How To Configure Varnish for Drupal with Apache on Debian and Ubuntu

Authored by: ***ASPHostServer Administrator*** *[asphostserver@gmail.com]*
Saved From: *http://faq.asphosthelpdesk.com/article.php?id=246*

Drupal is an open-source content management framework and platform written in PHP. Used for building rich, back-end based solutions for websites of any size and web applications alike, it is massively popular with extremely high statistics of adoption rates. Its distribution under the GNU General Public License means that anyone is free to use the software, contribute to the project in various ways and share it with others. The main distribution of Drupal is called Drupal Core and as of October 2013, it is expandable with over 20 thousand modules and almost 2 thousand themes.

Taking advantage of platforms like Drupal to quickly prototype and develop applications is great. However, just as quickly, issues such as tackling growth of your product start to arise. This will require you to scale rapidly to continue to serve your customers fast, in order to keep their interest in your product and to maintain their happiness.

In this article in a series to*help developers with scaling*, we will be talking about configuring Varnish with Drupal to greatly reduce the amount of time it takes users to load your Drupal based website and increase the simultaneous serving capacity by passing requests through the Varnish Cache software first.

Scaling

When talking about scaling in terms of software, what people usually mean is [a system's] ability to handle and accommodate a larger-than-its-built-for amount of work (which is to be processed) and data *which is to be kept and used in real-time or later). Regarding networks, the definition and identification of the issue gets simpler, despite a thorough solution usually being more costly to implement as it can require further incorporation of hardware to the system.

The two most common approaches of scaling a webpage (or a web application) are: **Vertical** and/or **Horizontal** scaling.

Vertical Scaling (Scaling Up!)

Simply put, Vertical Scaling means increasing the resources of that one, single computer which is our server or of any of the computers forming our system as a whole vertically, by adding more resources to them or increasing their capacity. Adding more memory, replacing the CPU with a faster one, getting higher capacity and faster drives with excellent read-write speeds can all be considered as Vertical Scaling. Sometimes, this is also called **Scaling Up**.

Horizontal Scaling (Scaling Out!)

When we decide to take advantage of excellent low pricing of today's private servers and decide to incorporate more of them in our system, it is called Horizontal Scaling, or **Scaling Out**. This could happen in

a variety of ways, such as running each software of our application stack on its own machine or cloning the web servers to create several others of the same and distributing incoming requests from clients across them by making use of a reverse proxy, such as NGINX. For more information on reverse proxies visit http://en.wikipedia.org/wiki/Reverse_Proxy.

Varnish

Sitting in front of your website, **Varnish Cache** deals with serving static or quasi-static content directly, without passing the request back to the web server (i.e. Apache) to process over and over again. Since a lot of content (despite the amount of time they will be accessed) needs to be computed and generated only once, storing and then serving them from the fast access memory greatly reduces the load bore by the web server and increases the amount of requests that can be simultaneously handled by our system as a whole.

How Can*Varnish* Help?

Using Varnish in (almost) any web site means a faster web site by multitudes of force (depending on the architecture, of course!). It also means a more reliable product (for reasons you will see during configuration) thanks to the extended functionality of Varnish Cache. All this translates to happier customers who enjoy using your product more and a system that you can better count on to deliver.

To Note: Unless we use a separate application server instance for Varnish (which we should, for many*good* reasons!) this is technically not scaling (neither up, nor out), but having a good system architecture and basically being smart. However, due to the increase of load that can be handled by a single server, this could be referred to as scaling in some ways, as we are going to have much better overall performance from our system. This is really the result expected from scaling, and we end up with a system architecture that is ready to truly scale if necessary; however, it must be noted that better performance alone does not mean scaling.

1. Preparing Your Web Server for Varnish

In this article, our goal is to set up Varnish to serve pages quickly, directly from the memory. This is called **caching** and for this to work, Varnish needs to be able to handle the incoming requests first. Given that your web server (ex. Apache) is set for doing the exact same thing (handling incoming requests), we need to make some modifications for Varnish to take its place. The way to achieve this is to modify the port it listens to (i.e. port 80).

Modifying Apache

By default, Apache runs on (listens to) port 80 and we need to modify that.

Modifying Apache's settings on *ports.conf*:

The port setting we need to modify is defined in a file called`ports.conf`which resides in`/etc/apache2/` folder.

Let's modify that using a text editor. We will be working with*nano* here, which is known for its user friendliness compared to some others.

