Building a cluster with OpenHPC & Warewulf

Overview

Step 0: Base install of the master

We want to start our master from a clean slate. We will need to redo the following steps to get going:

1. Re-install the master so we're starting from a clean slate. Go ahead and set up the network for both `em1` and `em2`.
2. Set your hostname: `hostnamectl set-hostname <hostname>`
3. Re-add our users.
4. Disable selinux (`setenforce 0`, and edit `/etc/selinux/config`)
5. Disable firewalld (`systemctl stop firewalld`, `systemctl disable firewalld`)
6. Configure yum repos (found at http://10.0.52.146/repo/yum.repos.d.tar.gz)
7. `[root@master]# yum -y install tmux vim ntp`

Don't worry, this will be the last time we have to do these steps by hand.

Step 1: Install OpenHPC components

Verify OpenHPC repo setup

Ordinarily, you would have to set up access to the *OpenHPC* repository. Fortunately, we've had it all along from our local mirror list!

To verify this:

```
[root@master ~]# yum repolist
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
repo id             repo name           status
OpenHPC             OpenHPC             327
OpenHPC-updates     OpenHPC Updates     739
base                CentOS - Base       10,019
epel                EPEL                13,578
extras              CentOS - Extras     260
updates             CentOS - Updates    994
repolist: 25,917
```

Note `OpenHPC` and `OpenHPC-updates`. Also, a look at `/etc/yum.repos.d/OpenHPC.repo` to verify that the repository is configured and enabled.

To see what packages are available from OpenHPC, you can use the following:

```
[root@master ~]# yum --disablerepo='*' --enablerepo='OpenHPC*' list
available
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
Available Packages
EasyBuild-ohpc.x86_64       3.8.1-5.1.ohpc.1.3.7    OpenHPC-updates
R-gnu7-ohpc.x86_64          3.5.0-2.1               OpenHPC-updates
R-gnu8-ohpc.x86_64          3.5.2-4.1.ohpc.1.3.7    OpenHPC-updates
R_base-ohpc.x86_64          3.3.3-22.3              OpenHPC-updates
adios-gnu-impi-ohpc.x86_64  1.12.0-10.1             OpenHPC-updates
...
```

As you can see, OpenHPC provides quite a few (~1000) useful packages for OpenHPC. We'll be using quite a few of these.

Installing OpenHPC packages we'll need

To get started, we'll install the OpenHPC base packages, and Warewulf itself:

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

This will install quite a few packages (>150). Note: the `-y` option tells yum to just install the packages, don't prompt us to verify that we want to install them.

Step 2: Initial Warewulf setup

Setup Warewulf `provision.conf`

We need to edit the file `/etc/warewulf/provision.conf` to set some basic parameters about our cluster. Specifically, we need to set `device = em1`. Your `provision.conf` file should look like this:

```
# What is the default network device that the master will use to
# communicate with the nodes?
network device = em1

# Which DHCP server implementation should be used?
dhcp server = isc

# What is the TFTP root directory that should be used to store the
# network boot images? By default Warewulf will try and find the
# proper directory. Just add this if it can't locate it.
#tftpdir = /var/lib/tftpboot

# Automatically generate and manage a dynamnic_host virtual file
# object in the datastore? This is useful for provisioning this
# out to nodes so they always have a current /etc/hosts file.
generate dynamic_hosts = yes

# Should we manage and overwrite the local hostfile file on this
# system? This will cause all node entries to be added
# automatically to /etc/hosts.
update hostfile = yes

# If no cluster/domain is set on a node, should we add 'localdomain'
# as the default domain
use localdomain = yes

# The default kernel arguments to pass to the nodes boot kernel
default kargs = "net.ifnames=0 biosdevname=0 quiet"
```

