Scalable System Administration

Chris Moates
cmoates@gaggle.net

Fair Warning

You will likely lose internet access during portions of this presentation. If you are not comfortable with this, please attend a different talk.

Who am I?

- Started in the Internet industry around 1993
- 10 years with EarthLink, left as Staff Engineer for Internal Systems and Monitoring Infrastructure
- Lead System Administrator/ Systems Architect for Gaggle.Net

Purpose and Audience

- Explain why scalable administration is important
- Detail one set of methods for doing so
- Targeted at people managing systems, primarily UNIX based.
- Management types will probably get something out of it too.

Why do you care?

- You have a lot of systems, and not enough time.
- Your boss complains your systems are always broken.
- You have 12 months of requests outstanding to get around to finishing.
- You don't remember how this or that is configured, and waste time figuring it out each time you have to do it, which is only once every 3 years.
- You enjoy surfing the web instead of

What does scalable mean for us?

- We can do more without having more resources.
- We can keep up with the growth of our organization.
- We spend less time doing the same things over and over again, and more time doing new things for the first time.
- Our company doesn't see us as a

What can we do to scale?

- System Patterns
- Monitoring Systems
- Standardized Processes
- Automated Processes

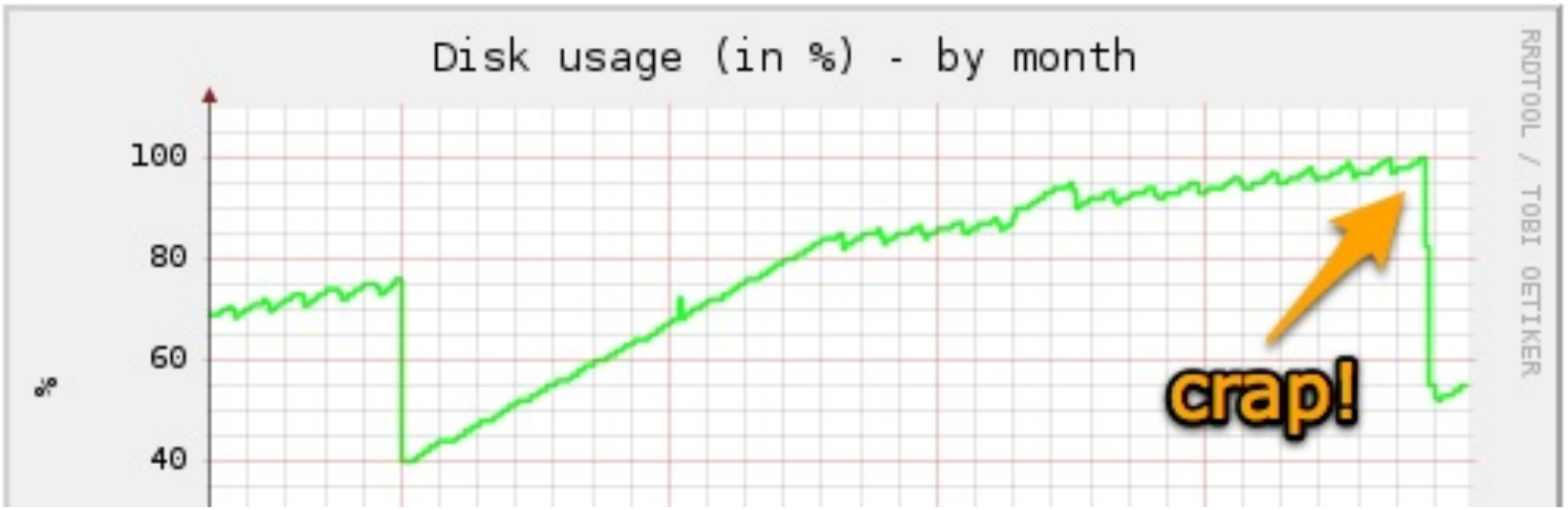
System Patterns

- Like design patterns for systems
- Reusable components
- Predictable I/O

Monitoring Systems

- Knowing what's wrong is far better than searching for what is wrong
- Good monitoring helps point out potential flaws in the system before they cause downtime

Disk utilization monitoring



Monitoring Systems

- Knowing what's wrong is far better than searching for what is wrong
- Good monitoring helps point out potential flaws in the system before they cause downtime
- Pretty pictures make bosses happy
- It's easier to justify expensive purchases when you can back it up with hard data

Standardized Processes

- Yes, I know.
- Writing documentation sucks.
- But it doesn't suck when you need it.
- Standardize your documentation!
- Machine readable documentation makes for easier automation.
- Documentation can assist with better monitoring, as well.
- Documented processes are repeatable processes.

