

Scibian 8 HPC Installation guide

Date:

Pages: 84

Authors: CCN-HPC



Contents

1	About this document	4
1.1	Purpose	4
1.2	Structure	4
1.3	Typographic conventions	4
1.4	Build dependencies	5
1.5	License	5
1.6	Authors	5
2	Reference architecture	6
2.1	Hardware architecture	6
2.1.1	Networks	6
2.1.2	Administration cluster	7
2.1.3	User-space cluster	8
2.1.4	Storage system	8
2.2	External services	8
2.2.1	Base services	8
2.2.2	Optional services	9
2.3	Software architecture	10
2.3.1	Overview	10
2.3.2	Base Services	11
2.3.3	Additional Services	14
2.3.4	High-Availability	15
2.4	Conventions	16
2.5	Advanced Topics	16
2.5.1	Boot sequence	16
2.5.2	Frontend nodes: SSH load-balancing and high-availability	20
2.5.3	Service nodes: DNS load-balancing and high-availability	21
2.5.4	Consul and DNS integration	23
2.5.5	Scibian diskless initrd	24
3	Installation procedure	26
3.1	Overview	26
3.2	Requirements	26
3.3	Temporary installation node	28
3.3.1	Base installation	28
3.3.2	Administration environment	28
3.4	Internal configuration repository	28
3.4.1	Base directories	29
3.4.2	Organization settings	29
3.4.3	Cluster directories	30
3.4.4	Puppet configuration	30
3.4.5	Cluster definition	31
3.4.6	Service role	39
3.4.7	Authentication and encryption keys	39
3.4.8	Initial files	40
3.5	Generic service nodes	44
3.5.1	Temporary installation services	44
3.5.2	First Run	44
3.5.3	Second Run	45

3.5.4	Base system installation	45
3.5.5	Ceph deployment	46
3.5.6	Migrate configuration	51
3.5.7	Consul deployment	52
3.5.8	Temporary installation node sweep	53
3.6	Admin node	53
3.6.1	Base system	53
3.6.2	Administration environment	55
3.7	Service virtual machines	55
3.7.1	Libvirt settings	55
3.7.2	Clara configuration	56
3.7.3	Virtual machine definitions	59
3.7.4	Required virtual machines	60
3.8	LDAP Authentication	61
3.8.1	Directory replica	61
3.8.2	Clients setup	61
3.9	Slurm	62
3.9.1	Base Configuration	62
3.9.2	Shared State Location	63
3.9.3	Miscellaneous Tuning	64
3.9.4	MariaDB security hardening	64
3.9.5	Bootstrap	65
3.9.6	Configuration deployment	65
3.10	Frontend and compute nodes	65
3.10.1	Diskless image generation	65
3.10.2	Boot nodes	69
3.11	Optional features	69
3.11.1	Tuning	69
3.11.2	Firewall	69
3.11.3	Kerberos	70
3.11.4	Internal APT repository	70
3.11.5	Storage Multipath	70
3.11.6	Monitoring	70
3.11.7	Metrics	70
3.11.8	HPCStats	70
3.11.9	Slurm WCKeys	70
3.11.10	NFS High-Availability	70
3.11.11	Slurm power management	70
4	Bootstrap procedures	72
4.1	LDAP bootstrap	72
4.2	MariaDB/Galera bootstrap	73
4.3	SlurmDBD bootstrap	73
4.4	Ceph	74
4.4.1	Mon	74
4.4.2	OSD	74
4.4.3	CephFS	74
4.5	NFS HA bootstrap	75
5	Production procedures	76
5.1	MAC address change	76
5.2	Password/keys changes	76
5.2.1	Root password	76
5.2.2	Root SSH key	77
5.2.3	SSH host keys	77
5.2.4	Eyaml keys	77
5.2.5	Internal repository encoding key	78

5.2.6	Replication account password	79
5.2.7	Monitoring certificates	79
5.2.8	Munge key	79
5.2.9	Repo keyring	80
5.2.10	MariaDB users	80
5.3	Administration node re-installation	81
5.4	Service node re-installation	81
5.5	Frontend access	82
5.5.1	Draining	82
5.6	NFS HA	83
5.6.1	Starting a node	83
5.6.2	Manual Fail Over	83
5.7	Services	84
5.7.1	Packages Caching purge	84
5.8	Virtual Machines	84
5.8.1	Deleting a Virtual Machine	84

Chapter 1

About this document

1.1 Purpose

The present document presents the reference architecture, the bootstrap and installation procedures of an HPC system called Scibian HPC.

The main goal is to provide exhaustive information regarding the configuration and system settings based on the needs expressed by users. This information may be useful to business and technical stakeholders, as well as to all members of the scientific computing community at EDF.

1.2 Structure

This document is divided into five chapters:

1. About this document: refers to the present chapter.
2. Reference architecture: gives an overview of the software and hardware architecture of a Scibian HPC system. It also includes a detailed description of the boot sequence of the HPC System and some other advanced topics.
3. Installation procedures: describes how to install the Puppet-HPC software stack used to configure the administration and generic nodes of the HPC system. This chapter also explains how to use Ceph for sharing the configuration files across all the nodes and how to handle the virtual machines providing all the services needed to operate the HPC system.
4. Bootstrap procedures: contains all the procedures to bootstrap all the crucial services for the Scibian HPC system: LDAP, Ceph, MariaDB with Galera, SlurmDBD, etc.
5. Production procedures: contains all the technical procedures to follow for regular operations occurring during the production phase of the supercomputer. This notably includes changing any encryption or authentication key, changing passwords, reinstalling nodes, etc.

1.3 Typographic conventions

The following typographic conventions are used in this document:

- Files or directories names are written in italics: */admin/restricted/config-puppet*.
- Hostnames are written in bold: **genbatch1**.
- Groups of hostnames are written using [the nodeset syntax from clustershell](#). For example, **genbatch[1-2]** refers to the servers **genbatch1** and **genbatch2**.
- Commands, configuration files contents or source code files are set off visually from the surrounding text as shown below:

```
$ cp /etc/default/rcS /tmp
```

1.4 Build dependencies

On a Debian Jessie system, these packages must be installed to build this documentation:

- asciidoctor \geq 0.1.4
- asciidoctor-edf-tpl-latex \geq 2.0
- inkscape
- rubber
- texlive-latex-extra

1.5 License

Copyright © 2014-2017 EDF S.A.

CCN-HPC <dsp-cspito-ccn-hpc@edf.fr>

This document is governed by the CeCILL license under French law and abiding by the rules of distribution of free software. You can use, modify and/ or redistribute the document under the terms of the CeCILL license as circulated by CEA, CNRS and INRIA at the following URL "<http://www.cecill.info>".

As a counterpart to the access to the source code and rights to copy, modify and redistribute granted by the license, users are provided only with a limited warranty and the document's author, the holder of the economic rights, and the successive licensors have only limited liability.

In this respect, the user's attention is drawn to the risks associated with loading, using, modifying and/or developing or reproducing the document by the user in light of its specific status of free software, that may mean that it is complicated to manipulate, and that also therefore means that it is reserved for developers and experienced professionals having in-depth computer knowledge. Users are therefore encouraged to load and test the document's suitability as regards their requirements in conditions enabling the security of their systems and/or data to be ensured and, more generally, to use and operate it in the same conditions as regards security.

The fact that you are presently reading this means that you have had knowledge of the CeCILL license and that you accept its terms.

Full license terms and conditions can be found at http://www.cecill.info/licences/Licence_CeCILL_V2.1-en.html.

1.6 Authors

In alphabetical order:

- Benoit Bocard
- Ana Guerrero López
- Thomas Hamel
- Camille Mange
- Rémi Palancher
- Cécile Yoshikawa

Chapter 2

Reference architecture

This chapter gives an overview of the software and hardware architecture of a Scibian HPC system. It also includes a detailed description of the boot sequence of the HPC System and some other advanced topics.

2.1 Hardware architecture

The following diagram represents the reference high-level hardware architecture of Scibian HPC clusters:

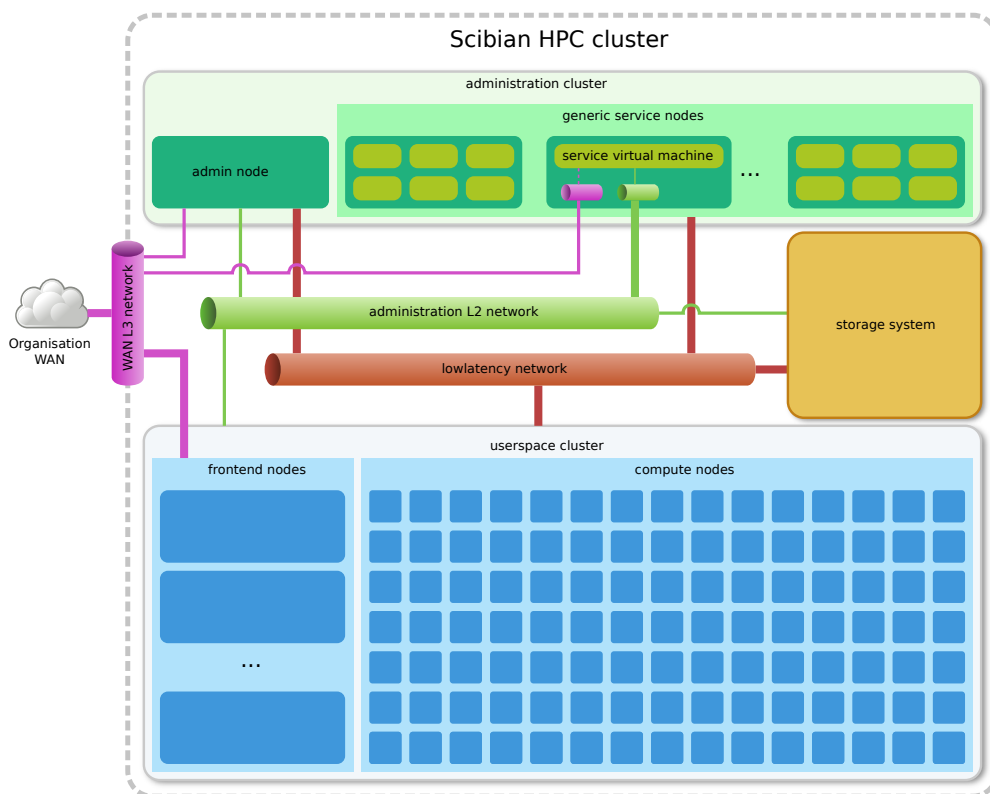


Figure 2.1: Scibian HPC cluster hardware reference architecture

2.1.1 Networks

The cluster is connected to three physically separated networks:

- The **WAN network**, an Ethernet based network with L3 network routers which connect the IP networks of the HPC cluster to the organization network.

- The **low-latency network** for both I/O to the storage system and distributed computing communications (typically MPI messages) between compute nodes. The hardware technologies of this network may vary upon performance requirements but it generally involves high bandwidth (10+GB/s) and low latency technologies such as InfiniBand, Omni-Path or 10GB Ethernet.
- The **administration network** used for basically every other internal network communications: deployment, services, administrator operations, etc. It must be a L2 Ethernet network with dedicated switches.

It is recommended to split the administration Ethernet network with a VLAN dedicated to all management devices (BMC ¹, CMC ², etc). This has significant advantages:

- It significantly reduces the size of Ethernet broadcast domains which notably increases DHCP reliability and drops Ethernet switches load.
- It slightly increases security since the IP access to the management devices can be restricted to nodes accessing the VLAN or by a firewall on an IP router.

2.1.2 Administration cluster

The administration cluster is composed by two types of nodes: the **admin node** and the **generic service nodes**.

The **admin node** is the access node for administrators and the central point of administrative operations. All common administrative actions are performed on this node. It does not run any intensive workloads, just simple short-lived programs and it does not need to be very powerful. It does not store sensible data nor run critical services, so it does not need to be very reliable either. Example of hardware specifications:

CPU	1 × 4 cores
RAM	8GB ECC
Network	<ul style="list-style-type: none"> ▪ 1 × 1GB bonding on administration network ▪ 1 × 1GB bonding on WAN network ▪ 1 link on low-latency network
Storage	2 × 300GB RAID1 SATA hard disk
PSU	Non-redundant

The **generic service nodes** run all critical infrastructure services (within service virtual machines) and manage all production administrative data. Scibian HPC requires a pool from 3 (minimum) to 5 (recommended) generic service nodes. The pool works in active cluster mode, the load is balanced with automatic fail-over. All generic service nodes of a cluster must be fairly identical for efficient load-balancing.

The generic service nodes manage the production data into a distributed object-storage system. It is highly recommended that the nodes have a dedicated block storage device for this purpose. The workload is mostly proportional to the number of compute nodes but the generic service nodes must be quite powerful to comfortably handle load peaks happening during some operations (**ex:** full cluster reboot). Also, since services are run into virtual machines, a fairly large amount of RAM is required. Services can generate a lot of traffic on the administration network, it is relevant to provide a network adapter with high bandwidth. Even though high-availability is ensured at the software level with automatic fail-over between generic service nodes, it is nevertheless recommended to get hardware redundancy on most devices of the generic service nodes to avoid always risky and hazardous service migrations as much as possible. Example of hardware specifications:

CPU	2 × 16 cores
RAM	64GB ECC
Network	<ul style="list-style-type: none"> ▪ 2 × 10GB bonding on administration network ▪ 2 × 1GB bonding on WAN network ▪ 1 link on low-latency network

¹Baseboard Management Card

²Chassis Management Card

Storage	<ul style="list-style-type: none"> ▪ 2 × 300GB RAID1 SATA hard disk for host ▪ 2 × 1TB SSD SAS or NVMe PCIe for object-storage system
PSU	Redundant

All physical nodes must be connected to all three physical networks. There are virtual bridges on the host of the generic service nodes connected to the WAN, administration (and eventually management) networks. The service virtual machines have connections to the virtual bridges upon their hosted service requirements.

2.1.3 User-space cluster

The user-space cluster is composed of **frontend nodes** and **compute nodes**.

The nodes of the user-space cluster are deployed with a diskless live system stored in RAM. It implies that, technically speaking, the nodes do not necessarily need to have local block storage devices.

The **frontend nodes** are the access hosts for users so they must be connected to all three physical networks. It is possible to have multiple frontend nodes in active cluster mode for load-balancing and automatic fail-over. The exact hardware specifications of the frontend nodes mostly depend on user needs and expectations. Users may need to transfer large amount of data to the cluster, it is therefore recommended to provide high-bandwidth network adapters for the WAN network. These nodes can also be designed to compile computational codes and in this case, they must be powerful in terms of CPU, RAM and local storage I/O.

The **compute nodes** run the jobs so they must provide high performances. Their exact hardware specifications totally depend on user needs. They must be connected to both the administration and the low-latency networks.

2.1.4 Storage system

The storage system is designed to host user data. It provides one or several shared POSIX filesystems. The evolved storage technologies depend on user needs ranging from a simple NFS NAS to a complex distributed filesystem such as Lustre or GPFS with many SAN and I/O servers.

2.2 External services

A Scibian HPC cluster is designed to be mainly self contained and to continue running jobs even if it is cut off from the rest of the organization network. There is some limits to this though and some external services are needed. Critical external services are replicated inside the cluster though, to avoid losing availability of the cluster if the connection to external service is cut.

2.2.1 Base services

LDAP

The reference cluster architecture provides a highly available LDAP service, but it is only meant as a replica of an external LDAP service. The organization must provide an LDAP service with suitable replica credentials.

Only the LDAP servers (Proxy virtual machines) connect to these servers.

NTP

The generic service nodes are providing NTP servers for the whole cluster. Those servers must be synchronized on an external NTP source. This could be an organization NTP or a public one (eg. `spool.ntp.org`).

Only the NTP servers (Generic Service nodes) connect to these servers.

Package repositories

The normal way for a Scibian HPC Cluster to handle package repositories (APT) is to provide a proxy cache to organization or public distribution repositories. Alternatively, it is possible to mirror external repositories on the cluster (with `c1ara` and `Ceph/S3`).

Proxy cache needs less maintenance and is the preferred solution. Local mirrors can be used when reliable connection to external repositories is unreliable.

Only the Proxy Cache servers (Generic Service nodes) connect to these servers. In the mirror mode, only the admin node uses them.

DNS

External DNS service is not strictly necessary but is hard to not configure if the cluster must use organization or public services (License servers, NAS...).

The external DNS servers are configured as recursive in the local DNS server configuration.

Only the DNS servers (Generic Service nodes) connect to these servers.

2.2.2 Optional services

NAS

It is frequent to mount (at least on the frontend nodes) an external NAS space to copy data in and out of the cluster.

Graphite

In the reference architecture all system metrics collected on the cluster (by `collectd`) are pushed to an external graphite server. This is usually relayed by the proxy virtual machines.

InfluxDB

In the reference architecture all jobs metrics collected on the cluster are pushed to an external InfluxDB server. This is usually relayed by the proxy virtual machines.

HPCStats

HPCStats is a tool that frequently connects to the frontend as a normal user to launch job. It also connects to the SlurmDBD database to get batch job statistics. The database connection needs a special NAT configuration on the Proxy virtual machines.

Slurm-Web Dashboard

The Slurm-Web Dashboard aggregates data coming from multiple clusters in the same web interface. To get those data, the client connect to an HTTP REST API that is hosted on the Proxy virtual machines.

2.3 Software architecture

2.3.1 Overview

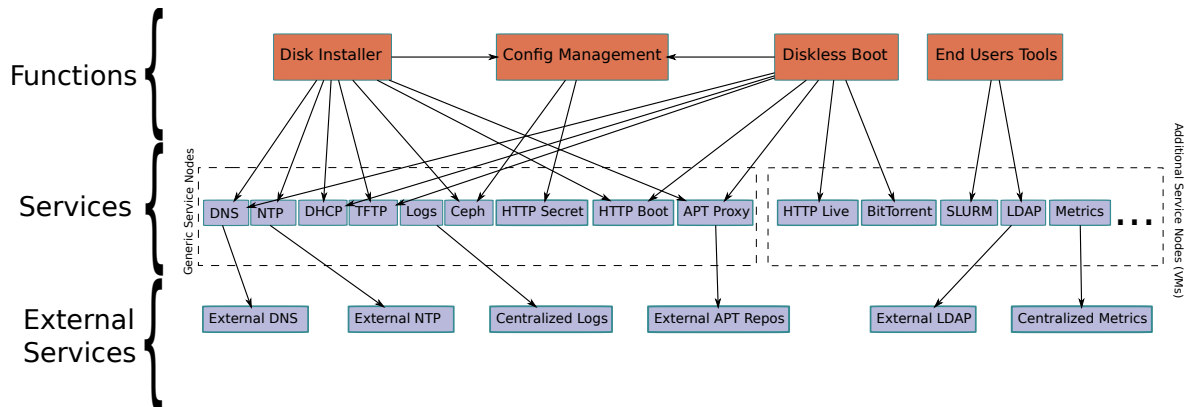


Figure 2.2:

Functions

The software configuration of the cluster aims to deliver a set of functions. Functions can rely on each other, for example, the disk installer uses the configuration management to finish the post-install process.

The main functions provided by a Scibian HPC cluster are:

- **Configuration Management**, to distribute and apply the configuration to the nodes
- **Disk Installer**, to install an OS from scratch on the node disks through the network
- **Diskless Boot**, to boot a node with a live diskless OS through the network
- **Administrator Tools**, tools and services used by the system administrator to operate the cluster
- **User Tools**, tools and services used by end users

The Scibian HPC Cluster will use a set of services to deliver a particular function. If a cluster can provide **Configuration Management** and a **Disk Installer**, it is able to operate even if it cannot do something useful for the users. These two core functions permit to create a self sufficient cluster that will be used to provide other functions.

Services

The software services of the cluster are sorted into two broad categories:

- **Base Services**, necessary to provide core functions: install and configure a physical or virtual machine
- **Additional Services**, to boot a diskless (live) machine, provide all end user services (batch, user directory, licenses...), and system services not mandatory to install a machine (monitoring, metrics...)

The Base Services run on a set of physical machines that are almost identical, those hosts are called **Service Nodes**. The services are setup to work reliably even if some of the service nodes are down. This means that a service node can be re-installed by other active service nodes.

The Additional Services can be installed on a set of other hosts that can be either physical or virtual. VMs (Virtual Machines) are usually used because those services do not need a lot of raw power and the agility provided by virtual machines (like live host migration) are often an advantage.

If the cluster is using virtualized machines for the Additional Services, the service nodes must also provide a consistent virtualization platform (storage and hosts). In the reference architecture, this is provided with Ceph RBD and Libvirt running on service nodes.

A particular service runs on service nodes even if it is not mandatory for Disk Installer or Config Management: the low-latency network manager (Subnet Manager for InfiniBand, Fabric Manager for Intel

Omni-Path). This exception is due to the fact that this particular service needs raw access to the low-latency network.

In the Puppet configuration, services are usually associated with **profiles**. For example, the puppet configuration configures the **DNS Server** service with the profile: `profiles::dns::server`.

2.3.2 Base Services

Infrastructure

Infrastructure-related services provide basic network operations:

- DHCP and TFTP for PXE Boot
- DNS servers, with forwarding for external zones
- NTP servers, synchronized on external servers

These services are configured the same way and running on each service nodes.

Consul

Consul is a service that permits to discover available services in the cluster. Client will query a special DNS entry (`xxx.service.virtual`) and the DNS server integrated with Consul will return the IP address of an available instance.

Ceph

Ceph provides an highly available storage system for all system needs. Ceph has the advantage to work with internal storage on service nodes. It does not require a storage system shared between servers (NAS or SAN).

Ceph provides:

- A Rados Block Device (RBD) that is used to store Virtual Machines disk images
- A Rados GateWay to provide storage for configuration management, Amazon S3 compatible REST API for write operations and plain HTTP for read.
- A Ceph FS that can provide a POSIX filesystem used for Slurm Controller state save location

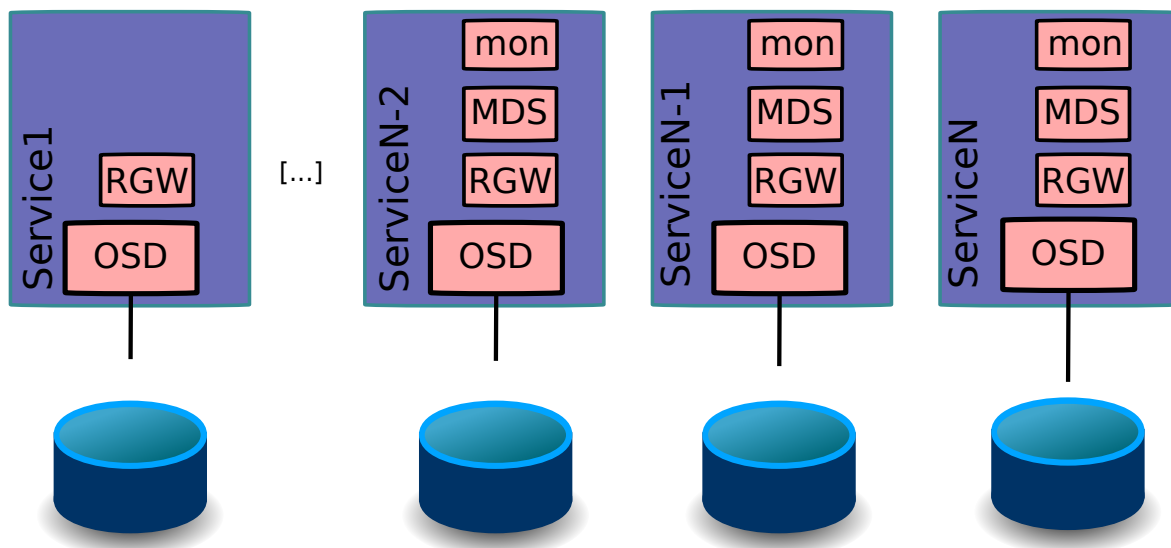


Figure 2.3:

A Ceph cluster is made of four kinds of daemons. All generic service nodes run the following daemons:

- **osd**, Object Storage Daemons actually holding the content of the ceph cluster
- **rgw**, Rados GateWay (sometimes shortened radosgw) exposing an HTTP API like S3 to store and retrieve data in Ceph

Two other kind of service are only available on three of the generic service nodes:

- **mon**, Monitoring nodes, this is the orchestrator of the ceph cluster. A quorum of two active mon nodes must be maintained for the cluster to be available
- **mds**, MetaData Server, only used by CephFS (the POSIX implementation above ceph). At least one must always be active.

With this configuration, any server can be unavailable. As long as at least two servers holding critical services are available, the cluster might survive losing another non-critical server.

