openSUSE Build Service

THE BUILDER

The openSUSE Build Service is a free online service for building binary packages. **BY KLAAS FREITAG**

One reason Novell created the community-based openSUSE project was to invite contributions from independent developers. To highlight this emphasis on outside help, Novell also unveiled a new online service that automatically builds binary packages from source code. The openSUSE Build Service [1] initially focused on building packages for the SUSE variants – openSUSE, SLES, and SLED – but the tool was later extended to support packages for other major distributions, such as Red Hat, Fedora, Mandriva, Debian, and Ubuntu.

The openSUSE Build Service (Figure 1), which builds packages for the i586, x86_64, and PowerPC platforms, also tracks changes to dependencies and rebuilds packages automatically to ensure that all the pieces fit together. A developer who makes a change to the project source code just needs to configure the appropriate build specifications, and the Build Service automatically generates new binary packages for all the major Linux distributions.

Packages created by the Build Service then become available to Internet users. The system is equipped with a powerful software search mechanism and a redirector infrastructure, which automatically redirects users to efficient mirror servers. YaST and other software management systems can add the project directories as installation sources.

Architecture

The openSUSE Build Service is organized around two principal components (Figure 2). The *front end* acts as an interface for receiving user input. An API receives requests from Internet clients to create packages, edit repositories, and change metadata. Because the front end is accessible from the Internet, no important data is stored there so that, in the event of an attack, the project source files are safe from tampering.

The *back end*, which is not directly accessible from the Internet, holds the necessary project data and controls the process of building packages. As shown in Figure 2, the back end supports a series of build host systems, where the packages are built in a native environment for the the appropriate Linux variant. The build hosts use Xen virtualization (or an alternative chroot jail) to operate a secure build environment.

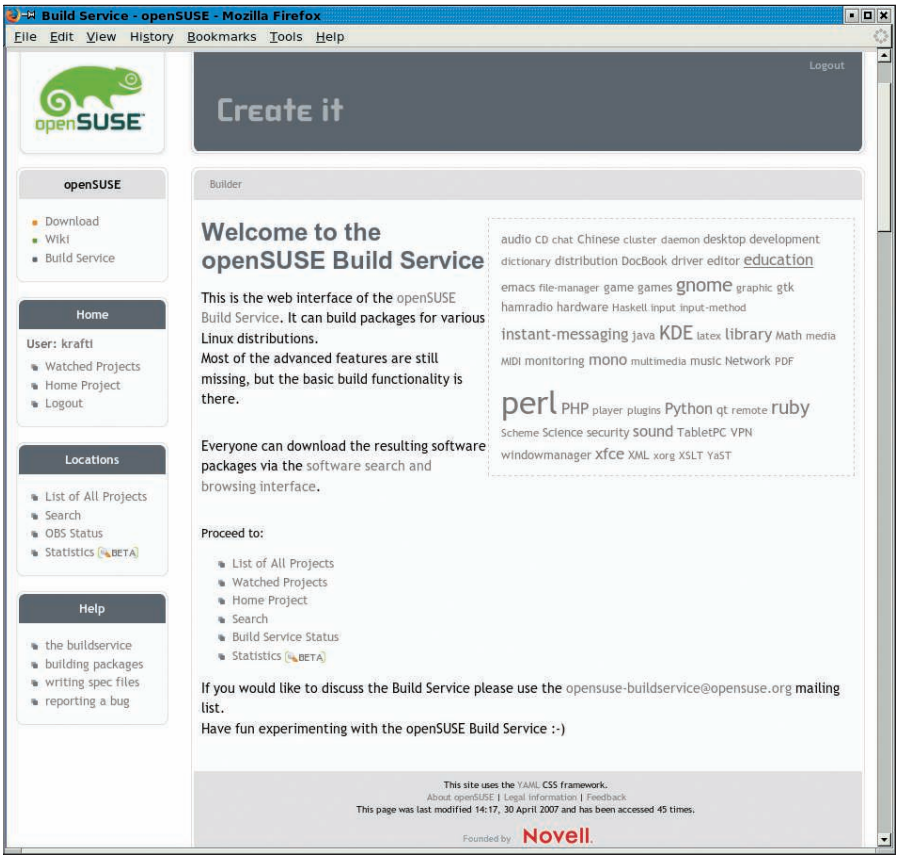

Figure 1: The web interface for the openSUSE Build Service.