Run the following command to open the editor to edit the contents of *ports.conf* file:

```
sudo nano /etc/apache2/ports.conf
```

Inside this file (by default) you should see something similar to:

```
NameVirtualHost *:80
Listen 80
<IfModule mod_ssl.c>
    # If you add NameVirtualHost *:443 here, you will also have to change
    # the VirtualHost statement in /etc/apache2/sites-available/default-ssl
    # to <VirtualHost *:443>
    # Server Name Indication for SSL named virtual hosts is currently not
    # supported by MSIE on Windows XP.
    Listen 443
</IfModule>
<IfModule mod_gnutls.c>
    Listen 443
</IfModule>
```

So go ahead and modify the first two lines where the port number is specified, from its original to, say, **8000** (and remember this number):

```
NameVirtualHost *:8000
Listen 8000
```

Now in order to save and close, press `CTRL+ X` and when prompted, type `Y` and then press enter. This is is going to save the file.

If you also have _Virtual Host(vhost) Configurations_

In case you are hosting multiple websites (or have domain specific options set), you are likely to have*Virtual Host*configurations which we need to update as well.

Run the following command to open the editor to edit the settings for your site:

```
sudo nano /etc/apache2/sites-availbable/my-domain-dot-com
```

***!!** Do not forget to replace **my-domain-dot-com** with the appropriate name.*

***!!** If you do not have custom named Virtual Host configurations, you can find the default settings on `/etc/apache2/sites-available/default`.*

Depending on your current settings, you will see a document, beginning with (or containing) something like the following:

```
<VirtualHost *:80>
```

Which we now need to replace with*same port from ports.conf*:

```
<VirtualHost *:8000>
```

Close and save the file by pressing `CTRL+X` and then typing `Y` and pressing enter the same way.

In order to bring changes to effect, we need to restart Apache:

```
sudo service apache2 reload
```

As of now, Apache will accept incoming requests on the new set port of 8000.
And we are ready to move to the next step!

2. Installing Varnish Cache

For Debian and Ubuntu:

Setting up Varnish on Debian and Ubuntu is quite simple, as it is distributed with default system package manager:*apt*. However, the recommended way, in order to ensure that we are getting the latest version, is as follows (ref. https://www.varnish-cache.org/installation/debian):

We need to add download URL for Varnish to aptitude package manager's sources list. In order to verify the source, we first need to add the security key provided by http://repo.varnish-cache.org.

Let's begin with adding the security key [Debian and Ubuntu]:

```
wget http://repo.varnish-cache.org/debian/GPG-key.txt
apt-key add GPG-key.txt
```

Now add the package URL to apt-get repository sources list.

Debian:

```
echo "deb http://repo.varnish-cache.org/debian/ wheezy varnish-3.0" >>
/etc/apt/sources.list
```

Ubuntu:

```
echo "deb http://repo.varnish-cache.org/ubuntu/ precise varnish-3.0" | sudo tee
-a /etc/apt/sources.list
```

Finally, let's update the package manager and download/install Varnish Cache [Debian and Ubuntu]

```
apt-get update
apt-get install varnish
```

3. Modifying Varnish's Settings

After having Apache's port modified and Varnish installed, we are ready to continue with modifying the settings of Varnish in order for it to behave the way we need it to.

Putting Varnish into Production Mod

As one of the reasons of enabling administrators to test Varnish upon installation, the default settings are*not* set to run on front-facing port of 80, and we need to change that.

The configuration file on Debian and Ubuntu reside at`/etc/default/varnish`.

Open the editor to edit the file:

```
nano /etc/default/varnish
```

Upon running this command, you will be faced with a rather long but self-explanatory document. If you scroll down, you will see a block of text defining the Varnish *daemon* options starting with the text **DAEMON_OPTS** , similar to:

```
DAEMON_OPTS="-a :6081 \
             -T localhost:6082 \
             -f /etc/varnish/default.vcl \
             -S /etc/varnish/secret \
             -s malloc,256m"
```

Let's modify it to change the port from 6081 to 80:

```
DAEMON_OPTS="-a :80 \
             -T localhost:6082 \
             -f /etc/varnish/default.vcl \
             -S /etc/varnish/secret \
             -s malloc,256m"
```

VCL Language and File

Varnish uses a`.vcl` file (default located at`/etc/varnish/asdefault.vcl`) containing instructions written in VCL Language in order to run its program. This is used to define how Varnish should handle the requests and how the document *caching* system should work. When a new set of instructions (through a`. vcl` file) gets loaded, a (manager) process spun by Varnish converts them to **C code** and compiles it-- which then gets used to do the work.

Modifying the Default VCL File under `/etc/varnish/`

VCL Language and files can appear complicated at first, due to the wide array of tasks that can be defined for Varnish to perform. Luckily, we are supplied quite a bit of great examples, accompanied with schemas (for some) on https://www.varnish-cache.org/trac/wiki/VCLExamples. If you have any questions or ideas about what you want Varnish to do, reading some of these examples will surely allow you to easily have your way around it just the way you need.

