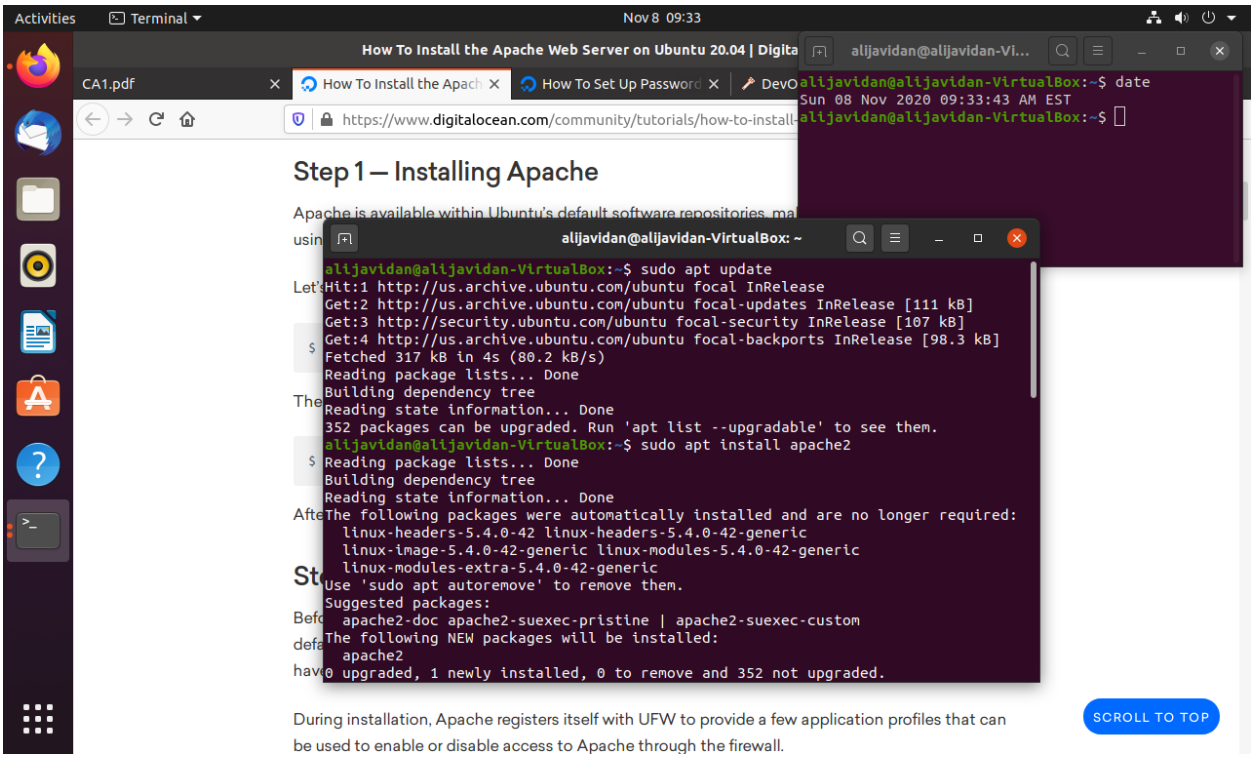
1. Set up a HTTP server with user authentication

Step 1 — Installing Apache

Let's begin by updating the local package index to reflect the latest upstream changes.

Then, install the apache2 package.

After confirming the installation, apt will install Apache and all required dependencies.



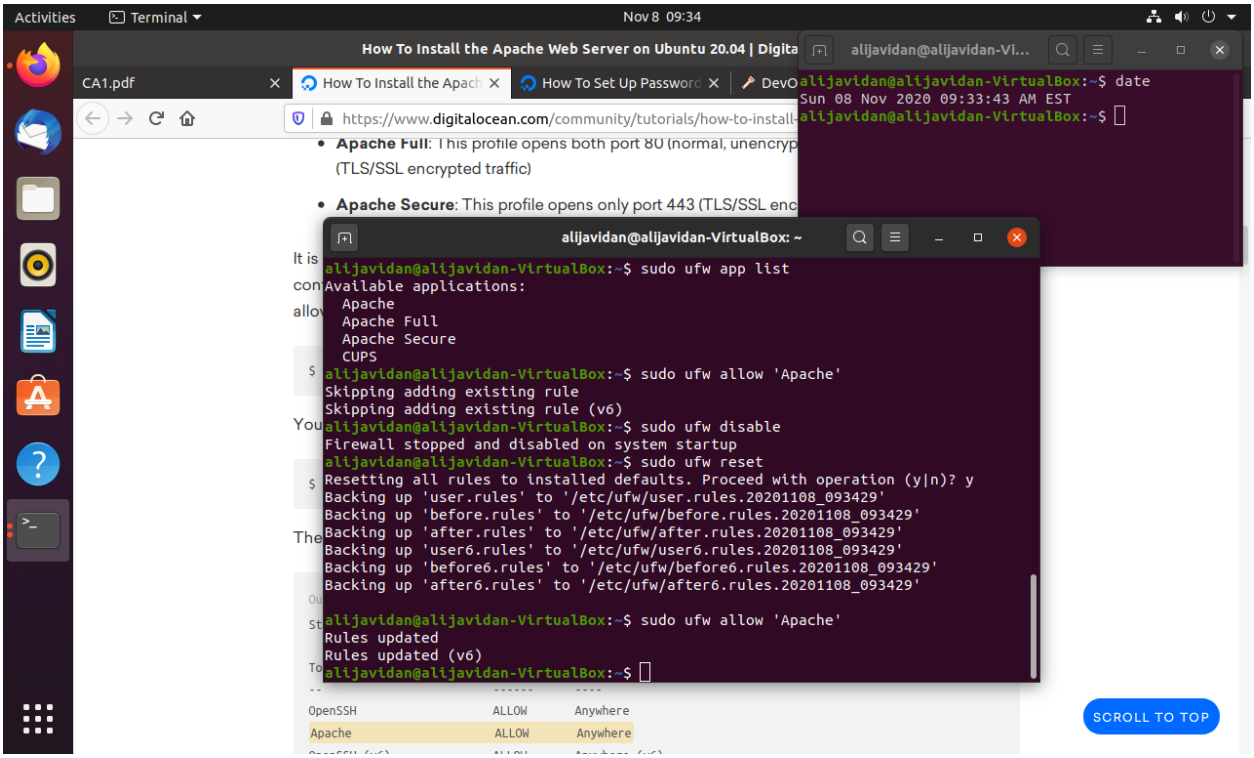
Step 2 — Adjusting the Firewall

During installation, Apache registers itself with UFW to provide a few application profiles that can be used to enable or disable access to Apache through the firewall.

We can list the ufw application profiles by typing:

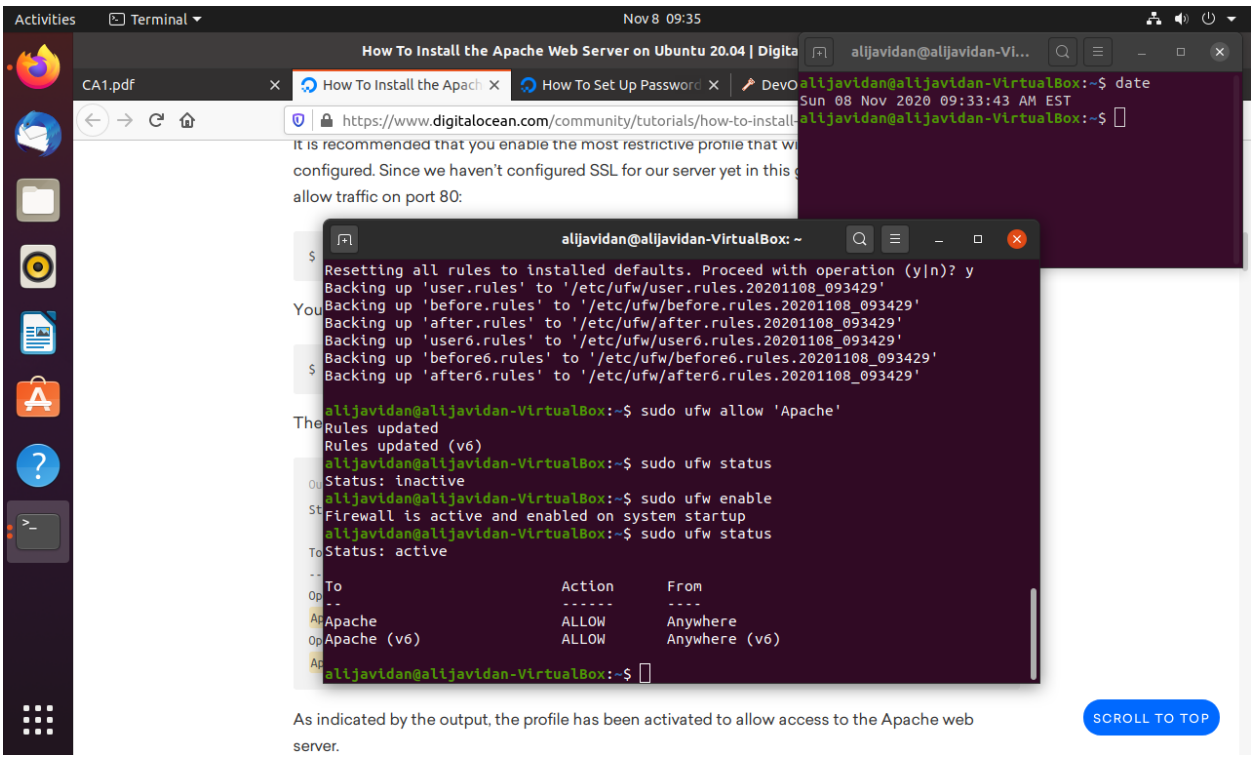
```
sudo ufw app list
```

It is recommended that we enable the most restrictive profile that will still allow the traffic we've configured. Since we haven't configured SSL for our server yet, we will only need to allow traffic on port 80.



We can verify the change by typing:

```
sudo ufw status
```



But since ufw status is inactive, we enable it by typing:

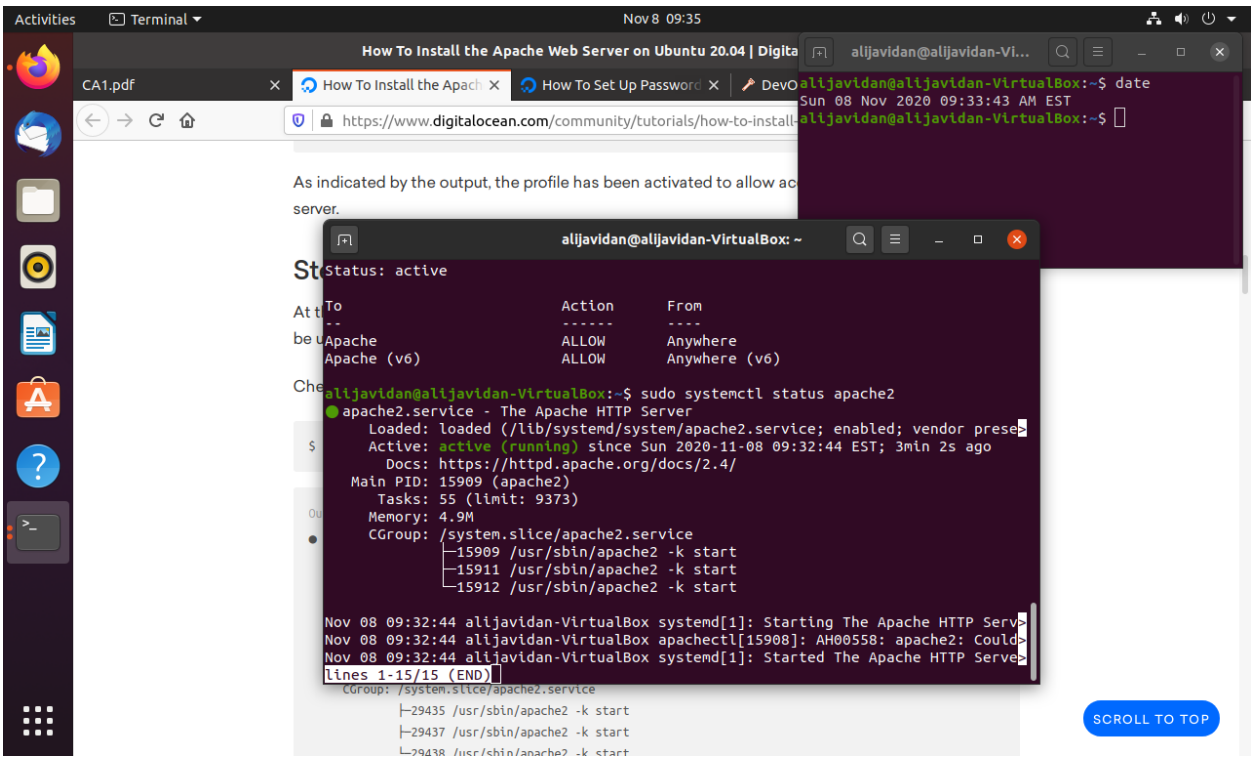
```
sudo ufw enable
```

And check its status again by typing:

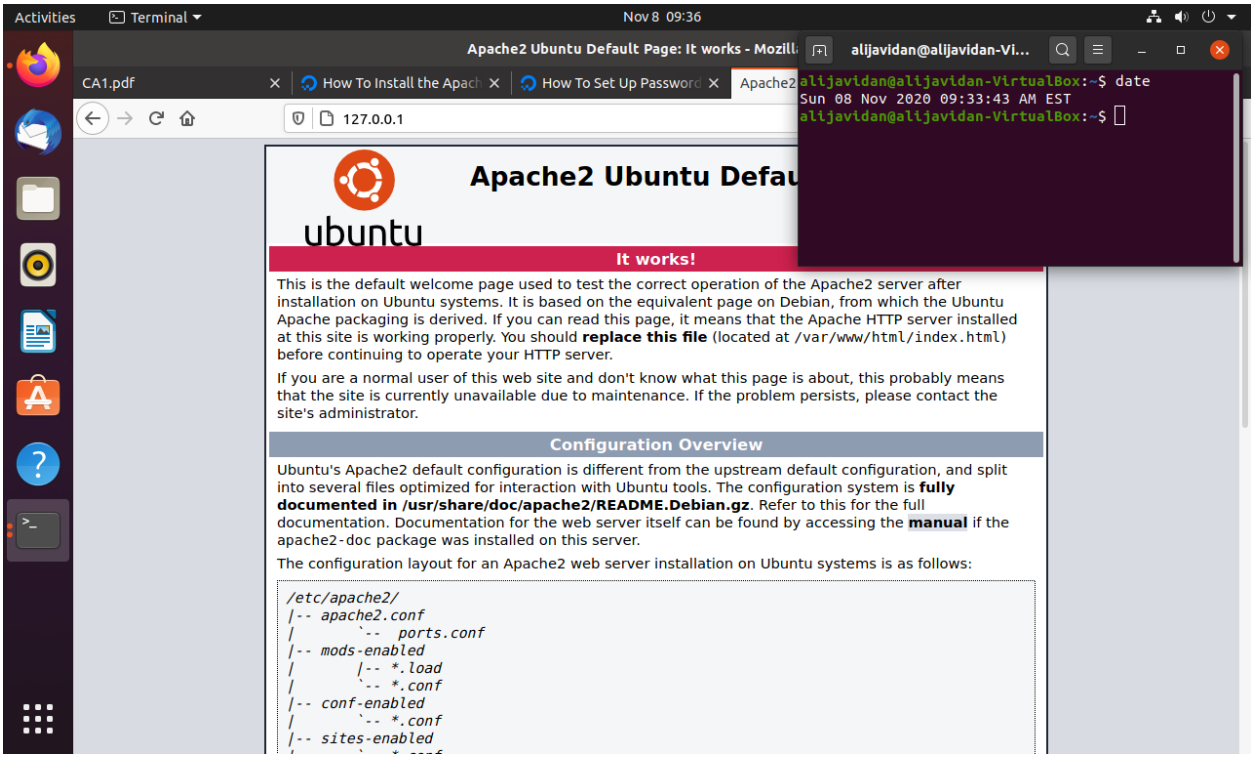
```
sudo ufw status
```

again.

Step 3 — Checking our Web Server
Wec check with the systemd init system to make sure the service is running.



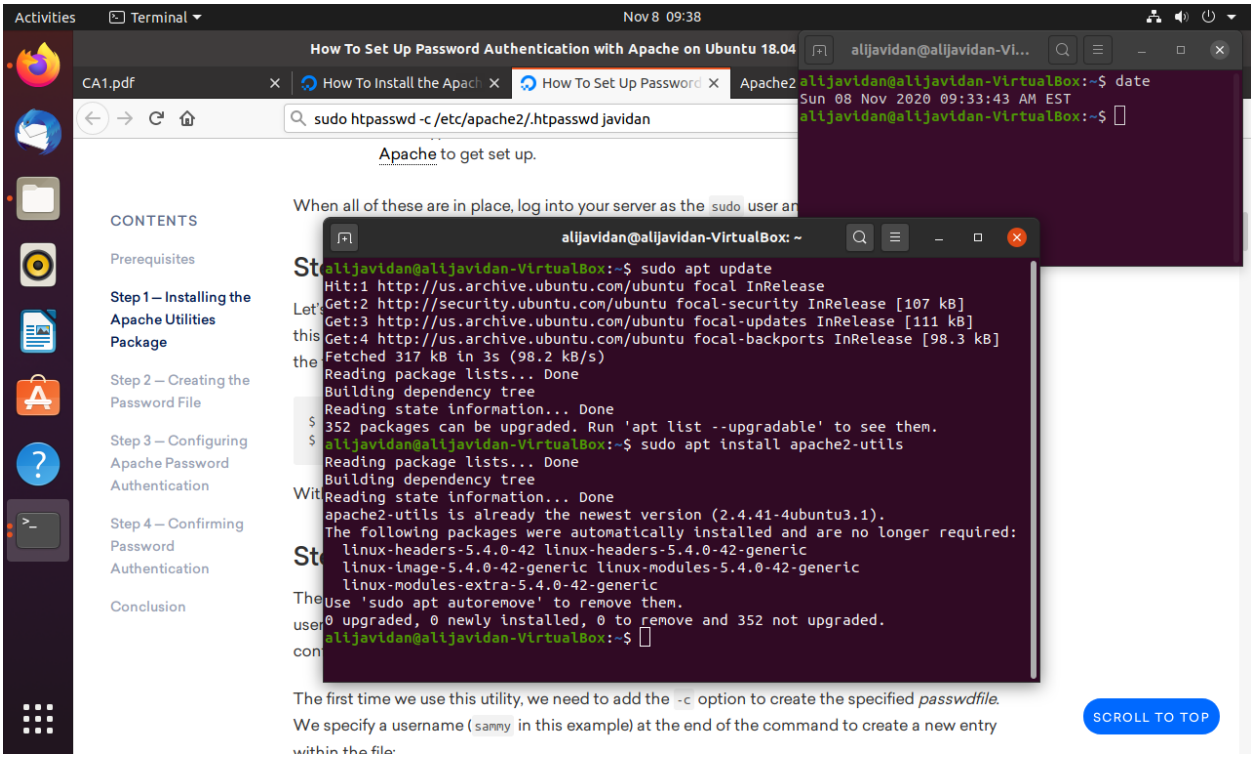
Now we enter 127.0.0.1 into our browser’s address bar, and we could see apache2 homepage gets shown up.



Set Up Password Authentication

Step 1 — Installing the Apache Utilities Package

Let’s begin by updating our server and installing a package that we’ll need. We will be using a utility called htpasswd, part of the apache2-utils package, to create the file and manage the username and passwords needed to access restricted content.



With this installed, we now have access to the htpasswd command.

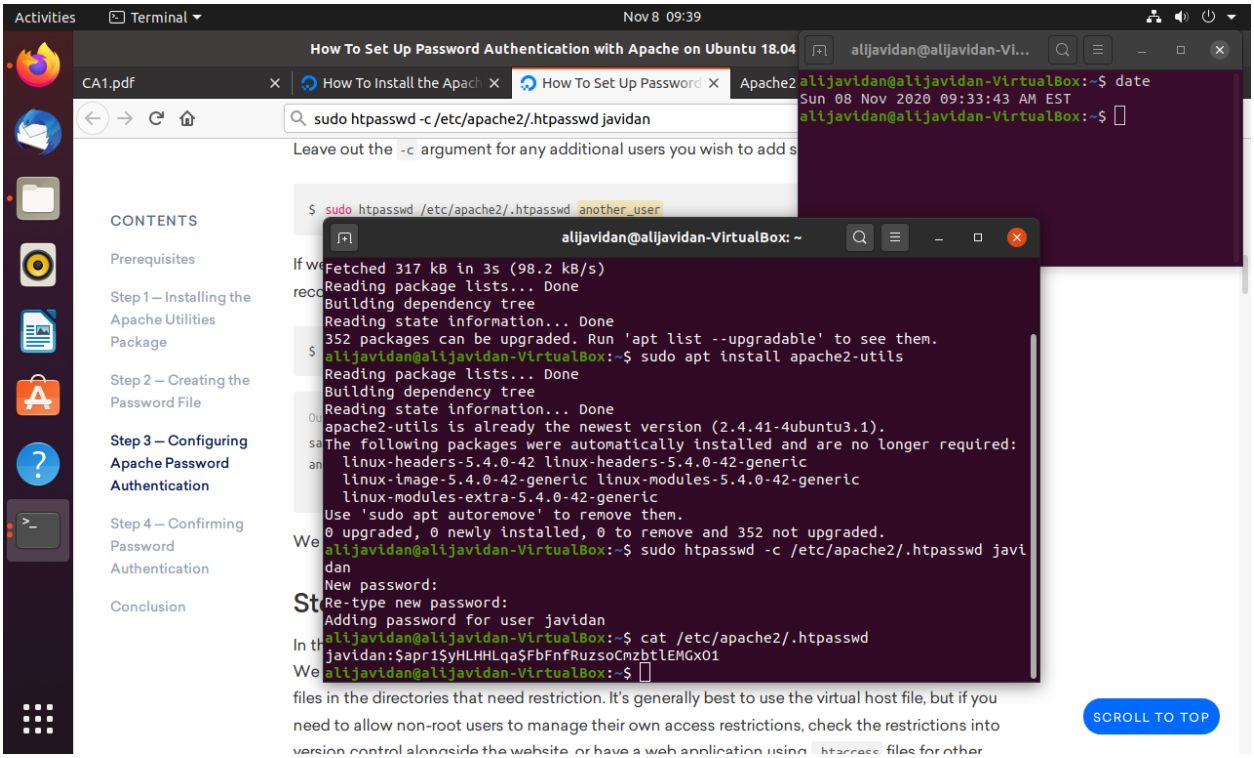
Step 2 — Creating the Password File

The htpasswd command will allow us to create a password file that Apache can use to authenticate users. We will create a hidden file for this purpose called .htpasswd within our /etc/apache2 configuration directory.

The first time we use this utility, we need to add the -c option to create the specified passwdfile. We specify javidan as username at the end of the command to create a new entry within the file.

We will be asked to supply and confirm a password for the user.

If we view the contents of the file, we can see the username and the encrypted password for each record:

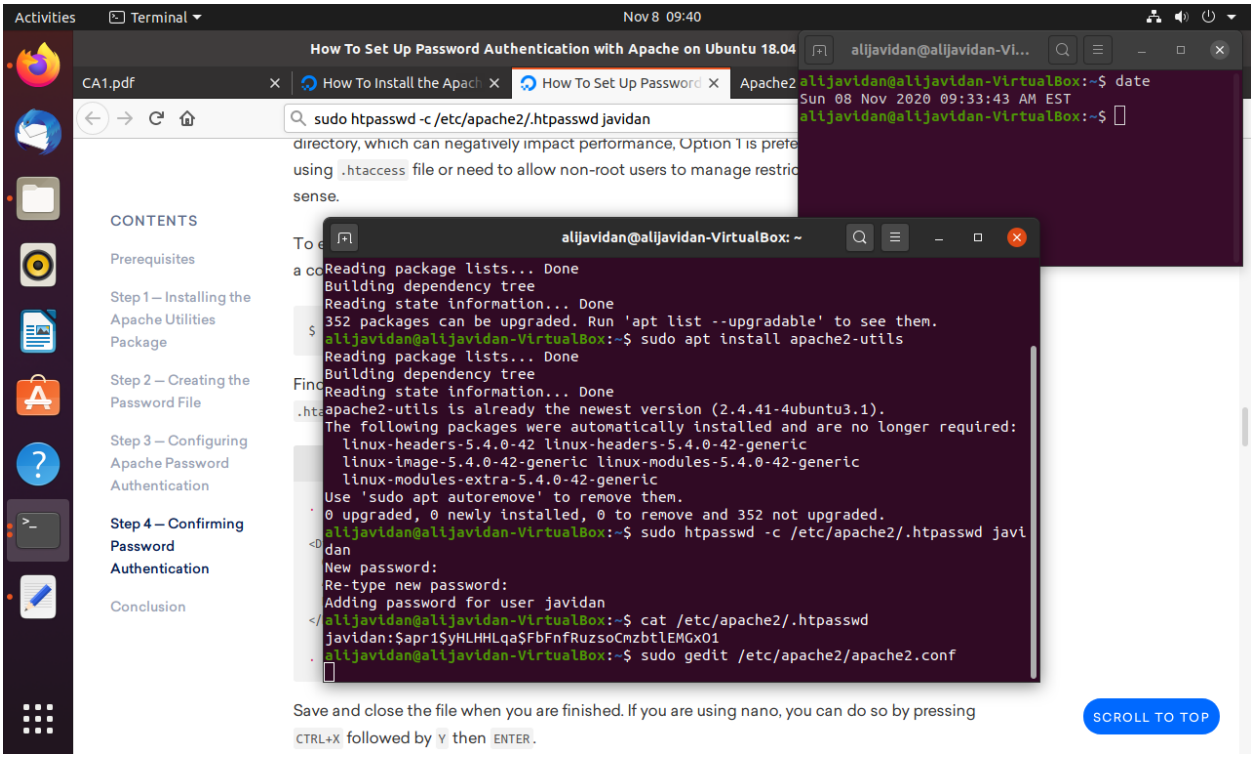


Step 3 — Configuring Apache Password Authentication

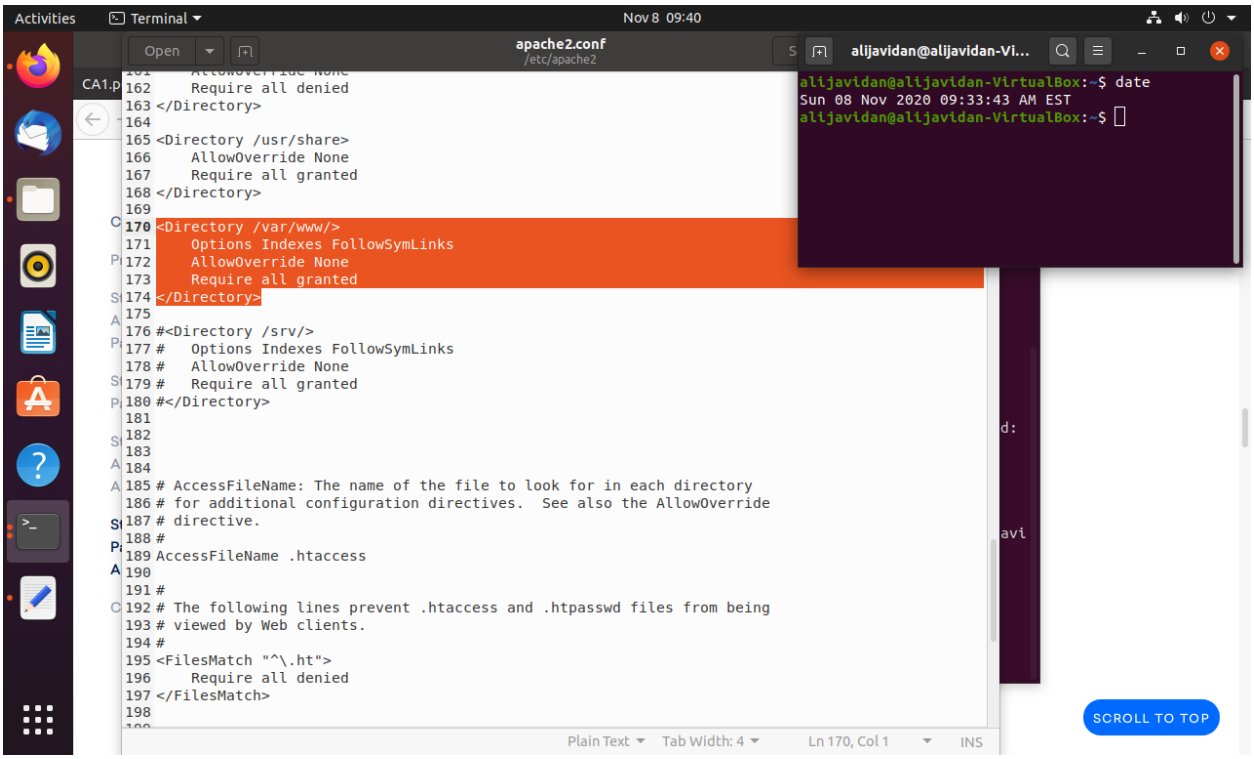
Configuring Access Control with .htaccess Files

