Configuration Management with Puppet

Introduction

What is Puppet

- is a configuration management system
- is a programming language
- offers a Client/Server architecture
- has a huge community
- widely used in the IT industry
- commercial support available if needed

What else is needed

- central software repositories
  - yum
  - apt
- Provisioning system
  - kickstart
  - preseed
- Version control
  - GIT
  - subversion

Puppet process - Step 1

Registration

Example: foreman, satellite, spacewalk ...

Creates: Kickstart File and Puppet Node definition

Puppet process - Step 2

Provisioning

Tool: Redhat anaconda, Fai, cobbler, preseed

Input:

anaconda configured by kickstart file

preseed config file

…

Result: Installing minimal linux and puppet, and start Puppet after reboot

Puppet process - Step 3

Configuration

Tool: puppet

Input:

node definition

Result: puppet defined system state

Contents

- Resources
- Manifests
- Ordering
- Variables, Conditionals, and Facts
- Classes
- Module

Resources

All elements of a node will be described as resources

- Files
- User
- Services
- ... (about 50)

Resources

- Abstraction layer to access the resources (**RAL)**
- Resources have attributes
  - a File has a Path.
- The RAL gives you os independence
  - BUT!! not all resources are available on every platform
- Resources well documented

Example Resource


```
user { 'dave':
       ensure     => present,
       uid        => '507',
       gid        => 'admin',
       shell      => '/bin/zsh',
       home       => '/home/dave',
       managehome => true,
     }
```

Ressource shell

You can interact with the RAL directly.

- `puppet resource user root`
- ```
  puppet resource user dave \
  ensure=present shell="/bin/zsh" \
  home="/home/dave" managehome=true
  ```
  ```

Resource Documentation

```
$ puppet describe -s user

user
====
Manage users.  This type is mostly built to manage system
users, so it is lacking some features useful for managing normal
users.

This resource type uses the prescribed native tools for creating
groups and generally uses POSIX APIs for retrieving information
about them.  It does not directly modify `/etc/passwd` or
anything.


Parameters
----------
    allowdupe, auth_membership, auths, comment, ensure, expiry, gid, groups,
    home, key_membership, keys, managehome, membership, name, password,
    password_max_age, password_min_age, profile_membership, profiles,
    project, role_membership, roles, shell, uid

