Techniques and lessons for improvement of deployment processes

Sander van der Burg

Delft University of Technology, EEMCS, Department of Software Technology

Philips Healthcare, Philips Informatics Infrastructure (PII), Best

August 27, 2012
NixOS

A GNU/Linux distribution using the Nix package manager

Sander van der Burg

Techniques and lessons for improvement of deployment processes
Main idea: store all packages in isolation from each other:

```
/nix/store/rpdqxnilb0cg...-firefox-3.5.4
```

Paths contain a 160-bit **cryptographic hash** of all inputs used to build the package:

- Sources
- Libraries
- Compilers
- Build scripts
- ...

```
/nix/store
|-- l9w6773m1msy...-openssl-0.9.8e
    |-- bin
    |   |-- ssh
    |   |   |-- sshd
    |   |   /smkabrbibqv7...-openssl-0.9.8e
    |   |   |-- lib
    |   |   |   |-- libssl.so.0.9.8
    |   |   /c6jbqm2mc0a7...-zlib-1.2.3
    |   |   |-- lib
    |   |   |   |-- libz.so.1.2.3
    |   |   /im276akmsrhv...-glibc-2.5
    |   |   |-- lib
    |   |   |   |-- libc.so.6
```
Nix expressions

{ stdenv, fetchurl, openssl, zlib }:

stdenv.mkDerivation {
    name = "openssh-4.6p1";
    src = fetchurl {
        url = http://.../openssh-4.6p1.tar.gz;
        sha256 = "0fpj1r3bfind0y94bk442x2p...";
    };
    buildCommand = ''
        tar xjf $src
        ./configure --prefix=$out --with-openssl=${openssl}
        make; make install
    '';}

Sander van der Burg
Techniques and lessons for improvement of deployment processes
Nix expressions

**all-packages.nix**

```nix
openssl = import ../development/libraries/openssl {
    inherit fetchurl stdenv perl;
};

zlib = ...;
perl = ...;

stdenv = ...;
openssl = ...
```

- `nix-env -f all-packages.nix -iA openssh`
- Produces a `/nix/store/l9w6773m1msy...-openssl-4.6p1` package in the Nix store.
$ disnix-env -s services.nix -i infrastructure.nix -d distribution.nix
Hydra

Continuous build & integration server
Built upon the Nix package manager
Used at TU Delft for building, integration and testing of several projects on multiple platforms

Sander van der Burg
Techniques and lessons for improvement of deployment processes
Nix applications’ advantages

- **Reliability.** Dependencies are always complete, present and correct. No files overwritten, modified or removed.

- **Reproducibility.** Because side-effects are removed, build functions always yield same result, regardless on what machine they are executed.

- **Generic.** We can invoke arbitrary build processes in Nix functions. Nix is supported on many operating systems: Linux, FreeBSD, Mac OS X, Windows (Cygwin)

- **Efficient.** Only components that must be built are build. Nix store serves as a cache.
How to apply these tools on large codebases?
Software components

We must write Nix expressions for all components:

“The software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.”
Challenges

- **Contractually specified interfaces.** Important for static verification of compositions.
- **Explicit context dependencies.** These are often not always known and may be implicit.
- **Deployed independently.** Without explicit dependencies it is hard to guarantee that this is possible.
- **Subject to composition by third parties.** Third parties must be able to provide or replace dependencies.
Recommendations

- Deal with the deployment complexity from the beginning in your development process
- Make your deployment process scriptable
Decompose your system and their build processes:

- Build times can reduced
- Components can be replaced
- Common components can be shared
Make build-time and run-time properties of components configurable:

- No implicit assumptions, e.g. `/usr/bin/echo`
- Use environment variables: `PATH`, `CLASSPATH` etc.
- Use configuration files
Recommendations

Isolate your build artifacts:

- Store components in separate directories
- Not in shared directories, e.g. /usr/lib or C:\Windows\System32 etc.
Recommendations

Think about a component naming convention:

- `name-version`
- `name-version-architecture`
- `hash-name-version`, e.g.
  
  `/nix/store/l9w6773m1msy...-openssh-4.6p1`
Recommendations

Think about component composition:

- Environment variables
- Symlink trees
- Single folder
Granularity of dependencies:

- How big should components be?
- Big components may trigger unnecessary rebuilds
Conclusion

With these recommendations, deployment processes can be made more reliable, reproducible and efficient.