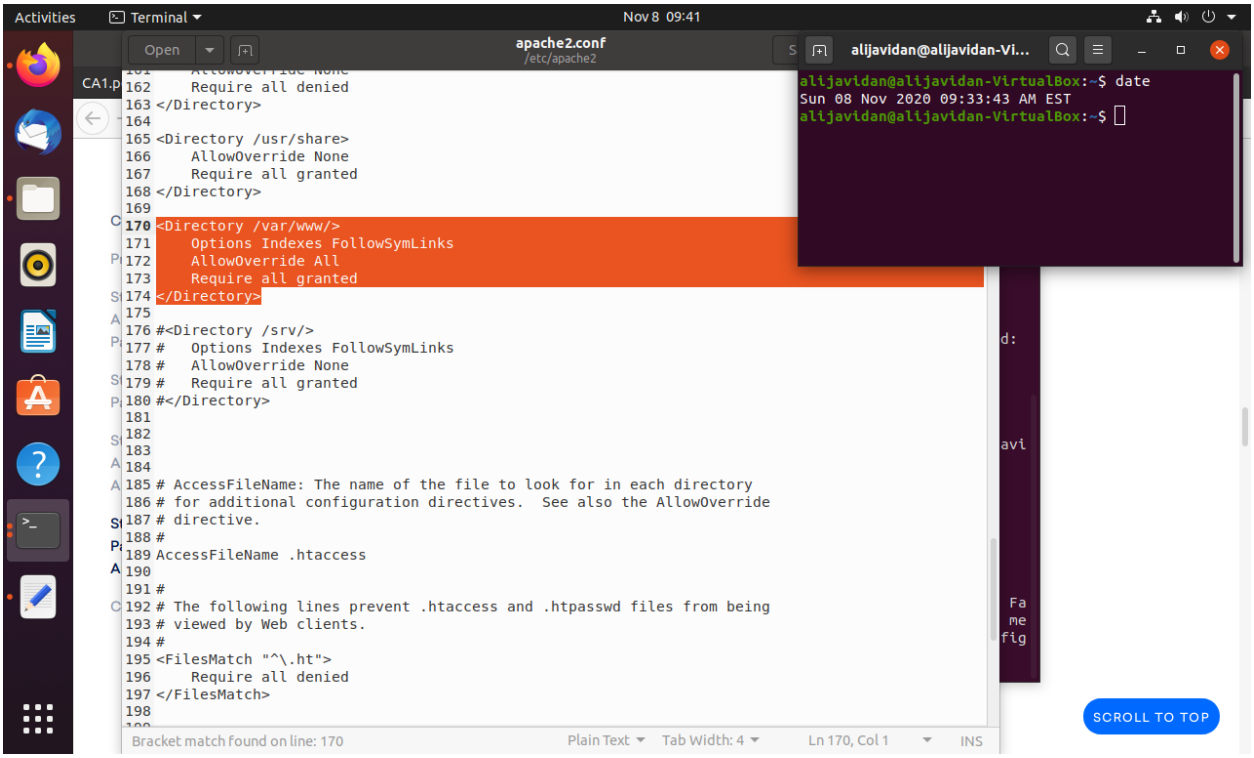
Apache can use .htaccess files in order to allow certain configuration items to be set within a content directory.

To enable password protection using .htaccess files, we open the main Apache configuration file with a command-line text editor such as gedit.

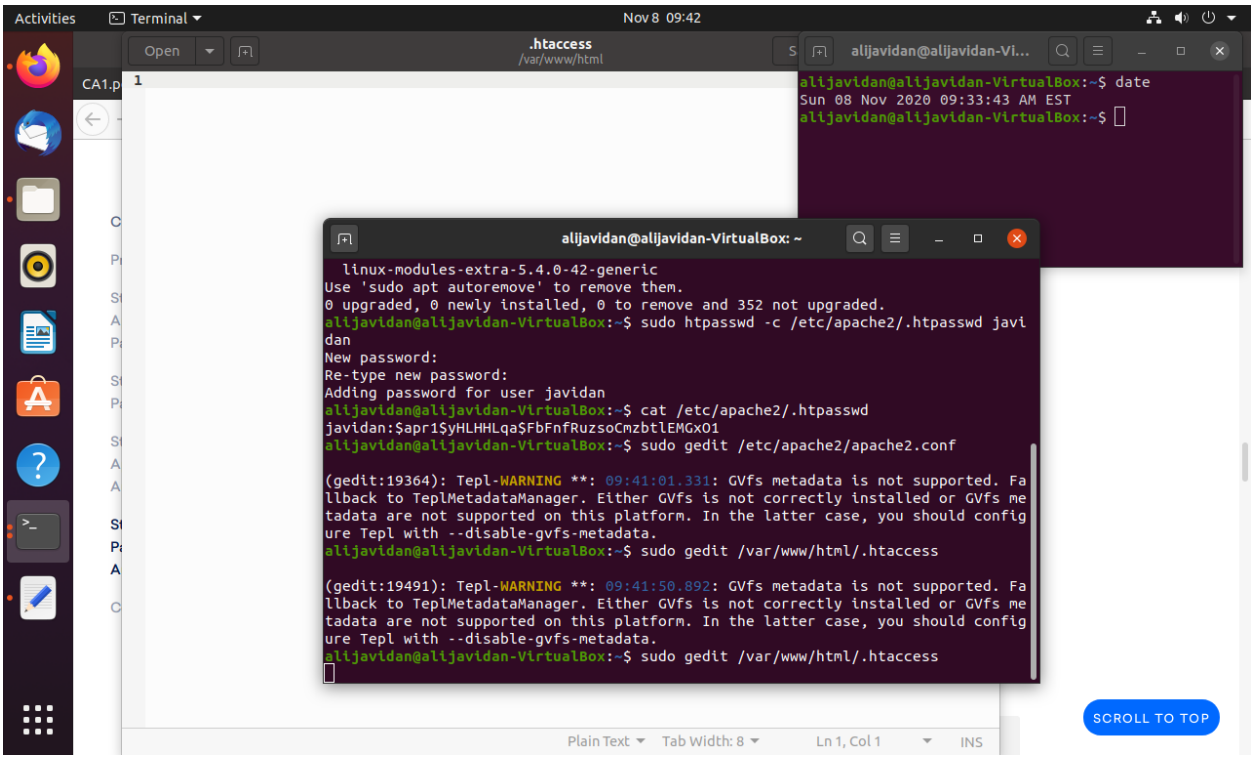


We find the <Directory> block for the /var/www directory that holds the document root. We turn on .htaccess processing by changing the AllowOverride directive within that block from None to All:

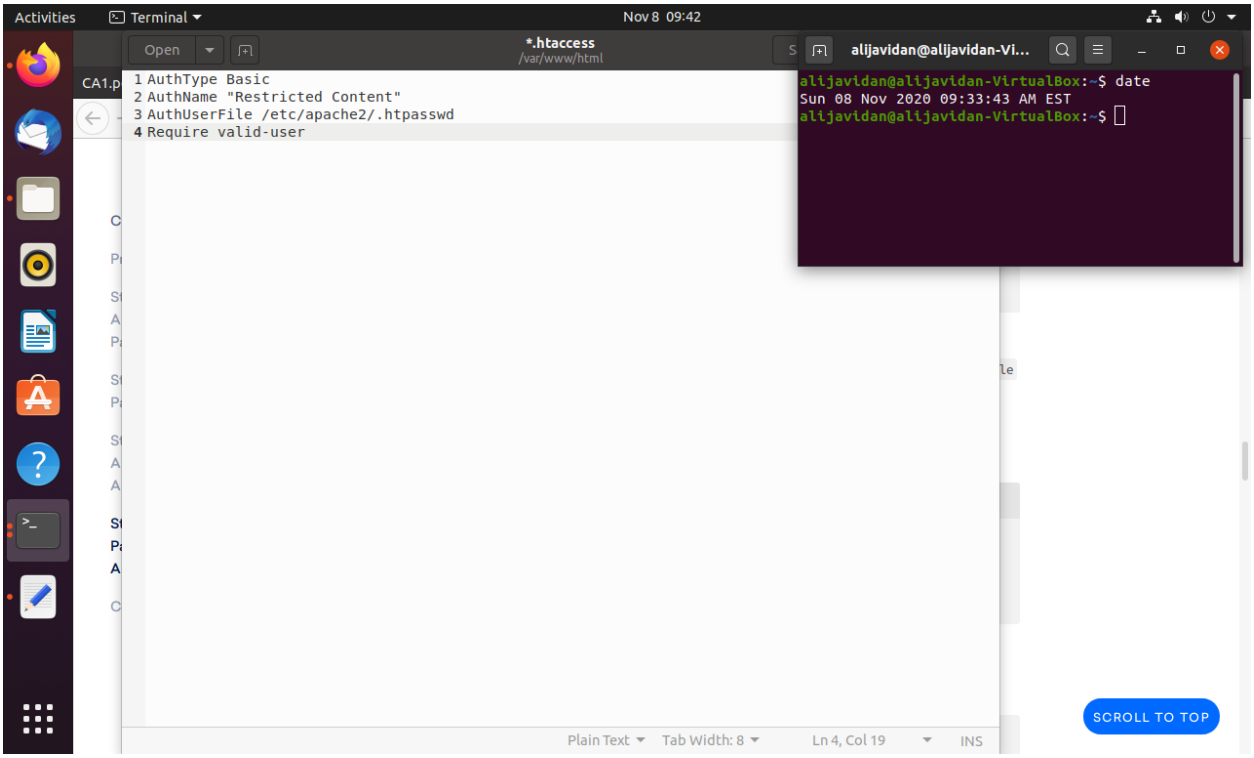




Next, we need to add an .htaccess file to the directory we wish to restrict. In our demonstration, we'll restrict the entire document root (the entire website) which is based at /var/www/html .



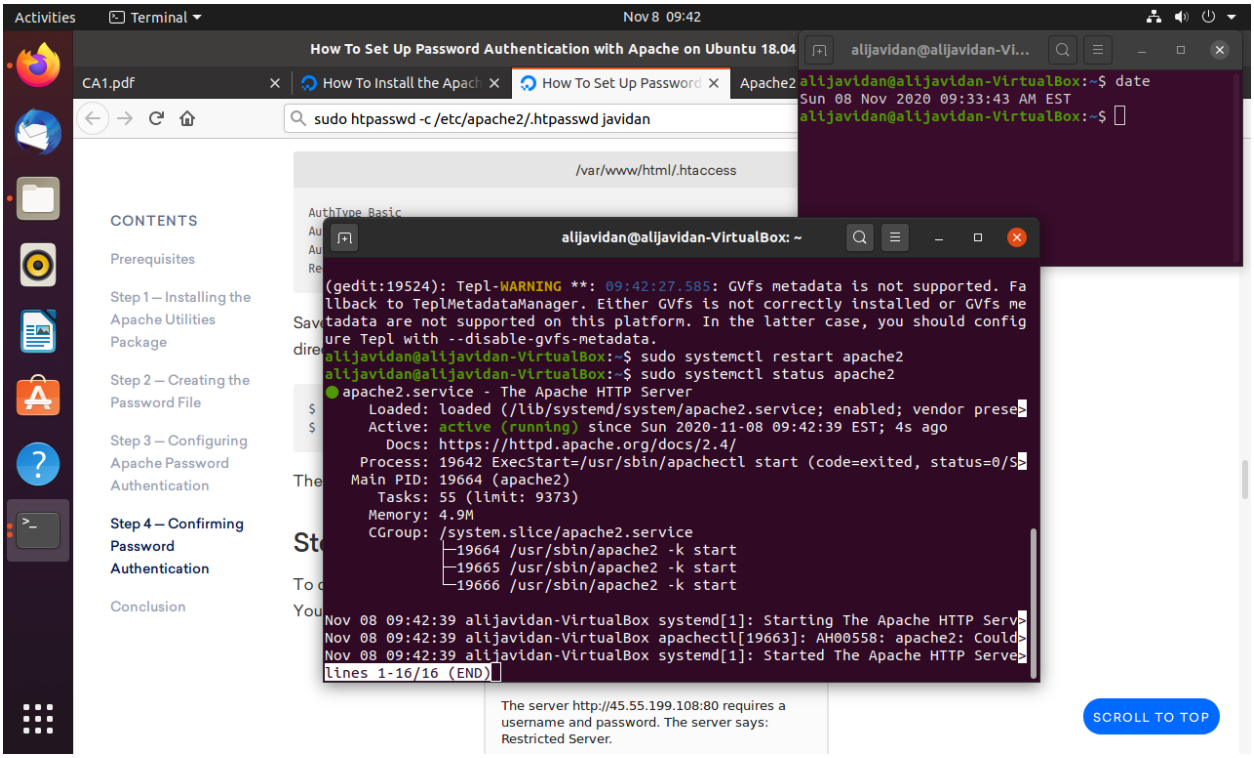
Within this file, we specify that we wish to set up Basic authentication. For the AuthName, we choose a realm name that will be displayed to the user when prompting for credentials. We use the AuthUserFile directive to point Apache to the password file we created. Finally, we will require a valid-user to access this resource, which means anyone who can verify their identity with a password will be allowed in:



We restart the web server to password protect all content in or below the directory with the .htaccess file and use

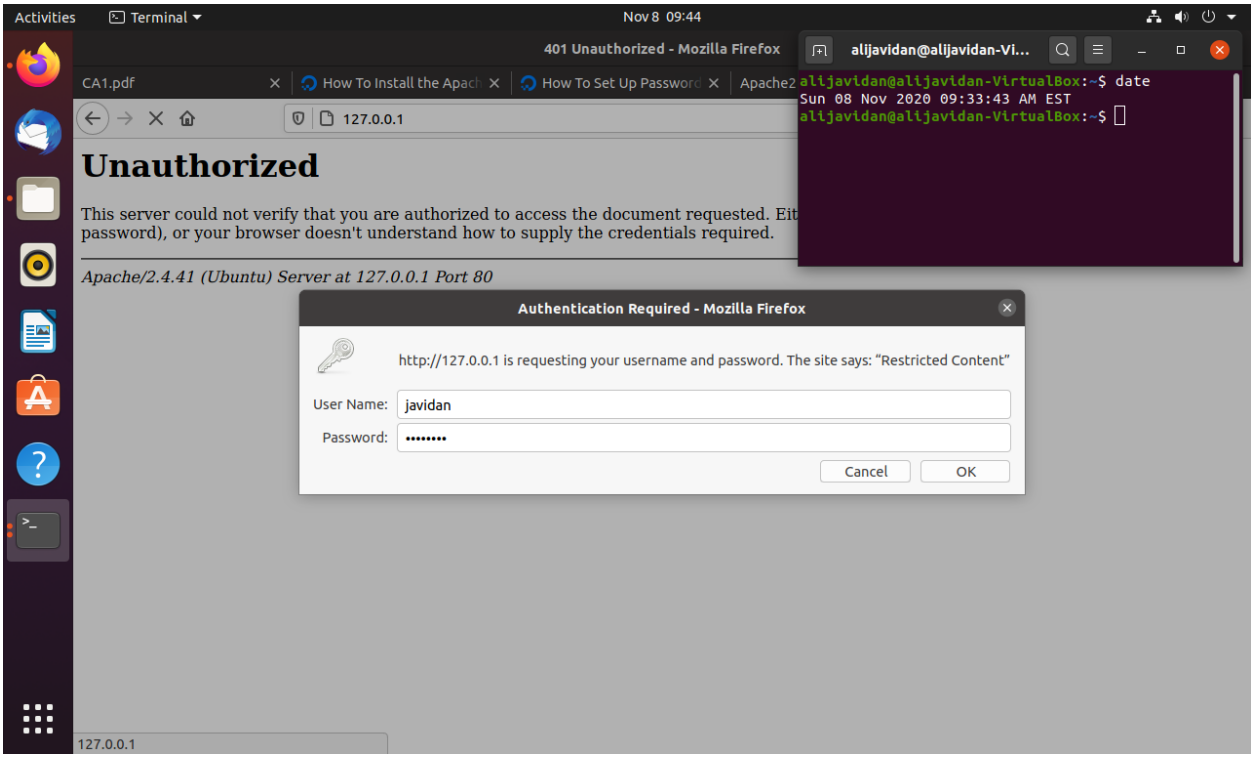
```
sudo systemtcl status apache2
```

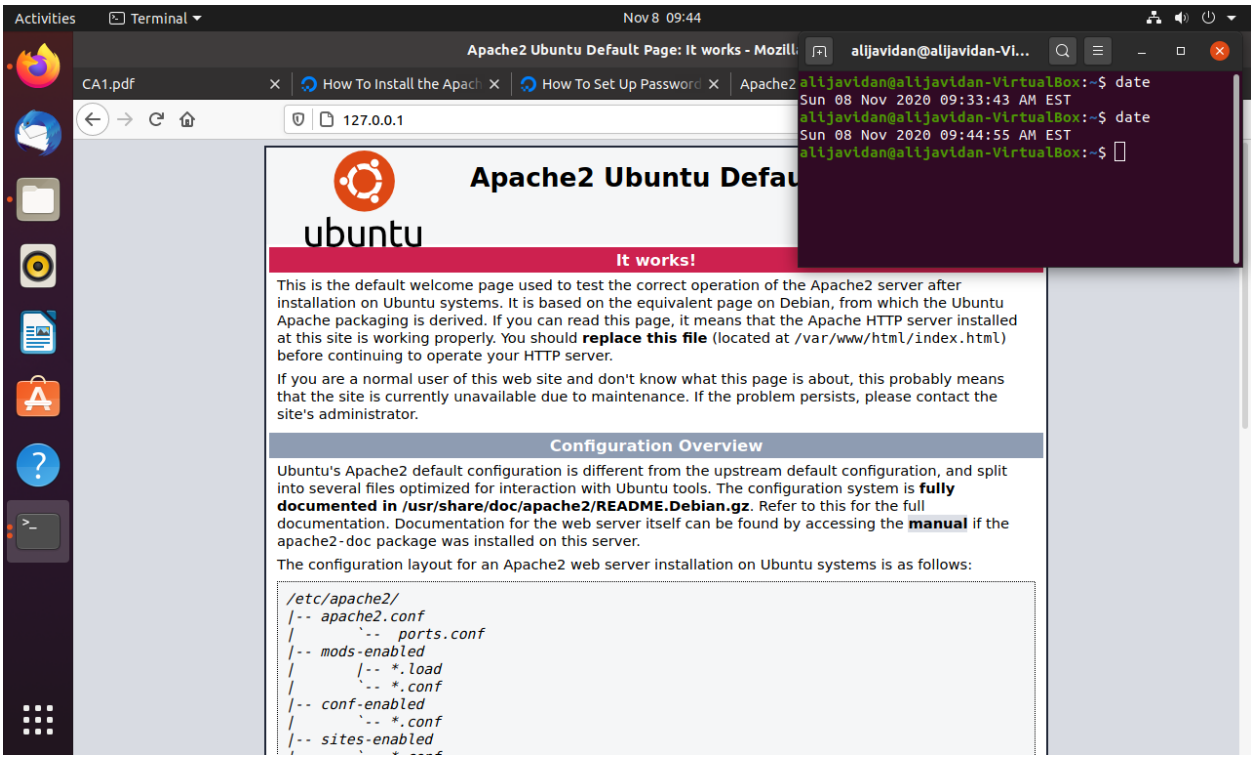
to verify the success of the restart.



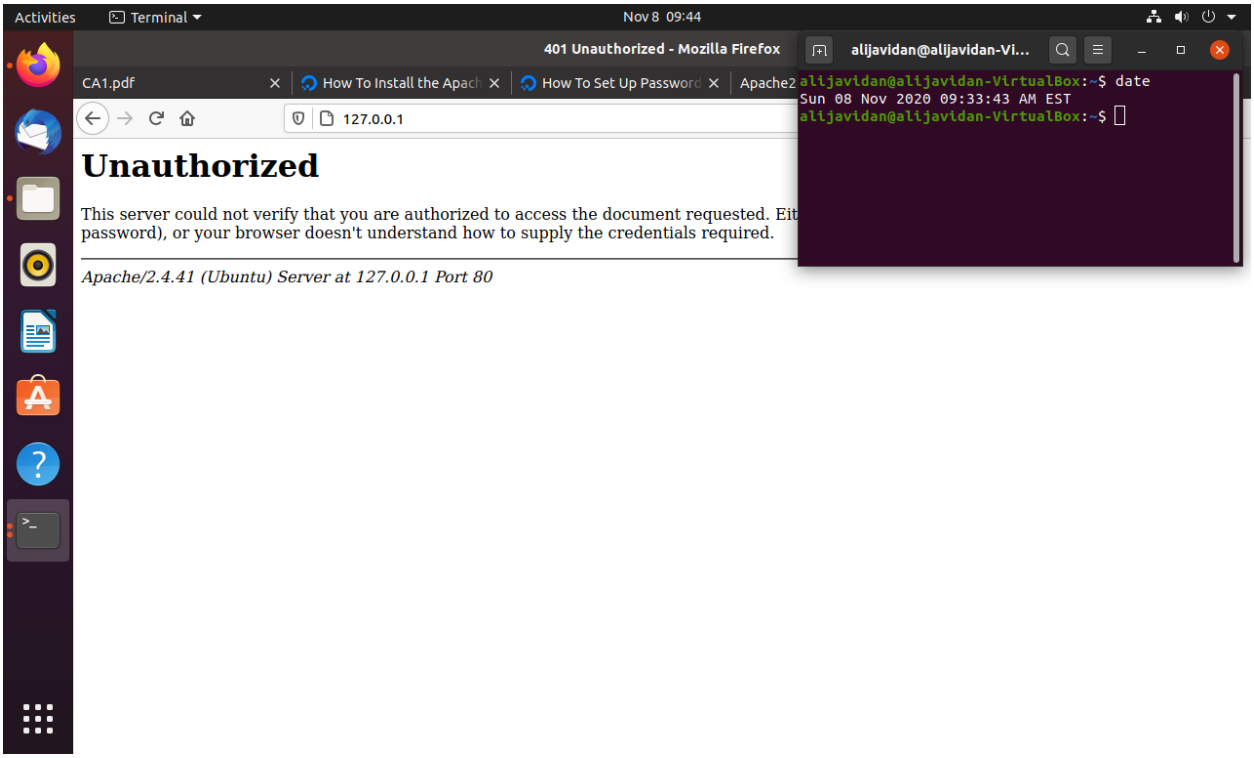
Step 4 — Confirming Password Authentication

To confirm that our content is protected, we try to access our restricted content in a web browser. We should be presented with a username and password prompt that looks like below screeshot.
If we enter the correct credentials, we will be allowed to access the content.





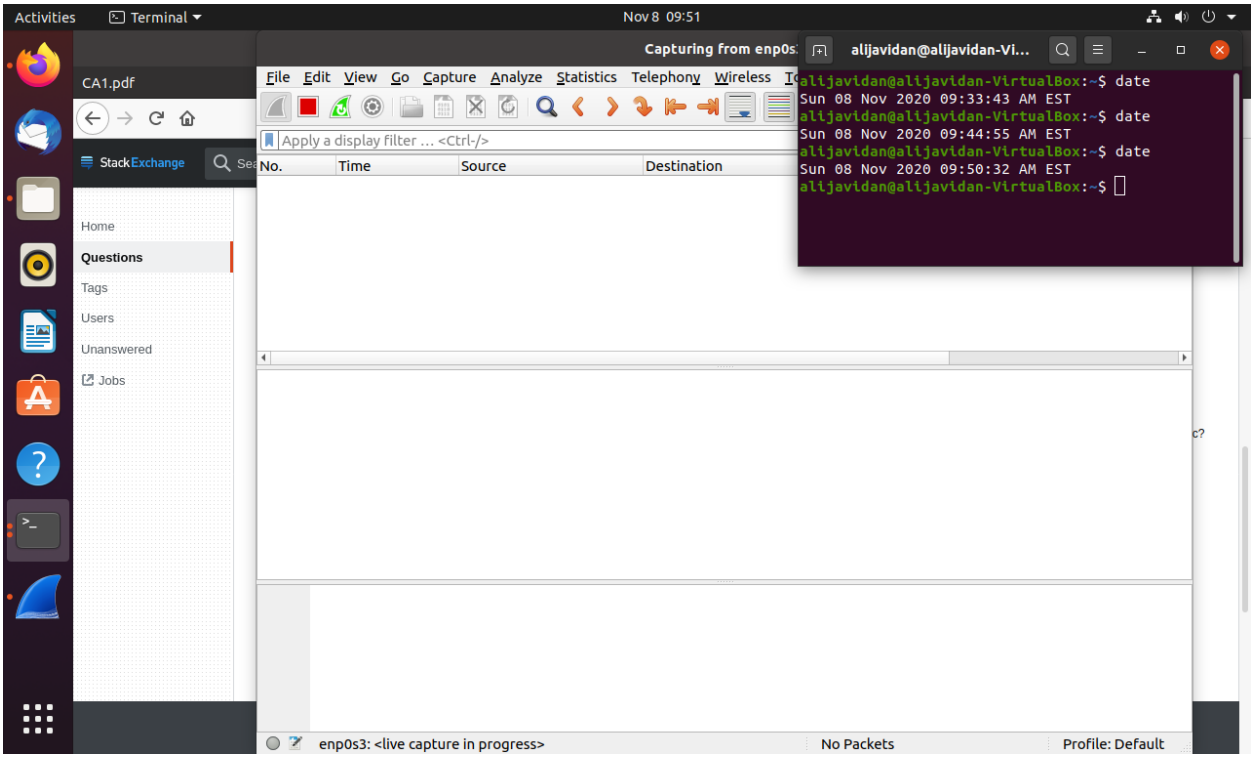
If we enter the wrong credentials or hit “Cancel”, we will see the “Unauthorized” error page:



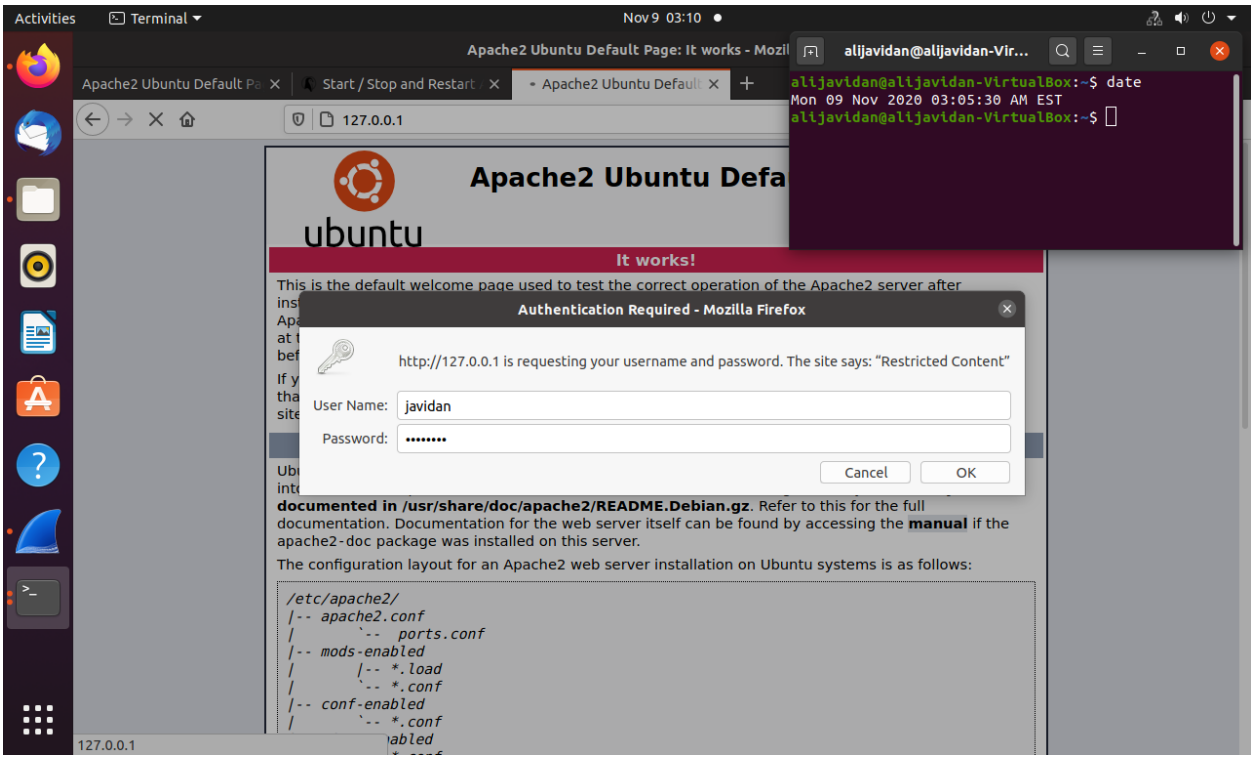
Wireshark

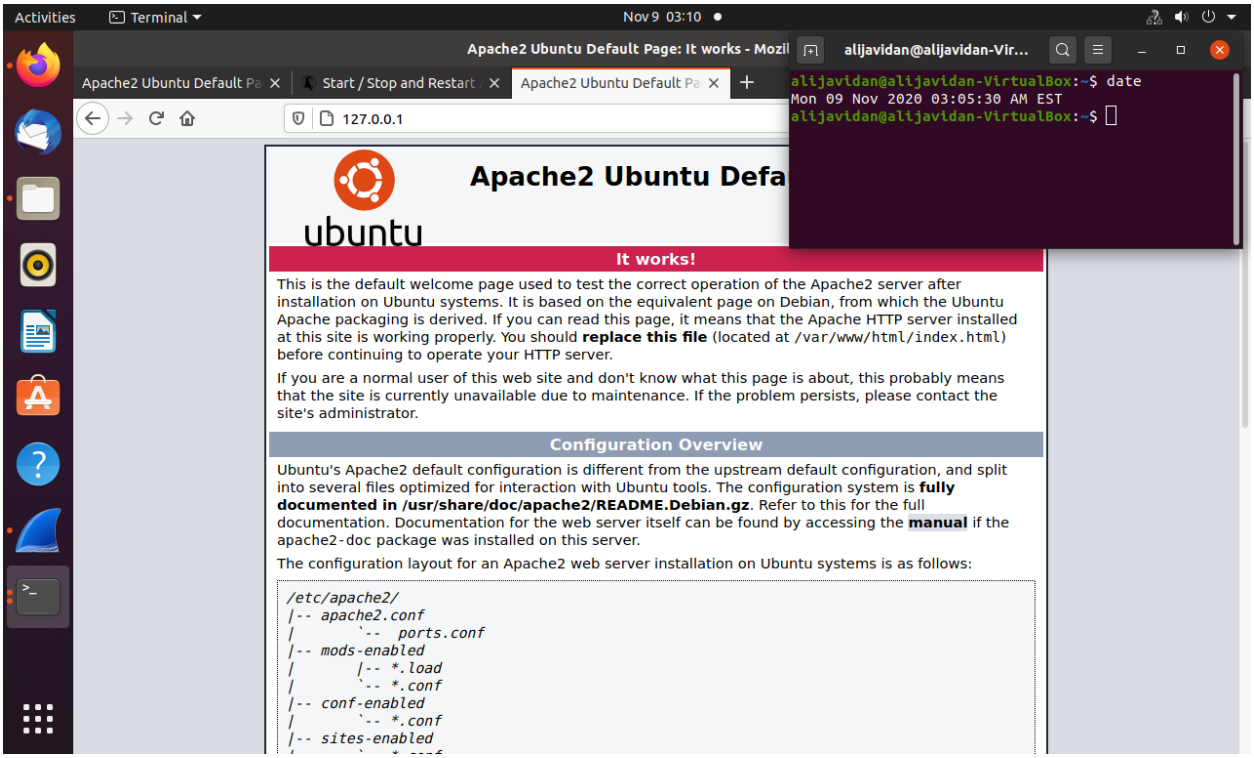
Since all localhost (127.0.0.1) traffic get passed through loopback port, we could use tools like Wireshark to capture them.

We start Wireshark capture,

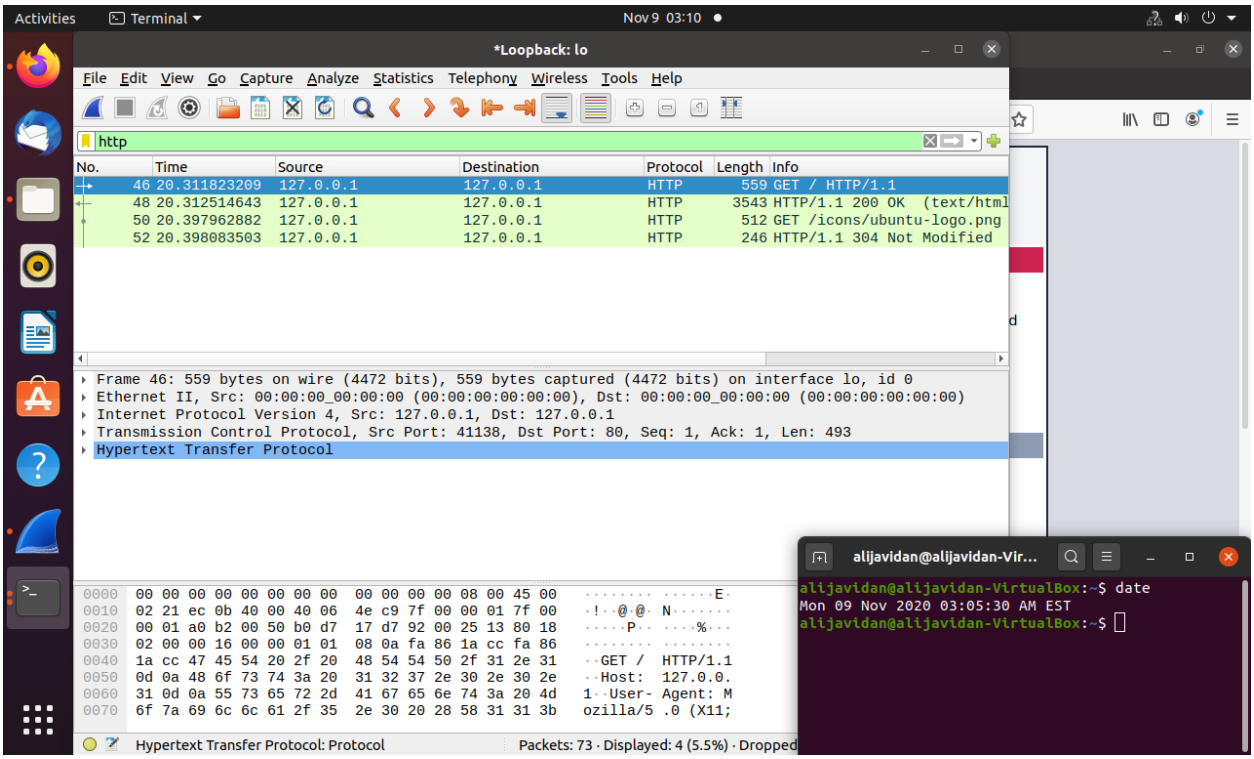


and login to Apache homepage with our creditioanlal (javidan as User Name) through authentication procedure we have previously built up.

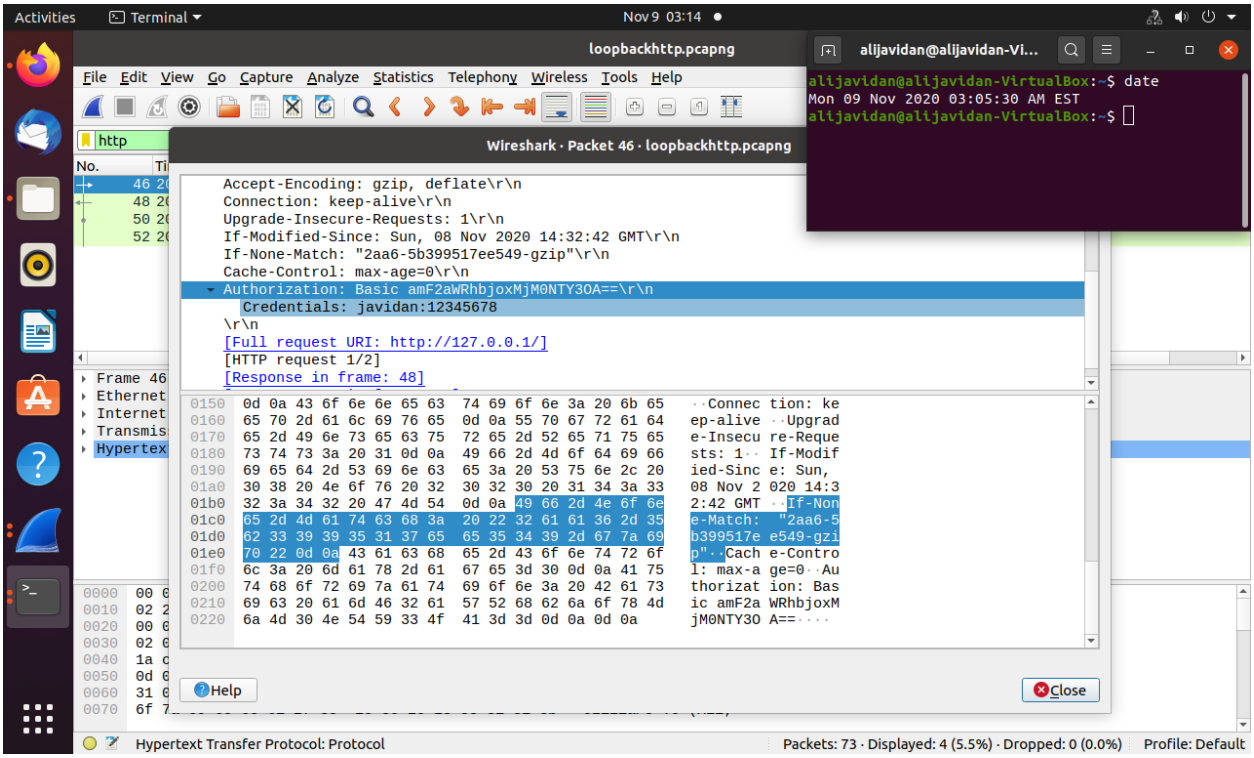




Now we stop Wireshark capture, filter out http packets as below:



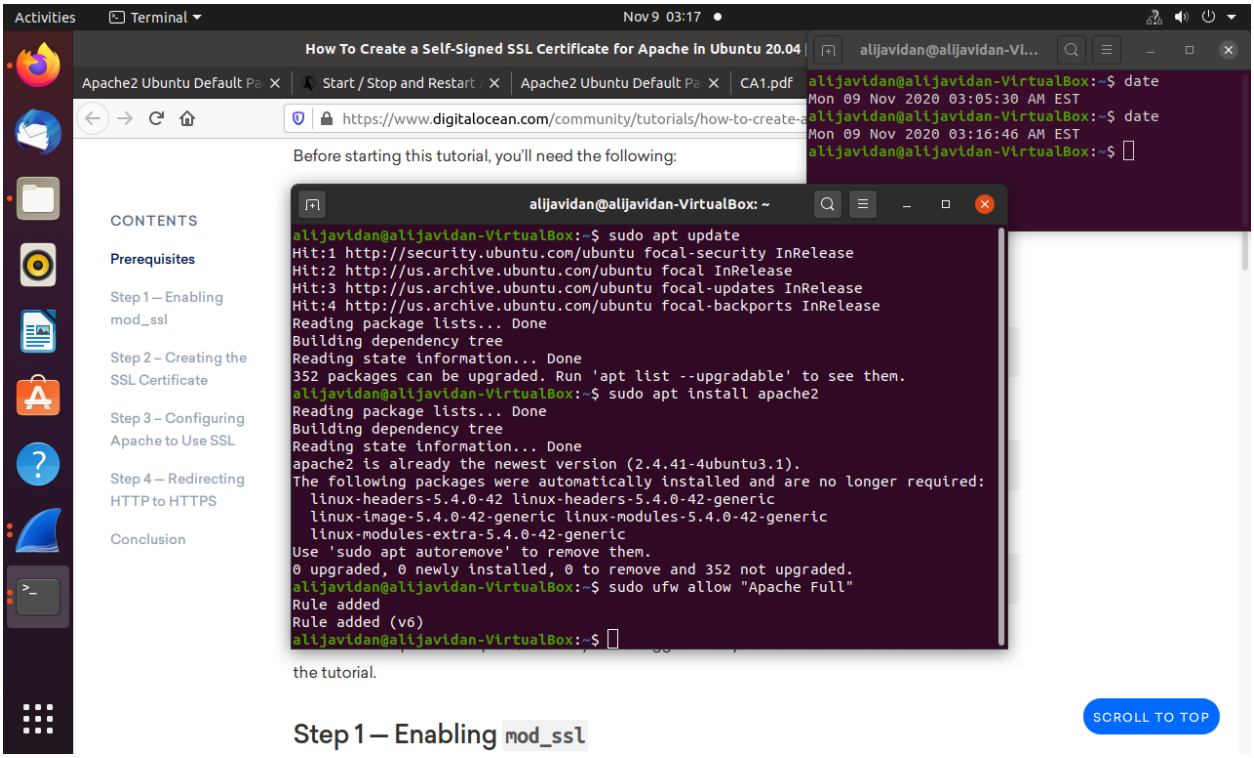
We could see creditionals we used to login under Authorization parameter as follow:



This is because datas sent on channel are not encroyted, resulting in them being easily sniffed.

