Netboot cluster guide

Overview

This guide contains the steps necessary to get the master to manually PXE boot a single stateless node. This guide is meant as a side reference, and the class should follow the presentation slides.

Assumptions

This guide assumes that the following steps have already been completed:

- CentOS 7 has been installed on the master and updated.
- Interface em2 has been configured as the uplink.
- Interface em1 has been statically configured with IP 172.16.0.254/24 and connected to the cluster switch.
- The MAC address for the first compute node's iDRAC has been collected, and the iDRAC login set to `admin`/`admin`.

Step 0: Preliminaries

The following steps will get the system prepared for our pxeboot setup.

1. There is supplementary content on the google drive. Copy the file `netboot.tar.gz` to your master. Extract it in `/root`:

   ```
   # tar zxvf netboot.tar.gz -C /root
   ```

   This will create the directory `/root/netboot` that we will be using for several steps, and populate it with some files we'll need.

2. Disable selinux:

```
# setenforce 0
# vi /etc/sysconfig/selinux
```

Change the line:

`SELINUX=enforcing`

to

`SELINUX=disabled`

3. Disable firewall (firewalld):

```
# systemctl stop firewalld
# systemctl disable firewalld
```

4. Install some useful utilities:

```
# yum install tmux lsof links
```

Step 1: Configuring DHCP

The heart of the PXEBoot process is controlled through DHCP. DHCP not only has the responsibility of handing out IP addresses, but also instructing the node as to where it can find a pxe executable (pxelinux.0) file that will launch the PXE process.

1. Install dhcp:

```
yum install dhcp
```

2. Configure dhcp. DHCPd is controlled through the file `/etc/dhcp/dhcpd.conf`. Open this file and create the following contents (a copy of this file can be found at `/root/netboot/master/etc/dhcp/dhcpd.conf`):

```
default-lease-time 7200;
max-lease-time 7200;

subnet 172.16.0.0 netmask 255.255.255.0 {
   option domain-name "blatz";
   option broadcast-address 172.16.0.255;
}

host node0 {
   hardware ethernet 00:11:22:33:44:55;
```

```
    fixed-address 172.16.0.1;
    next-server 172.16.0.254;
    filename "pxelinux.0";
}

host node0-bmc {
    hardware ethernet 00:11:22:33:44:56;
    fixed-address 172.16.0.101;
}
```

We don't know the real MAC address ("hardware ethernet") for the node yet, but we do know the BMC MAC address. Replace the dummy mac address (`00:11:22:33:44:56`) with the real MAC address for the BMC of the first node. We will later use the BMC to get the MAC address of this node.

Pay special attention to the lines `next-server` and `filename`. `next-server` is the IP address of a server (the master) that will provide a TFTP service where a PXE file can be downloaded. `filename` is the name of the file that should be downloaded.

Save the file and exit the editor.

If you copied the example file into place, make sure the ownership of the file is correct. It should be owned by root:root.

3. Validate dhcpd.conf syntax. Any time we configure a service, it's a good idea to verify our configuration. Most services have the ability to confirm that a config is valid. To validate our dhcp config, run:

```
# dhcpd -t
```

If everything is configured correctly, you should get an output like this:

```
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
```

If something is wrong with the file syntax, you will get an output something like this:

```
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/
/etc/dhcp/dhcpd.conf line 6: semicolon expected.
subnet
 ^
```

```
Configuration file errors encountered -- exiting
...
```

4. Enable/start dhcpd.

```
# systemctl enable dhcpd
# systemctl start dhcpd
```

5. Verify dhcp status. We can make sure dhcpd start and is running with:

```
# systemctl status dhcpd
```

We should see:

```
● dhcpd.service - DHCPv4 Server Daemon
Loaded: loaded (/usr/lib/systemd/system/dhcpd.service; enabled; vendor
preset: disabled)
Active: active (running) since Wed 2018-04-04 18:31:35 UTC; 7min ago
...
```

6. Watch the logs to see that our BMC gets its IP address. We can watch the logs with `journalctl`. We should see that our BMC completes an IPMI handshake. If other nodes are connected, we may see that they attempt to get leases but get a "no leases available" error. This is how it should work since we have not configured those BMCs in the `dhcpd.conf` file. To watch the logs:

```
# journalctl _SYSTEMD_UNIT=dhcpd.service -f
```