Initialize Warewulf

Warewulf uses a database (`mariadb`) as its backend store for information about your cluster. This has already been installed as a dependency for `ohpc-warewulf` but we need to tell warewulf to initialize the database tables it needs. Run:

```
[root@master ~]# wwinit database
database:     Checking to see if RPM 'mysql-server' is installed
NO
database:     Checking to see if RPM 'mariadb-server' is installed
OK
database:     Activating Systemd unit: mariadb
database:      + /bin/systemctl -q enable mariadb.service
OK
```

```
database:      + /bin/systemctl -q restart mariadb.service
OK
database:      + mysqladmin --defaults-extra-
file=/tmp/0.6cgpUIwx0LMt/my.cnf OK
database:     Database version: UNDEF (need to create database)
database:      + mysql --defaults-extra-file=/tmp/0.6cgpUIwx0LMt/my.cnf
ware OK
database:      + mysql --defaults-extra-file=/tmp/0.6cgpUIwx0LMt/my.cnf
ware OK
database:      + mysql --defaults-extra-file=/tmp/0.6cgpUIwx0LMt/my.cnf
ware OK
database:     Checking binstore kind
SUCCESS
Done.
```

Warewulf will use ssh keys to provide access to nodes, just like we did in the netboot tutorial. It will manage this for us, but we have to initialize them:

```
[root@master ~]# wwinit ssh_keys
ssh_keys:     Checking ssh keys for root
NO
ssh_keys:     Generating ssh keypairs for local cluster access:
ssh_keys:      + ssh-keygen -t rsa -f /root/.ssh/cluster -N
OK
ssh_keys:     Updating authorized keys
OK
ssh_keys:     Checking root's ssh config
NO
ssh_keys:     Creating ssh configuration for root
DONE
ssh_keys:     Checking for default RSA host key for nodes
NO
ssh_keys:     Creating default node ssh_host_rsa_key:
ssh_keys:      + ssh-keygen -q -t rsa -f
/etc/warewulf/vnfs/ssh/ssh_host_rsa OK
ssh_keys:     Checking for default DSA host key for nodes
NO
ssh_keys:     Creating default node ssh_host_dsa_key:
ssh_keys:      + ssh-keygen -q -t dsa -f
/etc/warewulf/vnfs/ssh/ssh_host_dsa OK
ssh_keys:     Checking for default ECDSA host key for nodes
NO
ssh_keys:     Creating default node ssh_host_ecdsa_key:

OK
ssh_keys:     Checking for default Ed25519 host key for nodes
NO
ssh_keys:     Creating default node ssh_host_ed25519_key:

OK
Done.
```

It's worth looking through the output of this command to see exactly what it did. Where did it put the keys it generated?

Setup NFS exports

We will need a couple of shared filesystems for our cluster to work properly. We will use the Network File System (NFS) for this. NFS will let our nodes easily attach to directories on the master over our Cluster LAN network. Our nodes will need to mount `/home` so that our users can have their home directories on all of the nodes. We will also need `/opt/ohpc/pub`, for sharing things like add-on software with our nodes.

To share a filesystem via NFS, we need to add entries to `/etc/exports`. The format of this file is:

```
<directory_to_share>     <network|host|*>(<options>)
```

Where `<directory_to_share>` is the directory we want to give access to (e.g. `/home`), `<network|host|*>` is either a network spec (e.g. `172.16.0.0/24`), a specific host by IP or hostname, or `*` to indicate *any* host is allowed to mount the NFS share. `(<options>)` specifies special options we want for our share. Some common options are:

- `rw` or `ro` to indicate "read-write" or "read-only"
- `root_squash` or `no_root_squash`: `root_squash` indicates that the `root` user on the remote system should be treated as the `nobody` user, i.e. have no special permissions on the share. This is usually a good idea for security.
- `no_subtree_check` disables an expensive check that NFS does by default. It's common to use this option. For details, see: `man 5 exports`
- `fsid=<num>` sets a unique identifier for each mountpoint.

Our `/etc/exports` file should look like:

```
/home           172.16.0.0/24(rw,no_subtree_check,fsid=10,root_squash)
/opt/ohpc/pub   172.16.0.0/24(ro,no_subtree_check,fsid=11)
```

Note that `/opt/ohpc/pub` doesn't have the `root_squash` option. Since this is exported `ro`, `root` on a remote system won't be able to modify anything here, but we do need to make sure that `root` on the remote system can *read* everything we share here in order to use some of the software that will be provided from this share.

Now we need to actually enable the service that will make these shares available.

