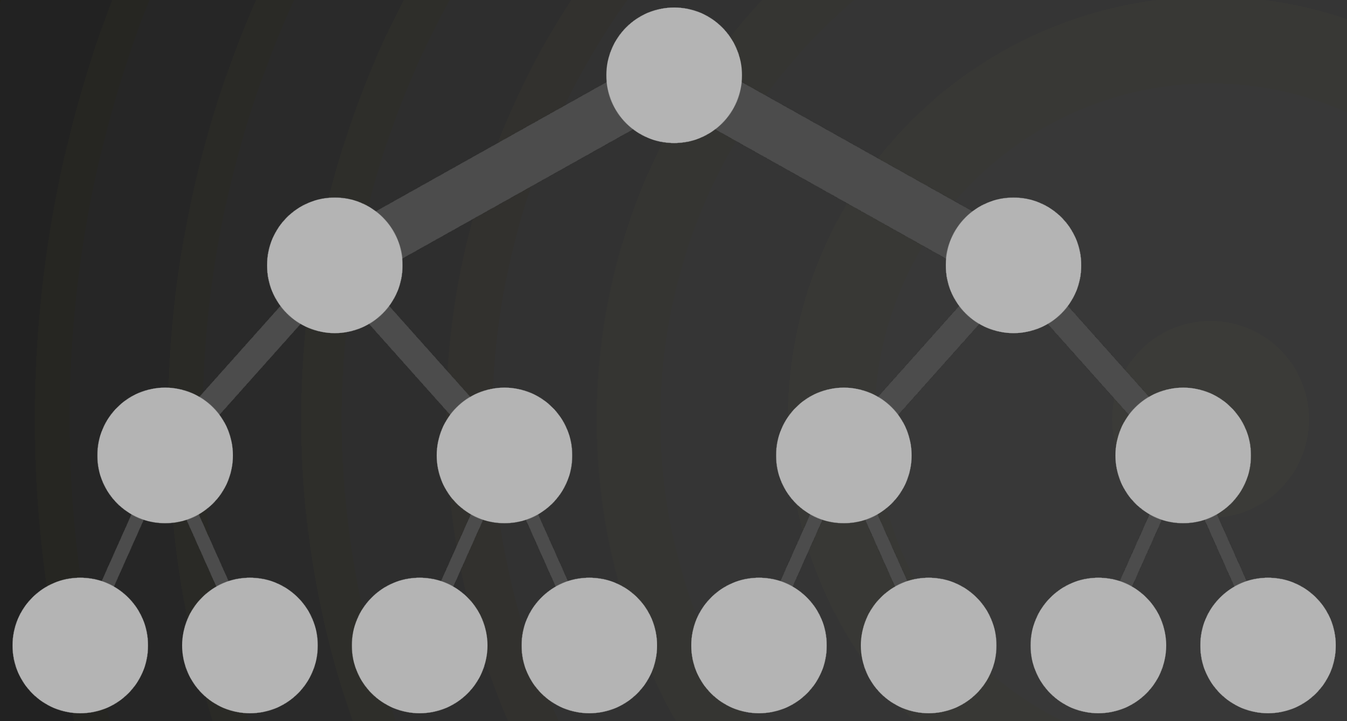
# Los Alamos

## NATIONAL LABORATORY

EST. 1943

NNSA
National Nuclear Security Administration
Managed by Triad National Security, LLC for the U.S. Department of Energy's NNSA



Los Alamos
NATIONAL LABORATORY
EST. 1943

Delivering science and technology
to protect our nation
and promote world stability

# Netboot Practicum

## Manually building a stateless cluster

Presented by CSCNSI

# Goals of this section

- Deploy a cluster node without a cluster provisioner
- Build a simple cluster compute node image manually
- Explore the internals of how clusters boot
- Learn basic configuration of the core cluster booting services
- Use some tools for verifying and diagnosing cluster booting services

# Why build a cluster by hand?

- Most cluster provisioners use similar methods “under the hood”
- By learning how to build a cluster by hand, we learn how these provisioners work at their foundation
- Gives us an opportunity to learn how individual services can be deployed and how to troubleshoot them
- Provides an appreciation for tools that do this work for us!