Libvirt/KVM

Service nodes are also the physical hosts for the Virtual Machines of the cluster. Libvirt is used in combination with QEMU/KVM to configure the VMs. A Ceph RBD pool is used to store the image of the VMs. With this configuration, the only state on a service node is the VM definition.

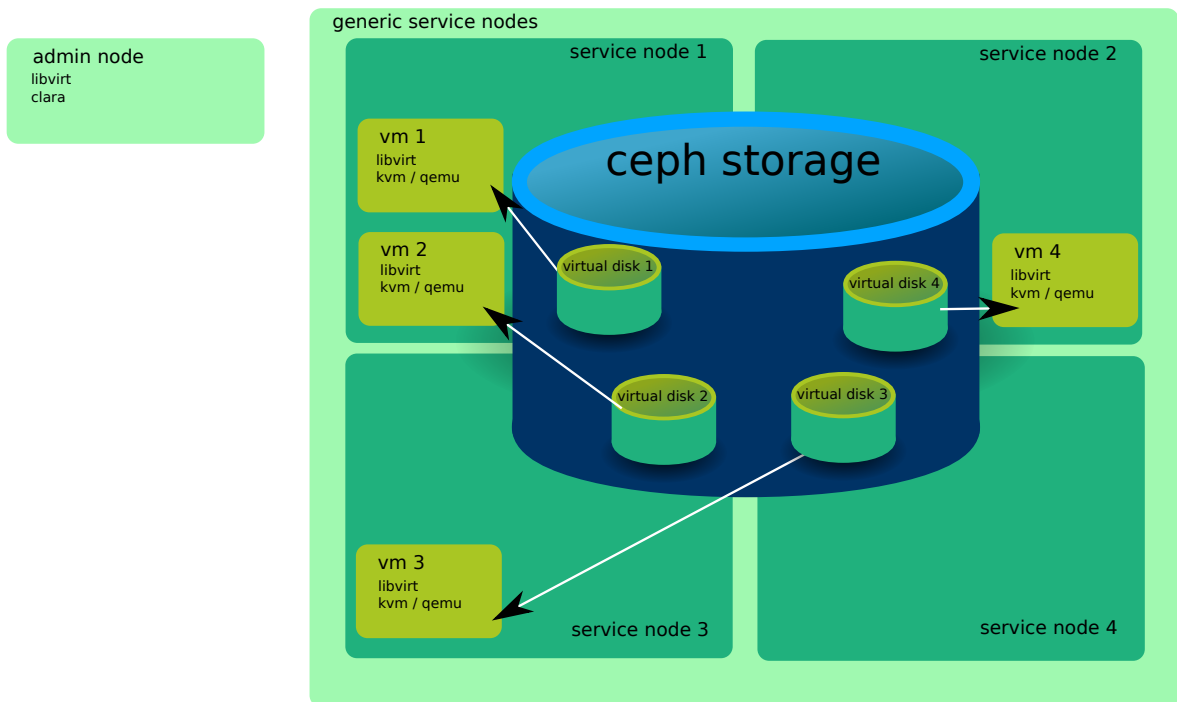


Figure 2.4: How the service machines, ceph and vm interact

Integration with Clara makes it easy to move VMs between nodes.

HTTP secret and boot

The process to boot a node needs a configuration obtained through HTTP and computed by a CGI (in Python). This is hosted on the service nodes and served by Apache. This is also used to serve files like the kernel, initrd and pre-seeded configuration.

A special Virtual Host on the Apache configuration is used to serve secrets (Hiera-Eyaml keys). This VHost is configured to only serve the files on a specific port. This port is only accessible if the client connects from a port below 1024 (is root), this is enforced by a Shorewall rule.

APT proxy

There is no full repository mirror on the cluster. APT is configured to use a proxy that will fetch data from external repositories and cache it. This permits to have always up-to-date packages without overloading external repositories and without having to maintain mirror sync (internally and externally).

Logs

Logs from all nodes are forwarded to a Virtual IP address running on the service nodes. The local rsyslog daemon will centralize those logs and optionally forward the result to an external location.

Low-latency network manager

The Low-latency network manager (InfiniBand Subnet Manager or Intel Omni-Path Fabric Manager) is not mandatory to achieve the feature set of Base Services (Configuration Management and Disk Installation) but it must run on a physical machine, so it is grouped with the Base Services to run on the service nodes.

NFS HA Service

A NFS HA Service can serve two purpose:

- Shared state for servicing using Posix to share their state (like **slurmctld**) when CephFS does not provided sufficient performance
- Shared storage for the users if a distributed file system like GPFS or Lustre is not used (only works for smaller cluster sizes)

The NFS HA Service is provided with a Keepalived setup.

2.3.3 Additional Services

LDAP

There is no standalone LDAP servers configured. The servers are replica from an external directory. This means that both are configured independently and are accessed only for read operations.

If the organization uses Kerberos, all Kerberos requests and password checks are done directly by the external Kerberos server.

Bittorrent

Diskless image files are downloaded by the nodes with the BitTorrent protocol. The cluster provides a redundant tracker service with OpenTracker and two server machines are configured to always seed the images.

An Apache server is used to serve the torrent files for the diskless images (HTTP Live).

Slurm

Slurm provides the job management service for the cluster. The controller service (SlurmCtld) runs in an Active/Passive configuration on a pair of servers (**batch** nodes). The state is shared between the controller nodes. This can be achieved with a CephFS mount or with an NFS HA server. CephFS does not permit to support a large number (thousands) of jobs yet.

The SlurmDBD service also runs on these two servers.

MariaDB/Galera

SlurmDBD uses a MySQL like database to store accounting information and limits. On Scibian HPC Clusters this is provided by a MariaDB/Galera cluster which provides an Active/Active SQL server compatible with MySQL.

This cluster is usually co-located with SlurmDBD service and Slurm Controllers (**batch** nodes).

Relays

The Additional Services include a set of relay services to the outside of the cluster for:

- Email (Postfix Relay)
- Network (NAT configured by Shorewall)
- Metrics (Carbon C Relay)

Monitoring

Cluster monitoring is done by Icinga2, the cluster is integrated inside an organization Icinga infrastructure. The cluster hosts a redundant pair of monitoring satellites that checks the nodes. The monitoring master

is external to the cluster.

2.3.4 High-Availability

All services running on the cluster should be highly available (HA). Some services not critical for normal cluster operation can be not highly available, but this should be avoided if possible.

The following section lists the different techniques used to achieve high-availability of the cluster services.

Stateless

Stateless services are configured the same way on all servers and will give the same answer to all requests. These services include:

- **dhcp**
- **tftp**
- **ntp**
- **dns**
- **LDAP Replica**
- **HTTP Secret**
- **HTTP Boot**
- **HTTP Live**
- **Ceph RadosGW**
- **APT Proxy**
- **Carbon Relay**
- **Bittorrent Tracker**
- **Bittorrent Seeder**
- **SMTP Relay**

Clients can provide a list of potential servers that will be tried in turn. If the client do not automatically accept multiple servers, it is possible to use the Consul service to get a DNS entry ("xxx.service.virtual") that will always point to an available instance of the service.

As a last resort and for services that do not need Active/Active (Load Balancing) capabilities, it is possible to use a Virtual IP address (VIP). **HTTP Live** and **Carbon Relay** uses this technique.

Native Active/Active

Some services have native internal mechanisms to share states between the servers. Contacting any server will have the same effect on the state of the service, or the service has an internal mechanism to get the right server. These services behave this way:

- **Ceph Rados**
- **MariaDB/Galera**
- **consul**

Native Active/Passive

Services that have only one active server at any time, but the mechanism to select the active server is internal to the service. This means all servers are launched in the same way and not by an external agent like Keepalived or Pacemaker/Corosync. Services using this technique are:

- **Ceph MDS** (Posix CephFS server)
- **Slurm Controller**

- **Omni-Path Fabric Manager** or **InfiniBand Subnet Manager**

Controlled Active/Passive

The service can only have one active server at any one time and this failover must be controlled by an external service. On the current configuration the only service requiring this setup is:

- **NFS HA Server**

2.4 Conventions

In order to restrain the complexity of the configuration of a Scibian HPC cluster, some naming and architecture conventions have been defined. Multiple components of the software stack expect these conventions to be followed in order to operate properly. These conventions are actually rather close to HPC cluster standards, then they should not seem very constraining.

- The operating system short hostname of the nodes must have the following format: <prefix><role><id>. This is required by the association logic used in Puppet-HPC to map a node to its unique Puppet role. This point is fully explained in the role section of Puppet-HPC reference documentation.
- The FQDN³ hostnames of the nodes must be similar to their network names on the administration network. In other words, the IP address resolution on the cluster of the FQDN hostname of a node must return the IP address of this node on the administration network.

2.5 Advanced Topics

2.5.1 Boot sequence

Initial common steps

The servers of the cluster can boot on their hard disks or via the network, using the PXE protocol. In normal operations, all service nodes are installed on hard disks, and all nodes of the userspace (**compute** and **frontend** nodes) use the network method to boot the diskless image. A service node can use the PXE method when it is being installed. The boot sequence between the `power on` event on the node and the boot of the `initrd` is identical regardless of the system booted (installer or diskless image).

The steps of the boot sequence are described on the diagram below:

³Fully-Qualified Domain Name

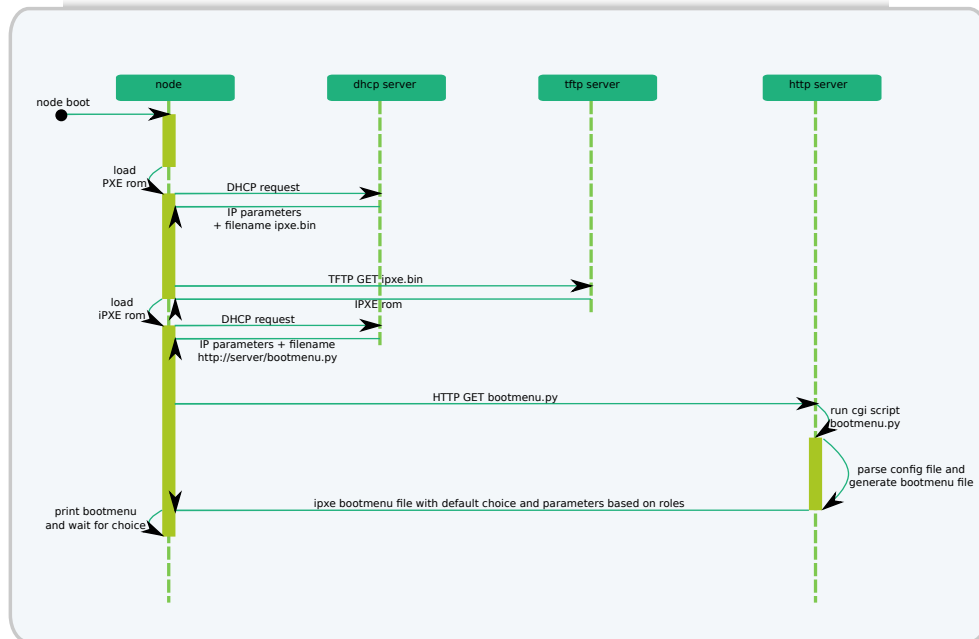


Figure 2.5:

When a node boots on its network device, after a few (but generally time-consuming) internal checks, it loads and runs the PXE ROM stored inside the Ethernet adapter. This ROM first sends a DHCP request to get an IP address and other network parameters. The DHCP server gives it an IP address alongside the filename parameter. This filename is the file the PXE ROM downloads using the TFTP protocol. This protocol, which is rather limited and unreliable is used here because the PXE ROM commonly available in Ethernet adapters only supports this network protocol.

The file to download depends on the type of nodes or roles. On Scibian HPC clusters when using the Puppet-HPC software stack, the required filename for the current node is set in the profile `bootsystem` and therefore its value is usually specified in Hiera in the `profiles::bootsystem::boot_params` hash. It is set to launch the open source iPXE software, because it delivers many powerful features such as HTTP protocol support. This way, it is used as a workaround to hardware PXE ROM limitations.

The virtual machines boot like any other node, except QEMU uses iPXE as the PXE implementation for its virtual network adapters. This means that the virtual machines go directly to this step.

The iPXE bootloader must perform another DHCP request since the IP settings are lost when the bootloader is loaded. The DHCP server is able to recognize this request originates from an iPXE ROM. In this case, it sets the filename parameter with an HTTP URL to a CGI written in Python: `bootmenu.py`. If the DHCP server already knows the originating node and its MAC address (with a statically assigned IP address), it also sends the hostname in the answer. Otherwise, the DHCP request is not honored.

Then, the iPXE bootloader sends the GET HTTP request to this URL. In this request, it also adds to the parameters its hostname as it was given by the DHCP server.

On the HTTP server side, the Python script `bootmenu.py` is run as a CGI program. This script parses its configuration file `/etc/hpc-config/bootmenu.yaml` to get the parameters to properly boot the node: serial console and Ethernet device to use, default boot mode (diskless, installer, etc ..). Then it generates an iPXE profile with a menu containing all possible boot entries. Finally, a timeout parameter is added to the iPXE profile.

The iPXE bootloader downloads and loads this dynamically generated profile. Without any action from the administrator, iPXE waits the timeout and loads the default entry set by the Python script.

Note

If one of the following conditions is satisfied: either the hostname parameter is empty, or the node could not be found in the `/etc/hpc-config/bootmenu.yaml` file, then, the default choices from the config file are used.

Disk installation

Here is the sequence diagram of a Scibian server installation on disk, right after the PXE boot common steps:

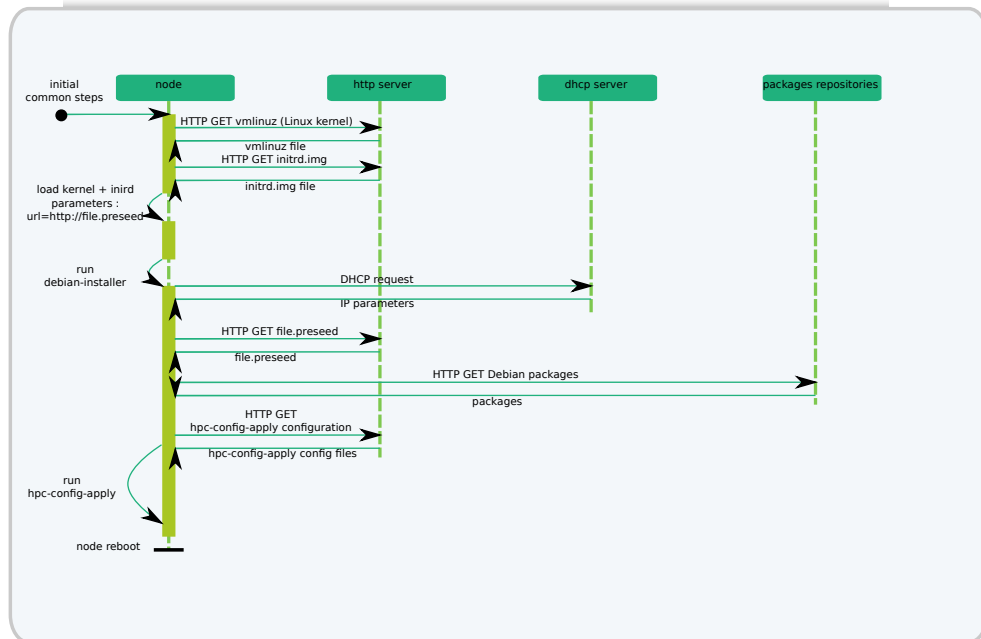


Figure 2.6:

The iPXE ROM downloads the Linux kernel and the initrd archive associated with the boot menu entry. The kernel is then run with all the parameters given in the menu entry, notably with the HTTP url to the preseed file.

The initrd archive contains the Debian Installer program. This program starts by sending a new DHCP request to get an IP address. Then, it downloads the preseed file located at the URL found in the 'url' kernel parameter. This preseed file contains all the answers to the questions asked by the Debian Installer program. This way, the installation process is totally automated and does not require any interaction from the administrator.

During the installation, many Debian packages are retrieved from Debian repositories.

At the end of the installation, Debian Installer runs the commands set in the `late_command` parameter of the preseed file. On Scibian HPC clusters, this parameter is used to run the following steps:

- Download through HTTP the `hpc-config-apply` script,
- Run `hpc-config-apply` inside the chroot environment of the newly installed system.

Detailed functioning of the `hpc-config-apply` script is not described here, but it involves:

- downloading and installing additional Debian packages depending on the node role,
- executing various types of software
- and writing various configuration files on the installed system.

Please refer to [hpc-config-apply\(1\) man page](#) for a full documentation on how to use this script.

Finally, when the execution of the commands are over, the server reboots.

Once the servers are installed, they are configured through IPMI with Clara to boot on their disk devices first. Please refer to Clara documentation for further details.

Diskless boot

Here is the sequence diagram of the boot process for diskless nodes, right after the PXE boot common steps:

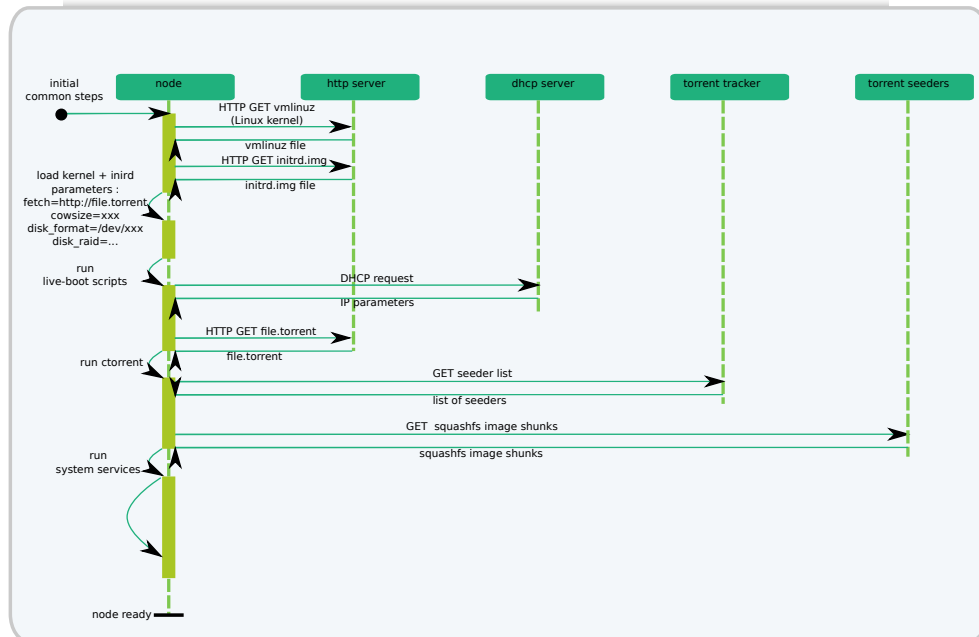


Figure 2.7:

The iPXE bootloader downloads the Linux kernel and the initrd image defined within the default boot menu entry and runs them with the provided parameters. Among these parameters, there are notably:

- `fetch` whose value is an HTTP URL to a torrent file available on the HTTP server of the supercomputer,
- `cowsize` whose value is the size of the ramfs filesystem mounted on `/lib/live/mount/overlay`,
- `disk_format` if this parameter is present the device indicated is formatted on node boot,
- `disk_raid` if this parameter is present a software raid is created with the parameters indicated on node boot.

Within the initrd images, there are several specific scripts that come from `live-boot`, `live-torrent` and specific Scibian Debian packages. Please refer to the following sub-section [Advanced Topics, Generating diskless initrd](#) for all explanations about how these scripts have been added to the initramfs image.

These scripts download the torrent file at the URL specified in the `fetch` parameter, then they launch the `ctorrent` BitTorrent client. This client extracts from the torrent file the IP address of the BitTorrent trackers and the names of the files to download using the BitTorrent protocol. There is actually one file to download, the SquashFS image, that the client will download in P2P mode by gathering small chunks on several other nodes. Then, once the file has been fully retrieved, the image is mounted after executing some preliminary tasks like formatting the disk or setting up a raid array if it has been indicated in the kernel options passed by the boot menu. Then, the real init system is started and it launches all the system services. One of these services is `hpc-config-apply.service` which runs the `hpc-config-apply` script.

As for the part regarding the installation with a disk, how the `hpc-config-apply` script works is not described here. Please refer to [hpc-config-apply\(1\) man page](#) for a full documentation on this topic.

Finally, the node is ready for production.

2.5.2 Frontend nodes: SSH load-balancing and high-availability

The frontend nodes offer a virtual IP address on the WAN network that features both an highly-available and load-balanced SSH service for users to access the HPC cluster. The load-balancing feature automatically distributes users on all available frontend nodes. This load-balancing is operated with persistence so that users (based on their source IP address) are always redirected to the same frontend node in a time frame. Behind the virtual IP address, the high-availability of the SSH service is also ensured in case of outage on a frontend node. These load-balancing and high-availability features are ensured by the Keepalived software.

For security reasons, a firewall is also set up on the frontend nodes to control outgoing network traffic. This firewall service is managed by Shorewall, a high-level configuration tool for Linux netfilter. Because of all the various network flows involved in Keepalived, it must be tightly integrated with the firewall rules. The following diagram illustrates both the network principles behind the high-availability/load-balancing mechanisms and the integration with the software components of the firewall:

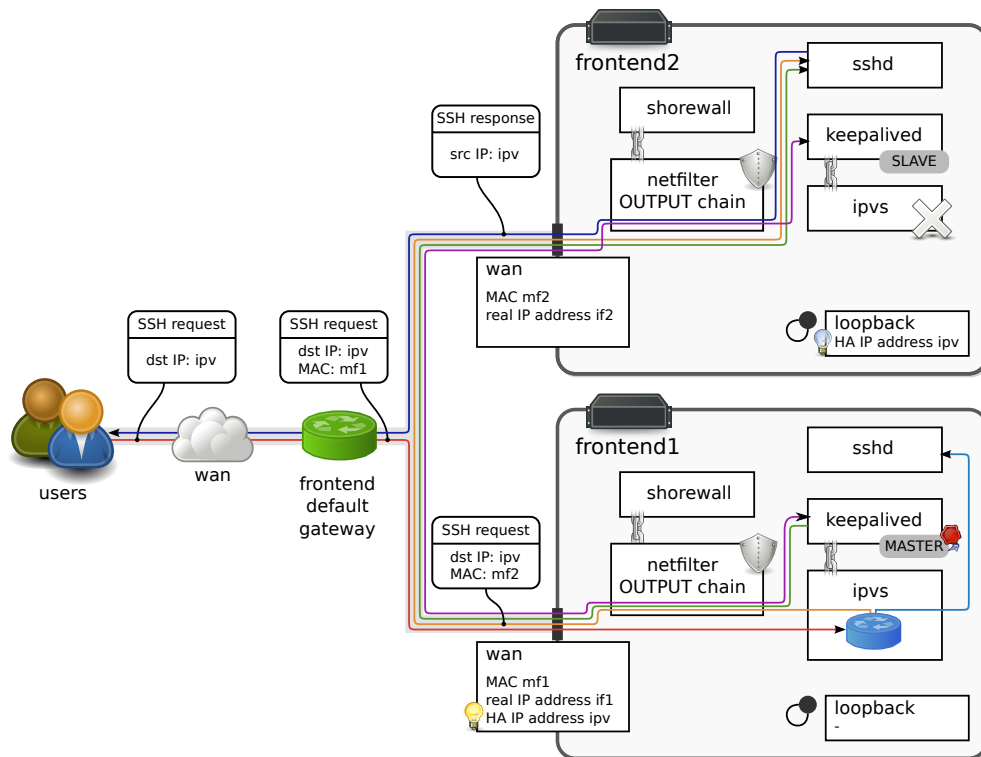


Figure 2.8: sshd load-balancing HA mechanism with firewall integration

The Keepalived software checks all the frontend nodes using the VRRP⁴ protocol on the WAN network interfaces (purple arrow in the diagram). This protocol must be allowed in the OUTPUT chain of the firewall so that Keepalived can work properly.

On the master frontend node, the HA virtual IP address is set on the network interface attached to the WAN network. The Keepalived software configures the IPVS⁵ Linux kernel load-balancer to redirect new TCP connections with a Round-Robin algorithm. Therefore, a part of the TCP connections is redirected to the `sshd` daemon of other frontend nodes (orange arrow in the diagram). An exception must be specified in the OUTPUT chain of the firewall to allow these redirected connections.

To perform such redirections, IPVS simply changes the destination MAC address, to set the address of the

⁴Virtual Router Redundancy Protocol

⁵IP Virtual Server

real destination frontend, in the Ethernet layer of the first packet of the TCP connection. However, the destination IP address does not change: it is still the virtual IP address.

