PRESENTED BY RED HAT

LEARN. NETWORK.
EXPERIENCE OPEN SOURCE.

www.theredhatsummit.com

Case Study: Deploying Data Centers with Puppet

Rafael Brito
Lead Analyst, NYSE Euronext
Michael DeHaan
Product Manager, Puppet Labs
Jun 23, 2010

Objective of this Presentation

Share to the community the solution arrived by NYSE Euronext System Architecture and Engineering (SAE) to address the technical challenges imposed by the Business Strategy of the company:

- Consolidate various Data Centers across the globe in two brand-new state-of-the-art facilities, one in the US and another in Europe;
- Adapt a Global standard process to build and configure servers and applications across multiple countries and teams;
- Quickly deploy changes in large scale in an automated way.

Main Tools Used in the Solution

- RHEL Kickstart

  We have a single, very frugal kickstart profile for each RHEL release for the entire company.

- RH Network Satellite

  - Manage packages of the OS, Third Parties and Home-grown RPMs.

  - Manage patches and upgrades.

- Puppet

  - Work horse of this model

  - Common framework to apply and configure the Global standard server configuration and particularities of each environment and applications.

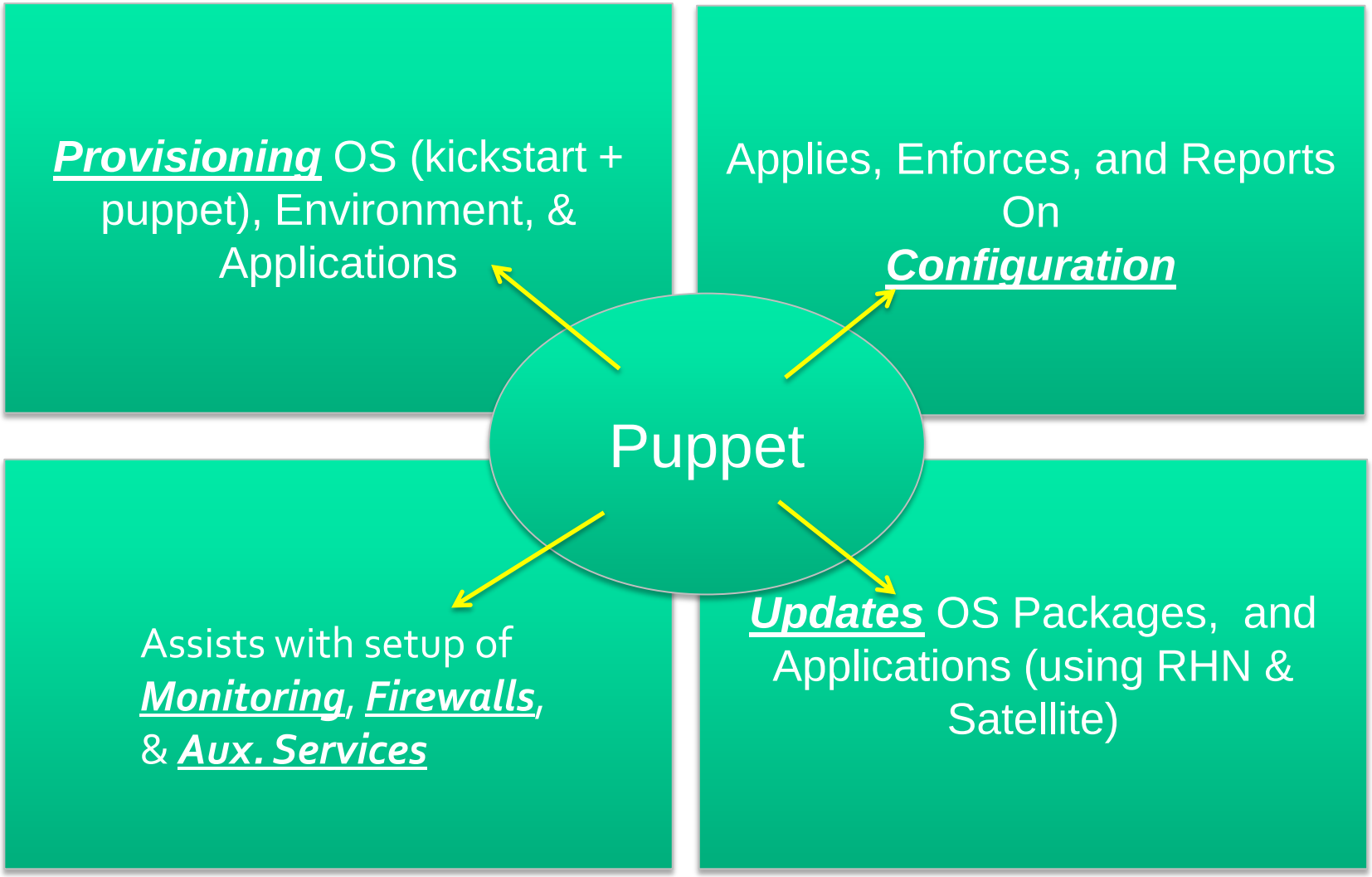
What is Puppet? Not a child's toy.

