EDF HPC

# HPC System: Puppet Configuration

Date:

Pages: 26

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

- 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 written in the format below:

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

## 1.3 License

## 1.4 Authors

In alphabetical order:

- Thomas HAMEL

# Chapter 2

# Overview

## 2.1 What is provided?

The goal of the configuration is to provide a common configuration for HPC clusters. The Puppet HPC configuration provides:

- Generic Puppet modules for tools commonly used in HPC clusters, these modules can be provided directly or coming from third party sources (mainly the puppetlabs forge)
- Specialized Puppet modules that can only work with the rest of the configuration, these modules rely on data provided by **hiera**
- A generic **hiera** configuration to provide some defaults
- A series of scripts and tools to apply, fetch and debug the configuration

## 2.2 What is missing?

The Puppet HPC configuration can not be applied directly, it must be configured for a specific cluster or site. This configuration is done by using **hiera**. This configuration should at least define the cluster (See: Cluster Definition).

It's also possible to expand the puppet modules to include custom modification or support for software and product not implemented in the common configuration.

## 2.3 Workflow

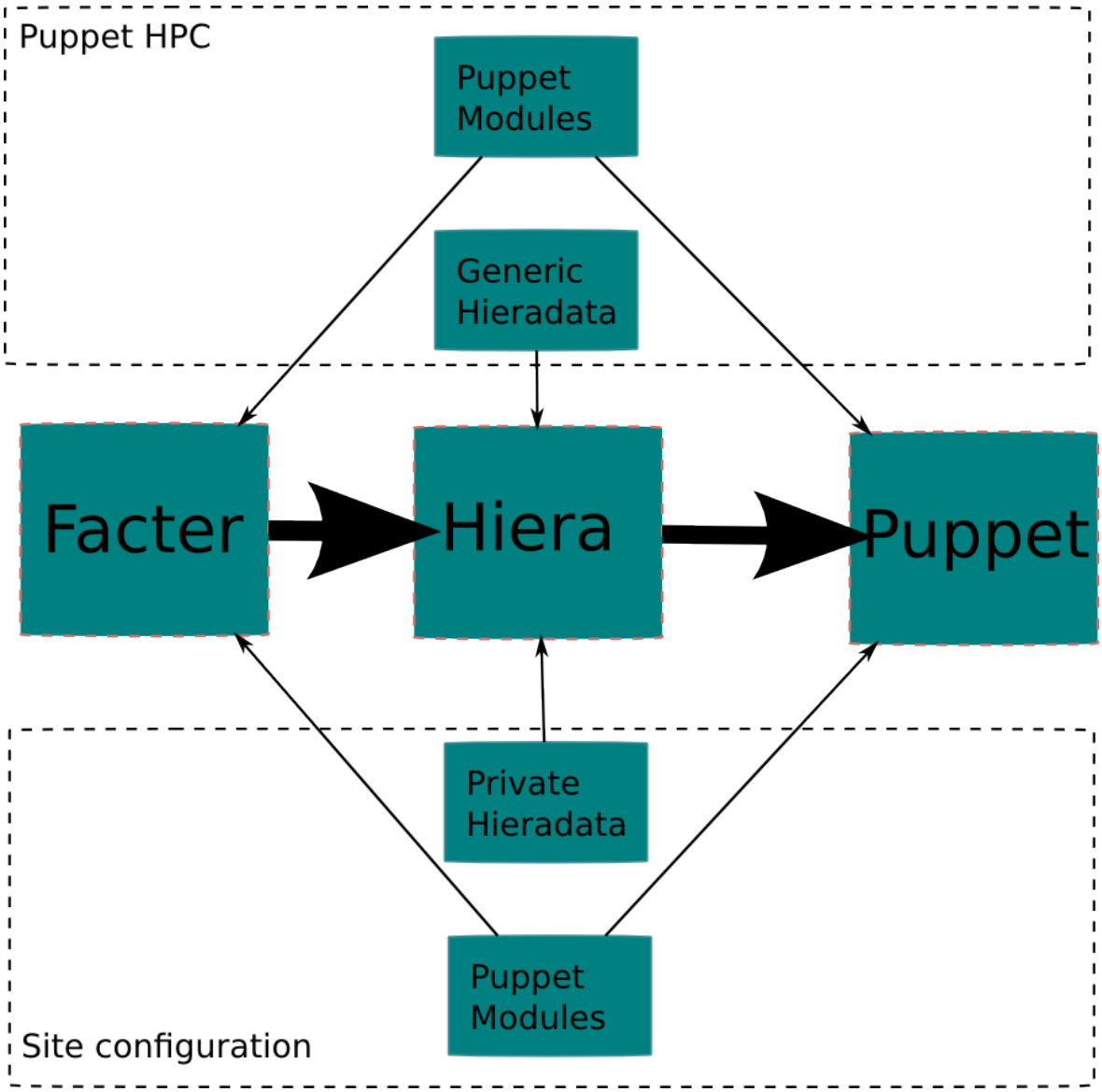


Figure 2.1:

The final configuration in Puppet is called the **catalog**, the catalog is computed from:

- The site main **manifest** (`site.pp`)
- The puppet modules included
- **hiera**
- The facts (native or from the modules)

In the hiera configuration, it is possible to reference facts and top level variables. Top level variable are defined by the site manifest.

The Puppet HPC configuration is particular because some facts (from the `hpclib` module) are defined from the cluster definition that is stored in hiera.

## 2.4 Customization

The aim of the configuration is to provide a common configuration base between multiple clusters and to cleanly separate configuration data from logic. This means that the primary tool for a system administrator should be to modify the hiera private configuration.

If a configuration can not be handled only by modifying hiera, it should be added as a custom module and not by modifying the Puppet HPC manifest directly.

Systems administrators are encouraged to open issues with the Puppet HPC configuration project on github in those situation. The modification might make sense on other sites or be a new default.

# Chapter 3

# Cluster Definition

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

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

## 3.1 Architecture

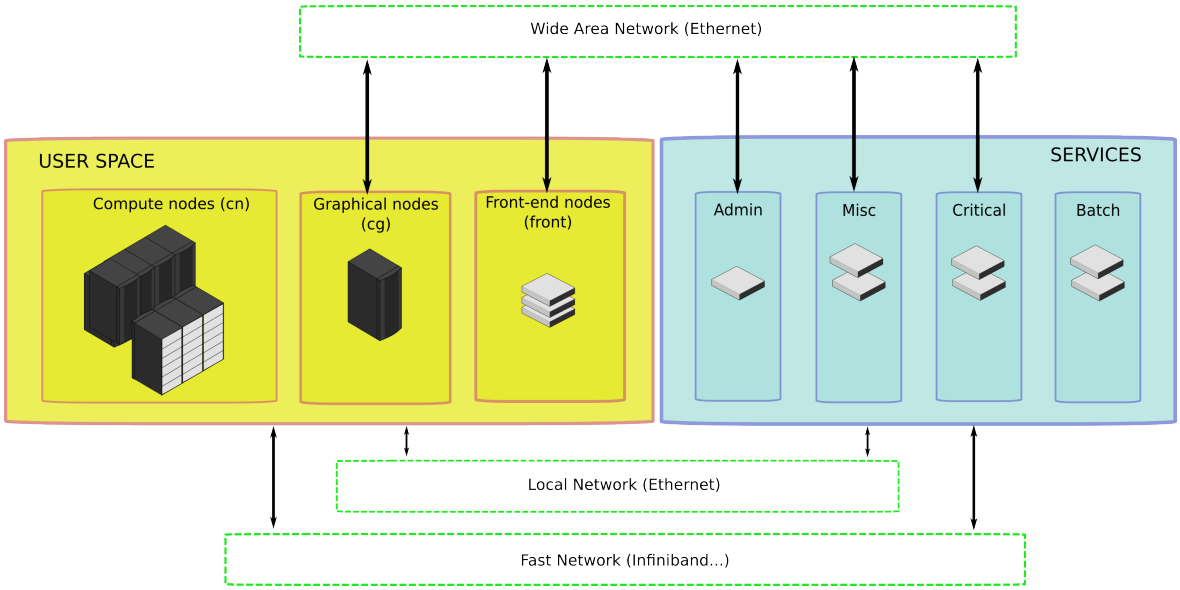


Figure 3.1:

## 3.2 Network definitions

### 3.2.1 Topology

Network topology is defined in the *cluster* level of the Hiera hierarchy. This means it is common to all nodes.

```
## Network topology of the cluster
net::allloc::ipnetwork: '172.16.0.0'
net::allloc::netmask: '255.255.0.0'
net::clusterloc::ipnetwork: '172.16.0.0'
net::clusterloc::netmask: '255.255.248.0'
net::clusterloc::prefix_length: '/21'
net::clusterloc::broadcast: '172.16.7.255'
net::clusterib::ipnetwork: '172.16.40.0'
net::clusterib::prefix_length: '/21'
```

```
net::clusteradm::ipnetwork: '172.16.80.0'
net::clusteradm::netmask: '255.255.240.0'
net::clusteradm::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'
    'allloc':
        'ipnetwork':      '172.16.0.0'
        'netmask':        '255.255.0.0'
    'clusterloc':
        'name':           'CLUSTER'
        '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'
    'clusterib':
        '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'
    'clusteradm':
        'name':            'ADMIN'
        'prefixes':        'adm'
        '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'
    'mgmt':
        'name':            'MANAGEMENT'
        'prefixes':        'mgmt'
        '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'
```

### 3.2.2 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 (same bond interface on same slaves 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*).

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

## 3.3 Node definitions

### 3.3.1 Master Network

Nodes are listed in a hiera array called `master_network`. This structure is derived from an internal CSV file format. Each array "line" defines one node and its network configuration.

Each line consist of five comma separated lists of values, and three lists of associations between those values.

The value lists are:

- MAC addresses
- Interface devices
- Hostnames
- IPv4 Addresses
- IPv4 Net Masks

The associations reference each list with an index starting at 0. The associations are:

- DHCP Configuration, "MAC addresses" ←→"Hostnames" ←→"IPv4 Addresses"
- Node Configuration, "Interfaces devices" ←→"IPv4 Addresses" ←→"IPv4 Netmask"
- DNS/Hosts Configuration, "Hostnames" ←→"IPv4 Addresses"

Example:

```
master_network:
  #MAC_Addr;Interfaces;Hostnames;Addresses;NetMask;DHCP(Mac_Addr@Hostname@Address);Config(
    Interface@Address@Netmask);Hosts(Hostname@Address)
  - 52:54:00:ba:9d:ac,52:54:00:43:d9:45,52:54:00:8a:aa:30,52:54:00:8a:0b:d2;bond0,bond1;
    genmisc1,wangenmisc1;172.16.2.21,172.17.42.45;255.255.248.0,255.255.255.0;0@0@0;0@0@0,1
    @1@1;0@0,1@1
```

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

- DHCP
  - `52:54:00:ba:9d:ac genmisc1 172.16.2.21`
- Network configuration on the node

  - bond0 172.16.2.21 255.255.248.0
  - bond1 172.17.42.45 255.255.255.0

- DNS and Hosts

  - genmisc1 172.16.2.21
  - wangenmisc1 172.17.42.45

All lists are optional, so it's possible to define element that just define a Hosts/DNS configuration (for virtual IP addresses for instance):

```
master_network:
  #MAC_Addr;Interfaces;Hostnames;Addresses;NetMask;DHCP(Mac_Addr@Hostname@Address);Config(
    Interface@Address@Netmask);Hosts(Hostname@Address)
  - ;;genmisc;172.16.2.20;;;;0@0
```

### 3.3.2 Roles and Profiles

Each host in the cluster must provides features to fullfill its role. Each feature is called a profile. Each role is defined by a name and a set of profiles.

Each node has a fact `puppet_role` that is referenced in the hiera configuration. This way it's possible to define hiera variable with values common between all the hosts with the same role. This is used to define a hiera variable `profiles` that list the name of all the profile applied to host with this role.

