Future of the software packaging

Tomáš Chvátal
tchvatal@suse.com
SUSE/L3 - Packaging

2018/10/06

Introduction

Who

Who the hell is this sod presenting here?

- SUSE employee working as a teamlead of Packaging team
- One of the people that created Tumbleweed rolling release distro
- Formerly Gentoo developer and Council member

What

- First we will dig up some brief history of the packaging
- Then we will check up on why do we even bother with the work
- And at the end we will make fun of everyone else, or should we?

Why bother

What actually is packaging

- Postal services, software? It does not matter.
- The core goal is to get something that wrapped from point A to B
- If we focus on offline Amazon usecase the software does all the steps:
  - Get request for delivery of goods
  - Gather the goods and put them to box
  - Send the box to delivery center (here we diverge with reality as we can deliver 1 resource endlessly)
  - Send the delivery to destination
  - Customer gets the goods and can enjoy his new bath duck

Packages on Linux

- Most distributions are leveraging some package/software management
  - Debian/Ubuntu (apt + dpkg)
  - Arch linux (pacman)
  - Gentoo linux (portage)
  - rpm based distributions (rpm + zypper/yum/dnf)
- The same applies for phones with Android/iOS (F-Droid, etc.)
- Windows provide the application store in Windows 10 and newer

Languages

Many popular languages have their own package system/registry

- CPAN (perl)
- pypi (python)
- cabal (haskell)
- npm (JS)
- ...

Why do we need packages at all?

- We need to be able to deliver software to users
- We need to isolate required components
- We need to ensure proper testing of the components
- We need to compile all various software stacks together
- We need to provide comprehensive solutions for some tasks
  - Dependencies
  - Updating and migration
  - User management
  - Post-installation configuration (first config, no wizzards)

What are packages providing to the user

- Collection of files and their respective permissions
- Metadata containing information about the software, runtime dependencies
- Something that can be verified (vendor, signature, CVE inclusion, etc.)
- Initial configuration provider
- Version migration management

Bit of history; openSUSE POV

Slackware

- Slackware was the first implementation of packaging overall
- The monolithic installation image was split to particular sets of files that were separately installable
- The Slacware packages are plain compressed tar archives a simple installation script is the only added feature
- The packaging tool just unpacks these files into the correct location
- The packages were compiled directly on the package maintainer system

Slackware continued

- Slackware has no concept of dependencies which asks for trouble:
    - Program can fail due to a missing dependency
    - Compilation can result in feature-restricted version due to a missing optional dependency
    - Program can be compiled with extra feature that was supposed to be avoided

Move to RPM

- RPM was a big step forward
- A compressed archive, but with metadata and dependencies
- Provides package description format - specfile
- But introduces a new problem: RPM is able to report dependencies, but it is not able to evaluate and install them
- Packages are organized into repositories, and a front-end resolves them

Autobuild

- Raising number of packages require CPU power
- Packages must be rebuilt in various cases due to changes
- It is hard to determine which packages need a rebuild
- First automated build system Autobuild evaluated and scheduled all the builds
- Packages were no more built inside a live system! chroot FTW

zypper/libsolv

- Rising number of packages introduced a new major problem
- Evaluation of package dependencies in the YaST installer became very slow
- Evaluation of the dependencies also required a lot of RAM
- Several SUSE employees were not happy with that, and they revived their mathematical occupation, and invented a new dependency resolver libsolv (now used as backend for both dnf and zypper)
- Prior to zypper SUSE also tried ZENworks Management Daemon (zmd) coded in C#

OBS

- Autobuild was a large monolithic tool that could not be scaled up easily
- openSUSE Build Service was designed to expand the autobuild concept:
  - Create many repositories for many products
  - Allow user access and provide ACL controls
  - Improve scheduler to utilize resources better
  - Provide solver to better figure out rebuilds
  - Better separation of builds (VMs)
  - Reproducible builds (no Tumbleweed otherwise)
- Also we got bored by doing it all on our own and provide API and UI to allow external contributors
- Nowadays, all SUSE Linux products are built inside OBS

openQA

- Newest friend on the SUSE packaging block ensuring quality of the stuff we produce
- With Tumbleweed no mere man could test all that mess 3-5x per week
- Machine pretending to be user that writes and clicks on stuff
- Compares partial screenshots and serial output
- Tests also weird stuff like s390x

Future?

RPM plans

Well I can’t say much for Debian as I am not involved there :-)

- Strictification/unification in tests/reviews
- Boolean dependencies (read as conditional dependencies)
- Internal speedups
- Linter improvements (rpmlint)

Flatsnaps

What motivation generally is behind this

- Create one package for all distributions
- Provide latest software version for OLD distribution
- Avoid problems with distribution dependencies
- Isolation of applications
- Allow multiple versions of installed aplications
- Consistent environment for the application

Flatpak overview

- Developed by the freedesktop.org project
- Can use libraries from other Flatpaks
- Provides sandboxing
- flathub.org repository with apps

Snap overview

- Developed by Canonical (Ubuntu)
- .snap is filesystem snapshot (squashfs)
- Sandbox using AppArmor
- Apps need to bundle all the libraries they use
- Ubuntu-centric

Appimage overview

- Image mounted via FUSE
- One file per application
- Formerly klik/PortableLinuxApps
- No sandboxing - but can utilize firejail
- Really nicely integrated with OBS

Sounds okay, where is the catch

- Ever heard of the "DLL hell"?
- Offloading security auditing to extra provider (who updates the packs)
- In long time we might end up with many "runtimes"(Visual C++ Redistributable analogy)
- Appstores will need to be audited a lot (like distribution is)
- Maintainers sometimes protect you from some crazy upstream ideas

Containers

Why bother with containers

- Really really really blazing fast to deploy something
- Awesome for various CI integration
- Small startup image and lower resource usage
- Blazing fast startup (comparing to virtual machines)

What can bite your arse

- Many instances of libraries to patch
- People must use dockerhub or similar and update their containers
- The initial containers must be trusted (shall not download randomly from the web)
- Updates mean redeploying all the containers

Endnote

RECAP

- Distribution packages are important for the user
- It is not always best idea to just download something to your system from the web
- Containers can be very useful for quick deployment but it is never fire and forget

Should you devote some of your time?

- Hell yes, get involved (regardless of the distribution)
- There are always some bugs to be fixed in packages that you personaly use
- Remember to have fun!

Thanks/Questions

Thank you for your attention.
Are there any questions?