CS 470: Unix/Linux Sysadmin
Spring 2025 Lab 3
Red Hat-style Linux, with NFS file server and CUPS print server

Things from prior labs that are required to begin this lab:

- lab 2: FreeBSD up, online, and accepting mail

Things in this lab that are on the critical path for upcoming labs:

- getting Rocky Linux up and online
- getting your NFS service running and sharing home directories with FreeBSD

The CUPS side quest near the end of this lab is due the day of the final exam.

Three gigs for the Professors under the sky,
Seven for the Grad-Students in their halls of stone,
Nine for Mortal Users doomed to `vi`,
One for the Hacker Lord on his dark throne,
In the Land of Unix where the Daemons lie.
One Kernel to rule them all, One Kernel to find them,
One Disk to hold the files and in the Darkness grind them,
In the Land of Unix where the Daemons lie.
– *from the web, via Dr. Carroll's old CS470 labs,*
and with sincere apologies to J.R.R. Tolkien

This lab gets our hands on the first Linux distribution in the class, Rocky Linux, and we set up that One Disk™ of a file server to hold the files, and in the darkness grind them. Our laptop SSDs don't make the grinding noise of magnetic read/write heads moving across the spinning hard disks of yore, but that is a welcome change. Also, where OpenBSD used `ypldap` to get user/group and FreeBSD used `nss_ldap`, Linux now has `sssd`, a unified One Ring™ to rule all naming, federation, directory services, and authentication.

Red Hat was the first blockbuster Linux distribution. Historically, it always offered its Linux free for download, but for many years now, it has stopped the free downloads and charged a subscription fee for support and access to downloads, security fixes, and updates. Because the cost is onerous to some, and the software is free and open source, this business practice created an opportunity for others to effectively steal revenue share away from Red Hat by re-compiling its operating system, without the Red Hat logos and without the subscription-enforcing software. They are able to offer a compatible distribution with a relatively low level of effort in putting it together and maintaining it.

Numerous have, and still do, make Linux distributions intended to be 100% binary-compatible with Red Hat Linux; see this Wikipedia page for more information. To do so means that they follow Red Hat constantly, and re-release the same operating system, package-by-package, version-by-version from their own repositories, separate from Red Hat. CERN, the European particle physics labs, used to make their own, called Scientific Linux, with cool atom and particle-based iconography and themes. Oracle also makes its own Red Hat clone, also 100% compatible aside from its value-add "unbreakable" kernel.

CentOS was the first and biggest of all the clones. Starting in 2004, from version 2.1 of Red Hat Enterprise Linux ("RHEL") and the same version of CentOS, it was the primary design goal of CentOS to be 100% free for all use cases and 100% binary compatible with Red Hat Enterprise Linux ("RHEL"), Red Hat's primary operating system offering, which had always charged for support and updates. In 2014, Red Hat bought the CentOS

Foundation, making CentOS the **official** subscription-free version of Red Hat for use in testing and development. When this happened, CERN ended the roadmap for Scientific Linux (which finally ended its support lifetime on June 30th 2024) and transitioned everybody to CentOS.

IBM was always a large stakeholder in Red Hat, but in 2018, IBM announced its intent to purchase the rest of Red Hat for $34B, and the purchase closed in July 2019. For all intents and purposes, Red Hat is IBM Linux, and its American ownership makes RHEL the favorite Linux of the American government sector. It is for this reason we use a derivative of RHEL in lab 3: RHEL and its derivatives are the second most widely-used in the world. You got Red Hat when you needed paid-for on-call support for critical production instances of the operating system. If you didn't feel that you needed support, if you just wanted a development or test system, or you just didn't want to pay, then CentOS was what most people used ... but then, in December 2020, Red Hat killed it, dead.

In the middle of the timeline for CentOS 8.x, Red Hat announced it was turning CentOS into a rolling "stream" distribution that no longer tracked Red Hat – which is what most of the people using CentOS were using it for – but a rolling stream that Red Hat tracked itself to preview upcoming releases of RHEL. The response in the community was swift and unforgiving to Red Hat. In response, they announced a couple of programs where small business deployments and individual developers could get RHEL for free, provided that they register for accounts with Red Hat ... but the damage was already done, and they should have had that plan ready from the get-go, not as a band-aid a few weeks after they made everybody really angry.

I always used CentOS as my subscription-free Red Hat derivative, both personally and for this lab. I briefly considered using either Oracle Linux (also a Red Hat clone) or Red Hat Enterprise Linux itself, but both of these options would have required a free sign-up. Even though Red Hat promises nobody will contact you, I did not want to ask you all to sign up for anything if it wasn't absolutely necessary.

The original founders of the CentOS project decided to start a new distribution to mirror RHEL almost instantly, Rocky Linux, and we used it in 2021, but in 2022, they couldn't get their version 9.0 out and new RHEL clone AlmaLinux could, so we used AlmaLinux just once, that summer. RHEL 9.0 was released on May 17th, and Alma was just nine days behind with their 9.0 release on the 26th. Rocky Linux didn't release their RHEL 9.0-compatible release until July 14th. However, they're at version parity right now, and the Rocky project is taking way less long to get its releases out. AlmaLinux's branding, however, is hideously ugly at best, and **they** should have to pay **us** to look at that hideous logo … I'd happily be a little behind on package versions to never look at that logo. Rocky Linux it is again.

If somebody says they need/want Red Hat, Rocky will do just fine as a development and test platform. The version of Rocky Linux we'll be using in this lab lines up, package-for-package, with the same version of RHEL.

This introduction wouldn't be complete without introducing the things we're going to do in this lab. Our lab 3 VM is going to be our lab-net's file server, using the file service protocol most commonly associated with Unix, NFS. NFS stands for network file system, and it began its life as an experiment inside Sun Microsystems. Sun was founded by Andy Bechtolsheim, a graduate of the Stanford team that started making Unix workstations for the first campus-wide network (Stanford University Network or SUN), and Bill Joy, one of the key people in the early development of BSD Unix at Berkeley and the original author of `vi`, `csh`, and that highly-regarded TCP/IP stack in the BSD kernel.

Version 1 of NFS never made it out of Sun. Version 2 was the first public release, and is still the most widely-used. NFS is built on top of RPC ("remote procedure call"), an API for modularizing and distributing functions across a network … perfect for a distributed file system like NFS. They kind of grew up together, and it's been widely said that they branded the first public release as version two in order to test the version interoperability and fallback features of RPC.

NFS has widely been regarded as insecure for the majority of its life. This is in part because of its reliance on RPC, its traditional lack of encryption, and primarily because NFS versions 2 and 3 forced administrators of systems subscribing to a pool of NFS file shares to truly trust one another. This is because NFS uses a user's UID number (from `/etc/passwd`) on the remote system as the UID on the file server. Though the capability existed to map specific user IDs to arbitrary other ones on a system-by-system basis, this rapidly turns into a pain, and the reality of it is: user IDs must generally match for each individual user across NFS client systems, and user IDs must not collide. If you have the same user ID number as me on two systems that are both clients to the same NFS server, you have full access to my files and vice-versa because the file server historically could not distinguish between us.

NFS version 4 (NFSv4) was jointly developed by Sun and the IETF, the standards body behind the RFC system, and gave us the way out of this security problem, by authenticating each individual user on each client system. Though we'll be configuring NFSv4 in this lab, we'll **not** be configuring per-user authentication. In stark contrast with other technologies like NIS and NIS+ (also grown inside Sun), NFS has always been widely used, as its benefits are widely perceived to be worth the risks. NFS continues to be developed and extended by the IETF.

part zero: get it

Go to https://rockylinux.org/ and click the "download" button. On the download page, choose your architecture, x86_64 if you're on Intel, or ARM64 (aarch64, ARM architecture 64, is another common abbreviation for the 64-bit ARM instruction set) if you're using an ARM Mac. The "DVD" download is BIG, just over 7 GB. Fortunately, we're not going to need all that stuff from that DVD ISO, and that you can download the "minimal" ISO. It's right under 2 GB no matter which architecture you're running.

part one: install it

Create a new custom VM on your shared VM network. In UTM, choose to virtualize a new Linux, since we're again using the native instruction set.

Also for VMware users, if you're presented with a nuisance calling itself "easy install," disable it. Here's how to do so on VMware Workstation, where it's a little less "easy."

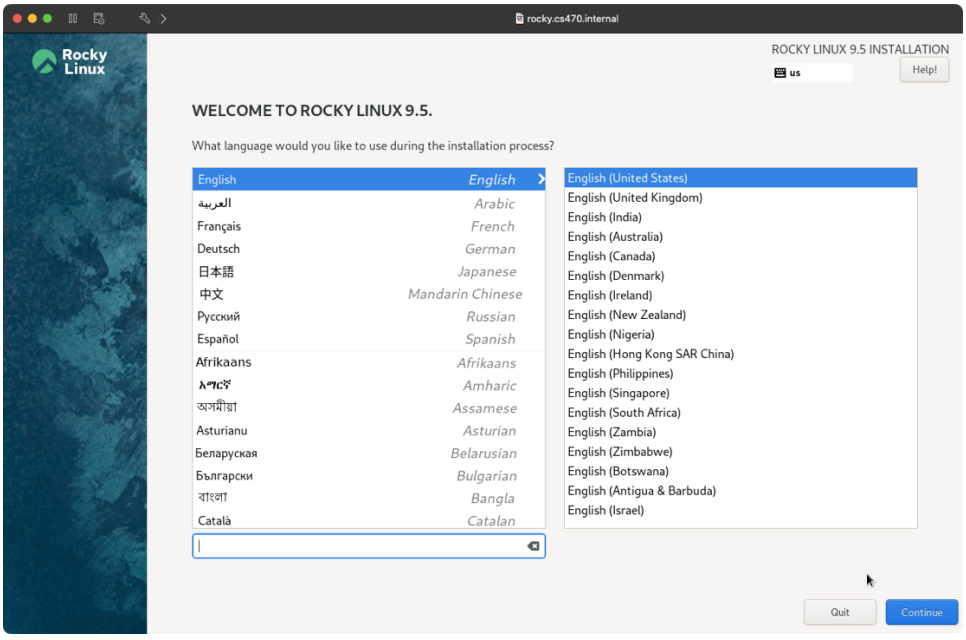
Use the following specifications:

- guest OS:
  - Intel/VMware: Red Hat Enterprise Linux 9 64-bit or Rocky Linux 64-bit
  - ARM/UTM: virtualize, Linux
- CPU: two virtual cores or processors
  one processor, two cores if you're on VMware Workstation
- boot firmware: UEFI
  (this is the default on ARM/UTM)
- RAM: 1 GB (1024 MB)
- hard disk: 16 GB

ARM Mac / UTM users: do not enable "hardware OpenGL acceleration," and go in and open your VM's settings before booting it up. I recommend you use a virtio-gpu-pci emulated display card, so that you can pause our VM if needed. "GPU Supported" options do not support pausing a VM. I initially recommended a virtio-ramfb display card here, but received reports of this not working well after the first reboot.

VMware users: If you have problems or hangs during the early stages of firing up or booting your Rocky VM, shut down your virtual machine, go to its display settings, and make sure that any 3D graphics features are **turned off**. Also, if you're running VMware, check that you're running the latest version of Workstation or Fusion. Workstation should be at version 17.6.3, Fusion should be at 13.6.3. If you're not running the latest version, go into and shut down your lab 1 and lab 2 VMs, then tell VMware to check for and install whatever updates are available. The options to check version number and to check for updates are in the "help" menu in VMware Workstation on Windows or Linux, and in the VMware Fusion menu on Macs.

When you start the VM, you should get a boot loader (`grub`, short for grand unified boot loader) menu offering you the ability to check the install media, or to just go ahead and install. We don't need to check the media – this is for use cases involving burning a physical disc – so go ahead and select to install … it may take a minute or two to load the graphical installer. Once it does, you'll be presented with a list of language choices from the installation media (the ISO).



Choose your language, and you'll be greeted by the installation screen. All of the options in the following screen come before installing Rocky Linux. A number of things can be set up before or after the installation.

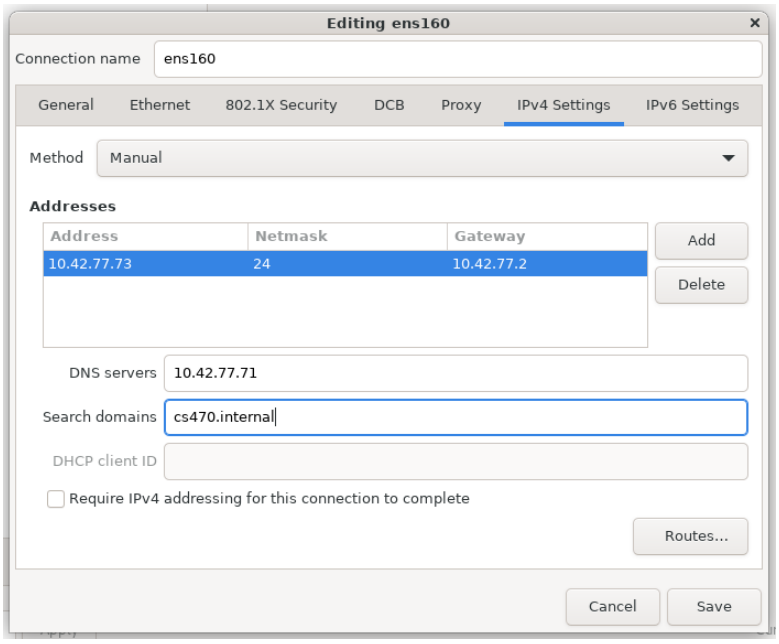
Let's start counting steps now, with this screen.

