Pi Cluster with Rocky Linux *(Stable)*

Following this document to create a rocky version:

- https://aoterodelaroza.github.io/devnotes/diskless-rpi-cluster/
- https://docs.rockylinux.org/guides
- https://j3b.in/pihpc/
-

Introduction

We are building a Raspberry Pi cluster using Rocky Linux.
One head node and 4 compute nodes.
Hardware:

- 5 Raspberry Pi Model 4 B with power cables
- Netgear network switch
- 5 ethernet cables
- 10 USB flash drives 32GB

Commands issued on **head** node will be denoted as:

```
head$ <command>
```

Commands issued on **compute** node will be denoted as:

```
compute$ <command>
```

Commands issued on **chroot** node will be denoted as:

```
chroot$ <command>
```

Commands issued on **auxiliary laptop** will be denoted as:

```
aux$ <command>
```

Follow this step by step guide to make a linux breakthrough in this AI world.

Flash the USB with Rocky

First will start by configuring the head node so we need to flash the usb drive Rocky Linux (Raspberry Pi ARM specific version).

It can be done either by Raspberry Pi's imager or just using shell.
First download latest Rocky linux image for raspberry pi ARM https://rockylinux.org/download

By Raspberry Pi's imager:

Use your auxiliary laptop
Download the imager from the site https://www.raspberrypi.com/software/
Now flash the usb by following the steps in imager.

By using shell:

Open terminal/shell in your auxiliary laptop and download the rocky image. Then uncompress it and flash the usb using dd.

Warning: BE CAREFUL, be sure which device you are writing to before you do it! Don't accidentally blow away your laptop/desktop hard drive!

Make sure to go through the readme file that comes with the rocky image as it contains lots of useful information like default login credentials.

Check if the image file correctly downloaded

```
aux$ file RockyLinux-RPi-9.5-aarch64.raw.xz
```

Uncompress the file

```
aux$ unxz RockyLinux-RPi-9.5-aarch64.raw.xz
```

Identify your USB drive mount point

```
aux$ diskutil list
```

Unmount the USB Drive

Unmount the drive (do not eject it yet):

```
aux$ sudo diskutil unmountDisk /dev/disk4
```

Flash the Image to the USB Drive

Warning: This erases all data!

```
aux$ sudo dd if=RockyLinux-RPi-9.5-aarch64.raw of=/dev/rdisk4 bs=1m
```

Eject the USB Drive

```
aux$ sudo diskutil eject /dev/disk4
```

Creating users

First setup root password

By default our rocky comes with a root user but we have to set its password.

```
head$ sudo passwd root
...
```

Now will create a new user

Will create a new user ***user1*** and add it to the **wheel** group for sudo access.

```
head$ useradd -m user1
head$ passwd user1
...
head$ sudo usermod -aG wheel user1
```

June 19:

Command used to create users and group:

```
$ echo '%hpcadmins ALL=(ALL) ALL' > /etc/sudoers.d/hpcadmins
$ chmod 440 /etc/sudoers.d/hpcadmins
$ groupadd hpcadmins
$ useradd -m -s /bin/bash -g hpcadmins -G hpcadmins ls565
$ passwd ls565
```

Transferring these users and group to chroot

```
[root@head /]# grep -E '^(ls565|kjc59|as4938|am2455|hz3):' /etc/passwd >
/tmp/passwd.hpc
[root@head /]# cat /tmp/passwd.hpc
ls565:x:5002:5002::/home/ls565:/bin/bash
kjc59:x:5003:5002::/home/kjc59:/bin/bash
am2455:x:5004:5002::/home/am2455:/bin/bash
hz3:x:5005:5002::/home/hz3:/bin/bash
as4938:x:5006:5002::/home/as4938:/bin/bash
[root@head /]# grep -E '^(ls565|kjc59|as4938|am2455|hz3):' /etc/shadow >
/tmp/shadow.hpc
[root@head /]# grep '^hpcadmins:' /etc/group > /tmp/group.hpc
[root@head /]# cat /tmp/passwd.hpc >> /image/pi4/etc/passwd
[root@head /]# cat /tmp/shadow.hpc >> /image/pi4/etc/shadow
[root@head /]# cat /tmp/group.hpc >> /image/pi4/etc/group
[root@head /]# cp /etc/sudoers.d/hpcadmins /image/pi4/etc/sudoers.d/
[root@head /]# chmod 440 /image/pi4/etc/sudoers.d/hpcadmins
```

Change hostname of head node

Currently our head node might be displaying locahost as hostname so let’s change it

```
head$ nmtui
...
```

Now follow the instruction to set new hostname. After that it will start displaying **head**.
You can also verify this by check the file ***/etc/hostname***

Setup Network

Will set up a network connection for our head node to be able to talk to the outside world. Will connect to **NJIoT** wifi. The ethernet interface on head node will be used to connect with internal cluster via network switch.

First turn off randomized mac address

Your mac address will keep on changing on every reboot. Apparently NetworkManager can randomize your mac to make it harder to people to snoop on traffic.
To disable this you need to create a new file inside ***/etc/NetworkManager*** and add following lines to it.

```
head$ vi /etc/NetworkManager/conf.d/100-disable-wifi-mac-randomization.conf
```

```
...
[connection]
wifi.mac-address-randomization=1

[device]
wifi.scan-rand-mac-address=no
```

Connecting to wifi

First scan the list of available wifi and note down your desired SSID

```
head$ nmcli device wifi list
```

Then use the following command to connect

```
head$ nmcli device wifi connect <SSID> password <your_wifi_password>
```

Congrats!! Now you can ***ssh*** into your head node from other device.

Configure the package manager

Note: We won't be changing dnf or yum source urls unlike the tutorial because in Rocky the url scheme is pretty complex and might mess up the whole process. So instead will just use the default urls provided. If something doesn't work then will revisit this step.

Upgrade the system

Grab your snacks because this command is gonna take a while.

```
head$ dnf -y upgrade
```

Create the compute node image

```
head$ sudo mkdir -p /image/pi4
head$ sudo chmod go-rX /image  # Restrict access to the image directory
```

Install Required Tools

Ensure dnf and necessary tools are installed

```
head$ sudo dnf install dnf-plugins-core
```

Build the Base System Image for compute node

Lets create a minimal Rocky base system inside ***/image/pi4*** directory. This will be mounted on later to the compute node via NFS.
Before we proceed make sure you have enough space in the root partition. If not then extend the partition for at least 4gb.

```
head$ sudo dnf --installroot=/image/pi4/ --releasever=9 groupinstall
"Minimal Install"
```

This will create a minimal system that we can chroot.
Now we want install packages inside this chroot environment, so we need to have more functionality from /dev, /proc and /sys. So let's mount them

```
head$ mount --rbind /dev /image/pi4/dev
head$ mount --make-rslave /image/pi4/dev
head$ mount -t proc /proc /image/pi4/proc
head$ mount --rbind /sys /image/pi4/sys
head4 mount --make-rslave /image/pi4/sys
head$ mount --rbind /tmp /image/pi4/tmp
# Now check it
head$ chroot /image/pi4
```

More info about these commands
https://www.kernel.org/doc/Documentation/filesystems/sharedsubtree.txt

Since we will be chrooting into the image quite a number of times, it is best if the system mounts the above directories automatically on start-up. To do this, put the following lines in the head node's fstab file:

```
head$ vi /etc/fstab
...
/dev /image/pi4/dev none defaults,rbind,rslave 0 0
```

```
/proc /image/pi4/proc proc defaults 0 0
/sys /image/pi4/sys none defaults,rbind,rslave 0 0
/tmp /image/pi4/tmp none defaults,rbind 0 0
```

Configure the local (wired) network

Edit the ***/etc/hosts*** file and fill in the names of the head node and the compute nodes. In my cluster the head node is **head** and the compute nodes are n1, n2…. etc.

```
head$ vi /etc/hosts
127.0.0.1       localhost.localdomain localhost

192.168.1.1     head
192.168.1.10    n1

::1             localhost6.localdomain6 localhost6
```

Note that we are using the IPs 192.168.1.1, 192.168.1.10,… for the local network, with the first one being the IP for the head node.

