Puppet/chef/ansible/saltstack 🡪configuration management tools

terraform/cloudformation/resource manager 🡪Infrastructure provisioning tools

For updating patches puppet works very well.

Security is very good in puppet.

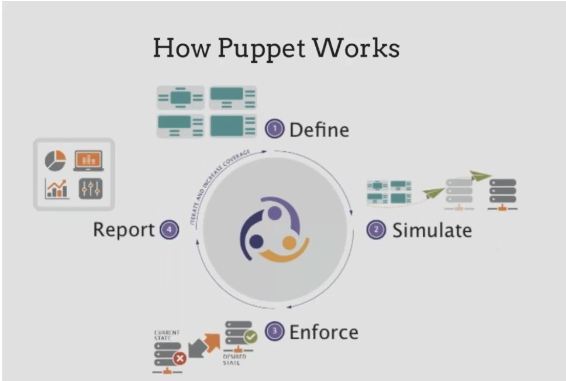
Puppet works as below:

define: define a module

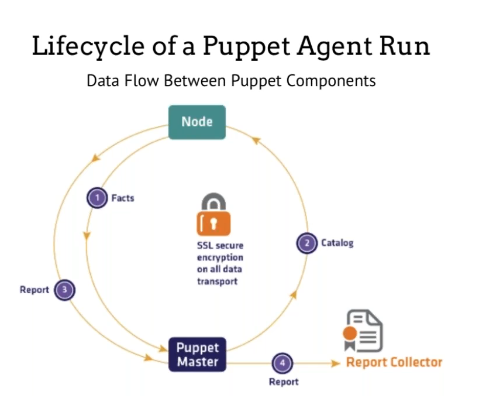
simulate: map it

enforce: execute

report



Lifecyce of Puppet gent run:



Remote execution:

Master

agent

Puppet works on pull mechanism. Puppet master cannot initiate anything.Puppet agent will request puppet master to cofigure something.Puppet master will send the information to agent and code will be executed in agent and sent back to master and stored and represented in dashboard.

Everything is secured with SSL communication.

Puppet agent 🡪facts🡪Puppetmaster🡪catlog🡪execute in agent🡪send report to master

3 ways we can bring the environment

1. Puppet enterprise --can provide dashboard
2. Puppet foreman --can dashboard
3. Puppet standalone -- everything in cli no dashboard

Infra as a code

idempotent

reporting

remote execution

Modules are pulled from puppet master as catalogs.

we need 2 ec2 instances one for puppet master other is puppet agent.

Puppet master installation: In ubuntu 14

sudo sh -c 'echo "deb http://deb.theforeman.org/ trusty 1.5" > /etc/apt/sources.list.d/foreman.list' && \

sudo sh -c 'echo "deb http://deb.theforeman.org/ plugins 1.5" >> /etc/apt/sources.list.d/foreman.list' && \

wget -q http://deb.theforeman.org/pubkey.gpg -O- | sudo apt-key add - && \

sudo apt-get update && sudo apt-get install foreman-installer && \

sudo foreman-installer

below is usefull info after installation:

**Installing Done [100%] [.....................................................]**

**Success!**

**\* Foreman is running at https://ip-172-31-95-234.ec2.internal**

**Default credentials are 'admin:changeme'**

**\* Foreman Proxy is running at https://ip-172-31-95-234.ec2.internal:8443**

**\* Puppetmaster is running at port 8140**

**The full log is at /var/log/foreman-installer/foreman-installer.log**

Puppet agent:

sudo apt-get install -yq puppet

AWS or other cloud providers support puppet chef and ansible config management tools using opswork.

Now agent should communicate with master and we need to update master internal ip address in agent config file (puppet.conf). So we can use puppet utility to locate config directory using below command

**puppet config print confdir**

once this command is executed we get the path where config file is located.so open the config file in vi editor and at last add below lines.

**[agent]**

**server=ip-172-31-95-234.ec2.internal**

ip-172-31-95-234.ec2.internal is internal ip of puppet master updated in puppet agents puppet.conf file.

once done give **service puppet restart.**

Puppet agent and puppet master will communicate in port 8140.

Use puppet master public dns in https mode to access the puppet foreman.

usename:admin

password:changeme

Puppet agent --help

SYNOPSIS

--------

Retrieves the client configuration from the puppet master and applies it to

the local host.

This service may be run as a daemon, run periodically using cron (or something

similar), or run interactively for testing purposes.

In agent give below command

**puppet agent -t**

Exiting; no certificate found and waitforcert is disabled

In master:

Now there must be authentication from agent to master using ssl certificates.In master use below command to locate ssl certificates

**puppet config print ssldir**

/var/lib/puppet/ssl

now got to library

**cd /var/lib/puppet/ssl/ca/signed**

**ls -lrt**

you should see the agent dns in it. Like below

root@ip-172-31-95-234:/var/lib/puppet/ssl/ca/signed# ls -ltr

total 4

-rw-r----- 1 puppet puppet 2041 Nov 18 16:03 ip-172-31-95-234.ec2.internal.pem

now go to

**cd /var/lib/puppet/ssl/ca/requests**

**ls -lrt**

you will find private dns of agent. As below

root@ip-172-31-95-234:/var/lib/puppet/ssl/ca/requests# ls -ltr

total 4

-rw-r----- 1 puppet puppet 1610 Nov 18 17:07 ip-172-31-95-57.ec2.internal.pem

to sign the certificate go for below command

**puppet cert sign ip-172-31-95-57.ec2.internal**

root@ip-172-31-95-234:/var/lib/puppet/ssl/ca/requests# puppet cert sign ip-172-31-95-57.ec2.internal

