HPC Tools

- HPC Tools
  - Step 1: Working with `pdsh`
    - Setting up the rest of the BMCs
  - Step 2: Powerman & Conman
    - Powerman
    - Conman
  - Step 3: Setting up Infiniband
    - Install necessary software & drivers on the master
    - Enable/Start services on the master
    - Install necessary software & drivers in the BOS image
    - Re-build the VNFS & setup warewulf IB
    - Re-start the nodes & verify IB
  - Step 4: Deploying Slurm
    - Setup software/services on the master
    - Setup software/services on the nodes
    - Map necessary configuration to the nodes
    - Create a VNFS and test it on one node
    - Remap our image to the rest of the nodes and reboot
  - Step 5: Program environments with `lmod`

Step 1: Working with `pdsh`

We're going to explore using the `pdsh` command (commonly pronounce "p-dish"). `pdsh` will give us an easy way to run commands on more than one node at once. It does this by `ssh`ing to the nodes, possibly in parallel and running a specified command. This is the default behavior, though in a moment we will learn another way it can be used.

Let's first look at the usage info for `pdsh`:

```
[root@master ~]# pdsh -h
Usage: pdsh [-options] command ...
-S                  return largest of remote command return values
-h                  output usage menu and quit
-V                  output version information and quit
-q                  list the option settings and quit
-b                  disable ^C status feature (batch mode)
-d                  enable extra debug information from ^C status
-l user             execute remote commands as user
-t seconds          set connect timeout (default is 10 sec)
-u seconds          set command timeout (no default)
-f n                use fanout of n nodes
-w host,host,...    set target node list on command line
-x host,host,...    set node exclusion list on command line
-R name             set rcmd module to name
-M name,...         select one or more misc modules to initialize first
```

```
-N                disable hostname: labels on output lines
-L                list info on all loaded modules and exit
available rcmd modules: ssh,exec (default: ssh)
```

We see that the `-w` option can be used to specify a list of hosts to run on. The `-f` option can be used to control the "fanout" of the command, i.e. how many nodes it runs on in parallel. Let's start simple:

```
[root@master ~]# pdsh -f1 -w n01,n02 hostname
n01: n01
n02: n02
```

This ran `hostname` on nodes `n01` and `n02`. By default, `pdsh` puts "`<host>:`" in front of the output for each host it runs on.

Specifying a comma-separated list of hosts could get very long, fortunately, `pdsh` supports range specifications:

```
[root@master ~]# pdsh -w n[01-03] hostname
n01: n01
n03: n03
n02: n02
```

Note that we let it run in parallel, so our answers came out of order (by default `-f` is 32).

This syntax is probably fine for our small cluster, but let's take it a step further. `pdsh` supports add-on modules. We're going to use a module that applies `gender` specifications through `libgender` to our nodes. To do this, we need an additional package:

```
[root@master ~]# yum -y install pdsh-mod-genders-ohpc
```

Now edit the file `/etc/genders` and put the following in it:

```
n01 first
n10 last
n[01-03] three
n07 lucky=true,seven
n[01-06],n[08-10] lucky=false
n[01-10] compute
```

The format of this file is:

```
<nodespec> attr[=value],...
```

If we look at `pdsh -h` we see we have new options. Let's try a couple of these:

```
[root@master ~]# pdsh -A hostname
n02: n02
n01: n01
n03: n03
... (the rest of the nodes)
```

We should see that we get a response from all of our nodes.

We can also specify based on attributes and attribute/value pairs:

```
[root@master ~]# pdsh -g first hostname
n01: n01
```

```
[root@master ~]# pdsh -g lucky=true hostname
n07: n07
```

```
[root@master ~]# pdsh -f1 -g lucky=false hostname
n01: n01
n02: n02
n03: n03
n04: n04
n05: n05
n06: n06
n08: n08
n09: n09
n10: n10
```

```
[root@master ~]# pdsh -f1 -A -x n07 hostname
n01: n01
n02: n02
n03: n03
n04: n04
n05: n05
n06: n06
n08: n08
n09: n09
n10: n10
```

Both of these last commands exclude `n07`.

Note that you may want `-f1` if you want to see a consistent, sequential list. It will run much slower though.

Setting up the rest of the BMCs

`pdsh` is pretty handy. Let's use it for something useful. Let's get the remaining BMC MAC addresses using `pdsh` to run `ipmitool` locally on the nodes (we made sure we installed it in the image earlier):

```
[root@master ~]# pdsh -f1 -w n[04-10] ipmitool lan print
n04: Set in Progress         : Set Complete
n04: Auth Type Support       : MD5
n04: Auth Type Enable        : Callback : MD5
n04:                         : User     : MD5
n04:                         : Operator : MD5
n04:                         : Admin    : MD5
n04:                         : OEM      :
n04: IP Address Source       : DHCP Address
n04: IP Address              : 172.16.0.101
n04: Subnet Mask             : 255.255.255.0
n04: MAC Address             : 18:66:da:68:3e:40
...(lots and lots of output)
```

We got our MAC addresses, but we got way too much output. Let's write a command to filter for just what we want.

```
[root@master ~]# pdsh -f1 -w n[04-10] ipmitool lan print | grep "MAC
Address"
n04: MAC Address             : 18:66:da:68:3e:40
n05: MAC Address             : 18:66:da:68:4b:14
n06: MAC Address             : 18:66:da:68:41:3a
...
```

Better, but let's grab ***just*** the MAC address, and let's store it in a file:

```
[root@master ~]# pdsh -f1 -w n[04-10] ipmitool lan print | grep "MAC
Address" | awk '{print $NF}' | tee /tmp/bmc-macs.txt
18:66:da:68:3e:40
18:66:da:68:4b:14
18:66:da:68:41:3a
...
```

We used `awk` to get the last space separated column (`NF` is a special variable that means "number of columns", `$NF` then means "value of the last column"). `tee` is a special command that both prints the output and outputs it to a file.

Here's an `awk` line that will generate the "host" lines we need for our `dhcpd.conf`:

```
[root@master ~]# awk '{printf "host n%02d-bmc { hardware ethernet %s;
fixed-address 172.16.0.1%02d; }\n", NR+3, $1, NR+3}' /tmp/bmc-macs.txt
...
host n08-bmc { hardware ethernet 18:66:da:68:4b:14; fixed-address
172.16.0.108; }
host n09-bmc { hardware ethernet 18:66:da:68:41:3a; fixed-address
172.16.0.109; }
host n10-bmc { hardware ethernet 18:66:da:68:41:3a; fixed-address
172.16.0.110; }
```

Copy/paste after the other entries in `/etc/warewulf/dhcpd-template.conf`.

We now need to regenerate the `dhcpd.conf` and restart `dhcpd`:

```
[root@master ~]# wwsh dhcp update
Rebuilding the DHCP configuration
Done.
```

```
[root@master ~]# systemctl restart dhcpd
```

We should see that our BMC get their IP address now. Let's force the issue by running the following commands on all of the BMCs:

```
[root@master ~]# pdsh -A ipmitool user set name 2 admin
[root@master ~]# pdsh -A ipmitool user set password 2 admin
[root@master ~]# pdsh -A ipmitool lan set 1 ipsrc dhcp
[root@master ~]# pdsh -A ipmitool chassis bootdev pxe options=persistent
[root@master ~]# pdsh -A ipmitool mc reset cold
```

These will:

1. make sure the username is "admin"
2. set the password to "admin"
3. set the BMC to use DHCP
4. make PXE the persistent first boot option
5. restart the BMC (won't restart the node itself)

Step 2: Powerman & Conman

Powerman

`ipmitool` is a powerful command, but it can get tedious entering all of the options and remembering all of the commands.