On the slave frontend nodes, the HA virtual IP address is set on the loopback interface. This is required to make the kernel accept the redirected packets from the master frontend node addressed to the virtual IP address. In order to avoid endless loops, the IPVS redirection rules are disabled on slave frontend nodes or else, packets would be redirected endlessly.

By default, the Linux kernel answers the ARP requests coming from any network device for any IP address attached to any network device. For example, on a system with two network devices: `eth0` with `ip0` and `eth1` with `ip1`, if an ARP request is received for `ip1` on `eth0`, the kernel positively responds to it, with the MAC address of `eth0`. Though it is convenient in many cases, this feature is annoying on the frontend nodes, since the virtual IP address is set on all of them. Consequently all frontend nodes answer the ARP requests coming from the WAN default gateway. In order to avoid this behaviour, the `net.ipv4.conf.<netif>.arp_ignore` and `net.ipv4.conf.<netif>.arp_announce` sysctl Linux kernel parameters, where `<netif>` is the network interface connected to the WAN network, are respectively set to 1 and 2. Please refer to the Linux documentation for more details on these parameters and their values: <http://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt>

The Keepalived software also checks periodically if the `sshd` service is still available on all frontend nodes by trying to perform a TCP connection to their real IP addresses on the TCP/22 port (green arrow in the diagram). An exception must be present in the OUTPUT chain of the firewall to allow these connections.

There is an unexplained behaviour in the Linux kernel where the Netfilter `conntrack` module considers that new TCP connections redirected by IPVS to the local `sshd` daemon have an invalid `cstate`. This point can be verified with well placed iptable rules using the LOG destination. This causes the TCP SYN/ACK answer from the `sshd` to be blocked by the OUTPUT chain since it considers the connection is new and not related to any incoming connections. To workaround this annoying behaviour, an exception has been added in the OUTPUT chain of the firewall to accept connections with a source port that is TCP/22 and a source IP address that is the virtual IP address. This is not totally satisfying in terms of security but there is no known easy or obvious way to exploit this security exception from a user perspective for other purposes.

If a slave frontend node becomes unavailable, Keepalived detects it either with VRRP checks, or with TCP checks in case only the `sshd` daemon is crashed. The IPVS rules are changed dynamically to avoid redirecting new TCP connections to this failing node.

If the master frontend node becomes unavailable, the Keepalived software selects a new master node within the other frontend nodes. Then, on this new master node, Keepalived restores the IPVS redirection rules (since they were previously disabled to avoid loops) and moves the virtual IP address from the loopback interface to the WAN network interface.

If a frontend node is scheduled to be turned off, it is possible to drain it.

2.5.3 Service nodes: DNS load-balancing and high-availability

This diagram gives an overview of the load-balancing and high-availability mechanisms involved in the DNS service of the Scibian HPC clusters:

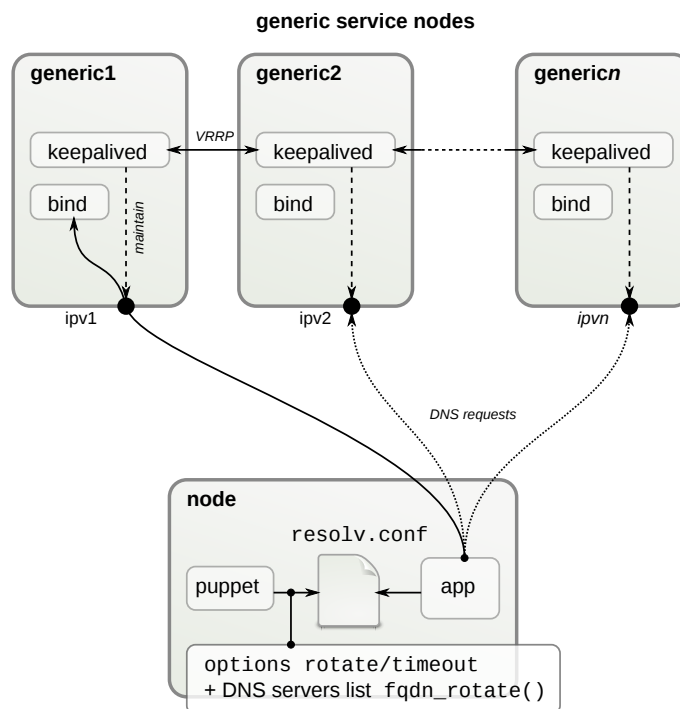


Figure 2.9: DNS service load-balancing and high-availability

On Linux systems, when an application needs to resolve a network hostname, it usually calls the `gethostbyname*()` and `getaddrinfo()` functions of the `libc`. With a common configuration of the Name Service Switch (in the file `/etc/nsswitch.conf`), the `libc` searches for the IP address in the file `/etc/hosts` and then fallbacks to a DNS resolution. The DNS solver gathers the IP address by sending a request to the DNS nameservers specified in the file `/etc/resolv.conf`. If this file contains multiple nameservers, the solver sends the request to the first nameserver. If it does not get the answer before the timeout, it sends the request to the second nameserver, and so on. If the application needs another DNS resolution, the solver will follow the same logic, always trying the first nameserver in priority. It implies that, with this default configuration, as long as the first nameserver answers the requests before the timeout, the other nameservers are never requested and the load is not balanced.

This behavior can be slightly altered with additional options in the file `/etc/resolv.conf`

- `options rotate`: this option tells the `libc` DNS solver to send requests to all the nameservers for successive DNS requests of a process. The DNS solver is stateless and loaded locally for the processes as a library, either as a shared library or statically in the binary. Therefore, the rotation status is local to a process. The first DNS request of a process will always be sent to the first nameserver. The rotation only starts with the second DNS request of a process. Notably, this means that a program which sends one DNS request during its lifetime, launched n times, will send n DNS requests to the first nameserver only. While useful for programs with long lifetime, this option can not be considered as an efficient and sufficient load-balancing technique.
- `options timeout:1`: this option reduces the request timeout from the default value i.e. 60 seconds to 1 second. This is useful when a nameserver has an outage since many processes are literally stuck waiting for this timeout when it occurs. This causes many latency issues. With this option, the `libc` DNS solver quickly tries the other nameservers and the side-effects of the outage are significantly reduced.

On Scibian HPC clusters, Puppet manages the file `/etc/resolv.conf` and ensures these two options are present. It also randomizes the list of nameservers with the `fqdn_rotate()` function of the Puppet `stdlib` community module. This function randomizes the order of the elements of an array but uses the `fqdn` fact to ensure the order stays the same for a node with a given FQDN. That is, each node will get a different random rotation from this function, but a given node's result will be the same every time unless its hostname changes. This prevents the file content from changing with every Puppet runs. With this function, all the DNS nameservers are equivalently balanced on the nodes. Combined with the `options rotate`, it forms an efficient load-balancing mechanism.

The DNS servers are managed with the `bind` daemon on the generic service nodes. Each generic service node has a virtual IP address managed by a `keepalived` daemon and balanced between all the generic service nodes. The IP addresses of the nameservers mentioned in the file `/etc/resolv.conf` on the nodes are these virtual IP addresses. If a generic service node fails, its virtual IP address is automatically routed to another generic service node. In combination with `options timeout:1`, this constitutes a reliable failover mechanism and ensures the high-availability of the DNS service.

2.5.4 Consul and DNS integration

This diagram illustrates how Consul and the DNS servers integrate to provide load-balanced and horizontally scaled network services with high-availability:

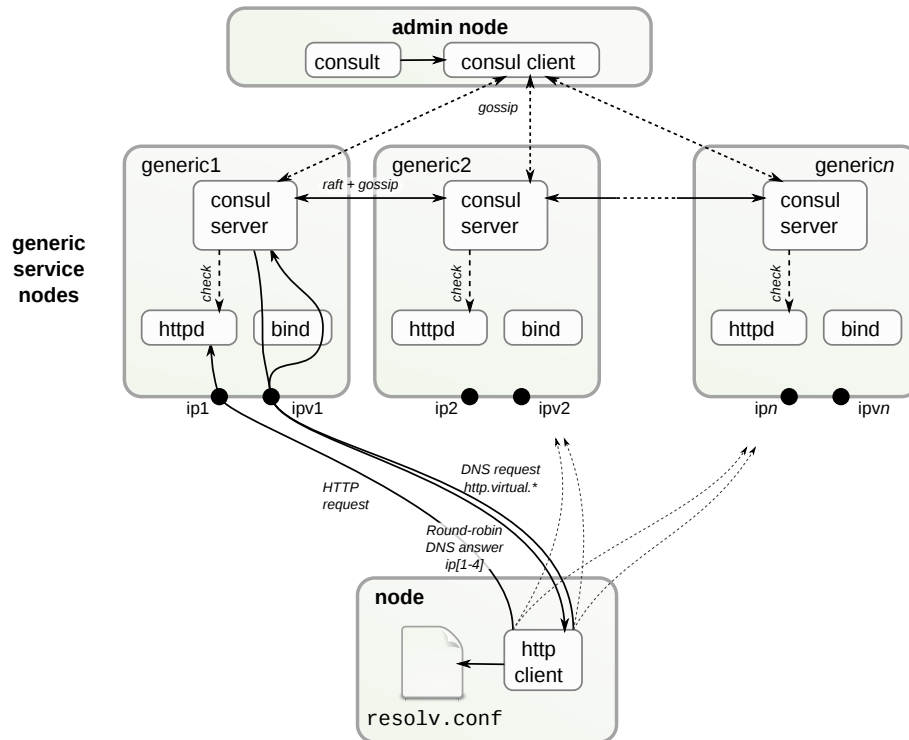


Figure 2.10: Consul, DNS server and services integration

The Consul agent daemon can run in two modes: server and client. The cluster of Consul servers maintains the state of the cluster using the raft protocol. The clients communicate with the servers to detect failures using the gossip protocol. Both agents expose the data of the Consul cluster through a HTTP REST API. On Scibian HPC clusters, the Consul servers run on the generic service nodes while the admin node runs a client agent.

As explained in the [Software architecture](#) section, Consul *discovers* network services on a pool of nodes. The services discovered by Consul on Scibian HPC clusters are hosted on the generic service nodes. Each Consul server is responsible for checking its locally running services, such as an HTTP server for example. The state being constantly shared by all Consul agents, every agent is actually able to tell where the services are available. Consul notably provides a DNS interface. Given a particular virtual hostname referring to a service, Consul can give the IP addresses of the servers currently running this service.

Consul is not designed to operate as a full DNS server. It listens for incoming requests on an alternative UDP port for a particular sub-domain `virtual.<domain>`, where `<domain>` is configurable and depends on the cluster.

On the nodes, the clients are configured to connect to services in this particular sub-domain, for example `http.virtual.<domain>` for the HTTP service. The DNS requests sent by the clients are received by the `bind` daemon through the virtual IP addresses of the generic service nodes, as explained in [DNS Load-balancing and High-availability](#) section. The DNS `bind` daemon is configured to forward the requests on

the virtual sub-domain to the local Consul agent. The Consul agent answers the DNS request with the static IP address of the generic service nodes running this service, in random order.

In this architecture, both the DNS requests to the Consul servers and the services (eg. HTTP) requests are load-balanced on all the generic service nodes in high-availability mode. The same mechanism also applies to APT proxies, Ceph RADOS gateways, and so on.

The `Consult` utility is installed on the admin node to request the current status of the Consul cluster. It connects to the REST API of the Consul client running locally and prints the status on the standard output.

2.5.5 Scibian diskless initrd

The `scibian-diskless-initramfs-config` package

This package contains the necessary configuration in order to build an initramfs disk suitable for Scibian diskless nodes. It depends on the following packages:

- `initramfs-tools`
- `live-torrent`
- `live-boot`
- `mdadm`
- `parted`

initramfs-tools `initramfs-tools` is a Debian package that provides tools to create a bootable initramfs for Linux kernel packages. The initramfs is a compressed cpio archive. At boot time, the kernel unpacks that archive into RAM, mounts and uses it as the initial root file system. The mounting of the real root file system occurs in early user space.

live-boot The `live-boot` package provides support for live systems. It depends on the `live-boot-initramfs-config` package, which is a backend for `live-boot` in `initramfs` config. In particular, it provides the "live" script in `/usr/share/initramfs-tools/scripts/live`. This script is copied in the generated initramfs and can download and unpack live system images used as the root filesystem for diskless nodes.

live-torrent The `live-torrent` package provides support for BitTorrent downloading for live systems. It depends on the `live-torrent-initramfs-tools` package, which provides the `ctorrent` binary (a bittorrent client) in the initramfs.

Generating the initramfs

With the packages described above installed on a Scibian system, it is possible to generate an initramfs able to download the root live system image via the BitTorrent protocol.

On a Scibian HPC cluster, it is recommended to use the `Clara` tool to generate the root live system image, and to generate the corresponding initramfs. It is possible to specify in the Clara configuration file which packages are mandatory in the image before generating the initramfs.

Here is an example of the "images" section of the Clara configuration file:

```
[images]
files_to_remove=/etc/udev/rules.d/70-persistent-net.rules,/root/.bash_history,/etc/hostname
etc_hosts=10.0.0.1:service,10.0.0.2:admin1
extra_packages_image=hpc-config-apply,scibian-hpc-compute
packages_initrd=scibian-diskless-initramfs-config
```

With this configuration, Clara follows these steps to generate the initramfs:

1. Uncompress the squashfs image

2. Chroot in the directory created
3. Install the packages defined by the `packages_initrd` key in the Clara config file
4. Generate the `initramfs`
5. Do not re-compress the `squashfs` image

This method is used to guarantee consistency with the kernel in the `squashfs` image. It is also possible to generate an `initramfs` for an image based on Scibian9 with a machine installed on Scibian8, for example.

Chapter 3

Installation procedure

This chapter describes how to install the Scibian HPC cluster software stack on a hardware infrastructure compliant with the *reference architecture*. The first section gives a quick overview of the main steps of the installation process. There are few requirements before starting the installation, they are listed in the following sections. Then, the successive steps are described in details. Finally, the chapter ends with the installation documentation of various optional features.

3.1 Overview

The installation process of a Scibian HPC cluster starts with the administration cluster of the *reference architecture*. The administration cluster is composed of the admin node and a pool of generic services nodes. The generic services nodes run the base services required by all nodes, then they are the entry point of the installation procedure.

The first generic service node takes the role of the temporary installation in order to install all the other generic service nodes. When the generic services nodes are fully operational with the base software services stack, the admin node is installed. Then, the process continues with the services virtual machines and the set of additional services are installed.

Finally, the frontend and compute nodes of the userspace cluster are deployed and all the additional services are setup to make the Scibian HPC cluster fully operational.

3.2 Requirements

There are a few requirements before starting up the cluster installation. This section aims to inventory all of these requirements, with example values.

Note

For the sake of simplicity, the examples values are used all along the rest of the installation procedure documentation in various commands or code excerpts. These examples values must be replaced with values corresponding to your environment where appropriate.

Description	Example
Cluster name	foobar or \$CLUSTER
Cluster prefix	fb
Network domain name	hpc.example.org or \$NETDOMAIN
Remote Git internal configuration repository (cf. following note)	ssh://forge/hpc-privatedata
DNS servers	<ul style="list-style-type: none">1.1.1.12.2.2.2
NTP servers	<ul style="list-style-type: none">ntp1.example.orgntp2.example.org

SMTP servers	smtp.example.org
LDAP server	ldap.example.org
Groups of users in LDAP directory	<ul style="list-style-type: none"> ▪ grpusers1 ▪ grpusers2
IP networks and addressing plan	4 IP networks: <ul style="list-style-type: none"> ▪ <i>administration</i>: 10.1.0.0/24 ▪ <i>management</i>: 10.2.0.0/24 ▪ <i>wan</i>: 10.3.0.0/24 ▪ <i>lowlatency</i>: 10.4.0.0/24
All MAC addresses	
Network interfaces configuration of all the nodes and equipments	Please refer to the following diagram for an example of generic service network configuration.
Local block storage configuration of all the nodes	For generic services nodes: * sda for system * sdb for Ceph

Note

The deployment of Scibian HPC cluster is mainly based on [Puppet-HPC](#). As explained in the *Software Architecture* chapter of Puppet-HPC documentation, it works in combination with an internal configuration repository containing all configuration settings and data specific to your organization. This Git repository does not have to be populated to proceed the installation. If it empty, the Internal repository section of this chapter explains how to initialize it from scratch for Puppet-HPC.

This diagram represents an exemple network interfaces configuration for the generic services nodes of a Scibian HPC cluster:

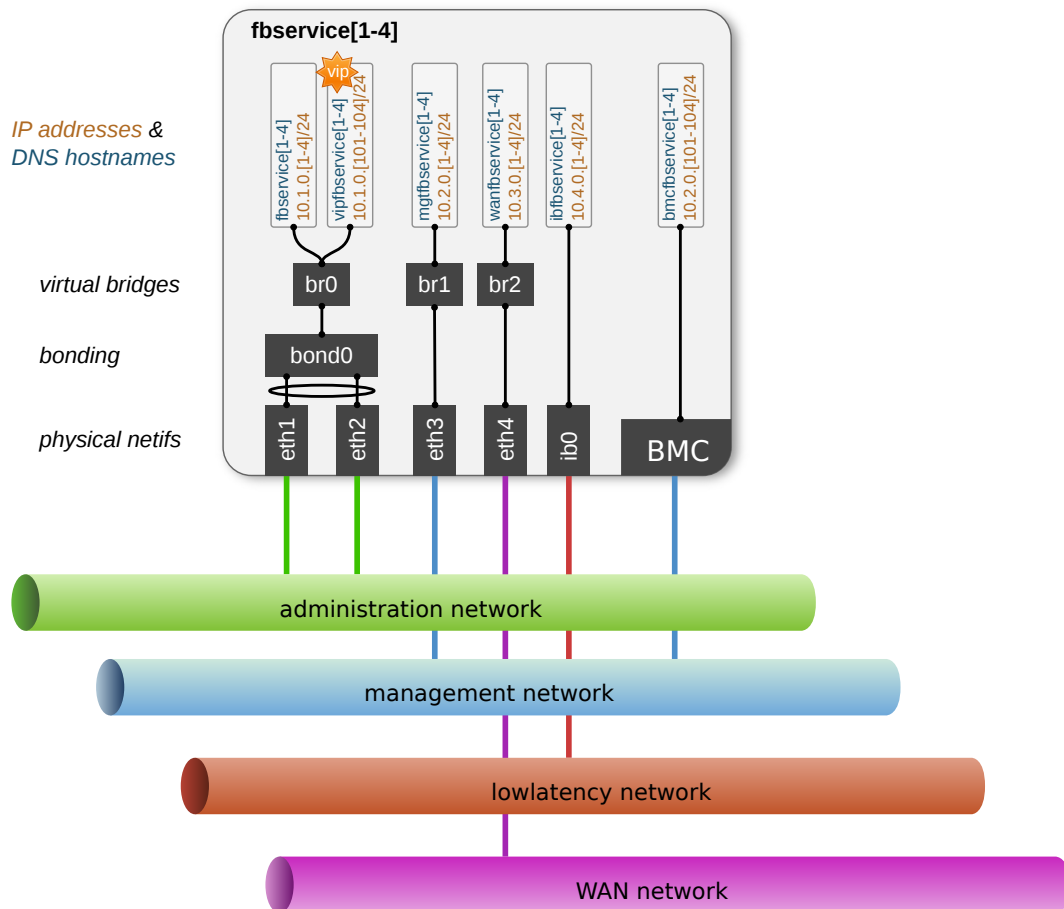


Figure 3.1: Example generic service nodes network interfaces

3.3 Temporary installation node

The first step of the installation process is to install the first generic service node. This node will ensure the role of temporary installation node for the other generic service nodes. Before the admin node is installed, all operations (unless explicitly stated) are realized on this temporary installation node.

3.3.1 Base installation

Install Debian 8 Jessie base system using any of the official Debian installation media (CD, DVD, USB key, PXE server, etc) at your convenience. Configure the network interfaces with static IP addresses in compliancy with the cluster IP addressing plan. Set the hostname following the architecture conventions, for example: fbserve1.

Once the node has rebooted on freshly installed system, add the Scibian 8 APT repositories to the configuration:

```
# echo <<EOF >/etc/apt/sources.list.d/scibian8.list
deb http://scibian.org/repo/ scibian8 main
EOF
```

Download and enable Scibian repository keyring:

```
# wget https://scibian.org/repo/scibian-auto-keyring.pub -O- | apt-key add -
```

Update the packages repositories local database:

```
# apt-get update
```

Install the following Scibian HPC administration node meta-package:

```
# apt-get install scibian-hpc-admin
```

3.3.2 Administration environment

All the files manipulated during the installation process will be placed into a dedicated working directory. The location of this directory is arbitrary, for example: `root/install`. This directory will be designated as `$ADMIN` in the following section of the installation procedure documentation.

```
# export ADMIN=~root/install
# mkdir $ADMIN && cd $ADMIN
```

Clone both Puppet-HPC and internal configuration repositories into this dedicated working directory:

```
# git clone https://github.com/edf-hpc/puppet-hpc.git
# git clone ssh://forge/hpc-privatedata.git
```

At this stage, the internal repository can be populated with all files and data initially required to install the cluster.

3.4 Internal configuration repository

The internal configuration repository required by Puppet-HPC is designed to be shared upon multiple clusters in an organization. Then, it has to be initialized only for the first cluster installation. Its structure and content is fully explained in the *Software Architecture* chapter of Puppet-HPC documentation.

This section provides examples configurations snippets to quickstart this internal configuration repository from scratch.

3.4.1 Base directories

If the internal configuration repository is fully empty and is initialized from scratch, a few base directories must be created under its root.

Important

This step must not be realized if the internal configuration repository is not empty, typically if it has already been initialized for another cluster.

```
# cd $ADMIN/hpc-privatedata
# mkdir files hieradata puppet-config
```

3.4.2 Organization settings

Some settings are common to all HPC clusters of an organization, in particular settings regarding the external services. To avoid duplication of these settings in all HPC cluster configurations, they are defined once in the organization layer of the hiera repository shared by all HPC clusters.

Important

This step must be done only once for the organization. It can be skipped safely if the organization layer YAML file already exists.

Initialize the file `$ADMIN/hpc-privatedata/hieradata/org.yaml` with the following content:

```
##### Common #####

org: 'company' # lower-case name of the organization

##### DNS #####

domain: "%{hiera('cluster_name')}.hpc.example.org"

profiles::dns::client::search: "%{hiera('domain')} hpc.example.org"

profiles::dns::server::config_options:
  forwarders:
    - '1.1.1.1'
    - '2.2.2.2'

##### NTP #####

profiles::ntp::server::site_servers:
  - "ntp1.example.org"
  - "ntp2.example.org"

##### APT #####

profiles::cluster::apt_sources:
  'jessie':
    location: "http://%{hiera('debian_mirror_server')}/%{hiera('debian_mirror_dir')}"
    release: 'jessie'
    repos: 'main contrib non-free'
    required_packages: 'debian-archive-keyring'
    pin: '500'
    include_src: false
    architecture: 'amd64,i386'
  'jessie-updates':
    location: "http://%{hiera('debian_mirror_server')}/%{hiera('debian_mirror_dir')}"
    release: 'jessie-updates'
    repos: 'main contrib non-free'
    required_packages: 'debian-archive-keyring'
    pin: '500'
    include_src: false
    architecture: 'amd64,i386'
```

```

'jessie-security':
  location: "http://{%hiera('debian_mirror_server')}/%{hiera('debian_sec_mirror_dir')}}"
  release: 'jessie/updates'
  repos: 'main contrib non-free'
  required_packages: 'debian-archive-keyring'
  pin: '500'
  include_src: false
  architecture: 'amd64,i386'
'scibian8':
  location: "http://{%hiera('scibian_mirror_server')}/%{hiera('scibian_mirror_dir')}}"
  release: 'scibian8'
  repos: 'main'
  required_packages: 'scibian-archive-keyring'
  pin: '1000'
  include_src: false
  architecture: 'amd64,i386'

##### SMTP/Postfix #####

profiles::postfix::relay::config_options:
  relay_domains:      '$mydestination example.org'
  relayhost:          'smtp.example.org'

##### LDAP/SSSD #####

ldap_external: 'ldap.example.org'

profiles::auth::client::sssd_options_domain:
  ldap_search_base:      'dc=example,dc=org'
  ldap_user_search_base: 'ou=people,dc=example,dc=org'
  ldap_group_search_base: 'ou=groups,dc=example,dc=org'

```

Note

This configuration supposes the APT, NTP, SMTP, DNS and LDAP settings are similar on all the HPC clusters of your organization. This might not be true in some specific organization environments. In this case, the settings of the affected services must be defined in the cluster specific layers of the hiera repository instead.

The examples values must be replaced with the settings corresponding to your organization environment.

