined;

    # pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000
    #
    location ~ \.php$ {
       root           /var/www/test.local;
       fastcgi_pass   127.0.0.1:9000;
       fastcgi_index  index.php;
       fastcgi_param  SCRIPT_FILENAME  $document_root$fastcgi_script_name;
       include        fastcgi_params;
    }

}
```

Again, we are keeping the access_log format in order to make use of web traffic analysis with GoAccess. Save and exit the file when you're happy with the setup.

Now we can create and edit the 'extra.local.conf' file in much the same way as the first file:

```
server {
    listen 80;
    server_name extra.local;

    # this root path is to the webdir
    root /var/www/extra.local;

    # index pages to load in this order
    index index.php index.html index.htm;

    #This will be needed for GoAccess ** make sure vcombined map is defined
in nginx.conf
    access_log /var/log/nginx/access.log vcombined;

    # pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000
    #
    location ~ \.php$ {
       root           /var/www/extra.local;
       fastcgi_pass   127.0.0.1:9000;
       fastcgi_index  index.php;
       fastcgi_param  SCRIPT_FILENAME  $document_root$fastcgi_script_name;
       include        fastcgi_params;
    }

}
```

Note that port:80 is still being used in both cases, as is the existing 'localhost' site, but the server_name and root path are different to reflect that this is a seperate entity. The server_name is how 'nginx' will know which site to direct you to when it is requested. You can of course use other ports. For example, if you required 'test.local' to be accessible on port:1080 then just specify that with

a 'listen 1080;' parameter and access it as 'http://test.local:1080'.

The web directories need files in them in order to display any content. So, as before, you could create a simple 'index.php' file and put it in the directory that relates to the new site(s). Or you could crate a simple 'index.html' file to identify the site. For example:

```
~# nano -w /var/www/test.local/index.php

<?php
  phpinfo();
?>
```

And/or...

```
~# echo '<html><body><h1>TEST : It works!</h1></body></html>' >
/var/www/test.local/index.html
~# echo '<html><body><h1>EXTRA : It works!</h1></body></html>' >
/var/www/test.local/index.html
```

When all that's complete, the final task is to test the 'nginx' configuration and restart the daemon.

```
~# nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
~# /etc/rc.d/rc.nginx restart
Shutdown Nginx gracefully...
Starting Nginx server daemon...
```

So now, to access the new site content on your 'nginx' web server use the following, remembering to request the file you've previously created. For example:

```
~# links http://test.local/index.php
~# links http://extra.local/index.html
```

One thing to note is that if you use 'http://127.0.0.1/index.html' it will direct you to the 'localhost' site because that has been specified as the default in '/etc/nginx/nginx.conf'. In order to acess the content of any sites other than 'localhost' or '127.0.0.1' [default] the server_name must be used. You can also access the site remotely using the IP address [and port - if specified] of your ARM device instead of a domain name. Or you could add a domain name alias to your 'hosts' file and use that instead. Either method works.

Sources

Software used in this guide:
http://slackware.uk/slackwarearm/slackwarearm-current/ # Slackware ARM -current OS.
https://nginx.org/download/ # Nginx 1.17.9 web server software [gzip tarball].
https://slackbuilds.org/slackbuilds/14.2/network/nginx/ # SlackBuilds.org nginx package build script files.

https://slackbuilds.org/repository/14.2/system/goaccess/ # SlackBuilds.org goaccess package build script files.
https://goaccess.io/download # GoAccess 1.3 web server analysis software [gzip tarball].
https://sarpi.fatdog.eu/index.php?p=downloads # SARPi Project pre-built [txz] packages.

Documentation which assisted in this guide:
https://w3techs.com/technologies/comparison/ws-apache,ws-nginx # Web server software usage statistics.
https://www.nginx.com/resources/faq/ Nginx.com FAQ.
https://nginx.org/en/docs/faq.html Nginx.org FAQ.
https://www.aosabook.org/en/nginx.html # Overview of nginx architecture.
https://goaccess.io/faq # GoAccess FAQ.
https://github.com/allinurl/goaccess # GoAccess GitHub README.

Special thanks:
Aal for suggesting Nginx/GoAccess on Slackware ARM and help/advice with configs [on Debian].
MoZes for making a lot of Slackware ARM things clearer and more understandable via Slackchat Podcast.

* Originally written by Exaga - 2020-02-25 20:06:09 [GMT]

howtos, slackware, arm, nginx, php, fpm, fastcgi, web, server, goaccess, slackbuild, author exaga