Maria Clare Lusardi

pawprint: mclntg

CS/IT 3530 – Unix Operating Systems

Implementing HTTPS on a RHEL Web Server

**Introduction**

This report outlines the importance of using HTTPS (Hypertext Transfer Protocol Secure) and SSL certificates for securing websites and demonstrates a method for achieving this on an Elastic Cloud Computing Instance (ec2) from Amazon Web Services (AWS) running a Red Hat Enterprise Linux 8 (RHEL 8) operating system. I selected this topic because this past semester I have been studying web development, which has made me more aware of the security challenges for websites. If I ever run a website in the future, HTTPS is a security measure I will definitely need to be able to implement.

Many companies and organizations right now are pushing for HTTPS to become the standard. Most web browsers today display some kind of indication, whether by a ‘lock’ symbol or a green address bar, which shows that the website has an SSL certificate and is operating with HTTPS. There has also been a push to educate the public on the dangers of inputting sensitive information to websites which are not using HTTPS. Since regular HTTP does not encrypt data in transit or verify the identity of the website owner, these websites are susceptible to man-in-the middle attacks. In these kinds of attacks, an attacker will insert themselves between the client and server to intercept data passing between the two. Using HTTPS encrypts this data, so that an attacker cannot read transferred information, and it verifies the owner of the website, so an attacker can’t trick the user to connecting with an illegitimate server. Because of this, HTTPS can improve customer confidence when purchasing things on a website, makes submitting data through forms much more secure, keep membership info safe, and can protect site information from being tampered with or injected with advertisements. In addition, Google will prioritize websites in search results which are using HTTPS. Typically, websites have had to pay a certificate authority, or CA, to receive an SSL certificate, but with the start of Let’s Encrypt in 2015, certificates are now available for free. In short, there is no reason to leave a website unsecured with HTTP.

**Research**

In my demonstration, I will first cover how to set up a RHEL 8 ec2 instance as a web server and demonstrate that it can display a basic HTML page (pulled from course content). Then I will set up SSL/TLS encryption and a self-signed SSL certificate to allow a user to navigate to the site via HTTPS. Lastly, I will discuss how to obtain a certificate from Let’s Encrypt, an official certificate authority, and further steps that can be taken to improve website security. Before continuing, here are some key terms to know:

* HTTP – Hypertext Transfer Protocol, a set of rules governing how information can be exchanged between a client, typically via a web-browser, and a web server.
* SSL – Secure Sockets Layer, technology that encrypts data sent between two systems, such as a web server and a client browser.
* TLS – Transport Layer security, a more secure version of SSL, though the term SSL and TLS are often used interchangeably.
* SSL Certificates – used to certify that the website is using SSL/TLS encryption and that the website owner is legitimate. Though commonly sold to websites by certificate authorities, this report demonstrates how to obtain a free certificate through Let’s Encrypt.
* HTTPS – Hyper Text Transfer Protocol Secure, this appears when a website is using an SSL certificate. It can also be seen via the lock symbol or green address bar on some browsers.
* CA – Certificate authority, an organization that issues SSL or TLS certificates. In addition to encrypting data, CAs will verify the website owner’s identity, which makes it more secure.
* Domain Validation SSL certificates: the most basic level, only requires control of the domain name to be established. Generating this kind of certificate is demonstrated as it is the only option for individuals.
* ACME – Automatic Certificate Management Environment, a software which automatically reviews a web server, certifies it, and renews certificates. Let’s Encrypt’s recommended ACME is Certbot, so that is what is demonstrated in this report.

Process:

When a website receives an SSL certificate from a CA, it is installed on the server. The encryption process is triggered when a web-browser connects to the website. This is called the SSL handshake, which occurs after the TCP handshake. After the SSL handshake, the server sends the certificate to the client to specify what version SSL/TLS should be used for encryption. There are usually sets of encryption methods of varying strengths supported. These are called Cipher suites. The client computer will check the certificate validity and select the highest possible encryption that is supported by both parties. Lastly, the SSL or TLS protocol authenticates the messages sent from the client with message authentication codes (MAC). While encrypting, SSL/TLS do not affect the functions of other application layers.

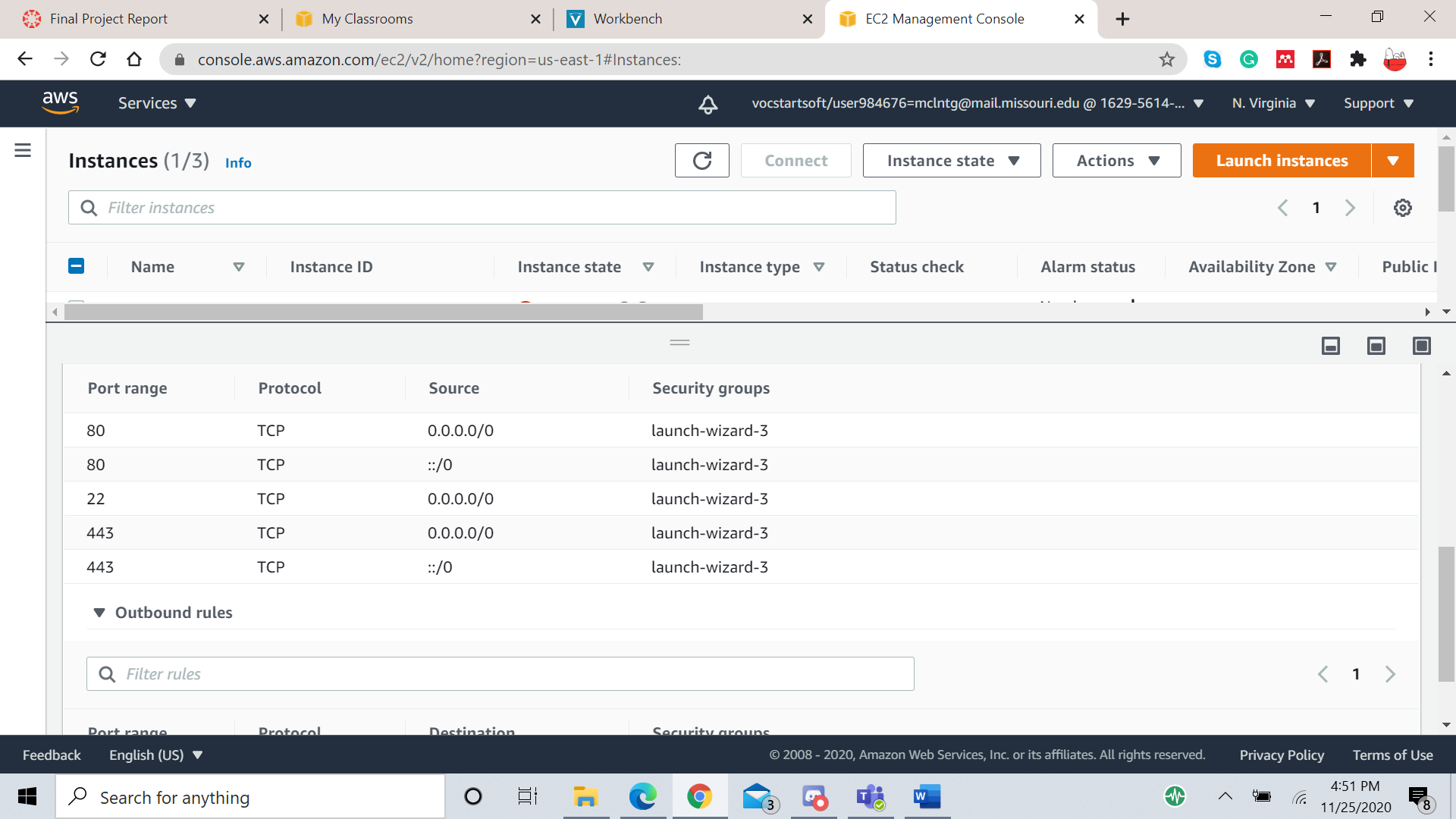
Source: <https://www.websecurity.digicert.com/security-topics/what-is-ssl-tls-https>

<https://www.ssl.com/faqs/faq-what-is-ssl/>

**Application**

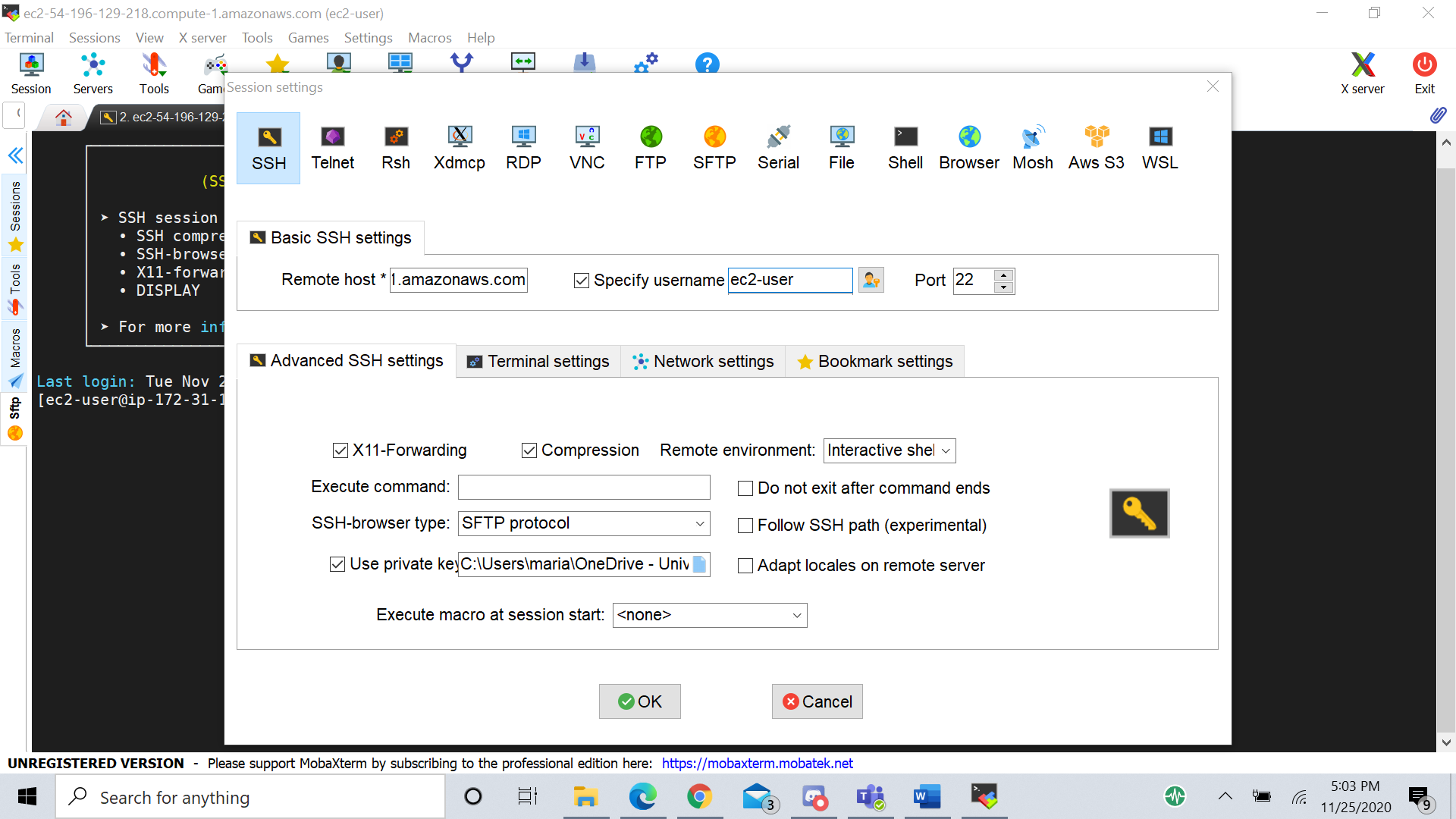
1. Create a RHEL 8 ec2 instance in AWS

Click on ‘Launch instance’ and input the settings. For the security groups, port 22 for SSH, port 80 for HTTP, and port 443 for HTTPS must be open for any source. See below for the correct security group settings. When creating, generate a key pair for the instance and save the private key file. After creating the instance start it.