3.4.3 Cluster directories

Some directories are required to store cluster specific file and settings inside the internal configuration repository. Create these directories with the following command:

```
# mkdir $ADMIN/hpc-privatedata/puppet-config/$CLUSTER \
        $ADMIN/hpc-privatedata/files/$CLUSTER \
        $ADMIN/hpc-privatedata/hieradata/$CLUSTER \
        $ADMIN/hpc-privatedata/hieradata/$CLUSTER/roles
```

3.4.4 Puppet configuration

The hpc-config-push Puppet-HPC utility expects to find a Puppet and Hiera configuration files for the cluster under the puppet-config directory of the internal configuration repository. Simply copy examples configuration files provided with Puppet-HPC:

```
cp $ADMIN/puppet-hpc/examples/privatedata/{puppet.conf,hiera.yaml} \
   $ADMIN/hpc-privatedata/puppet-config/$CLUSTER/
```

The hiera.yaml file notably specifies the layers of YAML files composing the hiera repository. It can eventually be tuned for additional layer to fit your needs.

Puppet-HPC requires the cluster name to be a *fact* (according to [Facter](#)). For this purpose, define the file `$ADMIN/hpc-privatedata/puppet-config/$CLUSTER/hpc-config-facts.yaml` with the following content:

```
---
cluster_name: 'foobar'
```

This YAML file is deployed by Puppet-HPC `hpc-config-push`, apply on every nodes in `/var/lib/puppet/facts.d` directory. The content of the YAML files located this directory are considered as valid custom facts.

Eventually, additional arbitrary custom facts can be defined for the cluster in this file at your convenience.

3.4.5 Cluster definition

The cluster specific layers of the Hiera repository must be initialized with a sufficient description of the HPC cluster. This description is the *cluster definition*.

Networks definition

A specific layer in the hiera repository stack is dedicated to all the networks settings of the HPC cluster. This layer is defined in file `$ADMIN/hpc-privatedata/hieradata/$CLUSTER/network.yaml`. Initialize this file with the following content:

```
network::ib_enable: false
network::opa_enable: true

net::administration::ipnetwork:    '10.1.0.0'
net::administration::netmask:     '255.255.255.0'
net::administration::prefix_length: '/24'
net::administration::broadcast:   '10.1.0.255'
net::lowlatency::ipnetwork:       '10.4.0.0'
net::lowlatency::prefix_length:   '/24'
net::management::ipnetwork:      '10.2.0.0'
net::management::netmask:        '255.255.255.0'
net::management::broadcast:      '10.2.0.255'

net_topology:
  wan:
    name:           'WAN'
    prefixes:       'wan'
    ipnetwork:      '10.3.0.0'
    netmask:        '255.255.255.0'
    prefix_length:  '/24'
    gateway:        '10.3.0.254'
    broadcast:      '10.3.0.255'
    ip_range_start: '10.3.0.1'
    ip_range_end:   '10.3.0.254'
    firewall_zone:  'wan'
  administration:
    name:           'CLUSTER'
    ipnetwork:      '10.1.0.0'
    netmask:        '255.255.255.0'
    prefix_length:  '/24'
    gateway:        '10.1.0.0' # fbproxy
    broadcast:      '10.1.0.255'
    ip_range_start: '10.1.0.1'
    ip_range_end:   '10.1.0.254'
    firewall_zone:  'clstr'
  pool0:
    ip_range_start: '10.1.0.1'
    ip_range_end:   '10.1.0.254'
  lowlatency:
    name:           'LOWLATENCY'
```



```

    prefixes:          'opa'
    ipnetwork:         '10.4.0.0'
    netmask:           '255.255.255.0'
    prefix_length:     '/24'
    broadcast:         '10.4.0.255'
    ip_range_start:    '10.4.0.1'
    ip_range_end:      '10.4.0.254'
    firewall_zone:     'clstr'
management:
  name:               'MGT'
  prefixes:           'mgt'
  ipnetwork:          '10.2.0.0'
  netmask:             '255.255.255.0'
  prefix_length:      '/24'
  broadcast:          '10.2.0.255'
  ip_range_start:     '10.2.0.1'
  ip_range_end:       '10.2.0.254'
  firewall_zone:      'clstr'
bmc:
  name:               'BMC'
  prefixes:           'bmc'
  ipnetwork:          '10.2.0.0'
  netmask:             '255.255.255.0'
  prefix_length:      '/24'
  broadcast:          '10.2.0.255'
  ip_range_start:     '10.2.0.1'
  ip_range_end:       '10.2.0.254'
  firewall_zone:      'clstr'

network::bonding_options:
  bond0:
    slaves:
      - eth1
      - eth2
    options:          'mode=802.3ad primary=eth2 miimon=100 updelay=200 downdelay=200'
    description:      'service nodes on adm/mgt networks'

network::bridge_options:
  br0:
    ports:
      - bond0
    description:      'service nodes on administration network'
  br1:
    ports:
      - eth3
    description:      'service nodes on management network'
  br2:
    ports:
      - eth4
    description:      'service nodes on WAN network'

master_network:
  fbsevice1:
    fqdn: "fbsevice1.%(hiera('domain'))"
    networks:
      administration:
        'DHCP_MAC': 'aa:bb:cc:dd:ee:00'
        'IP':       '10.1.0.1'
        'device':   'br0'
        'hostname': 'fbsevice1'
      lowlatency:
        'IP':       '10.4.0.1'
        'device':   'ib0'
        'hostname': 'opafbsevice1'
    bmc:

```

```
'DHCP_MAC': 'aa:bb:cc:dd:ee:01'
'IP': '10.2.0.101'
'hostname': 'bmcfbservice1'
management:
  'IP': '10.2.0.1'
  'device': 'br1'
  'hostname': 'mgtfbservice1'
wan:
  'IP': '10.3.0.1'
  'device': 'br2'
  'hostname': 'wanfbservice1'
fbservice2:
  fqdn: "fbservice2.#{hiera('domain')}}"
  networks:
    administration:
      'DHCP_MAC': 'aa:bb:cc:dd:ee:02'
      'IP': '10.1.0.2'
      'device': 'br0'
      'hostname': 'fbservice2'
    lowlatency:
      'IP': '10.4.0.2'
      'device': 'ib0'
      'hostname': 'opafbservice2'
    bmc:
      'DHCP_MAC': 'aa:bb:cc:dd:ee:03'
      'IP': '10.2.0.102'
      'hostname': 'bmcfbservice2'
    management:
      'IP': '10.2.0.2'
      'device': 'br1'
      'hostname': 'mgtfbservice2'
    wan:
      'IP': '10.3.0.2'
      'device': 'br2'
      'hostname': 'wanfbservice2'
fbservice3:
  fqdn: "fbservice3.#{hiera('domain')}}"
  networks:
    administration:
      'DHCP_MAC': 'aa:bb:cc:dd:ee:04'
      'IP': '10.1.0.3'
      'device': 'br0'
      'hostname': 'fbservice3'
    lowlatency:
      'IP': '10.4.0.3'
      'device': 'ib0'
      'hostname': 'opafbservice3'
    bmc:
      'DHCP_MAC': 'aa:bb:cc:dd:ee:05'
      'IP': '10.2.0.103'
      'hostname': 'bmcfbservice3'
    management:
      'IP': '10.2.0.3'
      'device': 'br1'
      'hostname': 'mgtfbservice3'
    wan:
      'IP': '10.3.0.3'
      'device': 'br2'
      'hostname': 'wanfbservice3'
fbservice4:
  fqdn: "fbservice4.#{hiera('domain')}}"
  networks:
    administration:
      'DHCP_MAC': 'aa:bb:cc:dd:ee:06'
      'IP': '10.1.0.4'
```

```

    'device': 'br0'
    'hostname': 'fbservice4'
  lowlatency:
    'IP': '10.4.0.4'
    'device': 'ib0'
    'hostname': 'opafbservice4'
  bmc:
    'DHCP_MAC': 'aa:bb:cc:dd:ee:07'
    'IP': '10.2.0.104'
    'hostname': 'bmcfbservice4'
  management:
    'IP': '10.2.0.4'
    'device': 'br1'
    'hostname': 'mgtfbservice4'
  wan:
    'IP': '10.3.0.4'
    'device': 'br2'
    'hostname': 'wanfbservice4'

#### High-Availability Virtual IP addresses #####
vips:
  service1:
    network: 'administration'
    ip: '10.1.0.101'
    hostname: 'vipfbservice1'
    router_id: 161
    master: 'fbservice1'
    members: 'fbservice[1-4]'
    notify:
      common:
        - "%{hiera('private_files_dir')}/ha/rndc_reload.sh"
    advert_int: '2'
  service2:
    network: 'administration'
    ip: '10.1.0.102'
    hostname: 'vipfbservice2'
    router_id: 162
    master: 'fbservice2'
    members: 'fbservice[1-4]'
    notify:
      common:
        - "%{hiera('private_files_dir')}/ha/rndc_reload.sh"
  service3:
    network: 'administration'
    ip: '10.1.0.103'
    hostname: 'vipfbservice3'
    router_id: 163
    master: 'fbservice3'
    members: 'fbservice[1-4]'
    notify:
      common:
        - "%{hiera('private_files_dir')}/ha/rndc_reload.sh"
  service4:
    network: 'administration'
    ip: '10.1.0.104'
    hostname: 'vipfbservice4'
    router_id: 164
    master: 'fbservice4'
    members: 'fbservice[1-4]'
    notify:
      common:
        - "%{hiera('private_files_dir')}/ha/rndc_reload.sh"

```

The first `network::ip,opa_enable` define which high-performance interconnect network technology is involved in the HPC cluster (InfiniBand or Intel Omni-Path).

The `net::*` parameters and `net_topology` hash basically define the addressing maps of the various IP networks of the clusters, along with some metadata such as the network hostname prefixes, the DHCP dynamic pools and the firewall zones associated to these IP networks.

The `network::bonding_options` and `network::bridge_options` hashes respectively define all the network interfaces bondings and virtual bridges involved on the nodes of the HPC cluster. Note that these settings are global to all nodes.

The `master_network` hash defines the list of nodes and all their network interfaces with the associated IP addresses, network hostnames and eventually MAC addresses (on the administration and bmc networks).

Finally, the `vips` hash define the virtual highly-available IP addresses (VIP) managed by nodes of the HPC cluster.

Initially, the YAML file must contain all the IP network definitions and the network settings of all the generic service nodes with their VIP.

General cluster settings

The cluster specific general parameters and services settings are located in file `$ADMIN/hpc-privatedata/hieradata/$O`. Initialize this file with the following content:

```
cluster_prefix: 'fb' # Prefix starting all nodes hostnames

user_groups: # Array of user groups allowed to access to the cluster
- 'grpusers1'
- 'grpusers2'

##### DNS Cluster settings #####

profiles::dns::client::nameservers:
- '10.1.0.101' # VIP addresses of generic service nodes on administration
- '10.1.0.102' # network
- '10.1.0.103'
- '10.1.0.104'

profiles::dns::server::config_options:
listen-on:
- '127.0.0.1'
- '10.1.0.1' # Static IP addresses of generic service nodes on
- '10.1.0.2' # administration network
- '10.1.0.3'
- '10.1.0.4'
- '10.1.0.101' # VIP addresses of generic service nodes on administration
- '10.1.0.102' # network
- '10.1.0.103'
- '10.1.0.104'
- '10.2.0.1' # Static IP addresses of generic service nodes on
- '10.2.0.2' # management network
- '10.2.0.3'
- '10.2.0.4'

##### Bootsystem #####

boot_params:
defaults: # default boot params to all nodes
domain: "%{hiera('domain')}}"
kernel_opts: 'persistence formatcow nosmap'
cowsize: '2G'
dhcp_timeout: '120'
diskinstall_server: "%{hiera('server_web_boot')}}"
diskless_server: '10.1.0.50:3137' # VIP adresse of P2P servers
nameserver: '10.1.0.101' # VIP address of 1st generic
# service on administration network

boot_os: 'scibian8_ram'
boot_dev: 'eth0'
```

```

console:                'ttyS0,115200n8'
fbSERVICE[1-4]:       # generic service nodes specific
                        # boot params

boot_os:                'scibian8_disk'
ipxebin:                'ipxe_noserial.bin'

# iPXE ROM downloaded by TFTP. There are 2 versions of the same ROM: with
# software serial console enabled for virtual machines and w/o software
# serial console for bare metal machines, as the serial console is transparently
# redirected to the BMC virtual serial port by the BIOS/UEFI on these nodes.
boottftp::hpc_files:
  "${hiera('tftp_dir')}/ipxe_serial.bin":
    source: "${hiera('private_files_dir')}/boot/ipxe/ipxe_serial.bin"
  "${hiera('tftp_dir')}/ipxe_noserial.bin":
    source: "${hiera('private_files_dir')}/boot/ipxe/ipxe_noserial.bin"

# Configuration files downloaded by HTTP by the debian-installer: the partition
# schema and hpc-config-apply configuration file.
boothttp::hpc_files:
  "${hiera('website_dir')}/disk/scibian8/partition-schema":
    source: "${hiera('private_files_dir')}/boot/disk-installer/scibian8/partition-schema"
  "${hiera('website_dir')}/disk/scibian8/hpc-config.conf":
    source: "file:///etc/hpc-config.conf"

# The tarball containing the Debian network installer
boothttp::archives:
  "${hiera('website_dir')}/disk/scibian8/netboot.tar.gz":
    source: "${hiera('private_files_dir')}/boot/disk-installer/scibian8/netboot.tar.gz"
    extract_path: "${hiera('website_dir')}/disk/scibian8"
    extract:      true

boothttp::install_options:
  scibian8:
    ### Localization
    'd-i debian-installer/locale string':      'en_US.UTF-8'
    'd-i debian-installer/language string':    'en'
    'd-i debian-installer/country string':     'en'
    'd-i localechooser/supported-locales multiselect': 'en_US.UTF-8'
    # Keyboard selection.
    'd-i keyboard-configuration/xkb-keymap select': 'en'
    ### Apt setup
    'apt-setup-udeb apt-setup/services-select multiselect': 'none'
    'apt-mirror-setup apt-setup/contrib boolean': 'true'
    'apt-mirror-setup apt-setup/no_mirror boolean': 'false'
    'apt-mirror-setup apt-setup/non-free boolean': 'true'
    'apt-mirror-setup apt-setup/use_mirror boolean': 'true'
    ### Mirror settings
    'd-i mirror/protocol select': 'http'
    'd-i mirror/country string': 'manual'
    'd-i mirror/http/hostname string': "${hiera('debian_mirror_server')}"
    "
    'd-i mirror/http/directory string': "/${hiera('debian_mirror_dir')}"
    'd-i mirror/suite string': 'jessie'
    'd-i debian-installer/allow_unauthenticated boolean': 'true'
    ### Network configuration
    'd-i netcfg/enable boolean': 'true'
    'd-i netcfg/choose_interface select': 'auto'
    'd-i netcfg/use_autoconfig boolean': 'true'
    'd-i netcfg/no_default_route boolean': 'true'
    'd-i netcfg/get_domain string': "${hiera('domain')}"
    # VIP addresses of the generic service nodes on the administration network
    'd-i netcfg/get_nameservers string': '10.1.0.101 10.1.0.102 10.1.0.103
    10.1.0.104'
    'd-i netcfg/dhcp_failed': 'note'
    'd-i netcfg/dhcp_options select Configure network': 'manually'

```

```

'd-i netcfg/dhcp_timeout string': '25'
'd-i netcfg/dhcpv6_timeout string': '3'
# Hardware detect
'd-i hw-detect/start_pcmcia boolean': 'false'
'd-i hw-detect/load_firmware boolean': 'true'
### Partitioning
'd-i partman/early_command string': >
  env -u http_proxy wget http://%{hiera('server_web_boot')}/disk/scibian8/partition-
schema -O /partition-schema;
  debconf-set partman-auto/disk "$(list-devices disk | head -n1)"
'd-i partman-auto/method string': 'lvm'
'd-i partman-partitioning/choose_label select': 'gpt'
'd-i partman-partitioning/confirm_write_new_label boolean': 'true'
'd-i partman-auto/expert_recipe_file string': '/partition-
schema'
'partman-basicfilesystems partman-basicfilesystems/no_mount_point boolean': 'false'
'd-i partman-lvm/device_remove_lvm boolean': 'true'
'd-i partman-auto/purge_lvm_from_device boolean': 'true'
'd-i partman-md/device_remove_md boolean': 'true'
'd-i partman-md/confirm_nochanges boolean': 'true'
'd-i partman-lvm/confirm boolean': 'true'
'd-i partman-lvm/confirm_nooverwrite boolean': 'true'
'd-i partman-auto/choose_recipe select': 'multiraid'
'd-i partman-md/confirm boolean': 'true'
'd-i partman-md/confirm_nooverwrite boolean': 'true'
'd-i partman/confirm_write_new_label boolean': 'true'
'd-i partman/confirm_nooverwrite boolean': 'true'
'd-i partman/choose_partition select': 'finish'
'd-i partman/confirm boolean': 'true'
### Boot loader
'grub-installer grub-installer/only_debian boolean': 'true'
'grub-installer grub-installer/bootdev string': 'default'
'grub-installer grub-installer/skip boolean': 'false'
'grub-installer grub-installer/multipath boolean': 'true'
'grub-installer grub-installer/with_other_os boolean': 'false'
'grub-installer grub-installer/make_active boolean': 'true'
# Don't create a normal user account.
'user-setup-udeb passwd/make-user boolean': 'false'
'user-setup-udeb passwd/root-login boolean': 'true'
'user-setup-udeb passwd/shadow boolean': 'true'
### Clock and time zone setup
'clock-setup clock-setup/utc boolean': 'true'
'clock-setup clock-setup/system-time-changed boolean': 'false'
'clock-setup clock-setup/hwclock-wait boolean': 'false'
'clock-setup clock-setup/ntp boolean': 'false'
'tzsetup-udeb time/zone select': 'GMT'
### Base system installation
'bootstrap-base base-installer/kernel/image select': 'linux-image-amd64'
'bootstrap-base base-installer/initramfs-tools/driver-policy select': 'dep'
'base-installer base-installer/install-recommends boolean': 'false'
### Package selection
'tasksel tasksel/first_multiselect': 'standard, ssh-server'
'd-i pkgssel/include string': >
  apt-transport-https
  scibian-archive-keyring
  hpc-config-apply
# Additional repositories, local[0-9] available
'd-i apt-setup/local0/repository string': "http://%{hiera('scibian_mirror_server')}/%{
hiera('scibian_mirror_dir')} scibian8 main contrib non-free"
'd-i pkgssel/upgrade select': 'full-upgrade'
# Other packages ##
'd-i popularity-contest/participate boolean': 'false'
'console-setup console-setup/charmap47 select': 'UTF-8'
'console-setup console-setup/codesetcode string': 'Lat15'
'console-setup console-setup/store_defaults_in_debconf_db boolean': 'true'

```

```

'console-setup console-setup/fontsize-fb47 select':           '8x16'
'console-setup console-setup/fontsize string':               '8x16'
### Late command ###
'd-i preseed/late_command string': >
  /bin/in-target sed -i 's/PermitRootLogin without-password/PermitRootLogin yes/' /etc/
ssh/sshd_config;
  env -u http_proxy wget http://[{hiera('server_web_boot')}] /disk/scibian8/hpc-config.conf
  -O /target/etc/hpc-config.conf;
  /bin/in-target mkdir -p /var/lib/puppet/facts.d;
  /bin/in-target /usr/bin/env -u http_proxy hpc-config-apply -vvv;
# Root password, either in clear text
'd-i passwd/root-password-crypted password': "%{hiera('profiles::cluster::
root_password_hash')}}"
# Avoid that last message about the install being complete.
'd-i finish-install/reboot_in_progress note': ''

##### DHCP #####

profiles::dhcp::default_options:
- 'INTERFACES=br0 br1' # bridge interfaces of the generic service nodes on the
# administration and management networks

profiles::dhcp::includes:
  adm-subnet:
    'pool_name':           'subnet'
    'subnet_name':        'adm'
    'tftp':                true
    'pool':
    'use-host-decl-names': 'on'
    'deny':                'unknown-clients'
    'max-lease-time':      '1800'
    # Range of IP addresses on the administration network
    'range':               '10.1.0.1 10.1.0.254'
    'include':              '/etc/dhcp/adm_subnet'
  mgt-subnet:
    'pool_name':           'subnet'
    'subnet_name':        'mgt'
    'tftp':                false
    'pool':
    'use-host-decl-names': 'on'
    'deny':                'unknown-clients'
    'max-lease-time':      '1800'
    # Range of IP addresses on the management network
    'range':               '10.2.0.1 10.2.0.254'
    'include':              "/etc/dhcp/mgt_subnet"

profiles::dhcp::sharednet:
  'name':                  "%{hiera('cluster_name')}net"
  'subnet':
  - 'name':                'adm'
    'network':              "%{hiera('net::administration::ipnetwork')}}"
    'netmask':              "%{hiera('net::administration::netmask')}}"
    'domain-name':         "%{hiera('domain')}}"
    # VIP addresses of the generic service nodes on the administration network
    'domain-name-servers': '10.1.0.101, 10.1.0.102 10.1.0.103 10.1.0.104'
    'broadcast':           "%{hiera('net::administration::broadcast')}}"
  - 'name':                'mgt'
    'network':              "%{hiera('net::management::ipnetwork')}}"
    'netmask':              "%{hiera('net::management::netmask')}}"
    'domain-name':         "%{hiera('domain')}}"
    # Static IP addresses of the generic service nodes on the management network
    'domain-name-servers': '10.2.0.1, 10.2.0.2, 10.2.0.3, 10.2.0.4'
    'broadcast':           "%{hiera('net::management::broadcast')}}"

```

Additionally to some general parameters (`cluster_prefix` and `user_groups`), the initial version of this file notably contains the configuration of the base services required to install nodes on disk (DNS, TFTP,

HTTP, DHCP, Debian installer, etc).

Also, in order to prevent user to access the cluster during the installation process, it is recommended to enable the maintenance mode in this file:

```
profiles::access::maintenance_mode: true
```

3.4.6 Service role

The Puppet role `service` associated to the generic service nodes must be defined with the corresponding profiles. This is achieved by initializing file `$ADMIN/hpc-privatedata/hieradata/$CLUSTER/roles/service.yaml` with the following content:

```
profiles:
  # common
  - profiles::cluster::common
  - profiles::systemd::base
  - profiles::ssmtp::client
  - profiles::network::base
  - profiles::dns::client
  - profiles::access::base
  - profiles::openssh::server
  - profiles::openssh::client
  - profiles::environment::base
  - profiles::environment::limits
  - profiles::environment::service
  - profiles::log::client
  # HW host
  - profiles::hardware::ipmi
  - profiles::hardware::admin_tuning
  # service
  - profiles::hpcconfig::push
  - profiles::hpcconfig::apply
  - profiles::ntp::server
  - profiles::openssh::client_identities
  - profiles::clush::client
  - profiles::ha::base
  - profiles::http::secret
  - profiles::apt::proxy
  - profiles::log::server
  - profiles::dns::server
  - profiles::bootssystem::server
  - profiles::dhcp::server

profiles::network::gw_connect: 'wan'
```

The first profiles (below the *common* comment) are common to all nodes of the cluster. The profiles after the *HW host* comment are common to all bare metal nodes. The last profiles, after the *service* comment, carry the base services hosted by the generic service nodes.

The last parameter `profiles::network::gw_connect` defines on which network's gateway the nodes use as their default route.

3.4.7 Authentication and encryption keys

Configuration encryption keys

As explained in the *Puppet-HPC reference documentation* (chapter *Software Architecture*, section *Sensitive Data Encryption*), the sensitive data stored in the internal configuration repository are encrypted using two keys:

- asymmetric PKCS7 key pair for encrypting values in Hiera,

- symmetric AES key for encrypting files.

These keys must be generated following the bootstrap procedure available in the Puppet-HPC Reference Documentation.

SSH host keys

The SSH host keys must stay consistent between node re-installations and/or diskless reboots. To ensure this, the SSH host keys are generated in the cluster's files directory of the internal configuration repository before their first installation and/or diskless boot.

Firstly, create the `hostkeys` directory:

```
# cd $ADMIN
# mkdir -p hpc-privatedata/files/$CLUSTER/hostkeys
```

During this step, the `known_hosts` file will also be synchronized with the generated keys. This file will be stored in `hpc-privatedata/files/$CLUSTER/ssh/known_hosts`. To build this file, the script needs to know the local domain name for the cluster, the script will use the local domain of the machine where it runs by default. If this is not correct you must provide the domain.

Run the following script:

```
# puppet-hpc/scripts/sync-ssh-hostkeys.sh hpc-privatedata $CLUSTER $CLUSTER.$NETDOMAIN
```

This script ensures that all nodes present in the `master_network` hash have valid SSH host keys.

