

Scotti-BYTE Enterprise Consulting Services

How to Build a Webserver from Scratch

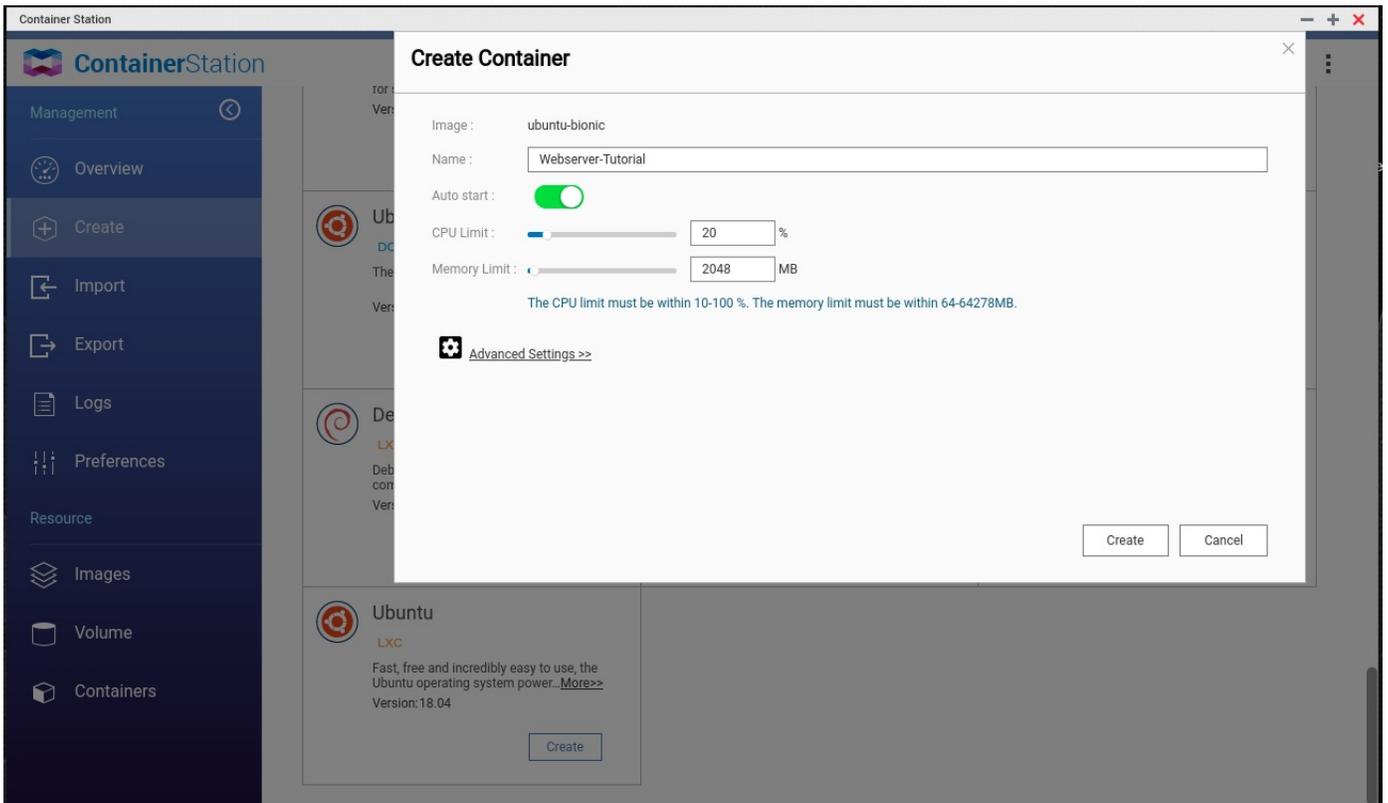
Most websites are based in the cloud on hosting services and many of those hosting services are very inexpensive and even free. This tutorial will discuss the process to building a completely self-hosted web server on private cloud. The consensus regarding this approach is that it is easier to build a web presence in the public cloud. The reason for this is that hosting services already have an accessible cloud presence and their servers are based in data centers with proper redundancy and backup.

However, there is an increasing thought that since private cloud devices are more and more prevalent, having complete control and ownership of your hosting might be the way to go. There are several ways to build a webserver. In this tutorial we will talk about using an open source website builder called "Wordpress". Wordpress is a content management system (CMS) that lets you build and maintain your website and was first released in 2003. Since that time, Wordpress has become the top (CMS) with 62% of the market share. It is believed that 35% of the Internet is powered by Wordpress Webservers and that number is still growing.