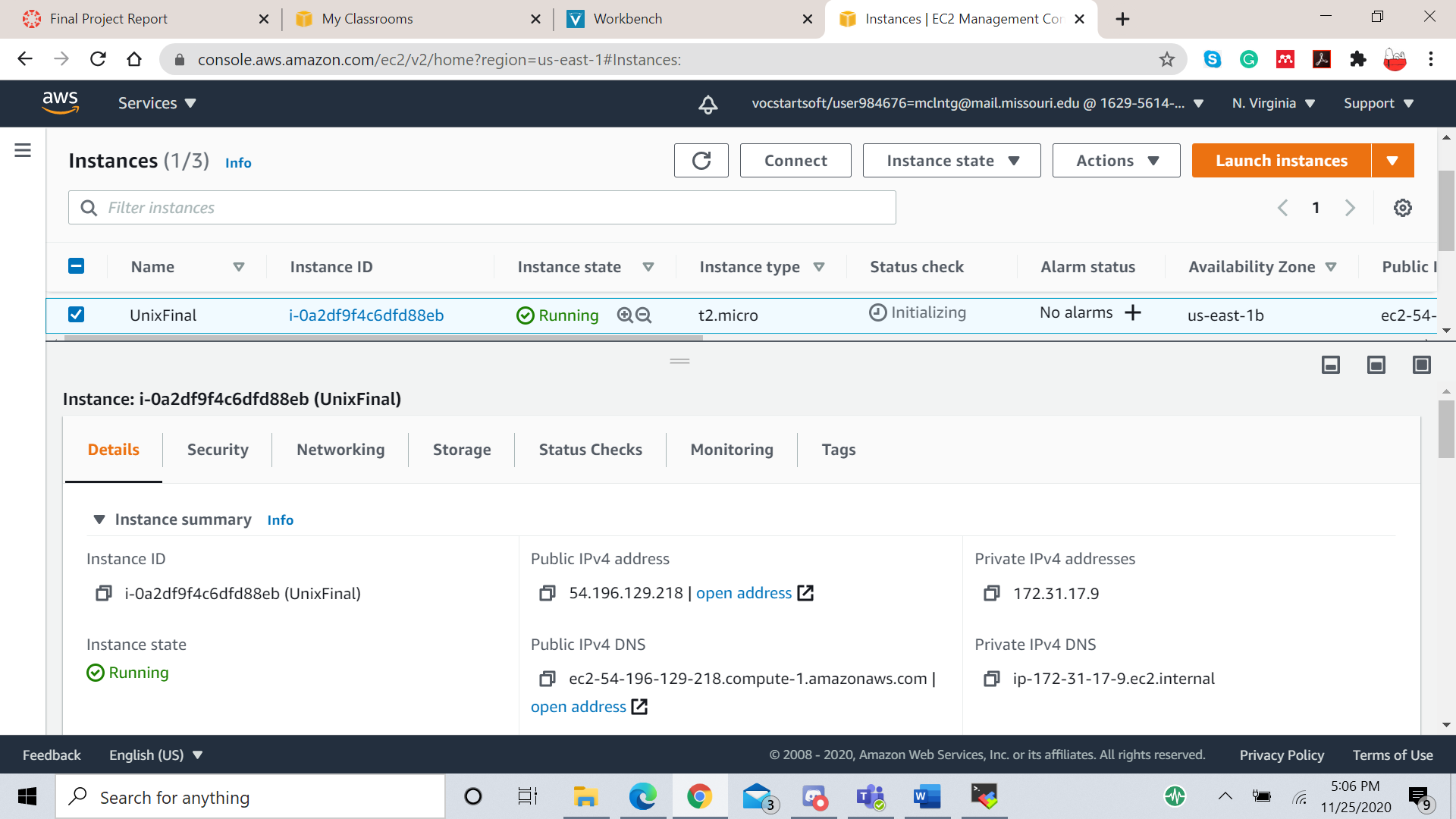


1. SSH into the instance

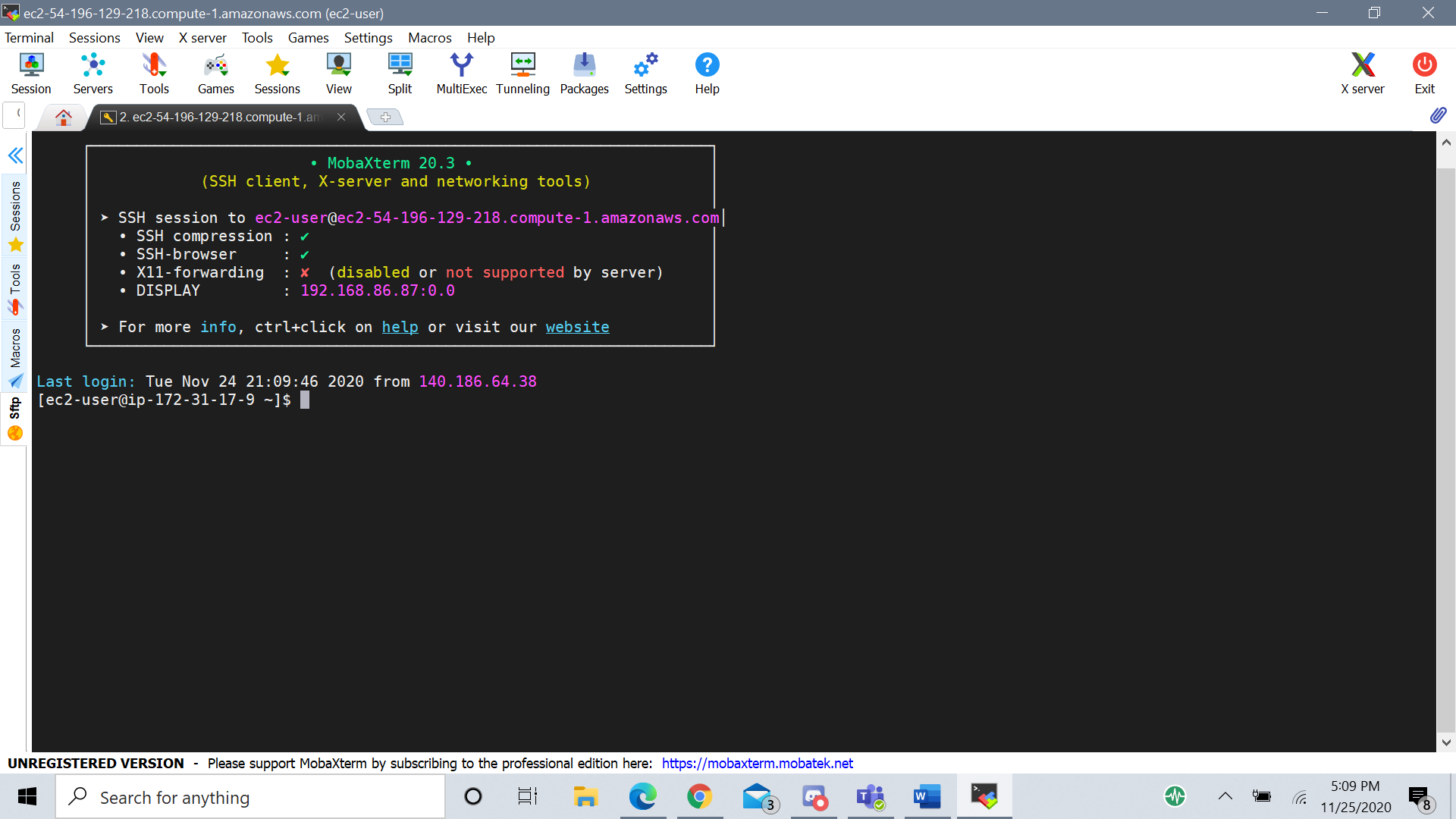
I used MobaXterm to connect to the instance via SSH. To do this, click on ‘Session’ to start a new session and select ‘SSH’. Input the public IPv4 DNS for the Remote Host and ‘ec2-user’ for the username. Click on ‘Advanced SSH settings’ and select ‘Use private key’. Select the private key file which was downloaded in step one. Click on ‘OK’ to log on to the instance.



Here is where to find the public DNS for the instance.



If you have successfully logged into the instance, then you should see a screen like this.



1. Install apache

Apache is a service that allows a server to host webpages. To download apache, enter the command

$ sudo dnf install httpd

Once this service is installed, it needs to be enabled and started. Enter the commands

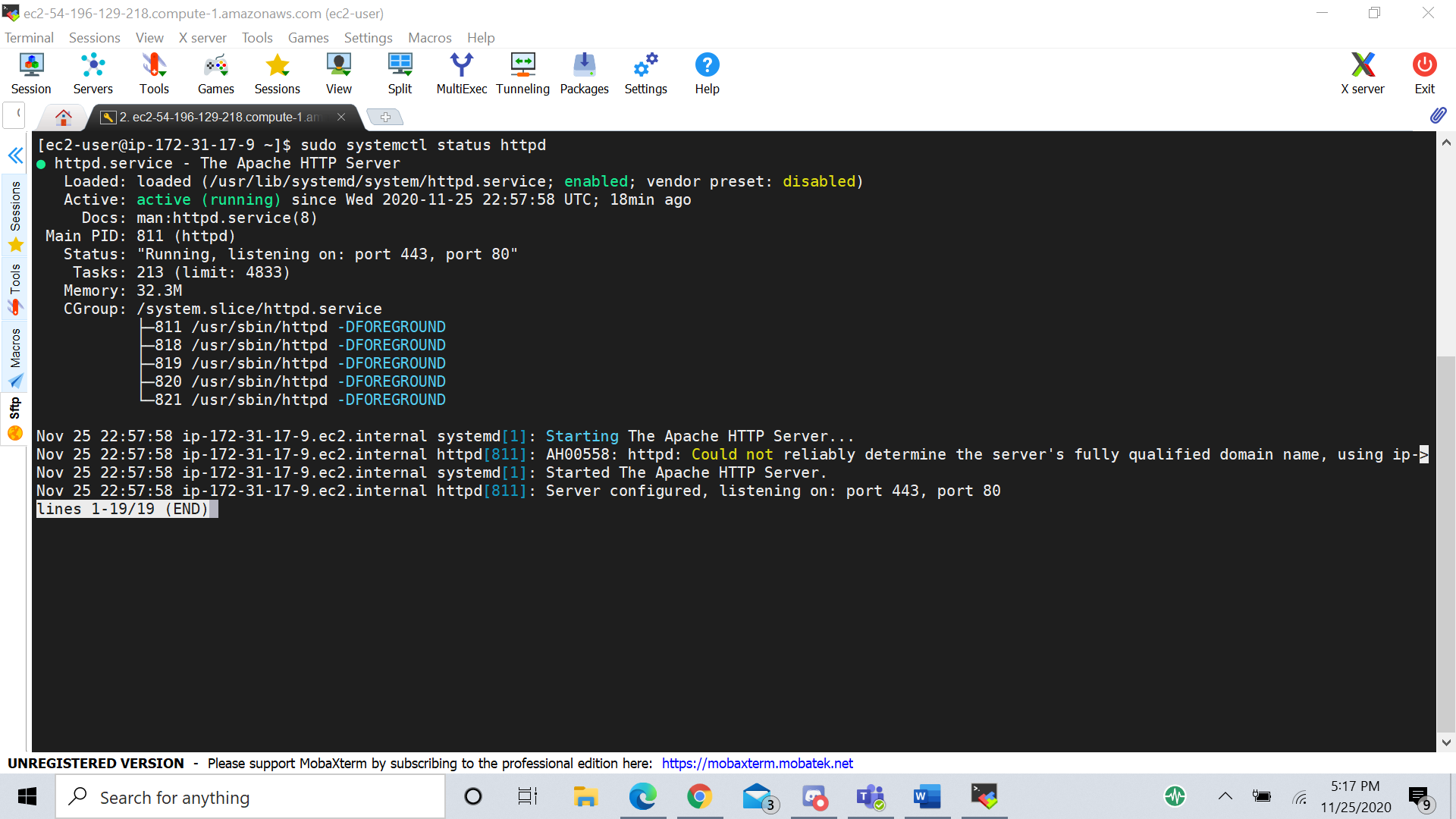
$ sudo systemctl enable httpd

$ sudo systemctl start httpd

To verify that these commands were successful, you can enter

$ sudo systemctl status httpd

If the service says ‘enabled’ and ‘running’ then it worked. See the output below.