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

If you change `/etc/exports` while NFS is running, you will need to tell NFS to re-read the file. You can either do this by restarting `nfs-server`, or by running:

```
[root@master ~]# exportfs -a
```

This will also check the syntax of your exports file before changing anything, which makes it a bit safer than restarting `nfs-server`.

We can use the `showmount` command to get information on who is connected to a share, as well as what is being exported by a host (including ourselves). To see what is being exported, run:

```
[root@master ~]# showmount -e
Export list for master:
/opt/ohpc/pub 172.16.0.0/24
/home         172.16.0.0/24
```

To see clients mounting our shares, run `showmount` with no options:

```
[root@master ~]# showmount
Hosts on master:
```

Of course, we haven't mounted the share yet, so this is empty.

To mount an NFS share, we can either make an entry in `fstab` (we'll do this later), or mount by hand with (we can mount our own NFS volume to verify):

```
[root@master ~]# mount -t nfs 172.16.0.254:/home /mnt
[root@master ~]# grep nfs /proc/self/mounts
172.16.0.254:/home /mnt nfs4
rw,relatime,vers=4.1,rsize=1048576,wsize=1048576,namlen=255,hard,proto=tcp
,timeo=600,retrans=2,sec=sys,clientaddr=172.16.0.254,local_lock=none,addr=
172.16.0.254 0 0
[root@master ~]# ls /mnt
lowell
[root@master ~]# umount /mnt
```

This mounted `/home` on `172.16.0.254` onto our local directory of `/mnt`.

Our NFS shares are now ready.

Setup necessary services

We need to also set up the following services (we've set up all of these before, so we won't go into detail):

1. NTP:

    1. Remove all current lines starting with `server` from `/etc/ntp.conf`
    2. Add a line with `server 172.16.0.146`
    3. `systemctl enable ntpd`
    4. `systemctl start ntpd`
    5. Verify with `ntpq` then `lpeers`

2. TFTP:

    1. Edit `/etc/xinetd.d/tftp`, change `disabled = yes` to `disabled = no`
    2. `systemctl enable xinetd`
    3. `systemctl start xinetd`
    4. Verify this by downloading a file from `/var/lib/tftpboot` with the `tftp` client (`tftp` may need to be installed).

3. HTTPD (Apache):

    1. `systemctl enable httpd`
    2. `systemctl start httpd`
    3. Verify this by running `wget http://localhost/` and making sure you get an HTML response. This may be an error response (e.g. 403); that's fine, as long as it gets something (`wget` may need to be installed).

4. *Enable* (but don't start) DHCPD. DHCPD isn't configured yet; warewulf will do this for us later. But, we want to make sure it's enabled.

    1. `systemctl enable dhcpd`

Step 3: Building the BOS

In this step, we'll build the Base Operating System (BOS). This will be what our node image, called the VNFS, is made from. The steps are similar to what we did in the netboot guide, but warewulf provides some tools to help us out.

Building the initial chroot

We can build out our base image with a single warewulf command. Note: this will take a little while:

```
[root@master ~]# wwmkchroot centos-7 /opt/ohpc/admin/images/centos7
Loaded plugins: fastestmirror
Determining fastest mirrors
os-base                                | 3.6 kB  00:00:00
(1/2): os-base/x86_64/group_gz         | 166 kB  00:00:00
(2/2): os-base/x86_64/primary_db       | 6.0 MB  00:00:00
Resolving Dependencies
--> Running transaction check
... (lots of package installs)

Complete!
```

If we look in `/opt/ohpc/admin/images/centos7` we'll see there is a root filesystem there now:

```
[root@master ~]# ls /opt/ohpc/admin/images/centos7/
bin  boot  dev  etc  fastboot  home  lib  lib64  media  mnt  opt  proc
root  run  sbin  srv  sys  tmp  usr  var
```

That's it. We now have the base of the image. If we inspect the image, we'll notice that warewulf works hard to keep this image small:

```
[root@master ~]# du -sh /opt/ohpc/admin/images/centos7
454M    /opt/ohpc/admin/images/centos7
```

Warewulf does this by leaving out a lot of software you'd ordinarily want. For instance, even `yum` isn't installed in the BOS, because we anticipate that all installs would happen on the master when we create the image.

