National Synchrotron Light Source II

Brookhaven National Laboratory

U.S. DEPARTMENT OF ENERGY

On EPICS Build and Deployment

EPICS Collaboration Meeting

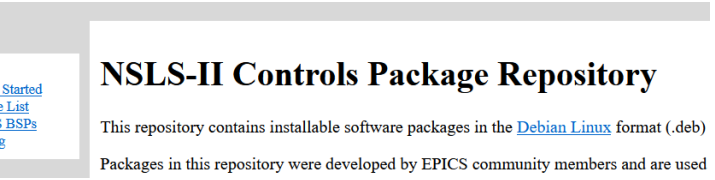
Anton A. Derbenev, Group Lead, NSLS-II DSSI

September 20, 2022

Why standard build/deployment is good?

- Code can be used (and tested!) in many places and environments
- More people can contribute to fixes and updates
- Easier general availability expands EPICS footprint
  - Easy to try if it's easy to install
  - Vendors might be more inclined to provide EPICS support
- More people using EPICS, more hardware supported = worth!

The Change of Landscape



- NSLS-II was a Debian shop since construction
- Had been building, testing, publishing packages for debianized code – collaboration effort at **github.com/epicsdeb** and internal
- Some time ago, OS standard has been changed to RHEL
- Debian repo is still up but deprecated at **epicsdeb.bnl.gov**
- A massive migration of IOCs – which required new packaging!

Some lessons learnt from Debian times

- Packaging itself needs to be maintained, it's not free
  - New upstream versions, bug fixes, package patches, structure changes
  - Thinking about and fixing compatibility, dependencies, [architectures]
  - Deceptively easy to do it (wrong), but best practices are not cheap
- Building, testing, publishing packages (from debianized code) also takes effort
  - Need infrastructure and process to do it all, better be pipeline
  - Availability of "production testing grounds" is what turns a set of "works for me" software into an easily adoptable product
  - It's quite a learning curve "from zero to maintainer", but a well-defined and robust process can compensate for that
- It's not just the coding fun
  - Collaborative work implies engagement(e.g., review a PR on debianized code, publish new version)
  - Implicit responsibility to product users (e.g., make sure mirror is up, keys are good)
  - "Formal" considerations like security, licensing, mode of sharing/use

The RHEL experience

- How to address a facility-wide need without incurring too much debt on the way?
  - Take note of overall prior experience, carry on with what worked well
  - Make necessary compromises along the way
- It's surely good to have packaging for EPICS, taking advantage of everything packaging provides
  - Availability (just yum install), standard install location and contents, versioning, centralized distribution
- Employ various powerful tools to automate and expand the reach – Jenkins, Ansible/Satellite
- Ended up using some prior work (EPICS base and modules build tool) to create a mega-package

From .deb to .rpm

- Debian packaging approach was: follow strict practices, tight about signing and versioning, base and modules are split, multiple architectures, manage dependency trees
- RHEL packaging approach was: make it work good fast easy
- Use e.g. **Jenkins** automate build and package:
  - **A specialized in-house build tool** pulls, configures, builds, restructures all required modules and base using customizable build configuration (heavy lifting is done here)
  - **rpmbuild** produces a single .srpm from a relatively unsophisticated .spec (artifacts are pre-structured)
  - Packaging pipeline builds and publishes to the **internal repo** (with basic versioning)
- Something like **RH Tower** to handle install and update:
  - Centralized, **automated install** on relevant machines (IOC servers, CA client workstations)
  - The package comes from internal repo, all installs go in **/usr/lib/epics**
  - From IOC perspective, very similar to how it was done on Debian (none or minor reconfiguration needed)

Wanted Looking Forward

- More flexible customization and versioning of the package, consider split in a collection of base/module/driver packages
- Use GH Actions, do more advanced testing (e.g., with a “virtual beamline”)
- Externally accessible repo – perhaps from **epicsrpm** collaboration effort?
  - Licensing!
- Automated deployment pipeline for IOCs