Defined Types and Parameterised Classes

Grouping Resources

In this session, we are going to learn how to group resources into reusable clumps that you can refer to by name, making it easy to create lots of similar resources at once. You can also make your Puppet manifests shorter, neater, and more readable by eliminating duplicated code.

Resources of Same Type

```
package { 'php5-cli':
        ensure => installed,
}

package { 'php5-fpm':
        ensure => installed,
 }

package { 'php-pear':
        ensure => installed,
 }
```

Grouping Together

```
package { 'php5-cli':
          ensure =>
installed,
  }

  package { 'php5-fpm':
          ensure =>
installed,
   }

  package { 'php-pear':
          ensure =>
installed,
   }
```

```
package { [ 'php5-cli',
            'php5-fpm',
            'php-pear' ]:
   ensure => installed,
}
```

Array

A comma-separated list in square brackets, shown in the following code line, is called an array:

```
[ 'php5-cli', 'php5-fpm', 'php-pear' ]
```

We have split it over multiple lines to make it more readable, but it's all the same to Puppet.

Arrays are acceptable in many places where otherwise you might use a single value:

```
require => [ Package['ntp'],
File['/etc/ntp.conf'] ],
```

Grouping resources into arrays is very helpful, but it only works with instances of a single

resource type. What if you want to **group resources of different types**?

Example

Adding Ubuntu Repositories

Adding repositories is a two step process

- Run a command to add repo
- Create a sources file for apt

Example

```
exec { "add-apt-repository-nginx":

  command     => "/usr/bin/add-apt-repository ppa:nginx-org-daily/ppa",
  unless      => "/usr/bin/test -s /etc/apt/sources.d/nginx}",

}

file { "/etc/apt/sources.d/nginx/nginx.conf":
  ensure  => file,
  require => Exec["add-apt-repository-nginx"],
}
```

So far, so good. But when you have many repos to add it would get repetitive

```
exec { "add-apt-repository-repo1": }

file { "/etc/apt/sources.d/repo1/repo.conf":}

exec { "add-apt-repository-repo2": }

file { "/etc/apt/sources.d/repo1/repo.conf":}

exec { "add-apt-repository-repo3": }

file { "/etc/apt/sources.d/repo1/repo.conf":}
                   ⋮

exec { "add-apt-repository-repon": }

file { "/etc/apt/sources.d/repon/repo.conf":}
```

Define

```
define apt::ppa{

  exec { "add-apt-repository-${name}":
     command     => "/usr/bin/add-apt-repository  ${name}",
     unless      => "/usr/bin/test -s ${sources_list_d_filename}",

   }

   file { "/etc/sources.d/${name}/repo.conf":
     ensure  => file,
     require => Exec["add-apt-repository-${name}"],
   }
}
```

Calling Defined Type

```
node 'demo' {

    apt::ppa{'nginx':}

}
```

$name

- special parameter
- title/instance of defined type is available inside the definition as $name
- Its possible to pass additional parameters to defined types

Passing Params

```
node 'demo' {

   apt::ppa{'nginx':
     repo_name => 'nginx'
     repo_file => 'nginx.conf;
   }

}
```

Accepting Params

```
 define apt::ppa(
    repo_name,
    repo_file,
){

   exec { "add-apt-repository-${name}":
      command     => "/usr/bin/add-apt-repository ${repo_name} ${name}",
      unless      => "/usr/bin/test -s ${sources_list_d_filename}",

    }

    file { "/etc/sources.d/${name}/${repo_file}":
      ensure  => file,
      require => Exec["add-apt-repository-${name}"],
    }
}
```

Setting Params Defaults

```
 define apt::ppa(
    repo_name = 'nginx',
    repo_file = 'nginx.conf',
){

   exec { "add-apt-repository-${name}":
      command     => "/usr/bin/add-apt-repository ${repo_name} ${name}",
      unless      => "/usr/bin/test -s ${sources_list_d_filename}",

    }

    file { "/etc/sources.d/${name}/${repo_file}":
      ensure  => file,
      require => Exec["add-apt-repository-${name}"],
    }
}
```

Calling define with defautls

```
node 'demo' {

    apt::ppa{'nginx':}

}
```

Resource Titles

When you use defined types, the resource titles should always be derived from a uniq parameter, typically

${name}

```
exec { "add-apt-repository-${name}"

file { "/etc/sources.d/${name}/repo.conf"
```

Lets enhance nginx module...

