This tutorial was written with setting up a Linode server as the goal. You can use this information with any VPS option or on your own machine. If you’re working on a machine that you are not using SSH to access, you can ignore the SSH parts. But it is still good info have.

**Basic Security**

After getting the server turned on, connect to it using either:

ssh root@ipaddress

In a Linux or Mac terminal, or by downloading PuTTY on Windows and connecting via SSH with that.

Now that we are in, before going any further it would be wise to take a few basic steps to harden, or secure, the server. First order of business, updates:

yum update

Making sure that our server always has the most recent versions of software goes quite a long way in keeping us secure. Next, we’ll create our user, since only having root adds a whole slew of security issues; hackers would be able to log straight into root over SSH giving them access to anything and everything, lack of the sobriety check that having to use sudo gives, and having to setup software with root privileges rather than with a more limited user. To make our new user we can do:

adduser *username*passwd *username*  
adduser –aG wheel *username*

*username* can be whatever you desire. Using the adduser command creates a user and a home directory for that user on the server, we then use passwd to give that user a password, since they won’t have on by default. Using it the second time adds our new user to the wheel group, allowing us to run commands as root without having to be logged in as root, but avoids allowing the user to have complete access to everything. Since we now have the option to use a non-root user, let’s do so:

su *username*

su means *substitute user*. It is somewhat but not the same as logging out and then in as a new user. Processes start as root continue running and you don’t enter a new shell, but any commands that you run will be done as the new user rather than root.

With all this done, we should now move on to making our SSH connection more secure. Disabling root login and changing which port SSH connections are made on will add a layer of security and stop most automated bots that find websites with insecure SSH rules. We can make these changes in the sshd\_config file, which can be accessed by doing:

sudo nano /etc/ssh/sshd\_config

However, I suggest making a backup first by using the command:

sudo cp /etc/ssh/sshd\_config /etc/ssh/sshd\_config.bak

Just so that if changing a setting breaks everything, we can make it a little less broken by restoring the original configuration.

The parts that interest us are:  
Port 22  
PermitRootLogin yes  
#PasswordAuthentication yes  
#PermitEmptyPassword no

Set Port to be whatever you desire, just make sure that you will remember it (and don’t worry about suddenly getting kicked off the server by the change, it won’t come into effect until we restart the service). Set PermitRootLogin to no, so that root will only be accessible by doing su root rather than being able to log straight into it (people trying to get in having to crack two passwords is much preferred to having to crack one, no?). Finally, remove the # before PasswordAuthentication and PermitEmptyPassword, to make it so that in order to SSH into the server, the user being logged into must have a password. Feel free to play around with other settings in this file, like motd (message of the day, a line of text that will display whenever someone logs into the server over SSH). These changes will come into effect after doing:

sudo systemctl restart sshd.service

And disconnecting and then reconnecting to the server. Since the SSH port is changed, modify the command you use to connect to be:

ssh *username*@*ipaddress* -p*port*

As far as basic SSH security goes, we are set. There are a few more steps that could be taken, such as setting up SSH Keys to make it so that only someone with a private key that matches the server’s public key can log in over SSH, but I won’t get into that in this tutorial. Perhaps in a future one. Now we’ll get a firewall running.

Unlike Debian and Ubuntu where you have options such as UFW for a firewall, CentOS comes standard only with iptables. There are ways to get options such as UFW to work, but there are a slight pain and iptables is good to know. To start with, lets see if there are any rules in place:

sudo iptables -L

This should come up with output saying that all traffic is being accepted. If not, flush your firewall rules with the –F flag:

sudo iptables –F

Now that we have fresh firewall rules, we should make it so that all traffic going into localhost is allowed. Software like MariaDB and some parts of Wordpress require this.

sudo iptables –A INPUT –i lo –j ACCEPT

Now that internal traffic within our server is allowed, lets create rules for ingoing and outgoing traffic:

sudo iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT  
sudo iptables –A INPUT –p tcp –-dport *sshport* –j ACCEPT   
sudo iptables –A INPUT –p tcp –-dport 80 –j ACCEPT   
sudo iptables –A INPUT –p tcp –-dport 443 –j ACCEPT  
sudo iptables –P INPUT DROP  
sudo iptables –P FORWARD DROP  
sudo iptables –P OUTPUT ACCEPT  
sudo iptables –L –v

The first line sorts traffic by whether it is NEW, which means traffic not started by the server, ESTABLISHED, traffic started by the server, or RELATED, traffic created because of ESTABLISHED packets. The second through fourth rule allow traffic for SSH, HTTPS, and HTTPS to enter the server, the fifth blocks everything that is not on those three ports. The sixth rule blocks all forwarded traffic. The seventh rules allows all traffic to leave the server. And finally, the eighth command shows us all of this.

At the moment if we were to reboot, our iptable configuration would be reset. To avoid this we can run a script provided by iptables.

Sudo /sbin/iptables-save

With all this done, let’s get into making us a web server!

**Web Server!**

Step one in making a web server is installing the software needed to make one run. In our case, this will be Apache2 as our server, PHP 5 as our scripting language, and MariaDB as our database (this is known as a LAMP stack. One can substitute out Apache2 for Nginx and have a LEMP stack. I will make a post on how to make WordPress work with Nginx at a future date). To get all that we will need installed, run:

sudo yum install httpd mariadb-server mariadb php php-mysql wget  
sudo systemctl enable mariadb.service  
sudo systemctl enable httpd.service

The last two lines tell the system to start MariaDB and Apache when the computer boots up.

