Puppet ile Linux Sistem Yönetimi Otomasyonu

Çağrı Ersen
cagri.ersen@bga.com.tr
http://linuxakademi.com.tr
http://syslogs.org
https://twitter.com/CagriErsen

Sistem Yönetimi Temelleri

Nedir, Ne Yapar ?

Ağ servisleri sunan sistemlerin yönetilmesi.

- Kurma/Kaldırma/Güncelleme/Yapılandırma
- Disk / Storage / Dosya Sistemi Yönetimi
- Kullanıcı / Grup Yönetimi
- Yedekleme İşleri
- Performans Analizi / Micro Management
- Güvenlik Yönetimi

Anahtar Kelime: Yapılandırma

Sonuç olarak; sistem yönetimi sürekli olarak “bir şeylerin” yapılandırılmasından ibarettir.

Ömrünüz genellikle yapılandırma komutları ve dosyaları içerisinde geçer.

Yapılandırma Seçenekleri

Genel olarak üç yol tercih edilebilir,

1. Manual Yapılandırma
2. Kurulum Sırasında Otomatik Yapılandırma (Post-Installation Script)
3. Bir Otomasyon Aracı Kullanarak Yapılandırma

Manual Yapılandırma

Manual Yapılandırma (1)

Az sayıda sunucu yönetiyorsanız sorun yok!

Herşeyi manual yapsanız da olur (mu acaba ?)

- Kolay
- Az sunucuda az zamanda az iş

Manual Yapılandırma (2)

Fakat…

- Kesin olarak unuttuğunuz bir şeyler olur.
- Değişiklikleri takip edemezsiniz.
- Aynı ayarlarla kolayca yeni bir sunucu kuramazsınız.

Manual Yapılandırma (3)

Çok sayıda sunucu yönetiyorsanız “en kötü” ve “en uygulanamaz” seçenektir!

Kurulum Sırasında Otomatik Yapılandırma (Post-Installation Scripts)

Post Installation Scripts (1)

Standart kurulumun ardından, yazdığımız küçük yapılandırma scriptlerini çalıştırmak.

- Çok sayıda sunucu hızlıca yapılandırılabilir.
- Kısmen merkezi bir yönetim sağlanmış olur.

Post Installation Scripts (2)

Fakat:

- Yapılandırmanın sürekli aynı kalacağını garanti edemezsiniz.
- Değişiklik yapmak kolay değildir.
- Değişiklik geçmişini takip edemezsiniz.

Post Installation Scripts (3)

ve kalabalık bir ekipseniz...

Bir Otomasyon Aracı Kullanarak Yapılandırma

Otomasyon Araçları Kullanmak (1)

Seçenekler arasında açık ara en iyi alternatif.

- Sunucularınızı nasıl yapılandırmak istediğinizi tarif edersiniz,
- Otomasyon aracı yapılandırmayı tarifinize göre uygular.
- Ayarların hep aynı durumda kalmasını sağlar

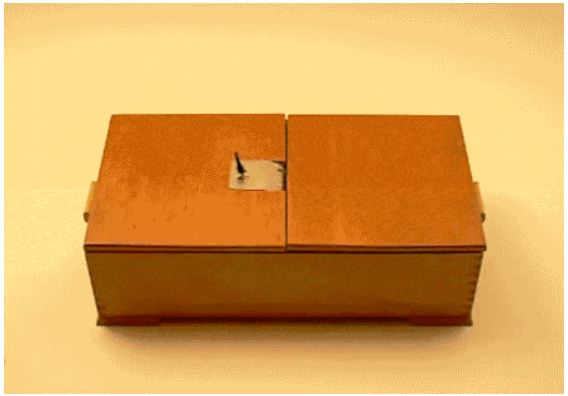
Otomasyon Araçları Kullanmak (2)

- Altyapıya yeni bir sunucu eklemek kolaydır.
- Çöken bir sunucu eski durumuna süratle döndürülebilir.
- Yapılandırma değişiklikleri zahmetsizdir.
- Değişiklik geçmişi tutulabilir.

Otomasyon Araçları Kullanmak (3)

HIZ

TUTARLILIK



PUPPET

Puppet Nedir ?

