# BYOC: Build Your Own Cluster
# Part 1: Design

Nathan R. Vance, Michael L. Poublon, and William F. Polik
Hope College, Holland MI, 49423

Computer clusters are a standard tool in scientific computing. Clusters speed up calculations by computing a single task in parallel or by computing multiple tasks simultaneously, thus rapidly solving extremely large or computationally intensive problems. They utilize commodity hardware, resulting in an excellent performance-to-price ratio.

In a basic computer cluster, one computer (the head node) relays instructions to the rest of the computers (compute nodes) across an isolated local network. The compute nodes then carry out their assigned tasks, optionally communicate among themselves, and return the results to the head node. This structure is analogous to a work force: the head node is the manager, who receives jobs from a customer and subcontracts with the compute node workers. When a worker is done, it signals the manager that it is available for another task. When the job is completed, the manager returns the final product to the customer, in this case a calculated result.

Building a cluster is typically accomplished in one of two different ways. One way is to use configuration software such as ROCKS to set up a cluster automagically. While this method has the obvious strength of convenience, weaknesses include being constrained to the assumptions and supported architectures of the tool. Additionally, it can be difficult to diagnose problems that arise because these tools conceal what's happening under the hood.

The other method of cluster building is to build your own. It may take more time to complete, but the level of control and understanding of the cluster gained by this method makes it worth it in the long run. Cluster design goals include:

- **Robustness** - The cluster is flexible enough to support many different applications, whether known at the time of its initial setup or not.
- **Reliability** - The cluster, both hardware and software, should be stable for the long-term after the initial setup.
- **Portability** - The same process and tools used in this guide can be used on different distributions with little change.
- **Scalability** - The procedures should be practical if there are 2, 10, or 100 nodes.
- **No "Magic"** - Common problems are resolved by straightforward, well documented, easy-to-follow solutions.
- **Heterogeneity** - Upgrading in the future is still possible when the hardware may be different.
- **Simplicity** - The approach outlined below should allow a person with modest Linux/Unix experience to build a cluster of their own.
- **Low-Cost** - The cluster uses readily available hardware and free software (Linux).

This three-part series walks through the process of building a cluster that has all of these attributes. The first article gives an overview of the cluster's design, including the setup of computing hardware, networking, and storage. The second article deals with installing the operating system and software on the cluster in a scalable manner. The third article covers the

configuration of tools and services that transform the loose collection of computers to a tightly integrated cluster.

# Cluster Hardware

There are many hardware components that go into building your cluster. These can generally be broken down into six general categories: the computers themselves, networking supplies, physical storage of the computers, power distribution equipment, a console to access the computers, and spare parts.

- **Computers** - Clusters can be constructed from generic tower computers. But for large clusters, computers specifically designed for high performance computing can be purchased from suppliers such as Supermicro. These systems are preferable to tower PC's because they come in high-density packages, such as 1 or 2 U rack-mountable cases (a U is 1.75” of vertical rack space). In either case the computers must work well with Linux. For scalability, the compute nodes need to support PXE booting and IPMI control.
    - The head node typically is more capable than the compute nodes since it is the entry point for the entire cluster. It should support RAIDed hard drives (more on that later) and must have at least two ethernet ports, one to connect to the outside world and the other for the isolated internal compute node network.
    - Compute nodes should ideally be small, cheap, and plentiful. Depending on the application of the cluster, they could have any combination of powerful CPUs, large amounts of RAM, large hard disks, or GPUs.
- **Networking** - Your network switch needs to have at least as many ports as you have compute nodes. Extra ports are always handy in case you decide to add to your cluster in the future. Network cables are also essential.
- **Physical Configuration** - A cluster can be constructed from tower PCs on a shelf. However, a professionally-built cluster will typically use special rack-mounted computers. Computers will often be on rails allowing them to be slid out far enough to remove the lid without physically detaching them from the rack. It is advisable to leave enough slack in the cables on the backs of the computers so that they can be running while pulled out for diagnostic purposes.
  An important consideration is the location for the cluster. A cluster can be rather noisy due to the fans, so put it in a place where you won't mind some extra white noise. A cluster can also generate a lot of heat. If it’s large, you’ll need ventilation or a dedicated air conditioning unit.
- **Power Distribution** - Computers draw a lot of power, and lots of computers draw lots of power. The circuits they’re on must be able to handle the draw, and you’ll need power strips to distribute the power. If your cluster is small, a few consumer grade power strips should be adequate. Otherwise, large rack mountable power strips exist that report current.
  Additionally, the head node and storage unit should be plugged into an uninterruptible power supply (UPS) so that they don't immediately halt on a power outage, potentially corrupting data.
- **Access** - It isn’t practical to have a Keyboard, Video monitor, and Mouse (KVM) for every node in the cluster. It is a good idea, however, to have a local KVM hooked up to the head node. This guarantees that you will always be able to access your cluster to perform administrative tasks. There are specialty products such as a rack-mountable LCD monitor and keyboard that can serve this purpose well.

Once the cluster is set up, you will be able to access compute nodes using SSH from the head node. Under normal operation, the nodes can then be headless (operate without a KVM). Under abnormal operation, such as when initially setting up the cluster or when diagnosing a problem, you can access the compute nodes using a crash-cart, which is a mobile KVM with long cables that you can plug into whatever node is having troubles. Another option is a KVM switch. These switches can use standard IO cables (such as USB and VGA), or they can work entirely over IP if the computers' BIOSes support it. The more nodes involved, the pricier the KVM and cables will be.

- **Spares** - Stuff breaks. When you have a lot of stuff (as in a cluster), it breaks often. For example, let's say that you have 100 hard disks among all of the computers in your cluster, and each hard disk is rated to operate for 20 years. This is an annual failure rate of 5%, so you can expect 5 of them to fail in a year, or roughly one every 10 weeks. This analysis applies to all computer components, meaning that in addition to spare hard disks, it's a good idea to also purchase spare RAM, power supplies, and possibly even motherboards and CPU's. In a large cluster it's wise to have spare networking equipment, and in a production environment, an entire spare head node. Spare parts for compute node repairs are not as necessary since dysfunctional nodes can simply be taken offline or cannibalized for parts.

In summary, the parts needed to build your own cluster are as follows:

- Head node (optionally a storage node and a spare as well)
- RAID storage (integrated directly into the head node or storage node, or as a separate device)
- Compute nodes
- Networking switch(es)
- Networking cables
- Rack and mounting hardware
- Power strips
- Uninterruptible Power Supply
- KVM switch and cables
- Spare parts kit (hard drives, RAM, power supplies)

# Network Setup

Once you settle on hardware, you need to plan how to connect it up. We'll start with communication. First, give your cluster a name. Names are used as aliases for IP addresses, making it much easier for a human to identify individual computers on a network. A cluster computer uses two different networks: the external network (aka "the internet") that only the head node connects to, and the internal network that the cluster uses for internal communication. Therefore, two names must be configured, one for the external network and one for the internal network.

- **External Network** - This is used only by the head node. The name on the external network is typically formatted as `hostname.domain.suffix`, where the `hostname` is whatever you want, and the `domain.suffix` pertains to the organization using the cluster. The example used in this guide is `name.university.edu`.
- **Internal Network** - This is used by all nodes in the cluster. The internal name is typically the hostname component from the external name used in conjunction with a numbering scheme. For example, we append two digits to the end of the hostname for each node: `name00` (head node), `name01` (first compute node), etc. This scheme limits us to 100 nodes, but can easily be expanded to accommodate future upgrades.

Naming computers is vital for humans to be able to maintain the cluster, but the computers themselves deal with numeric IP addresses. The method for obtaining an IP address on the external network is up to your network administrator, but you get full reign over the internal network. Two methods exist for assigning IP addresses in the internal network: static and dynamic assignment.

- **Static Assignment** - Each compute node is configured individually with its own IP address. This contradicts the scalability goal of this guide because manually configuring IP addresses for a large number of nodes is not practical.
- **Dynamic Assignment** - Each compute node has an identical configuration and receives its IP address from the head node through the network based on its unique MAC address. This guide uses dynamic IP assignment.

# Storage Node

So far in our description of a cluster we have mentioned a single head node that acts as an access point to the cluster, along with many compute nodes to perform the tasks the cluster receives. Many large clusters will further separate out tasks, especially if the head node becomes a bottleneck for cluster operation. For example, it is common to have a separate storage node to manage the files to which the compute nodes need access, such as application software and each user's home directory.

# Disk Partitioning

As opposed to Windows where partitions are referred to as lettered drives, in Linux they are mounted under directories called "mount points" in the file system. Partitions are useful for keeping data, applications, and system software separate for easy backups and reinstallations. This section highlights useful Linux partitions assumed in this article:

- **root (/)** - This partition is where the actual operating system resides, and other partitions will be mounted in its file system.
- **/boot** - The files Linux uses to boot, including the kernel itself, are located here. For mostly historical reasons some administrators prefer to keep this on a separate partition, but we will keep them on the same partition as root.
- **/admin** - Disk images, software distributions, kickstart files, and backups are stored here. This is vital for the installation of all compute nodes in a scalable way.
- **/home** - User files are located here. We will make this a separate partition from root for ease of backups, upgrading, and reinstallations.
- **/export** - System wide application software to be run on compute nodes is stored here. While sometimes a partition of its own, it can be subsumed under /home/export instead.
- **/scratch** - Hefty computations like those done on clusters often involve writing large temporary files to the hard disk over the process of the computation, then reading these files to complete a result. It is recommended to have a large partition on all compute nodes set aside for this purpose.
- **swap** - Swapping is the process by which, should Linux run out of memory, it writes pages of memory to the swap partition on the hard disk. This can result in allowing memory intensive software to run on systems with too little RAM. However, swapping is orders of magnitude slower than using RAM. Perhaps a decade ago when RAM was expensive it would have been advisable to have a large swap partition. But now that

RAM is cheap, it is best not to swap at all but instead to buy more RAM if memory constraints become a problem, or use software that is designed to use the /scratch partition.