Adding software to the BOS

We need to add a little more software to our master. In general, we just add software to the BOS using `yum` with the `--installroot=` option.

```
[root@master ~]# yum -y --installroot=/opt/ohpc/admin/images/centos7
install ohpc-base-compute ntp kernel ipmitool lmod-ohpc
Loaded plugins: fastestmirror
Determining fastest mirrors
OpenHPC               | 2.9 kB  00:00:00
OpenHPC-updates       | 2.9 kB  00:00:00
base                  | 3.6 kB  00:00:00
epel                  | 3.6 kB  00:00:00
extras                | 2.9 kB  00:00:00
updates               | 2.9 kB  00:00:00
Resolving Dependencies
... (installs a bunch of packages)

Complete!
```

Adding our NFS mounts to the BOS image

We need to add `fstab` entries so that our NFS shares get mounted when our nodes boot. Open `/opt/ohpc/admin/images/centos7/etc/fstab` and add these lines to the bottom of the file:

```
172.16.0.254:/home /home nfs nfsvers=3,nodev,nosuid,noatime 0 0
172.16.0.254:/opt/ohpc/pub /opt/ohpc/pub nfs nfsvers=3,nodev,noatime 0 0
```

These specify our NFS mounts as default mounts to add at startup.

Similar changes can be made to other configuration files in the image (this is all we need for now), but we will see in a moment that warewulf has a feature that makes managing some of these files easier.

Enabling services in the BOS image

We need to enable the NTPD service *inside* our BOS. Fortunately, `systemctl` has a `--root=` option. We can do this with:

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

This will make sure NTPD starts on our compute nodes.

NTPD is the only service we need to enable for the moment.

Importing files into Warewulf images

Warewulf has a feature that allows the synchronization of files from the master into the image. This is especially useful for things like the `passwd` file that we would like to update easily when we want to add a user.

This is achieved with the `wwsh file import` command. Note that `wwsh` on our own would drop us into a special shell where we could enter various warewulf commands. Try `wwsh help` to get a sense of what is available.

Let's import some useful files:

```
[root@master ~]# wwsh file import /etc/passwd
[root@master ~]# wwsh file import /etc/group
[root@master ~]# wwsh file import /etc/shadow
```

We can see what we have imported with:

```
[root@master ~]# wwsh file list
group                   :  rw-r--r-- 1   root root              619
/etc/group
passwd                  :  rw-r--r-- 1   root root             1370
/etc/passwd
shadow                  :  rw-r----- 1   root root              890
/etc/shadow
```

Note that warewulf also keeps track of the correct ownership and permissions for the files.

If we ever change one of these files, we need to re-sync them with warewulf. We can do this with:

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

Our BOS is now complete.

Step 4: Assembling Bootstrap/VNFS & adding the first 3 nodes

Just like we did in the netboot example, warewulf needs to assemble a "bootstrap" (i.e. a kernel and initramfs) as well as make a useable version of the BOS image (we did this with `cpio`).

Assembling the VNFS

Warewulf calls the packaged image a VNFS (Virtual Node File System). A Warewulf cluster could potentially have many different images available, and it maintains a mapping that keeps track of which nodes should use which VNFSes.

The VNFS is stored in the `mariadb` database that keeps stores all of the warewulf configuration information. We create and import our BOS as a VNFS with a single command:

```
[root@master ~]# wwvnfs --chroot=/opt/ohpc/admin/images/centos7 centos7-
base
Creating VNFS image from centos7-base
Compiling hybridization link tree                           : 0.13 s
Building file list                                          : 0.40 s
Compiling and compressing VNFS                              : 11.88 s
Adding image to datastore                                   : 31.97 s
Wrote a new configuration file at: /etc/warewulf/vnfs/centos7-base.conf
Total elapsed time                                          : 44.37 s
```

Here, `--chroot` specified where or BOS lives. The last argument, `centos7-base` specifies the name of our VNFS. This argument is optional. Had we left it off, our VNFS gets named the same name as the directory, in our case, `centos7`.