# Overview

# High-level steps to build a cluster

- Physically rack & cable (already done)
- Install the operating system on the master (already done)
- Configure necessary services on the master (`DHCP`, `TFTP`,…)
- Build OS images that the cluster will run
- Provide a set of tools to get those images running on nodes (typically in `initramfs`)
- The following pieces will not be implemented by our DIY cluster. We will implement them in later steps:
- Manage configuration across the cluster (*Not today*)
- Provide a job scheduler/resource manager to run tasks on the cluster (*Not today*)

# Stateful vs. Stateless clusters

- **Stateful cluster** — each node has a physical disk. The local disks have at least a partial operating install. Typically, these use some mechanism to deploy the stateful image automatically.
  - fewer challenges booting
  - less RAM consumption
- **Stateless cluster** — the nodes temporarily store the OS image in RAM. The OS image is deployed to the cluster node at each node reboot.
  - keeps system state consistent
  - easy to deploy new images
  - no disks to fail on nodes
- Both have advantages and disadvantages. Our test cluster will be *stateless*.

# Stateless cluster init process

- Node powers on
- (node) requests `DHCP` lease
- (master) provides `IP` and where to get a `PXE` boot file
- (node) gets `PXE` boot file(s), executes
- (master) provides a `PXE` config, specifying where the kernel & `initramfs` are
- (node) downloads `kernel` & `initramfs` and boots into “stage 1”
- (master) provides full OS image to node
- (node) extracts image into a ramdisk
- (node) uses cluster image to boot into “stage 2”
- (node) starts services, etc.
- Node is up

