

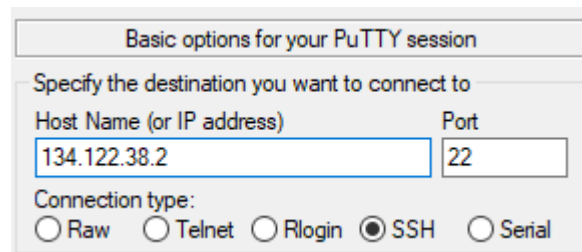
COSC 419

Lab 1: Getting Started with Remote Servers

For our first lab, we'll be starting the configuration of our remote servers. We'll do some initial configuration, and then get to installing the first major components of the software stack: the Apache web server, and Python.

Connecting to Your Remote Server

We'll want to begin by establishing a **secure shell** (SSH) connection to our remote server. If you're on Windows, I would recommend [you download the PuTTY client](#). It should prompt you for a hostname or IP address – enter the IP address sent to you for your remote server.



Once you've done that, hit the "Open" button. You'll be prompted to enter a username and password. You should use "root" as your username, and then use the password provided to you.

```
root@EXAMPLE-SERVER:~  
login as: root  
root@134.122.38.2's password:  
Access denied  
root@134.122.38.2's password:
```

Once you've logged in successfully, you should be connected and currently in your user home (~) folder.

Initial Server Preparation

Now that you've connected successfully to your server, we can begin to configure and prepare our virtual server to act as a web server. We'll first want to begin by updating any packages that may be out of date. To do this, we'll use the **yum** package manager:

```
yum update
```

It will then most likely prompt you to accept the package updates. Enter 'y' to accept:

```
Transaction Summary
=====
Install    5 Packages
Upgrade   39 Packages

Total download size: 176 M
Is this ok [y/N]: y
```

This process may take some time as packages are downloaded. It may appear to hang at the end of the process, but it should eventually complete. Once you've updated all of your current packages, we'll go ahead and install a new package: the **nano** text editor. Nano is a simple, no-frills text editor which is good for when you want to quickly edit a file directly in the SSH terminal. We can install it using **yum install**:

```
yum install nano
```

Again, you will be prompted to confirm the installation. Make sure you select 'y'. Once the installation is complete, you should be able to open a new file (or an existing file) for editing using the following command:

```
nano <filename>
```

```
Running transaction
Preparing      :
Installing     : nano-2.9.8-1.el8.x86_64
Running scriptlet: nano-2.9.8-1.el8.x86_64
Verifying      : nano-2.9.8-1.el8.x86_64

Installed:
  nano-2.9.8-1.el8.x86_64

Complete!
[root@EXAMPLE-SERVER ~]# nano myFile.txt
```

Now that we've done that, we'll move on to creating a new user. It is usually considered bad practice to use the **root** user for most tasks, as the root user account can pose a great security threat if it is compromised. For this reason (and so you can set your own password), we'll create a new account. Use the following command:

```
adduser <username>
```

Where **<username>** is whatever you want your username to be. Make sure it's something you'll remember. This will create a user with the specified username. Next, we'll set the password for the user using the **passwd** command:

```
passwd <username>
```

```
[root@EXAMPLE-SERVER ~]# adduser frit
[root@EXAMPLE-SERVER ~]# passwd frit
Changing password for user frit.
New password: 
```

Make sure you enter the correct username (for the new user you just created). It will then prompt you to enter a password. Try to ensure you use a good password – you don't want anyone breaking into your server, and you don't want to forget your password. Once you've done that, it'll confirm that the password has been changed.

Now we've got our new user, but they don't have any permissions yet. We'll want this user to have **sudo** (Super User Do) permissions so that we can edit files and install software. To do this, we'll need to add the user to the **wheel** group. The wheel group denotes users that have sudo permissions. We can add the user to this group with the following command:

```
usermod -aG wheel <username>
```

The **-aG** flags stand for **append Group**. Here's an example of me adding my new user 'frit' as a sudo user:

```
[root@EXAMPLE-SERVER ~]# usermod -aG wheel frit
```

We can switch to our newly created user using the **su** command:

```
su <username>
```

Try logging out (or just closing your PuTTY connection) and then logging in using your newly created account to verify that it works properly before continuing onto the next step. Once your new user works, you should disable the ability to remotely log in using the **root** user. We can do this by editing the SSH config file, located at **/etc/ssh/sshd_config**:

```
sudo nano /etc/ssh/sshd_config
```

Look for a line that says 'PermitRootLogin yes' and change it to 'PermitRootLogin no'. Then use CTRL+X to close the file. It will prompt you to ask if you wish to save your changes. Enter 'y', and then hit enter again to save under the same filename. Now, we'll restart the SSH service so that our changes can take effect:

```
sudo service sshd restart
```

```
[frit@EXAMPLE-SERVER ~]$ sudo service sshd restart  
Redirecting to /bin/systemctl restart sshd.service
```

Installing Apache and Python

Now that we've finished updating our packages and securing our root account, we can move onto installing the first major packages in the software stack. We'll start by install Apache:

```
sudo yum install httpd
```

In the RedHat/Fedora/CentOS ecosystem, Apache is usually referred to as **httpd**. As before, confirm that you want to install the packages, and wait for it to complete the installation. It should be fairly quick. Once you've done that, we'll want to do two things: enable Apache on

