Yum at scale
Keeping the software of your infrastructure under control

Luis Fernando Muñoz Mejías

Universiteit Gent

CentOS Dojo, Madrid 2013
Outline

1 Introduction
   - Large installations
   - What we’ll be talking about

2 Declarative package manager
   - Portage as a role model
   - Making Yum declarative
   - Configuring repositories wisely

3 Keeping a lot of hosts consistent
   - Upgrade policies
   - Repository-based upgrade policies
   - Operating with versioned repositories
   - Real case

4 Conclusion
Large installations are a mess
Yum’s not meant for clusters

- All tools we use for managing our clusters are declarative
  - Quattor, Puppet, Chef...
- But Yum is imperative
  - `yum install foo`
  - `yum remove bar`
  - `yum upgrade`
- We’ll show how to “abuse” Yum into a declarative model
You never upgrade all your infrastructure at once

- Test environments
- Critical services
- Very exposed services
- ...

And you want to control *when* upgrades take place
What comes next?

- Making Yum an almost declarative package manager
- Keeping software installations of lots of hosts aligned
Lessons learned from Gentoo
Package sets in Portage

- Package sets can contain other sets
- The **world** set contains all the packages declared by the user
- The **system** set contains all the “protected” packages
- Any package not in world or system is either a dependency or not needed anymore

**Figure**: Package sets
Virtual dependencies

- Allow to choose the implementation of certain services
- Better than RPM’s provides and requires

Figure: Virtual dependencies
Update vs. distro-sync

**update** install any newer versions available

**distro-sync** align yourself to the state of the repositories

- Install newer versions
- Remove packages that leave the repository
- Downgrade versions that leave the repository

- In Yum, repositories *declare the desired state*
- In Yum, repositories *are the source of truth*
Controlling which versions to install

- Some packages cannot change versions at the same speed as the repositories

Example

- Upgrade kernel to 2.6.32-358.456
- But the Infiniband drivers are available only for 2.6.358.123
- You lose Infiniband if you upgrade
- There is a critical security exploit on OpenSSL and you must upgrade
Controlling which versions to install

- Blacklisting unwanted versions may work...

**Example**

```
exclude=kernel-2.6.32-358.456.el6
```

- ... until a newer kernel version appears and you have to blacklist that one too
The versionlock plugin

- We should be able to declare the versions of some packages, and not let them move
- ... and that’s what the versionlock plugin does

Example

```
yum versionlock 0:kernel-2.6.32-358.123.el6.x86_64
```

- When your new IB drivers appear, just unlock the kernel or upgrade the version lock
Cleaning up the system

@world

@editors

firefox  nano  emacs  vim

gtk

glib

chrome  mplayer

@gcc  @gmake  @glibc  @autoconf  @python

@system
Cleaning up the system

- There is no equivalent for the *world* set in Yum
- Your configuration management tool should wrap it
- Quattor does
- On CentOS 6, set `clean_requirements_on_remove`
The post-transaction-actions plugin

- After a package operation, some things should happen

**Example**

- Restart Tomcat after upgrading/downgrading OpenJDK
- Restart Apache after upgrading/downgrading OpenSSL

- The post-transaction-actions plug-in does this
We can run arbitrary scripts on install, remove, upgrade, downgrade of packages

Example

openssl:upgrade:service httpd restart
java*:upgrade:service tomcat restart
Yum problems

- The shell doesn’t report errors correctly

Example

```shell
$ yum -y shell
install idonotexist
transaction run
exit
$ echo $?
```

- Large, complicated transactions may raise conflicts
  - Lots of operations with repoquery to prevent them
- The Quattor wrapper around Yum is 411 lines of Perl code
The case of repoforge

- Repoforge (AKA RPMForge) contains lots of useful packages
- But it overlaps with CentOS, and the builds may be incompatible with each other
- Use it only for selected packages

Example

```
[rpmforge]
name=rpmforge
...
enabled=1
includepkgs=dstat perl-IO-Pty-Easy
```
Advise for repository mirroring

- Enable only the minimum set of repositories you need on your system
- Mirror existing upstream repositories when possible
- Don’t mix different upstream repositories in a single local repository
- ... but you still need repositories for stuff you package yourself
- Try to keep them small
Avoid Yum priorities
Avoid Yum priorities

Desired state

Raising the priority of repoforge
Avoid Yum priorities

CentOS
- git
- perl-JSON-XS
- perl-IO-String

RPMForge
- git
- perl-JSON-XS
- perl-IO-String

Desired state

CentOS (priority=5)
- git
- perl-JSON-XS
- perl-IO-String

system

wrong!

RPMForge
- git
- perl-JSON-XS
- perl-IO-String

Raising the priority of CentOS
Requirements for an upgrade policy

- Upgrades must happen frequently
  - Bug fixes
  - Security fixes
  - New cool features

- But we need stability
  - And changes carry some risk
  - Risk depends on the service, exposure...

- Different portions of the infrastructure need different upgrade frequencies

- Surprises are not allowed
Requirements for an upgrade policy

- Upgrades must happen frequently
  - Bug fixes
  - Security fixes
  - New cool features

- But we need stability
  - And changes carry some risk
  - Risk depends on the service, exposure...

- Different portions of the infrastructure need different upgrade frequencies

- Surprises are not allowed
Many dimensions of an upgrade

\[ \vec{\text{system packages}} = f (\text{service}, \text{date}, \text{risk}_{\text{upgrade}}, \text{risk}_{\text{upgrade not}}, \ldots) \]

\[ \text{risk} = g (\text{exposure}, \text{cost of downtime}, \text{resources}, \text{service}, \ldots) \]
Repository management

- Probably we all mirror a lot of repositories into our organisation
- What do you use for the mirroring? And for the access control?
Repository management tools

- Does any of you use SpaceWalk? Satellite? Pulp?
- What is your opinion on those?
Most tools are based on having different repositories for different needs.

Example:

epel, epel_prod, epel_test, epel_devel, epel_uat, ....

- How do things move from _test to _prod?
- On large environments, the number of repositories explodes
- And reverting to old states is not simple
Versioned repositories

\[
\begin{pmatrix}
\text{centos} & v_{\text{centos}} \\
\text{epel} & v_{\text{epel}} \\
\text{rpmforge} & v_{\text{rpmforge}} \\
\text{local} & v_{\text{local}} \\
\ldots & \ldots
\end{pmatrix} = f (\text{service}, \text{date}, \text{risk}_{\text{upgrade}}, \text{risk}_{\text{upgrade not}}, \ldots)
\]
Figure: Subscriptions to repository versions per host
Repository versions must be immutable
  - Having _dev, _test, ... repositories doesn’t guarantee that

All systems on version X of a repository must see the exact same repository at all times

Easy to do with filesystem snapshots
  - LVM
  - ZFS
  - BTRFS
  - NetApp
  - ...

Upgrading systems

- Declare a newer version for your repository
- Run `yum distro-sync`
Rolling back a failed upgrade

- Declare a previous version for the repository
- Run `yum distro-sync`
Locking the kernel among upgrades

- Use the versionlock plugin
- Move the versions of any repositories at will
- Run `yum distro-sync`
HPC@UGent snapshotting architecture

GET /snapshot_a/repodata
Figure: List of mirrored and home-made repositories
Figure: List of snapshots
Subscribing to a version of a repository

```bash
[epel]
name=epel
baseurl=http://a.host/20131008/epel_el5
enabled=1
...
```
Wrap-up
Questions?

![Image of a Mars rover labeled "CURIOSITY" and "KILLED THE CAT" with a link to 9GAG](quickmeme.com)
The $g$ logo is property of the Gentoo Foundation.