- Gelişmiş bir Configuration Management Tool
- 2005 - Luke Kanie tarafından
- Ruby
- Son Sürüm 3.4.3
- Açık Kaynak Kod
- Declarative
- Kendine ait DSL (Domain Spesifik Language)
- Server / Client Mimari

Desteklediği Platformlar

RHEL	CentOS	Fedora	Debian
Ubuntu	SUSE	Gentoo	Mandriva
FreeBSD	OpenBSD	NetBSD	Solaris
Arch	HP-UX	Mac OS X	Windows

Yönetebildiği bazı “şeyler”

Paketler	**Servisler**	**Kullanıcılar Gruplar**	**Komutlar**
SSH Anahtarları	**Cron**	**Dosyalar Dizinler**	**Mount**

ve daha fazlası...

Çalışma Mod’ları

- Server / Client (Master / Agent)
- Standalone (Serverless)

Client / Server Mode (1)

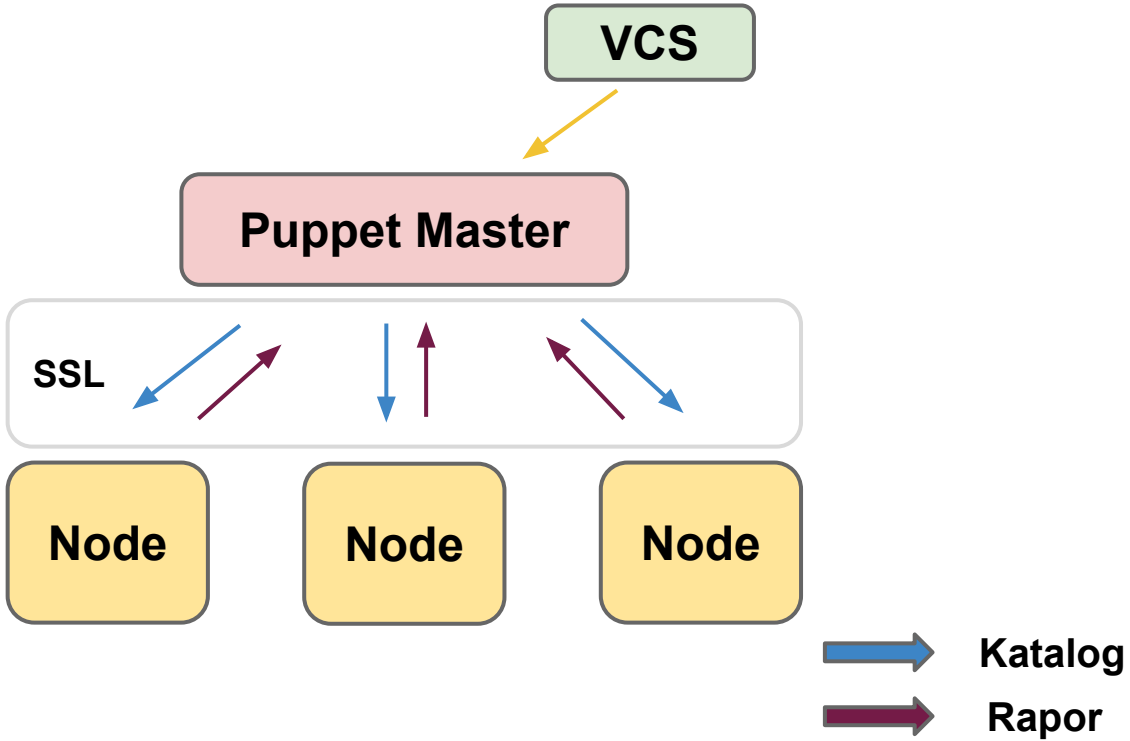
Puppet Server (Master)

- Tüm config dosyaları master'da tutulur.
- Üzerinde bir CA vardır, SSL keyleri tutar.