1. First, select "Network & Host Name" … let's set up the network.

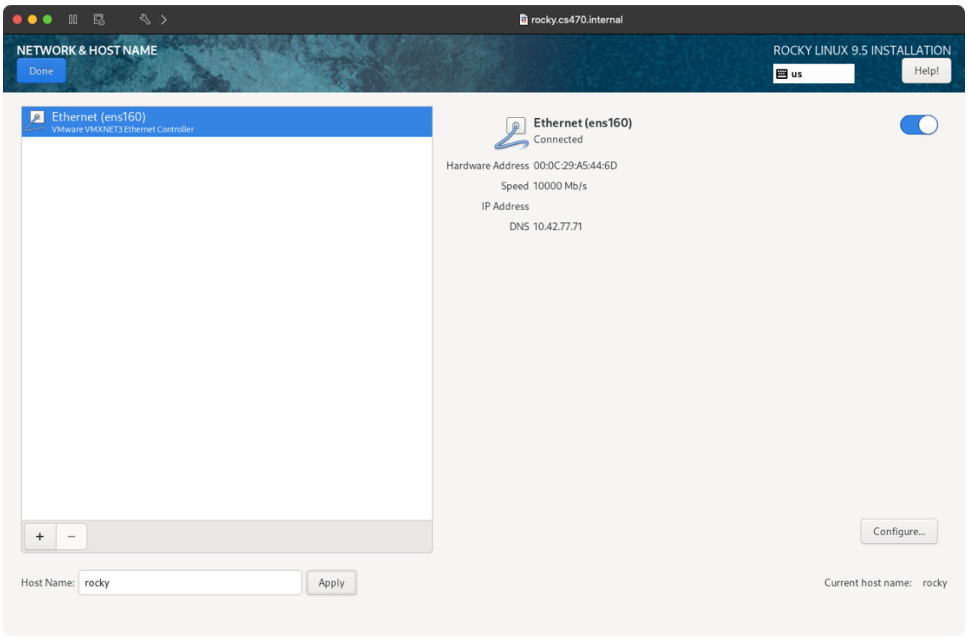
   You should see the network and host name configuration pane … first, let's use the switch in the upper right-hand corner to turn our network on, if it isn't already. If you had to do that, you should see it go out and get an address from DHCP on your local NATwork … if you didn't, like for me, it should already have been there. If the first three octets of the IP it gets don't match the rest of your VMs, you've set up your VM's network incorrectly, and need to check out its settings before you continue. Note that like a physical computer, you don't need to reboot a VM to re-wire its network. If you have to make changes to fix the network on your VM, just off/on the ethernet adapter here on this screen until you get an IP on the right subnet.

   Click the "configure" button, then in the next screen that pops up, click on the "general" tab and make sure that "connect automatically" is selected. Then, click on the IPv4 Settings tab, and change "method" to "manual." Click the "add" button to add an IP address, the IP ending with .73 on your network, as you entered into `/etc/hosts` on your host operating system, and into the name server on your lab 1 VM for the Rocky Linux VM instance. The netmask should be `24` – this is the number of bits in a 255.255.255.0 netmask, and enter the same default gateway that you've been using with your other VMs. Enter the IP of your OpenBSD VM (which should always be up and running at this point!) as your DNS server, and cs470.internal as your search domain.

   What you have after you're done with the IPv4 Settings pane should look like the screenshot below.



Go over to IPv6 Settings and change the method to “disabled.” Click “save.”



Finally, to complete the network setup, set the hostname of your VM to `rocky` and click “apply.” You should have something like the screenshot above, though your IP addresses may differ a little depending on the network VMware gave you, of course. Now click “done,” in the upper left hand corner, to go back to the main installer window.

2. Select “time and date” and set your timezone. We want “Americas/Los Angeles.” In the upper right-hand corner of the screen, you’ll see a widget to turn on network time, and as we’ve seen, time-keeping in VMs is very important. Turn it on, if it’s not already, and hit “done.”

3. The “installation source” should be “auto-detected installation media” … this is the ISO file you downloaded, pretending to be an optical disc. Since we grabbed the minimal DVD, you should only have the choice of the “minimal” repository.

   If “minimal” shows as an “additional repository,” it’s going to complain and refuse to select to install. If that’s the case, select to use the network (“on the network”) first (the second radio button in the top part), then change back to “auto-detected installation media.” Hit the button to “verify” the installation media. It should clear out the “additional repositories” pane, allowing you to select the “auto-detected installation media” again and then click “done.”

   Back at the root installer menu, select “Software Selection” then make sure that only “minimal install” is selected. Click on “done” again.

4. Select “installation destination” … you may notice that it wants to automatically configure partitioning … and that I’ve typically viewed this with scrutiny in the past.

Select to make a “custom” storage configuration, make sure your virtual hard disk is selected, then click “done.” If it didn’t take you to the manual partitioning screen, try it again … two of us had to do this twice to get there. Not very intuitive. Once you’ve arrived at the partitioning screen, change the partitioning scheme from LVM (logical volume manager) to “standard partition.”

‼ NOTE: LVM abstracts filesystems from actual physical disks, and might be helpful with real machines, but here in a VM, it’s an unnecessary bonus layer of abstraction in a device with already-virtual disks.

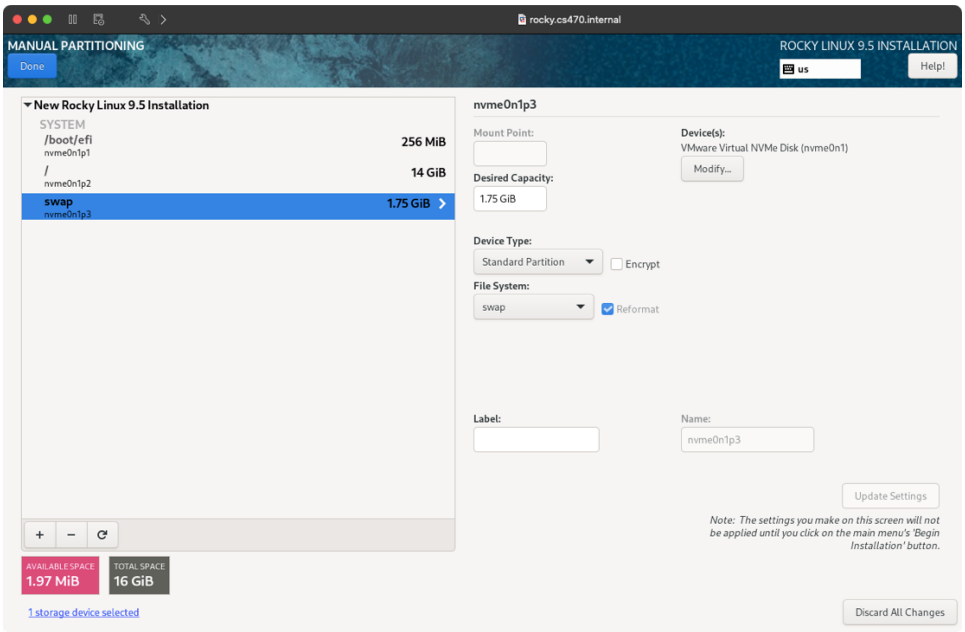
First, you will need to use the plus sign button to create a 256 MiB partition first, mounted on `/boot/efi`. The installer may realize right away to make the file system an EFI system partition, so you may not need to click the drop-down menu to set the filesystem yourself.

Next, use the plus sign button again to create a new partition (the installer confusingly says “mount point” here) with the mount point `/` and a desired capacity of "14 GiB" and click “add mount point.” Select to not encrypt it, and change it from the default xfs filesystem to ext4.

‼ IMPORTANT NOTE: in case you hadn’t been introduced to this particular law of physics, **attorneys ruin everything**. Did you notice the lower-case “i” in between “G” and “B?” This means we’re using gibibytes, not gigabytes. The difference, you ask? Base 10 vs. base 2. See this Wikipedia article for more information on that.

If you feel so inclined, remove your root filesystem, and re-create it with “GB” as units and check out the difference in size. **DO**, however, use 14 GiB, **not** 14 GB as the final size of your root partition.

Do the plus sign again to create another partition, and use the drop-down menu to select “swap” … which is not really a mount point at all, and use all remaining space available, about 1.75 GiB, for swap space.



Your manual partitioning screen should look like the above screenshot. If it looks good, click "done." You'll be asked to confirm your changes; you may "accept changes" quickly and without hesitation, since this is a virtual disk in a VM and a blank one at that, not a real disk.

5. Back at the main installation window, the RHEL installer is just now getting with accepted best practices, and is now allowing you to leave the root account locked. This has been the default on Ubuntu for a long time; Ubuntu doesn't even ask if you want to set a root password.

   Please, however, **do** take a moment to set a root password, and then begin the installation, but **don't** create a user yet. We'll create a non-root user shortly after the first reboot, because we want to take this opportunity to show off command line tools for user provisioning.

   When the installation is complete, you should see a screen letting you know, and waiting for you to click to reboot into the installed system. As always, while it reboots, make sure the ISO is removed from your VM's virtual optical drive.

Just like with BSD, always tell your Linux VMs you want them to shut down or reboot cleanly dismount all filesystems before using any VMware or UTM interface widgets or menus to change their power state. All operating systems in our labs have a `reboot` command. Like FreeBSD, there is a `poweroff` command, which requires no switches. There is also a `shutdown` command, but the switches behave slightly differently with the implementation of shutdown typically used in Linux: `-h` powers the system off, and so does `-P`, whereas in BSD a lowercase `-p` is used to `poweroff`, and `-h` halts but doesn't power the system down.

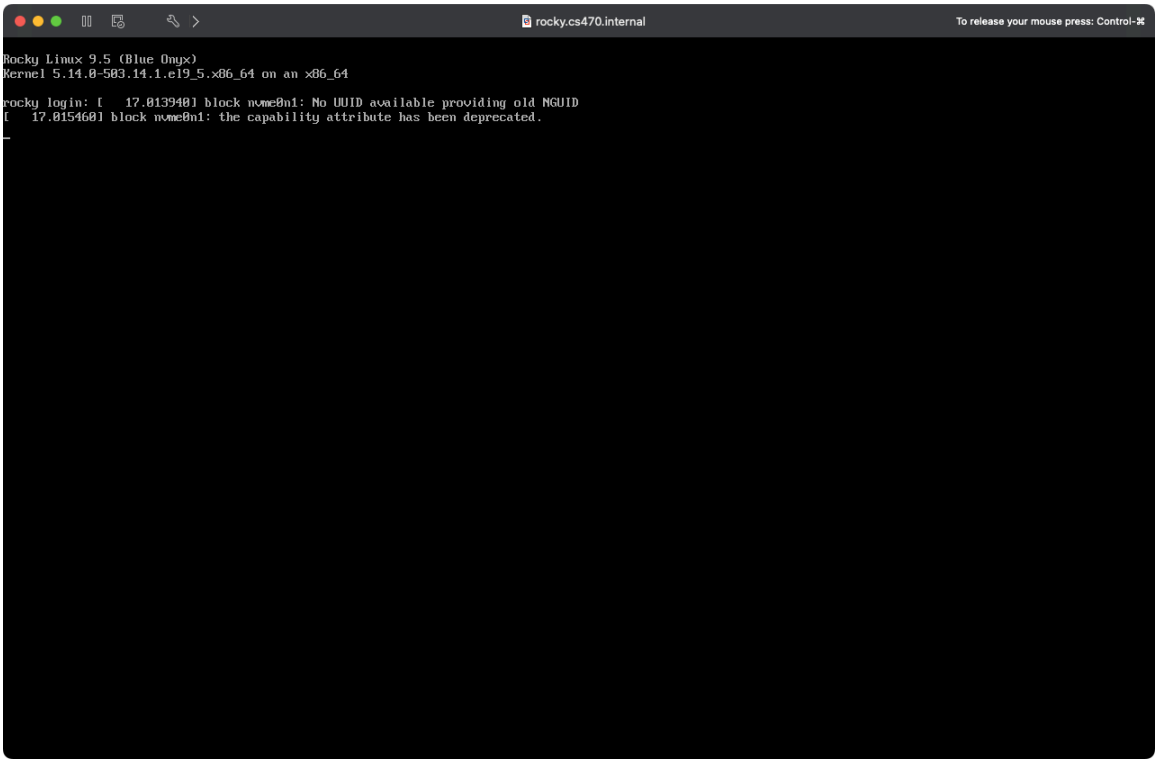
part two: configure it

Indeed, configuration is over as the exception, rather than a rule. As we go through our lab operating systems to generally easier and more full-featured OSs, many things we had to do post-install are now done for us during the installation. We still have a few things to set up, though.

6. During the reboot, you'll briefly see the `grub` boot loader menu again. This is the boot loader that corresponds to the OpenBSD boot prompt and the FreeBSD boot menu. Do nothing here; it will timeout and do the default boot configuration, which is what we want here. I'm just bringing your attention to a familiar feature, in a new OS … we want to fully boot the system anyways.