Then propagate the hosts file to the compute node image:

```
head$ cp /etc/hosts /image/pi4/etc/hosts
```

Also copy the dns configuration file

```
head$ cp /etc/resolv.conf /image/pi4/etc/resolv.conf
```

Lets also set static ip for our head node

```
head$ vi /etc/NetworkManager/system-connections/eth0.nmconnection
...
[connection]
id=eth0
uuid=<generate uuid using uuidgen command>
type=ethernet
timestamp=1739226118
```

```
[ethernet]

[ipv4]
address1=192.168.1.1/24,192.168.1.1
dns=8.8.8.8;8.8.4.4;
method=manual
route-metric=200

[ipv6]
method=ignore

[proxy]
```

Make sure to put newly generated **uuid** at the place
Just restart the network manager and check static ip with ip a
If it doesn't show then turn on eth0 using nmcli connection up eth0

Note: Now your wifi might stop working due to network preference. You can check that by using ip route show command. If the wlan0 has higher route metric then make it lower then eth0.

On reboot the eth0 is not automatically connected despite autoconnect=yes.
So currently doing **nmcli connection up eth0** on every reboot.

Change the prompt name of chroot

This way, you will be able to tell when you are in the chroot and when you are connected to a compute node via SSH.

```
head$ vi /image/pi4/root/.bashrc
...
if [[ "$HOSTNAME" != "head" ]] ; then
   export PS1="\[\]\[\]\h:\W#\[\] "
else
   export PS1="chroot:\W# "
fi
```

Set timezone

I am in New York but you should set the time accordingly.

```
head$ sudo timedatectl set-timezone America/New_York
head$ date    # verify the time
```

Now let's set time for chroot. Since it doesn't have systemd so we have to do it other way.

```
chroot$ ln -sf /usr/share/zoneinfo/America/New_York /etc/localtime
chroot$ echo "America/New_York" > /etc/timezone
```

Use chrony to make sure all clocks are on the correct time. Head should use cronie and compute node should sync time with head. **USE NTP**
Use this a guide https://j3b.in/pihpc/modules/chrony

Upgrade and install packages

We need to install firmware, kernel image, ssh server and utility packages. However, it is a good practice to update the os and upgrade existing packages before new installations.

```
chroot$ dnf update
chroot$ dnf upgrade --refresh
```

Now we install necessary packages and also remove some which might not be needed
Install Kernel and Firmware

```
# Enable EPEL (Extra Packages for Enterprise Linux)
chroot$ dnf install epel-release

# Install firmware packages
chroot$ dnf install linux-firmware wireless-regdb
```

```
# Core networking/SSH tools
chroot$ dnf install openssh-server openssh-clients net-tools nfs-utils xz

# Optional utilities (Emacs/BIND tools)
chroot$ dnf install emacs bind-utils dbus
```

```
# Remove anacron and exim (if separate package exists)
chroot$ dnf remove exim* || echo "exim not installed"
chroot$ dnf remove anacron || echo "anacron not installed"
```

Exim is an MTA (mail transfer agent), which isn't needed on an HPC node, so removing it makes sense to reduce attack surface and resource usage(in our case exim is not installed). Anacron is for scheduling periodic tasks on systems that aren't always on, which an HPC cluster node probably is always on, so using cron instead is better. So the user should check if these packages exist in Rocky and remove them if present.

```
chroot$ dnf kernel install linux-firmware raspberrypi2-firmware
wireless-regdb bluez net-tools nfs-utils openssh-server openssh-clients
bind-utils dbus xz dracut-network
```

Modify Initramfs for Network Boot

Create a dracut config file in the chroot:

```
chroot$ vi /etc/dracut.conf.d/network.conf
omit_drivers+=" wireless iwlwifi "   # Exclude unnecessary drivers
add_dracutmodules+=" network nfs "
kernel_cmdline+=" root=/dev/nfs nfsroot=192.168.1.1:/image/pi4 ip=dhcp rw "
```

Errors in ***dracut -f /boot/initramfs-$(uname -r).img $(uname -r)***
ERROR

```
dracut: Turning off host-only mode: '/run' is not mounted!
dracut: dracut module 'nfs' cannot be found or installed.
```

Resolution:
Check if nfs is installed:

```
chroot:/# rpm -q nfs-utils
nfs-utils-2.5.4-27.el9.aarch64
```

That means /run is not mounted, mount the following:

But first reload daemon:

```
systemctl daemon-reload
```

Then run these commands to mount:

```
head$ mount --bind /proc /image/pi4/proc
mount --bind /sys /image/pi4/sys
mount --bind /dev /image/pi4/dev
mount --bind /run /image/pi4/run
```

Now run **dracut -f -v /boot/initramfs-$(uname -r).img $(uname -r)** this command again but with -v flag.

ERROR

```
chroot$ dracut -f -v /boot/initramfs-$(uname -r).img $(uname -r)

dracut: Executing: /usr/bin/dracut -f -v
/boot/initramfs-6.1.31-v8.1.el9.altarch.img 6.1.31-v8.1.el9.altarch
dracut: dracut module 'systemd-networkd' will not be installed, because
command 'networkctl' could not be found!
dracut: dracut module 'systemd-networkd' will not be installed, because
command '/usr/lib/systemd/systemd-networkd' could not be found!
dracut: dracut module 'systemd-networkd' will not be installed, because
command '/usr/lib/systemd/systemd-networkd-wait-online' could not be found!
dracut: dracut module 'systemd-resolved' will not be installed, because
command 'resolvectl' could not be found!
dracut: dracut module 'systemd-resolved' will not be installed, because
command '/usr/lib/systemd/systemd-resolved' could not be found!
dracut: dracut module 'systemd-timesyncd' will not be installed, because
command '/usr/lib/systemd/systemd-timesyncd' could not be found!
dracut: dracut module 'systemd-timesyncd' will not be installed, because
command '/usr/lib/systemd/systemd-time-wait-sync' could not be found!
dracut: dracut module 'busybox' will not be installed, because command
'busybox' could not be found!
dracut: dracut module 'dbus-daemon' will not be installed, because command
'dbus-daemon' could not be found!
dracut: dracut module 'rngd' will not be installed, because command 'rngd'
could not be found!
dracut: dracut module 'connman' will not be installed, because command
'connmand' could not be found!
dracut: dracut module 'connman' will not be installed, because command
'connmanctl' could not be found!
dracut: dracut module 'connman' will not be installed, because command
'connmand-wait-online' could not be found!
dracut: dracut module 'network-legacy' will not be installed, because
command 'dhclient' could not be found!
dracut: dracut module 'network-wicked' will not be installed, because
command 'wicked' could not be found!
dracut: dracut module 'lvmmerge' will not be installed, because command
'lvm' could not be found!
dracut: dracut module 'lvmthinpool-monitor' will not be installed, because
command 'lvm' could not be found!
dracut: dracut module 'btrfs' will not be installed, because command
'btrfs' could not be found!
dracut: dracut module 'dmraid' will not be installed, because command
'dmraid' could not be found!
```

```
dracut: dracut module 'lvm' will not be installed, because command 'lvm'
could not be found!
dracut: dracut module 'mdraid' will not be installed, because command
'mdadm' could not be found!
dracut: dracut module 'pcsc' will not be installed, because command 'pcscd'
could not be found!
dracut: dracut module 'tpm2-tss' will not be installed, because command
'tpm2' could not be found!
dracut: dracut module 'cifs' will not be installed, because command
'mount.cifs' could not be found!
dracut: dracut module 'iscsi' will not be installed, because command
'iscsi-iname' could not be found!
dracut: dracut module 'iscsi' will not be installed, because command
'iscsiadm' could not be found!
dracut: dracut module 'iscsi' will not be installed, because command
'iscsid' could not be found!
findmnt: can't read (null): No such file or directory
dracut: 95nfs: Could not find any command of 'rpcbind portmap'!
dracut: dracut module 'nvmf' will not be installed, because command 'nvme'
could not be found!
dracut: dracut module 'biosdevname' will not be installed, because command
'biosdevname' could not be found!
dracut: dracut module 'memstrack' will not be installed, because command
'memstrack' could not be found!
dracut: memstrack is not available
dracut: If you need to use rd.memdebug>=4, please install memstrack and
procps-ng
dracut: dracut module 'nfs' cannot be found or installed.
```