Providers
---------
    directoryservice, hpuxuseradd, ldap, pw, user_role_add, useradd
```

Resource basic

- file vs. augeas
- yumrepo stages
- package ensure latest ?
- exec only if needed

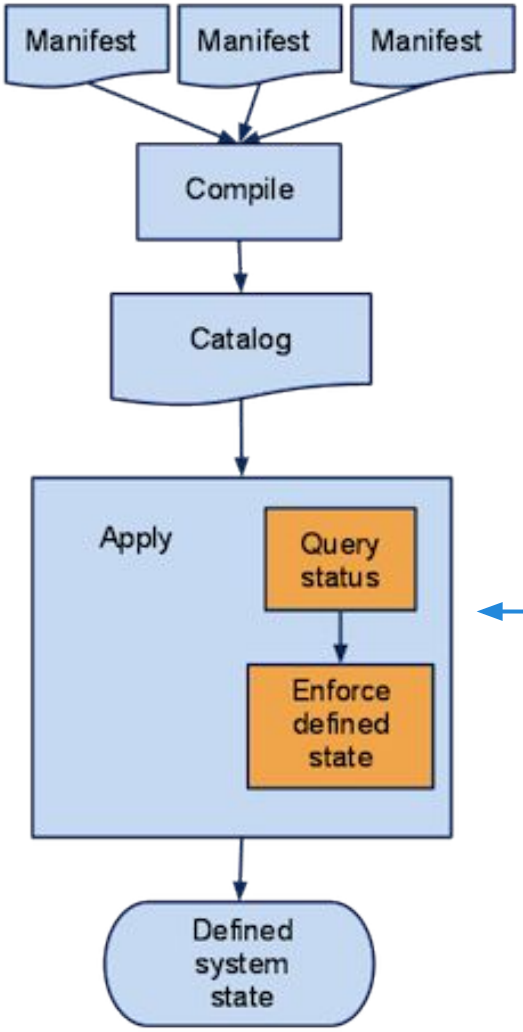
Manifests

- *manifests* are puppet programs
- puppet programs
  - declare resources
  - define conditions
  - group resources
  - generate text
  - link other manifests
  - define ordering

Manifests Compile

Puppet-Dev

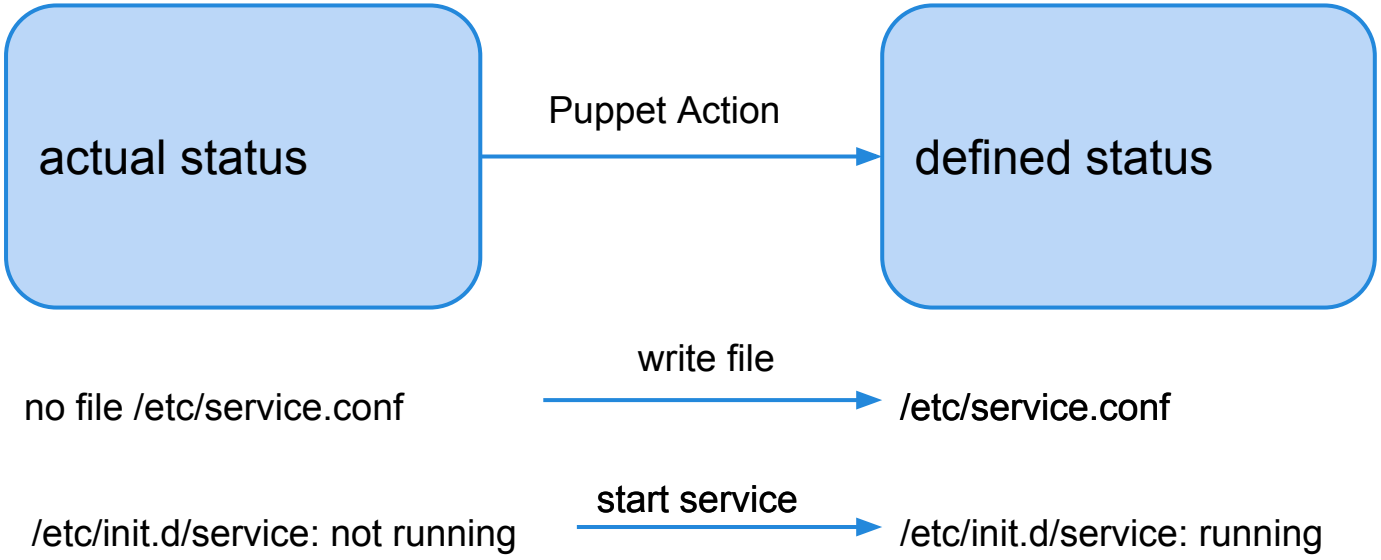
writes



transform

Managed-Node

Manifests Run



Manifests example


```
file {'/tmp/test1':
      ensure  => present,
      content => "Hi.",
    }

file {'/tmp/test2':
      ensure => directory,
      mode   => 0644,
}

file {'/tmp/test3':
      ensure => link,
      target => '/tmp/test1',
}

notify {"I'm notifying you.":}
notify {"So am I!":}
```

Ordering example


```
file { '/etc/ssh/sshd_config':
  ensure => file,
  mode   => 600,
  source => '/root/learning-manifests/sshd_config',
}

service { 'sshd':
  ensure     => running,
  enable     => true,
  subscribe  => File['/etc/ssh/sshd_config'],
}
```

Facts example


```
host {'self':
    ensure       => present,
    name         => $fqdn,
    host_aliases => ['puppet', $hostname],
    ip           => $ipaddress,
}

file {'motd':
    ensure  => file,
    path    => '/etc/motd',
    mode    => 0644,
    content => "Welcome to ${hostname},\na ${operatingsystem}
island in the sea of ${domain}.\n",
}
```

Conditionals example


