Name: Maravilla, Keith Dominic T.	**Date Performed: August 18, 2022**
Course/Section: CPE232/CPE31S23	**Date Submitted: August 22, 2022**
Instructor: Engr. Jonathan V. Taylar	**Semester and SY: 1st Sem - 3rd Year**

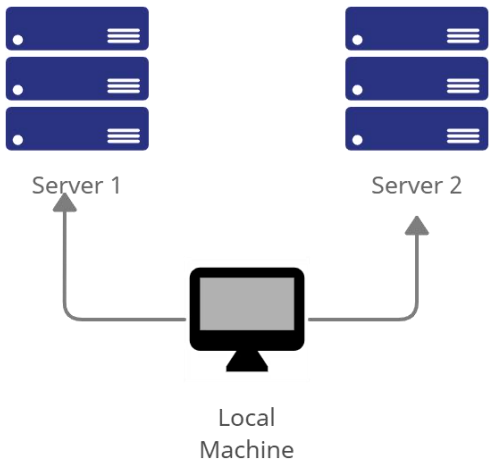
Activity 1: Configure Network using Virtual Machines

1. **Objectives:**
   1.1. Create and configure Virtual Machines in Microsoft Azure or VirtualBox
   1.2. Set-up a Virtual Network and Test Connectivity of VMs

2. **Discussion:**

Network Topology:

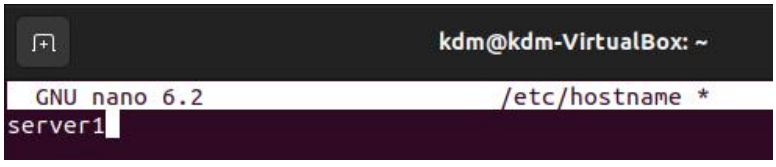
Assume that you have created the following network topology in Virtual Machines, *provide screenshots for each task*. (Note: *it is assumed that you have the prior knowledge of cloning and creating snapshots in a virtual machine*).



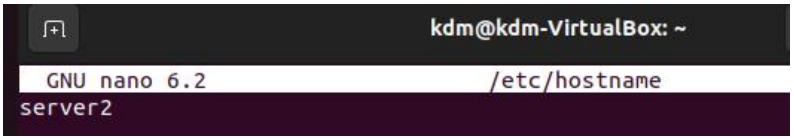
Task 1: Do the following on Server 1, Server 2, and Local Machine. In editing the file using nano command, press control + O to write out (save the file). Press enter when asked for the name of the file. Press control + X to end.

1. Change the hostname using the command *sudo nano /etc/hostname*

1.1 Use server1 for Server 1



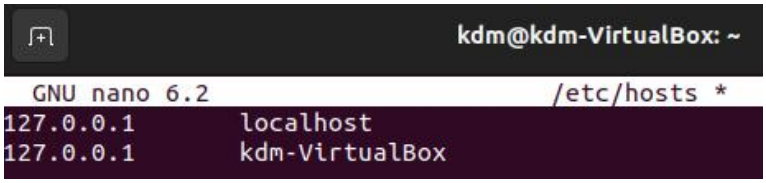
1.2 Use server2 for Server 2



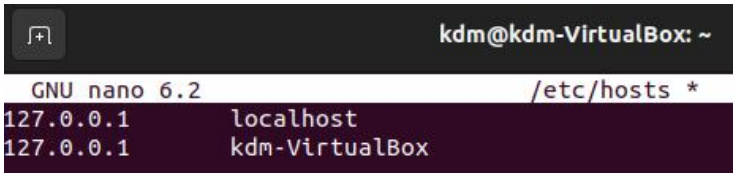
1.3 Use workstation for the Local Machine

2. Edit the hosts using the command *sudo nano /etc/hosts*. Edit the second line.

2.1 Type 127.0.0.1 server 1 for Server 1



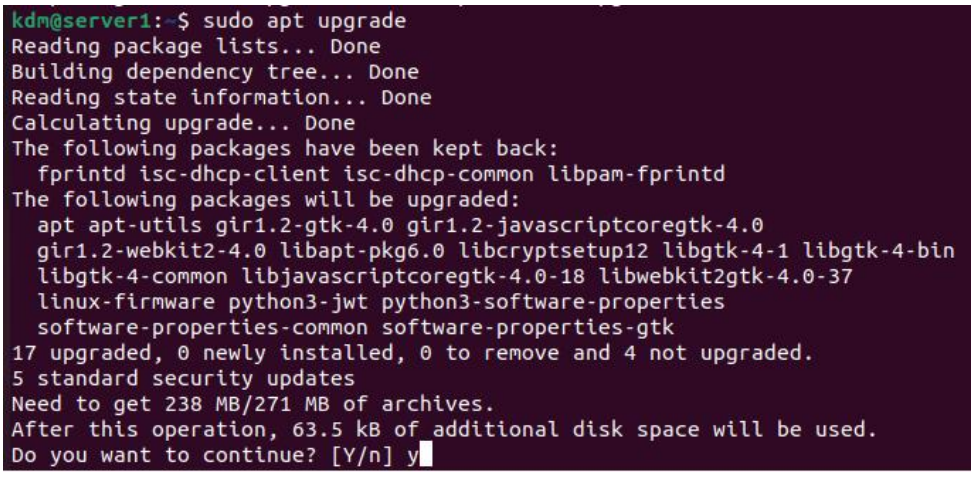
2.2 Type 127.0.0.1 server 2 for Server 2