Solved the error by installing following:

```
chroot$ dnf install -y lvm2 dracut-network rpcbind busybox dhcp-client
systemd-networkd lvm2 mdadm btrfs-progs dbus-daemon tpm2-tss cifs-utils
iscsi-initiator-utils nfs-utils kernel-core
```

Now run:

```
chroot$ dracut -f -v /boot/initramfs-$(uname -r).img $(uname -r)
```

You might still see some errors but don't worry, image will be created.
If you see an error related to “rpcuser” then its not good.

Debian: In Debian-based systems, the nfs-common package (which provides NFS client utilities) automatically creates the rpcuser user and group during installation. This happens silently, so you might not have noticed it.

Rocky Linux: In Rocky Linux (and other Red Hat-based distributions), the nfs-utils package does not automatically create the rpcuser user/group. Instead, it assumes the user/group already exists or will be created manually.

So let's fix it:

```
chroot$ groupadd -r rpcuser
chroot$ useradd -r -g rpcuser rpcuser
```

Then rebuild the initramfs:

```
chroot$ dracut -f -v /boot/initramfs-$(uname -r).img $(uname -r)
```

Configure ramdisk for temporary files in the image

The compute nodes will have a read-only root filesystem (/), so temporary files cannot be written to /tmp in the usual way. To prevent this from causing errors, create a RAM partition for temporary files, so they will be written to memory instead. To do this, insert the following lines in the image configuration files:

```
chroot$ vi /etc/fstab
...
tmpfs   /tmp    tmpfs   defaults,noatime,mode=1777   0 0
```

Setup TFTP

Use this guide:
https://computingforgeeks.com/setup-tftp-server-on-centos-rhel-rocky-linux/#google_vignette

Prepare the compute nodes to boot over the network

Follow this guide's step 17 except install the latest version of Raspbian OS.

Configure the DHCP Server

Install the DHCP Server

```
head$ dnf install dhcp-server -y
```

Now modify the configuration file.

Note we used the MAC address of each node and associated the corresponding IPs and names to them. We also indicated the location of the NFS root filesystem of the compute nodes. If you have different computers on your cluster that require different images, they can be served with different image directories by modifying this file.

```
head$ vi /etc/dhcp/dhcpd.conf
…
ddns-update-style none;
use-host-decl-names on;
default-lease-time 600;
max-lease-time 7200;

subnet 192.168.1.0 netmask 255.255.255.0 {  # Adjust if using a different subnet
  option routers 192.168.1.1;
  option subnet-mask 255.255.255.0;
  option broadcast-address 192.168.1.255;

  group {
    option tftp-server-name "192.168.1.1";  # Option 66
    option vendor-class-identifier "PXEClient";  # Option 60
    option vendor-encapsulated-options "Raspberry Pi Boot";  # Option 43
    filename "pxelinux.0";  # PXE bootloader

    host head {
      hardware ethernet D8:3A:DD:FB:3B:AF;  # MAC address of head node
      fixed-address 192.168.1.1;
      option host-name "head";
    }

    host n1 {
      hardware ethernet D8:3A:DD:DB:3E:0C;  # MAC address of compute node
n1
      fixed-address 192.168.1.10;
      option host-name "n1";
      option root-path
"192.168.1.1:/image/pi4,v3,tcp,hard,rsize=8192,wsize=8192";  # NFS root
    }
  }
}
```

Enable and start DHCP

```
head$ systemctl enable dhcpd
head$ systemctl start dhcpd
```

Keep checking the status and log

```
head$ systemctl status dhcpd
head$ journalctl -f
```

Open the Firewall for DHCP and PXE Booting

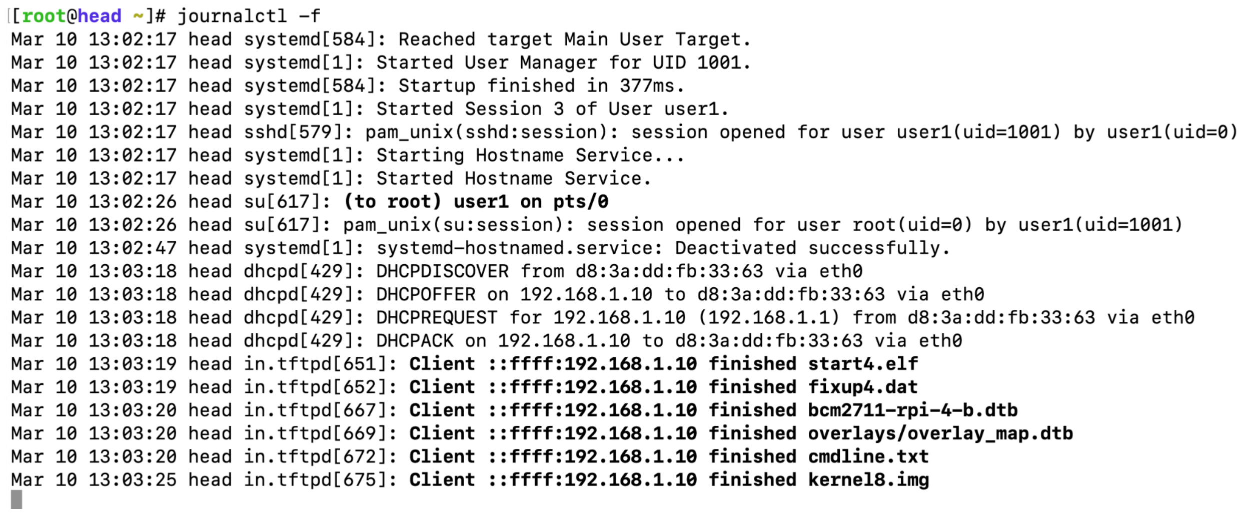
```
head$ firewall-cmd --add-service=dhcp --permanent
head$ firewall-cmd --add-port=67/udp --permanent
head$ firewall-cmd --add-port=69/udp --permanent
head$ firewall-cmd --add-port=4011/udp --permanent # PXE Boot port
head$ firewall-cmd --reload
```

Copy the boot files to tftp

Copy all the boot files of head node to tftp folder so that they can be served to compute node for provisioning. Also make sure to set proper read access

```
head$ cp -R /boot/* /srv/tftp/
head$ chmod 644 /srv/tftp
```

