Cluster Platform Knowledgebase Documentation

Alces Software

May 01, 2019

Introduction

This site documents the considerations and guidelines for designing and developing a HPC platform for cluster computing. The documentation describes general practices and considerations when designing a HPC platform as well as recommendations and guides used by Alces Software to configure HPC platforms.

CHAPTER 1

Acknowledgements

We recognise the respect the trademarks of all third-party providers referenced in this documentation. Please see the respective EULAs for software packages used in configuring your own environment based on this knowledgebase.

1.1 License

CHAPTER 2

HPC Cluster Platform

This knowledgebase document is broken down into the following sections:

2.1 Introduction

The purpose of this documentation is to provide a list of considerations and guidelines for the development of a High Performance Computing (HPC) environment. This documentation should be followed through in order to properly understand the structure of the environment and that certain considerations are not missed out along the way.

To generalise the entire process, it goes as follows:

```
Hardware Architecture Design -> Hardware Build -> Software Build -> Platform Delivery
```

Ensuring that a suitable hardware and network architecture is designed before the build process begins will allow you to create a stable base for your HPC platform.

Performing the hardware build before doing any software configuration guarantees that the network and hardware is properly setup. A partially built network during software setup can lead to unforeseen issues with communication and configuration.

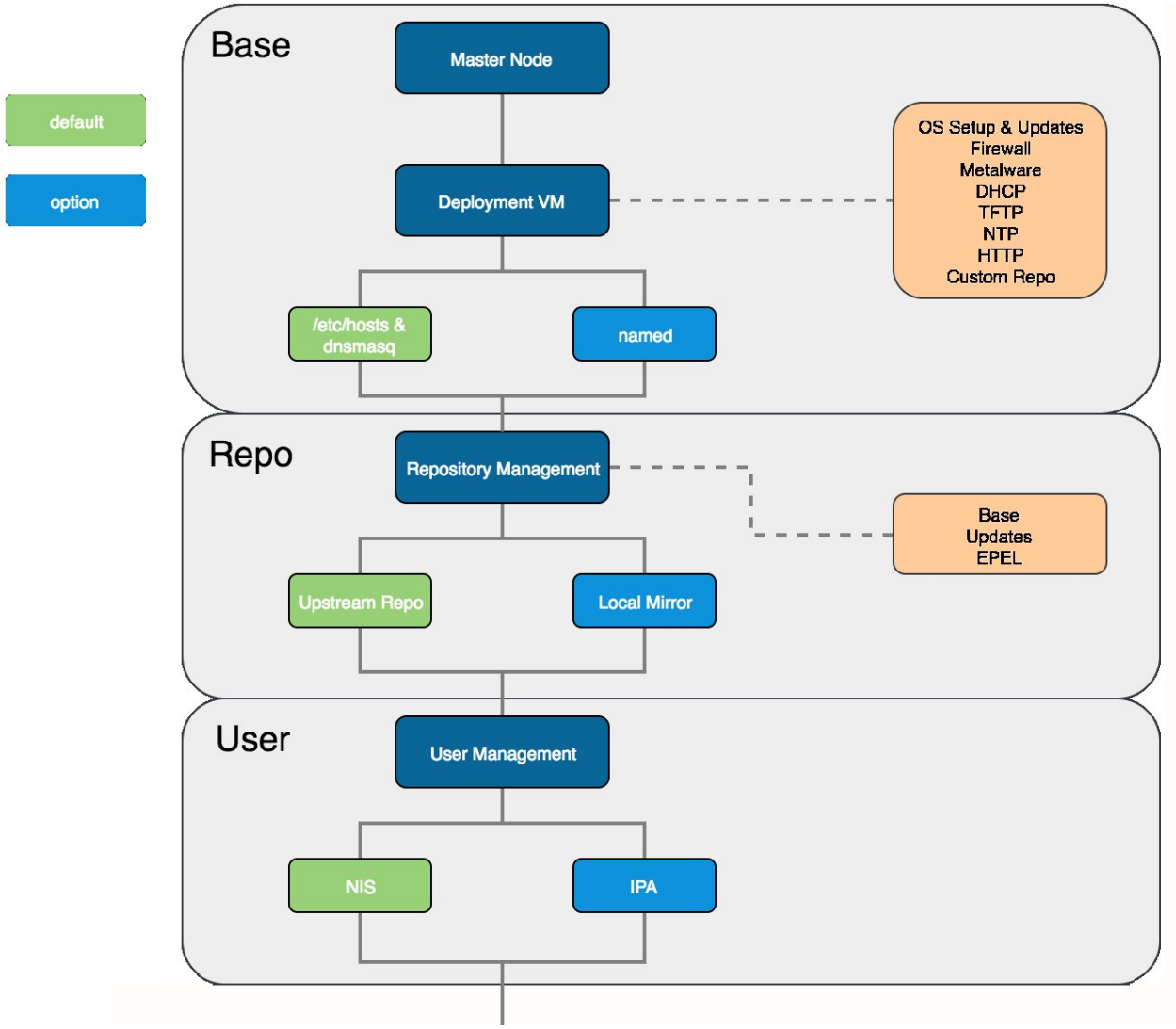
Once the infrastructure has been physically built the software build can proceed. Usually the central servers will be configured first before client and compute nodes are configured.

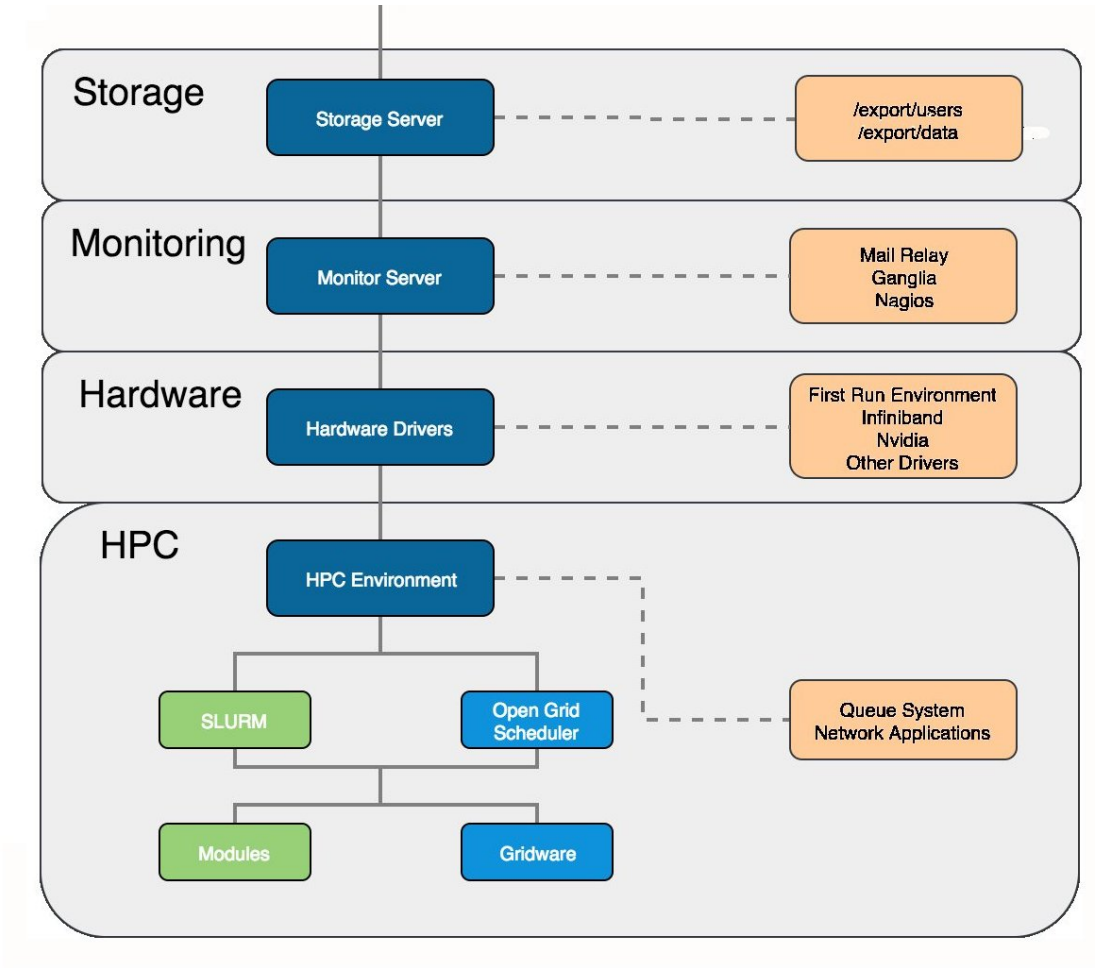
Finally, platform delivery includes a range of connectivity, performance and quality tests which ensure that the completed environment is stable, manageable and consistent.

Note: It is recommended to read through all of the documentation before starting to design the HPC platform to understand the scope and considerations.

2.2 Overviews

A successful HPC cluster environment is composed of a number of different packages. When designing your system, a range of different components can be chosen to enable different functionality, features and capabilities to be implemented. Your choice of packages will depend on your specific requirements - the diagram below shows an example breakdown of the different sections reviewed in this document which can be used to build a complete HPC system:





2.3 Considerations for Network and Hardware Design

In general, the things to consider when designing the hardware and network solution for a HPC platform are:

- The hardware environment
- The types of nodes required in the network
- The different networks to be used by the network
- The level of resilience desired
- The hostname and domain naming convention

These topics are covered in more detail below.

2.3.1 Hardware Environment

The hardware environment will generally be one of two setups, metal or cloud.

- **Metal** - Metal environments are those which are composed of on-site systems in a datacenter which are usually running 24/7.

- **Cloud** - Cloud environments are systems hosted in a third-party datacenter and are usually ephemeral systems that are being created and destroyed on demand.
- **Metal/Cloud Hybrid** - A hybrid environment usually consists of a core metal configuration that uses cloud as an overflow for additional capacity at times of high utilisation.

A hardware environment is mainly focussed on the location, capacity and permanence of the HPC platform and does not directly determine the hardware that will be used in the various systems.

2.3.2 Node Types

A complete HPC platform will be comprised of systems that serve different purposes within the network. Ideas of node types along with the services and purpose of those nodes can be seen below.

- **Login Node** - A login node will usually provide access to the HPC platform and will be the central system that users access to run applications. How users will access the system should be considered, usually this will be SSH and some graphical login service, such as, VNC.
- **Master Node** - A master node will usually run services for the HPC platform. Such as, the master process for a job scheduler, monitoring software and user management services.
- **Compute Node** - Compute nodes are usually used for running HPC applications that are queued through a job scheduler. Additionally, these can be used for VM deployments (via software like OpenStack) or other computational uses. Compute nodes usually have large amounts of cores and memory as well as high bandwidth interconnect (like Infiniband).
- **Special-purpose Node** - Some compute nodes may feature a particular specification to be used for a particular job, or stage in your workflow. Examples may include nodes with more memory, larger amounts of local scratch storage, or GPU/FPGA devices installed.
- **Storage Node** - The storage node will serve network storage solutions to systems on the network. It would have some sort of storage array connected to it which would provide large and resilient storage.

The above types are not strict. Services can be mixed, matched and moved around to create the desired balance and distribution of services and functions for the platform.

2.3.3 Different Networks

The network in the system will most likely be broken up (physically or virtually with VLANs) into separate networks to serve different usages and isolate traffic. Potential networks that may be in the HPC platform are:

- **Primary Network** - The main network that all systems are connected to.
- **Out-of-Band Network** - A separate network for management traffic. This could contain on-board BMCs, switch management ports and disk array management ports. Typically this network would only be accessible by system administrators from within the HPC network.
- **High Performance Network** - Usually built on an Infiniband fabric, the high performance network would be used by the compute nodes for running large parallel jobs over MPI. This network can also be used for storage servers to provide performance improvements to data access.
- **External Networks** - The network outside of the HPC environment that nodes may need to access. For example, the *Master Node* could be connected to an *Active Directory* server on the external network and behave as a slave to relay user information to the rest of the HPC environment.
- **Build Network** - This network can host a DHCP server for deploying operating systems via PXE boot kickstart installations. It allows for systems that require a new build or rebuild to be flipped over and provisioned without disturbing the rest of the network.

- **DMZ** - A demilitarised zone would contain any externally-facing services, this could be setup in conjunction with the external networks access depending on the services and traffic passing through.

The above networks could be physically or virtually separated from one another. In a physical separation scenario there will be a separate network switch for each one, preventing any sort of cross-communication. In a virtually separated network there will be multiple bridged switches that separate traffic by dedicating ports (or tagging traffic) to different VLANs. The benefit of the VLAN solution is that the bridged switches (along with bonded network interfaces) provides additional network redundancy.

Note: If a cloud environment is being used then it is most likely that all systems will reside on the primary network and no others. This is due to the network configuration from the cloud providers.

2.3.4 Resilience

How well a system can cope with failures is crucial when delivering a HPC platform. Adequate resilience can allow for maximum system availability with a minimal chance of failures disrupting the user. System resilience can be improved with many hardware and software solutions, such as:

- **RAID Arrays** - A RAID array is a collection of disks configured in such a way that they become a single storage device. There are different RAID levels which improve data redundancy or storage performance (and maybe even both). Depending on the RAID level used, a disk in the array can fail without disrupting the access to data and can be hot swapped to rebuild the array back to full functionality.[1]
- **Service Redundancy** - Many software services have the option to configure a slave/failover server that can take over the service management should the master process be unreachable. Having a secondary server that mirrors critical network services would provide suitable resilience to master node failure.
- **Failover Hardware** - For many types of hardware there is the possibility of setting up failover devices. For example, in the event of a power failure (either on the circuit or in a power supply itself) a redundant power supply will continue to provide power to the server without any downtime occurring.

There are many more options than the examples above for improving the resilience of the HPC platform, it is worth exploring and considering available solutions during design.

Note: Cloud providers are most likely to implement all of the above resilience procedures and more to ensure that their service is available 99.99% of the time.

2.3.5 Hostname and Domain Names

Using proper domain naming conventions during design of the HPC platform is best practice for ensuring a clear, logical and manageable network. Take the below fully qualified domain name:

```
node01.pri.cluster1.compute.estate
```

Which can be broken down as follows:

- `node01` - The hostname of the system
- `pri` - The network that the interface of the system is sat on (in this case, pri = primary)
- `cluster1` - The cluster that `node01` is a part of

[1] For more information on RAID arrays see https://en.wikipedia.org/wiki/RAID

- `compute` - The subdomain of the greater network that `cluster1` is a part of
- `estate` - The top level domain

2.3.6 Security

Network security is key for both the internal and external connections of the HPC environment. Without proper security control the system configuration and data is at risk to attack or destruction from user error. Some tips for improving network security are below:

- Restrict external access points where possible. This will reduce the quantity of points of entry, minimising the attack surface from external sources.
- Limit areas that users have access to. In general, there are certain systems that users would never (and should never) have access to so preventing them from reaching these places will circumvent any potential user error risks.
- Implement firewalls to limit the types of traffic allowed in/out of systems.

It is also worth considering the performance and usability impacts of security measures.

Much like with resilience, a Cloud provider will most likely implement the above security features - it is worth knowing what security features and limitations are in place when selecting a cloud environment.

Note: Non-Ethernet networks usually cannot usually be secured to the same level as Ethernet so be aware of what the security drawbacks are for the chosen network technology.

2.3.7 Additional Considerations and Questions

The below questions should be considered when designing the network and hardware solution for the HPC platform.

- How much power will the systems draw?
  - Think about the power draw of the selected hardware, it may be drawing a large amount of amps so sufficient power sources must be available.
- How many users are going to be accessing the system?
  - A complex, distributed service network would most likely be overkill and a centralised login/master node would be more appropriate.
- What network interconnect will be used?
  - It's most likely that different network technologies will be used for *Different Networks*. For example, the high performance network could benefit from using Infiniband as the interconnect.
- How could the hardware be optimised?
  - BIOS settings could be tweaked on the motherboard to give additional performance and stability improvements.
  - Network switch configurations could be optimised for different types of traffic
- What *types of nodes* will be in the system?
- What applications are going to be run on the system?
  - Are they memory intensive?
  - Is interconnect heavily relied upon for computations?

2.4 Recommendations for Network and Hardware Design

At Alces software, the recommended network design differs slightly depending on the number of users and quantity of systems within the HPC platform.

