Software deployment with Nix
IPA Lentedagen 2005

Eelco Dolstra
eelco@cs.uu.nl

Institute of Information & Computing Sciences
Utrecht University, The Netherlands

March 31, 2005
Software deployment: the art of **transferring software** (components) from one machine to another (and managing it).

The hard part: components should **work the same** on the target machine.

- “DLL hell”
- “Dependency hell”
So why is this hard?

- Difficult to have multiple versions; but we want this to
  - Test upgrades
  - Deal with conflicting dependencies
  - Support different user / service requirements
So why is this hard?

- Difficult to have multiple versions; but we want this to
  - Test upgrades
  - Deal with conflicting dependencies
  - Support different user / service requirements

Diagram:

- gtk+
- wxGTK
- wxPython
- BitTorrent
- zapping

Requires `gtk+-2.4`
So why is this hard?

- Difficult to have multiple versions; but we want this to
  - Test upgrades
  - Deal with conflicting dependencies
  - Support different user / service requirements

Diagram:

- gtk+
- wxGTK
- zapping
- wxPython
- BitTorrent

gtk+ requires gtk+-2.4
wxGTK fails with gtk+-2.4
zapping requires gtk+-2.4

So why is this hard?

- Unreliable dependency information
  - What components are needed?
  - What versions?
So why is this hard?

- Unreliable dependency information
  - What components are needed?
  - What versions?

```
gtk+

wxGTK

wxPython

BitTorrent
```

Eelco Dolstra eelco@cs.uu.nl  Software deployment with Nix
So why is this hard?
Central idea: store all components in isolation.

Unique paths:

/nix/store/605332199533e73b...-gtk+-2.2.4

which is an SHA-256 hash of all inputs used to build the component:

- Sources
- Libraries
- Compilers
- Build scripts
- Build parameters
- System type
- ...

- **Prevent** undeclared **build time** dependencies.
- **Scan** for **runtime** dependencies.
- Deploy only **closures** under the **depends-on** relation.
Unique paths for different versions

```
/nix/store
  └── bd6593219f8dcb63...-gtk+-2.2.4
      └── lib
          └── libgtk-x11-2.0.so.0
  └── ce2d7d2a41456bab...-wxGTK-2.4.2
      └── lib
          └── libwx_gtk2-2.4.so
  └── e889db0595672287...-wxPython-2.4.2.4
      └── (lots of Python bindings)
  └── 9ed8c4231bfde4af...-bittorrent-3.4.2
      └── bin
          └── btdownloadgui.py
  └── 300ccc1a41af3abc...-gtk+-2.4.13
      └── lib
          └── libgtk-x11-2.0.so.0
  └── f51ec7d5663c735e-zapping-0.7.3
      └── bin
          └── zapping
```
Nix expressions

**hello/default.nix**

```nix
{stdenv, fetchurl, perl}:

stdenv.mkDerivation {
  name = "hello-2.1.1";
  builder = ./builder.sh;
  src = fetchurl {
    url = 
    md5 = "70c9ccf9fac07f762c24f2df2290784d";
  };
  inherit perl;
}
```
Nix expressions

```nix
{stdenv, fetchurl, perl}:

stdenv.mkDerivation {
    name = "hello-2.1.1";
    builder = ./builder.sh;
    src = fetchurl {
        md5 = "70c9ccf9fac07f762c24f2df2290784d";
    };
    inherit perl;
}
```

Function arguments
Nix expressions

```
{stdenv, fetchurl, perl}:

stdenv.mkDerivation {
  name = "hello-2.1.1";
  builder = ./builder.sh;
  src = fetchurl {
    md5 = "70c9ccf9fac07f762c24f2df2290784d";
  };
  inherit perl;
}
```

Function arguments

Build attributes

Eelco Dolstra eelco@cs.uu.nl
Software deployment with Nix
Nix expressions

hello/builder.sh

. $stdenv/setup

PATH=$perl/bin:$PATH

tar xvfz $src
cd hello-*
./configure --prefix=$out
make
make install
hello/builder.sh

. $stdenv/setup

PATH=$perl/bin:$PATH

tar xvfz $src

cd hello-*

./configure --prefix=$out

make

make install

Environment initially empty; prevents undeclared dependencies
system/all-packages-generic.nix

```nix
hello = (import ../applications/misc/hello/ex-1) {
  inherit fetchurl stdenv perl;
};

perl = (import ../development/interpreters/perl) {
  inherit fetchurl stdenv;
};

fetchurl = (import ../build-support/fetchurl) {
  inherit stdenv; ...
};

stdenv = ...;
```