As an example, we define in `hieradata/default/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::opensshserver::server
  - profiles::opensshclient::client
  - profiles::environment::base
  - profiles::environment::codes
  - profiles::environment::modules
  - profiles::environment::vtune
  - profiles::environment::limits
  - profiles::auth::client
  - profiles::metrics::collect_base
  - profiles::metrics::collect_jobsched_exec
  - profiles::hardware::user_tuning
  - profiles::hardware::ipmi
  - profiles::log::client
```

Roles themselves are defined by the name of the hosts. The name is analysed by searching for a pattern `<prefix><role name><numerical ID>` (Example: `gencritical1`). This is done by the `hpclib` module which define the `puppet_role` fact. The prefix is the hiera variable: `cluster_prefix`.

# Chapter 4

# Glossary

- **Admin Server**, Node the system administrators use to connect interactively to the cluster
- **Batch Server**, Server hosting services related to the job scheduling system
- **cg**, see *Graphical Node*
- **cn**, see *Compute Node*
- **Compute Node**, Node with a lot of CPU power and mempory to handle the actual jobs
- **Critical Server**, Server hosting critical services used by the nodes
- **Front-End Node**, Node where the users log in interactively to manage jobs, edit files, compile codes or do command line pre/post processing. This type of node is sometime called a *Login Node*.
- **front**, see *Front-End Node*
- **Graphical Node**, Node with a GPU to handle pre/post processing tasks or GPGPU jobs
- **Misc Server**, Server hosting miscellaneous services used by the nodes

# Chapter 5

# Dependencies

## 5.1 Debian

```
# apt-get install clustershell python-yaml
```

## 5.2 Puppet

This is a list of current puppet Module dependencies not provided.

- puppetlabs-stdlib (debian: puppet-module-puppetlabs-stdlib)
- puppetlabs-concat (debian: puppet-module-puppetlabs-concat)
- puppetlabs-apache (debian: puppet-module-puppetlabs-apache)

  stdlib is special, because for "historical" reasons, it is in the current git archive. Installing and using `hiera-eyaml` is not recomended:

```
# apt-get install hiera-eyaml
```

# Chapter 6

# Installation

Installing the configuration will depend of your cluster topology. This page only describes most simple.

## 6.1 Shared `/admin`

In this setup, a storage space is mounted on every nodes of the cluster and the configuration is applied directly from this storage space. By default this space is mounted on `/admin`, using another mount point should not be difficult.

On simple systems, it is possible to use an NFS server to make `/admin` available on all nodes. It is also possible to bootstrap the cluster with a `/admin` on the *Admin Server* exported by NFS and later move it to a more resilient location (HA NFS, CephFS or GPFS).

### 6.1.1 Directory layout

The layout setup should be done on the first node with `/admin` available. This is generally the *Admin Server*.

- `/admin`
- `restricted`
  - `puppet-hpc` (A git clone of the puppet-hpc repository)
    - `puppet-config`
    - `hieradata`
  - `hpc-privatedata` (Frequently another git repository)
    - `hieradata`
    - `files`
  - `hieradata`
    - `generic` (Symbolic link to `/admin/restricted/puppet-hpc/hieradata`)
    - `private` (Symbolic link to `/admin/restricted/hpc-privatedata/hieradata`)
  - `privatefiles` (Symbolic link to `/admin/restricted/hpc-privatedata/files`)
- `public`
  - `http`

### 6.1.2 Puppet

Puppet must be configured to search for the modules in the shared `/admin`. The following file can be used on debian and also search modules installed with debian packages:

```
[main]
logdir=/var/log/puppet
vardir=/var/lib/puppet
ssldir=/var/lib/puppet/ssl
rundir=/var/run/puppet
```

```
basemodulepath=$confdir/modules:/usr/share/puppet/modules:/admin/restricted/puppet-hpc/puppet
    -config/cluster:/admin/restricted/puppet-hpc/puppet-config/modules:/admin/restricted/
    puppet-hpc/puppet-config/modules_3rdparty
prerun_command=/etc/puppet/etckeeper-commit-pre
postrun_command=/etc/puppet/etckeeper-commit-post
stringify_facts=false
hiera_config=/etc/puppet/hiera.yaml

[master]
# These are needed when the puppetmaster is run by passenger
# and can safely be removed if webrick is used.
ssl_client_header = SSL_CLIENT_S_DN
ssl_client_verify_header = SSL_CLIENT_VERIFY
```

### 6.1.3 Hiera-eyaml

It is recomended to use Hiera EYAML to store secret values. The keys must be created on the first node.

