

OpenHPC: Beyond the Install Guide for PEARC24

Sharon Colson Jim Moroney Mike Renfro

Tennessee Tech University

2024-07-22

Acknowledgments and shameless plugs

OpenHPC especially Tim Middelkoop (Internet2) and Chris Simmons (Massachusetts Green High Performance Computing Center). They have a BOF at 1:30 Wednesday. You should go to it.

Acknowledgments and shameless plugs

OpenHPC especially Tim Middelkoop (Internet2) and Chris Simmons (Massachusetts Green High Performance Computing Center). They have a BOF at 1:30 Wednesday. You should go to it.

Jetstream2 especially Jeremy Fischer, Mike Lowe, and Julian Pistorius. Jetstream2 has a tutorial at the same time as this one. Please stay here.

Acknowledgments and shameless plugs

OpenHPC especially Tim Middelkoop (Internet2) and Chris Simmons (Massachusetts Green High Performance Computing Center). They have a BOF at 1:30 Wednesday. You should go to it.

Jetstream2 especially Jeremy Fischer, Mike Lowe, and Julian Pistorius. Jetstream2 has a tutorial at the same time as this one. Please stay here.

NSF CC* for the equipment that led to some of the lessons we're sharing today (award #2127188).

Acknowledgments and shameless plugs

- OpenHPC** especially Tim Middelkoop (Internet2) and Chris Simmons (Massachusetts Green High Performance Computing Center). They have a BOF at 1:30 Wednesday. You should go to it.
- Jetstream2** especially Jeremy Fischer, Mike Lowe, and Julian Pistorius. Jetstream2 has a tutorial at the same time as this one. Please stay here.
- NSF CC*** for the equipment that led to some of the lessons we're sharing today (award #2127188).
- ACCESS** current maintainers of the project formerly known as the XSEDE Compatible Basic Cluster.

Where we're starting from

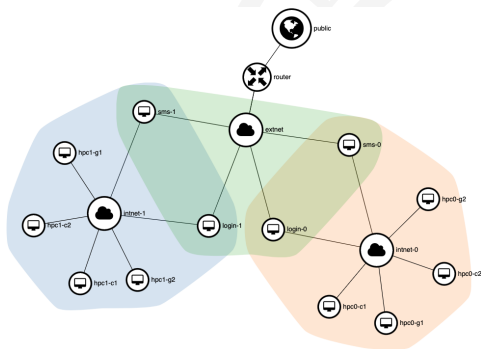


Figure 1: Two example HPC networks for the tutorial

You:

- ▶ have installed OpenHPC before
- ▶ have been issued a (basically) out-of-the-box OpenHPC cluster for this tutorial

Cluster details:

- ▶ Rocky Linux 9 (x86_64)
- ▶ OpenHPC 3.1, Warewulf 3, Slurm 23.11.6
- ▶ 2 non-GPU nodes
- ▶ 2 GPU nodes (currently without GPU drivers, so: expensive non-GPU nodes)
- ▶ 1 management node (SMS)
- ▶ 1 unprovisioned login node

Where we're starting from

We used the OpenHPC automatic installation script from Appendix A with a few variations:

1. Installed s-nail to have a valid MailProg for `slurm.conf`.
2. Created `user1` and `user2` accounts with password-less `sudo` privileges.
3. Changed `CHROOT` from `/opt/ohpc/admin/images/rocky9.3` to `/opt/ohpc/admin/images/rocky9.4`.
4. Enabled `slurmd` and `munge` in `CHROOT`.
5. Added `nano` and `yum` to `CHROOT`.
6. Removed a redundant `ReturnToService` line from `/etc/slurm/slurm.conf`.
7. Stored all compute/GPU nodes' SSH host keys in `/etc/ssh/ssh_known_hosts`.
8. Globally set an environment variable `CHROOT` to `/opt/ohpc/admin/images/rocky9.4`.

Where we're going

1. A login node that's practically identical to a compute node (except for where it needs to be different)
2. A slightly more secured SMS and login node
3. GPU drivers on the GPU nodes
4. Using node-local storage for the OS and/or scratch
5. De-coupling the SMS and the compute nodes (e.g., independent kernel versions)
6. Easier management of node differences (GPU or not, diskless/single-disk/multi-disk, Infiniband or not, etc.)
7. Slurm configuration to match some common policy goals (fair share, resource limits, etc.)

Assumptions

1. We have a VM named `login`, with no operating system installed.
2. The `eth0` network interface for `login` is attached to the internal network, and `eth1` is attached to the external network.
3. The `eth0` MAC address for `login` is known—check the **Login server** section of your handout for that. It's of the format `aa:bb:cc:dd:ee:ff`.
4. We're logged into the SMS as `user1` or `user2` that has `sudo` privileges.

Create a new login node

Working from section 3.9.3 of the install guide:

```
[user1@sms ~]$ sudo wwsh -y node new login --netdev eth0 \  
--ipaddr=172.16.0.2 --hwaddr=__:__:__:__:__:__  
[user1@sms ~]$ sudo wwsh -y provision set login \  
--vnfs=rocky9.4 --bootstrap=$(uname -r) \  
--files=dynamic_hosts,passwd,group,shadow,munge.key,network
```

Make sure to replace the `__` with the characters from your login node's MAC address!

What'd we just do?

Ever since `login` was powered on, it's been stuck in a loop trying to PXE boot. What's the usual PXE boot process for a client in an OpenHPC environment?

1. The client network card tries to get an IP address from a DHCP server (the SMS) by broadcasting its MAC address.

What'd we just do?

Ever since `login` was powered on, it's been stuck in a loop trying to PXE boot. What's the usual PXE boot process for a client in an OpenHPC environment?

1. The client network card tries to get an IP address from a DHCP server (the SMS) by broadcasting its MAC address.
2. The SMS responds with the client's IP and network info, a `next-server` IP (the SMS again), and a `filename` option (a bootloader from the iPXE project).

What'd we just do?

Ever since `login` was powered on, it's been stuck in a loop trying to PXE boot. What's the usual PXE boot process for a client in an OpenHPC environment?

1. The client network card tries to get an IP address from a DHCP server (the SMS) by broadcasting its MAC address.
2. The SMS responds with the client's IP and network info, a `next-server` IP (the SMS again), and a `filename` option (a bootloader from the iPXE project).
3. The network card gets the bootloader over TFTP and executes it.

What'd we just do?

Ever since `login` was powered on, it's been stuck in a loop trying to PXE boot. What's the usual PXE boot process for a client in an OpenHPC environment?

1. The client network card tries to get an IP address from a DHCP server (the SMS) by broadcasting its MAC address.
2. The SMS responds with the client's IP and network info, a `next-server` IP (the SMS again), and a `filename` option (a bootloader from the iPXE project).
3. The network card gets the bootloader over TFTP and executes it.
4. iPXE makes a second DHCP request and this time, it gets a URL (by default, `http://SMS_IP/WW/ipxe/cfg/${client_mac}`) for an iPXE config file.

What'd we just do?

Ever since `login` was powered on, it's been stuck in a loop trying to PXE boot. What's the usual PXE boot process for a client in an OpenHPC environment?

1. The client network card tries to get an IP address from a DHCP server (the SMS) by broadcasting its MAC address.
2. The SMS responds with the client's IP and network info, a `next-server` IP (the SMS again), and a `filename` option (a bootloader from the iPXE project).
3. The network card gets the bootloader over TFTP and executes it.
4. iPXE makes a second DHCP request and this time, it gets a URL (by default, `http://SMS_IP/WW/ipxe/cfg/${client_mac}`) for an iPXE config file.
5. The config file contains the URL of a Linux kernel and initial ramdisk, plus multiple kernel parameters available after initial bootup for getting the node's full operating system contents.

What'd we just do?

1. The node name, `--hwaddr`, and `--ipaddr` parameters go into the SMS DHCP server settings.

What'd we just do?