Creating a defined type for Nginx websites

Scenario

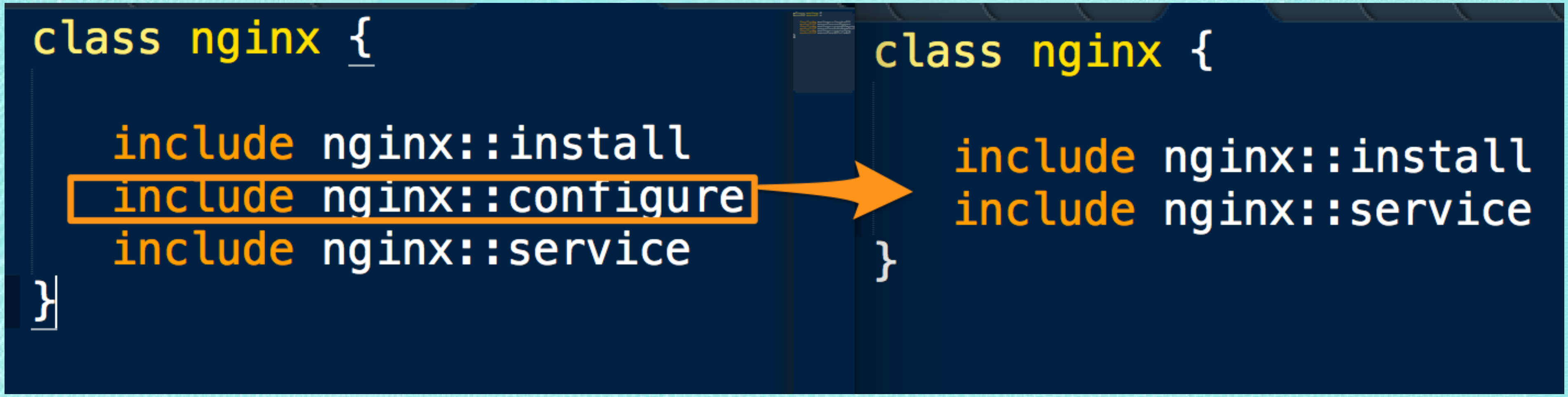
Problem Statement: We are limited to create one site per host. We need to be able to create more than one virtual hosts per node, dynamically.

Success Criteria: We should be able to create multiple instances of virtual hosts

Remove Configure Class

- We need to disable to current approach to write configuration and web site files.
- Since we have modular code, we could do so easily by just removing nginx::configure class from init.pp

init.pp


```
class nginx {

  include nginx::install
  include nginx::configure
  include nginx::service
}
```

```
class nginx {

  include nginx::install
  include nginx::service
}
```

Create nginx::vhost defined type in vhost.pp

```
define nginx::vhost (
       $port = 80,
       $site_domain

){
    include nginx

    file { "/etc/nginx/conf.d/${name}.conf":
      content => template('nginx/vhost.conf.erb'),
      notify => Service['nginx'],
    }

    file { "/var/www/${name}":
      ensure => directory,
    }

    file { "/var/www/${name}/index.html":
      content => template('nginx/index.html.erb'),
    }
}
```

src: https://gist.github.com/initcron/09d89f804e9514d56950

Update vhost.conf.erb template

```
server {
   listen <%=  @port  %>;
   root /var/www/<%=  @name  %>;
   server_name  <%= @site_domain %>;
}
```

https://gist.github.com/initcron/3e9211ae71faa253d616

Add the following line to html body in index.html.erb

```
<h1>hello <%= @name %></h1>
```

nodes.pp

```
node 'demo' {

   nginx::vhost{ 'hollywood':
      port   => 81,
      site_domain => 'hollywood.com'
   }

}
```

apply

```
$ puppet apply /vagrant/puppet/manifests/
```

validate

```
$ curl localhost:81

       Or

http://192.168.33.10:81
```

Classes

Recalling Classes

We've seen classes before, when we used the class keyword to group together the Puppet

resources that implement some particular service, such as Nginx:

```
# Manage nginx webserver
  class nginx {
                  package { 'nginx':
                  ensure => installed,
                  }
  }
```

Defining classes

A new class definition:

```
class nginx {
  ...
}
```

Class with Parameters

```
class appserver($domain,
$database) {
  ...
}
```

A class with Parameter and value

```
class hadoop($role = 'node') {
    ...
}
```

Putting classes inside modules

It's a good idea to organize your classes into modules, just as we did with the nginx class. Each class should be stored in the modules/MODULE_NAME/manifests directory, in a file named after the class, with each file containing just one class.

