Software Stack Provisioning for HPC

Build  Install  Configure  Run

Michael Mercier  michael.mercier@inria.net
GriCAD Workshop
Why Can't We Install Softwares?

Problem statement

**Shared Resources, Shared State**
- High Performance Computers (HPC) are shared resources
- Each HPC machine is a shared software environment

**HPC users requirements**
- Need to bring there own software
- They want reproducibility of results

Package Managers are modifying a shared state

If users have permissions to install:
- **side-effects** (See Alice and Bob story)
- Drift of the environment: Operator don’t know what is running
Alice and Bob Story
The Update problem
Classical Package Manager (YUM, APT, …)
Or Why Using Shared Libraries

Heavily use shared libraries \(\iff\) Shared state

Why Shared libraries?
- Loading libraries once in memory
- Storing libraries once on disk
- Can update without re-compilation

**Static libraries: glibc in Debian**
- 37% of the package depends of \texttt{glibc} in Debian Stable
- 19579 packages out of 51831
- 3 updates in Debian Stable \((\approx 1\text{ year})\)

\(\Rightarrow\) 58737 to recompile instead of 3
Software Provisioning Process

Build  Install  Configure  Run
Build
Requirements and Constraints

Reproducibility
- Bitwise reproducibility of the binaries
- Logging of the build process with all parameters
- Changing any build parameters ⇒ explicit new version

Combinatorial explosion of parameters
- Architecture (x86_64, ARM64, …)
- Platform (Linux, Cray, …)
- Compilers (GCC-6, GCC-7, Clang, Intel, …)
- MPI libraries (OpenMPI-1, OpenMPI-2, mvapich, IntelMPI, …)
- Other libraries (OpenBLAS, OpenMP, …)
- Compilation flags (-debug, -O, …)
Install
Requirements and Constraints

**For Users**
- Have permission to install!
- From source (build)
- From binary (provided by any tiers)
- Have multiple version of the same software
- Have distinct environments for development/test/run

**For Operators**
- Provides easy to install packages for users
- Possibility to push (security) updates to users
- Save resources by sharing libraries between users
- Keep track of what is installed and where
- Possibility to rollback to previous version if needed
Configure
Requirements and Constraints

Portability
- no pointers to user’s home :)

Traceability
- Every configuration versions are kept
- Sharing software with configuration attached automatically

Ease of use
- Operator provides sane defaults depending on the platform
- Users are able to change configuration of software AND operating system
- Possibility to have multiple configuration for the same software (and switch easily)
Run
Requirements and Constraints

The goal of provisioning
The final step that depends on the others

Avoid divergence between build time and run time
- Be sure to link to shared libraries used for build
- Or ABI-compatible ones with security patch

Traceability
- Have warranties that the right software is running
- Be sure that the right configuration is used
Containers don't solve the problem
Or the Unwanted Isolation

Containers Resource ans Job Management Systems (OAR, Slurm, …) already provides resources isolation. Other types of isolations are not wanted: Mount: prevent to load and share files (where is my CSV?) special hardware support is problematic (need drivers and libraries on the host) Process ID: prevent to find other local process Net: bad network performances Inter Process Communication: No shared memory between process User ID: Give you what you have if you were not isolated A good solution for services: ensure a configuration is used (using entrypoints)
Spack and EasyBuild

Spack **Build Install Configure Run** Patch binaries to force the RPATH
Nix and Guix

TODO
Conclusion

TODO
Thanks!

Questions?

Michael Mercier  michael.mercier@inria.net
GriCAD Workshop