Nix expressions

Eelco Dolstra eelco@cs.uu.nl

Software deployment with Nix
Nix expressions

system/all-packages-generic.nix

```nix
hello = (import ../applications/misc/hello/ex-1) {
  inherit fetchurl stdenv perl;
};

perl = (import ../development/interpreters/perl) {
  inherit fetchurl stdenv;
};

fetchurl = (import ../build-support/fetchurl) {
  inherit stdenv; ...
};

stdenv = ...;
```

Eelco Dolstra eelco@cs.uu.nl  Software deployment with Nix
bittorrent = (import ../tools/networking/bittorrent) {
    inherit fetchurl stdenv wxGTK;
};

wxGTK = (import ../development/libraries/wxGTK) {
    inherit fetchurl stdenv pkgconfig;
    gtk = gtkLibs22.gtk;
};

firefox = (import ../applications/browsers/firefox) {
    inherit fetchurl stdenv pkgconfig perl zip libIDL libXi;
    gtk = gtkLibs24.gtk;
};
{ localServer, stdenv, fetchurl
, openssl ? null, db4 ? null, ... }

assert localServer -> db4 != null;
assert sslSupport
    -> openssl != null &&
    && (httpServer -> httpd.openssl == openssl);

stdenv.mkDerivation {
    name = "subversion-1.1.3";
    builder = ./builder.sh;
    src = fetchurl {url=...};
    ...
}
Deployment as memory management

How to discover retained dependencies?

memory ⇔ disk
objects (values) ⇔ components
addresses ⇔ path names
pointer dereference ⇔ I/O
pointer arithmetic ⇔ string operations
dangling pointer ⇔ reference to absent component
Deployment as memory management

How to discover retained dependencies?

memory ⇔ disk
objects (values) ⇔ components
addresses ⇔ path names
pointer dereference ⇔ I/O
pointer arithmetic ⇔ string operations
dangling pointer ⇔ reference to absent component
Deployment as memory management

How to discover retained dependencies?

memory ⇔ disk
objects (values) ⇔ components
addresses ⇔ path names
pointer dereference ⇔ I/O
pointer arithmetic ⇔ string operations
dangling pointer ⇔ reference to absent component
Deployment as memory management

How to discover retained dependencies?

memory ⇔ disk
objects (values) ⇔ components
addresses ⇔ path names
pointer dereference ⇔ I/O
pointer arithmetic ⇔ string operations
dangling pointer ⇔ reference to absent component
Deployment as memory management

How to discover retained dependencies?

memory ⇔ disk
objects (values) ⇔ components
addresses ⇔ path names
pointer dereference ⇔ I/O
pointer arithmetic ⇔ string operations
dangling pointer ⇔ reference to absent component
Correct deployment of component $c$ requires distributing the smallest set of components $C$ containing $c$ closed under the “has-a-pointer-to” relation.

So we have to discover the pointer graph.

This is exactly what garbage collectors for programming languages have to do.
Correct deployment of component \( c \) requires distributing the smallest set of components \( C \) containing \( c \) closed under the “has-a-pointer-to” relation.

So we have to discover the pointer graph.

This is exactly what garbage collectors for programming languages have to do.
Correct deployment of component $c$ requires distributing the smallest set of components $C$ containing $c$ closed under the “has-a-pointer-to” relation.

So we have to discover the *pointer graph*.

This is exactly what garbage collectors for programming languages have to do.
Correct deployment of component $c$ requires distributing the smallest set of components $C$ containing $c$ closed under the “has-a-pointer-to” relation.

So we have to discover the *pointer graph*.

This is exactly what garbage collectors for programming languages have to do.
GC requires a *pointer discipline*:
- Ideally, entire memory layout is known, and no arbitrary pointer formation (e.g., integer ⇔ pointer casts).
- But even C/C++ has rules: pointer arithmetic is not allowed to move a pointer out of the object it points to.
- This is why conservative GC works: assume that everything that looks like a pointer is a pointer.

