

Building the Virtual Machine

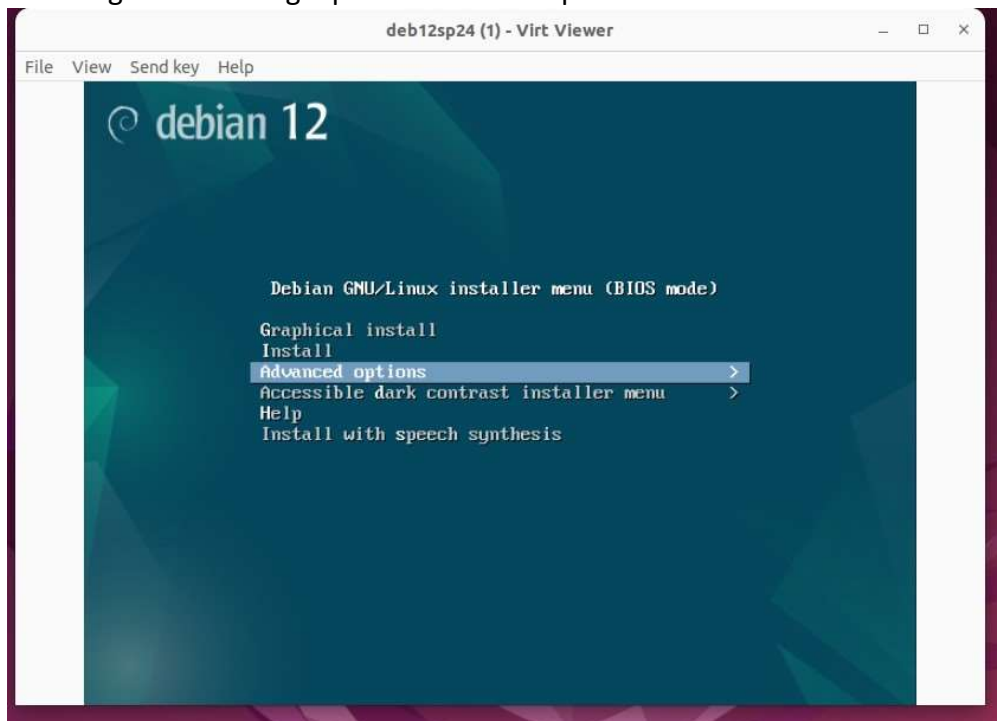
- Assuming all the necessary KVM packages have been installed for Ubuntu 22.04 LTS and we have an existing Debian12 ISO file we can simply use the virt_inst.txt file to begin the Virtual Machine, or VM, construction. To execute the text file, we use bash:

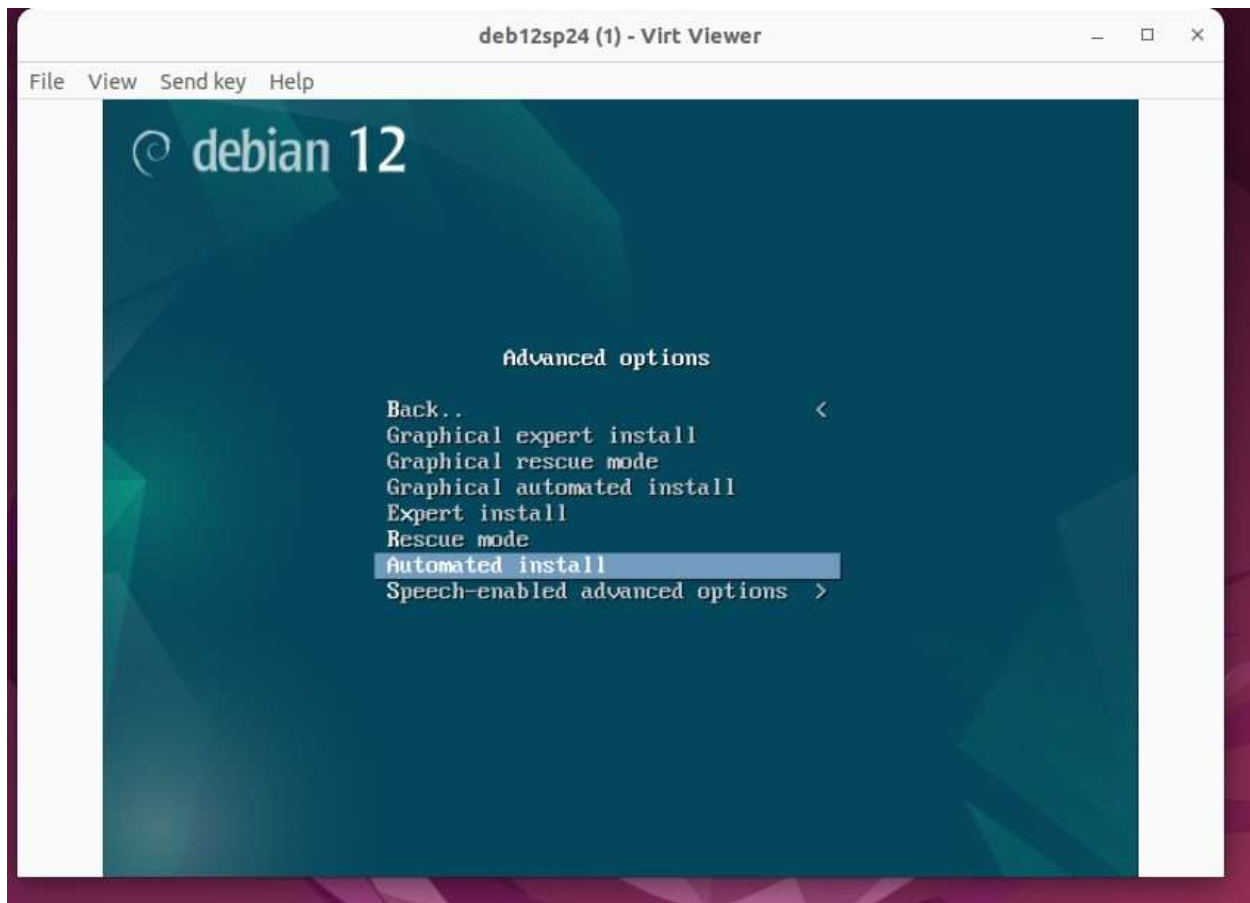
```
johnny@cheese:~/LinuxClass/automation_Scripts$ cat sp24_virt_ins.txt
# this is the command line to
# build a base deb12 guest on
# the kvm server

sudo virt-install \
--os-variant debian12 \
--name deb12sp24 \
--memory 8096 \
--vcpus 4 \
--disk size=22 \
--cdrom debian-12.1.0-amd64-netinst.iso

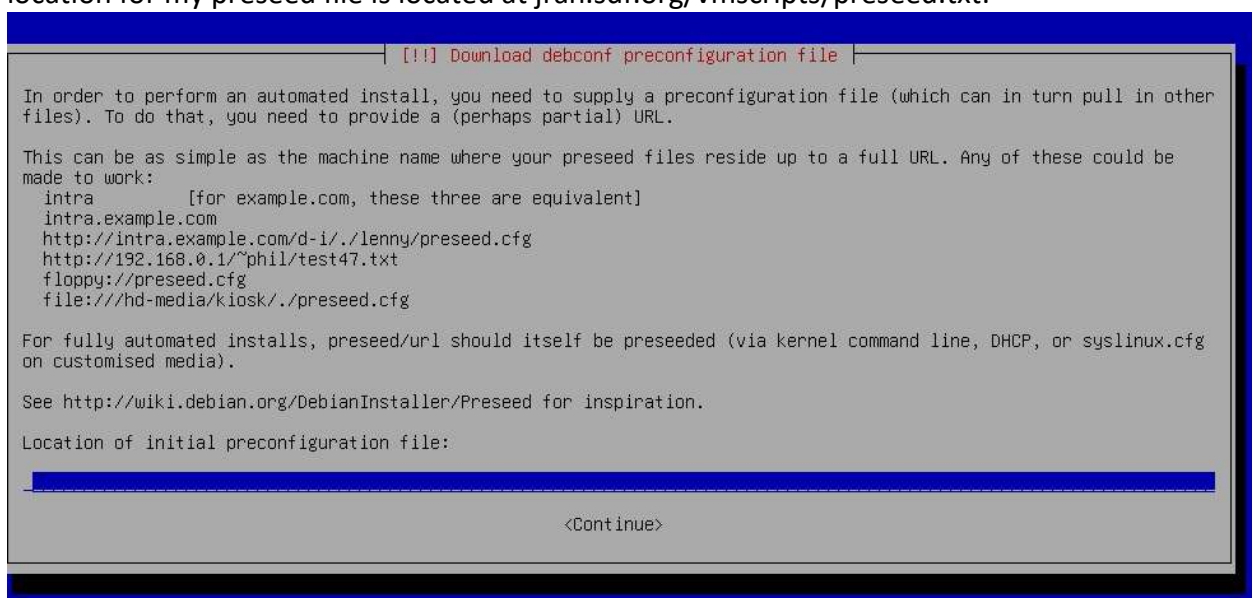
#--
johnny@cheese:~/LinuxClass/automation_Scripts$ vi sp24_virt_ins.txt
johnny@cheese:~/LinuxClass/automation_Scripts$ bash sp24_virt_ins.txt
```

- Next, we're prompted with the installation menu for the Debian 12 distribution. Normally, we'd choose Install option, but we're going to automate this entire process by choosing the following inputs: Advanced Options -> Automated Install

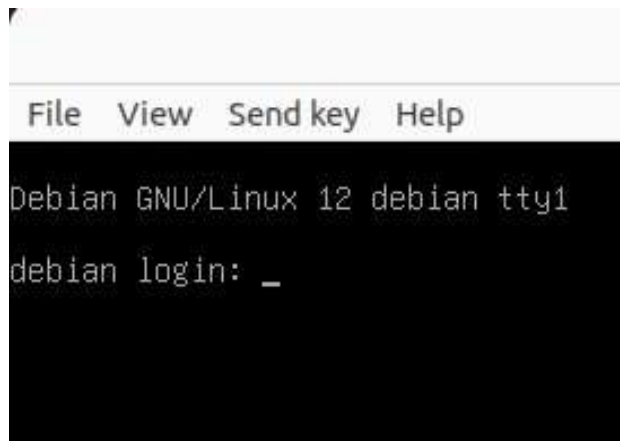




- Afterwards, the process will be set up for automated installation, it will then prompt us for the preseed file, the file that has all the configuration options defined for us. The location for my preseed file is located at jran.sdf.org/vmscripts/preseed.txt:

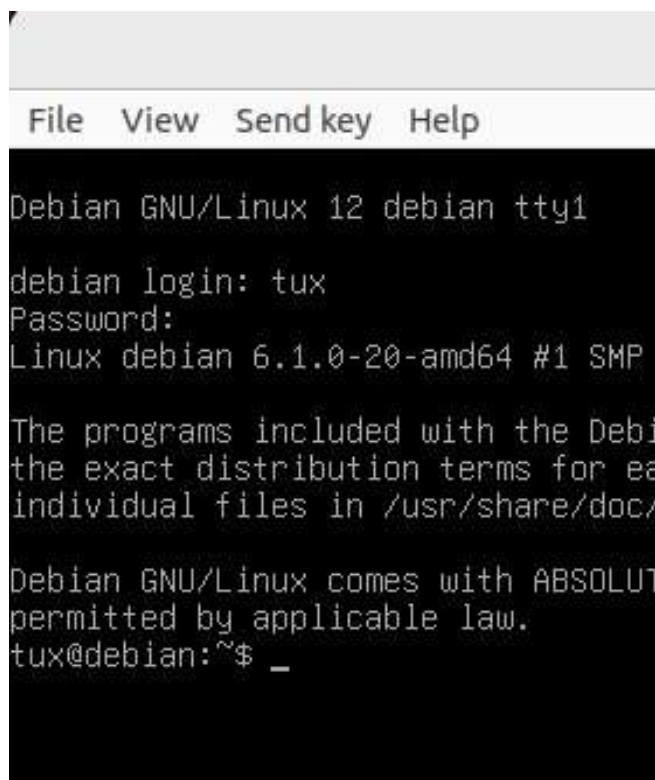


- If our minimal installation of Debian12 is successful we should see the login screen for tty1:



```
File View Send key Help
Debian GNU/Linux 12 debian tty1
debian login: _
```

- We can simply login to our VM, the user is tux and password being vmware12 as defined in our preseed file



```
File View Send key Help
Debian GNU/Linux 12 debian tty1
debian login: tux
Password:
Linux debian 6.1.0-20-amd64 #1 SMP
The programs included with the Debi
the exact distribution terms for ea
individual files in /usr/share/doc/
Debian GNU/Linux comes with ABSOLUT
permitted by applicable law.
tux@debian:~$ _
```

Setting Up the Path Environmental Variable

- Next, we simply set up our PATH environmental variable. This allows us to define paths to executable files or scripts for the system whenever a file or script is executed anywhere on the system. In other words, it allows us and the system to execute files without needing to specify the path to them. We achieve this by editing the .bashrc file with the following:

```
export PATH="$HOME/bin:$PATH"
".bashrc" 115L, 3557B
```

- This lets us define executable files and scripts within the bin directory inside the user's home directory. We can apply this to our current session by running: **source .bashrc**
- We know this works if after we source .bashrc on the commandline the PATH variable when echoed will have the bin directory within the user's home directory inside the PATH variable like the following:

```
tux@debian:~$ echo $PATH
/home/tux/bin:/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
tux@debian:~$
```

Gathering the Scripts within the Bin Directory

- First, we can actually use the SSH service to control the VM from our main terminal in the host system. We'd simply use the command **ip a** to get the IP address of the VM and utilize that for the SSH command in our host system.
- After connecting to tux through SSH from the host system to the VM system, we can gather executable scripts by using the wget command as showcased in the fol

```
tux@debian:~/bin$ wget http://jran.sdf.org/vmscripts/heredoc.txt
--2024-04-24 20:42:55-- http://jran.sdf.org/vmscripts/heredoc.txt
Resolving jran.sdf.org (jran.sdf.org)... 205.166.94.8
Connecting to jran.sdf.org (jran.sdf.org)|205.166.94.8|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1397 (1.4K) [text/plain]
Saving to: 'heredoc.txt'

heredoc.txt                               100%[=====]
2024-04-24 20:42:55 (240 MB/s) - 'heredoc.txt' saved [1397/1397]

tux@debian:~/bin$
```

- Once the scripts have been gathered within the bin directory, we now turn on the executable permission so we can execute these scripts by using the chmod command:

```
tux@debian:~/bin$ ls
addUser.txt  heredoc.txt  post_install.txt  upit.txt
tux@debian:~/bin$ chmod +x *
tux@debian:~/bin$ ls
addUser.txt  heredoc.txt  post_install.txt  upit.txt
tux@debian:~/bin$
```

Using the Scripts and Ensuring Success

- Next, we login as root to store an essential script, that updates the entire system, within the bin directory of root's home directory. Since the preseed file provided no password for root, we simply use the **passwd root** command and set one for root, otherwise we're locked out of root.
- After escalating privileges, we create the bin directory within root's home directory and store the upit file within it so quickly update the system at any time.

```
#!/bin/bash
# upit
# /root/bin/upit
apt-get check
apt clean
apt -y autoclean
apt remove
apt -y autoremove
apt update
apt -y upgrade
apt -y dist-upgrade
cat /proc/version
cat /etc/os-release
uname -a
#--
```

```
root@debian:~/bin# ls
upit
root@debian:~/bin# cat upit
```


- Once the script has been housed, we return to the tux user and execute the command to preform the script as the root user, then return to the original user, tux:

```
tux@debian:~/bin$ cat upit.txt | tail -22 | head -1
# sudo su - -c"/root/bin/upit"
tux@debian:~/bin$ sudo su - -c"/root/bin/upit"
```

```
Calculating upgrade... Done
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
Linux version 6.1.0-20-amd64 (debian-kernel@lists.debian.org) (gcc-12 (Debian 12.2.0-14) 12.2.0, GNU
NANIC Debian 6.1.85-1 (2024-04-11)
PRETTY_NAME="Debian GNU/Linux 12 (bookworm)"
NAME="Debian GNU/Linux"
VERSION_ID="12"
VERSION="12 (bookworm)"
VERSION_CODENAME=bookworm
ID=debian
HOME_URL="https://www.debian.org/"
SUPPORT_URL="https://www.debian.org/support"
BUG_REPORT_URL="https://bugs.debian.org/"
Linux debian 6.1.0-20-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.85-1 (2024-04-11) x86_64 GNU/Linux
/root/bin/upit: line 23: syntax error near unexpected token `('
/root/bin/upit: line 23: `apt install sudo (for tux to sudo)'
tux@debian:~/bin$
```

