Resource, RAL and Manifests

Resource, RAL and Manifests

Resources are the fundamental building blocks of Puppet

Resource

How puppet looks at your infrastructure

Resource

How puppet looks at your infrastructure

- User/ Group
- File
- Package
- Directory
- Service
- Cron
- Shell Command

How are Resources Written ?

How are Resources Written ?

Puppet's own Domain Specifc Language (DSL)

Writing Resources

Syntax

```
<type> { <title>:

   attribute1 => value1,

   attribute2 => value2,

}
```

Example

```
package { "apache2":

        ensure => present,

}
```

Resource are made up of

1. ***Type:*** system component being managed

2. ***Title:*** instance of that resource type

3. ***Attributes:*** characteristic of system component

Become root now

 # sudo su

- Most of the tasks that puppet does requires a privileged user.

Talking to your system

Lets start talking to our system using puppet's abstraction layer shell

$puppet
resource user
root

```
user { 'root':
        home => '/root',
        shell => '/bin/bash',
        uid => '0',
        ensure => 'present',
        password => '*',
        gid => '0',
        comment => 'SysAdmin'
}
```

Resource Shell

```
$ puppet resource <TYPE> [<NAME>]
[ATTRIBUTE=VALUE ...]
```

$ puppet resource user root

```
user { 'root':
    home => '/root',
    shell => '/bin/bash',
    uid => '0',
    ensure => 'present',
    password => '*',
    gid => '0',
    comment => 'SysAdmin'
}
```

Exploring Resource Shell

```
$ puppet resource package | less

$ puppet help resource
```

Creating new Resources

Create a new user

Create:

```
$ puppet resource user rajnikant \
        ensure=present  \
        home="/home/rajnikant" managehome=true
```

Verify:

```
$ puppet resource user rajnikant
```

Syntax note: you dont need to use those back slashes. Type the complete command on the same line

Lets now start making changes to the existing resource.

Lets Edit the Resource

```
$ puppet resource user rajnikant > rajnikant.pp
```

- Edit rajnikant.pp

```
1. root@demo:/home/vagrant (bash)
File Edit Options Buffers Tools Help
user { 'rajnikant':
  ensure           => 'present',
  gid              => '503',
  home             => '/home/rajnikant',
  comment          => 'Rajni, The Boss',
  password         => '!!',
  password_max_age => '99999',
  password_min_age => '0',
  shell            => '/bin/bash',
  uid              => '502',
}

-UU-:**--F1  rajnikant.pp   All L5     (Puppet)------------
Auto-saving...done
```

Example Changes

- Change the shell from /bin/bash to /bin/sh
- Add comment parameter

Syntax Check

```
$ puppet parser validate rajnikant.pp
```

Commit Changes

```
$ puppet apply rajnikant.pp
```

Puppet Apply

- Standalone puppet execution
- Local
- Serverless Puppet Site

Standalone

- Uses Puppet Apply
- Triggered by user as a command
- Fetches manifests, compiles into catalog.

Master/Agent

- Uses Puppet Agent
- Typically runs as a daemon/service
- Gets only catalogue and not manifests

Validate

```
$ puppet resource user rajnikant
```

Resource Properties

- Declarative
- Idempotent

Idempotent

A resource

can be applied to a system

multiple times and end result

will remain the same.

```
package { “nginx”:
   ensure => latest,
}
```

Declarative

Resource describe what

should be managed by

Puppet without bothering

about how

```
package { “nginx”:
    ensure => latest,
}
```

In above piece of code, all we tell to

Puppet is to install nginx on machine

without bothering about package

manager which will be used for this

very purpose.

Resource Abstraction Layer (RAL)

Type/ Resource **Provider**

What → How

References

Puppet Core Types Cheatsheet

Puppet Core Type Reference Doc (http://docs.puppetlabs.com/references/latest/type.html)

Discovering Resources

Puppet provides with **describe** subcommand to find informaiton about resources, parameters etc.

Describing Resources

```
$ puppet describe –list

$ puppet describe file

$ puppet describe file –short

$ puppet describe file --providers
```

Manifests

Lets write our first manifest....