But software components do not have any pointer discipline.
- Any string can be a pointer.
- Pointer arithmetic and dereferencing directories can produce pointers to any object in the file system.
Finding runtime dependencies

```
/nix/store
  └── bd6593219f8dcb63...-gtk+-2.2.4
      └── lib
          └── libgtk-x11-2.0.so.0
  └── ce2d7d2a41456bab...-wxGTK-2.4.2
      └── lib
          └── libwx_gtk2-2.4.so
  └── e889db0595672287...-wxPython-2.4.2.4
      └── (lots of Python bindings)
  └── 9ed8c4231bfde4af...-bittorrent-3.4.2
      └── bin
          └── btdownloadgui.py
  └── 300ccc1a41af3abc...-gtk+-2.4.13
      └── lib
          └── libgtk-x11-2.0.so.0
  └── f51ec7d5663c735e-zapping-0.7.3
      └── bin
          └── zapping
```

Eelco Dolstra eelco@cs.uu.nl
Software deployment with Nix
Finding runtime dependencies

Contents of libwx-gtk2-2.4.so

- Contents of libwx-gtk2-2.4.so
  - Contents of libwx-gtk2-2.4.so
    - Contents of libwx-gtk2-2.4.so
      - Contents of libwx-gtk2-2.4.so
        - Contents of libwx-gtk2-2.4.so
          - Contents of libwx-gtk2-2.4.so
            - Contents of libwx-gtk2-2.4.so
              - Contents of libwx-gtk2-2.4.so
                - Contents of libwx-gtk2-2.4.so
                  - Contents of libwx-gtk2-2.4.so
                    - Contents of libwx-gtk2-2.4.so
                      - Contents of libwx-gtk2-2.4.so
                        - Contents of libwx-gtk2-2.4.so
Finding runtime dependencies

Contents of libwx-gtk2-2.4.so

```
// Path to store
/nix/store
  bd6593219f8dcb63...-gtk+-2.2.4
    lib
      libgtk-x11-2.0.so.0
  ce2d7d2a41456bab...-wxGTK-2.4.2
    lib
      libwx_gtk2-2.4.so
  e889db0595672287...-wxPython-2.4.2.4
    bin
      btdownloadgui.py
  9ed8c4231bfde4af...-bittorrent-3.4.2
    bin
      zapping
  f51ec7d5663c735e-zapping-0.7.3
    bin
      zapping
```

Eelco Dolstra eelco@cs.uu.nl  Software deployment with Nix
Finding runtime dependencies

Contents of libwx-gtk2-2.4.so

Eelco Dolstra eelco@cs.uu.nl
Software deployment with Nix
Finding runtime dependencies

/nix/store

- bd6593219f8dcb63...-gtk+-2.2.4
  - lib
    - libgtk-x11-2.0.so.0
  - ce2d7d2a41456bab...-wxGTK-2.4.2
    - lib
      - libwx_gtk2-2.4.so
  - e889db0595672287...-wxPython-2.4.2.4
    (lots of Python bindings)
  - 9ed8c4231bfde4af...-bittorrent-3.4.2
    - bin
      - btdownloadgui.py
  - 300ccc1a41af3abc...-gtk+-2.4.13
    - lib
      - libgtk-x11-2.0.so.0
  - f51ec7d5663c735e-zapping-0.7.3
    - bin
      - zapping

Eelco Dolstra eelco@cs.uu.nl
Software deployment with Nix
Finding runtime dependencies

/nix/store
  └── bd6593219f8dcb63...-gtk+-2.2.4
      └── lib
          └── libgtk-x11-2.0.so.0
  └── ce2d7d2a41456bab...-wxGTK-2.4.2
      └── lib
          └── libwx_gtk2-2.4.so
  └── e889db0595672287...-wxPython-2.4.2.4
      └── (lots of Python bindings)
  └── 9ed8c4231bfde4af...-bittorrent-3.4.2
      └── bin
          └── btdownloadgui.py
  └── 300ccc1a41af3abc...-gtk+-2.4.13
      └── lib
          └── libgtk-x11-2.0.so.0
  └── f51ec7d5663c735e-zapping-0.7.3
      └── bin
          └── zapping
To build and install Hello:

```
$ nix-env -if .../all-packages.nix hello
```

When a new version comes along:

```
$ nix-env -uf .../all-packages.nix hello
```

If it doesn’t work:

```
$ nix-env --rollback
```

Delete unused components:

```
$ nix-collect-garbage
```
To build and install Hello:

$ nix-env -if .../all-packages.nix hello

When a new version comes along:

$ nix-env -uf .../all-packages.nix hello

If it doesn’t work:

$ nix-env --rollback

Delete unused components:

$ nix-collect-garbage
To build and install Hello:

```
$ nix-env -if .../all-packages.nix hello
```

When a new version comes along:

```
$ nix-env -uf .../all-packages.nix hello
```

If it doesn’t work:

```
$ nix-env --rollback
```

Delete unused components:

```
$ nix-collect-garbage
```
User operations

▶ To build and install Hello:

```
$ nix-env -if .../all-packages.nix hello
```

▶ When a new version comes along:

```
$ nix-env -uf .../all-packages.nix hello
```