Let's set up a handy tool called, `powerman`, that we can use to control and monitor system power through IPMI in a more convenient way.

First, we need to install it:

```
[root@master ~]# yum -y install powerman
```

We are going to want to refer to the BMC by hostname, so let's add some entries to `/etc/hosts`. In `/etc/hosts`, before the line:

```
### ALL ENTRIES BELOW THIS LINE WILL BE OVERWRITTEN BY WAREWULF ###
```

Add:

```
# cluster_compute_nodes.bmc
172.16.0.101 bmc-n01 bmc-n01.localdomain
172.16.0.102 bmc-n02 bmc-n02.localdomain
172.16.0.103 bmc-n03 bmc-n03.localdomain
172.16.0.104 bmc-n04 bmc-n04.localdomain
172.16.0.105 bmc-n05 bmc-n05.localdomain
172.16.0.106 bmc-n06 bmc-n06.localdomain
172.16.0.107 bmc-n07 bmc-n07.localdomain
172.16.0.108 bmc-n08 bmc-n08.localdomain
172.16.0.109 bmc-n09 bmc-n09.localdomain
172.16.0.110 bmc-n10 bmc-n10.localdomain
```

The configuration for powerman lives under `/etc/powerman`:

```
[root@master ~]# ls /etc/powerman
apc7900.dev appro-gb2.dev cyclades-pm10.dev hpmpblade.dev ipmipower.dev
rancid-cisco-poe.dev
apc7900v3.dev appro-greenblade.dev cyclades-pm20.dev hpmpcell.dev
ipmipower-serial.dev raritan-px4316.dev
apc7920.dev bashfun.dev cyclades-pm42.dev hpmp.dev kvm.dev raritan-
px5523.dev
apc8941.dev baytech.dev cyclades-pm8.dev hpmpdome.dev kvm-ssh.dev
sentry_cdu.dev
apc.dev baytech-rpc18d-nc.dev dli4.dev ibmbladecenter.dev lom.dev
swpdu.dev
apcnew.dev baytech-rpc22.dev dli.dev icebox3.dev openbmc.dev vpc.dev
apcold.dev baytech-rpc28-nc.dev eaton-epdu-blue-switched.dev icebox.dev
phantom.dev wti.dev
apcpdu3.dev baytech-rpc3-nc.dev eaton-revelation-snmp.dev ics8064.dev
plmpower.dev wti-rps10.dev
apcpdu.dev baytech-snmp.dev hp3488.dev ilom.dev powerman.conf.example
apc-snmp.dev cb-7050.dev hpilo.dev ipmi.dev powerman.dev
```

There are a lot of files here that are used to define different types of devices we know how to interact with. The only one we'll be using is `ipmipower`. The only file we need to modify is `/etc/powerman/powerman.conf`. Edit `/etc/powerman/powerman.conf`, and set its contents to:

```
include "/etc/powerman/ipmipower.dev"

device "ipmi0" "ipmipower" "/usr/sbin/ipmipower -D lanplus -u admin -p
admin -h bmc-n[01-10] |&"

node "n[01-10]" "ipmi0" "bmc-n[01-10]"
```

Powerman relies on a special service to operate. This service needs a special directory setup that doesn't get setup automatically:

```
[root@master ~]# mkdir -p /var/run/powerman
[root@master ~]# chown daemon:daemon /var/run/powerman
```

Now enable and start the service:

```
[root@master ~]# systemctl enable powerman
[root@master ~]# systemctl start powerman
```

Check its status:

```
[root@master ~]# systemctl status powerman
● powerman.service - PowerMan
   Loaded: loaded (/usr/lib/systemd/system/powerman.service; disabled;
vendor preset: disabled)
   Active: active (running) since Sat 2019-06-08 16:22:21 MDT; 4s ago
  Process: 32375 ExecStart=/usr/sbin/powermand (code=exited,
status=0/SUCCESS)
 Main PID: 32377 (powermand)
   CGroup: /system.slice/powerman.service
           └─32377 /usr/sbin/powermand

Jun 08 16:22:21 master systemd[1]: Started PowerMan.
...
```

The command for using powerman is `powerman`, but we can use the shorter alias `pm`. To query the status of power on our nodes:

```
[root@master ~]# pm -q
on:  n[01-10]
off:
unknown
```

We should see that all 10 nodes are on.

To power cycle we use:

```
[root@master ~]# pm -c n01
```

To power off:

```
[root@master ~]# pm -0 n01
```

To power on:

```
[root@master ~]# pm -1 n01
```

Conman

Conman is a similar utility to powerman that can use IPMI SOL (and other devices) to provide remote console access. First, we need to install conman:

```
[root@master ~]# yum -y install conman-ohpc
```

(it may already be installed)

The configuration for conman is in `/etc/conman.conf`. Set its contents to:

```
SERVER keepalive=ON
SERVER logdir="/var/log/conman"
SERVER logfile="/var/log/conman.log"
SERVER loopback=ON
SERVER pidfile="/var/run/conman.pid"
SERVER resetcmd="/usr/bin/powerman -0 %N; sleep 5; /usr/bin/powerman -1
%N"
SERVER tcpwrappers=ON
#SERVER timestamp=1h
GLOBAL seropts="115200,8n1"
GLOBAL log="/var/log/conman/console.%N"
GLOBAL logopts="sanitize,timestamp"
```

```
CONSOLE name="n01" dev="ipmi:bmc-n01"
ipmiopts="U:admin,P:admin,W:solpayloadsize"
CONSOLE name="n02" dev="ipmi:bmc-n02"
ipmiopts="U:admin,P:admin,W:solpayloadsize"
CONSOLE name="n03" dev="ipmi:bmc-n03"
ipmiopts="U:admin,P:admin,W:solpayloadsize"
CONSOLE name="n04" dev="ipmi:bmc-n04"
ipmiopts="U:admin,P:admin,W:solpayloadsize"
CONSOLE name="n05" dev="ipmi:bmc-n05"
ipmiopts="U:admin,P:admin,W:solpayloadsize"
CONSOLE name="n06" dev="ipmi:bmc-n06"
ipmiopts="U:admin,P:admin,W:solpayloadsize"
CONSOLE name="n07" dev="ipmi:bmc-n07"
ipmiopts="U:admin,P:admin,W:solpayloadsize"
CONSOLE name="n08" dev="ipmi:bmc-n08"
ipmiopts="U:admin,P:admin,W:solpayloadsize"
CONSOLE name="n09" dev="ipmi:bmc-n09"
ipmiopts="U:admin,P:admin,W:solpayloadsize"
CONSOLE name="n10" dev="ipmi:bmc-n10"
ipmiopts="U:admin,P:admin,W:solpayloadsize"
```

Now we need to enable and start the `conman` service:

```
[root@master ~]# systemctl enable conman
Created symlink from /etc/systemd/system/multi-
user.target.wants/conman.service to
/usr/lib/systemd/system/conman.service.
[root@master ~]# systemctl start conman
```

We should now be able to get a serial console on one of our nodes. Note, you may need to hit `<enter>` to get the login prompt after connecting.

```
[root@master ~]# conman n01

<ConMan> Connection to console [n01] opened.

CentOS Linux 7 (Core)
Kernel 3.10.0-957.12.2.el7.x86_64 on an x86_64

n01 login:
```

To escape conman, you need to hit the escape sequence: `& - .` (i.e. ampersand-period).