```
if $is_virtual == 'true' {
    service {'ntpd':
      ensure => stopped,
      enable => false,
    }
  }
  else {
    service { 'ntpd':
      name       => 'ntpd',
      ensure     => running,
      enable     => true,
      hasrestart => true,
      require => Package['ntp'],
    }
  }
```

Code entities

- ## Classes

  Classes are named blocks of Puppet code that are stored in modules for later use and are not applied until they are invoked by name.

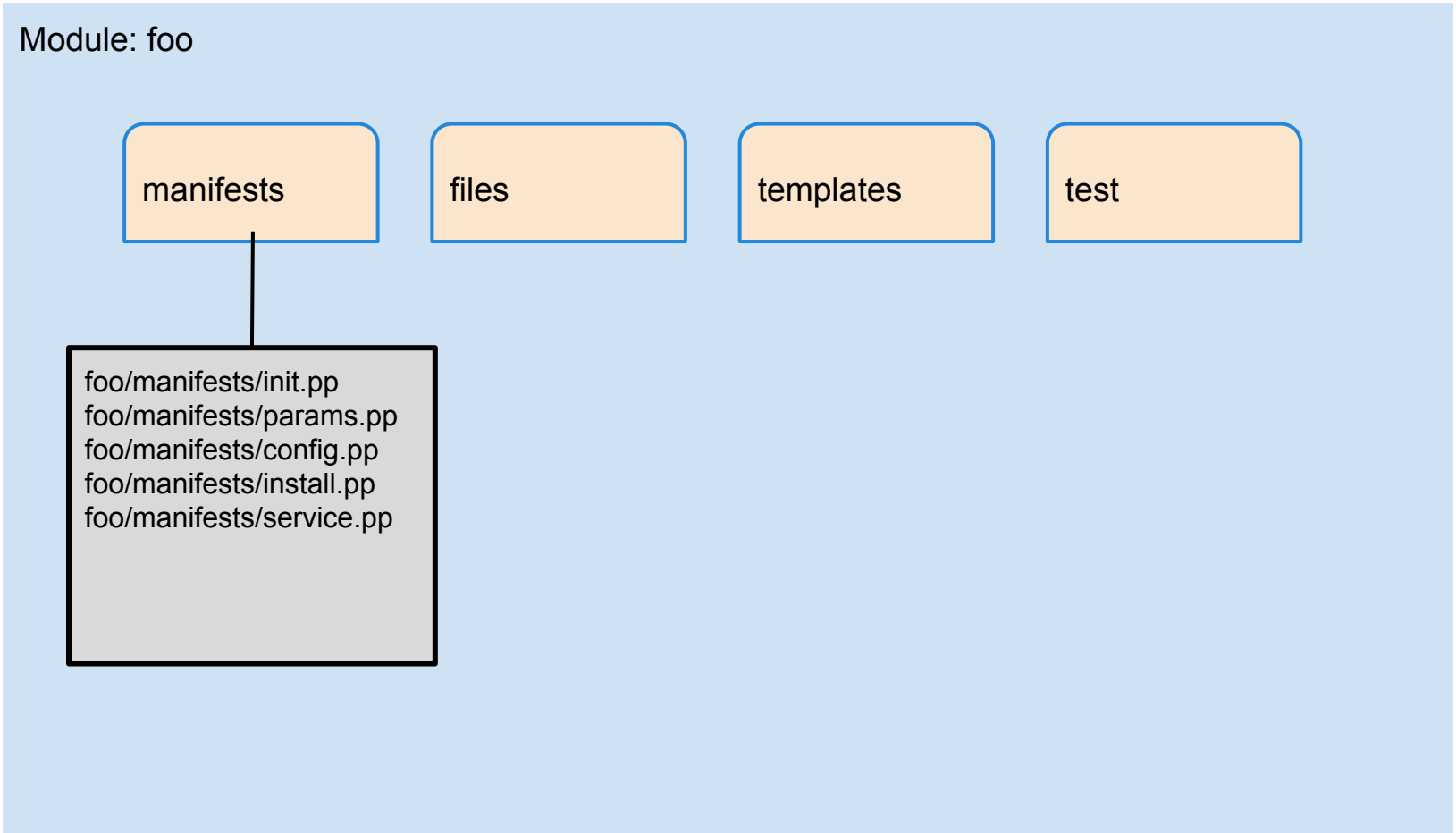
- ## Modules

  Modules are self-contained bundles of code and data. You can download pre-built modules from the Puppet Forge or you can write your own modules. They can be added to a node defnition.

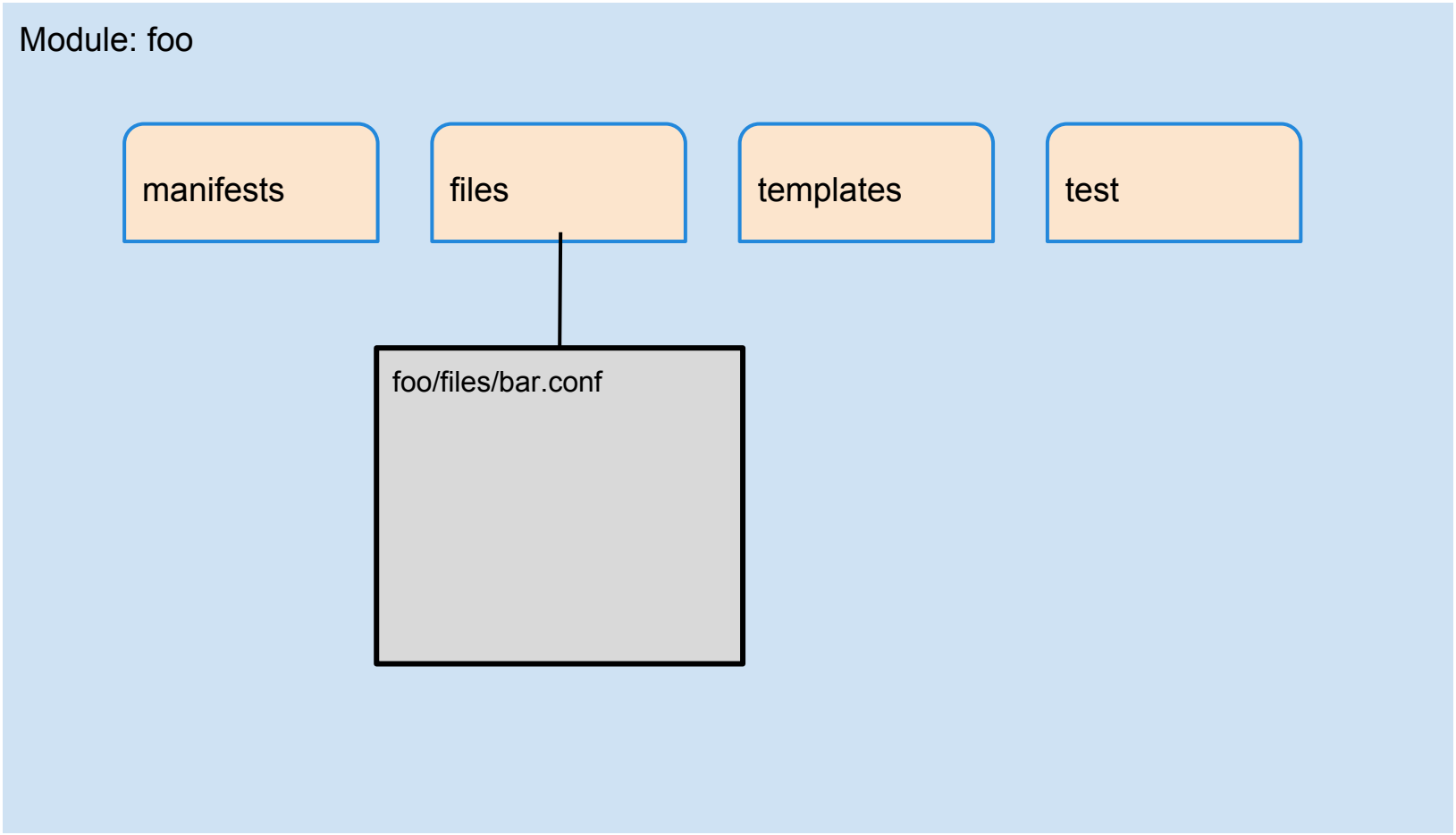
- ## Node definition / ENC