1. The node name, `--hwaddr`, and `--ipaddr` parameters go into the SMS DHCP server settings.
2. The `--bootstrap` parameter defines the kernel and ramdisk for the iPXE configuration.

What'd we just do?

1. The node name, `--hwaddr`, and `--ipaddr` parameters go into the SMS DHCP server settings.
2. The `--bootstrap` parameter defines the kernel and ramdisk for the iPXE configuration.
3. The node name, `--netdev`, `--ipaddr`, `--hwaddr` parameters all go into kernel parameters accessible from the provisioning software.

What'd we just do?

1. The node name, `--hwaddr`, and `--ipaddr` parameters go into the SMS DHCP server settings.
2. The `--bootstrap` parameter defines the kernel and ramdisk for the iPXE configuration.
3. The node name, `--netdev`, `--ipaddr`, `--hwaddr` parameters all go into kernel parameters accessible from the provisioning software.
4. During the initial bootup, the `--hwaddr` parameter is passed to a CGI script on the SMS to identify the correct VNFS for the provisioning software to download (set by the `--vnfs` parameter).

What'd we just do?

1. The node name, `--hwaddr`, and `--ipaddr` parameters go into the SMS DHCP server settings.
2. The `--bootstrap` parameter defines the kernel and ramdisk for the iPXE configuration.
3. The node name, `--netdev`, `--ipaddr`, `--hwaddr` parameters all go into kernel parameters accessible from the provisioning software.
4. During the initial bootup, the `--hwaddr` parameter is passed to a CGI script on the SMS to identify the correct VNFS for the provisioning software to download (set by the `--vnfs` parameter).
5. After downloading the VNFS, the provisioning software will also download files from the SMS set by the `--files` parameter.

Did it work? So far, so good.

```
[user1@sms ~]$ sudo ssh login
[root@login ~]# df -h
Filesystem
...
172.16.0.1:/home
172.16.0.1:/opt/ohpc/pub
```

Did it work? Not entirely.

```
[root@login ~]# sinfo
sinfo: error: resolve_ctls_from_dns_srv: res_nsearch error:
  Unknown host
sinfo: error: fetch_config: DNS SRV lookup failed
sinfo: error: _establish_config_source: failed to fetch config
sinfo: fatal: Could not establish a configuration source
```

systemctl status slurmd is more helpful, with
fatal: Unable to determine this slurmd's NodeName. So how do we fix this one?

Option 1: take the error message literally

So there's no entry for login in the SMS `slurm.conf`. To fix that:

1. Run `slurmd -C` on the login node to capture its correct CPU specifications. Copy that line to your laptop's clipboard.

Option 1: take the error message literally

So there's no entry for login in the SMS `slurm.conf`. To fix that:

1. Run `slurmd -C` on the login node to capture its correct CPU specifications. Copy that line to your laptop's clipboard.
2. On the SMS, run `nano /etc/slurm/slurm/slurm.conf` and make a new line of all the `slurmd -C` output from the previous step (pasted from your laptop clipboard).

Option 1: take the error message literally

So there's no entry for login in the SMS `slurm.conf`. To fix that:

1. Run `slurmd -C` on the login node to capture its correct CPU specifications. Copy that line to your laptop's clipboard.
2. On the SMS, run `nano /etc/slurm/slurm/slurm.conf` and make a new line of all the `slurmd -C` output from the previous step (pasted from your laptop clipboard).
3. Save and exit nano by pressing `Ctrl-X` and then `Enter`.

Option 1: take the error message literally

So there's no entry for login in the SMS `slurm.conf`. To fix that:

1. Run `slurmd -C` on the login node to capture its correct CPU specifications. Copy that line to your laptop's clipboard.
2. On the SMS, run `nano /etc/slurm/slurm/slurm.conf` and make a new line of all the `slurmd -C` output from the previous step (pasted from your laptop clipboard).
3. Save and exit nano by pressing `Ctrl-X` and then `Enter`.
4. Reload the new Slurm configuration everywhere (well, everywhere functional) with `sudo scontrol reconfigure` on the SMS.

Option 1: take the error message literally

So there's no entry for login in the SMS `slurm.conf`. To fix that:

1. Run `slurmd -C` on the login node to capture its correct CPU specifications. Copy that line to your laptop's clipboard.
2. On the SMS, run `nano /etc/slurm/slurm/slurm.conf` and make a new line of all the `slurmd -C` output from the previous step (pasted from your laptop clipboard).
3. Save and exit nano by pressing `Ctrl-X` and then `Enter`.
4. Reload the new Slurm configuration everywhere (well, everywhere functional) with `sudo scontrol reconfigure` on the SMS.
5. `ssh` back to the login node and restart `slurmd`, since it wasn't able to respond to the `scontrol reconfigure` from the previous step (`sudo ssh login systemctl restart slurmd` on the SMS).

Option 1: take the error message literally

Now an sinfo should work on the login node:

```
[root@login ~]# sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
normal*    up    1-00:00:00      1   idle  c[1-2]
```

Option 2: why are we running `slurmd` anyway?

The `slurmd` service is really only needed on systems that will be running computational jobs, and the login node is not in that category.

Running `slurmd` like the other nodes means the login node can get all its information from the SMS, but we can do the same thing with a very short customized `slurm.conf` with two lines from the SMS' `slurm.conf`:

```
ClusterName=cluster  
SlurmctldHost=sms
```

Interactive test

1. On the login node as root, temporarily stop the slurmd service with `systemctl stop slurmd`
2. On the login node as root, edit `/etc/slurm/slurm.conf` with `nano /etc/slurm/slurm.conf`
3. Add the two lines to the right, save and exit nano by pressing Ctrl-X and then Enter.

`/etc/slurm/slurm.conf` on login node

```
ClusterName=cluster  
SlurmctldHost=sms
```

Verify that `sinfo` still works without slurmd and with the custom `/etc/slurm/slurm.conf`.

```
[root@login ~]# sinfo  
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST  
normal*      up 1-00:00:00      1   idle c[1-2]
```

Make permanent changes from the SMS

Let's reproduce the changes we made interactively on the login node in the Warewulf settings on the SMS.

For the customized `slurm.conf` file, we can keep a copy of it on the SMS and add it to the Warewulf file store.

We've done that previously for files like the shared `munge.key` for all cluster nodes (see section 3.8.5 of the OpenHPC install guide).

We also need to make sure that file is part of the login node's provisioning settings.

Make permanent changes from the SMS

On the SMS:

```
[user1@sms ~]$ sudo scp login:/etc/slurm/slurm.conf \  
    /etc/slurm/slurm.conf.login  
slurm.conf          100%   40    57.7KB/s   00:00  
[user1@sms ~]$ sudo wwsh -y file import \  
    /etc/slurm/slurm.conf.login --name=slurm.conf.login \  
    --path=/etc/slurm/slurm.conf
```

Now the file is available, but we need to ensure the login node gets it. That's handled with wwsh provision.

A quick look at `wwsh` provision

What are the provisioning settings for compute node `c1`?

```
[user1@sms ~]$ wwsh provision print c1
#### c1 #####
c1: MASTER                = UNDEF
c1: BOOTSTRAP              = 6.1.96-1.el9.elrepo.x86_64
c1: VNFS                   = rocky9.4
c1: VALIDATE               = FALSE
c1: FILES                  = dynamic_hosts,group,munge.key,network,
    passwd,shadow
...
c1: KARGS                  = "net.ifnames=0 biosdevname=0 quiet"
c1: BOOTLOCAL              = FALSE
```

A quick look at `wwsh` provision

What are the provisioning settings for node login?

```
[user1@sms ~]$ wwsh provision print login
#### login #####
login: MASTER           = UNDEF
login: BOOTSTRAP         = 6.1.96-1.el9.elrepo.x86_64
login: VNFS              = rocky9.4
login: VALIDATE          = FALSE
login: FILES             = dynamic_hosts,group,munge.key,network,
    passwd,shadow
...
login: KARGS             = "net.ifnames=0 biosdevname=0 quiet"
login: BOOTLOCAL         = FALSE
```

A quick look at `wsh` provision

The provisioning settings for `c1` and `login` are identical, but there's a lot to read in there to be certain about it.

