CONSTRUCTION OF CLUSTERS FOR SCIENTIFIC COMPUTING

Presented by Stephen Lien Harrell
SLH@PURDUE.EDU
WHAT IS THIS WORKSHOP ABOUT

• This is:
  • A guide to building a simple scientific computing cluster
  • An outline for deploying a high performance computing cluster
  • Targeted at a corporate or academic IT person who is tasked with creating a scientific computing cluster for the first time

• This is NOT:
  • An exact blueprint for creating a cluster that meets your needs
  • An example of best practices throughout the industry
  • A complete solution at scale
WHO AM I, AND WHY AM I HERE

- Stephen Lien Harrell
  - Scientific Applications Analyst in Research Computing at Purdue University
  - Specialization in imaging and configuration management for moderately large (500-2000 machine) clusters and HPC in undergraduate education.
  - Teach similar classes to undergraduates at Purdue University
GOALS AND CAVEATS

• Goals
  • Illuminate the technologies needed to build a scientific computing cluster
  • Show how the technologies fit together
  • Show an iterative and scalable configuration management model

• Caveats
  • We have little time for deep dives
  • I will be using simpler technologies in some places than industry standards.
  • Hardened security is out of our scope.
  • Our cluster will be built in virtual machines which will be different in design and technologies than hardware clusters.
You must have a laptop or computing device that has an internet connection, ssh terminal and a modern browser.

You must have access to EAFIT's virtual machines.

Must be familiar with at least one text editor on the Linux command line.
TASK LIST FOR THE WORKSHOP

• Bootstrap Puppet
• Firewalls
• DNS
• Shared Storage
• Environment Modules
• Accounts
• Scheduler and Resource Manager
• Node Health Checks
• Run MPI pi calculator and HPL benchmark
• Log aggregation
• Nagios
I have created snippets of code and commands to help us move along quickly.

Each slide will be tagged with the snippet name we will be working with.

The snippets are available at
  * [http://web.rcac.purdue.edu/~sharrell/buildacluster/](http://web.rcac.purdue.edu/~sharrell/buildacluster/)

These slides are intentionally incomplete without these snippets.
SETUP HEAD NODE
We will be using a subset of the puppet configuration management tool

Puppet has many capabilities and language abstractions

My main goals are readability and manageability after the class is over
**SVN AND PUPPET PRIMER**

- **svn up**
  - Update an existing repository with the current commits

- **svn di**
  - Print the diff of your current tree vs the remote tree at your current commit

- **svn ci**
  - Check in changes

- **puppet apply**
  - This applies any changes in the puppet tree, this is followed by the path of site.pp.
  - You may want to make an alias for “puppet apply /etc/puppet/manifests/site.pp”
• SSH into the node as root
  • `ssh root@EAFIT-VM-IP`

• Install puppet and its dependencies
  • `yum install http://yum.puppetlabs.com/el/6.0/products/x86_64/facter-1.7.6-1.el6.x86_64.rpm`
  • `yum install http://yum.puppetlabs.com/el/6.0/dependencies/x86_64/ruby-augeas-0.4.1-3.el6.x86_64.rpm`
  • `yum install http://yum.puppetlabs.com/el/6.0/dependencies/x86_64/ruby-shadow-2.2.0-2.el6.x86_64.rpm`
  • `yum install http://yum.puppetlabs.com/el/6.0/products/x86_64/puppet-2.7.26-1.el6.noarch.rpm`

• Install git, subversion, apache with mod ssl and vim
  • `yum -y install git mod_ssl vim subversion`
• Create subversion repository and initialize puppet tree
  • `mkdir /var/svn`
  • `svnadmin create /var/svn/puppet`
  • `svn import /etc/puppet file:///var/svn/puppet -m "Initial puppet import."`

• Move original puppet directory out of the way and check out version controlled puppet
  • `mv /etc/puppet /etc/puppet.orig`
  • `svn co file:///var/svn/puppet /etc/puppet`
• Install puppet modules we will be using throughout the setup
  • for package in puppetlabs-apache puppetlabs-firewall spiette-selinux AlexCline-mounts torrancew-account saz-resolv_conf saz-rsyslog petems-swap_file; do puppet module install $package;done
  • git clone https://github.com/rlex/puppet-dnsmasq.git /etc/puppet/modules/dnsmasq
  • git clone https://github.com/haraldsk/puppet-module-nfs.git /etc/puppet/modules/nfs
• Set editor then add and check in modules
  • export EDITOR=vim
  • svn add /etc/puppet/modules/*
  • svn ci /etc/puppet