If your nodes use multiple disks, you will have the choice of which ones to use for which partitions. By convention, the root partition should go on the first hard disk, but the rest is up to you. The following is an example of single disk partitioning schemes using 1TB hard drives.

| **Head node** | | | **Compute Node** | |
|---|---|---|---|---|
| / (includes boot) | 200 GB | | / | 200 GB |
| /admin | 200 GB | | /scratch | rest of space |
| /home (includes export) | rest of space | | | |

Table 1: Head and compute node partitioning schemes for 1 TB drives

# RAID Devices

When storing large amounts of data, it is highly recommended to utilize a RAID (Redundant Array of Independent Disks) device. This may be integrated directly into the head node, a separate component connected to the head node, or part of a separate storage node.

A **RAID** device works by combining several small physical drives to form one larger, faster virtual drive. A RAID device can also introduce data redundancy, which allows a drive to fail while still preserving the data. This is absolutely vital in a production environment. As was mentioned earlier, with many components comes frequent component failures. A drive on a compute node failing isn't the end of the world since there's nothing on it that can't be reinstalled, so clusters will often use a single hard drive for each compute node. However, losing all of the cluster's application or user data would be a disaster, making RAIDing of head node partitions a must.

There are several commonly used RAID levels that achieve increased size and speed, redundancy, or both of these goals.

- **RAID 0** provides a storage size and performance increase by "striping" data across two or more drives. This means that consecutive data segments are stored on different disks. This may significantly improve read time in some applications; however, one failed drive causes all of the data to be lost. A RAID 0 drive is as large as the size of its smallest drive times the number of drives.
- **RAID 1** provides redundancy with no storage size or performance increase by "mirroring" data writes to two or more disks, allowing one to go down while still preserving the data. The size of a RAID 1 drive is the same as its smallest drive.
- **RAID 5** is similar to RAID 0 except that it includes redundant parity information spread across the 3 or more disks. This allows any one disk to fail without the loss of data. The RAID as a whole will store as much as the smallest drive times one less than the number of drives.

- **RAID 6** is similar to RAID 5 except that it has two disks worth of redundant parity information spread across 4 or more drives. This allows for two disks to fail without the loss of data. Storage will be limited to the size of the smallest drive times two less than the number of drives.

**Hot Spares** are blank drives included with a RAID 5 or 6 device. When one drive fails, the RAID device rebuilds the information formerly on that drive onto the hot spare using the redundant information spread across the other drives. If hot spares are not included, this process would only begin when you manually swap out the failed disk. If you happen to be on vacation and can't get a friend to perform this task, you run the risk of additional drives failing and resulting in data loss before you return.

No matter how many hot spares you provision yourself with, your data isn't 100% safe from disk failures wiping it out. Using a RAID device may make the likelihood of losing your data smaller, but it won't eliminate it. Therefore, if your data is at all important (you're going through the effort of building a cluster in order to obtain it, so it is), make sure you have access to another machine in a different physical location to which you can back up files.

# Assembly

After you have gathered all your hardware and planned the configuration, you can begin the fun part of cluster work: actually assembling your cluster. Arrange your nodes for good air flow. When running cables, make sure to use differently colored cables for the internal and external networks, and label them on both ends. It can be both time consuming and frustrating to track down a problem only to find that it was caused by a swapped network cable. If things are kept consistent among nodes, your life will be much easier when it comes to managing your cluster. Ideally when starting out, your compute nodes should all be identical, both in terms of internal hardware and external cable configuration. In our experience, however, this ideal is seldom maintained over the long-term as the cluster expands or specialized capabilities are added.

# Maintenance

Scalability for a cluster requires that it is easy to set up all of the compute nodes with identical configurations. This ability is useful in several scenarios: to initially install the cluster, to reinstall a node for maintenance, or to add new nodes to the cluster.

There are two methods for achieving this goal. The first is to perform a complete installation on a single node, save a disk image, and write that image to all other nodes. Unfortunately, this strategy results in a loss of support for heterogeneity. If you desire to add nodes of a different architecture than what's already in the cluster, you'd be forced to start from scratch in installing them.

The other method is to script all the changes to the operating system so that they can be applied during an automated install. Such installation scripts typically record basic settings similar to those that would be configured in the system installer, a list of software packages to install beyond the initial system, and a script to handle all other modifications. This installation method solves the issue of heterogeneity in that the installer handles the choice of software, allowing the same script to be used on multiple architectures assuming that all requested software is packaged for the different architectures. Furthermore, a script containing an

exhaustive list of modifications from a clean install is an excellent resource when diagnosing future issues or for performing an operating system upgrade. Most major distributions support this method. On CentOS it is called kickstart.

In practice, a combination of these two methods is used. For example, scripts are used to build the cluster, and an image can be used to replace a bad hard disk on a compute node.

# System Software

In part 2 of this series we will dive into installing CentOS using kickstart. This procedure involves performing a single manual installation to generate the base kickstart file, then iteratively making modifications until the operating system installs on the head node and compute nodes without manual intervention. The end result will be a functional operating system on every node with networking in place.

In part 3 we will describe the installation and configuration of software that makes these networked computers function as an integrated cluster. This software includes DHCP for IP addresses assignment, NFS to share file systems over the internal network, passwordless SSH between all nodes, a suite of administrative tools, a local software repository for supplying RPMs to compute nodes, Ganglia for monitoring the cluster, and SLURM as a resource manager.

# Application Software

When building a cluster, it's important to know what you're going to do with it. You should have already done some research to be sure that software exists to achieve your goal. For example, we built a high-throughput computational chemistry cluster that runs quantum chemical programs and molecular dynamics simulation software. This requires compute nodes with multi-core CPUs, GPUs, large amounts of RAM, and significant scratch space.

An important consideration is licensing of the application software that will run on the cluster. For example, there are many free or open source computational chemistry programs for which licensing isn't a problem, such as MOPAC, GAMESS, and ORCA. One can purchase a site license for commercial software such as Gaussian and use it across the cluster. Other commercial programs like QChem require being keyed to the specific nodes on which they will be running.

# Conclusion

In this article we discussed the considerations that go into designing a cluster computer. To start we outlined our design goals, a vital one being that our setup is scalable to accommodate any number of nodes without their installation and administration becoming impractical. We then discussed the hardware that goes into the cluster, including the computers, networking equipment, physical storage, power distribution, access, and spare parts. We also designed a disk partitioning scheme for the head node and compute nodes that allows for easy backups, upgrades, and reinstallations. We described the networking of a cluster, including an external network connection and an isolated internal network. Finally, we discussed the physical

assembly of the cluster, introduced the importance of maintenance, and touched on the cluster's application.

In the next article we will install the base operating system and set up network connectivity. In the process we will create two kickstart files, one for the head node and the other for the compute nodes. In the third article we will turn the group of computers into a single cluster by configuring vital system services to communicate and run cluster software.

# BYOC: Build Your Own Cluster
# Part 2: Installation

Nathan R. Vance, Michael L. Poublon, and William F. Polik
Hope College, Holland MI, 49423

In part 1 of this three-part series, we left off with bare metal hardware assembled, a disk partitioning scheme for the head node and compute nodes, and a design for the network.

In this article we will bootstrap ourselves up to performing fully automated operating system installations on both the head node and compute nodes, a vitally important step for the cluster to be scalable to large numbers of compute nodes. To create the kickstart scripts used in automated installations, we will perform the installation many times, each time with a larger amount of the process being automated. By the end of this article we'll have an operating system on all nodes and network connectivity.

## Head Node Manual Installation

We start with doing things manually for several reasons. First, it's a great opportunity to make sure that the hardware is correctly set up. Second, a manual installation generates the kickstart file template that will be used in subsequent installations. While examples of kickstart files can be found online, it's more useful to generate it yourself because then you can be sure that it contains the specifics for your hardware setup.

To perform the installation, you'll have to download the distribution as an ISO file, burn it to a DVD, and boot from it. This guide uses CentOS 7, the redistributable version of Red Hat Enterprise Linux. You could alternatively burn it to a USB drive; however, Linux currently names hard disks and flash drives in an arbitrary order, changing the sda and sdb labels on different boots. This will become a problem when we start kickstarting installations. There are workarounds like using the more verbose UUID naming scheme, but to avoid confusion, we will assume you use the DVD.

When installing, we will assume the disk partitions as described in part 1 of this series:

- **/** - 200 GB
- **/admin** - 200 GB
- **/home** - rest of space

For networking, configure the interface on the external network as your network administrator dictates. As you've probably discovered by now, it's important to always be on your network administrator's good side since they have nearly unlimited power over your internet connectivity. But you have full control over the internal network of the cluster.

When configuring the network interface on the internal network, set it to use a statically assigned IP address. The address should be selected from one of the private IP address ranges, which are:

192.168.0.0 - 192.168.255.255
172.16.0.0 - 172.31.255.255

10.0.0.0 - 10.255.255.255

In this article, we will use a subset of the 192.168 range, specifically 192.168.1.100 - 192.168.1.199, which provides 100 IP addresses. The head node should use the first IP address in the range. Therefore, the configuration for the head node on the internal network is:

Address: 192.168.1.100
Netmask: 255.255.255.0

When you select packages to install, choose Server With GUI, and choose E-mail Server and Development Tools as add-ons. We'll fine tune this selection later, but this is a good starting point.