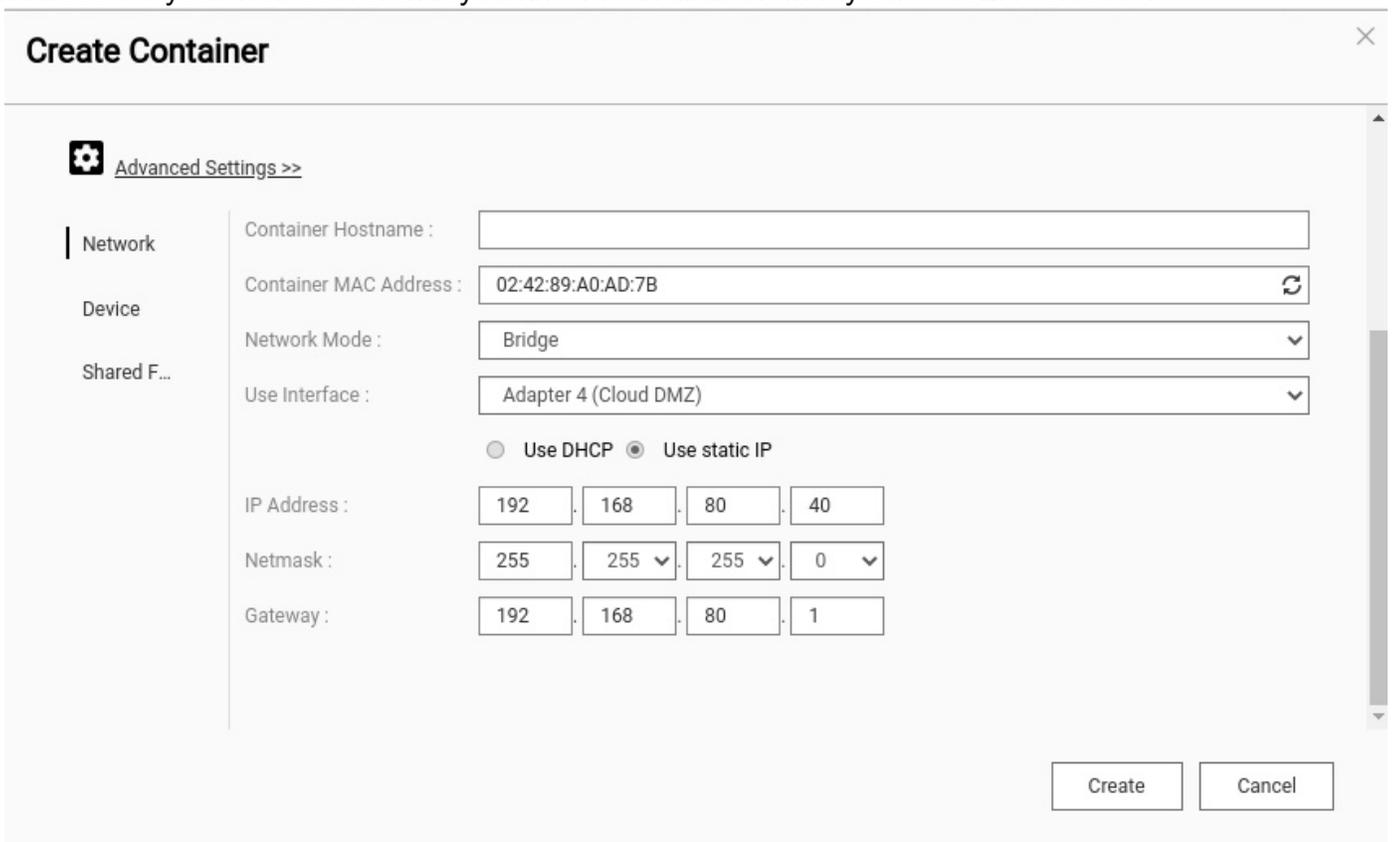
We will be building a Webserver on an LXC container. It is worthwhile to note that these steps would also work on something as minimal as a Raspberry Pi. To build our website, we need our LXC container, a user account with sudo privileges, and a LAMP stack. LAMP stands for Linux, Apache, MySQL and PHP. We will then secure our website with SSL. Once the LAMP stack is installed, we can install Wordpress and start to configure the website. In order to secure our website with SSL, we need our own domain name. There are lots of places to register a domain name. I registered a domain name on Google Domains for \$12 a year.

Once our website is built, we will be opening access to it on our router so that it can be contacted from the public Internet. Since we are creating an LXC container on a QNAP NAS, I will go through those steps. If you have a Raspberry Pi, you can skip ahead to where I install the create the sudo account and install the "ssh server". Our LXC container will be Ubuntu 18.04, but Raspberry Pi OS is Debian based and the commands are the same as for Ubuntu.

On QNAP, head on over to "Container Station" and choose "Create". Scroll to the very button and create an Ubuntu 18.04 LXC Container and make the settings like those in the following screen shot.



Next, click on advanced settings and change the Network settings to "Bridge", click the arrows to the right of the container MAC address to generate a unique/persistent hardware address, and select the adapter you want if you have more than one. Now select "Static Address" and enter an address for your web server that you know is available. In my case 192.168.80.40.



Click "Create" and on the summary page, click ok. Your LXC container is now being created. Go back to the "Overview" in container station and it should show up.



Click on the name in blue and the console will open.

```
Connected to tty 1
Type <Ctrl+a q> to exit the console, <Ctrl+a Ctrl+a> to enter Ctrl+a itself

Ubuntu 18.04.1 LTS Webserver-Tutorial pts/0

Webserver-Tutorial login: ubuntu (automatic login)

Last login: Mon Jul 6 17:26:29 UTC 2020 on pts/2
Welcome to Ubuntu 18.04.1 LTS (GNU/Linux 4.14.24-qnap x86_64)

* Documentation: https://help.ubuntu.com
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/advantage

ubuntu@Webserver-Tutorial:~$
```

This is the point where the commands would be the same for our LXC container or a Raspberry Pi from here on. At the command prompt, enter the following two commands:

```
sudo apt update
sudo apt upgrade
```

The default password in the LXC container is "ubuntu" and the username is ubuntu. Change that password:

```
sudo passwd
```

Now install the ssh server so we can access the system remotely:

```
sudo apt install openssh-server
```

Create a privileged user account:

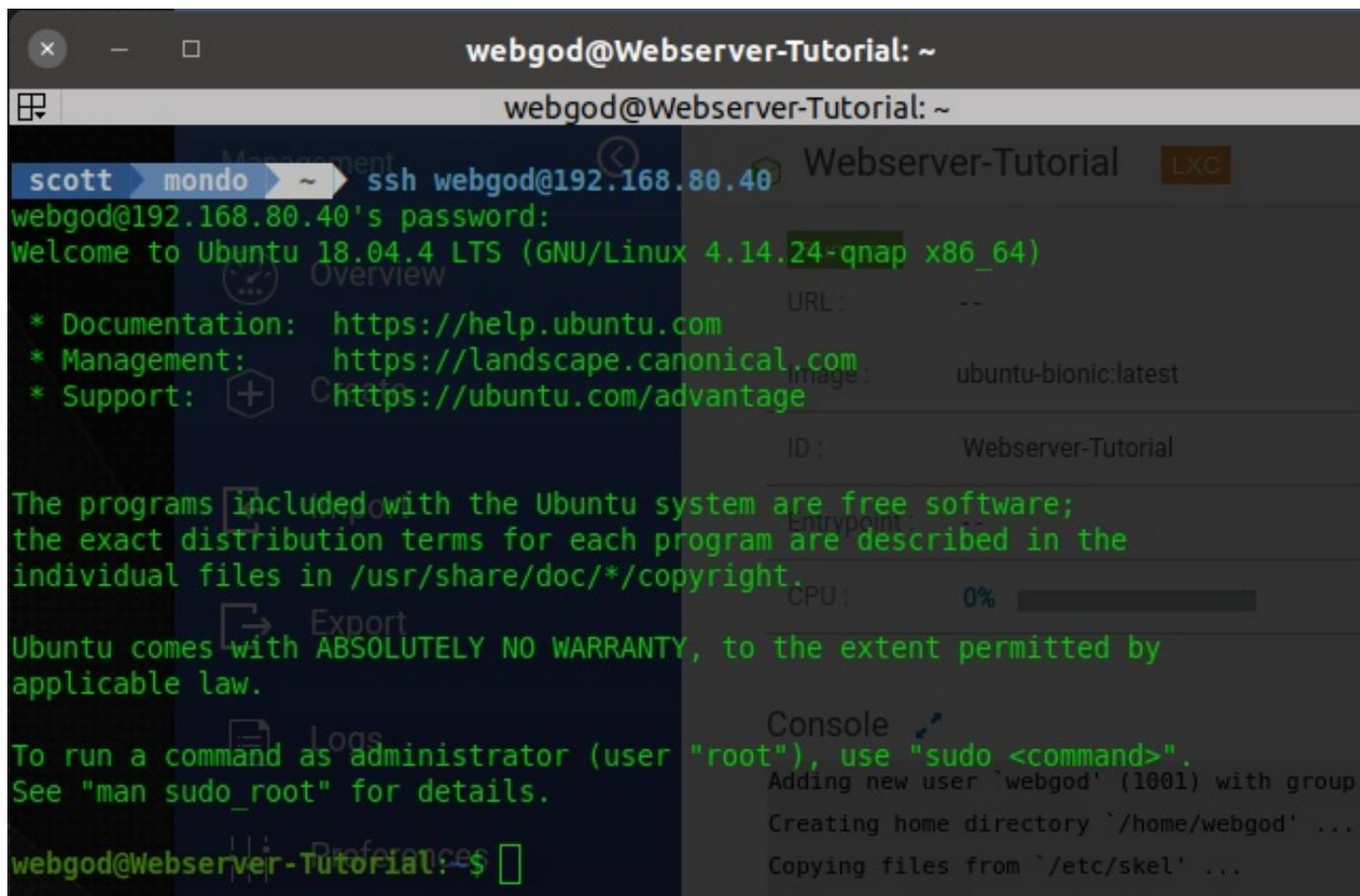
```
sudo adduser webgod
```

Grant the new user sudo privilege:

```
sudo usermod -aG sudo webgod
```

Now open a terminal and connect to the system remotely:

```
ssh webgod@192.168.80.40
```



Now we begin to install the "LAMP" stack. Install the Apache web server:

```
sudo apt install apache2
```

At this point you will be able to open a web browser and go to your address: <http://192.168.80.40/> and you will see an Apache web page.

Now we need a database. Install the mysql database package:

```
sudo apt install mysql-server
```

Secure your installation:

```
sudo mysql_secure_installation
```

You will be asked if you want to run the validate password plug-in. This is your choice and it just

improves security. Provide a password of your choosing for SQL, remove anonymous users, disallow root login remotely, and delete the test database. Finally reload the privilege tables.

Sign on to MySQL:

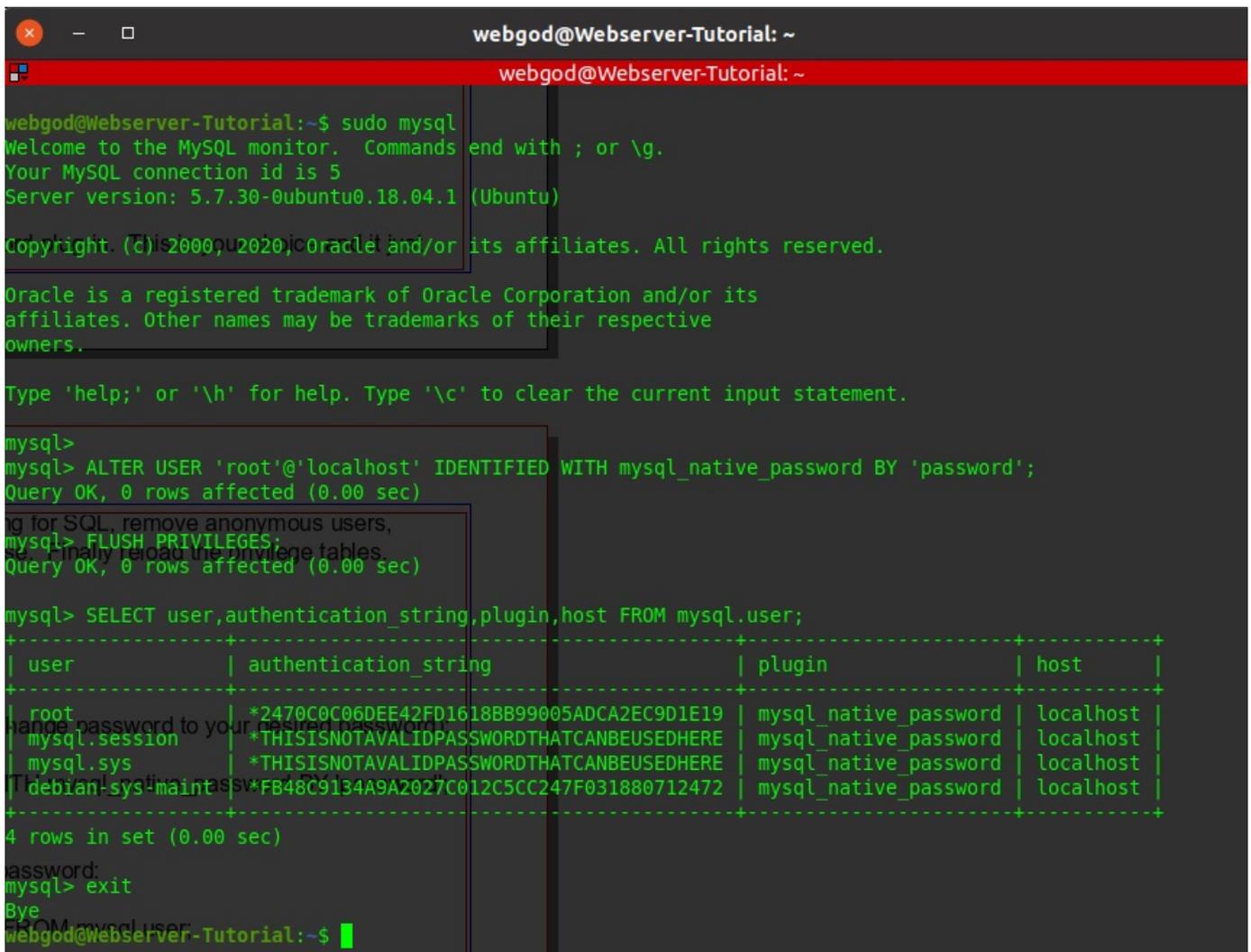
```
sudo mysql
```

Change the root account to sign on with a password (change password to your desired password):

```
ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY 'password';  
FLUSH PRIVILEGES;
```

