EDF HPC

Puppet-HPC reference documentation

Date: Pages: 38

Authors: CCN-HPC

Contents

Chapter 1

About this document

1.1 Purpose

This document contains a generic description of an HPC system in terms of its architectural views.

1.2 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
```

1.3 Build dependencies

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

- edf-doc-materials >= 2.0
- inkscape
- rubber
- texlive-latex-extra

1.4 License

1.5 Authors

In alphabetical order:

- Benoit BOCCARD
- Mehdi DOGGUY
- Thomas HAMEL
- Rémi PALANCHER
- Cécile YOSHIKAWA

Chapter 2

Overview

Puppet-HPC is a full Puppet-based software stack designed to easily deploy HPC clusters. Puppet is a popular open-source configuration management tool. The main goal of Puppet-HPC is to provide a common generic configuration management system that can be used effortlessly across multiple HPC clusters and organizations.

The Puppet-HPC software stack notably provides:

- Many generic Puppet modules for all technical components required on a HPC cluster.
- Defined data model for representing the description of an HPC cluster based on Hiera.
- Software patterns and code conventions conform to latest Puppet community standards.
- Tools to easily deploy and manage the configuration with high-scalability requirements.

The Puppet-HPC software architecture clearly separates code from data. This way, the code can be generic while the data can provide all specific contextual information. This has many advantages:

- The code base can be re-used and the development effort is shared.
- The same code is run on many different environments, it is therefore more tested and more reliable.
- The code can be easily tested on a small testing environment even if the data is different from the production environment.

All details about the software architecture of Puppet-HPC are documented in the Software Architecture chapter of this document.

Puppet-HPC is developed and actively maintained by the CCN-HPC (*Centre de Compétences Nationales en High Performance Computing*) of EDF (*Électrité de France*), one of the largest worldwide producers of electricity. The software is used to deploy and maintain the configuration of the largest HPC cluster in the company.

Puppet-HPC is open-source software and it is licensed under the terms of GPLv2+. Any external contribution is very welcome! It should be made under the form of a pull request or an issue creation on the project GitHub repository. Please refer to the Development Guidelines chapter for hints on doing awesome patches.

Chapter 3

Software architecture

3.1 Pattern

Puppet-HPC is based on the three following tools and principles:

- **facter** is used to report per-node facts. Moreover, some facts specific to the HPC context are used to convey the global information about the cluster that needs to be known when running Puppet on a node. These facts are implemented in the `hpclib` module.
- **hiera** is used to look up data. This tool helps separating site-specific or cluster-specific data from Puppet code. Specific data are excluded from Puppet-HPC, being kept, versioned and maintained in a separate internal repository. The cluster description it contains should follow certain rules though. These rules are detailed below in the Cluster Definition part.
- **The Roles/Profiles pattern** has been used to design the Puppet-HPC code. It is organized in different levels of abstraction:
  - Roles, which represent the business logic. A node includes one role, and one only. Each role lists one or more profiles.
  - Profiles, which represent the implementation of technical functionalities. A profile includes and manages modules to define a logical technical stack.
  - Modules, which are the elementary technical blocks. Modules should only be responsible for managing aspects of the component they are written for and should be as generic as possible.

Regarding the Roles/Profiles pattern, it is a common pattern in Puppet code organization. This pattern is explained in details in this presentation: https://puppet.com/presentations/designing-puppet-rolesprofiles-pattern

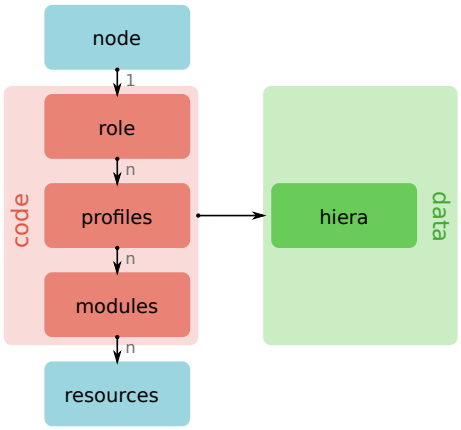

Figure 3.1: Code and data separation with roles and profiles pattern

One of the interesting aspects of the Roles/Profiles pattern is that modules should be as generic as possible. Whenever it is possible, **external community modules** should be used. They should come from a reliable source: distribution package or the Puppet Forge. In any case, external community modules should be properly reviewed.

3.2 Hiera layers

Hiera is a software designed to manage a repository of data formatted in key/value pairs. The key is the parameter name. The values can be of various types: strings, numbers, booleans, hashes or arrays. Puppet-HPC requires to use the default Hiera YAML backend, therefore the data is stored in YAML files.

Hiera is able to look up data out of a hierarchy - hence its name - of layers and manage overrides. This feature, combined with layers properly ordered by genericity levels, allows to define a maximum number of parameters once for multiple clusters and organization. The parameters are overridden in more specialized layers only when necessary. The following diagram illustrates the look up logic of a parameter `foo` into an example of a simplified hierarchy:

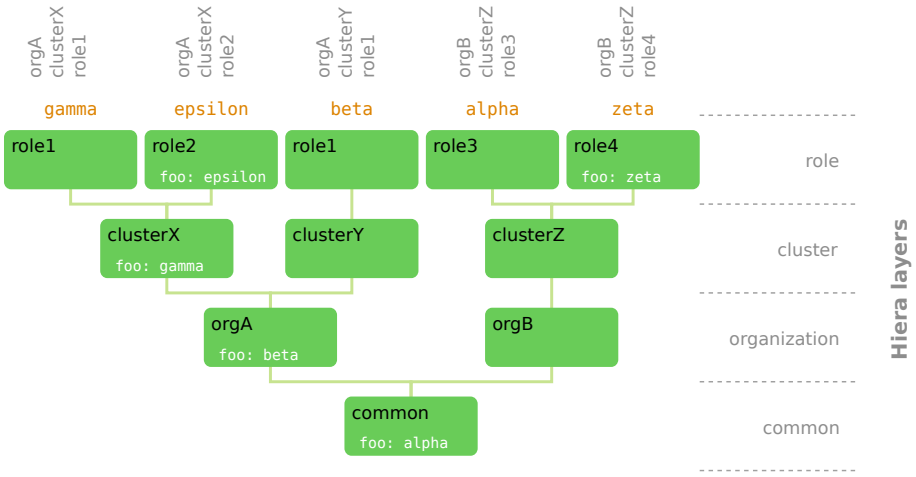

Figure 3.2: Simplified Hiera tree with specialization layers

The typical Hiera layers to use with Puppet-HPC are the following:

- `%cluster_name/roles/%puppet_role`
- `%cluster_name/cluster`
- `%cluster_name/network`
- `organization`
- `common`

The `common` layer is directly provided by Puppet-HPC with the YAML file `hieradata/common.yaml`. The upper layers are specific to an organization or a cluster, they must be defined in the internal repository as it's documented in the next section. Network data is separated in a specific file (`network.yaml`) only to keep the cluster YAML file readable.

The hierarchy (with all its layers) is setup in the `hiera.yaml` configuration file. An example of this file is provided with Puppet-HPC under the path `examples/privatedata/hiera.yaml`.

3.3 Internal repository

Puppet-HPC can not be used only on its own, it must be configured for a specific site and a specific cluster. It is recommended to work with an additional internal repository that will contain all specific data that can not be published on GitHub.

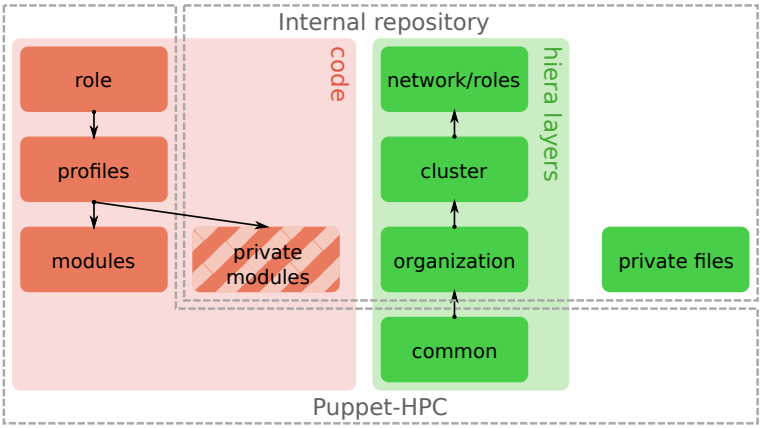

Figure 3.3: Combination of Puppet-HPC with an internal repository

The content and structure of this internal repository is explained below:

- `files` contains any configuration file that needs to be stored internally and that can be used as it is in a cluster configuration. It can be, for example , SSL certificates, SSH host keys, etc. If the content of these files is sensitive, they should be encrypted.
- `hieradata` contains all the site-specific and cluster-specific data necessary to configure a cluster with Puppet-HPC.
- `puppet-config` includes some Puppet configuration files to use with a specific cluster such as `puppet.conf`, `hiera.yaml`.

For each of these three directories, it is recommended to have a subdirectory per cluster being configured with Puppet-HPC.

3.4 External dependencies

Puppet-HPC provides a set of Puppet modules but also relies on some Puppet external community modules that can be considered as dependencies. The full list of these modules is:

- `puppetlabs-stdlib`
- `puppetlabs-concat`
- `puppetlabs-apache`
- `puppetlabs-apt`
- `arioch-keepalived`
- `herculesteam-augeasproviders-core`
- `puppet-archive`
- `puppet-collectd`
- `saz-rsyslog`
- `yo61-logrotate`

It is also strongly recommended to install the eyaml utility in order to encrypt sensitive data (such as passwords) inside the Hiera YAML files.

3.5 Genericity levels

The use of the Roles/Profiles pattern enables to control the level of genericity of each element of a Puppet configuration code base. Here are the genericity levels defined for all the components in the Puppet-HPC project:

- Roles: The code part of the roles is fully generic as it consists into one manifest (located under path `puppet-config/cluster/roles/manifests/init.pp`) which simply extract from Hiera the list of profiles included in the role. However, the name of the roles and the list of profiles are cluster specific due to technical properties of Puppet-HPC. This point is explained further in the roles section.
- Profiles are HPC specific and are highly related to the way Scibian HPC clusters are modeled. Nevertheless they can be reused from one cluster to the other. Their structure should follow the reference architecture defined in the Scibian HPC cluster installation guide.

- Modules are fully generic. They support multiple distributions and can even be used outside of the HPC clusters context when relevant.

As stated in the Hiera layers section, the Hiera repository is composed of multiple layers of datasets ordered by genericity levels. Then, each layer of the hierarchy has it own genericity level depending on its specialization.

This diagram gives a quick glance summary of the genericity levels for each element of the Puppet-HPC stack:

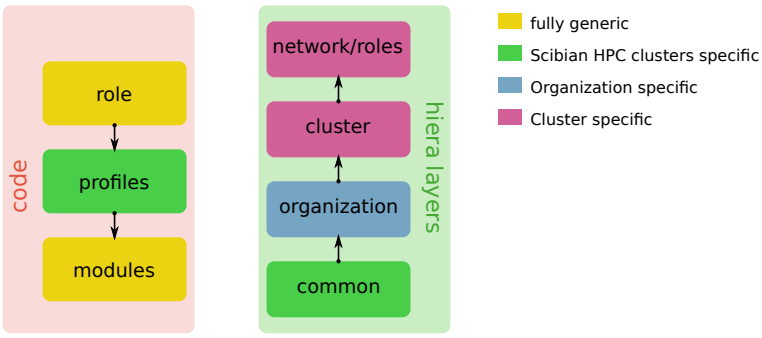

Figure 3.4: Stack components genericity goals

3.6 Roles

As previously stated in the pattern section, a node has exactly one role representing its *business logic*, a role being nothing more than a set of profiles. By design, all machines sharing the same role have the same set of profiles. In other words, if it is necessary to apply a different set of profiles to different machines, they must have different roles

In Puppet-HPC, the node-to-role association is set using a custom fact `puppet_role` provided by the `hpclib` module. The fact actually extracts the role name out of the hostname of the node, using the following pattern:

```
<prefix><role><id>
```

Where:

- `<prefix>` is the prefix of the cluster name (as provided by Hiera),
- `<role>` is the role name,
- `<id>` is a set of consecutive digits.

For example, on a cluster whose *prefix* is `foo`, the role names extracted from the following hostnames are:

- `foobar1`: `bar`
- `foocompute001`: `compute`
- `fooservice2boot001`: `service2boot`

This fact is then used in the Hiera layers definitions to get a role specific layer. The parameters values defined in this layer are specific to the nodes having this role. This role specific layer of the Hiera repository is primarily designed to set the list of profiles associated to the role, under the generic `profiles` parameter name. As example, here is a possible value of this parameter in file `<privatedata>/hieradata/<cluster_name>/roles/cn.yaml`:

```
profiles:
  - profiles::cluster::common
  - profiles::network::base
  - profiles::dns::client
  - profiles::access::base
  - profiles::ntp::client
  - profiles::ssmtp::client
  - profiles::jobsched::exec
  - profiles::openssh::server
  - profiles::environment::base
```

```
  - profiles::auth::client
  - profiles::metrics::collect_base
  - profiles::log::client
```

This parameter is extracted from the Hiera repository by the fully generic main manifest of roles module `puppet-config/cluster/roles/manifests/init.pp`:

```
class roles {
  hiera_include('profiles')
}
```

In order to work as expected, this mechanism has the following requirements:

- All nodes must follow this naming convention.
- The prefix of the cluster name must be set with `cluster_prefix` parameter at the cluster level in the Hiera repository.
- The `profiles` parameter must be defined at the role specific layer of the Hiera repository, for all the possible roles.

This diagram gives a summary of this node/role/profiles associations logic:

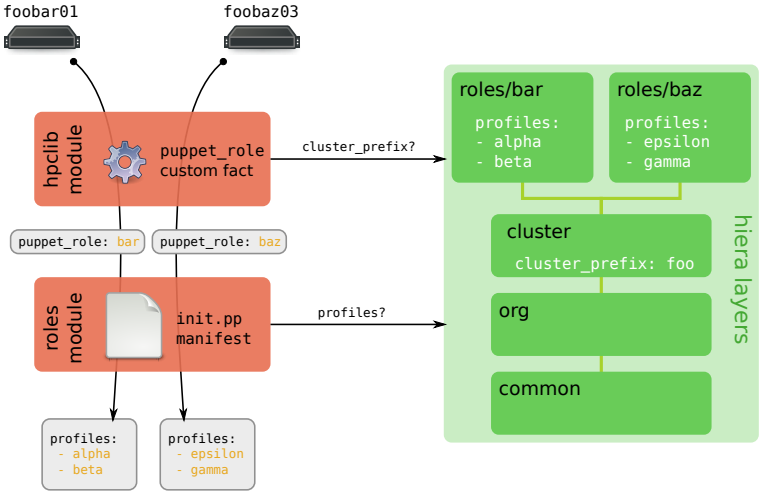

Figure 3.5: Node, role and profiles associations mechanism

3.7 Cluster definition

This cluster configuration is meant to be used with a standard cluster architecture, deviation from this architecture should be minimum. Some constraints are planned to be relaxed in the future.

Here, we are going to describe this architecture and how it should be defined to be used by the Puppet-HPC configuration.

3.7.1 Architecture

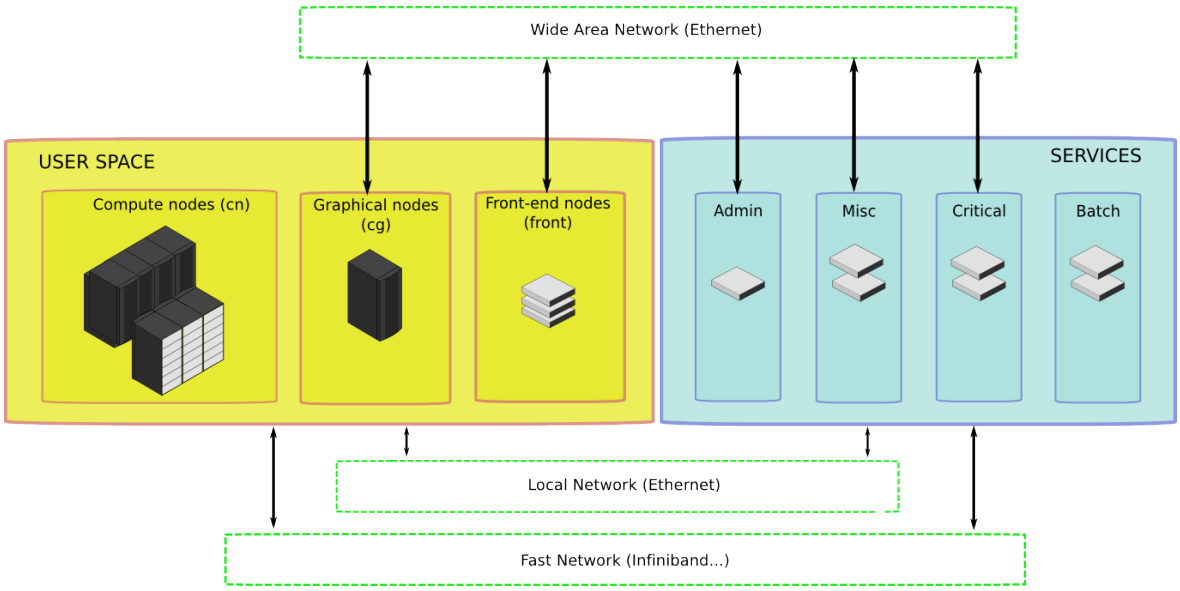

Figure 3.6:

As noted earlier, each machine is defined by its *role* (front, cn, batch, etc . . .). We can separate the machines into two groups: nodes with user access (frontends, compute nodes, etc ..) and nodes who provides services for the cluster, including storage. There's no particular distinction (except the *role*) between these two groups in the configuration.

3.7.2 Main shared parameters

Some parameters must be defined at the *cluster* level of the Hiera hierarchy. These variables are not associated to a unique profile and are aimed to be reused directly several times in Hiera or in puppet profiles.

- `cluster_name`: The complete name of the cluster. Can be used, for example, in the slurm configuration.
- `cluster_prefix`: The prefix used for all the hostnames in the cluster. Generally it will be composed of 2 or 3 letters ("gen" for a cluster named "generic", for example).
- `private_files_dir`: The directory where all the files copied by Puppet on the machines are stored. These files can be encrypted or not. It can be a shared directory between all the nodes, or an http export if the *hpclib::hpc_file* resource is used.
- `domain`: The domain name used across all the machines. Used in particular by the *bind* module.
- `user_groups`: Array of user groups authorized to connect and submit jobs to the cluster.
- `cluster_decrypt_password`: General password used by the *hpclib* module to decrypt encrypted files before copying them on the machines. This variable is usually itself encrypted using e-yaml.

3.7.3 Network definitions

Topology

The network topology is defined at the *cluster* level of the Hiera hierarchy. This means it is common to all nodes.

