What goes in the Profile part of Role/Profile?

Ramin Khatibi, SRE Twitter @ramin_dk

```
My Server = Role {
                Profile {
                  Module {} }}
```

Role, Profile, and Module

```
class role::mysql {
  include ::profile::mysql # just put colons everywhere
}

class profile::mysql {
  include ::mysql           # you need the colons for scope
}

class mysql {
  package { 'mysql-server': }
  service { 'mysqld': }
}
```

This example is too simple

```
class role::db_site {
  include ::profile::base, ::profile::mysql
}

class ::profile::base {
  include ::profile::postfix, ::profile::ssh
}

class ::profile::mysql {
  include ::mysql
  package { 'maatkit': }
}
```

One role, many everything else.

```
class profile::apache {
  include ::apache
  include ::profile::sslcerts

  $a_mods = hiera_array('apache::a2mods', {})
  create_resources('apache::a2mod', $a_mods)

  $a_vhosts = hiera_hash('apache::vhosts', {})
  create_resources('apache::vhost', $a_vhosts)

  Sslcerts::Cert<||> -> Class['apache::service']
}
```

Daemon installs require more than a binary

```
hieradata/stage/fe.yaml
---
apache::vhosts:
    statsstage.example.com:
        priority:   '99'
        template:   'apache/vhosts/stats.example.com.erb'
   stage.example.com:
        priority:   '00'
        template:   'apache/vhosts/example.com.erb'
    vmapstage.example.com:
        priority:   '99'
        template:   'apache/vhosts/vmap.example.com.erb'
```

Apache vhosts in yaml in our Hiera data

```
class profile::haproxy {
  include ::haproxy, ::profile::sslcerts

  logrotate::simple { 'haproxy': }
  rsyslog::simple { 'haproxy': }
  nrpe::checkprocs { 'haproxy': n_warning  => '1:1', n_critical => '1:1',  }

  $ha_default = hiera('haproxy::default',{})
  create_resources('haproxy::default',$ha_default)

  $ha_frontend = hiera_hash('haproxy::frontend',{})
  create_resources('haproxy::frontend',$ha_frontend)

  $ha_backend = hiera_hash('haproxy::backend',{})
  create_resources('haproxy::backend',$ha_backend)

  Sslcert::Cert<||> -> Class['haproxy::service'] }
```

Profiles can get complex quickly, that’s OK

```
class profile::redis {

  # resources we include because we
  # always want it to be there
  include ::redis
  include ::redis::service::disable

  # data we look up because it might be
  # different based on the role of the server
  $myredis = hiera('redis::servers', {})
  create_resources('redis::server', $myredis)
}
```

Static and dynamic resources

```
class profile::haproxy {

  include ::haproxy

  # The haproxy package ships broken versions
  # of these config files. :-(
  logrotate::simple { 'haproxy': }
  rsyslog::simple { 'haproxy': }
}
```

Profiles, used to monkey patch packages

```
class role::logger {
  include ::profile::disk::raid0
  include ::profile::logger
  include ::profile::mcollective
}

class profile::logger {
  include ::profile::rsyslog
  include ::rsyslog::remote
  include ::rsyslog::remote::rails
}
```

This Profile took a week of fail to get right.

```
class rsyslog::remote::rails {

  file { '/mnt/rsyslog/rails': ensure => directory, } ->
  file { '/var/log/rails':      ensure => symlink, target => '/mnt/rsyslog/rails',}
  file { '/etc/rsyslog.d/31-remote_rails.conf':
    content => template('rsyslog/remote_rails.conf.erb'),}

# you need two crons, one to compress and one to delete!!
  cron::simple { 'remote_rails_log_compress':
    payload   => 'find /mnt/rsyslog/rails/ -mmin +120 -type f -name "*.log" -print0 | xargs -r
gzip',
    minutes   => '17', }
  cron::simple { 'remote_rails_log_delete':
    command   => 'find /mnt/rsyslog/rails/ -mtime +15 -type f -name "*.log.gz" -print0 | xargs -r
rm',
    minutes   => '17', }

  Class['rsyslog::config'] -> Class['rsyslog::remote::rails'] ~> Class['rsyslog::service']
}
```

This is hard to model in data, so don’t

```
rsyslog::remote:
  rails:
    config:    'rsyslog/rails_remote.conf.erb'
    crons:
      - compress:
          command: 'gzip file'
          minute:  '17'
      - delete:
          command:  'xargs | rm'
          minutes  '17'
     directory_base: '/mnt/rsyslog'
     symlink:        'yes'
```

It’s not the data, it’s the code to consume it.

```
define redis::server (
  $port          = '6379',
  $bind          = '0.0.0.0',
  $master        = 'localhost',
) {
  file { "redis-server-${port}.conf":  }
  file { "redis-slave-${port}.conf":  }
  file { "redis-server-${port}.init":  }
  service { "redis-server-${port}":  }

  datadog::redis { $name: port => $port, }
  nrpe::redis { $name: port => $port, }
  backup::redis { $name: port => $port, }
  Class['redis::service'] -> Redis::Server[$name] }
```

20 Redis instances? Time to generalize.

```
class profile::puppetmaster {

  include ::profile::apache
  include ::profile::passenger
  include ::profile::puppet
}
---
apache::a2mods:
  ssl: {}
apache::vhosts:
  puppet.example.com: {}
```

Data and classes for Apache modules

- There are no rules, only guidelines.
- Only one role. (okay one rule)
- Profiles should reflect your system.
- Think about whether the resource are dynamic based on the role or static.
- Generalizing too early is the enemy of getting work done.

Thank You