   If you did want to get to the `grub` menu, you'd hold down the left `shift` key with legacy BIOS firmware. With EFI/UEFI, you'd use the `ESC` key.

   When it's done rebooting, you should see something like this …



… we got and used graphics for the installation because we weren't given a choice, but none afterwards, thank you very much. Note how my console window, above, received log messages after the login prompt … the login prompt remains there nonetheless, and the log messages aren't also waiting for input. If for some reason this bugs you, or if for some reason it lags before giving you a login prompt, try pressing the return key on your keyboard.

Log in as root and create your non-root user account with the same `failsafe` name as you used on your OpenBSD and FreeBSD VMs.

```
# useradd -mU -G wheel -s /bin/bash failsafe
```

Obviously inspired by Berkeley Unix, Red Hat Linux and its derivatives use the `wheel` group to allow access to `sudo` by default, so no additional work is required here. Check out the output of `id failsafe` and note that `useradd` creates a personal group for each user by default. Also run `ls -la ~failsafe` and see that `useradd` creates the user's home directory with default files from `/etc/skel` like we showed on EDORAS in lab 0.

EDORAS runs Oracle Linux 8.x, thus it is also descended from Red Hat Enterprise Linux, and a lot of userland is configured or behaves in a very similar fashion, even though it's now a major version of the operating system behind.

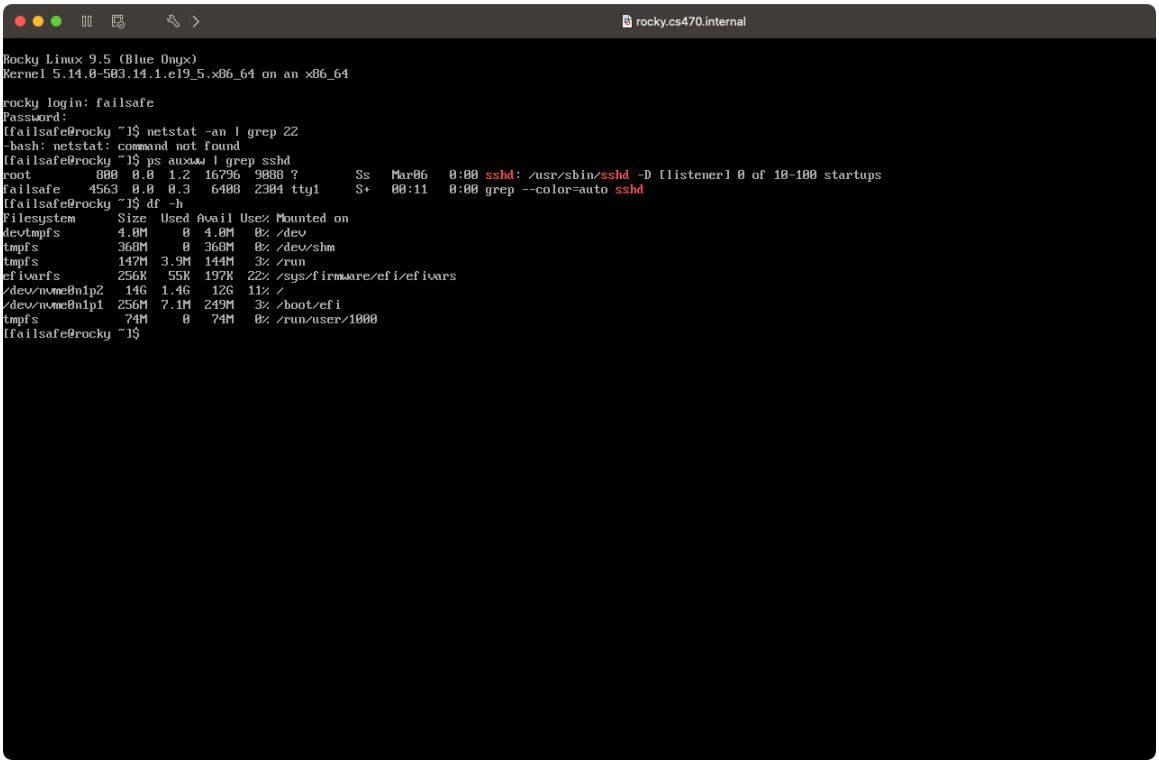
Now set the failsafe account's password.

```
# passwd failsafe
```

7. Log back in as `failsafe`.

   I wanted to see if SSH is running, so I used `netstat`, like in the prior labs … but got an error message saying `command not found`. We're getting what we asked for here, with a minimal installation. For Linux, a 1.4 GB installation footprint is fairly small.

   Since `netstat` wasn't available, I used the `ps` command to verify that SSH was running, and the line with `/usr/bin/sshd` ... that's it, and it's running.



8. Create your SSH directory, with 700 permissions …

   ```
   $ mkdir -m 700 ~/.ssh
   ```

   … and after you've created your `.ssh` directory, copy over your SSH key into the `authorized_keys` file so that you can use SSH and minimize the console window.

9. You'll probably be happy to know that `gpg` is already installed, even in the minimal configuration of Rocky. This is because it is used to sign and verify binary packages, which is the typical method of distributing software in Rocky Linux, both in the base operating system and third-party software.

   ```
   $ which gpg
   /usr/bin/gpg
   ```

10. Speaking of, let's configure the package manager for Rocky Linux, `dnf` ... type `sudo dnf update` …

    ```
    $ sudo dnf update
    ```

… first it will update its local cache of the packages available in each repository …

```
Rocky Linux 9 - BaseOS                                   35 kB/s | 2.2 MB     01:05
Rocky Linux 9 - AppStream                               1.8 MB/s | 7.9 MB     00:04
Rocky Linux 9 - Extras                                  242  B/s |  15 kB     01:01
```

… then it presents a full list of what it intends to do, for your approval …

```
Dependencies resolved.
================================================================================
 Package                 Architecture Version                 Repository    Size
================================================================================
Installing:
 kernel                  x86_64       5.14.0-503.26.1.el9_5   baseos       2.0 M
Upgrading:
 NetworkManager          x86_64       1:1.48.10-5.el9_5       baseos       2.3 M
 NetworkManager-libnm    x86_64       1:1.48.10-5.el9_5       baseos       1.8 M
 NetworkManager-team     x86_64       1:1.48.10-5.el9_5       baseos        39 k
 NetworkManager-tui      x86_64       1:1.48.10-5.el9_5       baseos       246 k
 bzip2-libs              x86_64       1.0.8-10.el9_5          baseos        39 k
 cronie                  x86_64       1.5.7-12.el9_5          baseos       115 k
 cronie-anacron          x86_64       1.5.7-12.el9_5          baseos        31 k
 firewalld               noarch       1.3.4-9.el9_5           baseos       452 k
 firewalld-filesystem    noarch       1.3.4-9.el9_5           baseos       8.6 k
 grub2-common            noarch       1:2.06-93.el9_5         baseos       903 k
 grub2-efi-x64           x86_64       1:2.06-93.el9_5         baseos       1.3 M
 grub2-tools             x86_64       1:2.06-93.el9_5         baseos       1.8 M
 grub2-tools-minimal     x86_64       1:2.06-93.el9_5         baseos       603 k
…
 kernel-tools            x86_64       5.14.0-503.26.1.el9_5   baseos       2.3 M
 kernel-tools-libs       x86_64       5.14.0-503.26.1.el9_5   baseos       2.0 M
 libbrotli               x86_64       1.0.9-7.el9_5           baseos       312 k
 libgcc                  x86_64       11.5.0-5.el9_5          baseos        84 k
…
 sssd-client             x86_64       2.9.5-4.el9_5.4         baseos       161 k
 sssd-common             x86_64       2.9.5-4.el9_5.4         baseos       1.6 M
 sssd-kcm                x86_64       2.9.5-4.el9_5.4         baseos       109 k
 tzdata                  noarch       2025a-1.el9             baseos       429 k
Installing dependencies:
 freetype                x86_64       2.10.4-9.el9            baseos       387 k
 graphite2               x86_64       1.3.14-9.el9            baseos        94 k
 grub2-tools-efi         x86_64       1:2.06-93.el9_5         baseos       540 k
 grub2-tools-extra       x86_64       1:2.06-93.el9_5         baseos       840 k
 harfbuzz                x86_64       2.7.4-10.el9            baseos       623 k
 kernel-core             x86_64       5.14.0-503.26.1.el9_5   baseos        18 M
 kernel-modules          x86_64       5.14.0-503.26.1.el9_5   baseos        36 M
 kernel-modules-core     x86_64       5.14.0-503.26.1.el9_5   baseos        30 M
 libpng                  x86_64       2:1.6.37-12.el9         baseos       116 k
```

… your results may vary slightly, especially if you're on an ARM CPU, but the general idea is the same. We're telling `dnf` to grab the latest list of packages, and it will come back and tell us it's got a few updates from between the time that ISO we used to install was published and the current date and time.

```
Transaction Summary
================================================================================
Install  10 Packages
Upgrade  67 Packages

Total download size: 604 M
Is this ok [y/N]:
```

Type `y` to allow it to upgrade, and it will go out to the Rocky Linux repositories, and download all the updated packages. You'll be asked near the end to accept the new official GPG signing key …

```
Rocky Linux 9 - BaseOS                                  1.6 MB/s | 1.7 kB     00:00
Importing GPG key 0x350D275D:
 Userid     : "Rocky Enterprise Software Foundation - Release key 2022 <releng@rockylinux.org>"
 Fingerprint: 21CB 256A E16F C54C 6E65 2949 702D 426D 350D 275D
```

```
 From       : /etc/pki/rpm-gpg/RPM-GPG-KEY-Rocky-9
Is this ok [y/N]: y
```

… as with other cryptographic signatures going back to lab 1, you could use the fingerprint to validate the key manually before accepting it. If your terminal shows the same key fingerprint as my output above did, you just validated the key through a peer and can again type `y` to allow it. It will then validate and install all the package updates.

‼ INTERESTING NOTE: Ubuntu Linux includes a feature called "livepatch" which does away with the need to reboot to patch your kernel. Of course, however, this is a paid feature … not for free users. Yes, I know this is the Rocky Linux lab, but thought it cool enough to mention here.

When/if your VM receives a new kernel version from `dnf`, you will clearly see a line in `dnf` output spell out `kernel` or `linux` something – mine did above – and when it does, the new kernel will be added as an option to the `grub` boot loader menu. If your new kernel has a problem, the `grub` menu is just one place where you could select to boot (one time) with a different kernel, to do process-of-elimination troubleshooting. Run `man -k grub` or `apropos grub` to see all the commands associated with the `grub` bootloader.

Run this command to see if you've gotten a new kernel configured in `grub` …

```
$ sudo grubby --default-title
```

… compare the version number segment of the output to the output of `uname -a` … if the version numbers match, you're running the same version of the kernel that's configured for the next boot. If they don't match, you've gotten a new kernel and you need to reboot to run the operating system on top of this new kernel instead. If you got a kernel update when you did this step, go ahead and `reboot`, but before you do, run `df -h` and note the device file names of your `/boot/efi` and root filesystems.

I ask you to do this because after rebooting my Rocky VM in UTM on my ARM Mac, it hung at the first stage loader with a "synchronous exception" and failed to boot. Please let me know if this happens to you. I needed to put the ISO image back in, and boot off of that to go into troubleshooting, then rescue/recovery mode.

If it looks like your computer is hung at a blank screen, that's because the Rocky installation ISO's rescue/recovery mode is just broken … hit `ctrl-option-fn-F2` or `F3` to flip to another text terminal like we discussed briefly in prior labs and you should get a command line. Once you have a command line, use `find` to locate `grubaa64.efi` in the rescue environment, then manually `mount` your `/boot/efi` filesystem and `cp` that file to wherever `/boot/efi/EFI/rocky/grubaa64.efi` would be on your virtual hard disk and `reboot` to get back up and running.

Fortunately, Red Hat, Rocky, or whomever was at fault here seems to have fixed this problem this time around, but I'm leaving the above text there just in case.

11. Note how `uname` only tells us that we're on Linux, not Rocky Linux. The `el` in the extended kernel version information will tip off experienced users – and hopefully you now too, after reading this – that we're running Red Hat Enterprise Linux or a derivative of it, but not which one.

    `uname` was intended, in the early days of Unix, to provide a userland utility to identify the local operating system by name and version number. In all Unix-like operating systems prior to Linux, the name and version of the kernel **was** the name of the operating system. As Linux is just a kernel, it broke that paradigm. A tool was still needed to identify the kernel, however, so `uname` kept that one task.

    Over time, the Linux filesystem hierarchy standard – yes, there is one – grew to address this need, with a standard for metadata files to identify the current operating system, version, and even the operating system's lineage, if it's a descendant of another operating system, just like Rocky and Red Hat. See `man os-release` and the contents of the file `/etc/os-release`. Both major Linux distributions we deploy in this class put a file at `/etc/os-release`.

    FreeBSD has lined up with this standard too, and leaves OS metadata at `/etc/os-release` as well! Take a look at this file on both operating systems, now!

