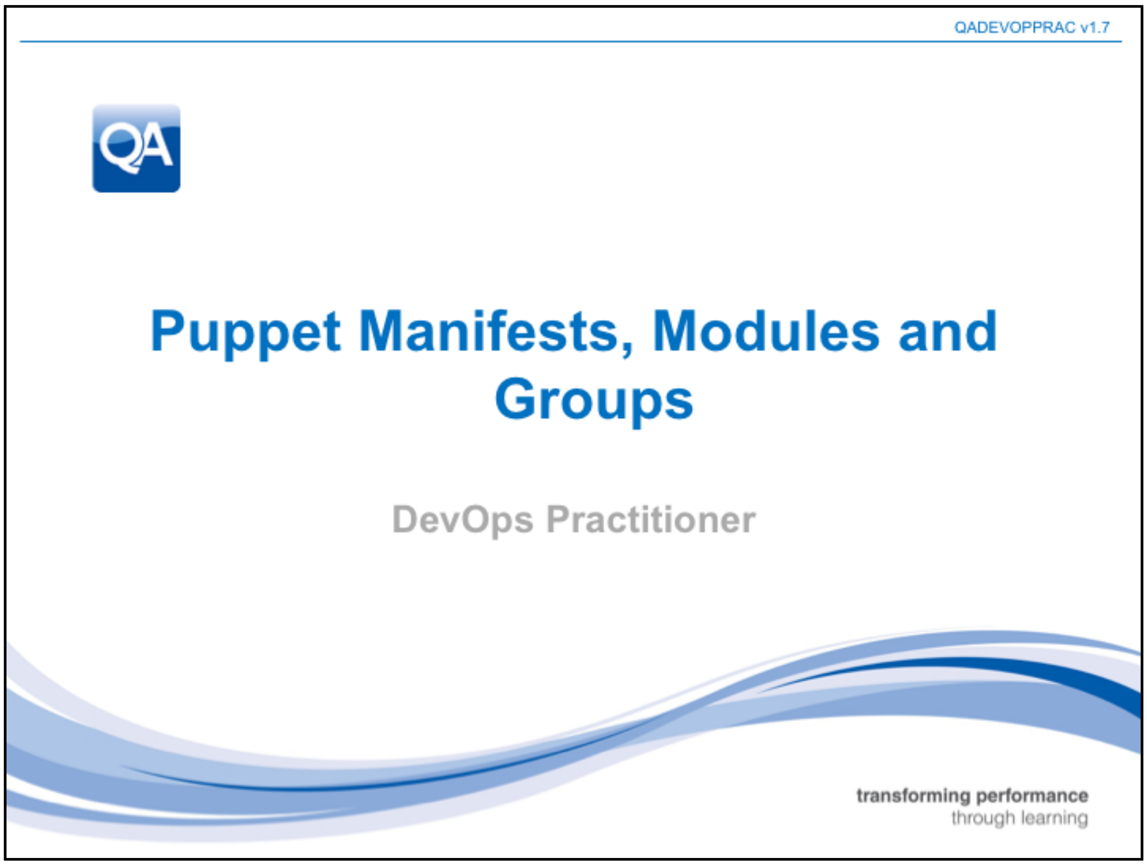
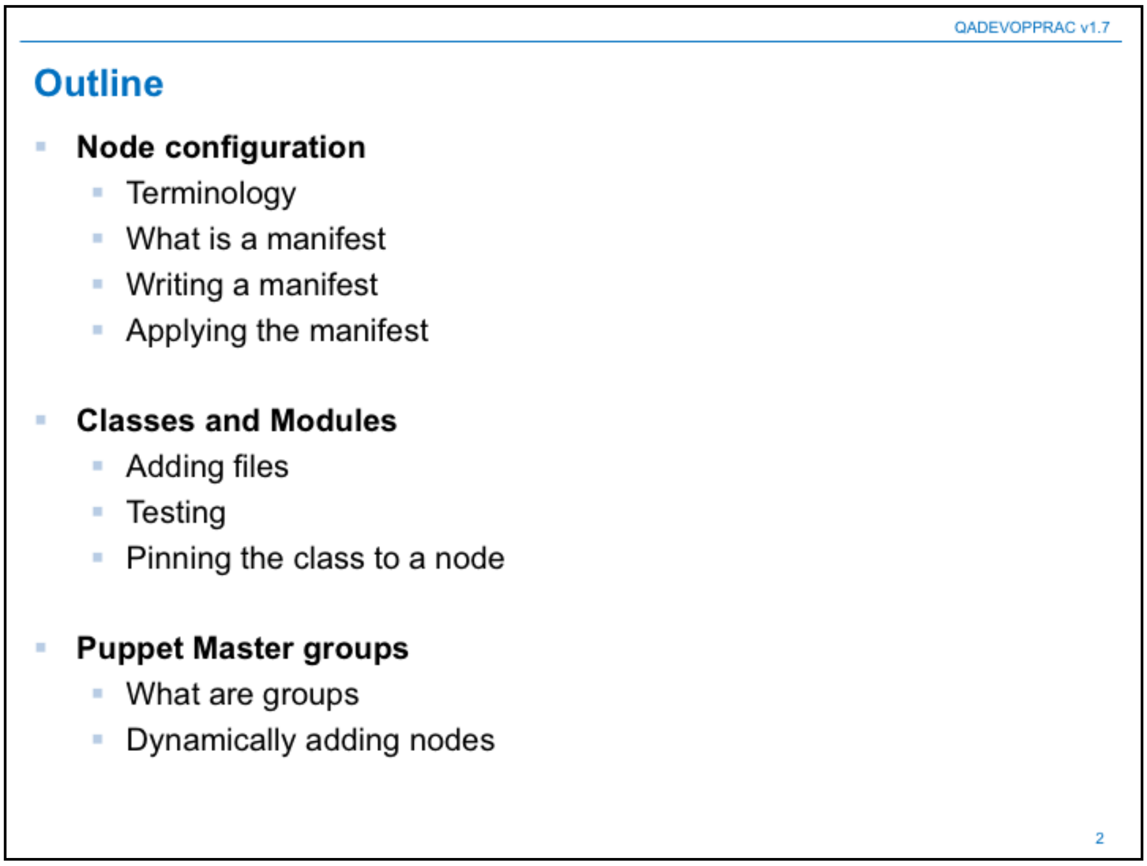