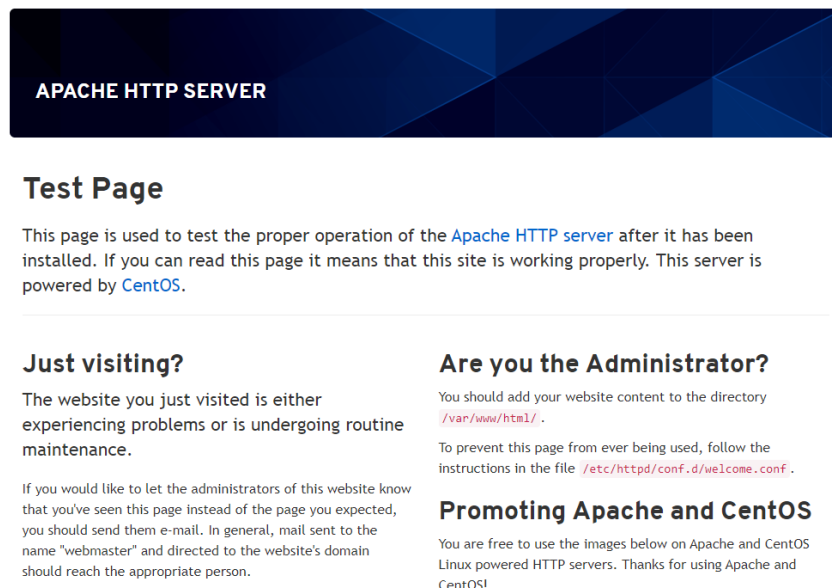
startup, so that when the server restarts our web server automatically starts again, and then manually start Apache. We can enable Apache at startup using this command:

```
sudo systemctl enable httpd
```

And then we can manually start Apache with a very similar command:

```
sudo systemctl start httpd
```

You can verify that Apache is installed correctly and currently running by entering the IP address for your server into a web browser. It should load up the default Apache test page, indicating that Apache is working properly.



Next up, we'll install Python. This is fairly straightforward, but make sure you're installing the correct version – Python 3, **not Python 2!** Use the following command:

```
sudo yum install python3
```

You should see that it will install Python 3.6, along with the Python 3 setup tools and the Pip utility:

```
Installed:
python3-pip-9.0.3-16.el8.noarch
python3-setuptools-39.2.0-5.el8.noarch
python36-3.6.8-2.module_el8.1.0+245+c39af44f.x86_64
```

Once you've done that, we're ready to create our first very basic Python web application.

Creating A Basic Application

We don't necessarily need a framework to build a very simple dynamic web application, we can do it with just standard Python and Apache. First, we'll need to create an **index.html** page. If an

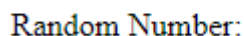

index page is found in the Apache web root directory, it will automatically be served by Apache. The web directory root can be found at **/var/www/html**. Run the following command:

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

Now, create a very basic HTML web page. The following HTML code is given as an example, but you can add your own flavor if you feel like it:

```
<html>
<body>
<h1>Frit's Page!</h1>
<p>Random Number: %</p>
</body>
</html>
```

Save and close the file. Visiting your server's IP address in a web browser should now show a page like this:



Now, navigate to the web root directory using the **cd** command, and then take a copy of the index file using the **cp** command:

```
sudo cp index.html index_original
```

You should now have two copies of the index file, one named **index.html**, and the other named **index_original**. We'll write a simple Python application which will read the **index_original** file, and then use it as a template to overwrite the **index.html** file, changing our web page.

First, create your Python file (in the web root directory):

```
sudo nano myapp.py
```

The following program will open the **index_original** file, modify it, and then save the modifications to the **index.html** file. It will do this every 10 seconds for as long as it is left running. Pay attention to the comments for an explanation of what the code is doing. Note that Python uses whitespace (spaces and tabs) to denote "blocks" of code, rather than brackets. Everything indented after the **while** statement is **inside the while loop**. Hash characters '#' denote a commented line.

```
#Import Dependencies
from random import randrange
import time
```

```

#Loop Infinitely
while True:
    #Open the original index file in "read" mode
    file_original = open("index_original", "r")
    #Read the contents of the file into a variable as a string
    html = file_original.read()
    #Replace the percent '%' character with a number from 0 to 9
    html = html.replace("%", str(randrange(10)))
    #Close the original file
    file_original.close()
    #Open the index.html file in "write" mode
    file_output = open("index.html", "w")
    #Write our edited HTML to the index.html file
    file_output.write(html)
    #Close the index.html file
    file_output.close()
    #Sleep for 10 seconds before looping again
    time.sleep(10)

```

Exit and save the changes to your Python file. Now, you can try running your Python file using the python3 package:

```
sudo python3 myapp.py
```

Now try visiting the page while our script is running. If you refresh the page after waiting for 10 seconds, you should see the number on the page change:

Frit's Page!

Random Number: 2

You may notice that we can now no longer use the SSH terminal, as our Python script is still running, and will be running forever. Use CTRL-C to forcibly close the script, returning the SSH terminal.

Congratulations, you've built a very basic full-stack web application. It has an OS (CentOS), a web server (Apache), a backend (Python), and our frontend (the HTML page). This application isn't particularly useful by itself, however, as we have no way of capturing or processing user input. In Lab 2, we'll see how we can directly interface Python with Apache using WSGI, rather than using Python to modify HTML files on the fly.

Submission

There is no required submission for this lab. You will be marked directly on the state of your server, using the following schema:

Marks	Item
1 Mark	Yum update
2 Marks	Setup new user and disabled root login
2 Marks	Installed Apache and Python
2 Marks	Set up basic Python Application