The Nix project

Sander van der Burg

Delft University of Technology, EEMCS, Department of Software Technology

August 16, 2012
Software deployment

All of the activities that make a software system available for use

*Carzaninga et al.*

Activities

- Building software components
- Installing systems
- Activating software components
- Upgrading software components
Modern systems are rarely *self-contained*:

- Systems are *composed* of many software components
- Dependencies must be *present* and *correct*, both at build-time and run-time
- Non-functional requirements must be met:
  - Reliability
  - Privacy
  - Component licenses
Software deployment

Sander van der Burg

The Nix project
Software deployment

Challenges

- Systems are becoming *bigger*, more distributed and more complicated
- Deployment takes a lot of *effort* and *time*
- Difficult to *reproduce* a configuration elsewhere
- *Upgrading* may break a system and may introduce significant downtimes
Deployment failures

Sander van der Burg

The Nix project
Challenges

- **Complexity.** Many steps must be performed, in the right order.

- **Reliability.** Dependencies may break, files get overwritten, modified or removed. Upgrades may yield different results as fresh installations.

- ** Agility.** In order to quickly deliver value, systems must be deployed faster and on-demand.

- **Genericity.** Most existing deployment solutions only support specific component-types and environments.

Sander van der Burg

The Nix project
NixOS

A GNU/Linux distribution using the Nix package manager
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
|-- 19w6773m1msy...-openssl-0.9.8e
|   |-- bin
|   |   |-- ssh
|   |       |-- sbin
|   |           |-- sshd
|   |-- smkabrbibqv7...-openssl-0.9.8e
|       |-- lib
|       |   |-- libssl.so.0.9.8
|       |       |-- libssl.so.0.9.8
|       |-- c6jbqm2mc0a7...-zlib-1.2.3
|           |-- lib
|           |   |-- libz.so.1.2.3
|           |       |-- libz.so.1.2.3
|           |-- im276akmsrhv...-glibc-2.5
|               |-- lib
|               |   |-- libc.so.6
```
{ stdenv, fetchurl, openssl, zlib }:

stdenv.mkDerivation {
    name = "openssh-4.6p1";
    src = fetchurl {
        url = http://.../openssh-4.6p1.tar.gz;
        sha256 = "0fpjlr3bfind0y94bk442x2p...";
    };
    buildCommand = ''
        tar xjf $src
        ./configure --prefix=$out --with-openssl=${openssl}
        make; make install
    '';"
all-packages.nix

```nix
openssh = import ../tools/networking/openssh {
  inherit fetchurl stdenv openssl zlib;
};

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

stdenv = ...;
openssl = ...;
zlib = ...;
perl = ...;
```

- `nix-env -f all-packages.nix -iA openssh`
- Produces a `/nix/store/l9w6773m1msy...-openssl-4.6p1` package in the Nix store.
In NixOS, all packages including the Linux kernel and configuration files are managed by Nix.

NixOS does not have directories such as: /lib and /usr

NixOS has a minimal /bin and /etc

But NixOS is more than just a distribution managed by Nix
/etc/nixos/configuration.nix

{pkgs,...}:

{
    boot.loader.grub.device = "/dev/sda";

    fileSystems = [ { mountPoint = "/"; device = "/dev/sda2"; } ];
    swapDevices = [ { device = "/dev/sda1"; } ];

    services = {
        openssh.enable = true;

        xserver = {
            enable = true;
            desktopManager.kde4.enable = true;
        };
    };

    environment.systemPackages = [ pkgs.mc pkgs.firefox ];
}
NixOS configuration

nixos-rebuild switch

- Nix package manager builds a complete system configuration
  - Includes all packages and generates all configuration files, e.g. OpenSSH configuration
- Upgrades are (almost) atomic
  - Components are stored safely next to each other, due to hashes
  - No files are automatically removed or overwritten
- Users can switch to older generations of system configurations not garbage collected yet
NixOS bootloader

GNU GRUB version 0.97  (636K lower / 129984K upper memory)

NixOS - Default
Windows
NixOS - Configuration 269 (2009-08-11 23:21:10 - 2.6.27.29-default)
NixOS - Configuration 268 (2009-08-11 18:24:09 - 2.6.27.29-default)
NixOS - Configuration 267 (2009-08-05 10:47:20 - 2.6.27.29-default)
NixOS - Configuration 266 (2009-08-05 10:35:27 - 2.6.27.29-default)
NixOS - Configuration 265 (2009-08-05 10:35:06 - 2.6.27.29-default)
NixOS - Configuration 264 (2009-08-04 15:27:25 - 2.6.27.29-default)
NixOS - Configuration 263 (2009-08-04 15:07:21 - 2.6.27.29-default)
NixOS - Configuration 262 (2009-08-04 14:11:27 - 2.6.27.29-default)
NixOS - Configuration 261 (2009-08-04 10:42:23 - 2.6.27.29-default)
NixOS - Configuration 260 (2009-08-04 10:29:25 - 2.6.27.29-default)

Use the ↑ and ↓ keys to select which entry is highlighted.
Press enter to boot the selected OS, 'e' to edit the
commands before booting, or 'c' for a command-line.

GNU/Linux
Nix/NixOS 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*.

- Scope is *limited* to packages on local machines.

Sander van der Burg

The Nix project
How to extend these concepts to the latest generation of systems?
Service Development Support System (SDS2)

CT  MR  Nuclear medicine  X-Ray  UM
A hospital contains a wide range of medical devices
Each produce status and event logs in their own format
Difficult to perform analysis on data
How can we transform these implicit datasets into something useful?
$ disnix-env -s services.nix -i infrastructure.nix -d distribution.nix
Service model