# Head Node Automated Installation

To achieve reliability, one needs to have a reproducible installation method that provides consistent results. Luckily, Red Hat's installer, anaconda, has a reliable and scalable method called "kickstart." Kickstarting means that the installer uses a configuration file to automatically install Linux. Anaconda generates a kickstart file after every installation, which can be found at `/root/anaconda-ks.cfg`.

After manually installing the head node, locate this file and copy it as `ks.cfg` to a separate computer as a backup. We will be editing `ks.cfg` over the course of this article, and if the only copy resides on the same machine being reinstalled, a typo could result in its destruction.

## Editing the ks.cfg File

The kickstart file can be edited to include all desired installation options and post-installation configurations. The kickstart file must use Unix end of line characters, so if you're going to edit the file on a Windows machine, use an editor like notepad++ that respects this difference. Otherwise, it's easiest to just edit it on the head node and back it up to some other machine.

Kickstart files have a specific formatting so that they can be parsed by the system installer. A few important features include:

- **Comments** - Any line with a leading `#` is ignored by the installer. They are used to comment on code or to disable small sections of code.
- **Partitioning** - There are several lines with partition info that you selected during the manual installation. Add the option `--noformat` to the partition entries that you don't want to be formatted by the installer such as `/admin` and `/home`. Do not add this option to the / partition as it contains the previously installed OS, which should be erased and replaced during a reinstallation.
- **Repository** - This tells the installer where to find the repository to install from. Currently, it is set to install from a cd:
  ```
  # Use CDROM installation media
  cdrom
  ```
  The repository line will be modified several times during this guide.
- **Miscellaneous Settings** - Settings may be applied such as disabling selinux, changing bootloader options, and much more. Modify them to disable selinux:
  ```
  selinux --disabled
  ```
  Visit Red Hat's Kickstart Options guide for an exhaustive list of kickstart options at `https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Installation_Guide/sect-kickstart-syntax.html`

- **Packages** - After the `%packages` tag is a list of packages to be installed. Those that start with a `@` correspond to groups of packages, such as `@ Base`. The others are individual packages.
- **Postscript** - At the end of the installation some additional commands may be executed. Add the following to the end of the file:
  ```
  %post
  %end
  ```
  Between these tags we will add post installation scripts that will be executed once the installation completes. In the DHCP configuration section later in this article a sample script to automatically configure DHCP is given. It is highly recommended to include all similar system modifications here as well for documentation, backup, and reinstallation purposes.

Creating an automated installation is an iterative process in which one modifies the kickstart file, reinstalls the system, and verifies that the changes took hold as intended. For example, to check that `/admin` and `/home` (and any others that you specified) don't get formatted during the install, create a file on the partition and see if it exists after anaconda finishes reinstalling. Also check that selinux is indeed disabled. You will perform a multitude of installations during the creation of the cluster, each time automating and then testing a new capability or feature added to the cluster. Hence, it is vital to automate the process.

Accessing the kickstart file for an automated installation can be done in a variety of ways. The kickstart file can be located on an ext2 or fat32 formatted usb drive, or on a partition on the hard drive from where it can be read by the installer.

The following table summarizes the different installation methods we will use to bootstrap ourselves up to a fully automated installation without removable media. In this table, the boot/installer is the collection of files from which the CentOS installer boots, which currently reside on the DVD. Likewise, the distro is the repository of software that the installer uses to set up the CentOS operating system, also currently located on the DVD.

| Installation Method | Boot/Installer Location | Kickstart Location | Distro Location |
|---|---|---|---|
| Manual | DVD | None | DVD |
| Automated DVD | DVD | USB | DVD |
| Automated Hard Drive | DVD | Hard Drive | Hard Drive |
| Sans Removable Media | Hard Drive | Hard Drive | Hard Drive |

Table 1: Installation plan for the head node

The end goal is to eliminate the need for removable media and instead to use the hard disk for the entire installation process.

## DVD Based Booting with Kickstart on USB

Before we start, this method has one caveat: depending on the hardware configuration of your system, when the installer boots, the usb drive *may* show up as a different drive than normal. For example, in a configuration with two hard drives sda and sdb, when you insert a usb drive it comes up as sdc. However, upon booting into the installer the usb drive might be sda while the hard drives are sdb and sdc. This scenario would require the kickstart file to be edited so that all references to any sdX hard drives are shifted one letter.

To install CentOS using a kickstart file on a USB drive:

1. Copy the kickstart file to a blank ext2 or fat32 formatted usb drive. Make sure the repository line is set to
   `cdrom`
2. With the usb inserted into the head node, you can now use the kickstart file while booting from the DVD by pressing tab at the initial welcome screen and appending the following to the boot options:
   `inst.ks=hd:sd`**`X`**`1:/ks.cfg`
   Note that sdX corresponds to the usb device when booting with it in place, and that the drive letter X might not be the same as when you inserted it into a running operating system. The installer will allow you to edit this line if it fails to find the kickstart file on the specified device.

By performing an installation this way we have verified that the kickstart file is properly formatted. The next step is to eliminate the USB drive and to reduce our reliance on the DVD.

# DVD Based Booting with Kickstart and Distro on Hard Drive

The first step in migrating away from external media is to move the kickstart and distro to the hard drive, thus eliminating the USB drive and reducing the responsibility of the DVD down to just booting.

To use the kickstart and distro from the hard drive:

1. Make sure you know the device name of the partition mounted as `/admin`. This information can be discovered using the `lsblk` command.
2. Create the following folder hierarchy in `/admin`
   ```
   # mkdir -p /admin/iso/centos7/
   # mkdir -p /admin/ks/headnode/
   ```
3. Copy the installation media to `iso/centos7`. If you have the original ISO file floating around you can simply copy it across the network. If not, you can use the dd command to create it from the DVD.
   ```
   # dd if=/dev/cdrom of=/admin/iso/centos7/CentOS-7-x86_64-DVD-1511.iso
   ```
4. Copy the kickstart file to the newly created `ks/headnode` folder. In the `ks.cfg` file, comment out the repository line and replace it with the following:
   `harddrive --partition=sd`**`XY`**` --dir=/iso/centos7/`
   Note that `sd`**`XY`** is the partition in which `/admin` is located. This can be discovered using the `lsblk` command. Also, while you have `ks.cfg` open, verify that it isn't set to format `/admin`. You've just made some fairly significant additions to that partition, and it would be a bummer to wipe them all out.
5. Insert the DVD and reboot. Press tab at the initial welcome screen and append the following options:
   `inst.ks=hd:sd`**`XY`**`:/ks/headnode/ks.cfg`

The installation should now proceed using both `ks.cfg` and the ISO from the hard drive. This makes the installation go much faster since it doesn't have to read everything from the DVD.

The next step in eliminating external media is to set up the `/admin` partition to be the installer.

## Installation Sans Removable Media

Currently, the only role the DVD serves is to boot the installer. In this final step we will transfer that role to the hard drive, eliminating the need for all external media. To do so we will configure grub to boot directly into the installer, pass it the boot options for locating the kickstart file, and install CentOS completely automatically, without any external media or typing `inst.ks=<location>` options into the terminal.

To run the installer from the hard drive:

1. Make a directory to house boot files.
   ```
   # mkdir /admin/boot/
   ```
2. Mount the iso and copy the internal files to the boot directory.
   ```
   # mount -o loop /admin/iso/centos7/CentOS-7-x86_64-DVD-1511.iso /mnt
   # cp -a /mnt/* /admin/boot/
   ```
3. At the bottom of `/etc/grub.d/40_custom` insert the following:
   ```
   menuentry "Install" {
          set root=(hdW,msdosZ)
          linux /boot/images/pxeboot/vmlinuz ks=hd:sdXY:/ks/headnode/ks.cfg
          initrd /boot/images/pxeboot/initrd.img
   }
   ```
   Note the `(hdW,msdosZ)` and `ks=hd:sdXY` lines. These correspond to the drive and partition for `/admin`. Use the command `lsblk` to find the partition in `sdXY` format. **`W`** then corresponds to the drive number; sda is 0, sdb is 1, and so forth. **`Z`** is the partition number `/admin` is on. So, if `/admin` is located on sda2, use `(hd0,msdos2)`.
4. Regenerate grub.cfg.
   ```
   # grub2-mkconfig -o /boot/grub2/grub.cfg
   ```
5. At the end of the ks.cfg file, between `%post` and `%end`, insert the following:
   ```
   #grub configuration
   cp -p /boot/grub/grub.conf /boot/grub/grub.conf.000
   cat >> /etc/grub.d/40_custom << EOF
   menuentry "Install" {
          set root=(hdW,msdosZ)
          linux /boot/images/pxeboot/vmlinuz ks=hd:sdXY:/ks/headnode/ks.cfg
          initrd /boot/images/pxeboot/initrd.img
   }
   EOF
   grub2-mkconfig -o /boot/grub2/grub.cfg
   ```
   This script will now automatically modify your grub.cfg file as done in steps 3 and 4. Make sure to modify the `(hdW,msdosZ)` and `sdXY` lines as in step 3. The method used here for writing this text to `/etc/grub.d/40_custom` is called a here file. As opposed to using the echo command, here files have far fewer characters that must be escaped, though there are some characters that still need to be. For example, every $ or ` symbol must be escaped with \$ or \`. Otherwise, bash will attempt to resolve it as a variable or executable script.
6. Reboot. At grub's splash screen, arrow down to the entry titled "Install" and press enter. The system should now install.