We should eventually see a handshake like this:

```
Apr 02 17:20:52 pxe-master.blah dhcpd[19694]: DHCPDISCOVER from
00:11:22:33:44:01 via em1
Apr 02 17:20:52 pxe-master.blah dhcpd[19694]: DHCPOFFER on
176.16.0.101 to 00:11:22:33:44:01 via em1
Apr 02 17:20:53 pxe-master.blah dhcpd[19694]: DHCPREQUEST for
176.16.0.101 (192.168.33.2) from 00:11:22:33:44:01 via em1
Apr 02 17:20:53 pxe-master.blah dhcpd[19694]: DHCPACK on 176.16.0.101
to 00:11:22:33:44:01 via em1
```

Once we see this handshake has completed, we should be able to ping the BMC:

```
# ping 172.16.0.101
PING localhost (172.16.0.101): 56 data bytes
64 bytes from 172.16.0.101: icmp_seq=0 ttl=64 time=0.052 ms
64 bytes from 172.16.0.101: icmp_seq=1 ttl=64 time=0.083 ms
```

DHCP is now functional, and we have an IP address for our BMC.

Step 2: IPMI control of the compute node

IPMI (Intelligent Platform Management Interface) is a fairly universal protocol for interacting with BMCs from various vendors. It can be used to control power, attach to serial consoles, configure some BIOS settings (like boot order), and discover information about the server. We will use it to: collect MAC addresses, control power, and attach to the serial console.

1. Install ipmitool:

```
# yum install ipmitool
```

2. Get an IPMI shell. The `ipmitool` command will allow us to access the features of the BMC available through the IPMI protocol. The general syntax of an IPMI command is:

```
# ipmitool -I lanplus -H <IP address> -U <user> -P <password> <IPMI
command>
```

The "lanplus" option specifies a version of the protocol that we will need to use with these Dell servers. There is a special `shell` IPMI command that will drop us in a shell, where we can run various IPMI commands.

```
# ipmitool -I lanplus -H 172.16.0.101 -U admin -P admin shell
```

3. Let's get the MAC addresses of the node:

```
> delloem mac
```

This will give us MAC addresses for all of the ports on this node. We need the MAC of the first interface. Write it down (or copy/paste).

4. *IMPORTANT:* Now that we have the MAC address, we should update the "node0" section of `/etc/dhcp/dhcpd.conf` before we forget. Edit the file and put this MAC address as "hardware ethernet" for the node0 section. Perform the config check, and restart the dhcpd service:

```
# systemctl restart dhcpd
```

Verify status and look at the log to make sure it is working correctly. We are now done with the dhcpd config.

5. Other useful IPMI commands for later reference:
    - `power on/off/cycle` - turn power on, off or power cycle the node
    - `power status` - get current power on/off status
    - `sol activate` - connect to the serial console. To exit the serial console, use the key sequence `<enter>~.` (period is part of the sequence).
    - `lan print 1` - view the network configuration of the BMC
    - `chassis bootdev pxe/disk/bios` - set the first boot device to be PXE, disk, or into the BIOS.

Step 3: Configure TFTP

Recall that the "next-server" line of the `dhcpd.conf` file references a server (master) where a TFTP service can be found. We need to set up that service now. The standard TFTP service is managed through the `xinetd` service, so we will need to install it as well, and enable/start it. We also need to let xinetd know that it should start TFTP.

1. Install xinetd and tftp (and the tftp client for testing):

```
# yum install xinetd tftp tftp-server
```

2. Tell xinetd to start tftp. Edit the file `/etc/xinetd.d/tftp` and set the line from `disable = yes` to `disable = no`. Notice the line `server_args = -s /var/lib/tftpboot`. This line tells where TFTP files will be served from. The default is `/var/lib/tftpboot`, and this is what we will use in this tutorial.

3. Start/enable xinetd

```
# systemctl enable xinetd
# systemctl start xinetd
```

4. Verify and test tftp. We should now see that the xinetd service is active if we run `systemctl status xinetd`. We can also make sure it is listening for tftp requests. Run:

```
# lsof -i
COMMAND   PID    USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
...
xinetd    880    root    5u  IPv4  16940      0t0  UDP *:tftp
...
```

We should see a line like the above showing the xinetd is, in fact, listening on the UDP tftp port.

Finally, we can test that tftp is actually working. For this, we need a file in `/var/lib/tftpboot`. In principle, any file would do.

```
# cp -v /etc/redhat-release /var/lib/tftpboot/
‘/etc/redhat-release’ -> ‘/var/lib/tftpboot/redhat-release’
# tftp 172.16.0.254
tftp> get redhat-release
tftp> quit
# diff /root/redhat-release /etc/redhat-release
```

We should be able to complete these steps, and see that `diff` returns no differences between the files.