List the users and see that "root" is set to login with a password:

```
SELECT user,authentication_string,plugin,host FROM mysql.user;  
exit
```



The screenshot shows a terminal window titled 'webgod@Webserver-Tutorial: ~'. The user runs 'sudo mysql' and is prompted with the MySQL monitor welcome message. The user then enters the following commands:

```
mysql> ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY 'password';  
Query OK, 0 rows affected (0.00 sec)  
  
mysql> FLUSH PRIVILEGES;  
Query OK, 0 rows affected (0.00 sec)  
  
mysql> SELECT user,authentication_string,plugin,host FROM mysql.user;
```

The output of the last command is a table with 4 rows and 4 columns:

user	authentication_string	plugin	host
root	*2470C0C06DEE42FD1618BB99005ADCA2EC9D1E19	mysql_native_password	localhost
mysql.session	*THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE	mysql_native_password	localhost
mysql.sys	*THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE	mysql_native_password	localhost
debian-sys-maint	*FB48C9134A9A2027C012C5CC247F031880712472	mysql_native_password	localhost

The user then enters 'exit' and the terminal returns to the shell prompt.

You will note above that I checked to see whether the root account was set to login with a password. Do not set your password to "password" as I did above. Choose a secure password.

PHP is the component of your setup that will process code to display dynamic content in your web pages. It can run scripts, connect to your MySQL databases to get information, and hand the processed content over to your web server to display. Install PHP as follows:

```
sudo apt install php libapache2-mod-php php-mysql
```

By default, if a user requests a directory from the server, Apache will first look for a file called `index.html`. We want to tell the web server to prefer PHP files over others, so make Apache look for an `index.php` file first.

We need to install the nano editor to edit a configuration file:

```
sudo apt install nano
```

To edit the configuration file:

```
sudo nano /etc/apache2/mods-enabled/dir.conf
```

Change the contents to look like this:

```
<IfModule mod_dir.c>
    DirectoryIndex index.php index.html index.cgi index.pl index.xhtml index.htm
</IfModule>
```

The change is to make `index.php` first and `index.html` second in the list. After you make the changes, CTRL-X and "Y" to exit and save the changes.

Restart the Apache server for the changes to take effect:

```
sudo systemctl restart apache2
```

Apache on Ubuntu 18.04 has one server block enabled by default that is configured to serve documents from the `/var/www/html` directory. While this works well for a single site, it can become unwieldy if you are hosting multiple sites. Instead of modifying `/var/www/html`, let's create a directory structure within `/var/www` for our domain site.

At some point, you need to go to a domain registrar and buy a domain name. I bought my domain name on Google Domains and it cost \$12 a year. Once you have a registered domain name, you will be able to give the website we are creating a name and you will also be able to get a security certificate to allow end to end encryption using Secure Socket Layer (SSL).

The next step we need is to create the directory for your `_domain` as follows:

```
sudo mkdir /var/www/your_domain
```

For our purposes, my domain name is scottibyte.com, so I will use it in the examples to follow:

```
sudo mkdir /var/www/scottibyte.com
```

The permissions of your web roots should be set by the following commands:

```
sudo chown -R $USER:$USER /var/www/scottibyte.com
```

```
sudo chmod -R 755 /var/www/scottibyte.com
```

Next, create a sample index.html page using nano:

```
sudo nano /var/www/scottibyte.com/index.html
```

Inside, add the following sample HTML:

```
<html>
  <head>
    <title>Welcome to Your_domain!</title>
  </head>
  <body>
    <h1>Success! The your_domain server block is working!</h1>
  </body>
</html>
```

Save and close the file when you are finished.

In order for Apache to serve this content, it's necessary to create a virtual host file with the correct directives. Instead of modifying the default configuration file located at /etc/apache2/sites-available/000-default.conf directly, let's make a new one at /etc/apache2/sites-available/scottibyte.com.conf

```
sudo nano /etc/apache2/sites-available/scottibyte.com.conf
```

Paste in the following configuration block, which is similar to the default, but updated for our new directory and domain name (I had to use www2, because I already have a www):

```
<VirtualHost *:80>
  ServerAdmin vmsman@scottibyte.com
  ServerName www2.scottibyte.com
  ServerAlias www2.scottibyte.com
  DocumentRoot /var/www/scottibyte.com
  ErrorLog ${APACHE_LOG_DIR}/error.log
  CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Make the changes to the file above for your site appropriately and save and exit the editor.

Let's enable the file with the a2ensite tool:

```
sudo a2ensite scottibyte.com.conf
```

Disable the default site defined in 000-default.conf:

```
sudo a2dissite 000-default.conf
```

Next install net tools:

```
sudo apt install net-tools
```

Install curl:

```
sudo apt install curl
```

We need to know what IP address to assign to our website. You can assign a IPv4 address if you are not using port 443 for SSL on your router. In that case you can create a port forward rule for port 443 to the internal IPv4 address of the server we are building. We need to know the external address of your router on the Internet though, because the DNS entry will point to your router. To find your router WAN address:

```
curl http://icanhazip.com
```

You would use the IPv4 address from the router obtained above to create a DNS "A" record named www2.scottibyte.com, in my particular case.

Since IPv6 gives every system on your network a local address, you can issue the following command to see the ipv6 address of the server we are creating:

```
ifconfig
```

As an example, I have highlighted my IPv6 address for the example server we are creating below. Your IPv6 global address will be different.