Now the system installs automatically without external media. This is a very useful ability to have when rebuilding the head node. It will be essential for building compute nodes!

# Head Node DHCP

Before installing the compute nodes, Dynamic Host Configuration Protocol (DHCP) must be configured on the head node. DHCP allows the compute nodes to receive their network configuration from a central server. This step is vital for a scalable system because it allows the nodes to be identically configured but have unique IP numbers.

To configure DHCP on the head node:

1. Install dhcp
   ```
   # yum install dhcp
   ```
2. Add the following to `/etc/dhcp/dhcpd.conf`
   ```
   #dhcpd config options
   authoritative;
   default-lease-time -1;
   option broadcast-address 192.168.1.255;
   ddns-update-style none;
   next-server 192.168.1.100;
   filename "pxelinux.0";

   subnet 192.168.1.0 netmask 255.255.255.0 {
          range 192.168.1.101 192.168.1.199;
          option subnet-mask 255.255.255.0;
          option domain-name-servers 8.8.8.8;
          option routers 192.168.1.100;
   }
   ```
   Change the range option to include enough addresses for all of your nodes. Note that some bolded options may need to be adjusted if the head node's internal IP is not 192.168.1.100. The domain-name-servers option may point to whatever DNS you desire, in this example we use Google's at 8.8.8.8, though some network administrators may require you to use their own.
3. To start the DHCP service on the head node at boot time, execute the command
   ```
   # systemctl enable dhcpd
   ```
   And to start the service immediately, execute the command
   ```
   # systemctl start dhcpd
   ```
4. To automate this process, in `ks.cfg`, append dhcp to the `%packages` list, and between the `%post` and `%end` tags add the following:
   ```
   #dhcp
   cp -p /etc/dhcp/dhcpd.conf /etc/dhcp/dhcpd.conf.000
   cat > /etc/dhcp/dhcpd.conf << EOF
   [Contents of dhcpd.conf as determined in step 2 go here]
   EOF
   systemctl enable dhcpd
   ```
5. To use DHCP, the firewall must allow it. Since we want all communications on the internal network to go unobstructed, we can simply add the interface on that network to firewalld's trusted zone.
   ```
   # firewall-cmd --permanent --zone=trusted --change-interface=[INTERNAL
   INTERFACE]
   # echo "ZONE=trusted" >> /etc/sysconfig/network-scripts/ifcfg-[INTERNAL
   INTERFACE]
   # nmcli con reload
   # firewall-cmd --reload
   ```

Use the command `ip addr` to discover **`[INTERNAL INTERFACE]`**. Add these changes to the kickstart using the following lines

```
firewall-offline-cmd --zone=trusted --change-interface=[INTERNAL INTERFACE]
echo "ZONE=trusted" >> /etc/sysconfig/network-scripts/ifcfg-[INTERNAL INTERFACE]
```

A more thorough explanation on firewalld will be forthcoming in the next article.

Now that the head node is fully automatically installed and support for a basic network is in place, we can proceed to the compute nodes.

# Compute Node Manual Installation

On a single compute node, boot from the installation DVD and perform a manual installation.

Compute nodes have a different partition table, network setup, and package selection than the head node. The example partitioning scheme we used in Part 1 of this series is:

- / - 200 GB
- /scratch - rest of space

When configuring networking for the compute node, connect using a dynamic IP address. If DHCP is set up on the head node correctly, you should receive an address.

For now, make the package selection a Minimal installation.

The generated kickstart file on the newly installed compute node is located at `/root/anaconda-ks.cfg` and should be transferred to the head node. In `/admin`, create a directory to house the compute node kickstart file:

```
# mkdir /admin/ks/computenode/
```

And copy it over the network from the compute node to the head node

```
# scp 192.168.1.101:/root/anaconda-ks.cfg /admin/ks/computenode/ks.cfg
```

substituting the node's actual ip address as determined by running the command `ip addr` on it.

# Compute Node Automated Installation

Automated installations are incredibly important for compute nodes in a cluster. While manual installations may be practical (though still undesirable because of reliability) for small test clusters, they scale poorly and are impractical for medium to large clusters.

For the compute nodes, the process of automating the installation will be similar to that of the head node. In the kickstart file, be sure to disable selinux as was done for the head node, and also disable the firewall because the compute nodes won't be connected to the outside world anyway.

```
selinux --disabled
firewall --disabled
```

Like with the head node, we will transfer responsibilities away from the DVD. This time, rather than the final destination for installation files being the compute node itself, it will be the head node accessed over the network. In the final configuration, the compute nodes will boot over

Pre-boot eXecution Environment (PXE), and will retrieve their kickstart files and distributions via a Network File System (NFS).

| Method | Boot/Installer Location | Kickstart Location | Distro Location |
|---|---|---|---|
| Manual | DVD | None | DVD |
| Automated DVD | DVD | USB | DVD |
| Automated NFS | DVD | NFS | NFS |
| Automated PXE | PXE | NFS | NFS |

Table 2: Installation plan for the compute nodes

## DVD Based Booting with Kickstart on USB

The procedure here is the same as under the head node. If you are still learning about kickstart files, feel free to follow the DVD Based Booting with Kickstart on USB instructions under the head node section. But otherwise, save yourself time and use NFS as described in the next section.

## DVD Based Booting with Kickstart and Distro on NFS

NFS allows compute nodes to access files stored on the head node over the internal network. This is a necessary step for the scalability of the cluster, since it would be impractical to have a copy of the kickstart and distro locally on each node at installation time.

In this section it is assumed that the directory structure on the head node is configured as follows:

- `/admin/boot` contains the contents of the DVD as in Head Node Installation Sans Removable Media
- `/admin/ks/computenode/ks.cfg` is the kickstart file for the compute nodes
- The head node's IP address on the internal network is 192.168.1.100
- The head node is assigning addresses using DHCP (this was verified during the compute node manual installation)

If your setup differs, modify the following commands accordingly.

To install over NFS:

1. On the head node, open the file `/etc/exports` in a text editor and add the following:
   `/admin 192.168.1.100/255.255.255.0(ro,sync,no_root_squash)`
   This gives all computers on the 192.168.1.0/24 subnet read only (ro) access to the `/admin` directory.
2. Restart the NFS service so that this change takes effect. If NFS wasn't already running, this will start it.
   `# systemctl restart nfs`
3. In the compute node `ks.cfg` file (located on the head node), comment out the repository line and replace it with:
   `nfs --server=192.168.1.100 --dir=/admin/boot`

This tells the installer to look for the installation files on the network rather than on a DVD.

4. On a compute node, determine the name of the network interface that connects to the internal network using the command `ip addr`. The interface name could be formatted in one of several different ways depending on your motherboard, such as ethX, enoX, enpXsY, or if you're unlucky, en<mac address>.
5. You can now use the kickstart file while booting from the DVD by pressing tab at the initial welcome screen and appending the following to the boot options:
   `ks=nfs:192.168.1.100:/admin/ks/computenode/ks.cfg ksdevice=`**`ethX`**
   substituting `eth`**`X`** for the interface name found in step 4.
6. To make the changes on the head node persistent between installs, add the following to the end of the ks.cfg file for the head node between the %post and %end tags:
```
#nfs
echo "/admin 192.168.1.100/255.255.255.0(rw,sync,no_root_squash)" >>
/etc/exports
systemctl enable nfs
```

The compute nodes are now capable of retrieving kickstarts and installation files over the network, but that is only half of the story. To make the installation proceed without any media, the nodes must boot over the network as well.

# PXE based NFS

Pre-boot eXecution Environment (PXE), called MBA on some BIOSes, allows nodes to retrieve their boot media via the network. Here are some basic prerequisites before using PXE:

- Your motherboard supports PXE
- PXE is enabled in your BIOS
- PXE is set before local boot methods on the BIOS boot order

PXE allows us to perform kickstart installations on the nodes without having to physically load any disks. It's then possible to start an automated installation merely by powering on the cluster. Think about it: throw a switch, take a lunch break, and when you get back, the entire cluster is installed. That's scalability!

To make this claim a reality and install the compute nodes from the head node:

1. On the head node, install syslinux, tftp-server, and tftp using yum. Add these to the Packages section of the kickstart file for documentation and reinstallation purposes.
2. On the head node, make and populate the directory `/admin/tftpboot`
```
# mkdir /admin/tftpboot
# cp /usr/share/syslinux/pxelinux.0 /admin/tftpboot/
# cp /usr/share/syslinux/menu.c32 /admin/tftpboot/
# mkdir -p /admin/tftpboot/images/centos7/
```
3. Copy in a compressed kernel and initial ramdisk from which the compute nodes can boot
```
# cp /admin/boot/images/pxeboot/vmlinuz /admin/tftpboot/images/centos7/
# cp /admin/boot/images/pxeboot/initrd.img /admin/tftpboot/images/centos7/
```
   These two files are necessary for booting a bare bones linux system. Vmlinuz is a compressed linux kernel, and initrd.img is a temporary root filesystem. When we boot a compute node over PXE, these two files will be transferred to the compute node, giving it the software required to access the kickstart file and the rest of the installation files over NFS.
4. Create a directory to hold the PXE configuration files
```
# mkdir /admin/tftpboot/pxelinux.cfg
```
5. Create the new file `/admin/tftpboot/pxelinux.cfg/default` containing the following:
```
DEFAULT menu.c32
PROMPT 0
```

```
TIMEOUT 100
ONTIMEOUT kickstart
MENU TITLE PXE Menu
MENU seperator
LABEL local
LOCALBOOT 0
MENU seperator
LABEL   kickstart
        kernel images/centos7/vmlinuz
        append initrd=images/centos7/initrd.img
ks=nfs:192.168.1.100:/admin/ks/computenode/ks.cfg ksdevice=ethX
```

[Editor Note: the append line is all one line including the ks and ksdevice arguments.] Modify the ksdevice as needed. Notice that when the menu times out (`ONTIMEOUT`), it defaults to the `kickstart` option. This is useful for installing the cluster without manual intervention. But this must later be changed to `local` for the cluster to reboot without reinstalling the OS.

