

Exercise – Puppet Manifests

Objective

Part 1 - Create a stand-alone manifest and apply it

Connect to your puppet agent machine.

We are going to write a simple manifest which will add a new user to the system and install a package.

1. Create a new file called site.pp
2. In the file enter the following:

```
user { 'jane':  
  ensure      => 'present',  
  home        => '/home/jane',  
  managehome  => true,  
  shell       => '/bin/bash',  
}
```

This is a resource declaration. It will check for a new user called Jane. If one doesn't exist then puppet will create one. It also sets up the default shell Jane uses and her home directory.

3. Save the file
4. On the command line run the command

```
$ sudo puppet apply -t site.pp
```

5. You should see that a user 'jane' has been created with a new home directory in /home/jane. To check this has been created, run the following

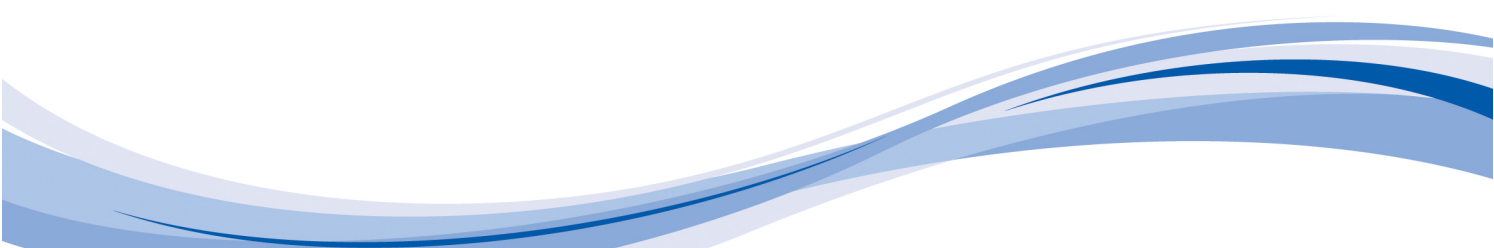
```
$ id jane
```

You can reverse engineer the commands needed to create new users with the following:

```
$ sudo puppet resource user jane
```

6. Drift is a huge problem with servers. Puppet can notice and correct drift when reapplying the manifest. To test this we can intentionally cause some drift. First re-apply your puppet manifest

```
$ sudo puppet apply -t site.pp
```



Now change the user slightly with the following command. **Don't worry about the error it states**, that's not important to us, we just want to change something in the user settings.

```
$ sudo usermod -d /home/extra/jane -m jane
```

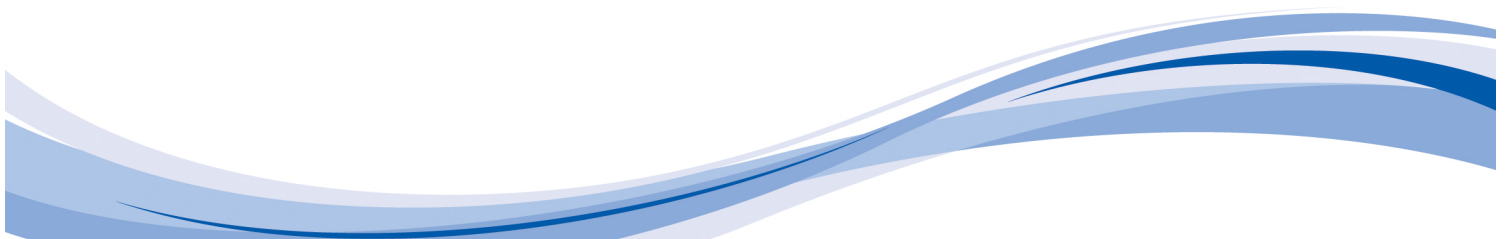
7. What is important is that puppet notices a change has occurred. Run the puppet resource command and check that the change in user directory has occurred

```
$ sudo puppet resource user jane
user { 'jane':
  ensure      => 'present',
  gid         => '1001',
  home        => '/home/extra/jane',
  password    => '!',
  password_max_age => '99999',
  password_min_age => '0',
  shell       => '/bin/bash',
  uid         => '1001',
}
```

8. Re-run the puppet apply command to correct the 'drift'

```
$ sudo puppet apply -t site.pp
Info: Loading facts
Notice: Compiled catalog for agent1.eu-west-1.compute.internal
in environment production in 0.06 seconds
Info: Applying configuration version '1443014472'
Notice: /Stage[main]/Main/User[jane]/home: home change
```

9. Finally, delete the site.pp file, we won't need this again!



Part 2 - Create a module on the puppet master

A module is a collection of manifests and files that are used to set up a machine. These are stored in the modules directory. We will be creating a module that will install apache and show a test page.