Step 4: Configure pxelinux

When our node gets its DHCP offer, it will get the `next-server``` and ```filename``` options along with its IP address. Recall that the ```filename` we set was "pxelinux.0". This is a special pxe "kernel" that can serve a purpose similar to grub on a diskfully installed system. It can present a menu of options, download more files from TFTP among other things.

In our setup, we will just have one PXE menu option. It will instruct pxelinux to download a kernel and an initramfs and boot the kernel with some specified options.

1. Install pxelinux:

   ```
   # yum install syslinux-tftpboot
   ```

   If you look in `/var/lib/tftpboot``` there should now be quite a few files, including ```pxelinux.0`.

2. Write the pxelinux configuration. `pxelinux.0` will look, through TFTP, in the directory `pxelinux.cfg`. It will try a sequence of files that map to the node's MAC address or IP address that could load node-specific PXE configurations. If it doesn't find any of these files, it will look for a file called `default`. We will only use the `default` file.

   First we need to create the directory:

   ```
   # mkdir /var/lib/tftpboot/pxelinux.cfg
   ```

   Now we need to create the file `/var/lib/tftpboot/pxelinux.cfg/default`, and place the following contents in the file:

   ```
   ```

```
DEFAULT menu.c32
PROMPT 0
TIMEOUT 50
SERIAL 0 115200 0
MENU TITLE Main Menu

LABEL node
   MENU LABEL Boot x86_64 Compute Node (diskless)
   KERNEL vmlinuz
   APPEND initrd=initramfs.img img=http://172.16.0.254/image.cpio
console=tty0 console=ttyS0,115200
```

The `KERNEL` line says that the kernel to boot is in a file called `vmlinuz`. We will eventually need that file under `/var/lib/tftpboot` (Step 7).

The `APPEND` line contains kernel parameters. Of special note, the `initrd` paramters says that we need an initramfs file at `/var/lib/tftpboot/initramfs.img`. Also, the `img` parameter is a custom parameter that we will use in our boot process later to know where to download our compute node image. Note: we will need to have an http service for this (Step 6).

Finally, the `SERIAL` line and the `console=ttyS0...` parameters make sure both pxelinux and the kernel know that we want to use a serial port as our console (through the BMC).

3. Testing it out: We don't have everything in place yet to actually get our node to boot, but at this point we can verify that pxelinux works. To do this, I recommend going into split window in tmux (key sequence `<ctrl>-b "`, use `<ctrl>-b <up>/<down>` to switch windows). In one window, follow the dhcpd logs with `# journalctl _SYSTEMD_UNIT=dhcpd.service -f`, and in the other window start an IPMI shell as we did in Step 2. Power on the node (`ipmi> power on`), wait a few seconds for it to initialize, then attach to the serial console (`ipmi> sol activate`). You should see the node boot through the BIOS/UEFI, then negotiate DHCP, which will appear in the logs. Then you should see the PXE menu we specified in the serial console. It won't boot beyond this point, because we need the kernel, initramfs and compute image still.

Step 5: Create our compute image

Next, we need to create the install image that our node is going to run. We are going to make a very minimal image that would be a bit useless for production, but illustrates the basic steps of how this is done.

1. First, we need a directory structure where we will put the image contents. Make the directory `/opt/img/root`. This will be the root of our compute node's filesystem. Under `/opt/img/root` create an `etc` directory. There is a `yum.conf` file in the `/root/netboot` directory. Copy that file into `/opt/img/root/etc`.

2. Now we can use the `--installroot` feature of `yum` to install the set of packages required for a "Minimal Install" of CentOS. We need to use the repositories defined in our yum.conf file we just copied in. To do that we also need to make sure the default repositories on the master are disabled. The repos in the yum.conf are conveniently named [img-base] and [img-updates] (it's not a bad idea to take a look at the contents of the yum.conf). This can all be accomplished with the following yum command (***this will take several minutes to complete***):

```
# yum --installroot=/opt/img/root --disablerepo='*' --enablerepo='img-*' groupinstall "Minimal Install"
```

Once this has finished, you should have a fairly complete looking Linux install in `/opt/img/root`.

3. There are several configuration files that need to be set up in our image file. In the interest of brevity, I have pre-made these files so they can be copied into the image. We first make sure the file ownership is correct before moving them into place.

```
# chown -R root:root /root/netboot/root
# cp -av /root/netboot/root/* /opt/img/root/
```

There are a couple of files of particular interest:

- `/etc/ssh/sshd_config` - contains the configuration for the sshd service that will run on our node. Specifically, this is altered to allow root to log in with a public key.
- `/init` - *IMPORTANT!* CentOS 7 systems use systemd as their init process; however, systemd requires more work to get to start correctly than we are going to bother with here. This `/init` is a short script that will handle the couple of basic bootup tasks we need by hand, such as starting sshd and agetty (login services). Take a look inside this script. Once the image is loaded, it will run this script as Process ID (PID) 1.