6. Edit the file `/usr/lib/systemd/system/tftp.service`, changing the line
   `ExecStart/usr/sbin/in.tftpd -s /var/lib/tftpboot`
   to
   `ExecStart/usr/sbin/in.tftpd -s /admin/tftpboot`
   and restart the service so that this change takes effect. This change could be added to the head node's kickstart file using a here file, but it's more concise to use sed:
   ```
   # sed -i.000 's|/var/lib/tftpboot|/admin/tftpboot|'
   /usr/lib/systemd/system/tftp.service
   ```
   In this command, the flag -i.000 specifies that sed is to do the modification in place, that is, it will perform the change and write it back to the original file. The .000 part makes it so that sed will save the original file with a .000 extension as a backup. The next component, in single quotes, specifies the operation that sed is to perform. The 's' tells it to do a substitution (as opposed to a deletion or insertion), the pipe (|) serves as a delimiter between parts of the command, and the two strings are the parts to exchange. Finally, the file path at the end of the command specifies the file that sed operates on.
7. Reboot a compute node. Or all of them. If everything is set up correctly, they will install automatically.
8. Change `ONTIMEOUT kickstart` to `ONTIMEOUT local`. Otherwise, every reboot will result in a reinstall.

An explanation on the inner workings of PXE booting is in order. When `pxelinux.0` boots on a machine, it tftp's back to the boot server and tries to find a configuration file in the directory `pxelinux.cfg`. The filename is determined by converting the IP address given to it by the DHCP server to hexadecimal. For example:

| | | | | |
|---|---|---|---|---|
| 192 | 168 | 1 | 101 | |
| C0 | A8 | 01 | 65 | → C0A80165 |

`pxelinux.0` attempts to find files in `pxelinux.cfg` in the following order:

C0A80165
C0A8016
C0A801
C0A80
C0A8
C0A
C0

C
default

This PXE feature is useful for supplying specific kickstart files for different sets of compute nodes because of hardware differences; for example, it can supply different partitioning schemes for different hard disk sizes. Right now DHCP on the head node is configured to dole out IP addresses in an arbitrary order, making it hard to take advantage of this feature. In the next article we will fix that.

When booting a compute node over the network, vmlinuz (the compressed kernel) and initrd.img (the compressed initial file system) are transferred back to the compute node, along with the boot options.

In our case with the `kickstart` option, the boot options tell the installer (included in initrd.img) the location from which to retrieve `ks.cfg`, which in turn includes the location of the distro.

PXE is also useful for booting other images such as Memtest and other diagnostic tools.

# Conclusion

In this article we covered the basics of kickstart files on CentOS, and we set up a scalable method for installing the entire cluster. The resulting system is capable of intercommunication over ssh as root, but it doesn't have any useful cluster-wide application software or users on it yet.

In the final article we will address the Linux services that are vital for cluster operation, culminating on a resource manager called SLURM. With this software in place, the cluster will be fully-fledged and ready for its end users.

# BYOC: Build Your Own Cluster
# Part 3: Configuration

Nathan R. Vance, Michael L. Poublon, and William F. Polik
Hope College, Holland MI, 49423

In part 1 of this series we designed and assembled a computer cluster, and in part 2 we set up a scalable method to perform reproducible installations across an entire cluster. At this point, we have a cluster consisting of Linux (CentOS 7) installed to every node and a network that currently allows SSH between nodes as root. Additionally, we have kickstart files, making it possible to reinstall the entire cluster in a scalable manner.

We must now configure various Linux services to convert our collection of networked computers into a tightly integrated cluster. In this article, we will address:

- **Firewalld** - a firewall on the head node to protect the cluster from external threats
- **DHCP** - a more in-depth look at assigning IP addresses to compute nodes in a deterministic manner
- **/etc/hosts** - the file that maps node names to IP addresses
- **NFS** - share additional file systems over the network
- **SSH/RSH** - log into compute nodes without providing a password
- **btools** - scripts to run administrative tasks on all nodes in the cluster
- **NTP** - keep the cluster's clock in sync with the Network Time Protocol
- **Yum Local Repository** - install packages to compute nodes without traffic forwarding
- **Ganglia** - a cluster monitoring suite
- **Slurm** - a resource manager for executing jobs on the cluster

Each of the following sections gives a description of the service, how to configure it for use in a cluster, and some simple usage cases as appropriate. Installing these services converts our networked computers into a useful functioning cluster.

# Firewalld

Firewalld is an abstraction layer for netfilter and is the default firewall for CentOS. Only the head node needs a firewall because it is the only node that is in direct contact with the outside world. The compute nodes should already have had their firewalls disabled in their kickstart so that their firewalls don't interfere with internal communication.

An article in Linux Journal in September 2016 describes how firewalld works. Provided here are a few commands to set up a basic firewall that allows ssh and http at your local institution, drops traffic from the rest of the world, and allows all traffic on the internal network:

```
# firewall-cmd --permanent --zone=internal --add-source=[IP/MASK OF YOUR INSTITUTION]
# firewall-cmd --permanent --zone=internal --remove-service=dhcpv6-client
# firewall-cmd --permanent --zone=internal --remove-service=ipp-client
# firewall-cmd --permanent --zone=internal --add-service=ssh
# firewall-cmd --permanent --zone=internal --add-service=http
# firewall-cmd --permanent --zone=public --remove-service=ssh
# firewall-cmd --permanent --zone=public --remove-service=dhcpv6-client
# firewall-cmd --permanent --zone=public --set-target=DROP
```

```
# firewall-cmd --permanent --zone=public --change-interface=[EXTERNAL INTERFACE]
# echo "ZONE=public" >> /etc/sysconfig/network-scripts/ifcfg-[EXTERNAL INTERFACE]
# firewall-cmd --permanent --zone=trusted --change-interface=[INTERNAL INTERFACE]
# echo "ZONE=trusted" >> /etc/sysconfig/network-scripts/ifcfg-[INTERNAL INTERFACE]
# nmcli con reload
# firewall-cmd --reload
```

In this listing, **[IP/MASK OF YOUR INSTITUTION]** is the network address and netmask for your school, business, etc. formatted 123.45.67.0/24. **[EXTERNAL INTERFACE]** is the interface connected to the outside world, such as eno1. **[INTERNAL INTERFACE]** is the interface connected to the internal compute node network, such as eno2.

As with all of the services we will configure, these changes should be added to your ever growing kickstart file. Note that when adding firewalld commands to your kickstart file in the `%post` section, you need to use `firewall-offline-cmd` instead of `firewall-cmd`. In addition, `--remove-service` will have to be changed to `--remove-service-from-zone`, and `--permanent` is not necessary. For example, the command

```
firewall-cmd --permanent --zone=internal --remove-service=ipp-client
```

becomes

```
firewall-offline-cmd --zone=internal --remove-service-from-zone=ipp-client
```

# Reserved Address DHCP

Sooner or later a compute node will have problems. To associate physical devices with IP addresses one must configure DHCP to give out IP addresses based on hardware MAC addresses.

The following procedure logs the MAC addresses of machines that receive an IP address over DHCP in the order requested, then uses this information to set up DHCP to give each node the same number each time.

To associate IP addresses with specific computers:

1. If it exists, delete the file `/var/lib/dhcpd/dhcpd.leases`. Whether or not it existed, create it as a new file
   ```
   # touch /var/lib/dhcpd/dhcpd.leases
   ```
2. Power off all compute nodes
3. Power on the nodes in order
4. On the head node view the file `/var/lib/dhcpd/dhcpd.leases`. This should contain entries with the IP and MAC addresses of recently connected compute nodes
5. Append entries to the very bottom of `/etc/dhcp/dhcpd.conf` for each node, associating the node number with the order they appear in the file. For example:
   ```
   host name01 {
          hardware ethernet 00:04:23:c0:d5:5c;
          fixed-address 192.168.1.101;
   }
   ```
   To automate this task, use the following commands, either run from the command line or in a script. Tweak as necessary for your naming scheme:
   ```
   #!/bin/sh
   index=1
   for macaddr in `cat /var/lib/dhcpd/dhcpd.leases | grep 'ethernet' | sed
   's/.*hardware ethernet //'`; do
          printf 'host name%.2d {\n\thardware ethernet %s\n\tfixed-address
   192.168.1.1%.2d;\n}\n' "$index" "$macaddr" "$index"
   ```

```
        index=`expr $index + 1`
done
```

This script will dump the output to the terminal where it can be copy/pasted into `dhcpd.conf`. If you are running a terminal that doesn't support copy/paste you can instead redirect the output.

6. Reboot the cluster and ensure that this change takes effect

As with all configuration changes in this article, make sure this revised `dhcpd.conf` file finds its way into the head node kickstart file.