Notice: Signed certificate request for ip-172-31-95-57.ec2.internal

Notice: Removing file Puppet::SSL::CertificateRequest ip-172-31-95-57.ec2.internal at '/var/lib/puppet/ssl/ca/requests/ip-172-31-95-57.ec2.internal.pem'

ec2-34-207-67-216.compute-1.amazonaws.comwill list certificates in master.

once certificate is signed in master it will move from requests to signed

Now in agent give below command

**puppet agent -t**

Notice: Skipping run of Puppet configuration client; administratively disabled (Reason: 'Disabled by default on new installations');

Use 'puppet agent --enable' to re-enable.

**puppet agent --enable**

once done give below command

**puppet agent -t**

Info: Caching certificate\_revocation\_list for ca

Warning: Unable to fetch my node definition, but the agent run will continue:

Warning: Error 400 on SERVER: Failed to find ip-172-31-95-57.ec2.internal via exec: Execution of '/etc/puppet/node.rb ip-172-31-95-57.ec2.internal' returned 1:

Info: Retrieving plugin

Info: Caching catalog for ip-172-31-95-57.ec2.internal

Info: Applying configuration version '1574094832'

Info: Creating state file /var/lib/puppet/state/state.yaml

Notice: Finished catalog run in 0.01 seconds

Now here agent has received empty catalog from master. Puppet master sent something but nothing is mapped in agent.

where it is mapped in master usually is /etc/puppet manifests directory

**cd /etc/puppet/manifests**

In manifests directory we will find site.pp file.

edit site.pp file in vi editor

**vi site.pp**

**# Empty site.pp required (puppet #15106, foreman #1708)**

**node 'ip-172-31-95-57.ec2.internal'{**

**notify {"Hi I am in $fqdn node def":}**

**}**

**node default{**

**notify {"Hi I am in default node def":}**

**}**

now run **puppet agent -t** in agent. You can see below message in agent.

root@ip-172-31-95-57:/etc/puppet# puppet agent -t

Info: Retrieving plugin

Info: Caching catalog for ip-172-31-95-57.ec2.internal

Info: Applying configuration version '1574100989'

Notice: Hi I am in ip-172-31-95-57.ec2.internal node def

Notice: /Stage[main]/Main/Node[ip-172-31-95-57.ec2.internal]/Notify[Hi I am in ip-172-31-95-57.ec2.internal node def]/message: defined 'message' as 'Hi I am in ip-172-31-95-57.ec2.internal node def'

Notice: Finished catalog run in 0.02 seconds

So here agent is pulling the catalog from master.

Facts are sent to puppet master from agent like global variables. From this only it will pick fqdn values $fqdn.

To get the fact information from agent use **facter –p.**

So here we are sending facts from agents to master and agent will pull catalog (site.pp) from manifests directory from master and execute it.

Also in below script if node (agent ip) is matching then it will execute else it will go to default section and print Hi I am in default def.

**node 'ip-172-31-95-57.ec2.internal'{**

**notify {"Hi I am in $fqdn node def":}**

**}**

**node default{**

**notify {"Hi I am in default node def":}**

**}**

Puppet language files are called manifests, and are named with the .pp file extension.

Puppet DSL contains different resource. So we should call those resource.

Puppet has already developed resources. In programming language we call as methods. Here we call resources. Here resource is like a method.

For ex apt install tree is used to install tree package. So in puppet to perform all these we have resource like install a package, create a file, create a user, and create service everything can be done using resources.

You can check set of resources using command **puppet resources –t**

Below are resources:

augeas

computer

cron

exec

file

filebucket

group

host

interface

k5login

macauthorization

mailalias

maillist

mcx

mount

nagios\_command

nagios\_contact

nagios\_contactgroup

nagios\_host

nagios\_hostdependency

nagios\_hostescalation

nagios\_hostextinfo

nagios\_hostgroup

nagios\_service

nagios\_servicedependency

nagios\_serviceescalation

nagios\_serviceextinfo

nagios\_servicegroup

nagios\_timeperiod

notify

package

resources

router

schedule

scheduled\_task

selboolean

selmodule

service

ssh\_authorized\_key

sshkey

stage

tidy

user

vlan

whit

yumrepo

zfs

zone

zpool

To install tree package:

**puppet resource package tree ensure=latest**

to uninstall:

**puppet resource package tree ensure=purged**

In order to install multiple packages or perform multiple tasks we cannot execute commands instead we use resources and specify them in site.pp file in manifests directory of master and execute it in agent by **puppet agent -t**.

If you want to install tree then in site.pp file in master add script as below.

# Empty site.pp required (puppet #15106, foreman #1708)

node 'ip-172-31-95-57.ec2.internal'{

notify {"Hi I am in $fqdn node def and it is using $kernel":}

**package {"tree":**

**ensure => latest**

**}**

}

node default{

notify {"Hi I am in default node def":}

}

Once added. Go to agent and execute **puppet agent -t** command.

We can get all the syntaxes of ruby dsl or puppet descriptive language from

<https://www.puppetcookbook.com/>

We can get description of each resource by below command

**puppet describe <resource name>**

**puppet describe package**

If we have package installation and file creation for 50 nodes then we cannot write the code in single page. So we can use modularity for reusability. So we will be separating the executions in different approach. Here in site.pp file we can simply call one module include.