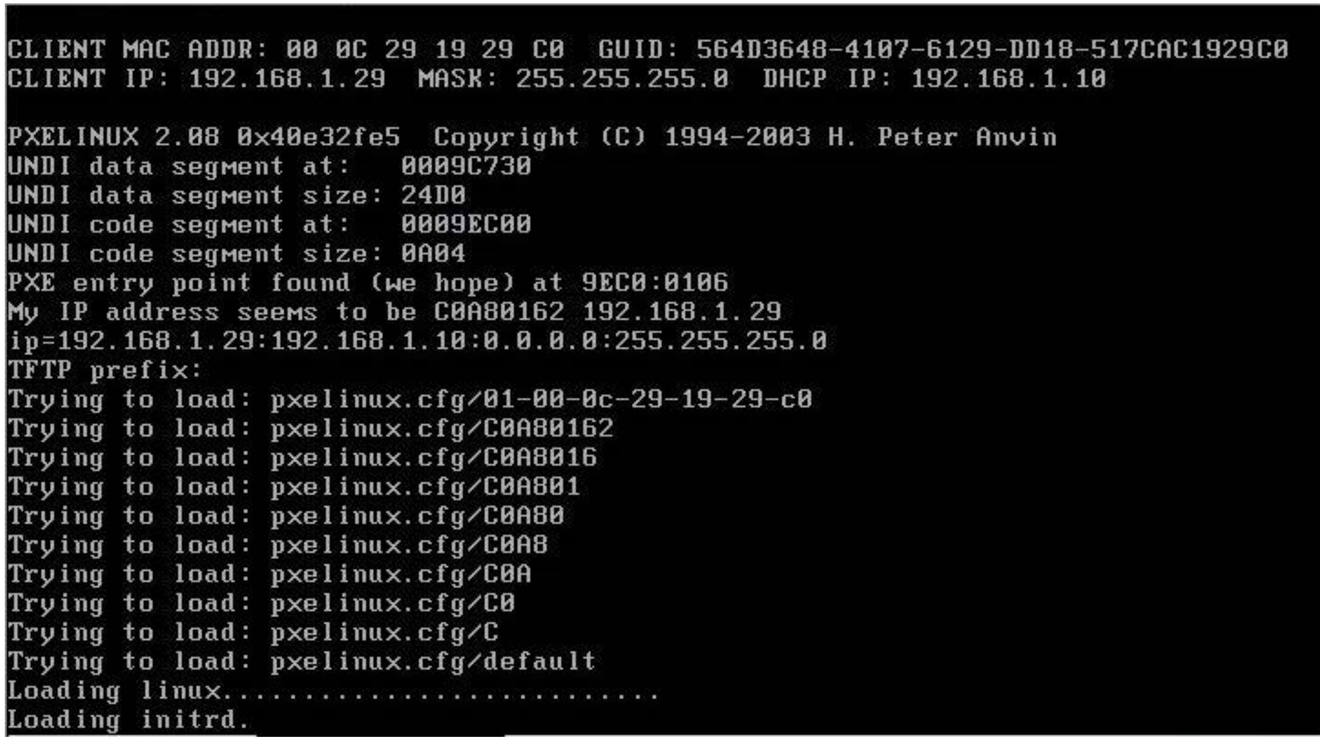
# Starting assumptions

- The following steps should 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.

# Netboot services

# PXE

- “Preboot Execution Environment”
- A standard supported by (some) network cards
- Network card can load a special executable that can be used to provision an un-provisioned system over the network
  - Configuration information is provided by DHCP
  - Executable & further config are fetched from TFTP
    - Executable can act as a bootloader for a boot/install process