**Database**

With our toolset installed, let’s first configure MariaDB. The installation comes with a lovely script that you can run by calling:

sudo systemctl start mariadb.service  
sudo mysql\_secure\_installation

The first line starts the service for Maria, which lets us access the databases with our Maria install. The second runs the script that I mentioned, which will take of some basic hardening of our database. Say yes to everything it asks, and including setting a secure root password.

And then try the secure installation script again, this time entering the password you just set. What we just did is go into the main mysql database, change all parts that mention a root password to be something rather than blank, update privileges, and then exit. This makes the secure installation script happy because it now does not have to begin functioning with no password.

Now that we have MariaDB running, let’s make a database for our WordPress install. To start with, let’s get into the MariaDB shell:

mysql -u root -p

This tells MariaDB to open a shell with user root, and that the user does have a password. Sometimes you will need to run sudo with this command, sometimes you will not. Now that we have our MariaDB shell, let’s make a database. I am going to be going the setup for a website that will be called CooperEditing from this point forward. So any time that I mention CooperEditing, change it to match what you desire.

create database CooperEditing;

And now for the sake of security, we will make a user within MariaDB with the ability to edit this database, rather than have root do it.

create user *username*;  
set password for *username*@localhost= PASSWORD("*password*");  
grant all privileges on CooperEditing.\* TO *username*@localhost identified by '*password*';  
flush privileges;  
exit

The first line makes a new MariaDB user, the second sets their password, the third gives them an all access pass to our database, the fourth updates privileges, and the last exits the MariaDB shell. This should be all that we need for our database, so let’s move on to PHP!

**PHP**

Of everything, PHP needs the least configuration to make it work. Actually, it does not need any, but there is one bit that you might want to change in php.ini limits how big of a file can be uploaded to the server. By default it is 2MB. Uploading certain themes or large images and such will require this to be changed. To do so:

sudo nano /etc/php.ini

The line that reads upload\_max\_filesize (you can use ctrl+w to find this line, similar to ctrl+f in many browsers and graphical text editors) can be changed to adjust how large you want to limit files. I suggest max 5M. If you ever need bigger, adjust it higher, upload the file, and then change it back. Or setup an FTP server.

That’s all that is needed for PHP. So finally, Apache2!

**Apache2**

Apache on RHEL distributions is quite different than Debian based counterparts. For one, all configuration is in a folder named httpd in /etc. On a cool note, though, we can make it act a bit more like it’s Debian brother, and also make activating and deactivating websites hosted on our server pretty simple. To do this, add:

IncludeOptional sites-enabled/\*.conf

To the bottom of /etc/httpd/conf/httpd.conf. To make this work properly, make two directories in the httpd folder. One called /etc/httpd/sites-available and the other /etc/httpd/sites-enabled. With both folders made, we can now make the configuration file for our website.

sudo mkdir /etc/httpd/sites-available/CooperEditing.conf

And to fill in the blank config, you can change the info from here to match what is correct for your website:

<VirtualHost \*:80>

ServerAlias cooperediting.com

ServerName www.cooperediting.com  
 ServerAdmin bailey@gingertechnology.net

DocumentRoot /var/www/cooperediting.com/

ErrorLog /var/www/cooperediting.com/error.log

CustomLog /var/www/cooperediting.com/requests.log combined

</VirtualHost>

ServerAlias is the website’s domain name, ServerName is how one could set a subdomain, ServerAdmin is who one should report errors to, DocumentRoot is where the files for the website are, ErrorLog and CustomLog are log files for the website that can be used to troubleshoot issues. And now to activate this website, we simply run:

sudo ln –s /etc/httpd/sites-available/CooperEditing.conf / etc/httpd/sites-enabled/CooperEditing.conf

And then restart the apache service.

**Website!**

Let’s start off by making sure we are in our user’s home directory.

cd ~

My reason for doing this is that we will be downloading the files for WordPress, and this makes it easier to keep organized. Now that know we are in our home folder, let’s download the file that we need.

wget http://wordpress.org/latest.tar.gz

Since this is an archive with the most recent build of WordPress in it, we will need to expand it out by running:

tar -xzvf latest.tar.gz

Some text should fly by and you will now have a folder called wordpress in your home directory. We’re almost there! Our next step is to setup the config file our database info from before.

cp wordpress/wp-config-sample.php wordpress/wp-config.php  
nano wordpress/wp-config.php

The lines that we need to change are right at the top.

// \*\* MySQL settings - You can get this info from your web host \*\* //  
/\*\* The name of the database for WordPress \*/  
define('DB\_NAME', 'database\_name\_here');

/\*\* MySQL database username \*/  
define('DB\_USER', 'username\_here');

/\*\* MySQL database password \*/  
define('DB\_PASSWORD', 'password\_here');

Fill in the spots requesting info and save the file. After doing so we need to get the files from our home folder to the DocumentRoot. To do this we can run:

sudo mkdir /var/www/cooperediting.com  
sudo cp -R wordpress/\* /var/www/cooperediting.com

Transferring the files will end with them not having the right permissions for WordPress to run and Apache2 to display them, so let’s fix that with:

sudo chown -R www-data:www-data /var/www/\*

And if you want your user to be easily able to edit the source files:

sudo adduser -aG *username* www-data

Try going to your server’s IP in your browser, and you should get a WordPress installation page!