Conman stores console logs under `/var/log/conman/console.<hostname>`. Try:

```
[root@master ~]# cat /var/log/conman/console.n01
<ConMan> Console [n01] log opened at 2019-06-08 16:50:20 MDT.

<ConMan> Console [n01] connected to <bmc-n01>.

<ConMan> Console [n01] joined by <root@localhost> on pts/0 at 06-08 16:50.
2019-06-08 16:50:40
2019-06-08 16:50:40 CentOS Linux 7 (Core)
2019-06-08 16:50:40 Kernel 3.10.0-957.12.2.el7.x86_64 on an x86_64
2019-06-08 16:50:40
2019-06-08 16:50:40 n01 login:
2019-06-08 16:50:40 CentOS Linux 7 (Core)
2019-06-08 16:50:40 Kernel 3.10.0-957.12.2.el7.x86_64 on an x86_64
2019-06-08 16:50:40
2019-06-08 16:50:40 n01 login:
<ConMan> Console [n01] departed by <root@localhost> on pts/0 at 06-08
16:50.
```

This can be a very useful feature since it will keep a log of, e.g. the boot console messages of each node.

Step 3: Setting up Infiniband

At this point, all of the hardware on our cluster is functional with the exception of our Infiniband fabric (through much of this guide we will refer to Infiniband as IB). There are six steps to getting the Infiniband fabric fully functional.

1. Install necessary software & drivers on the master
2. Enable/Start services on the master
3. Install necessary software & drivers in the BOS image
4. Re-build the VNFS & setup warewulf IB
5. Re-start the nodes & verify IB

We will go through each of these steps, while also learning some of the commands that help us diagnose the IB fabric.

Install necessary software & drivers on the master

We need to install a suite of software known as Mellanox OFED (OpenFabrics Enterprise Distribution). This is the suite of drivers and tools provided by the vendor that will allow us to enable and use the Mellanox IB cards/switch we have in our clusters.

We should already have a repository setup that contains the OFED software.

```
[root@master ~]# yum repolist
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
repo id     repo name       status
...
mlnx        Mellanox OFED   127
```

```
...
repolist: 26,044
```

We can see what it provides:

```
[root@master ~]# yum --disablerepo='*' --enablerepo=mlnx list available
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
Available Packages
ar_mgr.x86_64               1.0-0.42.g750eb1e.46101            mlnx
cc_mgr.x86_64               1.0-0.41.g750eb1e.46101
mlnx
dapl.x86_64                 2.1.10mlnx-OFED.3.4.2.1.0.46101    mlnx
dapl-devel.x86_64           2.1.10mlnx-OFED.3.4.2.1.0.46101    mlnx
dapl-devel-static.x86_64    2.1.10mlnx-OFED.3.4.2.1.0.46101    mlnx
...
```

Of particular interest to us are several packages named `mlnx-ofed-*`. These packages are "meta-packages" that will install the software needed for particular tasks. Of particular interest to us is `mlnx-ofed-hpc`:

```
[root@master ~]# yum info mlnx-ofed-hpc
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
Available Packages
Name        : mlnx-ofed-hpc
Arch        : noarch
Version     : 4.6
Release     : 1.0.1.1.rhel7.6
Size        : 6.1 k
Repo        : mlnx
Summary     : MLNX_OFED hpc installer package (with KMP support)
URL         : http://mellanox.com
License     : GPLv2 or BSD
Description : MLNX_OFED hpc installer package (with KMP support)
```

Install this on the master. It will take a little while. Let's use it with the `time` command to see how long:

```
[root@master ~]# time yum -y install mlnx-ofed-hpc
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
Resolving Dependencies
--> Running transaction check
...(lots of package installs)

Complete!

real    5m35.542s
```

```
user    5m7.848s
sys     2m53.027s
```

Running `yum` inside of the `time` command allowed us to track the time spent on the command. This can be pretty handy. We can see here that the command ran for 5 minutes 35 seconds.

We now have quite a few commands available on our master for dealing with IB. By typing `ib<tab><tab>` we'll get a list of commands starting with "ib". Most of these are for working with the Infiniband.

```
[root@master ~]# ib<tab><tab>
ib2ib_setup           ibdiagpath            ib_send_lat
ibacm                 ibdiscover.pl         ibstat
ib_acme               ibdmchk               ibstatus
ibaddr                ibdmtr                ibswitches
ib_atomic_bw          ibfindnodesusing.pl   ibswportwatch.pl
ib_atomic_lat         ibgenperm             ibsysstat
ibcacheedit           ibhosts               ibtopodiff
ibccconfig            ibidsverify.pl        ibtracert
ibccquery             iblinkinfo            ibv_asyncwatch
ibcheckerrors         iblinkinfo.pl         ibv_cc_pingpong
ibcheckerrs           ibmirror              ibv_dcini
ibchecknet            ibnetdiscover         ibv_dctgt
ibchecknode           ibnetsplit            ibv_devices
ibcheckport           ibnlparse             ibv_devinfo
ibcheckportstate      ibnodes               ibv_intf
ibcheckportwidth      ibping                ibv_polldcinfo
ibcheckstate          ibportstate           ibv_rc_pingpong
ibcheckwidth          ibprintca.pl          ibv_srq_pingpong
ibclearcounters       ibprintrt.pl          ibv_task_pingpong
ibclearerrors         ibprintswitch.pl      ibv_uc_pingpong
ibcongest             ibqueryerrors         ibv_ud_pingpong
ibdatacounters        ibqueryerrors.pl      ibv_umr
ibdatacounts          ib_read_bw            ibv_xsrq_pingpong
ibdev2netdev          ib_read_lat           ib_write_bw
ibdiagm.sh            ibroute               ib_write_lat
ibdiagnet             ibrouters
ibdiagnet_csv2xml.py  ib_send_bw
```

Fortunately, we'll only need a few of these for normal operation.

At the moment, these will generally fail because we don't have the IB interface configured. One of the handy ones is `iblinkstat`, that check our IB port's link status. Let's try it.

```
[root@master ~]# ibstat
```

This will likely result in an error because the appropriate drivers aren't loaded yet.

Enable/Start services on the master

There are two services we'll need on the master:

1. `openibd` - this initializes the Infiniband hardware (we'll need this one on the nodes too), including loading the necessary kernel modules. To see what modules it loads, let's first list our modules:

```
[root@master ~]# lsmod
Module                  Size  Used by
udp_diag               12801  0
inet_diag              18949  1 udp_diag
nfsd                  351388  11
nfs_acl                12837  1 nfsd
... (lots more modules)
```

Now enable and start the service:

```
[root@master ~]# systemctl enable openibd
[root@master ~]# systemctl start openibd
```

Run `lsmod` again:

```
[root@master ~]# lsmod
Module                  Size  Used by
rdma_ucm               26930  0
ib_ucm                 22602  0
rdma_cm                60234  1 rdma_ucm
iw_cm                  43514  1 rdma_cm
ib_ipoib              176895  0
ib_cm                  53141  3 rdma_cm,ib_ucm,ib_ipoib
ib_umad                22093  0
mlx5_fpga_tools        14392  0
mlx5_ib               345327  0
mlx5_core             984868  2 mlx5_ib,mlx5_fpga_tools
mlxfw                  18227  1 mlx5_core
mlx4_en               146420  0
ptp                    19231  2 mlx4_en,mlx5_core
pps_core               19057  1 ptp
mlx4_ib               212206  0
ib_uverbs             127130  4 mlx4_ib,mlx5_ib,ib_ucm,rdma_ucm
ib_core               300520  10
rdma_cm,ib_cm,iw_cm,mlx4_ib,mlx5_ib,ib_ucm,ib_umad,ib_uverbs,rdma_ucm,
ib_ipoib
mlx4_core             360639  2 mlx4_en,mlx4_ib
mlx_compat             29590  15
rdma_cm,ib_cm,iw_cm,mlx4_en,mlx4_ib,mlx5_ib,mlx5_fpga_tools,ib_ucm,ib_
core,ib_umad,ib_uver
devlink                48345  4 mlx4_en,mlx4_ib,mlx4_core,mlx5_core
```

`lsmod` lists modules in reverse order to when they are loaded (newest on top). These are all modules that the `openib` service loads.