We could run the two outputs through `diff`, but every line contains the node name, so **no lines are literally identical**.

Let's simplify and filter the `wsh` provision output to make it easier to compare.

Filter the `wwsh` provision output

- I only care about the lines containing `=` signs, so

```
wwsh provision print c1 | grep =
```

is a start.

Filter the `wwsh` provision output

- ▶ I only care about the lines containing `=` signs, so

```
wwsh provision print c1 | grep =
```

is a start.

- ▶ Now all the lines are prefixed with `c1:`, and I want to keep everything after that, so

```
wwsh provision print c1 | grep = | cut -d: -f2-
```

will take care of that.

Filtered result

```
wwsh provision print c1 | grep = | cut -d: -f2-
```

```
MASTER          = UNDEF
BOOTSTRAP        = 6.1.96-1.el9.elrepo.x86_64
VNFS             = rocky9.4
VALIDATE         = FALSE
FILES            = dynamic_hosts , group , munge.key , network ,
                  passwd , shadow
...
KARGS            = "net.ifnames=0 biosdevname=0 quiet"
BOOTLOCAL        = FALSE
```

Much more useful.

Make a function for this

We may be typing that command pipeline a lot, so let's make a shell function to cut down on typing:

```
[user1@sms ~]$ function proprint() { \  
    wwsh provision print $@ | grep = | cut -d: -f2- ; }  
[user1@sms ~]$ proprint c1  
MASTER                = UNDEF  
BOOTSTRAP              = 6.1.96-1.el9.elrepo.x86_64  
...
```

diff the outputs

We could redirect a `proprint c1` and a `proprint login` to files and `diff` the resulting files, or we can use the shell's `<()` operator to treat command output as a file:

```
[user1@sms ~]$ diff -u <(proprint c1) <(proprint login)
[user1@sms ~]$
```

Either of those shows there are zero provisioning differences between a compute node and the login node.

Add the custom `slurm.conf` to the login node

Add a file to login's FILES property with:

```
[user1@sms ~]$ sudo wvsh -y provision set login \  
--fileadd=slurm.conf.login
```

(refer to section 3.9.3 of the install guide for previous examples of `--fileadd`).

Check for provisioning differences

```
[user1@sms ~]$ diff -u <(proprint c1) <(proprint login)
--- /dev/fd/63      2024-07-06  11:11:07.682959677  -0400
+++ /dev/fd/62      2024-07-06  11:11:07.683959681  -0400
@@ -2,7 +2,7 @@
     BOOTSTRAP          = 6.1.96-1.el9.elrepo.x86_64
     VNFS               = rocky9.4
     VALIDATE           = FALSE
-   FILES              = dynamic_hosts ,group ,munge.key ,network ,
passwd ,shadow
+   FILES              = dynamic_hosts ,group ,munge.key ,network ,
passwd ,shadow ,slurm.conf.login
     PRESHELL           = FALSE
     POSTSHELL          = FALSE
     POSTNETDOWN        = FALSE
```

Ensure `slurmd` doesn't run on the login node

To disable the `slurmd` service on just the login node, we can take advantage of conditions in the `systemd` service file. Back on the login node as root:

```
[user1@sms ~]$ sudo ssh login  
[root@login ~]# systemctl edit slurmd
```

Insert three lines between the lines of `### Anything between here...` and `### Lines below this comment...`:

```
[Unit]  
ConditionHost=|c*  
ConditionHost=|g*
```

This will only run the service on nodes whose hostnames start with `c` or `g`.

Ensure `slurmd` doesn't run on the login node

Once that file is saved, try to start the `slurmd` service with `systemctl start slurmd` and check its status with `systemctl status slurmd`:

```
o slurmd.service - Slurm node daemon
...
Condition: start condition failed at Sat 2024-07-06 18:12:17
EDT; 4min 22s ago
...
Jul 06 17:14:16 login systemd[1]: Stopped Slurm node daemon.
Jul 06 18:12:17 login systemd[1]: Slurm node daemon was skipped
because of an unmet condition check (ConditionHost=c*).
```

Make the changes permanent

The `systemctl edit` command resulted in a file `/etc/systemd/system/slurmd.service.d/override.conf`. Let's:

- ▶ make a place for it in the chroot on the SMS, and
- ▶ copy the file over from the login node.

```
[user1@sms ~]$ sudo mkdir -p \  
    ${CHROOT}/etc/systemd/system/slurmd.service.d/  
[user1@sms ~]$ sudo scp \  
    login:/etc/systemd/system/slurmd.service.d/override.conf \  
    ${CHROOT}/etc/systemd/system/slurmd.service.d/  
override.conf                               100%   23    36.7KB/s   00:00
```

(**Note:** we globally pre-set the `CHROOT` environment for any account that logs into the SMS so that you didn't have to.)

Make the changes permanent

Finally, we'll:

- ▶ rebuild the VNFS, and
- ▶ reboot both the login node and a compute node to test the changes.

```
[user1@sms ~]$ sudo wwnfs --chroot=${CHROOT}
Using 'rocky9.4' as the VNFS name
...
Total elapsed time
: 84.45 s
[user1@sms ~]$ sudo ssh login reboot
[user1@sms ~]$ sudo ssh c1 reboot
```

Verify the changes on the login node

Verify that the login node doesn't start slurmd, but can still run sinfo without any error messages.

```
[user1@sms ~]$ sudo ssh login systemctl status slurmd
o slurmd.service - Slurm node daemon
...
Jul 06 18:26:23 login systemd[1]: Slurm node daemon was
skipped because of an unmet condition check
(ConditionHost=c*).
[user1@sms ~]$ sudo ssh login sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
normal*      up 1-00:00:00      1   idle c[1-2]
```

Verify the changes on a compute node

Verify that the compute node still starts slurmd (it can also run sinfo).

```
[user1@sms ~]$ sudo ssh c1 systemctl status slurmd
o slurmd.service - Slurm node daemon
...
Jul 06 19:03:22 c1 slurmd[1082]: slurmd: CPUs=2 Boards=1
    Sockets=2 Cores=1 Threads=1 Memory=5912 TmpDisk=2956
    Uptime=28 CPUSpecList=(null) FeaturesAvail=(null)
    FeaturesActive=(null)
[user1@sms ~]$ sudo ssh c1 sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
normal*    up    1-00:00:00      1   idle  c2
normal*    up    1-00:00:00      1   down  c1
```

(Yes, c1 is marked down—we'll fix that shortly.)

Problem: the login node doesn't let users log in

What if we ssh to the login node as someone other than root?

```
[user1@sms ~]$ ssh login
Access denied: user user1 (uid=1001) has no active jobs on this
node.
Connection closed by 172.16.0.2 port 22
```

which makes this the exact opposite of a login node for normal users. Let's fix that.

Make the login node function as a login node

- ▶ The Access denied is caused by the `pam_slurm.so` entry at the end of `/etc/pam.d/sshd`, which is invaluable on a normal compute node, but not on a login node.
- ▶ On the SMS, you can also do a
`diff -u /etc/pam.d/sshd ${CHROOT}/etc/pam.d/sshd`
- ▶ You'll see that the `pam_slurm.so` line is the only difference between the two files.

Test a PAM change to the login node

- ▶ Temporarily comment out the last line of the login node's `/etc/pam.d/ssh` and see if you can ssh into the login node as a normal user (i.e., `ssh user1@login`).
- ▶ Your user should be able to log in now.
- ▶ In case the PAM configuration won't let root log in, **don't panic!** Instructors can reboot your login node from its console to put it back to its original state.