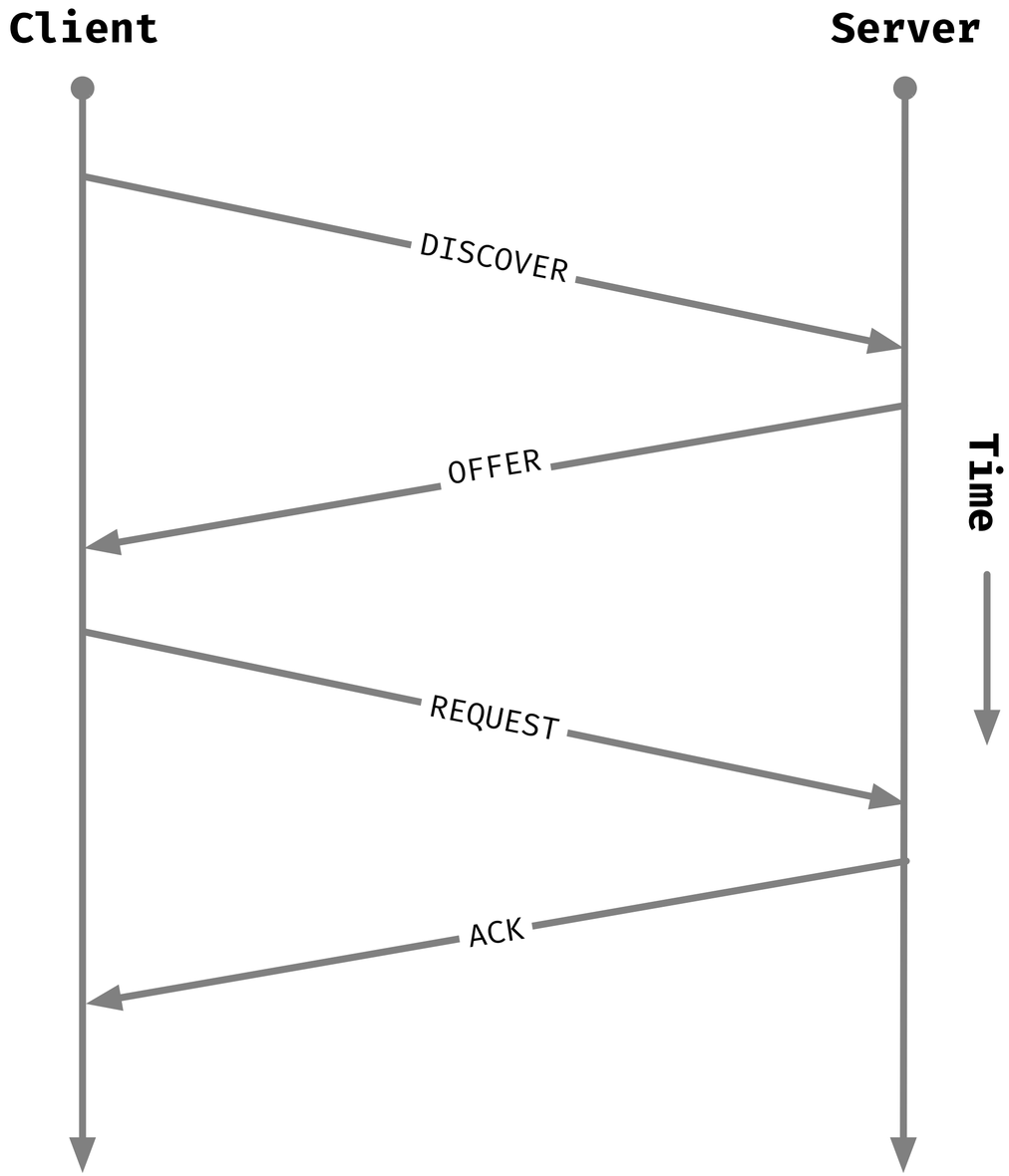


# DHCP

- Dynamic Host Control Protocol
- Can assign IP addresses and network information to hosts on a network:
  - Can *dynamically* assign IP addresses on demand
  - Or, can assign fixed IP addresses to known clients
- Can also pass a lot of extra configuration information with the IP address
- Special options used for PXE booting
  - “next-server” says there’s a file that should be grabbed at a TFTP server
  - “filename” says what filename to grab

# DHCP: The handshake

- A successful handshake:
  - DISCOVER—the client sends out a request to the broadcast address asking if anyone can tell it how to setup its network.
  - OFFER—A server offers an IP address (and maybe other config too).
  - REQUEST—The client tells the server it would like the configuration it was offered.
  - ACK—The server acknowledges the request, and the client takes its configuration.
- Other packets
  - NACK—Server rejects a request
  - INFORM-–Client requests more info (e.g. proxy)
  - RELEASE—Client is giving up address



# TFTP

- Trivial File Transfer Protocol
- A simple, UDP based file transfer protocol
- Has only trivial access control
- Very simple to use, but:
  - Very limited features (1981 technology)
  - Very unreliable (should only be used for small transfers)
  - Not very efficient

# IPMI

- Intelligent Platform Management Interface
- A specification & protocol for management and monitoring of hardware “out-of-band”
- Is implemented on (most) Baseboard Management Controllers (BMCs) (e.g. Dell iDRAC)
- We will heavily use:
  - The ability to control system power
  - The serial-over-LAN (sol) capability for remote console

- IPMI will likely be obsoleted by *Redfish*

# iPXE

- iPXE is a newer, open-source version of PXE
- Not all hardware supports it
  - Hardware that *does* may need to be flashed with special firmware;
  - It may be possible to emulated iPXE support by having PXE load an iPXE executable.
- More flexible; can use other file transfer protocols (HTTP, iSCSI,…)
- Some HPC management tools have started using iPXE instead of PXE for this flexibility

- *We will use PXE for this exercise because it's a bit easier to setup*

# Steps

# The Steps

1. Preliminaries
2. Configure `DHCP`
3. `IPMI` control of the compute node
4. Configure `TFTP`
5. Configure `PXELinux`
6. Create our compute image
7. Setup http & publish image
8. Create the “bootstrap” (`kernel` & `initramfs`)
9. Boot the compute node

# Step 0: Preliminaries

*These steps are required to get our master ready.*

- Get the supplementary materials! Grab netboot.tar.gz from the google drive. Keep a copy on your local system, but also copy to your master:
```
# scp <user>@10.0.52.xx:bootcamp-pxe.tar.gz /root
# cd /root ; tar zxvf bootcamp
```
- Disable selinux. selinux just complicates initial setup. We could try to re-enable it later, but some common cluster services aren't selinux friendly (e.g. Lustre).
- Disable firewall. Like selinux, this is just a convenience for getting setup. Any production cluster master would have the firewall enabled.
- Install some useful utilities. We will use these utilities for debug, verification and collaboration.