So if we create an nginx::loadbalancer class, the definition should look like this:

```
class nginx::loadbalancer {
    ...
}
```

It should go in the file modules/nginx/manifests/loadbalancer.pp.

The exception is the class named after the module (for example, nginx). This should be in the file modules/nginx/manifests/init.pp.

Declaring classes

There are different ways to declare a class (that is, to create an instance of it and apply it to the current node) once you've defined it. If you don't need to give the class parameters, the simplest way is to use include, as we did before:

```
include nginx
```

Requiring Classes

Alternatively, you can use require. This behaves just like include, except it specifies that everything in the required class must be applied immediately, before Puppet moves on to the next part of the code:

```
require nginx
```

Parameterized Class

If the class does need parameters, declare it like this (a bit like a resource):

```
class { 'cluster_node':
        role => 'master',
}
```

You can include the same class from several different places, and Puppet won't mind. But you can only use a resource-like declaration once (because resources have to be unique).

Why parameterize a class ?

Why not just use a arbitrary and unique variable scope variable and have a class retrieve it?

```
$some_variable
include some_class
# This class will reach outside its own scope, and hope
# it finds a value for $some_variable.
```

Problems

- If you do it this way, you would have to be very careful to document all of your variables that the class will refer to
- There is no standard place you could go to to see what all variables class is referring to

Parameters

Class parameters were added to Puppet in version 2.6.0, to address a need for a standard and visible way to configure clases.

Classes vs Defined Types

A parameterized class looks much like a defined types. What's the difference? Why would you use a class instead of a defined type, or vice versa?

Classes and Defined Types

classes and defined types both bundle a group of different resources into a single named entity that you can create single instances of, with some optional parameters.

Classes vs Defined Types

In older versions of Puppet, classes didn't take parameters, which made the two types more distinct.

However, there are important differences. Classes are **singletons**; that is, Puppet only allows one instance of a class to exist on a node at a time.

Classes or Defined Types

So if you're wondering which to use, consider:

- Will you need to have multiple instances of this on the same node (for example,   a website)? If so, use a defined type.
- Could this cause conflicts with other instances of the same thing on this node (for example, a web server)? If so, use a class.

Lets write some more code...

Creating an ntp Class

Let's build an example class that manages the NTP time service. The class will take an optional parameter specifying an NTP server to sync from.

Ntp module skeleton..

```
$ mkdir modules/ntp
$ cd modules/ntp
$ mkdir manifests templates
```

modules/ntp/manifests/init.pp

```
class ntp($server = undef) {
    package { 'ntp':
            ensure => installed,
    }

    file { '/etc/ntp.conf':
            content => template('ntp/ntp.conf.erb'),
            notify => Service['ntpd'],
     }
     service { 'ntpd':
             ensure => running,
             enable => true,
             require  => [ Package['ntp'], File['/etc/ntp.conf'] ],
     }
}
```

modules/ntp/templates/ntp.conf.erb

```
driftfile /var/lib/ntp/ntp.drift
<% if @server -%>
server <%= @server %> prefer
<% end -%>
server 0.ubuntu.pool.ntp.org
server 1.ubuntu.pool.ntp.org
server 2.ubuntu.pool.ntp.org
server 3.ubuntu.pool.ntp.org
server ntp.ubuntu.com
restrict -4 default kod notrap nomodify nopeer noquery
restrict -6 default kod notrap nomodify nopeer noquery
restrict 127.0.0.1
restrict ::1
```

nodes.pp

```
node 'demo'   {
    class { 'ntp':
        server => 'us.pool.ntp.org',
    }
}
```

apply

```
$ puppet apply /vagrant/puppet/manifests/
```