# Empty site.pp required (puppet #15106, foreman #1708)

node 'ip-172-31-95-57.ec2.internal'{

include example

}

node default{

notify {"Hi I am in default node def":}

}

So we can keep the module example in path **/etc/puppet/environments/production/modules.**

Now in modules directory give below command to generate example module.

**puppet module generate ip-example**

Here we cannot give module name example directly instead there should be dash-separated username and module name ip-example.

**root@ip-172-31-95-234:/etc/puppet/environments/production/modules# puppet module generate ip-example**

**Notice: Generating module at /etc/puppet/environments/production/modules/ip-example**

**ip-example**

**ip-example/Modulefile**

**ip-example/README**

**ip-example/manifests**

**ip-example/manifests/init.pp**

**ip-example/spec**

**ip-example/spec/spec\_helper.rb**

**ip-example/tests**

**ip-example/tests/init.pp**

Main manifest to modular manifest. So we are writing reusable code in modular manifest instead of writing in **site.pp** file.

So here class name and module name should be same. Class name can be found in **/etc/puppet/environments/production/modules/ip-example/manifests/init.pp and it will be example** but module name is ip-example. So we have to change the module name from ip-example to example from **/etc/puppet/environments/production/modules/**.

**mv ip-example example**

Class contains multiple resources. So if I call the class the all the resources will be executed. A particular configuration can be enabled by using class.

Manifest contains classes and classes contains resources.

Main manifest contain site.pp file and it will call init.pp file from module manifests.

Main manifest

🡪site.pp

🡪node def

🡪class

🡪resources

So we are calling resources in classes and in turn classes are called my site.pp file.

In init.pp file we can call the resources like below. Here we are using package and file resource.

class example {

package {"tree":

ensure => latest

}

file {"/tmp/a.txt/":

ensure => file,

content => "Hi I am alok"

}

}

Note: Here ensure has different values. File then file is created, directory then directory is created, installed, present etc.

In site.pp file we will be including class name example.

node 'ip-172-31-95-57.ec2.internal'{

include example

}

node default{

notify {"Hi I am in default node def":}

}

Note: Idempotent is denoting an element of a set which is unchanged in value when multiplied or otherwise operated on by itself.

exec resource will not exhibit idempotent behavior.

Ex: To create nginx module

**puppet module generate ip-nginx** in /etc/puppet/environments/production/modules/ directory

**mv ip-nginx nginx** class name similar to module name

Now go to init.pp file in manifests directory /etc/puppet/environments/production/modules/nginx/manifests/init.pp

In init.pp give the following code.

**class nginx{**

**package{"nginx":**

**ensure => latest**

**}**

**file{"/etc/nginx/server.xml":**

**ensure => file,**

**source => "puppet:///modules/nginx/server.xml**

**}**

**service{"nginx":**

**ensure => running**

**}**

**}**

Here if content of file is huge then we can use source option under file resource. Below is description for source option.

=====================================================================================

- \*\*source\*\*

A source file, which will be copied into place on the local system.Values can be URIs pointing to remote files, or fully qualified paths to files available on the local system (including files on NFS shares or Windows mapped drives). This attribute is mutually exclusive with `content` and `target`. The available URI schemes are \*puppet\* and \*file\*. \*Puppet\* URIs will retrieve files from Puppet's built-in file server, and are usually formatted as:

`puppet:///modules/name\_of\_module/filename`

This will fetch a file from a module on the puppet master (or from a local module when using puppet apply). Given a `modulepath` of `/etc/puppetlabs/puppet/modules`, the example above would resolve to `/etc/puppetlabs/puppet/modules/name\_of\_module/files/filename`. Unlike `content`, the `source` attribute can be used to recursively copy directories if the `recurse` attribute is set to `true` or `remote`. If a source directory contains symlinks, use the `links` attribute to specify whether to recreate links or follow them. Multiple `source` values can be specified as an array, and Puppet will use the first source that exists. This can be used to serve differentfiles to different system types:

file { "/etc/nfs.conf":

source => [

"puppet:///modules/nfs/conf.$host",

"puppet:///modules/nfs/conf.$operatingsystem",

"puppet:///modules/nfs/conf"

]

}

We have to create directory files in /etc/puppet/environments/production/modules/nginx/. So in /etc/puppet/environments/production/modules/nginx/files create file called server.xml and add the content into it.

Now we have to create mount point.

go to fileserver.conf file in /etc/puppet and open in vi editor and here we should give mount point.

[modules]

path /etc/puppet/environments/production/modules/

allow \*

here modules is mount point. If we give puppet:///modules automatically it will point to /etc/puppet/environments/production/modules/.

restart apache2 server

**service apache2 restart**

Now in below code

class nginx{

package{"nginx":

ensure => latest

}

file{"/etc/nginx/server.xml":

ensure => file,

source => "puppet:///modules/nginx/server.xml

}

service{"nginx":

ensure => running

}

}

If you execute it, it will not execute sequentially like package 🡪file🡪service.

