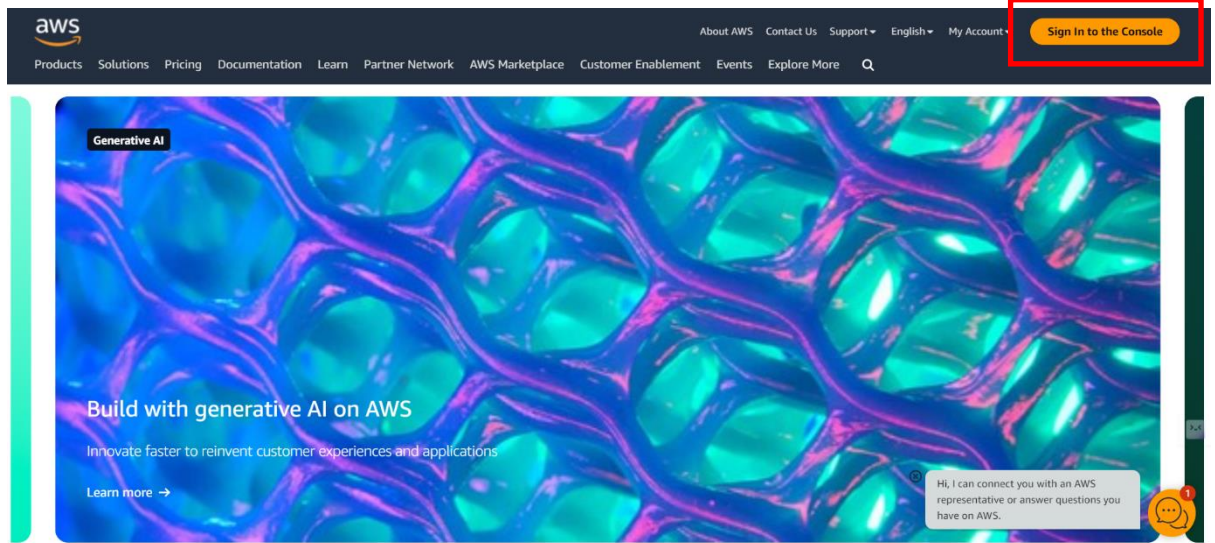


James Chellew - 35377181

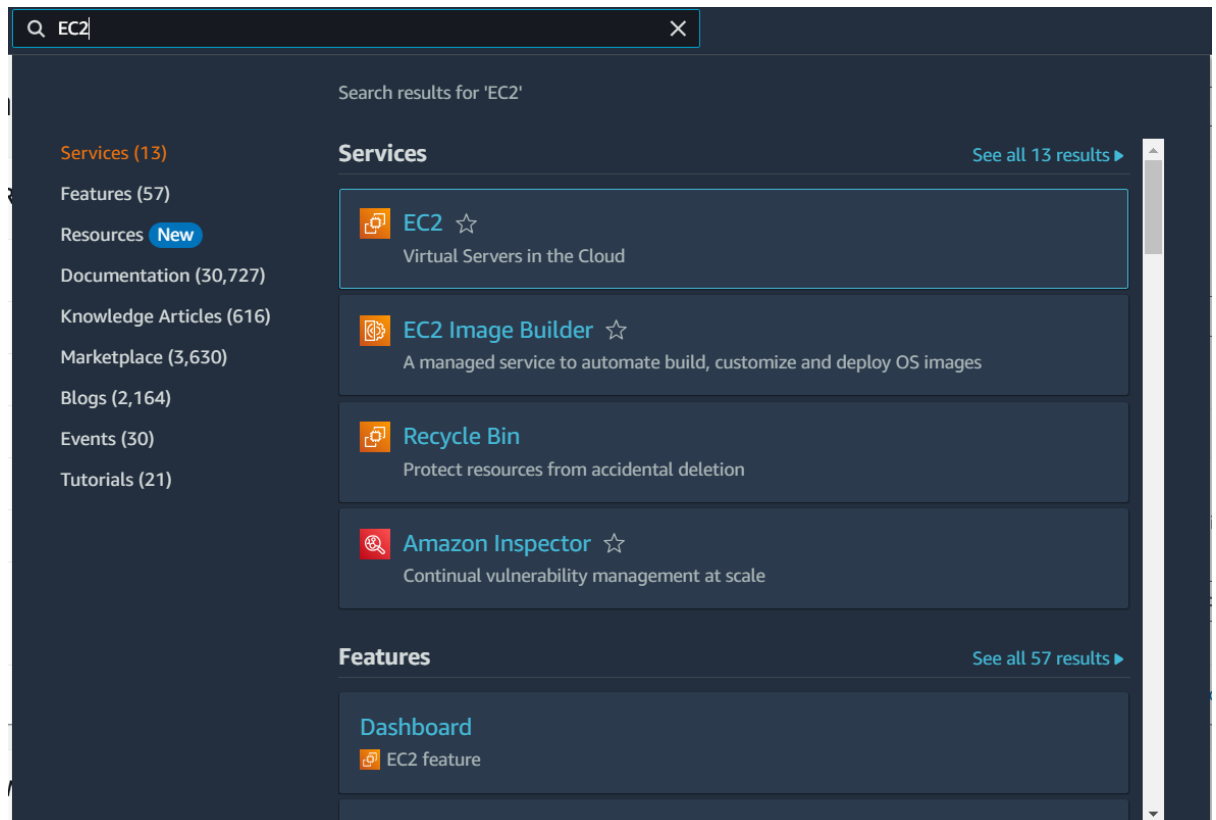
Documentation For www.jameschellew.xyz Server