Puppet Manifests, Modules and Groups

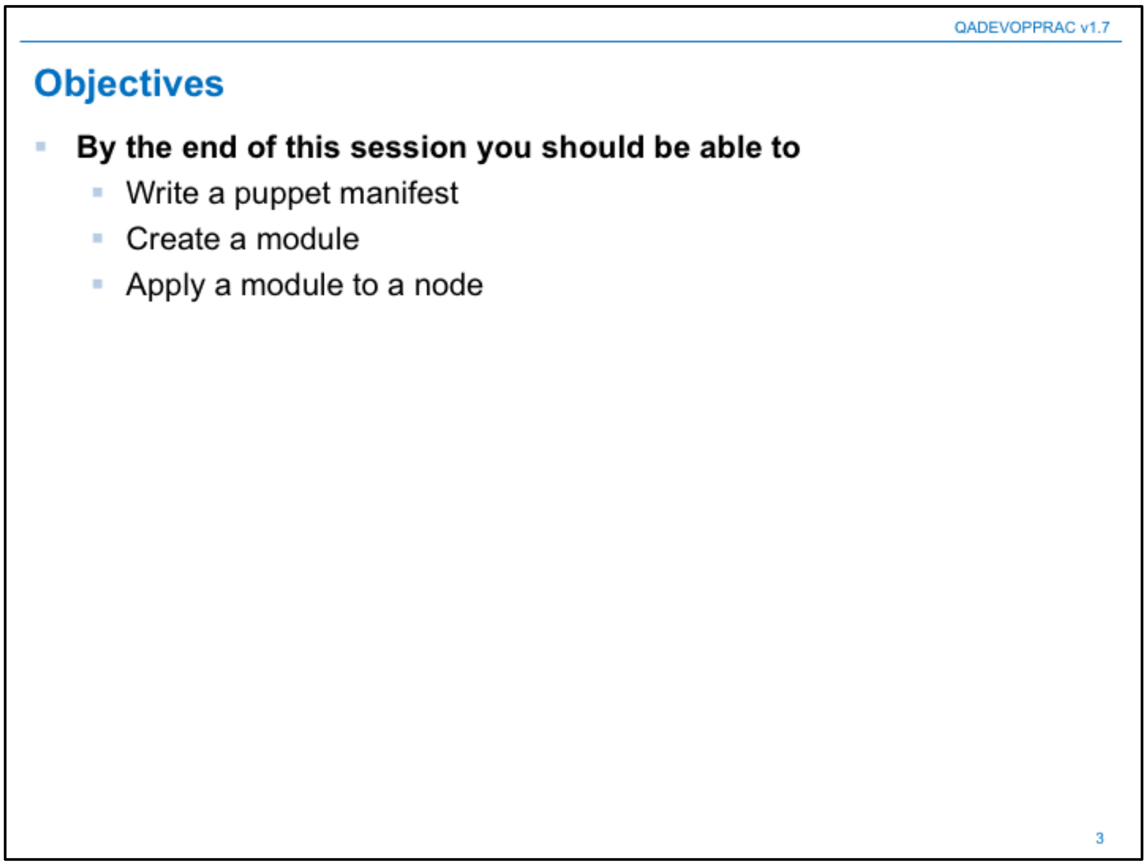
DevOps Practitioner

transforming performance
through learning



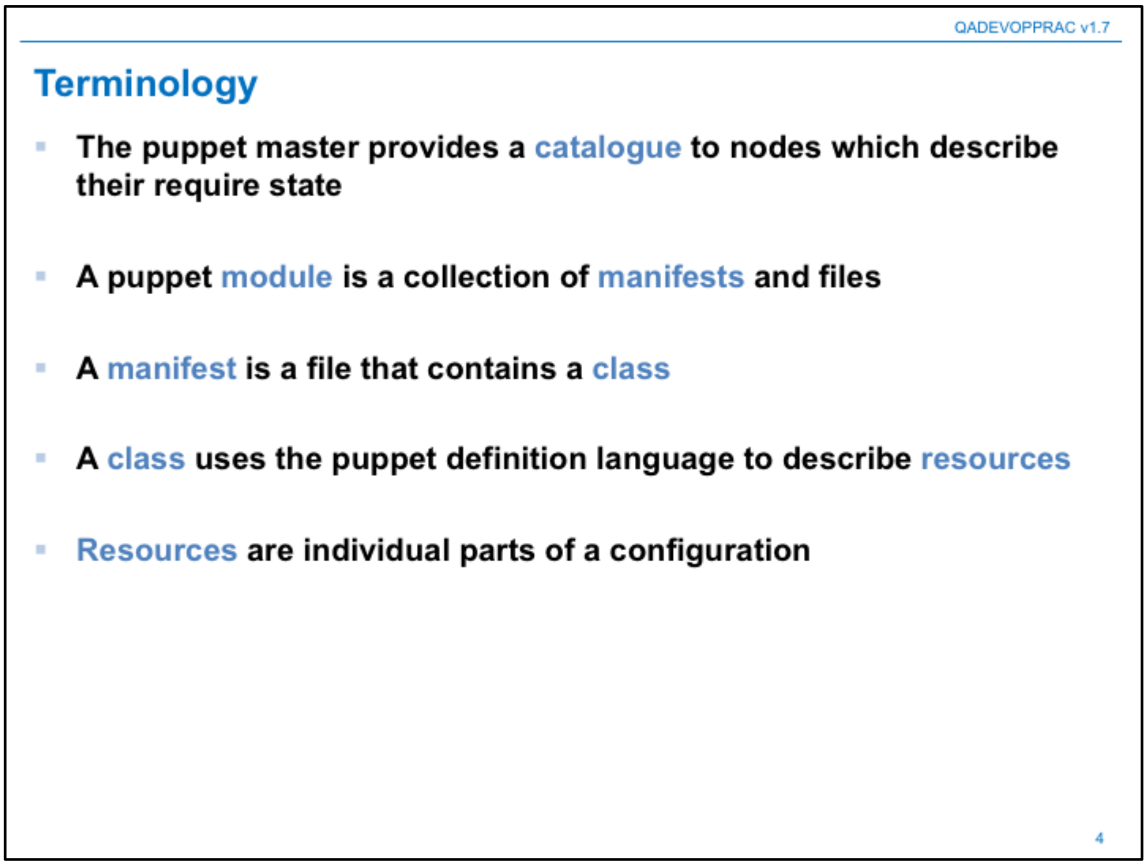
Outline

- **Node configuration**
  - Terminology
  - What is a manifest
  - Writing a manifest
  - Applying the manifest
- **Classes and Modules**
  - Adding files
  - Testing
  - Pinning the class to a node
- **Puppet Master groups**
  - What are groups
  - Dynamically adding nodes



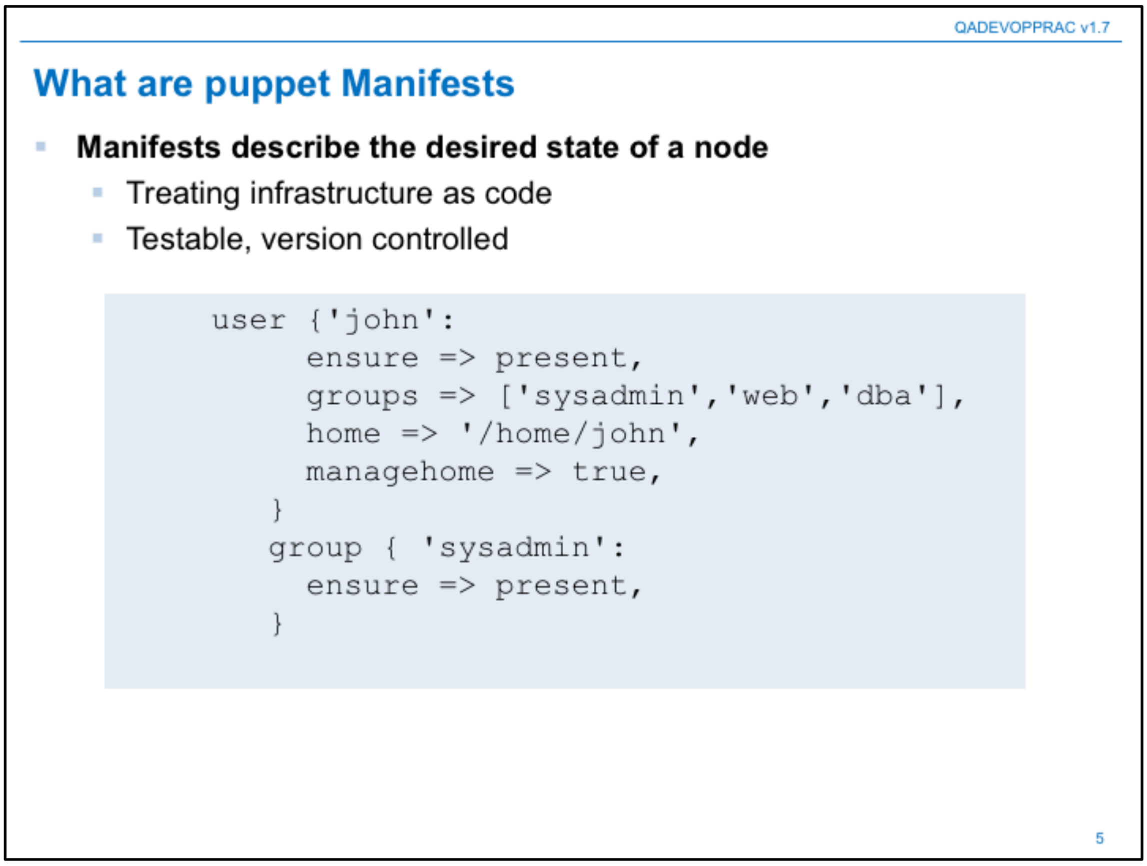
Objectives

- **By the end of this session you should be able to**
  - Write a puppet manifest
  - Create a module
  - Apply a module to a node



Terminology

- The puppet master provides a catalogue to nodes which describe their require state
- A puppet module is a collection of manifests and files
- A manifest is a file that contains a class
- A class uses the puppet definition language to describe resources
- Resources are individual parts of a configuration



What are puppet Manifests

- **Manifests describe the desired state of a node**
  - Treating infrastructure as code
  - Testable, version controlled

```
user {'john':
     ensure => present,
     groups => ['sysadmin','web','dba'],
     home => '/home/john',
     managehome => true,
   }
   group { 'sysadmin':
     ensure => present,
   }
```