- High Level Configuration Management Tool and Framework. No more bash scripts.
- We focus on *what* things that need to be done instead on *how* things are done.
- Idempotent
- Example of puppet code (called manifests) to set permission a file:

```
file { "/etc/resolv.conf":
  permission => 0444,
  owner => root, group => root,
}
```

- Michael will give a quick overview of the tool

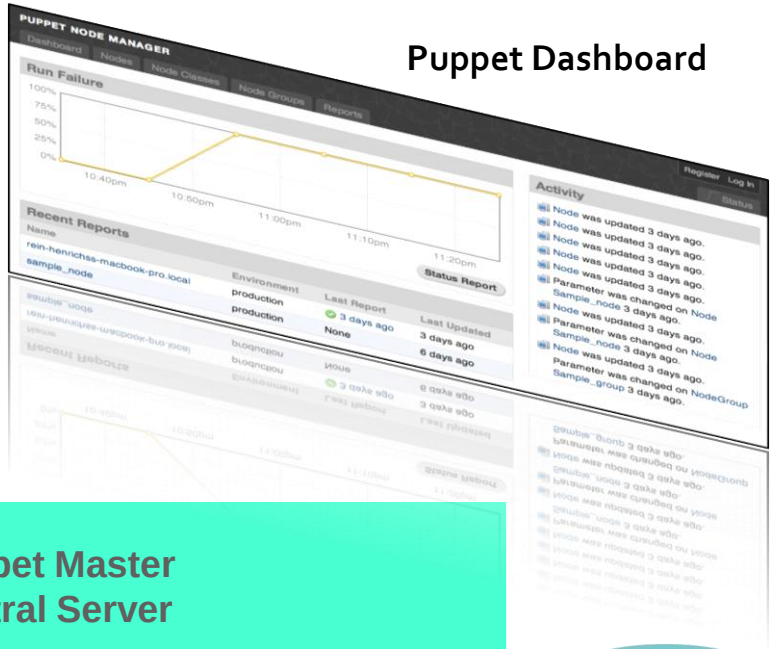
Puppet: An Infrastructure Hub



Puppet At A Glance

Puppet Modules In Source Control

Puppet Dashboard



Puppet Modules

- Configurations
- Applications
- Operating System

Puppet Master
Central Server

Reports

Database

Your Systems

Can Help Assist in Red Hat Migrations

Puppet: Cross Platform Management of:

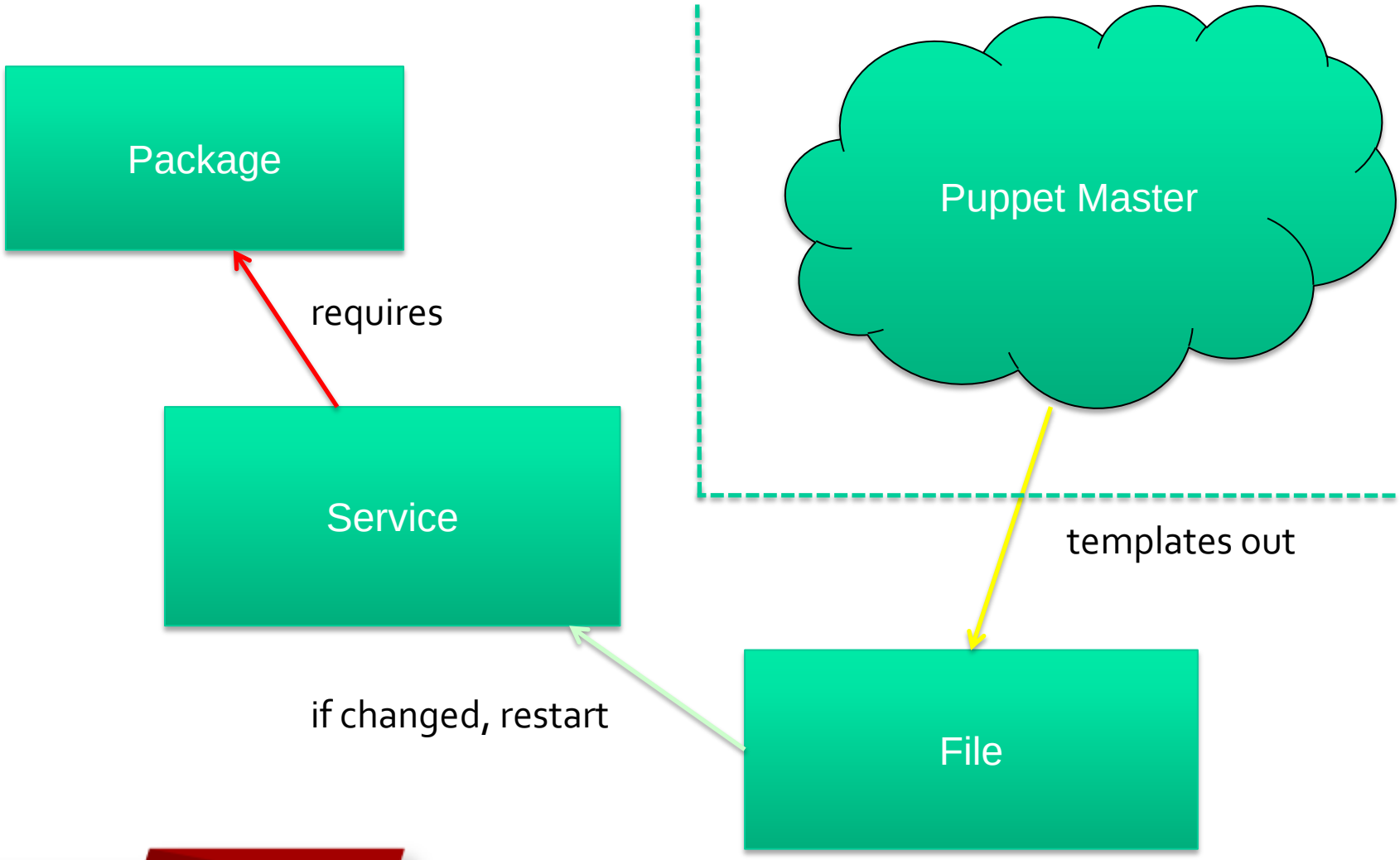
Web Applications	Databases	Users	Groups
Monitoring	Commands (Exec)	Cron	Services
Config Files	Mount Points	SSH Keys	Packages

And More. It's Also User Extensible.

Puppet: A Declarative System

```
user { ‘mpdehaan’:
    home => ‘/home/mpdehaan’,
    shell => ‘/bin/bash’,
    ensure => present,
    groups => [ ‘wheel’, ‘magic’, ’cdrom’],
    comment => ‘’,
}
```


Puppet: A Model Based System



Back to NYX Solution: The Three Configuration Layers Approach

We have organized our configuration needs in three levels:

Base: All NYX global customizations on top of the RHEL default settings. These must be propagated across the enterprise.