12. Let's install some common packages that you might find useful, and I'll be using later to grade your work: `wget`, `csh` (via `tcsh`), `host` (via `bind-utils`), `binutils`, and `ifconfig` and `netstat` (via `net-tools`) on Rocky Linux.

    ```
    $ sudo dnf install wget tcsh net-tools binutils bind-utils
    ```

13. As with our other VMs, let's make sure our system can send mail to our mail server on the FreeBSD VM. You might remember me mentioning `postfix`, the more modern mail server … that's what we're going to use, and it was left out of our minimalist installation. Let's install it.

    ```
    $ sudo dnf install postfix
    ```

    As I said in lab 2 and you'll shortly see, `postfix` is far, far easier to configure than `sendmail`. We are trying to do less with `postfix` here than we did we `sendmail`, but despite that, let's back up the main `postfix` configuration file as a matter of best practice.

    ```
    $ sudo cp -p /etc/postfix/main.cf /etc/postfix/main.cf.orig
    ```

    Then, let's edit the configuration file …

    ```
    $ sudo vi /etc/postfix/main.cf
    ```

    … and set the following values under the areas for each variable's sample:

    ```
    myhostname = rocky.cs470.internal
    mydomain = cs470.internal
    myorigin = $myhostname
    ```

```
inet_protocols = ipv4
```

… once you're done, save out the file, and let's enable and start up `postfix`.

```
$ sudo systemctl enable postfix
$ sudo systemctl start postfix
```

See, much easier than `sendmail`, no?

We have also introduced you here to `systemctl`, a component of `systemd` and Linux's corresponding tool to OpenBSD's `rcctl`. `rcctl` is actually derivative of `systemctl` and a product of OpenBSD trying to modernize itself a bit … this is evident when you see that FreeBSD has no such tool or built-in alternative. This is a rare place where OpenBSD is more helpful than FreeBSD.

14. Let's fix the aliases database, to make sure we get e-mail intended for the system administrator. In Rocky Linux (locations can and will vary from Linux distribution to Linux distribution), it's located at `/etc/aliases`.

    ```
    $ sudo vi /etc/aliases
    ```

    Find the line aiming root's e-mail (the last line in my file) and replace the default with your @cs470.internal e-mail address from lab 2. Save the file and …

    ```
    $ sudo newaliases
    ```

    … tell the mail subsystem to re-read the alias database.

15. Some of you might have already noticed that because we're running the "minimal" installation of Rocky Linux here, it's not provided us the `mail` command we were using to test e-mail connectivity between our VMs. `dnf` has a solution for that …

    ```
    $ dnf whatprovides mail
    ```

    `dnf` will go through grabbing some databases the first time you run this command, but if you wanted to know which package to install to add a particular command, this is one way to do. `dnf` told me:

    ```
    s-nail-14.9.22-6.el9.x86_64 : Environment for sending and receiving mail
    repo        : appstream
    Matched from:
    Filename    : /usr/bin/mail
    Provide     : /bin/mail
    ```

    So I installed it …

    ```
    $ sudo dnf install s-nail
    ```

    … and was able to use `mail` to test mail transmission in between systems, like in lab 2.

‼ IMPORTANT NOTE ‼ Just about every Unix and Unix-like OS will have logging for its mail subsystem and **all** its subsystems, somewhere. I recommend perusing the file or `/etc/rsyslog.conf` and `man rsyslog`. `rsyslog` is a favorite modern `syslogd` replacement for a few modern Linux distributions, and we'll be using it more in lab 5.

16. We discussed `htop` in class, an improved version of the `top` utility for process monitoring. `htop` isn't found in the default Rocky Linux repositories.

```
$ dnf search htop
Last metadata expiration check: 1:11:50 ago on Fri 07 Mar 2025 11:54:33 AM PST.
No matches found.
```

Let's install the EPEL repository; EPEL is short for "extra packages for enterprise Linux," and for many years now has been the storehouse for additional software not included in the mainline repositories. Let's add it to `dnf` …

```
$ sudo dnf -y install epel-release && sudo dnf -y check-update
```

```
…
================================================================================================
 Package                    Architecture        Version                 Repository          Size
================================================================================================
Installing:
 epel-release               noarch              9-7.el9                 extras              19 k

Transaction Summary
================================================================================================
Install  1 Package

Total download size: 19 k
Installed size: 26 k
Downloading Packages:
epel-release-9-7.el9.noarch.rpm                                 77 kB/s |  19 kB     00:00
------------------------------------------------------------------------------------------------
Total                                                           40 kB/s |  19 kB     00:00
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :                                                                       1/1
  Installing       : epel-release-9-7.el9.noarch                                           1/1
  Running scriptlet: epel-release-9-7.el9.noarch                                           1/1
Many EPEL packages require the CodeReady Builder (CRB) repository.
It is recommended that you run /usr/bin/crb enable to enable the CRB repository.

  Verifying        : epel-release-9-7.el9.noarch                                           1/1

Installed:
  epel-release-9-7.el9.noarch

Complete!
Extra Packages for Enterprise Linux 9 - x86_64                  10 MB/s |  23 MB     00:02
Extra Packages for Enterprise Linux 9 openh264 (From Cisco) - x86_6 2.8 kB/s | 2.5 kB     00:00
```

Lots of useful third-party packages are routinely found in EPEL; I usually add EPEL to all the instances of Red Hat and its derivatives that I install. The note about needing to enable the CRB repository is new-ish. I have yet to run into any packages I've wanted that need that repo turned on.

Please also note how this repository … is installed as a package. It was a whole whopping 19k of download, because it's just the configuration files for the repository. After it's done installing, please check out the folder `/etc/yum.repos.d` …

```
$ ls -l /etc/yum.repos.d
total 1
-rw-r--r--. 1 root root 1142 Aug 27  2023 epel-cisco-openh264.repo
-rw-r--r--. 1 root root 1453 Aug 27  2023 epel.repo
-rw-r--r--. 1 root root 1552 Aug 27  2023 epel-testing.repo
-rw-r--r--. 1 root root 6610 Oct 31 20:27 rocky-addons.repo
-rw-r--r--. 1 root root 1165 Oct 31 20:27 rocky-devel.repo
-rw-r--r--. 1 root root 2387 Oct 31 20:27 rocky-extras.repo
-rw-r--r--. 1 root root 3417 Oct 31 20:27 rocky.repo
```

… this is where repository configurations are stored. Note the base package repositories (`rocky-*`) and the new EPEL repository files. Use `more` to page through a couple of the repository configuration files, and note how we configure just a few things for each repo: an HTTP URL for the repo, the location of a key for validating package signatures, and not much else.

In the meantime, however, we can now install `htop`.

```
$ sudo dnf -y install htop
…
Dependencies resolved.
================================================================================
 Package              Architecture        Version                 Repository      Size
================================================================================
Installing:
 htop                 x86_64              3.3.0-1.el9             epel           198 k
Installing dependencies:
 hwloc-libs           x86_64              2.4.1-5.el9             baseos         2.1 M

Transaction Summary
================================================================================
Install  2 Packages

Total download size: 2.3 M
Installed size: 3.5 M
Downloading Packages:
(1/2): hwloc-libs-2.4.1-5.el9.x86_64.rpm                 5.0 MB/s | 2.1 MB     00:00
(2/2): htop-3.3.0-1.el9.x86_64.rpm                       157 kB/s | 198 kB     00:01
--------------------------------------------------------------------------------
Total                                                    1.3 MB/s | 2.3 MB     00:01
Extra Packages for Enterprise Linux 9 - x86_64           1.6 MB/s | 1.6 kB     00:00
Importing GPG key 0x3228467C:
 Userid     : "Fedora (epel9) <epel@fedoraproject.org>"
 Fingerprint: FF8A D134 4597 106E CE81 3B91 8A38 72BF 3228 467C
 From       : /etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL-9
Key imported successfully
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :                                                        1/1
  Installing       : hwloc-libs-2.4.1-5.el9.x86_64                          1/2
  Installing       : htop-3.3.0-1.el9.x86_64                                2/2
  Running scriptlet: htop-3.3.0-1.el9.x86_64                                2/2
  Verifying        : htop-3.3.0-1.el9.x86_64                                1/2
  Verifying        : hwloc-libs-2.4.1-5.el9.x86_64                          2/2

Installed:
  htop-3.3.0-1.el9.x86_64                hwloc-libs-2.4.1-5.el9.x86_64

Complete!
```

Note that we had to import a new package signing key, and that package key is from the Fedora Project. Fedora is Red Hat's upstream "bleeding edge" distribution; it moves rapidly, is less stable for development and day-to-day use, and is used as a testing ground for features and new code before they get rolled into RHEL.

The EPEL repository and the Fedora project are both run by Red Hat itself, and require no subscription.

Since Rocky Linux is 100% binary compatible, the packages from EPEL work just fine with Rocky, and there's no need for the Rocky Linux project to set up its own compatible repository.

part three: LDAP

Now let's get our LDAP directory plugged into our new instance.

!! THE WARNINGS FROM BEFORE ABOUT LOCKING YOURSELF OUT OF YOUR VM APPLY AGAIN HERE.

17. As usual, we need to pull over our CA certificate from the OpenBSD VM for the LDAP client.

```
$ scp -p openbsd:/home/pki/data/cacert.pem ~failsafe
```

Now let's look what we're working with here, for chains of trust …

```
$ ls -l /etc/ssl/
total 0
lrwxrwxrwx. 1 root root 49 Aug 21  2024 cert.pem -> /etc/pki/ca-
trust/extracted/pem/tls-ca-bundle.pem
lrwxrwxrwx. 1 root root 18 Aug 21  2024 certs -> /etc/pki/tls/certs
lrwxrwxrwx. 1 root root 28 Aug 21  2024 ct_log_list.cnf ->
/etc/pki/tls/ct_log_list.cnf
lrwxrwxrwx. 1 root root 24 Aug 21  2024 openssl.cnf -> /etc/pki/tls/openssl.cnf
```

… and what do you know … it's symlinked to somewhere else, but Rocky Linux also has a CA certificate bundle at `/etc/ssl/cert.pem`. It turns out this bundle is dynamically generated. This post led me to the directory `/etc/pki/ca-trust/source` where there is a `README` file:

```
This directory /etc/pki/ca-trust/source/ contains CA certificates and
trust settings in the PEM file format. The trust settings found here will be
interpreted with a high priority - higher than the ones found in
/usr/share/pki/ca-trust-source/.

================================================================================
QUICK HELP: To add a certificate in the simple PEM or DER file formats to the
            list of CAs trusted on the system:

            Copy it to the
                    /etc/pki/ca-trust/source/anchors/
            subdirectory, and run the
                    update-ca-trust
            command.

            If your certificate is in the extended BEGIN TRUSTED file format,
            then place it into the main source/ directory instead.
================================================================================

Please refer to the update-ca-trust(8) manual page for additional information.
```

First, let's create a reference / backup copy of our root CA certificate at `/etc/ssl/cacert.pem` as we have done previously.

```
$ sudo mv ~failsafe/cacert.pem /etc/ssl/cacert.pem
$ sudo chown root:root /etc/ssl/cacert.pem
```

Now `sudo cp` that root CA certificate into `/etc/pki/ca-trust/source/anchors/` then run the command `update-ca-trust` as suggested by the `README` file.

18. In what should be a completely unsurprising next move, we're going to set up the OpenLDAP client, yet again. First, let's query the package manager, and make sure it's not already installed. You've seen `dnf`, which is a "smart" Python-based front-end to the Red Hat package manager `rpm`. When we want to query what's installed, we do something like this, and just happen to discover that OpenLDAP is already installed.

```
$ rpm -qa | grep -i ldap
openldap-2.6.6-3.el9.x86_64
```

`dnf`, which you've already met, is a full-fledged package management system. You may have noticed `.rpm` at the end of the packages `dnf` was downloading in the initial update step above. RPM stands for Red Hat Package Manager, because often times in software when we figure out a better way of doing something, we don't throw away everything we used before … we just build on top of it.

`dnf` is actually "Dandified YUM," is the successor to `yum`, an acronym too, which stands for Yellow Dog Updater, Modified … because the original Yellow Dog Linux Updater was called ... you guessed it: `yup`. Yellow Dog Linux was a Red Hat-based distribution that was really popular on the PowerPC architecture, and notably used to get Linux running on the PlayStation 3. It's not much in use anymore, but lives on through widespread use of `yum`, which was just the best tool to date for managing RPM packages, often just called "RPMs." An `rpm` tool is there, underneath it all … but we won't use it much, because it doesn't do much, and it's why they made `yup`, `yum`, and `dnf` to do more useful and complicated things on top of it.

Enough story time, though … again, that output above told us some OpenLDAP package is already installed … let's find out which.

```
$ dnf search openldap
…
================================ Name Exactly Matched: openldap ================================
openldap.x86_64 : LDAP support libraries
openldap.i686 : LDAP support libraries
================================ Name & Summary Matched: openldap ================================
collectd-openldap.x86_64 : OpenLDAP plugin for collectd
================================== Name Matched: openldap ==================================
openldap-clients.x86_64 : LDAP client utilities
openldap-compat.x86_64 : Package providing legacy non-threaded libldap
openldap-compat.i686 : Package providing legacy non-threaded libldap
openldap-devel.x86_64 : LDAP development libraries and header files
openldap-devel.i686 : LDAP development libraries and header files
openldap-servers.x86_64 : LDAP server
================================ Summary Matched: openldap =================================
lua-ldap.x86_64 : LDAP client library for Lua, using OpenLDAP
…
```

This output tells us we have LDAP support libraries, but not the client utilities. We certainly need those, and we also need the `nss_ldap` package that we used with FreeBSD in lab 2, or something like it to connect the operating system to LDAP for user and group data. Red Hat's own documentation about configuring LDAP authentication used to refer to it for older versions of RHEL, but now most Linux is getting behind `sssd`, a "One Ring to Rule Them All" system security services daemon. In fact,

this is exactly what `sssd` stands for, and you might have thought that preamble channeling the Lord of the Rings was out of place, but it was not.