Making an Amazon EC2

1. Go to <https://aws.amazon.com/>
2. Sign into console.



3. Search for EC2



4. On the Dashboard, look for "Launch Instance".
5. Select Ubuntu as the OS
6. Select the t2.micro instance type
7. Generate a Key Pair
 - a. Save the private key somewhere safe. You will use this to log into the machine later.
8. Ensure SSH is enabled in the security group

A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

☒ Create security group

☐ Select existing security group

We'll create a new security group called '**launch-wizard-3**' with the following rules:

- ☒ Allow SSH traffic from
Helps you connect to your instance

Anywhere
0.0.0.0/0
- ☐ Allow HTTPS traffic from the internet
To set up an endpoint, for example when creating a web server
- ☐ Allow HTTP traffic from the internet
To set up an endpoint, for example when creating a web server

9. Add a "New Volume" of storage up to 22GB (the free upper limit is 30GB)
10. When you are happy with all the changes. Press launch instance.
11. Head back to the EC2 dashboard and check that your instance has been set up.



Initialising Website Resources from GitHub Repo

The below script will automatically sync your website with the site's current git repository. It will also back up any changes made to the website's information to the git repository. There are a few steps we need to take before we can run this script.

1. If you haven't already, set up your username and email.

```
git config --global user.name "name here"
git config --global user.email "email here"
```

2. Start by cloning the repository into the directory /var/www/html/

```
cd /var/www/html
git clone https://github.com/JamesChellew/sites
```

3. Back to your VM, if you haven't already, set up your username and email.

```
git config --global user.name "name here"
git config --global user.email "email here"
```

4. Go into sites directory and set up a remote origin and the script's remote origin.
Important note: the script requires the SSH URL to work. The other origin is the normal https link. The origin should already be set up if you clone the repo, but we will set it anyway for completeness.

```
cd /var/www/html/sites
git remote add script_origin git@github.com:JamesChellew/sites.git
git remote add origin https://github.com/JamesChellew/sites
```

5. To perform git commands, you need to be in inside the sites directory.
6. To make git pull requests:

```
git pull origin main
```

7. To stage all changes

```
git add .
```

8. To commit changes locally

```
git commit -m "message here"
```

9. To push these to remote

```
git push origin main
```

Pull and Push script

10. We must make an RSA key and register the public key with the GitHub repository in order to run the script. Cd into your .ssh directory and run the ssh-keygen command.