4. To set up public key authentication for our node, we need to copy a public key from the master to the image. First, generate a public key on the master (if you haven't already):

```
# ssh-keygen
```

And just hit Enter for all of the prompts. This will create `/root/.ssh/{id_rsa,id_rsa.pub}` on the master. Now, we need to copy this file to a special location in the image: `/opt/img/root/root/.ssh/authorized_keys`, and make sure all permissions are correct:

```
# mkdir /opt/img/root/root/.ssh
# chmod 700 /opt/img/root/root/.ssh
# cp -v /root/.ssh/id_rsa.pub /opt/img/root/root/.ssh/authorized_keys
# chmod 600 /opt/img/root/root/.ssh/authorized_keys
```

IMPORTANT! if permissions are incorrect on these files and directories, public key authentication will not work.

Our image is now ready. All that remains is to package it up and make it available through http, which we will do in the next step.

Step 6: Setup httpd and publish image

We will not require much from http. It just needs to send a single file. We will use the default configuration, which makes things easy.

1. Install, enable and start httpd:

```
# yum install httpd
# systemctl enable httpd
# systemctl start httpd
```

2. Verify the httpd service. We can use the tools we used before to verify services. `systemctl status httpd` should return an "active" status. `lsof -i` should show that httpd is listening on TCP on "*:http". Finally, Apache comes with a default page when installed and started. We can load that page in text mode with: `links http://172.16.0.254`. This should load a "welcome to Apache" page.

3. Bundling & publishing our compute image: Apache's default DocumentRoot is `/var/www/html`. We want to bundle up a version of our image in `/opt/img/root` that can be easily downloaded, and place it in that directory so Apache will serve the file. We will use the `cpio` file format to store our image. In a larger cluster, we might also compress it. The following commands will publish our image (note: it's important to start in the root directory of the image):

```
# cd /opt/img/root
# find . | cpio -oc > /var/www/html/image.cpio
```

   Cpio will print a message saying how many bytes it wrote. You should be able to see the file at `/var/www/html/image.cpio`. As a test, `# wget http://172.16.0.254/image.cpio` should download the file.

Step 7: Bootstrap (kernel & initramfs)

The final piece to get our node to boot is the most complicated. It involves some custom scripting to build an initramfs that will:

1. Load network device modules
2. Get an IP from the DHCP server
3. Setup a ramfs mount for our real root
4. Download and extract the compute image
5. `switch_root` to the new image and start the `init` in the image

...all while doing various system tasks, like mounting `/proc`.

Rather than have you actually go through this process, which would be a whole session, we have created a script that will generate the initramfs for you and copy a `vmlinuz` and `initramfs.img` file into `/var/lib/tftpboot` where they can be downloaded through TFTP.

To use this script:

```
# cd /root/netboot
# sh bootstrap.sh
```

Verify that the vmlinuz and initramfs.img files are in /var/lib/tftpboot.

Finally, we should at least get a sense of how things are done. Take a look at the bootstrap.sh script to see what it does.

To illustrate what boostrap made for us, do the following:

```
# cd /var/lib/tftpboot
# mkdir initramfs
# cd initramfs
# cpio -iv < ../initramfs.img
```

This will extract the initramfs image into the directory `/var/lib/tftpboot/initramfs`. There are some important files here:

- `/init` - this process is the first to launch. It manages everything else and ultimately performs the switch_root.
- `/load_img.sh` - this helper script is what downloads and extracts the compute image. It also extracts the `img=` kernel command line argument to know where to get the image.
- `/modules.txt` - this file lists modules that need to be loaded at this stage (mostly network device drivers).

There are a couple of other small helper scripts in this image as well. You are strongly encouraged to read through these files, especially `/init` and `/load_img.sh` to see how this process works.

Step 8: Boot the compute node

All of the pieces should be in place now! Go ahead and attach to the serial console and power cycle the node. You should see it go through these stages:

1. BIOS/UEFI init
2. pxelinux menu (times out in 5 seconds)
3. the kernel will load
4. the initramfs will load, and image will download
5. it will switch to the real root, load a couple of services, and give a login prompt

At this point, the node is up. We should be able to ssh to it using: `ssh 172.16.0.1`, and (after accepting the host key) it should public key authenticate.

That's it!