First, install `openldap-clients` with `dnf`, and then look at `/etc/nsswitch.conf`, since it was mentioned in the Red Hat guide for configuring LDAP authentication. Note the lines for `passwd` and `group`, which we configured in lab 2. They mention `sss`, which according to the preamble comments for `nsswitch.conf` refer to `sssd`.

```
$ dnf search ldap | grep sssd
sssd-ldap.x86_64 : The LDAP back end of the SSSD
```

Bingo. `sssd-ldap` is exactly what we need. Looking through the output of `dnf search sssd`, we also want to install `sssd-tools` to be able to test and probe `sssd`, and `sssd-nfs-idmap`, since we'll be setting up NFS after we get LDAP up and running. Go ahead and install all three packages with `dnf`.

19. Configure `sssd-ldap`. You will need to use `sudo` to create the `sssd` configuration file. I highly recommend that you check out `man sssd`, but will give you this one. Populate the file `/etc/sssd/sssd.conf` with the following contents …

```
[sssd]
config_file_version = 2
services = nss, pam
domains = default

[nss]
homedir_substring = /home

[pam]

[domain/default]
id_provider = ldap
auth_provider = ldap
chpass_provider = ldap
cache_credentials = False
ldap_tls_cacert = /etc/ssl/cert.pem
ldap_search_base = dc=cs470,dc=internal
ldap_uri = ldaps://openbsd.cs470.internal/
```

… if you don't know what most of the above configuration means at this point, take this as a wake-up call – you're probably sleepwalking and/or not reading the labs. Please use a web search to look it up. We have configured all but the `chpass_provider` (unsurprisingly a hook for changing passwords) and the caching options before (not mentioned / also not needed in prior LDAP clients).

```
$ sudo chmod 600 /etc/sssd/sssd.conf
```

Now test your `sssd` configuration with the tool for doing just that. If your `sssd` configuration checks out, you should see something like this.

```
$ sudo sssctl config-check
Issues identified by validators: 0

Messages generated during configuration merging: 0
```

```
Used configuration snippet files: 0
```

20. Configure the LDAP client and the LDAP client libraries in `/etc/openldap/ldap.conf` … you should know what to do by now … back it up, and you can probably just copy over your configuration file from FreeBSD.

21. Now, configure `/etc/ldap.conf` … this file has the same format as the one in the prior step, so let's just create a hard link at that place in the filesystem again.

```
$ sudo ln /etc/openldap/ldap.conf /etc/ldap.conf
```

22. Restart `sssd`.

```
$ sudo systemctl restart sssd
```

Test. You should now be able to see your LDAP user with `id`.

If you don't, I recommend popping a root shell. Here are some commands I found useful during troubleshooting. Note that all the below are preceded by a `#` prompt, because you either need to be root or use `sudo` to query `sssd`.

To get `sssd` to check out its configuration files (same as a couple steps ago):

```
# sssctl config-check
```

To query `systemctl` for the status of `sssd` including any obvious errors:

```
# systemctl status sssd
```

To query `sssd` once it's running about connected authentication domains:

```
# sssctl domain-list
# sssctl domain-status default
```

To query `sssd` about your LDAP user and group:

```
# sssctl user-show peter
# sssctl group-show peter
# sssctl user-checks peter
```

To get `sssd` to flush its whole cache, or individual users from its cache:

```
# sssctl cache-remove
# sssctl cache-expire --user peter
```

To bump up `sssd` logging levels:

```
# sssctl debug-level 0x07f0
```

`sssd` logs are found in `/var/log/sssd`.

23. Make your LDAP user's home directory, and make sure they own it.

```
$ sudo mkdir /home/peter
$ sudo chown peter:peter /home/peter
```

Because your LDAP user and group IDs already resolve to the associated names, the second command should work without issue.

24. Edit `/etc/sudoers` and replicate the group line for the wheel group to make a line granting `sudo` rights to the sudoers group as well.

25. Create a symbolic link for `bash` so that it can be found at the location defined with your LDAP user …

    ```
    $ sudo ln -s /bin/bash /usr/local/bin/bash
    ```

    … and add `/usr/local/bin/bash` to the list of acceptable shells in `/etc/shells`.

26. Enable `sssd` for authentication. `authselect` is Red Hat's tool for configuring the PAM setup files in `/etc/pam.d` and `/etc/nsswitch.conf`.

    ```
    $ sudo authselect select --force sssd
    ```

    `authselect` makes backup files, in case you need to back out; see `man authselect`.

27. Test!

    Test logging into the console of your VM, in its VMware or UTM window, as both failsafe and your LDAP user.

    Try logging in via SSH as both.

    Test using `sudo`. Note that you can just use `sudo` to run something harmless like `ls` to test that it doesn't throw any errors and just works, even though you don't *need* rootly powers to run `ls`.

28. Create your LDAP user's `.ssh` directory and set up key-based authentication for your LDAP user on your Rocky VM. Once everything works with your LDAP user, abandon the failsafe user as usual.

part four: NFS server

As stated in the introduction, this Rocky Linux instance is going to become an NFS server, and the file server for all the VMs on our network. It's going to share `/home` from this machine to all file service clients, so that we have the same home directory, the same files, no matter which VM we log into. We're also going to create some other directories under `/home` to allow other things to be shared between our VMs.

The only VM that's going to be left out of the party here is our OpenBSD VM, so that we don't introduce a circular dependency. Our Rocky VM isn't going to come up gracefully without DNS and LDAP from our OpenBSD VM, and we can't put an NFS share on our Rocky VM in our OpenBSD VM's `/etc/fstab`, or Rocky will have to be up for our OpenBSD VM to boot. We can live without network home directories on our OpenBSD VMs.

Back to setting up our file server. Fortunately, the NFS server is already built into the minimal Rocky Linux distribution.

29. Let's look for other tools to supplement our NFS service.

    ```
    $ dnf search nfs
    ```

    Note that, like a lot of things, you don't need to be root unless you're actually making/writing changes to disk. You do NOT need to be root in order to search the package repositories. Look at the output generated by `dnf` when you search … you'll notice all packages are tagged with an architecture.

    Packages with `x86_64` as a suffix are 64-bit Intel packages; ones with `aarch64` are 64-bit ARM packages. `noarch` are packages made entirely of configuration files, headers, and/or interpreted scripts, so the architecture is not important in these instances. You probably won't see `i686` 32-bit Intel packages anymore, aside from fringe (cringe?) use cases and repositories. Nobody has sold or bought 32-bit Intel CPUs for a while now, and the architecture has been deprecated, as far as the Red Hat family of Linux distributions is concerned.

    Let's install `nfs-utils`, because I'm fairly sure we'll be using them.

    ```
    $ sudo dnf install nfs-utils
    ```

    Follow the prompts, and make sure it installs.

30. `systemctl` is the tool for controlling `systemd`, the subsystem responsible for managing services and system startup in current versions of Linux, as discussed earlier in lecture. Tell `systemd` to start an NFS server at boot time …

    ```
    $ sudo systemctl enable nfs-server.service
    ```

    … and tell it to fire it up immediately.

    ```
    $ sudo systemctl start nfs-server.service
    ```

31. RHEL and its derivatives come, by default, with a host firewall. Any services we want to get *into* our Rocky Linux VM will have to be explicitly allowed through the firewall.

    ```
    $ systemctl is-active firewalld
    ```

    The above command asks `systemd` if `firewalld` (the firewall daemon) is active; it should reply `active`, as the firewall is turned on by default.

    Let's see what services are allowed to all hosts in the `firewalld` logical area ("zone") called "public."

    ```
    $ sudo firewall-cmd --info-zone=public
    ```

    My output looked like this:

```
public (active)
  target: default
  icmp-block-inversion: no
  interfaces: ens160
  sources:
  services: cockpit dhcpv6-client ssh
  ports:
  protocols:
  forward: yes
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
```

Let's tell `firewalld` that we also want it to let NFS traffic through. Fortunately, it already knows what NFS is, and what NFS traffic looks like. We also need to include the service `mountd`, which separately tracks requests to mount shared filesystems and the permissions a client has for each share, and `rpc-bind`, the RPC portmapper, which is used to find and register RPC-based services like NFS.

```
$ sudo firewall-cmd --permanent --zone=public --add-service=nfs
$ sudo firewall-cmd --permanent --zone=public --add-service=mountd
$ sudo firewall-cmd --permanent --zone=public --add-service=rpc-bind
```

`firewalld` should reply with `success` for each, or you likely got something wrong. Now tell it to re-load its ruleset.

```
$ sudo firewall-cmd --reload
```

Again, it should reply with `success`. Our firewall is good, and now our NFS traffic will pass through it.

32. Many services built on top of RPC can run on a dynamic port number, over TCP or UDP or both, and can support different versions of a protocol. The port mapper (`rpc-bind` to the host firewall above, but `rpcbind` is the name of its executable) gives us a way of finding out which versions of which services are available, on which ports.

    You can probe the RPC port mapper with the `rpcinfo` utility …

```
$ rpcinfo -p localhost
   program vers proto   port  service
    100000    4   tcp    111  portmapper
    100000    3   tcp    111  portmapper
    100000    2   tcp    111  portmapper
    100000    4   udp    111  portmapper
    100000    3   udp    111  portmapper
    100000    2   udp    111  portmapper
    100024    1   udp  35319  status
    100024    1   tcp  40239  status
    100005    1   udp  20048  mountd
    100005    1   tcp  20048  mountd
    100005    2   udp  20048  mountd
    100005    2   tcp  20048  mountd
    100005    3   udp  20048  mountd
    100005    3   tcp  20048  mountd
    100003    3   tcp   2049  nfs
    100003    4   tcp   2049  nfs
```

```
    100227    3   tcp   2049  nfs_acl
    100021    1   udp  49804  nlockmgr
    100021    3   udp  49804  nlockmgr
    100021    4   udp  49804  nlockmgr
    100021    1   tcp  38917  nlockmgr
    100021    3   tcp  38917  nlockmgr
    100021    4   tcp  38917  nlockmgr
```

As you can see, each RPC-based program has a registered program number, shown in the first column, and a version number, shown in the second column. We're running both versions 3 and 4 of NFS by default, which is customarily fixed on port 2049. The RPC portmapper is always on port 111. After that, the rest here are dynamically assigned and primarily to do with NFS.

The ports shown above by the portmapper line up with the ports that would be shown by `netstat`, another way to confirm the service is running.

```
$ netstat -an | grep 2049
tcp        0      0 0.0.0.0:2049            0.0.0.0:*               LISTEN
tcp6       0      0 :::2049                 :::*                    LISTEN
```

As you can also see, the portmapper leaks a lot of information about local services on our system; we would be wise to limit access to port 111 (and everything else NFS too) on our servers to those clients that actually need it. All the systems on our little lab networks are going to use this NFS service, so that doesn't make sense here. The perimeter firewall provided by our hypervisor's NAT prevents systems outside our lab net from contacting these services, and that is adequate here.

33. Now let's set up the NFS service itself.

```
$ sudo vi /etc/exports
```

The file `/etc/exports` is there, but it has a zero size by default, so everything here we'll be adding in:

```
/home        10.42.77.0/24(rw,sync)
```

This tells the NFS server that we want to share `/home` on the local system (in this case, our Rocky VM) to all the hosts on our private VM network (remember to swap in your network numbers into the first three octets of the IP), to allow read/write access (`rw`), and synchronous I/O (`sync`). Note that there are NO spaces in between the address range and the NFS mount options.

Now, let's share it!

```
$ sudo exportfs -a
```

This tells the NFS service to immediately share everything (`-a`) declared in `/etc/exports`. We can verify this worked with the `showmount` command, which we installed along with the `nfs-utils` package a few steps ago.

```
$ showmount -e localhost
```

This queries the NFS server with a client program (`showmount`), using the loopback network adapter, as discussed in lecture. You should see something like the following:

```
Export list for localhost:
/home 10.42.77.0/24
```

34. Let's make a place to use as a mount point for data shared in between users. We'll use this repeatedly on our file server over the next few labs, in addition to using it for testing and playing with NFS behavior in general. As you can see from the last step, we're going to share out the file tree under `/home`, so let's put this storage device under the mount point `/srv/nfs`.

    `/srv` is a part of the Linux filesystem standard, reserved for data that is served by a system, where the filesystem hierarchy is exposed to clients. This wouldn't apply to our IMAP server in lab 2, because the IMAP server's storage files aren't directly exposed to clients. It also wouldn't apply because lab 2 runs FreeBSD and `/srv` is a part of the Linux filesystem standard, but we are going to apply it there anyways to keep things uniform.