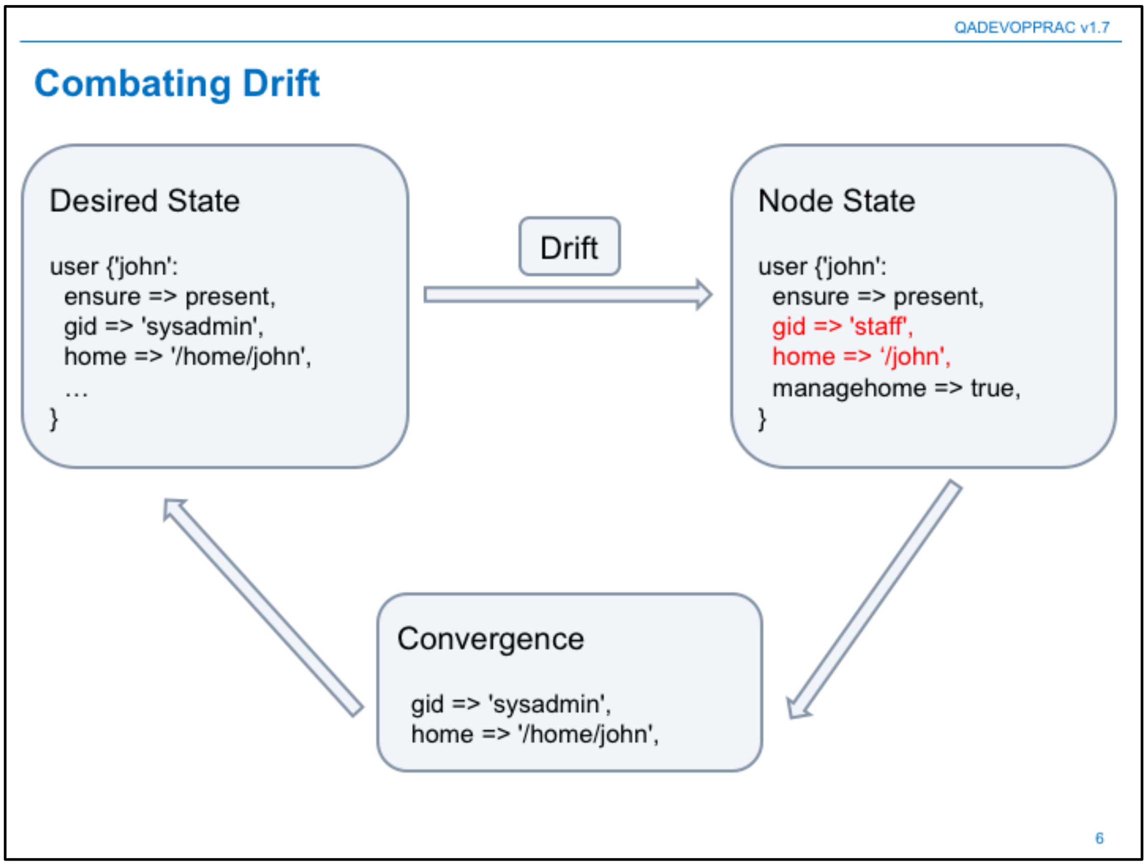
5

Describes state of the node.

It’s a collection of resources (stored in text file)

Its can be stored in version control and be tested.

So TDD is for the infrastructure code as well



Combating Drift

Desired State

```
user {'john':
  ensure => present,
  gid => 'sysadmin',
  home => '/home/john',
  …
}
```

Node State

```
user {'john':
  ensure => present,
  gid => 'staff',
  home => '/john',
  managehome => true,
}
```

Convergence

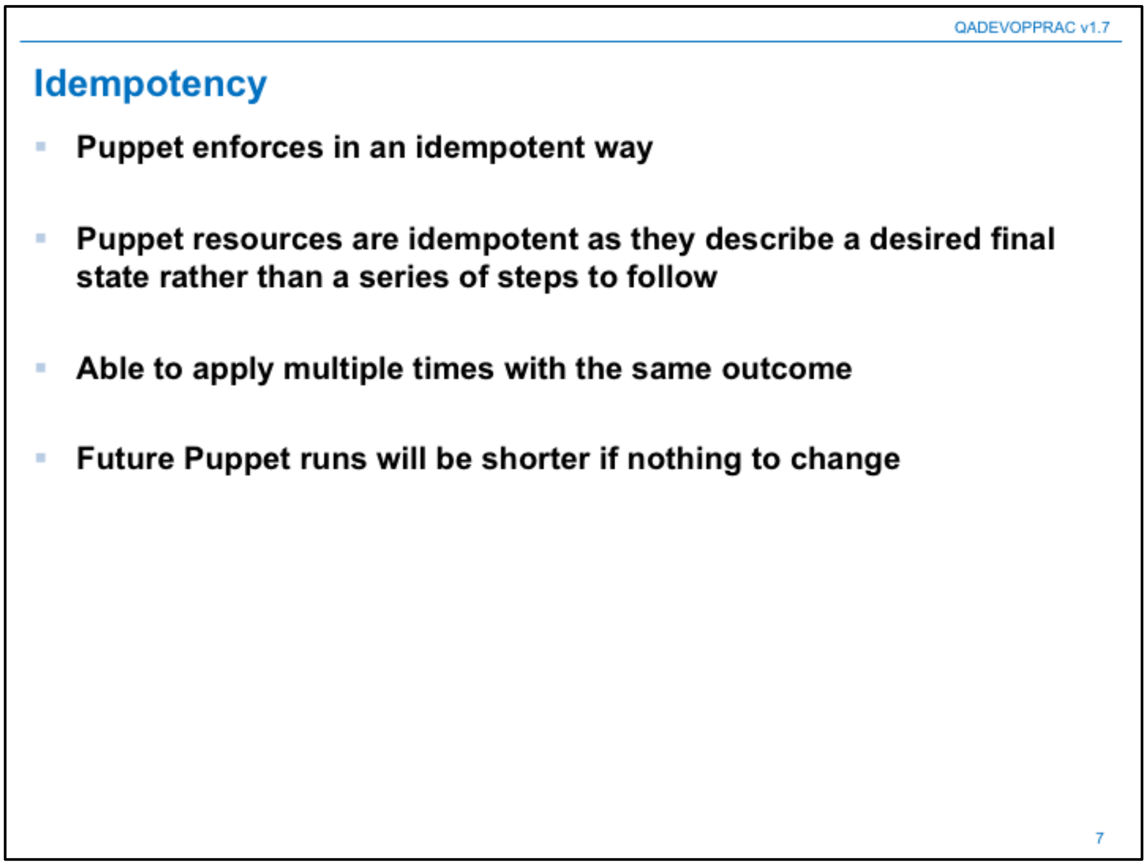
```
gid => 'sysadmin',
home => '/home/john',
```

The main thing we want to do – come back drift.

Lets say desired data are changed. We don't want this to be saved or continue happening.

Every 30min agent checks with master, master gives catalogue with original class and it will be recovered back to the node as it is written in manifest

That's a job of config management (you can do this on local machines as well)



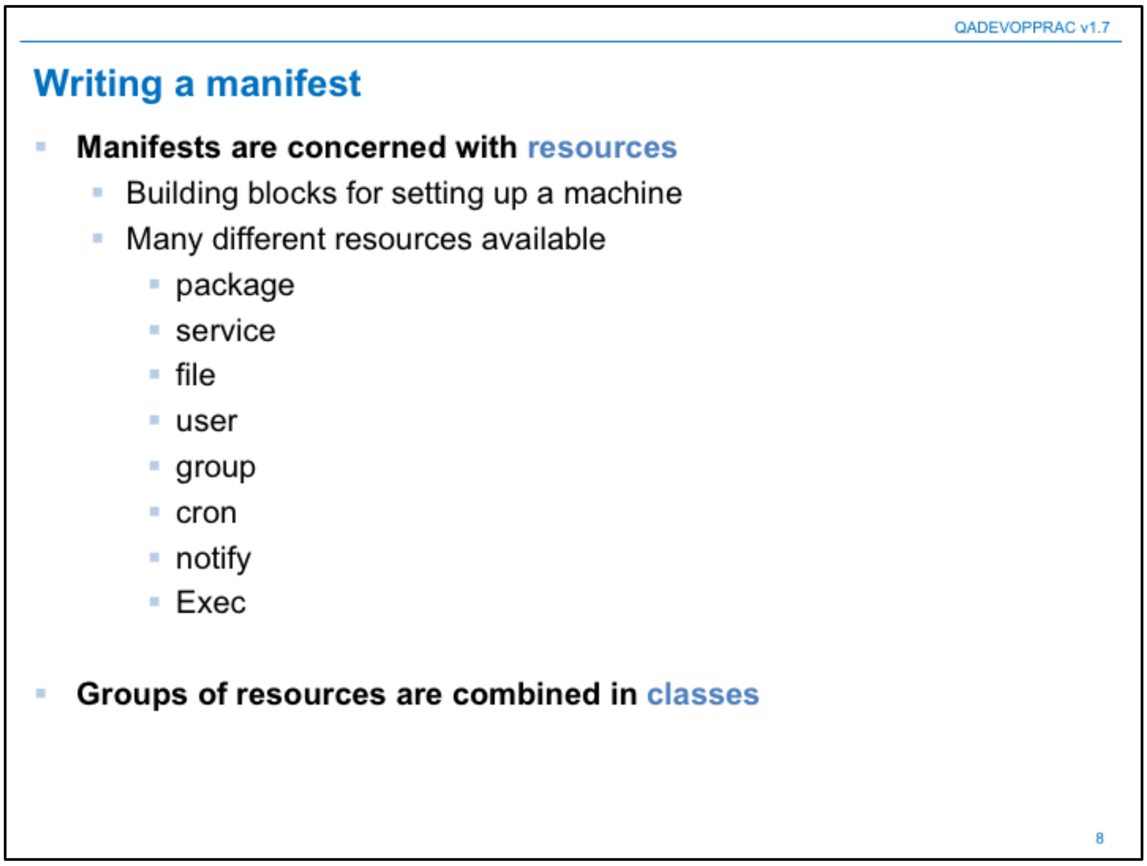
QADEVOPPRAC v1.7

Idempotency

- Puppet enforces in an idempotent way
- Puppet resources are idempotent as they describe a desired final state rather than a series of steps to follow
- Able to apply multiple times with the same outcome
- Future Puppet runs will be shorter if nothing to change