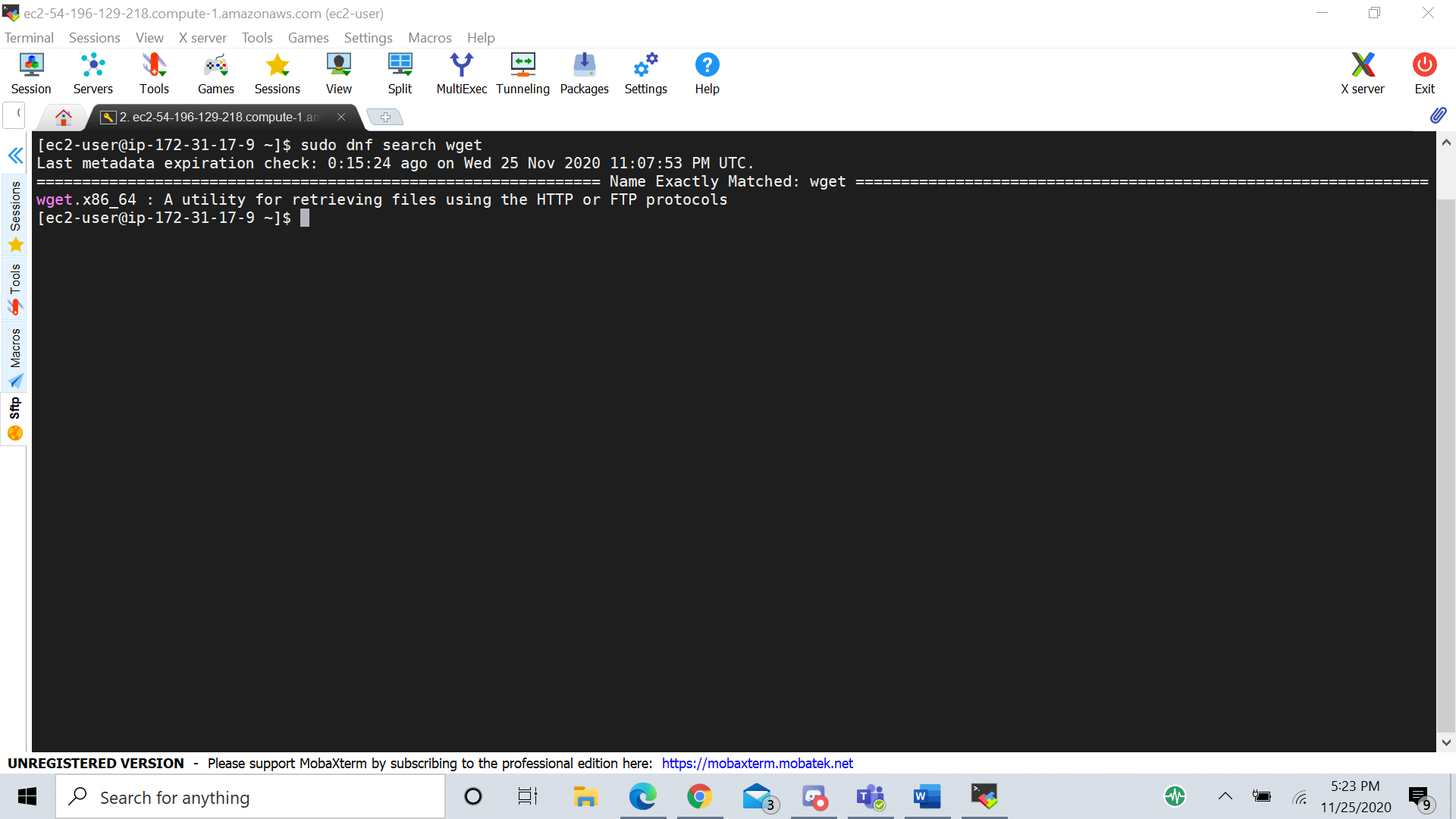
You can also see that it is listening on port 443 and port 80, the HTTPS and HTTP ports respectively.

1. Create index file and navigate to it via web-browser

To test if the web server works, we need to host a webpage. For this project, I downloaded the HTML file used in labs 8 and 12. The wget command does not come pre-installed on RHEL 8. First, find the package by typing

$ sudo dnf search wget

This searches for a package called wget. When it finds the package, it will show



Download the wget package with the command

$ sudo dnf install wget

The wget command will enable the index file to be downloaded directly to the instance. To do this, type

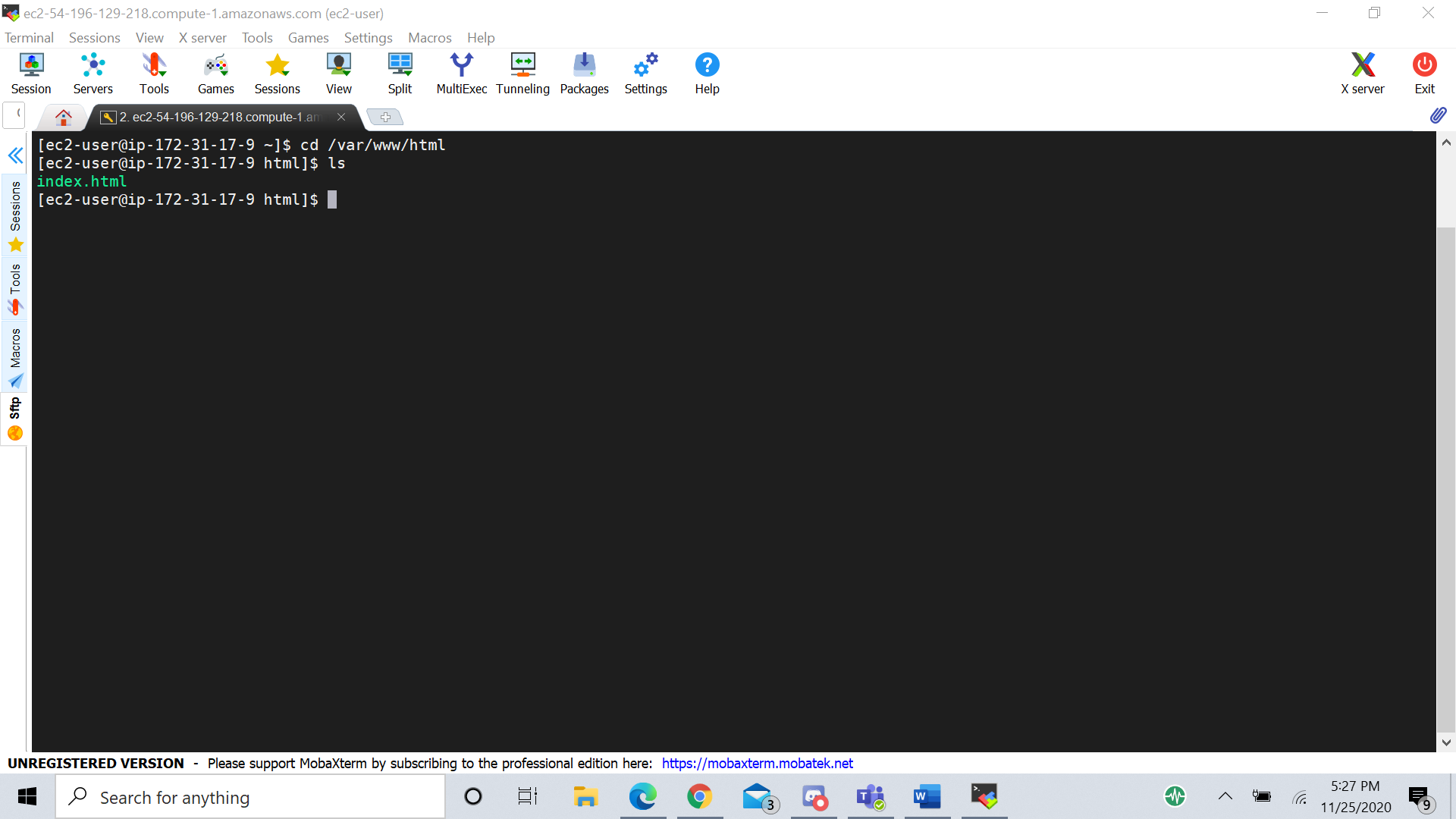
$ wget <http://34.238.159.47/UnixOperatingSystem/index>

The file will be called ‘index’. Change the name to ‘index.html’ and move the file into the /var/www/html directory.

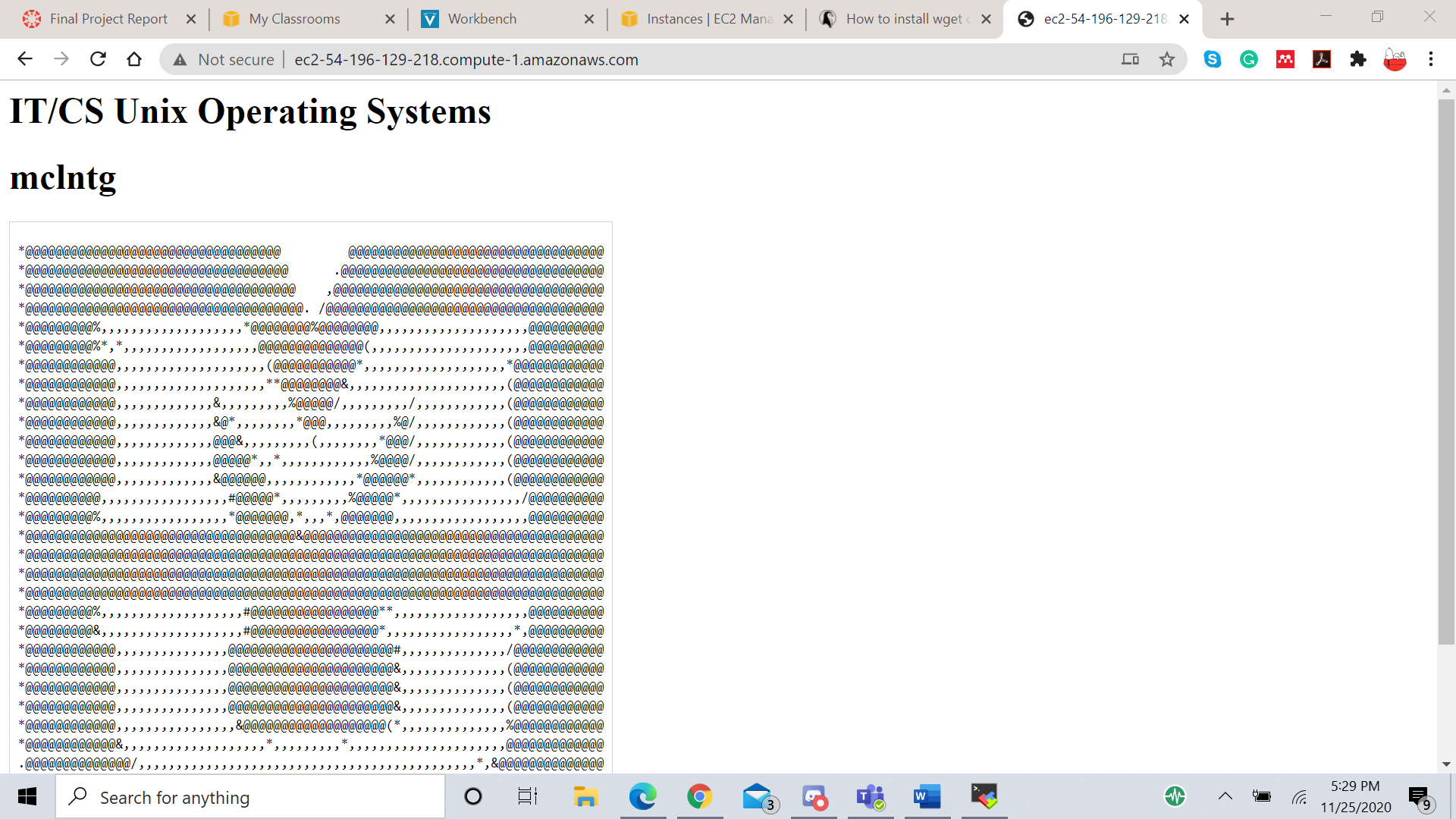
$ mv index index.html

$ mv index.html /var/www/html

If you change to the /var/www/html directory, you should see the file listed.