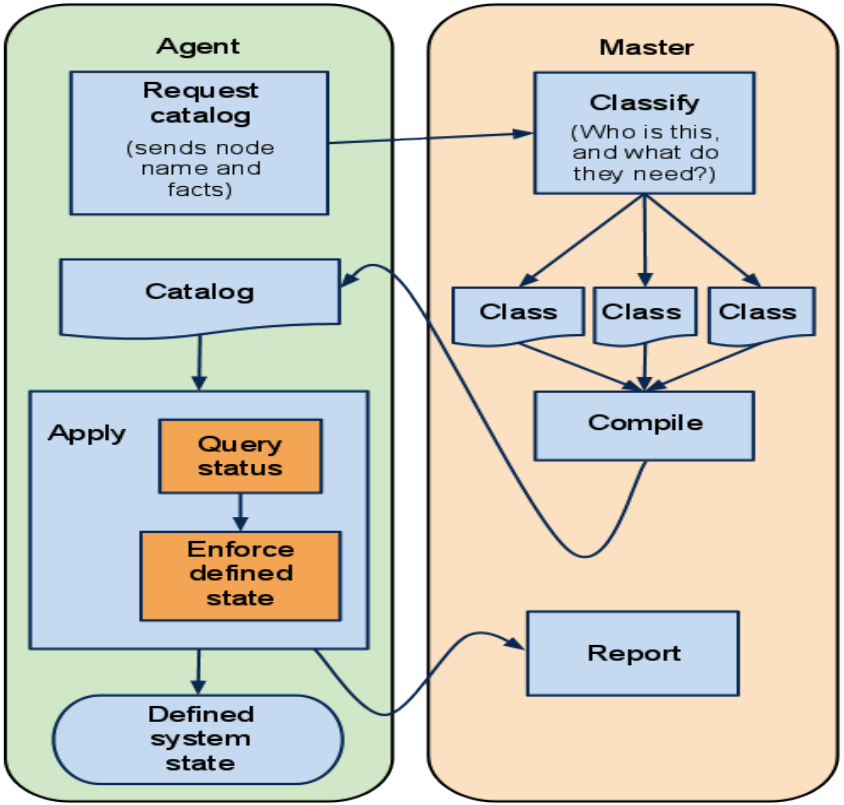
Bir ya da daha fazla client (Node)

- Node 30 dakikada bir sistem bilgilerini Master'a yollar.
- Master, node'a "istenilen yapılandırma durumunu" belirten bir katalog gönderir.
- Node kataloğu uygular, master'a raporlar.

Client / Server Mode (2)



Client / Server Mode (3)

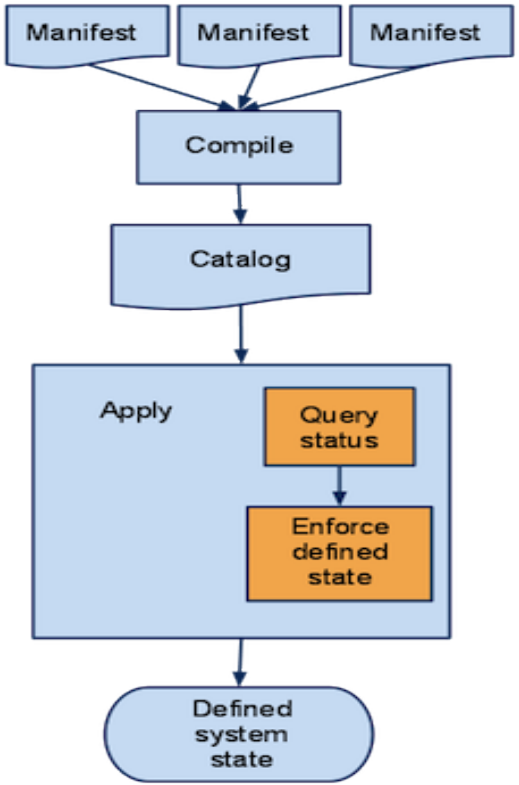


Puppet: Standalone Mode (1)

Üzerinde Puppet Paketi Kurulu Bir Host

- Master / Agent modundan tek fark yapılandırma dosyalarının localde tutuluyor olmasıdır.
- Çok sunucudan oluşan altyapılar için ideal değildir.

Standalone Mode (2)



Puppet Mimarisi ve Bileşenler

Declarative Language

Yapılandırma deklare edilir (Desired state)

```
user { 'crom':
     ensure     => present,
     uid        => '512',
     gid        => 'admin',
     shell      => '/bin/bash',
     home       => '/home/crom',
     managehome => true,
   }
```

OS Bağımsız

Sistem yöneticisi ile sistem arasındaki katmandır!