Make the change permanent

- ▶ We want to ensure that the login node gets the same `/etc/pam.d/sshd` that the SMS uses.
- ▶ We'll follow the same method we used to give the login node a custom `slurm.conf`:

```
[user1@sms ~]$ sudo wwsh -y file import /etc/pam.d/sshd \  
--name=sshd.login  
[user1@sms ~]$ wwsh file list  
...  
sshd.login :  rw-r--r-- 1    root root          727 /etc/pam.d/sshd
```

Make the change permanent

```
[user1@sms ~]$ sudo wwsh -y provision set login \  
--fileadd=sshd.login  
[user1@sms ~]$ diff -u <(proprint c1) <(proprint login)  
...  
VALIDATE                = FALSE  
- FILES                  = dynamic_hosts ,group ,munge.key ,network ,  
  passwd ,shadow  
+ FILES                  = dynamic_hosts ,group ,munge.key ,network ,  
  passwd ,shadow ,slurm.conf.login ,sshd.login  
...
```

(refer to section 3.9.3 of the install guide for previous examples of `--fileadd`).

Test the change

Reboot the login node and let's see if we can log in as a regular user.

```
[user1@sms ~]$ sudo ssh login reboot  
[user1@sms ~]$ ssh login  
[user1@login ~]$
```

A bit more security for the login node

Not too long after your SMS and/or login nodes are booted, you'll see messages in the SMS `/var/log/secure` like:

```
Jul 11 11:24:06 sms sshd[162636]: Invalid user evilmike from
68.66.205.120 port 1028
...
Jul 11 11:24:08 sms sshd[162636]: Failed password for invalid
user evilmike from 68.66.205.120 port 1028 ssh2
...
```

because people who want to break into computers for various reasons have Internet connections.

A bit more security for the login node

There's a lot of things that can be done to secure things, including:

1. Placing the SMS and login node external interfaces on protected network segment.
2. Allowing only administrative users to SSH into the SMS.
3. Replacing password-based authentication with key-based authentication.

Though #3 will eliminate brute-force password guessing attacks, it's usually not practical for a login node. So let's mitigate that differently with `fail2ban`.

How fail2ban works (by default)

1. Monitor `/var/log/secure` and other logs for indicators of brute-force attacks (invalid users, failed passwords, etc.)
2. If indicators from a specific IP address happen often enough over a period of time, use `firewalld` to block all access from that address for a period of time.
3. Once that period has expired, remove the IP address from the block list.

This reduces the effectiveness of brute-force password guessing by orders of magnitude (~10 guesses per hour versus ~100 or ~1000 guesses per hour).

Including `firewalld` could mean that some necessary services get blocked by default when `firewalld` starts. Let's see what those could be.

See what processes are listening on the login node

We'll use the `netstat` command to look for sockets that are `udp` or `tcp`, listening, and what process the socket is attached to. We omit anything only listening for `localhost` connections.

```
[user1@sms ~]$ sudo ssh login netstat -utlp | grep -v localhost
Active Internet connections (only servers)
Proto ... Local Address ... State PID/Program name
tcp      0.0.0.0:ssh      LISTEN 1034/sshd: /usr/sbi
tcp      0.0.0.0:sunrpc   LISTEN 1/init
tcp6     [::]:ssh       LISTEN 1034/sshd: /usr/sbi
tcp6     [::]:sunrpc    LISTEN 1/init
udp      0.0.0.0:sunrpc   0.0.0.0:* 1/init
udp      0.0.0.0:37036    0.0.0.0:* 1143/rsyslogd
udp6     [::]:sunrpc    [::]:* 1/init
```

See what processes are listening on the login node

`sshd` secure shell daemon, the main thing we want to protect against brute force attempts

`init` the first process started during booting the operating system. Effectively, this shows up when you participate in NFS file storage, as a server or a client (and login is a client).

`rsyslogd` message logging for all kinds of applications and services

Of these, `sshd` is the only one that we need to ensure `firewalld` doesn't block by default. In practice, the `ssh` port (22) is always in the default list of allowed ports.

Test installing fail2ban on the login node

Install the fail2ban packages into the CHROOT with

```
[user1@sms ~]$ sudo yum install --installroot=${CHROOT} \
    fail2ban
[user1@sms ~]$ sudo chroot ${CHROOT} systemctl enable \
    fail2ban firewalld
```

(the yum command will also install firewalld as a dependency of fail2ban).

Add the following to the chroot's sshd.local file with
`sudo nano ${CHROOT}/etc/fail2ban/jail.d/sshd.local:`

```
[sshd]
enabled = true
```

Should I run `fail2ban` everywhere?

`fail2ban` is probably best to keep to the login node, and not the compute nodes:

- ▶ Nobody can SSH into your compute nodes from outside.
- ▶ Thus, the only things a compute node could ban would be your SMS or your login node.
- ▶ A malicious or unwitting user could easily ban your login node from a compute node by SSH'ing to it repeatedly, which would effectively be a denial of service.

Test installing fail2ban on the login node

```
[user1@sms ~]$ sudo mkdir -p \  
  ${CHROOT}/etc/systemd/system/fail2ban.service.d/ \  
  ${CHROOT}/etc/systemd/system/firewalld.service.d/
```

Test installing fail2ban on the login node

```
[user1@sms ~]$ sudo nano \  
    ${CHROOT}/etc/systemd/system/fail2ban.service.d/override.conf
```

Add the lines

```
[Unit]  
ConditionHost=|login*
```

save and exit with Ctrl-X.

Test installing fail2ban on the login node

Finally, duplicate the override file for firewalld:

```
[user1@sms ~]$ sudo cp \  
${CHROOT}/etc/systemd/system/fail2ban.service.d/override.conf \  
${CHROOT}/etc/systemd/system/firewalld.service.d/override.conf
```


Test installing fail2ban on the login node

Before we go further, check if there's anything in /var/log/secure on the login node:

```
[user1@sms ~]$ sudo ssh login ls -l /var/log/secure  
-rw----- 1 root root 0 Jul  7 03:14 /var/log/secure
```

Nope. Let's fix that, too.

- ▶ Looking in /etc/rsyslog.conf, we see a bunch of things commented out, including the line #authpriv.* /var/log/secure.
- ▶ Rather than drop in an entirely new rsyslog.conf file that we'd have to maintain, rsyslog will automatically include any *.conf files in /etc/rsyslog.d.
- ▶ Let's make one of those for the chroot.

Make an rsyslog.d file, rebuild the VNFS, reboot the login node

```
[user1@sms ~]$ echo "authpriv.* /var/log/secure" | \
  sudo tee ${CHROOT}/etc/rsyslog.d/authpriv-local.conf
authpriv.* /var/log/secure
[user1@sms ~]$ cat \
  ${CHROOT}/etc/rsyslog.d/authpriv-local.conf
authpriv.* /var/log/secure
[user1@sms ~]$ sudo wwnfs --chroot=${CHROOT}
[user1@sms ~]$ sudo ssh login reboot
```

Post-reboot, how's fail2ban and firewalld on the login node?

```
[user1@sms ~]$ sudo ssh login systemctl status firewalld
[root@login ~]# systemctl status firewalld
x firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service;
          enabled; preset>
   Active: failed (Result: exit-code) since Thu 2024-07-11
          16:49:47 EDT; 46mi>
...
Jul 11 16:49:47 login systemd[1]: firewalld.service: Main
   process exited, code=exited, status=3/NOTIMPLEMENTED
Jul 11 16:49:47 login systemd[1]: firewalld.service: Failed
   with result 'exit-code'.
```

Not great.

Diagnosing 3/NOTIMPLEMENTED

- **So many** Google results amount to “reboot to get your new kernel”, but we’ve just booted a new kernel.

