SCOTT GOSE

Grid Packaging Software

Today's Grid environments are constructed from many software layers. From the operating system to middleware to applications, all this software needs to be packaged so that it can be distributed and used by people on the Grid. This column examines fundamental tools for packaging and higher-level utilities that can simplify the deployment of Grid software.

Package Formats

As the first thing users deal with, package formatting can have a tremendous effect on the success of your software. This section describes three popular formats and weighs their advantages and disadvantages.

GPT

Grid Packaging Tools (GPT) consist of utilities for building and packaging software. GPT was designed by the National Center for Supercomputing Applications (NCSA) as a cross-platform, location-independent method that allows dynamic or static libraries and source or binary packaging.

The information defining each GPT package is stored in an XML metadata file that defines such things as the package name, version, and any dependencies. With GPT, binary and source packages can be created from this file. But there is much more to GPT than metadata and tools to build and package the software.

The common practice used with GPT is to create a larger number of smaller packages, rather than a smaller number of larger packages. The benefit of this architecture is modularity. If you need to release an update to one package, you can do so relatively easily by releasing just the small, updated package (in most cases) and not all the other packages that make up a particular software release.

Having multiple GPT packages per software release can be cumbersome to manage, however. To help with this, GPT introduces the notion of a "bundle." A bundle is a collection of GPT packages that have been tarred and compressed (.tar.gz), making them easy to move. Moreover, because these bundles can be built and installed directly with GPT tools, they can be the primary method by which the software is released.

Recall that GPT packages come in both source and binary form. When source packages are built, they produce different types of binary packages. A GPT source package consists of one I<tarball>, in .tar.gz format. However, when the GPT source package is built using "gpt-build," it can produce up to six different binary packages, depending on what is contained in the source package.

The GPT binary package types possible are shown in *Table One*.

With its bundling capability, GPT easily manages the possibility of a 6:1 source-to-binary package ratio by preventing the user from seeing the increased number of packages. And because most GPT software is released in bundled form, users don't have to deal with packages at all. Only when updates to a particular package are released is the user exposed to a single GPT package.

GPT is written in perl and therefore usable on most platforms. Its main drawback is that, as a relatively new packaging tool that is still evolving, GPT has not yet been included as standard software on many operating systems. While requiring users to install GPT before they install the packaged Grid middleware is not optimal, the slight complexity added by this step can be mitigated with fairly simple build and install scripts.

RPM

RPM (for Red Hat Package Manager) is a format that dates to the beginning of Red Hat Linux. Back in the early days, Slackware was the most popular Linux distribution, and .tar.gz's were the standard way to manage packages. However, with its added convenience, RPM helped vault Red Hat to the most popular Linux distribution.

Despite its origins in the Linux environment, RPM offers the Grid

TABLE ONE

GPT Binary Package Types

TYPE	CONTENTS
`pgm`	Dynamically linked executables
`pgm_static`	Statically linked executables
`rtl`	Runtime libraries
`data`	Data files
`dev`	Header files
`doc`	Documentation

community a powerful package manager that can be used to deploy and manage software efficiently — something the Grid needs.

RPM does have some drawbacks. For example, GPT allows you to easily relocate binary packages. RPM has this ability, too, but it isn't required by an RPM creator to support when building a package. This can make relocating a binary by the user nearly impossible. In addition, non-root installs of RPMs are tricky, whereas in GPT they are trivial.

But RPM also offers several advantages. First, it is commonplace. A majority of the widespread Linux distributions today use RPM as their native package format. With standards gradually being accepted for commonplace file locations, RPM's incompatibilities across multiple Linux distributions will continue to converge. The goal is to have a single RPM that can be installed on any RPM-based system. This would be a tremendously powerful asset if it became a reality.

Second, RPM is stable. Since it has been around for several years and is so widely used, it has been tested considerably. The result is a reliable and robust abstraction layer for managing software on a system. It is difficult (and frustrating) to develop software using a build and packaging tool that itself is under development. With the common, refined base RPM provides, packaging software almost becomes a non-issue.

Tar and Zip

In the world of multiple architectures, various operating systems, and splintered Linux distributions, packaging can be downright daunting. GPT tries to address this problem by being a universal packager, and in turn, becomes another requirement for your software.