If we look at our network interfaces, we'll also see that we now have an `ib0` interface.

```
[root@master ~]# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
mode DEFAULT group default qlen 1000 link/loopback 00:00:00:00:00:00
brd 00:00:00:00:00:00
...(the em* interfaces)
5: ib0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 2044 qdisc mq state UP
mode DEFAULT group default qlen 256 link/infiniband
20:00:08:a6:fe:80:00:00:00:00:00:00:11:22:33:44:77:66:77:12 brd
00:ff:ff:ff:ff:12:40:1b:ff:ff:00:00:00:
```

Now that the kernel modules are loaded, let's try our `ibstat` command again:

```
[root@master ~]# ibstat
CA 'mlx5_0'
    CA type: MT4116
    Number of ports: 1
    Firmware version: 12.25.1020
    Hardware version: 0
    Node GUID: 0x1122334477667711
    System image GUID: 0x248a07030029cc48
    Port 1:
        State: Active
        Physical state: LinkUp
        Rate: 100
        Base lid: 12
        LMC: 0
        SM lid: 11
        Capability mask: 0x2651ec48
        Port GUID: 0x1122334477667712
        Link layer: InfiniBand
```

Infiniband does not natively use the IP protocol, but there is a commonly used emulation layer called IPoIB (IP-over-IB) that allows you to use the IB network for normal IP traffic. This is important for some applications to work. It is often used for applications that use IB to bring up initial IB connections, for instance.

Let's configure the IP address for our `ib0` interface. Create the file `/etc/sysconfig/network-scripts/ifcfg-ib0` with the following contents:

```
TYPE=Infiniband
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=static
```

```
DEFROUTE=no
IPV4_FAILURE_FATAL=no
IPV6INIT=no
NAME=ib0
DEVICE=ib0
ONBOOT=yes
IPADDR=192.168.0.254
NETMASK=255.255.255.0
```

Now, `ifup` the interface and verify that it's working:

```
[root@master ~]# ifup ib0
[root@master ~]# ip addr show ib0
5: ib0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 2044 qdisc mq state UP
group default qlen 256
    link/infiniband
20:00:08:a6:fe:80:00:00:00:00:00:00:11:22:33:44:77:66:77:12 brd
00:ff:ff:ff:ff:12:40:1b:ff:ff:00:00:00:
    inet 192.168.0.254/24 brd 192.168.0.255 scope global noprefixroute
ib0
       valid_lft forever preferred_lft forever
    inet6 fe80::1322:3344:7766:7712/64 scope link
       valid_lft forever preferred_lft forever
```

2. `opensm` - this runs the subnet manager that we will need to resolve routes on our IB fabric. Every IB fabric (or other HSN fabric, generally) needs to have at least one of these on the fabric. Some IB switches support running it on the switch, but we still will generally run it on a server, often the master. Sometimes we'll run more than one for failover, but that requires extra configuration.

   Without the subnet manager, the IB will be unusable. There are a couple of commands we can use to verify the subnet manager. The simplest is `sminfo`. This just lists what is known about the subnet manager:

   ```
   [root@master ~]# sminfo
   sminfo: iberror: failed: query
   ```

   This is clearly broken.

   Another, more generic diagnostic tool is `ibdiagnet`. This runs a sequence of diagnostics and will report on their status:

   ```
   [root@master ~]# ibdiagnet
   ...
   Plugin Name                                   Result     Comment
   libibdiagnet_cable_diag_plugin-2.1.1          Succeeded  Plugin loaded
   libibdiagnet_phy_diag_plugin-2.1.1            Succeeded  Plugin loaded
   ```

```
-------------------------------------------
Discovery
-I- Discovering ... 12 nodes (1 Switches & 11 CA-s) discovered.
-I- Fabric Discover finished successfully
...(lots more output
```

This command has a lot of useful information, and will often be your first tool for diagnosing an IB problem. Errors will start with "`-E-`", we can `grep` for these:

```
[root@master ~]# ibdiagnet | grep -E '^-E'
-E- Subnet Manager Check finished with errors
-E- Not found master subnet manager in fabric
-E- FW Check finished with errors
```

This is telling us pretty clearly that we need to have a subnet manager running and we don't. Let's start/enable it:

```
[root@master ~]# systemctl enable opensmd
[root@master ~]# systemctl start opensmd
```

Now let's try again:

```
[root@te-hyperv ~]# sminfo
sminfo: sm lid 11 sm guid 0x248a07030029cc48, activity count 349
priority 0 state 3 SMINFO_MASTER
```

If we run `ibdiagnet` we will also see that the subnet manager errors have gone away. Our fabric is now functional.

Install necessary software & drivers in the BOS image

We have the IB setup for the master. Now we need it in the image. As we saw before it can take a while:

```
[root@master ~]# yum -y --installroot=/opt/ohpc/admin/images/centos7
install mlnx-ofed-hpc
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
Resolving Dependencies
--> Running transaction check
...

Complete!
```

We only need the `openibd` service in the image. We can enable it with:

```
[root@master ~]# systemctl --root=/opt/ohpc/admin/images/centos7 enable
openibd
```

Re-build the VNFS & setup warewulf IB

We've modified our image, now we'll need to rebuild the VNFS.

```
[root@master ~]# wwvnfs --chroot=/opt/ohpc/admin/images/centos7 centos7-
mlnx
Creating VNFS image from centos-mlnx
Compiling hybridization link tree                          : 0.17 s
Building file list                                         : 0.54 s
Compiling and compressing VNFS                             : 19.82 s
Adding image to datastore                                  : 46.42 s
Wrote a new configuration file at: /etc/warewulf/vnfs/centos-mlnx.conf
Total elapsed time                                         : 66.96 s
```

Note that we used a new image name, `centos7-mlnx`, instead of `centos7-base` that we used before. We'll see why in a moment.

We can see that the VNFS is imported:

```
[root@master ~]# wwsh vnfs list
VNFS NAME            SIZE (M)   ARCH       CHROOT LOCATION
centos7-mlnx         399.6      x86_64     /opt/ohpc/admin/images/centos7
centos7-base         261.8      x86_64     /opt/ohpc/admin/images/centos7
```

We are also going to need our `ib0` interface's IPoIB configured on the nodes. We will use warewulf files and network config objects to do this.