Now try to boot your compute node without any pendrive, sdcard but just with ethernet connected to head node. It should boot upto some point because nfs is still not mounted.
You can check the logs:



```
[root@head ~]# journalctl -f
Mar 10 13:02:17 head systemd[584]: Reached target Main User Target.
Mar 10 13:02:17 head systemd[1]: Started User Manager for UID 1001.
Mar 10 13:02:17 head systemd[584]: Startup finished in 377ms.
Mar 10 13:02:17 head systemd[1]: Started Session 3 of User user1.
Mar 10 13:02:17 head sshd[579]: pam_unix(sshd:session): session opened for user user1(uid=1001) by user1(uid=0)
Mar 10 13:02:17 head systemd[1]: Starting Hostname Service...
Mar 10 13:02:17 head systemd[1]: Started Hostname Service.
Mar 10 13:02:26 head su[617]: (to root) user1 on pts/0
Mar 10 13:02:26 head su[617]: pam_unix(su:session): session opened for user root(uid=0) by user1(uid=1001)
Mar 10 13:02:47 head systemd[1]: systemd-hostnamed.service: Deactivated successfully.
Mar 10 13:03:18 head dhcpd[429]: DHCPDISCOVER from d8:3a:dd:fb:33:63 via eth0
Mar 10 13:03:18 head dhcpd[429]: DHCPOFFER on 192.168.1.10 to d8:3a:dd:fb:33:63 via eth0
Mar 10 13:03:18 head dhcpd[429]: DHCPREQUEST for 192.168.1.10 (192.168.1.1) from d8:3a:dd:fb:33:63 via eth0
Mar 10 13:03:18 head dhcpd[429]: DHCPACK on 192.168.1.10 to d8:3a:dd:fb:33:63 via eth0
Mar 10 13:03:19 head in.tftpd[651]: Client ::ffff:192.168.1.10 finished start4.elf
Mar 10 13:03:19 head in.tftpd[652]: Client ::ffff:192.168.1.10 finished fixup4.dat
Mar 10 13:03:20 head in.tftpd[667]: Client ::ffff:192.168.1.10 finished bcm2711-rpi-4-b.dtb
Mar 10 13:03:20 head in.tftpd[669]: Client ::ffff:192.168.1.10 finished overlays/overlay_map.dtb
Mar 10 13:03:20 head in.tftpd[672]: Client ::ffff:192.168.1.10 finished cmdline.txt
Mar 10 13:03:25 head in.tftpd[675]: Client ::ffff:192.168.1.10 finished kernel8.img
```

Configure nfs server on head node

Install nfs server

```
head$ dnf install -y nfs-utils
```

Enable and start the NFS server:

```
head$ systemctl enable --now nfs-server rpcbind
```

Verify it's running:

```
head$ systemctl status nfs-server
```

Configure the NFS Exports File

```
head$ vi /etc/exports
...
/image/pi4  192.168.1.0/24(ro,sync,no_root_squash,no_subtree_check)
/home       192.168.1.0/24(rw,sync,no_root_squash,no_subtree_check)
```

Apply the NFS Configuration

```
head$ exportfs -rv
head$ exportfs -v
```

Open Firewall for NFS

```
head$ firewall-cmd --add-service=nfs --permanent
head$ firewall-cmd --add-service=rpc-bind --permanent
head$ firewall-cmd --add-service=mountd --permanent
head$ firewall-cmd --reload
```

Test NFS on the Head Node

```
head$ mkdir -p /mnt/test-nfs
head$ mount -t nfs 192.168.1.1:/image/pi4 /mnt/test-nfs
head$ df -h
```

You should see /image/pi4 mounted via NFS.

```
head$ umount /mnt/test-nfs
head$ rm -r /mnt/test-nfs
```

Install the NFS Client in the Compute Node Image

```
chroot$ dnf install -y nfs-utils
```

Reboot the compute node.

Final tweaks and compute node boot up

Configure the mtab in the image by doing:

```
chroot$ ln -sf /proc/self/mounts /etc/mtab
```

Disable the /etc/mtab creation rule. Check the file containing mtab and comment it out.

```
chroot$ grep -r "/etc/mtab" /usr/lib/tmpfiles.d/
...
#L+ /etc/mtab    -    -    -    -   ../proc/self/mounts
```

Disable rpcbind in the Compute Node Image

```
chroot$ systemctl disable rpcbind
```

This ensures rpcbind won't start on boot.

Remove the Hostname from the Compute Node Image

```
chroot$ rm -f /etc/hostname
```

Make sure NFS is running on the head node:

```
head$ systemctl restart nfs-server
head$ exportfs -vsy
```

Reboot the compute node and monitor logs on the head node:

```
head$ journalctl -f
```

If still stuck then try to remove all rpcbind files

```
chroot$ rm /usr/lib/systemd/system/rpcbind.*
```

Make sure to comment out any rpcbind file in /usr/lib/tmpfiles.d/ because it might try to start rpcbind on boot

```
chroot$ vi /usr/lib/tmpfiles.d/rpcbind.conf
...
#Type Path         Mode  UID  GID Age Argument
#D     /run/rpcbind 0700  rpc  rpc  -  -
```

Finally check /srv/tftp/cmdline.txt on the head node

```
head$ vi /srv/tftp/cmdline.txt
console=tty0 ip=dhcp root=/dev/nfs ro
nfsroot=192.168.1.1:/image/pi4,vers=3,nolock panic=60 ipv6.disable=1
rootwait systemd.gpt_auto=no
```

Now try to boot compute node and you should see login screen. CONGRATULATIONS!!

Configure the fstab file and the volatile directories in the image

The root filesystem in the compute nodes is mounted read-only, so we need to provide RAM partitions for every directory where the compute node's OS needs to write. Note that the contents of the RAM mounts will disappear once the compute node is rebooted so things like, for instance, log files, will not survive a reboot. First set up the temporary filesystems in the fstab file, so the compute node mounts them automatically on boot-up:

```
#### /image/pi4/etc/fstab
# <file system> <mount point>   <type>  <options>       <dump>  <pass>
192.168.1.1:/image/pi4    /            nfs     tcp,nolock,ro,v4   1       1
proc                   /proc             proc    defaults           0       0
none                   /tmp              tmpfs   defaults           0       0
none                   /media            tmpfs   defaults           0       0
none                   /var              tmpfs   defaults           0       0
none                   /run              tmpfs   defaults           0       0
192.168.1.1:/home      /home             nfs     tcp,nolock,rw,v4   0       0
```

Then configure systemd in the image to create some directories on start-up. Even though /var is mounted as tmpfs, various services running on the compute nodes will complain if these directories do not exist:

```
#### /image/pi4/etc/tmpfiles.d/vardirs.conf
d /var/backups       0755 root  root -
d /var/cache         0755 root  root -
d /var/lib           0755 root  root -
d /var/local         4755 root  root -
d /var/log           0755 root  root -
d /var/mail          4755 root  root -
d /var/opt           0755 root  root -
d /var/spool         0755 root  root -
d /var/spool/rsyslog 0755 root  root -
d /var/tmp           1777 root  root -
```

Configure SELinux if stuck in login loop on compute node

```
head$ vi /image/pi4/etc/selinux/config
...
SELINUX=permissive
```

Setup SSH on Compute Node

```
head$ ssh-keygen
head$ cp -r /root/.ssh/ /image/pi4/root/
head$ cat /root/.ssh/id_rsa.pub >> /root/.ssh/authorized_keys
head$ cp /root/.ssh/authorized_keys /image/pi4/root/.ssh/
```

In the compute node check the status of OpenSSH

```
compute$ journalctl -xeu sshd.service
```

If you see warning regarding file permissions too open then fix it through chroot because compute node is read only:

```
chroot$ chmod 600 /etc/ssh/ssh_host_*
```

Reboot the compute node and ssh should work now.