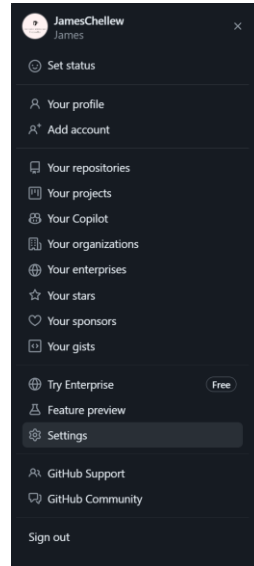
```
cd ~/.ssh && ssh-keygen
```

11. And define where they are located.

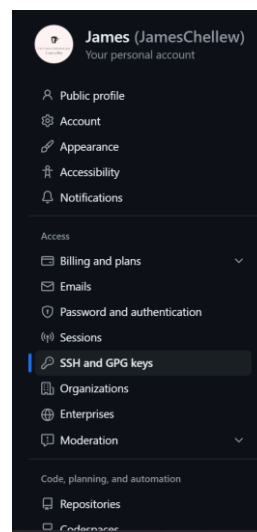
```
ssh-add ~/.ssh/id-rsa
```

12. You'll then need to copy the public key to your clipboard so we can create a new SSH key on GitHub

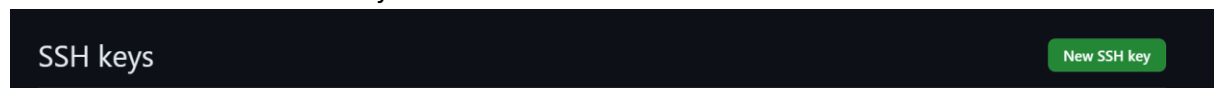
13. Got to "Settings"



14. Then SSH and GPG keys



15. And then make a new SSH Key



16. Name the key something meaningful and then paste in the public key in the key field.

A screenshot of the 'Add SSH key' form in the GitHub mobile app. The form has a dark background with white text. It includes a 'Title' field with the placeholder text 'NameItSomething'. Below it is a 'Key type' dropdown menu set to 'Authentication Key'. The 'Key' field is a large text area with placeholder text: 'Begins with 'ssh-rsa', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', 'ecdsa-sha2-nistp521', 'ssh-ed25519', 'sk-ecdsa-sha2-nistp256@openssh.com', or 'sk-ssh-ed25519@openssh.com''. At the bottom of the form is a green button labeled 'Add SSH key'.

17. Once added, go to the repository <https://github.com/JamesChellew/sites> and click on the green code button > SSH and copy the git@... line.

18. Touch a new file. Let's call it git-pull-push.sh. This will be our script file.

```
cd ~  
touch git-pull-push.sh
```

19. Edit the file with sudo nano.

```
sudo nano git-pull-push.sh
```

20. Paste in this script

```
#!/bin/bash

# Date in format Day-Month-Year
date=$(date +"%Y-%m-%d %T")

# Commit message
message="Backup Commit for ${date}"
cd /var/www/html/sites/
git pull script_origin main
git add .
git commit -m"${message}" -q
status="$(git status --branch --porcelain)"
echo $status >> /home/ubuntu/cron_echo.txt
if [ "$status" == "## main...origin/main" ]; then
    echo "${date}: IT IS CLEAN: Nothing to push" >> /home/ubuntu/cron_echo.txt
else
    echo "There is stuff to push" >> /home/ubuntu/cron_echo.txt
    echo "Pushing ${message}" >> /home/ubuntu/cron_echo.txt
    git push -u script_origin main
fi
```

21. Save the file

22. Save it somewhere. You may leave it in the home directory, but scripts should be put in /usr/local/bin.

23. Make sure the script is owned by your user and has execute permissions for the owner.

```
sudo cp git-pull-push.sh /usr/local/bin/git-pull-push.sh
sudo chmod 744 /usr/local/bin/git-pull-push.sh
sudo chown [user]:[user] /usr/local/bin/git-pull-push.sh
```

Setting up your domain

Getting a domain name

To get started, we want to look at domain registries. A domain registry manages the internet traffic that tries to look for your domain name. When someone tries to access your website from your domain name, the registry will take that incoming traffic and redirect it to the IP address you specified in what is commonly known as an address record.

You have some freedom to choose what domain registry you would like to use, but I will be using Porkbun (<https://porkbun.com/>) for this documentation.