SSH root key

For password-less SSH authentication from the admin and generic service nodes to all the other nodes of the cluster, SSH authentication keys pair are deployed for root on the nodes.

First, create the `rootkeys` sub-directory in the cluster's files directory of the internal configuration repository:

```
# cd $ADMIN
# mkdir -p hpc-privatedata/files/$CLUSTER/rootkeys
```

Then, generate the key pair:

```
# ssh-keygen -t rsa -b 2048 -N '' -C root@$CLUSTER \
  -f hpc-privatedata/files/$CLUSTER/rootkeys/id_rsa_root
```

Eventually, adjust the key type and size. Encode the private key with the following helper script provided by Puppet-HPC:

```
# puppet-hpc/scripts/encode-file.sh hpc-privatedata $CLUSTER \
  hpc-privatedata/files/$CLUSTER/rootkeys/id_rsa_root
generating encoded file hpc-privatedata/files/foobar/rootkeys/id_rsa_root.enc
```

Do not forget to remove the generated unencrypted private key:

```
# rm hpc-privatedata/files/$CLUSTER/rootkeys/id_rsa_root
```

Finally, publish the public key with the following parameter in the cluster specific layer of the hiera repository `$ADMIN/hpc-privatedata/hieradata/$CLUSTER/cluster.yaml`:

```
openssh::server::root_public_key: <pubkey>
```

3.4.8 Initial files

There are few files that are required in the cluster's file directory of the internal configuration repository to deploy the generic service nodes.

First, create the needed sub-directories:

```
mkdir -p $ADMIN/hpc-privatedata/files/$CLUSTER/boot/ipxe \
        $ADMIN/hpc-privatedata/files/$CLUSTER/boot/disk-installer/scibian8 \
        $ADMIN/hpc-privatedata/files/$CLUSTER/ha
```

Download the Debian network installer archive:

```
# wget http://deb.debian.org/debian/dists/jessie/main/installer-amd64/current/images/netboot/
  netboot.tar.gz \
  -O $ADMIN/hpc-privatedata/files/$CLUSTER/boot/disk-installer/scibian8/netboot.tar.gz
```

The iPXE ROM must be built from sources downloaded on [iPXE website](#):

```
# wget https://git.ipxe.org/ipxe.git/snapshot/HEAD.tar.gz -O $ADMIN/ipxe.tar.gz
# tar -C $ADMIN -xzf $ADMIN/ipxe.tar.gz
# cd $ADMIN/ipxe-*/src
# make bin/undionly.kpxe
# cp bin/undionly.kpxe $ADMIN/hpc-privatedata/files/$CLUSTER/boot/ipxe/ipxe_noserial.bin
```

The ipxe_serial.bin variant must be generated after modifying hard-coded serial console parameters in iPXE source code as documented in [iPXE official documentation](#).

The iPXE menu boot entries are generated dynamically, depending on the source node name, by the CGI Python script \$ADMIN/hpc-privatedata/files/\$CLUSTER/boot/cgi/bootmenu.py:

```
#!/usr/bin/python
import ClusterShell.NodeSet
import cgi
import yaml
import sys
import syslog

sys.stderr = sys.stdout
params = cgi.FieldStorage()
target = params.getvalue('node')

print("Content-Type: text/plain")
print("")

config = yaml.load(open('/etc/hpc-config/bootmenu.yaml'))

def lookup_param(key):
    global config, target
    for nodeset in config:
        if nodeset == 'defaults':
            continue
        if target in ClusterShell.NodeSet.expand(nodeset) and key in config[nodeset]:
            return config[nodeset][key]
    return config['defaults'][key]

ipxe_menu = {
    'params' : {
        'timeout' : '3000',
        'boot' : 'boot || goto failed',
        'goto' : 'goto start',
    },
    'menu' : {
        'scibian8_ram' : {
            'label' : 'Run Scibian8 in RAM',
            'url' : "http://" + lookup_param('diskless_server') + "/scibian8",
            'initrd' : '${base-url}/initrd-3.16.0-4-amd64',
            'kernel' : "${base-url}/vmlinuz-3.16.0-4-amd64 initrd=initrd-3.16.0-4-amd64 " +
                " console=" + lookup_param('console') +
                " ethdevice=" + lookup_param('boot_dev') +
                " ethdevice-timeout=" + lookup_param('dhcp_timeout') +
                " cowsize=" + lookup_param('cowsize') +
                " transparent_hugepage=always " +
```

```

        " disk-format=" + lookup_param('disk_format') +
        " disk-raid=" + lookup_param('disk_raid') +
        " boot=live " +
        " fetch=http://" + lookup_param('diskless_server') + "/scibian8/
scibian8.squashfs.torrent " +
        lookup_param('kernel_opts'),
    },
    'scibian8_disk' : {
        'label' : 'Install Scibian8',
        'url' : "http://" + lookup_param('diskinstall_server') + "/disk/scibian8",
        'initrd' : '${base-url}/debian-installer/amd64/initrd.gz',
        'kernel' : "${base-url}/debian-installer/amd64/linux initrd=initrd.gz console="+
lookup_param('console')+" url=${base-url}/install_config auto interface="+lookup_param('
boot_dev')+" locale=en_US console-keymaps-at/keymap=fr keyboard-configuration/xkb-keymap=
fr languagechooser/language-name=English netcfg/get_domain="+lookup_param('domain')+"
netcfg/get_nameservers="+lookup_param('nameserver')+" netcfg/no_default_route=true debian
-installer/add-kernel-opts=console="+lookup_param('console')+" priority=critical scibian-
installer",
    },
    'discovery' : {
        'label' : 'Discover/Rescue system',
        'url' : "http://" + lookup_param('diskless_server') + "/scibian8",
        'initrd' : '${base-url}/initrd-3.16.0-4-amd64',
        'kernel' : "${base-url}/vmlinuz-3.16.0-4-amd64 initrd=initrd-3.16.0-4-amd64
console="+lookup_param('console')+" boot=discovery interface="+lookup_param('boot_dev'),
    },
},
}

print("#!ipxe")
print("#####")
print("# IPXE Linux menu specification")
print("#####")
print("set esc:hex 1b          # ANSI escape character - ^[")
print("set cls ${esc:string}[2J # ANSI clear screen sequence - ^[[2J")
print("menu Please choose an operating system to boot")
print("echo ${cls}")
for key in ipxe_menu["menu"]:
    print("item "+key+"          "+ipxe_menu["menu"][key]["label"])
    if key == lookup_param('boot_os'):
        menudft = key
if not menudft:
    menudft = 'discovery'
print("choose --default "+menudft+" --timeout "+ipxe_menu["params"]["timeout"]+" target &&
goto ${target}")

syslog.syslog("generating menu for " + target + " with default kernel: " + ipxe_menu["menu"][
menudft]["kernel"] )

for key in ipxe_menu["menu"]:
    print("")
    print(": "+key)
    print("set base-url "+ipxe_menu["menu"][key]["url"])
    print("initrd "+ipxe_menu["menu"][key]["initrd"])
    print("kernel "+ipxe_menu["menu"][key]["kernel"])
    print(ipxe_menu["params"]["boot"])
    print(ipxe_menu["params"]["goto"])

```

Debian installer partman utility requires a partition schema with rules and constraints to create the disk partitions. This file must be located in `$ADMIN/hpc-privatedata/files/$CLUSTER/boot/disk-installer/scibian8/p`. Here is a complete example for this file:

```

512 512 1024 free
    $iflabel{ gpt }
    $reusethold{ }

```

```

        method{ efi }
        format{ } .

256 256 256 ext2
    $primary{ }
    $bootable{ }
    method{ format }
    format{ }
    use_filesystem{ }
    filesystem{ ext2 }
    mountpoint{ /boot } .

1 250000 -1 ext4
    $defaultignore{ }
    $primary{ }
    method{ lvm }
    vg_name{ sysvg } .

20000 100000 100000 ext4
    $lvmok{ }
    in_vg{ sysvg }
    lv_name{ root }
    method{ format }
    format{ }
    use_filesystem{ }
    filesystem{ ext4 }
    mountpoint{ / } .

30000 100000 100000 ext4
    $lvmok{ }
    in_vg{ sysvg }
    lv_name{ var }
    method{ format }
    format{ }
    use_filesystem{ }
    filesystem{ ext4 }
    mountpoint{ /var } .

2000 10000 20000 ext4
    $lvmok{ }
    in_vg{ sysvg }
    lv_name{ tmp }
    method{ format }
    format{ }
    use_filesystem{ }
    filesystem{ ext4 }
    mountpoint{ /tmp } .

2000 100000 2000+50% linux-swap
    $lvmok{ }
    in_vg{ sysvg }
    lv_name{ swap }
    method{ swap }
    format{ } .

```

Finally, the version bind9 DNS server available in Debian Jessie is not able to automatically listen for incoming connections to IP addresses configured after the daemon has started. This is annoying when virtual IP addresses are gathered to failover a failing generic service node. This behaviour is workaround with the following script located in `$ADMIN/hpc-privatedata/files/$CLUSTER/ha/rndc_reload.sh`:

```

#!/bin/sh

# This is needed with bind 9.5 to make it:
#
# - start listening on new interfaces

```

```
# - stop listening on interfaces that don't exist anymore
#
# This becomes useless starting from bind >= 9.10 thanks to new
# automatic-interface-scan feature.

rndc reconfig
```

This script is run when the virtual IP addresses move from one generic service node to another. It asks bind9 daemon to reload its configuration which also makes it listen on new IP addresses.

3.5 Generic service nodes

3.5.1 Temporary installation services

The goal of this section is to configure the Temporary Installation Services on the Temporary Installation Node. This process is done in two steps:

- A First Run only using local files or external services
- A Second Run reconfiguring the Temporary Installation Node to use the services setup during the First Run with values that will also be used to install the other generic nodes remotely.

The reference architecture (with Ceph/S3) does not use the local apache server in the final configuration. Some other architectures use a local apache Virtual Host instead of Ceph/S3 (*profiles::http::system*). If this is the case, `/var/www/html` can be replaced by `hieradata('profiles::http::system::docroot')` and `http.service` by `web-system.service` in the `private_files_dir` URL.

3.5.2 First Run

Consul is not available because the consul cluster needs quorum to work. Quorum can only be achieved when more than half of the generic service nodes are configured. The DNS server is therefore configured to only return the temporary installation node for all requests on the consul domain. This is done simply by adding temporarily the following parameters in file `$ADMIN/hpc-privatedata/hieradata/$CLUSTER/cluster.yaml`:

```
dns::server::virtual_relay: false
install_server_ip: '10.1.0.1' # static IP address of the temporary
                           # installation node on the administration
                           # network
```

Technically speaking, these parameters make bind authoritative on the *virtual* DNS zone before Consul service discovery utility is available. The virtual zone contains all the symbolic names to the network services (ex: `http.virtual`). This way, all services will be directed to the temporary installation node with the IP address provided in `install_server_ip` parameter.

The first run also needs to work properly without a local DNS server and without a local repository cache proxy. These services will be configured during this first run. Local repositories must also be disabled during the first run.

```
private_files_dir: "file:///var/www/html/hpc-config/{environment}/latest/files"
apt::proxy_host: ''
profiles::dns::client::nameservers:
  - '172.16.1.1' # External DNS server
hpcconfig::push::config_options:
  global:
    cluster:    "%{hieradata('cluster_name')}}"
    mode:       'posix'
    destination: "/var/www/html/hpc-config"
```

The configuration will be pushed on local files while the temporary installation is used. The settings above configure this, but the first push must use a configuration that will be created manually in the file: `/etc/hpc-config/push.conf`.

```

---
[global]
environment=production
version=latest
destination=/var/www/html/hpc-config
cluster=<CLUSTER NAME>
mode=posix
---

```

The directory where the keys were generated cannot be used as a key source for apply because it will be overwritten during the apply. So it must be copied before doing the apply. To deploy the configuration of the temporary installation node, run the following commands:

```

# cd $ADMIN
# hpc-config-push
# cp -a /etc/puppet/secure/keys keys
# hpc-config-apply --source file:///var/www/html/hpc-config \
                  --keys-source=file://$ADMIN/keys \
                  --verbose
# rm -rf $ADMIN/keys

```

3.5.3 Second Run

At this stage the unique temporary installation node should work, the configuration is modified to use it. The following hiera keys must be changed in file `$ADMIN/hpc-privatedata/hieradata/$CLUSTER/cluster.yaml`:

```

private_files_dir: "http://http.service.%(hiera('virtual_domain'))/hpc-config/%{environment}/
                  latest/files"
# apt:proxy_host is left to the default values
# profiles::dns::client::nameservers is configured as detailed above

```

Check the new settings with:

```

# cd $ADMIN
# hpc-config-push
# hpc-config-apply --verbose

```

If the two commands run without error, the initial setup succeeded.

At this stage, the temporary installation service are fully configured and available to install other generic service nodes.

3.5.4 Base system installation

The other generic service nodes must now be rebooted in PXE mode to run the Debian installer and configure the base system:

```

for BMC in $(nodeset -e bmcfbSERVICE[2-4]); do
    ipmitool -I lanplus -U ADMIN -P ADMIN -H $BMC chassis bootdev pxe
    ipmitool -I lanplus -U ADMIN -P ADMIN -H $BMC power reset
done

```

Replace the BMC credentials with the appropriate values.

Once the base system is fully installed, the nodes reboot and become available with SSH. Check this with:

```

# clush -bw fbSERVICE[2-4] uname
-----
fbSERVICE[2-4] (3)
-----
Linux

```

3.5.5 Ceph deployment

Deployment is based on a tool called `ceph-deploy`. This tool performs the steps on a node to setup a ceph component. It is only used for the initial setup of the Ceph cluster. Once the cluster is running, the configuration is reported in the Puppet configuration in case it is re-deployed.

The reference configuration uses one disk (or hardware RAID LUN) to hold the system (`/dev/sda`) and another to hold the Ceph OSD data and journal (`/dev/sdb`). Three or five nodes must be chosen to setup the **mon** and **mds** services, the remaining nodes are used only as **osd** and **radosgw** nodes.

The `ceph-deploy` utility generates authentication keys for Ceph. Once the cluster is running, these keys are manually collected and encrypted with `eyaml` to be included in the **hier**a configuration.

In the following example MONs/MDS are installed on nodes `fb-service[2-4]` while the node `fb-service1` only has OSD and RGW.

Packages installation

Install the `ceph-deploy` utility and the S3 CLI client `s3cmd`:

```
# apt-get install ceph-deploy s3cmd
```

The deployment of Ceph cluster generates a bunch of files (keyrings, configuration file, etc). Create a temporary directory to store these files:

```
# mkdir ~root/ceph-deploy
# cd ~root/ceph-deploy
```

Install the Ceph software stack on all nodes of the Ceph cluster:

```
# ceph-deploy install --no-adjust-repos $(nodeset -e fb-service[1-4])
```

Cluster bootstrap

Initialize the cluster with the first MON server of the Ceph cluster in parameter:

```
# ceph-deploy new fb-service2
```

Install admin credentials

```
# ceph-deploy admin $(nodeset -e fb-service[1-4])
```

Create the MON servers:

```
# ceph-deploy mon add $(nodeset -e fb-service[2-4])
```

Create the OSD servers:

```
# ceph-deploy disk zap $(nodeset -e fb-service[1-4]:sdb)
# ceph-deploy osd prepare $(nodeset -e fb-service[1-4]:sdb)
```

Create the MDS servers:

```
# ceph-deploy mds create $(nodeset -e fb-service[2-4])
```

Check the Ceph cluster status:

```
# ceph status
```

The command must report `HEALTH_OK`.

RadosGW

Enable RadosGW with the following command:

```
# ceph-deploy rgw create $(nodeset -e fb-service[1-4])
```

Libvirt RBD pool

The virtual machines will use a specific libvirt storage pool to store the disk images. This libvirt storage pool uses ceph RBD, so a specific ceph pool is necessary. This is not handled by `ceph-deploy`:

```
# ceph osd pool create libvirt-pool 64 64
```

If the cluster has five OSDs or more, the numbers of PG and PGP can be set to 128 instead of 64.

The client credentials must be manually generated:

```
# ceph auth get-or-create client.libvirt \
    mon 'allow r' \
    osd 'allow class-read object_prefix rbd_children, allow rwx pool=libvirt-pool'
```

CephFS initialization

In high-availability mode, Slurm controller requires a shared POSIX filesystem between the primary and the backup controllers. In the Scibian HPC cluster reference architecture, CephFS is used for this filesystem. Create this CephFS filesystem with the following commands:

```
# ceph osd pool create cephfs_data 64 64
pool 'cephfs_data' created
# ceph osd pool create cephfs_metadata 64 64
pool 'cephfs_metadata' created
# ceph fs new cephfs cephfs_metadata cephfs_data
new fs with metadata pool 15 and data pool 14
```

If the cluster has five OSDs or more, the numbers of PGs can be set to 128 for data and metadata pool.

RadosGW S3

A user must be created to access the RadosGW S3 API:

```
# radosgw-admin user create --uid=hpc-config --display-name="HPC Config push"
```

This commands gives an `access_key` and a `secret_key` that can be used by `hpc-config-push(1)` or `s3cmd(1)`.

Create a temporary configuration file for `s3cmd` with these keys:

```
# cat <<EOF >> ~/.s3cfg
[default]
access_key=<ACCESS_KEY>
secret_key=<SECRET_KEY>
host_bucket=%(bucket)s.service.virtual:7480
host_base=rgw.service.virtual:7480
EOF
```

With the `access_key` and the `secret_key` provided by `radosgw-admin user create` command.

To work properly with Amazon S3 tools and consul DNS, RadosGW must be configured to accept requests on `rgw.service.virtual` and on `<bucket_name>.service.virtual`. To configure this, it is necessary to re-define the default realm, region and zonegroup.

The region is configured by writing a JSON region file (`rgw-region.json`):

```
{"name": "default",
  "api_name": "",
  "is_master": "true",
  "endpoints": [],
  "hostnames": ["rgw.service.virtual", "service.virtual"],
  "master_zone": "",
  "zones": [
    {"name": "default",
```



```

    "endpoints": [],
    "log_meta": "false",
    "log_data": "false"}
  ],
  "placement_targets": [
    {"name": "default-placement",
     "tags": [] }],
  "default_placement": "default-placement"
}

```

Inject this region file into RadosGW configuration:

```

# radosgw-admin realm create --rgw-realm=default --default
# radosgw-admin region set --infile rgw-region.json
# radosgw-admin region default --rgw-zonegroup=default
# radosgw-admin zonegroup add --rgw-zonegroup=default --rgw-zone=default

```

Define default zone and zonegroup:

```

# radosgw-admin zone default --rgw-zone=default
# radosgw-admin zonegroup default --rgw-zonegroup=default

```

Update the period:

```

# radosgw-admin period get
# radosgw-admin period update --commit

```

After this step the RadosGW daemons must be restarted on every nodes:

```

clush -w fbSERVICE[1-4] 'systemctl restart ceph-radosgw@rgw.${HOSTNAME}.service'

```

Finally, create the bucket with s3cmd:

```

# s3cmd mb --acl-public s3://s3-system
Bucket 's3://s3-system/' created

```

Transfer to Hiera

When the Ceph cluster is fully initialized, its configuration must be reported into the cluster specific layer of the hiera repository `$ADMIN/hpc-privatedata/hieradata/$CLUSTER/cluster.yaml`. Here is an example a Ceph cluster configuration in Hiera:

```

profiles::ceph::config_options:
  global:
    fsid:                '9ce06246-210f-4f59-866a-1f384852da38'
    mon_initial_members: 'fbSERVICE2, fbSERVICE3, fbSERVICE4'
    mon_host:            'fbSERVICE2, fbSERVICE3, fbSERVICE4'
    auth_cluster_required: 'cephx'
    auth_service_required: 'cephx'
    auth_client_required: 'cephx'

ceph_client_admin_key: <eyaml encrypted key>

ceph::keyrings:
  client.admin.keyring:
    client.admin:
      key: "%{hiera('ceph_client_admin_key')}}"
  ceph.mon.keyring:
    'mon.':
      key: <eyaml encrypted key>
      'caps mon': 'allow *'
  ceph.bootstrap-mds.keyring:
    client.bootstrap-mds:
      key: <eyaml encrypted key>
  ceph.bootstrap-osd.keyring:

```

```
client.bootstrap-osd:
  'key': <eyaml encrypted key>
ceph.bootstrap-rgw.keyring:
  client.bootstrap-rgw:
    key: <eyaml encrypted key>

ceph::osd_config:
  fbsservice1:
    id: '0'
    device: '/dev/sdb1'
    key: <eyaml encrypted key>
  fbsservice2:
    id: '1'
    device: '/dev/sdb1'
    key: <eyaml encrypted key>
  fbsservice3:
    id: '2'
    device: '/dev/sdb1'
    key: <eyaml encrypted key>
  fbsservice4:
    id: '3'
    device: '/dev/sdb1'
    key: <eyaml encrypted key>

ceph::mon_config:
- fbsservice2
- fbsservice3
- fbsservice4

ceph::mds_config:
- fbsservice2
- fbsservice3
- fbsservice4

ceph::mds_keyring:
  fbsservice2:
    mds.fbsservice2:
      key: <eyaml encrypted key>
  fbsservice3:
    mds.fbsservice3:
      key: <eyaml encrypted key>
  fbsservice4:
    mds.fbsservice4:
      key: <eyaml encrypted key>

ceph::rgw_config:
- fbsservice1
- fbsservice2
- fbsservice3
- fbsservice4

ceph::rgw_client_keyring:
  fbsservice1:
    client.rgw.fbsservice1:
      key: <eyaml encrypted key>
  fbsservice2:
    client.rgw.fbsservice2:
      key: <eyaml encrypted key>
  fbsservice3:
    client.rgw.fbsservice3:
      key: <eyaml encrypted key>
  fbsservice4:
    client.rgw.fbsservice4:
      key: <eyaml encrypted key>
```

Many parameters in this example must be replaced to match the exact Ceph cluster configuration.

First, the fsid can be obtained with:

```
# ceph fsid
```

The bootstrap keys have been generated in the temporary Ceph deployment directory:

```
# cd ~root/ceph-deploy
# cat ceph.client.admin.keyring
# cat ceph.mon.keyring
# cat ceph.bootstrap-mds.keyring
# cat ceph.bootstrap-osd.keyring
# cat ceph.bootstrap-rgw.keyring
```

The OSD keys can be gathered with:

```
# clush -bw fbservice[1-4] 'cat /var/lib/ceph/osd/ceph-*/keyring'
```

The MDS keys can be gathered with:

```
# clush -bw fbservice[2-4] 'cat /var/lib/ceph/mds/ceph-#{HOSTNAME}/keyring'
```

The RGW keys can be gathered with:

```
# clush -bw fbservice[1-4] 'cat /var/lib/ceph/radosgw/ceph-rgw.#{HOSTNAME}/keyring'
```

Then, add the `ceph::server` profile into the service role:

```
--- a/hpc-privatedata/hieradata/foobar/roles/service.yaml
+++ b/hpc-privatedata/hieradata/foobar/roles/service.yaml
@@ -28,5 +28,6 @@
- profiles::bootsystem::server
- profiles::dhcp::server
- profiles::environment::limits
+ - profiles::ceph::server

profiles::network::gw_connect: 'wan'
```

Then push the new configuration:

```
# hpc-config-push
```

Theoretically, at this stage, the Ceph cluster can be fully configured with Puppet. It is really recommended to check this by re-installing one of the generic service nodes (excepting the temporary installation node) before going further. Please mind that in case of generic service node reinstallation after the initial configuration, bootstrap steps may be necessary:

- **mds** and **radosgw**, those services have no state outside of Rados, so no additional bootstrap is necessary
- **mon** Always necessary to bootstrap
- **osd** Must be bootstrapped if the OSD volume (`/dev/sdb`) is lost.

Please refer to the bootstrap procedure section for all details.

Once the re-installation of a generic service node with Ceph is validated, the `ceph-deploy` temporary directory can be removed from the temporary installation node:

```
# rm -r ~root/ceph-deploy
```

Network restrictions

By default with Puppet-HPC, Ceph daemons socket are binded to the administration network interface of the generic service nodes. This setup is done on purpose for security reasons and avoid access to the Ceph cluster from outside of the administration network (typically from the wan network, outside of the cluster).

However, this can be easily changed by overriding this parameter in the hiera repository:

```
profiles::ceph::listen_network: 'wan' # Make ceph listen the wan network for
                                   # connections, default is 'administration'
```

It is also possible to totally disable the network restriction settings on Ceph daemons with:

```
ceph::restrict_network: false
```

3.5.6 Migrate configuration

At this stage, the configuration was published on the HTTP server of the temporary installation node. Now that Ceph cluster is available, the configuration can be migrated to the S3 backend of Ceph RadosGW.

