Staging Package Deployment via Repository Management

Chris St. Pierre
Matt Hermanson

Background

- (Mostly) homogeneous environment
- Organizational structure
- Bcfg2

Our Approach

- Control what packages are available in the repository
- Define classes of repositories
  - Upstream/Stable/Unstable
  - Infra/HPSS/clusters
- Clients are always up-to-date with repository
- Centralized management

Other solutions

- Yum excludes
- Spacewalk
- Bcfg2 version specification
- Yum versionlock

A solution: Pulp

- Part of Red Hat’s CloudForms
- Repos can be “cloned” efficiently
- Sync mediated by filters
- Manual manipulation

Workflow

- Tiered repositories
  - Upstream – daily sync from upstream
  - Unstable – filtered sync from upstream
  - Stable – filtered sync from unstable
- Custom repositories branched from upstream
- Package promotion separated by time and/or manual intervention

Workflow

- How do we implement filters
  - Whitelist and blacklist packages
- Manual package promotion and removal

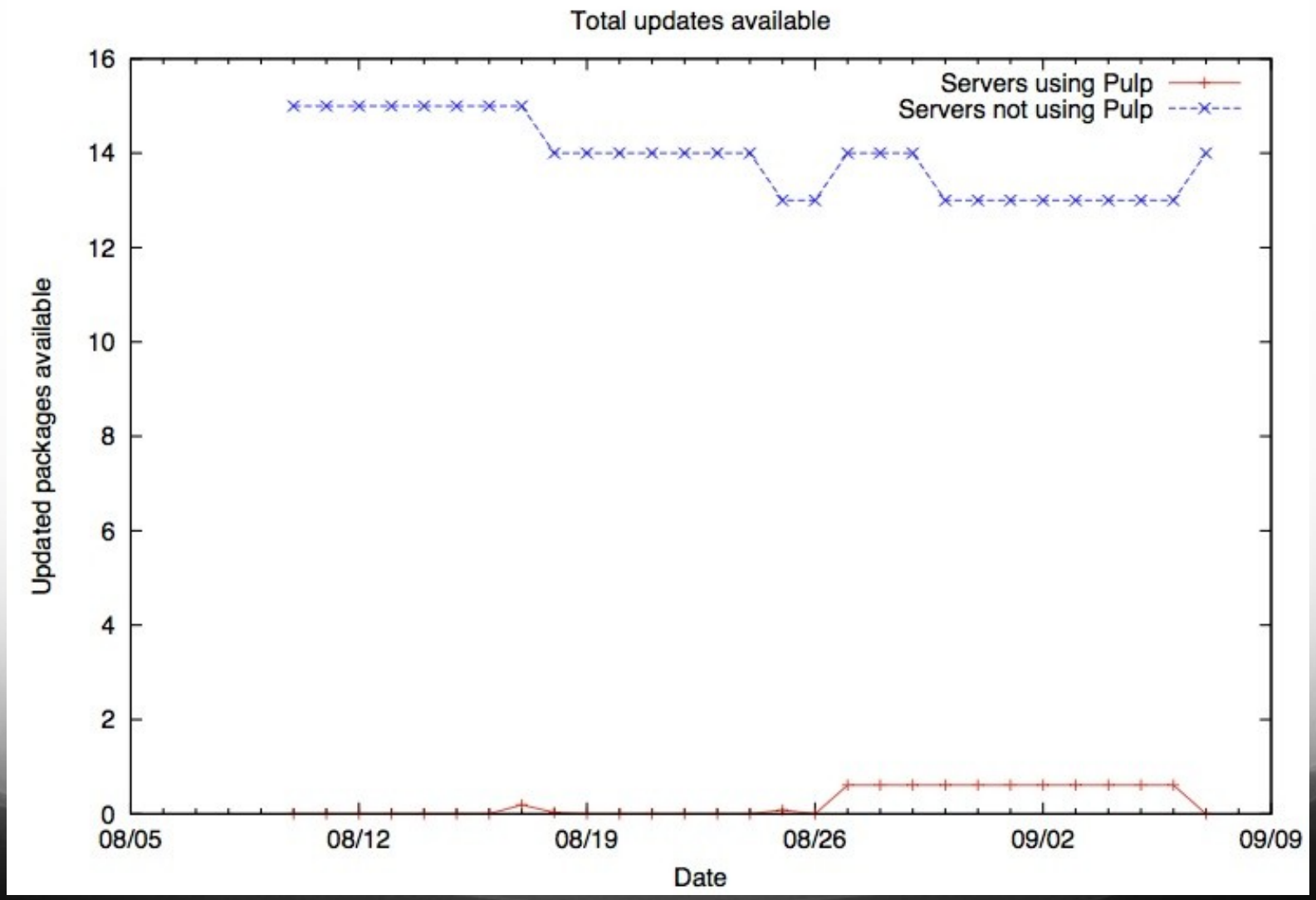
Workflow

- Patches are promoted to stable after at least a week in unstable
- Security patches receive immediate attention
- Choosing Impactful packages
  - Kernel and kernel-space
  - Impacts customers
  - Lustre and Infiniband related