Internet users who want to access the Build Service connect to the front end through some form of client application. Three popular client options are:

- osc – A command-line tool, similar to Subversion, written in Python. Experienced users love its command-line efficiency. Developers can check out packages, make changes, and build locally as well as on the server. Osc is modular and expandable through plugins.
- Web client (Figure 1) – The web option lets users manage projects and packages on the server by means of Ajax technology. The web interface is especially useful when several packages are in play simultaneously.
- An experimental KDE client [2] – Installs locally and allows the developer to work within a comfortable KDE interface. This project was part of Google's "Summer of Code."

Developers also can write custom client applications that connect to the service using the front end's REST interface.

The final piece of the service is the mirror interface system (Figure 2), which makes the binary packages created by the Build Service available to Internet users though YaST and other software-management systems.

REST Interfaces

Data is passed between the components of the Build Service using the REST (Representational State Transfer) architecture. REST uses XML documents to transport information. System components can then use ordinary http Requests – such as GET, PUT, or DELETE – to exchange the data.

Rolling Up Your Sleeves

The easiest way to experience the openSUSE Build Service is to create a package step by step. The Build Service uses the same account as the openSUSE wiki or Bugtracker *http://bugzilla.novell.com*. First create an account from the start page of the wiki: *http://en.opensuse.org*.

After you log in, you automatically are brought to the home project page (Figure 3). This home project area is a playground where the user can experiment without breaking anything important.

The Build Service asks for a home project name, title, and description (Figure 4). Clicking the *Add User* link defines additional project users.

The project now exists as a framework for packages, which are then created within the project. By default, the packages within a project inherit all project characteristics. The developer can override these project defaults through the package configuration.

Base repositories are also defined through the project settings. These repositories are normally for stable distributions such as openSUSE 10.3 or SUSE Linux Enterprise Platform 10. In special cases, the developer can point to special repositories that are created because specific aspects of the base system have

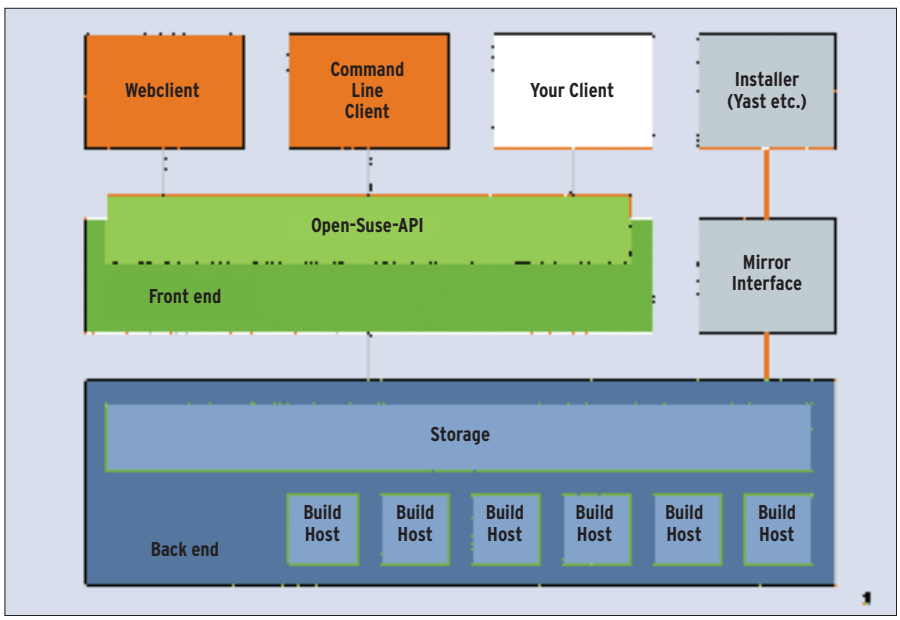

Figure 2: Clients connect to the Build Service front end. The back end handles the package-building process.

