Using the openSUSE Build Service to Create Kernel Module Packages

Ann Davis
Ann.Davis@suse.com
August 2011

Overview

- Target Audience: developers who wish to provide kernel modules for multiple distributions
- Agenda
  - Linux Driver Model: The preferred approach
  - Kernel module packages: What are they? Why use them?
  - openSUSE Build Service (OBS): Overview
  - Demo: Build a sample kernel module package for SLE and RHEL using the openSUSE Build Service
  - Summary, questions

The Linux Driver Model

- http://ldn.linuxfoundation.org/node/3759
- Preferred approach for providing kernel modules
- Basic message:
  - Open-source modules
  - Push modules upstream (into mainline kernel)
  - Less work long-term
  - Better end-user experience
    - No kernel-update problems
  - Resources exist to help developers upstream their modules

But what to do if upstreaming isn't possible? One option: create a kernel module package...

Kernel Module Package – What Is It?

- Generically:
  - Binary package (rpm, deb, etc.) that installs drivers or other kernel modules onto the end-user's system
- Ideally:
  - Should also integrate modules correctly with the host OS and kernel
    - Set up module dependencies (depmod)
    - Rebuild initrd if necessary
    - Set up package dependencies to handle kernel updates
    - Provide distro-specific functionality (support tags, etc.)
- Different distros use different terms for kernel module packages
  - kmods, KMPs, etc.

Why Use Kernel Module Packages?

- There are other ways to provide out-of-tree drivers
  - Rebuild from source on the end-user system
  - Try to provide pre-built modules for every existing kernel
  - Other approaches (some proprietary)
- Advantages of Kernel Module Packages
  - Don't require development tools or source code to be installed on the end-user's system
  - Package-level dependency checks
    - Warn/prevent user from installing an kernel update that will break out-of-tree modules
    - (Some) distros include technology to work with kernel module packages (module-init-tools scripts)

How to build a kernel module package? One option: Use the openSUSE Build Service...

openSUSE Build Service – What Is It?

- Online public and free package build and repository hosting service: http://build.opensuse.org
- Public instance of Open Build Service (OBS) technology
- Supports building packages for most Linux distributions
- Creates packages from source code and packaging files
- Web and command-line interfaces
- Provides built packages via YUM repositories

OBS has 20,358 projects, 147,841 packages, 30,234 repositories, 29,039 confirmed developers

(as of July 25, 2011)

Why use OBS?

Advantages

- OBS is public (everyone can view source code)
- Can build for multiple distros and architectures w/o setting up any local build servers
- Can distribute packages from OBS (instead of maintaining local repo hosts)

Disadvantages

- OBS is public (everyone can view source code)
- Can't look at actual build structure (must rely on error reports)
- Not as much personal control (what if OBS is unavailable?)

Note: The Open Build Server technology is open, so organizations and/or individuals can also set up their own OBS instances.

Demo: Using OBS to Build a Kernel Module Package

Demo Overview

- Use single source to create kernel module packages for SUSE Linux Enterprise and Red Hat Enterprise Linux
  - Step 1: Set up local build structure (source code and packaging files)
  - Step 2: Move the local build structure to OBS
  - Step 3: Build the packages on OBS
  - Step 4: Test installing the kernel module packages from the OBS YUM repo

Note: The demo source code and packaging files will also build successfully for several other rpm-based distros.

Step 1 – Create the Local Build Structure

- Create directories
  - Usually put all the source code in a %name-%version directory
- Source code
  - *.c and *.h files along with Makefile/Kbuild file(s)
  - Should build as described in /usr/src/linux/Documentation/kbuild/modules.txt
    - Test: “make -C /lib/modules/`uname -r`/build M=`pwd` modules”
    - Remember to clean up: “make -C /lib/modules/`uname -r`/build M=`pwd` clean”
  - Compress the source code into a tarball

Step 1 – Create the Local Build Structure (cont'd)

- Create a spec file
  - Cross-distro KMP spec file template: http://www.linuxfoundation.org/collaborate/workgroups/driver-backport/samplekmpspecfile
    - Uses standard macros that are defined differently depending on distro
      - SUSE: see /etc/rpm/macros.kernel-source and /usr/lib/rpm/kernel-module-subpackage (installed by kernel-source package)
      - RHEL: see /etc/lib/rpm/redhat/macros and kmodtool (installed by redhat-rpm-config package)
    - %kernel_module_package does the real work
      - Calls kernel-module-subpackage (SUSE) or kmodtool (RHEL)
      - Configurable via options (can completely replace kernel-module-subpackage or kmodtool)
      - Sets up rpm scripts to run depmod, mkinitrd, and weak-modules as necessary at install/uninstall time

Step 2 – Move the Local Build Structure to OBS

- OBS Basics:
  - Structure:
    - Containers:Projects:(Subprojects):Packages
    - Everyone gets a home:<login> project
  - Projects have Build Targets <== distros to build packages for
    - Accessed via “Repositories” tab
  - Projects have Distribution Repos <== where built packages are provided
  - Build targets and distribution repos can be enabled/disabled at project *and* package level
  - Distribution repo from one project can be build target for another project

Step 2 – Move the Build Structure to OBS (cont'd)

- Create the project/subproject
  - home:andavis:linuxcon2011
- Specify the build targets for the project
  - SLES 11 SP1, RHEL 6.0, openSUSE 11.4
- Add a package to the project
  - “sampledriver”
  - Enable/disable building and publishing for each build target
- Upload the package files
  - sampledriver-1.0.tar.bz2
  - sampledriver.spec

Step 3 – Build the Packages

- (Re)Build happens automatically whenever package source files change
  - View build status on the *package* page (“Overview” tab)
    - View full build log by clicking on the “Succeeded” or “Failed” status link
  - View/download built packages by clicking the desired build target on the *package* page
  - View/download from YUM repository by clicking the desired build target on the *project* page
    - Only available if publishing has been enabled

Step 4 – Test: Install the Package(s)

- Test initial install:
  - Register the OBS YUM repo as an install source
  - Use the system's software management tools to install the package(s) from the YUM repo
- Test updates:
  - On OBS:
    - Update the package version
  - On the test system:
    - Ensure that the system recognizes that updated packages are available
    - Install a kABI-compatible kernel update, ensure that the modules continue to work
    - Install a kABI-*incompatible* kernel update, ensure that the installation process warns about conflict

Caveats, Notes

- Not all distros support using kernel module packages to provide out-of-tree modules
- Not all distros implement the LF Driver Backport Workgroup distro-independent macros
- However: OBS supports distro-specific tags, so spec files can be expanded to cover other distro-specific functionality
  - http://en.opensuse.org/openSUSE:Build_Service_cross_ distribution_howto
  - Review other OBS kernel module packages for more complex examples
  - How to build kernel module packages for deb-based distros?
- Packages and repos built on OBS have OBS signatures
  - Some organizations use OBS to build but not distribute packages

References

- Linux Driver Model
  - http://ldn.linuxfoundation.org/node/3759
- LF Driver Backport Workgroup
  - http://www.linuxfoundation.org/collaborate/workgroups/driver-backport
- OpenSUSE Build Service Documentation
  - http://en.opensuse.org/Category:Build_Service
- SUSE Partner Linux Driver Program Site
  - http://wiki.novell.com/index.php/Category:Partner_Linux_Driver_Program
- Red Hat Driver Update Program Site
  - http://dup.et.redhat.com


suse®