  A node definition or node statement is a block of Puppet code that will only be included in matching nodes' catalogs.
  An external node classifier is an arbitrary script or application which can tell Puppet which modules a node should have.
  This allows you to assign specific configurations to specific nodes.

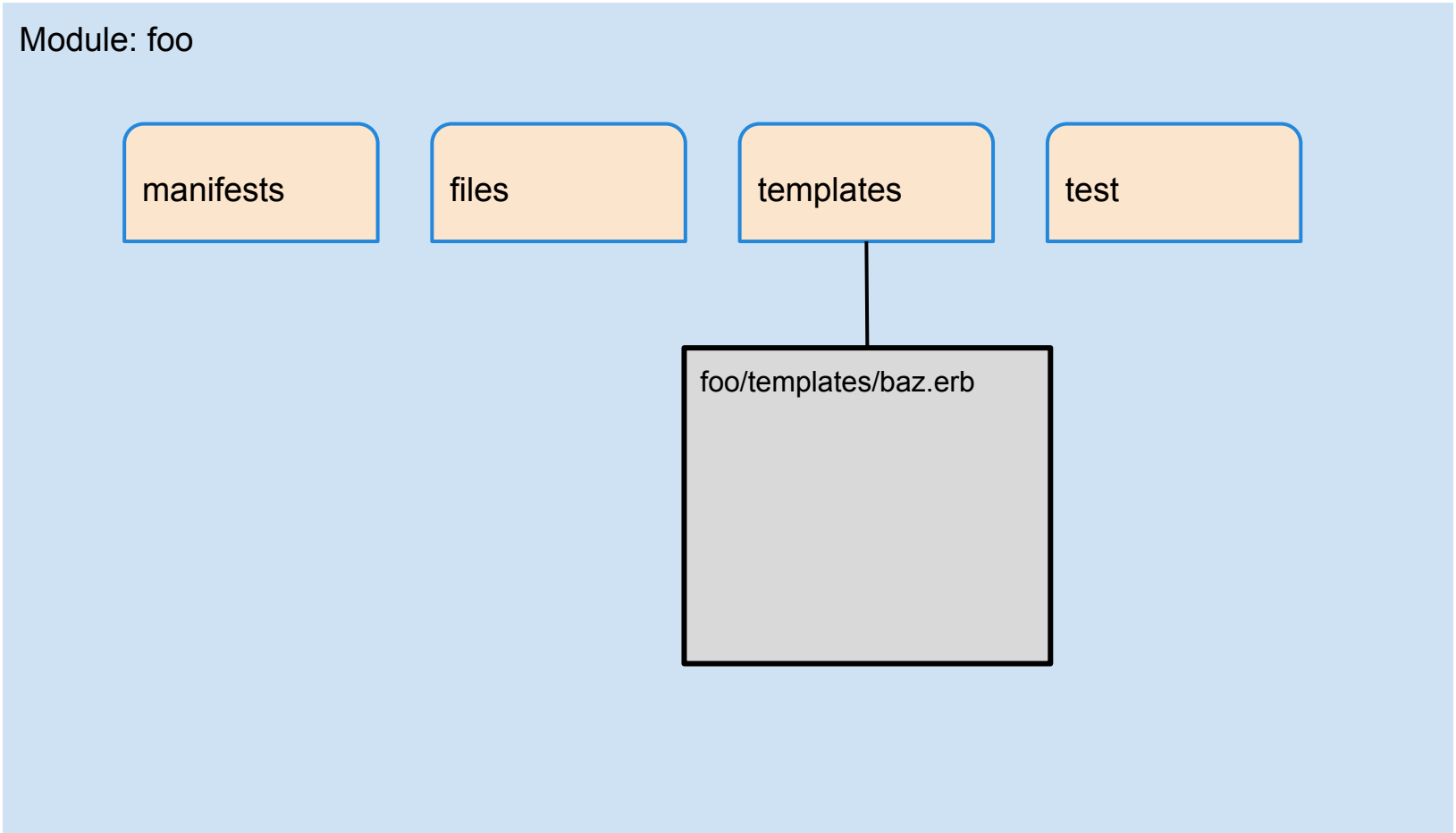
Structure of a Module



Structure of a Module



Structure of a Module



Class example

/ntp/manifests/init.pp