Edit the cluster specific layer of the hiera repository `$ADMIN/hpc-privatedata/hieradata/$CLUSTER/cluster.yaml` to make the `hpc-config` utilities push and download configuration in dedicated Ceph S3 bucket:

```
hpcconfig::apply::config_options:
  DEFAULT:
    source:
      value: "http://s3-system.service.{{hiera('virtual_domain')}}:7480/hpc-config"
    keys_source:
      value: "http://secret.service.{{hiera('virtual_domain')}}:{{hiera('secret_port')}}"
hpcconfig::push::config_options:
  global:
    cluster:    "{{hiera('cluster_name')}}"
    mode:       's3'
  s3:
    access_key: "{{hiera('s3::access_key')}}"
    secret_key: "{{hiera('s3::secret_key')}}"
    bucket_name: 's3-system'
    host:        'rgw.service.virtual'
    port:        '7480'

s3::access_key: DEC::PKCS7[<access_key>]!
s3::secret_key: DEC::PKCS7[<secret_key>]!

s3cmd::config_options:
  default:
    access_key: "{{hiera('s3::access_key')}}"
    secret_key: "{{hiera('s3::secret_key')}}"
    host_bucket: '%(bucket)s.service.virtual:7480'
    host_base:    'rgw.service.virtual:7480'
```

Push and apply configuration:

```
# hpc-config-push
# clush -bg service hpc-config-apply -v
```

At this stage, the `hpc-config` utilities are configured to use Ceph. Try this new configuration by running them once again:

```
# hpc-config-push
# clush -bg service hpc-config-apply -v
```

Finally, tune the `private_files_dir` in file `$ADMIN/hpc-privatedata/hieradata/$CLUSTER/cluster.yaml` to use Ceph S3 as well:

```
-private_files_dir: "http://http.service.{{hiera('virtual_domain')}}/hpc-config/{{environment
  }}/latest/files"
+private_files_dir: "http://s3-system.service.{{hiera('virtual_domain')}}:7480/hpc-config/{{
  environment }}/latest/files"
```

Run a final check:

```
# hpc-config-push
# clush -bg service hpc-config-apply -v
```

Starting from this moment, all the cluster configuration is hosted in the Ceph distributed filesystem.

3.5.7 Consul deployment

All the base services are now deployed on all the generic service nodes. It is time to enable load-balancing and high-availability with Consul service discovery tool.

Consul needs a shared secret key to encrypt communication between its distributed agents. Generate this key with:

```
# dd if=/dev/urandom bs=16 count=1 2>/dev/null | base64
```

The output of this command must be reported in the cluster's specific layer of the hiera repository `$ADMIN/hpc-privatedata/hieradata/$CLUSTER/cluster.yaml` with eyaml:

```
consul::key: DEC::PKCS7[<key>!]
```

Add `consul::server` profile to the `service` role:

```
--- a/hpc-privatedata/hieradata/foobar/roles/service.yaml
+++ b/hpc-privatedata/hieradata/foobar/roles/service.yaml
@@ -29,5 +29,6 @@
- profiles::dhcp::server
- profiles::environment::limits
- profiles::ceph::server
+ - profiles::consul::server

profiles::network::gw_connect: 'wan'
```

Then, run Puppet on all services nodes:

```
# hpc-config-push && clush -bg service hpc-config-apply -v
```

Check that all the generic service nodes are members of the Consul cluster with this command:

```
# clush --pick -Ng service consul members
Node      Address      Status  Type    Build  Protocol  DC
fbservice1 10.1.0.1:8301 alive   server  0.6.4   2         foobar
fbservice2 10.1.0.2:8301 alive   server  0.6.4   2         foobar
fbservice3 10.1.0.3:8301 alive   server  0.6.4   2         foobar
fbservice4 10.1.0.4:8301 alive   server  0.6.4   2         foobar
```

The output should report that all the services nodes are members and *alive*.

Remove `dns::server::virtual_relay` and `install_server_ip` parameters from `$ADMIN/hpc-privatedata/hiera`

```
--- a/hpc-privatedata/hieradata/foobar/roles/cluster.yaml
+++ b/hpc-privatedata/hieradata/foobar/roles/cluster.yaml
@@ -225,8 +225,3 @@
- # Static IP addresses of the generic service nodes on the management network
- 'domain-name-servers':      '10.2.0.1, 10.2.0.2, 10.2.0.3, 10.2.0.4'
- 'broadcast':                "%{hiera('net::management::broadcast')}"
-
- dns::server::virtual_relay: false
- install_server_ip: '10.1.0.1' # static IP address of the temporary
-                               # installation node on the administration
-                               # network
```

With this new configuration, Bind DNS server relays all DNS requests on the *virtual* zone to Consul DNS interface.

Push and the apply the new configuration:

```
# hpc-config-push && clush -bg service hpc-config-apply -v
```

Finally, check DNS requests on virtual zone are managed by Consul with:

```
# dig +short http.service.virtual
10.1.0.4
10.1.0.2
10.1.0.3
```

The output must report multiple generic service nodes static IP addresses in random order.

3.5.8 Temporary installation node sweep

Since the beginning of the installation process, the temporary installation node hosts installation files and services required to install the other generic service nodes. Now, all the other generic service nodes host the same files and services. Finally, the temporary installation node must be re-installed to be strictly identical to the other generic service nodes in terms of configuration.

Note

The disks of the temporary installation node are going to be formatted and all data hosted of this node will be lost. Then, it is probably time to backup all the manual modifications realized on this node and push all modifications in the remote internal configuration Git repository.

Reboot the node in PXE mode through its BMC:

```
# export BMC=bmcfbservice1
# ipmitool -I lanplus -U ADMIN -P ADMIN -H $BMC chassis bootdev pxe
# ipmitool -I lanplus -U ADMIN -P ADMIN -H $BMC power reset
```

Wait for the network installation to proceed and the node to reboot on the system freshly installed on its disks.

3.6 Admin node

Once the Service nodes are fully configured (Ceph, DNS, Consul, DHCP, TFTP, HTTP for boot...), the cluster is able to reinstall any physical or virtual machine with load-balancing and high-availability.

The first other node to install is the admin node, the central point of the HPC cluster administration.

3.6.1 Base system

Add the *admin* role by creating the file `$ADMIN/hpc-privatedata/hieradata/$CLUSTER/roles/admin.yaml` with the following content:

```
profiles:
  # common
  - profiles::cluster::common
  - profiles::systemd::base
  - profiles::ssmtp::client
  - profiles::network::base
  - profiles::dns::client
  - profiles::access::base
  - profiles::openssh::server
  - profiles::openssh::client
  - profiles::environment::base
  - profiles::environment::limits
  - profiles::environment::service
  - profiles::log::client
  # HW host
  - profiles::hardware::ipmi
  - profiles::hardware::admin_tuning
  # admin
  - profiles::hpcconfig::push
  - profiles::hpcconfig::apply
  - profiles::ntp::client
  - profiles::openssh::client_identities
  - profiles::clush::client
  - profiles::consul::client
  - profiles::conman::client
  - profiles::clara::base
  - profiles::ceph::client
  - profiles::s3::s3cmd
  - profiles::jobsched::client
```

```
profiles::network::gw_connect: 'wan'
```

```
profiles::environment::service::packages:
- scibian-hpc-admin
```

The profiles listed after the *admin* comment carry the software required on the admin node. The `profiles::environment` has a specific value for this role in order to install the admin meta-package.

Append the node definition in the `master_network` hash, for example:

```
master_network:
[...]
```

```
fbadmin1:
  fqdn: "fbadmin1.#{hiera('domain')}}"
  networks:
    administration:
      'DHCP_MAC': 'aa:bb:cc:dd:ee:08'
      'IP':       '10.1.0.10'
      'device':   'eth0'
      'hostname': 'fbadmin1'
    management:
      'IP':       '10.2.0.10'
      'device':   'eth1'
      'hostname': 'mgtfbadmin1'
    lowlatency:
      'IP':       '10.4.0.10'
      'device':   'ib0'
      'hostname': 'opafbadmin1'
    bmc:
      'DHCP_MAC': 'aa:bb:cc:dd:ee:09'
      'IP':       '10.2.0.110'
      'hostname': 'bmcfbadmin1'
    wan:
      'IP':       '10.2.0.10'
      'device':   'eth2'
      'hostname': 'wanfbadmin1'
```

Adjust the node boot parameters in the `boot_params` hash, for example:

```
boot_params:
[...]
```

```
fbadmin1:
  boot_os:          'scibian8_disk'
  boot_dev:         'eth0'
  console:          'ttyS0,115200n8'
  ipxebin:          'ipxe_serial.bin'
```

Synchronize SSH host keys:

```
# puppet-hpc/scripts/sync-ssh-hostkeys.sh hpc-privatedata $CLUSTER
```

Push and apply the new configuration:

```
# hpc-config-push && clush -bg service hpc-config-apply -v
```

And reboot the node in PXE mode to proceed the network installation:

```
# export BMC=bmcfbadmin1
# ipmitool -I lanplus -U ADMIN -P ADMIN -H $BMC chassis bootdev pxe
# ipmitool -I lanplus -U ADMIN -P ADMIN -H $BMC power reset
```

Wait for the network installation to proceed. Once the installation is over, the node reboot on its freshly installed system on its disks and it becomes available through SSH. Starting from this point, all the following operations of the installation process are realized from this admin node.

3.6.2 Administration environment

The administration environment must be re-created following the same instructions given in the temporary installation node `administration environment` section.

The Clara utility is available on the admin node. Its ipmi plugin can be configured with this small snippet added with eyaml to the cluster specific layer of the hiera repository:

```
##### Clara #####

clara::ipmi_options:
  command: "%{hiera('cluster_prefix')}}%{my_conman_server}1"
  prefix: 'bmc'

clara::password_options:
  ASUPASSWD: "%{hiera('cluster_decrypt_password')}}"
  IMMUSER: 'ADMIN'
  IMPASSWORD: DEC::PKCS7[<password>!]
```

Push and apply configuration on the admin node:

```
# hpc-config-push && hpc-config-apply -v
```

Then, the clara ipmi plugin can be used as explained in its documentation (`man clara-ipmi (1)`).

3.7 Service virtual machines

On Scibian HPC clusters, the additional services are hosted inside virtual machines for more flexibility and better resources partitioning. These service virtual machines run on the generic service nodes. On the generic services nodes, the virtual machines are managed by Libvirt service. The distributed instances of Libvirt are controlled centrally from the admin node with Clara utility. The following sub-sections explain how to setup these software components.

3.7.1 Libvirt settings

The Libvirt service must create various virtual networks to connect the virtual machines to the HPC cluster and a storage pool on Ceph RDB interface to store the virtual disks of the virtual machines. These virtual resources are setup with the following configuration in the cluster specific layer of the hiera repository:

```
virt_ceph_uuid: '<uuid>'

profiles::virt::networks:
  'administration':
    'mode': 'bridge'
    'interface': 'br0'
  'management':
    'mode': 'bridge'
    'interface': 'br1'
  'wan':
    'mode': 'bridge'
    'interface': 'br2'

profiles::virt::secrets:
  'client.libvirt':
    'type': 'ceph'
    'uuid': "%{hiera('virt_ceph_uuid')}}"
    'value': DEC::PKCS7[<key>!]
```

```
profiles::virt::pools:
  'rbd-pool':
    'type': 'rbd'
    'hosts':
```



```

- 'fbservice2'
- 'fbservice3'
- 'fbservice4'
'auth':
  'type':      'ceph'
  'username':  'libvirt'
  'uuid':      "%{hiera('virt_ceph_uuid')}}"

```

In this example, the following values must be replaced:

- <key> is given by the following command:

```
# ceph auth get-key client.libvirt'
```

- <uuid> is an arbitrary UUID¹ to identify uniquely the secret. For example, it can be generated with this command:

```
# python -c 'import uuid; print uuid.uuid1()'
```

Push and apply configuration on the generic service nodes:

```
# hpc-config-push && clush -bg service hpc-config-apply
```

3.7.2 Clara configuration

Clara has dedicated configuration for its *virt* plugin. This configuration is set with the following two hashes in the cluster specific layer of the hiera repository:

```

clara::virt_options:
  'nodegroup:default':
    'default':      'true'
    'nodes':        'fbservice1,fbservice2,fbservice3,fbservice4'
  'pool:default':
    'default':      'false'
  'pool:rbd-pool':
    'default':      'true'
  'vol_pattern':
    'default':      '{vm_name}_{vol_role}'
  'template:default':
    'default':      'true'
    'xml':          'domain_default_template.xml'
    'vol_roles':    'system'
    'vol_role_system_capacity': '60000000000'
    'networks':     'administration'

```

```

clara::virt_tpl_hpc_files:
  '/etc/clara/templates/vm/domain_default_template.xml':
    source: "%{hiera('private_files_dir')}/virt/domain_default_template.xml"

```

The `clara::virt_options` hash notably specifies the list of generic services nodes that hosts the virtual machines and the domain templates and parameters associated to each service virtual machine. For the moment, only the default domain template and parameters are set. The second hash `clara::virt_tpl_hpc_files` defines the templates of Libvirt XML domains definitions. In this example, there is one default domain XML template for all virtual machines which should be fine for most Scibian HPC clusters.

The domain XML template must be located in `$ADMIN/hpc-privatedata/files/$CLUSTER/virt/domain_default_t`. Here is a full example of this file:

```

<domain type='kvm'>
  <name>{{ name }}</name>
  <memory unit='KiB'>{{ memory_kib }}</memory>
  <currentMemory unit='KiB'>{{ memory_kib }}</currentMemory>
  <vcpu placement='static'>{{ core_count }}</vcpu>
  <resource>

```

¹Universally Unique Identifier, a 128-bit number used to identify information in computer systems

```

    <partition>/machine</partition>
</resource>
<os>
  <type arch='x86_64' machine='pc-i440fx-2.1'>hvm</type>
  <bootmenu enable='yes' />
  <boot dev='hd' />
  <boot dev='network' />
</os>
<features>
  <acpi />
  <apic />
  <pae />
</features>
<cpu mode='custom' match='exact'> </cpu>
<clock offset='utc'>
  <timer name='rtc' tickpolicy='catchup' />
  <timer name='pit' tickpolicy='delay' />
  <timer name='hpet' present='no' />
</clock>
<on_poweroff>destroy</on_poweroff>
<on_reboot>restart</on_reboot>
<on_crash>restart</on_crash>
<pm>
  <suspend-to-mem enabled='no' />
  <suspend-to-disk enabled='no' />
</pm>
<devices>
  <emulator>/usr/bin/kvm</emulator>
  <disk type='network' device='disk'>
    <source protocol='rbd' name='{ volumes.system.path }'>
      <host name='<ip_mon_server_1' /> />
      <host name='<ip_mon_server_2' /> />
      <host name='<ip_mon_server_3' /> />
    </source>
    <auth username='libvirt'>
      <secret type='ceph' uuid='<uuid' />
    </auth>
    <target dev='vda' bus='virtio' />
    <alias name='virtio-disk0' />
  </disk>
  <disk type='block' device='cdrom'>
    <driver name='qemu' type='raw' />
    <backingStore />
    <target dev='hda' bus='ide' />
    <readonly />
    <alias name='ide0-0-0' />
  </disk>
  <controller type='usb' index='0' model='ich9-ehci1'>
    <alias name='usb0' />
  </controller>
  <controller type='usb' index='0' model='ich9-uhci1'>
    <alias name='usb0' />
    <master startport='0' />
  </controller>
  <controller type='usb' index='0' model='ich9-uhci2'>
    <alias name='usb0' />
    <master startport='2' />
  </controller>
  <controller type='usb' index='0' model='ich9-uhci3'>
    <alias name='usb0' />
    <master startport='4' />
  </controller>
  <controller type='pci' index='0' model='pci-root'>
    <alias name='pci.0' />
  </controller>

```

```

<controller type='ide' index='0'>
  <alias name='ide0' />
</controller>
<controller type='virtio-serial' index='0'>
  <alias name='virtio-serial0' />
</controller>
{% for network_name, network in networks.iteritems() %}
<interface type='network'>
  <mac address='{{ network.mac_address }}' />
  <source network='{{ network_name }}' />
  <model type='virtio' />
</interface>
{% endfor %}
<!--
<serial type='tcp'>
  <source mode='bind' host='{{ serial_tcp_host }}' service='{{ serial_tcp_port }}' />
  <protocol type='telnet' />
  <target port='0' />
  <alias name='serial0' />
</serial>
-->
<serial type='pty'>
  <target port='0' />
  <alias name='serial0' />
</serial>
<channel type='spicevmc'>
  <target type='virtio' name='com.redhat.spice.0' />
</channel>
<input type='tablet' bus='usb'>
  <alias name='input0' />
</input>
<input type='mouse' bus='ps2' />
<input type='keyboard' bus='ps2' />
<graphics type='spice' port='5901' autoport='yes' listen='127.0.0.1'>
  <listen type='address' address='127.0.0.1' />
</graphics>
<sound model='ich6'>
  <alias name='sound0' />
</sound>
<video>
  <model type='qxl' ram='65536' vram='65536' heads='1' />
  <alias name='video0' />
</video>
<redirdev bus='usb' type='spicevmc'>
  <alias name='redir0' />
</redirdev>
<redirdev bus='usb' type='spicevmc'>
  <alias name='redir1' />
</redirdev>
<redirdev bus='usb' type='spicevmc'>
  <alias name='redir2' />
</redirdev>
<redirdev bus='usb' type='spicevmc'>
  <alias name='redir3' />
</redirdev>
<memballoon model='virtio'>
  <alias name='balloon0' />
</memballoon>
<rng model='virtio'>
  <backend model='random'>/dev/random</backend>
  <alias name='rng0' />
</rng>
</devices>
</domain>

```

In this example, the following values must be replaced:

- `<ip_mon_server_*>` are the static IP addresses of the Ceph MON servers on the administration network.
- `<uuid>` is the UUID for Libvirt Ceph RBD secret generated in the previous sub-section.

Deploy these new settings by pushing and applying the configuration on the admin node:

```
# hpc-config-push && hpc-config-apply -v
```

3.7.3 Virtual machine definitions

Now that Libvirt and Clara virt plugin are properly setup, the various service virtual machines can be defined. The steps to define the service virtual machines are mostly generic and common to all of them. As an example for this documentation, the two service virtual machines `fbdoe[1-2]` will be defined.

The first step is to define the `boot_params` of the virtual machines in `$ADMIN/hpc-privatedata/hieradata/$CLUSTER`

```
boot_params:
  [...]
  fbdoe[1-2]:
    boot_os:          'scibian8_disk'
    ipxebin:         'ipxe_noserial.bin'
```

Eventually, in the same file, an additional domain template and parameters association can be appended to the `clara::virt_options` for these new virtual machines, if the default domain parameters are not appropriate:

```
clara::virt_options:
  [...]
  'template:proxy':
    'vm_names':          'fbdoe[1-2]'
    'xml':               'domain_default_template.xml'
    'vol_roles':        'system'
    'vol_role_system_capacity': '60000000000'
    'networks':         'administration,wan'
    'core_count':       '16'
    'memory_kib':       '16777216'
```

In this example, the following settings are overridden from the defaults:

- the virtual block storage device has a size of 60GB,
- 2 network devices attached to the *administration* and *wan* networks,
- 16 virtual CPU cores,
- 16GB of RAM.

Then, the new role *doe* must be defined in file `$ADMIN/hpc-privatedata/hieradata/$CLUSTER/roles/doe.yaml` with all the appropriate profiles.

Push and apply configuration on admin node:

```
# hpc-config-push && hpc-config-apply -v
```

Extract MAC address of the virtual machine on the administration network:

```
# clara virt getmacs <VM>
```

Then add the network settings of the virtual machines in the `master_network` hash with their MAC addresses:

```
master_network:
  fbdoe1:
    fqdn: "fbdoe1.#{hiera('domain')}}"
    networks:
      administration:
```

```

    'DHCP_MAC': 'aa:bb:cc:dd:ee:0a'
    'IP':       '10.1.0.11'
    'device':   'eth0'
    'hostname': 'fbdoe1'
  wan:
    'IP':       '10.3.0.11'
    'device':   'eth1'
    'hostname': 'wanfbdoe1'
fbdoe2:
  fqdn: "fbdoe2.#{hiera('domain')}}"
  networks:
    administration:
      'DHCP_MAC': 'aa:bb:cc:dd:ee:0b'
      'IP':       '10.1.0.12'
      'device':   'eth0'
      'hostname': 'fbdoe2'
    wan:
      'IP':       '10.3.0.12'
      'device':   'eth1'
      'hostname': 'wanfbdoe2'

```

Eventually, virtual IP addresses can also be defined for the virtual machines in the `vips` hash of the same file.

Generate the SSH host keys in synchronization with the `master_network`:

```
# puppet-hpc/scripts/sync-ssh-hostkeys.sh hpc-privatedata $CLUSTER
```

Push and apply the new configuration on the generic service nodes:

```
# hpc-config-push && clush -bg service hpc-config-apply -v
```

Define the new virtual machines with Clara on the generic service node of your choice, for example `fbSERVICE1`:

```
# clara virt define fbdoe[1-2] --host=fbSERVICE1
```

Note

The choice of the generic service node is not critical as the service virtual machines can be migrated from one generic service node to another at any time.

Then start the virtual machine by wiping its virtual block storage devices and boot in PXE mode:

```
# clara virt start fbdoe[1-2] --wipe
```

Eventually, watch the serial console with:

```
# ssh -t fbSERVICE1 -- virsh console fbdoe1
```

3.7.4 Required virtual machines

You are free to define the service virtual machines you want on Scibian HPC clusters. The service virtual machines can run any software services you would like. However, some specific generic virtual machines are required in the reference architecture to run some mandatory additional services.

The required service virtual machines are:

- two **proxy** virtual machines with the `auth::replica` profile for managing the LDAP directory replica. The installation of the LDAP directory replica of the *proxy* nodes is documented in the *Directory replica* sub-section of the *LDAP Authentication* section of this installation procedure.
- two **batch** virtual machines with the `jobsched::server` and `db::server` profiles for Slurm controller, SlurmDBD accounting service and MariaDB galera database. The installation of the Slurm server-side components on the *batch* nodes is documented in the *Slurm* section of this installation guide.

- two **p2p** virtual machines with the `p2p::seeder`, `p2p::tracker` and `http::diskless` profiles for serving files to boot diskless nodes with Bittorrent. The installation of the *p2p* nodes is pretty straightforward as long as the required profiles are enabled. The creation of the diskless environment is documented in the *Build diskless image* sub-section of the *Frontend and compute nodes* section of the installation procedure.

3.8 LDAP Authentication

3.8.1 Directory replica

User authentication on Scibian HPC clusters is based on LDAP directory using `ldaps` protocol (LDAP over SSL/TLS). This protocol requires the LDAP replica to have valid SSL certificate and asymmetric keys.

For production use, it is recommended to obtain a certificate signed by a valid PKI CA², either a public CA on the Internet or a CA internal to your organization. Otherwise, it is possible to use self-signed certificates.

Copy the private key and the certificate under the following paths:

- certificate: `$ADMIN/hpc-privatedata/files/$CLUSTER/auth/$CLUSTER_ldap.crt`
- private key: `$ADMIN/hpc-privatedata/files/$CLUSTER/auth/$CLUSTER_ldap.key`

Encrypt these files with `clara enc` plugin:

```
# clara enc '$ADMIN/hpc-privatedata/files/$CLUSTER/auth/$CLUSTER_ldap.crt'
# clara enc '$ADMIN/hpc-privatedata/files/$CLUSTER/auth/$CLUSTER_ldap.key'
```

Remove the unencrypted files:

```
# rm '$ADMIN/hpc-privatedata/files/$CLUSTER/auth/$CLUSTER_ldap.crt'
# rm '$ADMIN/hpc-privatedata/files/$CLUSTER/auth/$CLUSTER_ldap.key'
```

Then, append the `auth::replica` profile and set certificate owner to `openldap` in the `proxy` role:

```
--- a/hieradata/foobar/roles/proxy.yaml
+++ b/hieradata/foobar/roles/proxy.yaml
@@ -14,7 +14,7 @@ profiles:
  # Proxy
  - profiles::ntp::client
  - profiles::network::wan_nat
+ - profiles::auth::replica
  - profiles::postfix::relay
  - profiles::ha::base
  - profiles::hardware::admin_tuning
@@ -30,3 +30,24 @@ profiles:

  profiles::network::gw_connect: 'wan'
  shorewall::ip_forwarding:     true
+
+certificates::certificates_owner: 'openldap'
```

Push and apply the configuration on the proxy nodes:

```
# hpc-config-push && clush -bg proxy hpc-config-apply -v
```

Finally, follow the steps documented in *LDAP bootstrap procedure*.