7

Idempotency – if something runs once: we know beginning and end points. So it can run N times, and it will give same end state. (e.g. installing the package: if it there, you don't need to install. If is not – install it)



QADEVOPPRAC v1.7

Writing a manifest

- **Manifests are concerned with resources**
  - Building blocks for setting up a machine
  - Many different resources available
    - package
    - service
    - file
    - user
    - group
    - cron
    - notify
    - Exec
- **Groups of resources are combined in classes**

8

Package – to be installed

Service – to be started

File – to be in

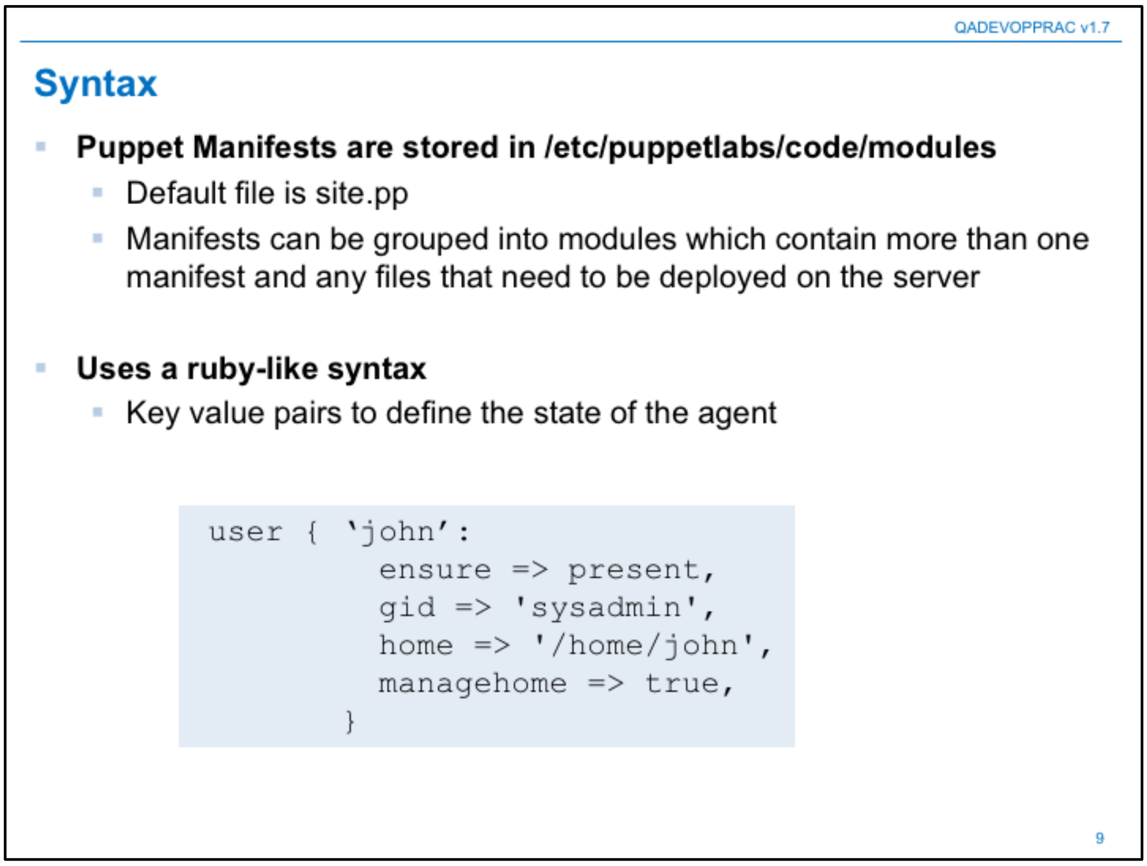
User – witch user should exist

Group – what group user belongs to

Cron – cron jobs

Notify -

Exec – if something fails it will run script or shell command



QADEVOPPRAC v1.7

Syntax

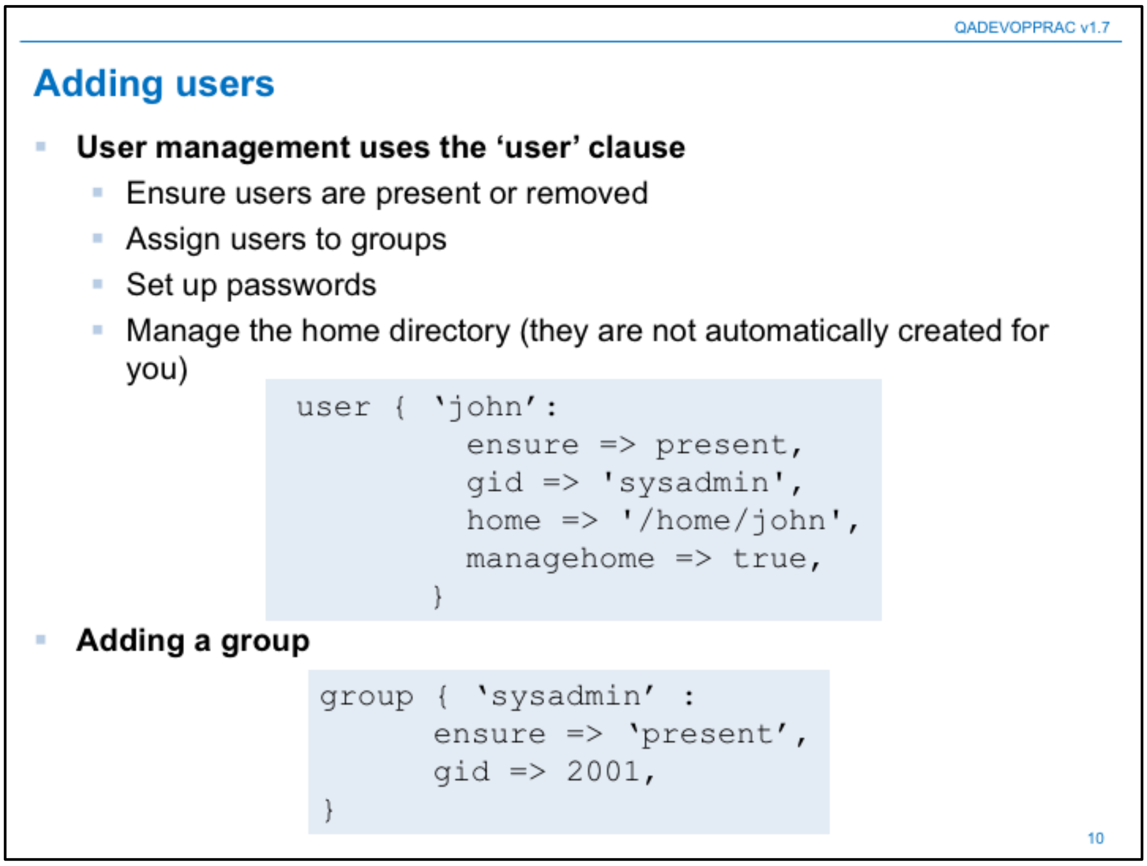
- **Puppet Manifests are stored in /etc/puppetlabs/code/modules**
  - Default file is site.pp
  - Manifests can be grouped into modules which contain more than one manifest and any files that need to be deployed on the server
- **Uses a ruby-like syntax**
  - Key value pairs to define the state of the agent

```
user { ‘john’:
         ensure => present,
         gid => 'sysadmin',
         home => '/home/john',
         managehome => true,
       }
```

9

The puppet cookbook is a very good resource for manifests – www.puppetcookbook.com

Manifest (unlike modules) only runs locally. They will be ran and tested by the agent.