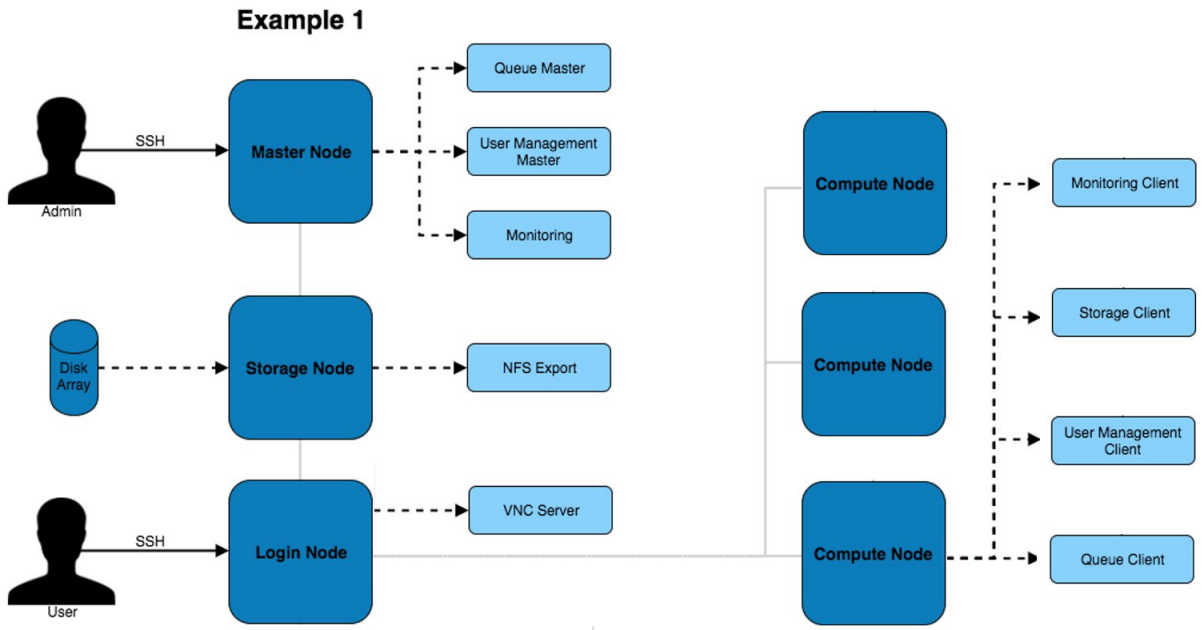
2.4.1 Hardware Recommendations

Recommendations for blades, network switches and storage technologies can be found here - https://github.com/alces-software/knowledgebase/wiki

2.4.2 Cluster Architectures

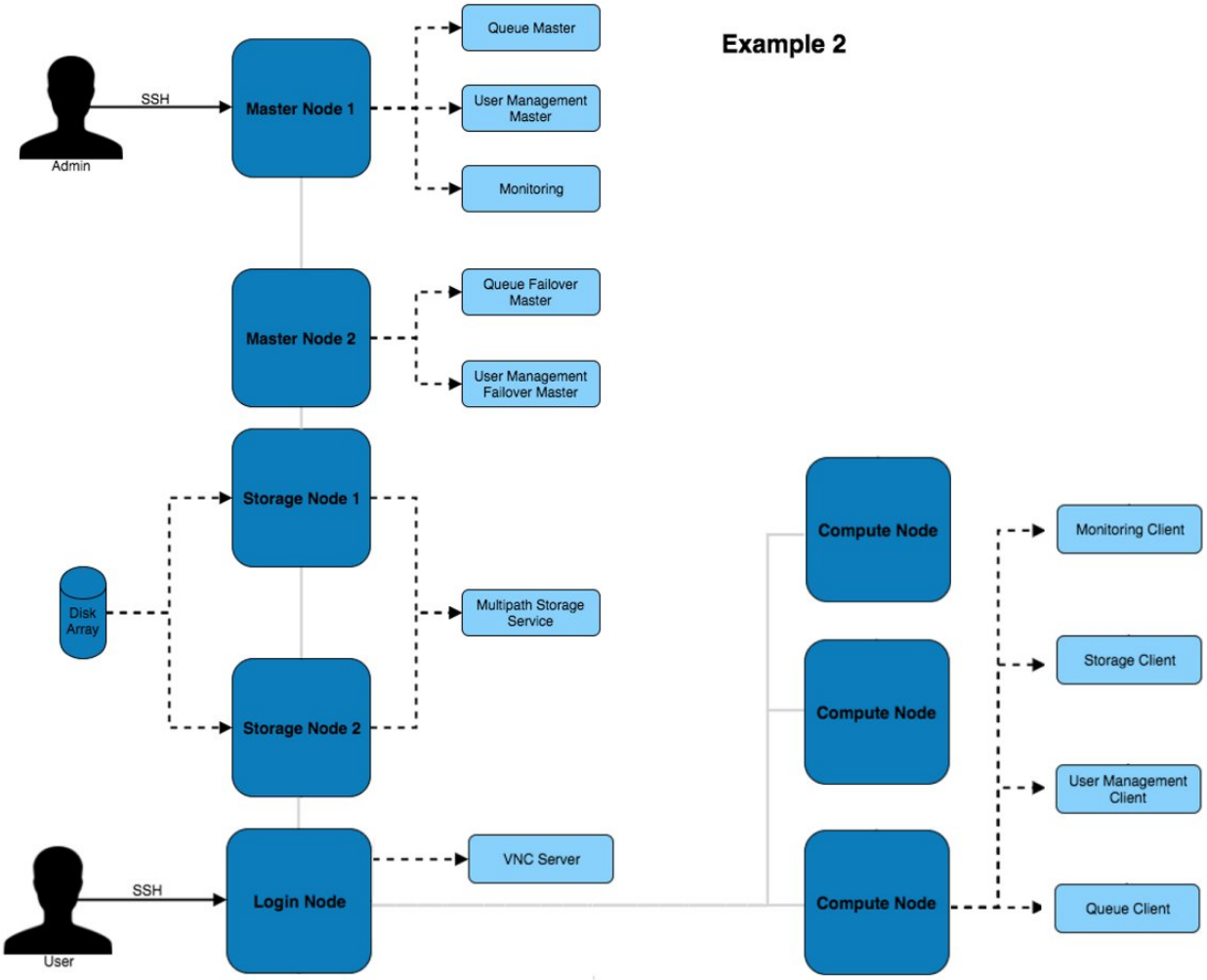
With the *Network and Hardware Design Considerations* in mind, diagrams of different architectures are below. They increase in complexity and redundancy as the list goes on.

Example 1 - stand-alone



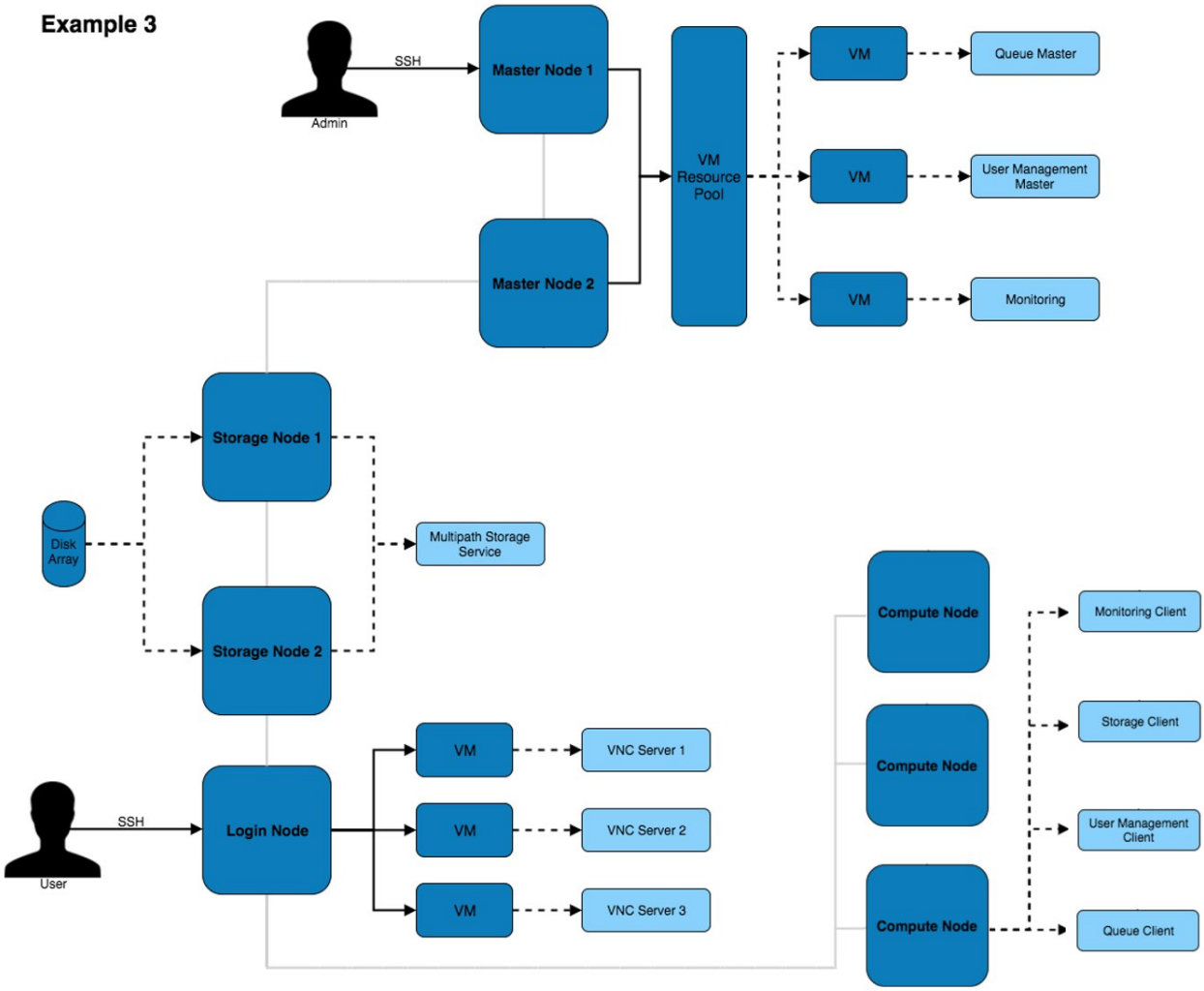
The above architecture consists of master, login and compute nodes. The services provided by the master & login nodes can be seen to the right of each node type. This architecture only separates the services for users and admins.

Example 2 - high-availability



This architecture provides additional redundancy to the services running on the master node. For example, the disk array is connected to both master nodes which use multipath to ensure the higher availability of the storage device.

Example 3 - HA VMs

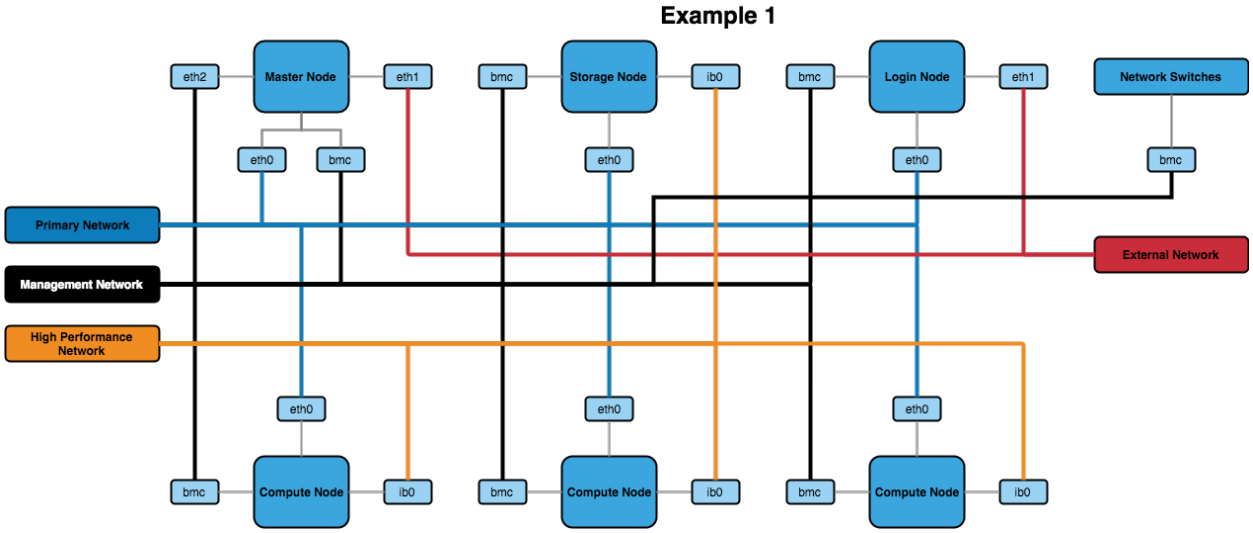


This architecture puts services inside of VMs to improve the ability to migrate and modify services with little impact to the other services and systems on the architecture. Virtual machines can be moved between VM hosts live without service disruption allowing for hardware replacements to take place on servers.

2.4.3 Network Designs

The above architectures can be implemented with any of the below network designs.

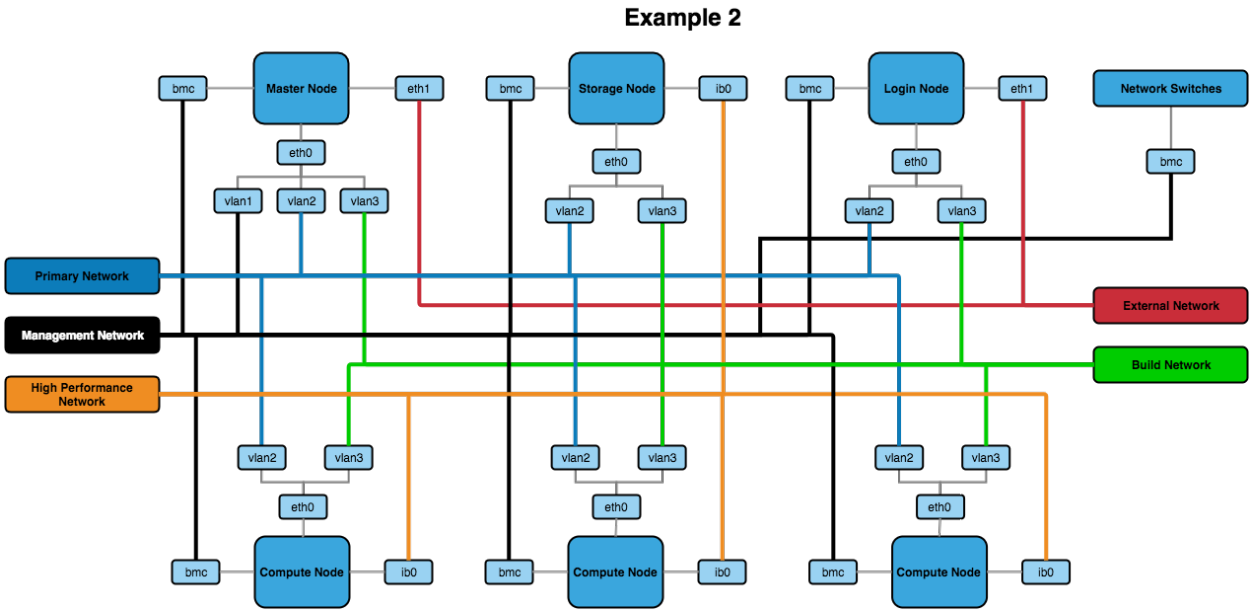
Example 1- simple



The above design contains the minimum recommended internal networks. A primary network (for general logins and navigating the system), a management network (for BMC management of nodes and switches) and a high performance Infiniband network (connected to the nodes). The master and login nodes have access to the external network for user and admin access to the HPC network.

Note: The master node could additionally be connected to the high performance network so that compute nodes have a faster network connection to storage.

Example 2 - VLANs



The above network design has a few additions to the first example. The main change is the inclusion of VLANs for the primary, management and build networks (with the build network being a new addition to this design). The build

network allows for systems to be toggled over to a DHCP system that uses PXE booting to kickstart an OS installation.

2.4.4 Other Recommendations

BIOS Settings

It's recommended to ensure that the BIOS settings are reset to default and the latest BIOS version is installed before optimising the settings. This can ensure that any issues that may be present in the configuration before proceeding have been removed.

When it comes to optimising the BIOS settings on a system in the network, the following changes are recommended:

- Setting the power management to maximum performance
- Disabling CPU CStates
- Disabling Hyperthreading
- Enabling turbo mode
- Disabling quiet boot
- Setting BMC to use the dedicated port for BMC traffic
- Setting the node to stay off when power is restored after AC power loss

Note: The wordings for settings above may differ depending on the hardware that is being used. Look for similar settings that can be configured to achieve the same result.