```
# mkdir /etc/puppet/secure
# cd /etc/puppet/secure/
# eyaml createkeys
[hiera-eyaml-core] Created key directory: ./keys
Keys created OK
# chown -R puppet:puppet /etc/puppet/secure/keys
# chmod -R 0500 /etc/puppet/secure/keys
# chmod 0400 /etc/puppet/secure/keys/*.pem
```

To configure `eyaml(1)` itself, the following file should be created: `/etc/eyaml/config.yaml`

```
---
pkcs7_private_key: '/etc/puppet/secure/keys/private_key.pkcs7.pem'
pkcs7_public_key: '/etc/puppet/secure/keys/public_key.pkcs7.pem'
```

Hiera is configured to search for values in the generic configuration repository, then in a few files for all nodes, then in files specific for each *role*.

```
:backends:
  - eyaml
:eyaml:
  :datadir:           /admin/restricted/hieradata
  :pkcs7_private_key: /etc/puppet/secure/keys/private_key.pkcs7.pem
  :pkcs7_public_key:  /etc/puppet/secure/keys/public_key.pkcs7.pem
  :extension:         'yaml'
:hierarchy:
  - private/default/roles/%{puppet_role}
  - generic/default/roles/%{puppet_role}
  - private/cluster
  - private/network
  - private/default
  - generic/common
  - generic/%{osfamily}/common
```

### 6.1.4 Cluster Decrypt Password

A decrypt password is used by clara to decrypt files. Once you have generated this password, it should be in your hiera under this structure:

```
cluster_decrypt_password: 'PASSWORD_TO_PUT_IN_EYAML'
clara::password_options:
  ASUPASSWD:   "%{hiera('cluster_decrypt_password')}"
```

This password is used elsewhere in the hiera, generally under the name `decrypt_password` (for example: `opensshserver::decrypt_passwd`). So we define a top level variable (`cluster_decrypt_password`) to reuse it more easily.

### 6.1.5 Cluster keyring

The cluster must use a private cluster keyring. This keyring is used to sign packages generated locally and the local repositories.

You should generate it in your privatedata. You will be asked for a passphrase, this passphrase must be provided interactively when you call `clara repo add|del`. The following command can be pretty long to execute if you don't use a hardware Random Number Generator (RNG).

```
 # LANG=C gpg --no-default-keyring --keyring files/repo/cluster_keyring.gpg --secret-keyring
    files/repo/cluster_keyring.secret.gpg --gen-key                    gpg (GnuPG) 1.4.18;
     Copyright (C) 2014 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

gpg: keyring 'files/repo/cluster_keyring.secret.gpg' created
gpg: keyring 'files/repo/cluster_keyring.gpg' created
Please select what kind of key you want:
   (1) RSA and RSA (default)
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
Your selection? 1
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048) 4096
Requested keysize is 4096 bits
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0)
Key does not expire at all
Is this correct? (y/N) y

You need a user ID to identify your key; the software constructs the user ID
from the Real Name, Comment and Email Address in this form:
    "Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"

Real name: HPC Team Example cluster
Email address: hpc@example.com
Comment:
You selected this USER-ID:
    "HPC Team Example cluster <hpc@example.com>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O
You need a Passphrase to protect your secret key.

passphrase not correctly repeated; try again.
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
..+++++
.....+++++
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
+++++
.+++++
gpg: key 241FB865 marked as ultimately trusted
public and secret key created and signed.

gpg: checking the trustdb
```

```
gpg: public key of ultimately trusted key 1F2607DD not found
gpg: public key of ultimately trusted key 94DEFA86 not found
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:   3  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 3u
pub   4096R/241FB865 2016-05-19
      Key fingerprint = D192 11C0 2EB6 BE80 A3BC  7928 1CB4 3266 241F B865
uid                  HPC Team Example cluster <hpc@example.com>
sub   4096R/C7027D3A 2016-05-19
```

Clara will use this key in its encrypted form, if you have a working `clara enc`, you can use `clara enc encode` directly. Otherwise you can use the following command:

```
$ openssl aes-256-cbc -in cluster_keyring.secret.gpg -out cluster_keyring.secret.gpg.enc -k <
    cluster decrypt password>
```

### 6.1.6 Subsystems initializations

Some subsytems needs specific initialization steps that are not handled by the profiles, refer to the following sections for specific steps:

- MariaDB/Galera: Init/Start
- OpenLDAP: Replica
- SlurmDBD: Init

### 6.1.7 Node bootstraping

Setting up the directory layout can be done once, but you will still have to do some bootstraping on other newly installed nodes. Those steps will be handled by the bootsystem eventually.

The steps are:

- Distributing the puppet configuration
- Distributing the hiera configuration and keys
- Mounting `/admin`

# Chapter 7

# Operations - MariaDB/Galera

## 7.1 Init/Start

You have to perform this operation anytime the cluster is completely down (first boot or full reboot).