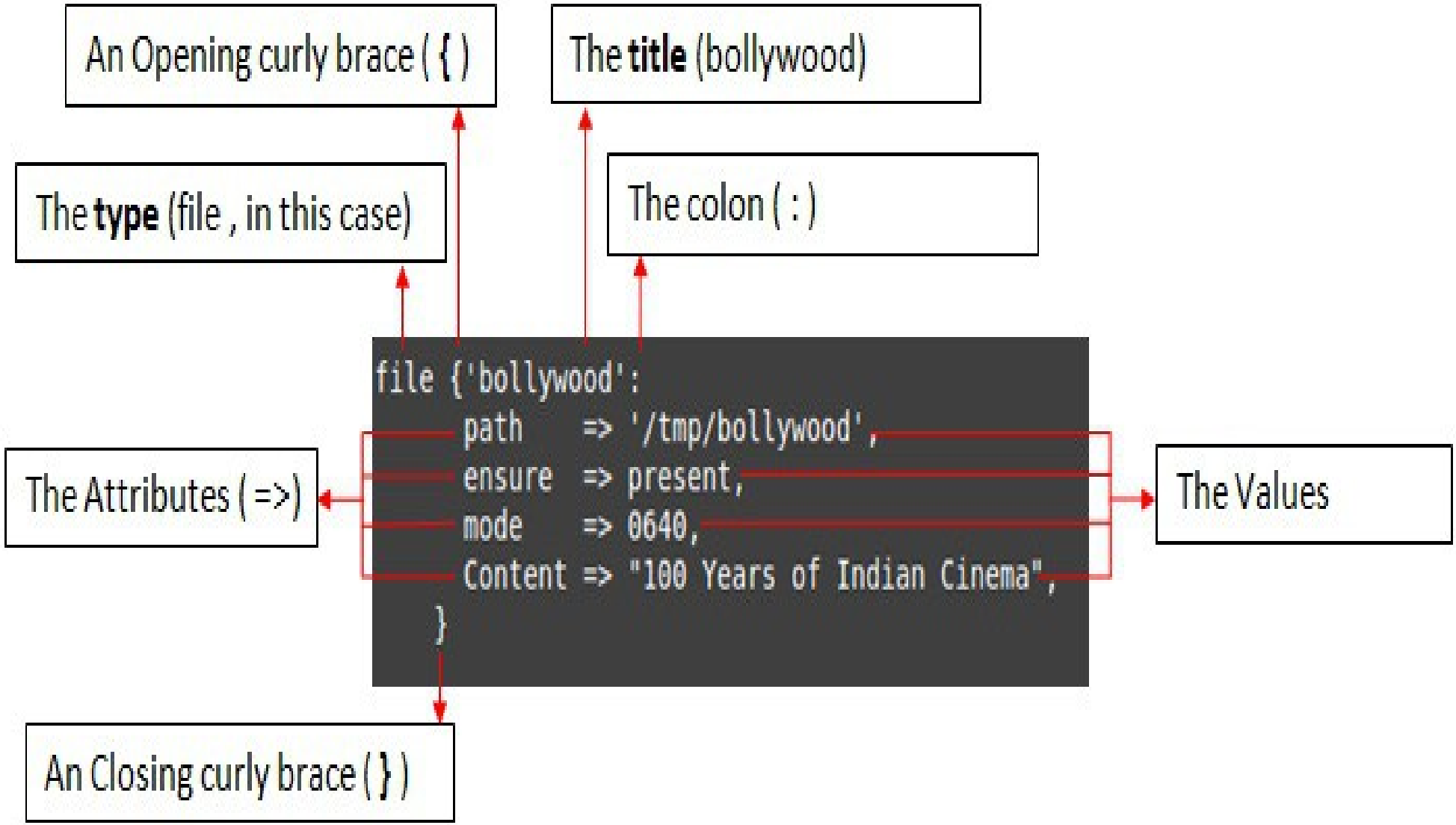
$ mkdir /vagrant/puppet

Edit /vagrant/puppet/test**.pp**

```
file {'bollywood':
    path    => '/tmp/bollywood',
    ensure  => present,
    mode    => 0640,
    content => "100 Years of Indian Cinema",
  }
```

Since the directory that contains Vagrantfile gets automatically mounted on the VM (at /vagrant) even if you edit the files from the host machine (your desktop/laptop) it will get reflected on the VM.

Dissecting Resource



test.pp

1. root@demo:/home/vagrant/puppet (bash)

```
File Edit Options Buffers Tools Help
file {'bollywood':
        path    => '/tmp/bollywood',
        ensure  => present,
        mode    => 0640,
        content => "100 Years of Indian Cinema",
      }

-UU-:----F1  site.pp        All L1     (Puppet)-------------
For information about GNU Emacs and the GNU system, type C-\
h C-a.
```

Time to apply the code

Before we apply, lets first check syntax and do a dry run

Preparing to Apply

Syntax Check

```
$ puppet parser validate /vagrant/puppet/test.pp
```

Dry Run

```
$ puppet apply --noop /vagrant/puppet/test.pp
```

Dry run

lets apply....

```
$ puppet apply /vagrant/puppet/test.pp
```

Validation

```
[root@demo puppet]# cat /tmp/bollywood

100 Years of Indian Cinema
```

Lets try changing the contents of the file manually and see how Puppet reacts...

```
$ echo 'I love Hollywood !!'  >
/tmp/bollywood

$ puppet apply test.pp
```

Configuration Drift

Good Practice

It's a good idea to add a notice to the file you manage with puppet, in the comments section, so that manual updates are avoided.

ensure == state

Exercise

Now that we have learnt the resource DSL, lets write a few more resources ourselves. Following slides describe one resource at a time. You are supposed to write a resource for each.