For hardware-specific BIOS configuration settings see https://github.com/alces-software/knowledgebase/wiki#servers

2.5 Considerations for Infrastructure Design

Infrastructure design largely relates to the considerations made for the *Cluster Architectures*. Depending on the design being used, some of the infrastructure decisions may have already been made.

2.5.1 Infrastructure Service Availability

There are typically 3 possible service availability options to choose from, these are:

- All-in-one
- VM Platform
- High Availability VM Platform

Note: If using a Cloud Platform then the service availability will be handled by the cloud provider. The only additional considerations are how services will be provided in terms of native running services or containerised.

These are covered in more detail below.

All-in-one

This is the most common solution, an all-in-one approach loads services onto a single machine which serves the network. It is the simplest solution as a single OS install is required and no additional configuration of virtual machine services is needed.

This solution, while quick and relatively easy to implement, is not a recommended approach. Due to the lack of redundancy options and the lack of service isolation there is a higher risk of an issue effecting one service (or the machine) to have an effect on other services.

VM Platform

A VM platform provides an additional layer of isolation between services. This can allow for services to be configured, migrated and modified without potentially effecting other services.

There are a number of solutions for hosting virtual machines, including:

- Open-source solutions (e.g. VirtualBox, KVM, Xen)
- Commercial solutions (e.g. VMware)

The above software solutions provide similar functionality and can all be used as a valid virtualisation platform. Further investigation into the ease of use, flexibility and features of the software is recommended to identify the ideal solution for your HPC platform.

High Availability VM Platform

For further redundancy, the virtualisation platform can utilise a resource pool. The service will be spread across multiple machines which allows for VMs to migrate between the hosts whilst still active. This live migration can allow for one of the hosts to be taken off of the network for maintenance without impacting the availability of the service VMs.

2.5.2 Node Network Configuration

In addition to the availability of services, the network configuration on the node can provide better performance and redundancy. Some of the network configuration options that can improve the infrastructure are:

- **Channel Bonding** - Bonding interfaces allows for traffic to be shared between 2 network interfaces. If the bonded interfaces are connected to separate network switches then this solution
- **Interface Bridging** - Network bridges are used by interfaces on virtual machines to connect to the rest of the network. A bridge can sit on top of a channel bond such that the VM service network connection is constantly available.
- **VLAN Interface Tagging** - VLAN management can be performed both on a managed switch and on the node. The node is able to create subdivisions of network interfaces to add VLAN tags to packets. This will create separate interfaces that can be seen by the operating system (e.g. eth0.1 and eth0.2) which can individually have IP addresses set.

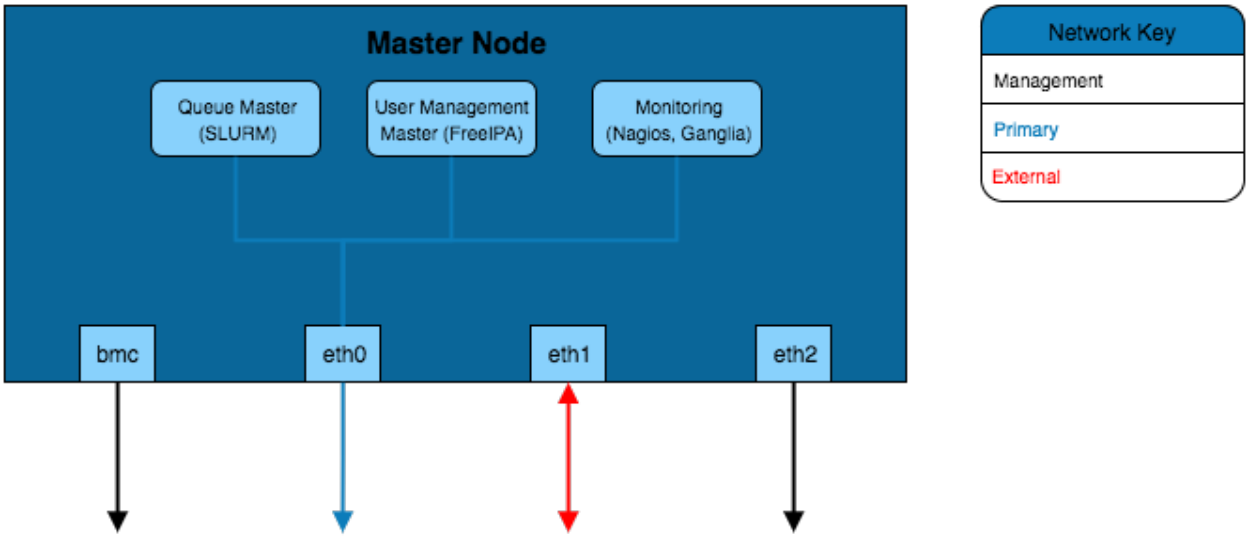
2.5.3 Additional Considerations and Questions

- Could containerised solutions be used instead of VMs?
  - *Docker* or *Singularity* containerised services can provide similar levels of isolation between services as VMs without the additional performance overhead.

2.6 Recommendations for Infrastructure Design

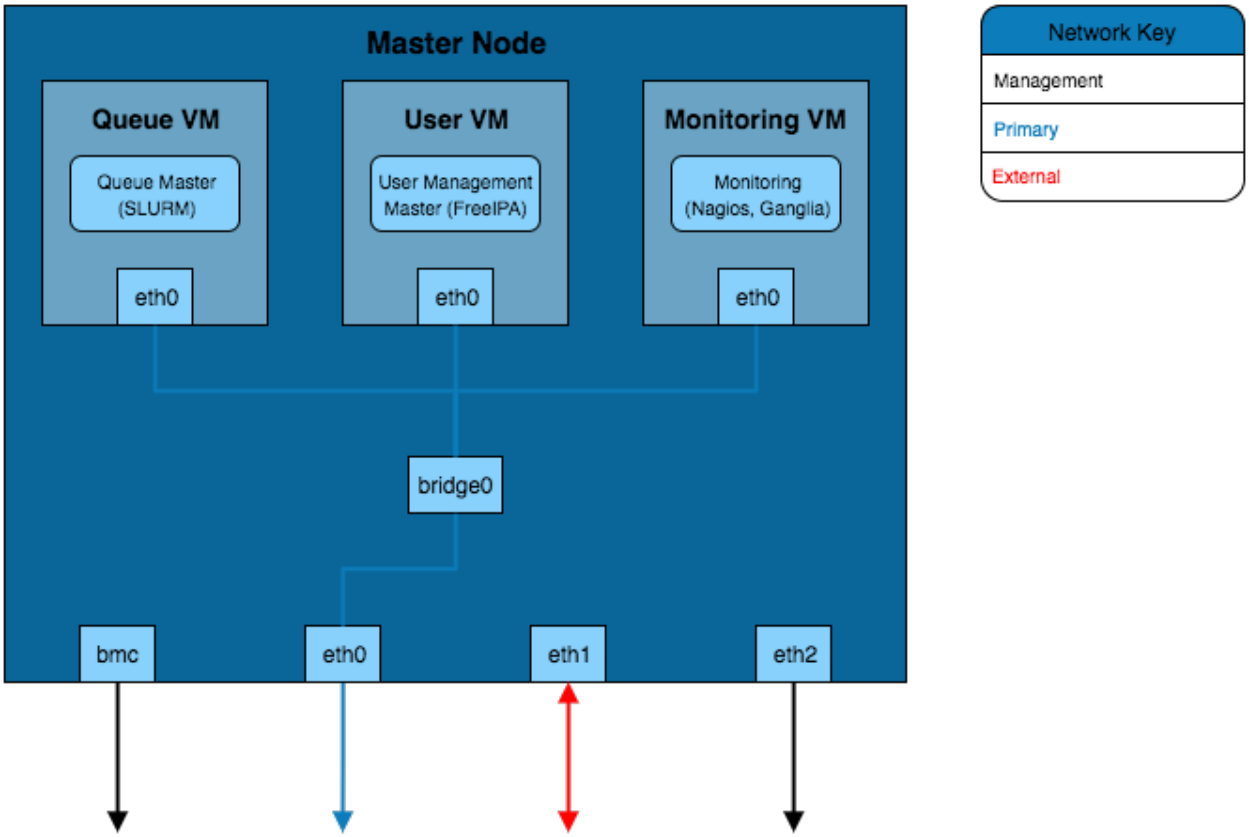
The example configurations here combine elements of the *Recommendations for Network and Hardware Design* as well as the different infrastructure solutions from *Considerations for Infrastructure Design*. These focus on the internal configuration of the master node but these examples can be extrapolated for configuring login, storage, compute or any other nodes that are part of the HPC environment.

2.6.1 Simple Infrastructure



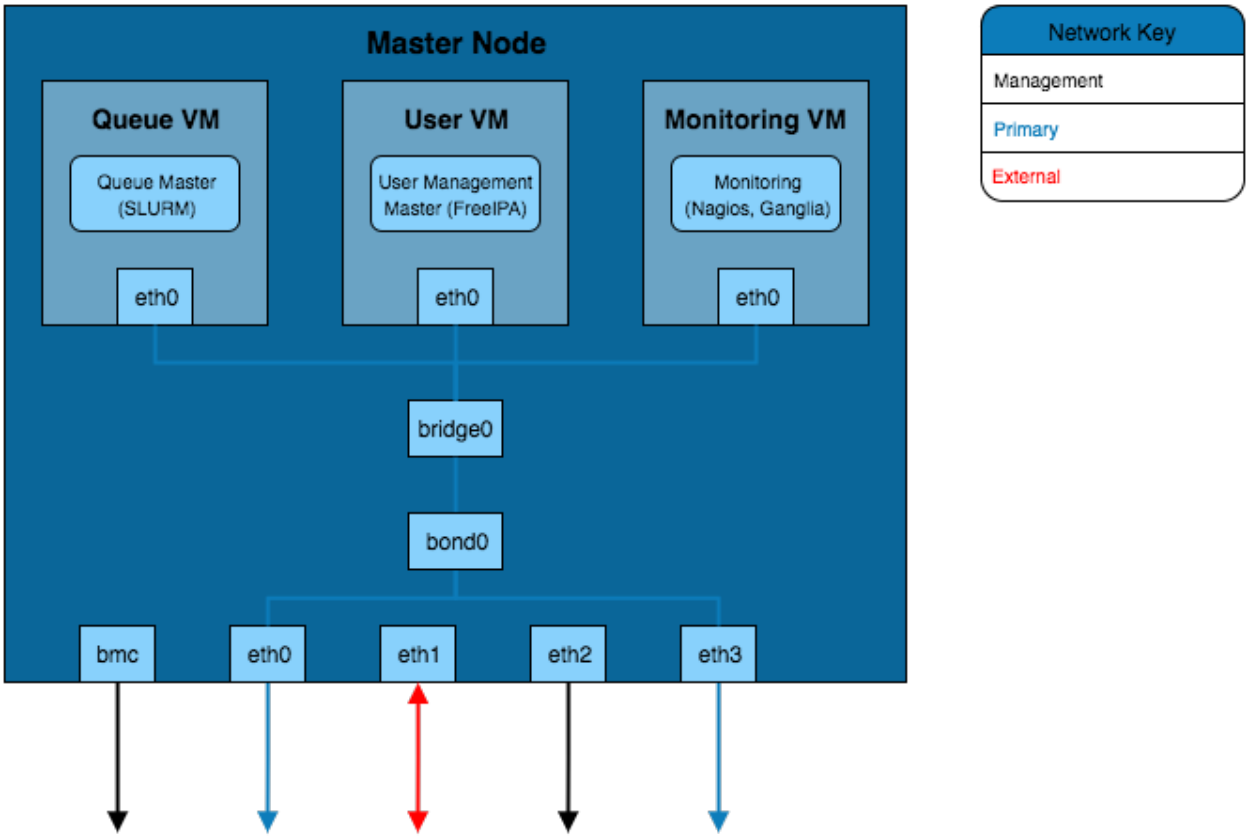
The simplest infrastructure configuration uses the all-in-one approach where services are configured on the master node's operating system.

2.6.2 Virtual Machine Infrastructure



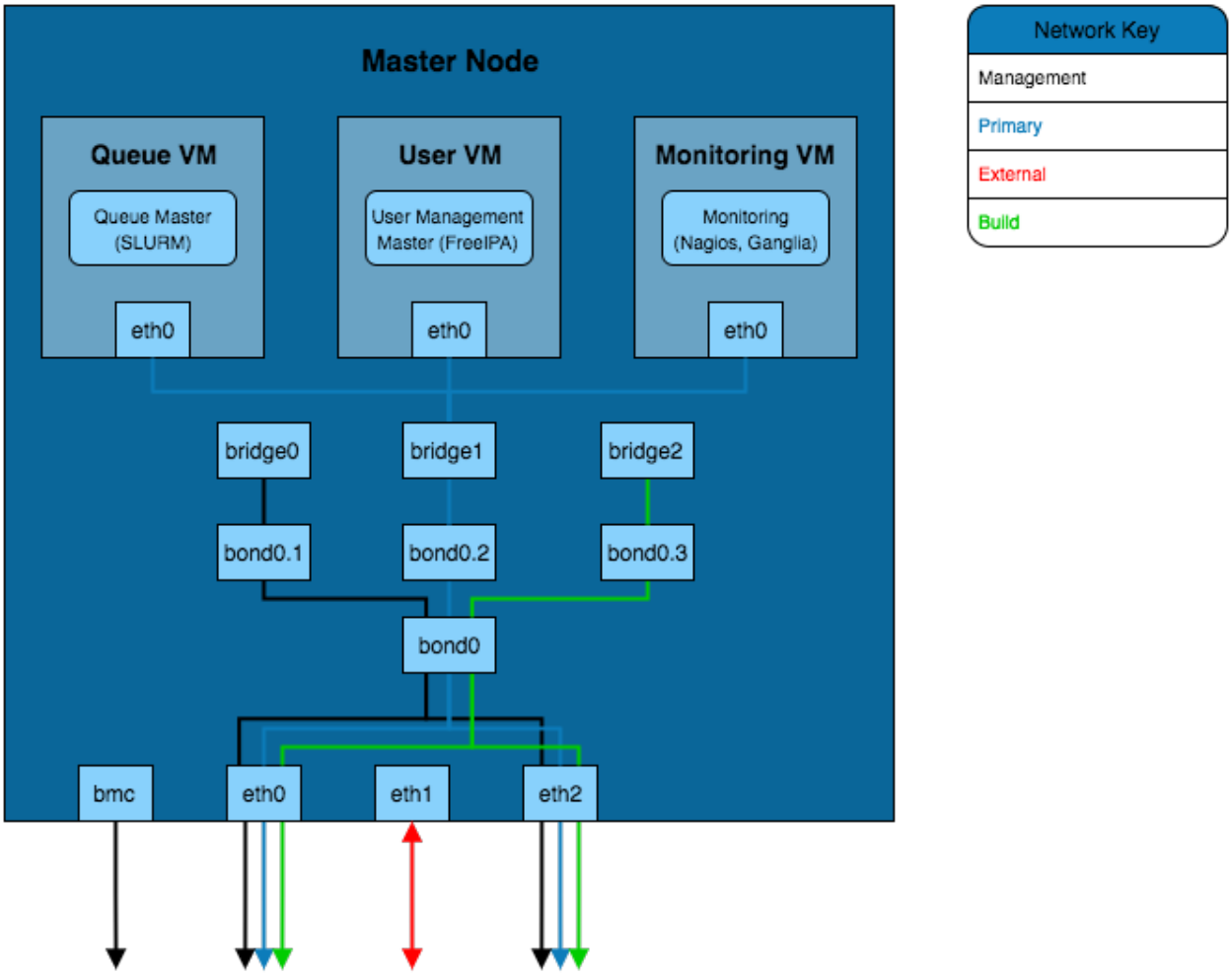
This solution separates the services into VMs running on the master node. In order for these VMs to be able to connect to the primary network a network bridge is created that allows the VM interfaces to send traffic over the eth0 interface.

2.6.3 Channel Bonded Infrastructure



This example adds a layer of redundancy over the *VM Infrastructure* design by bonding the eth0 and eth3 interfaces. These interfaces are connected to separate network switches (the switches will be bridged together as well) which provides redundancy should a switch or network interface fail. Bonding of the two interfaces creates a new *bond* interface that the bridge for the virtual machines connects to.

2.6.4 VLAN Infrastructure



The above solution implements the channel bonded infrastructure in a network with VLANs. The VLANs have bond and bridge interfaces created for them. This allows some additional flexibility for VM bridging as virtual interfaces can be bridged onto specific VLANs whilst maintaining the redundancy provided by the bond. This adds additional security to the network as the master node can be left without an IP on certain VLAN bond interfaces which prevents that network from accessing the master node whilst VMs on the master node are able to reach that VLAN.