```
## Network topology of the cluster
net::administration::ipnetwork: '172.16.0.0'
net::administration::netmask: '255.255.248.0'
net::administration::prefix_length: '/21'
net::administration::broadcast: '172.16.7.255'
net::lowlatency::ipnetwork: '172.16.40.0'
net::lowlatency::prefix_length: '/21'
net::management::ipnetwork: '172.16.80.0'
```

```
net::management::netmask: '255.255.240.0'
net::management::broadcast: '172.16.95.255'
net_topology:
    'wan':
        'name':           'WAN'
        'prefixes':       'wan'
        'ipnetwork':      '172.17.0.0.0'
        'netmask':        '255.255.255.0'
        'prefix_length':  '/24'
        'gateway':        '172.17.0.1'
        'broadcast':      '172.17.0.255'
        'ip_range_start': '172.17.0.1'
        'ip_range_end':   '172.17.0.254'
        'firewall_zone':  'wan'
    'administration':
        'name':           'ADM'
        'prefixes':       ''
        'ipnetwork':      '172.16.0.0'
        'netmask':        '255.255.248.0'
        'prefix_length':  '/21'
        'gateway':        '172.16.0.1'
        'broadcast':      '172.16.7.255'
        'ip_range_start': '172.16.0.1'
        'ip_range_end':   '172.16.7.254'
        'firewall_zone':  'clstr'
        'pool0':
            'ip_range_start':
                          '172.16.0.1'
            'ip_range_end':
                          '172.16.5.254'
        'pool1':          # IP reserved for the discovery process
            'ip_range_start':
                          '172.16.6.1'
            'ip_range_end':
                          '172.16.7.254'
    'lowlatency':
        'name':           'IB'
        'prefixes':       'ib'
        'ipnetwork':      '172.16.40.0'
        'netmask':        '255.255.248.0'
        'prefix_length':  '/21'
        'gateway':         ''
        'broadcast':       '172.16.47.255'
        'ip_range_start':  '172.16.40.1'
        'ip_range_end':    '172.16.47.254'
        'firewall_zone':   'clstr'
    'management':
        'name':            'MGT'
        'prefixes':        'mgt'
        'ipnetwork':       '172.16.80.0'
        'netmask':         '255.255.240.0'
        'prefix_length':   '/20'
        'gateway':         ''
        'broadcast':       '172.16.95.255'
        'ip_range_start':  '172.16.80.1'
        'ip_range_end':    '172.16.95.254'
        'firewall_zone':   'clstr'
    'bmc':
        'name':            'BMC'
        'prefixes':        'bmc'
        'ipnetwork':       '172.16.80.0'
        'netmask':         '255.255.248.0'
        'prefix_length':   '/21'
        'gateway':         ''
        'broadcast':       '172.16.87.255'
```

```
        'ip_range_start':  '172.16.80.1'
        'ip_range_end':    '172.16.87.254'
        'firewall_zone':   'clstr'
```

The `bmc` network connects all the management cards (bmc, imm, etc ...). The `management` network connects the servers who must access these management devices. That is the reason why they share an IP networks settings and ranges.

Bonding

Some network interfaces are bonded together for load balancing and high availability. The bonding definition is done in Hiera. If the bonding is uniform (i.e. same bond interface on same slave interfaces) between nodes, this can be done at the *cluster* level. In case of differences between nodes, it must be done higher in the hierarchy (*role* or *node*).

```
network::bonding_options:
    'bond0':
      'slaves':
         - 'eth0'
         - 'eth1'
      'options':  'mode=active-backup primary=eth0'
    'bond1':
      'slaves':
         - 'eth2'
         - 'eth3'
      'options':  'mode=active-backup primary=eth2'
```

This variable from Hiera uses *Auto Lookup* to be passed to the `network` class.

Bridges

When using a machine as a physical host for VMs, it is often necessary to setup bridge interfaces. These bridge interfaces will be configured in the `master_network` hash with the right IP addresses. The physical device will be added automatically without an IP address. It is possible to create a bridge above a bonding interface.

```
network::bridge_options:
    'br0':
      'ports':
         - 'bond0'
      'description': 'Administration network bridge.'
    'br2':
      'ports':
         - 'bond2'
      'description': 'WAN network bridge'
```

3.7.4 Node definitions

Master Network

Nodes are listed in a Hiera hash called `master_network`. It must be defined at the at the *cluster* level of the Hiera hierarchy, but for readability reasons it is in a separate `network.yaml` file. Each key defines one node and its network configuration. Each node is described by a hash containing its fully qualified hostname and the networks attached to it. Each network must have a name corresponding to the ones used in the `net_topology` hash described in the topology section.

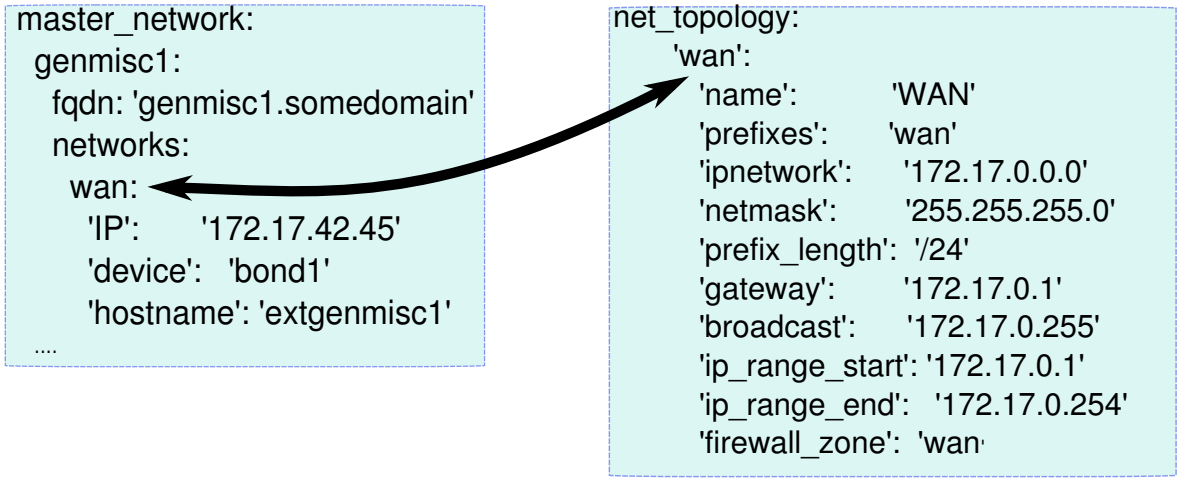

Figure 3.7:

These values can be defined for each network:

- MAC address (`DHCP_MAC`): The MAC address of the physical device connected to the network. It is used to build the dhcpd server configuration.
- Interface device (`device`): The device where the configuration must be applied, this means that with a bonded interface, the configuration must be applied on the `bond` interface. So it is not necessary that the physical interface is attached to the MAC address quoted above. The interfaces enslaved to the `bond` interfaces can be omitted from this configuration.
- Hostname (`hostname`): The hostname of the machine on the considered network.
- IPv4 Address (`IP`): The IPv4 address of the machine on the considered network. The netmask comes from the `net_topology` variable.
- External config (or not) : External configuration means the interface is configured on the system but should not be setup by the Puppet-HPC configuration. It is useful if another subsystem sets up the network interface: VPN, libvirt... On Debian, it means the interface is not added to `/etc/network/interfaces`. This boolean can take the value `true` or `false` and is considered `false` if omitted

Example:

```
master_network:
  genmisc1:
    fqdn: 'genmisc1.somedomain'
    networks:
      administration:
        'DHCP_MAC': '52:54:00:ba:9d:ac'
        'IP':       '172.16.2.21'
        'device':   'bond0'
        'hostname': 'genmisc1'
      lowlatency:
        'IP':       '172.16.42.21'
        'device':   'ib0'
        'hostname': 'llgenmisc1'
      management:
        'IP':       '172.16.88.21'
        'device':   'bond0'
        'hostname': 'mgtgenmisc1'
      bmc:
        'DHCP_MAC': '40:F2:E9:CD:53:CE'
        'IP':       '172.16.82.21'
        'hostname': 'bmcgenmisc1'
      wan:
        'IP':       '172.17.42.45'
        'device':   'bond1'
        'hostname': 'extgenmisc1'
```

This example defines one node (`genmisc1`) with the following configuration:

- DHCP
  - `52:54:00:ba:9d:ac genmisc1 172.16.2.21`
  - `40:F2:E9:CD:53:CE mgtgenmisc1 172.16.82.21`
- Network configuration on the node
  - `bond0 172.16.2.21 255.255.248.0` External Config: `false`
  - `bond0 172.16.88.21 255.255.248.0` External Config: `false`
  - `bond1 172.17.42.45 255.255.255.0` External Config: `false`
- DNS and Hosts
  - `genmisc1 172.16.2.21`
  - `extgenmisc1 172.17.42.45`

All lists are optional, so it is possible to define an element that just defines a Host/DNS configuration (for virtual IP addresses for instance):