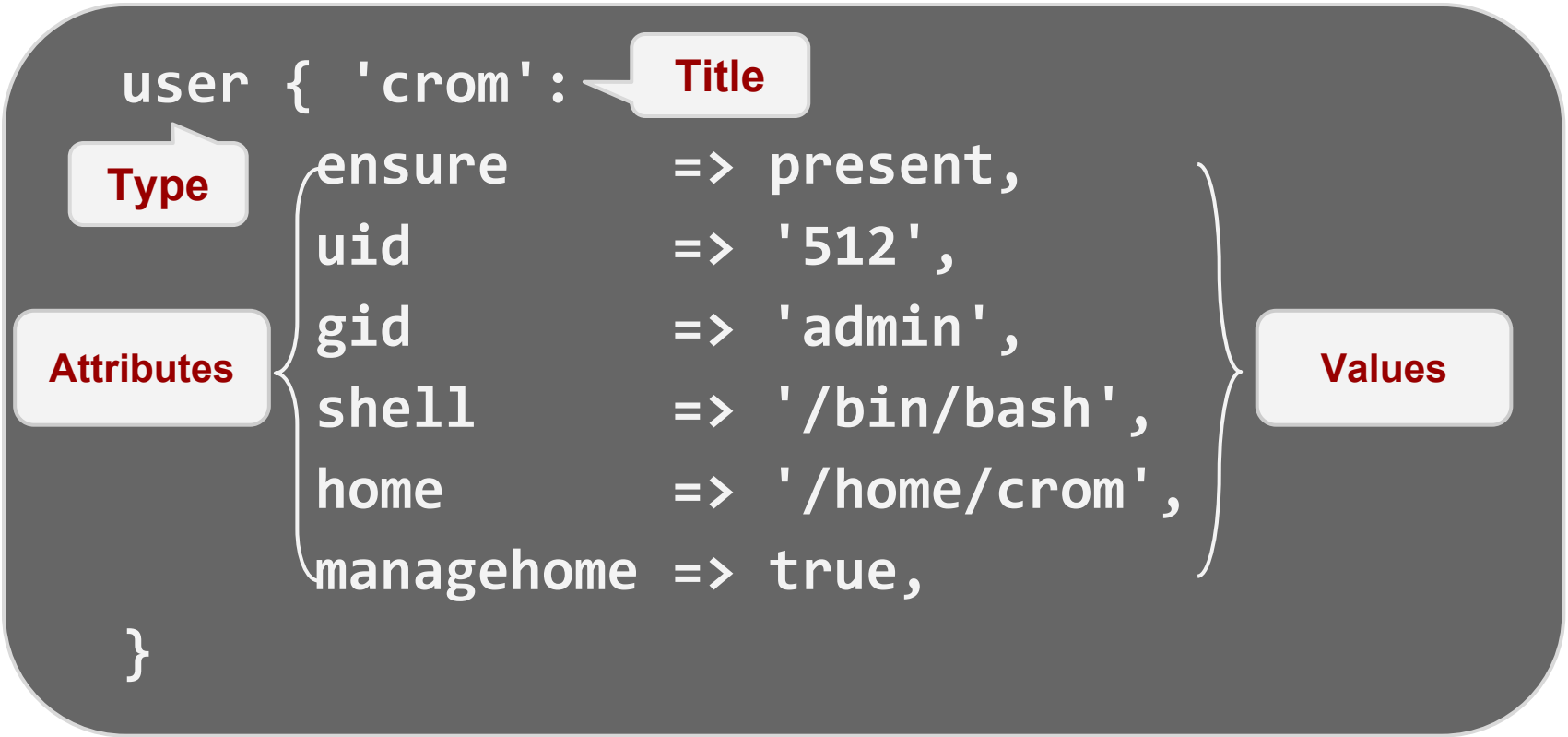
```
package { 'openssh-server':
  ensure => present,
}
```

Puppet: Resources (1)

Resource'lar sistem yapılandırmasını meydana getiren her bir bileşene verilen isimdir.

- Bir kullanıcı
- Bir paket
- Bir dosya
- Bir servis
- ...