2.7 Base System Overview

2.7.1 About

The base system is comprised of the integral services required for a deployment environment.

It is recommended that periodic updates are run in the future with the source tree for the minor OS version. The systems would require careful testing after any updates have been applied to ensure system functionality has persisted.

Note: If a *local repository* has been setup then the local repo mirrors will need to be resynced before deploying updates.

The controller node also provides IP Masquerading on its external interface. All slave nodes are configured to default route out via the controller node's external interface.

A tftp service, dhcpd service and webserver are installed on the controller node, these enable slave systems to be booted and pickup a series of automated deployment scripts that will result in a blank system being deployed and joining the environment.

2.7.2 Components

This package will set up and configure:

- Up-to-date OS installation (CentOS 7.3 with full upstream 'update' tree applied)
- Firewall rules for network interfaces
- Metalware cluster management software, providing:
  - Custom yum repository for providing additional packages to nodes
    - The directory `/opt/alces/repo/custom/Packages` can be used to store RPMs that are then be served to client nodes, allowing for custom, additional or non-supported packages to be installed.
  - DHCP and TFTP server configuration for network booting
    - DHCP provides host identity management, such as, serving IPs and hostnames to client systems based on the hardware MAC address of the client. This information is used during installation to configure the node uniquely.
    - TFTP provides the boot configuration of the system in order to provide the build or boot environment of client systems.
  - NTP for keeping the cluster clocks in sync
- Name resolution services either:
  - DNSmasq using `/etc/hosts`
    - Alongside the server providing lookup responses, the client systems will also have a fully populated `/etc/hosts` files for local queries.
  - *or*
  - Named from bind packages
    - Named creates forward and reverse search zones on the controller node that can be queried by all clients. Unlike DNSmasq, the client systems have an empty `/etc/hosts` as named is serving all of the additional host information.
- Management tools built around ipmitool, pdsh and libgenders
  - These management tools allow for running commands on multiple systems defined in groups, improving the ease and flexibility of environment management.
- Combined logging for clients with rsyslog server
  - All clients built from the master write out build progress and syslog messages to `/var/log/slave/<NODENAME>/messages.log`

2.7.3 Key Files

- `/etc/hosts`
- `/etc/dhcp/dhcpd.*`

- `/etc/dnsmasq.conf` or `/etc/named/metalware.conf`
- `/opt/metalware/*`
- `/var/lib/metalware/*`
- `/var/lib/tftpboot/*`
- `/etc/ntp.conf`
- `/etc/rsyslog.d/metalware.conf`

2.7.4 Licensing

The CentOS Linux distribution is released under a number of Open-source software licenses, including GPL. A copy of the relevant license terms is included with the CentOS software installed on your cluster. The CentOS Linux distribution does not require any per-server license registration.

Additionally, the applications and services installed have similar open-source licensing which can be viewed either online or through the manual pages for the specific package.

2.8 Considerations for Software and Application Deployment

Before considering *how* the OS and applications will be deployed it is worth making a few decisions regarding the OS that will be used:

- Will the same OS be used on all systems? (it's strong recommended to do so)
- What software will be used? (and therefore will need to be supported by the OS)
- How stable is the OS? (bleeding edge OSes may have bugs and instabilities that could negatively impact the HPC environment)
- If you are using bare-metal hardware, is the OS supported by your hardware vendor? (running an unsupported OS can lead to issues when attempting to obtain hardware support)

2.8.1 Deployment

The deployment of the HPC platform can be summarised in two main sections, these being:

- Operating System Deployment
- Software Package Repository Management

Operating System Deployment

When it comes to performing many operating installations across nodes in the network it can be tricky to find a flexible, manageable, automated solution. Performing manual installations of operating systems may be the ideal solution if there are only a few compute nodes, however, there are many other ways of improving the speed of OS deployment:

- **Disk Cloning** - A somewhat inelegant solution, disk cloning involves building the operating system once and creating a compressed copy on a hard-drive that can be restored to blank hard-drives.
- **Kickstart** - A kickstart file is a template for automating OS installations, the configuration file can be served over the network such that clients can PXE boot the installation. This can allow for easy, distributed deployment over a local network.

- **Image Deployment** - Cloud service providers usually deploy systems from template images that set hostnames and other unique system information at boot time. Customised templates can also be created for streamlining the deployment and customisation procedure.

It is worth considering manual, cloning and kickstart solutions for your OS deployment, any one of them could be the ideal solution depending on the number of machines that are being deployed.

Repository Management

It is worth considering how packages, libraries and applications will be installed onto individual nodes and the network as a whole. Operating systems usually have their own package management system installed that uses public repositories for pulling down packages for installation. It is likely that all of the packages required for a system are not in the public repositories so it's worth considering where additional packages will come from (e.g. a 3rd party repository, downloaded directly from the package maintainer or manually compiled).

Further to managing packages on the local system, the entire network may require applications to be installed; there are a couple of options for achieving this:

- **Server Management Tools** - Management tools such as puppet, chef or pdsh can execute commands across multiple systems in parallel. This saves time instead of having to individually login and run commands on each node in the system.
- **Network Package Managers** - Software such as Alces Gridware can install an application in a centralised storage location, allowing users simply to modify their `PATH` and `LD_LIBRARY_PATH` on a node in order to start using the application.

For more information regarding network package managers and application deployment, see *Application Deployment*

2.8.2 Additional Considerations and Questions

- How will applications outside of the repositories be installed?
- Will the application need to be useable by all nodes in the HPC network? (e.g. an NFS export for apps may solve the issue of multiple installations)
- How will new package versions be installed when the HPC environment being maintained in the future?
- How will you create and maintain a consistent software environment on all nodes over time?

2.9 Master Node Setup

2.9.1 Manual Setup

- Run a minimal CentOS installation on the system; this can be performed manually or via an automated install service if you have one already setup
- It is recommended to update the packages on the system for any bug fixes and patches that may have been introduced to core packages:

```
yum -y update
```

Note: If using kickstart OS installation on many nodes it is worth considering a *local mirror repository* for the OS image and packages so that all nodes receive an equivalent software installation, and aren't trying to connect to the internet at the same time.

- Disable and stop NetworkManager:

```
systemctl disable NetworkManager
systemctl stop NetworkManager
```

- Set the hostname:

```
echo 'master.cluster.local' > /etc/hostname
```

- Configure bridge interface for primary network (`/etc/sysconfig/network-scripts/ifcfg-pri`):

```
DEVICE=pri
ONBOOT=yes
TYPE=Bridge
stp=no
BOOTPROTO=static
IPADDR=10.10.0.11
NETMASK=255.255.0.0
ZONE=trusted
PEERDNS=no
```

Note: Replace `DEVICE`, `IPADDR` and `NETMASK` with the appropriate values for your system

- Create bridge interfaces for all other networks (e.g. management [`mgt`] and external [`ext`])

Note: The external interface may require getting it's network settings over DHCP; if so then set `BOOTPROTO` to `dhcp` instead of static and remove the `IPADDR` lines.

- Setup config file for network interfaces (do this for all interfaces sitting on the bridges configured above):

```
TYPE=Ethernet
BOOTPROTO=none
NAME=p1p1
DEVICE=p1p1
ONBOOT="yes"
BRIDGE=pri
```

Note: In the above example, the interface `p1p1` is connected to the primary network but instead of giving that an IP it is set to use the `pri` bridge

- Enable and start firewalld to allow masquerading client machines via the external interface and to improve general network security:

```
systemctl enable firewalld
systemctl start firewalld
```

- Add `ext` bridge to external zone; the external zone is a zone configured as part of firewalld:

```
firewall-cmd --add-interface ext --zone external --permanent
```

- Add all the other network interfaces to the trusted zone; replace `pri` with the other network names, excluding `ext`:

```
firewall-cmd --add-interface pri --zone trusted --permanent
```

- Reboot the system
- Install components for VM service:

```
yum groupinstall -y virtualization-platform virtualization-tools
yum install -y virt-viewer virt-install
```

- Enable and start the virtualisation service:

```
systemctl start libvirtd
systemctl enable libvirtd
```

- Create VM pool:

```
mkdir /opt/vm
virsh pool-define-as local dir - - - - "/opt/vm/"
virsh pool-build local
virsh pool-start local
virsh pool-autostart local
```

2.9.2 Automated Setup

If using metalware, a controller can be used to deploy it's own master. By setting up the controller on a separate machine to the master, the master can then be defined and hunted (see *Deployment Example* for hunting instructions). The following will add the build config and scripts to configure a functional master (much like the above).

Note: In the following guide the group is called `masters` and the master node is `master1`

- Configure certificate authority for libvirt from the controller as described in VM Deployment from the Controller
- Create a deployment file for the master node (`/var/lib/metalware/repo/config/master1.yaml`) containing the following (the network setup configures network bridges and the external interface):

```
files:
  setup:
    - /opt/alces/install/scripts/10-vm_master.sh
  core:
    - /opt/alces/ca_setup/master1-key.pem
    - /opt/alces/ca_setup/master1-cert.pem
networks:
  pri:
    interface: pri
    type: Bridge
    slave_interfaces: em1
  mgt:
    interface: mgt
    type: Bridge
```

(continues on next page)

(continued from previous page)

```
    slave_interfaces: em2
  ext:
    defined: true
    domain: ext
    ip: 10.101.100.99
    network: 10.101.0.0
    netmask: 255.255.0.0
    gateway: 10.101.0.1
    short_hostname: <%= node.name %>.<%= config.networks.ext.domain %>
    interface: ext
    firewallpolicy: external
    slave_interfaces: p1p4
```

Note: If additional scripts are defined in the domain level `setup` and `core` lists then be sure to include them in the master1 file

- Additionally, download the VM master script to `/opt/alces/install/scripts/10-vm_master.sh`:

```
mkdir -p /opt/alces/install/scripts/
cd /opt/alces/install/scripts/
wget -O 10-vm_master.sh https://raw.githubusercontent.com/alces-software/
↪knowledgebase/master/epel/7/libvirt/vm_master.sh
```

2.10 Controller VM Setup

2.10.1 On Master Node

- Create `/opt/vm/controller.xml` for provisioning a VM called controller (`Available here`)
  - This template creates 3 interfaces on the VM (on the primary, management and external networks)
- Create base qcow2 image `controller.qcow2`:

```
qemu-img create -f qcow2 controller.qcow2 80G
```

- Create the VM:

```
virsh define controller.xml
```

- Start the VM:

```
virsh start controller
```

- Connect a VNC-like window to the VM to watch it booting and interact with the terminal:

```
virt-viewer controller
```

Note: Much like the host system, a minimal installation of CentOS 7 is recommended (as is ensuring that the system is up-to-date with `yum -y update`)

2.10.2 On Controller VM

OS Setup

- Set the hostname of the system (the fully-qualified domain name for this system has additionally added the cluster name):

```
echo 'controller.testcluster.cluster.local' > /etc/hostname
```

- Setup the network interfaces (if setting a static IP then ensure to set `IPADDR`, `NETMASK` and `NETWORK` for the interface)
  - Eth0 is bridged onto the primary network - set a static IP for that network in `/etc/sysconfig/network-scripts/ifcfg-eth0`
  - Eth1 is bridged onto the management network - set a static IP for that network in `/etc/sysconfig/network-scripts/ifcfg-eth1`
  - Eth2 is bridged onto the external network - this will most likely use DHCP to obtain an IP address `/etc/sysconfig/network-scripts/ifcfg-eth2`

Note: Add `ZONE=trusted` to eth0 & eth1, `ZONE=external` to eth2 to ensure the correct firewall zones are used by the interfaces.

- Enable and start firewalld:

```
systemctl enable firewalld
systemctl start firewalld
```

- Add the interfaces to the relevant firewall zones:

```
firewall-cmd --add-interface eth0 --zone trusted --permanent
firewall-cmd --add-interface eth1 --zone trusted --permanent
firewall-cmd --add-interface eth2 --zone external --permanent
```

- Disable network manager:

```
systemctl disable NetworkManager
```

- Reboot the VM
- Once the VM is back up it should be able to ping both the primary and management interfaces on the master node. If the ping returns properly then metalware can be configured to enable deployment capabilities on the VM.

Metalware Install

- Install metalware (to install a different branch, append `alces_SOURCE_BRANCH=develop` before `/bin/bash` in order to install the `develop` branch):

```
curl -sL http://git.io/metalware-installer | sudo alces_OS=el7 /bin/bash
```

- Set metalware to use default repository:

```
metal repo use https://github.com/alces-software/metalware-default.git
```

Domain Configuration

- Configure the domain settings (this will prompt for various details regarding the domain setup, such as, root password, SSH RSA key [which can be created with `ssh-keygen`] and default network parameters):

```
metal configure domain
```

- Uncomment the `PASSWORD=` line in `/opt/metalware/etc/ipmi.conf` and replace `password` with the IPMI password from the domain configuration
- Sync the configuration changes:

```
metal sync
```

Controller Build

- Configure the controller node:

```
metal configure local
```

- Sync the configuration changes:

```
metal sync
```

- Template the scripts and sync them into place:

```
metal template local
metal sync
```

- Build the controller:

```
metal build local
```

Note: If you wish to install an OS other than CentOS 7 then see the *Configure Alternative Kickstart Profile* instructions.

Platform Scripts

Deploying on different hardware and platforms may require additional stages to be run on systems when being deployed. This is handled by an additional scripts key `platform:` in `/var/lib/metalware/repo/config/domain.yaml`.

There is currently a script for configuring the AWS EL7 platform available on github which can be downloaded to the scripts area:

```
mkdir -p /opt/alces/install/scripts/
cd /opt/alces/install/scripts/
wget https://raw.githubusercontent.com/alces-software/knowledgebase/master/epel/7/
↪platform/aws.sh
```

2.11 Deployment Example

2.11.1 Client Deployment Example

- Configure a node group (this example creates a `nodes` group for compute nodes):

```
metal configure group nodes
```

- Start the controller VM listening for PXE requests:

```
metal hunter
```

- Boot up the client node
- The controller VM will print a line when the node has connected, when this happens enter the hostname for the system (this should be a hostname that exists in the group configured earlier)
- Once the hostname has been added the previous metal command can be cancelled (with ctrl-c)
- Generate DHCP entry for the node:

```
metal dhcp
```

- Start the controller VM serving installation files to the node (replace slave01 with the hostname of the client node):

```
metal build slave01
```

Note: If building multiple systems the genders group can be specified instead of the node hostname. For example, all compute nodes can be built with `metal build -g nodes`.

- The client node can be rebooted and it will begin an automatic installation of CentOS 7
- The `metal build` will automatically exit when the client installation has completed
- Passwordless SSH should now work to the client node

2.11.2 Configuring Alternative Kickstart Profile

In this example, a CentOS 6 kickstart profile is configured. This method should be transferrable to other operating systems with little modification to the general practice.

- Download the boot files to the `PXE_BOOT` directory:

```
PXE_BOOT=/var/lib/tftpboot/boot/
curl http://mirror.ox.ac.uk/sites/mirror.centos.org/6/os/x86_64/images/pxeboot/
↪initrd.img > "$PXE_BOOT/centos6-initrd.img"
curl http://mirror.ox.ac.uk/sites/mirror.centos.org/6/os/x86_64/images/pxeboot/
↪vmlinuz > "$PXE_BOOT/centos6-kernel"
```

- Create `/var/lib/metalware/repo/pxelinux/centos6` template PXE boot file for the OS:

```
DEFAULT menu
 PROMPT 0
 MENU TITLE PXE Menu
```

(continues on next page)

(continued from previous page)

```
TIMEOUT 5
TOTALTIMEOUT 5
<%= alces.firstboot ? "ONTIMEOUT INSTALL" : "ONTIMEOUT local"%>

LABEL INSTALL
     KERNEL boot/centos6-kernel
        APPEND initrd=boot/centos6-initrd.img ksdevice=<%= config.networks.pri.
↪interface %> ks=<%= node.kickstart_url %> network ks.sendmac _ALCES_BASE_
↪HOSTNAME=<%= node.name %> <%= config.kernelappendoptions %>
        IPAPPEND 2

LABEL local
     MENU LABEL (local)
     MENU DEFAULT
     LOCALBOOT 0
```

- Create `/var/lib/metalware/repo/kickstart/centos6` template file for kickstart installations of the OS:

```
#!/bin/bash
##(c)2017 Alces Software Ltd. HPC Consulting Build Suite
## vim: set filetype=kickstart :

network --onboot yes --device <%= config.networks.pri.interface %> --bootproto␣
↪dhcp --noipv6

#MISC
text
reboot
skipx
install

#SECURITY
firewall --enabled
firstboot --disable
selinux --disabled

#AUTH
auth  --useshadow  --enablemd5
rootpw --iscrypted <%= config.encrypted_root_password %>

#LOCALIZATION
keyboard uk
lang en_GB
timezone  Europe/London

#REPOS
url --url=<%= config.yumrepo.build_url %>

#DISK
%include /tmp/disk.part

#PRESCRIPT
%pre
set -x -v
exec 1>/tmp/ks-pre.log 2>&1
```

(continues on next page)

(continued from previous page)

```
DISKFILE=/tmp/disk.part
bootloaderappend="console=tty0 console=ttyS1,115200n8"
cat > $DISKFILE << EOF
<%= config.disksetup %>
EOF

#PACKAGES
%packages --ignoremissing

vim
emacs
xauth
xhost
xdpyinfo
xterm
xclock
tigervnc-server
ntpdate
vconfig
bridge-utils
patch
tcl-devel
gettext

#POSTSCRIPTS
%post --nochroot
set -x -v
exec 1>/mnt/sysimage/root/ks-post-nochroot.log 2>&1

ntpdate 0.centos.pool.ntp.org

%post
set -x -v
exec 1>/root/ks-post.log 2>&1

# Example of using rendered Metalware file; this file itself also uses other
# rendered files.
curl <%= node.files.main.first.url %> | /bin/bash | tee /tmp/metalware-default-
↪output

curl '<%= node.build_complete_url %>&event=complete&msg=Build%20is%20complete'
```

- When building nodes, use the new template files by specifying them as arguments to the `metal build` command:

```
metal build -k centos6 -p centos6 slave01
```

2.11.3 Configuring UEFI Boot

UEFI network booting is an alternative to PXE booting and is usually the standard on newer hardware, support for building nodes with UEFI booting can be configured as follows.

