Deploying Community Codes

Prasad Maddumage
mhemantha@fsu.edu

Research Computing Center
Florida State University
July 31, 2017
Overview

- What are community packages?
- Who installs what?
- How to compile and install?
  - Dependency hell
- Setup at FSU RCC
  - Where to install?
- Using RPMs vs regular install
  - Getting RPMs
  - How to build an RPM
- Automated package building
  - EasyBuild
  - Spack
What are community packages?

- **Libraries**
  - Scalapack (linear algebra routines)
  - SuperLU (solving sparse matrices)

- **Languages**
  - Python (2 and 3)
  - R (several versions per year)
  - Julia (relatively new yet powerful)

- **Software packages**
  - LAMMPS (molecular dynamics simulation)
  - TopHat (RNA sequencing)
Who Installs What?

● Two policies
  – Administrators install the basics and users install packages on their home directories
    • Cluster maintenance is relatively simple
    • User support could become complicated
  – Support staff install packages system-wide for users
    • Cluster upgrades and maintenance is complicated
    • Eliminates most package install and version related issues
How to compile and install?

- `configure/cmake, make, make install`
  - Most packages install this way
  - Best if only had to do once
  - `cmake` offers many configuration options
  - May need lot of researching (Google) to find best options

- Binaries from the developer
  - No need to compile
  - Library version incompatibilities (eg: `boost`)
  - Only use if source is not available

- Use RPMs (on RHEL and CentOS)
Dependency hell

- gtools
- bitops
- gdata
- caTools
- numDeriv
- gplots
- SparseM
- mvtnorm
- quantreg
- matrixcalc
gLMM
Dependency hell
FSU RCC manages 550 custom packages
- 171 R packages (mostly bioconductor)
- Only install basic Python packages and Python 3
  - Users can install Python packages in their home directories via `virtualenv` (`pip` installs dependencies automatically)

All packages are installed via RPMs
- Few exceptions for very large packages installed on parallel file system (e.g., `orca`)

Only support the packages we install
- `WRF` is widely used but managed by users and we respond to support requests
Where to install?

- Package Request
  - Group of Users
  - Single User
    - Install size too big?
      - yes
        - Shared file system
      - no
        - Local disk /usr, /opt etc.
    - User's home directory
Where to install?
Using RPMs vs regular install

• Pros
  – No need to figure out how to install a package if a pre-built RPM exists
  – Self documenting
  – Easy file lookup (using `yum provides ...`)
  – Easy up/downgrade to different versions (using `yum`)
  – Clean uninstall

• Cons
  – Need local disks on every node

• Best practices
  – Local repo for custom built RPMs
  – Minimal (clean) system for building RPMs
Getting RPMs

- **Multiple sources**
  - Public repos (EPEL, fedora, rpmfusion, …)
  - Some packages offer RPMs (eg: LAMMPS)
  - Customize an existing source RPM

- **Create a custom RPM**
  - Get the source
  - Find the installation instructions
  - Create a spec file
  - Use `rpmbuild` to create the RPM
  - Resulting source RPM contains the `.spec` file and all the source files
  - The RPM(s) preserve the install directory structure
How to build an RPM

- Need the package source and .spec file

Name: R2spec
Version: 4.2.1
Release: 11{%?dist%
Summary: Python script to generate R spec file

Group: Development/Languages
License: GPLv3+
URL: https://fedorahosted.org/r2spec/
Source0: https://fedorahosted.org/releases/r/2/r2spec/R2spec-%{version}.tar.gz
BuildRoot: %{_tmppath}/%{name}-%{version}-%{release}-root-%{__id_u} -n

Requires: R python-jinja2 wget fedora-packager
Requires: python >= python-2.6 python-argparse >= python-argparse-1.2.1
Provides: R2rpm >= 1.0.0

%description
R2spec is a small python tool that generates spec file for R libraries.
How to build an RPM

%prep
%setup -q

%build
%{__python} setup.py build
sed -i '1i %define Rver 3.4.0' r2spec/specfile.tpl
sed -i '2i %define _prefix \'/opt/hpc/R/R-%{Rver}' r2spec/specfile.tpl
sed -i '3i %define distnum %%(\'/usr/lib/rpm/redhat\'/dist.sh --distnum)' r2spec/specfile.tpl
sed -i 's|%%{?dist}.*|%%{?dist}%%{distnum}3|' r2spec/specfile.tpl
sed -i 's|Name:             R-%%{packname}|Name:             R-%{Rver}%{packname}|' r2spec/specfile.tpl
sed -i '44i module purge;module load R/%{Rver}' r2spec/specfile.tpl
sed -i '62,70d' r2spec/specfile.tpl
sed -i '62i %{rlibdir}/%%{packname}/*' r2spec/specfile.tpl

%install
rm -rf %{buildroot}
%{__python} setup.py install --root=%{buildroot}
install r2spec/specfile.tpl %{buildroot}/%{python_sitelib}/r2spec/
chmod -x %{buildroot}/%{python_sitelib}/r2spec/specfile.tpl

%clean
rm -rf %{buildroot}

%files
%defattr(-,root,root,-)
%doc README LICENSE CHANGELOG
%{python_sitelib}/*
%config(noreplace) %{sysconfdir}/%{name}/repos.cfg
%{_bindir}/%{name}
%{_bindir}/R2rpm
%{_mandir}/man1/%{name}.1.gz
%{_mandir}/man1/R2rpm.1.gz
Automated package building

- Dependencies make package building very tedious
- Fedora uses Koji RPM build system
  - [https://pagure.io/koji](https://pagure.io/koji)
  - Used by CERN, Caltech, and, Amazon etc.
  - Very complicated and less flexible
- RPM building process can be scripted in many cases
  - R package RPM creation was completely automated
    - Recursively download all dependencies
    - R2spec package was used to create spec files for RPMs
  - General RPM creation at FSU RCC was mostly automated
    - Package source locations had to be manually supplied
EasyBuild

- Automatic build and installation of (scientific) programs
- Flexible and configurable (build recipes)
- Automatic dependency resolution
- Module file generation, logging, archiving
- Good documentation, increasing community acceptance
- Relatively simple to set up and use when using defaults
- Due to its flexibility, more complicated to customize
- Best deployed as a fresh build-out
Spack

- Package management tool designed to support multiple versions and configurations of software
- Designed for large HPC clusters
- Automatic installation of scientific packages through prebuilt recipes
- Strong CLI support
- Different versions of packages can coexist
- Easy to integrate with existing systems
- Module files are auto generated (Tcl and LMOD)
Demo