```
master_network:
  genmisc:
    fqdn: 'genmisc.somedomain'
    networks:
      administration:
        'IP':       '172.16.2.20'
        'hostname': 'genmisc'
      management:
        'IP':       '172.16.82.20'
        'hostname': 'mgtgenmisc'
      wan:
        'IP':       '172.17.42.44'
        'hostname': 'extgenmisc'
```

3.8 Deployment

Scibian clusters use a simple *puppet apply* command with a set of modules, manifests and data. Puppet-HPC is not designed to work in a traditional "Puppet server" environment, as it must be used in a very simple system environment, like a post installation script inside Debian Installer.

3.8.1 Push and apply scripts

Two tools have been developed in order to apply the puppet-hpc configuration on the nodes of a cluster. One of the tools, "push" the entire configuration (modules, Hiera data, files) in a shared space, and another one is aimed to apply the configuration on the nodes.

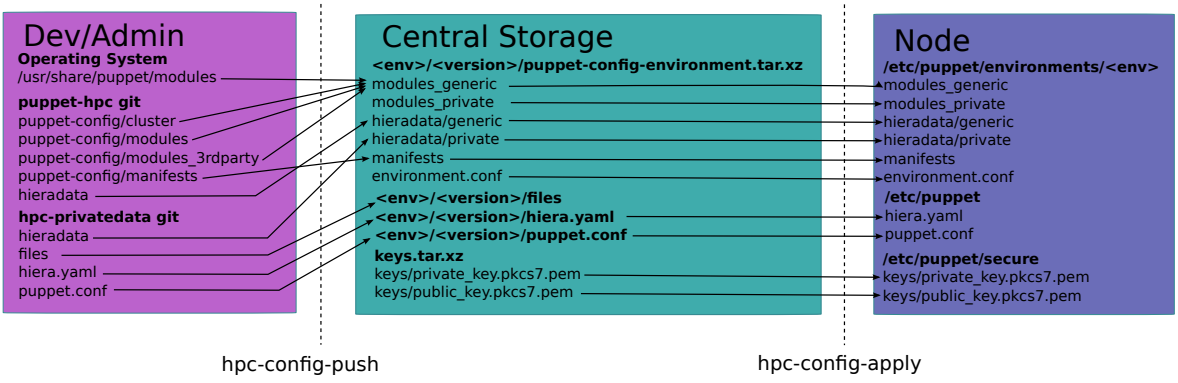

Figure 3.8: How the push and apply scripts work

hpc-config-push

This script merge all the data necessary to apply the puppet configuration in one archive and push it into a shared storage:

- Puppet modules installed in operating system via packages
- Puppet modules from the *puppet_hpc* git repository, including profiles
- Optionally other puppet modules
- Configuration files for Puppet and Hiera
- YAML files for Hiera: generic ones from puppet-hpc git repository and specific ones from the internal repository
- Files to copy on nodes from the internal repository

Two methods can be used to push the data: * **posix**: simply copies the tarball into a shared directory on all nodes (a nfs export, for example) * **s3**: uses the Amazon S3 RESTful API to send data on a compatible storage (Ceph Rados Gateway, for example)

The script can manage several Puppet environments with the `-e, -environment` parameter. A default environment can be defined in the configuration file.

The file `/etc/hpc-config/push.conf` allows to configure all the options for this script.

Please refer to `hpc-config-push(1)` man page for full usage documentation.

hpc-config-apply

This downloads the Puppet configuration (modules and hieradata) as a tarball and installs it as an environment in `/etc/puppet`. Private data files are not downloaded with the configuration, instead they are available from the central storage and are directly downloaded from the Puppet modules. If eyaml is used, this script needs a source to download the keys used to encrypt and decrypt data.

The command `puppet apply` is executed afterward with the environment previously untarred.

The configuration file indicating where to download the files is located in */etc/hpc-config.conf*.

Please refer to `hpc-config-apply(1)` man page for full usage documentation.

3.8.2 Packages

These two scripts are provided in the Scibian distribution as Debian packages:

- `hpc-config-apply`
- `hpc-config-push`

Chapter 4

Development Guidelines

This chapter gives some guidelines to help contributing to Puppet-HPC source code development. Additionally to these guidelines, this chapter also set some rules to follow, in order to make sure the code base stay consistent in the long term.

Puppetlabs, the company who maintain Puppet software, provides a reference style guide available online at this URL: https://docs.puppet.com/guides/style_guide.html

Puppet-HPC source code must respect the conventions defined in this reference style guide. All the guidelines defined in this chapter aim to be complementary to this reference style guide. In case of conflict between the 2 documents, the rules published in the reference style guide take precedence over the rules defined in this chapter.

All portions of code that do not fully respect those rules must be considered as bugs and must be tracked as such.

4.1 Main rules

There are few goals and principles that rule the overall architecture of Puppet-HPC code base:

- **Wise genericity**: respect genericity goals but stay practical.
- **Simple profiles**: only parameters that could not be defined in other layers of the stack.
- **Minimized hieradata**: only useful parameters defined at the right level.
- **Convention over configuration**: prefer clearly defined conventions and specifications rather than systematic configurability and overridability.

The following sections fully explain in details how to achieve those principles in every components of the stack.

4.2 Directories structure

The sources root directories contains the following sub-directories:

- `conf/`: examples configuration files of the scripts
- `debian/`: Debian packaging related files
- `doc/`: sources of the documentation
- `examples/`: examples of code for reference and learning purposes
- `hieradata/`: common down-most level of hieradata
- `init/`: init system configuration files for the scripts
- `puppet-config/`: all Puppet manifests
  - `cluster/`: definitions of profiles
  - `manifests/`: core manifests with nodes definitions
  - `modules/`: generic modules
- `scripts/`: deployment scripts

4.3 Language settings

The Puppet-HPC configuration uses the parser from Puppet < 4 (not the future parser). Modules must not use constructs that are only available with the parser from Puppet 4 (*ex:* `foreach`). Compatibility with the future parser is encouraged though.

It is assumed that the manifest will be applied with the following setting in `puppet.conf`:

```
stringify_facts=false
```

This setting permits facts to define advanced data structures such as hashes and arrays.

4.4 Hieradata

The hieradata is a database of parameters. As stated in the Hieradata levels section of the software architecture chapter, the hieradata is a stack of levels, each upper level being more specific to a smaller context. The parameters must always be defined at the lowest possible level of the stack (**ie.** the most generic). Obviously, if the parameter is defined in files below the cluster specific files, the parameter does not have to be duplicated from cluster to cluster and the cluster configuration is simpler.

There are multiple types of parameters in Hiera. In the first sub-section, the various types are defined. Then, all details and rules for each type are given.

4.4.1 Parameter types

The hieradata contains many parameters that can classified into four main categories:

- **shared parameters**: generic parameters used in several places accross hieradata, facts, functions and profiles.
- **simple parameters**: parameters dedicated to a module or a profile.
- **advanced parameters**: complex structures providing either a set of configuration settings or resources definitions to a profile.

The following schema represents how these various types of parameters can be used inside Puppet-HPC:

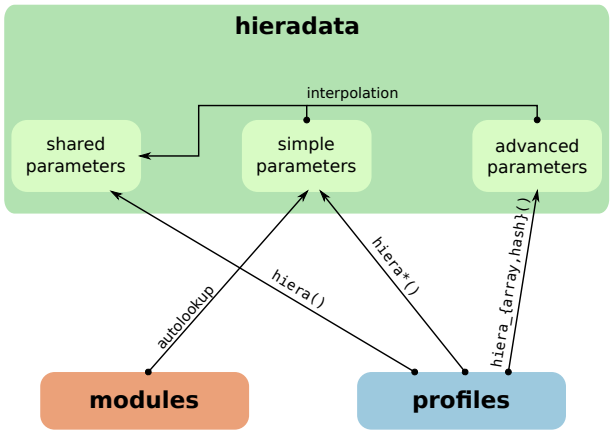

Figure 4.1: Parameters types and workflow

Full details are given in the following sub-sections.

4.4.2 Shared parameters

Shared parameters are generic parameters that are used several times in hieradata (using interpolation), custom facts, custom parser functions and profiles. The shared parameters should be used wisely. They should be used only where relevant to avoid clear duplication of data.

Their value must either be a string or an integer.

The shared parameters names can only contain letters and underscores. **Ex:**

- `timezone`
- `slurm_master`

All shared parameters required by Puppet-HPC must be defined in the common down most level of hieradata. Some shared parameter cannot have a sane default value at this level though, **eg.** the domain name. In this case, the value must clearly state it is wrong (**ex:** `FROM_COMMON_LVL_CHANGEME`). This way, users can easily spot them and clearly figure out they must be overridden in upper levels of their private hieradata.

4.4.3 Simple parameters

Strictly speaking, simple parameters are all parameters that are not of the other types of parameters (**ie**: shared parameters, configuration set or resource definition).

Their value can be of any type: string, integer, boolean, array or hash.

A simple parameter must either be:

- a module public class auto-lookup parameter,
- or a profile parameter.

A module parameter must not be imported by a profile.

Module parameters names are necessarily prefixed by the public class, this is requirement for hiera. For example, the parameter **opt** of the public class `soft::server` must be named `soft::server::opt`. Profiles parameters rules are given in the profiles section.

4.4.4 Advanced parameters

Advanced parameters provide either provide a set of configuration settings or resource definitions.

Their value must either be a hash or an array.

Configuration sets must be imported by profiles using the hiera function `hiera_array()` and `hiera_hash()`. Unlike the `hiera()` function or hiera autoloopkups, these functions can merge the elements coming from multiple levels of the hieradata. This is really convenient because all the configuration settings can be defined at their highest level of genericity in the hieradata stack. This way, most settings can be defined once for several clusters and the cluster specific hieradata levels only contain the cluster specific values.

This behaviour is illustrated in the following schema:

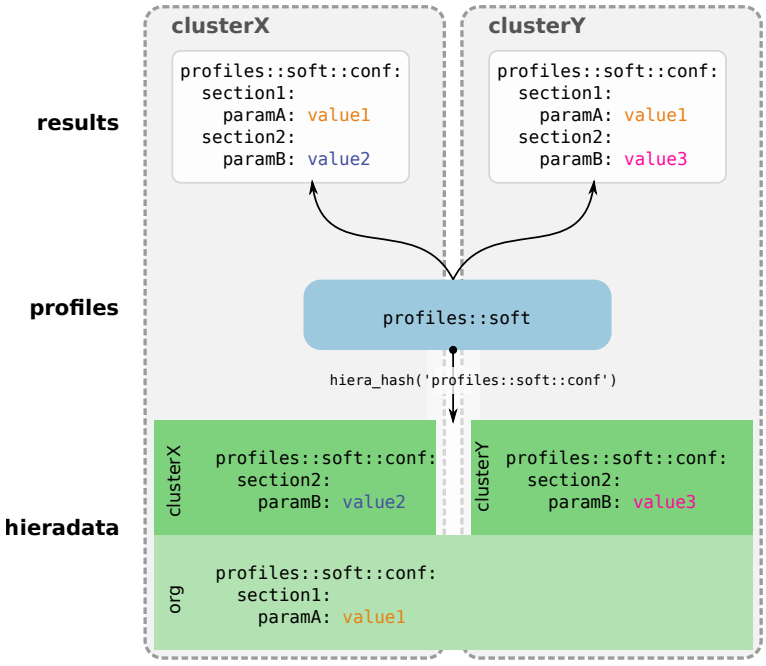

Figure 4.2: Merge behaviour of hiera advanced functions

4.4.5 Interpolation

Hiera support internal interpolation of parameters. This feature is really useful to help factorizing many settings. Its usage is definitely recommended in Puppet-HPC but it must be limited to the following parameters:

- Shared parameters,
- Standard facts,
- **hpclib** custom facts.

This implies it is not allowed to interpolate profile or module autoloopkup parameters. For example, this is considered safe:

```
shared_parameter: value
profiles::soft::param: "%{hiera('shared_parameter')}"
```

However, the following two examples are considered unsafe in Puppet-HPC:

```
soft::param: value
profiles::soft::param: "%{hiera('soft::param')}"

profiles::soft::param: value
soft::param: "%{hiera('profiles::soft::param')}"
```

4.5 Modules

4.5.1 Dependencies

Puppet-HPC internal modules must be fully autonomous and must not depend to any other module except on the stdlib external community module, hpclib and systemd internal modules. These modules do not manage real resources (excepting the `systemd` public class). They mostly provide a set of usefull functions, facts and defined types. Therefore, they can safely be considered as libraries for the other modules.

The special `hpc_*` internal modules are also exceptions: they can depend on one (and only one) other external community module (ex: `hpc_ha` depends on `keepalived`). Those modules are actually *wrappers* over other external community modules in order to give an high-level interface both more practical and specialized for the HPC clusters specific needs to the profiles without modifying the structure of the underlying module.

4.5.2 Classes inheritance

Modules are notably composed of a set of manifests containing classes. There are two types of classes:

- **public** classes: these classes can be called by the profiles and can receive arguments.
- **private** classes: these classes are called only by other classes from the same module. Generally, the private classes do not receive arguments as they inherit the public classes and get access to all their variables this way.

There must be a private class `params` for each public class. For example, there must be a private `software::params` private class for a `software` public class. The `params` private class defines the default values of the public class parameters, including its arguments. A public class must inherit of its corresponding `params` class.

A public class should not manage any resource by itself. The resources must be delegated to the private sub-classes. The resources must be grouped by sequentials *deployment steps*. A step must be managed by a specific private class. The common steps are:

- **install**: install packages and files (including directories) required by the technical component.
- **config**: manage the configuration parameters of the technical component.
- **service**: manage the service of the technical component.

This list is not exhaustive and can be adapted to specific cases.

> **Note**
> There is generally some confusions to define whether some files are part of the *install* or the *config* step. Considering the software component follow main rules of the FHS, a file under `/etc` is part of the *config* step. It is part of the *install* step otherwise.

Each step must be managed by a specific private class. For example, `software::install`, `software::config` and `software::service`. These private classes must be called sequentially by the public class, using ordering arrows delimited by anchors. Please refer to modules examples section for full examples.

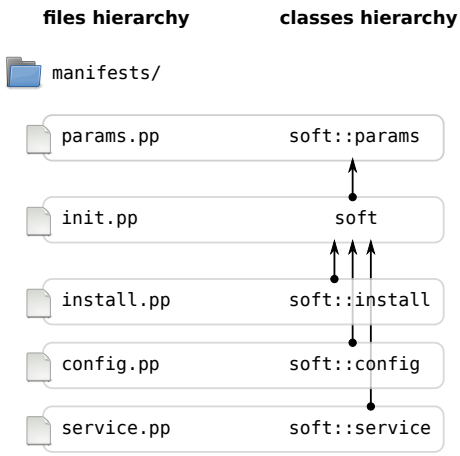

Figure 4.3: Files and classes hierarchies in a simple module

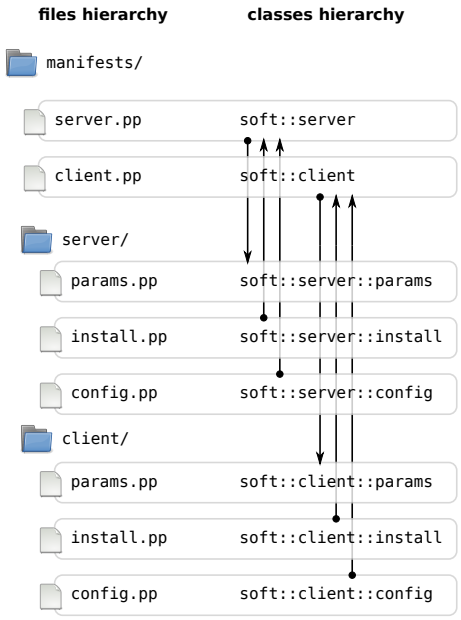

Figure 4.4: Files and classes hierarchies in a complex module

4.5.3 Parameters

Classification

Modules are controlled by a large set of parameters which defines their behavior, resources settings, paths and so on. Each parameter is a variable defined in a public class of a module.

There are two levels of parameters visibility:

- *public parameters* whose values can be set by Hiera auto-lookup or by profiles through the arguments.
- *private parameters* which are defined in the public class of the module. These parameters generally store the result of a function from `stdlib` or `hpclib` modules based on the values of some other public parameters. The variable name of these parameters must be prefixed by an underscore _ (*ex:* `$_config_options`).

There are two categories of parameters:

- **activation parameters**, detailed further in the following sub-section.
- **data parameters**. As their name suggest, they provide data the public class in order to control resources content and metadata. Many conventions have been defined for data parameters, detailed in the data parameters conventions sub-section.

Activation parameters

All public classes must have an **activation parameter** for each deployment step. These parameters must be named `<step>_manage` where <**step**> is replaced by the name of the step. For example, if a public class has 3 deployment steps `install`, `service` and `config`, there must be the following 3 public activation parameters:

- `install_manage`
- `service_manage`
- `config_manage`

If the public class also manages packages, typically within the install step, there must also be a `packages_manage` parameter.

These parameters are all booleans.

The activation parameter control the deployment steps activation **ie.** whether the resources of the corresponding step are actually managed by the public class or not.

Data parameters conventions

Data parameters are basically all other parameters except activation parameters. As previously stated in the classification sub-section, the data parameters hold the content and meta-parameters of the resources. Many data parameters are actually quite similar across modules. For these recurring parameters, several name and type conventions have been defined in Puppet-HPC.

For `install` step:

- `packages` (array) is the list of packages to install.
- `packages_ensure` (string) is the expected state of the packages. **Ex: latest** or **installed**.

For `service` step:

- `service_name` (string) is the name of the managed service.
- `service_ensure` (string) is the expected state of the service. **Ex: running** or **stopped**.
- `service_enable` (boolean) defines if the service start at boot time.

For `config` step, where `<file>` is replaced by the a symbolic name representing the nature of the file (**ex:** `config`):

- `<file>_file` (string), absolute file path of the file on the target system.
- `<file>_options` (hash), content of a configuration file with all its sections, parameters and values. This hash is typically processed by the `hpclib::print_config()` function.
- `<file>_enc` (string), URL of encrypted source of the file, typically processed with the `hpclib::decrypt()` function.
- `default_file` (string), absolute path to the configuration file path of the init-system service script or description file. **ex:** `/etc/default/service`.
- `default_options` (hash), content of the `default_file`.

Type checking

Parameters types must be checked at the beginning of public classes code using `validate_*()` functions of the stdlib module. The tests of parameters types must be conditioned by the activation parameters of the steps they are involved. For example, the `packages` parameter type must be checked only if `packages_manage` parameter is true.

4.5.4 Arguments

Public classes accept arguments. There must be arguments for every public parameters of a public class. The values of these arguments must default to the variables inherited from the corresponding `params` class, with 2 exceptions:

- When there is no sane possible default value, typically for security reasons (*ex:* password) or because it highly depends on the context (*ex:* network domain). In this case, the arguments must be placed in first positions in the arguments lists.

- For configuration structures. It is generally useful to combine settings given in arguments by profiles and default settings coming from the `params` class using `merge()` and/or `deep_merge()` functions from the stdlib module. This way, it becomes unnecessary to define **all** the parameters in the structure in argument, the profile can simply gives to parameters to add or to override in the defaults. In this case, the default values must of the argument be an empty hash or an empty array, depending its type.

4.5.5 Examples

This section contains two full examples of puppet module, one **simple** module with one public class and another **complex** modules with two public classes.

Simple example

The **simple** module simply install packages and launch a service. The private class `simple::config` is a no-op.

File `README.md`:

```
# simple

#### Table of Contents

1. [Overview](#overview)
2. [Module Description](#module-description)
3. [Setup](#setup)
    * [What iscdhcp affects](#what-iscdhcp-affects)
    * [Setup requirements](#setup-requirements)
    * [Beginning with iscdhcp](#beginning-with-iscdhcp)
4. [Usage](#usage)
5. [Limitations](#limitations)
6. [Development](#development)

## Module Description

The module deploys simple stuff.

## Setup

### What simple affects

The module installs simple software with its configuration file and manages its
service.

