**Nginx and Varnish Configuration**

**NGINX CONFIGURATION**

All Nginx configuration files are located in the /etc/nginx/ directory. The primary configuration file is /etc/nginx/nginx.conf.

**Worker\_Process**

Defines how many threads, or simultaneous instances, of Nginx to run. A worker process is a single-threaded process.

If Nginx is doing CPU-intensive work such as SSL or gzipping and you have 2 or more CPUs/cores, then you may set worker\_processes to be equal to the number of CPUs or cores.

If you are serving a lot of static files and the total size of the files is bigger than the available memory, then you may increase worker\_processes to fully utilize disk bandwidth.

Your OS may schedule all workers on single CPU/core this can be avoided using worker\_cpu\_affinity



**Gzip**

The gzip directive tells the server to use on-the-fly gzip compression to limit the amount of bandwidth used and speed up some transfers.:

**/etc/nginx/nginx.conf**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | Gzip on; |

Defining the Directives

We’ll begin by explaining the core directives in /etc/nginx/nginx.conf Let’s go back to that first section:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | **User** nginx;  **worker\_processes** 1;  **pid** var/run/nginx.pid;  **events** {  **worker\_connections** 1024;    } |

**user**

Defines which Linux system user will own and run the Nginx server. Most Debian-based distributions use **nginx** but this may be different in other distros. There are certain use cases that benefit from changing the user; for instance if you run two simultaneous web servers, or need another program’s user to have control over Nginx.

**worker\_process**

Defines how many threads, or simultaneous instances, of Nginx to run. You can learn more about this directive and the values of adjusting it .

**pid**

Defines where Nginx will write its master process ID, or PID. The PID is used by the operating system to keep track of and send signals to the Nginx process.

HTTP (Universal Configuration)

The next section of the nginx.conf file covers the universal directives for Nginx as it handles HTTP web traffic. The first part of the HTTP block is shown below:

**/etc/nginx/nginx.conf**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | **http** {    **sendfile** on;  #**tcp\_nopush** on;    **keepalive\_timeout** 65;    *# server\_tokens off;*    *# server\_name\_in\_redirect off;*  **include** /etc/nginx/mime.types;  **default\_type** **application/octet-stream**;  *##*  *# Logging Settings*  *##*  **access\_log** /var/log/nginx/access.log main;  **error\_log** /var/log/nginx/error.log warn;  *##*  *# Gzip Settings*  *##*  **gzip** on; |

Most of the http { } block should work as-is for most Nginx configurations. We do, however, want to draw your attention to the following configuration options:

**include**

The include statement at the beginning of this section *includes* the file mime.types located at/opt/nginx/conf/mime.types. What this means is that anything written in the file mime.types is interpreted as if it was written inside the http { } block. This lets you include a lengthy amount of information in the http { } block without having it clutter up the main configuration file. Try to avoid too many chained inclusions (i.e., including a file that itself includes a file, etc.) Keep it to one or two levels of inclusion if possible, for readability purposes. You can always include all files in a certain directory with the directive:

|  |  |
| --- | --- |
|  |  |

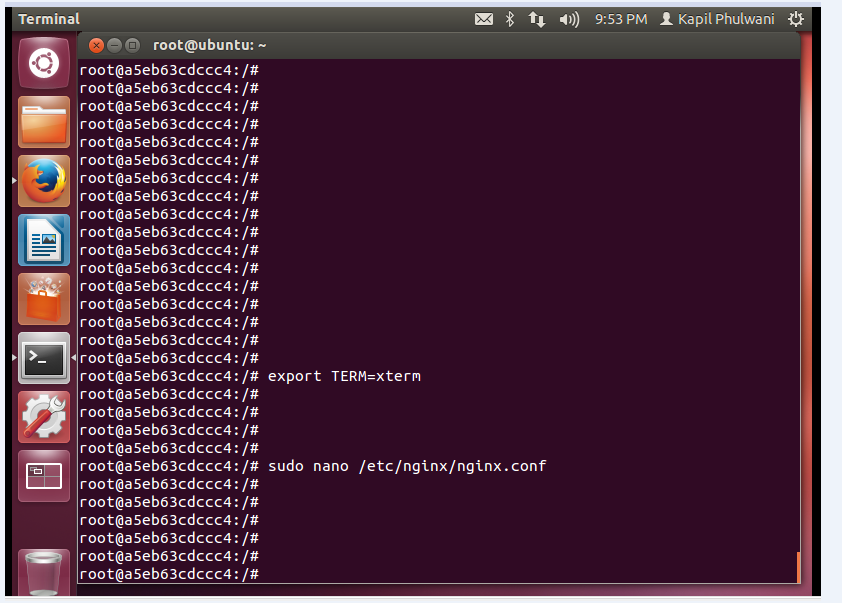
Or to be more specific, you can include all .conf files in a directory:

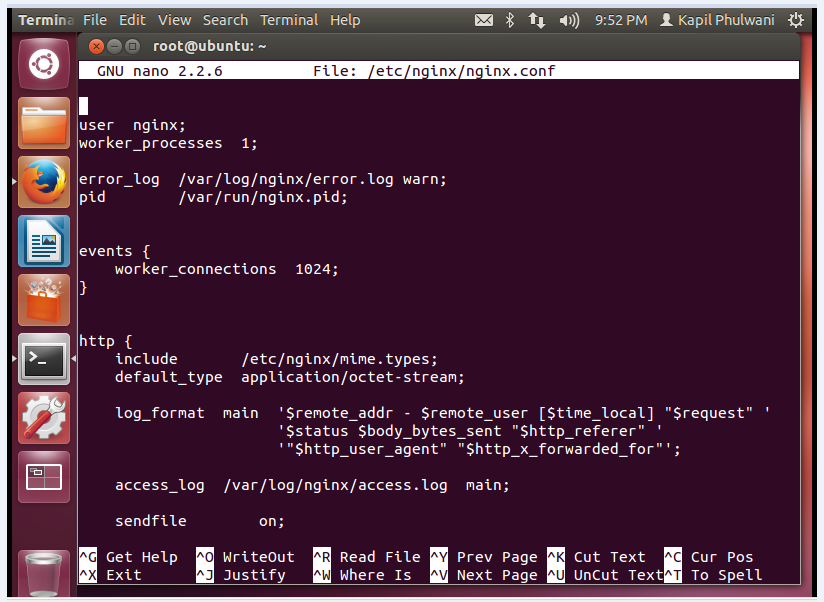
|  |  |
| --- | --- |
| 1 | include /etc/nginx/conf.d/\*.conf; |

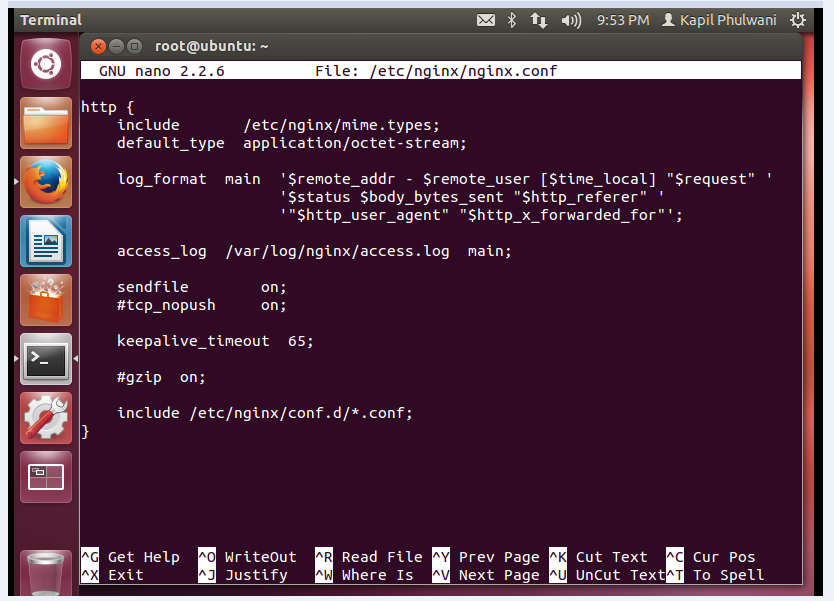
**gzip**

The gzip directive tells the server to use on-the-fly gzip compression to limit the amount of bandwidth used and speed up some transfers. This is equivalent to Apache’s mod\_deflate.

**Snapshots of nginx configuration**







**Varnish Configuration**

Configuration

Varnish has two categories of configuration:

* Command line configuration and tunable parameters
* VCL

Command line configuration

|  |  |
| --- | --- |
| -a *<[hostname]:port>* | |
|  | listen address |
| -f *<filename>* | VCL |
| -p *<parameter=value>* | |
|  | set tunable parameters |
| -S *<secretfile>* | |
|  | authentication secret for management |
| -T *<hostname:port>* | |
|  | Management interface |
| -s *<storagetype,options>* | |
|  | where and how to store objects |

All the options that you can pass to the varnishd binary are documented in the varnishd(1) manual page (man varnishd). You may want to take a moment to skim over the options mentioned above.

The only option that is strictly needed to start Varnish is the -f to specify a VCL file.

Though they are not strictly required, you almost always want to specify a -s to select a storage backend, -a to make sure Varnish listens for clients on the port you expect and -T to enable a management interface, often referred to as a telnet interface.

Both for -T and -a, you do not need to specify an IP, but can use :80 to tell Varnish to listen to port 80 on all IPs available. Make sure you don’t forget the colon, as -a 80 will tell Varnish to listen to the IP with the decimal-representation “80”, which is almost certainly not what you want. This is a result of the underlying function that accept this kind of syntax.

You can specify -p for parameters multiple times. The workflow for tuning Varnish parameters usually means that you first try the parameter on a running Varnish through the management interface to find the value you want, then store it in a configuration file that will pass it to Varnish with -p next time you start it up. We will look at these files later on.

The -S option specifies a file which contains a secret to be used for authentication. This can be used to authenticate with varnishadm -S as long as varnishadm can read the same secret file - or rather the same content: The content of the file can be copied to another machine to allow varnishadm to access the management interface remotely.

Configuration files

Most Varnish-installations use two configuration-files. One of them is used by the operating system to start Varnish, while the other contains your VCL.

| **File** | **Usage** |
| --- | --- |
| /etc/default/varnish | Used for parameters and command line arguments. When you change this, you need to run service varnish restart for the changes to take effect. On RedHat-based OS’s, this is kept in /etc/sysconfig/varnish instead. |
| /etc/varnish/default.vcl | The VCL file. You can change the file name by editing */etc/default/varnish* if you want to, but it’s normal to use the default name. This contains your VCL and backend-definitions. After changing this, you can run either service varnish reload, which will not restart Varnish, or you can run service varnish restart, which empties the cache. |