• Create directory for puppet configs and add and check in
  • mkdir /etc/puppet/manifests
  • touch /etc/puppet/manifests/site.pp
  • touch /etc/puppet/hiera.yaml
  • svn add /etc/puppet/manifests
  • svn add /etc/puppet/hiera.yaml
  • svn ci /etc/puppet
• Generate self-signed certificate for use with apache and subversion
  • mkdir /etc/httpd/ssl
  • openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/httpd/ssl/apache.key -out /etc/httpd/ssl/apache.crt
    • Common Name will be head.cluster

• Create http user for subversion remote access
  • htpasswd -c /etc/httpd/auth_user_file root
  • chown apache:apache /etc/httpd/auth_user_file
• /etc/puppet/manifests/site.pp

  • This is where we will be doing most of our configuration of the cluster

  • We want to minimize the amount of software setup commands that we run on the command line

  • We want to maximize the documentation and scripting of what needs to be done on the machines

  • We will be applying updates with puppet apply
    • This avoids a lot of extra setup of puppet and lets us get into the things that matter for our cluster.
• Head Node
  • Puppet repository
  • Scheduling server

• Storage Node
  • Shared file system server

• Compute Node
  • Scheduling client
  • User libraries

```bash
class base_cluster { }
class head_node { }
class storage_node { }
class compute_node { }

# head node
node 'head.cluster', {
    include head_node
    include base_cluster
}

# storage node
node 'storage.cluster' {
    include storage_node
    include base_cluster
}

# compute nodes
node 'compute1.cluster', 'compute2.cluster' {
    include compute_node
    include base_cluster
}
```
• Boilerplate apache module instantiation

class { 'apache':
    default_conf_files => false,
    purge_configs => false,
}
class { 'apache::mod::dav_svn': }

• Define apache vhost

apache::vhost { 'head.cluster':
    port => 443,
    docroot => '/var/www/html/',
    ssl => true,
    ssl_cert => '/etc/httpd/ssl/apache.crt',
    ssl_key => '/etc/httpd/ssl/apache.key',
    custom_fragment => '  
<Location /puppet>
  AuthType Basic
  AuthName "Puppet Cluster Repository"
  AuthUserFile "/etc/httpd/auth_user_file"
  Require valid-user
  DAV svn
  SVNPath /var/svn/puppet/
</Location>'
}

002-subversion-puppet-repo
PUPPET SUBVERSION REPOSITORY
DEPLOYED BY PUPPET – PART 2

• puppet apply
  • Will get error message with fqdn of host
  • Change the hostname of the head node
  • Run the command "hostname head"
  • Add the line "HEADNODE_IP  head.cluster head" to /etc/hosts

• puppet apply

• fix permissions
  • chcon -R -h -t httpd_sys_content_t /var/svn/puppet
  • chown -R apache:apache /var/svn/puppet
• Add puppet apply on boot
  • file_line { 'puppet-apply-on-boot':
    path => '/etc/rc.d/rc.local',
    ensure => present,
    line => '/usr/bin/puppet apply /etc/puppet/manifests/site.pp',
  }

• Add a swap file
  • swap_file::files { 'default':
    ensure => present,
  }

• Add Exec path globally for older module compatibility
  • Exec { path => [ "/bin/", "/sbin/", "/usr/bin/", "/usr/sbin/" ] }

• Make SELinux not bother us that much
  • class { 'selinux':
    mode => 'permissive',
  }

• Add miscellaneous utilities to make this easier.
  • package { 'bind-utils':
    ensure => present,
  }
  • wget
  • lsof
  • mlocate
  • strace
  • telnet
  • netcat
  • screen

• puppet apply
**PUPPET FIREWALL PREPARATION**

- Make directory and file for new module in puppet
  - mkdir -p /etc/puppet/modules/my_fw/manifests
  - touch /etc/puppet/modules/my_fw/manifests/init.pp
  - svn add /etc/puppet/modules/my_fw/