This page should now be accessible to a web browser. Go to a web browser and paste the instance’s public IPv4 DNS into the address bar to see the file.



If you instead get a page saying “Forbidden” and that you are not allowed to access this resource, then there is an issue with the permissions. Run the command

$ sudo restorecon -R /var/www

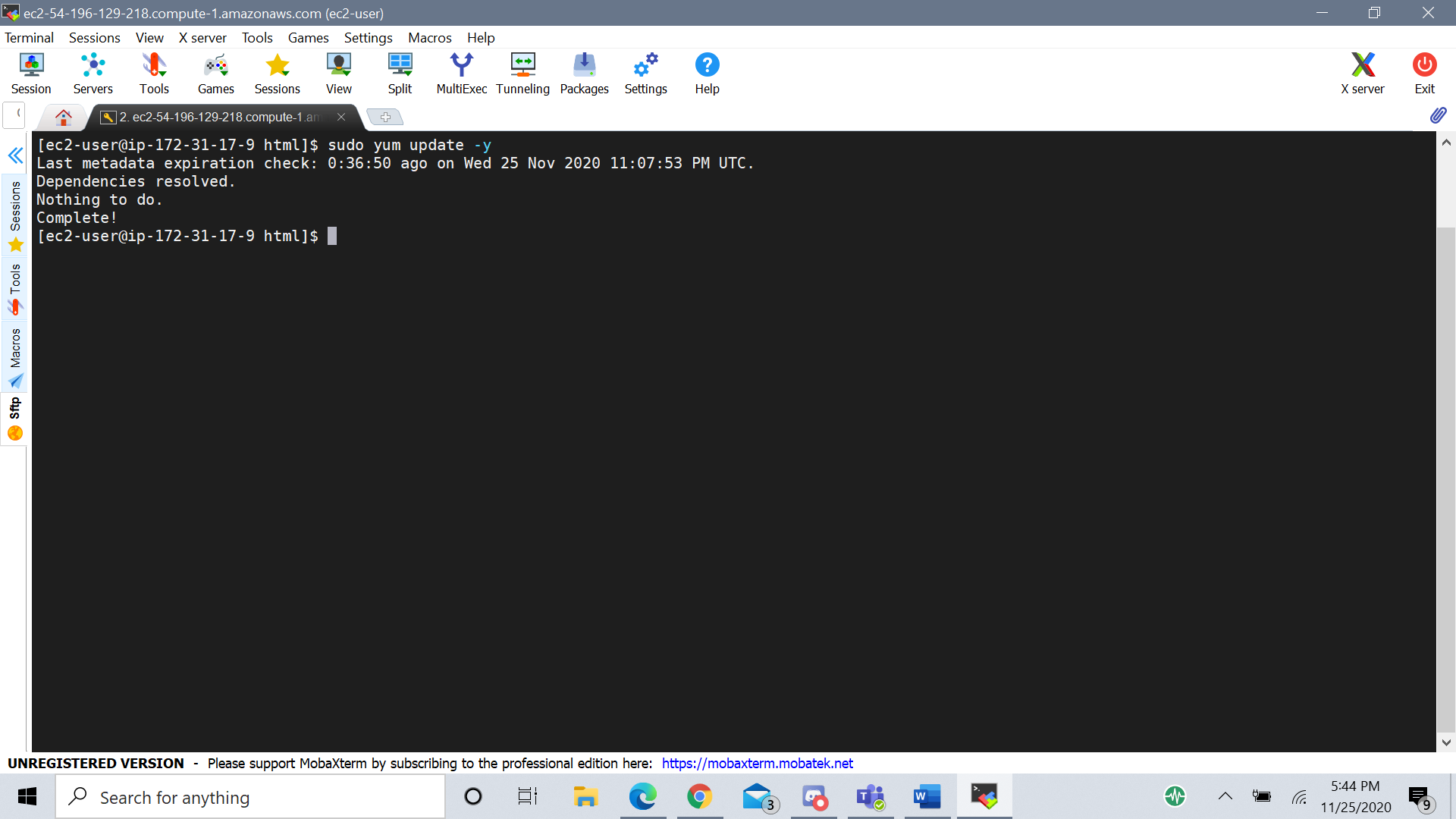
This will reset the permissions for all files within the www/ directory to the correct values and enable the browser to access the index.html file.

Notice that there is no ‘lock’ icon in the address bar. Instead it says ‘Not secure’. That is because the website is currently using HTTP and the data between the client and server is not encrypted.

1. Enable SSL/TLS on the server

TLS is the service that will encrypt the data. First, ensure all packages on the server are up to date by typing

$ sudo yum update -y



Next, install the mod\_ssl package from httpd with the command

$ sudo dnf install mod\_ssl

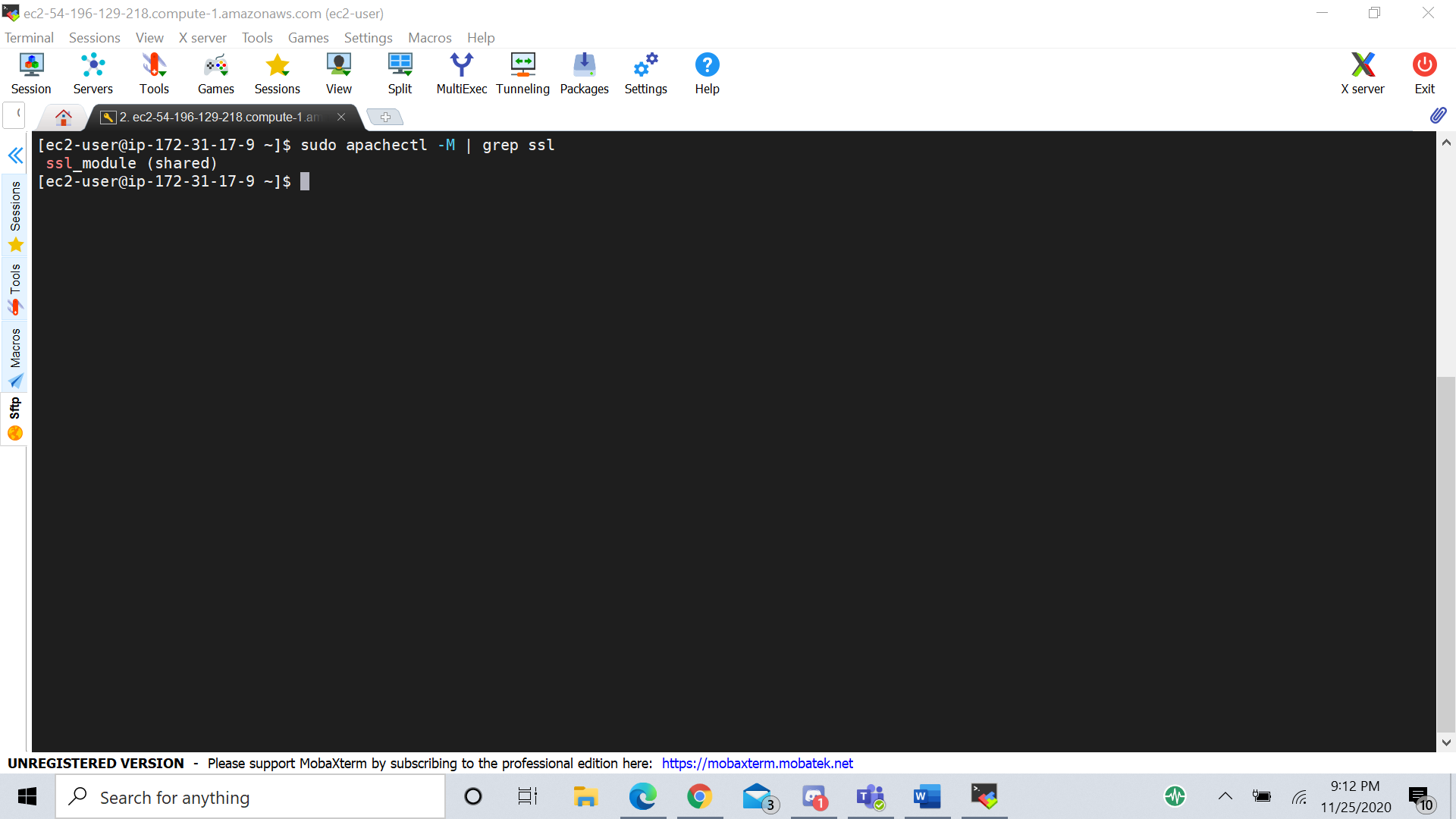
The httpd module must be restarted to enable mod\_ssl. Restart it with the command

$ sudo systemctl restart httpd

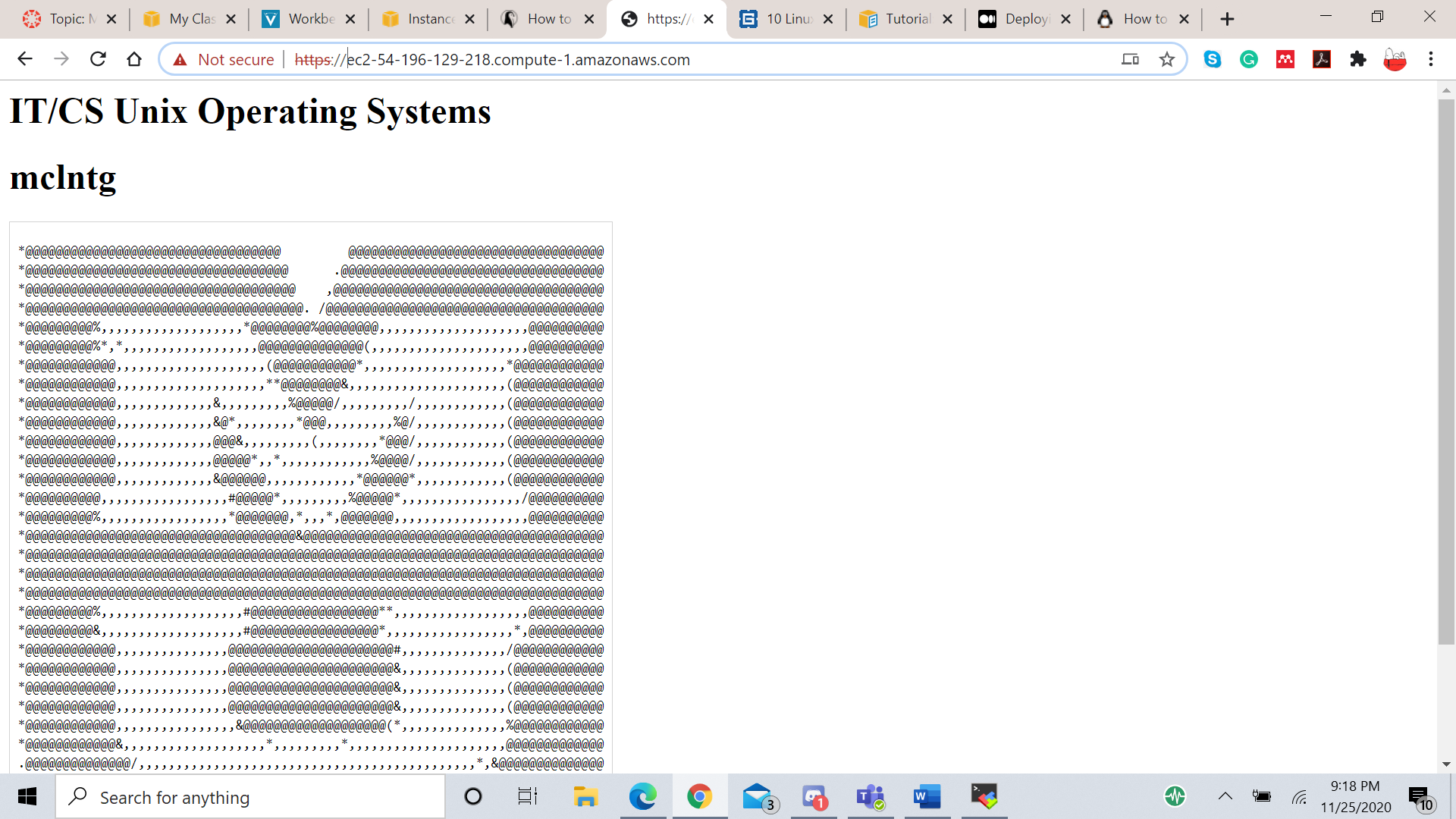
After restarting httpd, check that it worked with the command

$ sudo apachectl -M | grep ssl

This command searches the apache server control interface for the letters ‘ssl’. If SSL has been enabled, then the output from this command will be ‘ssl\_module (shared)’



Since port 443 is already opened in the security groups for this instance, the webpage can now be accessed by going to https.