```nix
{distribution, system}:

let pkgs = import ../top-level/all-packages.nix {
  inherit distribution system;
}; in

{ mobileeventlogs = {
    name = "mobileeventlogs";
    pkg = pkgs.mobileeventlogs;
    type = "mysql-database";
  };

MELogService = {
    name = "MELogService";
    pkg = pkgs.MELogService;
    dependsOn = { inherit mobileeventlogs; };
    type = "tomcat-webapplication";
  };

SDS2AssetTracker = {
    name = "SDS2AssetTracker";
    pkg = pkgs.SDS2AssetTracker;
    dependsOn = { inherit MELogService ...; };
    type = "tomcat-webapplication";
  };
...
```
Infrastructure model

{
    test1 = {
        hostname = "test1.net";
        tomcatPort = 8080;
        mysqlUser = "user";
        mysqlPassword = "secret";
        mysqlPort = 3306;
        targetEPR = http://test1.net/.../DisnixService;
        system = "i686-linux";
    };
    test2 = {
        hostname = "test2.net";
        tomcatPort = 8080;
        
        targetEPR = http://test2.net/.../DisnixService;
        system = "x86_64-linux";
    };
}

Captures machines in the network and their relevant properties and capabilities.
Distribution model

{infrastructure}:

{
    mobileeventlogs = [ infrastructure.test1 ];
    MELogService = [ infrastructure.test2 ];
    SDS2AssetTracker = [ infrastructure.test1 infrastructure.test2 ];
    ...
}

Maps services to machines
Specifications are used to derive deployment process:

- **Building** services from source code
- **Transferring** services to target machines
- **Deactivating** obsolete services and **activating** new services
Distributed infrastructure deployment

- Disnix deploys services, but does not manage the underlying infrastructure
- NixOS has good properties for deployment of a single machine configurations
- Can we extend these properties to distributed systems?
Motivating example: Trac

Welcome to Trac 0.11.7

Trac is a minimalist approach to web-based management of software projects. Its goal is to simplify effective tracking and handling of software issues, enhancements and overall progress.

All aspects of Trac have been designed with the single goal to help developers write great software while staying out of the way and imposing as little as possible on a team's established process and culture.

As all Wiki pages, this page is editable, this means that you can modify the contents of this page simply by using your web-browser. Simply click on the "Edit this page" link at the bottom of the page. WikiFormatting will give you a detailed description of available Wiki formatting commands.

"trac-admin yourenvdir initenv" created a new Trac environment, containing a default set of wiki pages and some sample data. This newly created environment also contains documentation to help you get started with your project.

You can use trac-admin to configure Trac to better fit your project, especially in regard to components, versions and milestones.

TracGuide is a good place to start.

Enjoy!

The Trac Team
Motivating example: Trac

Trac can be deployed in a *distributed* environment:

- Subversion server
- Database server
- Web server
Distributed NixOS configuration

