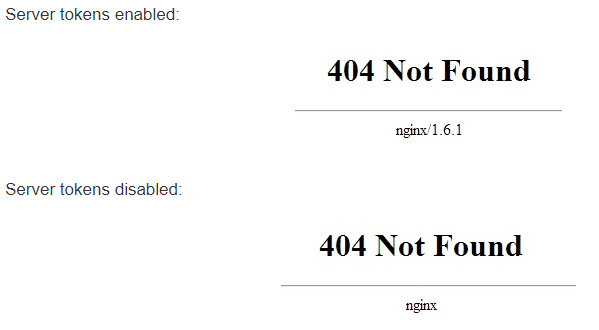
*Assuming NGINX is already installed…*

**NGINX SECURITY: Disable Server Tokens**

NGINX’s version number is visible by default with any connection made to the server, whether by a successful 201 connection by CURL, or a 404 returned to a browser. Disabling server tokens makes it more difficult to determine NGINX’s version, and therefore more difficult for an attacker to execute version-specific attacks.

****

Add the following line to the http block of /etc/nginx/nginx.conf:

**--- server\_tokens off;**

>>> To avoid a possible hash bucket memory problem that can arise from adding additional server names, it is necessary to adjust a single value in the /etc/nginx/nginx.conf file. Open the file:

**--- sudo nano /etc/nginx/nginx.conf**

Find the server\_names\_hash\_bucket\_size directive and remove the # symbol to uncomment the line:

**server\_names\_hash\_bucket\_size 64;**

Restart nginx web server.

**--- sudo systemctl restart nginx**

**NGINX SECURITY: Disable Content Sniffing**

Content sniffing allows browsers to inspect a byte stream in order to determine the file format of its contents. It is generally used to help sites that do not correctly identify the MIME type of their content, but it also presents a vector for cross-site scripting and other attacks. To disable content sniffing, add the following line to nginx configuration’s http block:

**--- sudo nano /etc/nginx/nginx.conf**

**add\_header X-Content-Type-Options nosniff;**

**NGINX SECURITY: Filter Cross-Site Scripting (XSS)**

This header signals to a connecting browser to enable its cross-site scripting filter for the request responses. XSS filtering is usually enabled by default in modern browsers, but there are occasions where it’s disabled by the user. Forcing XSS filtering for your website is a security precaution, especially when your site offers dynamic content like login sessions: add the following line to nginx configuration’s http block:

**--- sudo nano /etc/nginx/nginx.conf**

**add\_header X-XSS-Protection "1; mode=block";**

**NGINX MANAGEMENT: Setting Up Server Blocks (Essential for SSL Certificate Testing)**

***Server Blocks*** *e.g* ***hrmskra.briqsystems.com*** *that are to be created here must match the* ***Domain Name*** *that will be bought form the* ***Webhost or DNS Provider e.g GoDaddy*** *…*

In **NGINX**, a Server Block basically equates to a website (same as Virtual Host in Apache terminology). NGINX can host multiple websites, and each site’s configuration should be in its own file, with the name formatted as example.com.conf. That file should be located at /etc/nginx/conf.d/.

Create the directory for example.com as follows, using the -p flag to create any necessary parent directories:

**--- sudo mkdir -p /var/www/hrmskra.briqsystems.com/html**

Assign ownership of the directory with the $nginx environment variable:

**--- sudo chown -R $nginx:$nginx /var/www/hrmskra.briqsystems.com/html**

The permissions of your web roots should be correct if you haven’t modified your umask value, but you can make sure by typing:

**--- sudo chmod -R 755 /var/www/hrmskra.briqsystems.com**

Open your text editor and create an index.html file:

**--- sudo nano /var/www/hrmskra.briqsystems.com/index.html**

! Inside, add the following sample HTML

<html>

<head>

<title>Welcome to hrmskra.briqsystems.com!</title>

</head>

<body>

<h1>Success! The hrmskra.briqsystems.com server block is working!</h1>

</body>

</html>

Or you can simply type in some plain text to the html file then

**>>> SAVE & EXIT**

**NGINX MANAGEMENT: Configure Server Blocks**

! multiple site configuration files should be stored in /etc/nginx/conf.d/ as that's a standard convention, and should work anywhere.