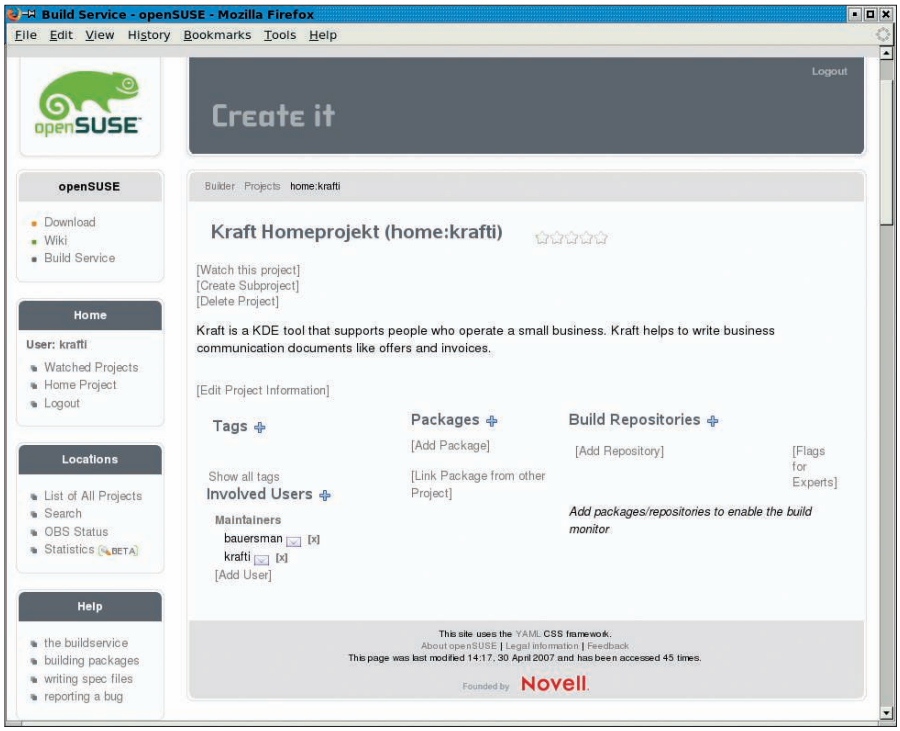

Figure 3: The home project for the Build Service user.

changed. For example, the developer could decide that the new package is not based on standard openSUSE 10.3, but rather on a 10.3 version with an optimized KDE 3.

The example build for this article adds a few important RPM-based repositories, such as the last stable versions of openSUSE 10.2 and 10.3, as well as Fedora and Mandriva.

OpenSUSE Factory has a special position. Factory is the not-yet-stable, next release of openSUSE. Thus, you can expect frequent updates and, because of the continually changing base packages, frequent rebuilds.

Luckily, this complication does not bother the end user because the rebuilds are done automatically. Thus, your own package is at the head of the distribution development and can be adapted quickly to incompatible libraries and similar problems.

Filling the packages in the example project with life is the next step. The command-line client *osc* takes care of this quickly and without complication. However, you first need to check out the project on the local workstation

```
> osc co home:krafti
A  home:krafti
A  home:krafti/kraft
A  home:krafti/kraft/kraft.spec
```

and can input the necessary package sources and update the specification file with the build directives necessary for building the specific package (Figure 5).

To test whether the package is built without errors, a developer can build it for testing locally. The build environment's setup is performed automatically. The developer specifies which hardware and software platforms the Build Service should build.