▶ If it doesn’t work:

```
$ nix-env --rollback
```

▶ Delete unused components:

```
$ nix-collect-garbage
```
- Users can have different sets of installed applications.
- `nix-env` operations create new user environments in the store.
- We can atomically switch between them.
- These are roots of the garbage collector.

```
PATH

/nix/links

/ex
current

/nix/store

```

- `eb3266df5c1a...-user-env
  bin

- `firefox`

- `hello`

- `90d0d25ee157...-hello-2.1.1
  bin`

- `hello`

- `e7cb4be9e7c4...-firefox-1.0
  bin`

- `firefox`
User environments

- Users can have different sets of installed applications.
- **nix-env** operations create new **user environments** in the store.
- We can atomically switch between them.
- These are roots of the garbage collector.

```
(nix-env -u hello)
```
User environments

- Users can have different sets of installed applications.
- **nix-env** operations create new **user environments** in the store.
- We can atomically switch between them.
- These are roots of the garbage collector.

**(nix-env -u hello)**
Users can have different sets of installed applications.

- **nix-env** operations create new **user environments** in the store.

- We can atomically switch between them.

- These are roots of the garbage collector.

```
(nix-env -u hello)
```
Users can have different sets of installed applications.

- **nix-env** operations create new **user environments** in the store.

We can atomically switch between them.

- These are roots of the garbage collector.

```
(nix-env -u hello)
```
User environments

- Users can have different sets of installed applications.
- `nix-env` operations create new user environments in the store.
- We can atomically switch between them.
- These are roots of the garbage collector.

(nix-env --remove-generations old)
Users can have different sets of installed applications.

**nix-env** operations create new **user environments** in the store.

We can atomically switch between them.

These are roots of the **garbage collector**.
This is conceptually a source deployment model. We get binary deployment by sharing pre-built components.

On the producer side:

```
$ nix-push $(nix-instantiate .../all-packages.nix) \ http://server/cache
```

On the client side:

```
$ nix-pull http://server/cache
$ nix-env -if .../all-packages.nix hello
```

Installation will now reuse pre-built components, iff they are exactly the same.
This is conceptually a **source deployment model**.

We get **binary deployment** by sharing pre-built components.

On the producer side:

```
$ nix-push $(nix-instantiate .../all-packages.nix) \ 
http://server/cache
```

On the client side:

```
$ nix-pull http://server/cache
$ nix-env -if .../all-packages.nix hello
```

Installation will now reuse pre-built components, **iff** they are exactly the same.
This is conceptually a **source deployment model**.

We get **binary deployment** by sharing pre-built components.

On the producer side:

```
$ nix-push $(nix-instantiate .../all-packages.nix) \ 
   http://server/cache
```

On the client side:

```
$ nix-pull http://server/cache
$ nix-env --if .../all-packages.nix hello
```

Installation will now reuse pre-built components, **iff** they are exactly the same.
An example deployment policy: channels

- Channels allow Nix expressions to be updated automatically.
- Subscribe to a channel:
  
  ```
  $ nix-channel --add http://.../channels/nixpkgs-unstable
  ```

- Fetch latest channel instance:
  
  ```
  $ nix-channel --update
  ```

- Update all installed packages:
  
  ```
  $ nix-env -u '*'
  ```
Channels allow Nix expressions to be updated automatically.

Subscribe to a channel:

$ nix-channel --add http://.../channels/nixpkgs-unstable

Fetch latest channel instance:

$ nix-channel --update

Update all installed packages:

$ nix-env -u '*'
Channels allow Nix expressions to be updated automatically.

Subscribe to a channel:

```
$ nix-channel --add http://.../channels/nixpkgs-unstable
```

Fetch latest channel instance:

```
$ nix-channel --update
```

Update all installed packages:

```
$ nix-env -u '*'
```
Conclusions

Contributions:

- Safe, automatic coexistence of versions/variants.
- Reliable dependencies.
- Multiple concurrent configurations.
- Atomic upgrades/rollbacks.
- Safe garbage collection.
- Binary deployment is automatic.
- Can accommodate many deployment policies.
- Useful for service deployment.
- Integrated continuous integration / release management.

Available at http://www.cs.uu.nl/groups/ST/Trace/Nix.
Further reading

**SCM’03**  E. Dolstra, *Integrating Software Construction and Software Deployment*

**ICSE’04**  E. Dolstra, E. Visser, and M. de Jonge, *Imposing a Memory Management Discipline on Software Deployment*


**CBSE’05**  E. Dolstra, *Efficient Upgrading in a Purely Functional Component Deployment Model*