CIRRAX

Beyond Basics: Hiera Hacks for Fun!

Benedikt Trefzer
benedikt.trefzer@cirrax.com

2.2.2026

Cirrax GmbH

Table of contents

Warmup: hiera

Warmup: hiera

Hiera is key/value data store. Data is organized in a hierarchy.

- keys/values are usually stored in yaml (or json) files
- custom backends allow to use other storage for key/value pairs[1]
- the file hiera.yaml defines the hierarchy and the backends to use
- several merge behaviours are available to combine values from hierarchy levels
- lookup_options configure how lookup is done and it's saved as a hiera data element per key

[1] Examples: `https://github.com/voxpupuli/hiera-eyaml`, `https://github.com/voxpupuli/hiera-file`

Warmup: functions in hiera

- **lookup**: for looking up string values in hiera
- **alias**: for looking up any data type in hiera
- **literal**: to get a '%' character
- Example:[2][3][4]

```
drinks::radler:
  - 'lemonade'
  - 'beer'

drinks::shandy: "%{alias('drinks::radler')}"
drinks::panache: "%{alias('drinks::radler')}"
drinks::panache2: "%{lookup('drinks::radler')}"  # wrong, since this is a String !
```

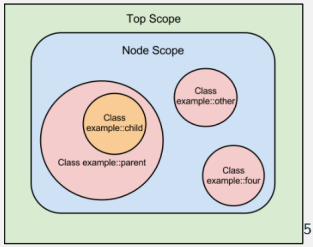
[2] 'Radler' is a mixed drink that consists of lemonade and beer in Germany

[3] 'panaché' is the equivalent of a 'Radler' used in Switzerland (yes it's french, but used in swiss german too)

[4] 'shandy' is the equivalent of a panaché in english (and do not read it as swiss german since there it means a mobile phone!)

variable scopes and lookups

scopes, definition



[5]

- scope isolates code parts from other code parts
- new scope for class, define, function etc
- lambdas inherit the local scope automatically
- with `inherits` keyword you can inherit a parent scope
- all other inherit top scope and node scope (:= no parent scope)
- inherited/top scope values can be overwritten in local scope[6]

[5] from https://help.puppet.com/core/current/Content/PuppetCore/lang_scope.htm

[6] as long as it is not a reserverd variable name such as trusted or facts

variable in hiera lookup

```
1 - name: "Per-node data (yaml version)"
2   path: "nodes/%{::trusted.certname}.yaml"
3
4 - name: "hpath"
5   path: "hpath/%{hpath}.yaml"
6
7 - name: "Common (fallback)"
8   path: "common.yaml"
```

- hiera ignores not defined variables
- for nodes we explicitly use topscope (line 2)
- for hpath local scope is used (which inherits top scope automatically) (line 5)

scope and hiera: test class

```
class presentation::scope (
  Optional[String] $hpath = undef,
  Optional[Any]    $param = undef,
){

  $res=lookup('presentation::scope::param', { 'default_value' => 'not found' })

  echo { 'Output presentation':
    withpath => false,
    message  => [
      '',
      '      ---- start presentation::scope output ----',
      "\$::hpath value (topscope)   : ${::hpath}",
      "\$hpath (local)              : ${hpath}",
      "\$param (autolookup)         : ${param}",
      "\$param (local scope lookup) : ${res}",
      '      ---- end presentation::scope output ----',
      '',
    ].join("\n")
  }
  include presentation::subclass
}
```

demo output of openvox run

```
      ---- start presentation::scope output ----
$::hpath value (topscope)   : from site.pp
$hpath (local)              : test
$param (autolookup)         : value in common.yaml
$param (local scope lookup) : value in hpath/test.yaml
      ---- end presentation::scope output ----

Notice:
      ---- start output from subclass::subclass  ----
$::hpath value (topscope)   : from site.pp
$hpath (local)              : test2
$param (autolookup)         : value in common.yaml
$param (local scope lookup) : value in hpath/test2.yaml
      ---- end output from presentation::subclass ----
```

scope and hiera: conclusions

- automatic parameter lookup for classes always uses top scope
- lookup code in manifests uses local scope[7]
- local scope inherits top scope[8]
- hiera lookup does not prevent filesystem traversal, you can lookup in data on the puppetserver ![9]
- reserved variable names (such as facts, trusted) cannot be overwritten in a local scope

[7] if not explicitly set to top scope in hiera.yaml

[8] so using local scope in hiera.yaml always works!