One tried-and-true method of package distribution is to distribute software simply tarred or zipped into a single file. Upon opening the package, the user is presented with an INSTALL file that details the installation process for the software. The install process is usually carried out by a "./configure," "make," "make install" process. If this is the case, however, note that the tarred or zipped package itself can likewise be built into a GPT or RPM package.

This type of package distribution is universal in that it can be handled on any type of Unix where the packaged software will build. The downside is that it doesn't provide any of the package management features of GPT or RPM. Although this seems like a significant drawback, the .tar.gz distribution is almost always needed because it not only forms the basis for creating GPT or RPM, but it also is the generic way to build and install software on any Unix system. Software releases generally start here and add GPT or RPM on top.

Higher-Level Packaging Utilities

The various packaging formats are useful, but they don't make package management easy in themselves. The biggest problem is that they need to automatically fulfill any dependencies between packaged software. A couple of higher-level package utilities — Pacman and Yum — respond to this need by adding a convenience layer on top of the packaging.

Pacman

Pacman is a high-level package manager that can handle almost any package format. It allows you to fetch, install, and manage software packages with ease by placing the burden of configuring those packages with a central authority. The intention is to have the software configured by one person and installed by many. If something doesn't work because it is configured incorrectly, it gets fixed in one place rather than in multiple places. This approach is unique among higher-level package managers because it can deal with highly customized configurations.

Additionally, Pacman's flexibility makes the package manager attractive for large-scale Grid deployment of software. For example, if a project needs a set of software installed at multiple sites, the task can be time-consuming or error-prone: Either someone familiar with installing and configuring the software must go to each site and do the work, or local site administrators have to install the software individually. With Pacman, one needs only to configure a Pacman cache of the software and have each site administrator run a couple of quick commands that will bring in all the project's software, install it, and configure it. Compared with the alternative, setting up a Pacman cache is almost a given for large-scale software deployment efforts that involve nontrivial software configurations.

Yum

Yum is a high-level package manager for RPM. It is similar to Pacman, but with several key differences. The first is that Yum works only with RPM. As such, software available through Yum generally comes with a vanilla configuration, requiring the user to make any configuration changes once the software is installed. Since Pacman can handle .tar.gz packages as well as RPMs, Yum might seem an unlikely contender as a higher-level package manager. However, Yum has some distinct advantages.

Yum is popular. Anyone who has ever used Debian knows that its "apt-get" utility for installing software and dependencies on the fly is incredibly convenient. Before Yum became

available for RPMs, RPM users had to play the dependency game when installing new RPMs, satisfying RPM dependencies by hand. Yum eliminates this annoying procedure.

For RPM-based systems, Yum supplies all the benefits of Debian's renowned "apt-get" utilities. Since RPMs are an order of magnitude more popular than Debian packages, Yum has solved a major problem for a large number of people. Because it deals only with RPMs, it handles them very well and is quickly being adopted by the RPM community. This is making it more and more commonplace.

With the ability to easily set up a Yum repository, creating specialized software caches is a snap if you have RPMs of the software you want to distribute to RPM-based environments. Additionally, those RPMs can be tailored so their default configuration is specific to a project. This feature makes RPM deployment attractive if the software being deployed is available in RPM in the first place and all of the distribution base can accept RPM.

The Correct Tool for the Environment

We have looked at two layers of packaging tools and software. The first layer comprises the packaging software itself, namely GPT and RPM. The second layer comprises manageability tools, namely, Pacman and Yum. Various combinations of these high-level utilities and lower-level packaging formats are possible, depending on your Grid environment.

For heterogeneous environments, which are most common, the decision to use Pacman for deployment of either GPT or RPM software is almost a given. However, if many of your users will be installing the software themselves, or if you have the luxury of having a more homogeneous deployment environment, it's generally best to use a common package format and distribution method such as RPM and Yum.

Resources

GPT
- www.gridpackagingtools.org

RPM
- www.rpm.org

Pacman
- physics.bu.edu/~youssef/pacman

Yum
- linux.duke.edu/projects/yum

Scott Gose is a Systems Engineer with the Globus Alliance, where he works on packaging, builds, and testing. He can be reached at gose@mcs.anl.gov.