- After executing the upit script, we can execute the heredoc script, we know its successful if the .funcs has been implemented within the .bashrc:

```
tux@debian:~$ heredoc.txt
Did you create you /home/tux/bin dir?
```

```

export PATH="$HOME/bin:$PATH"
if [ -f ~/.funcs ]; then
    . ~/.funcs
fi
#--
```

- After the heredoc, we then execute the `post_install.txt` script which installs additional packages for our VM:

```
tux@debian:~$ post_install.txt
Hit:1 http://security.debian.org/debian-se
Hit:2 http://httpredir.debian.org/debian b
Get:3 http://httpredir.debian.org/debian b
Fetched 55.4 kB in 1s (110 kB/s)
Reading package lists... Done
```

```
tux@debian:~$ ll
-bash: ll: command not found
tux@debian:~$ source .bashrc
tux@debian:~$ ll
total 56
drwx----- 4 tux  tux  4096 Apr 24 20:53 ./
drwxr-xr-x 3 root root 4096 Apr 24 20:14 ../
-rw-r--r-- 1 tux  tux   244 Apr 24 20:51 .bash_aliases
-rw----- 1 tux  tux  1089 Apr 24 20:37 .bash_history
-rw-r--r-- 1 tux  tux   220 Apr 24 20:14 .bash_logout
-rw-r--r-- 1 tux  tux  3604 Apr 24 20:51 .bashrc
-rw-r--r-- 1 tux  tux     8 Apr 24 20:51 .bcrc
drwxr-xr-x 2 tux  tux  4096 Apr 24 20:51 bin/
-rw-r--r-- 1 tux  tux   177 Apr 24 20:51 .funcs
-rw----- 1 tux  tux    68 Apr 24 20:37 .lessht
-rw-r--r-- 1 tux  tux   807 Apr 24 20:14 .profile
drwx----- 2 tux  tux  4096 Apr 24 20:23 .ssh/
-rw-r--r-- 1 tux  tux     0 Apr 24 20:15 .sudo_as_admin_successful
-rw----- 1 tux  tux  3823 Apr 24 20:53 .viminfo
-rw----- 1 tux  tux    52 Apr 24 20:25 .Xauthority
tux@debian:~$
```

- Afterwards, we verify that functions have been implemented by using the source command for both .brc and .funcs files:

```
tux@debian:~$ source .brc
tux@debian:~$ source .funcs
tux@debian:~$ wftr
</pre></body></html>
tux@debian:~$ whdr
<!DOCTYPE html><head><title>bash web</title></head><body><pre>
tux@debian:~$ calcit 8*8
64
tux@debian:~$ calcit 45/3
15.00
tux@debian:~$
```

- One additional check is to see if we can compile from source, here's a simple C++ program that is fully functional and compiles without error:

```
tux@debian:~/c++$ cat hello.cpp
#include <iostream>

int main()
{
    std::cout << "Hello World!\n";
    return 0;
}
tux@debian:~/c++$ ./a.out
Hello World!
tux@debian:~/c++$
```


Setting Up User Management

- Before we add any users, we have to set up the actual configuration file and the skeleton template within the /etc directory. Focusing on the /etc/skel directory we simply elevate privileges by becoming root and copying the tux user's files and directories into the /etc/skel. We also added additional directories that'll automatically be added to the user's home directory when we create their account:

```
root@debian:/etc/skel# ll
total 76
drwxr-xr-x  8 root root 4096 Apr 24 21:05 ./
drwxr-xr-x 80 root root 4096 Apr 24 20:59 ../
-rw-r--r--  1 root root  244 Apr 24 21:05 .bash_aliases
-rw-----  1 root root 1089 Apr 24 21:05 .bash_history
-rw-r--r--  1 root root  220 Apr 24 21:05 .bash_logout
-rw-r--r--  1 root root 3604 Apr 24 21:05 .bashrc
-rw-r--r--  1 root root    8 Apr 24 21:05 .bcrc
drwxr-xr-x  2 root root 4096 Apr 24 21:02 bin/
drwxr-xr-x  2 root root 4096 Apr 24 21:02 c++/
drwxr-xr-x  2 root root 4096 Apr 24 21:02 Downloads/
-rw-r--r--  1 root root  177 Apr 24 21:05 .funcs
drwxr-xr-x  2 root root 4096 Apr 24 21:02 inclass/
-rw-----  1 root root   68 Apr 24 21:05 .lessht
-rw-r--r--  1 root root  807 Apr 24 21:05 .profile
drwxr-xr-x  2 root root 4096 Apr 24 21:02 projects/
drwx-----  2 root root 4096 Apr 24 21:05 .ssh/
-rw-----  1 root root 7932 Apr 24 21:05 .viminfo
-rw-----  1 root root   52 Apr 24 21:05 .Xauthority
root@debian:/etc/skel#
```