Files from conf.d gets to be processed sooner, and, as such, if you have configurations that silently conflict with each other, then those from conf.d may take precedence over those in /sites-enabled and /sites-available. Sites-enabled is not used by the upstream packaging of nginx from <http://nginx.org/packages/>. Next, create the actual server block configuration:

**--- sudo nano /etc/nginx/conf.d/hrmskra.briqsystems.conf**

server {

listen 80;

listen [::]:80;

server\_name hrmskra.briqsystems.com;

root /var/www/hrmskra.briqsystems.com/html;

index index.html;

location / {

try\_files $uri $uri/ =404;

}

} **>>> SAVE & EXIT**

**NGINX MANAGEMENT: Restart Nginx Service**

**--- systemctl restart nginx**

! We can not test the domain yet because we need an actual domain name that is registered from a Domain Registrar on the internet.

**NGINX MANAGEMENT: Nginx Reverse Proxy to Upstream Servers**

**Reverse Proxy Server** is a type of proxy server that typically sits behind the firewall in a private network and directs client requests to the appropriate backend server. A reverse proxy provides an additional level of abstraction and control to ensure the smooth flow of network traffic between clients and servers.

**Proxy Server** is a go‑between or intermediary server that forwards requests for content from multiple clients to different servers across the Internet.

**Upstream Server** refers to a server that provides service to another server. In other words, upstream server is a server that is located higher in a hierarchy of servers. The highest server in the hierarchy is sometimes called the origin server.

! **Open the server block configuration that has been created in Step 3.1**

**--- sudo nano /etc/nginx/conf.d/hrmskra.briqsystems.conf**

server {

listen 80;

listen [::]:80;

server\_name hrmskra.briqsystems.com;

root /var/www/hrmskra.briqsystems.com/html;

index index.html;

location / {

try\_files $uri $uri/ =404;

}

}

Replace the try\_files directive with the following lines:

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_pass [http://localhost:8000](http://localhost:8000/);

>>> Where localhost is the ip address of your Virtual Server Instance e.g Ubuntu, CentOS with its corresponding port.

**>>> SAVE & EXIT**

X-Real-IP $remote\_addr - Forwards the real visitor remote IP address to the proxied server.

Host $host - The $host variable in the following order of precedence contains: hostname from the request line, or hostname from the Host request header field, or the server name matching a request.

**NGINX MANAGEMENT: Nginx Caching**

**Content Caching** is one of the most effective ways to dramatically improve the performance of a website.

A **content cache** sits in between a client and an “origin server”, and saves copies of all the content it sees. If a client requests content that the cache has stored, it returns the content directly without contacting the origin server. This improves performance as the content cache is closer to the client, and more efficiently uses the application servers because they don’t have to do the work of generating pages from scratch each time.

There are potentially multiple caches between the web browser and the application server: the client’s browser cache, intermediary caches, content delivery networks (CDNs), and the load balancer or reverse proxy sitting in front of the application servers. Caching, even at the reverse proxy/load balancer level, can greatly improve performance.

**NGINX MANAGEMENT: Create a folder to store cached content**

**--- mkdir /var/www/hrmskra.briqsystems.com/cache/**

Add the proxy\_cache\_path directive to NGINX’s default http block.

**--- sudo nano /etc/nginx/nginx.conf**

Add this line:

**proxy\_cache\_path** /var/www/hrmskra.briqsystems.com/cache/ keys\_zone=hrmskraCache:10m max\_size=500m inactive=24h use\_temp\_path=off;

**>>> SAVE & EXIT**

* keys\_zone=hrmskraCache:10m sets a 10 megabyte shared storage zone (simply called one, but you can change this for your needs) for cache keys and metadata.
* max\_size=500m sets the actual cache size at 500 MB.
* inactive=24h removes anything from the cache which has not been accessed in the last 24 hours.
* use\_temp\_path=off writes cached files directly to the cache path. This setting is [recommended by NGNIX](https://www.nginx.com/blog/nginx-caching-guide/).

Then add the proxy\_cache directive your server block configuration file.

**---sudo nano /etc/nginx/conf.d/hrmskra.briqsystems.conf**

Add this line above the location directive that defines the proxy\_pass reverse proxy code block:

**proxy\_cache** hrmskraCache;

! the parameter “briqCache” in this directive should match to the “keys\_zone=briqCache” directive in your NGINX’s default http block. And you can name it the same as your domain/subdomain name to eliminate confusion.

**>>> SAVE & EXIT**

If you need to **clear** the cache, the easiest way is with the command:

**find /var/www/hrmskra.briqsystems.com/cache/ -type f -delete**

**DNS MANAGEMENT: Pointing domain name (hrmskra.briqsystems.com) bought from GoDaddy to your Web Host or VPS (e.g Vultr Server Instances).**

A **domain name** is the address where Internet users can access your website. The domain name must be registered before you can use it.