Listing 1: Spec File Excerpt

```
01 %if 0%{?fedora_version} >= 5
02 source "%{_sysconfdir}/
   profile.d/qt.sh"
03 %configure \
04 --disable-dependency-tracking
   \
05 --with-xinerama \
06 --with-extra-libs=%{_libdir}
07 %else
08 export CFLAGS="$RPM_OPT_
   FLAGS"
09 export CXXFLAGS="$RPM_OPT_
   FLAGS"
10 ./configure \
11 --prefix=/opt/kde3 \
12 --with-qt-libraries=/usr/%_
   lib/qt3/%_lib \
13 %ifarch x86_64 ppc64 s390x
14   --enable-libsuffix=64 \
15 %endif
16 %endif
```

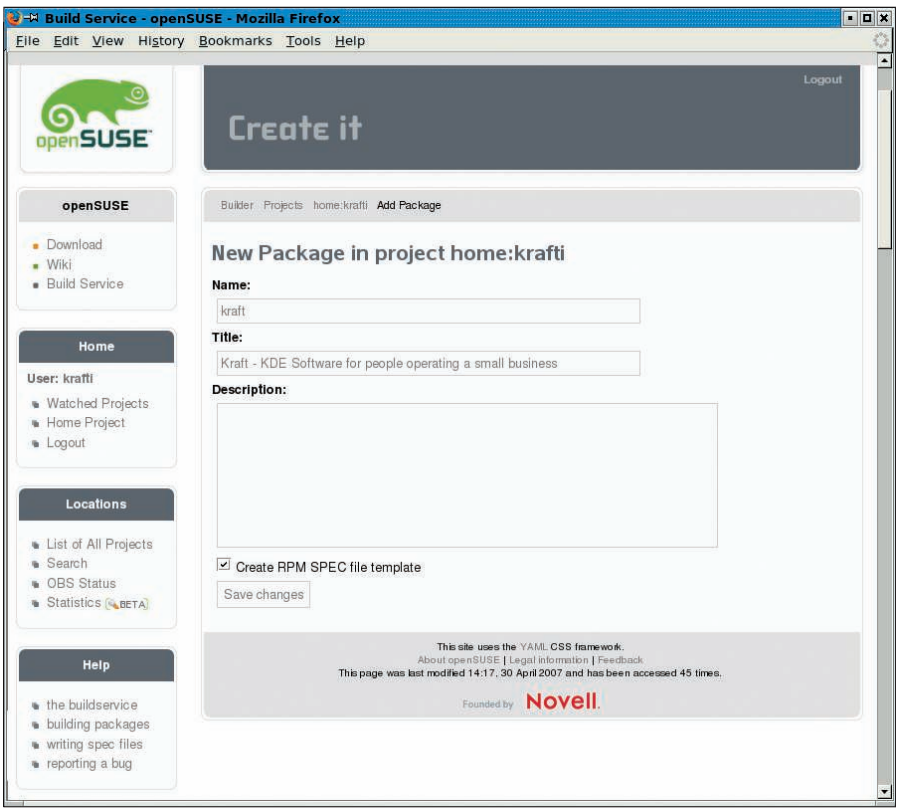

Figure 4: Creating a new package in the Build Service.

For an overview, enter

```
> osc repos
Fedora_7          i586
Fedora_7          x86_64
Mandriva_2007     i586
openSUSE_10.2     i586
openSUSE_10.2     x86_64
...
```

Although the actual build target is openSUSE, a package is create for Fedora for testing:

```
> osc build Fedora_7 ➥
i586 kraft.spec
Getting buildinfo ➥
from server
...
```

Because it pulls a complete base system for Fedora 7 from the server and installs it in a chroot environment, this command needs a little time to finish.

Debugging

Before the package works on all of the desired base systems, a number of runs might be necessary. Debugging directly in the chroot environment to find out why the build failed is helpful to developers. Typical errors range from incomplete build environments to portability problems (keyword: lib64) to file list errors.

Because macro definitions in the Build Service address the apparently inevitable differences in the base systems, it is possible to build a single spec file for all RPM-based distributions. Also, you can use skillfully crafted *if* statements that execute different commands during the build depending on the base system. When the spec file works without errors for all of the desired base systems, *osc* transfers the changes to the openSUSE server:

```
> osc add kraft-0.20. ➥
tar.bz2
> osc add kraft.spec
> osc commit
```

All projects automatically land in the software distribution of openSUSE. Many worldwide mirrors provide the binary packages to make fast downloads possible. However, the user does not have to worry about which mirror to use, as this happens by means of a sophisticated forwarding mechanism that assigns the best mirror for the user and passes this information along transparently.