- Focusing on the configuration file, we have to enable some default values within the **/etc/adduser.conf** file. Specifically the default values for the home directory location of the newly added users and where to acquire the template home directory. The home directory location would be /home while the template home directory would be located within /etc/skel.
- Once enabled, we can add an individual user using the adduser command. By default, adding a user this way is meant to be interactive by allow us to define the password of the user, a description that can be a full name, email, room number, or some other unique identifier besides the username. Utilizing the config file, we can define and refine more groups for different users. However, we can start by adding a simple user as follows:

```
tux@debian:~$ adduser johnny
-bash: adduser: command not found
tux@debian:~$ sudo adduser johnny
Adding user `johnny' ...
Adding new group `johnny' (1000) ...
Adding new user `johnny' (1000) with group `johnny (1000)' ...
Creating home directory `/home/johnny' ...
Copying files from `/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for johnny
Enter the new value, or press ENTER for the default
    Full Name []: Johnny@admin.com
    Room Number []:
    Work Phone []:
    Home Phone []:
    Other []:
Is the information correct? [Y/n] y
Adding new user `johnny' to supplemental / extra groups `users' ...
Adding user `johnny' to group `users' ...
tux@debian:~$ ls ../
johnny  tux
tux@debian:~$ sudo ls ../johnny/
bin  c++  Downloads  inclass  projects
tux@debian:~$
```

- On our original terminal within the VM, we can verify that this user was successfully added by using the tty1 login as follows:

```
Debian GNU/Linux 12 debian tty1

debian login: johnny
Password:
Linux debian 6.1.0-20-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.85-1 (2024-04-11) x86_64

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

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
johnny@debian:~$ echo $PATH
/home/johnny/bin:/home/johnny/bin:/home/johnny/bin:/home/johnny/inclass:/usr/local/bin:/
johnny@debian:~$
```

- We can take this a bit further by automating the process of user creation given a list of users. For example, say we have a list of emails that represent incoming employees, utilizing a script called addUser.txt we can easily implement multiple users at once without issue. Note that automating the reverse process of deleting users can be done as well.

```
tux@debian:~$ cat emails.txt
apple89@gmail.com
orange46@gmail.com
berry13@gmail.com
mic22@gmail.com
melons58@gmail.com
kingh63@gmail.com
queen82@gmail.com
ant11@gmail.com
overlord69@gmail.com
minion34@gmail.com
grape78@gmail.com
tux@debian:~$ addUser.txt emails.txt
Adding user `apple89'
```


- To verify if the users have been properly made, we use the **/etc/passwd** file and the **/etc/shadow** file to see if the users have been added and have assigned passwords:

```
tux:x:1010:1010:tux,,,:/home/tux:/bin/bash
johnny:x:1000:1000:Johnny@admin.com,,,:/home/johnny:/bin/bash
apple89:x:1001:1001:apple89@gmail.com,,,:/home/apple89:/bin/bash
orange46:x:1002:1002:orange46@gmail.com,,,:/home/orange46:/bin/bash
berry13:x:1003:1003:berry13@gmail.com,,,:/home/berry13:/bin/bash
mic22:x:1004:1004:mic22@gmail.com,,,:/home/mic22:/bin/bash
melons58:x:1005:1005:melons58@gmail.com,,,:/home/melons58:/bin/bash
kingh63:x:1006:1006:kingh63@gmail.com,,,:/home/kingh63:/bin/bash
queen82:x:1007:1007:queen82@gmail.com,,,:/home/queen82:/bin/bash
ant11:x:1008:1008:ant11@gmail.com,,,:/home/ant11:/bin/bash
overlord69:x:1009:1009:overlord69@gmail.com,,,:/home/overlord69:/bin/bash
minion34:x:1011:1011:minion34@gmail.com,,,:/home/minion34:/bin/bash
grape78:x:1012:1012:grape78@gmail.com,,,:/home/grape78:/bin/bash
tux@debian:~$ sudo cat /etc/shadow
```

- We can see that the users indicated by the left-most field have indeed been added.

```
johnny:$y$j9T$/j219Nwaek4VgAAG2VKyW.$3GQhRN0bG8IKq2Q6lzL/D5E3rqiMLsgYZ6iSHGjMPB0:1983
apple89:$y$j9T$4Hin652RjFq.SsO4TUI2.1$S.bgYajJiPVxtDB75fWoOvZoNs3c0xYQIgcJZwBHpN6:198
orange46:$y$j9T$XUkqHSyvXDq.IQW6gM6ao/$tK2ijf9mMo25GMW9bNMFPQQCotxBmpI/253cavaFg7C:19
berry13:$y$j9T$Ip7fDzELzgZYFFVBgDB8D1$krBENNSwnIsEUzujtUXGqyRCUUs1N7P.PHGLwmEhga3:198
mic22:$y$j9T$SYF0bhML1MLz9hVip2Tby.$p3hy7QLbND4arKbUBi4QWHwrnptH3jHIeHVUZotUr8/:19837
melons58:$y$j9T$dAF54/AlUjjPFANCmX0yt1$LSKOZPgZEUwQ6btSLO4TQLvL6Yyw8qGXb1XSYbThxM4:19
kingh63:$y$j9T$2PT9Zez2By0ykj97aC2xh1$hhPktRacTqPGvUsmNksZGwtjK..SjYjS0AlHqtLR43.:198
queen82:$y$j9T$S5E.rFFzdpMBQPcI.fNZF0$B7NL2rH1y762SJHoDtIyh6wbrUXmmRsp5tLHSAxyjV0:198
ant11:$y$j9T$5EvRG8r4Tcgw0Zg9pKcfi0$dwfM/NUFuyeV9jDEEA0Lo4tfNL2aLL6QdzaFaxYt8c1:19837
overlord69:$y$j9T$9DpXi1v1CGX9zrA/iuf.E.$IeSM16ruRfuGVsxZgYUJe5auAitLL0vY7sdRYGPtwA0:
minion34:$y$j9T$TJ/qE6PLqrJ0sOC/x/2F00$YwvpZdA0blx7OVkUyesukC79.9w140/jtF37bVJ2G9D:19
grape78:$y$j9T$dtxLXgT4ZLKAhi2uBnfFV.$uVYQwcgOMNLKbv4Gf90UQCIEqNrKJ7C7oteYlQvEE4:198
```