1. Your puppet master server manages modules. **Connect to the puppet master** and move to the modules directory.

(Please note, when you are in production you should be writing your modules and testing them on an agent not connected to your live system, then moving everything across to the master using some kind of source control system. We are avoiding this extra overhead for the exercise, but running untested code on the master is generally a very bad plan!)

```
$ cd /etc/puppetlabs/code/modules
$ sudo bash                                #become superuser
```

2. Create a new module called apache

```
$ puppet module generate qa-apache
```

This will generate for you a module using the username “qa”. The directory structure will look something like this.

```
.
├── apache
│   ├── examples
│   │   └── init.pp
│   ├── Gemfile
│   ├── manifests
│   │   └── init.pp
│   ├── metadata.json
│   ├── Rakefile
│   ├── README.md
│   └── spec
│       ├── classes
│       │   └── init_spec.rb
│       └── spec_helper.rb
```

3. Open the file in the **manifests** directory called **init.pp**

4. Add the following to the class part of the file. Check your brackets!

```
class apache {  
  package { 'apache2':  
    ensure => 'present',  
  }  
  
  file { '/var/www':  
    ensure => directory,  
  }  
}
```

This **manifest** does two things:

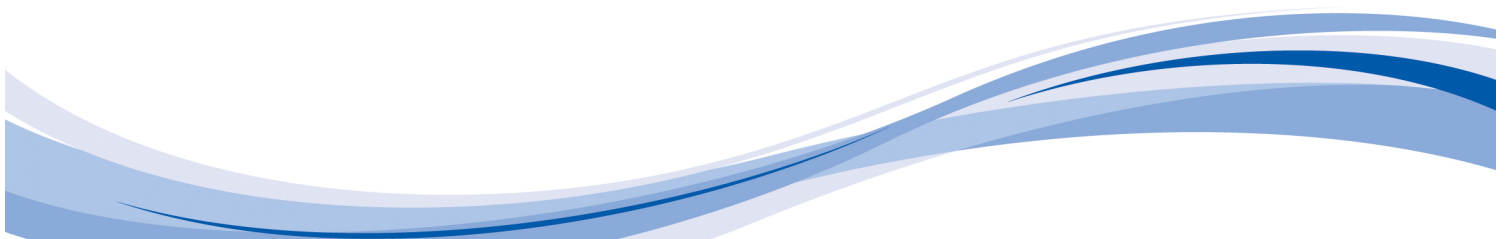
1-st ensures that the apache2 package has been installed.

2-nd ensures that the /var/www directory exists.

5. Add two file resources to your file before the final close bracket. These refer to the apache configuration file and a home page. It is no good having a webserver without anything to show for it!

```
file { '/etc/apache2/apache2.conf':  
  ensure => file,  
  owner => 'root',  
  group => 'root',  
  source => 'puppet:///modules/apache/apache2.conf',  
  require => Package['apache2'],  
}  
  
file { '/var/www/html/index.html':  
  ensure => file,  
  source => 'puppet:///modules/apache/index.html',  
}
```

This will pull in two files from the apache/files directory and add them to the server. We can omit the files directory as puppet will automatically assume it should be using that.



6. Finally, we want to start the service, so add the following at the bottom of the file:

```
service { 'apache2':  
  ensure => running,  
  subscribe => File['/etc/apache2/apache2.conf'],  
}
```

Your final file should contain:

```
class apache {  
  package { 'apache2':  
    ensure => 'present',  
  }  
  
  file { '/var/www':  
    ensure => directory,  
  }  
  
  file { '/etc/apache2/apache2.conf':  
    ensure => file,  
    owner => 'root',  
    group => 'root',  
    source => 'puppet:///modules/apache/apache2.conf',  
    require => Package['apache2'],  
  }  
  
  file { '/var/www/html/index.html':  
    ensure => file,  
    source => 'puppet:///modules/apache/index.html',  
  }  
  
  service { 'apache2':  
    ensure => running,  
    subscribe => File['/etc/apache2/apache2.conf'],  
  }  
}
```