Remove amitabh user

- Type of resource : amitabh
- State : absent

Remove rajnikant user

- Type of resource : user
- State : absent

Create a file

- Type of resource : file
- Path : /tmp/tollywood
- Permissions: 644
- Owner: root
- Group: root
- Content : “What is tollywood?”

What we just wrote in test.pp was a...

Manifest

Now, Lets try to make a definition ….

'** Manifest is a **collection of resources, classes and definitions '

We've already seen resources, We'll look at definitions and classes a little later

Quote

'Life is a journey, not a destination.'

- Ralph Waldo Emerson

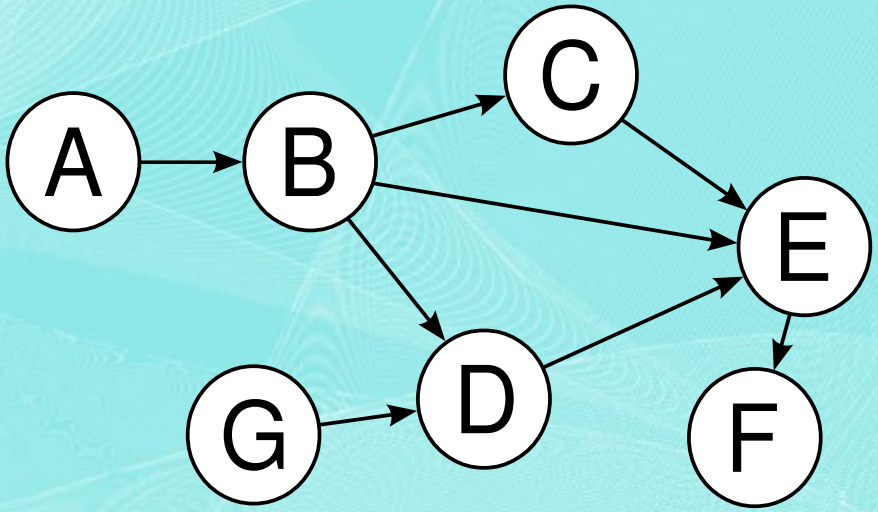
Quote

'Its only the destination that I care about...!

- A Puppet User

Manifests define the “desired state”. Thats the core of the thinking like a Puppet user.

Catalog



Before being applied, manifests get compiled
Into a “**catalog**”, which is a directed acyclic
Graph representing resources, and order in
Which they are to be synced.

Multi Machines

Node Declarations

So far we've only dealt with one server, the demo server. But of course Puppet can manage many machines, each with different configurations, so we need a way to tell Puppet which configuration belongs to each machine.

Node Declaration

```
node NODENAME {
        R 1
        R2
        ...
        Rn
}
```

Create Manifests Subdir

Within this directory, create a subdirectory named manifests:

```
root@demo:~$ cd puppet
root@demo:~/puppet$ mkdir manifests
```

test.pp => nodes.pp

Move your existing test.pp file into the manifests subdirectory:

```
root@demo:~/puppet$ mv ../test.pp manifests/nodes.pp
```

Edit nodes.pp

- Enclose resources in node definition

```
node 'demo' {
        file { ....
              .....
         }
  }
```

nodes.pp

1. root@demo:/home/vagrant/puppet/manifests (bash)

```
File Edit Options Buffers Tools Help
node 'demo' {
file {'bollywood':
        path    => '/tmp/bollywood',
        ensure  => present,
        mode    => 0640,
        content => "100 Years of Indian Cinema",
      }
file {'/tmp/hollywood':
        ensure  => file,
        content => "Hi.",
     }
file {'/tmp/tollywood':
        ensure => directory,
        mode   => 0644,
     }
file {'/tmp/mumbai':
        ensure => link,
        target => '/tmp/bollywood',
     }
file {'/tmp/chennai':
        ensure => '/tmp/tollywood',
     }
user {'rajnikant':
        ensure => absent,
     }
user {'amitabh':
       ensure => absent,
     }
     notify {"Removed Amitabh and Rajnikant.":}
     notify {"Added New File Types!":}
}
-UU-:**--F1  nodes.pp       All L30    (Puppet)----------------
```

Your directory structure should look like...

```
puppet
    |
    |__manifests
    |          |
    |          |__ nodes.pp
    |
```

Site Configurations

Your directory structure should look like...

```
puppet
   |
   |__manifests
   |          |
   |          |__ site.pp
   |          |__ nodes.pp
   |
```

Site.pp

Create site.pp with the following contents

```
import 'nodes.pp'
```

Check

```
$ puppet apply manifests/site.pp
```

Deprecation Warnings

- Puppet 3.6.1 onwards (May 22nd 2014 onwards)
- Site.pp will be deprecated

Fixing

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

Summary

- Resources Overview
- Resource Shell
- Manifests
- Puppet Apply
- Node Declarations