- We can also see a long string of characters after the left-most field indicating that each user has been assigned a password

Setting Up Serial port

- Next we set up the Serial capabilities of the VM by activating the serial service. We use the systemctl command to start and enable the service. We can check the status using the systemctl command as well:

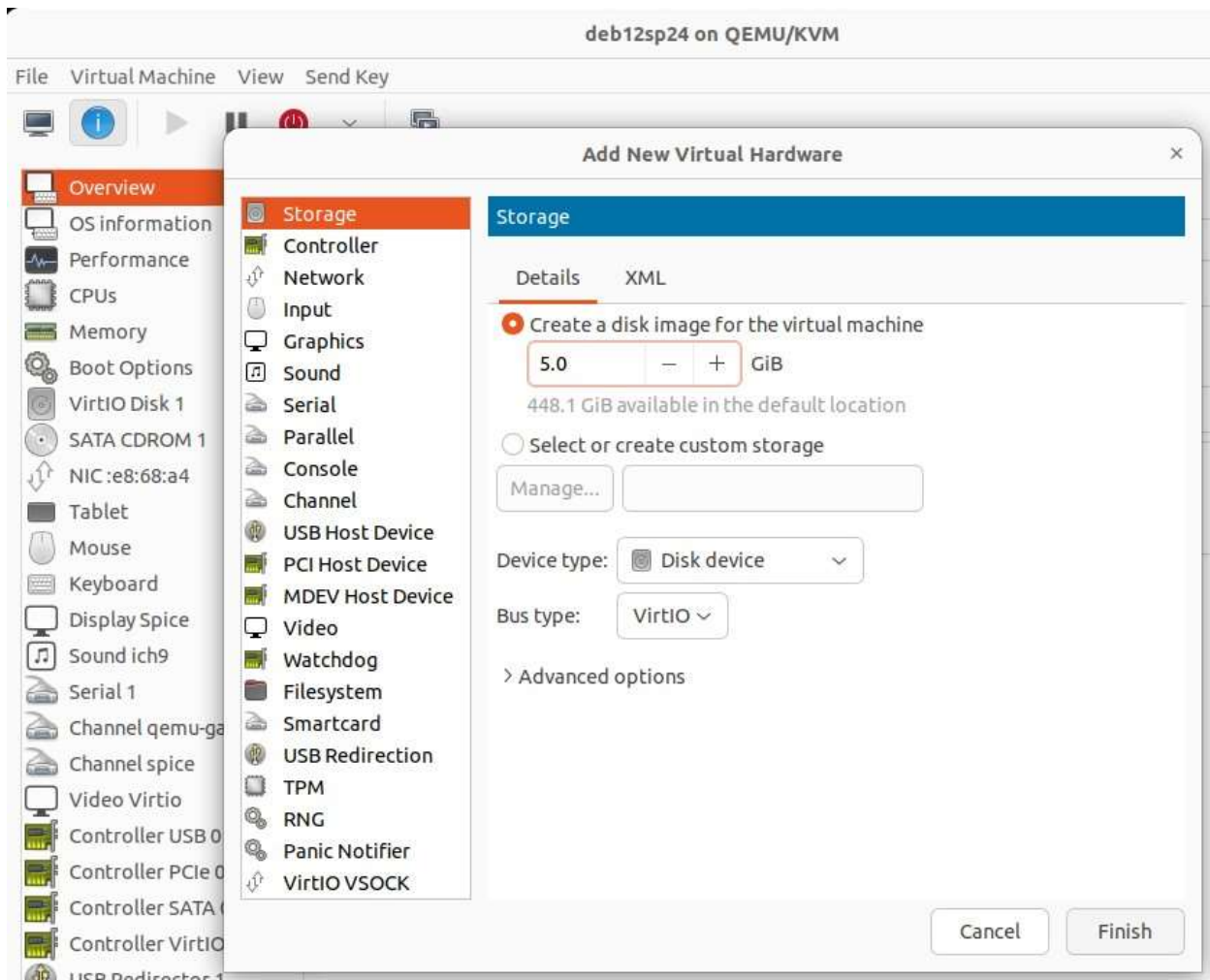
```
tux@debian:~$ systemctl status serial-getty@ttyS0.service
● serial-getty@ttyS0.service - Serial Getty on ttyS0
   Loaded: loaded (/lib/systemd/system/serial-getty@.service; enabled; preset: enabled)
   Active: active (running) since Wed 2024-04-24 20:51:38 UTC; 37min ago
     Docs: man:agetty(8)
           man:systemd-getty-generator(8)
           https://0pointer.de/blog/projects/serial-console.html
  Main PID: 1361 (agetty)
    Tasks: 1 (limit: 9357)
   Memory: 204.0K
      CPU: 2ms
   CGroup: /system.slice/system-serial\x2dgetty.slice/serial-getty@ttyS0.service
           └─1361 /sbin/agetty -o "-p -- \u" --keep-baud 115200,57600,38400,9600 - vt220

tux@debian:~$
```

- We can utilize this service by doing the following within the VM window: go to view -> console -> serial port]
- This service allows us to reboot the VM without losing our monitoring capabilities, this comes into play later when we're almost done adding extra storage to our VM

Setting Up Storage

- First, we can add virtual disks using the VM window. This simulates installing more hard disks onto the rack of a server. We're going to add two more 5 GB virtual disks



