TEMPLATES

class postfix {  
   
 package { 'postfix':  
 ensure => present,  
 }  
}  
   
class postfix::backend inherits postfix {  
   
 package { 'postfix-ldap':  
 ensure => present,  
 }  
   
 file { '/etc/postfix/main.cf':  
 ensure => present,  
 owner => 'root',  
 group => 'root',  
 mode => 644,  
 content => template('expressolivre/backend.erb'),  
 notify => Service['postfix'],  
 }  
   
 file { '/etc/postfix/master.cf':  
 ensure => present,  
 owner => 'root',  
 group => 'root',  
 mode => 644,  
 source => "puppet:///expressolivre/master.cf.debian",  
 notify => Service['postfix'],  
 }  
   
 service { "postfix":  
 ensure => running,  
 enable => true,  
 hasrestart => true,  
 hasstatus => true,  
 require => [  
 File["/etc/postfix/main.cf"],  
 File["/etc/postfix/master.cf"],  
 Package["postfix"],  
 Package["postfix-ldap"],  
 ]  
 }  
}

Primeiro vamos interpretar a configuração acima:

* Tenho uma classe chamada postfix
* A classe postfix instala o pacote postfix
* Tenho uma classe chamada postfix::backend
* A classe postfix::backend herda as configurações da postfix (inherits)
* A classe postfix::backend declara um arquivo main.cf e aponta a origem para um template
* A classe postfix::backend declara um arquivo master.cf como file (arquivo estático)
* A classe postfix::backend declara um serviço postfix com múltiplas dependências.

Legal, mas qual a diferença entre um arquivo estático e um template?

O Arquivo estático não muda não na execução do objeto, já o template é gerado de forma dinâmica.

Veja o arquivo estático do main.cf

# /etc/postfix/main.cf  
   
# HEADER: This file was autogenerated by puppet.  
# HEADER: While it can still be managed manually, it  
# HEADER: is definitely not recommended.  
   
smtpd\_banner = $myhostname ESMTP $mail\_name (detran)  
   
inet\_interfaces = all  
inet\_protocols = ipv4  
   
relayhost =  
recipient\_delimiter = +  
   
myorigin = host  
mydestination = $myhostname, localhost.$mydomain, localhost, host.dominio  
mynetworks = 127.0.0.0/8 10.138.26.0/24  
   
unknown\_local\_recipient\_reject\_code = 550  
   
alias\_maps = hash:/etc/aliases  
alias\_database = hash:/etc/aliases

Agora veja um exemplo de template do mesmo arquivo, chamado aqui de main.erb

# /etc/postfix/main.cf  
   
# HEADER: This file was autogenerated by puppet.  
# HEADER: While it can still be managed manually, it  
# HEADER: is definitely not recommended.  
   
smtpd\_banner = $myhostname ESMTP $mail\_name (detran)  
   
inet\_interfaces = all  
inet\_protocols = ipv4  
   
relayhost =  
recipient\_delimiter = +  
   
myorigin = <%= hostname %>  
mydestination = $myhostname, localhost.$mydomain, localhost, <%= fqdn %>  
mynetworks = 127.0.0.0/8 10.138.26.0/24  
   
unknown\_local\_recipient\_reject\_code = 550  
   
alias\_maps = hash:/etc/aliases  
alias\_database = hash:/etc/aliases

Observe as linhas do arquivo que o caracterizam como template

myorigin = <%= hostname %>  
mydestination = $myhostname, localhost.$mydomain, localhost, <%= fqdn %>

Neste exemplo estou usando a variável **fqdn** e a variável **hostname** ambas do facter, com isto, quando o template for chamado essas informações serão substituidas pelo valor de **fqdn** e **hostname** no node em que o template estiver sendo processado, na máquina que estou testando ficaria assim:

myorigin = puppetserverless  
mydestination = $myhostname, localhost.$mydomain, localhost, puppetserverless.hacklab

## **extensao**

Todo o template deve ter a extensão .erb obrigatóriamente

## **taggeando**

Quando formos referenciar alguma variável, vamos usar as seguintes tags

<%= %>

Exemplo prático

<%= variable %>

## **comentários**

Se quiser fazer um comentário no código use o exemplo abaixo

<%# This comment will be ignored. %>

## **lookupvar**

Quando queremos utilizar variáveis de alguma classe externa, devemos fazer um lookup, veja o exemplo abaixo:

<%= scope.lookupvar('apache::user') %>

## **concatenando**

Você pode fazer referências a vários templates de uma só vez

template('my\_module/template1.erb','my\_module/template2.erb')

## **uso de arrays**

Podemos criar iterações com arrays, abaixo defino um array em um manifest que referencia um template

$namservers = ["8.8.8.8", "8.8.4.4", "208.67.222.222", "208.67.220.220"]

No template eu posso entrar com o seguinte código

<% nameservers.each do |val| -%>  
nameserver <%= val %>  
<% end -%>

Com isso a saída seria a seguinte

nameserver 8.8.8.8  
nameserver 8.8.4.4  
nameserver 208.67.222.222  
nameserver 208.67.220.220

São muitas as possibilidades que temos com esse suporte a arrays e impressão por linha.

## **condicionais**

Podemos usar condicionais nos templates, veja o exemplo colocado em um template chamado vimrc.erb

<% if operatingsystem == "Debian" -%> runtime! debian.vim <% end %>

Caso o sistema operacional seja Debian, ele vai colocar a linha abaixo no arquivo

runtime! debian.vim

Simples e prático.

## **testando templates**

Se desejar testa a sintaxe de seus templates, pode fazê-lo usando o comando abaixo

erb -P -x -T '-' mytemplate.erb | ruby -c

## 