Deeper Understanding of Software Collections

955508

Ryan Hennessy
Solutions Architect, Red Hat
hennessy@redhat.com

Agenda

- Software Collections
- Red Hat Software Collections and Red Hat Developers Toolset
- Using Software Collections
- Basic building blocks for building your software collections

The needs of the System Engineer....

- RHEL software packages are designed for stability and long life cycles
- There is a need for updated software packages that can follow defined installation/patching mechanisms
- Provide for multiple version of the same software package on a single system

Go from this...

To this...

Image Source: GL Stock Images

Software Collections

- Allow for multiple versions of the same software to be installed on the system
- Does not override the RHEL requirements for specific version of software
- Packaged in RPM
- Installed in a standardized path
- Easy set of commands to interact and use installed software

Red Hat Software Collections and Developer Toolset

- Built with the software collection tool set
- Packages built and supported by Red Hat
- Red Hat Developer Toolset focused on system type software development and debugging
- Red Hat Software Collections (RHSC) provides recent versions of dynamic programming languages, database servers, and various related packages

Red Hat Software Collections Life Cycle

- Important bug and security fixes are supplied in same manner as RHEL errata
- Major version has three year life cycle of support
- New major version is released approximately every 18 months
- New components in RHSC have backward compatibility with the components in the previous major version of RHSC
- Available on supported 64-bit versions of RHEL 6

Red Hat Developer Toolset Life Cycle

- Important bug and security fixes are supplied in same manner as RHEL errata
- Major version has two year life cycle of support
- New major version is released annually
- Packages built/compiled on a particular version of RHEL can be run on RHEL n and RHEL n+1
- Available on supported versions of RHEL 5 and RHEL 6

Let's get down and dirty...

- Identifying software collections that are installed
- Enabling a software collection
- Running applications with a software collection

Before we do anything...

Install the package needed to invoke software collections

```
[root@testserver bin]# rpm -ql scl-utils
/etc/bash_completion.d/scl.bash
/etc/scl/prefixes
/opt/rh
/usr/bin/scl
/usr/bin/scl_enabled
/usr/share/man/man1/scl.1.gz
[root@testserver bin]#
```

Identifying the Software Collections Installed

Run the 'scl' command to list all installed software collections

```
[root@server ~]# scl --list
devtoolset-2
mysql55
perl516
php54
python33
[root@server ~]#
```

Enabling a Software Collection

After enabling the software collection you can see that the version of the python interpreter is different

```
[root@server ~]# python --version
Python 2.6.6
[root@server ~]#
[root@server ~]# scl enable python33 bash
[root@server ~]#
[root@server ~]#
[root@server ~]# python --version
Python 3.3.2
[root@server ~]#
```

Enabling a Software Collection

- Enable multiple software collections at once by providing each collection on the same enable command
- Environmental variable 'X_SCLS' can be used to determine which collections are currently enabled

```
[root@server ~]# scl enable python33 mysql55 bash
[root@server ~]# echo $X_SCLS
python33 mysql55
[root@server ~]#
```

Enabling a Software Collection

- Software Collection services are enabled the same way as any other system service

```
[root@server ~]# service mysql55-mysqld start
Starting mysql55-mysqld:                                   [  OK  ]
[root@server ~]# chkconfig mysql55-mysqld on
[root@server ~]#
```

Running an application using a Software Collection

Simple python script: (Don't worry if you don't know python)

```
def outer():
        x = 1
        print ("Pre innner call: ", x)
        def inner():
            nonlocal x
            x = 2
            print("inner:", x)
        inner()
        print("outer:", x)

if __name__ ==  "__main__":
         outer()
```

Let's run the script

Using the version of python installed with RHEL

```
[root@server ~]# python pythontest.py
  File "pythontest.py", line 5
    nonlocal x
             ^
SyntaxError: invalid syntax
```

Let's try running that script again

This time run the script with the python 3.3 software collection

```
[root@server ~]# scl enable python33 "python pythontest.py"
Pre innner call:  1
inner: 2
outer: 2
```

Now let's take a look under the hood..