2.3 Type 127.0.0.1 workstation for the Local Machine

Task 2: Configure SSH on Server 1, Server 2, and Local Machine. Do the following:

1. Upgrade the packages by issuing the command *sudo apt update* and *sudo apt upgrade* respectively.



Initializing update and upgrade of packages in Server 1

(NOTE: This will be the same output for Server 2)

2. Install the SSH server using the command *sudo apt install openssh-server*.

```
kdm@server1:~$ sudo apt install openssh-server
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  ncurses-term openssh-sftp-server ssh-import-id
Suggested packages:
  molly-guard monkeysphere ssh-askpass
The following NEW packages will be installed:
  ncurses-term openssh-server openssh-sftp-server ssh-import-id
0 upgraded, 4 newly installed, 0 to remove and 4 not upgraded.
Need to get 751 kB of archives.
After this operation, 6,046 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://ph.archive.ubuntu.com/ubuntu jammy/main amd64 openssh-sftp-server
amd64 1:8.9p1-3 [38.8 kB]
Get:2 http://ph.archive.ubuntu.com/ubuntu jammy/main amd64 openssh-server amd64
 1:8.9p1-3 [434 kB]
Get:3 http://ph.archive.ubuntu.com/ubuntu jammy/main amd64 ncurses-term all 6.3
-2 [267 kB]
Get:4 http://ph.archive.ubuntu.com/ubuntu jammy/main amd64 ssh-import-id all 5.
```

Installing the SSH server in Server 1

(NOTE: This will be the same output for Server 2)

3. Verify if the SSH service has started by issuing the following commands:
   3.1 *sudo service ssh start*
   3.2 *sudo systemctl status ssh*

```
kdm@server1:~$ sudo service ssh start
kdm@server1:~$ sudo systemctl status ssh
● ssh.service - OpenBSD Secure Shell server
     Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset:
     Active: active (running) since Mon 2022-08-22 21:10:55 PST; 4min 37s ago
       Docs: man:sshd(8)
             man:sshd_config(5)
   Main PID: 15341 (sshd)
      Tasks: 1 (limit: 1080)
     Memory: 1.7M
        CPU: 25ms
     CGroup: /system.slice/ssh.service
             └─15341 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups"

Aug 22 21:10:55 server1 systemd[1]: Starting OpenBSD Secure Shell server...
Aug 22 21:10:55 server1 sshd[15341]: Server listening on 0.0.0.0 port 22.
Aug 22 21:10:55 server1 sshd[15341]: Server listening on :: port 22.
Aug 22 21:10:55 server1 systemd[1]: Started OpenBSD Secure Shell server.
lines 1-16/16 (END)
```

Verifying the SSH service in Server 1

(NOTE: This will be the same output for Server 2)

4. Configure the firewall to all port 22 by issuing the following commands:
   4.1 *sudo ufw allow ssh*
   4.2 *sudo ufw enable*
   4.3 *sudo ufw status*

```
kdm@server1:~$ sudo ufw allow ssh
Rules updated
Rules updated (v6)
kdm@server1:~$ sudo ufw enable
Firewall is active and enabled on system startup
kdm@server1:~$ sudo ufw status
Status: active

To                         Action      From
--                         ------      ----
22/tcp                     ALLOW       Anywhere
22/tcp (v6)                ALLOW       Anywhere (v6)
```

Activating or enabling firewall in port 22 in Server 1
(NOTE: This will be the same output for Server 2)

Task 3: Verify network settings on Server 1, Server 2, and Local Machine. On each device, do the following:

