The gory details of the build service backend

Michael Schröder
Novell, Inc
Outline

Data management
  • storing projects and packages
  • the source repository

Server framework
  • data flow client to the backend
  • example: listing package files

Building packages
  • from scheduler to build host and back
  • scalability issues

Automatic rebuild triggering
  • package meta files
Backend basics

Projects, packages and repositories

KDE:KDE4

- kdelibs
- kde toxins
- ...

openSUSE:10.2

Apache

openSUSE 10.2
Fedora Core6

© February, 2007 Novell Inc.
How projects are stored

A project consists of

- a project name
- project meta data (summary, description, repositories...)
- project build configuration (setup information, preferred binary packages, rpm macros...)
- multiple packages

```
projects/home:mlschroe.xml
/home:mlschroe.conf  (if not empty)
/home:mlschroe.pkg/
```
How packages are stored

A package consists of

• a package name
• package meta data (summary, description, ...)
• package revision log

/project/home:mlschroe.pkg/
  screen.xml
  screen.rev

Revision log format (last line is latest revision):

12|76|f6d2e7a398f6bd801d7ddc18eaa76e2d|4.0.2|1155842871|mlschroe|
Revision                                version
Release number                          checkin time
Source repository identifier            checkin user
The source repository

Manages sources of packages

• allows retrieval of old versions
• switching to new source revisions must be atomic
• identical files in multiple projects and revisions should be shared, i.e. not take extra disk space

Implementation

• files are prefixed with the md5sum value of their content
• a revision is identified by the md5sum over a MD5SUMS file
• nothing gets deleted, just new files added
Example: the screen package

revision:

- screen_4.0.2-4.1.diff.gz
- screen_4.0.2-4.1.dsc
- screen_4.0.2-4.1.spec
- screen_4.0.2.orig.tar.gz

MD5SUMS:

<table>
<thead>
<tr>
<th>Hash</th>
<th>File</th>
</tr>
</thead>
<tbody>
<tr>
<td>8f8725fa9b3385042115e84a06866ce6</td>
<td>screen_4.0.2-4.1.diff.gz</td>
</tr>
<tr>
<td>64276af3d6f9c364528fb49223b995a3</td>
<td>screen_4.0.2-4.1.dsc</td>
</tr>
<tr>
<td>47cc233ceb7ba64bf43807978b52c40a</td>
<td>screen_4.0.2-4.1.spec</td>
</tr>
<tr>
<td>ed68ea9b43d9fba0972cb017a24940a1</td>
<td>screen_4.0.2.orig.tar.gz</td>
</tr>
</tbody>
</table>

md5sum MD5SUMS:

<table>
<thead>
<tr>
<th>Hash</th>
<th>MD5SUMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>f6d2e7a398fdbd801d7ddc18eaa76e2d</td>
<td>MD5SUMS</td>
</tr>
</tbody>
</table>
The source repository (cont.)

Example: the screen package

source/screen/
  8f8725fa9b3385042115e84a06866ce6-screen_4.0.2-4.1.diff.gz
  64276af3d6f9c364528fb49223b995a3-screen_4.0.2-4.1.dsc
  47cc233ceb7ba64bf43807978b52c40a-screen_4.0.2-4.1.spec
  ed68ea9b43d9fba0972cb017a24940a1-screen_4.0.2.orig.tar.gz
  ...
  ...
  f6d2e7a398fdbd801d7ddc18eaa76e2d-MD5SUMS
  ...

Revision identifier is MD5SUMS' md5sum:
  f6d2e7a398fdbd801d7ddc18eaa76e2d
The source repository (cont.)

How to retrieve a file from the repository:

Task: read file `screen_4.0.2-4.1.spec` from source revision `f6d2e7a398fdd801d7ddc18eaa76e2d` package `screen`:

- enter directory `source/screen`
- read `f6d2e7a398fdd801d7ddc18eaa76e2d-MD5SUMS`
- find md5sum of file `screen_4.0.2-4.1.spec`:
  `47cc233ceb7ba64bf43807978b52c40a`
- read file
  `47cc233ceb7ba64bf43807978b52c40a-screen_4.0.2-4.1.spec`
Server Framework

Flexible, extensible http server framework

• written for the build server, perl5
• supports file/cpio streaming
• automatic parameter type checking
• forks a process for every request, but requests taking long time can be passed to a special process
  - used for AJAX updates of build status and for logfile streaming
  - if load gets too high, return “retry after n seconds” error
Server Framework (cont.)

Example: getting a source listing

osc

http://api.opensuse.org/source/home:mlschroe/screen

API


src server

dispatch table[]

getfilelist()
Requests are dispatched via a dispatch array:

my $dispatches = [ ...

    '/source/$project/$package rev?' => \&getfilelist,
... ];

sub getfilelist {
    my ($cgi, $projid, $packid) = @_;
    my $rev = getrev($projid, $packid, $cgi->{'rev'});
    my $files = lsrep($projid, $packid, $rev->{'srcmd5'});
    my $dir = {'name' => $packid, 'rev' => $rev->{'rev'}};
    ...
    $dir->{'entry'} = @res;
    return ($dir, $BSXML::dir);
}
Server Framework (cont.)