3.8.2 Clients setup

Once the LDAP replica are bootstrapped and operational, it is possible to setup NSS LDAP backend and PAM LDAP authentication on the nodes.

²Public Key Infrastructure Certificate of Authority, an entity that issues digital certificates

On Scibian HPC clusters, NSS LDAP backend and PAM authentication over LDAP are both setup with the same `auth::client` profile. This profile must be used in combination with the `access::base` profile. This profile controls the remote access rules to the nodes. By default, the profile prevents remote access to the nodes with LDAP accounts. The access rules must explicitly whitelist users and/or administrators to allow remote access with SSH.

There are two main access whitelist parameters:

- `profiles::access:base_options` is the list of permanent access rules.
- `profiles::access:production_options` is the list of access rules disabled in maintenance mode.

The administrators related access rules must be listed in the `base_options` while the users related access rules must only be present in the `production_options` list. This way, only administrators can access the HPC cluster in maintenance mode. For example:

```
profiles::access::base_options:
- "+ : (admins) : ALL"
profiles::access::production_options:
- "+ : (grpusers1) : ALL"
- "+ : (grpusers2) : ALL"
```

These parameters must be set in the roles specific layer of the hiera repository as access rules depends on the role of the nodes. For example, users may access the frontend nodes but not the admin node.

Additionally, it is also possible to setup sudo rules with the `sudo::base` profile and the `sudo::sudo_config_opts` list. This parameter is basically a list of sudo rules. For example, to allow the group of administrator to sudo any command on the admin node, add the following excerpt to file `$ADMIN/hpc-privatedata/hieradata/$CLUSTER/ro`

```
profiles::sudo::sudo_config_opts:
- "%admins ALL = (ALL) ALL"
```

By default, the PAM and NSS LDAP backend connect to the HPC cluster internal LDAP replica. This replica is hosted by service virtual machine. In order to make LDAP authentication on the admin nodes and generic service nodes possible for the administrators when the virtual machines are offline (typically during maintenances), it is possible to add the following parameter in the associated roles:

```
profiles::auth::client::external_ldap: true
```

This way, the nodes will connect to the organization reference LDAP directory instead of the internal LDAP replica.

Push and apply the configuration on all the affected nodes with:

```
# hpc-config-push && clush -bg all hpc-config-apply
```

3.9 Slurm

Slurm workload manager is distributed among the HPC cluster nodes with multiple daemons and clients software. On Scibian HPC clusters, the server part of Slurm, *ie.* the controller and the accounting services, run in high-availability mode on the *batch* nodes. These components are managed by the `jobsched::server`. The *batch* nodes also need the `db::server`, and the `ceph::client` or `nfs::mount` that respectively setup the MariaDB galera RDBMS³, and CephFS or NFS filesystem client.

3.9.1 Base Configuration

Slurm communications between nodes are secured using Munge which is based on a secret shared key. Generate this munge key with the following command:

```
# mkdir -p $ADMIN/hpc-privatedata/files/$CLUSTER/munge
# dd if=/dev/urandom bs=1 count=1024 > $ADMIN/hpc-privatedata/files/$CLUSTER/munge/munge.key
```

³Relational Database Management System

Encrypt the key using Clara:

```
# clara enc encode $ADMIN/hpc-privatedata/files/$CLUSTER/munge/munge.key
```

Remove the unencrypted key:

```
# rm $ADMIN/hpc-privatedata/files/$CLUSTER/munge/munge.key
```

Setup the nodes and partitions managed by Slurm in the `slurm::partitions_options` hash in the cluster specific layer of the Hiera repository. For example:

```
slurm::partitions_options:
  - 'NodeName=fbcn[01-04] Sockets=2 CoresPerSocket=14 RealMemory=64000 State=UNKNOWN'
  - 'NodeName=fbgn01 Sockets=2 CoresPerSocket=4 RealMemory=64000 Gres=gpu:k80:2 State=UNKNOWN'
  - 'PartitionName=cn Nodes=fbcn[01-04] Default=YES MaxTime=INFINITE State=UP'
  - 'PartitionName=gn Nodes=fbgn01 MaxTime=INFINITE State=UP'
  - 'PartitionName=all Nodes=fbcn[01-04],fbgn01 MaxTime=INFINITE State=UP'
```

Please refer to [Slurm documentation](#) for more details about these settings.

In the same, setup the LDAP/SlurmDBD users synchronization utility, for example:

```
profiles::jobsched::server::sync_options:
  main:
    cluster: "%{hiera('cluster_name')}}"
    org:      "%{hiera('org')}}"
    policy:   'global_account'
  global_account:
    name:     'users'
    desc:     'Main users account'
```

Please refer to the [example configuration file](#) for more details.

3.9.2 Shared State Location

Still in the cluster specific layer of the Hiera repository, setup the shared storage directory.

CephFS

If you are using CephFS, configure the client mount with the following excerpt:

```
profiles::jobsched::server::ceph::keys:
  client:
    key: "%{hiera('ceph_client_admin_key')}}"

profiles::jobsched::server::ceph::mounts:
  slurmctld:
    servers: # list of Ceph MON servers
      - fb-service2
      - fb-service3
      - fb-service4
    device:    '/slurmctld'
    mountpoint: "%{hiera('slurm_state_save_loc')}}"
    user:      'admin'
    key:       'client'
    mode:      'kernel'
```

NFS

If you are using an NFS HA Server:


```
profiles::jobsched::server::ceph::enabled: false

profiles::jobsched::slurm_config_options:
  [...]
  StateSaveLocation:      '/admin/restricted/backup/slurm_state_save'
```

For NFS HA, at the role level, configure the NFS mount:

```
profiles:
  [...]
  - profiles::nfs::mounts

profiles::nfs::to_mount:
  home:
    server:      'fbnas'
    exportdir:   '/srv/admin'
    mountpoint:  '/admin'
    options:     'bg,rw,hard,vers=4'
```

3.9.3 Miscellaneous Tuning

Eventually, it is possible to tune Slurm, GRES, SlurmDBD, job submit LUA script with the following parameters:

```
profiles::jobsched::slurm_config_options:
  PrivateData:      'jobs,reservations,usage'
  AccountingStorageEnforce: 'associations,limits,qos'
  GresTypes:        'gpu'
  SlurmCtldDebug:   'verbose'
  PriorityFlags:     'FAIR_TREE'

slurm::gres_options:
  - 'nodeName=fbgn01 Name=gpu Type=k80 File=/dev/nvidia0'

profiles::jobsched::server::slurmdbd_config_options:
  PrivateData: 'accounts,jobs,reservations,usage,users'

slurm::ctld::submit_lua_options:
  CORES_PER_NODE: '28'
```

3.9.4 MariaDB security hardening

By default, the MariaDB server is setup with parameters to harden its security. Notably, the following settings are deployed by default:

- `max_user_connections` to 100 (default is 0 ie. unlimited), in order to prevent one user from grabbing all 151 available `max_connections` (default MariaDB value).
- `secure_file_priv` is set to an empty value in order to disable potentially dangerous command `LOAD DATA INFILE`.
- the client histfile `/.mysql_history` is disabled by default.

Obviously, these settings can be altered in the hiera repository. Here is an example yaml excerpt to change these default values:

```
mariadb::disable_histfile: false
mariadb::galera_conf_options:
  mysql:
    max_user_connections: '0' # unlimited
    secure_file_priv:     ''
```

3.9.5 Bootstrap

Some software components need to be manually bootstrapped on the *batch* nodes before being started:

- MariaDB database
- SlurmDBD service

The shared storage can be on CephFS or on NFS HA, the suitable bootstrap procedure must be performed:

- CephFS filesystem
- NFS HA filesystem

Please refer to the [Bootstrap procedure](#) chapter of this document for all details.

3.9.6 Configuration deployment

Once the configuration is set in the Hiera repository, push and apply the configuration on the *admin* and *batch* nodes:

```
# hpc-config-push && clush -bg admin,batch hpc-config-apply -v
```

Check Slurm is available by running the `sinfo` command on the admin node. If the command report the nodes and partitions state without error, Slurm is properly running.

3.10 Frontend and compute nodes

On Scibian HPC clusters, the frontend and compute nodes download at boot time a system image in deployed in RAM which notably gives possibility to have diskless nodes. For more details about this technique, please refer to the [Diskless boot](#) sub-section in the *Advanced Topics* section of the *Architecture* chapter of this document. The diskless image must be generated with Clara *images* plugin on the *admin* node before booting the frontend and the compute nodes. These steps are explained in the following sub-sections.

3.10.1 Diskless image generation

The diskless image is generated by the Clara *images* plugin. This plugin need some configuration in the cluster specific layer of the Hiera repository. Here is an example of such configuration:

```
clara_images_target_dir: "%{hiera('admin_dir')}/scibian8"

clara::common_options:
  allowed_distributions:
    value: 'scibian8'

clara::images_options:
  extra_packages_image: "scibian-archive-keyring,hpc-config-apply,scibian-hpc-commons"
  packages_initrd:      "scibian-diskless-initramfs-config"
  etc_hosts:            "10.1.0.101:vipfbSERVICE1,10.1.0.101:apt.service.virtual,10.1.0.10:fbadmin1"

clara::config_options:
  images-scibian8:
    debiandist:          'jessie'
    debmirror:          "http://%{hiera('debian_mirror_server')}/%{hiera('debian_mirror_dir')}"
    kver:                "3.16.0-4-amd64"
    list_repos:          "deb [arch=amd64,i386] http://%{hiera('debian_mirror_server')}/"
    trg_dir:             "%{hiera('clara_images_target_dir')}"
    trg_img:             "%{hiera('clara_images_target_dir')}/scibian8.squashfs"
```

```

preseed_file:           "%{hiera('clara_images_config_dir')}/scibian8/preseed"
package_file:          "%{hiera('clara_images_config_dir')}/scibian8/packages"
script_post_image_creation: "%{hiera('clara_images_config_dir')}/scibian8/post.sh"
list_files_to_install:  "%{hiera('clara_images_config_dir')}/scibian8/filelist"
dir_files_to_install:   "%{hiera('clara_images_config_dir')}/scibian8/files_dir"
foreign_archs:         'i386'

clara::live_dirs:
  "%{hiera('clara_images_config_dir')}":
    ensure: directory
  "%{hiera('clara_images_config_dir')}/scibian8":
    ensure: directory
  "%{hiera('clara_images_config_dir')}/scibian8/files_dir":
    ensure: directory

clara::live_files:
  "%{hiera('clara_images_config_dir')}/scibian8/post.sh":
    source: "%{hiera('private_files_dir')}/boot/live/scibian8/post.sh"
    mode: '755'
  "%{hiera('clara_images_config_dir')}/scibian8/preseed":
    source: "%{hiera('private_files_dir')}/boot/live/scibian8/preseed"
  "%{hiera('clara_images_config_dir')}/scibian8/filelist":
    source: "%{hiera('private_files_dir')}/boot/live/scibian8/filelist"
  "%{hiera('clara_images_config_dir')}/scibian8/files_dir/resolv.conf":
    source: "%{hiera('private_files_dir')}/boot/live/scibian8/files_dir/resolv.conf"
  "%{hiera('clara_images_config_dir')}/scibian8/files_dir/no-cache":
    source: "%{hiera('private_files_dir')}/boot/live/scibian8/files_dir/no-cache"
  "%{hiera('clara_images_config_dir')}/scibian8/files_dir/no-recommends":
    source: "%{hiera('private_files_dir')}/boot/live/scibian8/files_dir/no-recommends"
  "%{hiera('clara_images_config_dir')}/scibian8/files_dir/interfaces":
    source: "%{hiera('private_files_dir')}/boot/live/scibian8/files_dir/interfaces"
  "%{hiera('clara_images_config_dir')}/scibian8/files_dir/proxy":
    source: "%{hiera('private_files_dir')}/boot/live/scibian8/files_dir/proxy"
  "%{hiera('clara_images_config_dir')}/scibian8/files_dir/mk_ipmi_dev.sh":
    source: "%{hiera('private_files_dir')}/boot/live/scibian8/files_dir/mk_ipmi_dev.sh"
  "%{hiera('clara_images_config_dir')}/scibian8/files_dir/hpc-config.conf":
    source: "%{hiera('private_files_dir')}/boot/live/scibian8/files_dir/hpc-config.conf"

```

The `clara::live_files` parameter contains a list of files deployed under the configuration directory of Clara. Their files are:

- `$ADMIN/hpc-privatedata/files/$CLUSTER/boot/live/scibian8/post.sh` is a post-generation script run by Clara inside the image environment:

```

#!/bin/bash -e

# Fix Timezone data
echo GMT > /etc/timezone
dpkg-reconfigure -f noninteractive tzdata

# Fix hostname
echo "localhost" > /etc/hostname

# Create needed directory for Puppet
mkdir -p /var/lib/puppet/facts.d/

# Enable setuid on /bin/ping to let users run it because AUFS does not support
# xattr and therefore capabilities.
chmod 4755 /bin/ping

```

This script can notably be used to customize the image or set files and directories that are required very early in the live boot process before Puppet run.

- `$ADMIN/hpc-privatedata/files/$CLUSTER/boot/live/scibian8/preseed` contains the answers to the Debconf packages configuration questions:

```

console-common console-data/keymap/full select en
console-common console-data/keymap/policy select Select keymap from full list
console-data console-data/keymap/full select en
console-data console-data/keymap/policy select Select keymap from full list
console-setup console-setup/charmap47 select UTF-8
locales locales/default_environment_locale select en_US.UTF-8
locales locales/locales_to_be_generated multiselect en_US.UTF-8 UTF-8, en_US ISO-8859-1
keyboard-configuration keyboard-configuration/layout select English
keyboard-configuration keyboard-configuration/variant select English
keyboard-configuration keyboard-configuration/unsupported_layout boolean true
keyboard-configuration keyboard-configuration/model select International (with dead keys)
keyboard-configuration keyboard-configuration/layoutcode string intl
keyboard-configuration keyboard-configuration/ctrl_alt_bksp boolean false
keyboard-configuration keyboard-configuration/variantcode string oss
keyboard-configuration keyboard-configuration/modelcode string pc105
postfix postfix/main_mailer_type select No configuration
tzdata tzdata/Areas select Europe
tzdata tzdata/Zones/Europe select London
libpam-runtime libpam-runtime/conflicts error
mdadm mdadm/start_daemon boolean false
postfix postfix/mailname string localdomain

```

- `$ADMIN/hpc-privatedata/files/$CLUSTER/boot/live/scibian8/filelist` specifies the list of files to copy inside the generated image:

<code>hpc-config.conf</code>	<code>etc/</code>	0644
<code>resolv.conf</code>	<code>etc/</code>	0644
<code>proxy</code>	<code>etc/apt/apt.conf.d/</code>	0644
<code>no-cache</code>	<code>etc/apt/apt.conf.d/</code>	0644
<code>no-recommends</code>	<code>etc/apt/apt.conf.d/</code>	0644
<code>interfaces</code>	<code>etc/network/</code>	0644
<code>mk_ipmi_dev.sh</code>	<code>usr/local/sbin/</code>	0755

All the files under the `files_dir` directory are copied without modification into the image. The required files are:

- `$ADMIN/hpc-privatedata/files/$CLUSTER/boot/live/scibian8/files_dir/resolv.conf` is the configuration file for DNS solvers with the virtual IP addresses of the cluster's internal DNS servers:

```

domain foobar.hpc.example.org
search foobar.hpc.example.org hpc.example.org
nameserver 10.1.0.101
nameserver 10.1.0.102
nameserver 10.1.0.103
nameserver 10.1.0.104

```

- `$ADMIN/hpc-privatedata/files/$CLUSTER/boot/live/scibian8/files_dir/no-cache` disables packages local caching in APT package manager:

```

Dir::Cache::srcpkgcache "";
Dir::Cache::pkgcache "";

```

- `$ADMIN/hpc-privatedata/files/$CLUSTER/boot/live/scibian8/files_dir/no-recommends` disables *recommends* soft-dependencies installation in APT package manager:

```
APT::Install-Recommends "0";
```

- `$ADMIN/hpc-privatedata/files/$CLUSTER/boot/live/scibian8/files_dir/interfaces` is a default network interfaces configuration file to enable DHCP on `eth0` interface:

```

auto lo
iface lo inet loopback

auto eth0
iface eth0 inet dhcp

```

- `$ADMIN/hpc-privatedata/files/$CLUSTER/boot/live/scibian8/files_dir/proxy` setup cluster's internal packages proxy in APT configuration:

```
Acquire::http::Proxy "http://apt.service.virtual:3142";
```

- `$ADMIN/hpc-privatedata/files/$CLUSTER/boot/live/scibian8/files_dir/mk_ipmi_dev.sh` is a workaround script to create the BMC devices inodes the `/dev` virtual filesystem very early in the diskless nodes boot process:

```
#!/bin/sh

DEVICE='/dev/ipmi0'

if [ -e ${DEVICE} ]
then
  exit 0
else
  MAJOR=$(grep ipmidev /proc/devices | awk '{print $1}')
  mknod --mode=0600 ${DEVICE} c ${MAJOR} 0
fi
```

- `$ADMIN/hpc-privatedata/files/$CLUSTER/boot/live/scibian8/files_dir/hpc-config.conf` is a configuration file for Puppet-HPC `hpc-config-apply` utility:

```
[DEFAULT]
environment=production
source=http://s3-system.service.virtual:7480/hpc-config
keys_source=http://secret.service.virtual:1216
# Using /var/tmp to more easily manipulate /tmp mount
# point during a puppet run.
tmpdir=/var/tmp
```

Once all these files have been added to the cluster specific files directory, the Clara *p2p* plugin must be configured in the cluster specific layer of the Hiera repository with the following parameter:

```
clara::p2p_options:
  seeders:      "%{hiera('cluster_prefix')}%{my_p2p_tracker}1,%{hiera('cluster_prefix')}%{
    my_p2p_tracker}2:%{hiera('clara_images_target_dir')}/scibian8.squashfs.torrent"
  trackers:     "10.1.0.51,10.1.0.52:%{hiera('clara_images_target_dir')}/scibian8.squashfs
    .torrent"
  trackers_port: '6881'
  trackers_schema: 'http'
```

Push and apply the configuration on the admin node:

```
# hpc-config-push && hpc-config-apply
```

Now that Clara is setup, the image can be created with the following command:

```
# clara images create scibian8
```

Also create the associated initrd environment:

```
# clara images initrd scibian8
```

Deploy the generate image and initrd to the `_p2p` nodes with!

```
# clush -g p2p mkdir -p /var/www/diskless/scibian8
# clush -g p2p \
  --copy /var/cache/admin/scibian8/{initrd-3.16.0-4-amd64,vmlinuz-3.16.0-4-amd64} \
  --dest /var/www/diskless/scibian8
# clush -g p2p \
  --copy /var/cache/admin/scibian8/{scibian8.squashfs.torrent,scibian8.squashfs} \
  --dest /var/www/diskless/scibian8
```

Restart peer-to-peer services to load new files:

```
# clara p2p restart
```

The diskless environment is finally ready and available to frontend and compute nodes.

3.10.2 Boot nodes

Before booting the frontend and compute nodes, they must be declared in the internal configuration repository in the first place. Append the nodes to the `boot_params` hash in `$ADMIN/hpc-privatedata/hieradata/$CLUSTER`

```
boot_params:
  [...]
  fbfront[1-2]:
    cowsize:      '8G'
    boot_os:       'scibian8_ram'
    ipxebin:       'ipxe_noserial.bin'
  fbcn[01-04]:
    boot_os:       'scibian8_ram'
    ipxebin:       'ipxe_noserial.bin'
  fbgn01:
    cowsize:      '8G'
    boot_os:       'scibian8_ram'
    ipxebin:       'ipxe_noserial.bin'
```

The `cowsize` must be increased to 8GB from default 2GB on frontend and graphical nodes because these nodes need much more packages to be installed at boot time.

Then define the roles associated to the frontend and the compute nodes, for example `front`, `cn` and `gn`. For these roles definitions, keep in mind the following rules:

- The frontend role must include the `jobsched::client` while the compute nodes require the `jobsched::exec` profile instead.
- The `profiles::environment::userspace::packages` must include the `scibian-hpc-frontend` meta-package in the frontend nodes role, `scibian-hpc-compute` meta-package in the standard compute nodes and `scibian-hpc-graphical` meta-package on the graphical nodes.

The nodes must be added into the `master_network` hash in file `$ADMIN/hpc-privatedata/hieradata/$CLUSTER/net` with all their network interfaces and the MAC addresses of their network interface connected to the *administration* and their BMC.

Generate all the SSH host keys:

```
# puppet-hpc/scripts/sync-ssh-hostkeys.sh hpc-privatedata $CLUSTER
```

Push and apply the configuration to the admin and generic service nodes:

```
# hpc-config-push && clush -bg admin,service hpc-config-apply -v
```

Finally, boot all the nodes in PXE mode with Clara:

```
# clara ipmi pxe @front,@cn,@gn
# clara ipmi boot @front,@cn,@gn
```

3.11 Optional features

3.11.1 Tuning

tbd

3.11.2 Firewall

tbd

3.11.3 Kerberos

tbd

3.11.4 Internal APT repository

tbd

3.11.5 Storage Multipath

tbd

3.11.6 Monitoring

tbd

3.11.7 Metrics

tbd

3.11.8 HPCStats

tbd

3.11.9 Slurm WCKeys

tbd

3.11.10 NFS High-Availability

tbd

3.11.11 Slurm power management

Generate and encrypt the SSH key used to poweroff the nodes from the batch nodes:

```
# mkdir hpc-privatedata/files/$CLUSTER/pwmgmt
# ssh-keygen -N '' -C root@pwmgmt -f hpc-privatedata/files/$CLUSTER/pwmgmt/id_rsa_slurm
# clara enc encode hpc-privatedata/files/$CLUSTER/pwmgmt/id_rsa_slurm
# rm hpc-privatedata/files/$CLUSTER/pwmgmt/id_rsa_slurm
```

Then add those settings in the cluster specific layer of the hiera repository:

```
profiles::jobsched::pwmgmt::enabled: true
slurmutils::pwmgmt::ctld::config_options:
  ipmi:
    prefix:  "%{hiera('ipmi_prefix')}}"
    user:    "%{hiera('ipmi_user')}}"
    password: "%{hiera('ipmi_password')}}"
slurmutils::pwmgmt::ctld::priv_key_enc:  "%{hiera('private_files_dir')}/pwmgmt/id_rsa_slurm.
enc"
slurmutils::pwmgmt::ctld::decrypt_passwd: "%{hiera('cluster_decrypt_password')}}"

slurmutils::pwmgmt::exec::pub_key: <PUBKEY>
```

Where <PUBKEY> is the public key in file `hpc-privatedata/files/$CLUSTER/pwmgmt/id_rsa_slurm.pub`.

Finally, apply the new configuration on the batch nodes and all the compute nodes:

```
# hpc-config-push
# clush -bg batch hpc-config-apply -v
# clush -bg compute hpc-config-apply -v
```


Chapter 4

Bootstrap procedures

This chapter contains all the procedures to bootstrap all the crucial services for a Scibian HPC system: LDAP, Ceph, MariaDB with Galera, SlurmDBD, etc.

4.1 LDAP bootstrap

As stated in *external services* section of the Reference Architecture chapter, a central LDAP directory server external to the Scibian HPC cluster is required. The LDAP directory server on the cluster is just is a *replica* of this central external server.

The Puppet-HPC `openldap` module expects a LDIF file containing a full dump of the LDAP replica configuration. The easiest way to produce this bootstrap LDIF file is to install and configure an LDAP server replica manually and dump the live configuration.

First, install an LDAP server with common LDAP utilities:

```
# apt-get install slapd ldap-utils
```

Select the HDB database backend. Then, configure the base DN, the domain name, the organization name according to your environment, and set the administration password.