# /etc/hosts

The `/etc/hosts` file is used to pair hostnames with IP addresses. The general format is:

```
XXX.XXX.XXX.XXX     hostname.domain            shorthostname
```

The shorthostname field is optional, but saves typing in many situations. Make sure that the first line is as follows since it is required for the proper function of certain linux programs:

```
127.0.0.1           localhost.localdomain      localhost
```

After this line will be a mapping for the head node's fully qualified domain name to its external IP address:

```
123.45.67.89        name.university.edu        name
```

The rest of the file will contain mappings for every node on the internal network. For example, a cluster with a head node and two compute nodes will have a `/etc/hosts` as follows:

```
127.0.0.1           localhost.localdomain      localhost
123.45.67.89        name.university.edu        name
192.168.1.100       name00
192.168.1.101       name01
192.168.1.102       name02
```

Remember that 123.45.67.89 and 192.168.1.100 both correspond to the same machichine (the head node), but over different network adapters. The hosts file should be identical on the head node and compute nodes. As always, add these modifications to your kickstart files.

# NFS

NFS (Network File System) is used to share files between the head node and the compute nodes. We already configured it in the previous article, but this section goes more in depth. The general format of the `/etc/exports` configuration file is:

```
/exportdir    ipaddr/netmask(options)
```

To configure nfs on your cluster:

1. Modify `/etc/exports` on the head (or storage) node to share `/home` and `/admin`
   ```
   /home        192.168.1.100/255.255.255.0(rw,sync,no_root_squash)
   /admin       192.168.1.100/255.255.255.0(ro,sync,no_root_squash)
   ```
   This allows read/write access to /home, and read only access to /admin.
2. Restart nfs on the head node
   ```
   # systemctl restart nfs
   ```
3. Import the shares on the compute nodes by appending the following lines to `/etc/fstab`:
   ```
   name00:/home        /home        nfs    rw,hard,intr 0 0
   name00:/admin       /admin       nfs    ro,hard,intr 0 0
   ```

`name00` is the name of the head node, and `/home` is one of the shares defined in `/etc/exports` on the head node. The second `/home` is the location to mount the share on the compute node, and nfs lets Linux know that it should use nfs to mount the share. The remaining items on the line are options that specify how the mountpoint is treated.

4. The shares will be mounted on the compute nodes automatically on boot up, but may be mounted manually as follows:
```
# mount /home
# mount /admin
```

As always, once you have tested your configuration using one or two compute nodes, modify both kickstart files so that you don't have to manually apply it to all of them!

# Internal Access - SSH and RSH

SSH (Secure SHell) and RSH (Remote SHell) both allow access between nodes. In most situations when using Linux, ssh should be used because it uses encryption, making it much more difficult for a malicious person to gain unauthorized access. Trusted networks like the internal network in the cluster are exceptions to this rule because everything is under the protective shelter of the head node. As such, security can and should be more relaxed. It's all one big machine, after all.

Since ssh was built with security in mind, connections require a heftier authentication overhead than is the case for rsh. For this reason many people building high performance clusters that use multiple nodes to run a given job will favor rsh for its low latency communication. However, this can be a moot point. When using parallelization software such as OpenMPI, ssh or rsh is only used to start jobs and OpenMPI handles the rest of the communication.

Many nerd wars have been fought about using ssh vs. rsh in a cluster. This isn't the place to duke them out, so we provide instructions for both ssh and rsh in this section. For the rest of this article we'll assume that you chose the ssh route, but with some minor modifications you can make everything work with rsh as well.

## SSH

By default, ssh requires a password in order to access another machine. However, it can be configured to use rsa keys instead. By doing this, you will be able to ssh between machines without using passwords.

This step is absolutely vital since most cluster software such as slurm (addressed later in this article) assume passwordless ssh for communication. Additionally, cluster administration can be a pain when constantly juggling passwords around. Once an administrator or user has access to the head node, he or she should be able to access any other node within the cluster without providing any additional authentication.

For each user that you desire to have passwordless ssh (this includes root):

1. Start by generating rsa keys. As the user for which you are setting up passwordless ssh, execute the command
```
ssh-keygen
```
Accept the default location, `~/.ssh/id_rsa`, and leave the passphrase empty. If you supply a passphrase, they key will be encrypted and a passphrase will be necessary to use it, which defeats the purpose.

2. Copy the contents of id_rsa.pub to authorized_keys
   `cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys`

When you set up passwordless ssh for root, you will need to share `/root/.ssh` over nfs for the compute nodes. Since `/home` was shared in the nfs section of this article, this does not need to be done for regular users. Edit `/etc/exports` and make `/root/.ssh` a read only nfs share. On a single compute node (for testing purposes) add `/root/.ssh` to `/etc/fstab` and mount it. You should now be able to ssh as root in both directions between the head node and the compute node.

You may have noticed something annoying. The first time you ssh'd from the head node to a compute node as root, whether just now or earlier on in your cluster building, ssh complained that the authenticity of that compute node couldn't be established and prompted you to continue. You said 'yes' and went on your merry way, and ssh hasn't complained to you since. However, now it's upset again, and every time you ssh from a compute node back to the head node you get something like this:

```
The authenticity of host 'name00 (192.168.1.100)' can't be established.
ECDSA key fingerprint is 01:23:45:67:89:ab:cd:ef:01:23:45:67:89:ab:cd:ef.
Are you sure you want to continue connecting (yes/no)? yes
Failed to add the host to the list of known hosts (/root/.ssh/known_hosts).
```

The reason adding the host fails (causing you to go through this dialog every time you connect) is because the nfs share is read only. One way to fix this is to make it read/write, but that only solves half the issue. With that solution, you'd still have to accept this message manually for every node, which is unacceptable for scalability. Instead, edit `/etc/ssh/ssh_config` on all nodes, and after the line that looks like `Host *`, insert the following:

```
StrictHostKeyChecking no
UserKnownHostsFile=/dev/null
LogLevel error
```

If you omit the log level modification, ssh will warn you every time you connect that it's adding the host to the nonexistent (`/dev/null`) list of known hosts. While this isn't a major problem, it can fill system logs with non-issues, making debugging future problems more difficult.

When configuring the root user, you may need to edit `/etc/ssh/sshd.conf` on all nodes to allow passwordless root logins. Make sure it has the following values:

```
PermitRootLogin without-password
PubkeyAuthentication yes
```

Remember to continue adding these changes to your kickstart files!

# RSH

If you wish to use rsh instead of ssh because it doesn't needlessly encrypt communication, you can do the following. On each node in the cluster,

1. Install rsh. On the head node you can do so using yum:
   `# yum install rsh rsh-server`
   On the compute nodes you will have to add these to the packages section of the kickstart file and reinstall.
2. Append the following lines to `/etc/securetty`
   ```
   rsh
   rexec
   rsync
   rlogin
   ```

3. Create the file `/root/.rhosts` where each line follows the pattern `[HOSTNAME] root`. For example,
```
name00 root
name01 root
name02 root
```
4. Create the file `/etc/hosts.equiv`, in which each line is a hostname. For example,
```
name00
name01
name02
```
5. Enable and start the sockets
```
# systemctl enable rsh.socket
# systemctl enable rexec.socket
# systemctl enable rlogin.socket
# systemctl start rsh.socket
# systemctl start rexec.socket
# systemctl start rlogin.socket
```
6. You should now be able to access any computer from any other computer in the cluster as any user (including root) by executing
```
$ rsh [HOSTNAME]
```

Add these changes to your kickstart files.

# btools

Btools are a set of scripts used to automate the execution of commands and tasks across all compute nodes in a cluster. While "atools" would be the commands themselves typed in by some poor system administrator, and "ctools" (which actually exist) are part of a complex gui cluster management suite, btools are short scripts that fit somewhere in the middle. The listing of btools (bsh, bexec, bpush, bsync) and required support files (bhosts, bfiles) follows. These should be located on the head node. Feel free to modify them as needed.

## btool files

A few support files are required for the rest of the tools to function. Bhosts (Listing 1) contains the list of hostnames of all compute nodes, allowing tools that perform operations across the cluster to iterate over these hosts. Bfiles (Listing 2) contains the names of files that define the users on the system. If a script copies these files from the head node to all compute nodes, any users on the head node will be recognized on the compute nodes as well.

Listing 1. `/usr/local/sbin/bhosts`
```
name01
name02
```

Listing 2. `/usr/local/sbin/bfiles`
```
/etc/passwd
/etc/group
/etc/shadow
/etc/gshadow
```

## btool commands

Bsh (Listing 3) loops through the hosts in bhosts, executing ssh <some command> for each. Another tool, bexec (Listing 4), is similar to bsh in that it executes commands over all nodes, but

it executes them in parallel. While bsh waits for the first node to finish before moving on to the second, bexec gets them all started, then collects and displays the logs for the operations.

Besides executing commands, it is often useful to copy files to all nodes. Bpush (Listing 5) copies a file to all compute nodes. Similarly to bexec, it executes simultaneously, then displays logs.

Finally, bsync (Listing 6) copies the files defined in bfiles to all compute nodes. This causes all users on the head node to be users on the compute nodes as well.