Fixing systemctl failed services

Run the following command and check the failed service if any.

```
compute$ systemctl | grep fail
```

Most of the services might be failing because you don’t have systemd user for that service. This is typical Rocky linux workflow.

For systemd-networkd → create systemd-networkd user and group with no-login
For auditd → create /etc/log/audit file using etc/fstab
For gssproxy → read only file system is not allowing compute to write systemd files, so just create one manually in /image/pi4

Time synchronization

Install chronyd on head node and compute node

```
head$ dnf install chrony
head$ systemctl enable -now chronyd
head$ vi /etc/chony.conf
...
```

```
# Use public servers from the pool.ntp.org project.
# Please consider joining the pool (https://www.pool.ntp.org/join.html).
pool 2.rocky.pool.ntp.org iburst

# Use NTP servers from DHCP.
sourcedir /run/chrony-dhcp

# Record the rate at which the system clock gains/losses time.
driftfile /var/lib/chrony/drift

# Allow the system clock to be stepped in the first three updates
# if its offset is larger than 1 second.
makestep 1.0 3

# Enable kernel synchronization of the real-time clock (RTC).
rtcsync

# Enable hardware timestamping on all interfaces that support it.
#hwtimestamp *

# Increase the minimum number of selectable sources required to adjust
# the system clock.
#minsources 2

# Allow NTP client access from local network.
allow 192.168.1.0/24

# Serve time even if not synchronized to a time source.
local stratum 10

# Require authentication (nts or key option) for all NTP sources.
#authselectmode require

# Specify file containing keys for NTP authentication.
keyfile /etc/chrony.keys

# Save NTS keys and cookies.
ntsdumpdir /var/lib/chrony

# Insert/delete leap seconds by slewing instead of stepping.
#leapsecmode slew

# Get TAI-UTC offset and leap seconds from the system tz database.
```

```
leapsectz right/UTC

# Specify directory for log files.
logdir /var/log/chrony

# Select which information is logged.
#log measurements statistics tracking
```

Now setup chrony on compute node via chroot env.

```
chroot$ dnf install chrony
chroot$ vi /etc/chrony.conf
...
server 192.168.1.1 iburst

sourcedir /run/chrony-dhcp
driftfile /var/lib/chrony/drift
makestep 1.0 3
rtcsync
cmdport 0
logdir /var/log/chrony

chroot$ systemctl enable -now chronyd
```

Enable port 123 because the firewall might block it.

```
head$ sudo firewall-cmd --permanent --add-port=123/udp
head$ sudo firewall-cmd --permanent --add-port=123/tcp
head$ sudo firewall-cmd --reload

head$ systemctl restart chronyd
compute$ systemctl restart chronyd
```

Installing Slurm

Need to install Munge before Slurm
Slurm and munge groups and users must be synchronized in whole cluster
Don’t worry about avahi, timesync or other users if they differ in head node and compute nodes

Install Munge

```
head$ dnf install munge
```

it will install munge and munge-libs
it will create a user which is very important and must be synchronized among all nodes:
munge:x:991:990:Runs Uid N Gid Emporium:/run/munge:/sbin/nologin

Generate munge.key

```
head$ sudo /usr/sbin/create-munge-key
```

make sure its permission is 400 and user should be munge

Start munge

```
head$ systemctl enable --now munge
```

Check munge status

To check that munge is working on the head node, generate a credential on stdout

```
head$ munge -n
```

Also, check if a credential can be locally decoded:

```
head$ munge -n | unmunge
```

and you can also run a quick benchmark with:

```
head$ remunge
```

Install Slurm on head node

```
head$ dnf install slurm slurm-devel slurm-libs slurm-slurmctld slurm-slurmd
slurm-slurmdbd
```

Create slurm user+group

Make sure you choose uid and groupid which is not used before

```
head$ sudo groupadd -g 5000 slurm
head$ sudo useradd -u 5000 -g slurm -m -s /bin/bash slurm
```

Transfer these users to compute node using chroot

```
grep munge /etc/passwd >> /image/pi4/etc/passwd
grep slurm /etc/passwd >> /image/pi4/etc/passwd

grep munge /etc/group >> /image/pi4/etc/group
grep slurm /etc/group >> /image/pi4/etc/group

grep munge /etc/shadow >> /image/pi4/etc/shadow
grep slurm /etc/shadow >> /image/pi4/etc/shadow
```

Install slurm on compute node

```
chroot$ dnf install munge
```

Set up tmpfiles for MUNGE in image

```
head$ echo "d /var/log/munge 0700 munge munge -" >>
/image/pi4/etc/tmpfiles.d/vardirs.conf
head$ echo "d /var/lib/munge 0700 munge munge -" >>
/image/pi4/etc/tmpfiles.d/vardirs.conf
```

Copy the munge.key file from the head node to the image:

```
head$ cp /etc/munge/munge.key /image/pi4/etc/munge/
```

and change the permissions of the munge key in the image:

```
chroot$ chown munge.munge /etc/munge/munge.key
chroot$ chmod 400 /etc/munge/munge.key
```

Boot up (or reboot) the compute node and check that munge is working by decoding a credential remotely. (You can also restart munge without rebooting with `systemctl restart munge` if the compute node is already up.):

```
head$ munge -n | ssh n1 unmunge
```