OpenHPC provided us with a useful template to get started with this. This template will auto-fill an `ifcfg-ib0` interface on each node with the information that we tell warewulf about their `ib0` interfaces.

```
[root@master ~]# cat /opt/ohpc/pub/examples/network/centos/ifcfg-ib0.ww
DEVICE=ib0
BOOTPROTO=static
IPADDR=%{NETDEVS::IB0::IPADDR}
NETMASK=%{NETDEVS::IB0::NETMASK}
ONBOOT=yes
NM_CONTROLLED=no
DEVTIMEOUT=5
```

We need to import the file into warewulf's file list and add it to the provision set for each node.

```
[root@master ~]# wwsh file import
/opt/ohpc/pub/examples/network/centos/ifcfg-ib0.ww
[root@master ~]# wwsh file list
dynamic_hosts           :  rw-r--r-- 0   root root            1501
/etc/hosts
group                   :  rw-r--r-- 1   root root             619
/etc/group
ifcfg-ib0.ww            :  rw-r--r-- 1   root root             133
/opt/ohpc/pub/examples/network/centos/ifcfg-ib0.ww
passwd                  :  rw-r--r-- 1   root root            1370
/etc/passwd
shadow                  :  rw-r----- 1   root root             890
/etc/shadow
```

We need to tell warewulf that this file needs to live in a different location on the nodes than it does on the master. We can do this with the `wwsh file set` command:

```
[root@master ~]# wwsh -y file set ifcfg-ib0.ww --
path=/etc/sysconfig/network-scripts/ifcfg-ib0
About to apply 1 action(s) to 1 file(s):

     SET: PATH                = /etc/sysconfig/network-scripts/ifcfg-ib0

Proceed?
```

This remapped its location to `/etc/sysconfig/network-scripts/ifcfg-ib0`.

We need to tell warewulf's object store about the `ib0` information so this template can get filled. We can do this in one command with a `for` loop:

```
[root@master ~]# for i in $(seq -w 1 10); do wwsh -y node set n$i -D ib0 -
-ipaddr=192.168.0.$i --netmask=255.255.255.0; done
```

To verify that the changes were made:

```
[root@master ~]# wwsh node print n01
#### n01
================================================================================
...
            n01: ib0.HWADDR       = UNDEF
            n01: ib0.HWPREFIX     = UNDEF
            n01: ib0.IPADDR       = 192.168.0.1
            n01: ib0.NETMASK      = 255.255.255.0
            n01: ib0.NETWORK      = UNDEF
```

```
            n01: ib0.GATEWAY      = UNDEF
            n01: ib0.MTU          = UNDEF
            n01: ib0.FQDN         = UNDEF
...
```

Warewulf now knows about the `ib0` IP address and netmask, so it can fill out the template for each node.

Finally, we need to add this file to the file list for all of our nodes:

```
[root@master ~]# wwsh -y provision set n[01-10] --fileadd=ifcfg-ib0.ww
```

To verify:

```
[root@master ~]# wwsh provision print n01
#### n01
################################################################################
...j
            n01: FILES            = dynamic_hosts,group,ifcfg-
ib0.ww,passwd,shadow
...
```

We're ready to actually use these changes.

Re-start the nodes & verify IB

The final step to deploying our changes on the nodes is to reboot them into the new image. This is the general workflow for modifying software/configuration on nodes. Part of the stateless model for clustering is making sure that the master fully specifies the operating system for the node. The way we make sure this is the case is by (generally, sometimes we make exceptions) installing new software by loading a clean operating system image on the node, i.e. rebooting.

Remember that we stored this new VNFS with a new name, `centos-mlnx`. One reason to do this in a real-world situation is that it allows us to test a change on, e.g. a single node without changing the VNFS for all nodes. If we look at the provision list:

```
[root@master ~]# wwsh provision list
NODE                VNFS            BOOTSTRAP             FILES
================================================================================
======
DEFAULT             centos7-base    3.10.0-957.12.2.el...
dynamic_hosts,grou...
n01                 centos7-base    3.10.0-957.12.2.el...
dynamic_hosts,grou...
n02                 centos7-base    3.10.0-957.12.2.el...
dynamic_hosts,grou...
n03                 centos7-base    3.10.0-957.12.2.el...
```

```
dynamic_hosts,grou...
...
```

All of our nodes are still mapped to the `centos7-base` image. We'll start by remapping just `n01`:

```
[root@master ~]# wwsh provision set n01 -V centos7-mlnx
Are you sure you want to make the following changes to 1 node(s):

     SET: VNFS                = centos7-mlnx

Yes/No> Yes
```

```
[root@master ~]# wwsh provision list
NODE                VNFS            BOOTSTRAP             FILES
================================================================================
======
DEFAULT             centos7-base    3.10.0-957.12.2.el...
dynamic_hosts,grou...
n01                 centos7-mlnx    3.10.0-957.12.2.el...
dynamic_hosts,grou...
n02                 centos7-base    3.10.0-957.12.2.el...
dynamic_hosts,grou...
n03                 centos7-base    3.10.0-957.12.2.el...
dynamic_hosts,grou...
...
```

Now let's reboot the node. We could use `pm` to do a ***hard*** reboot, but it's safer to log in to the node and tell it to reboot.

```
[root@master ~]# ssh n01 systemctl reboot
Connection to n01 closed by remote host.
```

It's often a good idea, especially when testing a new image, to watch the node boot with `conman`:

```
[root@master ~]# conman n01

<ConMan> Connection to console [n01] opened.
[73286.941232] Restarting system.
...
Starting the provision handler:
 * adhoc-pre
OK
 * ipmiconfig Auto configuration not activated
SKIPPED
 * filesystems
```

```
RUNNING
   * mounting /
OK
 * filesystems
OK
 * getvnfs
RUNNING
   * fetching centos7-mlnx (ID:15)
...
[  OK  ] Started mlnx_interface_mgr - configure ib0.
[  OK  ] Started openibd - configure Mellanox devices.
         Starting LSB: Bring up/down networking...
...
CentOS Linux 7 (Core)
Kernel 3.10.0-957.12.2.el7.x86_64 on an x86_64

n01 login:
```

Looks like it booted, let's exist `conman` (`&.`) and ssh to the node to check it.

```
[root@master ~]# ssh n01
[root@n01 ~]# ls
yum-ww.conf
[root@n01 ~]# ibstat
CA 'mlx5_0'
    CA type: MT4115
    Number of ports: 1
    Firmware version: 12.17.2020
    Hardware version: 0
    Node GUID: 0x248a07030029cfdc
    System image GUID: 0x248a07030029cfdc
    Port 1:
        State: Active
        Physical state: LinkUp
        Rate: 100
        Base lid: 1
        LMC: 0
        SM lid: 11
        Capability mask: 0x2651e848
        Port GUID: 0x248a07030029cfdc
        Link layer: InfiniBand
```

Make sure it sees the subnet manager:

```
[root@n01 ~]# sminfo
sminfo: sm lid 11 sm guid 0x248a07030029cc48, activity count 1392 priority
0 state 3 SMINFO_MASTER
```

Here's another handy command to see what's going on with the IB fabric (especially after more than one node is up):

```
[root@n01 ~]# iblinkinfo
...(output is too big to print here)
```

`iblinkinfo` will tell you the status of all of the links on the IB fabric.