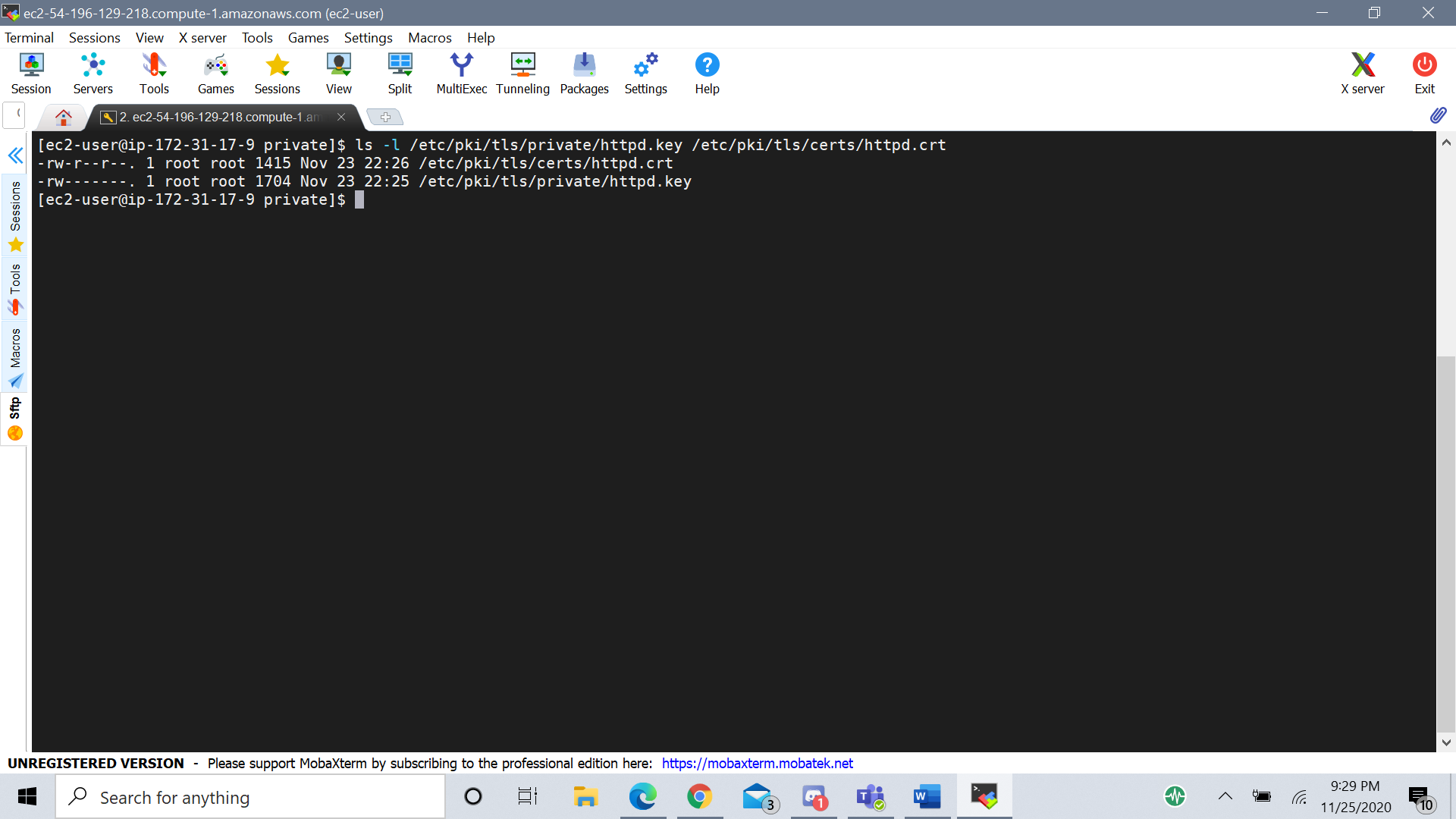
1. Generate self-signed certificate

Before requesting a certificate from a CA, a self-signed certificate can be created. These are not secure and should only be used for testing. To generate one, enter the following command.

$ sudo openssl req -newkey rsa:2048 -nodes -keyout /etc/pki/tls/private/httpd.key -x509 -days 365 -out /etc/pki/tls/certs/httpd.crt

After running this command, an interactive menu will appear requiring information for the certificate. Not all of the fields are required.

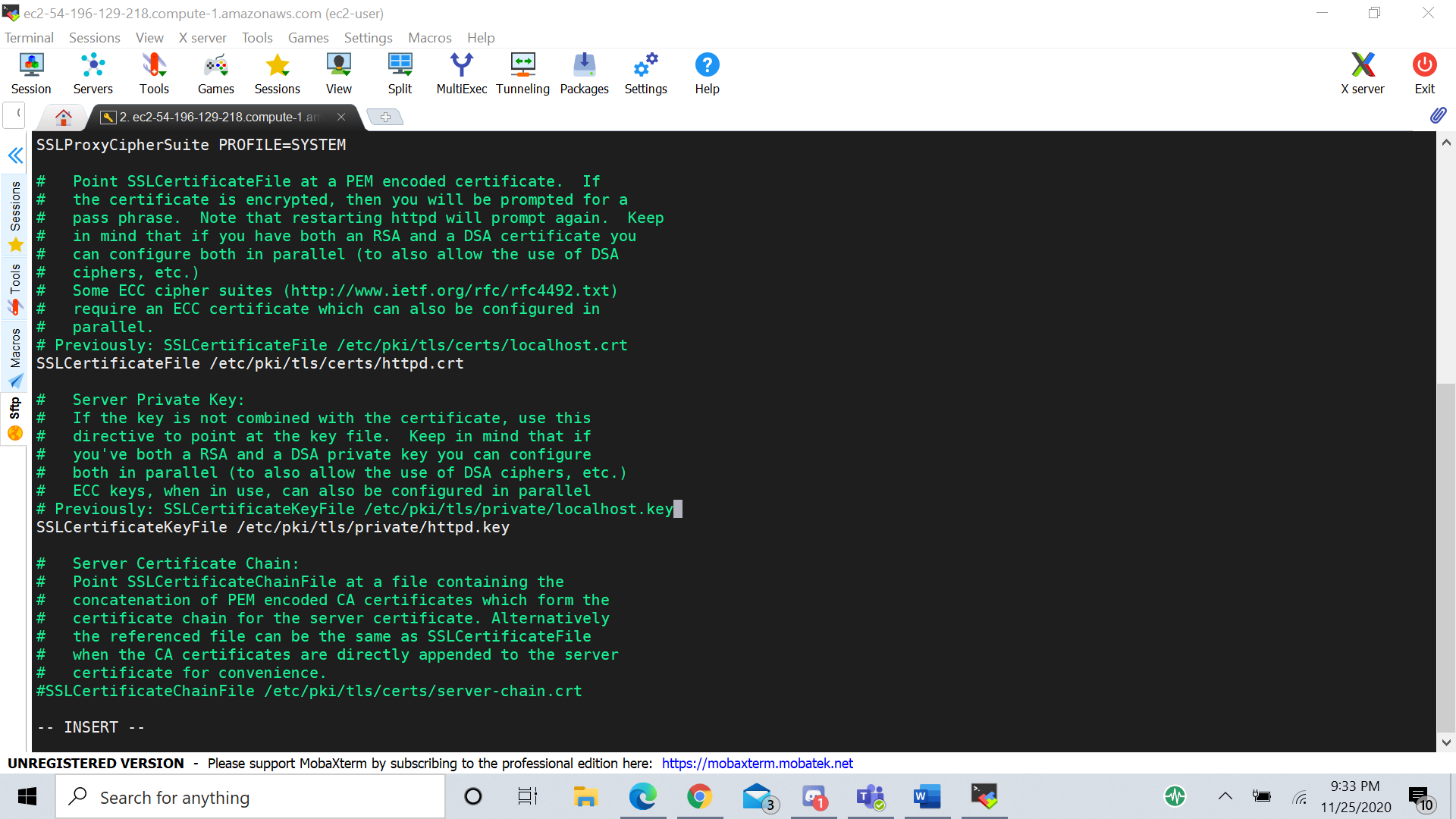
This command generates two files, /etc/pki/tls/private/httpd.key and /etc/pki/tls/certs/httpd.crt. The httpd.key file holds a new private key encoded with rsa 2048-bit encryption which can be used to access the certificate. The httpd.crt file holds the self-signed certificate. This certificate is valid for 365 days.



To incorporate this certificate into the webserver, the /etc/httpd/conf.d/ssl.conf file has to be edited. Open the file with the command

$ sudo vi /etc/httpd/conf.d/ssl.conf

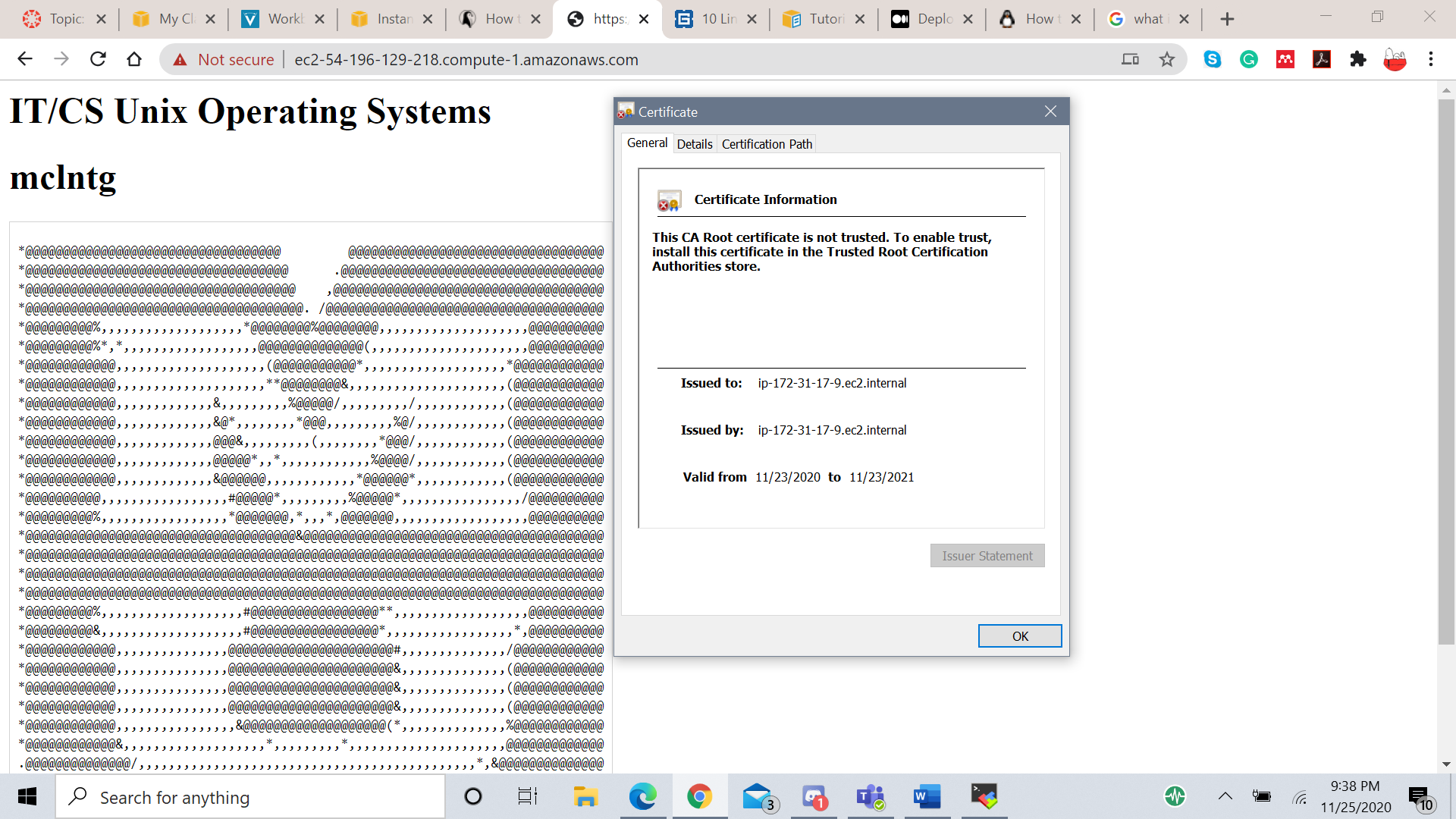
The lines indicated below show that the SSLCertificateFile and SSLCertificateKeyFile must be changed to the files created in the above command. Before, they were set to the localhost.crt and localhost.key files.