### Setup Requirements

N/A

### Beginning with simple

N/A

## Usage

The simple module has only one public class named ‘simple‘. It can be easily
instanciated with its defaults argument:

‘‘‘
include ::simple
‘‘‘

## Limitations

This module is mainly tested on Debian, but is meant to also work with RHEL and
derivatives.
```

```
## Development

Patches and issues can be submitted on GitHub:
https://github.com/edf-hpc/puppet-hpc
```

File `init.pp`:

```
################################################################################
#  Puppet configuration file                                                   #
#                                                                              #
#  Copyright (C) 2014-2016 EDF S.A.                                            #
#  Contact: CCN-HPC <dsp-cspit-ccn-hpc@edf.fr>                                 #
#                                                                              #
#  This program is free software; you can redistribute in and/or               #
#  modify it under the terms of the GNU General Public License,                #
#  version 2, as published by the Free Software Foundation.                    #
#  This program is distributed in the hope that it will be useful,             #
#  but WITHOUT ANY WARRANTY; without even the implied warranty of              #
#  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the               #
#  GNU General Public License for more details.                                #
################################################################################

# Deploys simple stuff.
#
# @param install_manage  Public class manages the installation (default: true)
# @param packages        Array of packages to install (default:
#                        ['simple-package'])
# @param packages_manage Public class installs the packages (default: true)
# @param packages_ensure Target state for the packages (default: 'latest')
# @param service_manage  Public class manages the service state (default: true)
# @param service_name    Name of the service to manage (default:
#                        'simple-service')
# @param service_ensure  Target state for the service (default: 'running')
# @param service_enable  The service starts at boot time (default: true)
# @param config_manage   Public class manages the configuration (default: true)

class simple (
  $install_manage   = $::simple::params::install_manage,
  $packages_manage  = $::simple::params::packages_manage,
  $packages         = $::simple::params::packages,
  $packages_ensure  = $::simple::params::packages_ensure,
  $service_manage   = $::simple::params::service_manage,
  $service_name     = $::simple::params::service_name,
  $service_ensure   = $::simple::params::service_ensure,
  $service_enable   = $::simple::params::service_enable,
  $config_manage    = $::simple::params::config_manage,
) inherits simple::params {

  validate_bool($install_manage)
  validate_bool($packages_manage)
  validate_bool($service_manage)
  validate_bool($config_manage)

  if $install_manage and $packages_manage {
    validate_array($packages)
    validate_string($packages_ensure)
  }

  if $service_manage {
    validate_string($service_name)
    validate_string($service_ensure)
    validate_bool($service_enable)
  }

  anchor { 'simple::begin': } ->
  class { '::simple::install': } ->
```

```
  class { '::simple::config': } ->
  class { '::simple::service': } ->
  anchor { 'simple::end': }

}
```

File `params.pp`:

```
##############################################################################
#  Puppet configuration file                                                 #
#                                                                            #
#  Copyright (C) 2014-2016 EDF S.A.                                          #
#  Contact: CCN-HPC <dsp-cspit-ccn-hpc@edf.fr>                               #
#                                                                            #
#  This program is free software; you can redistribute in and/or             #
#  modify it under the terms of the GNU General Public License,              #
#  version 2, as published by the Free Software Foundation.                  #
#  This program is distributed in the hope that it will be useful,           #
#  but WITHOUT ANY WARRANTY; without even the implied warranty of            #
#  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the             #
#  GNU General Public License for more details.                              #
##############################################################################

class simple::params {

  $install_manage  = true
  $packages_manage = true
  $packages        = ['simple-package']
  $packages_ensure = 'latest'
  $service_manage  = true
  $service_name    = 'simple-service'
  $service_ensure  = 'running'
  $service_enable  = true
  $config_manage   = true

}
```

File `install.pp`:

```
##############################################################################
#  Puppet configuration file                                                 #
#                                                                            #
#  Copyright (C) 2014-2016 EDF S.A.                                          #
#  Contact: CCN-HPC <dsp-cspit-ccn-hpc@edf.fr>                               #
#                                                                            #
#  This program is free software; you can redistribute in and/or             #
#  modify it under the terms of the GNU General Public License,              #
#  version 2, as published by the Free Software Foundation.                  #
#  This program is distributed in the hope that it will be useful,           #
#  but WITHOUT ANY WARRANTY; without even the implied warranty of            #
#  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the             #
#  GNU General Public License for more details.                              #
##############################################################################

class simple::install inherits consul {

  if $::simple::install_manage {

    if $::simple::packages_manage {

      package { $::simple::packages:
        ensure => $::simple::packages_ensure,
      }

    }

  }
```

```
}
```

File `config.pp`:

```
#############################################################################
#  Puppet configuration file                                                #
#                                                                           #
#  Copyright (C) 2014-2016 EDF S.A.                                         #
#  Contact: CCN-HPC <dsp-cspit-ccn-hpc@edf.fr>                              #
#                                                                           #
#  This program is free software; you can redistribute in and/or            #
#  modify it under the terms of the GNU General Public License,             #
#  version 2, as published by the Free Software Foundation.                 #
#  This program is distributed in the hope that it will be useful,          #
#  but WITHOUT ANY WARRANTY; without even the implied warranty of           #
#  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the            #
#  GNU General Public License for more details.                             #
#############################################################################

class simple::config inherits simple {

  if $::simple::config_manage {

    notice("nothing to configure in this example simple module")

  }

}
```

File `service.pp`:

```
#############################################################################
#  Puppet configuration file                                                #
#                                                                           #
#  Copyright (C) 2014-2016 EDF S.A.                                         #
#  Contact: CCN-HPC <dsp-cspit-ccn-hpc@edf.fr>                              #
#                                                                           #
#  This program is free software; you can redistribute in and/or            #
#  modify it under the terms of the GNU General Public License,             #
#  version 2, as published by the Free Software Foundation.                 #
#  This program is distributed in the hope that it will be useful,          #
#  but WITHOUT ANY WARRANTY; without even the implied warranty of           #
#  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the            #
#  GNU General Public License for more details.                             #
#############################################################################

class simple::service inherits simple {

  if $::simple::service_manage {

    service { $::simple::service_name:
      ensure  => $::simple::service_ensure,
      enable  => $::simple::service_enable,
    }

  }

}
```

Complex example

The **complex** module has two public classes: `complex::client` and `complex:server`. There is not `init.pp` manifest file but there are instead manifests files for each public class. The privates classes are defined in manifests located inside a the sub-directory of their respective public class. **Ex:** `complex::server::install`

private class associated to `complex::server` public class is defined in `install.pp` manifest inside `server` sub-directory.

File `README.md`:

```
# complex

#### Table of Contents

1. [Overview](#overview)
2. [Module Description](#module-description)
3. [Setup](#setup)
    * [What iscdhcp affects](#what-iscdhcp-affects)
    * [Setup requirements](#setup-requirements)
    * [Beginning with iscdhcp](#beginning-with-iscdhcp)
4. [Usage](#usage)
5. [Limitations](#limitations)
6. [Development](#development)

## Module Description

The module deploys complex client and server.

## Setup

### What complex affects

The module installs complex software in both client and server modes. It manages
the configuration files and server service.

### Setup Requirements

The module depends on:

* ‘stdlib‘ module (for ‘deep_merge()‘ function),
* ‘hpclib‘ module (for ‘print_config()‘ function).

### Beginning with complex

N/A

## Usage

The complex module has two public classes:

* ‘complex::client‘
* ‘complex::server‘

As their name suggest, they respectively manage the client and server parts of
complex software.

The client public class expects a password:

‘‘‘
class { '::complex::client':
  password       => 'CHANGEME',
}
‘‘‘

The server public class mainly expects a partial configuration options hashes
and a password:

‘‘‘
class { '::complex::server':
  config_options => {
    'section1' => {
```

```
      'param1' => 'value7',
      'param2' => 'value8',
    },
    'section3' => {
      'param5' => 'value5',
      'param6' => 'value6',
    },
  },
  password       => 'CHANGEME',
}
```

The `config_options` hash is deep-merged (using `stdlib` `deep_merge()` function) with the default hash from manifest `server::params.pp`. Ideally, the hash given in argument should only contain the difference with the default hash. The default hash value is:

```
  $config_options  = {
    'section1' => {
      'param1' => 'value1',
      'param2' => 'value2',
    },
    'section2' => {
      'param3' => 'value3',
      'param4' => 'value4',
    },
  }
```

After the deep merge, the resulting hash is:

```
  $config_options  = {
    'section1' => {
      'param1' => 'value7',
      'param2' => 'value8',
    },
    'section2' => {
      'param3' => 'value3',
      'param4' => 'value4',
    },
    'section3' => {
      'param5' => 'value5',
      'param6' => 'value6',
    },
  }
```

Limitations

This module is mainly tested on Debian, but is meant to also work with RHEL and derivatives.

Development

Patches and issues can be submitted on GitHub:
https://github.com/edf-hpc/puppet-hpc

File `client.pp`:

```
################################################################################
#  Puppet configuration file                                                   #
#                                                                              #
#  Copyright (C) 2014-2016 EDF S.A.                                            #
#  Contact: CCN-HPC <dsp-cspit-ccn-hpc@edf.fr>                                 #
```

```
#                                                                      #
#  This program is free software; you can redistribute in and/or       #
#  modify it under the terms of the GNU General Public License,        #
#  version 2, as published by the Free Software Foundation.            #
#  This program is distributed in the hope that it will be useful,     #
#  but WITHOUT ANY WARRANTY; without even the implied warranty of      #
#  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the       #
#  GNU General Public License for more details.                        #
########################################################################

# Deploys complex client stuff.
#
# @param install_manage  Public class manages the installation (default: true)
# @param packages_manage Public class installs the packages (default: true)
# @param packages        Array of packages to install (default:
#                        ['complex-client-package'])
# @param packages_ensure Target state for the packages (default: 'latest')
# @param config_manage   Public class manages the configuration (default: true)
# @param config_file     Absolute path to client configuration file  (default:
#                        '/etc/complex/client.conf')
# @param user            Name of client system user (default:
#                        'complex-client-user')
# @param password        Client password (no default)
class complex::client (
  $install_manage      = $::complex::client::params::install_manage,
  $packages_manage     = $::complex::client::params::packages_manage,
  $packages            = $::complex::client::params::packages,
  $packages_ensure     = $::complex::client::params::packages_ensure,
  $config_manage       = $::complex::client::params::config_manage,
  $config_file         = $::complex::client::params::config_file,
  $user                = $::complex::client::params::user,
  $password,
) inherits complex::client::params {

  validate_bool($install_manage)
  validate_bool($packages_manage)
  validate_bool($config_manage)

  if $install_manage and $packages_manage {
    validate_array($packages)
    validate_string($packages_ensure)
  }

  if $install_manage or $config_manage {
    validate_string($user)
  }

  if $config_manage {
    validate_absolute_path($config_file)
  }

  anchor { 'complex::client::begin': } ->
  class { '::complex::client::install': } ->
  class { '::complex::client::config': } ->
  anchor { 'complex::client::end': }