Conversion to XML is done via the XML::Structured module

```perl
$BSXML::dir = [
    'directory' =>
      'name',
    'rev',
    [ $BSXML::entry ],
];

$BSXML::entry = [
    'entry' =>
      'name',
    'size',
];

$dirxml = XMLOut($BSXML::dir, $dir);
$dir = XMLIn($BSXML::dir, $dirxml);
```
Backend Architecture

- **Src server**
- **Rep server**
- **Build Clients**
- **Scheduler**
  - i586
  - x86_64
- **Repos**
- **Job queue**
- **Job dispatcher**
### Backend Architecture

- **Src server**
- **Rep server**
- **Build Clients**

**Scheduler**
- i586
- x86_64

**getprojpack**

- Return configuration data about every project and package
- Select newest source revision for every package
- Select right spec/dsc file for every package / repository
- Parse spec/dsc files and return package dependencies
Backend Architecture

- **Src server**
- **Rep server**
- **Scheduler**: i586, x86_64
- **Job queue**
- **Job dispatcher**
- **Repos**

- **Read repository data for every project**
- **This includes provides/requires for every binary package**
Backend Architecture

- **Src server**
- **Rep server**
- **Scheduler**
  - i586
  - x86_64
- **Repos**
- **Job queue**
- **Job dispatcher**

- expand package dependencies with repository data
- sort packages by dependencies
- ignore packages blocked by other packages
- add job to job queue if package needs to be rebuilt
Backend Architecture

- **Src server**
  - x86_64

- **Rep server**
  - i586

- **Scheduler**
  - i586
  - x86_64

- **Job queue**

- **Repos**

- **Build Clients**

- **Job dispatcher**

- **Scans job queue for unassigned jobs**
- **Assign to idle build clients**

- **Build**
Backend Architecture

- **Src server**
- **Rep server**
- **Job dispatcher**
- **Build Clients**

- get package sources and build configuration from source server
- get binary packages from repository server
- build the package

- getsources, getconfig
- getbinaries
Backend Architecture

- transmit built binary packages and the log file back to the repository server

© February, 2007 Novell Inc.
Backend Architecture

- Inform scheduler that package build is completed
- Scheduler moves packages into repository and searches for new packages that need building
Scalability

Simplest configuration: everything on one host

- repsrv/ dispatch
- sched i586
- sched x86_64
- srcsrv
- worker
Scalability

Put build client on extra host

- repsrv/dispatch
- sched i586
- sched x86_64
- srcsrv
- worker
Scalability

Add more clients

- repsrv/dispatch
- sched i586
- sched x86_64
- srcsrv
- worker
- worker
- worker
Scalability

Put source server on extra host

- repsrv/dispatch
- sched i586
- sched x86_64
- srcsrv
- worker
- worker
- worker
Scalability

Partition repositories

- repsrv/dispatch
- sched i586
- sched x86_64

- srcsrv

- worker
- worker
- worker
Packages automatically get rebuild if a package they depend on is changed

- naïve implementation that just looks for changed binary packages would lead to endless builds in case of dependency cycles
- Example:
  - gnome-keyring: BuildRequires: CASA-devel
  - CASA: BuildRequires: gtk-sharp
  - gtk-sharp: BuildRequires gtkhtml2-devel
    → gtkhtml2 → libgnome-keyring.so.0
- Solution: use source identifiers, track change propagation, and cut cycles
The dependency meta file

The dependency meta file consists of all package source identifiers used (direct and indirect)

Example: CASA package

- level 0: own source identifier
  e71a2f9181669e4c787e6fbf7ce63414 CASA

- level 1: source md5 of direct dependencies
  ...
  8e72d8e96df97cbea7707ba26a3fc31d gtk-sharp
  ...

- level 2: source md5 of dependencies over one hop
  ...
  0724adba09a56cea17d41f0b35450d45 gtk-sharp/gnome-keyring
  ...

Packages are triggered for rebuild *iff* the meta file of the last build is different from the calculated one
Conclusion

Data storage
- project and package data is stored in simple xml files
- the source repository stores package files in an efficient way, nothing gets deleted

Server Framework
- new functions can be added with just a couple of lines of perl code

Package building
- scheduler → dispatcher → build client → repository
- all data is transferred via http, no NFS

Rebuild triggering
- works by comparing meta files containing source md5sums
General Disclaimer
This document is not to be construed as a promise by any participating company to develop, deliver, or market a product. Novell, Inc., makes no representations or warranties with respect to the contents of this document, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc., reserves the right to revise this document and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes. All Novell marks referenced in this presentation are trademarks or registered trademarks of Novell, Inc. in the United States and other countries. All third-party trademarks are the property of their respective owners.

This work is licensed under the Creative Commons Attribution-Noncommercial-Share Alike 2.5 License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/2.5/.

For other licenses contact author.