Create a Self-Signed SSL Certificate

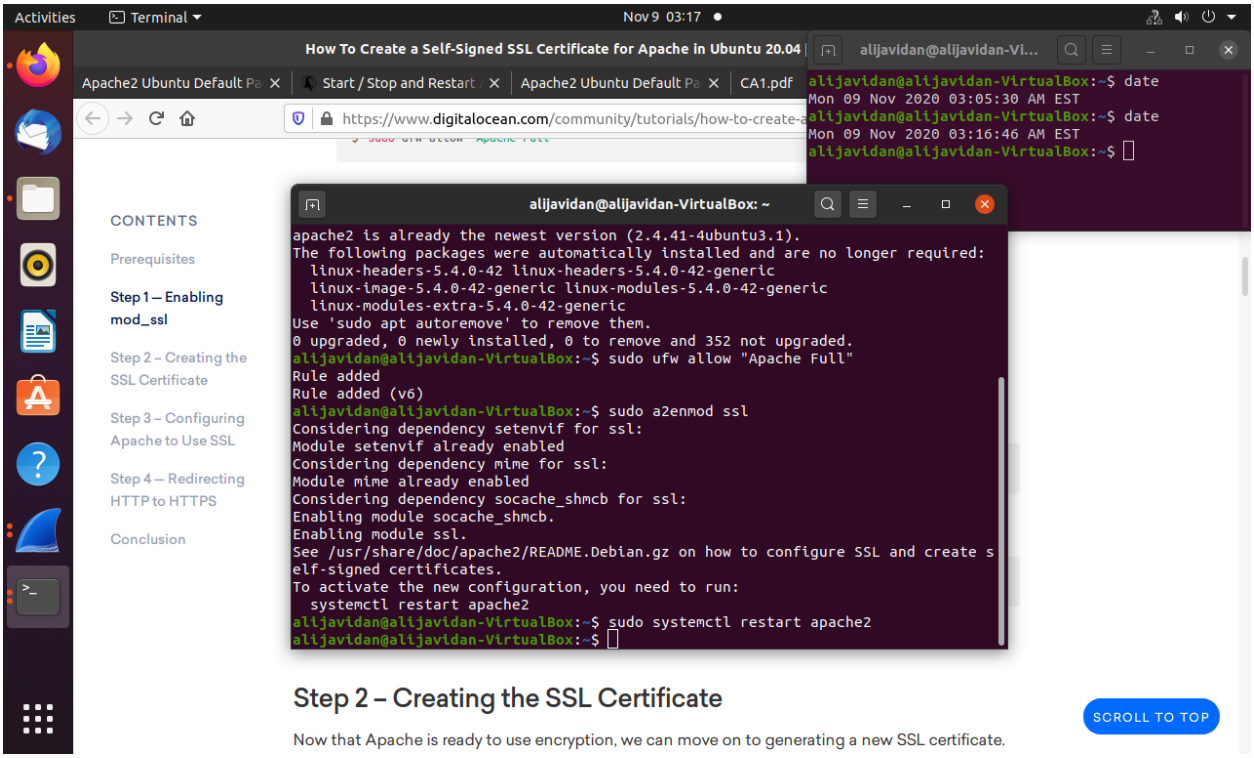
Since we have a ufw firewall set up, we need to open up the http and https ports:



2. Set up a HTTPS server with user authentication

Step 1 — Enabling mod_ssl

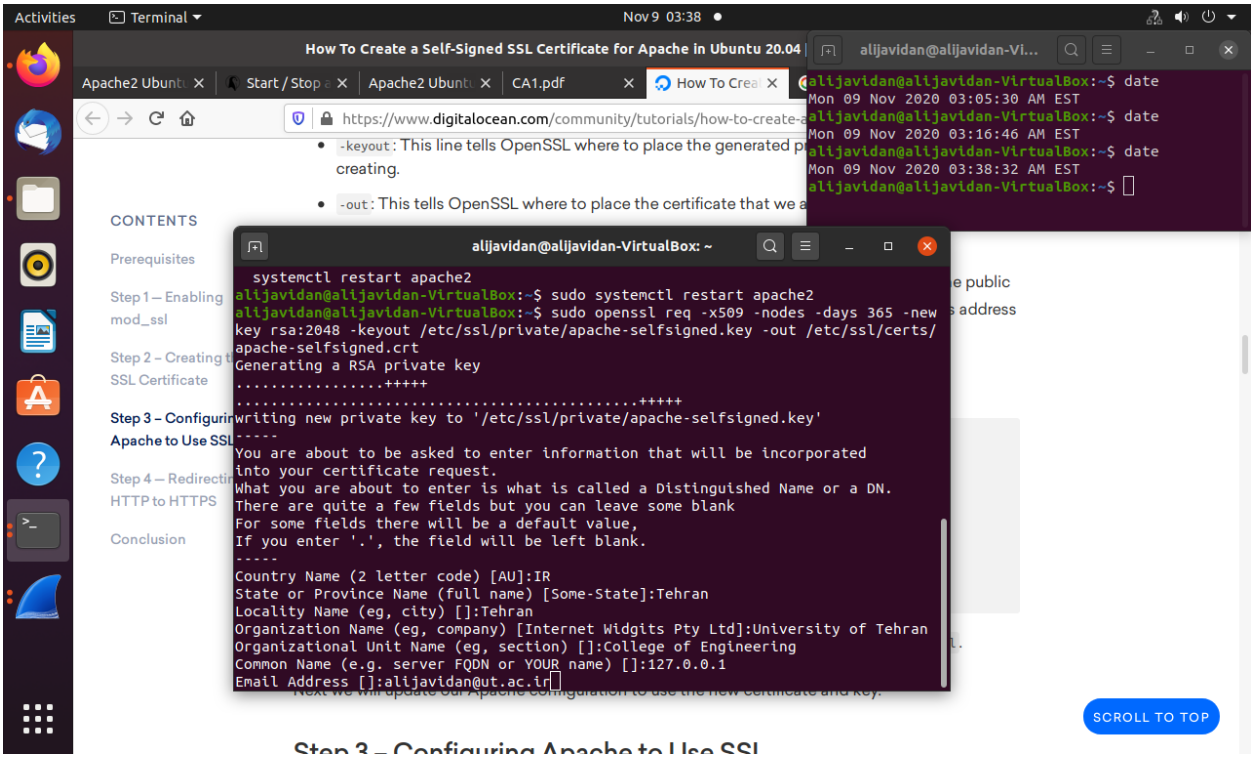
Before we can use any SSL certificates, we first have to enable mod_ssl, an Apache module that provides support for SSL encryption.
We Restart Apache to activate the module.



Step 2 – Creating the SSL Certificate

Now that Apache is ready to use encryption, we can move on to generating a new SSL certificate. The certificate will store some basic information about our site, and will be accompanied by a key file that allows the server to securely handle encrypted data.

After we enter the command, we will be taken to a prompt where we can enter information about our website.

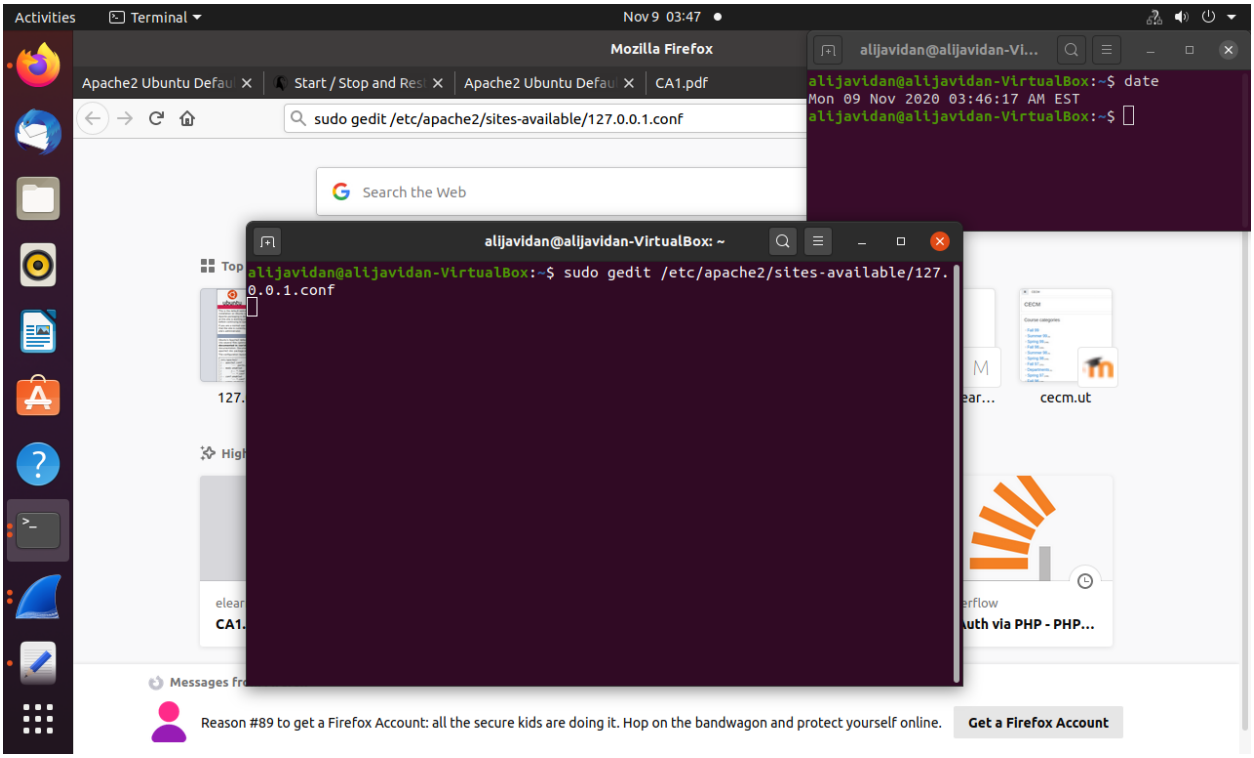


Both of the files we created will be placed in the appropriate subdirectories under /etc/ssl.

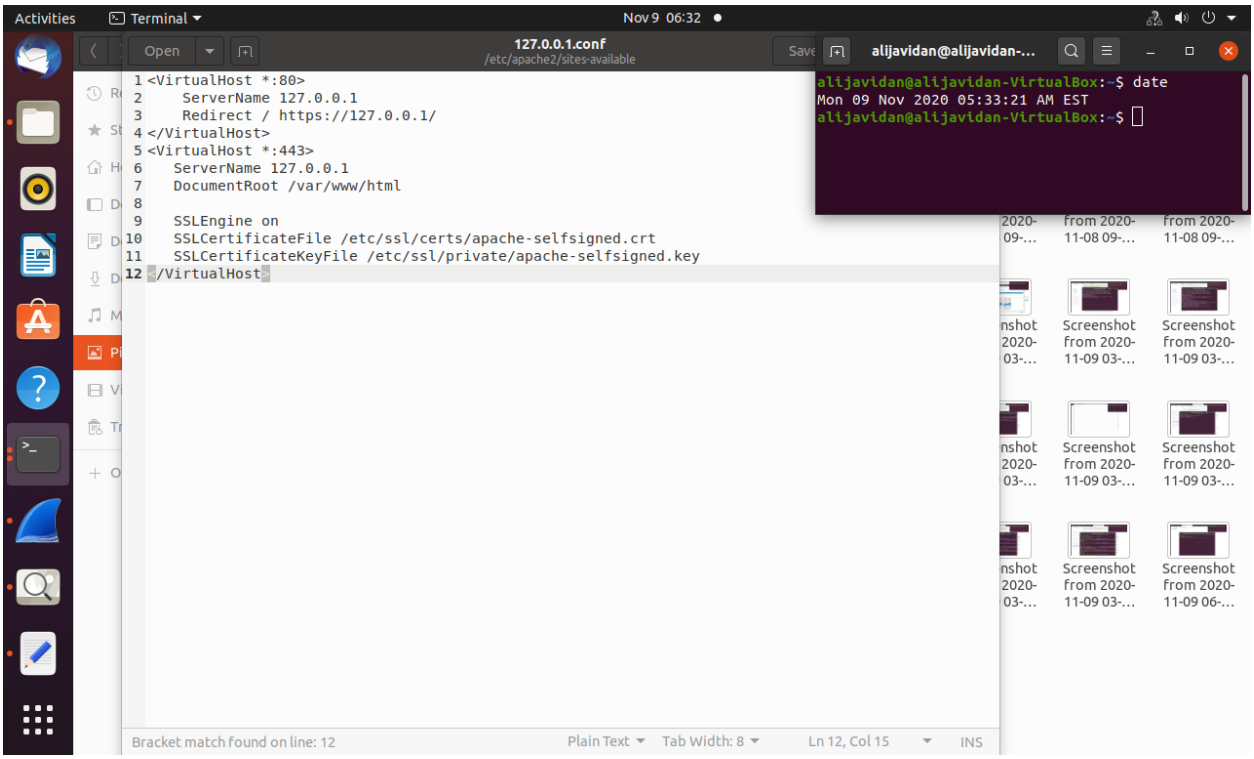
Step 3 – Configuring Apache to Use SSL

Now that we have our self-signed certificate and key available, we need to update our Apache configuration to use them. On Ubuntu, we can place new Apache configuration files (they must end in .conf) into /etc/apache2/sites-available/and they will be loaded the next time the Apache process is reloaded or restarted.