Slurm conf file

```
[root@head /]$ cat /opt/slurm/slurm.conf
# slurm.conf file generated by configurator.html.
# Put this file on all nodes of your cluster.
# See the slurm.conf man page for more information.
#
ClusterName=WulverXPi
#SlurmctldHost=head
ControlMachine=head
ControlAddr=192.168.1.1
#SlurmctldHost=
#
#DisableRootJobs=NO
#EnforcePartLimits=NO
#Epilog=
#EpilogSlurmctld=
#FirstJobId=1
#MaxJobId=67043328
#GresTypes=
#GroupUpdateForce=0
#GroupUpdateTime=600
#JobFileAppend=0
#JobRequeue=1
#JobSubmitPlugins=lua
#KillOnBadExit=0
```

```
#LaunchType=launch/slurm
#Licenses=foo*4,bar
#MailProg=/bin/mail
#MaxJobCount=10000
#MaxStepCount=40000
#MaxTasksPerNode=512
MpiDefault=pmix
#MpiParams=ports=#-#
#PluginDir=
#PlugStackConfig=
#PrivateData=jobs
ProctrackType=proctrack/pgid
#Prolog=
#PrologFlags=
#PrologSlurmctld=
#PropagatePrioProcess=0
#PropagateResourceLimits=
#PropagateResourceLimitsExcept=
#RebootProgram=
ReturnToService=1
SlurmctldPidFile=/var/run/slurm/slurmctld.pid
SlurmctldPort=6817
SlurmdPidFile=/var/run/slurm/slurmd.pid
SlurmdPort=6818
SlurmdSpoolDir=/var/spool/slurm/d
SlurmUser=root
SlurmctldParameters=slurmd_port_range=30000-40000
#SlurmdUser=root
#SrunEpilog=
#SrunProlog=
StateSaveLocation=/var/spool/slurm/ctld
SwitchType=switch/none
#TaskEpilog=
TaskPlugin=task/none
#TaskPluginParam=PortRange=30000-40000
#TaskProlog=
#TopologyPlugin=topology/tree
#TmpFS=/tmp
#TrackWCKey=no
#TreeWidth=
#UnkillableStepProgram=
#UsePAM=0
#
```

```
#
# TIMERS
#BatchStartTimeout=10
#CompleteWait=0
#EpilogMsgTime=2000
#GetEnvTimeout=2
#HealthCheckInterval=0
#HealthCheckProgram=
InactiveLimit=0
KillWait=30
#MessageTimeout=10
#ResvOverRun=0
MinJobAge=300
#OverTimeLimit=0
SlurmctldTimeout=120
SlurmdTimeout=300
#UnkillableStepTimeout=60
#VSizeFactor=0
Waittime=0
#
#
# SCHEDULING
#DefMemPerCPU=0
#MaxMemPerCPU=0
#SchedulerTimeSlice=30
SchedulerType=sched/backfill
SelectType=select/linear
#
#
# JOB PRIORITY
#PriorityFlags=
#PriorityType=priority/basic
#PriorityDecayHalfLife=
#PriorityCalcPeriod=
#PriorityFavorSmall=
#PriorityMaxAge=
#PriorityUsageResetPeriod=
#PriorityWeightAge=
#PriorityWeightFairshare=
#PriorityWeightJobSize=
#PriorityWeightPartition=
#PriorityWeightQOS=
#
```

```
#
# LOGGING AND ACCOUNTING
#AccountingStorageEnforce=0
#AccountingStorageHost=
#AccountingStoragePass=
#AccountingStoragePort=
AccountingStorageType=accounting_storage/none
#AccountingStorageUser=
AccountingStoreFlags=job_comment
#JobCompHost=
#JobCompLoc=
#JobCompPass=
#JobCompPort=
JobCompType=jobcomp/none
#JobCompUser=
#JobContainerType=job_container/none
JobAcctGatherFrequency=30
JobAcctGatherType=jobacct_gather/none
SlurmctldDebug=info
SlurmctldLogFile=/var/log/slurm/slurmctld.log
SlurmdDebug=info
SlurmdLogFile=/var/log/slurm/slurmd.log
#SlurmSchedLogFile=
#SlurmSchedLogLevel=
#DebugFlags=
#
#
# POWER SAVE SUPPORT FOR IDLE NODES (optional)
#SuspendProgram=
#ResumeProgram=
#SuspendTimeout=
#ResumeTimeout=
#ResumeRate=
#SuspendExcNodes=
#SuspendExcParts=
#SuspendRate=
#SuspendTime=
#
#
# COMPUTE NODES
#NodeName=head CPUs=4 Boards=1 SocketsPerBoard=1 CoresPerSocket=4
ThreadsPerCore=1 RealMemory=3793 State=UNKNOWN
NodeName=n1 CPUs=4 Boards=1 SocketsPerBoard=1 CoresPerSocket=4
```

```
ThreadsPerCore=1 RealMemory=3794 State=UNKNOWN
NodeName=n2 CPUs=4 Boards=1 SocketsPerBoard=1 CoresPerSocket=4
ThreadsPerCore=1 RealMemory=3794 State=UNKNOWN
PartitionName=debug Nodes=ALL Default=YES MaxTime=INFINITE State=UP
```

Enable and start Slurm daemons

```
head$ systemd start slurmctld
compute$ systemd start slurmd
```

Create a backup

Warning!!

Always use UUID to refer devices in /etc/fstab. Don’t use /dev/sda or /dev/sdb because these are dynamically assigned and might mess up booting process.

Also check /boot/cmdline.txt and blkid for any repeating PARTUUID.

Create a backup for safety of main head node flash drive:

```
head$ sudo dd if=/dev/sdb bs=4M status=progress | gzip >
/project/flash_backup_$(date +%F).img.gz
```

—--CLUSTER BACKUP CREATED UPTO THIS POINT—----

Filesystem and storage plan

##
SSD on head node:
/project for shared storage
/softwares for installing softwares (Spack+Lmod)

SD cards on compute nodes:
/scratch for each node's local storage

Need mounting scripts to run on boot for mounting sd cards on compute nodes. Is there a better way? Using UUID?
Put it in /etc/fstab? But it is shared
Create a systemd service? Yes [Kevin suggested on May 22 and also best practice in modern linux]

/software mounted on each compute node read only (POC as of now, will be automated later)

Solution:
Partition SSD into /software ~350GB and /project ~600GB
Make sure to note down UUID, PartUID and put label
Then mount these to software and project by using UUID because /sda or /sdb keeps changing.
Don't directly put mount configurations in fstab otherwise the system will get stuck on boot if ssd is not attached. Try conditional mounts.
Since we will be sharing these mounts to compute nodes via nfs, so start nfs only after mount is successful. Try conditional nfs as well to solve this issue.
##

Mounting /software and /project from ssd

Put this in /etc/fstab to mount on boot safely.

```
[root@head ~]# cat /etc/fstab
UUID=05fa8335-82c7-4c1d-b57a-5b6565c00cc4  / ext4    defaults,noatime 0 0
UUID=ABC3-EED9  /boot vfat    defaults,noatime 0 0
UUID=e92e62d3-b879-414d-97cb-47904a256f15  swap swap    defaults,noatime 0
0

/dev /image/pi4/dev none defaults,rbind,rslave 0 0
/proc /image/pi4/proc proc defaults 0 0
/sys /image/pi4/sys none defaults,rbind,rslave 0 0
/tmp /image/pi4/tmp none defaults,rbind 0 0
```

```
############################################################################
#
# 29May,2025 /LS
# Mount SSD partitions attached to head node safely or
# skip if not attached
############################################################################
#
UUID=0a72f6d2-f943-486a-a31d-7682727882ed  /project   ext4
defaults,nofail,x-systemd.device-timeout=10  0  0
UUID=4aac8b8f-dce8-42c4-8060-90db82acd459  /software  ext4
defaults,nofail,x-systemd.device-timeout=10  0  0
```

Export to compute nodes

Mount /software and /project from SSD — done via fstab with nofail.
Export them via NFS only if mounted.
NFS should still serve other exports (like rootfs for compute nodes) even if /software or /project is missing.

Create a new systemd unit to conditionally export /software and /project

Create script inside /usr/local/sbin and make it executable

```
[root@head /]# cat /usr/local/sbin/export_optional_nfs.sh
#!/bin/bash

############################################################################
########
# 29May,2025 /LS
# Checks if software and project partitions from ssd are successfully
# mounted and only then exports them to compute node via nfs
############################################################################
########
MOUNTED=0

if mountpoint -q /software; then
  echo "/software 192.168.1.0/24(ro,sync,no_subtree_check,no_root_squash)"
> /etc/exports.d/software.exports
  MOUNTED=1
else
  rm -f /etc/exports.d/software.exports
```

```
fi

if mountpoint -q /project; then
  echo "/project 192.168.1.0/24(rw,sync,no_subtree_check,no_root_squash)" >
/etc/exports.d/project.exports
  MOUNTED=1
else
  rm -f /etc/exports.d/project.exports
fi

# Reload exports if anything was added
if [ $MOUNTED -eq 1 ]; then
  exportfs -ra
fi

[root@head /]# chmod +x /usr/local/sbin/export_optional_nfs.sh
```

Create a systemd service for this script. It should be oneshot, with timeout and proper configurations to avoid boot hangs.

```
[root@head /]# cat /etc/systemd/system/export-optional-nfs.service
#################################
# 29May,2025 /LS
#################################

[Unit]
Description=Conditionally export /software and /project over NFS if mounted
After=local-fs.target nfs-server.service
Requires=nfs-server.service

[Service]
Type=oneshot
ExecStart=/usr/local/sbin/export_optional_nfs.sh
RemainAfterExit=yes
SuccessExitStatus=0
TimeoutSec=10

[Install]
WantedBy=multi-user.target
```