    ```
    $ sudo mkdir /srv/nfs
    ```

    Lots of temporary storage is needed on our file server, to build the Linux kernel from source in lab 4, and to create a clone of our lab 2 VM when piloting backups in lab 6. This data won't be needed if you want to keep your labs after the class is over, so we're going to separate this data that can be thrown out later.

    Shut down your Rocky VM with `sudo poweroff`, and let's create a second 40 GB virtual storage device in our Rocky VM.

    In VMware Workstation, you'll go to your VM's settings, in the hardware tab, and click "add" near the bottom and create a new hard disk. In VMware Fusion, you'll go to your VM's settings and click the "add device" button in the upper right-hand corner. In UTM, edit your VM from the main VM library window, and under "drives," click on "new."

    Beyond setting the size of your local storage to 40 GB, and storing the new virtual storage device's backing files in the same directory with the rest of your Rocky VM files, the defaults for the remainder of settings should be fine.

    Start your Rocky VM back up and log back into it. Use the partition editing tool for GPT partitions to list all your available storage devices and find the name of your second storage device.

    ```
    $ sudo parted -l
    [sudo] password for peter:
    Model: VMware Virtual NVMe Disk (nvme)
    Disk /dev/nvme0n1: 17.2GB
    Sector size (logical/physical): 512B/512B
    Partition Table: gpt
    Disk Flags:

    Number  Start   End     Size    File system     Name                  Flags
     1      1049kB  269MB   268MB   fat16           EFI System Partition  boot, esp
    ```

```
 2      269MB   15.3GB  15.0GB  ext4
 3      15.3GB  17.2GB  1877MB  linux-swap(v1)                           swap

Error: /dev/nvme0n2: unrecognised disk label
Model: VMware Virtual NVMe Disk (nvme)
Disk /dev/nvme0n2: 42.9GB
Sector size (logical/physical): 512B/512B
Partition Table: unknown
Disk Flags:
```

No matter which architecture you're on, what kind of virtual storage device your hypervisor created, or which bus your new virtual storage device is attached to, this much is for sure: it will be the 42.9 GB (remember gigabytes and gibibytes?) storage device that generates an error for an `unrecognized disk label` from `parted`, because it hasn't yet been partitioned. The output above is from my Rocky VM on my Intel Mac, where Fusion prefers to create a virtual NVMe SSD for storage, so it's the second NVMe device `/dev/nvme0n2`. On my ARM Mac, UTM preferred to create a VIrtIO device, so the second VirtIO storage device is `/dev/vdb`. I'm not sure what you'll get on VMware Workstation, but I **am** sure that you'll be able to figure out what to do with the below no matter your device.

35. Create a new partition table on your device …

```
$ sudo parted /dev/nvme0n2 mklabel gpt
```

… then create an ext4 partition to use up all space on the disk. Start as early on the disk as you can (`0`), try to go all the way to the end of the device (`42.9GB`), and feel free to ignore the warning about disk alignment. This would only matter if it were a real storage device …

```
$ sudo parted /dev/nvme0n2 mkpart ext4
[sudo] password for peter:
File system type?  [ext2]? ext4
Start? 0
End? 42.9GB
Warning: The resulting partition is not properly aligned for best performance: 34s
% 2048s != 0s
Ignore/Cancel? I
Information: You may need to update /etc/fstab.
```

… you can use `parted -l` again to check your work. Then, finally, make an ext4 filesystem on that new partition.

```
$ sudo mkfs.ext4 /dev/nvme0n2p1
```

Note that traditional hard disks, or whatever is perceived by the Linux kernel as one of them, use a different device instance and partition naming scheme than NVMe devices. On my ARM Mac, I had to run `mkfs.ext4` on `/dev/vdb1`.

36. Now look at the contents of `/etc/fstab` on your Rocky VM. Note how UUIDs are typically used in place of device filenames under `/dev` here … this is for a very good reason. Which device is detected as the first or second NVMe or SATA device can sometimes change from boot to boot of a computer, or if the insides of a computer need to be re-wired. By using the UUID, we always make sure to get the right partition.

By listing the contents of the directory `/dev/disk/by-uuid` you can easily get the UUID of your new partition.

```
$ ls -l /dev/disk/by-uuid/
total 0
lrwxrwxrwx. 1 root root 15 Mar  9 14:17 018B-AD8C -> ../../nvme0n1p1
lrwxrwxrwx. 1 root root  9 Mar  9 14:09 2024-11-16-01-52-31-00 -> ../../sr0
lrwxrwxrwx. 1 root root 15 Mar  9 14:17 30e62b31-9522-4917-b284-e30bf5a2d726 ->
../../nvme0n1p3
lrwxrwxrwx. 1 root root 15 Mar  9 14:18 c2c331ad-1e9e-42aa-bc78-2237aefaeaee ->
../../nvme0n2p1
lrwxrwxrwx. 1 root root 15 Mar  9 14:17 c3c46cd8-f894-485a-b418-6fffb035d898 ->
../../nvme0n1p2
```

Use whatever you find for the UUID to add the new partition to `/etc/fstab` on `/srv/nfs` with the same other options as the root filesystem. Then, `mount /srv/nfs` and test that it works. If everything worked you should be able to run `df -h` and see a new entry there with about 40 GB of storage space.

```
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        4.0M     0  4.0M   0% /dev
tmpfs           368M     0  368M   0% /dev/shm
tmpfs           147M  3.9M  144M   3% /run
efivarfs        256K   55K  197K  22% /sys/firmware/efi/efivars
/dev/nvme0n1p2   14G  1.9G   12G  15% /
/dev/nvme0n1p1  256M  7.1M  249M   3% /boot/efi
tmpfs            74M     0   74M   0% /run/user/1001
/dev/nvme0n2p1   40G   24K   38G   1% /srv/nfs
```

37. Now let's play with NFS and mount points a bit. First, let's change the permissions on `/srv/nfs` so that everybody can write to it, but nobody can delete anybody else's loot …

    ```
    $ sudo chmod 1777 /srv/nfs
    ```

    … this is done by setting it with full permissions allowed for everybody, plus the sticky bit (the `1` in the four-digit permissions code).

    ```
    $ ls -ld /srv/nfs /tmp

    drwxrwxrwt.  3 root root 4096 Mar  9 14:18 /srv/nfs
    drwxrwxrwt. 15 root root 4096 Mar  9 14:28 /tmp
    ```

    Looks just like `/tmp` … perfect.

    Now unmount `/srv/nfs` …

    ```
    $ sudo umount /srv/nfs
    ```

    … and check its permissions again with the `ls` command from above. Did the permissions change? Why? Where is the directory `/srv/nfs` physically located when the second storage device is mounted?

Set the permissions on the mount point directory to match `/tmp` and then re-mount `/srv/nfs`.

Finally, edit `/etc/exports` and add a line for `/srv/nfs` with the same options as `/home`. After you're done editing `/etc/exports`, run `exportfs -a` again to have the NFS server parse the file again.

38. **Go to your FreeBSD VM**, and look at the contents of your home directory.

```
$ ls -la $HOME
```

Then `cd` to the root of the filesystem.

```
$ cd /
```

We can't mount anything at a point above us in the file namespace. That is, if you're at `/home/peter`, some OSs will flatly refuse to mount something at `/home`, which is exactly what we're about to do, in the process sharing all home directories across our two systems. `mount /home` from Rocky VM's root filesystem …

```
$ sudo mount rocky:/home /home
```

Let's create the general file share moint point `/srv/nfs` …

```
$ sudo mkdir -p /srv/nfs
```

… and then `mount` it from your Rocky VM and that new filesystem you just made.

```
$ sudo mount rocky:/srv/nfs /srv/nfs
```

If you don't get a command line right back from each of the above two `mount` commands, odds are you screwed up one of the last few steps. If it hangs for a while, hang out and see if an error message tips you off as to what went wrong.

You can now see it in the filesystem mount table. Type `mount` … my output looked like this.

```
$ mount
/dev/da0p2 on / (ufs, local, soft-updates, journaled soft-updates)
devfs on /dev (devfs)
/dev/da0p1 on /boot/efi (msdosfs, local)
rocky:/home on /home (nfs)
rocky:/srv/nfs on /srv/nfs (nfs)
```

And type `df -h` to look at filesystem utilization …

```
$ df -h
Filesystem        Size    Used   Avail Capacity  Mounted on
/dev/da0p2         14G    5.2G    7.3G    42%    /
devfs             1.0K      0B    1.0K     0%    /dev
/dev/da0p1        260M    1.3M    259M     1%    /boot/efi
rocky:/home        14G    1.9G     11G    14%    /home
rocky:/srv/nfs     39G     24K     37G     0%    /srv/nfs
```

Now check out the contents of your home directory again.

```
$ ls -la $HOME
```

Probably looks at least slightly different from a few minutes ago, huh?

Back on your Rocky Linux system, run the following command:

```
$ touch $HOME/testfile
```

On your FreeBSD system, run the following command:

```
$ ls -l $HOME/testfile
```

The file should look the same. Also on FreeBSD …

```
$ ps auxww > $HOME/testfile
```

… and on your Rocky Linux system, note the file is no longer empty.

‼ IMPORTANT NOTE: the output of `ls -l` **should** be listing your correct username, on both sides … because the defaults for every operating system so far are (or what I asked you to do was) to use user ID 1000 for the first non-root user.

39. Where did the files under your home directory on FreeBSD go? The short answer: nowhere. They're just sort of hidden, no longer accessible, because they're hidden "under" a mount point.

```
$ cd /
$ sudo umount /home
$ ls -la $HOME
```

Your FreeBSD home directory files are back! They never really went anywhere, and just weren't visible because `/home` became a mount point, and when that mount point is crossed, you've changed over to the root directory of the filesystem, or share, mounted "on" that mount point.

In this case, under `/home` on your FreeBSD VM … you were seeing the contents of `/home` on your Rocky VM.

40. Before we mount `/home` from NFS permanently on your FreeBSD system and can't get to the home directories on that VM, let's sync their contents to the new home directories over on the Rocky VM.

In moving files around thus far, we've made heavy use of `scp`, but `scp` just copies files you spell out from one place to another. It's not very good for synchronizing two directory trees with one another. We need another tool, and that tool is `rsync`.

Install `rsync` on Rocky Linux with `dnf` … you should know what to do here by now.

Let's also install it on FreeBSD … choose your poison, package …

```
$ sudo pkg install rsync
```

… or source.

```
$ cd /usr/ports/net/rsync && sudo make install
```

Now, run the following while logged in as each of your non-root users on your FreeBSD VM, your LDAP user and as your failsafe user, first test your file sync with a "dry run" …

```
$ rsync -avn ~/ rocky:
```

The trailing slash after the tilde is important! Per the documentation in `man rsync` (which you should peruse yourself!) …

> A trailing slash on the source changes this behavior to avoid creating an additional directory level at the destination. You can think of a trailing / on a source as meaning "copy the contents of this directory" as opposed to "copy the directory by name", but in both cases the attributes of the containing directory are transferred to the containing directory on the destination.

If you want to keep your `.bash_history` file of past commands run in either account on your Rocky VM, you can add in `--exclude=.bash_history` before the directories in the appropriate account(s). Then, with each account, once you're happy with the output of the dry run, remove the `n` switch from the command above to perform the synchronization for real.

41. Let's mount `/home` and `/srv/nfs` from Rocky Linux permanently on your FreeBSD VM, but not your OpenBSD VM.

    There's a very simple reason for this: we don't want to create a chicken/egg problem here and now, where your Rocky VM needs your OpenBSD VM for DNS, and your OpenBSD VM needs your Rocky VM for home directories. Your OpenBSD VM is your DNS server and your LDAP server, helps everything find everything else by name, and we want it to just start, without any dependencies on other systems.

    ‼ IMPORTANT NOTE: everything you add to `/etc/fstab` MUST SUCCESSFULLY mount at boot time, or the operating system will kick you out into single-user mode. After we make this change, you will have to have your Rocky VM booted up before your FreeBSD VM or your FreeBSD VM will fail to boot!

    On your FreeBSD box …

    ```
    $ sudo vi /etc/fstab
    ```

    Add these lines at the end of the file, using tabs to make the columns line up with prior entries:

    ```
    rocky:/home     /home           nfs     rw,nfsv4        0       0
    rocky:/srv/nfs  /srv/nfs        nfs     rw,nfsv4        0       0
    ```

… and reboot your FreeBSD VM. When it comes back, log in and run `df` again. If it doesn't come back, you almost certainly made a mistake in `/etc/fstab` and need to fix it on the VM console to reboot into multi-user mode.

42. With NFS, we don't want root on one system to be root on another, for security purposes. In some cases, that would be **VERY BAD** … as discussed in the introduction to this lab, we would have to completely trust the system administrators of all NFS client systems, or they could leverage root powers on their computer … against files on our filesystem. Just bad, all bad.

    On your FreeBSD box, run the following command …

    ```
    $ sudo touch /srv/nfs/test
    ```

    … and note that, with `sudo`, we ran that command as root. NFS does something with file access performed as root called "squashing." That is, if a user is root on the remote system, we make them just like everybody else. We make them a nobody, literally.