**Domain Name Servers** (DNS) are the Internet's equivalent of a **phone book**. They maintain a directory of domain names and translate them to Internet Protocol (IP) addresses. This is necessary because, although domain names are easy for people to remember, computers or machines, access websites based on **IP addresses.**

**Web Host** is where your website files and databases are stored same for **VPS**.

**DNS MANAGEMENT: Creating account on Godaddy.com and buy a domain name of your choice.**

**Buy a Domain:** hrmskra.briqsystems.com

Next, find out nameservers of your web host or VPS. in this case the nameserver of **Vultr** where your instances of servers are stored.

The name servers usually look like **ns1.example.com** and **ns2.example.com**.

*You can get the name servers from DNS setting or the documentations on your web host or ask it directly from the support team if you have no idea where to get it.*

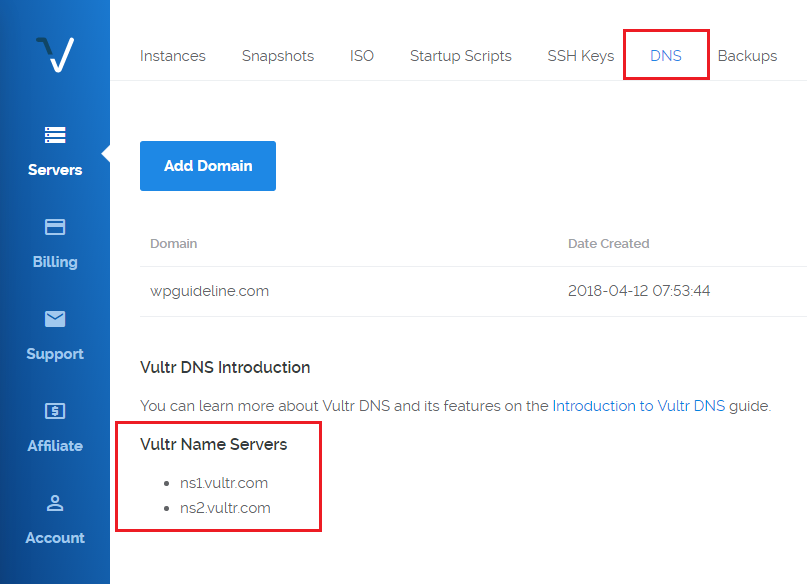
Add **nameservers** of web host or VPS into your domain registrar (GoDaddy).

**Nameserver** is a server on the internet specialized in handling queries regarding the location of a domain name's various services. Nameservers are a fundamental part of the Domain Name System (DNS). They allow using domains instead of IP addresses.

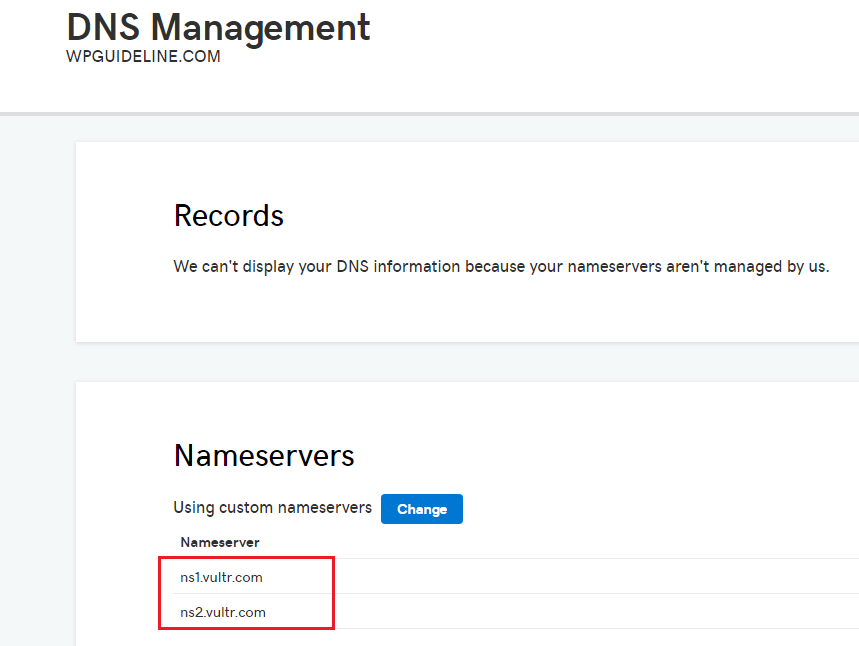
**DNS MANAGEMENT: Get nameservers from Vultr**

Log in Vultr and click DNS option on Vultr main **dashboard**. Click **Add Domain** button to enter your domain name and IP address.

Then you will find Vultr name servers listed at the bottom of this page.