Having said that, for a more-or-less default configuration combination for Drupal (and for most websites), we will go with the settings which can be found found below. Copy and paste all the code in each code block to form your`.vcl`file.

Let's open our editor once again to modify the contents of `default.vcl` (located under `/etc/varnish/`) for *Drupal*.

In order to open the editor, run the following command:

```
nano /etc/varnish/default.vcl
```

You will see a long document containing the default settings.

To begin, we need to define our web server and tell Varnish how to contact it. So let's modify the`backend default`section:

```
backend default {
    .host = "127.0.0.1";
    .port = "8000";
    .max_connections = 250;
    .connect_timeout = 300s;
    .first_byte_timeout = 300s;
    .between_bytes_timeout = 300s;
}
```

Note: Underneath, you will see a large block of commented out default VCL configuration code. You are free to keep it as long as you do not uncomment them by mistake, in which case compiling it would generate errors.

We will now set allowed purge client addresses (please refer to questions below to learn more on *purging*):

```
acl purge {
    "localhost";
    "127.0.0.1";
}
```

In order to define the allowed addresses which can access the cron.php or install.php, append the following:

```
acl internal {
  "192.10.0.0"/24;
  # For remote access, add your IP address here.
  # Ex: 162.xxx.xx.xx
}
```

Here we are going to write the program to handle the incoming (received) requests from clients. Continue with appending the following code block:

```
sub vcl_recv {
    # A great functionality of Varnish is to check
    # your web server's health and serve stale pages
    # if necessary.
    # In case of web server lag, let's return the
    # request with stale content.
    if (req.backend.healthy)
    {
        set req.grace = 60s;
```

```
}
else
{
    set req.grace = 30m;
}
# Modify (remove) progress.js request parameters.
if (req.url ~ "^/misc/progress\.js\?[0-9]+$")
{
    set req.url = "/misc/progress.js";
}
# Modify HTTP X-Forwarded-For header.
# This will replace Varnish's IP with actual client's.
remove req.http.X-Forwarded-For;
set    req.http.X-Forwarded-For = client.ip;
# Check if request is allowed to invoke cache purge.
if (req.request == "PURGE")
{
    if (!client.ip ~ purge)
    {
        # Return Error 405 if not allowed.
        error 405 "Forbidden - Not allowed.";
    }
    return (lookup);
}
# Verify HTTP request methods.
if (req.request != "GET"    && req.request != "HEAD" &&
    req.request != "PUT"    && req.request != "POST" &&
    req.request != "TRACE"  && req.request != "OPTIONS" &&
    req.request != "DELETE" && req.request != "PURGE")
{
        return (pipe);
}
# Handling of different encoding types.
if (req.http.Accept-Encoding)
{
    if (req.url ~ "\.(jpg|png|gif|gz|tgz|bz2|tbz|mp3|ogg)$")
    {
        remove req.http.Accept-Encoding;
    }
    elsif (req.http.Accept-Encoding ~ "gzip")
    {
        set req.http.Accept-Encoding = "gzip";
    }
    elsif (req.http.Accept-Encoding ~ "deflate")
    {
        set req.http.Accept-Encoding = "deflate";
    }
    else
    {
        remove req.http.Accept-Encoding;
    }
}
# Force look-up if request is a no-cache request.
```

```
    if (req.http.Cache-Control ~ "no-cache")
    {
        return (pass);
    }
    # Do not allow outside access to cron.php or install.php. Depending on your
access to the server, you might want to comment-out this block of code for
development.
    if (req.url ~ "^/(cron|install)\.php$" && !client.ip ~ internal)
    {
        # Throw error directly:
        error 404 "Page not found.";
        # Or;
        # Use a custom error page on path /error-404.
        # set req.url = "/error-404";
    }
    # Remove certain cookies.
    set req.http.Cookie = regsuball(req.http.Cookie, "has_js=[^;]+(; )?", "");
    set req.http.Cookie = regsuball(req.http.Cookie,
"Drupal.toolbar.collapsed=[^;]+(; )?", "");
    set req.http.Cookie = regsuball(req.http.Cookie, "__utm.=[^;]+(; )?", "");
    if (req.http.cookie ~ "^ *$")
    {
        unset req.http.cookie;
    }
    # Cache static content of themes.
    if (req.url ~ "^/themes/" && req.url ~ ".(css|js|png|gif|jp(e)?g)")
    {
        unset req.http.cookie;
    }
    # Do not cache these URL paths.
    if (req.url ~ "^/status\.php$" ||
        req.url ~ "^/update\.php$" ||
        req.url ~ "^/ooyala/ping$" ||
        req.url ~ "^/admin"        ||
        req.url ~ "^/admin/.*$"    ||
        req.url ~ "^/user"         ||
        req.url ~ "^/user/.*$"     ||
        req.url ~ "^/users/.*$"    ||
        req.url ~ "^/info/.*$"     ||
        req.url ~ "^/flag/.*$"     ||
        req.url ~ "^.*/ajax/.*$"   ||
        req.url ~ "^.*/ahah/.*$")
    {
        return (pass);
    }
    # Cache the following file types.
    if (req.url ~
"(?i)\.(png|gif|jpeg|jpg|ico|swf|css|js|html|htm)(\?[a-z0-9]+)?$")
    {
        unset req.http.Cookie;
    }
    # !! Do not cache application area
    if (req.url ~
```

```
"(^/app.php|^/app_dev.php|^)/([a-z]{2})/(payment|order|booking|media|autocomplete|moni
    {
        return (pass);
    }
    # !! Do not cache admin area
    if (req.url ~ "(^/app.php|^/app_dev.php|^)/admin" || req.url ~
"(^/app.php|^/app_dev.php|^)/(([a-z]{2})/admin)")
    {
        return (pass);
    }
    # !! Do not cache security area
    if (req.url ~
"(^/app.php|^/app_dev.php|^)/(([a-z]{2}/|)(login|logout|login_check).*)")
    {
        return (pass);
    }
    # Do not cache editor logged-in user sessions
    if (req.http.Cookie ~ "(sonata_page_is_editor)")
    {
        return (pass);
    }
    return (lookup);
}
sub vcl_hit {
    if (req.request == "PURGE")
    {
        purge;
        error 200 "Purged.";
    }
}
sub vcl_miss {
    if (req.request == "PURGE")
    {
        purge;
        error 200 "Purged.";
    }
}
```