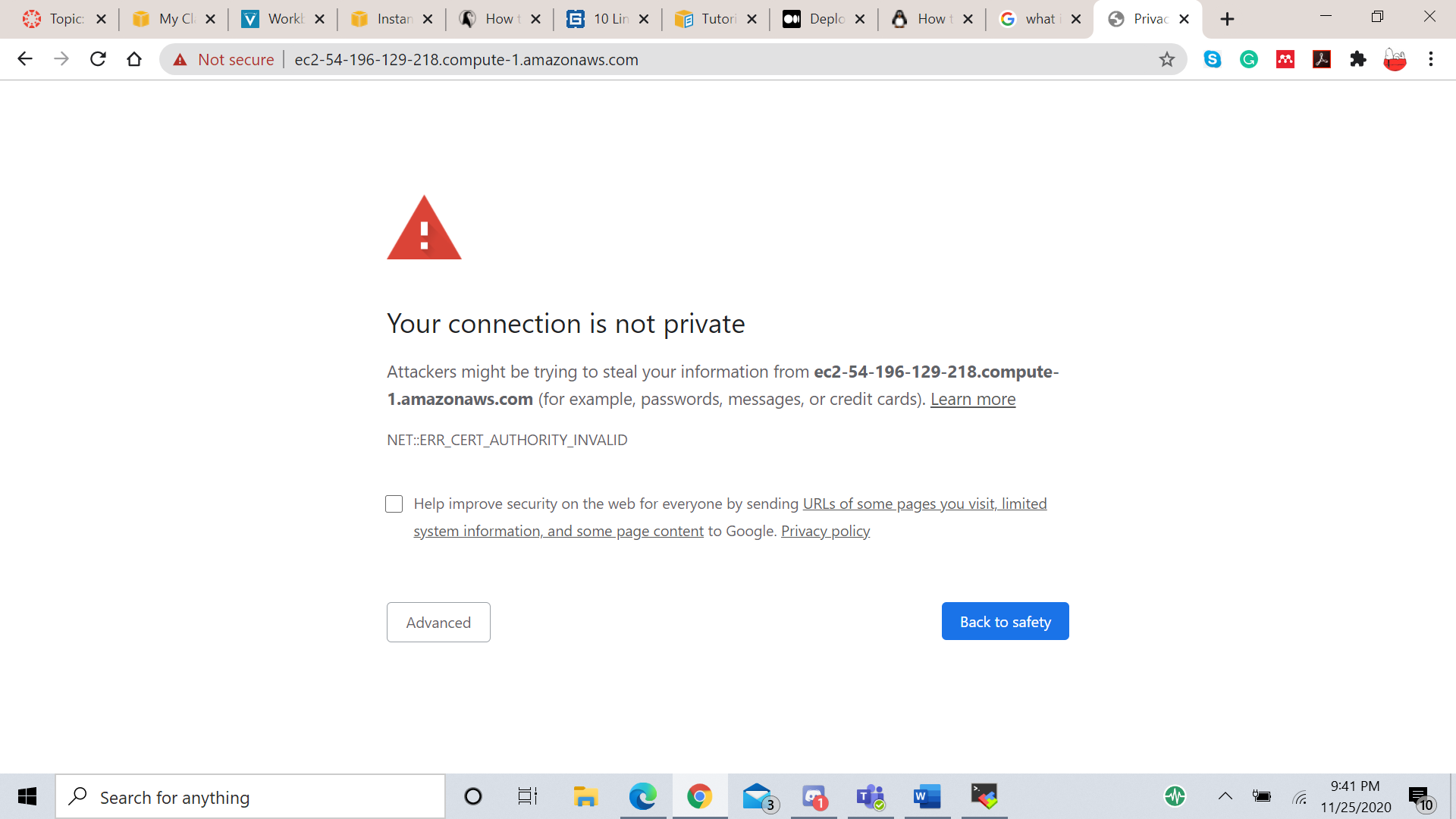
For these changes to take effect, the httpd service must be reloaded.

$ sudo systemctl reload httpd

Now when navigating to the website, the certificate will be visible.



This certificate was issued to and issued by the same address, making it a self-signed certificate. This is not a true verification of identity as anyone can generate their own self-signed certificate. Most web-browsers will issue security warnings and try to prevent users from reaching websites that have self-signed certificates. Clearly, this is not acceptable for a professional website.

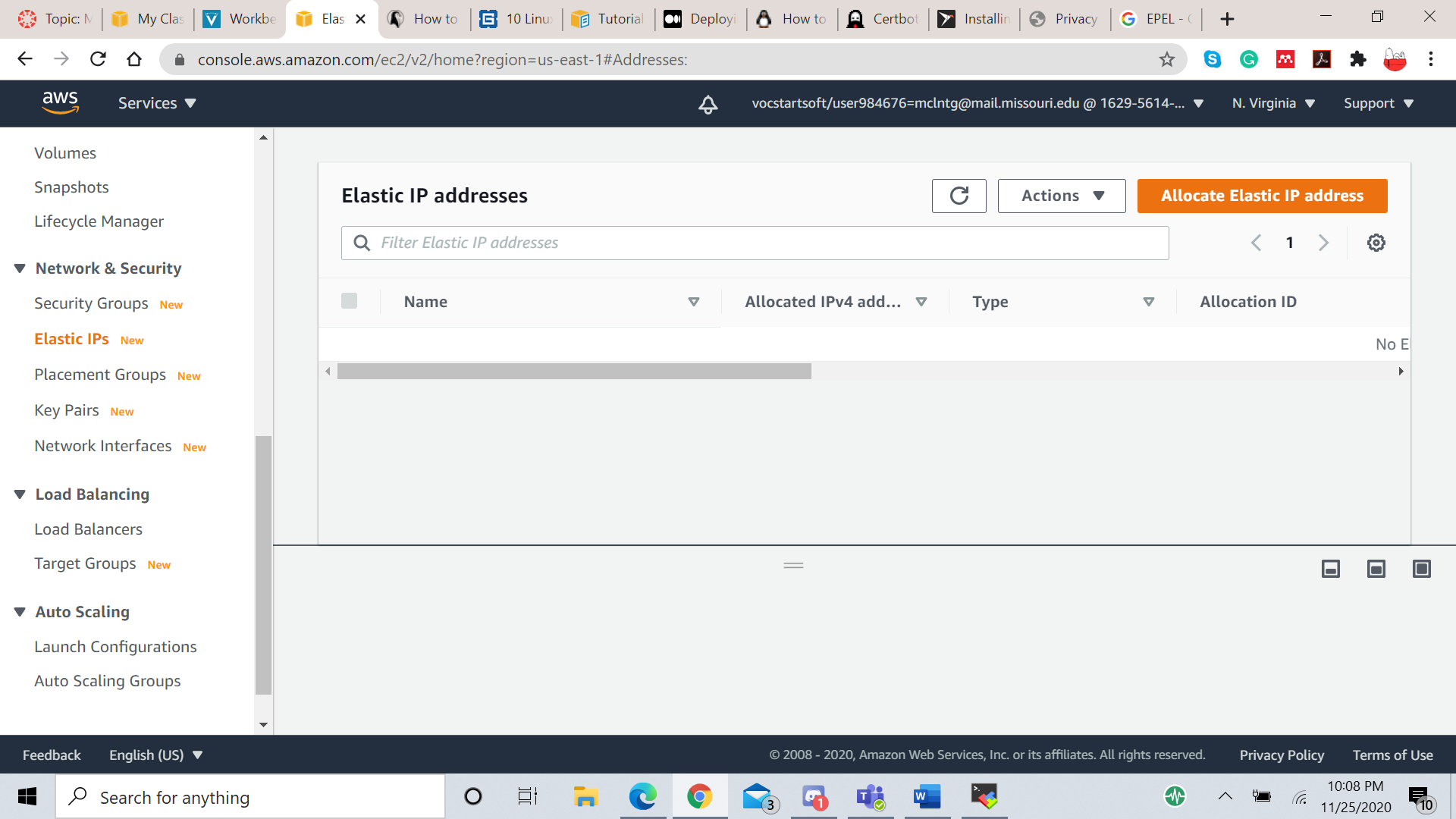


Despite this, the data is still being encrypted. The identity just hasn’t been verified.

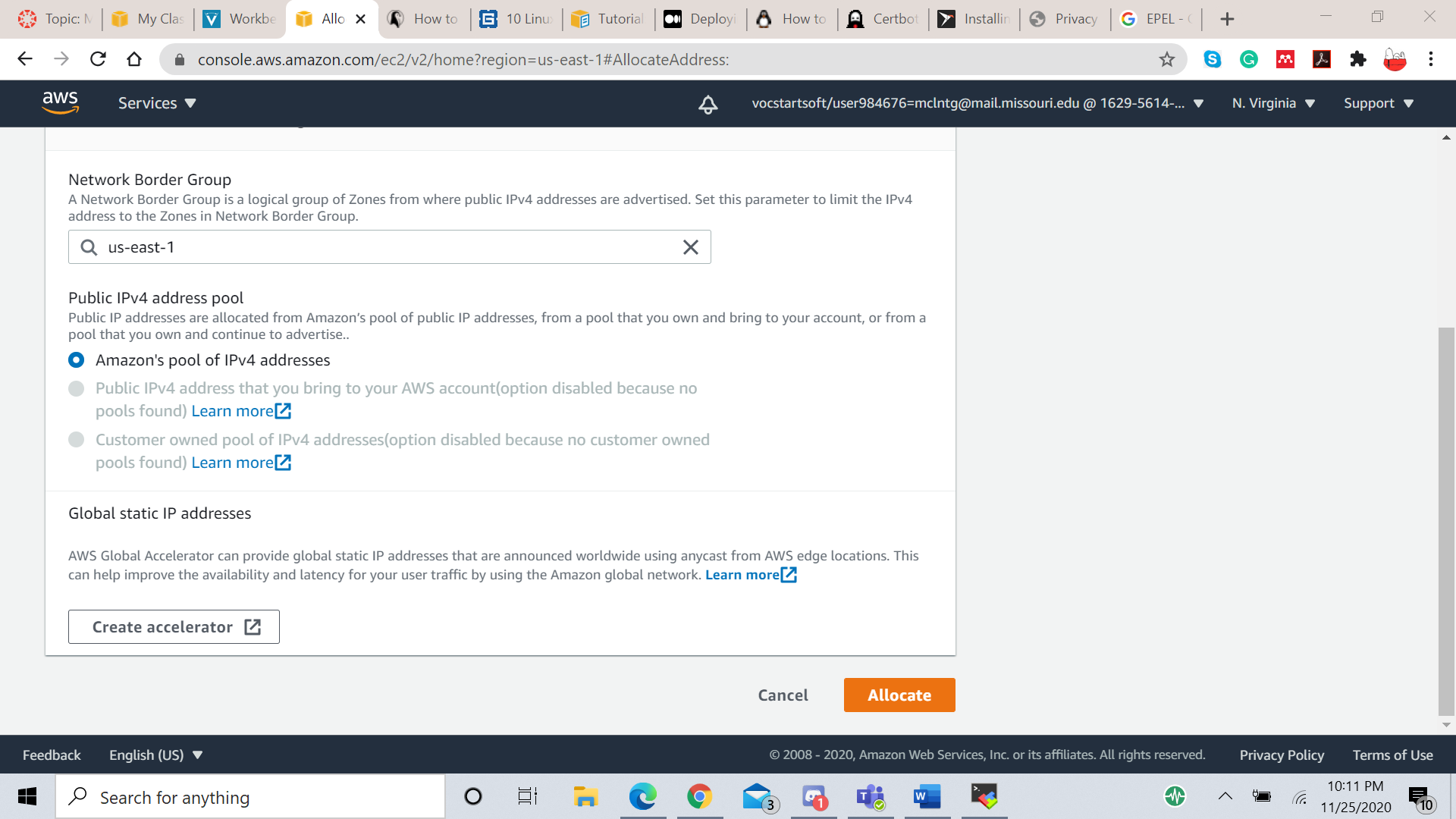
To continue, you must own a domain name. Certbot will not certify the public DNS provided for this instance by AWS because it is not permeant. An instance’s public IP and DNS only exist while the instance is running. Once the instance has been stopped, that IP and DNS are disassociated from the instance and put back into the pool of available IP and DNS addresses. When the instance is restarted, it is assigned a new IP and DNS from the available pool. For an ACME to certify a webpage the owner must prove that they own the domain name and that it is permanent. I cannot demonstrate this part, as I do not own a domain name. However, I can demonstrate the steps leading up to certification.