- We can verify the installation of our virtual disks by using the lsblk command:

```
tux@debian:~$ lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
sr0                                  11:0    1 1024M  0 rom
vda                                  254:0    0   22G  0 disk
├─vda1                              254:1    0  487M  0 part /boot
├─vda2                              254:2    0    1K  0 part
└─vda5                              254:5    0 21.5G  0 part
   ├─debian--vg-root                253:0    0 20.6G  0 lvm /
   └─debian--vg-swap_1              253:1    0  980M  0 lvm [SWAP]
vdb                                  254:16    0    5G  0 disk
vdc                                  254:32    0    5G  0 disk
tux@debian:~$
```

- Since our virtual disks are being detected as vdb and vdc, we simply use the fdisk command to create LVM ready partitions using the entire disk and with the type of 8e. The following is an example of a LVM ready partition on the vdb drive:

Device	Boot	Start	End	Sectors	Size	Id	Type
/dev/vdb1		2048	10485759	10483712	5G	8e	Linux LVM

- Using lsblk again, we can verify if our new LVM partitions on each disk is being detected and recognized:

```
vdb                                  254:16    0    5G  0 disk
└─vdb1                              254:17    0    5G  0 part
vdc                                  254:32    0    5G  0 disk
└─vdc1                              254:33    0    5G  0 part
root@debian:~#
```

- After verifying that our partitions are being recognized, we can then use the **pvccreate /dev/vdb1 /dev/vdc1** command to add both partitions into the physical volume pool. List our physical volume pool showcases that this was a success

```
root@debian:~# pvs
  PV          VG      Fmt  Attr  PSize   PFree
  /dev/vda5   debian-vg  lvm2  a--   <21.52g     0
  /dev/vdb1                   lvm2  ---   <5.00g <5.00g
  /dev/vdc1                   lvm2  ---   <5.00g <5.00g
root@debian:~#
```

- Now we dedicate the two partitions towards a volume group, which I'll name backupVG. This can be done using **vgcreate backupVG /dev/vdb1 /dev/vdc1** allowing us to create logical volumes. Using pvs command again allows us to see if these partitions were dedicated towards the backupVG volume group:

```
root@debian:~# pvs
  PV          VG      Fmt  Attr  PSize   PFree
  /dev/vda5   debian-vg  lvm2  a--   <21.52g     0
  /dev/vdb1   backupVG   lvm2  a--   <5.00g <5.00g
  /dev/vdc1   backupVG   lvm2  a--   <5.00g <5.00g
root@debian:~#
```

- Moving onto logical volumes, we simply define the size and which volume group to pull volume from. We can use the `lvcreate` command to create a logical volume named `backupLV` with the size of 5 GB. Using the `lsblk` command again showcases that the partitions are dedicated towards a volume group and a logical volume. In addition, we can use the `lvs` command to verify if the logical volume is 5 GB. Since we only use 5 GB of the total 10 GB from both partitions, we can extend the volume of `backupLV` if we need more volume.

```

root@debian:~# lvcreate --size 5G -n backupLV backupVG
Logical volume "backupLV" created.
root@debian:~# lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
sr0                                  11:0    1 1024M  0 rom
vda                                  254:0    0   22G  0 disk
├─vda1                              254:1    0  487M  0 part /boot
├─vda2                              254:2    0    1K  0 part
└─vda5                              254:5    0 21.5G  0 part
   ├─debian--vg-root                253:0    0 20.6G  0 lvm /
   └─debian--vg-swap_1              253:1    0  980M  0 lvm [SWAP]
vdb                                  254:16    0    5G  0 disk
├─vdb1                             254:17    0    5G  0 part
│   └─backupVG-backupLV            253:2    0    5G  0 lvm
vdc                                  254:32    0    5G  0 disk
├─vdc1                             254:33    0    5G  0 part
│   └─backupVG-backupLV            253:2    0    5G  0 lvm
root@debian:~# lvs
LV      VG      Attr      LSize   Pool Origin Data%
backupLV backupVG -wi-a----- 5.00g
root    debian-vg -wi-ao---- 20.56g
swap_1  debian-vg -wi-ao---- 980.00m
root@debian:~#

```

```

root@debian:~# vgs
VG      #PV #LV #SN Attr   VSize   VFree
backupVG 2   1   0 wz--n- 9.99g 4.99g
debian-vg 1   2   0 wz--n- <21.52g 0
root@debian:~#

```


Mounting Storage

- With our newly installed storage, we can now prime it for use by assigning it a location through mounting. First we have to apply the ext4 filesystem to the logical volume by using the mkfs command like the following:

```
root@debian:~# mkfs.ext4 /dev/backupVG/backupLV
mke2fs 1.47.0 (5-Feb-2023)
Discarding device blocks: done
Creating filesystem with 1310720 4k blocks and 327680 inodes
Filesystem UUID: 94bab2fd-f540-46c6-bfac-8111a7ba6559
Superblock backups stored on blocks:
    32768, 98304, 163840, 229376, 294912, 819200, 884736

Allocating group tables: done
Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done

root@debian:~#
```

- After setting the filesystem, we can use the **mount** command to mount the logical volume onto a specified location on our main vda virtual disk. In this scenario, we're going to mount the logical volume onto a backups folder within the /mnt directory.

```
root@debian:~# lsblk
vdb                254:16    0     5G  0 disk
└─vdb1              254:17    0     5G  0 part
   └─backupVG-backupLV 253:2     0     5G  0 lvm  /mnt/backups
vdc                254:32    0     5G  0 disk
└─vdc1              254:33    0     5G  0 part
   └─backupVG-backupLV 253:2     0     5G  0 lvm  /mnt/backups
root@debian:~#
```