Adding users

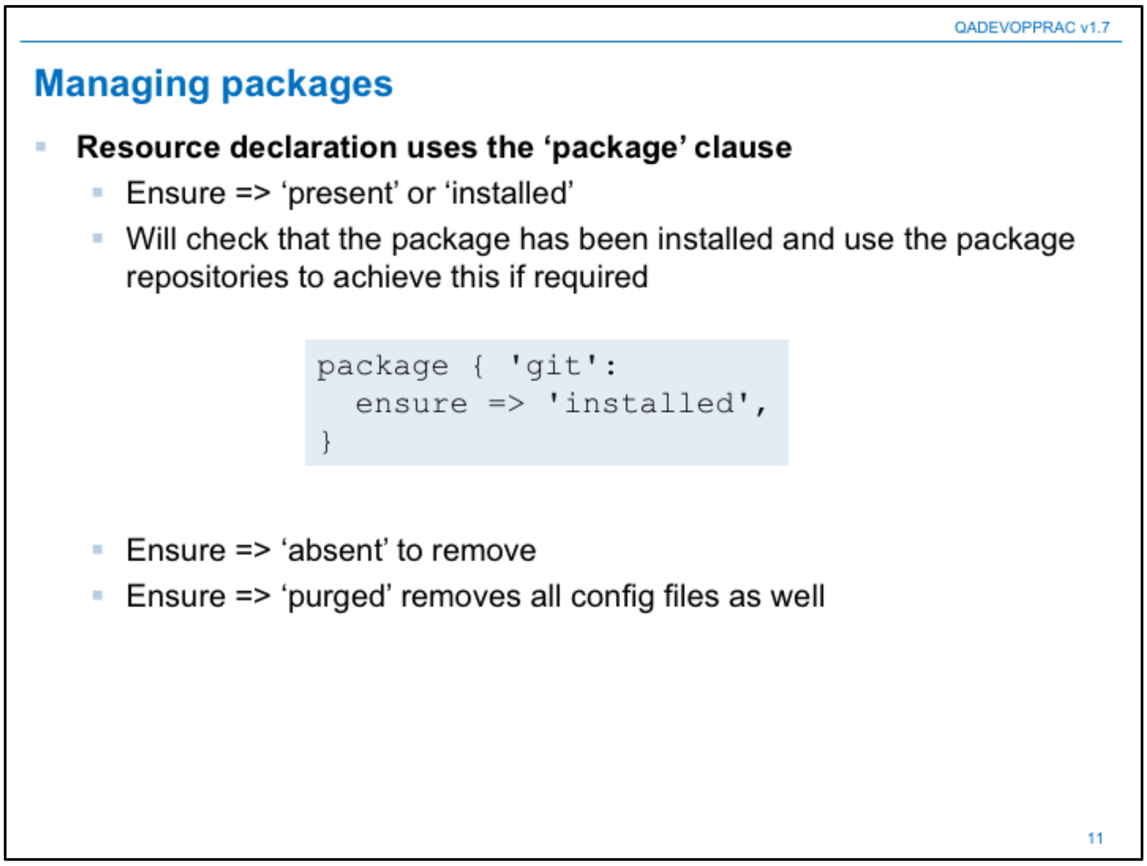
- **User management uses the 'user' clause**
  - Ensure users are present or removed
  - Assign users to groups
  - Set up passwords
  - Manage the home directory (they are not automatically created for you)

```
user { 'john':
         ensure => present,
         gid => 'sysadmin',
         home => '/home/john',
         managehome => true,
       }
```

- **Adding a group**

```
group { 'sysadmin' :
      ensure => 'present',
      gid => 2001,
}
```

10

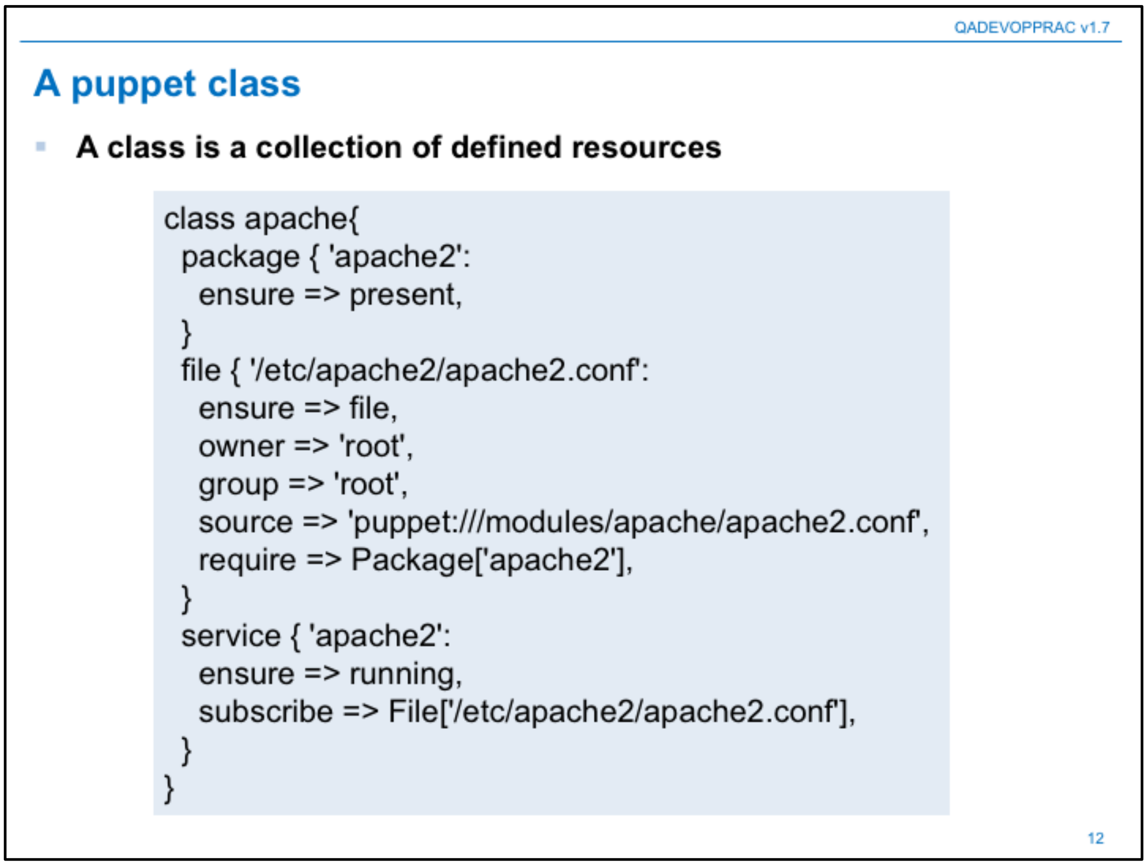


Managing packages

- **Resource declaration uses the ‘package’ clause**
  - Ensure => ‘present’ or ‘installed’
  - Will check that the package has been installed and use the package repositories to achieve this if required

```
package { 'git':
  ensure => 'installed',
}
```

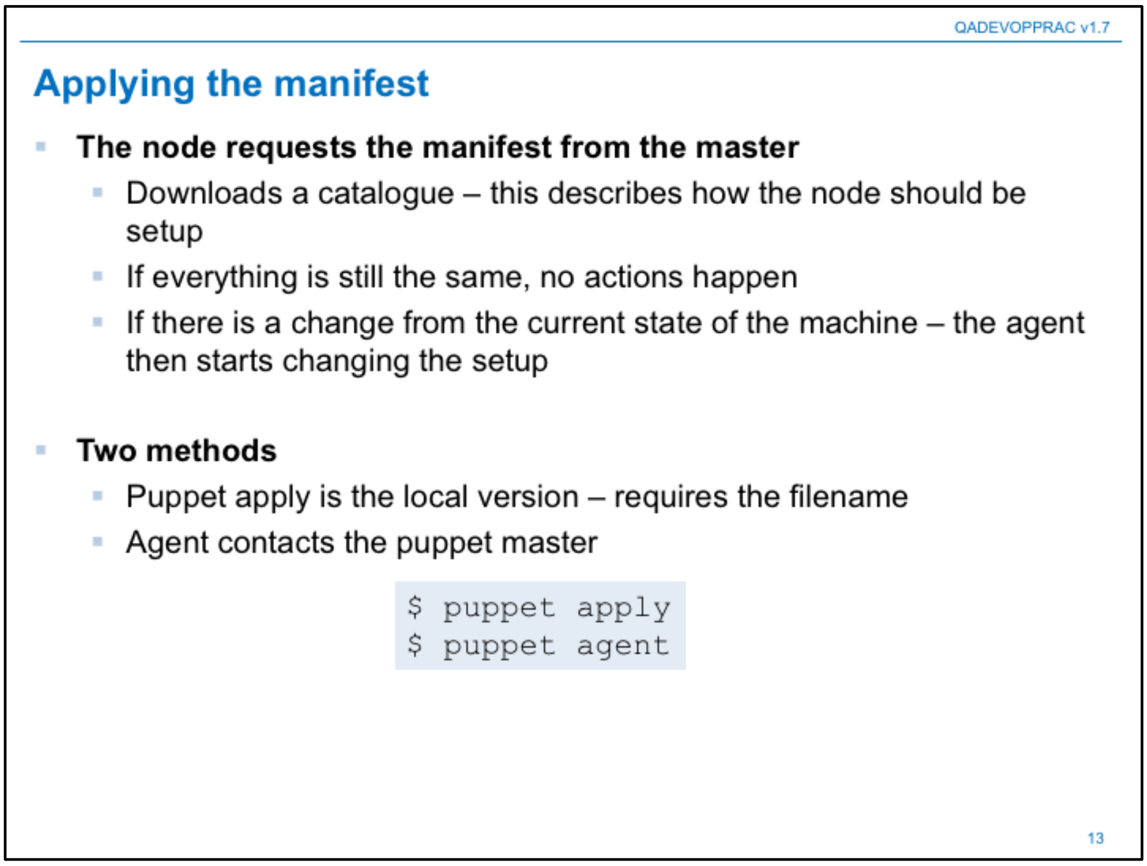
  - Ensure => ‘absent’ to remove
  - Ensure => ‘purged’ removes all config files as well