We can use `wwsh` to show that we have successfully imported the image, and get some information about it:

```
[root@master ~]# wwsh vnfs list
VNFS NAME            SIZE (M)   ARCH       CHROOT LOCATION
centos7-base         261.2      x86_64     /opt/ohpc/admin/images/centos7
```

Warewulf now knows how to use the BOS image we built.

Assembling the bootstrap

We now need the intramfs and kernel that will want to use. Just like we did with the netboot tutorial, the initramfs may need some extra pieces like extra kernel modules. Warewulf uses the file `/etc/warewulf/bootstrap.conf` to configure this. We will want to add the following line to this file:

```
drivers += updates/kernel/
```

This can safely be added to the end of the file.

Warewulf gives us a single command to build and assemble the bootstrap:

```
[root@master ~]# wwbootstrap $(uname -r)
Number of drivers included in bootstrap: 541
Number of firmware images included in bootstrap: 96
Building and compressing bootstrap
Integrating the Warewulf bootstrap: 3.10.0-957.12.2.el7.x86_64
Including capability: provision-adhoc
Including capability: provision-files
Including capability: provision-selinux
Including capability: provision-vnfs
Including capability: setup-filesystems
Including capability: setup-ipmi
Including capability: transport-http
Compressing the initramfs
Locating the kernel object
Bootstrap image '3.10.0-957.12.2.el7.x86_64' is ready
Done.
```

The only argument we need to `wwbootstrap` is the kernel version we want. We've taken a shortcut here by using `uname -r` (try it, it prints our current kernel version) because we know that the kernel version we are running on the master is the same as the kernel version we want on our nodes. This is not necessarily always the case.

Like the VNFS, we can view information about our bootstrap with `wwsh`:

```
[root@master ~]# wwsh bootstrap list
BOOTSTRAP NAME             SIZE (M)     ARCH
3.10.0-957.12.2.el7.x86_64 29.1          x86_64
```

Adding the first three compute nodes

We already have the MAC address information for the first three compute nodes, so we can add them to the warewulf configuration. We will get the rest of the nodes added in the next step.

We add nodes with the `wwsh node new` command. It takes several arguments to fully specify a node:

```
[root@master ~]# wwsh node new n01 --ipaddr=172.16.0.1 --
netmask=255.255.255.0 --gateway=172.16.0.254 --hwaddr=18:66:da:ea:34:7c -D
eth0 -g compute
Are you sure you want to make the following 7 change(s) to 1 node(s):
     NEW: NODE                     = n01
     SET: eth0.HWADDR              = 18:66:da:ea:34:7c
     SET: eth0.IPADDR              = 172.16.0.1
     SET: eth0.NETMASK             = 255.255.255.0
     SET: eth0.GATEWAY             = 172.16.0.254
     SET: GROUPS                   = compute

Yes/No [no]> Yes
```

These arguments should be mostly self-explanatory. The `-g` option allows you to add the node to a "group." Warewulf groups allow you to apply certain kinds of configuration to all nodes in the group or act on multiple nodes at once.

Go ahead and add the other two nodes now in the same way.

Once you're done, you can list your nodes:

```
[root@master ~]# wwsh node list
NAME                GROUPS              IPADDR              HWADDR
===========================================================================
===
n01                 compute             172.16.0.1
18:66:da:ea:34:7c
n02                 compute             172.16.0.2
18:66:da:ea:23:d8
n03                 compute             172.16.0.3
18:66:da:ea:4a:c8
```

You can get more information by using `wwsh node print`:

```
[root@master ~]# wwsh node print n01
#### n01
#######################################################################
            n01: ID                   = 7
            n01: NAME                 = n01
            n01: NODENAME             = n01
            n01: ARCH                 = x86_64
            n01: CLUSTER              = UNDEF
            n01: DOMAIN               = UNDEF
            n01: GROUPS               = compute
            n01: ENABLED              = TRUE
            n01: eth0.HWADDR          = 18:66:da:ea:34:7c
            n01: eth0.HWPREFIX        = UNDEF
            n01: eth0.IPADDR          = 172.16.0.1
            n01: eth0.NETMASK         = 255.255.255.0
            n01: eth0.NETWORK         = UNDEF
            n01: eth0.GATEWAY         = 172.16.0.254
            n01: eth0.MTU             = UNDEF
```