Running Varnish

Now we are ready to restart Varnish daemon.

Run the following command to get Varnish running:

```
/etc/init.d/varnish restart
```

Finishing

Verifying Applications' State

Let's run the two following commands to verify that both Apache and Varnish are bound to the correct ports:

Apache:

```
netstat -lp | grep apache2
```

Varnish:

```
netstat -lp | grep varnish
```

The outcome should be similar to the following:

```
tcp    0    0    localhost:8000    *:*     LISTEN   xxxx/apache2
-- or --
tcp6   0    0    [::]:8000         [::]:*  LISTEN   xxxx/apache2
-- and --
tcp    0    0    localhost:6082    *:*     LISTEN   xxxx/varnishd
tcp    0    0    *:http            *:*     LISTEN   xxxx/varnishd
tcp6   0    0    [::]:http         [::]:*  LISTEN   xxxx/varnishd
```

Finally

We have set up Apache and Varnish to work with our Drupal installation. If you would like to get more from your Drupal backend regarding Varnish, you might want to try the **Drupal Varnish Module** located at https://drupal.org/project/varnish.

As of October 2013, what this module does is stated as:

- This module provides admin-socket integration which allows Drupal to dynamically invalidate cache entries, and also lets you query the Varnish admin interface for status, etc.

Note: If you are using a Drupal system older than version 7, you will also need to use this module due to Drupal's way of dealing cookies to visitors.

Troubleshooting, Notes, and Tips

How can I find out my Linux Distribution and its version?

Since this article talks about multiple distributions and versions, you might want to make sure you are aiming for the right one. Luckily enough, the solution is rather simple. Just run the following command:

```
cat /etc/*-release
```

This should give you your distribution's name and version.

How can I decide which port to use? Where can I learn more about ports?

You can pretty much opt for any port you want but certain ones are better be avoided to prevent future issues or collisions with other applications. Some of these ports are:

- **Simple Mail Transfer Protocol (SMTP)** on *port* 25
- **Network News Transfer Protocol (NNTP) Sync** on *port* 119
- **HTTP over TLS/SSL (HTTPS)** on *port* 443
- **MySQL Database System** on *port* 3306

For a full list please refer to:
http://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers

What does purging mean in terms of Varnish's functionality?

Purging is invalidation of cached files. For more information on the subject, please refer to:
https://www.varnish-cache.org/docs/3.0/tutorial/purging.html

What is a *daemon* ?

In computing, a **daemon** is a background process that runs on its own instead of direct interaction with a user. In Varnish's case, its daemon, upon compiling the VCL configuration file, runs continuously, executing the pre-set and compiled instructions just like your web server.