A puppet class

- A class is a collection of defined resources

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



QADEVOPPRAC v1.7

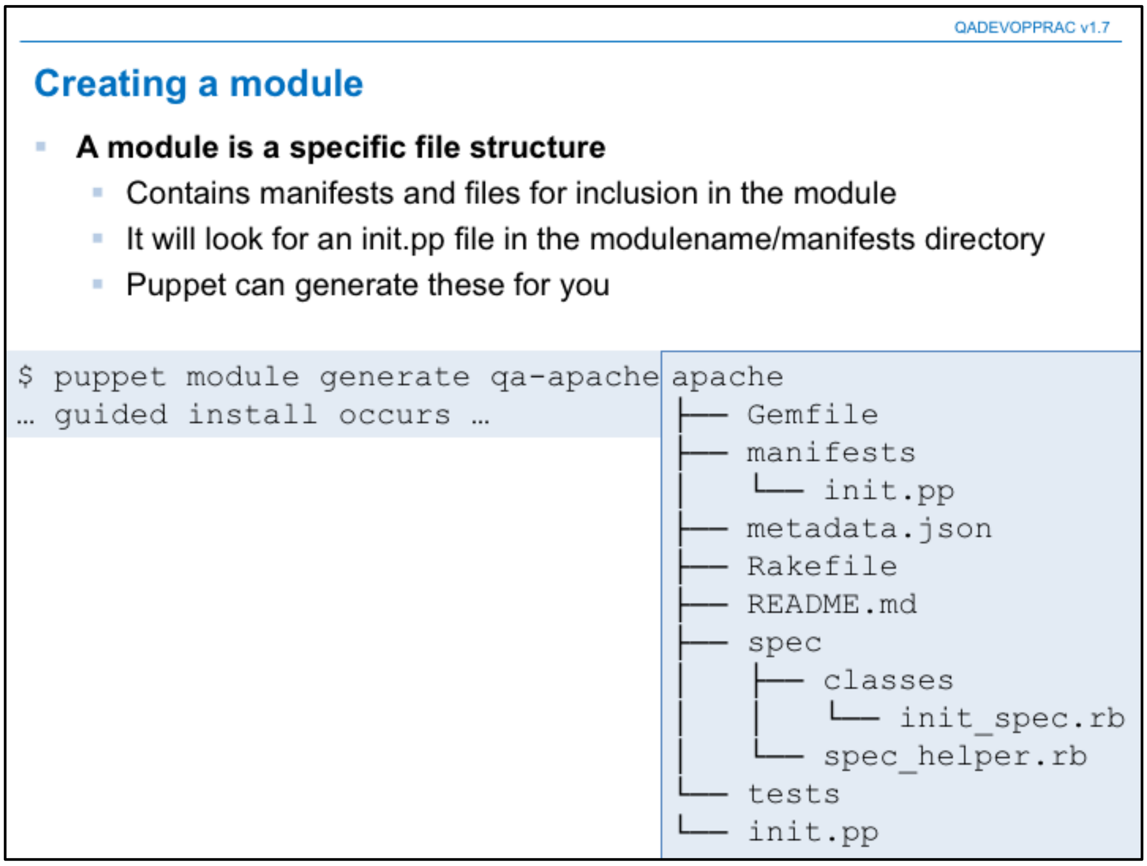
Applying the manifest

- **The node requests the manifest from the master**
  - Downloads a catalogue – this describes how the node should be setup
  - If everything is still the same, no actions happen
  - If there is a change from the current state of the machine – the agent then starts changing the setup
- **Two methods**
  - Puppet apply is the local version – requires the filename
  - Agent contacts the puppet master

```
$ puppet apply
$ puppet agent
```

13

```
$ puppet apply – for local
$ puppet agent – doing by agent
```

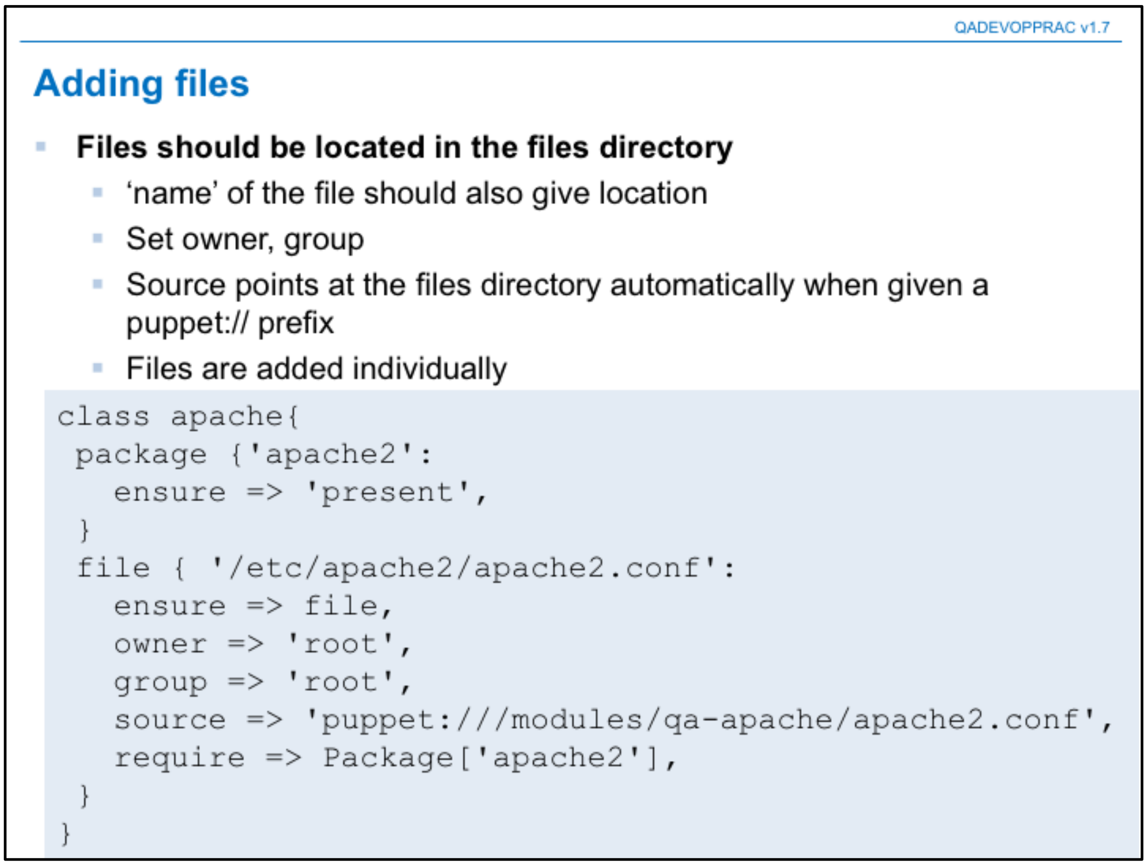


Creating a module

- **A module is a specific file structure**
  - Contains manifests and files for inclusion in the module
  - It will look for an init.pp file in the modulename/manifests directory
  - Puppet can generate these for you

```
$ puppet module generate qa-apache
… guided install occurs …
```