7. Save your file and go back to the command line.
8. We now need to create two files, index.html and apache2.conf in the files directory

Create a new file called index.html in the files directory

```
#make the directory
£ mkdir apache/files
#open the file
£ vim apache/files/index.html
```

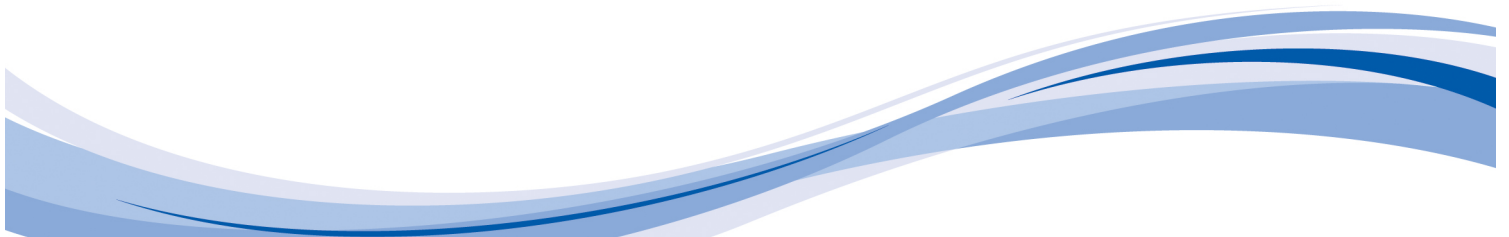
and add the following to it:

```
<html>
<head>
<title>Puppet test</title>
</head>
<body>
<h1>Hello World!</h1>
</body>
</html>
```

Save and exit.

9. To get an apache config file we will cheat and install apache, grab the file, then uninstall it again.

```
$ sudo apt-get install -y apache2
$ cp /etc/apache2/apache2.conf
/etc/puppetlabs/code/modules/apache/files/apache2.conf
$ sudo apt-get remove -y apache2
$ rm -rf /var/www
$ sudo apt-get purge -y apache2
```



Your directory structure should now look like this for your module:

```
.
└─ apache
    ├── examples
    │   └─ init.pp
    ├── files
    │   ├── apache2.conf
    │   └─ index.html
    ├── Gemfile
    ├── manifests
    │   └─ init.pp
    ├── metadata.json
    ├── Rakefile
    ├── README.md
    └─ spec
        ├── classes
        │   └─ init_spec.rb
        └─ spec_helper.rb
```

10. We can parse the files to check our syntax is correct with the puppet parser

```
$ puppet parser validate apache/manifests/init.pp
```

If nothing is output then there are no errors in your code

Part 3 - Pin a module to a node

We should now be able to pin this module to the node that we have connected to the puppet master.

1. Go to the puppet master dashboard.
2. Click on Nodes, then Classification
3. Add a new node group, we'll call it "webservers"

Classification

Create, edit, and remove node groups here.

▼ [Add group...](#)

Parent name	Group name	Environment
All Nodes	webservers	production
<input type="checkbox"/> Environment group ?		
Description (optional)		
apache webservers		
		Add

4. Click add, and then go to your new group.
5. On the first screen for 'rules' add a new rule where 'hostname' ~ 'webserver*'
This should set up a regular expression matching to all the nodes with a hostname starting with webserver
Click add rule
6. Next, click on the 'Classes' tab
7. In the text box type apache then click "add class" to assign that class to the node group.
8. Click "Commit 2 changes" at the bottom
9. Now we could wait for half an hour for the agent to check in to the master and find out what changes it has to apply. Or we can speed up the process. On the **agent machine** type:

```
$ puppet agent -t
```

You should see the output of the node completing the actions.

10. Test everything is working by pointing your browser at the **public IP** for your **agent** node. You should see your webpage saying "**Hello World!**"

Change your **index.html** file on the master server to say something else, save the file and then re-run puppet agent on the agent node to see your changes being applied and your website updated!

Stretcher

- Create some agents with those names and check your rule is working
- Have a look at: <http://www.codelord.net/2010/12/19/using-puppet-to-automatically-configure-new-ec2-instances/> and see if you can get puppet to auto sign your new certificates. You will need to modify your firewall rules for the puppet master to make it safe (it should only accept connections on 8140 from your machines, rather than anywhere)
- <http://www.logicworks.net/blog/2015/04/aws-puppet-devops-automation-configuration/> is a rather good blog post for how to use puppet inside aws