- Create additional TFTP directory and download EFI boot loader:

```
mkdir -p /var/lib/tftpboot/efi/
cd /var/lib/tftpboot/efi/
wget https://github.com/alces-software/knowledgebase/raw/master/epel/7/grub-efi/
↪grubx64.efi
chmod +x grubx64.efi
```

- For UEFI clients, add the following line to the client config file:

```
build_method: uefi
```

- Additionally, a `/boot/efi` partition will be required for UEFI clients, an example of this partition as part of the disk setup (in the client config) is below:

```
disksetup: |
  zerombr
  bootloader --location=mbr --driveorder=sda --append="$bootloaderappend"
  clearpart --all --initlabel

  #Disk partitioning information
  part /boot --fstype ext4 --size=4096 --asprimary --ondisk sda
  part /boot/efi --fstype=efi --size=200 --asprimary --ondisk sda
  part pv.01 --size=1 --grow --asprimary --ondisk sda
  volgroup system pv.01
  logvol  /  --fstype ext4 --vgname=system  --size=16384 --name=root
  logvol  /var --fstype ext4 --vgname=system --size=16384 --name=var
  logvol  /tmp --fstype ext4 --vgname=system --size=1 --grow --name=tmp
  logvol  swap  --fstype swap --vgname=system  --size=8096  --name=swap1
```

2.12 Repository Overview

2.12.1 About

Upstream repositories for CentOS and EPEL will be mirrored locally to a virtual machine which can provide the packages to the rest of the nodes in the cluster. The local repository will be used for deployment installations and package updates.

2.12.2 Components

Upstream distribution primary repos and EPEL will imported to `/opt/alces/repo/` with `reposync`, any upstream repo groups will also be imported to allow node redeployment without internet access (and a known working disaster recovery configuration).

2.12.3 Key Files

- `/etc/yum.repos.d/*`
- `/opt/alces/repo/*`

2.13 Repository Mirror Server

2.13.1 On Master Node

- Create `/opt/vm/repo.xml` for deploying the repo VM (`Available here`)
- Create disk image for the repo VM:

```
qemu-img create -f qcow2 repo.qcow2 150G
```

- Define the VM:

```
virsh define repo.xml
```

2.13.2 On Controller VM

- Create a group for the repo VM (add at least `repo1` as a node in the group, set additional groups of `services,cluster,domain` allows for more diverse group management):

```
metal configure group repo
```

- Customise `repo1` node configuration (set the primary IP address to 10.10.0.2):

```
metal configure node repo1
```

- Create a deployment file specifically for `repo1` at `/var/lib/metalware/repo/config/repo1.yaml` with the following content:

```
yumrepo:
  is_server: true
```

- Add the following to `/var/lib/metalware/repo/config/domain.yaml` (`build_url` is the URL for client kickstart builds to use, `source_repos` should be a comma-separated list of source files that repoman will use to generate client configurations, `clientrepofile` will need to be a URL to a repo config file for the client to curl):

```
yumrepo:
  # Repostiroy URL for kickstart builds
  build_url: http://mirror.ox.ac.uk/sites/mirror.centos.org/7/os/x86_64/
  # If true, this server will host a client config file for the network
  is_server: false
  # Repoman source files for repository mirror server to use (comma separate)
  source_repos: base.upstream
  # The file for clients to curl containing repository information [OPTIONAL]
  # clientrepofile: http://myrepo.com/repo/client.repo
  clientrepofile: false
```

Note: See the repoman project page for the included repository template files. To add customised repositories, create them in `/var/lib/repoman/templates/centos/7/` on the repository server.

- Additionally, add the following to the `setup:` namespace list in `/var/lib/metalware/repo/config/domain.yaml`:

```
- /opt/alces/install/scripts/00-repos.sh
```

- Download the `repos.sh` script to the above location:

```
mkdir -p /opt/alces/install/scripts/
cd /opt/alces/install/scripts/
wget  -O 00-repos.sh https://raw.githubusercontent.com/alces-software/
↪knowledgebase/master/epel/7/repo/repos.sh
```

Note: The script is renamed to `00-repos.sh` to guarantee that it is run before any other setup scripts.

- Follow *Client Deployment Example* to setup the repo node
- The repo VM will now be up and can be logged in with passwordless SSH from the controller VM and will have a clone of the CentOS upstream repositories locally. Modify `build_url` in `/var/lib/metalware/repo/config/domain.yaml` to be the, now built, repo server's URL so that new client builds will use that repository.

2.13.3 Custom Repository Setup

As well as using different sources for the upstream repositories it is beneficial to have a local repository that can be used to serve additional packages which are not part of upstream repos to clients. This will be known as the custom repository, details on setting up the custom repository are below. The purpose of this repository is to provide packages to the network that aren't available in upstream repositories or require higher installation priority than other available packages (e.g. a newer kernel package).

Manual

- Install package dependencies:

```
yum -y install createrepo httpd yum-plugin-priorities yum-utils
```

- Create custom repository directory:

```
mkdir -p /opt/alces/repo/custom/
```

- Define the repository:

```
cd /opt/alces/repo/
createrepo custom
```

- Create a repo source file to be served to clients at `/var/lib/repoman/templates/centos/7/custom.local`:

```
[custom]
name=custom
baseurl=http://myrepo.com/repo/custom/
description=Custom repository local to the cluster
enabled=1
skip_if_unavailable=1
gpgcheck=0
priority=1
```

- Add the custom repository to the source repos in `/var/lib/metalware/repo/config/domain.yaml`:

```
yumrepo:
   # Repostiroy URL for kickstart builds
   build_url: http://mirror.ox.ac.uk/sites/mirror.centos.org/7/os/x86_64/
   # If true, this server will host a client config file for the network
   is_server: false
   # Repoman source files for repository mirror server to use (comma separate)
   source_repos: base.upstream,custom.local
   # The file for clients to curl containing repository information [OPTIONAL]
   # clientrepofile: http://myrepo.com/repo/client.repo
   clientrepofile: false
```

Repoman Setup

Alternatively to manually creating the custom repository, the repoman command can handle the setup of a custom repository:

```
/opt/repoman/repoman.rb mirror --distro centos7 --include base.upstream --reporoot /
↪opt/alces/repo --configurl http://myrepo.com/repo --custom
```

2.14 User Management Overview

2.14.1 About

This package contains the services required to configure a central user management server for the HPC environment. This relieves the need to manage `/etc/passwd` locally on every system within the HPC environment and provides further authentication management of different services.

2.14.2 Components

For user management, one of the following software solutions will be implemented:

- NIS (Network Information Service)
  - The Network Information Service (NIS) is a directory service that enables the sharing of user and host information across a network.

or

- IPA (Identity, Policy, Audit)
  - FreeIPA provides all the information that NIS does as well as providing application and service information to the network. Additionally, FreeIPA uses directory structure such that information can be logically stored in a tree-like structure. It also comes with a web interface for managing the solution.

2.14.3 Key Files

- `/etc/sysconfig/network`
- `/etc/yp.conf`, `/var/yp/*`

or

- `/etc/ipa/*`

2.15 Considerations for User Management

2.15.1 User Authentication

User authentication is usually performed in a server/client setup inside the HPC environment due to the unnecessary overhead of manually maintaining `/etc/passwd` on a network of nodes. A few options for network user management are:

- **NIS** - The Network Information Service (NIS) is a directory service that enables the sharing of user and host information across a network.
- **FreeIPA** - FreeIPA provides all the information that NIS does as well as providing application and service information to the network. Additionally, FreeIPA uses directory structure such that information can be logically stored in a tree-like structure. It also comes with a web interface for managing the solution.
- Connecting to an **externally-managed** user-authentication service (e.g. LDAP, active-directory). This option is not recommended, as a large HPC cluster can put considerable load on external services. Using external user-authentication also creates a dependancy for your cluster on another service, complicating troubleshooting and potentially impacting service availability.

Note: If the user accounts need to be consistent with accounts on the external network then the master node should have a slave service to the external networks account management system. This will allow the account information to be forwarded to the HPC network, without creating a hard-dependancy on an external authentication service.

2.15.2 User Access

It is also worth considering how users will be accessing the system. A few ways that users can be accessing and interacting with the HPC environment are:

- **SSH** - This is the most common form of access for both users and admins. SSH will provide terminal-based access and X forwarding capabilities to the user.
- **VNC** - The VNC service creates a desktop session that can be remotely connected to by a user, allowing them to run graphical applications inside the HPC network.
- **VPN** - A VPN will provide remote network access to the HPC environment. This can be especially useful when access to the network is required from outside of the external network. Once connected to the VPN service, SSH or VNC can be used as it usually would be.

Note: If running firewall services within the environment (recommended) then be sure to allow access from the ports used by the selected user access protocols.

2.15.3 Additional Considerations and Questions

- What information will need to be shared between the systems?
- How will users want to access the system?

2.16 Recommendations for User Management

Below are guidelines for setting up both a NIS and an IPA server, only one of these should be setup to prevent conflicts and inconsistencies in user management.

2.16.1 NIS Server Setup

On Master Node

- Create `/opt/vm/nis.xml` for deploying the nis VM (`Available here`)
- Create disk image for the nis VM:

```
qemu-img create -f qcow2 nis.qcow2 80G
```

- Define the VM:

```
virsh define nis.xml
```

On Controller VM

- Create a group for the nis VM (add at least `nis1` as a node in the group, set additional groups of `services, cluster, domain` allows for more diverse group management):

```
metal configure group nis
```

- Customise `nis1` node configuration (set the primary IP address to 10.10.0.4):

```
metal configure node nis1
```

- Create a deployment file specifically for `nis1` at `/var/lib/metalware/repo/config/nis1.yaml` with the following content:

```
nisconfig:
  is_server: true
```

- Add the following to `/var/lib/metalware/repo/config/domain.yaml` (the nisserver IP should match the one specified for `nis1`):

```
nisconfig:
  nisserver: 10.10.0.4
  nisdomain: nis.<%= config.domain %>
  is_server: false
  # specify non-standard user directory [optional]
  users_dir: /users
```

- Additionally, add the following to the `setup:` namespace list in `/var/lib/metalware/repo/config/domain.yaml`:

```
- /opt/alces/install/scripts/02-nis.sh
```

- Download the `nis.sh` script to the above location:

```
mkdir -p /opt/alces/install/scripts/
cd /opt/alces/install/scripts/
wget -O 02-nis.sh https://raw.githubusercontent.com/alces-software/knowledgebase/
↪master/epel/7/nis/nis.sh
```

- Follow *Client Deployment Example* to setup the compute nodes

2.16.2 IPA Server Setup

On Master Node

- Create `/opt/vm/ipa.xml` for deploying the IPA VM (`Available here`)
- Create disk image for the IPA VM:

```
qemu-img create -f qcow2 IPA.qcow2 80G
```

- Define the VM:

```
virsh define IPA.xml
```

On Controller VM

- Create a group for the IPA VM (add at least `ipa1` as a node in the group, set additional groups of `services,cluster,domain` allowing for more diverse group management):

```
metal configure group ipa
```

- Customise `ipa1` node configuration (set the primary IP address to 10.10.0.4):

```
metal configure node ipa1
```

- Add the following to `/var/lib/metalware/repo/config/domain.yaml` (the ipaserver IP should match the one specified for `ipa1`):

```
ipaconfig:
  serverip: 10.10.0.4
  servername: ipa1
  insecurepassword: abcdef123
  userdir: /users
```

- Additionally, add the following to the `scripts:` namespace list in `/var/lib/metalware/repo/config/domain.yaml` (this script runs the client-side configuration of IPA):

```
- /opt/alces/install/scripts/02-ipa.sh
```

- Download the `ipa.sh` script to the above location:

```
mkdir -p /opt/alces/install/scripts/
cd /opt/alces/install/scripts/
wget -O 02-ipa.sh https://raw.githubusercontent.com/alces-software/knowledgebase/
↪master/epel/7/ipa/ipa.sh
```

- Follow *Client Deployment Example* to setup the IPA node and continue to the next session to configure the IPA server with a script

Setup IPA Server

- Download the server configuration script to the controller:

```
cd /opt/alces/install/scripts/
wget http://raw.githubusercontent.com/alces-software/knowledgebase/master/epel/7/
↪ipa/ipa_server.sh
```

- Render the script for the IPA server:

```
metal render /opt/alces/install/scripts/ipa_server.sh ipa1 > /tmp/ipa_server.sh
```

- Copy the script to the IPA server:

```
scp /tmp/ipa_server.sh ipa1:/root/
```

Note: Before launching the script it is currently necessary to disable named on the controller from serving the primary forward and reverse domains such that the IPA installation will work. This can be re-enabled once the IPA script has finished running.

- Launch the script on the IPA server (following any on-screen prompts):

```
ssh ipa1 "/root/ipa_server.sh"
```

2.16.3 IPA Replica Server Setup

For this example, the servers are as follows:

- `infra01-dom0` - The IPA server
- `infra01-domX` - The server to replicate IPA

Configuring Replica Host (on `infra01-domX`)

- Install IPA tools:

```
yum -y install ipa-server bind bind-dyndb-ldap ipa-server-dns
```

- Configure DNS to use `infra01-dom0` as nameserver in `/etc/resolv.conf`

Preparing Server (on `infra01-dom0`)

- Add `infra01-domX` as a host:

```
ipa host-add infra01-domX --password="MyOneTimePassword" --ip-address=10.10.2.51
```

- Add `infra01-domX` to `ipaservers` group:

```
ipa hostgroup-add-member ipaservers --hosts=infra01-domX
```

Connecting Replica Host (on `infra01-domX`)

- Run the replica installation:

```
ipa-replica-install --realm="PRI.CLUSTER.ALCES.NETWORK" --server="infra01-dom0.
↪pri.cluster.alces.network" --domain="pri.cluster.alces.network" --password=
↪"MyOneTimePassword"
```

2.17 Storage Overview

2.17.1 About

This package configures an NFS master server to provide user and data filesystems to all slave nodes.

2.17.2 Components

The storage solution is comprised of the following:

- NFS server
  - The NFS server serves redundant network storage (depending on hardware and network configuration) to the client systems in the environment. This allows for distributed access to project and user data.
- Filesystem formatting
  - Ext4

  or - XFS
- Exported filesystems
  - `/export/users` to be mounted at `/users` on clients
  - `/export/data` to be mounted at `/data` on clients

2.17.3 Key Files

- `/etc/sysconfig/nfs`
- `/etc/exports`
- `/etc/fstab`

2.18 Considerations for Storage Solution

2.18.1 Storage Hardware

When selecting the storage solution it is worth considering the size, performance and resilience of the desired storage solution. Usually some sort of storage array will be used; e.g. a collection of disks (otherwise known as JBOD - Just a Bunch Of Disks) in the form of an internal or external RAID array.

2.18.2 Type of filesystem

For many requirements, a simple NFS solution can provide sufficient performance and resiliency for data. Application, library and source-code files are often small enough to be stored on an appropriately sized NFS solution; such storage systems can even be grown over time using technologies like LVM and XFS as requirements increase.