Listing 3. `/usr/local/sbin/bsh`

```
#!/bin/sh
# bsh - broadcast ssh
for host in `cat /usr/local/sbin/bhosts`; do
       echo "*****${host}*****"
       ssh ${host} $*
done
```

Listing 4. `/usr/local/sbin/bexec`

```
#!/bin/sh
# bexec - broadcast ssh concurrently
# The total number of nodes (determined dynamically)
nhost=0
# Run the command on each node, logging output
for host in `cat /usr/local/sbin/bhosts`; do
       logfile="/tmp/${host}.$$.log"
       echo "***** ${host} *****" > $logfile
       ssh ${host} $* >> $logfile &
       pids[nhost]=$!
       let nhost=nhost+1
done
# Wait for all processes to finish
for i in `seq 0 $nhost`; do
       wait ${pids[$i]}
done
# Concatenate the results and cleanup
for host in `cat /usr/local/sbin/bhosts`; do
       logfile="/tmp/${host}.$$.log"
       cat $logfile
       rm $logfile
done
```

Listing 5. `/usr/local/sbin/bpush`

```
#!/bin/sh
# bpush - copy file(s) to nodes
# The total number of nodes (determined dynamically)
nhost=0
# Run the command on each node, logging output
for host in `cat /usr/local/sbin/bhosts`; do
       logfile="/tmp/${host}.$$.log"
       echo "***** ${host} *****" > $logfile
       scp $1 ${host}:$2 >> $logfile &
       pids[nhost]=$!
       let nhost=nhost+1
done
# Wait for all processes to finish
for i in `seq 0 $nhost`; do
       wait ${pids[$i]}
done
```

```
# Concatenate the results and cleanup
for host in `cat /usr/local/sbin/bhosts`; do
       logfile="/tmp/${host}.$$.log"
       cat $logfile
       rm $logfile
done
```

Listing 6. `/usr/local/sbin/bsync`

```
#!/bin/sh
# bsync - copy user files to nodes
for host in `cat /usr/local/sbin/bhosts`; do
       echo "Synching ${host}"
       for file in `cat /usr/local/sbin/bfiles`; do
              echo "Copying ${file}"
              rsync ${file} ${host}:${file}
       done
done
```

Each of the executable btools (bsh, bexec, bpush, and bsync) needs to be made executable before they can be run from the command line:

```
# chmod +x /usr/local/sbin/tool_name
```

This collection of btools is useful for a variety of administrative tasks. For example, bsh can be used to quickly **run a command on all nodes**, perhaps to check that they all have access to an nfs share:

```
# bsh ls /admin
```

A word of caution: bash doesn't pass some things such as redirection (>, <, |) as parameters to executables. If you want to use bsh to `echo "something" >> /some/file` on each compute node you will need quotes like this:

```
# bsh 'echo "something" >> /some/file'
```

For tasks that take more computation time, such as installing software, bexec should be used. For example, when we get a yum local repository set up, you can use bexec to install software from it.

```
# bexec yum -y install package_name
```

bpush is used to **copy a file to all nodes**. For example, if you want to test how a configuration works on all compute nodes without putting it in a kickstart and reinstalling the cluster, simply bpush it to all nodes and reload the relevant service

```
# bsh mv /etc/service/service.conf /etc/service/service.conf.000
# bpush /path/to/service.conf /etc/service/service.conf
# bexec systemctl restart service
```

Finally, every time you add a user to the cluster by using `useradd` on the head node, make sure to run bsync so that the **new users have access to all nodes**.

```
# useradd user_name
# passwd user_name
# bsync
```

# NTP

NTP is a protocol used by chrony to synchronize the time on a computer with some external source. It should be set up on both the head node to synchronize with the outside world, and on the compute nodes to synchronize with the head node. It is vital that the entire cluster is in agreement on the time so there aren't any errors regarding timestamps on files shared over the internal network.

The following procedure sets up the head node as a chrony server for the compute nodes:

1. Find a working timeserver (if your institution has its own, that's the best!) and add it to `/etc/chrony.conf` on the head node in the following format:
   ```
   server 192.43.244.18 #time.nist.gov
   ```
   The comment at the end of the server line is purely optional but can be helpful when looking at the file. If you have chosen more than one server you may add them in a similar fashion.
2. Comment out the
   ```
   server       X.centos.pool.ntp.org iburst
   ```
   lines on both the head node and compute nodes as they will interfere with the configuration.
3. Allow your compute nodes to be clients of the head node's ntp server by uncommenting the line on the head node:
   ```
   allow 192.168/16
   ```
4. Set a compute node to use the head node as its time server by adding the following line to `/etc/chrony.conf` on the compute node.
   ```
   server 192.168.1.100 # head node
   ```
5. Enable and start chrony on both head and compute nodes.
   ```
   # systemctl enable chronyd
   # systemctl start chronyd
   ```
6. After a few minutes have passed and NTP has had a chance to synchronize, execute the following commands to test your ntp configuration.
   ```
   # chronyc tracking
   # chronyc sources -v
   ```
   The -v option on the sources command prints cleverly formatted explanations for each column in the output. Together, these commands print out debugging information about the time server connection. They can be run on the head node to show info about your external time server(s) or on a compute node to print info about the time server on the head node.

We may be sounding like a broken record, but be sure to add all this to the kickstart files.

# Yum Local Repository

Creating a local repository on the head node is useful for installing software that isn't available from the installation media and installing updates to the compute nodes. While this could be achieved in a pinch with traffic forwarding, that method not only poses a security risk but it also bogs down the external network with redundant requests for the same software from each compute node. Instead, it is possible to download some software packages to the head node once, then set up the head node as a yum repository for the compute nodes to access.

The following procedure sets up the head node as a yum server and mirrors a repository onto the head node:

1. Prepare a good spot to store a CentOS repository
   ```
   # mkdir -p /admin/software/repo/
   ```
2. Sync with a mirror.
   ```
   # rsync -azHhv --delete some_mirror.org::CentOS/7* /admin/software/repo/
   ```
   Note the syntax for specifying the folder to be synchronized. If the folder is `rsync://mirrors.liquidweb.com/CentOS/7`, it would be written as `mirrors.liquidweb.com::CentOS/7`

This should use about 40G of disk space and take several hours. You can use the same command to update your local copy of the repo. Updates will proceed much more quickly than the first copy.

3. Edit the yum configuration files for both the head node and compute nodes in `/etc/yum.repos.d/` so that the head node will use itself as the update server, and the compute nodes will use their nfs mount of `/admin` as the update server. Change the line
   `#baseurl=http://mirror.centos.org/centos/$releasever...`
   to
   `baseurl=file:/admin/software/repo/$releasever...`
   and comment out all mirrorlist lines. This can be automated using the following sed commands:
   ```
   # sed -i.000 "s|#baseurl=http://mirror.centos.org/centos|baseurl=
   file:/admin/software/repo|" /etc/yum.repos.d/*.repo
   # sed -i "s|mirrorlist|#mirrorlist|" /etc/yum.repos.d/*.repo
   ```
   This should be performed on the head node and compute nodes (using bsh). It should also happen in their kickstarts. If you have installed additional repos (such as epel on the head node for ganglia) make sure not to modify their .repo files, otherwise yum will have trouble finding those repos on your hard drive!
4. Update the head node and compute nodes using the head node as a software source
   ```
   # yum -y upgrade
   # bexec yum -y upgrade
   ```

## Create your own repo

There are times when you will need to install software packages to your compute nodes that are not available in the repo that you cloned, for example, ganglia from epel. In this case, it is not efficient to clone the entire epel repository for only a few software packages. Thus, the possible solutions are to either download the rpm files to the head node, `bpush` them to the compute nodes, and `bexec rpm -ivh /path/to/software.rpm`, or to create your own specialized repository and install them using yum.

To create your own repository:

1. Create a directory to house the repo in the head node
   ```
   # mkdir /admin/software/localrepo
   ```
2. Download some rpms to populate your local repo. For example, get the ganglia compute node packages from epel, a third party repository.
   ```
   # cd /admin/software/localrepo
   # yum install epel-release
   # yumdownloader ganglia ganglia-gmond ganglia-gmetad libconfuse
   ```
3. Create the repo metadata
   ```
   # createrepo /admin/software/localrepo
   ```
4. Create `/etc/yum.repos.d/local.repo` containing the following:
   ```
   [local]
   name=CentOS local repo
   baseurl=file:///admin/software/localrepo
   enabled=1
   gpgcheck=0
   protect=1
   ```
5. Push `local.repo` to your compute nodes
   ```
   # bpush /etc/yum.repos.d/local.repo /etc/yum.repos.d/
   ```
6. Install the desired software on compute nodes
   ```
   # bexec yum -y install ganglia-gmond ganglia-gmetad
   ```
7. If you want to add software to the local repo, simply repeat steps 2 and 3. Note that yum keeps a cache of available software, so you may need to run

```
# bexec yum clean all
```
before you can install added software.

Be sure that the modifications to the files in `/etc/yum.repos.d/` are added to your kickstart files. The other changes occur in `/admin`, which remains unchanged during a reinstall.

# Ganglia

It's important to be able to monitor the performance and activity of a cluster to identify nodes that are having problems or to discover inefficient uses of resources. Ganglia is a cluster monitoring software suite that reports the status of all the nodes in the cluster over http.

Before installing ganglia, you must have httpd installed and have the ganglia, ganglia-gmond, ganglia-gmetad, and libconfuse packages available from the local repository.

To install Ganglia on the cluster:

1. Install httpd
   ```
   # yum install httpd
   # systemctl enable httpd
   # systemctl start httpd
   ```
2. On the head node, install ganglia-gmetad, ganglia-gmond, and ganglia-web
   ```
   # yum -y install ganglia ganglia-gmetad ganglia-gmond ganglia-web
   ```
3. On compute nodes, install ganglia-gmetad and ganglia-gmond
   ```
   # bexec yum -y install ganglia ganglia-gmetad ganglia-gmond
   ```
4. On the head node, backup and edit `/etc/ganglia/gmond.conf` such that
   - In the `cluster` block, `name` is something you will recognize
   - In the `udp_send_channel` block, `mcast_join` is commented out and the line `host = 192.168.1.100` (internal ip of head node) is added
   - In the `udp_recev_channel` block, `mcast_join` and `bind` are both commented out.