    ```
    $ ls -l /srv/nfs/test
    -rw-r--r--+ 1 nobody nobody 0 Mar  9 15:17 /srv/nfs/test
    ```

    As you should also see on your VMs, `/srv/nfs/test` is owned by "nobody" and associated with the group "nobody." This pseudo-user and group were created **specifically for** the purpose of "squashing" root access over NFS shares, a long time ago.

    You might also notice something different … the plus sign at the end of the permissions line. This is because we're using NFSv4 for the best security available. NFSv4 allow extended file system access controls (ACLs) to be passed from NFS server to client. The utility for checking out ACLs is `getfacl`.

    ```
    $ getfacl /srv/nfs/test
    # file: /srv/nfs/test
    # owner: nobody
    # group: nobody
               owner@:rw-p--aA--cC-s:-------:allow
               group@:r-----a---c--s:-------:allow
            everyone@:r-----a---c--s:-------:allow
    ```

    These are new to me as well; NFSv4 was just added for this lab. `man 9 acl` seems to have the best information I've found thus far on these ACLs.

    ‼ IMPORTANT NOTE: After completing this lab, your lab setup now has a change to its dependency-induced mandatory boot order.

    When bringing up your lab network, always first boot your OpenBSD VM to completion, because it hosts name service and LDAP, both required by every other VM. Then boot your Rocky Linux VM, because it will have the home directories for all your other systems. You may then bring up all the other VMs in any order you like.

When shutting down your VMs, shut down everything but Rocky Linux and OpenBSD first, in any order. Then shut down Rocky Linux, and finally OpenBSD.

‼ IMPORTANT REMINDER: `shutdown` on Linux just **has** to be different, of course, and uses `-P` with a capital `P` for a power-off shutdown, but it's also the default, so you don't need to have it in there. Because many Linux distributions have drawn inspiration from AT&T Unix, you also frequently have the command `poweroff` when on a Linux.

part five: print server with CUPS

In this part of the lab, we're going to set up the Common Unix Printing System ("CUPS") and aim it at a printer on campus, an HP Color LaserJet Pro in my office, so that you can print something and show that you got this part of the lab working for your grade.

This section of the lab is worth 1% of your class grade – separately from all other lab exercises – because it is the only one graded by a manual process. This task is due one day before the final exam.

For many years, printing in Unix-like operating systems was a tremendous headache. Every single operating system had their own implementation of mostly incompatible tool suites for implementing print services and print queues, and the predominant protocol used to connect to printers, `lpd`, short for line printer daemon, was riddled with shortcomings and really conceived to deal with older dot-matrix "line printers" rather than the rich color graphics we print today. It was a problem begging for a solution.

In 1997, Michael Sweet started working on the solution, CUPS, which stood for "Common Unix Printing System." CUPS adopted IPP as the standard protocol, not just for communication with printers, which really helped to drive adoption, but also to communicate with the server itself. IPP was originally called HTPP, HyperText Printing Protocol, and was built on top of HTTP, so CUPS only needs to provide one port for the submission of print jobs … and to offer an administration interface viewable with a web browser.

In order to print to a printer on campus, and make sure we get around firewall rules no matter where we're currently working on this lab, we're going to bounce our printing traffic off EDORAS. OpenSSH, it turns out, is not just a secure telnet replacement, but a rather handy network "Swiss army knife" of tools. Normally, it connects through TCP port 22 and just stands up an encrypted terminal emulation connection. However, that terminal connection can also be used to plumb arbitrary port forwarders both to the other of the encrypted connection, and also backwards from the remote end. For this reason, SSH is both an invaluable tool to network security administrators … and the source of many nightmares for those same administrators when in the hands of users, who can often use it to punch holes right through a network traffic policy.

We'll be using SSH port forwarding more in lab 7.

43. First, let's check for CUPS …

```
$ rpm -qa | grep -i cups
```

… this command produces no output, so unsurprisingly, print services are not a part of the minimal installation of Rocky Linux. Let's install it. As we did before, and should be generally considered a best practice when dealing with new, or larger, software packages, let's search the package repositories for CUPS and related packages.

```
$ dnf search cups
```

This returns a lot of results, as you'll see.

```
================================== Name & Summary Matched: cups
==================================
cups.x86_64 : CUPS printing system
apcupsd-cgi.x86_64 : Web interface for apcupsd
apcupsd-gui.x86_64 : GUI interface for apcupsd
bluez-cups.x86_64 : CUPS printer backend for Bluetooth printers
cups-client.x86_64 : CUPS printing system - client programs
cups-devel.x86_64 : CUPS printing system - development environment
cups-devel.i686 : CUPS printing system - development environment
cups-filesystem.noarch : CUPS printing system - directory layout
cups-filters.x86_64 : OpenPrinting CUPS filters and backends
cups-filters-libs.x86_64 : OpenPrinting CUPS filters and backends - cupsfilters and fontembed
                         : libraries
cups-filters-libs.i686 : OpenPrinting CUPS filters and backends - cupsfilters and fontembed
                       : libraries
cups-ipptool.x86_64 : CUPS printing system - tool for performing IPP requests
cups-libs.x86_64 : CUPS printing system - libraries
cups-libs.i686 : CUPS printing system - libraries
cups-lpd.x86_64 : CUPS printing system - lpd emulation
cups-pdf.x86_64 : Extension for creating pdf-Files with CUPS
cups-printerapp.x86_64 : CUPS printing system - tools for printer application
dymo-cups-drivers.x86_64 : DYMO LabelWriter Drivers for CUPS
gutenprint-cups.x86_64 : CUPS drivers for Canon, Epson, HP and compatible printers
python-cups-doc.x86_64 : Documentation for python-cups
python3-cups.x86_64 : Python3 bindings for CUPS API, known as pycups.
======================================== Name Matched: cups
========================================
apcupsd.x86_64 : APC UPS Power Control Daemon
cups-pk-helper.x86_64 : A helper that makes system-config-printer use PolicyKit
====================================== Summary Matched: cups
======================================
c2esp.x86_64 : CUPS driver for Kodak AiO printers
samba-krb5-printing.x86_64 : Samba CUPS backend for printing with Kerberos
```

Looks to me like we want all the packages starting with `cups` and the "GutenPrint" drivers, a really good package of open-source printer drivers, to make sure we can get this running. Go ahead and install it with the following command. Note how we can use wildcards with `dnf`, but need to escape out of them in order to **not** have the wildcard expanded ("globbed") by the shell.

```
$ sudo dnf install cups\* gutenprint-cups
```

With all of the recommended dependencies, this ended up being almost 100 packages of fonts, graphics, filters, and drivers. One can easily understand why it was left out of the minimal installation. Finally, let's enable and start CUPS if it's not already.

```
$ sudo systemctl enable cups
$ sudo systemctl start cups
```

You should now see `cupsd`, the CUPS daemon, show up in `ps` output. Output of `netstat` should also show a listener on loopback for the IPP protocol.

```
$ netstat -a | grep ipp
tcp        0      0 localhost:ipp           0.0.0.0:*               LISTEN
tcp6       0      0 localhost:ipp           [::]:*                  LISTEN
```

44. As you can see just above, `cupsd` only attaches to its host's loopback adapter by default, so we can't hit it directly over the network without some re-configuration.

    CUPS very recently had a grievous CVE associated with it, with severity rating 9.9 out of 10. Because this package installed in a configuration only bound to the loopback adapter, exploitation of this flaw would have required an attacker to already have a modicum access to the system, like a shell account. The package you just downloaded was almost certainly already patched.

    In order to talk directly to it, we could either allow it to listen on all adapters, in which case we should probably also add a certificate to protect that interface, or we can use a port forwarder. OpenSSH has the a VPN-like ability to tunnel arbitrary ports between the two systems involved in an SSH connection, which we can use to our Rocky VM's loopback ports from our hosts. Since we're not going to use this service very much and we might like to keep a security liability off the network, OpenSSH it is.

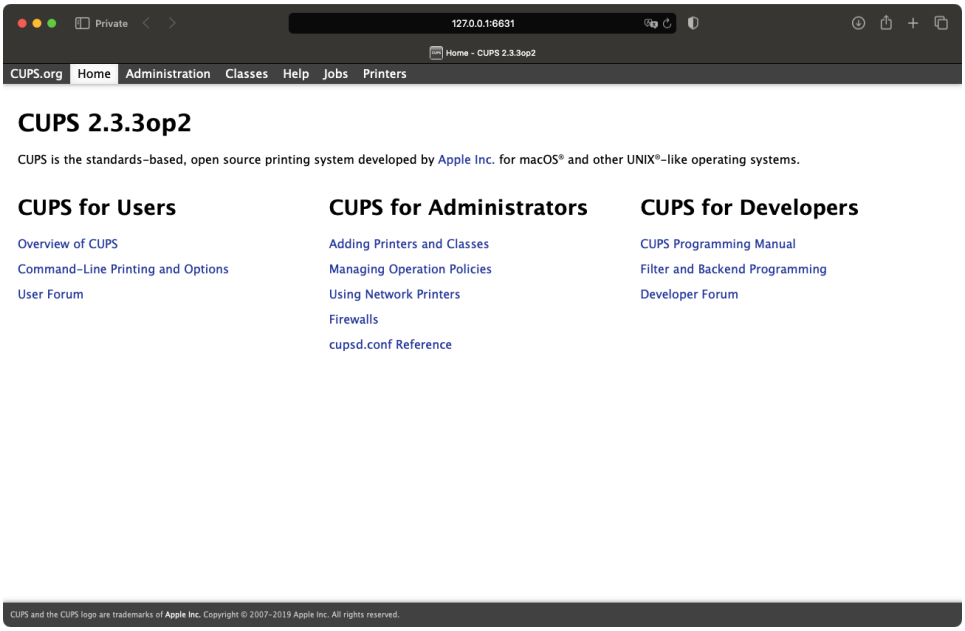
    We need to set up two forwarders, really: one from our lab computer's host operating system to Rocky, in order to access the CUPS service port, and another, from our Rocky VM to SDSUnet to get it around SDSU's firewall to the LaserJet in my office.

    First things first, let's set up the port forward to let us get into the CUPS service port on our Rocky VM's loopback interface. To implement the port forward once, manually, we'd run this command on our host operating system …

    ```
    $ ssh -L 6631:127.0.0.1:631 rocky
    ```

    … which tells OpenSSH we want loopback port 6631 on the local system, the SSH client, to forward to 127.0.0.1 port 631 on the SSH server, the port for IPP. We can override loopback here by supplying an IP and colon before the 6631, if we wanted to use a regular IP or real interface. After you run this command, on your SSH client (your host OS), you should be able to verify port 6631 is open, with `netstat`.

    We do this from our host operating system because … what do we want to use to connect to the CUPS service port? A web browser, of course. Aim your web browser at http://127.0.0.1:6631 and you should see something like the screenshot below. Please go ahead and explore the web management interface of CUPS, and note: authentication may not be configured (probably a huge reason it's bound to loopback) and neither are any printers. **If it asks you for a username and password, try your root credentials.**



45. Let's implement this port forward on a more permanent basis, in more seamless fashion. In labs 0 and 1, we stood up OpenSSH with key-based authentication and agent forwarding, and put this snippet in your `~/.ssh/config` file on your host operating system …

```
Host openbsd freebsd rocky ubuntu solaris *.cs470.internal 10.42.77.*
        ForwardAgent yes
```

… OpenSSH, however, will only match the first configuration stanza for any given hostname, and then stop processing rules. Remove `rocky` from the `Host` line in the configuration above, and create a configuration stanza for just your Rocky VM, right **above** the one with the rest …

```
Host rocky rocky.cs470.internal
        ForwardAgent yes
        LocalForward 6631 127.0.0.1:631
```

… we do this right above the other stanza because it contains a wildcard for all DNS hosts in the domain cs470.internal (`*.cs470.internal`); if we put our configuration for our Rocky VM after it, and called it by its fully-qualified DNS domain name, it would still match the other stanza and not include our port forwarder.

With this config snippet set up in SSH, you should now be able to just SSH into your Rocky VM from your host, and then use your browser to connect to CUPS.

46. Ordinarily, we'd connect our print server directly to a printer within our office, by DNS name or IP address … we'd be able to address it directly. However, for the purposes of this exercise, I needed to be able to receive your print job as a way of checking you'd succeeded. That meant there had to be a printer on campus, and that I needed a way to reach it from people working off campus.

A VPN just would have been too painful, so let's set up another SSH port forwarder, this time from your Rocky VM to EDORAS for the HP Color LaserJet Pro in my office in GMCS. It's on the IP address

130.191.39.112, and we want to try sending jobs to it over port 631 (IPP).

Let's skip the manual testing this time, and go ahead and set up `~/.ssh/config` in your Rocky VM. Note: your home directory is shared with your other VMs, and so is your SSH client configuration. I can't think of a drawback here, however, since we're not using a port at the other side and have no intention of printing from any of the other VMs.

Add the following block of configuration to `~/.ssh/config` on your Rocky VM …

```
Host edoras edoras.sdsu.edu
        LocalForward 9100 130.191.39.112:9100
```

… and log into EDORAS. This means we're forwarding port 9100 on the loopback adapter of our Rocky VMs to port 9100 of the IP 130.191.39.112 from EDORAS, while we're logged into it. I thought long and hard about how to test this step, but the best way is just going to be to print something.