For data-sets that require high capacity (>100TB) or high-performance (>1GB/sec) access, a parallel filesystem may be suitable to store data. Particularly well suited to larger files (e.g. at least 4MB per storage server), a parallel filesystem can provide additional features such as byte-range locking (the ability for multiple nodes to update sections of a large file simultaneously), and MPI-IO (the ability to control data read/written using MPI calls). Parallel filesystems work by aggregating performance and capacity from multiple servers and allowing clients (your cluster compute and login nodes) to mount the filesystem as a single mount-point. Common examples include:

- Lustre; an open-source kernel-based parallel filesystem for Linux
- GPFS (general purpose file system; a proprietary kernel-based parallel filesystem for Linux
- BeeGFS; an open-source user-space parallel filesystem for Linux

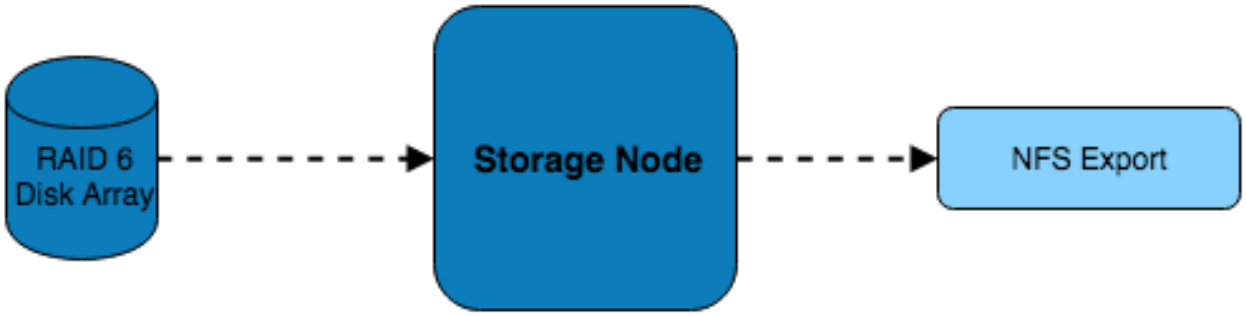
Your choice of filesystem will depend on the features you require, and your general familiarity with the technologies involved. As parallel filesystems do not perform well for smaller files, it is very common to deploy a parallel filesystem alongside a simple NFS-based storage solution.

If your data-set contains a large number of files which need to be kept and searchable for a long time (> 12-months) then an object storage system can also be considered. Accessed using client-agnostic protocols such as HTTPS, an object storage system is ideal for creating data archives which can include extended metadata to assist users to locate and organise their data. Most object storage systems include data redundancy options (e.g. multiple copies, object versioning and tiering), making them an excellent choice for long-term data storage. Examples of object-storage systems include:

- AWS Simple Storage Service (S3); a cloud-hosted service with a range of data persistence options
- Swift-stack; available as both on-premise and cloud-hosted services, the SWIFT protocol is compatible with a wide range of software and services
- Ceph; an on-premise object storage system with a range of interfaces (block, object, S3, POSIX file)

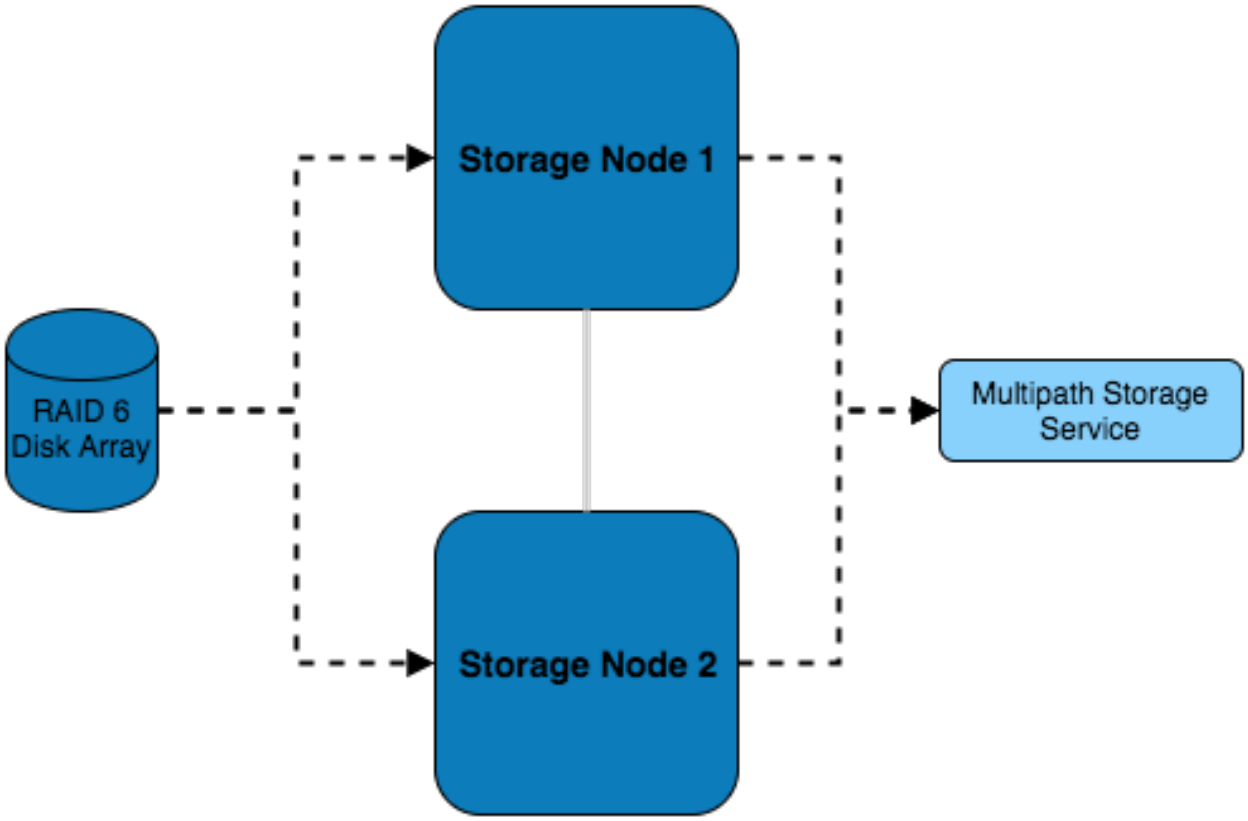
2.18.3 Network Storage Solutions

Single server with NFS



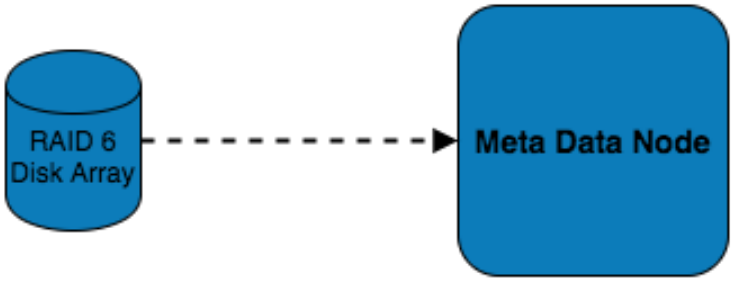
In this example, a single server is connected to a RAID 6 storage array which it is serving over NFS to the systems on the network. While simple in design and implementation, this design only provides redundancy at the RAID level.

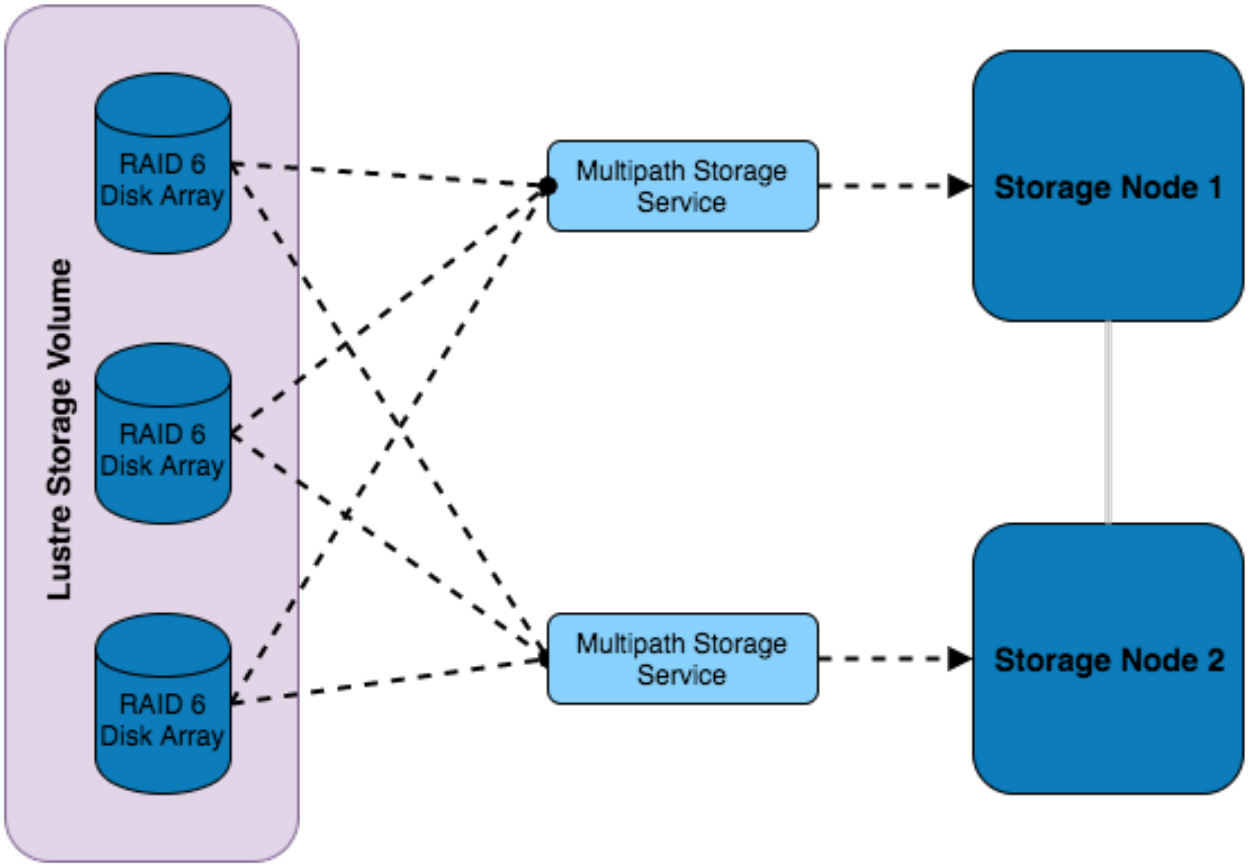
Multiple Servers with NFS



In addition to the previous example, this setup features multiple storage servers which balance the load of serving the disk over NFS.

Multiple Servers with Parallel filesystem





This setup features multiple RAID sets which are installed externally to the storage servers and are connected to both of them using multipath - this allows for multiple paths to the storage devices to be utilised. Using this storage, a Lustre volume has been configured which consists of a combination of all the external disks. Authorisation of access to the storage volume is managed by the metadata node, which also has dedicated storage.

2.18.4 Additional Considerations and Questions

- What data will need to be centrally stored?
- Where will data be coming from?

  - Are source files created within the HPC network or do they exist in the external network?
  - Will compute nodes be writing out logs/results from running jobs?
  - Where else might data be coming from?
- Is scratch space needed?
- What level of redundancy/stability is required for the data?
- How will the data be backed up?
  - Will there be off-site backups?
  - Should a separate storage medium be used?
  - Does all the data need backing up or only certain files?
  - For how long will we keep the data, and any backups created?
- What are my disaster recovery and business continuity plans?
  - If the storage service fails, how will my users continue working?
  - How can I recreate the storage service if it fails?
  - How long will it take to restore any data from backups?

2.19 Recommendations for Storage Solution

2.19.1 Storage Hardware

For recommended storage hardware technologies, see https://github.com/alces-software/knowledgebase/wiki#storage

2.19.2 Changing Disk Formatting for a Node/Group

In the metalware domain configuration files, the `disksetup` namespace configures the kickstart commands for disk formatting. A couple of example configurations are below.

Default disk configuration:

```
disksetup: |
  zerombr
  bootloader --location=mbr --driveorder=sda --append="$bootloaderappend"
  clearpart --all --initlabel

  #Disk partitioning information
  part /boot --fstype ext4 --size=4096 --asprimary --ondisk sda
  part pv.01 --size=1 --grow --asprimary --ondisk sda
  volgroup system pv.01
  logvol  /  --fstype ext4 --vgname=system  --size=16384 --name=root
  logvol  /var --fstype ext4 --vgname=system --size=16384 --name=var
  logvol  /tmp --fstype ext4 --vgname=system --size=1 --grow --name=tmp
  logvol  swap  --fstype swap --vgname=system  --size=8096  --name=swap1
```

Software RAID configuration:

```
disksetup: |
  zerombr

  bootloader --location=mbr --driveorder=sda --append="$bootloaderappend"
  clearpart --all --initlabel

  #Disk partitioning information
  part /boot --fstype ext4 --size=1024 --asprimary --ondisk sda
  part /boot2 --fstype ext4 --size=1024 --asprimary --ondisk sdb

  part raid.01 --size 60000 --ondisk sda --asprimary
  part raid.02 --size 60000 --ondisk sdb --asprimary

  raid pv.01 --level=1 --fstype=ext4 --device=md0 raid.01 raid.02
  volgroup system pv.01
  logvol  /  --fstype ext4 --vgname=system  --size=1  --name=root --grow
  logvol  /var  --fstype ext4 --vgname=system  --size=16384  --name=var
  logvol  swap  --fstype swap --vgname=system  --size=16384  --name=swap1

  part raid.03 --size=1 --ondisk sda --asprimary --grow
  part raid.04 --size=1 --ondisk sdb --asprimary --grow

  raid /tmp --fstype ext4 --fstype=ext4 --device=md1 --level=0 raid.03 raid.04
```

To override the default disk configuration, create a config file with the node/group name in `/var/lib/metalware/repo/config/` and add the new `disksetup:` key to it.

2.19.3 NFS Server Setup

On Master Node

- Create `/opt/vm/storage.xml` for deploying the storage VM (`Available here`)
- Create disk image for the storage VM:

```
qemu-img create -f qcow2 storage.qcow2 80G
```

- Define the VM:

```
virsh define storage.xml
```

On Controller VM

- Create a group for the storage VM (add at least `storage1` as a node in the group, set additional groups of `services,cluster,domain` allows for more diverse group management):

```
metal configure group storage
```

- Customise `storage1` node configuration (set the primary IP address to 10.10.0.3):

```
metal configure node storage1
```

- Create `/var/lib/metalware/repo/config/storage1.yaml` with the ip definition:

```
nfsconfig:
  is_server: true

nfsexports:
  /export/users:
  /export/data:
    # Modify the export options [optional]
    #options: <%= config.networks.pri.network %>/<%= config.networks.pri.netmask
↪%>(ro,no_root_squash,async)
```

Note: The `options:` namespace is optional, if not specified then the default export options will be used (`<%= config.networks.pri.network %>/<%= confignetworks.pri.netmask %>(rw, no_root_squash,sync)`)

- Add the following to `/var/lib/metalware/repo/config/domain.yaml` (toggle `defined` to `false` to prevent a client from creating an fstab entry for the mount on a node):

```
nfsconfig:
  is_server: false
nfsmounts:
  /users:
    defined: true
    server: 10.10.0.3
    export: /export/users
  /data:
    defined: true
    server: 10.10.0.3
    export: /export/data
    options: intr,sync,rsize=32768,wsize=32768,_netdev
```

Note: Add any NFS exports to be created as keys underneath `nfsmounts:`. The `options:` namespace is only needed if wanting to override the default mount options (`intr,rsize=32768,wsize=32768,_netdev`)

- Additionally, add the following to the `setup:` namespace list in `/var/lib/metalware/repo/config/domain.yaml`:

```
- /opt/alces/install/scripts/01-nfs.sh
```

- Download the `nfs.sh` script to the above location:

```
mkdir -p /opt/alces/install/scripts/
cd /opt/alces/install/scripts/
wget -O 01-nfs.sh https://raw.githubusercontent.com/alces-software/knowledgebase/
↪master/epel/7/nfs/nfs.sh
```

- Follow *Client Deployment Example* to setup the compute nodes

2.19.4 Lustre Server Setup

On Master Node

- Create `/opt/vm/lustre-mds.xml` for deploying the lustre metadata server VM (`Available here`)

- Create disk image for the lustre metadata server VM:

```
qemu-img create -f qcow2 lustre-mds.qcow2 80G
```

- Define the VM:

```
virsh define lustre-mds.xml
```

On Controller VM

- Create a group for the lustre VM (add at least `lustre-mds1` as a node in the group, set additional groups of `lustre,services,cluster,domain` allows for more diverse group management):

```
metal configure group lustre-mds
```

- Customise `lustre-mds1` node configuration (set the primary IP address to 10.10.0.10):

```
metal configure node lustre-mds1
```

- Create a deployment file specifically for `lustre-mds1` at `/var/lib/metalware/repo/config/lustre-mds1.yaml` with the following content:

```
lustreconfig:
  type: server
  networks: tcp0(<%= config.networks.pri.interface %>)
  mountentry: "10.10.0.10:/lustre    /mnt/lustre    lustre    default,_netdev
↪0 0"
```

Note: If the server has an Infiniband interface that can be used for storage access then set `networks` to a list of modules which includes Infiniband, e.g. `o2ib(<%= config.networks.ib.interface %>),tcp0(<%= config.networks.pri.interface %>)`

- Add the following to `/var/lib/metalware/repo/config/domain.yaml`:

```
lustreconfig:
  type: none
  networks: tcp0(<%= config.networks.pri.interface %>)
  mountentry: "10.10.0.10:/lustre    /mnt/lustre    lustre    default,_netdev
↪0 0"
```

Note: For clients to lustre, replicate the above entry into the group or node config file and change `type: none` to `type: client`, also ensuring that `networks` reflects the available modules and interfaces on the system

- Additionally, add the following to the `setup:` namespace list in `/var/lib/metalware/repo/config/domain.yaml`:

```
- /opt/alces/install/scripts/08-lustre.sh
```

- Download the `lustre.sh` script to the above location:

```
mkdir -p /opt/alces/install/scripts/
cd /opt/alces/install/scripts/
wget -O 08-lustre.sh https://raw.githubusercontent.com/alces-software/
↪knowledgebase/master/epel/7/lustre/lustre.sh
```

- Follow *Client Deployment Example* to setup the lustre node
- Once this has completed the lustre-mds node will have the necessary configuration to host a lustre metadata target or storage configuration. To configure the metadata disk or storage configuration see the below section.