```
webgod@Webserver-Tutorial:~$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 192.168.80.40  netmask 255.255.255.0  broadcast 192.168.80.255
    inet6 2601:2c4:8101:e1::200  prefixlen 128  scopeid 0x0<global>
    inet6 fe80::42:89ff:fea0:ad7b  prefixlen 64  scopeid 0x20<link>
    inet6 2601:2c4:8101:e1:42:89ff:fea0:ad7b  prefixlen 64  scopeid 0x0<global>
    ether 02:42:89:a0:ad:7b  txqueuelen 1000  (Ethernet)
    RX packets 267800  bytes 148704597 (148.7 MB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 21117  bytes 2011006 (2.0 MB)
    TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0
```

You would use your IPv6 global address to create a DNS "AAAA" record named www2.scottibyte.com, in my particular case.

Assuming you have defined either the A or the AAAA record or even both at your DNS provider, we are ready for the next step. Don't worry about opening the server on your router/firewall, because we will do that later.

Now edit the file the apache startup file:

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

At the very bottom of the file, create a new line:

```
ServerName localhost
```

CTRL-X and "Y" to save the file and exit the editor.

Restart Apache:

```
sudo systemctl restart apache2
```

Next, let's test for configuration errors:

```
sudo apache2ctl configtest
```

You should see the following output:

```
Syntax OK
```

Apache should now be serving your domain name. You can test this by navigating to <http://www2.scottibyte.com> (your address will be different) where you should see this:

Success! The your_domain server block is working!

In order to test that your system is configured properly for PHP, create a very basic PHP script called info.php. In order for Apache to find this file and serve it correctly, it must be saved to your web root directory.

Create the file at the web root you created in the previous step by running:

```
sudo nano /var/www/scottibyte.com/info.php
```

This will open a blank file. Add the following text, which is valid PHP code, inside the file:

```
<?php  
phpinfo();  
?>
```

When you are finished, save and close the file with CTRL-X and then "Y".

Now you can test whether your web server is able to correctly display content generated by this PHP script. To try this out, visit this page in your web browser (with your domain):

```
http://www2.scottibyte.com/info.php
```

The web browser should display a PHP Info page.

This page provides some basic information about your server from the perspective of PHP. It is useful for debugging and to ensure that your settings are being applied correctly.

If you can see this page in your browser, then your PHP is working as expected.

You probably want to remove this file after this test because it could actually give information about your server to unauthorized users. To do this, run the following command:

```
sudo rm /var/www/scottibyte.com/info.php
```

Now that we have a website in Apache, the next step will be to secure the website with SSL. WordPress serves dynamic content and handles user authentication and authorization. TLS/SSL is the technology that allows you to encrypt the traffic from your site so that your connection is secure.

Since we have a domain name and either an "A" record or an "AAAA" record (or both) for our web server that we added earlier, these are the key prerequisites to getting a security certificate.

We will use Let's Encrypt to obtain an SSL certificate is to install the Certbot software on your server.

Install the following prerequisite:

```
sudo apt-get install software-properties-common
```

Add the repository:

```
sudo add-apt-repository ppa:certbot/certbot
```

You'll need to press ENTER to accept.

Install Certbot's Apache package:

```
sudo apt install python-certbot-apache
```

Certbot needs to be able to find the correct virtual host in your Apache configuration for it to automatically configure SSL. It does this by looking for a ServerName directive that matches the domain you request a certificate for.

We set that value in your site configuration file `/etc/apache2/sites-available/scottibyte.com.conf` (your name will differ).

Before we can get a security certificate, we need to open access through your firewall/router. This is done either by port forwarding port 443 to the IPv4 address inside your network or by creating a IPv6 WAN IN router rule which is covered in my IPv6 tutorials.

Be sure to create your firewall rule now. Allow port 80 and 443 through your firewall.

The Certbot Apache plugin will take care of reconfiguring Apache and reloading the config whenever necessary. To use this plugin, type the following (with your server name):

```
sudo certbot --apache -d www2.scottitybe.com
```

This routine will ask for your email and also if you want to redirect all traffic through SSL. Choose the SSL option:

```
webgod@Webserver-Tutorial: ~
webgod@Webserver-Tutorial:~$ sudo certbot --apache -d www2.scottitybe.com
Saving debug log to /var/log/letsencrypt/letsencrypt.log
Plugins selected: Authenticator apache, Installer apache
Obtaining a new certificate
Performing the following challenges:
http-01 challenge for www2.scottitybe.com
Enabled Apache rewrite module
Waiting for verification...
Cleaning up challenges
Created an SSL vhost at /etc/apache2/sites-available/scottitybe.com-le-ssl.conf
Enabled Apache socache_shmcb module
Enabled Apache ssl module
Deploying Certificate to VirtualHost /etc/apache2/sites-available/scottitybe.com-le-ssl.conf
Enabling available site: /etc/apache2/sites-available/scottitybe.com-le-ssl.conf
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): 2
Enabled Apache rewrite module
Redirecting vhost in /etc/apache2/sites-enabled/scottitybe.com.conf to ssl vhost in /etc/apache2/sites-available/scottitybe.com-le-ssl.conf
-----
Congratulations! You have successfully enabled https://www2.scottitybe.com

You should test your configuration at:
https://www.ssllabs.com/ssltest/analyze.html?d=www2.scottitybe.com
-----

IMPORTANT NOTES:
- Congratulations! Your certificate and chain have been saved at:
  /etc/letsencrypt/live/www2.scottitybe.com/fullchain.pem
  Your key file has been saved at:
  /etc/letsencrypt/live/www2.scottitybe.com/privkey.pem
  Your cert will expire on 2020-10-05. To obtain a new or tweaked
  version of this certificate in the future, simply run certbot again
  with the "certonly" option. To non-interactively renew *all* of
  your certificates, run "certbot renew"
- If you like Certbot, please consider supporting our work by:

  Donating to ISRG / Let's Encrypt: https://letsencrypt.org/donate
  Donating to EFF: https://eff.org/donate-le

webgod@Webserver-Tutorial:~$
```

Let's Encrypt's certificates are only valid for ninety days. This is to encourage users to automate their certificate renewal process. The certbot package we installed takes care of this for us by adding a renew script to /etc/cron.d. This script runs twice a day and will automatically renew any certificate that's within thirty days of expiration.

To test the renewal process, you can do a dry run with certbot:

```
sudo certbot renew --dry-run
```

Now we have an operational website that uses SSL encryption. WordPress uses MySQL to manage and store site and user information. We have MySQL installed already, but we need to make a database and a user for WordPress to use.

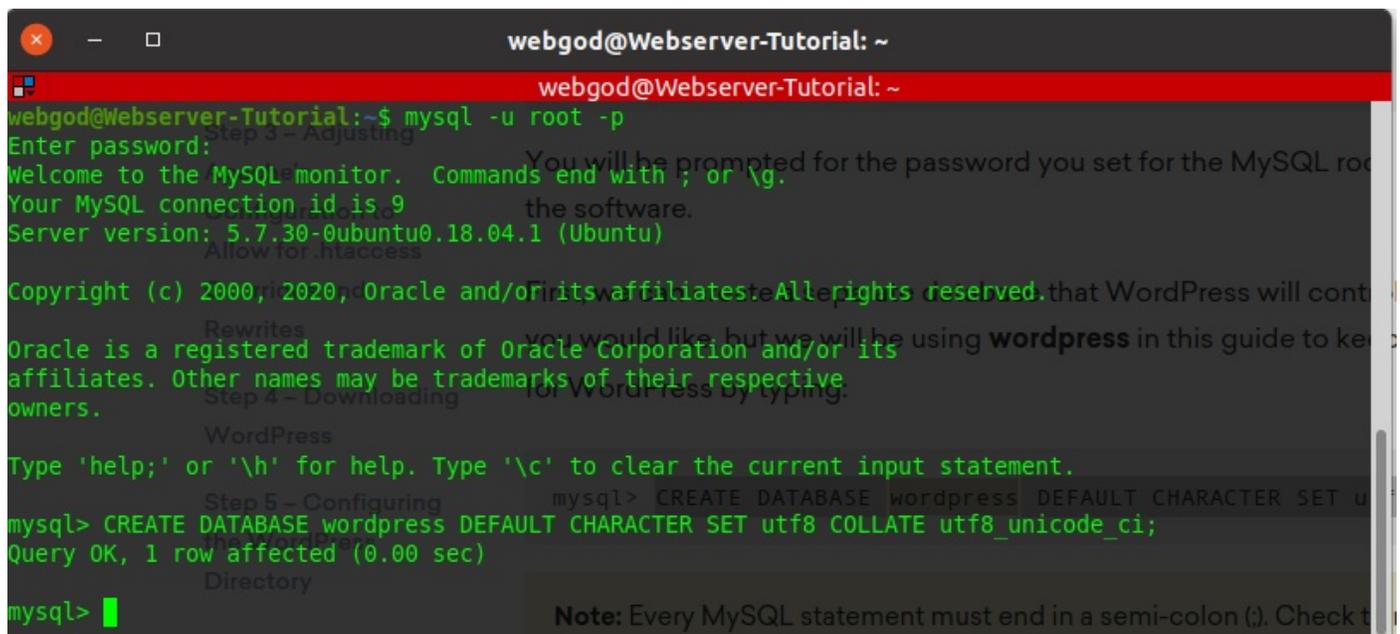
To get started, log into the MySQL root (administrative) account by issuing this command:

```
mysql -u root -p
```

You will be prompted for the password you set for the MySQL root account when you installed mysql earlier in the tutorial.

First, we can create a separate database that WordPress will control. You can call this whatever you would like, but we will be using wordpress in this guide to keep it simple. Create the database for WordPress at the mysql prompt by typing:

```
CREATE DATABASE wordpress DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;
```



```
webgod@Webserver-Tutorial: ~
webgod@Webserver-Tutorial: ~
webgod@Webserver-Tutorial:~$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 5.7.30-0ubuntu0.18.04.1 (Ubuntu)

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE DATABASE wordpress DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;
Query OK, 1 row affected (0.00 sec)

mysql> █
```

Next, we are going to create a separate MySQL user account that we will use exclusively to operate on our new database. Creating one-function databases and accounts is a good idea from a management and security standpoint. We will use the name wordpressuser in this guide.

We are going to create this account, set a password, and grant access to the database we created. We can do this by typing the following command. Remember to choose a strong password here for your database user (and not 'password'):

```
GRANT ALL ON wordpress.* TO 'wordpressuser'@'localhost' IDENTIFIED BY 'password';
```

You now have a database and user account, each made specifically for WordPress. We need to flush the privileges so that the current instance of MySQL knows about the recent changes we've made:

```
FLUSH PRIVILEGES;  
EXIT;
```

When setting up our LAMP stack, we only required a very minimal set of extensions in order to get PHP to communicate with MySQL. WordPress and many of its plugins leverage additional PHP extensions.

We can download and install some of the most popular PHP extensions for use with WordPress by typing:

```
sudo apt update  
sudo apt upgrade  
sudo apt install php-curl php-gd php-mbstring php-xml php-xmlrpc php-soap php-intl php-zip
```

Next, we will be making a few minor adjustments to our Apache configuration. Based on the prerequisite tutorials, you should have a configuration file for your site in the `/etc/apache2/sites-available/` directory. We'll use `/etc/apache2/sites-available/wordpress.conf` as an example here, but you should substitute the path to your configuration file where appropriate.

Additionally, we will use `/var/www/wordpress` as the root directory of our WordPress install. You should use the web root specified in your own configuration.

Currently, the use of `.htaccess` files is disabled. WordPress and many WordPress plugins use these files extensively for in-directory tweaks to the web server's behavior.

Open the Apache configuration file for your website:

```
sudo nano /etc/apache2/sites-available/scottibyte.com-le-ssl.conf
```

Add the following to this file:

```
<Directory /var/www/wordpress/>  
    AllowOverride All  
</Directory>
```

You will notice that the domain name with `"-le-ssl.conf"` was created for you with the SSL procedure. At this point, the file should look something like the following:

```
webgod@Webserver-Tutorial: /etc/apache2/sites-available
webgod@Webserver-Tutorial: /etc/apache2/sites-available
GNU nano 2.9.3 /etc/apache2/sites-available/scottitybe.com-le-ssl.conf Modified

<IfModule mod_ssl.c>
# Configuration file for your website:
<VirtualHost *:443>
    ServerAdmin vmsman@scottitybe.com
    ServerName www2.scottitybe.com
    ServerAlias www2.scottitybe.com
    DocumentRoot /var/www/scottitybe.com
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