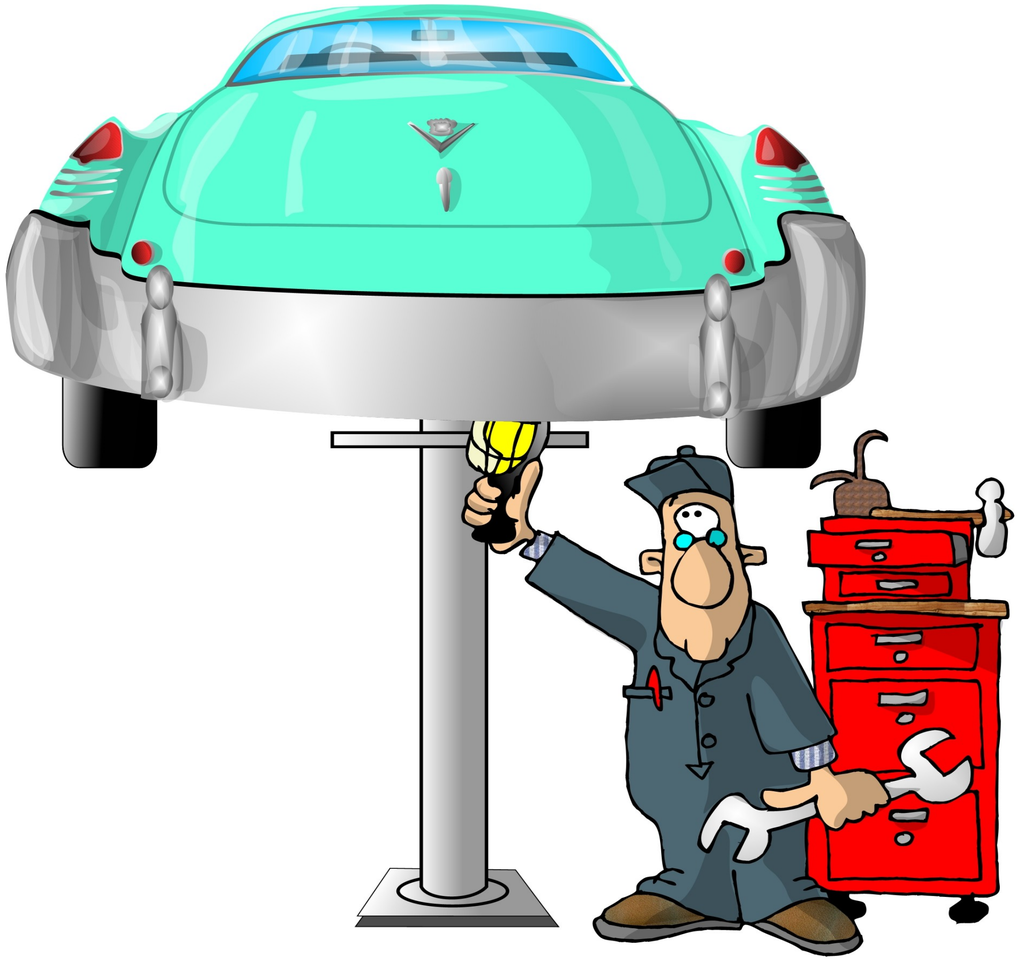
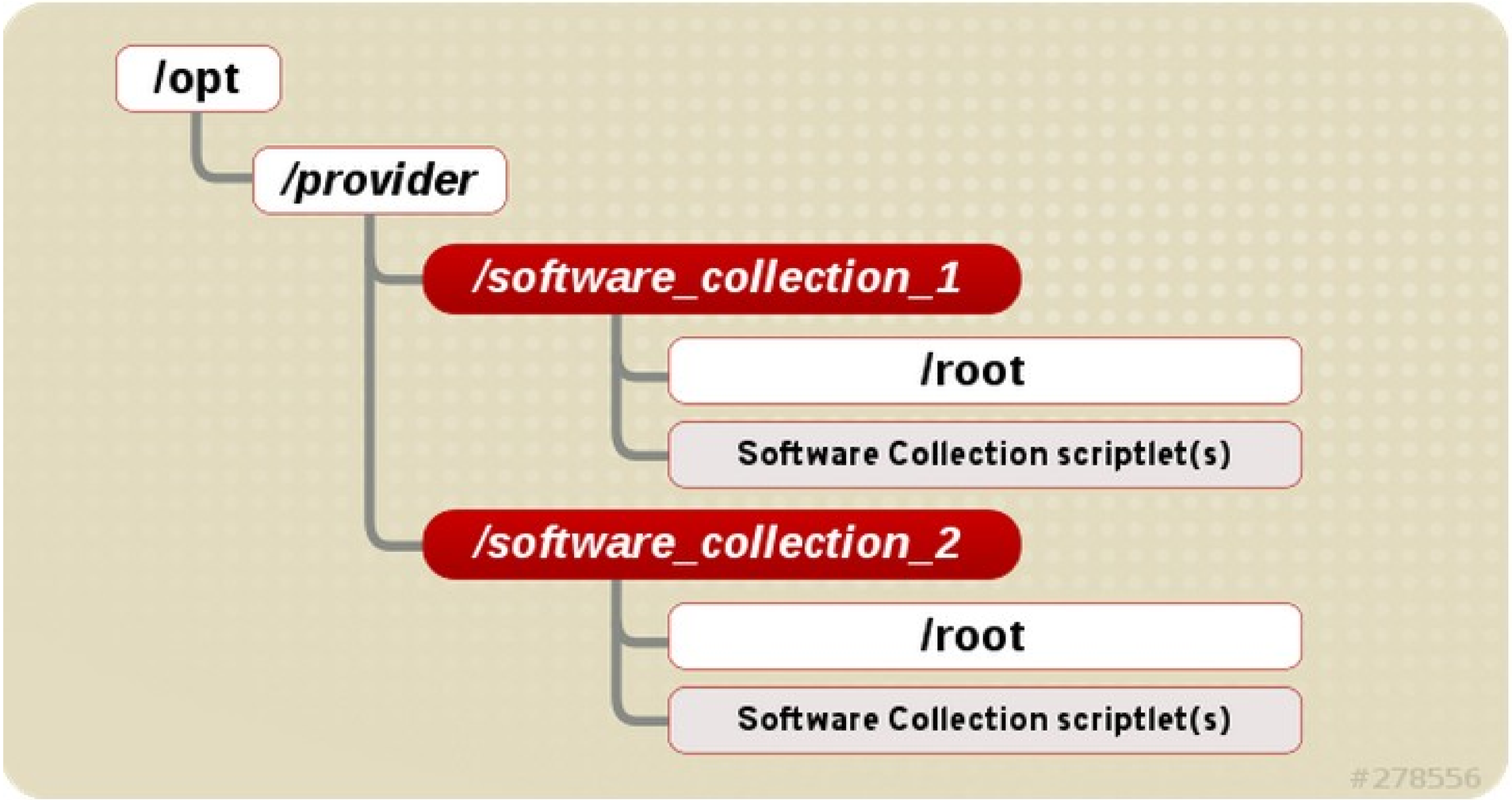