Diagnosing 3/NOTIMPLEMENTED

- ▶ **So many** Google results amount to “reboot to get your new kernel”, but we’ve just booted a new kernel.
- ▶ Red Hat has an article telling you to verify that you haven’t disabled module loading by checking `sysctl -a | grep modules_disabled`, but that’s not disabled either.

Diagnosing 3/NOTIMPLEMENTED

- ▶ **So many** Google results amount to “reboot to get your new kernel”, but we’ve just booted a new kernel.
- ▶ Red Hat has an article telling you to verify that you haven’t disabled module loading by checking `sysctl -a | grep modules_disabled`, but that’s not disabled either.
- ▶ The Red Hat article does tell you that packet filtering capabilities have to be enabled in the kernel, and that gets us closer.

Diagnosing 3/NOTIMPLEMENTED

- ▶ **So many** Google results amount to “reboot to get your new kernel”, but we’ve just booted a new kernel.
- ▶ Red Hat has an article telling you to verify that you haven’t disabled module loading by checking `sysctl -a | grep modules_disabled`, but that’s not disabled either.
- ▶ The Red Hat article does tell you that packet filtering capabilities have to be enabled in the kernel, and that gets us closer.
- ▶ It is possible to install and start `firewalld` on the SMS (you don’t have to verify this right now), and that’s using the same kernel as the login node.

Diagnosing 3/NOTIMPLEMENTED

- ▶ **So many** Google results amount to “reboot to get your new kernel”, but we’ve just booted a new kernel.
- ▶ Red Hat has an article telling you to verify that you haven’t disabled module loading by checking `sysctl -a | grep modules_disabled`, but that’s not disabled either.
- ▶ The Red Hat article does tell you that packet filtering capabilities have to be enabled in the kernel, and that gets us closer.
- ▶ It is possible to install and start `firewalld` on the SMS (you don’t have to verify this right now), and that’s using the same kernel as the login node.
- ▶ Or **is it?**

Diagnosing 3/NOTIMPLEMENTED

- ▶ How did we get the kernel that the login node is using?

Diagnosing 3/NOTIMPLEMENTED

- ▶ How did we get the kernel that the login node is using?
- ▶ Via `wwbootstrap $(uname -r)` on the SMS (section 3.9.1)

Diagnosing 3/NOTIMPLEMENTED

- ▶ How did we get the kernel that the login node is using?
- ▶ Via `wwbootstrap $(uname -r)` on the SMS (section 3.9.1)
- ▶ That section **also** had a command that most of us don't pay close attention to:

```
echo "drivers += updates/kernel/" >> /etc/warewulf/bootstrap.conf
```

Diagnosing 3/NOTIMPLEMENTED

- ▶ How did we get the kernel that the login node is using?
- ▶ Via `wwbootstrap $(uname -r)` on the SMS (section 3.9.1)
- ▶ That section **also** had a command that most of us don't pay close attention to:
`echo "drivers += updates/kernel/" >> /etc/warewulf/bootstrap.conf`
- ▶ So though the login node is running the same kernel **version** as the SMS, it may **not** have all the drivers included.

Diagnosing 3/NOTIMPLEMENTED

- ▶ How did we get the kernel that the login node is using?
- ▶ Via `wwbootstrap $(uname -r)` on the SMS (section 3.9.1)
- ▶ That section **also** had a command that most of us don't pay close attention to:
`echo "drivers += updates/kernel/" >> /etc/warewulf/bootstrap.conf`
- ▶ So though the login node is running the same kernel **version** as the SMS, it may **not** have all the drivers included.
- ▶ Where are the drivers we care about? `lsmod` on the SMS shows a lot of `nf`-named modules for the Netfilter kernel framework.

Diagnosing 3/NOTIMPLEMENTED

- ▶ How did we get the kernel that the login node is using?
- ▶ Via `wwbootstrap $(uname -r)` on the SMS (section 3.9.1)
- ▶ That section **also** had a command that most of us don't pay close attention to:
`echo "drivers += updates/kernel/" >> /etc/warewulf/bootstrap.conf`
- ▶ So though the login node is running the same kernel **version** as the SMS, it may **not** have all the drivers included.
- ▶ Where are the drivers we care about? `lsmod` on the SMS shows a lot of `nf`-named modules for the Netfilter kernel framework.
- ▶ `find /lib/modules/$(uname -r) -name '*nf*' shows these modules are largely located in the kernel/net folder (specifically kernel/net/ipv4/netfilter, kernel/net/ipv6/netfilter, and kernel/net/netfilter).`

Diagnosing 3/NOTIMPLEMENTED

Is kernel/net in our /etc/warewulf/bootstrap.conf at all?

```
[user1@sms ~]$ grep kernel/net /etc/warewulf/bootstrap.conf  
[user1@sms ~]$
```

Nope, let's add it.

```
[user1@sms ~]$ grep kernel/net /etc/warewulf/bootstrap.conf  
[user1@sms ~]$ echo "drivers += kernel/net/" | \  
    sudo tee -a /etc/warewulf/bootstrap.conf  
drivers += kernel/net/  
[user1@sms ~]$ grep kernel/net /etc/warewulf/bootstrap.conf  
drivers += kernel/net/
```

Diagnosing 3/NOTIMPLEMENTED

Let's re-run the wwbootstrap command and reboot the login node:

```
[user1@sms ~]$ sudo wwbootstrap $(uname -r)
...
Bootstrap image '6.1.97-1.el9.elrepo.x86_64' is ready
Done.
[user1@sms ~]$ sudo ssh login reboot
```


Did 3/NOTIMPLEMENTED go away?

```
[user1@sms ~]$ sudo ssh login systemctl status firewalld
o firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service;
          enabled; preset: enabled)
   Active: active (running) since Thu 2024-07-11 21:58:18
          EDT; 43s ago
...
Jul 11 21:58:18 login systemd[1]: Starting firewalld - dynamic
firewall daemon...
Jul 11 21:58:18 login systemd[1]: Started firewalld - dynamic
firewall daemon.
```

It did.

Does fail2ban actually work now?

```
[user1@sms ~]$ sudo ssh login grep 68.66.205.120 \  
/var/log/fail2ban.log  
...  
2024-07-11 22:02:27,030 fail2ban.actions ... [sshd] Ban \  
68.66.205.120
```

It does.

What does it look like from evilmike's side?

```
mike@server:~$ ssh evilmike@149.165.155.235
evilmike@149.165.155.235's password:
Permission denied, please try again.
evilmike@149.165.155.235's password:
Permission denied, please try again.
evilmike@149.165.155.235's password:
evilmike@149.165.155.235: Permission denied (publickey,
gssapi-keyex,gssapi-with-mic,password).
mike@server:~$ ssh evilmike@149.165.155.235
ssh: connect to host 149.165.155.235 port 22: Connection
refused
```

evilmike is thwarted, at least for now.

Why was c1 marked as down?

You can return c1 to an idle state by running
`sudo scontrol update node=c1 state=resume` on the SMS:

```
[user1@sms ~]$ sudo scontrol update node=c1 state=resume
[user1@sms ~]$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
normal*    up    1-00:00:00      1   idle c[1-2]
```

We should configure things so that we don't have to manually resume nodes every time we reboot them.

More seamless reboots of compute nodes

- ▶ Slurm doesn't like it when a node gets rebooted without its knowledge.
- ▶ There's an `scontrol reboot` option that's handy to have nodes reboot when system updates occur, but it requires a valid setting for `RebootProgram` in `/etc/slurm/slurm.conf`.
- ▶ By default, Slurm and OpenHPC don't ship with a default `RebootProgram`, so let's make one.

