Agenda

- Intro (being in the open source track)
- Terms
- Update Strategies
- Case Study
- After thoughts
- Conclusions
Who am I?

- Linux Administrator at a Fortune 100 Company
  - Worked in large shop for 5 years
  - 1000+ systems scattered throughout the world
- Separate Business Unit Now
  - Responsible for 7 Administrators
  - 300+ systems in 3 continents
- Author – Pro OpenSSH
- Involved in the Fedora Project
  - EPEL – steering committee
  - Infrastructure
  - Ruby SIG
- Former CISSP
Why are updates difficult?

- Number of Updates is staggering
- Hundreds per release per year
- 3rd Party Updates
- Security Updates
- Point releases
- Rebuilds from Bugzilla or people.redhat.com/fedorapeople.org
- Your own rpms
My Terminology

• System – a single instance of an OS (normally Linux in this case)

• Environment -- group of Systems performing the same/similar functions. (e.g. All web servers, all database servers, etc)

• Fleet – All Environments (and thus all systems) that are managed by your group. Fleet == 100% coverage.
My Terminology

Server Management
- Each Server Hand crafted by System Artisans
- Each server built to specifications known at the request time.
- Changes independent of other systems
- Smallest entity is the server

Legacy Hardware Management
Customer Support Issues

Environment Management
- Servers built to be in Environments in assembly-line like manner.
- Changes are cumulative per environment and distributed through configuration tool.
- All servers in environment managed holistically.
- Smallest entity is the application environment

Cluster management
VMware Farm Management
Class of servers (All Oracle Servers, all Weblogic servers)

Fleet Management
- Servers built to be in the Fleet in assembly-line like manner.
- Changes are cumulative per the entire fleet and distributed through configuration tool.
- All environments managed holistically.
- There is only one entity

Complete management
Configuration Settings and best practices
Package Management
Systems Architecture
Common Terms

• Release Manager – The person (or role) responsible for decisions on what gets released for an update on your fleet or environment

• Application Teams – The real customers of your services. This could be external, internal, the same team, business users, etc
Common Terms

- **Upstream** – The organization providing you with your Linux (Red Hat, Fedora, CentOS, etc)
- **3rd Parties** – Parties other than you or upstream. Software from EPEL, DAG, , RPMFusion, IBM, Oracle, etc
- **QA** - Quality Assurance. The testing process of updates in your environment. Ensure your applications work (3rd party, internal, home-grown, purchased, etc)
- **P0wned** – Having a new system administrator that isn't authorized
Common Strategies for Updates

- Do Nothing
- Update if you find a bug
- Update when major releases hit
- Update security patches only
- Update critical security patches only
- Update key components
- Apply all Upstream Updates
- Auto-apply updates at pace of upstream
Do Nothing

- Low cost (no matter size of environment)
- Low effort (no matter the size)
- Good for poorly designed environments
- High risk in terms of security
- Often requested in 24x7 factories, embedded systems, etc
Update when required

- When you find a bug, (and a fix) update the effected package(s)
- Low cost
- Relatively low amount of effort
- Still high security risks
- Good for reactively managed environments
- Merits little QA testing other than the package with the bug
Update when major updates hit

- Update quarterly (or so) per EL released Updates
- Cost is in relation to amount of QA
- Effort can vary based on variance in the environment
- Periodically closes security gaps
- Good for environments with strict change control, small outage windows, etc
Security Updates Only

• Update security-based updates only
• Cost varies depending on implementation
• Difficult to parse what is a security update < EL 5
  • Direct Yum feature not available < EL 5
  • Manual sorting of updates could be required
• Minimizes security risks
• Some security packages could have dependencies that are not security-related
• QA still required for updates, but is often shortened
Critical Security Updates Only

- Update security-based updates marked CRITICAL
- Cost varies depending on implementation
  - Cost goes up because more parsing of updates is required
  - Cost goes up because application process is probably home-grown. (Can't use security-only feature of yum)
  - Cost could be less, because few updates are critical
- Manual sorting of updates required
- Minimizes major security risks
- Some security packages could have dependencies that are not security-related
- QA is often shorted for critical updates
Update Key Components

- Update primary components used for system (e.g. Update MySQL, Apache on a LAMP system)
- Requires intimate knowledge of each system which can raise cost
- Cost is based on what components are deemed critical
- No out-of-box delivery method via yum, RHN, etc
- Unknown security benefits
- QA iterations will vary per environment type
- Not really addressing all aspects of updates
Apply All Upstream Updates

- Apply all upstream updates after QA validation
- Cost is high due to QA
- Requires testing of all environments in fleet
- Requires intimate knowledge of each system which can raise cost
- Easy to automate (yum -y update, up2date -u, etc)
- Security gap closed as quickly as you deploy after upstream releases
- Role of release manager minimalistic
Apply All Upstream Updates Automatically

- Apply upstream updates ASAP
- Cost lowered because you trust the QA of Upstream
- Risk is higher, you have tested nothing yourself
- Reboots sometimes required for updates to take effect
- Easy to automate (yum -y update, up2date -u, etc)
- Security gap closed faster than any other method
- Role of release manager minimalistic
- Unknown effect on 3rd party applications (support, functionality etc)
- Non-official upstream RPMs could cause dependency breakage, thus having no updates apply
## Strategy Analysis

<table>
<thead>
<tr>
<th>High Effort</th>
<th>Low Effort</th>
<th>* Do Nothing</th>
<th>* Update when applicable</th>
<th>* Key components</th>
<th>* Major Releases</th>
<th>Security Patches Only</th>
<th>* Critical Security Patches Only</th>
<th>* Everything upstream</th>
<th>* Auto-apply everything upstream</th>
</tr>
</thead>
<tbody>
<tr>
<td>* Proactive</td>
<td>* Reactive</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What's the right choice?