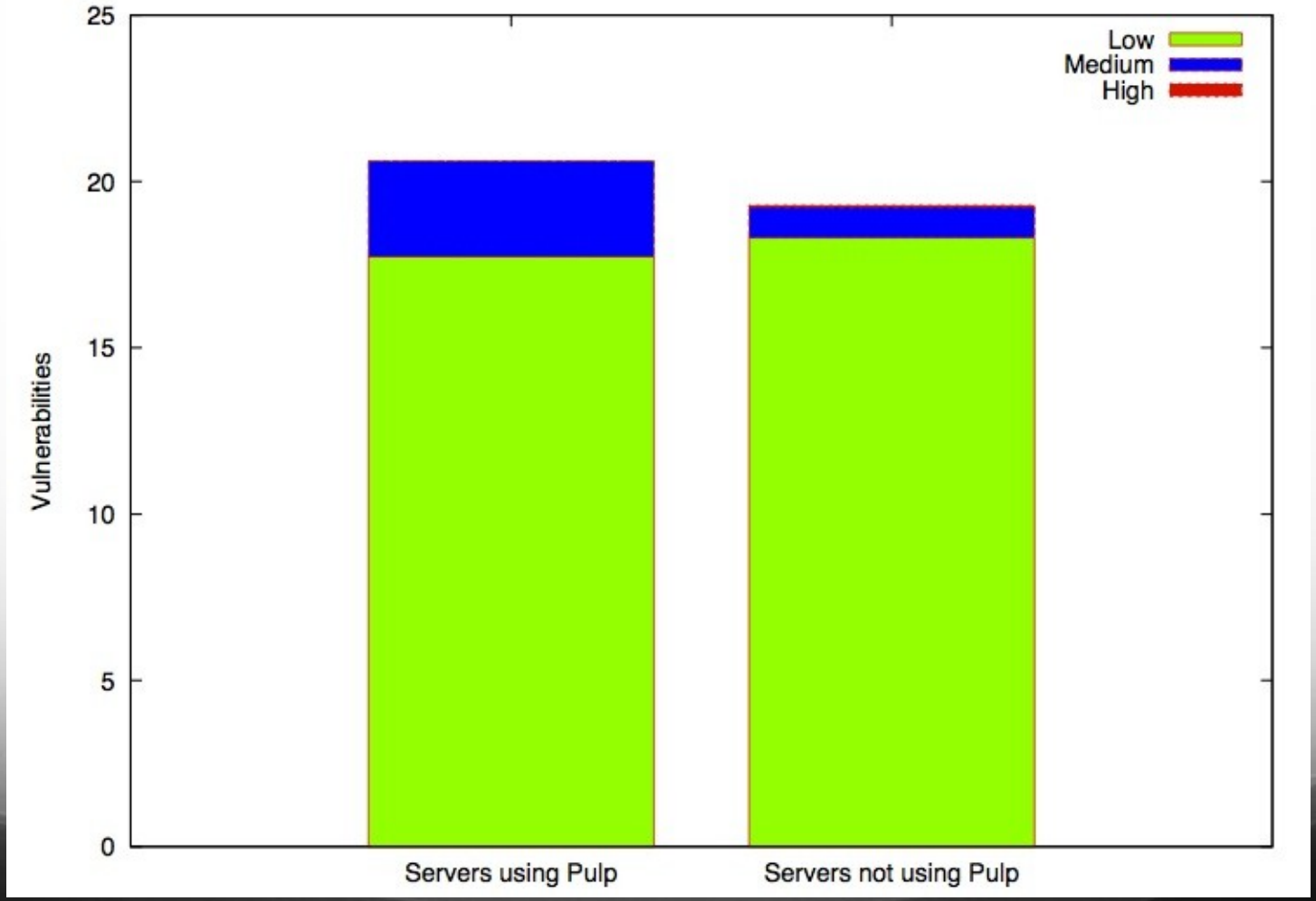
Results

- Improved automation results in less overhead
- Increased compartmentalization

Updates


Total updates available
Servers using Pulp
Servers not using Pulp
Updated packages available
16
14
12
10
8
6
4
2
0
08/05
08/12
08/19
08/26
09/02
09/09
Date

Vulnerabilities



What's next?

- Sponge
  - Web frontend for pulp
  - Django
  - More intuitive repository management
  - http://github.com/stpierre/sponge
- Apply an age attribute to individual packages
- Other packaging formats