```
# echo MYSQLD_ARGS=--wsrep-new-cluster > /etc/default/mysql
# systemctl start mysql
# rm /etc/default/mysql
```

# Chapter 8

# Operations - OpenLDAP

## 8.1 Replica

When you initialize your cluster, if you wish to use a local OpenLDAP replica, you have to execute the script `make_ldap_replica` on your replica node. This script will use an ldif file that you must provide.

## 8.2 Logging

### 8.2.1 Changing log level

To change the log level on a running server you must define a logging modification ldif file:

```
dn: cn=config
changetype: modify
replace: olcLogLevel
olcLogLevel: stats
```

The new level is applied with this command:

```
# ldapmodify -Q -Y EXTERNAL -H ldapi:/// -f /tmp/logging.ldif
```

# Chapter 9

# Operations - SlurmDBD

## 9.1 Init

After the batch server have been installed, you must create the cluster in slurmdbd:

```
# sacctmgr add cluster <cluster_name>
```

# Chapter 10

# Debugging

## 10.1 Facts

### 10.1.1 Listing facts

```
# puppet facts find $(hostname) --render-as=yaml
```

It's possible to use the -d (debug) flag with this command.

### 10.1.2 Debugging facts

This small script can be used to obtain a more precise error when a fact is failing:

```
#!/bin/bash

hpc_puppet_modules_dir=/admin/restricted/puppet-hpc/puppet-config/modules

for dir in /admin/restricted/puppet-hpc/puppet-config/modules/*
do
  FACTERLIB="$FACTERLIB:${dir}/lib/facter"
done

export FACTERLIB

facter "${@}"
```

All traditional facter flags are working:

```
# ./hpc_facter -p --trace
undefined method ‘empty?’ for nil:NilClass
/admin/restricted/puppet-hpc/puppet-config/modules/hpclib/lib/facter/network.rb:91:in ‘<top (
    required)>’
/usr/lib/ruby/vendor_ruby/facter/util/loader.rb:130:in ‘load’
/usr/lib/ruby/vendor_ruby/facter/util/loader.rb:130:in ‘kernel_load’
/usr/lib/ruby/vendor_ruby/facter/util/loader.rb:115:in ‘load_file’
/usr/lib/ruby/vendor_ruby/facter/util/loader.rb:49:in ‘block (2 levels) in load_all’
/usr/lib/ruby/vendor_ruby/facter/util/loader.rb:47:in ‘each’
/usr/lib/ruby/vendor_ruby/facter/util/loader.rb:47:in ‘block in load_all’
/usr/lib/ruby/vendor_ruby/facter/util/loader.rb:45:in ‘each’
/usr/lib/ruby/vendor_ruby/facter/util/loader.rb:45:in ‘load_all’
/usr/lib/ruby/vendor_ruby/facter/util/collection.rb:104:in ‘load_all’
/usr/lib/ruby/vendor_ruby/facter.rb:126:in ‘to_hash’
/usr/lib/ruby/vendor_ruby/facter/application.rb:46:in ‘run’
/usr/bin/facter:9:in ‘<main>’
```

# Chapter 11

# Internals - Roles and Profiles

## 11.1 Overview

The Puppet configuration for HPC clusters has been designed following the **Roles and Profiles** pattern. The Puppet code 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.

## 11.2 Background

Roles and Profiles is a common pattern in puppet code organisation. The pattern is explained in details by this presentation: https://puppet.com/presentations/designing-puppet-rolesprofiles-pattern

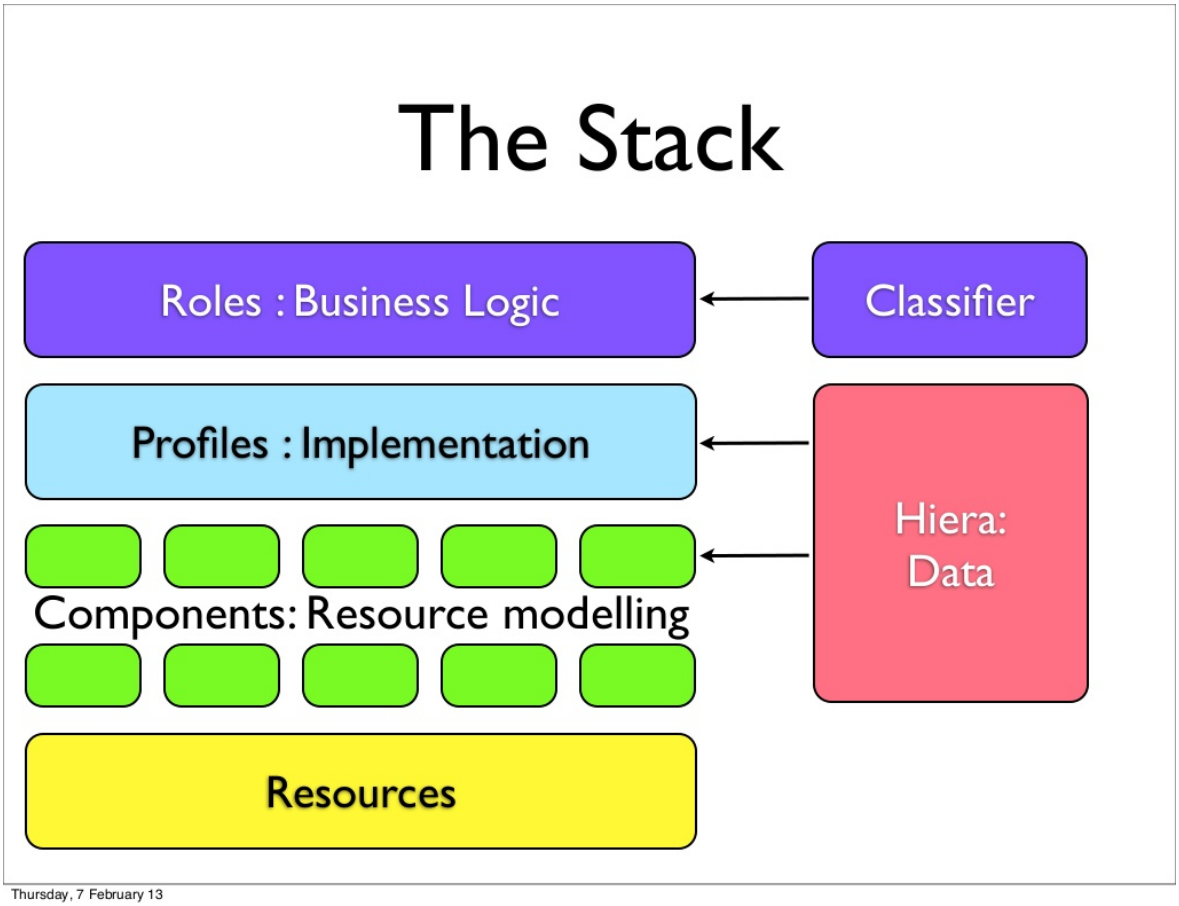


Figure 11.1: The stack (from *Designing Puppet: Roles/Profiles Pattern* by Craig Dunn)

## 11.3 Implementation

### 11.3.1 Node classifiers

A role must be associated to a node. In the Puppet HPC configuration, it is assumed the node is named after its role. The naming rule for a node follows the pattern :

- `<prefix><role name><numerical ID>` (Example: `gencritical1`)

  where `<prefix>` is the value of the variable `cluster_prefix` defined in Hiera.

  To retrieve the role of a node, its hostname is analyzed by searching for the previous pattern. This is done by the Ruby script `role.rb` of the `hpclib` module which defines the `puppet_role` fact.

### 11.3.2 Roles

In the Puppet HPC configuration, each role is associated to the list of profiles it includes. The content of each role is specified in Hiera (`hieradata/default/roles`) in the corresponding YAML file `<role>.yaml`. This YAML file includes the definition of a `profiles` array, which is the list of profiles the role is composed of.

A simple module (`puppet-config/cluster/roles`) is loading all the profile classes defined in the Hiera `profiles` array. The configuration is providing some role definitions (critical, misc, batch, etc) but it is not mandatory to use them. Any site may define its own roles. That is why referencing a role by referencing its name directly should be avoided.