So in order to execute it sequentially we have concept called meta parameters or dependency parameters.

class nginx{

package{"nginx":

ensure => latest,

before => File["/etc/nginx/server.xml"]

}

file{"/etc/nginx/server.xml":

ensure => file,

source => "puppet:///modules/nginx/server.xml

require => Package["nginx"],

notify => Service["nginx"]

}

service{"nginx":

ensure => running

subscribe => [ File["/etc/nginx/server.xml"], Package["nginx"] ]

}

}

Below are parameters Refer above code:

**Require:**

The required resources will be applied before this resource.

**Before:**

This resource will be applied before the dependent resources.

**Refresh events:**

**Subscribe:**

The subscribed resources will be applied before this resource.

If Puppet makes changes to any of the subscribed resources, it will cause this resource to refresh. (Refresh behavior varies by resource type: services will restart, mounts will unmount and re-mount, etc. Not all types can refresh.)

**Notify:**

This resource will be applied before the notified resources.

If Puppet makes changes to this resource, it will cause all of the notified resources to refresh. (Refresh behavior varies by resource type: services will restart, mounts will unmount and re-mount, etc. Not all types can refresh.)

Now place above code in init.pp file and can be validated using below command

**puppet parser validate init.pp**once validated execute **puppet agent -t** in agent.

Before that nginx module should be called in site.pp file like **include nginx.**

We can find multiple modules from puppet forge.

We can also do autosigning of certificates of any host agents by editing **autosign.conf** file in **/etc/puppet/** directory and add \* in it and save.

Once done we have to restart apache2 service **service apache2 restart.** So that if we bring another ec2 instance automatically certifficates will be signed.

We can create subclass in modules. For ex we have module nginx init.pp file in **/etc/puppet/environments/production/modules/nginx/manifests** directory .So we can create subclasses by creating service.pp or configure.pp.

But naming convention in service.pp or configure.pp will be as below **class nginx::service** here nginx is module and service is subclass. Similarly **class nginx::configure** in config.pp file.

So I can call individual classes from site.pp file.So in below script I have called example and nginx class from site.pp file. I can also call subclasses from site.pp file line **include nginx::service** , **include nginx::configure**.

If I give include nginx then init.pp file will be called. If I call include nginx::service then service.pp file will be called in nginx module. Similarly if I call include nginx::configure then configure.pp file will be called in nginx module.

ex:

node 'ip-172-31-30-130.ec2.internal'{

include example,nginx

include nginx::service

include nginx::configure

}

Also I can call subclasses in module manifests init.pp file like below. So nginx is main class in init.pp file and subclasses service,configure,install can be called in init.pp. So here multiple pp files can be called in init.pp file.

class nginx {

**include nginx ::service**

**include nginx::configure**

**include nginx::install**

package{"nginx":

ensure => latest,

before => File["/etc/nginx/server.xml"]

}

file{"/etc/nginx/server.xml":

ensure => file,

source => "puppet:///modules/nginx/server.xml",

require => Package["nginx"],

notify => Service["nginx"]

}

service{"nginx":

ensure => running,

subscribe => File["/etc/nginx/server.xml"]

}

}

**Variable:**

Variable is a pointer which holds value. Place holder. Used to separate data from code, instead of hard coding.

4 stages to define variables:

1. **Local variables**: Once you define value cannot be changed.

ex: In file{"/etc/nginx/server.xml":

I can declare $path=”/etc/nginx”

So where ever /etc/nginx is there I can give $path. So this cannot be changed.

1. **Global Variables**: These are facters which cannot be changed but varies from different environment. No need to declare as these are predefined and are automatically fetched. All variables and values from **faters -p** are global variables.

Ex:

If you give **facter fqdn** then the value will depend on environment. From one server it will give one value and other will have different value.

class nginx{

$package\_name = ”nginx”

$path = “/etc/nginx”

$fqdn = “node01.example.com”

notify {“Global vars $fqdn”:} -----------🡪 notify {“Global vars $::fqdn”:}

package{"nginx":

ensure => latest,

before => File["/etc/nginx/server.xml"]

}

In above example we have declared $fqdn in both local and global. Always local variables will be checked first and later global variables. In order to make global variables to be assigned first overriding local variable then we should give $::fqdn. So here $::fqdn will become static.

1. **Class variables:**

We can create own custom variables but problem is if one or 2 nodes we can create it. But if 100 nodes it is difficult. To overcome we can declare in class level also can be declared in UI also. These type of custom global variables are called **class variables**. While declaring class I can declare these variables.

So this can be declared at class level.

class nginx($version = "1.0.0"){

$package\_name = "nginx"

$path = "/etc/nginx"

$fqdn = "node01.example.com"

notify {"local vars $path":}

notify {"class vars $version":}

notify{“local vars $fqdn”:}

notify {"Global vars $::fqdn":}

package{$package\_name:

ensure => latest,

before => File["$path/server.xml"]

}

file{"$path/server.xml":

ensure => file,

source => "puppet:///modules/nginx/server.xml",

require => Package["nginx"],

notify => Service["nginx"]

}

service{$package\_name:

ensure => running,

subscribe => File["$path/server.xml"]

}

}

So wherever $version is there will be replaced with 2.0.0.

Now put above code in init.pp file and validate it by **puppet parser validate init.pp**. Make sure the module is called in site.pp file by **include nginx**.