We will create a new minimal configuration file.
We open a new file in the /etc/apache2/sites-available directory,



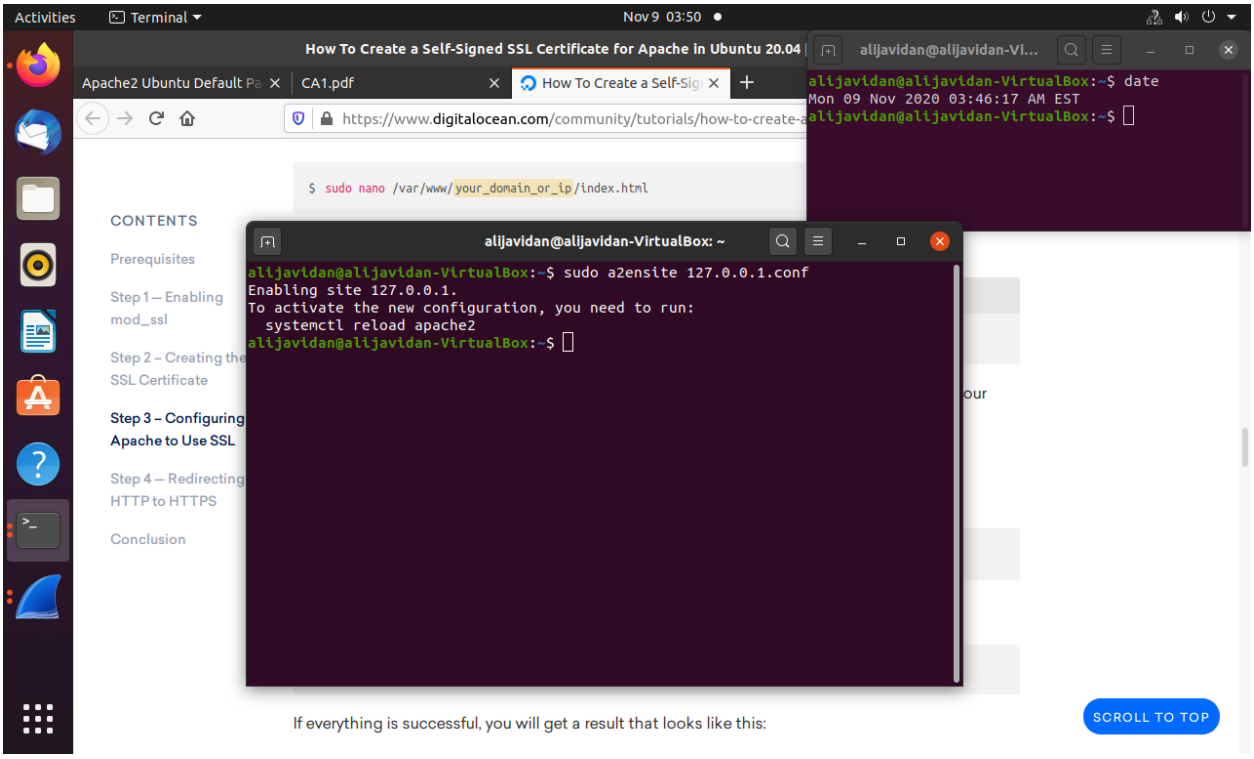
and paste in the following minimal VirtualHost configuration:



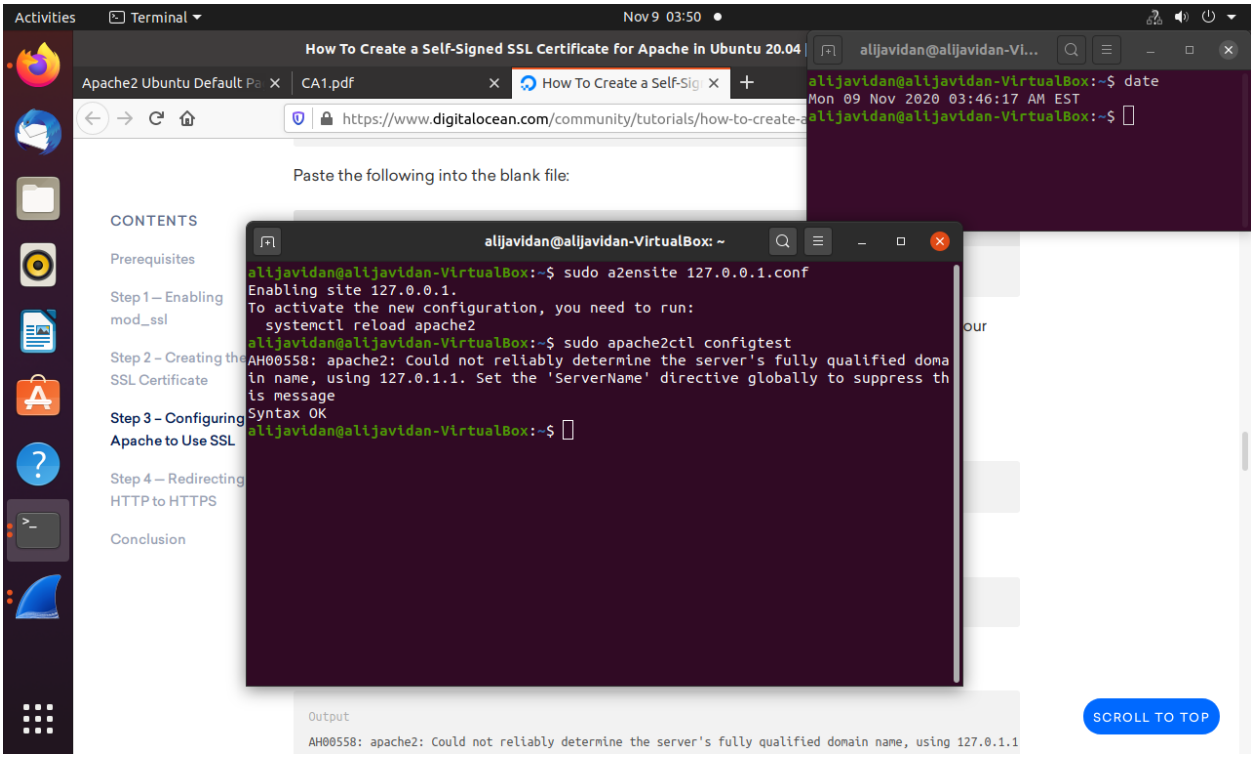
We should make sure ServerName matches the Common Name we chose when making the certificate.

The remaining lines specify a DocumentRoot directory to serve files from, and the SSL options needed to point Apache to our newly-created certificate and key.

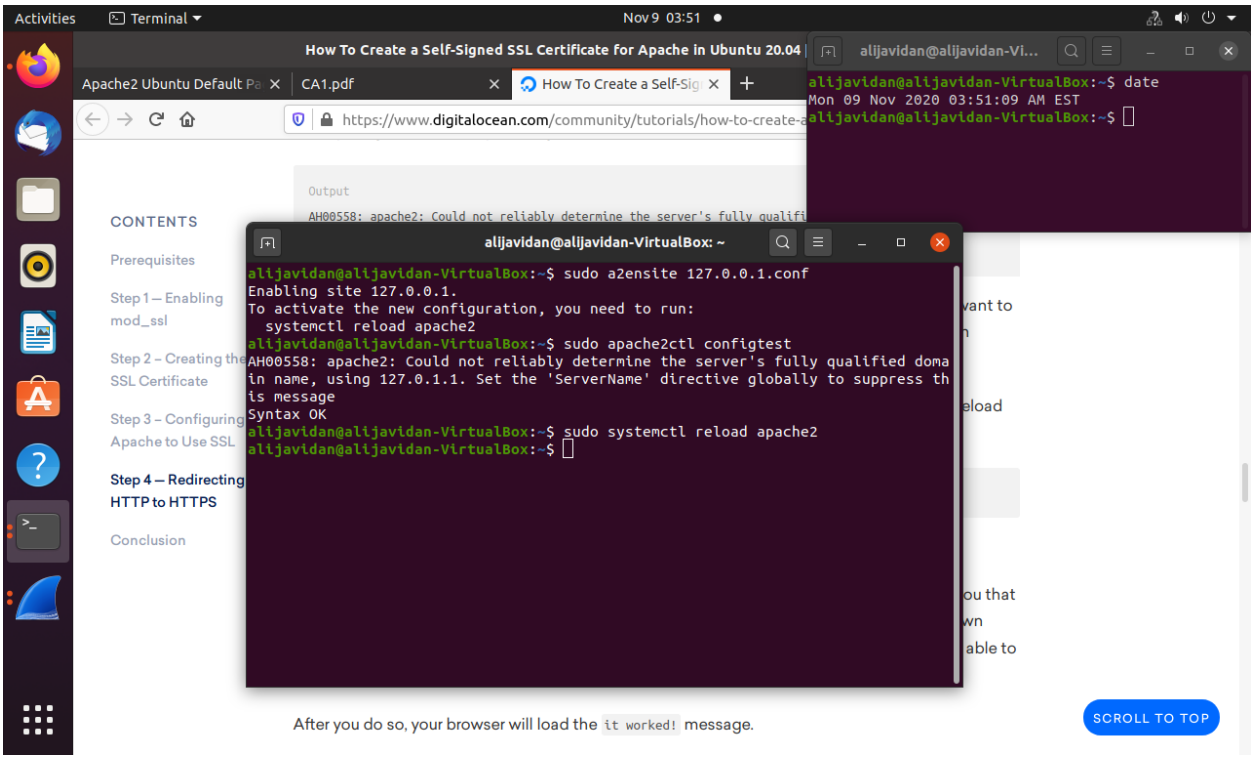
Next, let's test for configuration errors:



If everything is successful, you will get a result that looks like this:

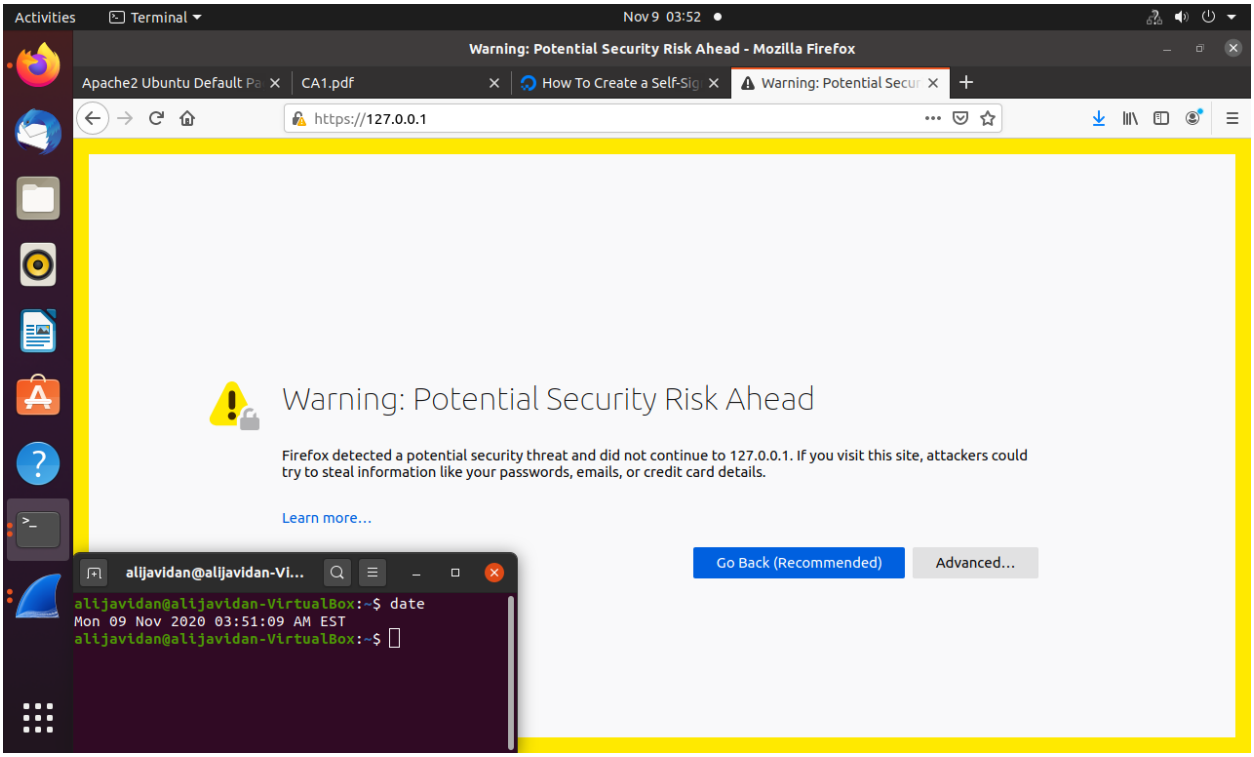


Now that output has Syntax OK in it, our configuration file has no syntax errors. We can safely reload Apache to implement our changes.