We've added our nodes, but we haven't yet said which VNFS or bootstrap they should use. This is handled using the `wwsh provision` command. We can see what we have now:

```
[root@master ~]# wwsh provision list
NODE                VNFS            BOOTSTRAP             FILES
==========================================================================
===
n01                 UNDEF           UNDEF                 group,passwd
n02                 UNDEF           UNDEF
dynamic_hosts,grou...
n03                 UNDEF           UNDEF
dynamic_hosts,grou...
```

We can set their provision information and use the "compute" group we created to do them all at once:

```
[root@master ~]# wwsh provision set n[01-03] --vnfs=centos7-base --
bootstrap=$(uname -r) --console="ttyS0,115200" --
files=dynamic_hosts,passwd,group,shadow
Are you sure you want to make the following changes to 3 node(s):

     SET: BOOTSTRAP             = 3.10.0-957.12.2.el7.x86_64
     SET: VNFS                  = centos7-base
     SET: FILES                 = dynamic_hosts,passwd,group,shadow
     SET: CONSOLE               = ttyS0,115200

Yes/No> Yes
```

Notice that we set the VNFS and bootstrap, but we also set `--console` and `--files`. `--console` sets our kernel command line parameters to include the serial console. `--files` attaches our imported files to these nodes.

Let's get the provision list again:

```
[root@master ~]# wwsh provision list
NODE                VNFS            BOOTSTRAP             FILES
==========================================================================
======
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...
```

That looks better.

Let's get a little more detail with `wwsh provision print`:

```
[root@master ~]# wwsh provision print n01
#### n01
######################################################################
            n01: BOOTSTRAP        = 3.10.0-957.12.2.el7.x86_64
            n01: VNFS             = centos7-base
            n01: FILES            = dynamic_hosts,group,passwd,shadow
            n01: PRESHELL         = FALSE
            n01: POSTSHELL        = FALSE
            n01: CONSOLE          = ttyS0,115200
            n01: PXELINUX         = UNDEF
            n01: SELINUX          = DISABLED
            n01: KARGS            = "net.ifnames=0 biosdevname=0 quiet"
            n01: BOOTLOCAL        = FALSE
```

This shows us our full file list and the console setting.

Note the file "dynamic_hosts". This is a special file that warewulf will auto-generate for us. It will create an `/etc/hosts` file that lists entries for all known nodes automatically. It uses `/etc/hosts` on the master as a base template to build off of.

Getting access to the BMCs

You'll note that the warewulf config did not include the BMCs. Warewulf will be generating our `dhcpd.conf` file for us, so we need a place to put some static entries. Fortunately, warewulf has a mechanism for this. We can modify the file `/etc/warewulf/dhcpd-template.conf` to add our BMC entries. We need to add them *before* the line that reads `# Node entries will follow below`. Our file should look like:

```
...

host n01-bmc { hardware ethernet 18:66:da:68:3e:40; fixed-address
172.16.0.101; }
host n02-bmc { hardware ethernet 18:66:da:68:4b:14; fixed-address
172.16.0.102; }
host n03-bmc { hardware ethernet 18:66:da:68:41:3a; fixed-address
172.16.0.103; }


# Node entries will follow below
```

Finishing it up

To generate the `dhcpd` config, we run:

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

We'll need to do this any time we make changes to `dhcpd-template.conf`. Let's take a look at `/etc/dhcp/dhcpd.conf` now. We'll see things that look like this:

```
host n01-bmc { hardware ethernet 18:66:da:68:3e:40; fixed-address
172.16.0.101; }
host n02-bmc { hardware ethernet 18:66:da:68:4b:14; fixed-address
172.16.0.102; }
host n03-bmc { hardware ethernet 18:66:da:68:41:3a; fixed-address
172.16.0.103; }

# Node entries will follow below


group {
   # Evaluating Warewulf node: n01 (DB ID:7)
   # Adding host entry for n01-eth0
   host n01-eth0 {
      option host-name n01;
      option routers 172.16.0.254;
      hardware ethernet 18:66:da:ea:34:7c;
      fixed-address 172.16.0.1;
      next-server 172.16.0.254;
   }
```

While a bit more complicated than what we did in the netboot tutorial, this should look very familiar.