Finally, let's make sure our IPoIB got set:

```
[root@n01 ~]# ip addr show ib0
7: ib0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 2044 qdisc pfifo_fast state
UP group default qlen 256
    link/infiniband
80:00:00:68:fe:80:00:00:00:00:00:00:24:8a:07:03:00:29:cf:dc brd
00:ff:ff:ff:ff:12:40:1b:ff:ff:00:00:00:
    inet 192.168.0.1/24 brd 192.168.0.255 scope global ib0
       valid_lft forever preferred_lft forever
    inet6 fe80::268a:703:29:cfdc/64 scope link
       valid_lft forever preferred_lft forever
```

Everything looks good here. Exit out of `ssh` and get back to the master.

Let's go ahead and transition all of the nodes to the new image and reboot them.

```
[root@master ~]# wwsh provision set n[2-10] -V centos7-mlnx
Are you sure you want to make the following changes to 2 node(s):

     SET: VNFS                = centos7-mlnx

Yes/No> Yes
```

```
[root@master ~]# wwsh provision list
NODE                VNFS            BOOTSTRAP             FILES
================================================================================
======
DEFAULT             centos7-base    3.10.0-957.12.2.el...
dynamic_hosts,grou...
n01                 centos7-mlnx    3.10.0-957.12.2.el...
dynamic_hosts,grou...
n02                 centos7-mlnx    3.10.0-957.12.2.el...
dynamic_hosts,grou...
n03                 centos7-mlnx    3.10.0-957.12.2.el...
dynamic_hosts,grou...
...
```

We can use `pdsh` to reboot all of the rest of the nodes:

```
[root@master ~]# pdsh -A -x n01 systemctl reboot
n02: Connection to n02 closed by remote host.
n01: Connection to n02 closed by remote host.
n03: Connection to n02 closed by remote host.
...
```

It's a good idea to watch one of these with `conman` to get a sense of when they come up:

```
[root@master ~]# conman n02

<ConMan> Connection to console [n02] opened.
[73910.939371] Restarting system.
...
CentOS Linux 7 (Core)
Kernel 3.10.0-957.12.2.el7.x86_64 on an x86_64

n02 login:
```

Now, verify that all of the nodes are up with `pdsh`:

```
[root@master ~]# pdsh -A uptime
n03:  12:08:54 up 2 min,  0 users,  load average: 0.14, 0.10, 0.04
n02:  12:08:56 up 2 min,  0 users,  load average: 0.13, 0.10, 0.04
n01:  12:08:54 up 12 min,  0 users,  load average: 0.00, 0.01, 0.04
...
```

Finally, run an `iblinkinfo` and see that all of your IB links in the fabric look correct. Things to look for include:

Every link should report as:

```
4X      25.78125 Gbps Active/  LinkUp
```

- Are all of the expected links there?
- Does everything report LinkUp?
- Does everything report the right speed?

Finally, make sure `ibdiagnet` doesn't report any errors.

We now have working Infiniband!

Step 4: Deploying Slurm

Up until now, we have no way to actually make our cluster do work than to ssh to a node and launch a process. Most clusters use a "Work Load Manager" (WLM) or "Queuing system" to manage work to be done on a cluster. This serves multiple purposes:

- Provides a nice interface for launch work on a cluster, possibly across multiple nodes at once
- Provides a queue to make sure we only run as much work as we can handle at once
- Generally provides SLA ("Service Level Agreement") policies to make sure different users get the share of the system that they should
- Provides accounting information on who has used what resources

We will now deploy a popular WLM called "Slurm". Our workflow will be similar to what we followed in the Infiniband step:

1. Setup software/services on the master
2. Setup software/services on the nodes
3. Map necessary configuration to the nodes
4. Create a VNFS and test it on one node
5. Remap our image to the rest of the nodes and reboot

Setup software/services on the master

Slurm has multiple pieces that are needed in different locations. The only piece strictly needed on the master is `slurmctld`. This is the services that control a Slurm cluster. The nodes will run a `slurmd` service that synchronizes them with the `slurmctld`. An optional component, `slurmdbd`, keeps accounting information in a database. We will not be deploying `slurmdbd`.

Additionally, all Slurm processes require a service called `munge` to be running, but on the master and compute nodes. `munge` handles internal authorization for Slurm. It has a special secret key that must be shared across the cluster.

We will only install `slurmctld` on the master. We will also install the package `slurm-example-configs-ohpc` so that we have some example configurations.

On the master:

```
[root@master ~]# yum -y install slurm-ohpc slurm-slurmctld-ohpc slurm-
example-configs-ohpc
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
Resolving Dependencies
--> Running transaction check
...

Complete!
```

We need to write a Slurm configuration. This is stored at `/etc/slurm/slurm.conf`. We need to edit this file. It has the form `<parameter>=<value>`. The following `<parameters>` need to be modified from the default:

```
ClusterName=<your_cluster_name>
ControlMachine=<your_master_hostname>
SallocDefaultCommand="/usr/bin/srun -n1 -N1 --mem-per-cpu=0 --pty --
preserve-env --mpi=none $SHELL"
NodeName=n[01-10] Sockets=2 CoresPerSocket=16 ThreadsPerCore=1
State=UNKNOWN
PartitionName=normal Nodes=n[01-10] Default=YES MaxTime=24:00:00 State=UP
```

Now let's enable and start services:

```
[root@master ~]# systemctl enable munge slurmctld
[root@master ~]# systemctl start munge slurmctld
```

```
[root@master ~]# systemctl status munge
● munge.service - MUNGE authentication service
   Loaded: loaded (/usr/lib/systemd/system/munge.service; enabled; vendor
preset: disabled)
   Active: active (running) since Sun 2019-06-09 12:37:33 MDT; 2min 33s
ago
...
```

```
[root@master ~]# systemctl status slurmctld
● slurmctld.service - Slurm controller daemon
   Loaded: loaded (/usr/lib/systemd/system/slurmctld.service; enabled;
vendor preset: disabled)
   Active: active (running) since Sun 2019-06-09 12:37:40 MDT; 2min 46s
ago
...
```

We'll see that `munge` created a `munge` key file:

```
[root@master ~]# ls -l /etc/munge/munge.key
-r--------. 1 munge munge 1024 Jun  9 12:28 /etc/munge/munge.key
```

We will have to share this key with our nodes. It's extremely important that the permissions on the key are correct, as seen here.

An extremely common reason for Slurm to be broken is that the `munge` service is broken. The most common reason for the `munge` service to be broken is that there is something wrong with the `munge` key.

Finally, if all has gone well, we can run some Slurm commands to test:

```
[root@master ~]# sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
normal*      up 1-00:00:00      0    n/a
```

`sinfo` is a general status command for Slurm. It can be used to tell you the state of your cluster. Everything is down right now because Slurm isn't running on our nodes yet.

When we need more detail, we can use the `scontrol` command. It has many options and functions. Here we can see the status of our "partition":

```
[root@master ~]# scontrol show partition
PartitionName=normal
   AllowGroups=ALL AllowAccounts=ALL AllowQos=ALL
   AllocNodes=ALL Default=YES QoS=N/A
   DefaultTime=NONE DisableRootJobs=NO ExclusiveUser=NO GraceTime=0
Hidden=NO
   MaxNodes=UNLIMITED MaxTime=1-00:00:00 MinNodes=0 LLN=NO
MaxCPUsPerNode=UNLIMITED
   Nodes=n[1-10]
   PriorityJobFactor=1 PriorityTier=1 RootOnly=NO ReqResv=NO
OverSubscribe=NO
   OverTimeLimit=NONE PreemptMode=OFF
   State=UP TotalCPUs=320 TotalNodes=10 SelectTypeParameters=NONE
   JobDefaults=(null)
   DefMemPerNode=UNLIMITED MaxMemPerNode=UNLIMITED
```

A "partition" is a logical segment of a slurm cluster. It contains a sub-grouping of nodes. Ours only has one called "normal", but you can have many different partitions. You may be able to spot the line in `slurm.conf` that defines this partition.