[9] as long as the lookup type is equal as defined in hiera.yaml

puppet function in hiera

Problem: Who has seen:

```
amodule::password: '***secret***'
# result::password needs to be the sha256sum of amodule::password
result::password: '2778d4beb5382487a08f6abc66e06ff87a8e717293ea5ace713163fe16119958'
```

```
cmodule::values: [ 'a', 'b', 'c' ]
# result::values must be a coma separated join of cmodule::values
result::values: 'a,b,c'
```

```
result::dburi: "mysql://%{lookup('database::username')}:%{lookup('database::password')}@%{lookup('database::hostname')}:33
```

```
 # json as string:
 result::json: '"baseURL": "https://api.example.com/v1", "timeout": 10000, "retries": 3, "headers": { "Content-Type": "appl
```

Solution

- Hiera functions are limited to do lookup and alias, no altering of data
- Puppet has the ability to create functions to alter data
- why not bring puppet functions into hiera lookups ?
- could a hiera backend run a puppet function ?
- but a hiera backend is a key/value store ![10]
- so we need to bring parameter(s) and function name into the key for the lookup
- something like:

```
parameters:
  - 'param1value'
  - 'param2value'

result: "%{lookup('parameters||functionname')}"
result2: "%{alias('parameters||functionname')}"
```

[10] a call to a function consists of a function name and n >=1 parameters

custom Hiera backends

- can be added as a puppet module
- written in ruby in the directory `<module>/lib/puppet/functions`
- referenced in the appropriate hiera.yaml file:

```
hierarchy:
  - name: 'pfunc run puppet function as hiera lookups'
    lookup_key: pfunc_lookup
```

- `pfunc_lookup` is the name of the function to call (and the name of the file)
- consider to place it at the begin of the hierarchy (top priority) to avoid overwrites

backend code structure [11]

```
Puppet::Functions.create_function(:pfunc_lookup) do
  dispatch :pfunc_lookup do
    param 'Variant[String, Numeric]', :key
    param 'Hash', :options
    param 'Puppet::LookupContext', :context
  end

  # actual function, to do the lookup and return the value
  def pfunc_lookup(key, options, context)

    # use cache:
    return context.cached_value(key) if context.cache_has_key(key)

    # search for the result.... here we just return a static value
    result='found'

    # put into cache and return
    context.cache(key, result)
  end
end
```

[11] file <module>/lib/puppet/functions/pfunc_lookup.rb

backend code doing puppet function

```
def pfunc_lookup(key, options, context)
  # return imediatly if it does not fit '.*||.*'
  unless key.match?(%r{\|\|})
    context.not_found
    return
  end

  return context.cached_value(key) if context.cache_has_key(key)

  param, func = key.split('||')

  result =  call_function(func, *lookup_in_hiera(options, context, param))

  context.cache(key, result)
end

def lookup_in_hiera(_options, context, lkup)
  lookedup = context.interpolate("%{alias('#{lkup}')}")
  raise ArgumentError, "pfunc_lookup hiera backend: could not lookup a value for #{lkup} in hiera" if lookedup.to_s.empty?
  lookedup
end
```

That's it

- magic done !
- available as module: `https://github.com/cirrax/hiera-pfunc`
- some test code is also available in the module
- the backend uses caching, which should not pose a problem for deterministic functions
- currently no options are supported

pfunc example: sha256sum

Problem:

```
amodule::password: '***secret***'
# result::password needs to be the sha256sum of amodule::password
result::password: '2778d4beb5382487a08f6abc66e06ff87a8e717293ea5ace713163fe16119958'
```

Solution:

```
amodule::password: '***secret***'

result::password: "%{lookup('amodule::password||stdlib::sha256')}"
```

pfunc example: join

Problem:

```
cmodule::values: [ 'a', 'b', 'c' ]
# result::values must be a coma separated join of cmodule::values
result::values: 'a,b,c'
```