Adding a valid RebootProgram

```
[user1@sms ~]$ grep -i reboot /etc/slurm/slurm.conf
#RebootProgram=
[user1@sms ~]$ echo 'RebootProgram="/sbin/shutdown -r now"' \
  | sudo tee -a /etc/slurm/slurm.conf
[user1@sms ~]$ grep -i reboot /etc/slurm/slurm.conf
#RebootProgram=
RebootProgram="/sbin/shutdown -r now"
```

Informing all nodes of the changes and testing it out

```
[user1@sms ~]$ sudo scontrol reconfigure  
[user1@sms ~]$ sudo scontrol reboot ASAP nextstate=RESUME c1
```

- ▶ `scontrol reboot` will wait for all jobs on a group of nodes to finish before rebooting the nodes.
- ▶ `scontrol reboot ASAP` will immediately put the nodes in a DRAIN state, routing all pending jobs to other nodes until the rebooted nodes are returned to service.
- ▶ `scontrol reboot ASAP nextstate=RESUME` will set the nodes to accept jobs after the reboot. `nextstate=DOWN` will leave the nodes in a DOWN state if you need to do more work on them before returning them to service.

Did it work?

```
[user1@sms ~]$ sudo ssh c1 uptime
15:52:27 up 1 min,  0 users,  load average: 0.09, 0.06, 0.02
[user1@sms ~]$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
normal*    up 1-00:00:00      1   idle c[1-2]
```


Downsides of stateless provisioning

Log into c1 as root, check available disk space and memory, then allocate a 5 GB array in memory:

```
[user1@sms ~]$ sudo ssh c1
[root@c1 ~]# df -h /tmp
Filesystem      Size  Used Avail Use% Mounted on
tmpfs           2.9G  843M  2.1G  29% /
[root@c1 ~]# free -m
              total            used              free    ...
Mem:           5912             3162             2862    ...
Swap:           0              0              0    ...
[root@c1 ~]# module load py3-numpy
[root@c1 ~]# python3 -c \
    'import numpy as np; x=np.full((25000, 25000), 1)'
[root@c1 ~]#
```

Downsides of stateless provisioning

Consume some disk space in /tmp, try to allocate the same 5 GB array again:

```
[root@c1 ~]# dd if=/dev/zero of=/tmp/foo bs=1M count=1024
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB, 1.0 GiB) copied, 0.63492 s, 1.7 GB/s
[root@c1 ~]# module load py3-numpy
[root@c1 ~]# python3 -c \
    'import numpy as np; x=np.full((25000, 25000), 1)'
```

Killed

Downsides of stateless provisioning

Clean off the disk usage, allocate the 5 GB array in memory once more, and log out from the node:

```
[root@c1 ~]# rm /tmp/foo
[root@c1 ~]# python3 -c \
    'import numpy as np; x=np.full((25000, 25000), 1)'
[root@c1 ~]# exit
[user1@sms ~]$
```

Summary of the default OpenHPC settings

1. The root filesystem is automatically sized to 50% of the node memory.
2. There's no swap space.
3. Consumption of disk space affects the workloads you can run (since disk space is really in RAM).

Even if we reformat node-local storage every time we reboot, moving file storage from RAM to disk is beneficial.

Strategies

Typical bare-metal node

- ▶ PXE handled by network card, all disks available for node-local storage
- ▶ Usually, the default kernel contains all the drivers you need

Jetstream2 instance

- ▶ First disk (`/dev/vda`) exists to provide iPXE support, so don't break that
- ▶ Some extra steps may be needed to enable storage and filesystem kernel modules

Examine the existing partition scheme (non-GPU nodes)

Log back into a compute node as root, check the existing partition table:

```
[user1@sms ~]$ sudo ssh c1 parted -l /dev/vda
```

```
Model: Virtio Block Device (virtblk)
```

```
Disk /dev/vda: 21.5GB
```

```
Sector size (logical/physical): 512B/512B
```

```
Partition Table: gpt
```

```
Disk Flags:
```

Number	Start	End	Size	File system	Name	Flags
1	1049kB	3146kB	2097kB		EFI	boot, esp

Examine the existing partition scheme (GPU nodes)

Log back into a gpu node as root, check the existing partition table:

```
[rocky@sms ~]$ sudo ssh g1 parted -l /dev/vda
```

```
parted -l /dev/vda
```

```
Model: Virtio Block Device (virtblk)
```

```
Disk /dev/vda: 64.4GB
```

```
Sector size (logical/physical): 512B/512B
```

```
Partition Table: gpt
```

```
Disk Flags:
```

Number	Start	End	Size	File system	Name	Flags
1	1049kB	3146kB	2097kB		EFI	boot, esp

Summary of existing partition schemes

1. GPT (GUID partition table) method on both node types
2. Different amounts of disk space on each node type
3. Each sector is 512 bytes
4. Bootable partition 1 (from 1049 kB = 1 MiB to 3146 kB = 3 MiB) for iPXE

Plan for new partition scheme

1. Non-destructive partitioning of /dev/vda once (outside of Warewulf, with a parted script).
2. 512 MiB partition for /boot.
3. 2 GiB partition for swap.
4. 2 GiB partition for /.
5. remaining space for /tmp.

Define new partition scheme

Could make a copy of an OpenHPC-provided example partition scheme (in `/etc/warewulf/filesystem/examples`), but we'll start one from scratch:

```
[user1@sms ~]$ sudo nano \  
/etc/warewulf/filesystem/jetstream.cmds
```

- ▶ The `.cmds` file is a mix of parted commands, `mkfs` parameters, and `/etc/fstab` information.
- ▶ It's typical, **but not 100% required**, to give a full set of `select`, `mkpart`, and `name` commands for parted.

Define new partition scheme

Contents of `jetstream.cmds` (part 1):

```
select /dev/vda
```

On Jetstream2:

- ▶ we leave `/dev/vda1` unmodified, since we need it for iPXE booting,
- ▶ we “semi-manually” (i.e., outside of Warewulf, but using a script) partition the rest of `/dev/vda` to include
 - ▶ 512 MiB for `/boot`
 - ▶ 2 GiB for swap
 - ▶ 2 GiB for `/`
 - ▶ remaining space for `/tmp`

Define new partition scheme

Contents of `jetstream.cmds` (part 2):

```
# mkpart primary ext4 3MiB 515MiB
# mkpart primary linux-swap 515MiB 2563MiB
# mkpart primary ext4 2563MiB 4611MiB
# mkpart primary ext4 4611MiB 100%
name 2 boot
name 3 swap
name 4 root
name 5 tmp
```

- ▶ Note how to create partitions, and add commands to label them.
- ▶ `mkpart` commands are intended to be comments here, so that Warewulf can ignore them, but we can keep everything in one place.

Define new partition scheme

Contents of `jetstream.cmds` (part 3):

```
## mkfs NUMBER FS-TYPE [ARGS...]  
mkfs 2 ext4 -L boot  
mkfs 3 swap  
mkfs 4 ext4 -L root  
mkfs 5 ext4 -L tmp  
## fstab NUMBER mountpoint type opts freq passno  
fstab 4 / ext4 defaults 0 0  
fstab 2 /boot ext4 defaults 0 0  
fstab 3 none swap defaults 0 0  
fstab 5 /tmp ext4 defaults 0 0
```

- ▶ Format partitions and mount them.
- ▶ Save and exit nano with Ctrl-X.

Partition the disks outside of Warewulf

- ▶ parted has a `--script` parameter helpful for passing in one or more commands at the command line.
- ▶ We want to pass in the commented `mkpart` commands of our `jetstream.cmds` file.