Setup software/services on the nodes

We need `slurmd` on our nodes, so we'll need to install into the BOS:

```
[root@master ~]# yum -y --installroot=/opt/ohpc/admin/images/centos7
install slurm-ohpc slurm-slurmd-ohpc slurm-example-configs-ohpc
```

We need our `munge.key` setup in the image. We could use `wwsh file` for this, but we don't expect the `munge.key` to ever change, so we can build it directly into our image:

```
[root@master ~]# cp -av /etc/munge/munge.key
/opt/ohpc/admin/images/centos7/etc/munge/
cp: overwrite ‘/opt/ohpc/admin/images/centos7/etc/munge/munge.key’? y
‘/etc/munge/munge.key’ ->
‘/opt/ohpc/admin/images/centos7/etc/munge/munge.key’
```

```
[root@master ~]# ls -l /opt/ohpc/admin/images/centos7/etc/munge/
total 4
-r--------. 1 munge munge 1024 Jun  9 12:28 munge.key
```

```
[root@master ~]# md5sum /etc/munge/munge.key
/opt/ohpc/admin/images/centos7/etc/munge/munge.key
a2a35e99d51f6c686b31046724196c72  /etc/munge/munge.key
a2a35e99d51f6c686b31046724196c72
/opt/ohpc/admin/images/centos7/etc/munge/munge.key
[root@master ~]#
```

We've verified that we have identical `munge.key` files in the image and master, and that the permissions are correct.

Now we need to enable the `munge` and `slurmd` services in the BOS:

```
[root@master ~]# systemctl --root=/opt/ohpc/admin/images/centos7 enable
munge slurmd
Created symlink /opt/ohpc/admin/images/centos7/etc/systemd/system/multi-
user.target.wants/slurmd.service, pointing to /usr/
```

Map necessary configuration to the nodes

We will map three configuration files into the node:

- `/etc/slurm/slurm.conf` - the main slurm configuration
- `/etc/slurm/slurm.epilog.clean` - a script that will run after each job finishes (we won't change it from the defaultj)
- `/etc/slurm/cgroup.conf` - a configuration file that slurm uses to setup `cgroups`. We won't need to modify this, but it must be in place for `slurmd`.

First, we need to create `/etc/slurm/cgroup.conf`. We'll just copy it from the example:

```
[root@master ~]# cp /etc/slurm/cgroup.conf.example /etc/slurm/cgroup.conf
```

Now, let's add these three files to warewulf. We'll use the `wwsh` shell this time:

```
[root@master ~]# wwsh
Warewulf> file import /etc/slurm/slurm.conf
Warewulf> file import /etc/slurm/cgroup.conf
Warewulf> file import /etc/slurm/slurm.epilog.clean
Warewulf> file list
```

```
cgroup.conf             :  rw-r--r-- 1   root root             216
/etc/slurm/cgroup.conf
dynamic_hosts           :  rw-r--r-- 0   root root            1637
/etc/hosts
group                   :  rw-r--r-- 1   root root             619
/etc/group
ifcfg-ib0.ww            :  rw-r--r-- 1   root root             133
/etc/sysconfig/network-scripts/ifcfg-ib0
passwd                  :  rw-r--r-- 1   root root            1370
/etc/passwd
shadow                  :  rw-r----- 1   root root             890
/etc/shadow
slurm.conf              :  rw-r--r-- 1   root root            2253
/etc/slurm/slurm.conf
slurm.epilog.clean      :  rwxr-xr-x 1   root root             888
/etc/slurm/slurm.epilog.clean
```

Now, we need to map those files to our nodes:

```
[root@master ~]# wwsh -y provision set n[01-10] --
fileadd=slurm.conf,slurm.epilog.clean,cgroup.conf
Are you sure you want to make the following changes to 3 node(s):

     ADD: FILES                = slurm.conf,slurm.epilog.clean,cgroup.conf

Yes/No>
[root@master ~]# wwsh provision print n01
#### n01
================================================================================
...
            n01: FILES            = cgroup.conf,dynamic_hosts,group,ifcfg-
ib0.ww,passwd,shadow,slurm.conf,slurm.epilog.clea
...
```

Our files are now mapped in.

The `slurm` and `munge` services each added new users to our cluster, so we need to resync our `passwd`, `group`, and `shadow` files, otherwise the special users won't exist for our nodes:

```
[root@master ~]# wwsh file resync
```

We can verify we have the right version of the `passwd` file:

```
[root@master ~]# wwsh file print passwd
#### passwd
================================================================================
passwd          : ID               = 1
```

```
passwd          : NAME             = passwd
passwd          : PATH             = /etc/passwd
passwd          : ORIGIN           = /etc/passwd
passwd          : FORMAT           = data
passwd          : CHECKSUM         = 2861405b284f1f4bc7979873bffcff54
passwd          : INTERPRETER      = UNDEF
passwd          : SIZE             = 1500
passwd          : MODE             = 0644
passwd          : UID              = 0
passwd          : GID              = 0
[root@master ~]# md5sum /etc/passwd
2861405b284f1f4bc7979873bffcff54  /etc/passwd
```

The checksum is correct.

Create a VNFS and test it on one node

Once again, we'll create a VNFS with a new name:

```
[root@master ~]# wwvnfs --chroot=/opt/ohpc/admin/images/centos7 centos7-
slurm
Creating VNFS image from centos7-slurm
Compiling hybridization link tree                           : 0.18 s
Building file list                                          : 0.50 s
Compiling and compressing VNFS                              : 21.68 s
Adding image to datastore                                   : 49.86 s
Wrote a new configuration file at: /etc/warewulf/vnfs/centos7-slurm.conf
Total elapsed time                                          : 72.23 s
```

```
[root@master ~]# wwsh vnfs list
VNFS NAME            SIZE (M)   ARCH       CHROOT LOCATION
centos7-base         261.8      x86_64     /opt/ohpc/admin/images/centos7
centos7-mlnx         399.6      x86_64     /opt/ohpc/admin/images/centos7
centos7-slurm        426.6      x86_64     /opt/ohpc/admin/images/centos7
```

Now we'll follow the same procedure as before.

```
[root@master ~]# wwsh provision set n01 -V centos7-slurm
Are you sure you want to make the following changes to 1 node(s):

     SET: VNFS                = centos7-slurm

Yes/No> Yes
```

```
[root@master ~]# ssh n01 systemctl reboot
Connection to n01 closed by remote host.
```

```
[root@master ~]# conman n01

<ConMan> Connection to console [n01] opened.
[ 4108.855476] Restarting system.
...
[  OK  ] Started MUNGE authentication service.
...
[  OK  ] Started Slurm node daemon.
...
CentOS Linux 7 (Core)
Kernel 3.10.0-957.12.2.el7.x86_64 on an x86_64

n01 login:
```

Now, on the master we can try `sinfo` again:

```
[root@master ~]# sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
normal*      up 1-00:00:00      2   unk* n[02-03]
normal*      up 1-00:00:00      1   idle n01
```

We see that `n01` is idle.