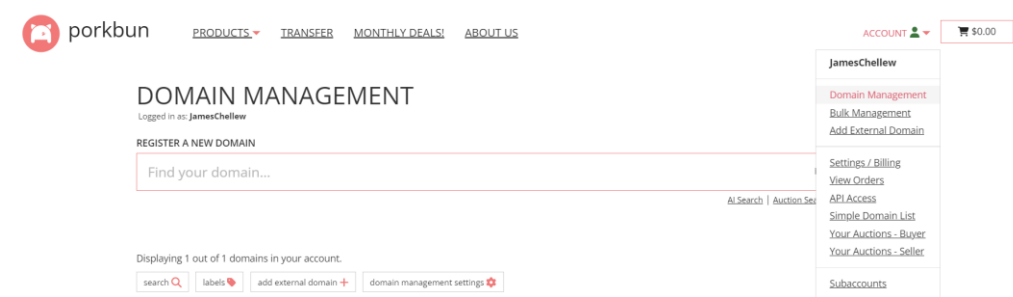
1. Make an account with Porkbun
2. Search for a domain name



3. What you have searched for is technically a second-level domain. The search results will show you all of the top-level domains available for purchase with your specified second-level domain. Prices will vary from one domain name to the next. More popular domains will tend to be more expensive (e.g. *.com will usually be more expensive than *.xyz). For our purposes, choose the cheapest one you like.

domainhere.name		\$7.29 / year renews at \$7.29	+
domainnamehere.xyz		unavailable ⓘ	
domainnamehere.inc	1st Year Sale!	\$2061.18 \$258.92 / year renews at \$2060.23	+
domainnamehere.it.com	1st Year Sale!	\$31.20 \$13.95 / year renews at \$31.20	+
domainnamehere.app		\$12.85 / year renews at \$12.85	+
domainnamehere.dev	1st Year Sale!	\$10.79 \$5.64 / year renews at \$10.79	+
domainnamehere.gay		error	
domainnamehere.info	1st Year Sale!	\$18.52 \$3.07 / year renews at \$18.52	+
domainnamehere.pro	1st Year Sale!	\$18.52 \$2.55 / year	+

4. Unfortunately, to reserve and use a domain is not free, so you must purchase one to proceed.
5. Once you have purchased a domain name, head to domain management.



6. Here, you will see the domain you have purchased. For example, at the time of writing the documentation I have the domain jameschellew.xyz

<input type="checkbox"/>	NAME	WEBSITE	EMAIL	RENEW	LOCK	WHOIS	MARKET	EXPIRES ↑	
<input type="checkbox"/>	jameschellew.xyz ⓘ						\$	2025-03-27	Details ▼

Displaying 1 out of 1 domains in your account.

7. Click on the details dropdown to see the details of your domain. In here, we are mainly interested in the DNS records section.



8. Click on this to manage the records.

9. At the bottom of this page, you will see some records set up by default. We do not need them for our purposes, so you can delete them.
10. First, we are going to set up an address record which will point to the IP address of the machine you are hosting your website on. In other words, we are telling Porkbun that we want people searching for www.jameschellew.xyz (in the case below) to be directed to the server we have set up.

Here you can configure custom DNS records for your domain.

Type
A - Address record

Host
Leave blank to create a record for the root domain. Use * to create a wildcard. Please note that ALIAS records do not support wildcards.
www
.jameschellew.xyz

Answer
[public ip address here]

TTL
600

Priority

Notes
This is for your own use and does not affect DNS.
[add notes for address record here]

Cancel Add

11. Once you have added this, you should be able to access your website via that domain name (note: it will not be a secure connection as we have not set up SSL certificates yet. As I have already done this, it is not showing the “not secure” warning in the search bar.)



12. To recreate this project, add another subdomain, “WordPress”.

Here you can configure custom DNS records for your domain.