- We can verify mounting by adding a file to see if when we unmount the logical volume, we'll be unable to access that file. This occurs as the mount point is a connection that allows use to access the other storage disks, without a mount point, we can never directly access it from our vda disk drive. Here's an example of what should occur when we unmount the /mnt/backups directory:

```
-rw-r--r-- 1 tux tux 78 Apr 24 22:24 note.txt
root@debian:/mnt/backups# cat note.txt
Hi! This is the logical volume stored in the Volume group of both vdb and vdc
root@debian:/mnt/backups#
```

```
root@debian:/mnt# umount backups
root@debian:/mnt# ls
backups
root@debian:/mnt# cd backups/
root@debian:/mnt/backups# ls
root@debian:/mnt/backups# ll
total 8
drwxr-xr-x 2 root root 4096 Apr 24 21:48 ./
drwxr-xr-x 3 root root 4096 Apr 24 21:48 ../
root@debian:/mnt/backups#
```

```
root@debian:/mnt/backups# ll
total 28
drwxrwxrwx 3 root root 4096 Apr 24 22:24 ./
drwxr-xr-x 3 root root 4096 Apr 24 21:48 ../
drwx----- 2 root root 16384 Apr 24 22:18 lost+found/
-rw-r--r-- 1 tux tux 78 Apr 24 22:24 note.txt
root@debian:/mnt/backups# cat note.txt
Hi! This is the logical volume stored in the Volume group of both vdb and vdc
root@debian:/mnt/backups#
```

- As shown in the series of screenshots, we have the initial note.txt file stored within the /mnt/backups directory stored in the logical volume of backupLV. When we unmount the /mnt/backups directory we are suddenly unable to access that file because we have no mount point for the logical volume. Remounting the backupLV logical volume back onto /mnt/backups directory grants us access to the note.txt file.

- The only issue with this method of mounting is that when we reboot the VM system, the mount point will be lost, meaning we can no longer access the files within that backupLV logical volume. Therefore, to make this mount point persistent we use the /etc/fstab file to permanently mount the backupLV logical volume onto the /mnt/backups directory. We need the **UUID** of the backupLV logical volume in order to have a consistent and reliable mount point that persists through shutdowns and reboots:

```

/dev/sr0          /media/cdrom0    udf,iso9660 user,noauto      0          0
UUID=94bab2fd-f540-46c6-bfac-8111a7ba6559 /mnt/backups     ext4 defaults 0 2
~
~
~
~
~
~

```

- Going back to our original serial port console on our VM window, we can simply command a reboot of the VM system and verify that the mount point persists. As showcased below, the note.txt file within the /mnt/backups was still accessible immediately after a reboot

```

tux@debian:~$ sudo reboot
[sudo] password for tux:

Broadcast message from root@debian on pts/0 (Wed 2024-04-24 22:41:42 UTC):

The system will reboot now!

tux@debian:~$

Debian GNU/Linux 12 debian ttyS0

debian login: tux
Password:
Linux debian 6.1.0-20-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.85-1 (2024-04-11)
x86_64

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

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Apr 24 22:41:32 UTC 2024 on ttyS0
tux@debian:~$ cat /mnt/backups/note.txt
Hi! This is the logical volume stored in the Volume group of both vdb and vdc
tux@debian:~$

```

Starting a Webserver Within Our VM

- To make this work use python, websocketd, and change localhost with ip address of vm, make count.sh executable
- We take our VM further by using the /var/www/html directory to host a web server. There are many approaches to this, but a simple implementation is using python3 package to jumpstart a http.server from within any directory, in our case this is the /var/www/html directory. Below I have a host of .html files and an executable count.sh file and use the following command to start a webserver within our VM

```
tux@debian:/var/www/html$ ls
count.html  count.sh  index.html  websocketd_info.txt
tux@debian:/var/www/html$ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
█
```

- To access the webserver, we have to use the ip address of the VM at the port 8000. Otherwise, we'll be unable to access the webserver. Entering this in the address of a web browser shows the default page of Apache2 meaning Success as the web server is properly serving the index.html file



- We can actually access bash executable files through the web server by utilizing websocketd to open a port on 8080. This executes the shell file and outputs it to the system requesting on that port. We can start this port using the following:

```
tux@debian:/var/www/html$ websocketd --port=8080 ./count.sh
Thu, 25 Apr 2024 19:40:32 +0000 | INFO | server | | S
Thu, 25 Apr 2024 19:40:32 +0000 | INFO | server | | S
█
```

- On our previous address, we just add /count.html to request the output of the count.sh. Here's what is outputted when we access the count.html file that is connected to the count.sh executable file through websocketd

```
← → ↻ 192.168.122.198:8000/count.html
CONNECT
MESSAGE: 1
MESSAGE: 2
MESSAGE: 3
MESSAGE: 4
MESSAGE: 5
MESSAGE: 6
MESSAGE: 7
MESSAGE: 8
MESSAGE: 9
MESSAGE: 10
MESSAGE: 11
MESSAGE: 12
MESSAGE: 13
MESSAGE: 14
MESSAGE: 15
MESSAGE: 16
MESSAGE: 17
MESSAGE: 18
MESSAGE: 19
MESSAGE: 20
MESSAGE: 21
MESSAGE: 22
MESSAGE: 23
MESSAGE: 24
MESSAGE: 25
MESSAGE: 26
MESSAGE: 27
MESSAGE: 28
MESSAGE: 29
MESSAGE: 30
DISCONNECT
```


- We can also modify the index.html default file to any design we desire. For example, I simply replaced the index.html file with a quickly written h1 html tag and a body html tag. Going back to the original address that automatically points to the index.html outputs the following