Show the `mkpart` lines

```
[user1@sms ~]$ grep mkpart \  
/etc/warewulf/filesystem/jetstream-vda.cmds  
# mkpart primary ext4 3MiB 515MiB  
# mkpart primary linux-swap 515MiB 2663MiB  
# mkpart primary ext4 2663MiB 4611MiB  
# mkpart primary ext4 4611MiB 100%
```

Take out the # signs from the `mkpart` lines

```
[user1@sms ~]$ grep mkpart \  
/etc/warewulf/filesystem/jetstream-vda.cmds | sed 's/#//g'  
mkpart primary ext4 3MiB 515MiB  
mkpart primary linux-swaps 515MiB 2663MiB  
mkpart primary ext4 2663MiB 4611MiB  
mkpart primary ext4 4611MiB 100%
```


Put all the commands on one line

```
[user1@sms ~]$ echo $(grep mkpart \  
    /etc/warewulf/filesystem/jetstream-vda.cmds | sed 's/#//g')  
mkpart primary ext4 3MiB 515MiB mkpart primary linux-swap  
515MiB 2663MiB mkpart primary ext4 2663MiB 4611MiB mkpart  
primary ext4 4611MiB 100%
```

Partition the drive

(all of the below goes on one literal line, no backslashes, line breaks, or anything else)

```
[user1@sms ~]$ sudo ssh g2 parted --script /dev/vda  
$(echo $(grep mkpart  
/etc/warewulf/filesystem/jetstream-vda.cmds |  
sed 's/#//g'))
```

Check your results

```
[user1@sms ~]$ sudo ssh g2 parted -l
Model: Virtio Block Device (virtblk)
Disk /dev/vda: 64.4GB
...
Number  Start   End     Size    File system  Name      Flags
  1      1049kB  3146kB  2097kB                EFI      boot, esp
  2      3146kB  540MB   537MB   ext4          primary
  3      540MB   2792MB  2252MB  linux-swap(v1) primary  swap
  4      2792MB  4835MB  2043MB  ext4          primary
  5      4835MB  64.4GB  59.6GB  ext4          primary
```

Now repeat the previous `sudo ssh NODE parted --script` command for the other nodes (c1, c2, and g1).

Apply the Warewulf filesystem provisioning commands to the nodes

```
[user1@sms ~]$ sudo wwsh provision set 'c*' \  
--filesystem=jetstream-vda  
[user1@sms ~]$ sudo wwsh provision set 'g*' \  
--filesystem=jetstream-vda
```

Do not reboot your nodes yet!

What could possibly go wrong?

- ▶ A lot, if you consider some edge cases and corner cases.
- ▶ This was by far the slowest-progressing and most error-prone section of the tutorial to develop.
- ▶ Using `wsh` provision set `NODE --preshell=1` and/or `--postshell=1` during debugging was invaluable.
- ▶ Rather than have y'all suffer through this without easy access to a console, I'll take you through what would have gone wrong if we'd rebooted just now.

What went wrong (part 1)

Starting the provision handler:

```
* adhoc-pre  
* ipmiconfig Auto configuration not activated  
* filesystems  
  * /dev/vdb not ready, retrying.....
```

OK
SKIPPED
RUNNING
ERROR

Figure 2: Device not ready, retrying

- ▶ Running `dmesg | grep vd` at the postshell command prompt confirmed that no `/dev/vd` devices were found.

How it got fixed (part 1)

- ▶ Comparing the `lsmod` output on the failing node versus the SMS indicated we were missing the `virtio_blk` kernel module.
- ▶ Running `modprobe virtio_blk` and `dmesg | grep vd` at the postshell command prompt confirmed this.
- ▶ Warewulf fix is to:
 - ▶ run
`echo modprobe += virtio_blk | sudo tee -a /etc/warewulf/bootstrap.conf`
 - ▶ run `sudo wwbootstrap KERNEL_VERSION`
 - ▶ reboot the node and try again.

What went wrong (part 2)

```
* filesystems                                RUNNING
* running parted on /dev/vdb                OK
* formatting /dev/vdb2                      OK
* formatting /dev/vdb3                      OK
* formatting /dev/vdb4                      OK
* formatting /dev/vdb5                      OK
* mounting /                                ERROR
                                           ERROR
```

Figure 3: Mounting /, error

How it got fixed (part 2)

```
/ # parted -l
Model: Virtio Block Device (virtblk)
Disk /dev/vdb: 10.7GB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Disk Flags:

Number   Start    End      Size    File system  Name      Flags
  1      1049kB   3146kB   2097kB   xfs          primary
  2      3146kB   538MB    535MB    ext4         boot
  3      538MB    2635MB   2097MB   linux-swap(v1) swap      swap
  4      2635MB   4732MB   2097MB   ext4         root
  5      4732MB   10.7GB   6004MB   ext4         tmp
```

Figure 4: parted -l looks ok

- ▶ running parted -l showed a valid partition table

How it got fixed (part 2)

- ▶ Trying to mount the proposed root partition with
`mkdir /mnt ; mount -t auto /dev/sdb4 /mnt` failed with
`mount: mounting /dev/vdb4 as /mnt failed: No such file or directory`

How it got fixed (part 2)

- ▶ Trying to mount the proposed root partition with
`mkdir /mnt ; mount -t auto /dev/sdb4 /mnt` failed with
`mount: mounting /dev/vdb4 as /mnt failed: No such file or directory`
- ▶ But both `/mnt` and `/dev/sdb4` both existed, as seen from `ls -l` on each of them.

How it got fixed (part 2)

- ▶ Trying to mount the proposed root partition with `mkdir /mnt ; mount -t auto /dev/sdb4 /mnt` failed with `mount: mounting /dev/vdb4 as /mnt failed: No such file or directory`
- ▶ But both `/mnt` and `/dev/sdb4` both existed, as seen from `ls -l` on each of them.
- ▶ Surprisingly, when I left the root partition as a ramdisk and tried to partition and mount swap and `/tmp` from disk partitions, provisioning threw errors, but post-provisioning, both swap and `/tmp` **were available** to the node!

How it got fixed (part 2)

- What was different? A missing filesystem module in the provisioning kernel (in my case, ext4).

How it got fixed (part 2)

- ▶ What was different? A missing filesystem module in the provisioning kernel (in my case, ext4).
- ▶ Running `modprobe ext4` at the `postshell` command prompt and re-running the `mount` command above caused the filesystem to mount.

How it got fixed (part 2)

- ▶ What was different? A missing filesystem module in the provisioning kernel (in my case, ext4).
- ▶ Running `modprobe ext4` at the `postshell` command prompt and re-running the `mount` command above caused the filesystem to mount.
- ▶ Warewulf fix is to:

How it got fixed (part 2)

- ▶ What was different? A missing filesystem module in the provisioning kernel (in my case, ext4).
- ▶ Running `modprobe ext4` at the `postshell` command prompt and re-running the `mount` command above caused the filesystem to mount.
- ▶ Warewulf fix is to:
 - ▶ `run echo modprobe += ext4 | sudo tee -a /etc/warewulf/bootstrap.conf`

How it got fixed (part 2)

- ▶ What was different? A missing filesystem module in the provisioning kernel (in my case, ext4).
- ▶ Running `modprobe ext4` at the `postshell` command prompt and re-running the `mount` command above caused the filesystem to mount.
- ▶ Warewulf fix is to:
 - ▶ run `echo modprobe += ext4 | sudo tee -a /etc/warewulf/bootstrap.conf`
 - ▶ run `sudo wwbootstrap KERNEL_VERSION`

How it got fixed (part 2)

- ▶ What was different? A missing filesystem module in the provisioning kernel (in my case, ext4).
- ▶ Running `modprobe ext4` at the `postshell` command prompt and re-running the `mount` command above caused the filesystem to mount.
- ▶ Warewulf fix is to:
 - ▶ run `echo modprobe += ext4 | sudo tee -a /etc/warewulf/bootstrap.conf`
 - ▶ run `sudo wwbootstrap KERNEL_VERSION`
 - ▶ reboot the node and try again.