The module `hpclib` defines some facts that indicate which role includes a profile with some specific keywords in its name (`server`, `relay`, `mirror`, `replica`, `tracker`). Looking at the facts on any node of the cluster, it is easy to know which role implements the DNS server or the LDAP replica for example.

### 11.3.3 Profiles

Profiles are classes grouped inside the `profiles` module (`puppet-config/cluster/profiles`). Each profile is a feature for a host. That feature is built from technical components defined by modules. Profiles can rely heavily on Hiera to get the configuration for the site. The profiles do not use auto-lookup but explicit `hiera` (or `hiera_hash` and `hiera_array`) calls, this permits to control how data coming from multiple hierarchy levels are merged.

The `profiles` module only defines classes, no resource, methods or facts. It should not use templates or files. Profiles are meant to rely on the rest of the Hiera/Puppet HPC configuration and may not work outside of it. Profiles can call any class except other profiles.

### 11.3.4 HPC modules

HPC modules are modules that provide a lower level implementation of a technical solution than a profile, but they still rely on the Puppet HPC configuration as a whole. This includes modules that use facts defined by `hpclib`.

Those modules can use other modules and define resources, templates, files, facts, functions... Their name can be prefixed by `hpc_`.

### 11.3.5 Generic modules

Generic modules implement a specific solution directly in a manner that is not specific to the Puppet HPC configuration. These modules can not rely on data other than class parameters or standard facts.

Modules must be able to work without Hiera. Despite the fact that these modules can be used independently of the Puppet HPC configuration, most of them have been created to be used in the context of an HPC cluster. As a result, the features they support might seem incomplete for other uses. If you feel this way, please feel free to contribute and add features to make our modules as generic as possible.

HPC and Generic modules are not strictly separated. Whether data specific to the Puppet HPC configuration is used in the module or not is what makes a module falls into a category or the other.

### 11.3.6 Third Party modules

Usage of third party generic modules is encouraged when possible. The limitation is that it should come from a reliable source: distribution package or puppetlabs forge. For modules from the forge, support level should be **approved** at least. Exceptions to this rule is possible if the module has been properly reviewed. The only exception right now is `puppetlabs-strings`.

Third party modules should be copied in the `puppet-config/modules_3rdparty` directory. This may change in the future.

# Chapter 12

# Writing Modules

## 12.1 Feature set

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 (`foreach...`). Compatibility with the future parser is encouraged though.

You must assume that the manifest will be applied with the following setting in `puppet.conf`:

```
stringify_facts=false
```

The configuration provides a script `validate.sh` that checks the syntax with `puppet-lint`. You can use it or run `puppet-lint` yourself with the following arguments:

- `-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

## 12.2 Style

Puppetlabs provides a style guide that should be respected by all modules: https://docs.puppet.com/guides/style_guide.h

## 12.3 Classes

A class with the same name as the service should exist, this class inherits a `params` class that defines default values for the software. Defaults should change when appropriate (eg. different package names for different OS).

Most modules follow a pattern where the following subclasses are defined:

- `install`, install the packages and other files
- `config`, setup the software (write configuration files)
- `service`, configure and launch the service

Classes `install`, `config` and `service` should inherit the main module class. They are included by the main module class with the proper requirements (typically `install` →`config` →`service`).

This pattern is used by the NTP module from the puppetlabs forge: https://github.com/puppetlabs/puppetlabs-ntp

If a module has multiple subclasses (a client and a server for example). This pattern can be replicated in a subdirectory of `manifests`. The `mysql` module from the puppetlabs forge does this: https://github.com/puppetlabs/pu mysql

## 12.4 Variables

To maintain consistency between modules, some recurring variables and/or parameters should use the following standardised names:

- `service`, the name of the service for this software (string)
- `service_ensure`, should the service be *running* or *stopped*
- `service_enable`, should the service be started on boot (boolean)
- `config_file`, absolute file path of the configuration file (string)
- `config_options`, hash with the values to put in the `config_file` (hash)
- `packages`, list of packages (array of strings)
- `packages_ensure`, state of the packages: *latest* or *installed*
- `default_file`, configuration file path for init system (*/etc/default/xxx* or */etc/sysconfig/xxxx*)
- `default_options`, values to put in the `default_file` (hash)

Last updated 2016-07-05 12:45:01 BST