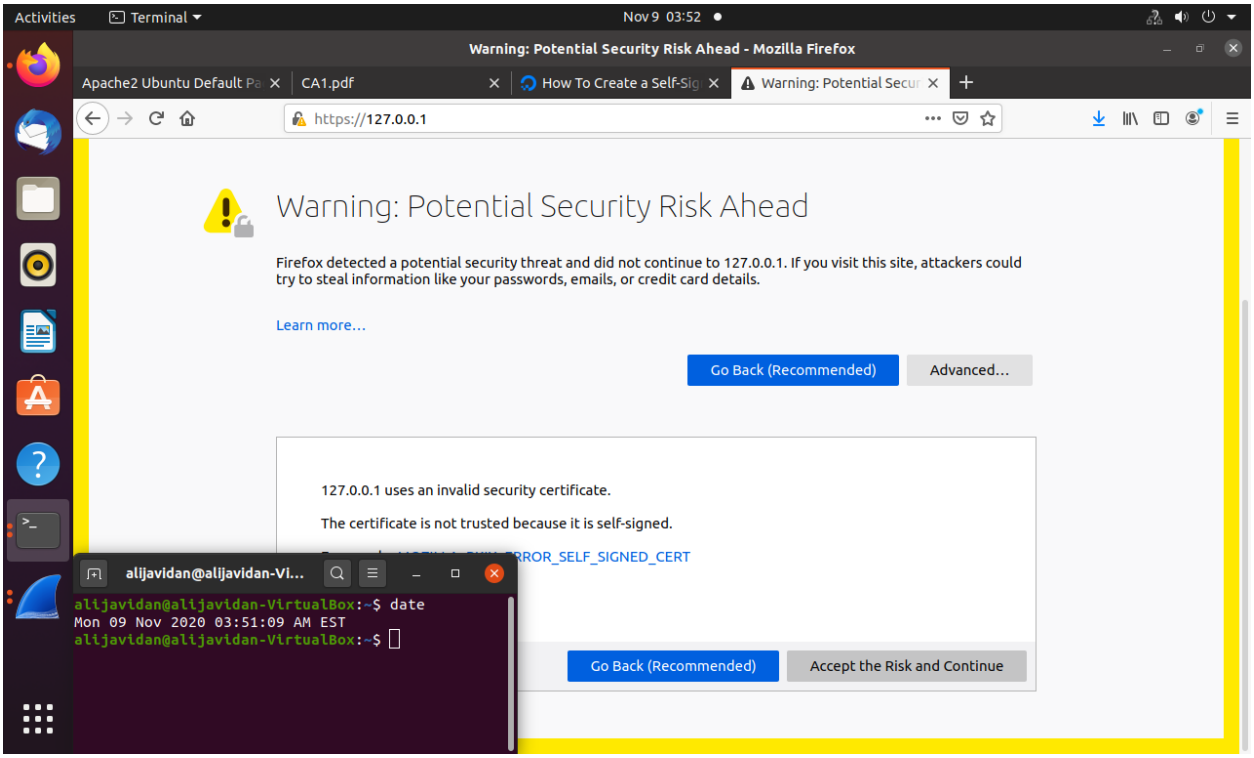


Now we load our site in a browser.

We should see an error. This is normal for a self-signed certificate! The browser is warning us that it can't verify the identity of the server, because our certificate is not signed by any of its known certificate authorities. For testing purposes and personal use this can be fine.

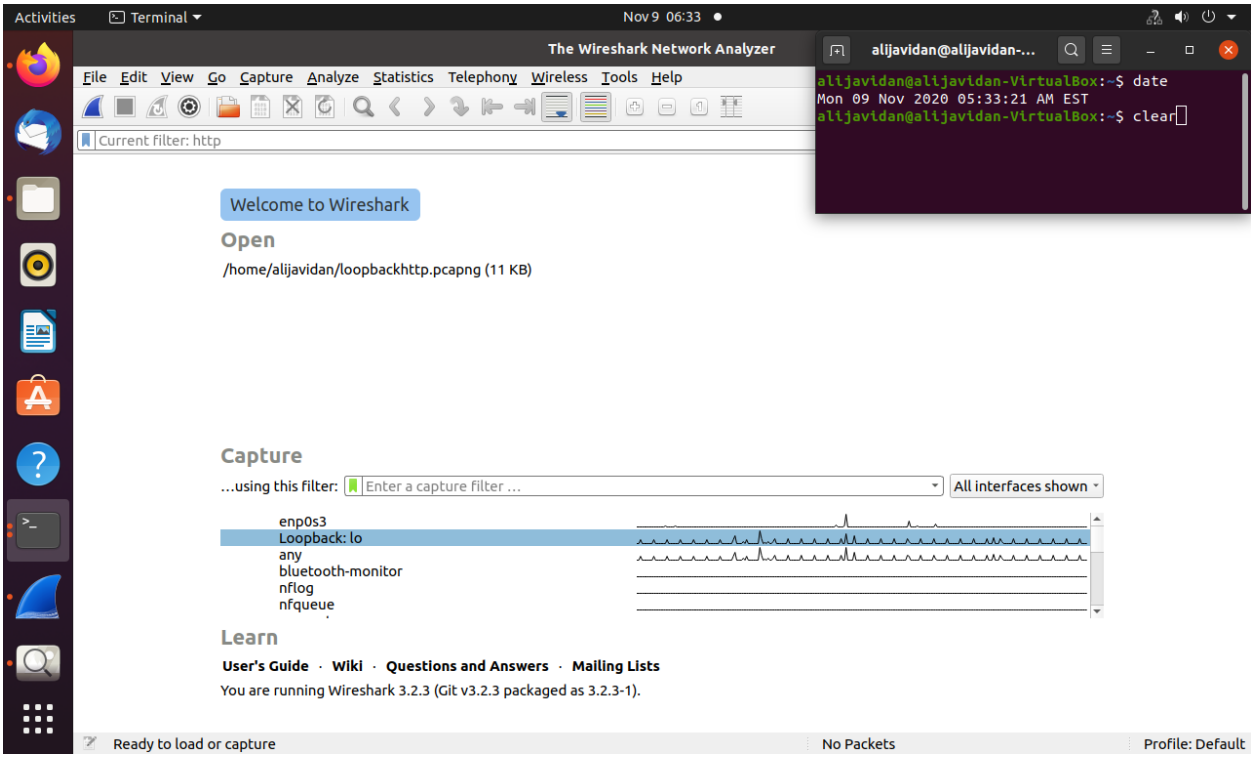


We should be able to click through to advanced or more information and choose to proceed.

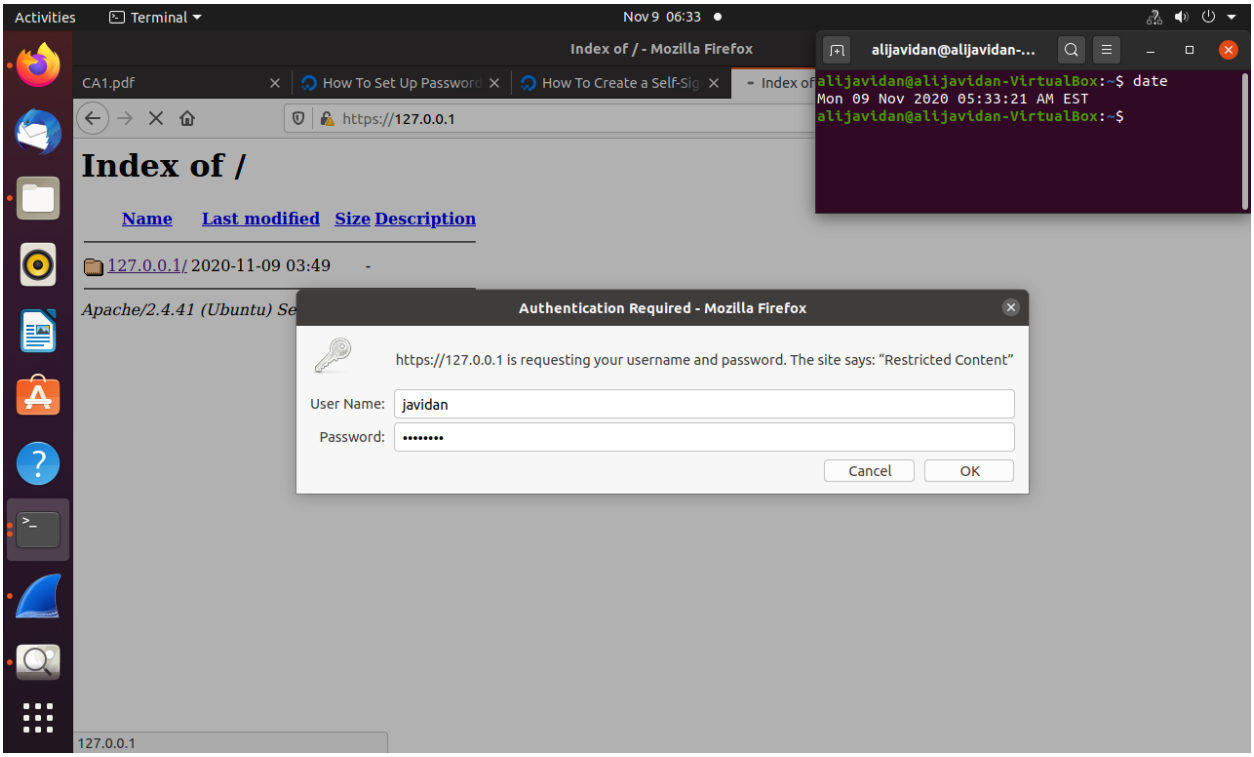


Wireshark

We start Wireshark capture,

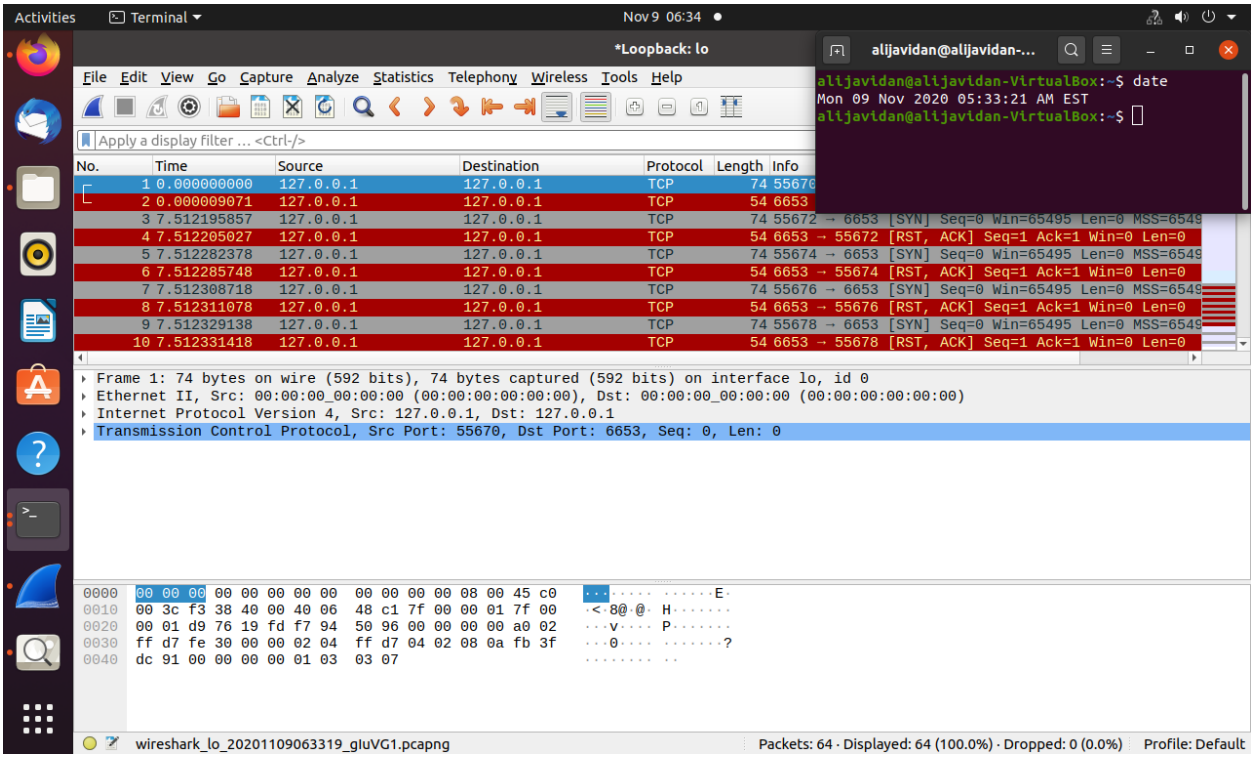


and login to Apache homepage with our creditioanlal (javidan as User Name) through authentication procedure we have previously built up.



Now we stop Wireshark capture,

but this time we could no longer extarct any creditionals from captured packets, *since in https they are encrypted and not sent as plain-text like what was like in http before.*


Activities
Terminal
Nov 9 06:34
*Loopback: lo
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
No. Time Source Destination Protocol Length Info
1 0.000000000 127.0.0.1 127.0.0.1 TCP 74 55670
2 0.000009071 127.0.0.1 127.0.0.1 TCP 54 6653
3 7.512195857 127.0.0.1 127.0.0.1 TCP 74 55672 → 6653 [SYN] Seq=0 Win=65495 Len=0 MSS=6549
4 7.512205027 127.0.0.1 127.0.0.1 TCP 54 6653 → 55672 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5 7.512282378 127.0.0.1 127.0.0.1 TCP 74 55674 → 6653 [SYN] Seq=0 Win=65495 Len=0 MSS=6549
6 7.512285748 127.0.0.1 127.0.0.1 TCP 54 6653 → 55674 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
7 7.512308718 127.0.0.1 127.0.0.1 TCP 74 55676 → 6653 [SYN] Seq=0 Win=65495 Len=0 MSS=6549
8 7.512311078 127.0.0.1 127.0.0.1 TCP 54 6653 → 55676 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
9 7.512329138 127.0.0.1 127.0.0.1 TCP 74 55678 → 6653 [SYN] Seq=0 Win=65495 Len=0 MSS=6549
10 7.512331418 127.0.0.1 127.0.0.1 TCP 54 6653 → 55678 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface lo, id 0
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 55670, Dst Port: 6653, Seq: 0, Len: 0
0000 00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 c0
0010 00 3c f3 38 40 00 40 06 48 c1 7f 00 00 01 7f 00
0020 00 01 d9 76 19 fd f7 94 50 96 00 00 00 00 a0 02
0030 ff d7 fe 30 00 00 02 04 ff d7 04 02 08 0a fb 3f
0040 dc 91 00 00 00 00 01 03 03 07
wireshark_lo_20201109063319_gIuVG1.pcapng
Packets: 64 · Displayed: 64 (100.0%) · Dropped: 0 (0.0%)
Profile: Default
alijavidan@alijavidan-...
alijavidan@alijavidan-VirtualBox:~$ date
Mon 09 Nov 2020 05:33:21 AM EST
alijavidan@alijavidan-VirtualBox:~$