Lustre Storage Setup

A lustre storage configuration usually consists of a metadata server (that is used to authorise mount, read and write requests to the lustre storage volume) and multiple storage servers (with disk arrays attached to them). The above configuration shows how a metadata server can be configured as part of the network but with some naming tweaks the lustre storage servers can also be added to the environment.

Metadata Storage Target

- To format a metadata storage disk from the metadata server run a command similar to the following (replacing `lustre` with the desired name of the lustre filesystem and `/dev/sda` with the path to the disk for storing metadata):

```
mkfs.lustre --index=0 --mgs --mdt --fsname=lustre --servicenode=10.10.0.10 --
↪reformat /dev/sda
```

- To activate the storage, mount it somewhere on the metadata server:

```
mount -t lustre /dev/sda /mnt/lustre/mdt
```

Lustre Storage Target

These commands should be performed from different systems connected to the same storage backends across the storage configuration (depending on the network configuration) to ensure that the device management is distributed.

- A storage target for the lustre filesystem can be formatted as follows (replacing `lustre` with the name of the filesystem from mdt configuration, repeat `--servicenode=IP-OF-OSSX` for each storage system that is also connected to the same storage backend and replace `/dev/mapper/ostX` with the path to the storage device):

```
mkfs.lustre --ost --index=0 --fsname=lustre --mgsnode=IP-OF-MDS-NODE --
↪mkfsoptions="-E stride=32,stripe_width=256" --servicenode=IP-OF-OSSX /dev/
↪mapper/ostX
```

- The device can then be mounted:

```
mount -t lustre /dev/mapper/ostX /mnt/lustre/ostX
```

Client Mount

- The following command will mount the example lustre volume created from the above steps:

```
mount -t lustre 10.10.0.10:/lustre /mnt/lustre
```

2.20 Monitoring Overview

2.20.1 About

This package will configure a monitoring master system with metric collection services and a web front-end. Slave nodes will have the client monitoring service setup to send metrics to the master system.

2.20.2 Components

The monitoring system will provide the following applications:

- Ganglia, a passive monitoring system with metric graphs
  - Clients send metrics to the Ganglia host system which are plotted on graphs for viewing data trends for the environment. The data is available through both a command-line utility and a web interface.
- Nagios, an active monitoring system with notifications
  - Nagios clients are configured on the server and are not required to run client software unless additional metrics are needed. The system actively monitors metrics and if the values go over a predefined, customisable threshold.

2.20.3 Key Files

- `/etc/ganglia/gmetad.conf`
- `/etc/ganglia/gmond.conf`
- `/etc/httpd/conf.d/ganglia`
- `/var/lib/ganglia/*`
- `/etc/nagios/*`
- `/etc/httpd/conf.d/nagios.conf`
- `/usr/share/nagios/*`

2.21 Considerations for Monitoring the HPC Platform

2.21.1 Types of Monitoring

There are 2 types of monitoring that can be implemented into a network; these are:

- **Passive** - Passive monitoring tools collect data and store from systems. Usually this data will be displayed in graphs and is accessible either through command-line or web interfaces. This sort of monitoring is useful for historical metrics and live monitoring of systems.
- **Active** - Active monitoring collects and checks metrics; it will then send out notifications if certain thresholds or conditions are met. This form of monitoring is beneficial for ensuring the health of systems; for example, email notifications can be sent out when systems start overheating or if a system is no longer responsive.

Both forms of monitoring are usually necessary in order to ensure that your HPC cluster is running properly, and in full working order.

2.21.2 Metrics

It is worth considering what metrics for the system will be monitored; a few common ones are listed here:

- CPU
  - Load average
  - Idle percentage
  - Temperature
- Memory
  - Used
  - Free
  - Cached
- Disk
  - Free space
  - Used space
  - Swap (free/used/total)
  - Quotas (if configured)
- Network
  - Packets in
  - Packets out

Note: Cloud service providers usually have both passive and active monitoring services available through their cloud management front-end.

2.21.3 Additional Considerations and Questions

- What metrics should be monitored?
- How frequently should metrics be checked?
- What level of notification is required?
  - Escalation upon repeated errors?
  - Acknowledgement of long-running outages?
  - How do we avoid over-saturation of notifications during major outages?
  - What tests will we run to ensure that notifications are working properly?

2.22 Recommendations for Monitoring the HPC Platform

2.22.1 Setting Up Monitor Server (Ganglia & Nagios)

On Master Node

- Create `/opt/vm/monitor.xml` for deploying the storage VM (`Available here`)
- Create disk image for the monitor VM:

```
qemu-img create -f qcow2 monitor.qcow2 80G
```

- Define the VM:

```
virsh define monitor.xml
```

On Controller VM

- Create a group for the monitor VM (add at least `monitor1` as a node in the group, set additional groups of `services,cluster,domain` allows for more diverse group management):

```
metal configure group monitor
```

- Customise `monitor1` node configuration (set the primary IP address to 10.10.0.5):

```
metal configure node monitor1
```

- Create `/var/lib/metalware/repo/config/monitor1.yaml` with the following network and server definition:

```
ganglia:
  is_server: true

nagios:
  is_server: true
```

- Add the following to `/var/lib/metalware/repo/config/domain.yaml`:

```
ganglia:
  server: 10.10.0.5
  is_server: false
nagios:
  is_server: false
```

- Additionally, add the following to the `setup:` namespace list in `/var/lib/metalware/repo/config/domain.yaml`:

```
- /opt/alces/install/scripts/03-ganglia.sh
- /opt/alces/install/scripts/04-nagios.sh
```

- Download the `ganglia.sh` and `nagios.sh` scripts to the above location:

```
mkdir -p /opt/alces/install/scripts/
cd /opt/alces/install/scripts/
wget -O 03-ganglia.sh https://raw.githubusercontent.com/alces-software/
↪knowledgebase/master/epel/7/ganglia/ganglia.sh
wget -O 04-nagios.sh https://raw.githubusercontent.com/alces-software/
↪knowledgebase/master/epel/7/nagios/nagios.sh
```

- Follow *Client Deployment Example* to setup the compute nodes

This will setup minimal installations of both Ganglia and Nagios. All nodes within the domain will be built to connect to these services such that they can be monitored. It is possible to expand upon the metrics monitored and notification preferences.

Once deployed, both Ganglia and Nagios services can be further configured and customised to deliver the metrics and notifications that you require. Review the documentation for each of the projects for assistance in settings up these tools.

2.23 Hardware Drivers Overview

2.23.1 About

Some clusters may require custom hardware drivers for your chosen operating system. This can be the case for both bare-metal hardware and cloud-based resources for devices such as Infiniband or other accelerated network technologies, GPU and FPGA cards or RAID controllers. This package will create an environment that allows nodes to run installers during first-boot to build the driver against the up-to-date operating system packages.

2.23.2 Components

The components in this package are:

- First boot environment for automatically executing installers at boot time
  - This provides a system service that can be used to run installation scripts when a system is turned on. For example, Nvidia graphics driver can be set to compile after the initial build of the system such that it is compiled against the correct kernel version.
- Infiniband driver first-boot script
  - A script that can install the Mellanox Infiniband driver either from upstream repositories or through local compilation from the source files
- Nvidia graphics driver first-boot script

2.23.3 Key Files

- `/etc/systemd/system/firstrun.service`
- `/var/lib/firstrun/*`
- `/var/log/firstrun/*`
- `/opt/alces/installers/*`

Note: In order to use first-boot, the system must be compatible with the base operating system. Review instructions for your chosen operating system if you need to use special drivers in order to allow your nodes to install the base OS.

2.24 First Boot Script Environment

The first boot environment is a service that allows for scripts to be executed on a node at startup, occurring only on the first boot after system build.

2.24.1 Creating a First Boot Script

A first boot script is made up of two main components:

- Setup script
- First run script

Setup Script

This script will run as part of the node build procedure and will be used to put the first run script into the correct location to be executed at boot time.

- Create a script like the following example (replace `myfile.bash` with the name of the program and between `cat` and `EOF` with the installation commands):

```
cat << EOF > /var/lib/firstrun/scripts/myfile.bash
curl http://www.system-driver.com/downloads/installer.sh > /tmp/installer.sh
sh /tmp/installer.sh --quiet
EOF
```

- The above script can then be saved somewhere under `/opt/alces/install/scripts/` on the deployment VM
- In `/var/lib/metalware/repo/config/domain.yaml` (or a group/node specific config file) add the path to the script in the `setup:` namespace

First Run Script

In the example setup script above it creates a file called `/var/lib/firstrun/scripts/myfile.bash` which is the first run script. Any files ending with `.bash` in `/var/lib/firstrun/scripts` will be executed on the first boot of the node.

2.25 Compute Node Setup

2.25.1 Infiniband Setup

- Create a configuration file specifically for the nodes group `/var/lib/metalware/repo/config/nodes.yaml` with the ib network setup:

```
networks:
  ib:
    defined: true
    ib_use_installer: false
    mellanoxinstaller: http://route/to/MLNX_OFED_LINUX-x86_64.tgz
    ip:
```

Note: If you want to install the Mellanox driver (and not use the IB drivers from the CentOS repositories), set `ib_use_installer` to `true` and set `mellanoxinstaller` to the location of the mellanox OFED installer.

- Download the `infiniband.sh` script from the knowledgebase:

```
mkdir -p /opt/alces/install/scripts/
cd /opt/alces/install/scripts/
wget -O 06-infiniband.sh https://raw.githubusercontent.com/alces-software/
↪knowledgebase/master/epel/7/infiniband/infiniband.sh
```

- Add the script to the `scripts:` namespace list in `/var/lib/metalware/repo/config/domain.yaml`:

```
- /opt/alces/install/scripts/06-infiniband.sh
```

- Follow *Client Deployment Example* to setup the compute nodes

2.25.2 Nvidia Driver Setup

- Download the Nvidia installer to `/opt/alces/installers/` on the controller VM as `nvidia.run`
- Download the `nvidia.sh` script from the knowledgebase:

```
mkdir -p /opt/alces/install/scripts/
cd /opt/alces/install/scripts/
wget -O 07-nvidia.sh https://raw.githubusercontent.com/alces-software/
↪knowledgebase/master/epel/7/nvidia/nvidia.sh
```

- Add the script to the `scripts:` namespace list in `/var/lib/metalware/repo/config/domain.yaml`:

```
- /opt/alces/install/scripts/07-nvidia.sh
```

- To run the installer on all nodes in a group (for example, `gpunodes`) add the following line to the group's config file (in this example, `/var/lib/metalware/config/gpunodes.yaml`):

```
nvidia: true
```

2.26 HPC Environment Overview

2.26.1 About

This package provides tools for queuing jobs and running applications across the cluster.

2.26.2 Components

The HPC environment will be comprised of:

- Queuing system for optimised resource utilisation
  - SLURM
    - The SLURM job scheduler is a centrally managed job scheduler that cant constrain resources based on grid utilisation, user/group assignment and job resource requirements.

  or

  - Open Grid Scheduler

    * Much like SLURM, OGS provides a centrally managed job scheduler with similar resource management possibilities.

- Application deployment solution
  - Environment modules
    * Modules allows for applications to be loaded dynamically in shell sessions. With the paths being updated on-the-fly, applications can be installed to a network storage location - minimising installation time and improving the ease of application use (both in interactive and non-interactive sessions).

    and/or

  - Alces Gridware
    * Gridware contains an implementation of environment modules and also provides a useful CLI tool for installing and managing applications from a large repository of HPC applications.

2.26.3 Key Files

- `/etc/slurm/*` *or* `/opt/gridscheduler/*`
- `/opt/apps/*` *or* `/opt/gridware/`

2.27 Considerations for HPC Environment Design

2.27.1 Job Scheduling

In a HPC environment there are large, distributed, multiple processor jobs that the users wish to run. While these jobs can be run manually by simply executing job scripts along with a hostfile containing the compute nodes to execute on, you will soon run into problems with multiple users, job queuing and priorities. These features are provided by a job scheduler, delivering a centralised server that manages the distribution, prioritisation and execution of job scripts from multiple users in the HPC network.

Popular job schedulers for HPC clusters include:

- Open Grid Scheduler (SGE)
- PBS / Torque-Maui / Torque-Moab / PBSPro
- SLURM
- Load-sharing facility (LSF) / OpenLava

All job schedulers provide a similar level of functionality and customisations so it is worth investigating the features of the available solutions to find the one best suited for your environment.

2.27.2 Application Deployment

General management of applications and libraries is mentioned *Repository Management* however this section focuses on installing applications into the entire HPC environment instead of individually to each node system.

A few things to consider when designing/implementing an application deployment system are:

- How will applications be stored? (central network storage location?)
- What parts of the application need to be seen by the nodes? (application data? program files? libraries?)
- How will multiple versions of the same application be installed, and how will users choose between them?

- How will dependencies be managed? (more on this below)

An application deployment system can be created yourself or Alces Gridware provides tools and an index of HPC applications for HPC platform installations.

Dependencies

When it comes to managing dependencies for applications it can either be done with local installations of libraries/packages or by storing these in a centralised location (as suggested with the applications themselves). Dependency control is one of the main reasons that using the same OS for all systems is recommended as it eliminates the risk of applications only working on some systems within the HPC environment.

Dependancies must be managed across all nodes of the cluster, and over time as the system is managed. For example, an application that requires a particular C++ library that is available from your Linux distribution may not work properly after you install distribution updates on your compute nodes. Dependancies for applications that utilise dynamic libraries (i.e. loaded at runtime, rather than compile-time) must be particularly carefully managed over time.

Reproducibility

It is important that your users receive a consistent, long-term service from your HPC cluster to allow them to rely on results from applications run at different points in your clusters' lifecycle. Consider the following questions when designing your application management system:

- How can I install new applications quickly and easily for users?
- What test plans have I created to ensure that applications run in the same way across all cluster nodes?
- How can I ensure that applications run normally as nodes are re-installed, or new nodes are added to the cluster?
- How can I test that applications are working properly after an operating system upgrade or update?
- How will I prepare for moving to a new HPC cluster created on fresh hardware, or using cloud resources?
- What are the disaster recovery plans for my software applications?

2.28 Recommendations for HPC Environment Design

2.28.1 SLURM Setup (From Controller VM)

The instructions below provide guidelines for installing the SLURM job-scheduler on your HPC cluster, and may be modified as required for alternative job-schedulers such as SGE, LSF, PBS or Torque/Maui.