Just like we did in the netboot tutorial, verify the config with `dhcpd -t`, and (re)start the dhcpd service:

```
[root@master warewulf]# dhcpd -t
Internet Systems Consortium DHCP Server 4.2.5
Copyright 2004-2013 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/
Not searching LDAP since ldap-server, ldap-port and ldap-base-dn were not
specified in the config file
```

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

```
[root@master warewulf]# systemctl status dhcpd
● dhcpd.service - DHCPv4 Server Daemon
   Loaded: loaded (/usr/lib/systemd/system/dhcpd.service; disabled; vendor
preset: disabled)
   Active: active (running) since Sat 2019-06-08 13:54:43 MDT; 3s ago
     Docs: man:dhcpd(8)
           man:dhcpd.conf(5)
 Main PID: 31440 (dhcpd)
```

```
   Status: "Dispatching packets..."
   CGroup: /system.slice/dhcpd.service
           └─31440 /usr/sbin/dhcpd -f -cf /etc/dhcp/dhcpd.conf -user dhcpd
-group dhcpd --no-pid

Jun 08 13:54:43 master systemd[1]: Started DHCPv4 Server Daemon.
... (some log entries)
```

To generate the `pxelinux.cfg` configs, we run:

```
[root@master warewulf]# wwsh pxe update
```

Booting our first three nodes

At this point (it may take a bit for DHCP to assign addresses), we should be able to reboot our nodes. Let's just do the first one to start:

```
[root@master warewulf]# ipmitool -I lanplus -H 172.16.0.101 -U admin -P
admin shell
ipmitool> power status
Chassis Power is on
ipmitool> power off
Chassis Power Control: Down/Off
ipmitool> power status
Chassis Power is off
ipmitool> chassis bootdev pxe options=persistent
Set Boot Device to pxe
ipmitool> power on
Chassis Power Control: Up/On
ipmitool> sol activate
[SOL Session operational.  Use ~? for help]
...
```

We should be able to watch the node fully boot.

We'll know warewulf is working when we see a screen like:

```
Now Booting Warewulf...

Setting the hostname (n01):
OK
Loading drivers: uhci-hcd ohci-hcd ehci-hcd whci-hcd isp116x-hcd isp1362-
hcd OKci-hcd sl811-hcd sd_mod
Detecting hardware: ahci ahci tg3 tg3 tg3 tg3 megaraid_sas mlx5_core
OK
Bringing up local loopback network:
OK
```

```
Checking for network device: eth0 (eth0)
OK
Configuring eth0 (eth0) statically: (172.16.0.1/255.255.255.0)
OK
Configuring gateway: (172.16.0.254)
OK
Creating network initialization files: (eth0)
OK
Trying to reach the master node at 172.16.0.254 .
OK
Probing for HW Address: (18:66:da:ea:34:7c)
OK
Starting syslogd:
OK
Getting base node configuration:
OK
Starting the provision handler:
 * adhoc-pre
OK
 * ipmiconfig Auto configuration not activated
SKIPPED
 * filesystems
RUNNING
   * mounting /
OK
 * filesystems
OK
 * getvnfs
RUNNING
   * fetching centos7-base (ID:4)
```

After a couple of minutes we should get a login prompt:

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

n01 login:
```

To exit the sol session, hit `<enter> ~ ~ .`

Now:

1. verify that you can SSH to n01 from the master
2. boot the other two nodes!

Step 5: Discovering your nodes (requires physical access)

We don't know the MAC addresses of nodes 3-10. Warewulf provides a tool we can use to discover these and add them to the node list all in one step called `wwnodescan`. We will have to have physical access to our systems to do this since we also don't have their BMCs configured.

`wwnodescan` works by listening for unknown MAC addresses and automatically adding a new node for each one it sees.

Because we also have the BMCs on the same network ***we must disconnect the BMCs (purple ethernet)*** through this procedure. We'll re-connect them when we're done. If we don't do this, warewulf will try to add the BMC addresses as new nodes, and we don't want this.

Our general procedure is:

1. start `wwnodescan` with appropriate options
2. power on a node
3. wait for warewulf to register it
4. power on the next node
5. repeat until all nodes are added

This process is a little tedious, but it's so much easier than collecting MAC addresses by hand!

Before we go to the server room, we'll want to set up what the default configuration for a new node is. We do this by adding a special node called "DEFAULT":