Example: global standard size for /usr file system, kernel parameter kernel.core_pattern = /var/crash/core.%e.%p.%h

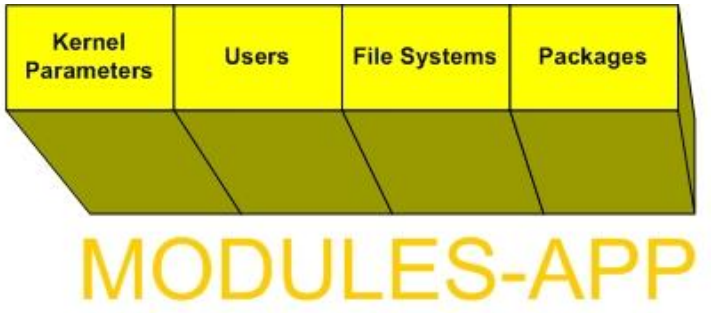
Zone (or environment or network): Configuration items common to a specific network or environment. Zone layer inherits the Base layer with the ability to override anything if necessary.

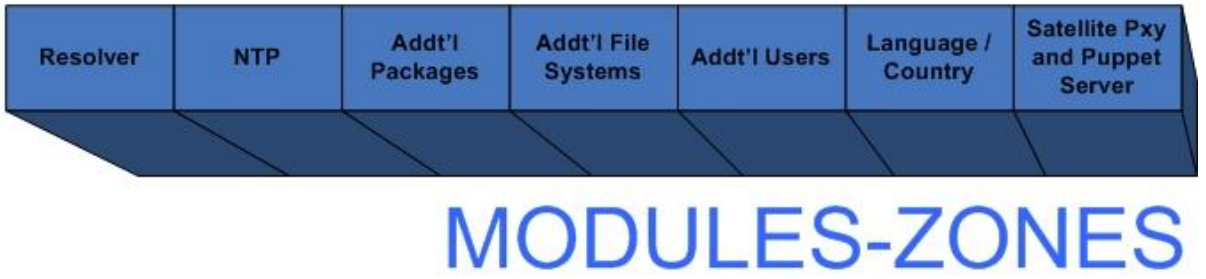
Example: /etc/resolv.conf file across one given production network.

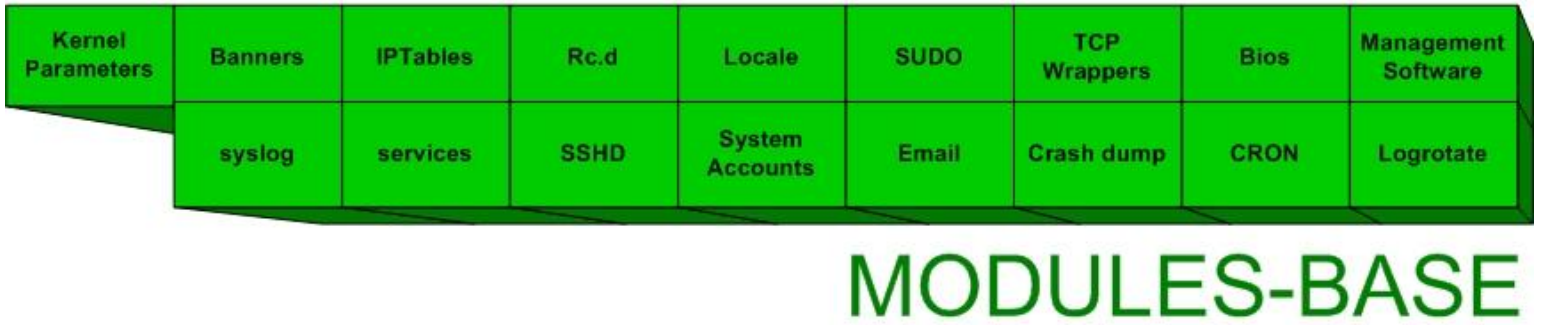
Application: Any specific configuration required by any given application. This layer inherits the Zone layer and has the ability to change anything set on the previous two layers.