# Step 1: Configuring DHCP

*DHCP is one of the most crucial services in the cluster boot process.*

- Install **`dhcp`**.
- Configure **`dhcp`**. Main config in: **`/etc/dhcp/dhcpd.conf`**
  Instructions in the guide.
  Sample in the bootcamp files under: **`master/etc/dhcp/dhcpd.conf`**
- Validate **`dhcpd.conf`** syntax. This type of validation is always a good idea, especially on production systems.
- Enable/start **`dhcp`**.
- Verify **`dhcp`** status. Does **`systemd`** think **`dhcp`** is running?
- Watch logs. Once we see the BMC get a DHCP lease, we should be able to contact it.

# Step 2: IPMI Control

*IPMI (Intelligent Platform Management Interface) can be used to control node power, attach to the serial console, configure some BIOS settings and discover information about the node.*

- Install `ipmitool`.
- Get an IPMI shell. The IPMI shell provides an easy way to explore IPMI capabilities or run a sequence of commands.  Try the "`help`" command.
- Get the node MAC address. We will later do this for all nodes, but for now, we just need this one.
- Update DHCP config.
- See the guide for a few useful IPMI commands, or explore possibilities with "`help`".

# Step 3: Configure TFTP

*TFTP (Trivial File Transfer Protocol) is a standard, very simple mechanism for transferring files in a local network. It's used by PXE. It also is commonly used for things like backing up network switch configuration.*

- Install `xinetd` & `tftp`.
- Tell `xinetd` to start `tftp`. `xinetd` is a service manager daemon that manages the `tftp` service for us.
- Get the node MAC address. We will later do this for all nodes, but for now, we just need this one.
- Start/Enable `xinetd`.
- Verify & test `tftp`.
  - Make sure we're listing with `` `lsof` ``
  - Download a test file with the `tftp` client

# Step 4: Configure pxelinux

*Setting up the PXE configuration and support files.*

- Install `pxelinux`.
- Write the `pxelinux` configuration.  Like most bootloaders (e.g. grub), pxelinux has many config options.  For documentation, go to: `https://www.syslinux.org/wiki/index.php?title=PXELINUX`
- Some `pxelinux` tests:
  - Get the `pxelinux.0` file with `tftp`
  - Watch the node boot, and see that you get a `pxelinux` menu

# Step 5: Create Compute Image

*We can use `yum` to create a compute image in a “chroot” location. This is how many provisioners do this, but there are other ways.*

- Create directory structure.
- Perform chroot yum install. This will take several minutes.
- Copy configuration files. Verify file permissions first!
- Setup public key SSH authentication. Verify file permissions for this too!

# Step 6: Setup httpd & publish image

*We will use NGINX to get the image to the node.  This is one of many possible methods.*

- We should already have NGINX installed and running from the previous practicum.
- Verify `nginx` service:

```
# systemctl status nginx
# lsof -i
# links http://172.16.0.254
```

- Bundle & publish our image.
- Use `wget` to verify that we can download the image.

# Step 7: Create Bootstrap

*Things the bootstrap needs to do (we've done most of this work for you):*

- Extract the kernel from the compute image and copy into the `tftpboot` directory
- Create the `initramfs` and copy it to `tftpboot`:
  - Copy network modules from the compute image
  - Build a minimal (`busybox`) linux system
  - Copy initialization scripts into place
  - Build the `initramfs` image (`cpio`)

# Step 7.5: Inspecting the initramfs

*The initramfs contains all of the logic to grab the image and boot into it*

- Extract the `initramfs` `cpio` archive.
- Inspect these files:
  `/init`
  `/load_img.sh`
  `/modules.txt`

# Step 8: Boot the compute node(s)!

*Everything should be in place now, let's boot & test.*

- Setup a `tmux` split screen to watch logs and serial console at the same time.
- Power on the node.
- We should see this sequence of events:

1. (node) BIOS/UEFI bootup
2. (node) Attempts PXE
3. (master) DHCP logs handshake
4. (node) Shows PXE menu
5. (node) Loads kernel
6. (node) Initramfs loads
7. (master) Sees HTTP request for image
8. (node) Loads image/switch_roots
9. (node) Login prompt

- SSH to the node.

# Questions?