```nix
{ storage = {pkgs, ...}:
  { 
    services.nfsKernel.server.enable = true; ...
  };

postgresql = {pkgs, ...}:
  { 
    services.postgresql.enable = true; ...
  };

webserver = {pkgs, ...}:
  { 
    fileSystems = [
      { mountPoint = "/repos"; device = "storage:/repos"; } ];
    services.httpd.enable = true;
    services.httpd.extraSubservices = [ { serviceType = "trac"; } ]; ...
  };

... }
```
Distributed deployment

charon create network.nix
charon deploy

- Build system configurations by the Nix package manager
- Transfer complete system and all dependencies to target machines in the network
  - Efficient: only missing store paths must be transferred
  - Safe: Existing configuration is not affected, because no files are overwritten or removed
- Activate new system configuration
  - In case of a failure, roll back all configurations
  - Relatively cheap operation, because old configuration is stored next to new configuration
Virtualization

nixos-build-vms network.nix; ./result/bin/nixos-run-vms

- Builds a network of QEMU-KVM virtual machines closely resembling the network of NixOS configurations
- We don’t create disk images
- The VM mounts the Nix store of the host system using SMB/CIFS
Virtualization

Welcome to Trac 0.11.5

Trac is a minimalistic approach to web-based management of software projects. Its goal is to simplify effective tracking and handling of software issues, enhancements and overall progress.

All aspects of Trac have been designed with the single goal to help developers write great software while staying out of the way and imposing as little as possible on a team's established process and culture.

As with all Wiki pages, this page is editable, this means that you can modify the contents of this page simply by using your web browser. Simply click on the "Edit this page" link at the bottom of the page. WikiFormatting will give you a detailed description of available Wiki formatting commands.

"Trac creates a "Trac environment," containing a default set of wiki pages and some sample data. This newly created environment also contains documentation to help you get started with your project.

You can use "trac-admin" to configure your Trac to better fit your project, especially in regard to components, versions and milestones.

TracGuide is a good place to start.

- **TracGuide** — Built-in Documentation
  - **The Trac project** — The Open Source Project
    - **FAQ** — Frequently Asked Questions
    - **TracSupport** — Trac Support

Sander van der Burg

The Nix project
testScript = ''
   $postgresql→waitForJob("postgresql");
   $postgresql→mustSucceed("createdb trac");

   $webserver→mustSucceed("mkdir -p /repos/trac");
   $webserver→mustSucceed("svnadmin create /repos/trac");

   $webserver→waitForFile("/var/trac");
   $webserver→mustSucceed("mkdir -p /var/trac/projects/test");
   $webserver→mustSucceed("trac-admin /var/trac/projects/test initenv ".
      "Test postgres://root\@postgresql/trac svn /repos/trac");

   $client→waitForX;
   $client→execute("konqueror http://webserver/projects/test &");
   $client→waitForWindow(qr/Test.*Konqueror/);

   $client→screenshot("screen");
   '';
nix-build tests.nix -A trac
Experience

- Distributed deployment of a Hydra build environment
- Continuous integration and testing of NixOS
  - NixOS installer
  - OpenSSH
  - Trac
  - NFS server
- Continuous integration and testing of various GNU projects
  - Install NixOS system with bleeding edge glibc
- Other free software projects
Various events may occur in a network of machines:

- Crashing machines
- Adding a new machine
- Change of a capability (e.g. increase of RAM)
- Dynamic Disnix generates infrastructure and distribution models and redeploy a system
License analysis

- We can also trace all files and processes involved in a build process.
- And we can determine the licenses of the original source files to say something about the result.

```
patchelf.cc  g++  patchelf.o  g++  patchelf  install  /usr/bin/patchelf
```
Hydra: Nix based continuous integration and testing

Sander van der Burg

The Nix project
We have shown a number of Nix applications
They provide fully automatic, reliable, reproducible, and efficient deployment for the latest generation of systems
NixOS website: http://nixos.org

- Nix. A purely functional package manager
- Nixpkgs. Nix packages collection
- NixOS. Nix based GNU/Linux distribution
- Hydra. Nix based continuous build and integration server
- Disnix. Nix based distributed service deployment
- Charon. NixOS-based multi cloud deployment tool

Software available under free and open-source licenses (LGPL/X11)