Why you should document

- Documentation helps everyone
- Move boring and/or repetitive tasks to minions, or better yet, automate them entirely
- Less time spent training minions and management
- No need to remember the details of each and every facet of your infrastructure
- More confidence in a task being done properly, the first time
- Lower likelihood of irrecoverable

Automate your job

- Automation reduces administrative load (i.e. how much work you have to do vs. how much time you can spend surfing EBay)
- We already automate, but we fall short of our potential
  - Cron jobs
  - Adduser
  - The 43,549 little scripts we all write to "make things easier for ourselves"
- So, what's missing?

Automating your configurations

- Yeah, yeah, I know, it's not simple.
- Oh, wait, I lied. It is simple, with the proper tools.
- There are three main components to a systems' lifecycle:
  - Birth
  - Life
  - Death
- Each of these can be automated, as necessary.

Automating birth

- Disk images can be used.
- But they are pathetically bad:
  - New hardware may require a new image
  - New software does require a new image
  - Images have to be stored somewhere, and they aren't small.
  - Creating images is time consuming and doesn't scale.
- Automated installers like kickstart resolve these issues:
  - Hardware detection is done automatically
  - Software updates only require updating base software repositories (because you are using packages, aren't you…)

The problem with kickstart

- As stated, it automates birth.
- Fire and forget
- Couldn't care less if the machine is actually configured to be useful in some way
- Doesn't offer any mechanism to change a system post-birth, without killing and rebirthing.
- Complex configuration is tedious and error prone.
- Interactivity with kickstart is near zero,

The good about kickstart

- It makes automating installs reasonably simple.
- If left to the basics, it's not hard to get running.
- Presents you with a working base system in under 10 minutes.
- Cobbler (a separate tool) helps with the automatic configuration of kickstart, making this dead simple.

Automating life

- What we need is a tool which can configure our systems for us, whenever a configuration changes, without having to resort to death/birth all over again.
- Unfortunately, no such tool exists at this time.
- Oh wait, I lied. (again)

Puppet

- Puppet helps us accomplish the following:
  - Constant insurance that our systems are up to date and configured the way we designed.
  - Self-documenting system configurations
  - Confidence knowing that it's NOT a software configuration problem.
  - Reporting of changes made to our systems.
  - Reduces the amount of time spent working on individual systems.
- Puppet comes into play AFTER the

Puppet features

- Guaranteed standardization
- Centralized configuration documentation, in the form of “recipes”
- Centralized file storage
- Cross-platform compatibility
- Idempotency
  - This is **really** important
  - How many of your scripts would do Really Bad Things™ if they were run more than once?
  - sed ‘s/the/then/g’

Puppet In Practice

- The WAP you were using in this room is a standard CentOS 5.2 Linux server configured using puppet.
- At the beginning of this talk, we destroyed the filesystem.
- By now, it's probably up and running again.
- Kickstart and puppet, combined, take 13.93 minutes to set up the

Puppet In Practice (Part 2)

- For our AP, several classes were defined:
  - YUM repositories
  - Handy stuff I like to have on every server
  - Nagios (server monitoring software)
  - Squid HTTP cache
  - Hostapd (Access point software)
  - Apache HTTP server
- Each of these were included in our “node” configuration, which puppet applies automatically.