}
```

File `client/params.pp`:

```
########################################################################
#  Puppet configuration file                                           #
#                                                                      #
#  Copyright (C) 2014-2016 EDF S.A.                                    #
#  Contact: CCN-HPC <dsp-cspit-ccn-hpc@edf.fr>                         #
#                                                                      #
```

```
#  This program is free software; you can redistribute in and/or         #
#  modify it under the terms of the GNU General Public License,          #
#  version 2, as published by the Free Software Foundation.              #
#  This program is distributed in the hope that it will be useful,       #
#  but WITHOUT ANY WARRANTY; without even the implied warranty of        #
#  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the         #
#  GNU General Public License for more details.                          #
##########################################################################

class complex::client::params {

  $install_manage  = true
  $packages_manage = true
  $packages        = ['complex-client-package']
  $packages_ensure = 'latest'
  $config_manage   = true
  $config_file     = '/etc/complex/client.conf'
  $user            = 'complex-client-user'

  # There is not any sane and secure possible default values for the following
  # params so it is better to not define them in this class.
  #   $password
}
```

File `client/install.pp`:

```
##########################################################################
#  Puppet configuration file                                             #
#                                                                        #
#  Copyright (C) 2014-2016 EDF S.A.                                      #
#  Contact: CCN-HPC <dsp-cspit-ccn-hpc@edf.fr>                           #
#                                                                        #
#  This program is free software; you can redistribute in and/or         #
#  modify it under the terms of the GNU General Public License,          #
#  version 2, as published by the Free Software Foundation.              #
#  This program is distributed in the hope that it will be useful,       #
#  but WITHOUT ANY WARRANTY; without even the implied warranty of        #
#  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the         #
#  GNU General Public License for more details.                          #
##########################################################################

class complex::client::install inherits complex::client {

  if $::complex::client::install_manage {

    if $::complex::client::packages_manage {

      package { $::complex::client::packages:
        ensure => $::complex::client::packages_ensure,
      }

    }

  }

}
```

File `client/config.pp`:

```
##########################################################################
#  Puppet configuration file                                             #
#                                                                        #
#  Copyright (C) 2014-2016 EDF S.A.                                      #
#  Contact: CCN-HPC <dsp-cspit-ccn-hpc@edf.fr>                           #
#                                                                        #
#  This program is free software; you can redistribute in and/or         #
#  modify it under the terms of the GNU General Public License,          #
```

```
#  version 2, as published by the Free Software Foundation.              #
#  This program is distributed in the hope that it will be useful,       #
#  but WITHOUT ANY WARRANTY; without even the implied warranty of        #
#  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the         #
#  GNU General Public License for more details.                          #
#########################################################################

class complex::client::config inherits complex::client {

  if $::complex::client::config_manage {

    file { $::complex::client::config_file:
      content => template('complex/client.erb'),
      owner   => $::complex::client::user,
      group   => $::complex::client::user,
      mode    => 0600,
    }

  }
}
```

File server.pp:

```
#########################################################################
#  Puppet configuration file                                            #
#                                                                       #
#  Copyright (C) 2014-2016 EDF S.A.                                     #
#  Contact: CCN-HPC <dsp-cspit-ccn-hpc@edf.fr>                          #
#                                                                       #
#  This program is free software; you can redistribute in and/or        #
#  modify it under the terms of the GNU General Public License,         #
#  version 2, as published by the Free Software Foundation.             #
#  This program is distributed in the hope that it will be useful,      #
#  but WITHOUT ANY WARRANTY; without even the implied warranty of       #
#  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the        #
#  GNU General Public License for more details.                         #
#########################################################################

# Deploys complex server stuff.
#
# @param install_manage  Public class manages the installation (default: true)
# @param packages_manage Public class installs the packages (default: true)
# @param packages        Array of packages to install (default:
#                        ['complex-server-package'])
# @param packages_ensure Target state for the packages (default: 'latest')
# @param service_manage  Public class manages the service state (default: true)
# @param service_name    Name of the service to manage (default:
#                        'complex-server-service')
# @param service_ensure  Target state for the service (default: 'running')
# @param service_enable  The service starts at boot time (default: true)
# @param config_manage   Public class manages the configuration (default: true)
# @param config_file     Absolute path to server configuration file  (default:
#                        '/etc/complex/server.conf')
# @param config_options  Hash of configuration default overrides (default: {})
# @param user            Name of server system user (default:
#                        'complex-server-user')
# @param password        Server password (no default)
class complex::server (
  $install_manage      = $::complex::server::params::install_manage,
  $packages_manage     = $::complex::server::params::packages_manage,
  $packages            = $::complex::server::params::packages,
  $packages_ensure     = $::complex::server::params::packages_ensure,
  $service_manage      = $::complex::server::params::service_manage,
  $service_name        = $::complex::server::params::service_name,
  $service_ensure      = $::complex::server::params::service_ensure,
  $service_enable      = $::complex::server::params::service_enable,
```

```
  $config_manage       = $::complex::server::params::config_manage,
  $config_file         = $::complex::server::params::config_file,
  $config_options      = {},
  $user                = $::complex::server::params::user,
  $password,
) inherits complex::server::params {

  validate_bool($install_manage)
  validate_bool($packages_manage)
  validate_bool($service_manage)
  validate_bool($config_manage)

  if $install_manage and $packages_manage {
    validate_array($packages)
    validate_string($packages_ensure)
  }

  if $service_manage {
    validate_string($service_name)
    validate_string($service_ensure)
    validate_bool($service_enable)
  }

  if $install_manage or $config_manage {
    validate_string($user)
  }

  if $config_manage {
    validate_absolute_path($config_file)
    validate_hash($config_options)
    validate_string($password)
    $_config_options = deep_merge(
      $config_options,
      $::complex::server::params::config_options)
  }

  anchor { 'complex::server::begin': } ->
  class { '::complex::server::install': } ->
  class { '::complex::server::config': } ->
  class { '::complex::server::service': } ->
  anchor { 'complex::server::end': }

  # config change must notify service
  Class['::complex::server::config'] ~> Class['::complex::server::service']

}
```

File server/params.pp:

```
################################################################################
#  Puppet configuration file                                                   #
#                                                                              #
#  Copyright (C) 2014-2016 EDF S.A.                                            #
#  Contact: CCN-HPC <dsp-cspit-ccn-hpc@edf.fr>                                 #
#                                                                              #
#  This program is free software; you can redistribute in and/or               #
#  modify it under the terms of the GNU General Public License,                #
#  version 2, as published by the Free Software Foundation.                    #
#  This program is distributed in the hope that it will be useful,             #
#  but WITHOUT ANY WARRANTY; without even the implied warranty of              #
#  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the               #
#  GNU General Public License for more details.                                #
################################################################################

class complex::server::params {
```

```
  $install_manage  = true
  $packages_manage = true
  $packages        = ['complex-server-package']
  $packages_ensure = 'latest'
  $service_manage  = true
  $service_name    = 'complex-server-service'
  $service_ensure  = 'running'
  $service_enable  = true
  $config_manage   = true
  $config_file     = '/etc/complex/server.conf'
  $config_options  = {
    'section1' => {
      'param1' => 'value1',
      'param2' => 'value2',
    },
    'section2' => {
      'param3' => 'value3',
      'param4' => 'value4',
    },
  }
  $user            = 'complex-server-user'

  # There is not any sane and secure possible default values for the following
  # params so it is better to not define them in this class.
  #   $password
}
```

File `server/install.pp`:

```
#############################################################################
#  Puppet configuration file                                                #
#                                                                           #
#  Copyright (C) 2014-2016 EDF S.A.                                         #
#  Contact: CCN-HPC <dsp-cspit-ccn-hpc@edf.fr>                              #
#                                                                           #
#  This program is free software; you can redistribute in and/or            #
#  modify it under the terms of the GNU General Public License,             #
#  version 2, as published by the Free Software Foundation.                 #
#  This program is distributed in the hope that it will be useful,          #
#  but WITHOUT ANY WARRANTY; without even the implied warranty of           #
#  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the            #
#  GNU General Public License for more details.                             #
#############################################################################

class complex::server::install inherits complex::server {

  if $::complex::server::install_manage {

    if $::complex::server::packages_manage {

      package { $::complex::server::packages:
        ensure => $::complex::server::packages_ensure,
      }

    }

  }

}
```

File `server/config.pp`:

```
#############################################################################
#  Puppet configuration file                                                #
#                                                                           #
#  Copyright (C) 2014-2016 EDF S.A.                                         #
#  Contact: CCN-HPC <dsp-cspit-ccn-hpc@edf.fr>                              #
```

```
#                                                                      #
#  This program is free software; you can redistribute in and/or       #
#  modify it under the terms of the GNU General Public License,        #
#  version 2, as published by the Free Software Foundation.            #
#  This program is distributed in the hope that it will be useful,     #
#  but WITHOUT ANY WARRANTY; without even the implied warranty of      #
#  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the       #
#  GNU General Public License for more details.                        #
########################################################################

class complex::server::config inherits complex::server {

  if $::complex::server::config_manage {

    hpclib::print_config{ $::complex::server::config_file:
      style      => 'keyval',
      data       => $::complex::server::::_config_options,
    }