Make the necessary `wwbootstrap` changes, then reboot your nodes

```
[user1@sms ~]$ echo modprobe += virtio_blk | \  
sudo tee -a /etc/warewulf/bootstrap.conf  
[user1@sms ~]$ echo modprobe += ext4 | \  
sudo tee -a /etc/warewulf/bootstrap.conf  
[user1@sms ~]$ sudo wwbootstrap $(uname -r)  
[user1@sms ~]$ sudo pdsh -w 'c[1-2],g[1-2]' reboot
```

Final result on a compute node (part 1)

```
[user1@sms ~]$ sudo ssh c1 "df -h; free -m"
```

Filesystem	Size	Used	Avail	Use%	Mounted on	
devtmpfs	2.9G	0	2.9G	0%	/dev	
/dev/vda4	1.9G	853M	914M	49%	/	
tmpfs	2.9G	0	2.9G	0%	/dev/shm	
tmpfs	1.2G	8.5M	1.2G	1%	/run	
/dev/vda2	488M	40K	452M	1%	/boot	
/dev/vda5	16G	72K	15G	1%	/tmp	
...						
	total	used	free	shared	buff/cache	available
Mem:	5912	382	4844	8	939	5530
Swap:	2047	0	2047			

Note that the used memory column has dropped by nearly 90% from before.

Final result on a compute node (part 2)

Consume 5 GiB of space in /tmp (we only used 1 GiB previously), then allocate 5 GB for an array in memory:

```
[user1@sms ~]$ sudo ssh c1
[root@c1 ~]# dd if=/dev/zero of=/tmp/foo bs=1M count=5120
5120+0 records in
5120+0 records out
5368709120 bytes (5.4 GB, 5.0 GiB) copied, 8.97811 s, 598 MB/s
[root@c1 ~]# module load py3-numpy
[root@c1 ~]# python3 -c \
    'import numpy as np; x=np.full((25000, 25000), 1)'
[root@c1 ~]# rm /tmp/foo
```

No Killed messages due to running out of memory. We're able to consume much more /tmp space and all practically the RAM without conflict.

Decoupling kernels from the SMS

- ▶ If you keep your HPC around for a long period, you might want/need to support different operating systems or releases.
- ▶ Maybe you need to run a few nodes on Rocky 8 while keeping the SMS on Rocky 9 (`wwmkchroot` supports that).
- ▶ Maybe you need to use a different kernel version for exotic hardware or new features, but don't want to risk the stability of your SMS.
- ▶ A simple `wwbootstrap $(uname -r)` won't do that.

Decoupling kernels from the SMS

Check `wwbootstrap --help`:

```
[user1@sms ~]$ wwbootstrap --help
USAGE: /usr/bin/wwbootstrap [options] kernel_version
...
  OPTIONS:
    -c, --chroot    Look into this chroot directory to find
                    the kernel
...
```

So if we install a kernel into the `${CHROOT}` like any other package, we can bootstrap from it instead of the SMS kernel.

Install a different kernel into the CHROOT, bootstrap it

```
[user1@sms ~]$ sudo yum -y install --installroot=$CHROOT kernel
...
Installing:
  kernel    x86_64  5.14.0-427.24.1.el9_4  ...
...
Complete!
[user1@sms ~]$ sudo wwbootstrap --chroot=${CHROOT} \
  5.14.0-427.24.1.el9_4.x86_64
Number of drivers included in bootstrap: 880
...
Bootstrap image '5.14.0-427.24.1.el9_4.x86_64' is ready
Done.
```


Check your nodes' provisioning summary

```
[user1@sms ~]$ wwsh provision list
```

NODE	VNFS	BOOTSTRAP	...
c1	rocky9.4	6.1.97-1.el9.elrep...	
c2	rocky9.4	6.1.97-1.el9.elrep...	
g1	rocky9.4	6.1.97-1.el9.elrep...	
g2	rocky9.4	6.1.97-1.el9.elrep...	
login	rocky9.4	6.1.97-1.el9.elrep...	

Change the default kernel for nodes, reboot them.

```
[user1@sms ~]$ sudo wssh provision set '*' \  
--bootstrap=5.14.0-427.24.1.el9_4.x86_64  
Are you sure you want to make the following changes to 5  
node(s):  
  
      SET: BOOTSTRAP                = 5.14.0-427.24.1.el9_4.x86_64  
  
Yes/No> y  
[user1@sms ~]$ sudo scontrol reboot ASAP nextstate=RESUME \  
c[1-2]  
[user1@sms ~]$ sudo pdsh -w 'g[1-2],login' reboot
```

Verify everything came back up

```
[user1@sms ~]$ sudo pdsh -w 'c[1-2],g[1-2],login' uname -r \
| sort
c1: 5.14.0-427.24.1.el9_4.x86_64
c2: 5.14.0-427.24.1.el9_4.x86_64
g1: 5.14.0-427.24.1.el9_4.x86_64
g2: 5.14.0-427.24.1.el9_4.x86_64
login: 5.14.0-427.24.1.el9_4.x86_64
[user1@sms ~]$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
normal*    up    1-00:00:00      2   idle c[1-2]
```

Management of GPU drivers

(installing GPU drivers – mostly rsync'ing a least-common-denominator chroot into a GPU-named chroot, copying the NVIDIA installer into the chroot, mounting /proc and /sys, running the installer, umounting /proc and /sys, and building a second VNFS)

See what we have, download the driver

```
[user1@sms ~]$ sudo ssh g1 lspci | grep -i nvidia
06:00.0 3D controller: NVIDIA Corporation GA100 [A100 SXM4
    40GB] (rev a1)
[user1@sms ~]$ export NV=550.90.07
[user1@sms ~]$ export B=https://us.download.nvidia.com/tesla/
[user1@sms ~]$ wget \
    ${B}/${NV}/NVIDIA-Linux-x86_64-${NV}.run
```

Prepare to install the driver

```
[user1@sms ~]$ chmod 755 NVIDIA-Linux-x86_64-${NV}.run  
[user1@sms ~]$ sudo mount -o rw,bind /proc ${CHROOT}/proc  
[user1@sms ~]$ sudo mount -o rw,bind /dev ${CHROOT}/dev  
[user1@sms ~]$ sudo cp NVIDIA-Linux-x86_64-${NV}.run \  
    ${CHROOT}/root
```

Install the driver, clean up, update VNFS

```
[user1@sms ~]$ sudo chroot ${CHROOT} \  
/root/NVIDIA-Linux-x86_64-${NV}.run \  
--kernel-name=5.14.0-427.24.1.el9_4.x86_64 \  
--disable-nouveau --silent --run-nvidia-xconfig --no-drm  
[user1@sms ~]$ sudo rm \  
${CHROOT}/root/NVIDIA-Linux-x86_64-${NV}.run  
[user1@sms ~]$ sudo umount ${CHROOT}/proc ${CHROOT}/dev  
[user1@sms ~]$ sudo wwnfs --chroot=${CHROOT}
```

Configuration settings for different node types

What tools have we used so far to define node settings?

1. `wwsh node` for node name and network information (MACs, IPs, provisioning interface)
2. `wwsh provision` for VNFS, kernel, kernel parameters, files
3. When the files include `systemd` services, other options become possible via `ConditionHost` or similar statements

Manually building these up over time and storing the results in the Warewulf database may be tedious to review, and we might want to easily port our setup to a dev/test environment, a new version of OpenHPC, etc.

Automation for Warewulf3 provisioning

Any kind of automation, scripting, or orchestration is beneficial for managing cluster settings:

- ▶ shell scripts,
- ▶ Python scripts,
- ▶ Ansible playbooks,
- ▶ Puppet manifests,
- ▶ etc.

Mike's used Ansible as part of the Basic Cluster project; Tim's `ohpc-jetstream2` repository does the same. TN Tech uses Python scripts for their Warewulf management.

Configuring Slurm policies

Can adapt a lot of Mike's CaRCC Emerging Centers talk from a couple years ago for this. Fair share, hard limits on resource consumption, QOSes for limiting number of GPU jobs or similar.