Image Source: MichiganToday.net

The Building Blocks of a Software Collection



The Building Blocks of a Software Collection

/etc/scl/prefixes

Configuration directory 'scl' command uses to determine the software collection file system

```
[root@server prefixes]# pwd
/etc/scl/prefixes
[root@server prefixes]# cat python33
/opt/rh
[root@server prefixes]#
```

The Building Blocks of a Software Collection

/opt/<provider>/<software collection>/enable

The environmental variables that are modified when a software collection is enabled

```
[root@server python33]# pwd
/opt/rh/python33
[root@server python33]# cat enable
export PATH=/opt/rh/python33/root/usr/bin${PATH:+:${PATH}}
export LD_LIBRARY_PATH=/opt/rh/python33/root/usr/lib64${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH}}
export MANPATH=/opt/rh/python33/root/usr/share/man:${MANPATH}
# For systemtap
export XDG_DATA_DIRS=/opt/rh/python33/root/usr/share${XDG_DATA_DIRS:+:${XDG_DATA_DIRS}}
# For pkg-config
export PKG_CONFIG_PATH=/opt/rh/python33/root/usr/lib64/pkgconfig${PKG_CONFIG_PATH:+:${PKG_CONFIG_PATH}}
```

The Building Blocks of a Software Collection

/opt/<provider>/<software collection>/root

complete file system layout containing all the files of the software collection

```
[root@server root]# pwd
/opt/rh/python33/root
[root@server root]# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  sbin  selinux  srv  sys  tmp  usr  var
[root@server root]#
```

Let's package it up..