You need to be logged into EDORAS for this port forward to work, so if you want to print the deliverable to the computer in my office, you must log into your Rocky VM in one SSH session to spool the job, and you must log into your Rocky VM in another SSH session, and SSH into EDORAS from your Rocky VM from there to enable the port forward.

47. Let's set up CUPS for the printer. In the CUPS web UI, click "administration" in the top banner, then hit the "add printer" button under "printers." Try using your Rocky VM's root credentials here to log in.

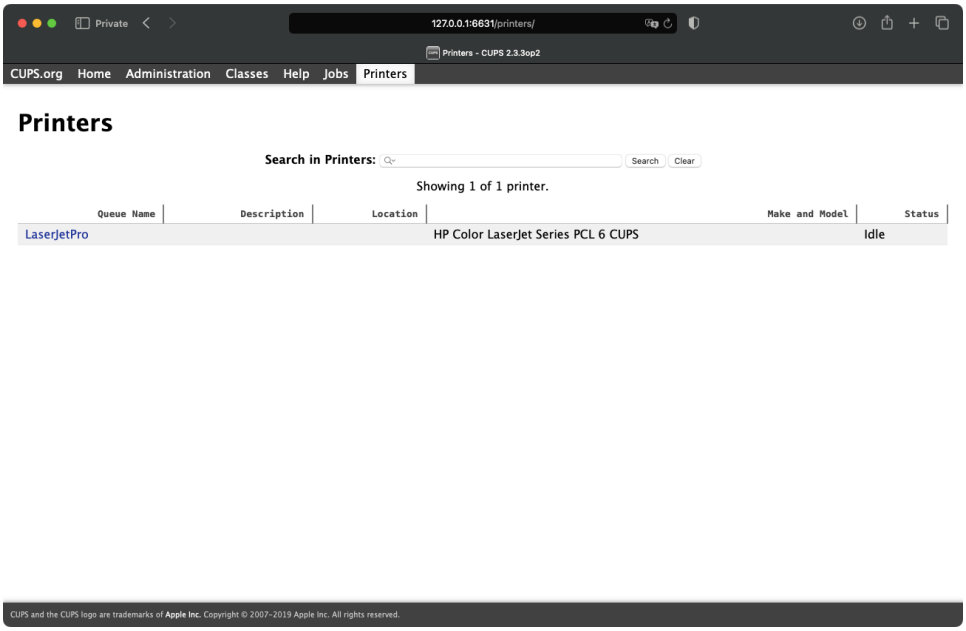
    On the first screen, choose "AppSocket / HP JetDirect" as the protocol to be used with the printer.

    On the second screen, enter `socket://localhost:9100` as the connection URI for the printer.

    On the third screen, enter `LaserJetPro` as the printer name, and leave the description and location blank. Do not share the printer.

    On the next, choose "HP" as the make of the printer, then click continue.

    On the next, choose "HP Color LaserJet Series PCL 6 CUPS (en)" as the model of the printer and then click the button to "add printer."



After completion, you should be able to go to the "printers" tab and see the printer there, idle … unless somebody else is printing on it, since we're all sharing a printer.

48. Let's print to the printer now. Use a word processor to create a single-page document on your computer, with appropriately large text spelling out CS470, your name, your red ID number, and your SDSUid e-mail address. Export that document to a PDF, and copy the PDF over to your Rocky VM.

    If you named your printer as suggested, you'd use the following command to send your PDF to the appropriate CUPS print queue. Naturally, substitute the name of your PDF for `deliverable.pdf`, the name of mine. **You must be SSH'd into EDORAS from your Rocky VM with the port forwarder properly configured for this to work.**

    ```
    $ lpr -P LaserJetPro deliverable.pdf
    ```

    If you keep the printers page open in CUPS and continually reload it, you should see the printer status go through all the phases as it processes the print job, sends it over the wire to the printer, and waits to see that the printer completed successfully. You can also check the logs for the local CUPS service with the following command:

    ```
    $ sudo journalctl -u cups -e
    ```

    I will be checking the printer in GMCS once a week, after lectures. If you want to know if I got your print job on the printer, send me a Discord PM or create a ticket In the class Discord server.

part six: latest kernel, via package

Our new Rocky Linux VM here is already our file server, and systems are already dependent on it. If we were

to log into our FreeBSD, or if we were logged in already, and our Rocky Linux VM was down or rebooting, files under `/home` would be inaccessible.

We can only really tolerate a reboot here, not a long and drawn-out process troubleshooting. Thus, we want the system to come right back up, so let's use a known-good kernel from a package. We'll build the Linux kernel from source in the next lab.

49. Whereas Ubuntu has a really cool menu-based tool, `mainline`, for installing newer kernels, the best way to get the newest kernel on Red Hat Enterprise Linux and its clones is via ELRepo. ELRepo is a reference to "enterprise Linux," but used to claim in jest to be Spanish, like *the repo*.

    First, let's import the repository's signing key.

    ```
    $ sudo rpm --import https://www.elrepo.org/RPM-GPG-KEY-elrepo.org
    ```

    Next, install the repository's configuration files into `dnf`.

    ```
    $ sudo dnf -y install https://www.elrepo.org/elrepo-release-9.el9.elrepo.noarch.rpm
    ```

    Just like we did previously when adding the EPEL repository, let's index the new repository's contents.

    ```
    $ sudo dnf check-update
    ```

50. Check the repository for the latest kernel.

    ```
    $ dnf list available --disablerepo='*' --enablerepo=elrepo-kernel | grep kernel-
    ```

    The command above is specifically leaving out all configured repositories, except for `elrepo-kernel`. You should see output similar to the following.

    ```
    kernel-lt.x86_64                        6.1.130-1.el9.elrepo       elrepo-kernel
    kernel-lt-core.x86_64                   6.1.130-1.el9.elrepo       elrepo-kernel
    kernel-lt-devel.x86_64                  6.1.130-1.el9.elrepo       elrepo-kernel
    kernel-lt-devel-matched.x86_64          6.1.130-1.el9.elrepo       elrepo-kernel
    kernel-lt-doc.noarch                    6.1.130-1.el9.elrepo       elrepo-kernel
    kernel-lt-headers.x86_64                6.1.130-1.el9.elrepo       elrepo-kernel
    kernel-lt-modules.x86_64                6.1.130-1.el9.elrepo       elrepo-kernel
    kernel-lt-modules-extra.x86_64          6.1.130-1.el9.elrepo       elrepo-kernel
    kernel-lt-tools.x86_64                  6.1.130-1.el9.elrepo       elrepo-kernel
    kernel-lt-tools-libs.x86_64             6.1.130-1.el9.elrepo       elrepo-kernel
    kernel-lt-tools-libs-devel.x86_64       6.1.130-1.el9.elrepo       elrepo-kernel
    kernel-ml.x86_64                        6.13.6-1.el9.elrepo        elrepo-kernel
    kernel-ml-core.x86_64                   6.13.6-1.el9.elrepo        elrepo-kernel
    kernel-ml-devel.x86_64                  6.13.6-1.el9.elrepo        elrepo-kernel
    kernel-ml-devel-matched.x86_64          6.13.6-1.el9.elrepo        elrepo-kernel
    kernel-ml-doc.noarch                    6.13.6-1.el9.elrepo        elrepo-kernel
    kernel-ml-headers.x86_64                6.13.6-1.el9.elrepo        elrepo-kernel
    kernel-ml-modules.x86_64                6.13.6-1.el9.elrepo        elrepo-kernel
    kernel-ml-modules-extra.x86_64          6.13.6-1.el9.elrepo        elrepo-kernel
    kernel-ml-tools.x86_64                  6.13.6-1.el9.elrepo        elrepo-kernel
    kernel-ml-tools-libs.x86_64             6.13.6-1.el9.elrepo        elrepo-kernel
    kernel-ml-tools-libs-devel.x86_64       6.13.6-1.el9.elrepo        elrepo-kernel
    ```

If you're on an ARM Mac, the packages above should show aarch64 instead of x86_64.

The `lt` in `kernel-lt`, and `ml` in `kernel-ml`, stand for "long-term" and "mainline," respectively. Sysadmins preferring stability and long-term maintenance could go with `kernel-lt`, but in our case, we want the latest mainline Linux kernel release. We want `ml`.

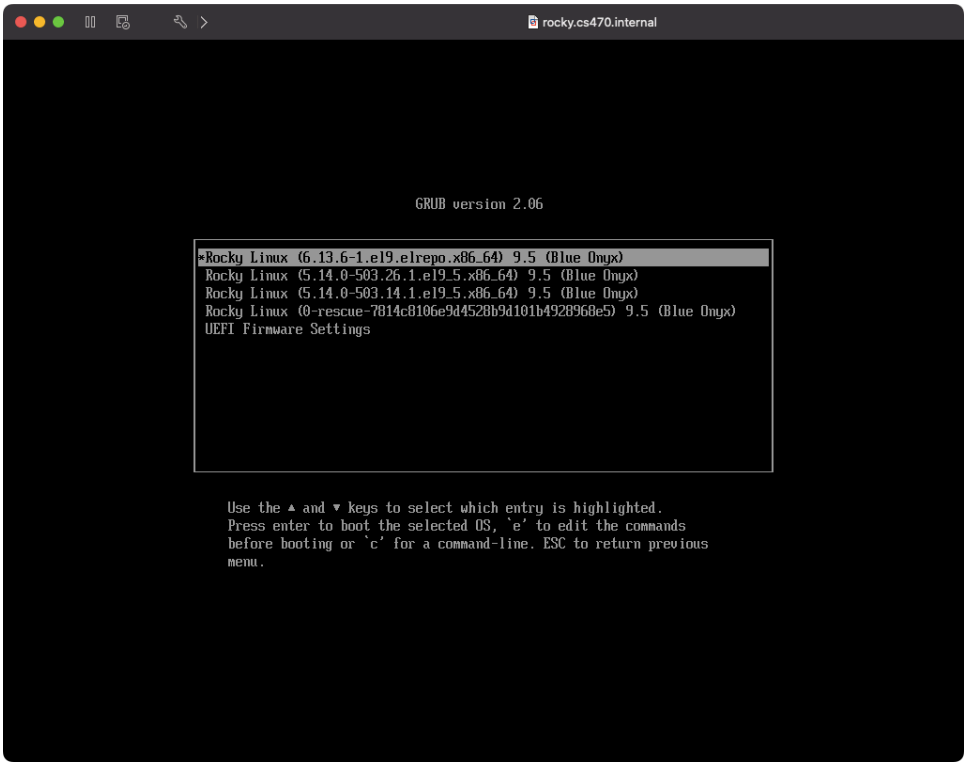
```
$ sudo dnf -y install --enablerepo=elrepo-kernel kernel-ml
```

51. After it's done, `sudo reboot`, and while your computer is rebooting, hold down the `escape` key to stop at the GRUB boot loader menu. We'll explore GRUB more in lab 4, but need to look at it a little now too. First and foremost, if your new kernel is installed properly, it'll show up in the GRUB menu. If the new kernel gives you problems, you may need to come back to the GRUB menu to boot a known-good kernel, and then remove, replace, or re-install the upgraded kernel.

    If you end up at a firmware menu asking you to choose between boot devices instead of kernel versions, just exit or continue out of that menu and hit `escape` again as you exit.

    If you end up at a GRUB command line, type `normal` and then hit the `escape` key right after the `return` key to end up back at the menu.

    If you screwed up and used legacy BIOS on an Intel-based VM, you won't get into the GRUB menu with `escape` at all; you'll want to hold down the left `shift` key on your keyboard as your VM reboots.



When you get to the GRUB menu, you should see something like the above, with your new kernel up

top as the new default.

52. Reboot normally into multi-user mode, without interacting with the GRUB menu, and use `uname` to check out your new kernel. It should reflect, like my output below, that you're running kernel version 6.13.x, or at least something far more recent than the 5.x kernel that came off the installation ISO.

```
$ uname -a
Linux rocky 6.13.6-1.el9.elrepo.x86_64 #1 SMP PREEMPT_DYNAMIC Fri Mar  7 15:29:37
EST 2025 x86_64 x86_64 x86_64 GNU/Linux
```

I'll be grading you on running a 6.x kernel from ELRepo, or something like that.

part seven: patching and tuning

53. As with all our other VMs, let's set up our Rocky Linux VM to do automatic checking for operating system and package updates, and to e-mail us the output.

```
$ sudo crontab -e
```

You'll be creating root's crontab with the above command … insert the following line …

```
0 0 * * * /usr/bin/dnf check-update
```

… and save it. Every night at midnight, your Rocky Linux VM will check the repositories for updates and patches.

54. Now let's manually check the package repositories for updates, and not wait for the automated task to run.

```
$ sudo dnf check-update
```

The prior command might have output a list of packages that are out of date; it might not. If it did, go ahead and patch your Rocky instance again. This is what we'll want to do when our `dnf cron` job displays that packages are behind, or out of date.

```
$ sudo dnf upgrade
```

55. Use `top` or `htop` to figure out how much memory you're using, and bring your system down …

```
$ sudo poweroff
```

… and play around with how little RAM your Rocky Linux VM will function with. The less resources you use here, the more are available here for our future VMs.

Make sure your NFS service and everything else we configured in this lab works. Once you're sure that it does, good job, you're done with lab 3.

```
</lab3>
```