Let's try it out:

```
[root@master ~]# salloc
salloc: Granted job allocation 5
[root@n01 ~]# hostname
n01
[root@n01 ~]# squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES
NODELIST(REASON)
                 5    normal       sh     root  R       0:08      1 n01
[root@n01 ~]# exit
```

We'll have the opportunity to use Slurm more later.

Remap our image to the rest of the nodes and reboot

Now that everything is working, let's set the VNFS for the rest of the nodes to `centos7-slurm` and reboot.

```
[root@master ~]# wwsh provision set n[02-10] -V centos7-slurm
Are you sure you want to make the following changes to 2 node(s):

     SET: VNFS                = centos7-slurm

Yes/No> Yes
```

```
[root@master ~]# wwsh provision list
NODE                VNFS            BOOTSTRAP             FILES
================================================================================
======
DEFAULT             centos7-base    3.10.0-957.12.2.el...
dynamic_hosts,grou...
n01                 centos7-slurm   3.10.0-957.12.2.el...
cgroup.conf,dynami...
n02                 centos7-slurm   3.10.0-957.12.2.el...
cgroup.conf,dynami...
n03                 centos7-slurm   3.10.0-957.12.2.el...
cgroup.conf,dynami...
...
```

```
[root@master ~]# pdsh -A -x n01 systemctl reboot
n03: Connection to n03 closed by remote host.
pdsh@master: n03: ssh exited with exit code 255
n02: Connection to n02 closed by remote host.
pdsh@master: n02: ssh exited with exit code 255
...
```

```
[root@master ~]# conman n02
...
```

Now let's check `sinfo`:

```
[root@master ~]# sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
normal*      up 1-00:00:00      1   idle n01
normal*      up 1-00:00:00      2   down n[02-10]
```

Why does it show them as down? This is a little misleading. If they were actually down the would say `*down` not `down`. The `*` means that they cannot be contacted. Whenever a resource goes away, by default we need to `resume` it.

```
[root@master ~]# scontrol update nodename=n[02-10] state=resume
```

```
[root@master ~]# sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
normal*      up 1-00:00:00     10   idle n[01-10]
```

That's better. We now have 10 nodes we can run slurm jobs on!

Step 5: Program environments with `lmod`

Users of a compute cluster often need various programming environments (PEs) available for their applications. For instance, one package may require OpenMPI 3.x while another uses OpenMPI 4.x. Or perhaps one requires GCC 7.x while the other requires the PGI compiler suite. We need a convenient way to manage this type of software for the user.

`lmod` is a package that takes different software environment specifications and makes them available to the user through a convenient user interface. Typically, the software that `lmod` manages is going to live on a shared filesystem. In our case, we will install this kind of tool under `/opt/ohpc/pub`, which the nodes have access to through the NFS share.

OpenHPC provides a number of packages that will automatically be set up to use `lmod`. We already built base `lmod` support into our images from the beginning. Let's start by installing some of the development tools OpenHPC provides.

On the master (this will take several minutes):

```
[root@master ~]# yum -y install EasyBuild-ohpc \
                    gnu8-compilers-ohpc \
                    hwloc-ohpc \
                    llvm5-compilers-ohpc \
                    lmod-defaults-gnu8-openmpi3-ohpc \
                    mpich-gnu8-ohpc \
                    ohpc-autotools \
                    ohpc-gnu8-io-libs \
                    ohpc-gnu8-mpich-parallel-libs \
                    ohpc-gnu8-openmpi3-parallel-libs \
                    ohpc-gnu8-perf-tools \
                    ohpc-gnu8-python-libs \
                    ohpc-gnu8-runtimes \
                    ohpc-gnu8-serial-libs \
                    openmpi3-pmix-slurm-gnu8-ohpc \
                    spack-ohpc \
                    valgrind-ohpc

...(lots of packages install)
```

Now, log out and log back into the master. We can list the available `lmod` modules:

```
[root@master ~]# module avail

------------------------ /opt/ohpc/pub/moduledeps/gnu8-openmpi3 ---------
-----------------
   adios/1.13.1     mpiP/3.4.1             pnetcdf/1.11.0      scorep/4.1
   boost/1.69.0     mumps/5.1.2            ptscotch/6.0.6
sionlib/1.7.2
   dimemas/5.3.4    netcdf-cxx/4.3.0       py2-mpi4py/3.0.0
slepc/3.10.2
   extrae/3.5.2     netcdf-fortran/4.4.5   py2-scipy/1.2.1
superlu_dist/6.1.1
   fftw/3.3.8       netcdf/4.6.2           py3-mpi4py/3.0.0    tau/2.28
   hypre/2.15.1     opencoarrays/2.2.0     py3-scipy/1.2.1
trilinos/12.12.1
   imb/2018.1       petsc/3.10.3           scalapack/2.0.2
   mfem/3.4         phdf5/1.10.4           scalasca/2.4

------------------------------ /opt/ohpc/pub/moduledeps/gnu8 -------------
-----------------
   R/3.5.2         metis/5.1.0     openblas/0.3.5        py2-numpy/1.15.3
   gsl/2.5         mpich/3.3       openmpi3/3.1.3 (L)    py3-numpy/1.15.3
   hdf5/1.10.4     mvapich2/2.3    pdtoolkit/3.25        scotch/6.0.6
   likwid/4.3.3    ocr/1.0.1       plasma/2.8.0          superlu/5.2.1

-------------------------------- /opt/ohpc/admin/modulefiles --------------
-----------------
   spack/0.12.1

---------------------------------- /opt/ohpc/pub/modulefiles ---------------
-----------------
   EasyBuild/3.8.1           gnu8/8.3.0  (L)    papi/5.6.0
valgrind/3.14.0
   autotools          (L)    hwloc/2.0.3        pmix/2.2.2
   charliecloud/0.9.7        llvm5/5.0.1        prun/1.3          (L)
   cmake/3.13.4              ohpc        (L)    singularity/3.1.0

  Where:
   L:  Module is loaded

Use "module spider" to find all possible modules.
Use "module keyword key1 key2 ..." to search for all possible modules
matching any of the
"keys".
```

Since these are shared to our nodes, we should be able to get them there as well. Let's start an interactive job to a node and see what we can do. You can do this as your own user:

```
[user@master ~]$ salloc
salloc: Granted job allocation 7
[user@n01 ~]$ module avail
...(should give the same module list)
```

To see what modules you currently have loaded (some modules get loaded by default):

```
[user@n01 ~]$ module list

Currently Loaded Modules:
  1) autotools   2) prun/1.3   3) gnu8/8.3.0   4) openmpi3/3.1.3   5) ohpc
```

To load a new module:

```
[user@n01 ~]$ R --version
bash: R: command not found
[user@n01 ~]$ module load R
[user@n01 ~]$ R --version
R version 3.5.2 (2018-12-20) -- "Eggshell Igloo"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under the terms of the
GNU General Public License versions 2 or 3.
For more information about these matters see
http://www.gnu.org/licenses/.
```

Note: we could have specified the full `R` version, e.g. `module load R/3.5.2`.

We can also unload modules:

```
[user@n01 ~]$ module unload R
[user@n01 ~]$ R --version
bash: R: command not found
```

There are a number of different things you can do with `lmod`. For instance, you could load several modules you need and "save" that state so you can restore it any time you need it later.

We'll be using `lmod` modules later. The specification for `lmod` modules are written in a language called `lua`. We will not be going into the particulars of making a module, but you can look at the files in the tree: `/opt/ohpc/admin/lmod`.