Execute in agent by **puppet agent -t**. Below is output

root@ip-172-31-95-57:~# puppet agent -t

Info: Retrieving plugin

Info: Caching catalog for ip-172-31-95-57.ec2.internal

Info: Applying configuration version '1574237634'

Notice: local vars node01.example.com

Notice: /Stage[main]/Nginx/Notify[local vars node01.example.com]/message: defined 'message' as **'local vars node01.example.com'**

Notice: **class vars 1.0.0**

Notice: /Stage[main]/Nginx/Notify[class vars 1.0.0]/message: defined 'message' as 'class vars 1.0.0'

Notice: **local vars /etc/nginx**

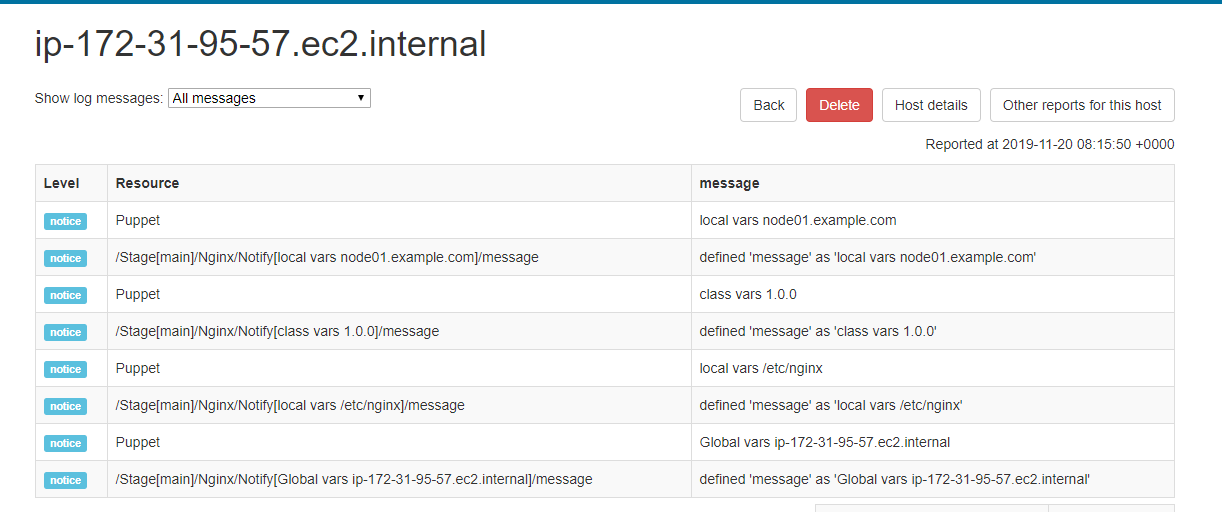
Notice: /Stage[main]/Nginx/Notify[local vars /etc/nginx]/message: defined 'message' as 'local vars /etc/nginx'

Notice: **Global vars ip-172-31-95-57.ec2.internal**

Notice: /Stage[main]/Nginx/Notify[Global vars ip-172-31-95-57.ec2.internal]/message: defined 'message' as **'Global vars ip-172-31-95-57.ec2.internal**'

Notice: Finished catalog run in 0.14 seconds

Also you can check above output in UI also. By logging in to UI and go to the agent and reports and latest time.



If you have different version other than 1.0.0. Then I can go to site.pp file instead of **include nginx** I can use below script.

node 'ip-172-31-95-57.ec2.internal'{

class{"nginx":

version => "2.0.0"

}

}

node default{

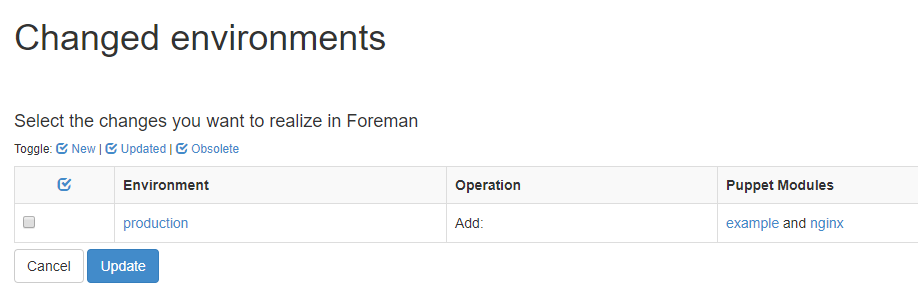
notify {"Hi I am in default node def":}

}

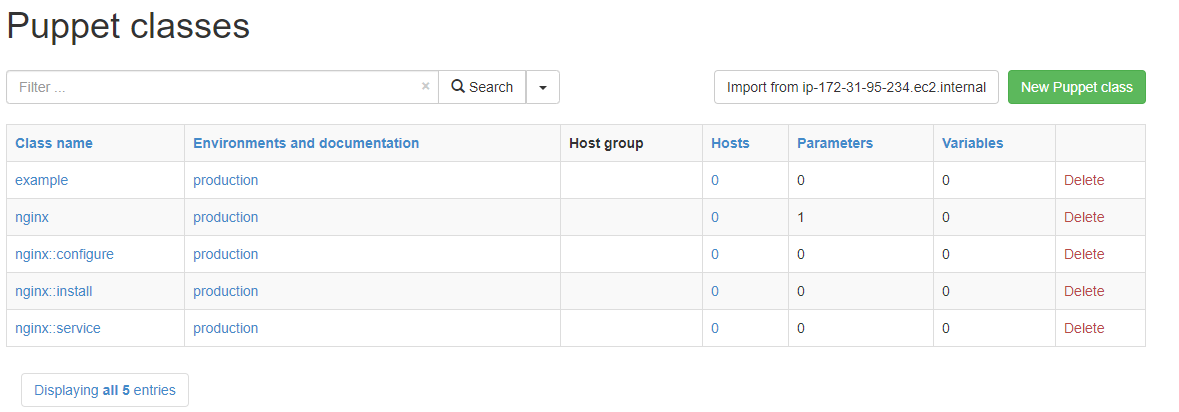
If we have 100 nodes and 100 parameters the site.pp file will be more complex. So we can avoid site.pp file and can be handled using UI itself. First mv site.pp site.pp.bkp.So it will not execute from site.pp.