- **Flush IPTables** to prevent ssh-blocking race condition
  - iptables -F
class my_fw {

    $ipv4_file = $operatingsystem ? {
        "debian" => '/etc/iptables/rules.v4',
        /(RedHat|CentOS)/ => '/etc/sysconfig/iptables',
    }

    firewall { "001 accept all icmp requests":
        proto => 'icmp',
        action => 'accept',
    }

    firewall { '002 INPUT allow loopback tcp':
        iface => 'lo',
        chain => 'INPUT',
        action => 'accept',
        proto => 'tcp',
    }

    firewall { '002 INPUT allow loopback udp':
        iface => 'lo',
        chain => 'INPUT',
        action => 'accept',
        proto => 'udp',
    }

    firewall { '000 INPUT allow related and established':
        state => ['RELATED', 'ESTABLISHED'],
        action => 'accept',
        proto => 'all',
    }

    firewall { '100 allow ssh':
        state => ['NEW'],
        dport => '22',
        proto => 'tcp',
        action => 'accept',
    }

    firewall { "998 deny all other requests":
        action => 'reject',
        proto => 'all',
        reject => 'icmp-host-prohibited',
    }

    firewall { "999 deny all other requests":
        chain => 'FORWARD',
        action => 'reject',
        proto => 'all',
        reject => 'icmp-host-prohibited',
    }
}

This will all go into: /etc/puppet/modules/my_fw/
manifests/init.pp

007-initpp-for-my_fw-module
Basic Firewall – Part 2

- **Back to site.pp**

- We will set the IPs for the entire cluster here, Update IPs for all 4 nodes now
  - $headnodeip='172.17.1.?'
  - $storagenodeip='172.17.1.?'
  - $computeoneip='172.17.1.?'
  - $computetwoip='172.17.1.?'

- Firewall Boilerplate
  - class base_cluster {
      resources {
          "firewall":
          purge => true
      }
      class { 'my_fw': }
      class { 'firewall': }
  }

- Create firewall rule for each machine to allow all machines to communicate freely.
  - firewall { '003 INPUT allow head ip':
      chain => 'INPUT',
      action => 'accept',
      proto => 'all',
      source => "${headnodeip}/32",
    }
  
  ... and repeat for the rest of the machines.

- Allow access to our web SVN tree from anywhere
  - firewall { '100 allow https access':
      state => ['NEW'],
      dport => 443,
      proto => tcp,
      action => accept,
    }

- **svn ci and puppet apply**
• Fix race condition
  • Class['dnsmasq'] -> Class['resolv_conf']

• DNSMasq module boilerplate
  • class { 'dnsmasq':
    interface         => 'lo',
    ...
  }

• Set outbound DNS server
  • dnsmasq::dnsserver { 'dns':
    ip => '192.168.1.5',
  }

• Hacky reverse name generation
  • $iparray_head = split($headnodeip,
    '[]
  )
  • $headnode_reverse = join(...,"")

• Set forward and reverse for the head node
  • dnsmasq::address { "head.cluster":
    ip => $headnodeip,
  }
  • dnsmasq::ptr {$headnode_reverse:
    value => 'head.cluster',
  }

• Setup resolv.conf to point to dnsmasq
  • class { 'resolv_conf':
    nameservers => ['127.0.0.1'],
    domainname => 'cluster',
  }

• `svn ci` and `puppet apply`
SETUP STORAGE
NODE
BOOTSTRAPPING STORAGE (AND COMPUTE) NODE(S)