**DNS MANAGEMENT: Add those nameservers into GoDaddy**

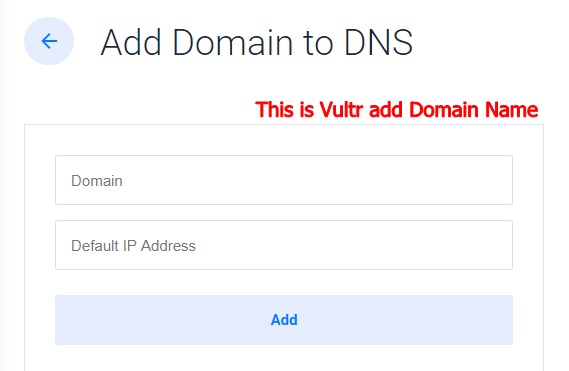
Once you get the name servers, sign in GoDaddy > **My Product** > **Domain name** > **DNS management** > **Nameservers**. Then, click Change button to customize your Nameservers and save your change.

Finally, it’s done, and it will take few hours or days to update the changes on the DNS cache of your DNS Server.

Once updated, you can visit your website with your domain name.

*NOTE: By doing this, We are* ***redirecting*** *the browser request made with your domain name (****hrmskra.briqsystems.com****) to the* ***IP Address*** *that has been created in your* ***VPS (Vultr)****.*

*Each IP Address represents each instance of a Virtual Server OS e.g Ubuntu, CentOS.*



Add the Domain Name (briqsystems.conf) you bought from GoDaddy here with the IP Address of the Virtual Server Instance (Ubuntu, CentOS) where you configured your Server Block (briqsystems.conf) using NGINX.

**Yes, You are going to make (2) domain names, one bought from GoDaddy & one from Vultr.**

**SLL MANAGEMENT: Setup Let's Encrypt With Nginx**

*Perform this SSL Configurations inside your Virtual Server Instance & apply SSL to the Server Block (hrmskra.briqsystems.com) that matches the Domain Name you bought from GoDaddy which is also (hrmskra.briqsystems.com).*

Let's Encrypt is a **Certificate Authority** (CA) that provides free **SSL certificates** with an automated client. By using a Let's Encrypt SSL certificate, you can **encrypt traffic** between your website and your visitors. The entire process is simple, and renewals can be automated. Also, note that the installation or renewal of certificates does not cause any downtime.

**SLL MANAGEMENT: Install Certbot**

To obtain a Let's Encrypt SSL certificate, you have to install the **Certbot** client on your server.

Add the repository. Press the **ENTER** key when prompted to accept.

**--- add-apt-repository ppa:certbot/certbot**

Update the package list.

**--- apt-get update**

Proceed by installing Certbot and Certbot's Nginx package.

**--- apt-get -y install python-certbot-nginx**

**SLL MANAGEMENT: Obtaining a Let's Encrypt SSL certificate**

**SSL** (Secure Sockets Layer) is the standard security technology for establishing an encrypted link between a web server and a browser. This link ensures that all data passed between the web server and browsers remain private and integral.

**--- certbot --nginx -d hrmskra.briqsystems.com**

If you are running Certbot for the first time, you will be prompted to enter an **email address** and agree to the **terms of service**. This e-mail address will be used for **renewal** and security notices. Once you have provided an email address, Certbot will request a certificate from Let's Encrypt and run a challenge to verify that you control the domain in question.

If Certbot can obtain an **SSL certificate**, it will ask how you would like to configure your **HTTPS** settings. You can either redirect visitors who visit your website over an unsecured connection or let them access it over the unsecured connection. This should usually be enabled because it ensures that visitors only access the SSL-protected version of your website. Select your choice, then hit **ENTER**.

**Certbot Prompt:**

Please choose whether or not to redirect HTTP traffic to HTTPS, removing HTTP access.

-------------------------------------------------------------------------------

1: No redirect - Make no further changes to the webserver configuration.

2: Redirect - Make all requests redirect to secure HTTPS access. Choose this for

new sites, or if you're confident your site works on HTTPS. You can undo this

change by editing your web server's configuration.

-------------------------------------------------------------------------------

Select the appropriate number [1-2] then [enter] (press 'c' to cancel):

Finally, Certbot will confirm that the process was **successful** and where your certificates are stored. Your certificates are now downloaded and installed.

! **Automating renewal** Because Let's Encrypt is a free certificate authority, and to encourage users to automate the renewal process, certificates are only valid for **90 days**. Certbot will take care of renewing certificates automatically. It does so by running **certbot renew** twice per day via **systemd**.

You can check that automatic renewal is working by running this command.

**--- certbot renew --dry-run**

You can also manually renew your certificate at any time by running the following command.

**--- certbot renew**

**------- END -------**