Puppet node configuration

```
node stormtrooper {
   include nice_stuff_to_have
   include "network_$hostname"
   include yumrepos
   include nagios_server
   include access_point
   include
  squid_transparent_cache
}
```

Puppet classes (squid)

```
class squid_transparent_cache {
  package {
    squid: ensure => latest;
  }

  file { "/etc/squid/squid.conf":
    source => "puppet:///files/apps/squid/squid.conf",
    require => Package['squid']
  }

  service { squid:
    require => Package['squid'],
    subscribe => File['/etc/squid/squid.conf']
  }
}
```

Puppet classes (apache)

```
class web_server {
  package {
    [httpd,php]: ensure => latest;
  }

  service { httpd:
    require => Package['httpd'],
    subscribe => Package['php']
  }
}
```

Puppet classes (base)

```
class nice_stuff_to_have {
  package {
     emacs: ensure => latest;
     jed: ensure => installed;
     screen: ensure =>
 installed;
     mtr: ensure => installed;
   }
}
```

Puppet classes (yum repos)

```
class yumrepos {
  yumrepo { centos-local:
    descr => "centos-local",
    baseurl => "http://puppet.mox.net/kickstart/32bit",
    gpgkey => "file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-5",
  }

  yumrepo { localrepo:
    descr => "localrepo",
    baseurl => "http://puppet.mox.net/kickstart/localrepo/i386/RPMS",
    gpgcheck => 0,
  }

  file { "/etc/yum.repos.d/CentOS-Base.repo":
    ensure => absent;
  }

  file { "/etc/yum.repos.d/CentOS-Media.repo":
    ensure => absent;
  }
}
```

Puppet classes (nagios)

```
class nagios_server {
  include web_server

  package {
    [nagios,nagios-plugins-all]: ensure => latest
  }

  file { "/etc/nagios/cgi.cfg":
    source => "puppet:///files/apps/nagios/cgi.cfg",
    notify => Service['nagios'],
    require => Package['nagios']
  }

  file { "/etc/httpd/conf.d/nagios.conf":
    source => "puppet:///files/apps/nagios/nagios.conf",
    notify => Service['httpd'],
    require => Package['httpd']
  }
```

Puppet classes (nagios)

```
file { "/etc/nagios/passwd":
    source => "puppet:///files/apps/nagios/
  passwd",
    require => Package['nagios']
  }

  service { nagios:
     require => [Package['nagios'],
                 Package['nagios-plugins-all'],
                 Class['web_server']],
  }
}
```

Puppet classes (access

```
class access_point {
  package {
    "madwifi-hal-kmdl-2.6.18-92.el5.i686": ensure => latest;
    madwifi: ensure => latest;
    dnsmasq: ensure => latest;
    hostapd: ensure => latest;
    iptables: ensure => latest;
  }

  file { "/etc/modprobe.conf":
    source => "puppet:///files/apps/hostapd/modprobe.conf"
  }

  exec { "/sbin/rmmod ath0;/sbin/modprobe ath0":
    subscribe => File['/etc/modprobe.conf'],
    notify => Service['hostapd'],
    refreshonly => true
```

Puppet node configuration

```
node stormtrooper {
   include nice_stuff_to_have
   include "network_$hostname"
   include yumrepos
   include nagios_server
   include access_point
   include
  squid_transparent_cache
}
```

Standard puppet workflow

- Kickstart a server
- Puppet brings it into sync
- 20 minute sync interval
- Modify class files as changes are needed
- Update package repositories
- Surf the web a lot more

Simplify Your Structure

- If a one-off server isn't required, don't build it
- Standardize your configurations
  - Enabled by puppet
  - You only need one sudoers file
  - LDAP for removes many headaches of user administration
  - Choose simplified tools when available
  - Reduce storage points
    - PowerDNS vs Bind
    - LDAP vs /etc/passwd, /etc/group, /etc/shadow

Conclusions

- Gaggle's infrastructure was unmanageable prior to standardization
- Was 8+ hours a day of keeping up with failures and inconsistencies
- Now 30 minutes a day of managing minor failures
- Changing architectures or distributions is nearly effortless
- More time is spent tweaking the infrastructure in new and interesting ways
- Backouts are trivially simple

Bonus Materials

- Load balancing
- Deeper puppet configurations
- High availability
- Virtualization