• Login to the storage node
• Install Puppet and its dependencies
  • `yum install http://yum.puppetlabs.com/el/6.0/products/x86_64/facter-1.7.6-1.el6.x86_64.rpm`
  • …
• Install puppet, vim and subversion
  • `yum -y install puppet vim subversion`
• Remove default puppet configs
  • `rm -rf /etc/puppet`
• checkout puppet svn to /etc/puppet
  • `svn co https://LOCAL_HEADNODE_IP/puppet /etc/puppet/`
• Flush IPTables
  • `iptables –F`
• Change the hostname of the storage node
  • Run the command "hostname storage.cluster"
  • Add the line "domain cluster" to /etc/resolv.conf
  • Add the line "127.0.0.1 storage.cluster localhost" to /etc/hosts
• `svn ci` and `puppet apply`
• `reboot`
• Create /apps/ directory for shared software
  • file { "/apps":
      ensure => "directory",
    }
• Create NFS exports for /home and /apps
  • include nfs::server
  • nfs::server::export{ '/home/':
      ensure => 'mounted',
      clients => '172.17.1.0/24(rw,insecure,async,no_root_squash) localhost(rw)',
    }
  • nfs::server::export{ '/apps/':
      ensure => 'mounted',
      clients => '172.17.1.0/24(rw,insecure,async,no_root_squash) localhost(rw)',
      require => File['/apps']
    }
• **svn ci** and **puppet apply**
• Mount /home
  • mounts { 'storage server home':
    ensure => present,
    source => "${storagenodeip}:~/home",
    dest => '/home',
    type => 'nfs',
    opts => 'nofail,defaults,vers=3,rw,noatime',
  }
• Mount /apps
  • mounts { 'storage server apps':
    ensure => present,
    source => "${storagenodeip}:~/apps",
    dest => '/apps',
    type => 'nfs',
    opts => 'nofail,defaults,vers=3,rw,noatime',
    require => File['/apps'],
  }
• Add this for the head node and cluster node classes and puppet apply
SETUP COMPUTE NODES
Go back to bootstrapping storage slide and bootstrap the two nodes
In an academic or corporate environment you will most likely be using LDAP or something similar. This method is an easy way around having to setup an LDAP.

```perl
account {
  'login_name_here':
    home_dir => '/home/login_name_here',
  groups   => [ 'wheel', 'users' ],
  comment   => 'Full Name',
  uid => 500,
}
```

• This will allow us to have a UID consistent user everywhere without setting up a full accounting system.
• puppet apply
Environment modules can provide pluggable software.

Install basic Packages
- package { 'environment-modules':
  ensure => present,
}

package { 'gcc-c++':
  ensure => present,
}

package { 'gcc-gfortran':
  ensure => present,
}

OpenMPI Software
- cd /apps/
- wget openmpi-1.7.5.tar.gz
- tar xfvz openmpi-1.7.5.tar.gz

OpenMPI Module
- Create the directory for the module files
  - file { "/usr/share/Modules/modulefiles/openmpi":
    ensure => "directory"
  }

OpenMPI Software
- cd /apps/
- wget openmpi-1.7.5.tar.gz
- tar xfvz openmpi-1.7.5.tar.gz

OpenMPI Module
- Create the directory for the module files
  - file { "/usr/share/Modules/modulefiles/openmpi":
    ensure => "directory"
  }
• Create the .version file. This file contains the default version for the module.
"#%Module1.0
set ModulesVersion "1.7.5"
"
• Create the actual module file
#%Module1.0
module-whatis "invoke openmpi-1.7.5"
set version 1.7.5
set app openmpi
set modroot /apps/openmpi-1.7.5/
pread-path PATH "$modroot/bin"
pread-path LD_LIBRARY_PATH "$modroot/lib"
setenv MPI_HOME "$modroot"
setenv CC mpicc
setenv CXX mpiCC
setenv F77 mpif77
setenv FC mpif90
"
OPENBLAS MODULE

- OpenBLAS Software
  - cd /apps/
  - wget openblas-0.2.10.tar.gz
  - tar xfvz openblas-0.2.10.tar.gz

- OpenBLAS Module
  - Create the directory for the module files
    - file { "/usr/share/Modules/modulefiles/openblas":
      ensure => "directory"
    }

  - Create the .version file. This file contains the default version for the module.
    - "#%Module1.0 set ModulesVersion "0.2.10"
    - Create the actual module file
      - "#%Module1.0 module-whatis "invoke openblas-0.2.10"
      - set version 0.2.10
      - set app openblas
      - set modroot /apps/openblas-0.2.10/
      - prepend-path PATH \$modroot/bin
      - prepend-path LD_LIBRARY_PATH \$modroot/lib"
• Setup torque libs and files across the whole cluster
  • package { 'libxml2':
      ensure => present,
  }
  package { 'torque':
      ensure => 'installed',
      source => 'torque-4.1.7-1.adaptive.el6.x86_64.rpm',
      provider => 'rpm',
  }
  file { '/var/spool/torque/server_name':
      content => "head.cluster\n",
  }