1. Connecting the domain name to the server

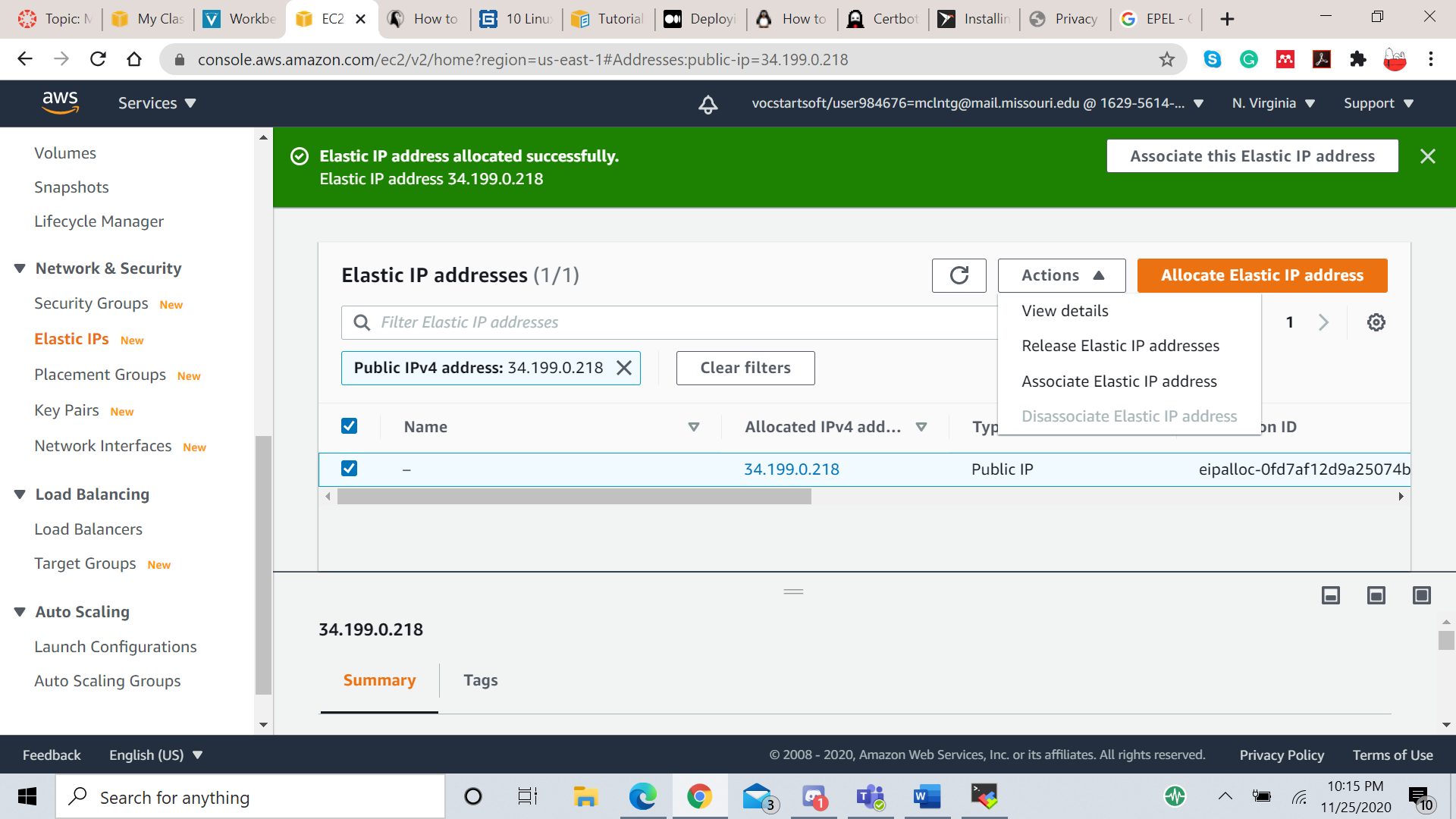
To associate a domain name with the ec2 instance, you must generate a static IP for the ec2. This IP will stay associated with the instance even if it is stopped or restarted. To do this, go to the AWS ec2 dashboard and click on ‘Elastic IPs’ and select ‘Allocate Elastic IP Address’.



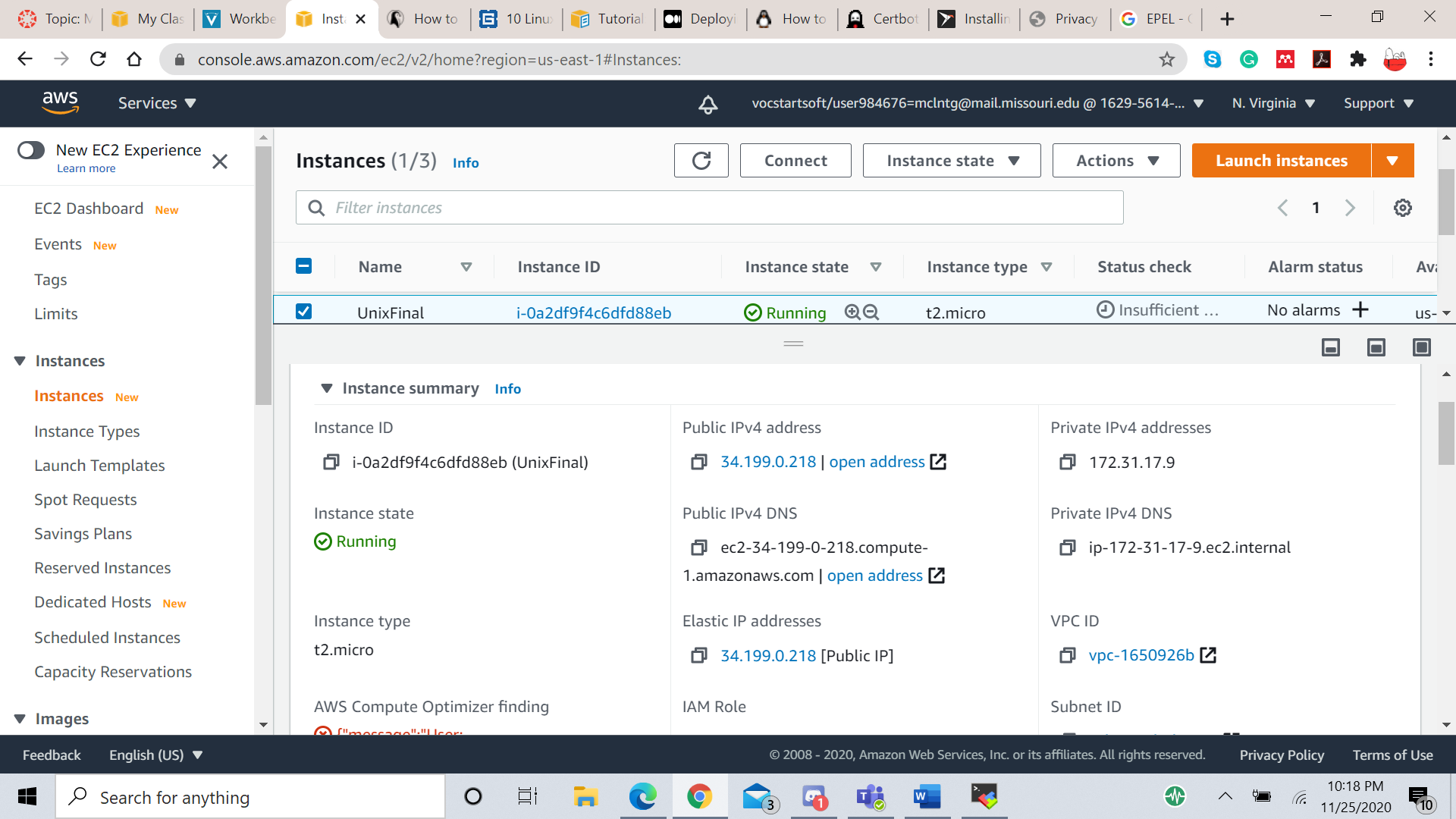
Next, select “Amazon’s pool of IPv4 addresses” for the address pool. Then select ‘Allocate’



Now that the Elastic IP has been allocated, it must be associated with an instance. From the ‘Actions’ drop-down menu, select ‘Associate Elastic IP Address’.



On the next page, select the instance you want to apply the IP to and click ‘Associate’. If successful, the elastic IP will be indicated in the settings for the instance.



Next, the DNS records for the domain name must be updated to point to the elastic IP address that was just allocated. These should be type A records.

For the ACME certification to work, the new domain name must be added to the /etc/httpd/conf/httpd.conf configuration file under the directive “Listen 80”. Run the following the command to edit the file.

$ sudo vi /etc/httpd/conf/httpd.conf

Add the following lines (replacing example.com with the actual domain names)