Search and Find

Searching for and finding software is possible through the software search at *http://software.opensuse.org/search* (Figure 6). Input a search term, and you'll see a list of projects in which this word appears.

A click on the *1-Click Install* button is enough to install the software. YaST automatically logs onto the appropriate software repository and installs the desired software with all of the dependent packages.

Distributing Software

The Build Service has a number of different ways to distribute software. For example, the developer can create a Live system for openSUSE 10.3 that contains a base system and the new package for demonstration purposes. OpenSUSE uses KIWI [3] as an open source system for creating Live images.

The following command checks out the official openSUSE 10.3 Live CD configuration from the repository:

```
svn co
https://forgesvn1.➥
novell.com/➥
svn/opensuse/trunk/➥
distribution➥
```

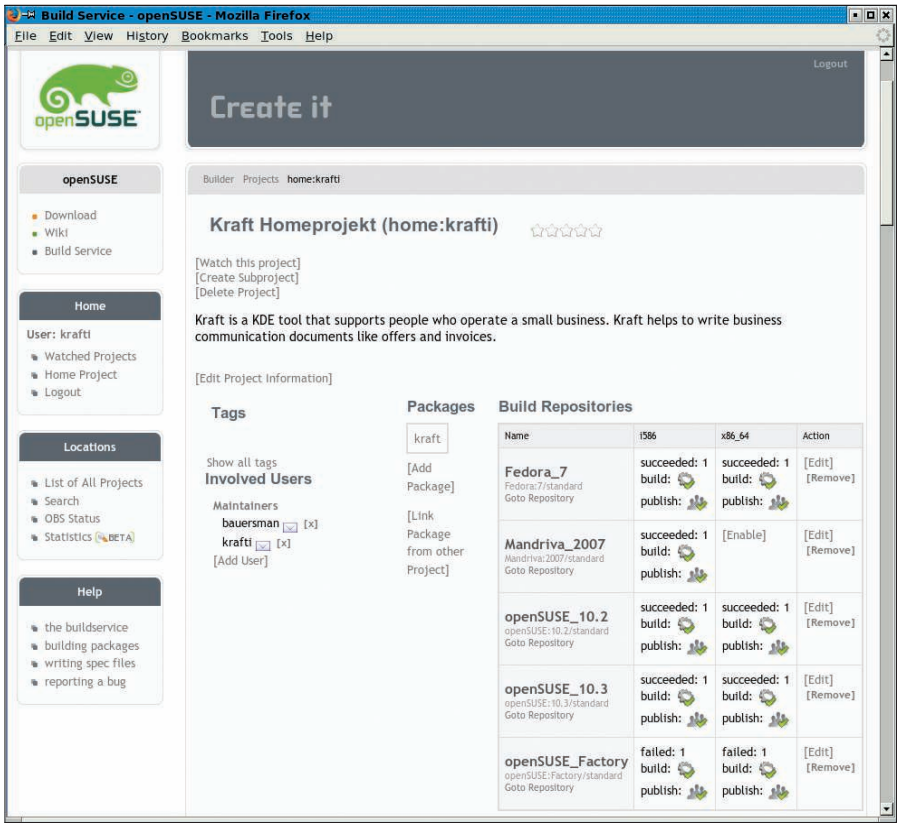

Figure 5: Project view with build status of the various base repositories.

```
/images/➥
kwliveCD-suse-10.3/
```

A new section adds the repository of the openSUSE Build Service to *config.xml*:

```
<repository type="rpm-md">
  <source path="opensuse:➥
  KDE:KDE4/openSUSE_➥
10.3"/>
</repository>
```

This example also adds the record *KDE:KDE4* to the project. Later, KIWI will register the source via *http://download.opensuse.org/repositories*. Also, the developer needs to add the desired packages in the *packages* section. With the command

```
kiwi --root $HOME/mydvd ➥
--prepare $PATH_TO➥
_CONFIG.XML
kiwi --create $HOME/mydvd ➥
-d $HOME
```

you can create a Live CD. The final result appears under the user's home directory in the subdirectoy *mydvd*.