# To make the directory available to SSL we need to add a <Directory> block
# files, we need to set the AllowOverride directive within a Directory block
    SSLCertificateFile /etc/letsencrypt/live/www2.scottitybe.com/fullchain.pem
    SSLCertificateKeyFile /etc/letsencrypt/live/www2.scottitybe.com/privkey.pem
    Include /etc/letsencrypt/options-ssl-apache.conf
</VirtualHost>
</IfModule>
<Directory /var/www/wordpress/>
    AllowOverride All
</Directory>

Save and close the file:
^G Get Help      ^O Write Out    ^W Where Is    ^K Cut Text     ^J Justify     ^C Cur Pos
^X Exit          ^R Read File    ^N Replace     ^U Uncut Text  ^T To Spell    ^_ Go To Line
```

Now do a CTRL-X and "Y" to save.

Restart Apache to implement the changes:

```
sudo systemctl restart apache2
```

Now that our server software is configured, we can download and set up WordPress.

Change into a writable directory and then download the compressed WordPress release by typing:

```
cd /tmp
curl -O https://wordpress.org/latest.tar.gz
```

Extract the compressed file to create the WordPress directory structure:

```
tar xzvf latest.tar.gz
```

We will be moving these files into our document root momentarily. Before we do, we can add a dummy .htaccess file so that this will be available for WordPress to use later.

Create the file by typing:

```
touch /tmp/wordpress/.htaccess
```

Copy over the sample configuration file to the filename that WordPress actually reads:

```
cp /tmp/wordpress/wp-config-sample.php /tmp/wordpress/wp-config.php
```

Create the upgrade directory, so that WordPress won't run into permissions issues when trying to do this on its own following an update to its software:

```
mkdir /tmp/wordpress/wp-content/upgrade
```

Copy the entire contents of the directory into our document root. We are using a dot at the end of our source directory to indicate that everything within the directory should be copied, including hidden files (like the .htaccess file we created). Make sure you replace "scottibyte.com" with your site name in the following command:

```
sudo cp -a /tmp/wordpress/. /var/www/scottibyte.com
```

One of the big things we need to accomplish is setting up reasonable file permissions and ownership for the Wordpress files.

We'll start by giving ownership of all the files to the www-data user and group. This is the user that the Apache webserver runs as, and Apache will need to be able to read and write WordPress files in order to serve the website and perform automatic updates.

Update the ownership with chown:

```
sudo chown -R www-data:www-data /var/www/scottibyte.com
```

Next we will run two find commands to set the correct permissions on the WordPress directories and files:

```
sudo find /var/www/scottibyte.com/ -type d -exec chmod 750 {} \;  
sudo find /var/www/scottibyte.com/ -type f -exec chmod 640 {} \;
```

Now, we need to make some changes to the main WordPress configuration file.

When we open the file, our first order of business will be to adjust some secret keys to provide some security for our installation. WordPress provides a secure generator for these values so that you do not have to try to come up with good values on your own. These are only used internally, so it won't hurt usability to have complex, secure values here.

To grab secure values from the WordPress secret key generator, type:

```
curl -s https://api.wordpress.org/secret-key/1.1/salt/
```

The lines that are output from the command above should be copied into your cut buffer and pasted into the file (your domain should be substituted):

```
sudo nano /var/www/scottibyte.com/wp-config.php
```

Search for the similar lines that you cut in the file above and replace them with the results from the curl command above.

Next, we need to modify some of the database connection settings at the beginning of the file. You need to adjust the database name, the database user, and the associated password that we configured within MySQL.

The other change we need to make is to set the method that WordPress should use to write to the filesystem. Since we've given the web server permission to write where it needs to, we can explicitly set the filesystem method to "direct". Failure to set this with our current settings would result in WordPress prompting for FTP credentials when we perform some actions.

This setting can be added below the database connection settings, or anywhere else in the file:

```
/var/www/scottibyte.com/wp-config.php
```

```
...
```

```
define('DB_NAME', 'wordpress');
```

```
/** MySQL database username */  
define('DB_USER', 'wordpressuser');
```

```
/** MySQL database password */  
define('DB_PASSWORD', 'password');
```

```
...
```

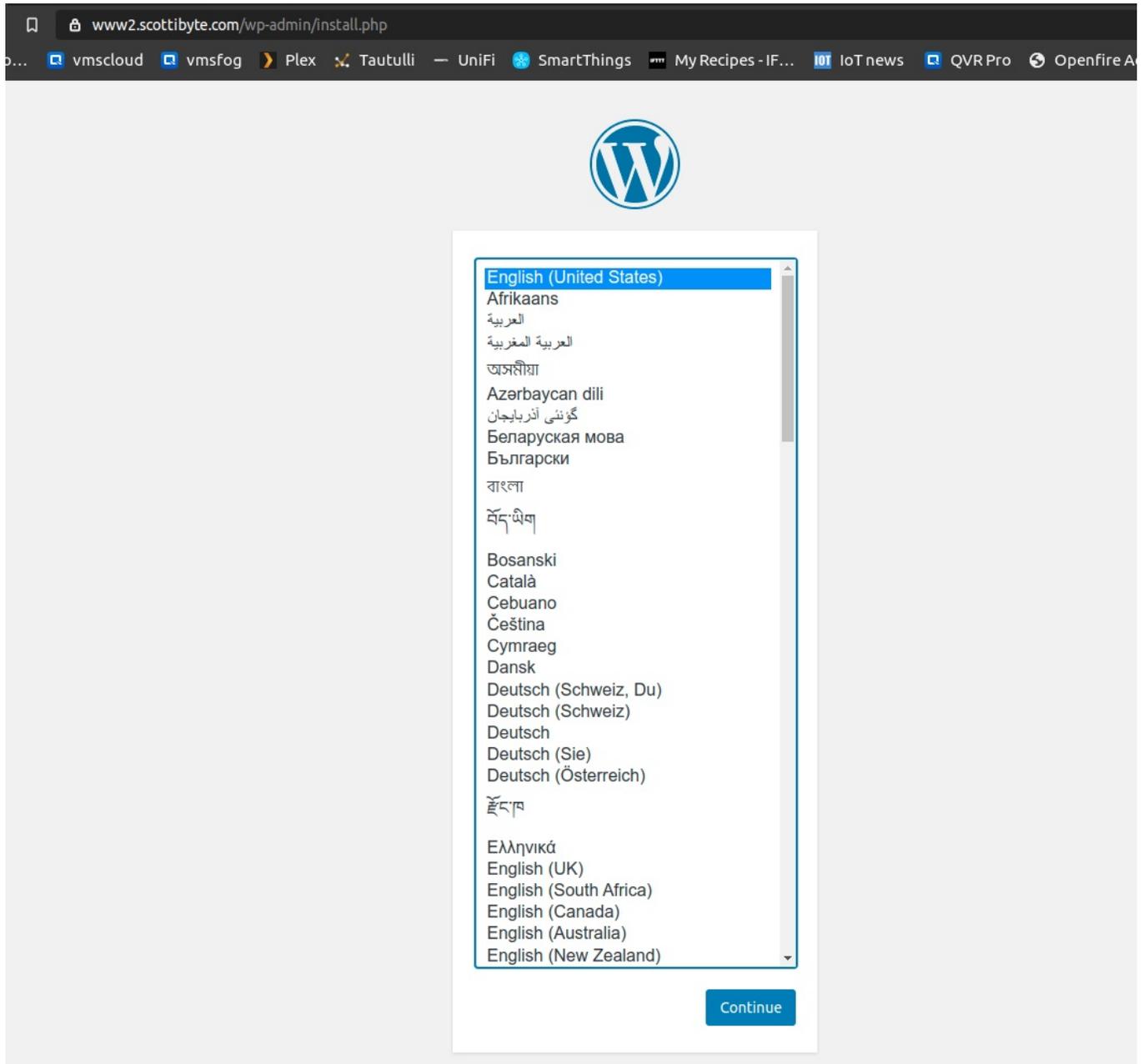
```
define('FS_METHOD', 'direct');
```

CTRL-X and "Y" to save and close the file.

Congratulations, we are ready to configure Wordpress. Open your web browser and go to the address of your website. In my case it is:

```
https://www2.scottibyte.com/
```

At this point, the web interface should show the following:



The next page, you give your website a name and establish a username and password.



Welcome

Welcome to the famous five-minute WordPress installation process! Just fill in the information below and you'll be on your way to using the most extendable and powerful personal publishing platform in the world.

Information needed

Please provide the following information. Don't worry, you can always change these settings later.

Site Title

Username
Usernames can have only alphanumeric characters, spaces, underscores, hyphens, periods, and the @ symbol.

Password [Hide](#)
Strong
Important: You will need this password to log in. Please store it in a secure location.

Your Email
Double-check your email address before continuing.

Search Engine Visibility Discourage search engines from indexing this site
It is up to search engines to honor this request.

[Install WordPress](#)

Next click on install Wordpress. You should see this screen after:



Success!

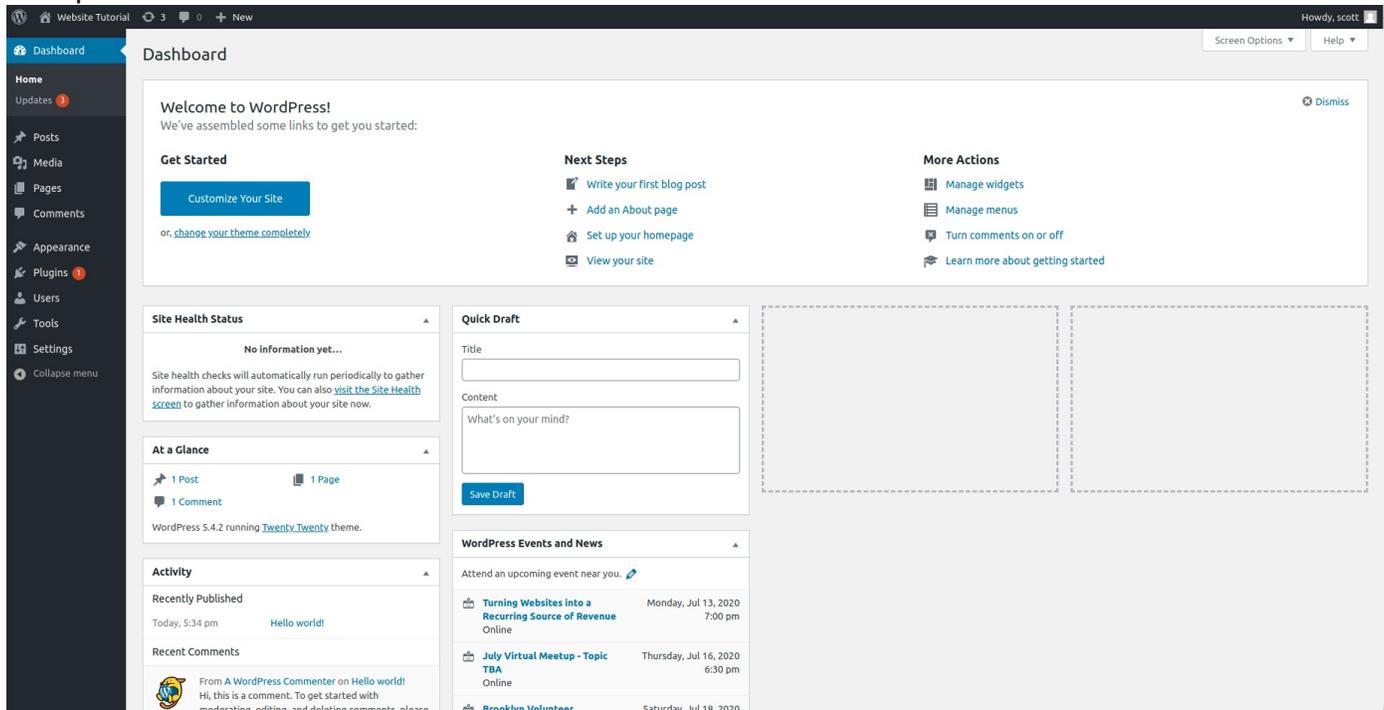
WordPress has been installed. Thank you, and enjoy!

Username scott

Password *Your chosen password.*

[Log In](#)

Click the login button and provide your username and password. If all is well, you will see the WordPress dashboard:



Now you have a fully functional SSL encrypted website accessible from the public internet. You will need to learn Wordpress to create your custom website. From this point on, all the options are point and click configurable.

The advantage to this solution is that your website is completely self hosted and controlled by you from your private network. All components of the website are locally based.

Wordpress has thousands of templates that you can access from the dashboard and configure to meet your needs. There are several great online videos which will show you how to use Wordpress to configure and customize your website.