• How many non-upstream packages do you have?
• Do your 3rd parties specify exact versions of libs, kernel, etc that you must be at for support?
• How much is downtime worth for your system/environment/fleet?
• What is your change control process?
• Do you have clear requirements from customers?
• What security policies do you have in place?
• Do you practice other risk mitigation methods?
• Are all of your systems network accessible?
• How confident are you in your ability to do proper QA?
A simple (but common) case study

- A group has 600 RHEL (3/4/5) Systems to manage
- Several 3rd party applications installed, using 3rd party repos also
- 36 application teams (environments), 10 Linux Administrators
- Using Home-grown RPMS on all versions
- Current state has systems at various update levels
A simple (but common) case study

- Unplanned downtime for any production system is 100,000$ per hour
- Planned downtime may be arranged monthly
- Some systems have highly confidential data
- Policy says all updates must be tested at least 2 weeks before being put in production
- No formal QA Process currently in place
Evaluate requirements

• Cost of downtime is high, and policy says updates must be tested before applied → can't use auto-updating upstream
• Large number of environments → difficult to use component-based updating
• Using RHEL3/4/5 so applying security only updates would require coding/process design
• Planned downtime not too hard to schedule
• Due to confidential nature of systems, updates should be applied as often as possible
Design

- Monthly updating/patching Cycle
- Minimally 2 weeks behind upstream in production
- Identify systems in each environment for QA usage
- All updates should have a back-out plan
- Various 3rd party tools to be validated during QA
- Home-grown RPMS to be rebuilt/updated for each release (if needed)
Release Map (upstream vs case)

- Deploy your own frozen update level to the fleet
- Deploy to your QA systems minimally 2 weeks prior to production releases
- Decide on what packages will be included in your release
- Have a reporting mechanism to ensure you have deployed successfully
- Create QA action plan
- Run two freezes in parallel (prod and pre-prod)
Release Deployment Methods

- RHN hosted
- RHN Satellite
- Private up2date/yum repositories
- Hand-pushed via scp, func,
- Puppet/Cfengine/bcfg2 (using another method internally)
RHN hosted (pros and cons)

Pros
• Always updated with latest packages
• Always at latest version of RHN
• Less overhead for managing

Cons
• At the mercy of RH for Downtime
• Speed can vary depending on location
• No ability to customize many portions
• Any custom packages will require additional deployment methods
• Doesn't work with CentOS, Fedora, Scientific, et al
• All systems must be able to access internet
RHN Satellite (pros and cons)

Pros
• Uses native updating techniques of RHEL
• Allows for custom channels and freezing of content
• Hosted behind firewalls, so not all systems need Internet access
• Should speed up content delivery
• Outages under your control
• Can schedule updates and report upon success/failure

Cons
• Again, no support for non-RHEL
• Have to manage the satellite server/application
• Not Open Source
Private Repository (pros and cons)

Pros

• Not tied to RHEL-specifically
• Maximum control allowed
• No software cost to have multiple mirrors or locations
• Easily add your own or 3rd party packages
• Outages under your control

Cons

• Support not standardized (requires more skilled maintainers)
• Lack of reporting built in
Hand Pushed (func, scp) (pros and cons)

Pros
- Minimal repository maintenance or setup
- Same tool may work on Non-Linux variants

Cons
- May not have dependency resolution
- Very manual to manage
- May require additional software to be installed on each node
Configuration Management Tools (puppet/cfengine/bcfg2) (pros and cons)

Pros

- Can easily incorporate most other methods
- Means environment is managed in a more mature manner
- Same tool may work on Non-Linux variants

Cons

- Steeper learning curve
- Other change control around configuration management software could slow deployments
- May require additional software to be installed on each node
Back to the case study

- Monthly Updating/Patching Cycle
- Deployment will be using a configuration tool in conjunction with custom yum/up2date repository
- QA process still to be determined
- Release manager to handle communication
Communication Plan

- Inform application teams of scope of release
- Provide them with information on what packages, specifically key packages (kernel, glibc, driver updates, etc) will be moving.
- Allow customers to develop a QA plan
- Give advance notice to customers for deployment in production and possibly QA
Define a QA process

- Often times the hardest (and therefore most-skipped) step
- QA is dependent on having a cost-benefit analysis
- QA requires dedicated systems
- Automated where possible (given enough time and money, it's all possible)
- Leverage available technologies and tools (Virtual Clones, LVM snapshots, testing suites, etc)
Developing QA Plan