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

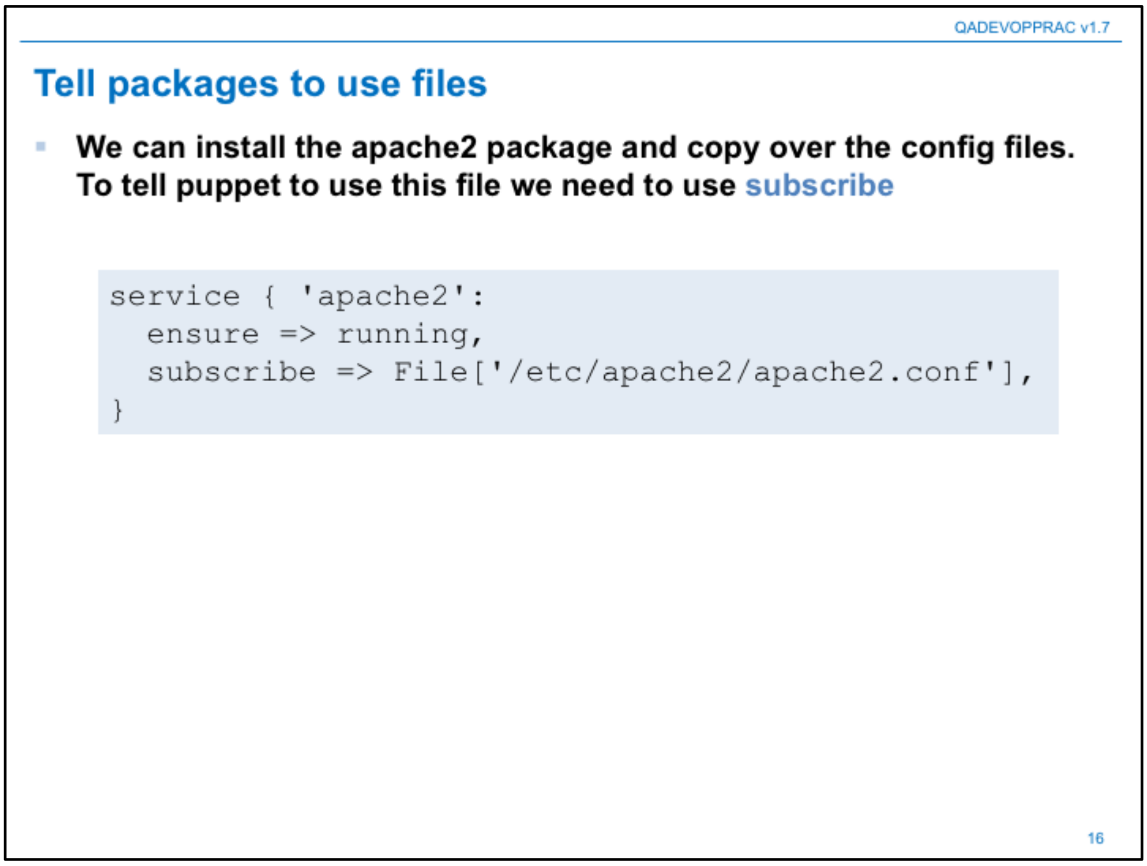


QADEVOPPRAC v1.7

Adding files

- **Files should be located in the files directory**
  - ‘name’ of the file should also give location
  - Set owner, group
  - Source points at the files directory automatically when given a puppet:// prefix
  - Files are added individually

```
class apache{
 package {'apache2':
   ensure => 'present',
 }
 file { '/etc/apache2/apache2.conf':
   ensure => file,
   owner => 'root',
   group => 'root',
   source => 'puppet:///modules/qa-apache/apache2.conf',
   require => Package['apache2'],
 }
}
```



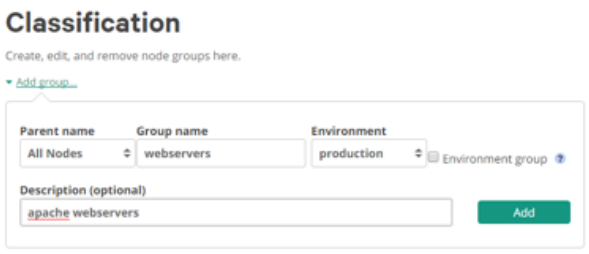
Tell packages to use files

- We can install the apache2 package and copy over the config files. To tell puppet to use this file we need to use subscribe

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

Pinning a manifest to an agent

- **On the puppet master UI**
  - Click on node, then classification
  - Add a node group, give it a name
  - In the group find your node, pin this node to the group
  - In the classes tab find your class, add the class to this group
  - Save your configuration

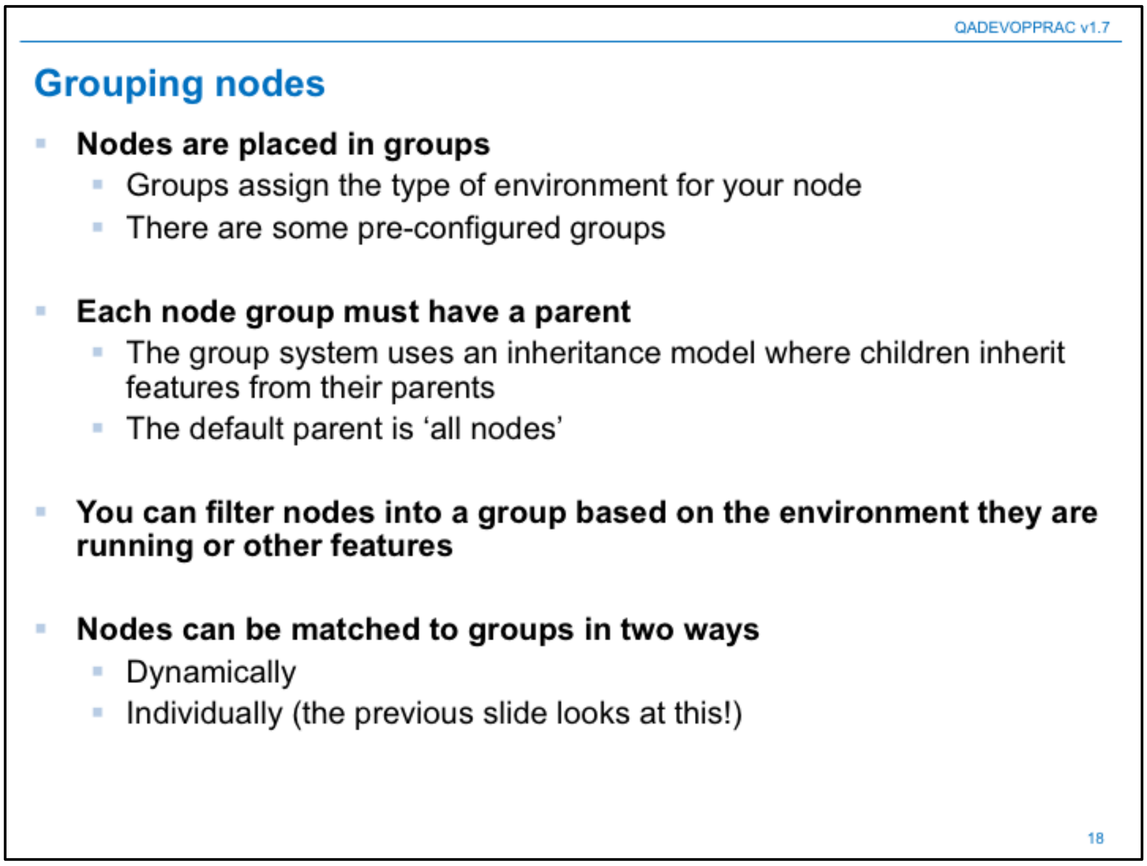


Classification

Create, edit, and remove node groups here.

Add group...

Parent name: All Nodes
Group name: webservers
Environment: production
Environment group
Description (optional): apache webservers