```
class ntp inherits ntp::params{
  include ntp::install
  include ntp::service
}
```

Class example

/ntp/manifests/params.pp

```
class ntp::params{
   $service_name ='ntpd'
   $conf_file    ='ntp.conf.el'
}
```

Class example

/ntp/manifests/install.pp

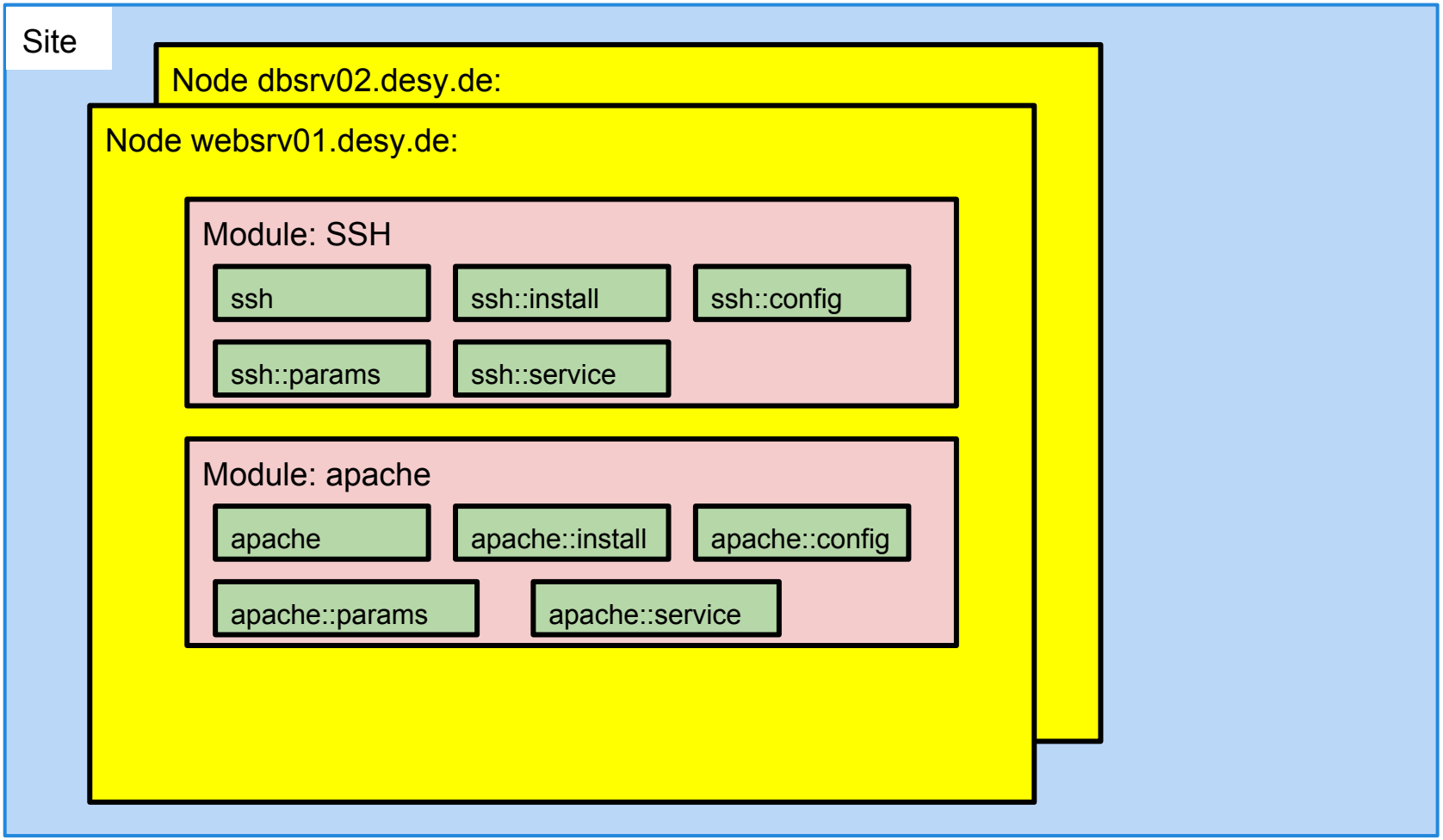
```
class ntp::install {
 package { 'ntp':
  ensure => installed,
 }
}
```

Class example

/ntp/manifests/service.pp

```
class ntp::service {
 service { 'ntp':
  name     => $service_name,
  ensure   => running,
  enable   => true,
  subscribe=> File['ntp.conf'],
 }
}
```

Structure of Configuration



Organization of the modules

```
node ‘foo21.desy.de’ {
  include it_desktop
}
```

modules / features	ntp	afs	apache
it_desktop	x	x	
xfel_webserver	x		x

It's the end my friend!