  }

}
```

File `server/service.pp`:

```
########################################################################
#  Puppet configuration file                                           #
#                                                                      #
#  Copyright (C) 2014-2016 EDF S.A.                                    #
#  Contact: CCN-HPC <dsp-cspit-ccn-hpc@edf.fr>                         #
#                                                                      #
#  This program is free software; you can redistribute in and/or       #
#  modify it under the terms of the GNU General Public License,        #
#  version 2, as published by the Free Software Foundation.            #
#  This program is distributed in the hope that it will be useful,     #
#  but WITHOUT ANY WARRANTY; without even the implied warranty of      #
#  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the       #
#  GNU General Public License for more details.                        #
########################################################################

class complex::server::service inherits complex::server {

  if $::complex::server::service_manage {

    service { $::complex::server::service_name:
      ensure  => $::complex::server::service_ensure,
      enable  => $::complex::server::service_enable,
    }

  }

}
```

4.6 Profiles

Profiles instantiate modules public classes. Optionally, they can provide a set of arguments to the public classes.

Technically speaking, profiles are Puppet classes defined in manifests grouped into one **profiles** module just like classic generic modules. It is possible to define arbitrary levels of sub-classes but classes inheritances must be avoided between profiles classes for the sake of clearness and simplicity. For example, the profiles classes `profiles::soft::server` and `profiles::soft::client` can be defined but they cannot inherit from an hypothetical parent `profiles::soft` class.

As previously stated in the main rules, the profiles must stay as simple as possible: they should only manipulate parameters that have to be manipulated at this layer of the stack. It includes the following

parameters:

- Results of `hiera_array()` and `hiera_hash()` calls on advanced parameters, for the reasons explained in the advanced parameters section.
- Results of **stdlib** and **hpclib** function calls because Puppet functions cannot be called directly in hiera.
- Results of `hiera*()` functions when multiple profiles provide different argument values to the same module public class (case C in schema).
- Parameters whose values are computed based on the above parameters.

It implies that hiera autolookups must be preferred over profiles parameters when possible (case A is preferred over case B in schema).

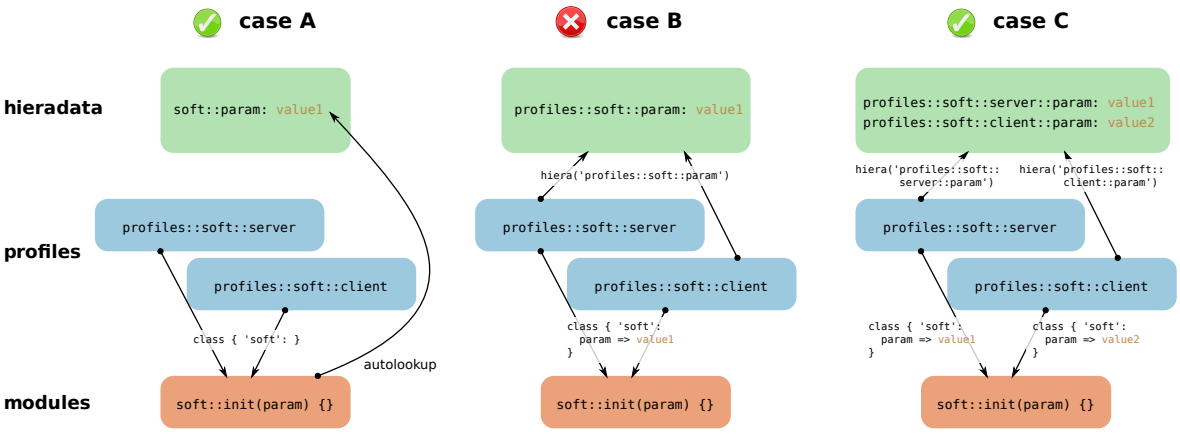

Figure 4.5: Autolookup **vs** profiles parameters

Profiles can only import the following types of parameters from hieradata:

- Shared parameters, as defined in parameters types section.
- Profiles parameters.

The profiles parameters must be prefixed by `profiles::<profile>` where `<profile>` is the name of the profile. A parameter can be imported by multiple profiles classes sharing sharing a sub-class namespace. For example, the `profiles::soft::server` and `profiles::soft::client` classes can share profile parameters because there are both in the `profiles::soft` sub-class namespace. However, `profiles::monitoring` and `profiles::scheduler` cannot share profile parameters. If a parameter is shared by multiple profiles classes, `<profile>` must be replaced by the highest common sub-class namespace. For example:

- a parameter shared by `profiles::monitoring::client` and `profiles::monitoring::server` must be prefixed by `profiles::monitoring`.
- a parameter shared by `profiles::env::soft::client` and `profiles::env::soft:server` must be prefixed by `profiles::env::soft`.

When importing hash or array parameters from hiera, the profiles must set the default value with an empty structure with the second argument of `hiera_hash,array()` functions. **Ex:**

```
hiera_hash('profile::soft::param', {})
```

Profiles cannot import parameters autolookup-ed by modules.

Profiles cannot define resources. The standard `create_resource()` function and **stdlib** `ensure_resource()` function are just wrappers over resources definitions. Therefore, they are also prohibited inside profiles classes.

4.7 Roles

As stated in the roles section of the software architecture chapter, a role is a set of profiles. Puppet-HPC requires that this list of profiles is an array in the `profiles` parameter of the hieradata. It has to be defined in the role level of the hieradata in order to be different from one role to another.

This a requirement in Puppet-HPC because **hpclib** module functions extracts roles and profiles definitions from the hieradata.

As explained in the project genericity goals, the roles are specific to each cluster. Owing to this charateristic, the role level of the hieradata must stay as small as possible to avoid duplications of parameters from one cluster to another. Ideally, it should only contain the profiles array. In particular, settings that are closely coupled to the general architecture of Scibian HPC clusters or settings that be directly be deduced from other parameters must not be defined in this level of the hieradata.

4.8 Advanced processing

For the sake of consistency and coherency, it is sometimes relevant to define advanced data structures in the hieradata (**ex**: networks settings) irrespectively of modules expectations. These data structures have to be processed then by some logic to generate other runtime temporary data structures ready to be consumed by modules.

There two ways to process data extracted from hieradata in Puppet:

- Custom facts,
- Custom parser functions.

Facts have the advantage of being usable directly in the hieradata, quite the opposite of Puppet parser functions. But facts are processed unconditionally, it is therefore important to keep them consistent and light. In Puppet-HPC, facts are considered relevant for very generic parameters used in many places across the hieradata. For other processing, typically for generating resources hash definitions, Puppet parser functions are largely preferred.

4.9 Git repository

All Puppet-HPC developments must happen in **master** branch. There are other branches for dedicated purpose:

- The **gh-pages** branch for publishing the documentation on GitHub,
- The **calibre/*** branches for Debian/Scibian packages maintenance.

All other branches are temporary development branches and should be removed regularly.

Merge commits (**ie.** commits with two parents) are forbidden in the **master** branch. Commits must be re-based on remote HEAD before being pushed.

The commit messages must follow the Git official documentation commit guidelines.

In a few words:

- First line summary length must be under 50 chars.
- Unless really obvious, there should be a long summary (separated by a blank line with first line) with a detailled description wrapped to 72 chars. This long summary should focus on what and why instead of how. The how must be wisely explained in codes comments or in documentation.
- Only one logical changeset per commit.
- `git diff --check` error free, notably with trailing white spaces.

The commit messages must be written in English.

The short summary must follow this format:

```
<prefix>: <summary>
```

Where `<prefix>` depends on commit modification target:

- `doc` for modifications in the `doc/` directory.
- `ex` for modifications in the `examples/` directory.
- `hieradata` for modifications in the `hieradata/` directory.
- `prof:<profile>`, where `<profile>` is the top profile name, for modifications on profiles.
- `mod:<module>`, where `<module>` is the name of the module, for modifications on a module.

For example, for modifications on:

- profile `profiles::foo`, the prefix is `prof:foo`,
- profile `profiles::base::bar`, the prefix is `prof:base`,

- module `simple`, the prefix is `mod:simple`
- public class `complex::server::params`, the prefix is `mod:complex`

This implies that a commit should modify only one type of content: one module, profiles sharing the same hierarchy, hieradata, documentation, and so on. Exceptions to this rule can happen but must be reserved to very specific corner cases (**ex:** large refactoring) and must be done wisely.

4.10 Debugging

4.10.1 Static analysis

Puppet-HPC provides a script `validate.sh` to check both the syntax and the code style of the modules. To check the syntax, run the following command:

```
$ puppet-config/validate.sh --syntax
```

The script must print `Syntax OK` for all files, otherwise errors must be fixed.
To check the code style, run the following command:

```
$ puppet-config/validate.sh --lint
```

The script must not print any `ERROR` or `WARNING`, otherwise they must be fixed.
The script can eventually take a module name in parameter to restrict the check on this module.
Internally , the script actually runs the `puppet-lint` command. The command can also be ran manually using the following additional parameters:

- `-no-class_inherits_from_params_class-check`: the configuration does not support puppet $<$ 3.0, so this check is ignored
- `-no-80chars-check`: the limit in the style guide is 140 characters, but `puppet-lint` in Debian Jessie is not up to date.

4.10.2 Scripts

4.10.3 Unit tests

4.11 Documentation

4.11.1 Module

Each module must be accompanied by a `README.md` file located at the root directory of the module. This file must be formatted in markdown markup language. The content of this file must respect the official Puppet modules documentation specifications in terms of content and format, except for the **reference** section. This section is replaced by inline manifest documentation using Puppet strings format.

4.11.2 Profiles

All profiles must be documented inline using Puppet strings format as well. In the heading comments of each profile manifests, there must be a `Hiera` section which contains the list of expected hieradata parameters along with their description, types and optionally examples of values.

Chapter 5

Reference API