Now you can get same nginx from UI.

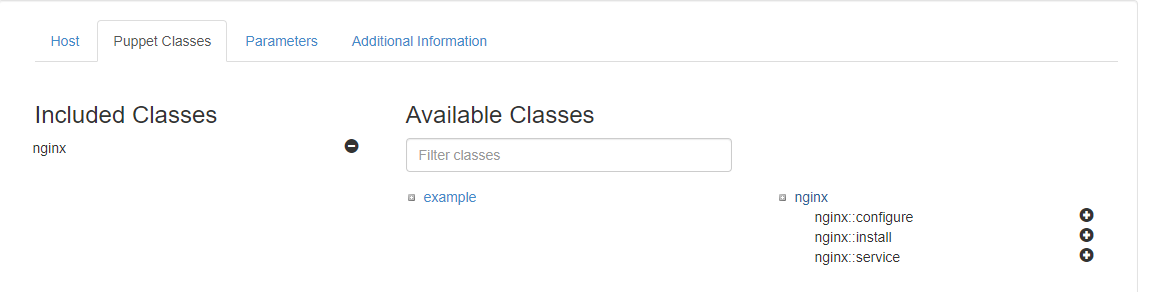
From UI go to all hosts and in configure Puppet classes and import from pupper master host will import all the modules into UI.



All the classes will be imported and can be applied to agents through UI and can eliminate site.pp file.

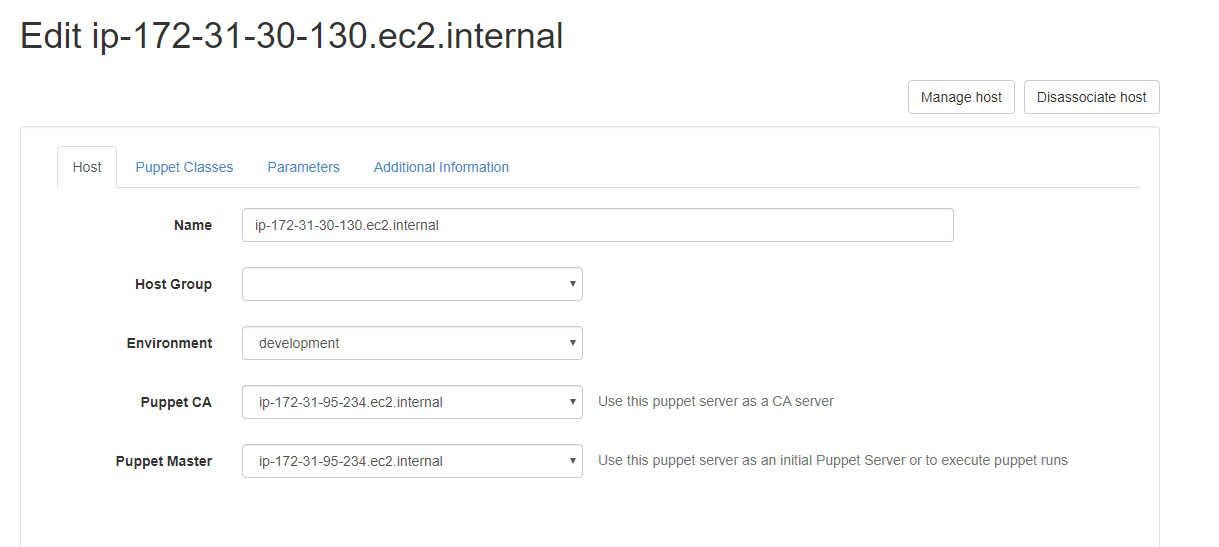


Now go to all hosts and click edit on any of agent and open it. We can add any number of classes.



Also parameters can be added.

Similarly we can copy the modules from production environment to development in master CLI and in UI under host for puppet agent2 we can assign it to development environment and map the class nginx. Once done we can execute puppet agent -t in puppet agent2.



Note: To install git package **puppet resource package git ensure=latest**

Refer simple project from <https://github.com/alokvk1234/puppet-webserver-demo2.git> and clone it in production environment of master and execute it.

Note: To find something in linux use **find / -name heira.yml**

**Hiera:**

Hierarchies are configured in a hiera.yaml configuration file. Each level of the hierarchy tells Hiera how to access some kind of data source. Hiera is the config file for your Puppet code

Puppet uses Hiera to do two things:

1. Store the configuration data in key-value pairs
2. Look up what data a particular module needs for a given node during catalog compilation

So we can declare these variables as like below in init.pp file:

class webserver (

$packagename = hiera('packagename')

$configfile = hiera('configfile')

$configsource = hiera('configsource')

$vhostfile = hiera('vhostfile')

) inherits ::webserver::parameters {

package { 'webserver-package':

name => $packagename,

ensure => present

}

file { 'config-file':

path => $configfile,

ensure => file,

source => $configsource,

require => Package['webserver-package'],

notify => Service['webserver-service'],

}

file { 'vhost-file':

path => $vhostfile,

ensure => file,

content => template('webserver/vhost.conf.erb'),

require => Package['webserver-package'],

notify => Service['webserver-service'],

}

service { 'webserver-service':

name => $packagename,

ensure => running,

enable => true,

hasrestart => true,

subscribe => [ File['config-file'], File['vhost-file'] ],

}

}

So all these values are fetched dynamically from source. That source we will declare in environment.

heira.yml is front controller whenever you hit heira method.