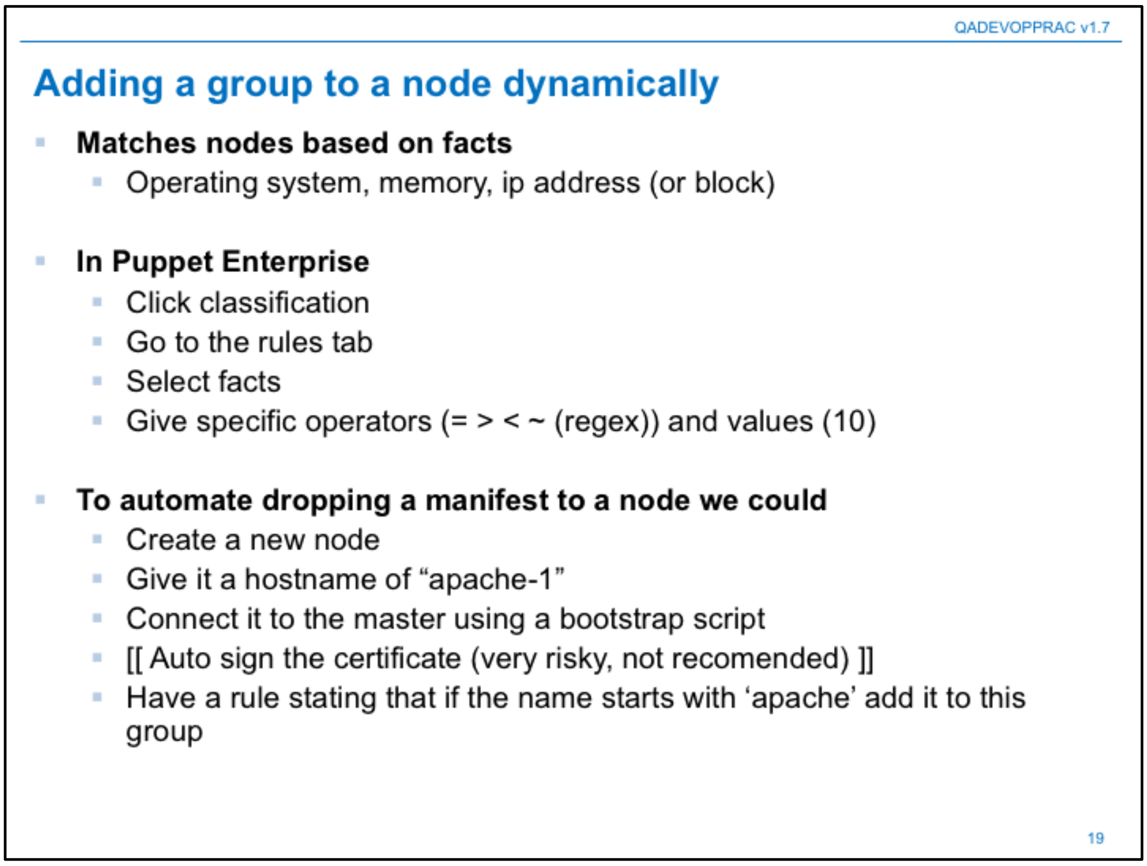


QADEVOPPRAC v1.7

Grouping nodes

- **Nodes are placed in groups**
  - Groups assign the type of environment for your node
  - There are some pre-configured groups
- **Each node group must have a parent**
  - The group system uses an inheritance model where children inherit features from their parents
  - The default parent is ‘all nodes’
- **You can filter nodes into a group based on the environment they are running or other features**
- **Nodes can be matched to groups in two ways**
  - Dynamically
  - Individually (the previous slide looks at this!)

18

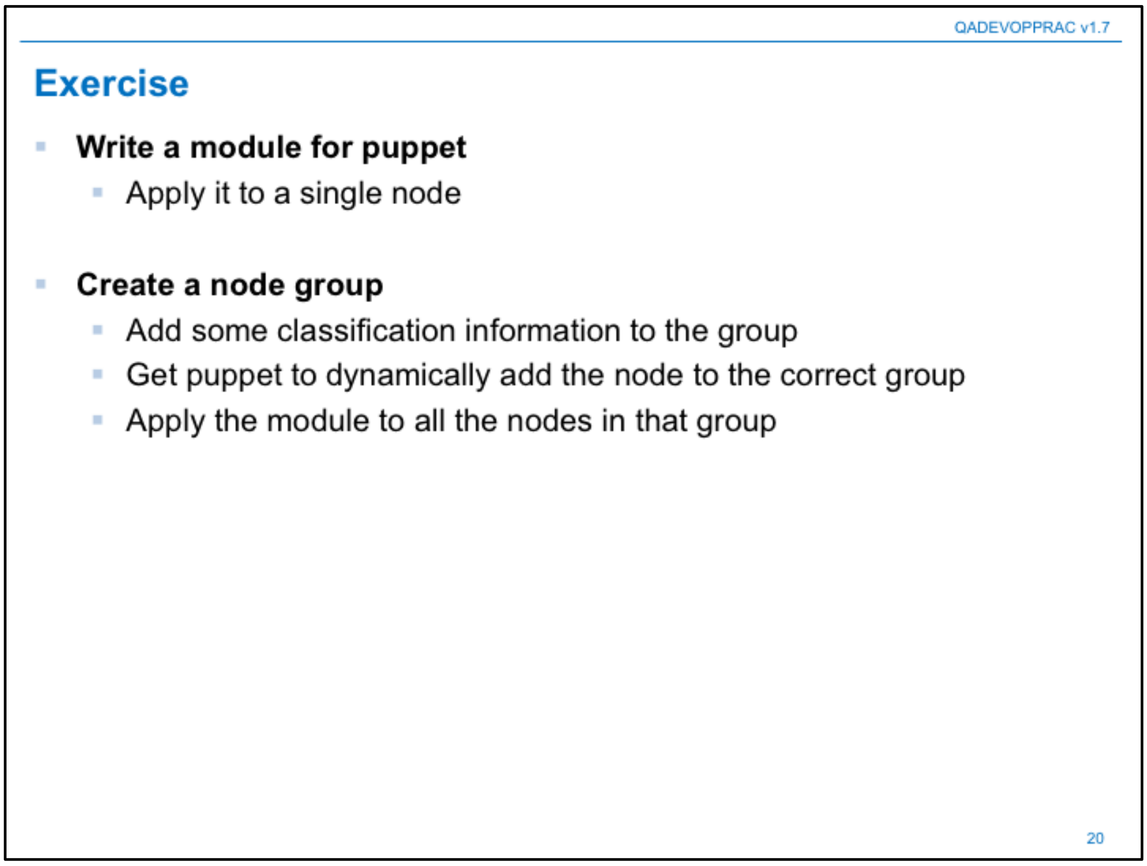


QADEVOPPRAC v1.7

Adding a group to a node dynamically

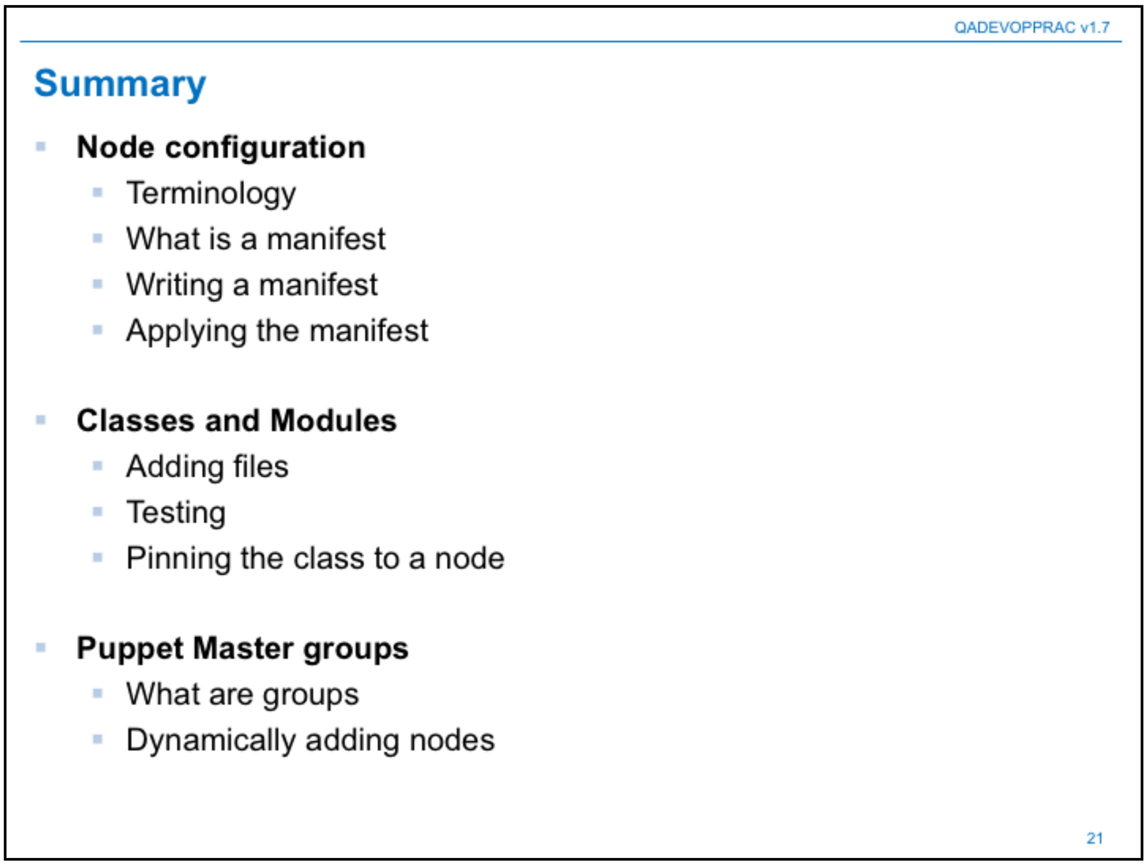
- **Matches nodes based on facts**
  - Operating system, memory, ip address (or block)
- **In Puppet Enterprise**
  - Click classification
  - Go to the rules tab
  - Select facts
  - Give specific operators (= > < ~ (regex)) and values (10)
- **To automate dropping a manifest to a node we could**
  - Create a new node
  - Give it a hostname of "apache-1"
  - Connect it to the master using a bootstrap script
  - [[Auto sign the certificate (very risky, not recomended)]]
  - Have a rule stating that if the name starts with 'apache' add it to this group

19



Exercise

- **Write a module for puppet**
  - Apply it to a single node
- **Create a node group**
  - Add some classification information to the group
  - Get puppet to dynamically add the node to the correct group
  - Apply the module to all the nodes in that group



QADEVOPPRAC v1.7

Summary

- **Node configuration**
  - Terminology
  - What is a manifest
  - Writing a manifest
  - Applying the manifest
- **Classes and Modules**
  - Adding files
  - Testing
  - Pinning the class to a node
- **Puppet Master groups**
  - What are groups
  - Dynamically adding nodes

21