Example: UTP application requires a 50GB /appl file system and a “qt” package installed.

The Puppet Modules behind of this approach







An example of the modules-base manifests

NYX Global Build customizes 250+ items on top of RHEL default OS.

All servers necessarily include the class “base” that setup the servers same way globally

```
class base::setup {
        include ssh::setup
        include banners::setup
        include sysctl::setup
        include services::setup
        include yum::setup
        include bootloader::setup
        include sudo::setup
        include hardware::setup
        # and many more classes
        }
```


An example of the modules-zones manifests

The organization has 30+ zones or environments across the globe, including production, QA, development networks, which one with own settings. Every server must belong to a zone.

Example of a zone class and some of its settings:

```
class us_trading::setup {

    $country = 'us'

    $searchpath = 'nyx.com'

    $domain = 'nyx.com'

    $nameservers = [  "10.0.X.X", "10.0.X.X" ]

    $timeservers = [ "10.0.X.X version 3 prefer", "10.0.X.X" ]

    $syslogserver   =   "10.0.X.X"

   # Including the base class

    include base::setup

}
```


An example of the modules-app manifests

We have more than 60+ different applications requiring their own settings (kernel parameters, file systems, users, packages).

```
class sg::setup {
    package { ["libxslt", "libmng"]:
        ensure => present,
    }
     lvm::config_lvm { "tcpdump":
        mount_point => "/tcpdump", lvname => "ltcpdump", vgname => vg00,
        lvsize => "5G", fstype => "ext3", owner => “myuser",
        group => "users", mode => "0755",
     }
}
```


How the three layers work together

- The zone class usually sets variables that will be used by modules-base for configuration.
- The zone class necessarily includes the modules-base.
- All servers must belong to one zone.
- The application module is coded to be pluggable to any zone (i.e. match engine application module can be on a QA environment or production environment)
- The ability to build anything, anywhere and mimic Production config to QA environment and vice-versa.
- The modules-app can override settings inherit on modules-zones. The modules-zones can override settings inherit on modules-base.