Puppet: Resources (2)



Puppet: Resource Types

- **packages**: Paket kurma kaldırma, update
- **file**: Dosya/Dizin işlemleri içerik, izinleri, sahibi
- **service**: Servis başlatma durdurma, enable/disable,
- **exec**: Komut çalıştırma işlemleri
- **cron**: Zamanlanmış görevlerin yönetilmesi, ekle/kaldır/düzelt
- **user**: Kullanıcı işlemleri ekle/kaldır
- **group**: Grup ekle/çıkar, kullanıcı ekle
- **mount**: disk bağlama işlemleri
- **sshkeys**: SSH Key'lerinin yönetimi

Tam Liste: augeas computer cron exec file filebucket group host interface k5login macauthorization mailalias maillist mcx mount nagios* notify package resources router schedule scheduled_task selboolean selmodule service ssh_authorized_key sshkey stage tidy uservlan yumrepo zfs zone zpool

Puppet: Resources (Örnek - 1)

Resource Type: Service

```
service { 'NetworkManager':
  ensure => 'stopped',
  enable => 'false',
}
```

Puppet: Resources (Örnek - 2)

Resource Type: File

```
file {'testfile':
  path=> '/tmp/testfile',
  ensure  => present,
  mode=> 0640,
  content => "I'm a test file.",
}
```

Puppet: Resources (Örnek - 3)

Resource Type: Cron

```
cron { logrotate:
  command => "/usr/sbin/logrotate",
  user    => root,
  hour    => 2,
  minute  => 0
}
```

Puppet: Manifests (1)

- .pp uzantılı Puppet scriptleri.
- Yapılandırma bu dosyalar içerisinde tanımlanır.
- Default → /etc/puppet/manifests/site.pp
- Tüm tanımlamalar site.pp içinde yapılabilir,
- Ya da birden çok .pp dosyası içersinde tanımlanıp, site.pp içerisinden “include” edilebilir.

Puppet: Manifests (2)

 # vi /etc/puppet/manifests/kullanici-sil.pp

```
user {'crom':
      ensure => absent,
   }
```

 # puppet apply /etc/puppet/manifests/kullanici-sil.pp

Puppet: Manifests - Ordering

```
package { 'openssh-server':
   ensure => present,
   before => File['/etc/ssh/sshd_config'],
 }
 file { '/etc/ssh/sshd_config':
   ensure => file,
   mode   => 600,
   source => 'puppet:///modules/sshd/sshd_config',
 }
 service { 'sshd':
   ensure     => running,
   enable     => true,
   subscribe  => File['/etc/ssh/sshd_config'],
 }
```

sshd_config değişikliklerini işlemeden önce ssh paketinin kurulu olduğundan emin ol.

Bu dosyada değişiklik yaparsan ssh servisini restart et.

Puppet: Variables

Puppet scriptleri içerisinde değişkenler tanımlayabilir; built-in değişkenleri (facts) kullanabilirsiniz.

```
$longthing = "BURADA-SSH-KEY-VAR"
   file {'authorized_keys':
   path    => '/root/.ssh/authorized_keys',
   content => $longthing,
   }
```

Puppet: Facts

- Önceden tanımlanmış built-in değişkenler:

```
file {'motd':
    ensure  => file,
    path    => '/etc/motd',
     mode    => 0644,
     content => "IP: ${ipaddress}, FQDN: ${fqdn}
OS: ${operatingsystem} Version:
${operatingsystemrelease} Puppet Version:
${puppetversion}.
     ",
     }
```

Puppet: Facter

- Fact'leri toplamak için kullanılan Puppet Tool'u

```
# facter
architecture => x86_64
bios_vendor => Phoenix Technologies LTD
bios_version => 6.00
blockdevice_sda_model => VMware Virtual S
blockdevice_sda_size => 5368709120
blockdevice_sda_vendor => VMware,
boardmanufacturer => Intel Corporation
domain => crom.lab
facterversion => 1.7.5
filesystems => ext4,iso9660
fqdn => puppetmaster01.crom.lab
hardwareisa => x86_64
.........
```

Puppet: Conditionals Statements

IF kullanımı:

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

Puppet: Conditionals Statements

Case kullanımı:

```
    case $operatingsystem {
      centos, redhat: { $apache = "httpd" }
      debian, ubuntu: { $apache = "apache2" }
      default: { fail("Unrecognized operating system for
webserver") }
    }

    package {'apache':
      name   => $apache,
      ensure => latest,
    }
```

Puppet: Classes

- Puppet yapılandırmalarını parçalara bölmek için kullanılan kod blokları.
- Amaç tanımlamaların okunaklı ve anlaşılır olmasını sağlamaktır.
- Tanımlanan class'lar manifest dosyaları içerisinden isimleri ile çağırılabilirler (include).