Type
A - Address record

Host
Leave blank to create a record for the root domain. Use * to create a wildcard. Please note that ALIAS records do not support wildcards.
wordpress
.jameschellew.xyz

Answer
[public ip address here]

TTL
600

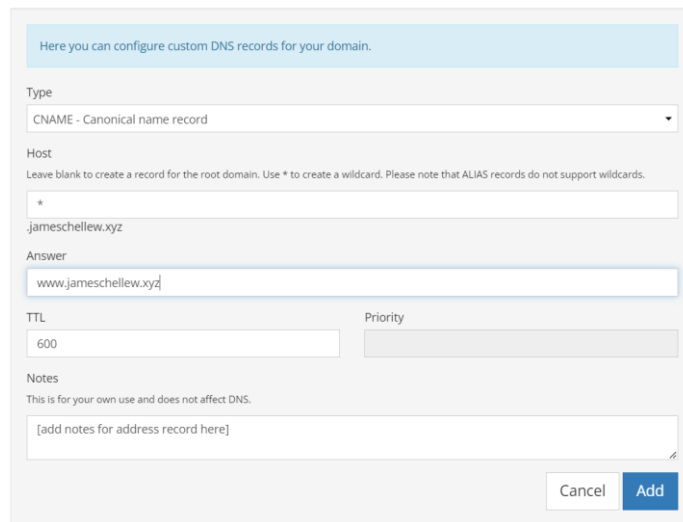
Priority

Notes
This is for your own use and does not affect DNS.
[add notes for address record here]

Cancel Add

13. And lastly, we will make a CName record that will point back to first address record we made. This redirects any other traffic to unspecified domains to the www subdomain. In

other words, it is just routing everything we haven't specified to refer to www.jameschellew.xyz



Here you can configure custom DNS records for your domain.

Type
CNAME - Canonical name record

Host
Leave blank to create a record for the root domain. Use * to create a wildcard. Please note that ALIAS records do not support wildcards.
*
jameschellew.xyz

Answer
www.jameschellew.xyz

TTL
600

Priority

Notes
This is for your own use and does not affect DNS.
[add notes for address record here]

Cancel Add

14. You should have something like this once you're all done:

A	wordpress.jameschellew.xyz
A	www.jameschellew.xyz
CNAME	*.jameschellew.xyz

Setting up Nginx Server Configuration File

Now that we have set up the address records for our server, we can start to tell nginx what files to look for when certain depending on the request (i.e. wordpress.* or www.*).

First, cd into the /etc/nginx/sites-available directory and sudo nano open the default configuration file

```
cd /etc/nginx/sites-available
sudo nano default
```

If you cloned the site, remove the current contents of your file and copy + paste the following code and change the highlighted bits where necessary. Make sure to change the php version to the one installed on your machine.

The first server block is a fall-through redirecting any unspecified traffic to <https://www.jameschellew.xyz>. The second block specifies that (in order) it should only respond to traffic looking for the server names listed, where the root directory for this traffic is on the machine and what default file name it should look for in the root. The location block says to try send stuff as a file first, as a directory second return a 404 not found code if it cannot resolve anything. The last block is the block for wordpress. It reads very similar to the previous block. The only thing of note, the last location block reads that for any file ending in .php, include some dependencies.

```
server {
    listen 80 default_server;

    server_name _;
    return 301 https://www.jameschellew.xyz;
}
server {
    server_name jameschellew.xyz www.jameschellew.xyz;
```

```

root /var/www/html/sites/jameschellew.xyz/main;

index index.html index.htm index.nginx-debian.html;

location / {
    # First attempt to serve request as file, then
    # as directory, then fall back to displaying a 404.
    try_files $uri $uri/ =404;
}
}
server {
    listen 80;
    root /var/www/html/sites/jameschellew.xyz/wordpress;
    index index.php index.html index.htm;

    server_name wordpress.jameschellew.xyz;
    client_max_body_size 500M;
    location / {
        try_files $uri $uri/ /index.php?$args;
    }
    location = /favicon.ico {
        log_not_found off;
        access_log off;
    }
    location ~* \.(js|css|png|jpg|jpeg|gif|ico)$ {
        expires max;
        log_not_found off;
    }
    location = /robots.txt {
        allow all;
        log_not_found off;
        access_log off;
    }
    location ~ \.php$ {
        include snippets/fastcgi-php.conf;
        fastcgi_pass unix:/var/run/php/php8.1-fpm.sock;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        include fastcgi_params;
    }
}

```

Setting Up LEMP Stack and WordPress

Installing WordPress – steps primarily inspired from

<https://www.ionos.com/digitalguide/hosting/blogs/wordpress-nginx/>

1. Always start by making sure the system is up to date:

```

sudo apt update
sudo apt upgrade

```

2. Ensure Nginx is installed. If Nginx is already installed, skip to step 4.

```

sudo apt install nginx

```

3. Check that nginx installed successfully

```

sudo service nginx status

```

4. Next, will install MySQL server as WordPress will require this service.

```

sudo apt install mysql-server

```


5. Check that MySQL server was installed successfully.

```
sudo systemctl status mysql
```

6. Now MySQL is installed. Next we need to configure MySQL. The first time we log in, we are going to log in as the root user. Do this command as follows to log into the MySQL server.