The Last Piece: the Node Definition File

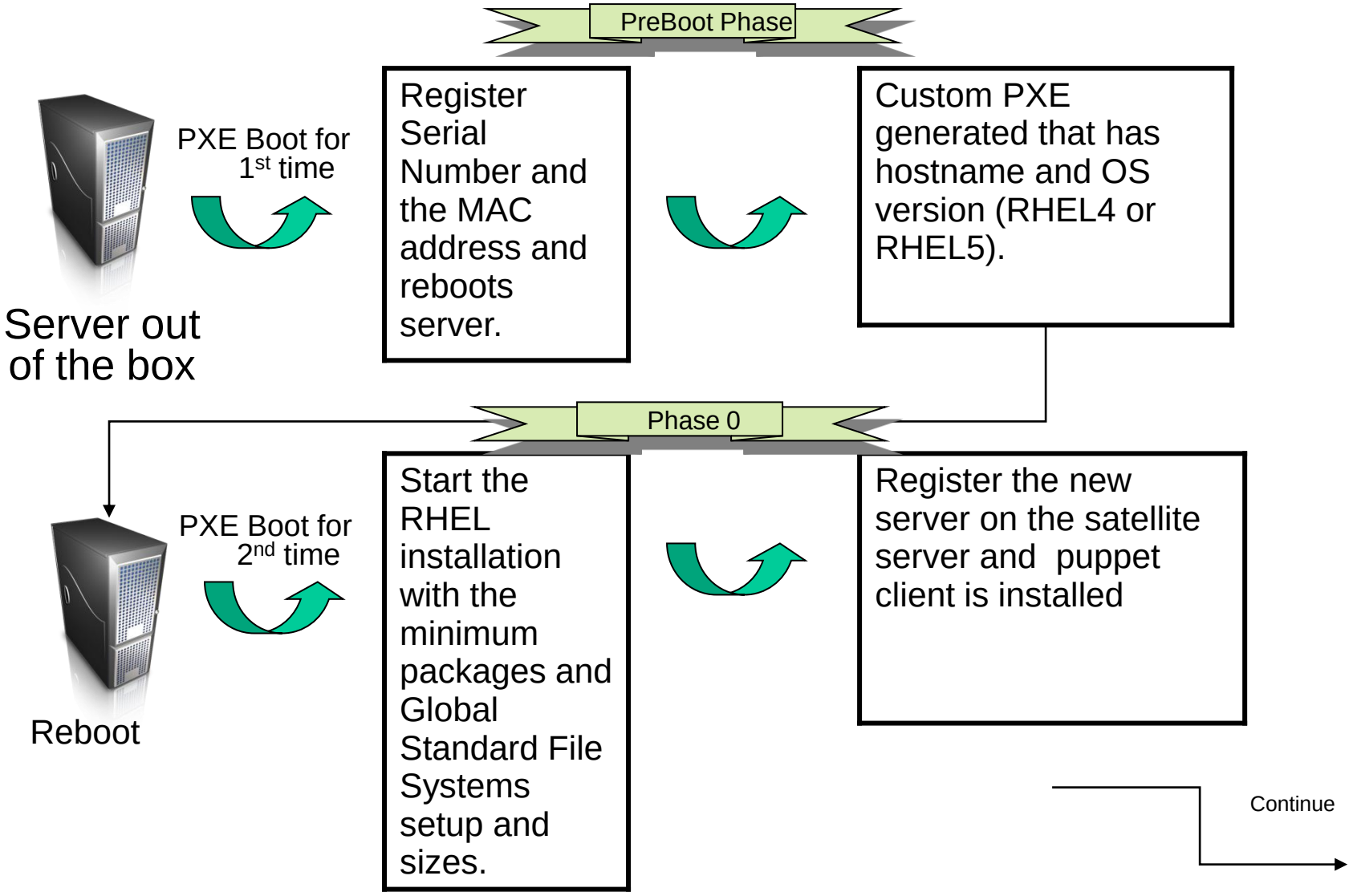
The node definition is the entity that assigns the server to the zone and application modules.

```
node "buildtest.nyx.com" {
                # Zone Class
                 include corplan::setup
                 # Application Class
                 include sg::setup
                 # Networking
                 network::interface::setup {"bond0":
                     ensure => present,
                     config_type => bonded,
                     ip => "10.0.0.100",
                     netmask => "255.255.255.0",
                     slave0 => "eth0",
                     slave1 => "eth1",
                     gateway => "10.0.0.254",
                     arp_target => "10.0.0.254",
                     arp_interval => "3000",
                 }
            }
```