Also put these mount points in chroot's fstab. Make sure to put proper options to avoid boot hangs

```
[root@head /]# cat /image/pi4/etc/fstab
# <file system> <mount point>   <type>  <options>       <dump>  <pass>
192.168.1.1:/image/pi4    /             nfs     tcp,nolock,ro,v4   1
1
proc                   /proc            proc    defaults           0
0
none                   /tmp             tmpfs   defaults           0
0
none                   /media           tmpfs   defaults           0
0
none                   /var             tmpfs   defaults           0
0
none                   /run             tmpfs   defaults           0
0
192.168.1.1:/home      /home            nfs     tcp,nolock,rw,v4   0
0

#############################################
# 29May,2025 LS
# Mount ssd partitions from head
#############################################
192.168.1.1:/software    /software    nfs
tcp,nolock,ro,v4,nofail,_netdev,x-systemd.automount    0    0
192.168.1.1:/project     /project     nfs
tcp,nolock,rw,v4,nofail,_netdev,x-systemd.automount    0    0
```

Bonus: Create a shutdown compute nodes script

Create a simple script to shutdown compute nodes because you don't shutdown head before compute nodes.

```
[root@head /]# cat /usr/local/sbin/shutdown_compute_nodes.sh
#############################################
# 29May,2025 LS
# Run this to shutdown compute nodes
# before head
#############################################
for host in n1 n2 n3; do
    ssh $host "shutdown -h now" &
done
wait
```

A utility script to check nodes status UP or DOWN.

We are shutting down and rebooting nodes multiple times to test our changes and its easy to get confused which node is up. So it will be very convenient to create a script to show node status.

```
[root@head /]# cat usr/local/bin/checknodes
#!/bin/bash

##############################################################
# 2 June, 2025 /LS
# Checks if a compute node is booted or shutdown.
##############################################################


# Add new nodes in this list
nodes=(n1 n2 n3)

# Create arrays for status
up_nodes=()
down_nodes=()

# Check each node
for host in "${nodes[@]}"; do
    if ping -c 1 -W 1 $host &>/dev/null; then
        up_nodes+=("$host")
    else
        down_nodes+=("$host")
    fi
done

# Join arrays into comma-separated strings
up_list=$(IFS=, ; echo "${up_nodes[*]}")
down_list=$(IFS=, ; echo "${down_nodes[*]}")

# Print the summary table
echo "| UP       | DOWN         |"
echo "|----------|--------------|"
printf "| %-8s | %-12s |\n" "$up_list" "$down_list"
```

Automatic scratch storage mount on compute nodes

Each compute node has a local sd card storage mounted on /scratch.
To automatically mount for each node, we have to create systemd service.

Please note that chroot image is share among all nodes, therefore putting mounting points in /etc/fstab will not help. So its better to create a script which detects mounts based on hostname of node.

```
chroot# cat /usr/local/sbin/mount_scratch.sh
#!/bin/bash

################################################################
# 2 June, 2025 /LS
# This script automatically mounts local
# storage (sd cards, usb) to compute nodes.
# It is paired with systemd service to run on boot.
################################################################
LOG_TAG="mount_scratch"
LOG_FILE="/var/log/mount_scratch.log"

HOST=$(hostname)
SCRATCH_DIR="/scratch"

log() {
    echo "[$(date +'%F %T')] [$LOG_TAG] $1" | tee -a "$LOG_FILE"
}

log "Starting scratch mount for $HOST"

# Map hostnames to UUIDs
case "$HOST" in
    n1)
        log "sd card slot is broken for n1"
        ;;
    n2)
        UUID="ac6b6864-3f9d-42f7-8a8d-cd2b0520f48b"
        ;;
    n3)
        UUID="668b0ddd-3a29-4fa9-bcd0-825e82b244c7"
        ;;
    *)
        log "[mount-scratch] No UUID configured for host $HOST"
        exit 0
        ;;
esac

# Resolve device from UUID
DEVICE=$(blkid -U "$UUID")
if [ -z "$DEVICE" ]; then
    log "[mount-scratch] No device found with UUID=$UUID for $HOST"
    exit 1
fi

# Mount if not already mounted
if ! mountpoint -q "$SCRATCH_DIR"; then
    log "[mount-scratch] Mounting $DEVICE to $SCRATCH_DIR"
    mount "$DEVICE" "$SCRATCH_DIR"
else
    log "[mount-scratch] $SCRATCH_DIR already mounted"
fi
```

Then create a systemd service

```
chroot# cat /image/pi4/etc/systemd/system/mount_scratch.service
[Unit]
Description=Mount local scratch storage
After=local-fs.target
ConditionPathExists=/usr/local/sbin/mount_scratch.sh

[Service]
Type=oneshot
ExecStart=/usr/local/sbin/mount_scratch.sh
RemainAfterExit=true
StandardOutput=journal
StandardError=journal

[Install]
WantedBy=multi-user.target
```

Spack + Lmod

######################## INITIALIZATION ##########################
source /software…/spack/…/setup-env.sh
spack load lmod
source $(spack location -i lmod)../init/bash
module –version
module use /software/modules/…/core
Module avail
Module install *package*
spack module lmod refresh --delete-tree -y
Module avail
##

Mounted /software as read and write to compute nodes.
Can't run source /software…/spack/…/setup-env.sh as root because it tries to create .spack/ in /root which is read only for compute nodes.
So run using user1. But takes huge time (slow nfs).
This might be problematic for slurm, so its better to install it from source and adjust install location prefix for isolation.

Slurm installation from source

Create a backup of slurm.conf from dnf installed slurm.

```
head$ mkdir -p /var/backups/slurm
head$ cp etc/slurm/slurm.conf /var/backups/slurm/slurm.conf.$(date +%F)
```

Now clean and remove dnf installed slurm.

```
head$ dnf remove slurm\*
head$ find / -name '*slurm*'
head$ rm -rf /var/spool/slurm
head$ rm -rf /var/log/slurm
head$ rm -rf /etc/slurm
```

Create a Slurm user and group with appropriate UID and GID on the head node. Will do the same on compute nodes as well.

```
head$ groupadd -g 450 slurm
head$ useradd -u 450 -g 450 -d /var/lib/slurm -s /bin/bash slurm
```

Create slurm log and process directories and change ownership to slurm

```
head$ mkdir -p /var/lib/slurm /var/spool/slurm /var/log/slurm
head$ chown -R slurm:slurm /var/spool/slurm /var/log/slurm /var/lib/slurm
```

Repeat these steps on chroot env as well.

Install munge on head and chroot (compute nodes)

Follow office guide: https://github.com/dun/munge/wiki/Installation-Guide

Install munge from source because dnf installed munge doesn't have dev lib tools.
Create installation dir for slurm /opt/slurm/25.05.0 and /opt/slurm/25.05.0/etc

Remove dnf installed munge

```
head$ dnf remove munge\*
head$ find / -name '*munge*'
```

Install dependency for munge

```
dnf install openssl openssl-devel
```

```
mkdir /opt/munge
cd /opt/munge
wget
https://github.com/dun/munge/releases/download/munge-0.5.16/munge-0.5.16.ta
r.xz
```

Open the tar and install into specific location using configure

```
./configure     --prefix=/opt/munge/0.5.16
--sysconfdir=/opt/munge/0.5.16/etc     --localstatedir=/var
--runstatedir=/run
make
make install
```

Add munge commands to path of every user

```
[user1@head ~]$ cat /etc/profile.d/munge.sh
###########################################################################
#####################
# 10 June, 2025 /LS
# Added munge commands to PATH of every user
# It is necessary to be accessible for all user because
# when a user submits a job or interacts with Slurm,
# their processes use the munge client library (which munge and unmunge
use)
# to create and verify credentials.
###########################################################################
#####################
export PATH=/opt/munge/0.5.16/bin:/opt/munge/0.5.16/sbin:$PATH
[user1@head ~]$
```

Make sure munge user and group exists.
Change ownership of /var/lib/munge, /var/log/munge to munge:munge

Create munge.service symlink to point towards your munge's folder

```
head$ sudo ln -s /opt/munge/0.5.16/lib/systemd/system/munge.service
/etc/systemd/system/munge.service
head$ sudo systemctl daemon-reload
head$ sudo systemctl enable munge
head$ sudo systemctl start munge
head$ sudo systemctl status munge
```

Do these steps inside chroot env as well!!