Solution:

```
cmodule::values:
  - [ 'a', 'b', 'c' ]
  - ','

result::values: "%{lookup('cmodule::values|pfunc::join')}"
```

pfunc example: split

Problem:

```
cmodule::values: 'a,b,c'
# result::values must be a coma separated join of cmodule::values
result::values: [ 'a', 'b', 'c' ]
```

Solution:

```
cmodule::values:
  - 'a,b,c'
  - ','

result::values: "%{alias('cmodule::values||pfunc::split')}"
```

remark the use of alias, the result should be an Array (not String)

pfunc example: template

Problem:

```
result::dburi: "mysql://%{lookup('database::username')}:%{lookup('database::password')}@%{lookup('database::hostname')}:33
```

Solution:

```
emodule::input:
  - 'mysql://<%= $username %>:<%= $password %>@<%= $hostname %>:3306/<%= $dbname %>'
  - username: 'auser'
    password: '*secret*'
    hostname: 'localhost'
    dbname: 'ghent'

result::dburi: "%{lookup('emodule::input||inline_epp')}"
```

use an `inline_epp` to set the scope for the parameters

pfunc example: json as a string

Problem:

```
result::json: '{"baseURL": "https://api.example.com/v1", "timeout": 10000, "retries": 3, "headers": { "Content-Type": "ap
```

Solution:

```
fmodule::input:
  - baseURL: 'https://api.example.com/v1'
    timeout: 10000
    retries: 3
    headers:
      Content-Type: 'application/json'
      User-Agent: 'MyApp/1.0'
    endpoints:
      users: '/users'
      aut: '/auth/login'
      orders: '/orders'

result::json: "%{lookup('fmodule::input||stdlib::to_json')}"
# result::json: "%{lookup('fmodule::input||stdlib::to_json_pretty')}"
```

pfunc example: json as a string in ruby

```
fmodule::with_ruby:
  - |
      <%= inline_template('<%=
        require "json"
        @data[0].to_json()
      %>') %>
  - data: "%{alias('fmodule::input')}"

result::json: "%{lookup('fmodule::with_ruby||inline_epp')}"
```

- use an `inline_epp` to run an `inline_template` (ERB)
- with EPP template we can set parameters in the scope !
- the data value is aliases to fmodule::input from the previous slide

pfunc conclusion

- helps making hiera yaml data more readable (sometimes ;))
- makes boundary between data and code more fuzzy
- no issues till now (also for production use)
- advice: use it for small data conversions (it's not a replacement for modules !)
- available as module: `https://github.com/cirrax/hiera-pfunc` or `https://forge.puppet.com/modules/cirrax/pfunc`
- contributions welcome !

bring function and scope together

functions and scope an example

- each function establish their own scope !
- let's group defines in separated hiera files (such as databases, apache vhosts etc.)
- use a function to establish the defines

```
classes:
  - presentation::services        # empty class for autolookup parameter

presentation::services::services: "%{lookup('services||presentation::vthings')}"

# should go to nodes/node.yaml
services:
  - - vhost1.cirrax.com
    - vhost2.cirrax.com
```

```
  function presentation::vthings( Array[String] $services) {
    lookup('resources', { 'merge' => 'deep' }).each | String[1] $k, Hash $v | {
      create_resources($k, $v)
    }
  }
```

functions and scope

```
# data/services/vhost1.cirrax.com.yaml
resources:
  presentation::vdummy:
    vhost1.cirrax.com:
      param: 'this is vhost1.cirrax.com'
```

hiera.yaml hierarchy (partly):

```
  - name: 'pfunc run puppet function as hiera lookups'
    lookup_key: pfunc_lookup

  - name: "services"
    mapped_paths: [ services, service, "services/%{service}.yaml"]
```

questions, remarks, discussions

- pfunc repository: `https://github.com/cirrax/hiera-pfunc`
- pfunc on forge: `https://forge.puppet.com/modules/cirrax/pfunc`
- contact: `benedikt.trefzer@cirrax.com`