- May require a multi-generational plan
- Difficult to automate until requirements are stable
- Be sure to test home-grown RPMs or any layered products (EMC PowerPath, IBM Sdd, Veritas Foundation Suite, Commercial Backup Software, etc)
- Look for specific conflicts, and obsoletes of packages
- Note key differences between version of your upstream provider’s OS
- You may need different QA plans for each environment
Case Study QA Methods

• Communicate package scope to application teams
• Take a full backup/snapshot of current environment before applying updates
• Create channel or folder for the current frozen update and subscribe QA systems to this channel
• Create scripted method of installing updates through configuration management tool(s)
• Apply updates to QA systems in each environment, representational of your fleet
• Test functionality of applications in conjunction with application teams
• Resolve issues with 3rd party packages, upstream and application teams (sometimes difficult)
Success Criteria

- If QA was successful and met all success criteria, re-communicate with application teams
- Schedule outages (if needed) for updates
- Ensure nothing has changed since QA deployment and testing in your release freeze
- Deploy release
- Ensure all is right in production
- Provide post-deployment analysis with particular attention to cost vs benefit
Afterthoughts
The challenge of 3\textsuperscript{rd} Party

- Some ISVs don't recognize the no ABI breakage of an EL update. So they only certify on a specific Update.
- Some Layered products or 3\textsuperscript{rd} party device drivers depend on specific kernels and often lag behind upstream releases.
- Some 3\textsuperscript{rd} parties deprecate support for an EL version before upstream does.
- Sometimes 3\textsuperscript{rd} parties are using specific bugs as features and when they are fixed, their tools break.
- RPMs of 3\textsuperscript{rd} parties are often mediocre at best. They often do lots in \texttt{%pre}, \texttt{%post} and may not own all files, or provide upgrade paths.
- Recommendation is to minimalize 3\textsuperscript{rd} party RPMs if possible.
The (in)sanity of point releases

- Upstream EL is not consistent in usage of update levels or point releases
- EL < 5 uses Update # methodology, but was commonly referred to as a X.Y release where Y is the update number
- EL 5 uses X.Y release formally, but offers two streams for packages → 5.1.1 vs 5.2.0
  - In this case some items may have ABI breakage, or at least newer versions in 5.1 → 5.2 update, but 5.1.1 won't.
  - This means more overhead for management and more confusion for Linux system administrators
  - You now may have ISVs required 5.1.1 and others requiring 5.2.x and thus causing another fork in your environment
  - 3rd Party repositories haven’t all figured out how they will handle these separate streams
A few notes on 3rd Parties

- Extra Packages for Enterprise Linux (EPEL) – sponsored by Fedora, follows Fedora packaging Guidelines and will never trample a package provided from core EL
- RPMFusion – also follows a majority of packaging guidelines but may offer kernel modules or certain types of US restricted content
- Be wary of 3rd parties that provide you with replacements for core RPM features of your upstream
- Be wary of attempting to use multiple 3rd party repositories simultaneously. This may cause issues. Can sometimes be overcome with yum priorities or other plugins.
Creating Your Release Process

- Cost-benefit Analysis
- Requirements
- Design
- Testing
- Implementation
- Evaluation
- Repeat
Risks you can't fix with updates

- Running clear-text protocols (mmm....telnet)
- SSL/SSH key vulnerability from Debian (you can update SSH, but your key could still be p0wned)
- Rootkits
- DNS Spoofing
- Misconfiguration
- Poor architecture
Other methods of Risk Mitigation

- Mandatory Access Control (SELinux, 3rd party commercial tools, etc)
- Host-based firewalling
- Disabling vulnerable services
- Network layer firewalling
- Intrusion Prevention/Detection
Afterthoughts

- Build Process
- Setting your customer expectations
- Multiple methods inside a fleet
- Technical Issues with Freezes
- Mock (your own updates)
- Freeze/merge/release cycle
- Other updates types – Firmware, BIOS, 3rd Party apps, etc
Sources


Pictures

http://flickr.com/photos/shellsonthefloor/2523236728/
http://farm4.static.flickr.com/3148/2316034104_abe020dc43.jpg?v=0
http://flickr.com/photos/innac/2044863930/
http://flickr.com/photos/dunechaser/518821732/
http://flickr.com/photos/eyes_manish/2114674028/
http://flickr.com/photos/tidewatermuse/134222791/
http://flickr.com/photos/lwr/241471902/
http://fedorahosted.org/func
http://flickr.com/photos/angelala77/2308000612/
http://flickr.com/photos/alwinsulaiman/280400640/
http://flickr.com/photos/mintytrina/2371224505/
http://flickr.com/photos/2inches/484133264/
http://flickr.com/photos/icesabre/2219876290/
http://flickr.com/photos/37925259@N00/159174764/
http://flickr.com/photos/existentist/666727475/
http://flickr.com/photos/drinus/123261849/
http://flickr.com/photos/tijfex/233597894/
http://flickr.com/photos/curoninja/2313807903/