Quick guide to setting up a website with a VPS

2022

I make absolutely no claims about the security of this procedure. Follow this at your own risk.

1. **Buy a domain**, which we'll assume it's something like **mydomain.com**. You can buy at sites like Epik, which is a DNS (*domain-name system*) registrar.

2. **Set up server**. In this guide I will assume you're renting a VPS, a *virtual private server*. You can do this at places like Vultr. Basically you rent a virtual machine on a server which is awake 24/7. You will be given the option of which operating system for you VPS, which I assume will be **Debian 10**. You can probably do this with other OS, but I refuse to do it in anything which is not Linux based. Should you be able to choose, select to have also IPv6 compatibility.

3. So, once you have rented a VPS you probably have:

   (a) a public IP (IPv4) for your server.

   (b) an IPv6 for your server.

   (c) a password for the **root** user of your VPS.

4. You can already **point your domain to your VPS**. To do this, go to your domain registrar (in my case it will be Epik) and make sure to add the following DNS Host records for your chosen domain:

   - type A record (IPv4) for your IPv4. Just fill in the blanks with your domain (mydomain.com) and your IPv4.
   - type AAA record (IPv6). Proceed as above

   You can also redirect subdomains. For example, should anyone enter **www.mydomain.com** on their browser, you probably want to redirect them to your webpage.

5. Now you want to **login into your server**, from your home/personal computer. If you use Linux you can do, from the terminal,

   ```
   $john@home:  ssh root@server.IPv4
   ```

   and then you'll prompted for the password. Once entered, you will be running a terminal *in your VPS*, but from the comfort of your own house. It's good practice to change the password for your root account, and make sure you **do not lose it**. Just do so with the `passwd` command:

   ```
   $root@server:  passwd
   ```

   Note that if you have already performed Step 4, you could also have ssh'd with your domain instead of the IPv4 address, like

   ```
   $john@home:  ssh root@mydomain.com
   ```

   Whenever you want to logout of your server and come back to your computer, simply type and execute `logout` or `exit`.

6. **Cryptographic keys**: it's generally a bad idea to access your server with a password, for security reasons. To remedy this, we can use cryptographic keys to access our server and disable logging in with passwords. It's simple enough: in your local computer (remember to `exit` from the session if you are in the server) generate a key pair with command

   ```
   $john@home:  ssh-keygen
   ```

   and store both keys, which I will assume will be in this path ~/.ssh/serverkey and ~/.ssh/serverkey.pub . Now you can login into your server carrying your public key by

   ```
   $john@home:  ssh-copy-id -i ~/.ssh/serverkey.pub root@mydomain.com
   ```

You will be prompted for your password to confirm you ARE the root user of your VPS. Once that is done, you can `exit` your server and try logging in with your private key instead, by using

```
$john@home:  ssh root@mydomain.com
```

or

```
$john@home:  ssh -i /PATH/TO/serverkey root@mydomain.com
```

if you have stored your keys in any other directory. You should not be prompted for the root password.

7. **Disable password login**: once you do this you want to disable logging in with passwords. That means, of course, you should be careful not to lose your private key if you want to keep being able to access your server. Once inside your server you need to edit the configuration file for **sshd**. You can use whatever command-line text editor you want, **nano** or **vim**. I will use vim:

   ```
   $root@server:  vim /etc/ssh/sshd_config
   ```

   and make sure you find and replace the commented or default lines to the following:

   - `PasswordAuthentication no`
   - `UsePAM no`
   - `ChallengeResponseAuthentication no`

   Once that is done, reload and restart your **sshd** daemon to update to the latest configuration:

   ```
   $root@server:  systemctl reload sshd
   ```

   ```
   $root@server:  systemctl restart sshd
   ```

   It should be impossible to access your server without your private key, now, which is a very good thing!

8. **Install nginx**, the software needed for hosting websites. Another popular option is apache, but I will stick to nginx in this guide. First, update and upgrade the software in your server:

   ```
   $root@server:  apt-get update
   ```

   ```
   $root@server:  apt-get upgrade
   ```

   Then, install the following 3 packages

   ```
   $root@server:  apt-get install nginx certbot python3-certbot-nginx
   ```

   The packages **certbot** and **python3-certbot-nginx** will allow us for enabling HTTPS in our website, for secure connections.

9. **Start setting up the website**: first thing you might want to do tell nginx that you have a site hosted in your server. You should do so creating a file under /etc/nginx/sites-available, which I will name johnswebsite. I will copy the default configuration file to be found at /etc/nginx/sites-available/default:

   ```
   $root@server:  cp /etc/nginx/sites-available/default /etc/nginx/sites-available/johswebsite
   ```

   and you can link it to **sites-enabled** with a symbolic link:

   ```
   $root@server:  ln -s /etc/nginx/sites-available/johnswebsite /etc/nginx/sites-enabled
   ```

   You can go ahead and edit the file at .../sites-available/johnswebsite and edit the necessary info to point to your (yet to be created) HTML folder where you will be putting you website's files. Use you text editor of choice. I will use vim:

   ```
   $root@server:  vim /etc/nginx/sites-available/johnsweb
   ```

   You simply need to add the domains in the right place, separated by a blank space: `mydomain.com` and probably `www.mydomain.com`. Also add the directory for your website (created in the next step). In this example I will use /var/www/johnswebsite. Just make sure to specify this in the file in a line "root /var/www/johnswebsite". Your configuration file should read (comments aside) like this:

```
server {
        listen 80 ;
        listen [::]:80 ;
        server_name mydomain.com www.mydomain.com;
        root /var/www/johnswebsite ;
        index index.html index.htm index.nginx-debian.html ;
        location / {
                try_files $uri $uri/ =404 ;
        }
}
```

10. **Website directory**: let us create the actual directory where the files for the site will be stored.

    ```
    $root@server:  mkdir /var/www/johnswebsite
    ```

    You can already create the first page in your website! It should be named **index.html**. You can put something really simple (and I won't cover HTML edition in this guide). Something like this will do:

    ```
    <h1>John's website </h1>
    <p>Here's some content </p>
    ```

    Save this as /var/www/johnswebsite/index.html. This should be already visible from the internet! Just go and browse for **mydomain.com**!

11. **Troubleshooting**: many VPS providers block traffick incoming to ports 80 and 443, which will prevent anyone to make connection with your site. If you cannot see your website already, it may be because the firewall **ufw** is blocking these ports. Unblocking them is easy enough:

    ```
    $root@server:  ufw allow 80
    $root@server:  ufw allow 443
    ```

12. **Enabling https**: with **certbot** it's pretty simple to certify your website as HTTPS compliant. Just run this:

    ```
    $root@server:  certbot -nginx
    ```

    and you will be prompted for the domains that you want to certify. It's pretty straighforward: make sure to select **mydomain.com** and **www.mydomain.com**.

    This certificates are not permanent, they last about a year if I recall correctly. However you can automate the renewal for these by adding this line to your crontab (edit it by running `crontab -e`). Add the following line to this file:

    ```
    1 1 1 * * certbot renew
    ```