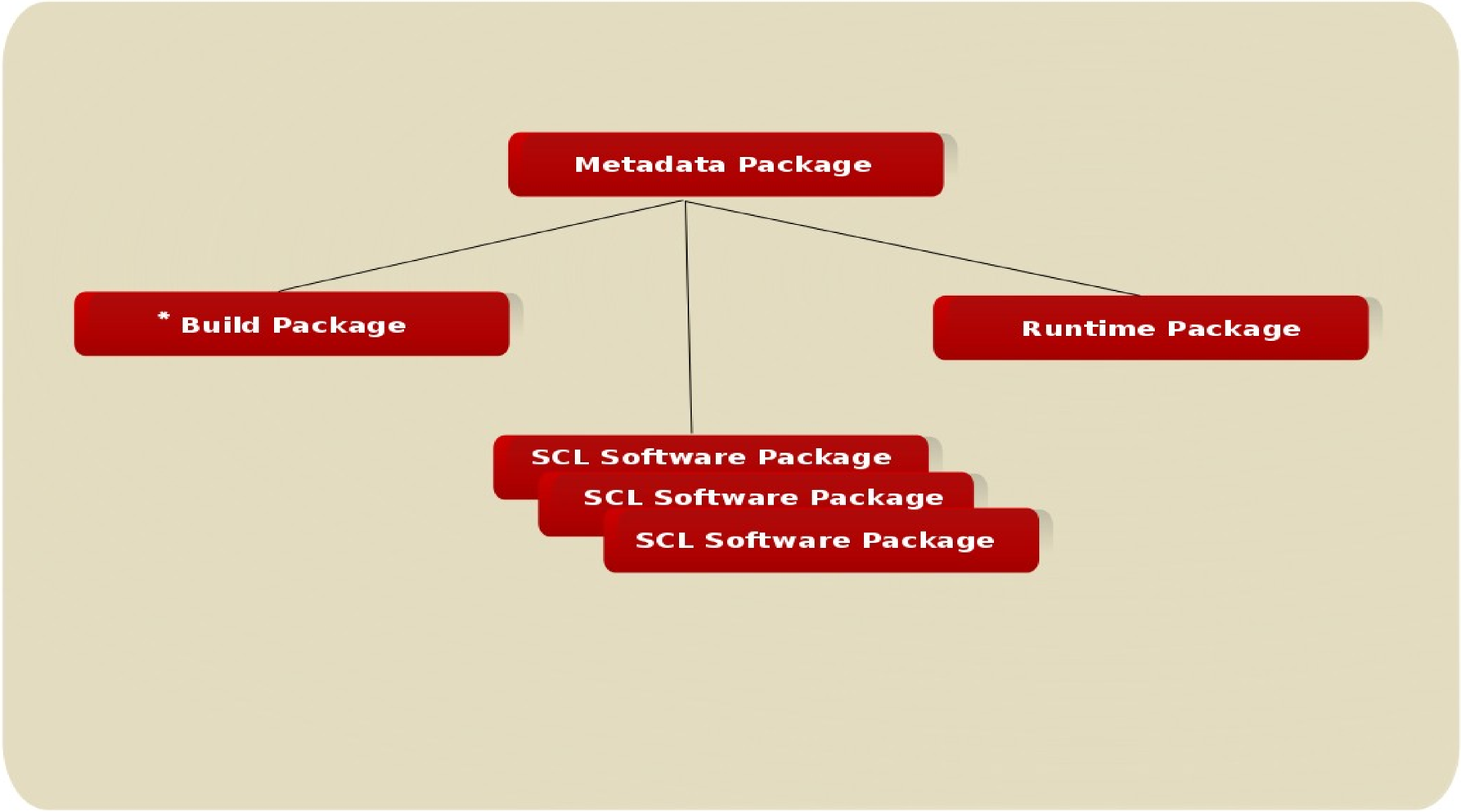
Why to build your own package

- Adding functionality to an existing software package
  - Library for python
- Copying a software collection package
  - Adding onto a software collection package that is provided by Red Hat
- Creating a software collection from scratch
  - Version control of your own software

What is with these extra packages

```
[root@server home]# rpm -qa | grep php54
php54-1-7.el6.x86_64
php54-php-common-5.4.16-7.el6.1.x86_64
php54-runtime-1-7.el6.x86_64
php54-php-xml-5.4.16-7.el6.1.x86_64
php54-php-process-5.4.16-7.el6.1.x86_64
php54-php-cli-5.4.16-7.el6.1.x86_64
php54-php-pear-1.9.4-7.el6.noarch
```

Parts of the software collections package



How to

- Install the scl-utils-build

- Convert your package by hand
  - Create the metadata spec file
  - Modify the software package(s) spec file

- Use the spec2scl tool
  - Create the metadata spec file
  - Use to the spec2scl tool to update the software package spec file

Creating the Software Collection package

- Follows the same rpm building process
  - Build the software packages
  - Build the metadata package
- Macros to be aware of
  - scl
  - _scl_prefix
- Example
  - rpmbuild -ba mypack.spec -- define 'scl mypack' --define '_scl_prefix /opt/henn'

Summary

- Software Collections tools are great for adding new software functionality as well as version control
- Red Hat's software collections uses:
  - Red Hat Software Collections = Updated runtimes/application
  - Red Hat Developer Tool Set = updated gcc and debugging tools
- Ease of use for existing software collections
- Building software packages
  - 3 RPM's for basic software collection functionality
  - Use documentation on the portal for spec file changes

Useful Information

- www.redhat.com/developers/rhel
- Many how-to's at developerblog.redhat.com
- The doc I finally read: http://red.ht/1ihMfTj
- spec2scl information: https://bitbucket.org/bkabrda/spec2scl/
- www.softwarecollections.org
- CORPs: http://developerblog.redhat.com/2014/03/11/intro-coprs