So find hiera.yml file by **find / -name hiera.yaml** in /etc/puppet directory.So we will get path **/usr/share/doc/puppet-common/examples/hiera/etc/hiera.yaml.**

We have to copy this file to /etc/puppet.

**root@ip-172-31-95-234:/etc/puppet# cp /usr/share/doc/puppet-common/examples/hiera/etc/hiera.yaml .**

Now in hiera.yaml

---

:backends:

- yaml

- puppet

:hierarchy:

- "%{::osfamily}"

- "%{environment}"

- common

:yaml:

:datadir: /etc/hieradata

:puppet:

:datasource: data

All the data is fetched from datadir. Data for redhat is in redhat.yaml and debian in debian.yaml files.

Whenever hiera package name comes first it will check debian.yaml file it will check. So in place of location we can use osfamily.If osfamily matches to debian it will check in hiera data directory debian.yaml file will be searched or if osfamily is redhat it will check hiera directory for redhat.yaml file. In that data will be picked and copied to source code dynamically.If nothing is matched it will come to environment if nothing matches it will come to common.

hiera()🡪hiera.yaml🡪hierarchy🡪yaml🡪data

**Roles and profiles:**

Abstraction layers:

Data is abstracted by hiera

Providers are abstracted by types

Resources are abstracted by classes

Modules are abstracted by profiles

Profiles are abstracted by roles

Ex:

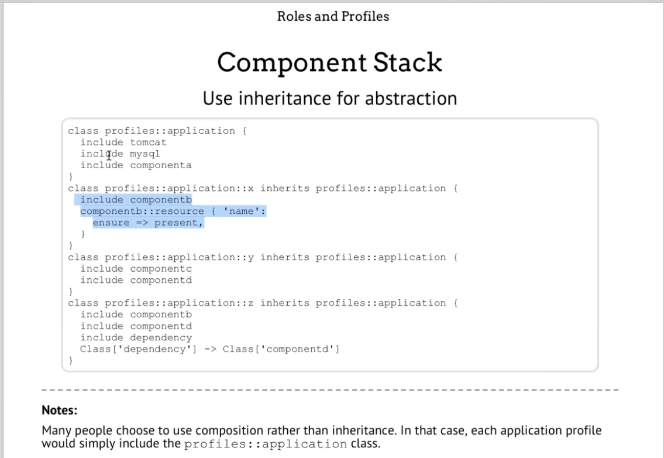
class role::webapp {

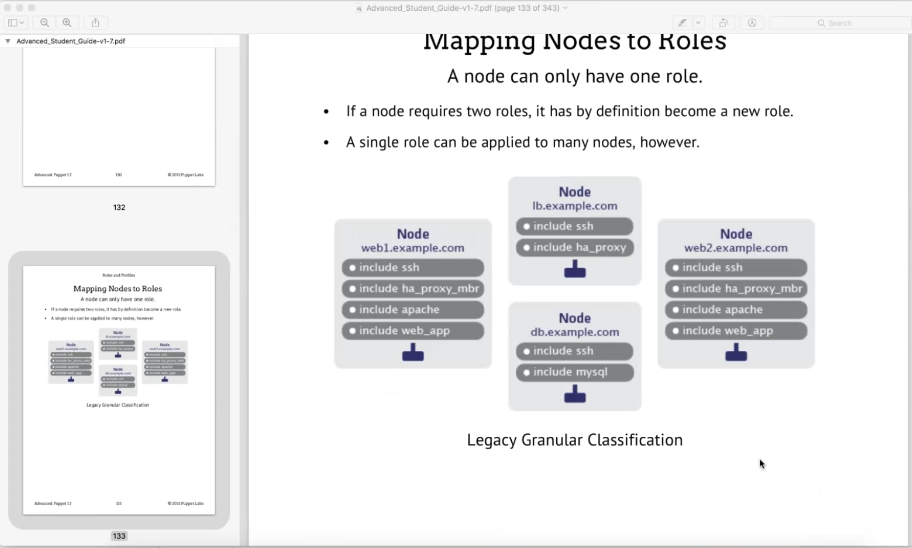
Include profiles::base

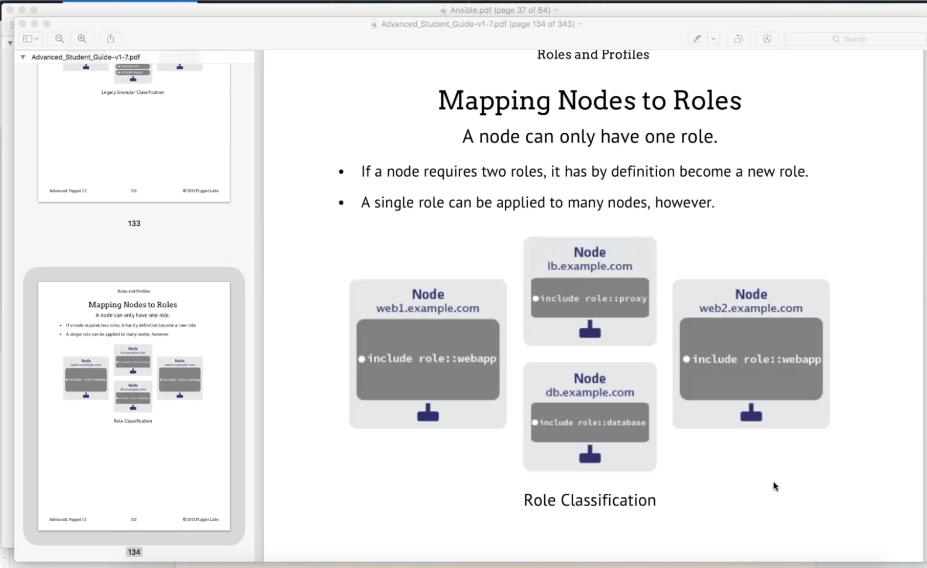
include profiles::customapp

include profiles::test\_tools

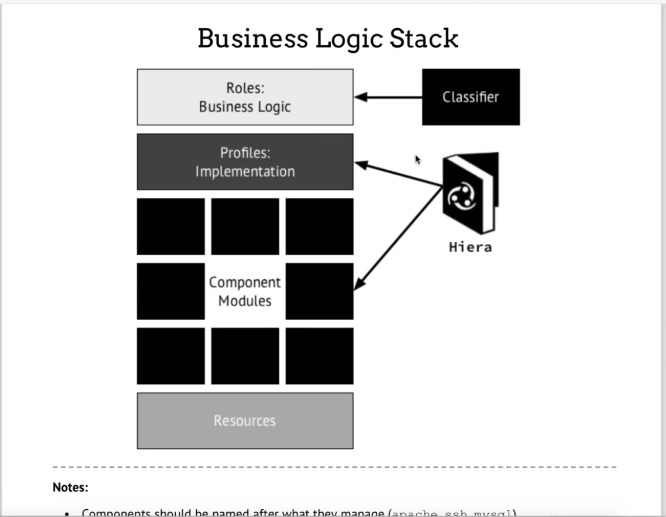
}







Node 🡪Role🡪Profile🡪module🡪manifests🡪classes🡪resources



Roles can be called from UI.

Refer <https://github.com/alokvk1234/puppet-roles>.

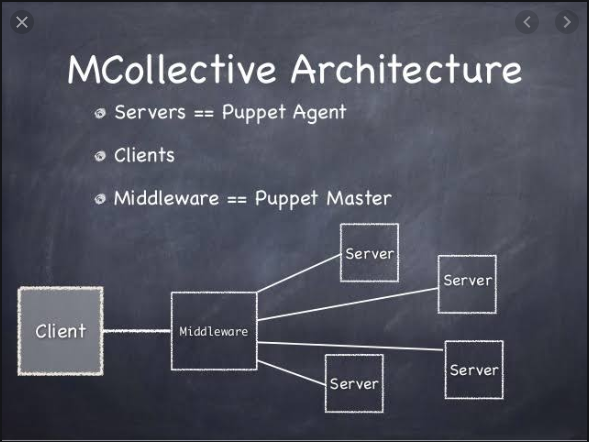
To run puppet agent remotely we have 2 ways:

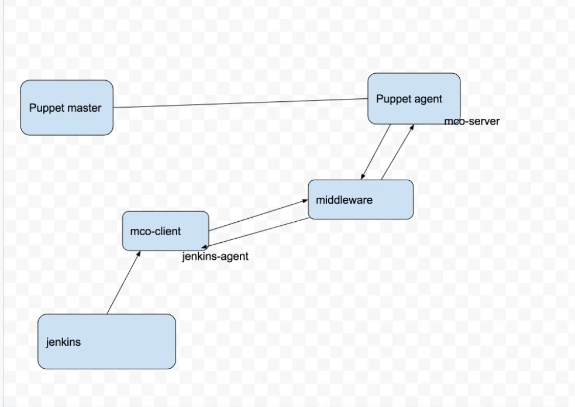
1. Puppet daemon: Like crontab scheduling.If it is 100% idempotent then puppet daemon is best used.
2. mcollective framework: Is orchestration framework used for remote method invocation.Use to run puppet agent automatically. The Marionette Collective, also known as MCollective, is a framework for building server orchestration or parallel job-execution systems. Most users programmatically execute administrative tasks on clusters of servers.

MCO Client

middleware

MCO server





Modules are collection of manifests . Each manifests inside a module consists of class.

So these Classes are functions which can be reused in program.

These classes are defined inside manifests which are basically put inside modules.

Modules 🡪Collection of manifests🡪Each manifest contain class which introduces code reusibility.

**sudo puppet module generate <moduename>** :which defines a file structure in a module.

Classes contain resources.Example below.

class hello{

exec { “Test”:

command =>’/bin/echo aache2 is installed > /tmp/status.txt’,

unless =>’/bin/which apache2’,

}

}

So we can call the class hello in defaut or main manufest just by **include hello**

If we use site.pp file to write entire code without using modules it will become clumsy and long.

In order to make code neat and clean we use modules.