1. Record the ip address of Server 1, Server 2, and Local Machine. Issue the command *ifconfig* and check network settings. Note that the ip addresses of all the machines are in this network 192.168.56.XX.
   1.1 Server 1 IP address: 192.168.56.103

```
kdm@server1:~$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::d688:b367:c904:1d28  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:cf:f2:10  txqueuelen 1000  (Ethernet)
        RX packets 213  bytes 227825 (227.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 156  bytes 16347 (16.3 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp0s8: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.56.103  netmask 255.255.255.0  broadcast 192.168.56.255
        inet6 fe80::9527:217:3c59:c58b  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:4b:83:d9  txqueuelen 1000  (Ethernet)
```

   1.2 Server 2 IP address: 192.168.56.104

```
kdm@server2:~$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::a4c1:9ac:ae21:d6bf  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:12:6c:d1  txqueuelen 1000  (Ethernet)
        RX packets 142749  bytes 206958658 (206.9 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 31728  bytes 1946261 (1.9 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp0s8: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.56.104  netmask 255.255.255.0  broadcast 192.168.56.255
        inet6 fe80::7477:472f:2185:4563  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:da:ae:0c  txqueuelen 1000  (Ethernet)
```

1.3 Server 3 IP address: 192.168.56.___

2. Make sure that they can ping each other.

2.1 Connectivity test for Local Machine 1 to Server 1: ☒ Successful ☐ Not Successful

```
keith@KEITHMARAVILLA MINGW64 ~
$ ping 192.168.56.103

Pinging 192.168.56.103 with 32 bytes of data:
Reply from 192.168.56.103: bytes=32 time<1ms TTL=64
Reply from 192.168.56.103: bytes=32 time<1ms TTL=64
Reply from 192.168.56.103: bytes=32 time<1ms TTL=64
Reply from 192.168.56.103: bytes=32 time<1ms TTL=64

Ping statistics for 192.168.56.103:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

2.2 Connectivity test for Local Machine 1 to Server 2: ☒ Successful ☐ Not Successful

```
keith@KEITHMARAVILLA MINGW64 ~
$ ping 192.168.56.104

Pinging 192.168.56.104 with 32 bytes of data:
Reply from 192.168.56.104: bytes=32 time<1ms TTL=64
Reply from 192.168.56.104: bytes=32 time<1ms TTL=64
Reply from 192.168.56.104: bytes=32 time<1ms TTL=64
Reply from 192.168.56.104: bytes=32 time<1ms TTL=64

Ping statistics for 192.168.56.104:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

2.3 Connectivity test for Server 1 to Server 2: ☒ Successful ☐ Not Successful

```
kdm@server1:~$ ping 192.168.56.104
PING 192.168.56.104 (192.168.56.104) 56(84) bytes of data.
64 bytes from 192.168.56.104: icmp_seq=1 ttl=64 time=0.907 ms
64 bytes from 192.168.56.104: icmp_seq=2 ttl=64 time=0.919 ms
64 bytes from 192.168.56.104: icmp_seq=3 ttl=64 time=0.806 ms
64 bytes from 192.168.56.104: icmp_seq=4 ttl=64 time=0.724 ms
64 bytes from 192.168.56.104: icmp_seq=5 ttl=64 time=0.320 ms
64 bytes from 192.168.56.104: icmp_seq=6 ttl=64 time=0.873 ms
64 bytes from 192.168.56.104: icmp_seq=7 ttl=64 time=0.943 ms
64 bytes from 192.168.56.104: icmp_seq=8 ttl=64 time=0.731 ms
64 bytes from 192.168.56.104: icmp_seq=9 ttl=64 time=0.955 ms
^C
--- 192.168.56.104 ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time 8059ms
rtt min/avg/max/mdev = 0.320/0.797/0.955/0.187 ms
kdm@server1:~$
```

Task 4: Verify SSH connectivity on Server 1, Server 2, and Local Machine.

1. On the Local Machine, issue the following commands:

1.1 ssh username@ip_address_server1 for example, *ssh jvtaylar@192.168.56.120*

```
keith@KEITHMARAVILLA MINGW64 ~
$ ssh kdm@192.168.56.103
The authenticity of host '192.168.56.103 (192.168.56.103)' can't be established
ED25519 key fingerprint is SHA256:rI/N3gsqcQjEHeAqrMm7/M4s09P2LqMS/tEHJXMDYI8.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.56.103' (ED25519) to the list of known host
```

Accessing SSH server of Server 1 through local machine

1.2 Enter the password for server 1 when prompted

```
keith@KEITHMARAVILLA MINGW64 ~
$ ssh kdm@192.168.56.103
kdm@192.168.56.103's password:
Welcome to Ubuntu 22.04.1 LTS (GNU/Linux 5.15.0-46-generic x86_64)

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

0 updates can be applied immediately.


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

kdm@server1:~$
```

Entering the password for server 1 using local machine

1.3 Verify that you are in server 1. The user should be in this format user@server1. For example, *jvtaylar@server1*

```
kdm@server1:~$
```