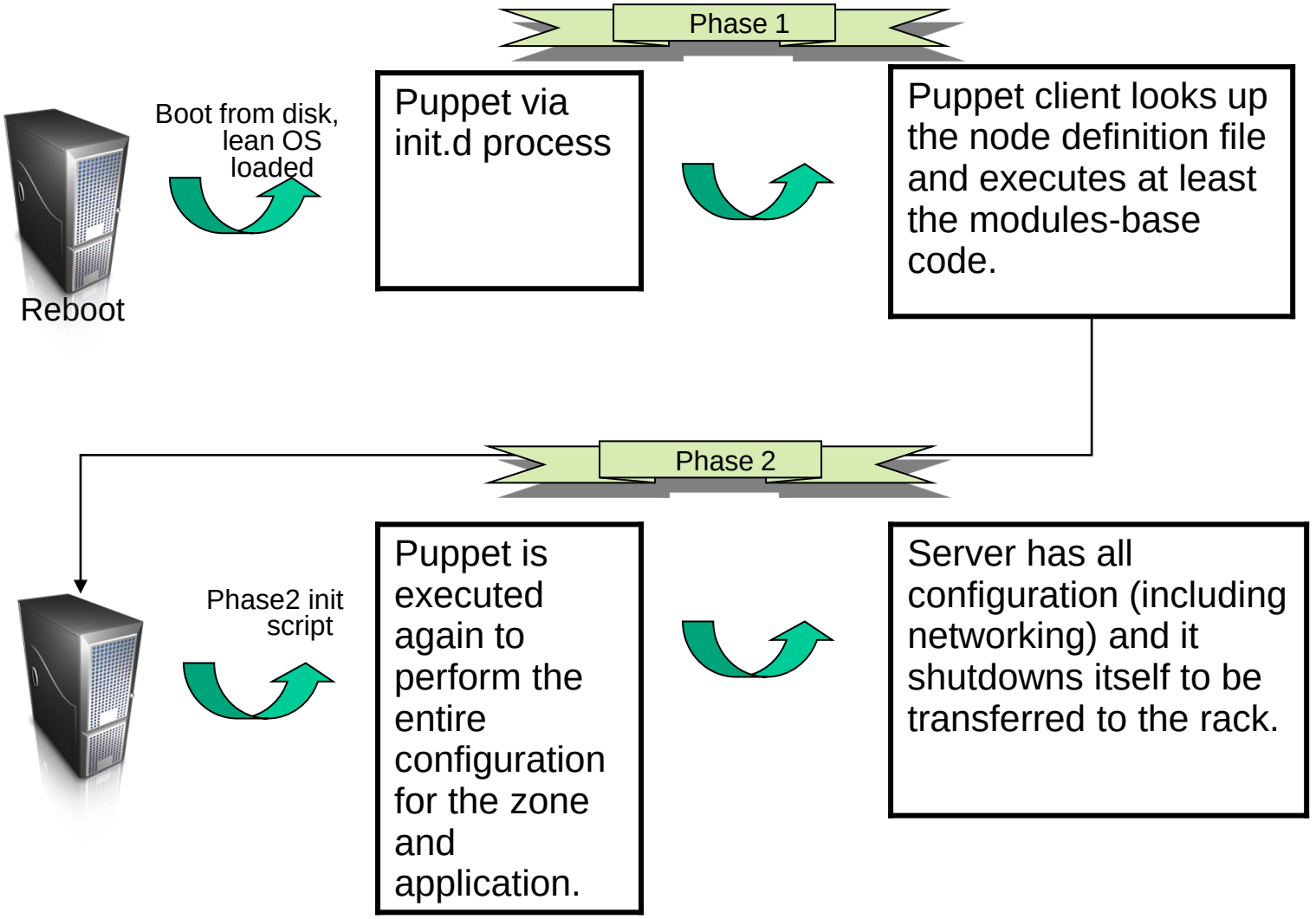

The NYX Build Server Components

- TFTP / PXE Server
- Proxy of the Satellite Server
- Apache Server
- MySQL Server
- Puppet Server

Automated Installation Process – In a Nutshell



Automated Installation Process – Overview Cont.



Data Center Deployment: Numbers and Lessons

- One server gets completely installed in 20 minutes in unattended mode
- We usually install up to 15 servers in parallel (after upgrading mongrel to apache).
- Usually, one individual in one day can install up to 50 servers by himself
- The most difficult part was getting the requirements of each application

Ancillary Tools for This Project

- Subversion
  - we had to disseminate the culture of version control among many individuals and teams.
  - Each layer of the modules and node definitions have its own subversion repositoriy
- Web Subversion
- OpenGrok (search the manifests)
- Redmine (ticket and changes control)
- Tidal Scheduler – To execute puppet client once the servers are in the production network

Next Steps on this Process

- Run puppet to deploy changes across the board on the day-by-day basis. This will require more a cultural change and coordination across more departments.

- Run puppet in report mode daily (no-intrusive) to detect discrepancies between the reality and the puppet manifests

- Use puppet dashboard as a central database of Node definitions (called External Node Classifier)

- Create more custom facts and types in ruby (diminish the number of execs inside the manifests).

Thank you!

Q & A

FOLLOW US ON TWITTER

www.twitter.com/redhatsummit

TWEET ABOUT IT

#summitjbw

READ THE BLOG

http://summitblog.redhat.com/