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

7. First we are going to create a database for WordPress.

```
CREATE DATABASE WordPress CHARACTER SET utf8mb4 COLLATE  
utf8mb4_general_ci;
```

8. Next, we are going to set up a username and password.

```
CREATE USER [user name here] IDENTIFIED BY '[password here]';
```

9. And finally we are going to allow the user we create to access and modify the database.

```
GRANT ALL PRIVILEGES ON WordPress.* TO [user name from before];
```

10. Now you can exit MySQL

```
EXIT;
```

11. Next up we are going to download some dependencies that WordPress will need to use – PHP. When installing this, pay attention to the version it downloads as it will be necessary for the next step.

```
sudo apt install php-fpm
```

12. Check that PHP was installed properly. At the time of writing, the version installed was php8.1.

```
sudo systemctl status php8.1-fpm
```

13. One last thing before we install WordPress, we are going to install an extension that will allow PHP to work with MySQL. With this we have completed a “LEMP” stack server (Linux, nginx, MySQL, Php).

```
sudo apt-get install php-mysql
```

14. If you have synced the website from GitHub, skip to step 18.

Make a directory; you may call it whatever you like, but it is convention to call it the domain name of the website. This is where we will be installing the WordPress so place it inside the directory as follows:

```
sudo mkdir -p /var/www/html/[directory name here]
```

15. Cd into that directory

```
cd /var/www/html/[directory name here]
```

16. Here, we’re going to download the latest version of WordPress in the form of a compressed file. If you are not doing a from scratch installation, continue with these steps, otherwise clone the git repository following the steps in that guide.

```
sudo wget https://wordpress.org/latest.tar.gz
```

17. Uncompress the file

```
sudo tar -xvzf latest.tar.gz
```

18. Change the ownership of the file and all subdirectories to www-data so that nginx can make direct changes to the directories.

```
sudo chown -R www-data: /var/www/html/[your directory]/wordpress
```

19. Change the directory into the WordPress directory.

20. Copy the sample file to the wp-config.php file. If you have synced from GitHub, do not worry about copying this again and continue to next step.

```
sudo cp wp-config-sample.php wp-config.php
```

21. and nano open the file you copied.

```
sudo nano wp-config.php
```

22. This file is used to configure WordPress to work with the database. The database name is what you set earlier, the same goes for the username and the password. The host can be left as is.

```
/** The name of the database for WordPress */
define( 'DB_NAME', '[Database Name]' );
/** Database username */
define( 'DB_USER', '[user you created]' );
/** Database password */
define( 'DB_PASSWORD', '[password you set]' );
/** Database hostname */
define( 'DB_HOST', 'localhost' );
```

23. Now, we are going to change the Nginx config file to access the WordPress route to the WordPress directory when a certain domain name is accessed. This domain name should be set up as an address record pointing to the public IP of our machine on your domain registry, so do that now if you have not. Nano open this file:

```
sudo nano /etc/nginx/sites-available/default
```

24. Paste this block into the file, modifying the highlighted bits. Be sure to check your php version

```
server {
    listen 80;
    root /var/www/html/[directory you made]/wordpress;
    index index.php index.html index.htm;
    server_name [your domain name];
    client_max_body_size 500M;
    location / {
        try_files $uri $uri/ /index.php?$args;
    }
    location = /favicon.ico {
        log_not_found off;
        access_log off;
    }
    location ~* \.(js|css|png|jpg|jpeg|gif|ico)$ {
        expires max;
        log_not_found off;
    }
    location = /robots.txt {
        allow all;
```

```

        log_not_found off;
        access_log off;
    }
    location ~ \.php$ {
        include snippets/fastcgi-php.conf;
        fastcgi_pass unix:/var/run/php/php8.1-fpm.sock;
        fastcgi_param SCRIPT_FILENAME
$document_root$fastcgi_script_name;
        include fastcgi_params;
    }
}

```

25. Test this configuration with:

```
sudo nginx -t
```

26. Then restart Nginx.

```
sudo systemctl restart nginx
```

27. Put the domain name into the server browser, and the WordPress admin screen should now show. Set up your account, and WordPress is now set up.

