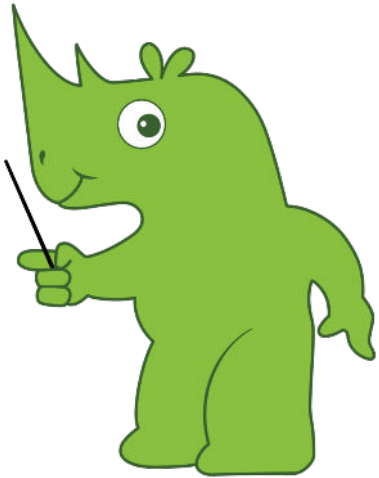
From Packaging Pulp to Using Pulp

A User Story by Markus, Maximilian, and Quirin

PulpCon 2022

2022-11-08-99a0fcfa

Why are we packaging Pulp?



Markus, Maximilian, and Quirin
ATIX AG

- orcharhino as downstream product of Foreman/Katello
- Katello relies on Pulp to manage content
- deliver backported bug fixes and features

How are we developing Pulp?

- stable branch based on upstream release
- feature branches for each change
- package for each push to GitLab

How are we packaging Pulp?

- combine upstream pulpcore_packaging (`.spec`) with downstream git repositories
- GitLab Pipeline to build RPM packages
- push packages to Pulp instance

What is our benefit using GitLab and Pulp?

- packaging-part tightly integrated with our actual code
- extend linting and testing with packaging in GitLab CI
- improved developer experience to build, package, and store packages

- simultaneously package for EL7 and EL8
- do not re-package a NEVRA
- use reference branches & 'NEVRA' to only package what's not packaged before

Why use Pulp?

- dedicated content management system with a single API
- less bandwidth required between build server and artifact store
- save disk space due to deduplicated artifacts in Pulp
- overall **better development experience** due to faster pipelines

RPM versioning pipeline design goals

- don’t package the same commit twice
- no duplicate NEVRAs
- newer code states must have a higher RPM version
- minimize manual SPEC file changes
- meaningful versions that tell us what the RPM contains

Into the weeds of our RPM versioning scheme

- Example: `pulp_deb` version `2.16.2`
- Upstream RPM package: `python38-pulp-deb-2.16.2-1.el8.noarch.rpm`

- N: `python38-pulp-deb`
- E: Empty sting implies `0`
- V: `2.16.2`
- R: `1.el8`
- A: `noarch`

- Next slide: R for Release

NEVRA: R is for Release version

- Example: `pulp_deb` version `2.16.2`

- Upstream SPEC file: `Release:1%{?dist}`
- Upstream RPM package Release: `1.el8`
- Upstream dist: `.el8`
- ATIX dist: `.2.0.atix.el8`
- ATIX RPM package Release: `1.2.0.atix.el8`
- ATIX RPM package: `python38-pulp-deb-2.16.2-1.2.0.atix.el8.noarch.rpm`

- ATIX dist scheme: `.<commit_count>.<atix_release>.atix.el<el_version>`

Demo time!

- GitLab Pipeline for new NEVRAs
- GitLab Pipeline for existing NEVRAs

Experiences with our Pulp Warehouse

- simple but versatile infrastructure
- easy to understand paths on Pulp Warehouse:
  `orcharhino/or_pulp_packaging/3-16/el7/x86_64`
- one branch in `or_pulp_packaging` results in one repository on Warehouse per EL version
- use git ref slugs instead of branch names
- switched to chunked upload API pretty soon

Pulp installer

- we used `pulp_installer` (Ansible installer)
- straightforward to use, but docs are spread out
- worked well with custom certificates
- stumbling blocks:
  - installed on Rocky Linux 8
  - some trouble getting the `ansible_fqdn` right
  - getting our HTTP proxy to play with Pulp

Talking to Pulp API

- using plain REST API requests from Python
- alternatives:
  - Pulp CLI
  - Pulp Squeezer
  - Pulp Client Bindings

How do we continue using Pulp?

- package Foreman and Katello based on `foreman_packaging`
- add `file` repositories to hold ISO images
- add `APT` repositories for orcharhino Clients for Debian and Ubuntu
- sync upstream repositories into Pulp

- GitLab CI + Pulp for artifact storage is a very powerful combination!

From Packaging Pulp to Using Pulp

- we **package** Pulp because we have to
- we **use** Pulp because we like it as a utility
- **thanks** to everyone in the Pulp community