Puppet: Classes (2)

 # vi /etc/puppet/manifests/site.pp

```
class add_my_user {
 user { 'crom':
   ensure     => present,
   uid        => '507',
   gid        => 'wheel',
   shell      => '/bin/bash',
   home       => '/home/crom',
   managehome => true,
 }
}
```


```
include add_my_user
```

Puppet: Modules

- Puppet scriptlerini ve dosyalarını hiyerarşik bir düzende tutmaya yarayan “dizin”’lerdir.
- Örneğin SSHd ve NTP servislerini yöneten manifest’ler birbirlerinden ayrı modüller olarak tutulabilir.
- Default module dizini: /etc/puppet/modules/

Puppet: Modules - Dizin Yapısı (1)

Module dizinlerinde üç önemli altdizin bulunur.

- /etc/puppet/modules/common/manifests
Bu dizinde .pp uzantılı manifest dosyaları ve bunların include edildiği init.pp isimli ana manifest dosyası bulunur.

```
-rw-r--r-- 1 root root 200 Mar 25 17:12 add-users.pp
-rw-r--r-- 1 root root  69 Mar 25 20:25 init.pp
-rw-r--r-- 1 root root 162 Mar 25 20:24 motd.pp
-rw-r--r-- 1 root root 239 Mar 25 20:18 sudoers.pp
```

Puppet: Modules - Dizin Yapısı (2)

- init.pp dosyasında modül ile aynı isimli bir class bulunur,
- Dizindeki diğer .pp dosyalarında tanımlı class'lar, bu default class içerisinden include edilir.

 # vi /etc/puppet/modules/common/manifests/init.pp

```
class common {
 include add-users
 include sudoers
 include motd
}
```

Puppet: Modules - Dizin Yapısı (3)

- /etc/puppet/modules/common/files

- Bu dizinde tarif edilen yapılandırma için gerekli olan static conf dosyaları bulunur. (örn: sudoers)
- Master / Agent mode'da, dosyalar manifest'ler / class'lar içerisinden **puppet:///modules/isim/conf_dosyasi** şeklinde çağırılır:

```
file { '/etc/sudoers':
  ensure => file,
  mode   => 440,
  source  => "puppet:///modules/common/sudoers",
}
```

Puppet: Modules - Dizin Yapısı (4)

```
class ntp {
  case $operatingsystem {
    centos, redhat: {
      $conf_file    = 'ntp.conf.el'
    }
    debian, ubuntu: {
      $conf_file    = 'ntp.conf.debian'
    }
  }

  file { 'ntp.conf':
    path    => '/etc/ntp.conf',
    ensure  => file,
    require => Package['ntp'],
    source  => "puppet:///modules/sshd/${conf_file}"
  }
```

RHEL tabanlı conf

Debian tabanlı conf

/etc/puppet/modules/ntp/files/ntp.conf.*

Puppet: Modules - Dizin Yapısı (5)

- /etc/puppet/modules/common/templates

- Templates dizininde, dinamik içerikli conf dosyaları tutulur.
- .erb uzantılıdırlar.
- Çok sayıda static conf dosyası oluşmasını önlemek içindir.

```
file { 'ntp.conf':
  path    => '/etc/ntp.conf',
  ensure  => file,
  require => Package['ntp'],
  content => template("ntp/ntp.erb"),
}
```

/etc/puppet/modules/ntp/templates/ntp.erb

Puppet: Modules - “Puppet Forge”

- Kendi modüllerinizi yazabileceğiniz gibi, başkalarının yazdığı modülleri de kurabilirsiniz.
- Puppet Forge Open Source Modül Reposu’dur. http://forge.puppetlabs.com/

```
# puppet module install puppetlabs-mysql
```

```
# puppet module list
```

Puppet: Nodes (1)

Hangi manifest, class ya da module'lerin hangi client'larda çalıştırılacağı "node" kod bloğu içerisinden belirtilir.

 # vi /etc/puppet/manifests/site.pp

```
node 'www1.example.com' {
  include common
  include apache
  include squid
}
node 'db1.example.com' {
  include common
  include mysql
}
```

Puppet: Nodes (2)

- Birden fazla node'u aynı block içinde tanımlayabilirsiniz:

```
node 'www1.example.com', 'www2.example.com' {
  include common
  include apache, squid
}
```

- Regex kullanabilirsiniz:

```
node /^(foo|bar)\.example\.com$/ {
  include common
}
```

Son Olarak

Otomatikleştirilebilecek hiç bir şeyi hiç bir zaman manual yapmayın….

Teşekkürler