   Push the modified `gmond.conf` file to all compute nodes
   ```
   # bpush /etc/ganglia/gmond.conf /etc/ganglia/
   ```
5. Enable and start gmetad and gmond
   ```
   # systemctl enable gmetad
   # systemctl start gmetad
   # systemctl enable gmond
   # systemctl start gmond
   # bexec systemctl enable gmetad
   # bexec systemctl start gmetad
   # bexec systemctl enable gmond
   # bexec systemctl start gmond
   ```
6. In `/usr/share/ganglia/conf.php`, between the `<?php` and `?>` tags, add the line
   ```
   $conf['gweb_confdir'] = "/usr/share/ganglia";
   ```
7. Edit `/etc/httpd/conf.d/ganglia.conf` such that it reflects the following:
   ```
   Alias /ganglia /usr/share/ganglia

   <Directory "/usr/share/ganglia">
          AllowOverride All
          Require all granted
   </Directory>
   ```
   And restart httpd
   ```
   # systemctl restart httpd
   ```
8. Monitor your cluster from a web browser by going to http://name.university.edu/ganglia.

Be sure to update your kickstart files.

# SLURM

In a cluster environment, multiple users share resources. Batch queueing systems run jobs on appropriate hardware resources and queue jobs when resources are unavailable. Some examples of batch queueing systems include PBS, Torque/Maui, SGE (now Oracle Grid Engine), and SLURM.

SLURM (Simple Linux Utility for Resource Management) is a tool for executing commands on available compute nodes. SLURM uses an authentication agent called munge.

To configure and install SLURM and munge:

1. On the head node, add slurm and munge users
```
# groupadd munge
# useradd  -m -d /var/lib/munge -g munge  -s /sbin/nologin munge
# groupadd slurm
# useradd  -m -d /var/lib/slurm -g slurm  -s /bin/bash slurm
```
And sync them to the compute nodes
```
# bsync
```
2. Slurm uses an authentication agent called munge, which is available from the epel repository.
```
# yum -y install munge munge-libs munge-devel
```
Munge must be installed to the compute nodes as well. See the section titled Yum Local Repository for details, then perform the installation using bexec.
3. Generate a key to be used by munge across the cluster
```
# dd if=/dev/urandom bs=1 count=1024 > /etc/munge/munge.key
# chown munge:munge /etc/munge/munge.key
# chmod 400 /etc/munge/munge.key
```
The same key must be in the `/etc/munge/` directory on all compute nodes. While it is possible to propagate it to all of them, it can be more convenient (vital for reinstallations, in fact) to make the entire directory a read only nfs share. See the section above on nfs for details.
4. Enable and start the munge service on all nodes. Test that munge is installed correctly.
```
# munge -n | ssh name01 unmunge
```
5. On the head node, install the packages necessary to build slurm
```
# yum -y install rpm-build gcc openssl openssl-devel pam-devel numactl numactl-devel hwloc hwloc-devel lua lua-devel readline-devel rrdtool-devel ncurses-devel gtk2-devel man2html libibmad libibumad perl-Switch perl-ExtUtils-MakeMaker mariadb-server mariadb-devel
```
6. On the head node, download and build the latest slurm distribution. In the following commands, substitute the latest version of slurm for **VERSION**. At the time of writing this article, the latest version is 16.05.7.
```
# cd /tmp
# curl -O https://www.schedmd.com/downloads/latest/slurm-VERSION.tar.bz2
# rpmbuild -ta slurm-VERSION.tar.bz2
```
7. Copy the built rpms to your local repository and update its metadata
```
# cp /root/rpmbuild/RPMS/x86_64/*.rpm /admin/software/localrepo
# createrepo /admin/software/localrepo
```
Make yum on all machines aware that this change occurred
```
# yum -y clean all
# bexec yum -y clean all
```
8. On the all nodes, install the following packages from the local repo
```
# bexec yum -y install slurm slurm-devel slurm-munge slurm-perlapi slurm-plugins slurm-sjobexit slurm-sjstat slurm-seff
```
And on the head node only, install the slurm database
```
# yum -y install slurm-slurmdbd slurm-sql
```

9. On all nodes, create and set permissions on slurm directories and log files
```
# mkdir /var/spool/slurmctld /var/log/slurm
# chown slurm: /var/spool/slurmctld /var/log/slurm
# chmod 755 /var/spool/slurmctld /var/log/slurm
# touch /var/log/slurm/slurmctld.log
# chown slurm: /var/log/slurm/slurmctld.log
```
10. On the head node, create the configuration file `/etc/slurm/slurm.conf`
```
ControlMachine=name
ReturnToService=1
SlurmUser=slurm
StateSaveLocation=/var/spool/slurmctld
# LOGGING AND ACCOUNTING
ClusterName=cluster
SlurmctldLogFile=/var/log/slurm/slurmctld.log
SlurmdLogFile=/var/log/slurm/slurmd.log
# COMPUTE NODES
NodeName=name[01-99] CPUs=1 State=UNKNOWN
PartitionName=debug Nodes=name[01-99] Default=YES MaxTime=INFINITE State=UP
```
This file must be the same across all nodes. To ensure it is, make the /etc/slurm directory a nfs share like you did for munge. Alternatively, you can place this file in `/home/export/slurm` (which already is a nfs share) and symbolically link it to `/etc/slurm` on all nodes
```
# ln -s /home/export/slurm/slurm.conf /etc/slurm/slurm.conf
# bsh ln -s /home/export/slurm/slurm.conf /etc/slurm/slurm.conf
```
We didn't use this trick when installing munge because munge checks that it's reading a regular file rather than a symbolic link.
11. On the head node, enable and start slurmctld
```
# systemctl enable slurmctld
# systemctl start slurmctld
```
And on the compute nodes, enable and start slurmd
```
# bsh systemctl enable slurmd
# bsh systemctl start slurmd
```
12. Test the system. First submit an interactive job on one node:
```
$ srun /bin/hostname
```
Then submit a batch job. First, in a home directory create the file job.sh
```
#!/bin/bash
sleep 10
echo "Hello, World!"
```
And execute the file using the command
```
$ sbatch job.sh
```
Before 10 seconds pass (the amount of time our sample job takes to run), check the job queue to make sure something's running
```
$ squeue
```
Which should display something like this
```
JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
    1     debug   job.sh     user  R       0:07      1 name01
```
Finally, if /home is nfs mounted read/write, you should see the job's output file slurm-1.out in your current directory with the contents
```
Hello, World!
```

# Next Steps

At this point the cluster is operational. You can run jobs on nodes using SLURM and monitor the load using Ganglia. You also have many tools and tricks for performing administrative tasks ranging from adding users to reinstalling nodes. A few things remain, most notably installing

application specific software, adding parallelization libraries like OpenMPI for running one job over multiple nodes, and employing standard administrative good practices.

When installing application software to a single directory (for example, `/usr/local/app_name/`), install it on the head node. Then share it to the compute nodes by moving the entire installation to `/home/export` and symbolically linking it to the install location:

```
# mv /usr/local/app_name /home/export/app_name
# ln -s /home/export/app_name /usr/local/app_name
# bsh ln -s /home/export/app_name /usr/local/app_name
```

When installing libraries, they are often in the standard repositories that you cloned to the head node when creating a local repository. This is the case with OpenMPI. Installation is as easy as executing a yum install:

```
# yum -y install openmpi openmpi-devel
# bexec yum -y install openmpi openmpi-devel
```

If you need a library that is not included in the standard repositories but is still packaged for CentOS, you can download it to your local repo following the procedure discussed in the Create your own repo section of this article. If the library isn't packaged at all for CentOS (perhaps you compiled it from source) you can still `bpush` it to the compute nodes:

```
# bpush /path/to/library.so /path/to/library.so
```

To prevent a computational Tragedy of the Commons, you will want to protect the communal resources, including the head node's disk and CPU. To keep a user from hogging the entire home partition, enforce a quota on disk usage. To prevent long jobs from running on the head node (that's what compute nodes are for), you can write a "reaper" daemon that checks for long running user processes and kills them.

Another useful tool to have in place is the ability to send email from the cluster. For example, many RAID devices have corresponding software that will send an email when a disk goes down. In order for this email to make its way to your inbox, the cluster must be configured to forward the email to the correct recipient(s).

While outside the scope of this article, it's important for a production cluster to be backed up regularly. You'll want a cron job that tars up the entire home directory and transfers it over the network to a backup machine, preferably in a different building.

# Conclusion

Computer clusters are important tools for massively parallelizable tasks (high performance computing) and for running many jobs concurrently (high throughput computing). All too often the setup and maintenance of clusters is left to specialists, or even worse, to complex “magical” configuration software. In the last three articles we wrote some complex configuration scripts ourselves, but we understand each step so that the process is stripped of all magic.

The resulting cluster is flexible enough to be used for many different computationally intensive problems. Furthermore, because of the redundancy built into the hardware and the ease of reinstalling compute nodes, the cluster is reliable as well. Our production cluster has been in operation for over 10 years, has seen multiple hardware and software upgrades, and still functions reliably.

In a few years you may decide to upgrade by adding some shiny new compute nodes, which won't be a problem since all you will have to do is tweak your kickstart file to take advantage of the new capacities or capabilities. Or perhaps you will want to upgrade to the latest operating system version. Since you used standard technologies and methodologies to build your own cluster (BYOC), future transitions should proceed as smoothly as the original installation!