Write the LDAP replication configuration LDIF file `syncrepl_config.ldif`, similarly to this example:

```
dn: olcDatabase={1}hdb,cn=config
changetype: modify
add: olcSyncrepl
olcSyncrepl: rid=001 provider=<LDAP_SERVER_URL> bindmethod=simple timeout=0
  tls_cacert=<CA_CERT_CHAIN>
  network-timeout=0 binddn="<BIND_DN>" credentials="<BIND_PASSWORD>"
  searchbase="dc=calibre,dc=edf,dc=fr"
  schemachecking=on type=refreshAndPersist retry="60 +"
-
add: olcUpdaterref
olcUpdaterref: <LDAP_SERVER_URL>
```

Where:

- `LDAP_SERVER_URL` is the URL to the organization central LDAP server, ex: `ldaps://ldap.company.tld`.
- If using TLS/SSL, `CA_CERT_CHAIN` is the absolute path to the CA certificate chain (up-to root CA certificate), ex: `/usr/local/share/ca-certificates/ca-chain.crt`
- `BIND_DN` is the replication user DN, ex: `cn=replication,dc=company,dc=tld`
- `BIND_PASSWORD` is the password of the replication user

Inject this LDIF replication configuration file into the LDAP server:

```
# ldapmodify -a -Y EXTERNAL -H ldapi:// -f syncrepl_config.ldif
```

Using the same technique, configure to your needs the indexes, ACLs, TLS/SSL, password policy, kerberos, etc. Finally, generate the full LDAP config dump with:

```
# slapcat -b cn=config > config_replica.ldif
```

or:

```
# ldapsearch -Y EXTERNAL -H ldapi:/// -b cn=config > config-replica.ldif
```

The `config_replica.ldif` file must be deployed encrypted within Puppet-HPC private files directory. Please refer to Puppet-HPC Reference Documentation for more details.

After a fresh installation the cluster's services virtual machines that host the LDAP directory replicas, the `config_replica.ldif` is deployed by Puppet and the LDAP replication must be bootstrapped with this script:

```
# make_ldap_replica.sh
```

The script will ask you to confirm by typing YES and press enter.

4.2 MariaDB/Galera bootstrap

The Puppet-HPC `mariadb` module configures an active/active MariaDB cluster based on galera replication library. On the service virtual machines that host this database system, the corresponding `mariadb` system service will not start unless it is already started on another service virtual machine. If it is not running anywhere else, the service must be bootstrapped with this command:

```
# galera_new_cluster
```

This command starts the MariaDB service on the local host in *new cluster* mode. The state of the local service can be checked with this command:

```
# systemctl status mariadb.service
```

This command must report on running `mysqld` process. In some case, typically when a MariaDB/Galera was not properly stopped, the command may fail and report this error:

```
[ERROR] WSREP: It may not be safe to bootstrap the cluster from this node. It
was not the last one to leave the cluster and may not contain all the updates.
To force cluster bootstrap with this node, edit the grastate.dat file manually
and set safe_to_bootstrap to 1 .
```

In this case, and if you are totally sure that MariaDB service is stopped on all nodes, the error can be ignored with the following command:

```
# sed -i 's/safe_to_bootstrap: 0/safe_to_bootstrap: 1/' /var/lib/mysql/grastate.dat
```

Then, the MariaDB/Galera cluster can be started again with `galera_new_cluster`.

Once the service is started on all service virtual machines, you can check the cluster replication status with:

```
# mysql -e "SELECT VARIABLE_VALUE as cluster_size \
FROM INFORMATION_SCHEMA.GLOBAL_STATUS \
WHERE VARIABLE_NAME='wsrep_cluster_size'"
```

This result must be the number of expected active nodes in the MariaDB/Galera cluster (ex: 2).

4.3 SlurmDBD bootstrap

After its first installation on the cluster, the SlurmDBD accounting database is empty. First, the cluster must be created in the database:

```
# sacctmgr --immediate add cluster <name>
```

Where <name> is the name of the cluster.

Then, once the `sync-accounts` utility is configured, run it to create all accounts and users:

```
# slurm-sync-accounts
```

Then, it is possible to create `QOS` and configure `fair-share` depending upon your needs.

If using wckey, they must be bootstrapped by adding the first key manually using the `sacctmgr` command and then run the importation script:

```
# sacctmgr -i add user root wckey=<init>
# slurm_wckey_setup.sh
```

4.4 Ceph

4.4.1 Mon

After the reinstallation of one of the generic service nodes with a mon, it must be re-initialized. This procedure only works on a running cluster, the initial mon creation uses another command.

From an **admin** node:

```
# cd <ceph deploy directory>
# ceph-deploy --overwrite-conf mon add <mon hostname>
```

4.4.2 OSD

This procedure only applies if the content of an OSD volume is lost. If the node is reinstalled without erasing the content of the OSD volume, the configuration in puppet will be enough to start the osd volume again.

The relevant OSD ID can be retrieved with:

```
# ceph osd tree
```

Before doing this procedure, make sure the OSD is really down and not mounted on the OSD node.

Removing old OSD

The old OSD must be removed from the configuration (stored by the MON).

```
# ceph osd crush remove osd.X
# ceph auth del osd.X
# ceph osd rm X
```

Re-creating the OSD

```
# cd <ceph deploy directory>
# ceph-deploy osd prepare clserviceY:sdb
# ceph-deploy disk zap clserviceY:sdb
```

The OSD id and authentication key should be updated on the hiera configuration. In most cases, the new OSD will take the same ID as the old one. You can get the new ID and the new key with:

```
# ceph osd tree
# ceph auth print-key osd.X
```

4.4.3 CephFS

CephFS filesystem is used between the batch nodes to shared Slurm controller state. The filesystem must be initialized before being used by Slurm.

First, mount temporarily the CephFS filesystem:

```
# mount -t ceph -o name=admin,secretfile=/etc/ceph/client.key fbsevice2,fbsevice3,fbsevice4:/ /mnt
```

Create a subdirectory for Slurm controller, set its ownership and restrict its mode:

```
# mkdir /mnt/slurmctld
# chown slurm: /mnt/slurmctld
# chmod 0700 /mnt/slurmctld
```

Finally, umount it:

```
# umount /mnt
```

Puppet-HPC is now able to use this filesystem for Slurm on batch nodes.

4.5 NFS HA bootstrap

The shared storage of the NFS server contains a directory that holds the state of the clients (mainly the locks). When the shared NFS storage is created, it must be formatted and this state directory must be created.

The shared storage must be on a specific LVM Volume Group. What the PVs are for this volume group and how they are configured depends on the hardware available.

In the following example, the PV/LV is VG_NAS/LV_NAS and is to be mounted as /srv/admin.

```
# mkfs.ext4 /dev/VG_NAS/LV_NAS
# mkdir /srv/admin
# mount /dev/VG_NAS/LV_NAS /srv/admin
# mkdir -p /srv/admin/restricted/state/nfs/v4recovery
# umount /srv/admin
```

After these steps, the keepalived daemon can be started on the nodes. The MASTER node will mount the storage and export it.

Chapter 5

Production procedures

In this chapter are listed all the technical procedures to follow for regular operations occurring during the production phase of the supercomputer. This notably includes changing any encryption or authentication key, changing passwords, reinstalling nodes, etc.

5.1 MAC address change

This procedure explains how to modify the Puppet-HPC configuration to change an hardware Ethernet address after a motherboard replacement, for example.

First, the yaml file in the hieradata repository containing the `master_network` hash must be edited to replace the old hardware address. A description of this hash can be found in the Installation section of this guide.

The modified configuration must be pushed to the shared administration directory with the `hpc-config-push` command:

```
# hpc-config-push
INFO: creating archive /tmp/puppet-config-push/tmp_ndq0ujz/puppet-config-environment.tar.xz
INFO: S3 push: pushing data in bucket s3-system
```

Then apply the configuration on the service nodes, who runs the DHCP server:

```
# hpc-config-apply
```

Note

It is not possible to run the `hpc-config-apply` command on all the service nodes at the same time exactly. A short delay must be respected as the Ceph service can be disturbed by a restart of the network service.

5.2 Password/keys changes

5.2.1 Root password

The hashed root password is stored in the variable `profiles::cluster::root_password_hash` in yaml files. The value must be encrypted using `eyaml`. It can be simply changed using the `eyaml` command.

```
# eyaml edit cluster.yaml
...
profiles::cluster::root_password_hash: DEC::PKCS7[hashed_password]!
...
```

Once changed, the new configuration must be applied on all the machines of the cluster.

5.2.2 Root SSH key

The root SSH keys are stored in the internal repository. The private keys must be encrypted. The SSH public rsa key is also in the variable `openssh::server::root_public_key`. It is necessary to change the files and the value of the variable at the same time. To avoid connections problems, it is necessary to follow these steps in this order:

1. Change the keys files and the variable `openssh::server::root_public_key` in the internal repository
2. Apply the configuration on all the machines except the **admin** one
3. Apply the new configuration on the **admin** server.

Note

In case of desynchronization between the keys on the **admin** node and those on the others nodes, it is always possible to use the root password to connect.

5.2.3 SSH host keys

The SSH host keys are stored, encrypted, in the internal repository. To avoid connections problems, it is necessary to follow these steps in this order:

1. Change the keys files in the internal repository
2. Apply the configuration on all the machines of the cluster, including the **admin** machine
3. Delete the file `/root/.ssh/known_hosts` on the **admin** node.
4. When connecting to the nodes, `/root/.ssh/known_hosts` will be automatically populated if the Scibian HPC default configuration is used.

5.2.4 Eyaml keys

Replacing the eyaml PKCS7 key pair consist in reality of two actions:

1. Generate a new pair of keys (`eyaml createkeys`)
2. Replace all the values encoded with the old pair with ones encoded with the new pair of keys.

Note

As these operations implies decoding files and re-encoding them with another key pair, it is not possible to perform other administrative operations (like applying the configuration on nodes) on the cluster at the same time. The changing keys operation must be fully completed before resuming "normal" administrative operations.

These steps must be followed in order to safely change the eyaml keys:

Save the old keys:

```
# cp /etc/puppet/secure/keys/private_key.pkcs7.pem \
    /etc/puppet/secure/keys/private_key.pkcs7.pem.old
# cp /etc/puppet/secure/keys/public_key.pkcs7.pem \
    /etc/puppet/secure/keys/public_key.pkcs7.pem.old
```

Copy the new keys in `/etc/puppet/secure/keys/`.

Decrypt all the yaml files encoded using the old keys:

```
# eyaml decrypt \
  --pkcs7-private-key /etc/puppet/secure/keys/private_key.pkcs7.pem.old \
  --pkcs7-public-key /etc/puppet/secure/keys/public_key.pkcs7.pem.old \
  -e hieradata/<cluster>/cluster.yaml \
  > hieradata/<cluster>/cluster.decrypt.yaml
```

The `decrypt.yaml` contains all the secret in plain text. It should be removed as soon as possible.

Encrypt the files with the new keys:

```
# eyaml encrypt -e hieradata/<cluster>/cluster.decrypt.yaml \
> hieradata/<cluster>/cluster.yaml
# rm hieradata/<cluster>/cluster.decrypt.yaml
```

Remove the old saved keys from the **admin** node:

```
# rm /etc/puppet/secure/keys/private_key.pkcs7.pem.old \
/etc/puppet/secure/keys/public_key.pkcs7.pem.old
```

Create a tarball, encode it with `clara enc` and add it to the `files` directory of the internal repository:

```
# tar cJf /tmp/keys.tar.xz /etc/puppet/secure/keys
# clara enc encode /tmp/keys.tar.xz
# mv /tmp/keys.tar.xz.enc <internal repository>/files/<cluster>/eyaml
```

Where:

- `<internal repository>` is the directory that contains the clone of the internal repository.
- `<cluster>` is the name of the cluster.

At this stage, the keys are now stored encrypted in the internal repository and are available locally in the standard eyaml paths.

In the default Scibian-HPC configuration, the PKCS7 keys propagation service runs on all the generic service nodes. First, the encoded tarball must be manually copied on the nodes:

```
# scp <internal repository>/files/<cluster>/eyaml/keys.tar.xz <generic server X>:/tmp
```

Where `<generic server X>` is the hostname of the generic service node.

Then apply the configuration using the new keys:

```
# hpc-config-apply -vv --keys-source=/tmp
```

This will copy the eyaml PKCS7 key pair in the right directory to be serviced by the propagation service to all others nodes when applying the puppet configuration. These last two operations must be executed on all the generic service nodes.

Don't forget to remove the keys from the `/tmp` directory on the admin node and on all the service nodes.

```
# rm /tmp/keys.tar.xz
# clush -w @service rm /tmp/keys.tar.xz
```

5.2.5 Internal repository encoding key

Note

As these operations implies decrypting files and re-encrypting them with another key, it is not possible to perform other administrative operations (like applying the configuration on nodes) on the cluster at the same time. The changing key operation must be fully completed before resuming "normal" administrative operations.

Replacing the AES key used to encode files in the internal repository consist in several steps.

Generate a new AES key:

```
# openssl rand -base64 32
```

For each encoded file in the internal repository, it is necessary to decode it with the old key and re-encode it with the new one.

```
# clara enc decode <internal repository>/files/<cluster>/<filename>.enc
# openssl aes-256-cbc \
-in <internal repository>/files/<cluster>/<filename> \
-out <filename>.enc -k <AES KEY>
# rm <internal repository>/files/<cluster>/<filename>
```

Where:

- `<internal repository>` is the directory that contains the clone of the internal repository
- `<cluster>` is the name of the cluster
- `<filename>` is the path of the file to encode
- `<AES KEY>` is the random 256 bits key.

Using `clara` for both operations, `decode` and `encode`, is not possible as it support only one AES key.

This re-encryption step can be automated with the `reencode-file.sh` script in the `puppet-hpc` scripts dir:

```
# cd <internal repository>/files/<cluster>
# find -name "*.enc" \
  -exec <puppet-hpc path>/scripts/reencode-file.sh\
    /tmp/oldkey /tmp/newkey '{}' ';' ;'
```

The files `/tmp/oldkey` and `/tmp/newkey` are files with just the old and new AES key respectively. This script does not depend on `clara` but basically performs the same steps as above.

The AES key must be placed in `cluster_decrypt_password` in the cluster layer of the Hiera repository:

```
# eyaml edit hieradata/<cluster>/cluster.eyaml
```

Replace the key:

```
cluster_decrypt_password: DEC::PKCS7[<AES KEY>]!
```

Apply the new configuration on the **admin** node, to update `clara` configuration:

```
# hpc-config-apply
```

5.2.6 Replication account password

The steps to change these credentials are described here:

1. Decode the configuration `ldif` file:


```
# clara enc edit <internal repository>/files/<cluster>/<filename>.enc
```
2. The field to change is `olcSyncrepl:`, it contains all the necessary informations to connect to the master LDAP server (login, password, URI, etc ..)
3. Apply the new configuration on the **proxy** nodes.
4. Follow the LDAP bootstrap procedure as described in `LDAP bootstrap` on each **proxy** node. It is recommended to wait until the first `ldap` replicate is complete before attempting to update the second, to not disrupt authentication across the cluster.

Note

It is possible to change others values with this procedure, for example the root LDAP password.

5.2.7 Monitoring certificates

The certificates used for monitoring are stored, encrypted, in the internal repository in `<internal repository>/files/<cluster>/icinga2/certs/`. Each host has a certificate and a key. The steps to follow to change them are:

1. Change the key and certificate files in the internal repository
2. Apply the configuration on the concerned node
3. Update the certificate on the `icinga2` server

5.2.8 Munge key

Note

Scheduling service and jobs must be stopped to change the munge key.

Warning

This will kill running jobs.

1. Stop the `slurmd` and `slurmctld` daemons.
2. Stop the munge daemon on all nodes.
3. Encrypt the new key with Clara and place it in `<internal repository>/files/<cluster>/munge/munge.key.enc`
4. Apply the new configuration on all nodes.
5. restart the daemons.

5.2.9 Repo keyring**Note**

The packages must be saved in another place.

The cluster must use a private cluster keyring. This keyring is used to sign the local packages repository.

It is stored in the internal repository: `<internal repository>/files/<cluster>/repo/`

Here are the steps to follow to change it:

1. Generates a new keyring:


```
# LANG=C gpg --no-default-keyring \  
--keyring <internal repository>/files/<cluster>/repo/cluster_keyring.gpg \  
--secret-keyring <internal repository>/files/<cluster>/repo/cluster_keyring.secret.gpg \  
--gen-key
```
2. Encode the secret file with `clara encode`.
3. Apply the configuration on the **admin** node.
4. Delete the folder containing the local repository.
5. Re-create the repository with `clara`:


```
# clara repo key  
# clara repo init scibian8-hpc
```
6. Add the previously saved packages with `clara`:


```
# clara repo add scibian8-hpc mypackage_1-2.dsc  
# ...
```

5.2.10 MariaDB users

Generate passwords conform with your organization policy and edit the following parameters with `eyaml` in the hiera repository:

- `slurmdbd_slurm_db_password`
- `slurmdbd_slurmro_db_password`

These parameters correspond to the passwords of the MariaDB having respectively R/W and R/O grants on the SlurmDBD database.

Once modified, push and apply the configuration with the following commands:

```
# hpc-config-push && \  
clush --fanout=1 -bg batch hpc-config-apply -v
```

The `hpc-config-apply` command will perform the following steps, on each batch node:

- Update the passwords in the configuration file of the Slurm `mysql-setup` utility.
- Update the passwords in the MariaDB database
- Update SlurmDBD configuration (if R/W password changed)
- Restart SlurmDBD (if R/W password changed)

The `-fanout=1` parameter of the `clush` command makes sure the configuration is not applied simultaneously on both batch nodes. This could cause the SlurmDBD daemon to be restarted at the same time and make this service unavailable for a short period of time.

5.3 Administration node re-installation

This procedure will wipe the first disk of the admin node, if some customizations are not in the Puppet configuration, this should be handled separately.

Before, powering off the administration node, check that:

- There is an alternative route to connect to the service node (can be the service nodes themselves)
- It is possible to connect to the BMC IPMI, and especially to the Serial Over LAN console
- It is possible to connect to the Ethernet administration network switch

The administration node has no critical service in the reference architecture, so it can simply be powered off:

```
# poweroff
```

Note

In some Ethernet bonding setups, the node cannot do a PXE boot with an active bonding configuration on the Ethernet switch. If this is the case, refer to the documentation of the network switch to disable the bonding configuration.

To be re-installed, the administration node must perform a network boot. This can be configured with `ipmitool(1)` installed on a host that has access to the BMC network interface:

```
# ipmitool -I lanplus -H <bmc host> -U <bmc username> -P chassis bootdev pxe
# ipmitool -I lanplus -H <bmc host> -U <bmc username> -P chassis power on
```

Next steps will happen once the node is installed and has rebooted, the installation can be followed through serial console:

```
# ipmitool -I lanplus -H <bmc host> -U <bmc username> -P sol activate
```

Note

If the Ethernet switch configuration had to be modified to setup PXE boot, the modification must be reverted to its nominal status.

5.4 Service node re-installation

Before re-installing a Service node, active Virtual Machines on the nodes should be migrated away from the node. Clara can be used to list the active VMs and do the live migration.

Listing the VMs:

```
# clara virt list | grep clserviceX
```

Migrate the live VMs with the command:

```
# clara virt migrate <vmname> --dest-host clserviceY
```

These points should be checked before turning off a Service Node:

- The ceph cluster should be `HEALTH_OK` (`ceph health`), with at least three OSD in
- `consult` should return services as passing on at least three nodes
- On an Intel Omni-Path cluster, the `opafabricinfo` should return at least one Master and one Standby node

Once there is no VM remaining on the node, it can be powered off safely, the other Service node should ensure there is no service outage. The power off can be done from the node itself:

```
# poweroff
```

Note

In some Ethernet bonding setups, the node cannot do a PXE boot with an active bonding configuration on the Ethernet switch. If this is the case, refer to the documentation of the network switch to disable the bonding configuration.

To be re-installed, the service node must perform a network boot. This can be configured with **clara**:

```
# clara ipmi pxe clserviceX
# clara ipmi on clserviceX
```

Next steps will happen once the node is installed and as rebooted, the installation can be followed through serial console:

```
# clara ipmi connect clserviceX
```

After a Service node re-installation, the ceph services: OSD, MDS and RadosGW should be reconfigured automatically by the Puppet HPC configuration. The Mon service (not present on every node), must be boot-strapped again. This procedure is described with other *Ceph bootstrap procedures*.

In order to validate the generic service node re-installation, there are some relevant checks to perform.

- High-Speed network manager (Intel Omni-Path):

```
# opafrabricinfo
```

The reinstalled node must appear as a **master** or **standby** node.

- Check the ceph cluster is healthy:

```
# ceph status
```

The cluster should be HEALTH_OK with all OSDs, Mons and MDSs.

- Consul:

```
# consult
```

All services on all nodes should have the state passing.

Note

If the Ethernet switch configuration had to be modified to setup PXE boot, the modification must be reverted to its nominal status.

5.5 Frontend access

5.5.1 Draining

To perform a scheduled reboot of a frontend it is better to avoid new connection going to the frontend node that will be rebooted. The new connections are highly available and load balanced with IPVS.

It is possible to remove a frontend from the pool of node accepting new connections without killing active connections with the `ipvsadm` command by setting the weight of a node to 0.

To list the current weight, on a frontend:

```
# ipvsadm -ln
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port          Forward Weight ActiveConn InActConn
TCP 172.16.1.16:22 rr persistent 600
  -> 172.16.1.11:22                Route    1      10         0
  -> 172.16.1.12:22                Route    1       6         0
  -> 172.16.1.13:22                Route    1       1         0
  -> 172.16.1.14:22                Route    1      15         0
  -> 172.16.1.15:22                Route    1       1         0
```

To avoid a frontend node being attributed to new sessions, the weight of the node can be manually set to 0. This setting does not completely forbid new connection to go to the node, if a user already has a session, new session will go to the same node regardless of the weight. This setting also does not block connections made directly to the node and not the virtual IP address.

```
# ipvsadm -e -t 172.16.1.16:22 -r 172.16.1.11:22 -w 0
# ipvsadm -ln
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port          Forward Weight ActiveConn InActConn
TCP 172.16.1.16:22 rr persistent 600
  -> 172.16.1.11:22                Route    0      10         0
  -> 172.16.1.12:22                Route    1       6         0
  -> 172.16.1.13:22                Route    1       1         0
  -> 172.16.1.14:22                Route    1      15         0
  -> 172.16.1.15:22                Route    1       1         0
```

The modification can be reversed by setting the weight back to 1 manually.

```
# ipvsadm -e -t 172.16.1.16:22 -r 172.16.1.12:22 -w 1
```

5.6 NFS HA

5.6.1 Starting a node

When a node start is should not start the keepalived service automatically. This permits a failed node to be started without it becoming master with an remaining problem.

Before starting the **keepalived** service, the following conditions must be met:

- The **multipath-tools** service must be active with a running multipathd process.
- The **keepalived** service must be disabled

When these conditions are met, the service can be started:

```
# systemctl start keepalived
```

If the node is to become master (master node in the VIP configuration or other node is down), check that the first check goes well. It runs every minutes and logs are in `/var/log/user.log`. The message following message must appear:

```
Mar 17 17:19:01 fbnfs1 hpc_nfs_ha_server_check.sh.info: INFO: fbnfs1 All checks are OK
```

5.6.2 Manual Fail Over

If the master node disappears, because it is turned off or because the keepalived service is stopped, the failover will happen, but it will take a bit of time (a little more than a minute). This timeout can be entirely avoided by doing a manual failover of the master node before cutting the keepalived service.

To do this, the keepalived configuration must be changed manually on the node. Edit the file `/etc/keepalived/keepalived.conf`. Find the configuration for the NFS VIP and change the priority to 40, and the role to BACKUP. The service must be reloaded:

```
# service keepalived reload
```

The failover should happen quickly. Once the node failed over, stop the keepalived service:

```
# systemctl stop keepalived
```

The original configuration must be restored before starting the service again. This will happen if you launch a **hpc-config-apply** manually or if you reboot the node.

5.7 Services

This section contains usefull procedures for casual operations on infrastructure services.

5.7.1 Packages Caching purge

In order to invalidate and purge the packages caching service `apt-cacher-ng` cache content, run the following commands consecutively:

```
# clush -bg service systemctl stop apt-cacher-ng.service
# clush -bg service rm -rf /var/cache/apt-cacher-ng
# clush -bg service mkdir /var/cache/apt-cacher-ng
# clush -bg service chown apt-cacher-ng: /var/cache/apt-cacher-ng
# clush -bg service systemctl start apt-cacher-ng.service
```

5.8 Virtual Machines

This section contains procedure related with virtual machines management with `clara`.

5.8.1 Deleting a Virtual Machine

A Virtual Machine is composed of two mostly independant objects:

- The disk image
- The definition on a host

The two objects must be deleted separately.

The first step is to stop the Virtual Machine:

```
# clara virt stop <vm_name>
```

Once it is in the state `SHUTOFF` you can undefine it:

```
# clara virt undefine <vm_name>
```

The VM will still appear on `clara virt list` with the state: `MISSING`. It means `clara` still sees the disk image but not the Virtual Machine definition.

You can then proceed with deleting the disk image, by checking the full disk image name with `clara virt list -details`, you must find the volume name and the pool name.

On a physical host:

```
# virsh vol-delete --pool <pool_name> <volume_name>
```

On all other physical hosts:

```
# virsh pool-refresh <pool_name>
```