Make sure to put munge ephemeral files configuration in /image/etc/tmpfile.d/vardirs.conf

```
chroot# echo "d /var/log/munge 0700 munge munge -" >>
/etc/tmpfiles.d/vardirs.conf
chroot# echo "d /var/lib/munge 0700 munge munge -" >>
/etc/tmpfiles.d/vardirs.conf
```

Install slurm on head and chroot (compute nodes)

https://slurm.schedmd.com/quickstart_admin.html

Create slurm user and group

```
head$ getent passwd slurm
slurm:x:450:450::/var/lib/slurm:/bin/bash
```

Install Slurm related softwares. Most of them are already installed during munge except database. Install mariadb, its good for rocky linux and comes with development libraries.

```
head$ dnf install mariadb-connector-c-devel
```

Some other dependencies

```
head$ dnf install numactl-devel pam-devel
```

Please note that we are just installing libraries for slurm installation, will setup main db server on head node later.

Create dir and download latest slurm

```
head$ mkdir -p /opt/slurm
head$ wget https://download.schedmd.com/slurm/slurm-25.05.0.tar.bz2
head$ mkdir -p /opt/slurm/25.05.0/etc
```

Then open tar slurm-25.05.0 and go to its dir for installation.

```
head$ sudo ./configure \
    --prefix=/opt/slurm/25.05.0 \
    --sysconfdir=/opt/slurm/25.05.0/etc \
    --localstatedir=/var \
    --runstatedir=/run \
    --enable-debug \
    --enable-pam \
    --enable-threads \
    --with-munge=/opt/munge/0.5.16
head$ sudo make install
```

Dynamically link slurm libraries using ldconfig

```
head$ echo "/opt/slurm/25.05.0/lib" | sudo tee /etc/ld.so.conf.d/slurm.conf
head$ sudo ldconfig
head$ sudo ldconfig -p | grep slurm
```

Create ephemeral directories and change ownership to slurm user.

```
head$ chown -R slurm:slurm /opt/slurm/25.05.0/etc
head$ mkdir -p /var/log/slurm /var/lib/slurm
head$ chown slurm:slurm /var/log/slurm /var/lib/slurm
head$ chmod 755 /var/log/slurm /var/lib/slurm
```

Create slurm.conf file for your version from slurm's configurator.html
Put it inside your slurm's etc/ directory
Copy slurm's provided systemd services to /etc/systemd

```
head$ vi opt/slurm/25.05.0/etc/slurm.conf
head$ cp opt/slurm/slurm-25.05.0/etc/slurmctld.service /etc/systemd/system/
```

Note: slurmctld.service is for head node and slurmd.service is for compute node unless you want to use head node as compute node.

Do all these steps in chroot as well.

Also make sure to add slurm ephemeral directories in tmpfiles.d for chroot. Since compute nodes are diskless booted so we need to make sure that these directories are automatically created during every boot.

```
chroot$ cat /etc/tmpfiles.d/vardirs.conf
d /var/backups       0755 root  root -
d /var/cache         0755 root  root -
d /var/lib           0755 root  root -
d /var/local         0755 root  root -
d /var/log           0755 root  root -
d /var/mail          0755 root  root -
d /var/opt           0755 root  root -
d /var/spool         0755 root  root -
d /var/spool/rsyslog 0755 root  root -
d /var/tmp           1777 root  root -
d /var/log/audit     0755 root  root -

# For munge tmp files during boot
d /var/log/munge 0700 munge munge -
d /var/lib/munge 0700 munge munge -

#################################################
# 16 June, 2025 /LS
# These directories will be created during
# boot and will be used by slurm.
#################################################
d /var/log/slurm 0755 slurm slurm -
d /var/lib/slurm 0755 slurm slurm -
d /var/lib/slurm/slurmctld 0700 slurm slurm - # Slurmctld state directory
d /var/lib/slurm/slurmd    0700 slurm slurm - # Slurmd spool directory
```

Put slurm's created slurmd.service file to systemd

```
chroot$ cp opt/slurm/slurm-25.05.0/etc/slurmd.service /etc/systemd/system/
```

Now start slurmctl on head node and slurmd on compute nodes

```
head$ systemctl enable slurmctld
head$ systemctl start slurmctl

compute$ systemctl start slurmd
```

If you see any cgroup errors, then you need to reconfigure slurm from start and install some additional packages like systemd-devel, libgudev-devel.

Serious problem!!

- Built Slurm 25.05.0 from source with `--with-munge`, `--enable-debug`, and default plugin support.
- Set `ProctrackType=proctrack/pgid` and `TaskPlugin=task/none` to bypass cgroup errors initially.
- Tried to enable `task/cgroup` + `proctrack/cgroup` using `cgroup_v2.so`.
- Ran into missing `dbus.h` errors — fixed it by installing `dbus-devel`.
- Recompiled Slurm's `cgroup_v2.so` manually using proper `CPPFLAGS` and `LDFLAGS` to link against `libdbus-1`.
- Successfully installed `cgroup_v2.so`, `task_cgroup.so`, and `proctrack_cgroup.so`.
- Updated `slurm.conf` and `cgroup.conf` for proper cgroup v2 usage.

- Modified PXE `cmdline.txt` to include:
  cgroup_enable=memory systemd.unified_cgroup_hierarchy=1

- Slurm plugins loaded successfully.

- PXE-booted nodes reached `slurmd` execution and attempted to use `cgroup_v2`.

- All configuration and userland parts (Slurm, systemd, chroot, NFS root) were functioning as expected.

- `dmesg` showed: `cgroup: Disabling memory control group subsystem`

- `/sys/fs/cgroup/cgroup.controllers` was missing `memory` — confirming no memory controller support.

- Kernel was missing `CONFIG_MEMCG` (memory cgroup support).

- Could not proceed with Slurm's task/cgroup because memory controller is mandatory for it in cgroup v2.

- That's why dnf installed slurm kept screaming memory not enabled!! memory not enablled!! ….. memory not enablled …. memory not enablled

.

- Rocky 9 + cgroupV2 + systemd → Effin tightly coupled

- On the head node which has full fledged kernel, you can see memory config is enabled

```
[root@head /]# grep CONFIG_MEMCG /boot/config-$(uname -r)
CONFIG_MEMCG=y
CONFIG_MEMCG_KMEM=y
```

- But on the compute nodes

```
n3:~# grep CONFIG_MEMCG /boot/config-$(uname -r)
grep: /boot/config-6.1.31-v8.1.el9.altarch: No such file or directory
```

###

TL;DR → PXE booted kernels don't have cgroup memory config, because authors thought people will just use them in small embedded systems and won't require mem config, so let's effin save space.

###

WORKAROUND SOLUTION: DISABLE CGROUP

Remove all cgroup plugins from slurm.conf

```
ProctrackType=proctrack/pgid
TaskPlugin=task/none
JobAcctGatherType=jobacct_gather/none
JobContainerType=job_container/none
UsePAM=no
```

Also delete or rename cgroup.conf file from etc/
Make sure the firewall is not blocking any ports on head and compute nodes.

—--CLUSTER BACKUP CREATED UPTO THIS POINT—----

LOGIN CREDENTIALS

User - $UCID@128.235.43.17
Password - *changeme*

Please DM for root credentials