• Install torque server and scheduler packages on head node
  • package { 'maui':
      ensure => 'installed',
      source => 'maui-3.3.1-x86_64-fpmbuild.rpm',
      provider => 'rpm',
      require => Package['torque']
  }
  • package { 'torque-server':
      ensure => 'installed',
      source => 'torque-server-4.1.7-1.adaptive.el6.x86_64.rpm',
      provider => 'rpm',
      require => Package['torque']
  }

• Setup services and config files for torque on the head node
  • service { "pbs_server":
      #ensure => "running",
      enable => "false",
  }
  • service {"maui.d":
      ensure => "running",
      enable => "false",
  }
  • file { '/var/spool/torque/server_priv/nodes':
      content => "compute1.cluster np=1\n      compute2.cluster np=1\n",
  }

• Puppet apply
• Stop Torque: /etc/init.d/pbs_server stop
• Run the Torque Setup: /usr/share/doc/torque-server-4.1.7/torque.setup
• Allow pbsnodes to work on the nodes
  • qmgr -c "set server managers = root@*.cluster"
• Change pbs_server and pbs_sched stanza to uncomment ensure running
• Setup torque on the compute nodes
  • package { 'torque-client':
    ensure => 'installed',
    source => 'torque-client-4.1.7-1.adaptive.el6.x86_64.rpm',
    provider => 'rpm',
    require => Package['torque']
  }

  service { "pbs_mom":
    ensure => "running",
    enable => "true",
    require => Package['torque-client'],
  }

  file { '/var/spool/torque/mom_priv/config':
    content => "\$pbsserver head.cluster
                  \$usecp */home /home
    ",
    require => Package['torque-client'],
    notify => Service['pbs_mom']
  }

  *svn ci* on head node and *svn up* on compute nodes, followed by *puppet apply*
Make sure that our compute nodes are free
• `pbsnodes -a`

• Start an interactive job
  • `su login_user`
  • `qsub -l`

• Start an interactive job with two nodes
  • `qsub -l -l nodes=2`

• Getting Debug Information
  • Show all jobs
    • `qstat -a`

  • Get information about specific job
    • `qstat {jobid}`
    • `tracejob {jobid}`

  • Show downed nodes
    • `pbsnodes --ln`

  • Important logs to check
    • `/var/spool/torque/server_logs/`
    • `*`
    • `/var/spool/torque/mom_logs/*`
• Install the NHC package
  package { 'warewulf-nhc':
    ensure => 'installed',
    source => 'http://
  warewulf.lbl.gov/downloads/repo/rhel6/
warewulf-nhc-1.3-1.el6.noarch.rpm',
    provider => 'rpm',
  }

• Run the health check at jobstart and offline the node if problems
  • \$node_check_script /usr/sbin/nhc
  • \$node_check_interval jobstart
  • \$down_on_error 0

• Add these lines to the existing mom_config file. Watch for the “,” file contents terminator.

• Set the checks
  • Check if / is mounted
    • /./ || check_fs_mount_rw /
    • check_fs_mount_rw /apps
  • Check if SSH is running
    • * || check_ps_daemon sshd root
  • Check if there is the correct amount of physical memory
    • * || check_hw_physmem 1024 1073741824
  • Check if there is any free
    • * ||
      check_hw_physmem_free 1

• Are there any other checks that could be important for job starts?
• Check to make sure both nodes are up and test a 2 node job
  • qsub -l -l nodes=2

• Unmount /apps on compute1.cluster
  • umount /apps

• Wait for the node too offline itself (should take 45 seconds or less)
  • pbsnodes --a
COMPILE AND RUN APPLICATIONS
COMPILING AND RUNNING MPI PI CALCULATOR

- Change user on head node to login user
  - su login_user

- Start an interactive job
  - qsub -l -l nodes=2

- Generate ssh keys and authorize them
  - ssh-keygen
  - cp ~/ssh/id_rsa.pub ~/ssh/authorized_keys

- Get the MPI pi calculator
  - wget

- List available modules
  - module avail

- Load the mpi module
  - module load openmpi

- Compile the program
  - mpicc pi.c -o pi

- Test pi single threaded
  - ./pi

- Run mpiexec to execute pi across two nodes
  - mpiexec -prefix /apps/openmpi-1.7.5/-machinefile $PBS_NODEFILE/home/login_user/pi
COMPILING HPL