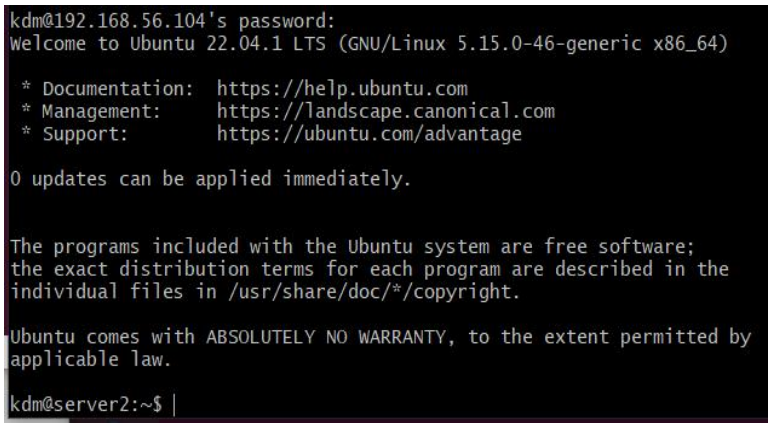
Verification that local machine is connected to server 1

2. Logout of Server 1 by issuing the command *control + D.*

```
kdm@server1:~$
logout
Connection to 192.168.56.103 closed.
```

Logging out to server 1 in a local machine

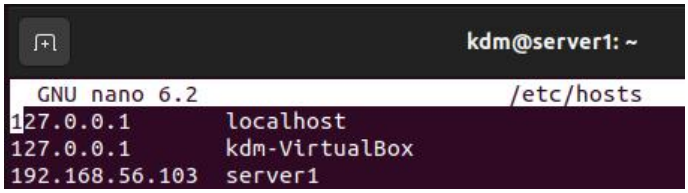
3. Do the same for Server 2.



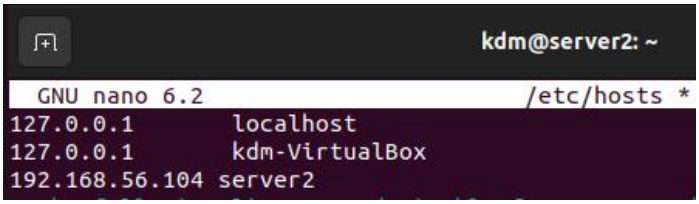
Verification that local machine is connected to server 2

4. Edit the hosts of the Local Machine by issuing the command *sudo nano /etc/hosts.* Below all texts type the following:

4.1 IP_address server 1 (provide the ip address of server 1 followed by the hostname)



4.2 IP_address server 2 (provide the ip address of server 2 followed by the hostname)



4.3 Save the file and exit.

5. On the local machine, verify that you can do the SSH command but this time, use the hostname instead of typing the IP address of the servers. For example, try to do *ssh jvtaylar@server1*. Enter the password when prompted. Verify that you have entered Server 1. Do the same for Server 2.

```
keith@KEITHMARAVILLA MINGW64 ~
$ ssh kdm@server2
kdm@server2's password:
Welcome to Ubuntu 22.04.1 LTS (GNU/Linux 5.15.0-46-generic x86_64)

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

0 updates can be applied immediately.

Last login: Mon Aug 22 22:14:05 2022 from fe80::3149:8f1:6fd9:3c2b%enp0s8
kdm@server2:~$
```

Issuing and verifying SSH command using hostname

Reflections:

Answer the following:

1. How are we able to use the hostname instead of IP address in SSH commands?

   We were able to use the hostname in SSH commands instead of using IP address because there is an addition of hostname associated to the IP address of a certain server in /etc/hosts. In this way, we could access and issue SSH commands of a server by simply typing the hostname since it was registered or added in the registry or file containing the IP addresses along with its server or unit.

2. How secured is SSH?

   Secure Shell or SSH is highly secured as it uses encryption and authentication to all available connections. Another reason why SSH is highly secured is due to remote managing of SSH clients. Using secure or SSH keys/credentials for SSH connections basically tells us that accessing SSH connection between devices is not easy. However, the moment SSH will not be secured if it is not properly managed which causes a typical SSH brute force attacks.

Conclusion/Learning:

After doing this activity, I learned how to clone Virtual Machines for different servers to work on. I also learned and observed how to test the connection between a local machine and server/s. In addition to that, I also learned how to use and issue SSH commands in a local machine and I was able to verify connections between those systems. Overall, this activity gave me a brief idea on what our next topic will be and this serves as a preparation to manage SSH connections.

I affirm that I will not give or receive any unauthorized help on this activity/exam and that all work will be my own.

-Keith Maravilla