New Directions

The openSUSE Build Service is a useful tool that solves many problems in a transparent, equitable, and efficient way. As a software service on the Internet, the Build Service frees the developer from the ballast of building packages for different distributions.

The next big milestone is support for cooperation: How can I modify a package for which I am not the maintainer? The Build Service will one day make it possible to offer patches to the maintainer. Letting several Build Service installations communicate with each other – for example, to exchange repositories – would be another important improvement. ■

INFO

[1] Build Service wiki: *http://en.opensuse.org/Build_Service*

[2] Build Service KDE client: *http://en.opensuse.org/Build_Service/Rich_Client*

[3] KIWI: *http://en.opensuse.org/Build_Service/KIWI*

[4] OpenSUSE Build Service redirector: *http://en.opensuse.org/Build_Service/Redirector*

[5] Build Service roadmap: *http://en.opensuse.org/Build_Service/Roadmap*

[6] Build Service tutorial: *http://en.opensuse.org/Build_Service_Tutorial*

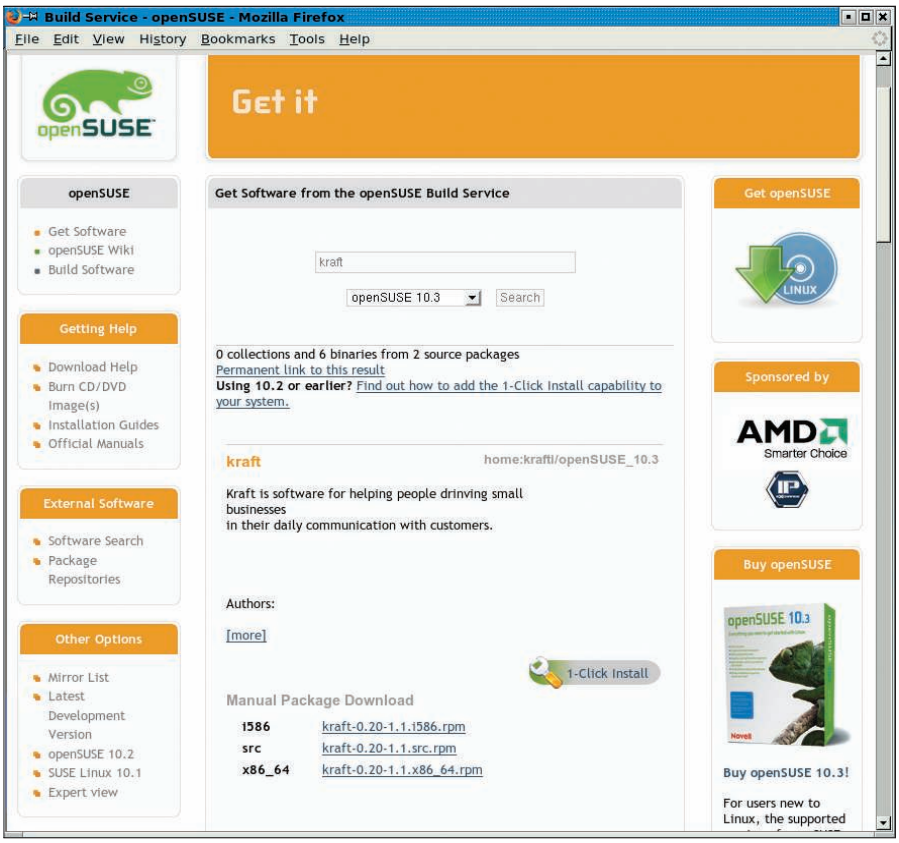

Figure 6: The search form with the search results for one-click installation.