• Remaining in the interactive job
• Download HPL
  • wget hpl-2.1.tar.gz
  • tar xfvz hpl-2.1.tar.gz
  • mv hpl-2.1 hpl
• Load openmpi module
  • module load openmpi
• Grab a working makefile
  • cd hpl
  • cp setup/
  • Make.Linux_PII_CBLAS_gm ./
• Edit the makefile and set the correct LAdir and LAlib paths
  • LAdir = /apps/openblas-0.2.10/lib/
  • LAlib = $(LAdir)/libopenblas.a
• Compile HPL
  • make
  • arch=Linux_PII_CBLAS_gm
RUNNING HPL

• Modify HPL.dat
  • cd bin/
  Linux_PII_CBLAS_gm
• Edit HPL.dat
• Modify the Ps and Qs
  • 1 1 1 Ps
  • 1 1 1 Qs

• HPL tuning
  • http://www.netlib.org/benchmark/hpl/tuning.html

• Execute hpl
  • mpiexec -prefix /apps/
    openmpi-1.7.5/ -np 2 -machinefile
    $PBS_NODEFILE /home/sharrell/hpl/bin/
    Linux_PII_CBLAS_gm/xhpl

• Marvel at the speed of our cluster in comparison to the top 500.
  • http://www.top500.org/lists/2014/06/
LOG AGGREGATION

- Add rsyslog server to the head node
  - Create directory to hold all of the logs
    - file {'/var/log/multi/':
      ensure => 'directory',
      before => Class['rsyslog::server'],
    }

- Add rsyslog module
  - class { 'rsyslog::server':
      server_dir => '/var/log/multi/',
    }

- Add rsyslog client to send logs to server
  - class { 'rsyslog::client':
      remote_type => 'tcp',
      server => 'head.cluster',
    }
NAGIOS HEAD NODE CONFIGURATION

- Edit in class head_node { ... }
- Install Nagios base packages and plugins
  package { ['nagios-common', 'nagios', 'nagios-plugins']:
    ensure => 'present',
    require => Package['epel-release.noarch'],
  }
  package { ['nagios-plugins-ssh', 'nagios-plugins-ping']:
    ensure => 'present',
    require => Package['nagios-plugins'],
  }
- Remove the default localhost config that was created
  exec { 'remove_localhost_conf':
    command => 'mv /etc/nagios/objects/localhost.cfg /etc/nagios/objects/localhost.cfg.orig; touch /etc/nagios/objects/localhost.cfg',
    require => Package['nagios'],
  }
- Now define our own set of hosts that we shall monitor
  file { '/etc/nagios/conf.d/hosts.cfg':
    ensure => 'file',
    content => "
      define host{
        use     linux-server
        host_name head.cluster
        alias   head
        address ${headnodeip}
      }
      ..... # Add other nodes here
      ....."
  }
- Define a hostgroup for setting up checks easily (in hosts.cfg)
  define hostgroup{
    hostgroup_name linux-servers
    alias Linux Servers
    members head.cluster, storage.cluster, compute1.cluster, .....,
  }
  require => Package['nagios'],
  notify => Service['nagios'],
} #end file_hosts.cfg
Now create checks that we want: ping and ssh

```yaml
file { '/etc/nagios/conf.d/services.cfg':
  ensure => file,
  content => "
  define service{
    use       local-service
    hostgroup_name linux-servers
    service_description PING
    check_command check_ping!
    100.0,20%!500.0,60%
  }
  define service{
    use       local-service
    hostgroup_name linux-servers
    service_description SSH
    check_command check_ssh
    notifications_enabled 0
  }",
  require => [Package['nagios'], File['/etc/nagios/conf.d/services.cfg']],
}
```

Create the nagios service

```yaml
service { "nagios":
  ensure => "running",
  enable => "true",
  require => [Package['nagios'], File['/etc/nagios/conf.d/services.cfg']],
}
```

Check status by browsing the Nagios UI.
- https://headnode-public-IP/nagios
- Default username/password are nagiosadmin/nagiosadmin
- Click “Hosts” in left panel and view current node status
WE FINISHED!

QUESTIONS? COMMENTS?

Contact:
Stephen Lien Harrell
SLH@PURDUE.EDU