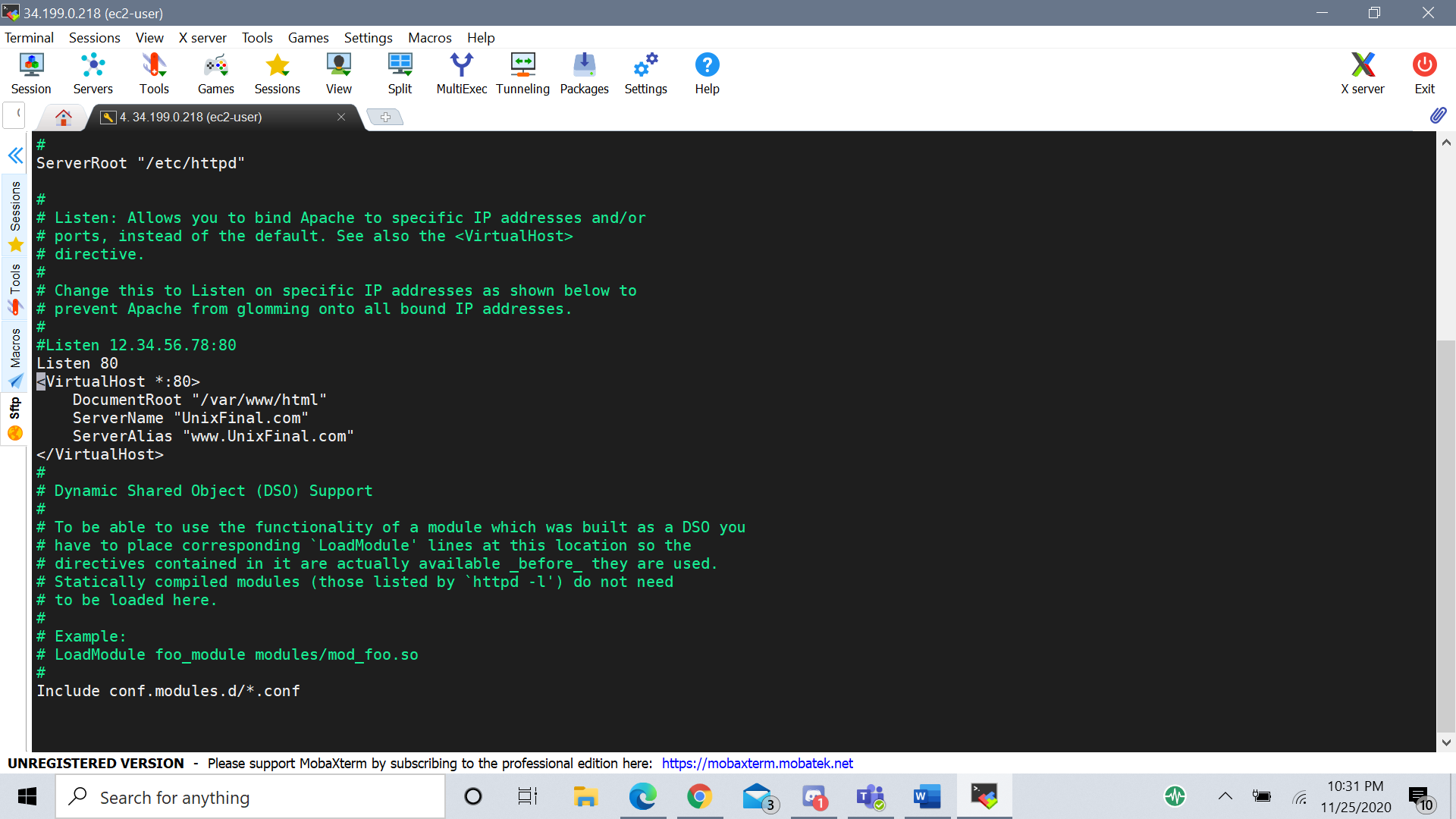
<VirtualHost \*:80>

DocumentRoot "/var/www/html"

ServerName "example.com"

ServerAlias "www.example.com"

</VirtualHost>



Save the changes and restart apache to apply the changes to the server.

$ sudo systemctl restart httpd

1. Obtaining a certificate from a CA

First, the snap daemon must be installed. It will be used to install Certbot. Run the following command to add the EPEL repository (Extra Packages for Enterprise Linux) which contains snapd. Then update dnf

$ sudo dnf install <https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm>

$ sudo dnf upgrade

Then run these commands to install and enable snapd.

$ sudo yum install snapd

$ sudo systemctl enable --now snapd.socket

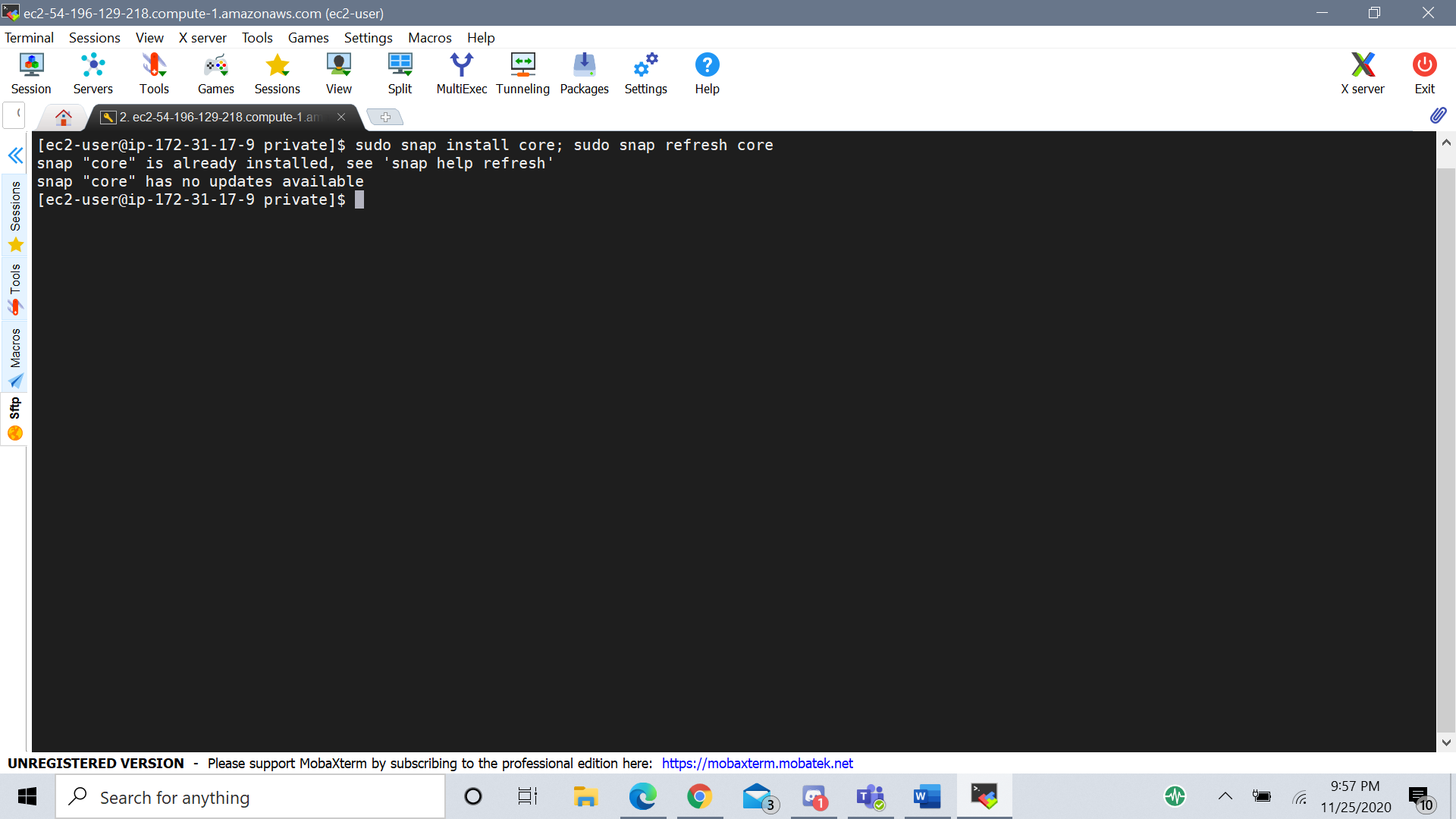
Lastly, for snapd to work, a symbolic link must be created between /snap and /var/lib/snapd/snap.

$ sudo ln -s /var/lib/snapd/snap /snap

Restart the system for these changes to take effect.

If snapd was just installed, then it should be up to date, but to check you can run the command

$ sudo snap install core; sudo snap refresh core



Before running Certbot, make sure any Certbot packages from the OS installation are removed, or else the certbot command may run that version instead of the snap version. To remove Certbot packages, type

$ sudo dnf remove certbot



To install Certbot, enter the following command

$ sudo snap install --classic certbot

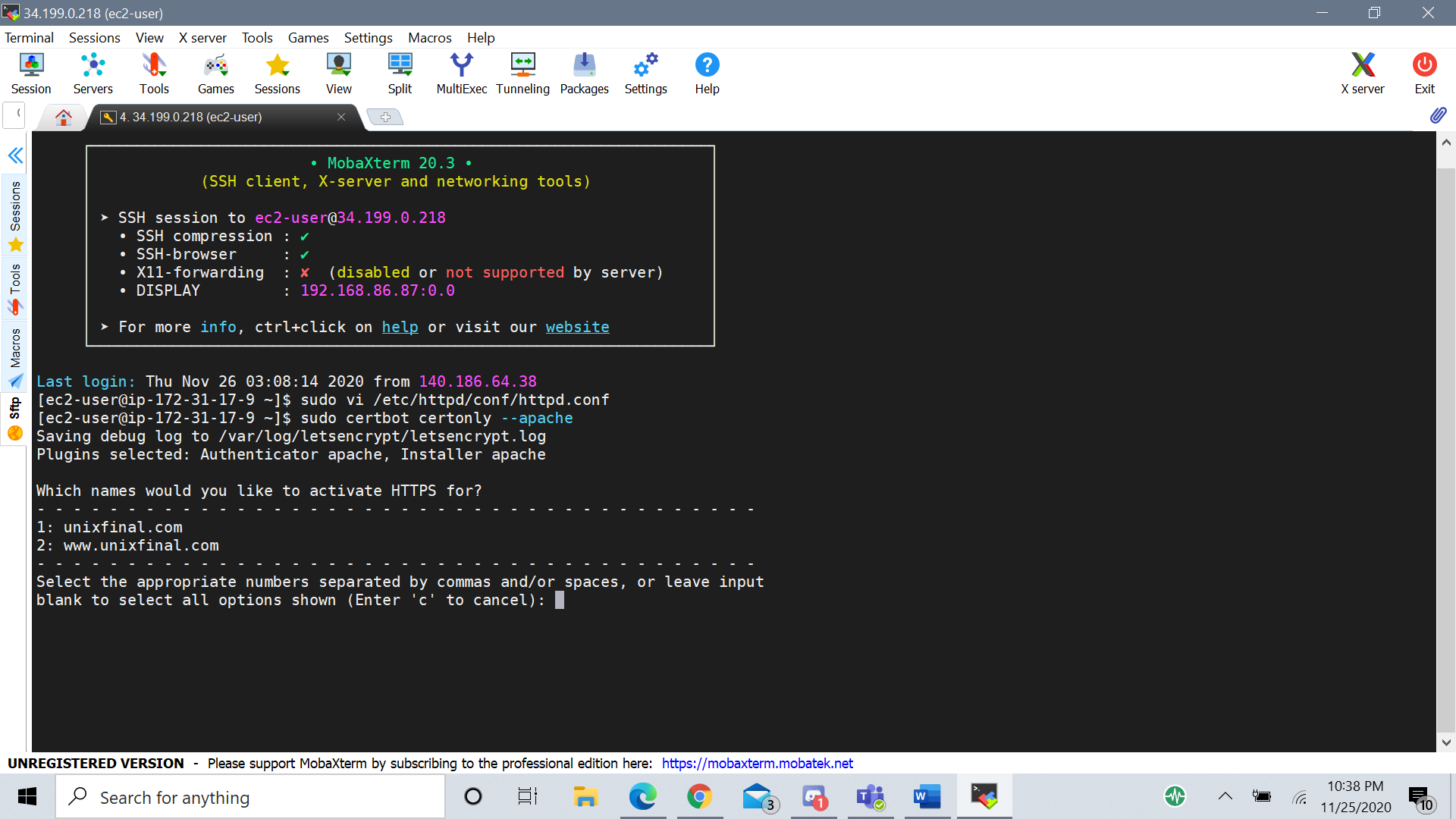
In order for Certbot to run correctly, a symbolic link must be created between the files /snap/bin/certbot and /usr/bin/certbot. Run the following command.

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

Lastly, run Certbot with this command.

$ sudo certbot certonly –apache

This will enter an interactive menu. Select which domain names you want to certify and Certbot will automatically test them and obtain the certificate from Let’s Encrypt. Now you have an official HTTPS website!



**Sources**

<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/SSL-on-amazon-linux-2.html#letsencrypt>

<https://medium.com/@gnowland/deploying-lets-encrypt-on-an-amazon-linux-ami-ec2-instance-f8e2e8f4fc1f>

<https://linuxconfig.org/how-to-install-mod-ssl-on-redhat-8>

<https://www.cyberciti.biz/faq/how-to-install-wget-on-rhel-8-redhat-enterprise-linux/>

<https://dev.to/maybebored/how-to-set-up-a-custom-domain-for-your-ec2-instance-without-using-route53-f9>

<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/elastic-ip-addresses-eip.html>

<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-instance-addressing.html#working-with-ip-addresses>

<http://httpd.apache.org/docs/2.4/mod/core.html#virtualhost>

<https://certbot.eff.org/lets-encrypt/centosrhel8-apache>

<https://letsencrypt.org/getting-started/>

<https://www.websecurity.digicert.com/security-topics/what-is-ssl-tls-https>

<https://www.ssl.com/faqs/faq-what-is-ssl/>