```
[root@master etc]# wwsh node new DEFAULT --groups=compute
Are you sure you want to make the following 3 change(s) to 1 node(s):

     NEW: NODE                 = DEFAULT
     SET: GROUPS               = compute

Yes/No [no]> Yes
```

Now we can set what the default provision settings are by "provisioning" the DEFAULT node:

```
[root@master etc]# wwsh provision set DEFAULT --vnfs=centos7-base --
bootstrap=$(uname -r) --files=dynamic_hosts,passwd,shadow,group --
console=ttyS0,115200
Are you sure you want to make the following changes to 1 node(s):

     SET: BOOTSTRAP            = 3.10.0-957.12.2.el7.x86_64
     SET: VNFS                 = centos7-base
     SET: FILES                = dynamic_hosts,passwd,shadow,group
     SET: CONSOLE              = ttyS0,115200

Yes/No> Yes
```

We can verify with:

```
[root@master ~]# wwsh provision print DEFAULT
#### DEFAULT
##################################################################
```

```
          DEFAULT: BOOTSTRAP           = 3.10.0-957.12.2.el7.x86_64
          DEFAULT: VNFS                = centos7-base
          DEFAULT: FILES               = dynamic_hosts,group,passwd,shadow
          DEFAULT: PRESHELL            = FALSE
          DEFAULT: POSTSHELL           = FALSE
          DEFAULT: CONSOLE             = ttyS0,115200
          DEFAULT: PXELINUX            = UNDEF
          DEFAULT: SELINUX             = DISABLED
          DEFAULT: KARGS               = "net.ifnames=0 biosdevname=0 quiet"
          DEFAULT: BOOTLOCAL           = FALSE
```

Here are the steps in detail (performed in the server room):

1. Make sure all of nodes `n[04-10]` are powered off

2. Disconnect the BMCs (purple ethernet)

3. Start `wwnodescan`. We want to give it options to set everything we need up from the beginning. Because we set defaults, we don't need to give provisioning options to `wwnodescan`. Our command looks like (it's recommended to run this inside a `tmux` that can be attached to):

   ```
   [root@master ~]# wwnodescan -v --ipaddr=172.16.0.4 --
   netmask=255.255.255.0 --listen=em1 n[04-10]
   Successfully connected to database!
    Assuming the nodes are booting over eth0Listening on em1 for DHCP
   requests
    Scanning for node(s) (Ctrl-C to exit)...
   ```

4. Power on node 4 (with the power button)

5. Wait a couple of minutes for the node to try to PXE boot. When it tries, you'll see warewulf add it to the list:

   ```
   WARNING:  Auto-detected "n04", cluster "", domain ""
   Loading event handler: Warewulf::Event::Bootstrap
   Loading event handler: Warewulf::Event::DefaultNode
   Loading event handler: Warewulf::Event::DefaultProvisionNode
   Loading event handler: Warewulf::Event::Dhcp
   Loading event handler: Warewulf::Event::DynamicHosts
   Loading event handler: Warewulf::Event::NewObject
   Loading event handler: Warewulf::Event::ProvisionFileDelete
   Loading event handler: Warewulf::Event::Pxe
   Loading event handler: Warewulf::Event::UniqueNode
   Building default configuration for new node(s)
   Building default configuration for new provision node(s)
   Looking for duplicate node(s)
   Building default configuration for new object(s)
   Writing DHCP configuration
   ```

```
Building iPXE configuration for: n04/18:66:da:68:3f:d0
Added to data store:  n04:  172.16.0.4/255.255.255.0/18:66:da:68:3f:d0
```

If this doesn't come up, your node may not be set to PXE boot. You'll have to connect the KVM to fix this (by hitting `<F12>` on boot).

6. Repeat with the rest of the nodes

Once you're done, you should have all of your nodes booted.

Note: you ***won't*** have BMC access for nodes 04 - 10. We'll use one of our tools to get those next.