- Create a group for the slurm VM (add at least `slurm1` as a node in the group, set additional groups of `services,cluster,domain` allows for more diverse group management):

```
metal configure group slurm
```

- Customise `slurm1` node configuration (set the primary IP address to 10.10.0.6):

```
metal configure node slurm1
```

- Create `/var/lib/metalware/repo/config/slurm1.yaml` with the following network and server definition:

```
slurm:
  is_server: true
```

- Add the following to `/var/lib/metalware/repo/config/domain.yaml` (set `server` to the hostname of the SLURM VM):

```
slurm:
  server: slurm1
  is_server: false
  mungekey: ff9a5f673699ba8928bbe009fb3fe3dead3c860c
```

- Additionally, add the following to the `setup:` namespace list in `/var/lib/metalware/repo/config/domain.yaml`:

```
- /opt/alces/install/scripts/06-slurm.sh
```

- Download the `slurm.sh` script to the above location:

```
mkdir -p /opt/alces/install/scripts/
cd /opt/alces/install/scripts/
wget -O 06-slurm.sh https://raw.githubusercontent.com/alces-software/
↪knowledgebase/master/epel/7/slurm/slurm.sh
```

- Build SLURM RPMs in custom repo (`/opt/alces/repo/custom/Packages`), a guide to building the SLURM RPMs can be found in the SLURM documentation. Once the packages have been moved to the previously mentioned custom repo directory, rebuild the repo with `createrepo custom`
- Follow *Client Deployment Example* to setup the SLURM node

Note: All systems that are built will have SLURM installed and the SLURM daemon running which will allow that node to submit and run jobs. Should this not be desired then the service can be permanently stopped and disabled with `systemctl disable slurmd && systemctl stop slurmd` on the node which is no longer to run SLURM.

Once your job-scheduler is installed and running, follow the documentation specific to your chosen package for instructions on how to configure the software for your particular requirements.

2.28.2 Modules Setup (From Deployment VM)

The environment modules software allows users to control their environment variables for a particular login session. This enables flexibility to use a library of installed application software whilst maintaining the correct dependencies for each package. Modules can also control environment variables that deliver user assistance (e.g. manual pages, help-files, usage examples), curate software package usage terms and manage license availability for commercial packages.

The instructions below provide a simple modules installation for your HPC environment:

- Create a group for the modules VM (add at least `apps1` as a node in the group, set additional groups of `services,cluster,domain` allows for more diverse group management):

```
metal configure group apps
```

- Customise `apps1` node configuration (set the primary IP address to 10.10.0.7):

```
metal configure node apps1
```

- Create `/var/lib/metalware/repo/config/apps1.yaml` with the following network and server definition:

```
modules:
  is_server: true
```

- Add the following to `/var/lib/metalware/repo/config/domain.yaml` (set `server` to the IP of the apps VM):

```
modules:
  server: 10.10.0.7
  directory: /opt/apps
  is_server: false
```

- Additionally, add the following to the `setup:` namespace list in `/var/lib/metalware/repo/config/domain.yaml`:

```
- /opt/alces/install/scripts/07-modules.sh
```

- Download the `modules.sh` script to the above location:

```
mkdir -p /opt/alces/install/scripts/
cd /opt/alces/install/scripts/
wget -O 07-modules.sh https://raw.githubusercontent.com/alces-software/
↪knowledgebase/master/epel/7/modules/modules.sh
```

- Follow *Client Deployment Example* to setup the apps node

Note: The apps directory can be setup on the *storage node* if one was created, this allows for all NFS exports to come from a centralised server.

2.29 Considerations for HPC Platform Verification

Before putting the system into a production environment it is worth verifying that the hardware and software is functioning as expected. The 2 key types of verification are:

- **Configuration** - Verifying that the system is functioning properly as per the server setup.
- **Performance** - Verifying that the performance of the system is as expected for the hardware, network and applications.

2.29.1 Verifying the configuration

Simple configuration tests for the previous stages of the HPC platform creation will need to be performed to verify that it will perform to user expectations. For example, the following could be tested:

- Passwordless SSH between nodes performs as expected
- Running applications on different nodes within the network
- Pinging and logging into systems on separate networks

Best practice would be to test the configuration whilst it is being setup at regular intervals to confirm functionality is still as expected. In combination with written documentation, a well practiced preventative maintenance schedule is essential to ensuring a high-quality, long-term stable platform for your users.

2.29.2 Testing System

There are multiple parts of the hardware configuration that can be tested on the systems. The main few areas are CPU, memory and interconnect (but may also include GPUs, disk drives and any other hardware that will be heavily utilised). Many applications are available for testing, including:

- Memtester
- HPL/Linpack/HPC-Challenge (HPCC)
- IOZone
- IMB
- GPUBurn

Additionally, benchmarking can be performed using whichever applications the HPC platform is being designed to run to give more representable results for the use case.

2.29.3 Additional Considerations and Questions

- How will you know that a compute node is performing at the expected level?
  - Gflops theoretical vs actual performance efficiency
  - Network performance (bandwidth, latency, ping interval)
- How can you test nodes regularly to ensure that performance has not changed / degraded?

2.30 Recommendations for HPC Platform Verification

2.30.1 Checking System Configuration

For the system configuration (depending on which previous sections have been configured), the cluster administrator should verify the following settings:

- Clocks - The current date/time is correct on all machines; clients are syncing with the controller
- User Login - Users from the chosen verification method can login to:
  - Infrastructure node(s) (should be possible for admin users only)
  - Login node(s)
  - Compute node(s)
- Storage system mounts - Mounted with correct mount options and permissions
- Job-scheduler - Jobs can be submitted and are run; nodes are all present and available on the queue
- Ganglia - All nodes present, metrics are logging
- Nagios - All nodes present, services are in positive state

2.30.2 Checking Hardware Configuration

Benchmarking Software

For general notes on running memtester, IOZone, IMB and HPL see - https://github.com/alces-software/knowledgebase/wiki/Burn-in-Testing

Further details can be found at:

- Memtester
- IOZone
- IMB
- HPL

Hardware Information

- Check CPU type:

```
pdsh -g groupname 'grep -m 1 name /proc/cpuinfo'
```

- Check CPU count:

```
pdsh -g groupname 'grep processor /proc/cpuinfo |wc -l'
```

- Check RAID active:

```
pdsh -g groupname 'cat /proc/mdstat | grep md[0-1]'
```

- Check Infiniband up/active:

```
pdsh -g groupname 'ibstatus |grep phys'
```

- Check free memory:

```
pdsh -g groupname 'free -m |grep ^Mem'
```

- Check GPU type and count:

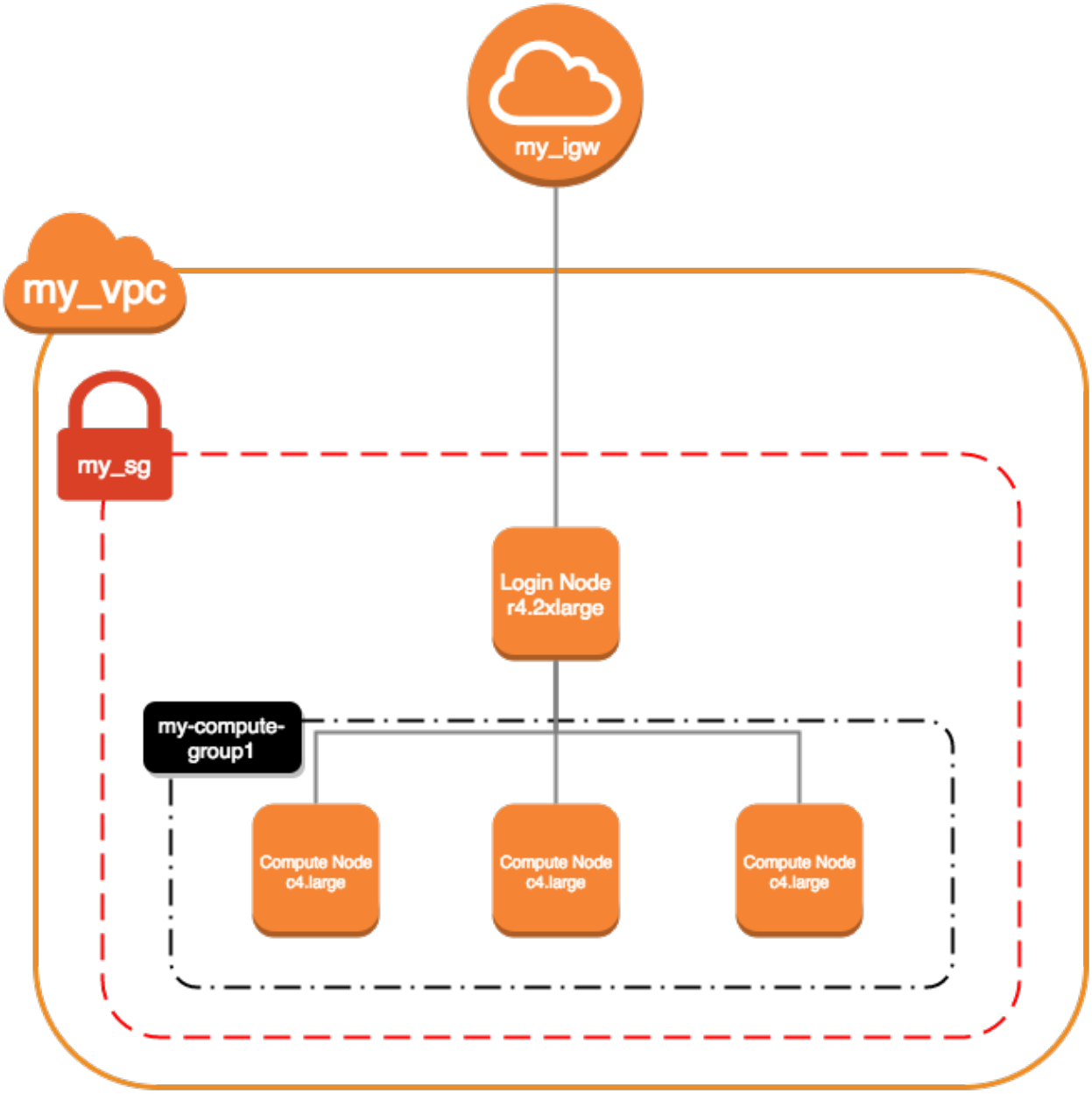
```
pdsh -g groupname 'nvidia-smi'
```

- Grab serial numbers:

```
pdsh -g groupname 'dmidecode -t baseboard |grep -i serial'
```

2.31 Setting Up AWS Environment for HPC Platform

HPC platforms can be deployed in the cloud instead of on local hardware. While there are many cloud providers out there, this guide focusses on setting up login and compute nodes in the cloud on the AWS platform.



2.31.1 General Network Setup

AWS has a command line tool that can be used to create and manage resources. These will need to be run from a Linux/Mac machine.

- Create a VPC for the network:

```
aws ec2 create-vpc --cidr-block 10.75.0.0/16
```

Note: Optionally, a name tag can be created for the VPC (which can make it easier to locate the VPC through the AWS web console) with `aws ec2 create-tags --resources my_vpc_id --tags Key=Name, Value=Name-For-My-VPC`

- Create a security group (replacing `my_vpc_id` with the VpcId from the above command output):

```
aws ec2 create-security-group --description my-sg1 --group-name my-sg1 --vpc-id␣
↪my_vpc_id
```

- Create a file `sg-permissions.json` in the current directory with the following content:

```
[
  {
    "IpProtocol": "-1",
    "IpRanges": [
      {
        "CidrIp": "10.75.0.0/16"
      }
    ]
  },
  {
    "IpProtocol": "tcp",
    "FromPort": 22,
    "ToPort": 22,
    "IpRanges": [
      {
        "CidrIp": "0.0.0.0/0"
      }
    ]
  },
  {
    "IpProtocol": "tcp",
    "FromPort": 443,
    "ToPort": 443,
    "IpRanges": [
      {
        "CidrIp": "0.0.0.0/0"
      }
    ]
  },
  {
    "IpProtocol": "tcp",
    "FromPort": 80,
    "ToPort": 80,
    "IpRanges": [
      {
        "CidrIp": "0.0.0.0/0"
      }
    ]
  }
]
```

- Add rules to security group (replacing `my_sg_id` with the GroupId from the above command output):

```
aws ec2 authorize-security-group-ingress --group-id my_sg_id --ip-permissions␣
↪file://sg-permissions.json
```

- Define subnet for the VPC (replacing `my_vpc_id` with the VpcId from earlier):

```
aws ec2 create-subnet --vpc-id my_vpc_id --cidr-block 10.75.0.0/16
```

- Create an Internet gateway:

```
aws ec2 create-internet-gateway
```

- Attach the Internet gateway to the VPC (replacing `my_igw_id` with InternetGatewayId from the above command output):

```
aws ec2 attach-internet-gateway --internet-gateway-id my_igw_id --vpc-id my_vpc_id
```

- Locate route table for the VPC:

```
aws ec2 describe-route-tables --filters Name=vpc-id,Values=my_vpc_id
```

- Create a route within the table (replacing `my_rtb_id` with RouteTableId from the above command output):

```
aws ec2 create-route --route-table-id my_rtb_id --destination-cidr-block 0.0.0.0/
↪0 --gateway-id my_igw_id
```

- Create a file `ec2-role-trust-policy.json` in the current directory with the following content:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": { "Service": "ec2.amazonaws.com"},
      "Action": "sts:AssumeRole"
    }
  ]
}
```

- Create autoscaling role:

```
aws iam create-role --role-name autoscaling --assume-role-policy-document file://
↪ec2-role-trust-policy.json
```

- Create a file `ec2-role-access-policy.json` in the current directory with the following content:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "autoscaling:DescribeAutoScalingGroups",
        "autoscaling:SetDesiredCapacity",
        "autoscaling:UpdateAutoScalingGroup",
        "autoscaling:TerminateInstanceInAutoScalingGroup",
        "ec2:DescribeTags"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

- Set role policy for above role:

```
aws iam put-role-policy --role-name my-autoscaling-role --policy-name My-
↪Autoscaling-Permissions --policy-document file://ec2-role-access-policy.json
```

- Create instance profile for autoscaling:

```
aws iam create-instance-profile --instance-profile-name autoscaling
```

- Join the role and instance profile:

```
aws iam add-role-to-instance-profile --instance-profile-name autoscaling --role-
↪name autoscaling
```

- Create a file `mapping.json` in the current directory with the following content:

```
[
  {
    "DeviceName": "/dev/sda1",
    "Ebs": {
      "DeleteOnTermination": true,
      "SnapshotId": "snap-00f18f3f6413c7879",
      "VolumeSize": 20,
      "VolumeType": "gp2"
    }
  }
]
```

2.31.2 Autoscaling Group Configuration

- Setup autoscaling launch configuration (`ami-061b1560` is the ID for the Official CentOS 7 minimal installation):

```
aws autoscaling create-launch-configuration --launch-configuration-name my-
↪compute-group1 --image-id compute_node_template_ami_id --key-name my_key_pair --
↪security-groups my_sg_id --associate-public-ip-address --iam-instance-profile␣
↪my-autoscaling-profile --instance-type c4.large --spot-price 0.113
```

- Create autoscaling group which can scale from 0 to 8 nodes and initially starts with 1:

```
aws autoscaling create-auto-scaling-group --auto-scaling-group-name my-compute-
↪group1 --launch-configuration-name my-compute-group1 --vpc-zone-identifier my_
↪subnet_id --min-size 0 --max-size 8 --desired-capacity 1
```

2.31.3 Node Creation Example

- Create node (`ami-061b1560` is the ID for the Official CentOS 7 minimal installation, replace `my_key_pair`, `my_sg_id` and `my_subnet_id` with the related values from earlier commands):

```
aws ec2 run-instances --image-id ami-061b1560 --key-name my_key_pair --instance-
↪type r4.2xlarge --associate-public-ip-address --security-group-ids my_sg_id --
↪block-device-mappings file://mapping.json --subnet-id my_subnet_id --iam-
↪instance-profile Name=my-autoscaling-profile
```

- Wait for node to launch (`instance_id` being the ID from the above command):

```
aws ec2 describe-instances --instance-id instance_id | jq '.Reservations[0].
↪Instances[0].State.Name'
```

- Identify public IP for the node to login to (`instance_id` being the ID from the above command):

```
aws ec2 describe-instances --instance-id instance_id |jq '.Reservations[0].
↪Instances[0].PublicIpAddress'
```

2.31.4 Controller Node Setup

- Follow *Node Creation Example*
- Follow *Metalware Install*

2.31.5 Repository Node Setup

- Follow *Node Creation Example*
- Follow *Repo Configuration*

2.31.6 Storage Node Setup

- Follow *Node Creation Example*
- Follow *Storage Configuration*

2.31.7 User Management Setup

- Follow *Node Creation Example*
- Follow *User Management Setup*

2.31.8 Monitor Node Setup

- Follow *Node Creation Example*
- Follow *Monitor System Setup*