Installing Certbot using Snapd

Inspired from <https://www.inmotionhosting.com/support/website/ssl/lets-encrypt-ssl-ubuntu-with-certbot/> and <https://www.digitalocean.com/community/tutorials/how-to-secure-nginx-with-let-s-encrypt-on-ubuntu-20-04>

Before setting up and running Certbot, you should make address records for your domain and configure Nginx to know how to resolve requests to those domain names.

Certbot Install Method 1:

1. Install snapd (snap daemon)

```
sudo apt install snapd
```

2. Make sure snapd is up to date:

```
sudo snap install core
sudo snap refresh core
```

3. Install Certbot

```
sudo snap install --classic certbot
```

4. Make a symbolic link to ensure that Certbot runs.

```
sudo ln -s /snap/bin/certbot /usr/bin/certbot
```

5. Ensure current nginx configuration works

```
6. sudo nginx -t
```

7.

8. Now run this command to generate SSL certificates for all of the domain names specified in the nginx config file. If you haven't configured this file, do so first

```
sudo certbot --nginx
```

Certbot Install Method 2:

1. Install Certbot and the plugin for nginx.

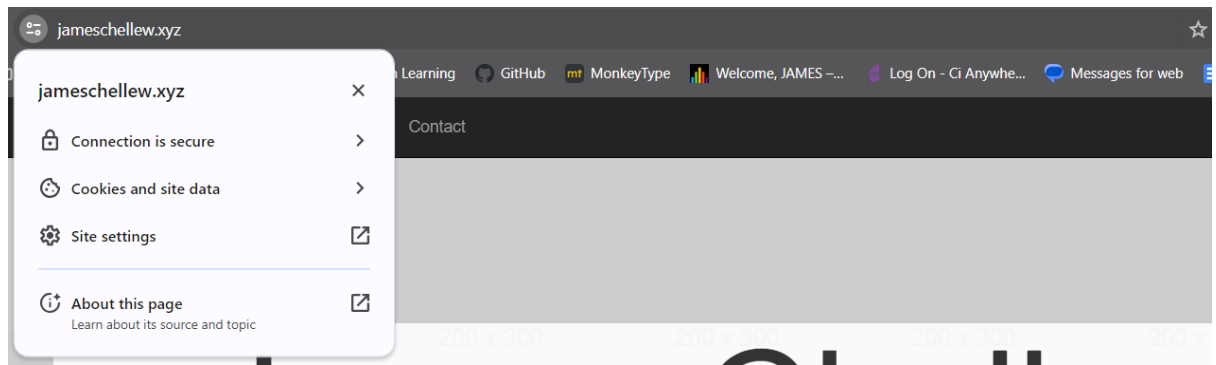
```
sudo apt install certbot python3-certbot-nginx
```

2. Run Certbot to generate SSL certificates for all domains in nginx configuration file

```
sudo certbot --nginx
```

After Running Certbot

1. Choose that yes, you want all of the domain names to have an SSL certificate set up.
2. If it is your first time running Certbot, it will ask you for an email to send renewal and security notices. Enter your email so that you are notified of this. It will ask you to agree to terms of service. It will ask if you want to receive further emails, which I recommend you decline. Lastly, we do not want to make further changes to the webserver, so we will select option 1.
3. Verify that the SSL certificate is working by visiting the site and checking that it has a valid certificate. On Chrome, it should look something like this:



4. If this does not work.

Additional note:

Certbot will automatically check your SSL certificates and manage your nginx configuration files. It will automatically renew the certificates when they are within 30 days of expiration. To manually update these certificates, run the command on step 5 again and choose the renew option when prompted.