**Services**: sets of running programs that provide some useful facility on a system or network.

Eelco Dolstra, Martin Bravenboer, Eelco Visser {eelco, martin, visser}@cs.uu.nl
Subversion Server

Administrative tasks

- You can create a new repository.
- You can add a new user (only within the cs.uu.nl domain).
- You can edit your user information.

Online information

- Subversion homepage.
- Subversion: The Definitive Guide.

Repositories

The following repositories are hosted on this server:

<table>
<thead>
<tr>
<th>Name</th>
<th>Owner</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3bate</td>
<td>kupers</td>
<td>Repository van 3bate</td>
</tr>
<tr>
<td>adaptive-wavelets</td>
<td>clijena</td>
<td>Adaptive Wavelet Project</td>
</tr>
<tr>
<td>aers</td>
<td>eeeco</td>
<td>Portable cryptographic file system</td>
</tr>
<tr>
<td>afp-exercises-jenn</td>
<td>neus</td>
<td>afp</td>
</tr>
<tr>
<td>afp-exercises</td>
<td>bdumitri</td>
<td>Advanced Functional Programming exercises (Boqdan, Ze)</td>
</tr>
<tr>
<td>afp-exercises-ra</td>
<td>rjlanro</td>
<td>AFP 2005 Exercises, Rjlanro, Amiddelk</td>
</tr>
<tr>
<td>afp-project</td>
<td>bdumitri</td>
<td>Advanced Functional Programming project (Ze, Boqdan, Yvanow, info)</td>
</tr>
</tbody>
</table>
Example: Issue tracking service

[NIXOS-16] NixOS should not wipe my hard drive - Stratego/XT JIRA - Mozilla Firefox

**Key:** NIXOS-16  
**Type:** Bug  
**Status:** Open  
**Priority:** Blocker  
**Assignee:** Armin Hemel  
**Reporter:** Eelco Dolstra  
**Votes:** 0  
**Watchers:** 0

**Original Estimate:** Unknown  
**Remaining Estimate:** Unknown  
**Time Spent:** Unknown

**Description**

The installer should

a) ask for confirmation before installing  
b) not wipe the existing Nix store but reuse it

This site is running on Atlassian JIRA with a free Open Source Project / Non-profit License (license details). JIRA is an issue and bug tracking application. Evaluate JIRA for your organisation.
Service deployment involves a number of steps:

- Deploy software components (e.g., Apache, PostgreSQL, Subversion)
- Edit configuration files (e.g., `httpd.conf`, `viewcvs.conf`)
- Initialise state (e.g., logging directories, database tables)
- Start/stop processes
- ... and all of this possibly on multiple machines / platforms
Problems

- Poor reproducibility (bad CM)
- Hard to support parallel configurations
- Cross-cutting configuration choices
Problem 1: Poor reproducibility

- **Goal:** it should be possible to realise a service by running a single command.
  - E.g., to move it to another machine
  - So no manual installing of missing software components, tweaking of configuration files, creating missing directories, etc.

- **Why is reproducibility hard?**
  - Admins often manually edit configuration files and initialise state
  - Service configuration doesn’t express software component dependencies
Gap between package management and service configuration

- Software components are typically deployed through package managers such as RPM
- Service configuration is typically kept under version management
- However, there is no good way to express the dependencies of the service on the software components
Problem 2: Parallel configurations

- It should be easy to create different instances of a service
  - Test vs. production servers (running on different ports, using different databases, etc.)
  - Instantiations for different users
  - Evolution through time (rollbacks)
- This is hard to support because there are typically lots of configuration files and control scripts that refer to lots of paths for components, state, static data files, etc.
  - /etc/apache/httpd.conf, /etc/init.d/apache, /etc/apache/viewcvs.conf, ...
Example

/etc/apache/httpd.conf for Subversion service (fragment)

ServerRoot "/var/httpd"
ServerName svn.cs.uu.nl:8080
LoadModule dav_svn_module /usr/lib/modules/mod_dav_svn.so
<Location /repos>
  AuthType Basic
  AuthDBMUserFile /data/subversion/db/svn-users
  ...
  SVNParentPath /data/subversion/repos
</Location>
ScriptAlias /viewcvs /usr/viewcvs/www/cgi/viewcvs.cgi

Use cases

▶ Try out with a different set of repositories.
▶ Try out a different Apache.
▶ Try out a different Subversion module.
/etc/init.d/httpd for Subversion service (fragment)
/usr/sbin/apachectl -k start -f /etc/apache/httpd.conf
Many configuration choices are cross-cutting, i.e., impact many different (parts of) configuration files, scripts, etc.

Examples:
- Port numbers
- Host names
- Paths (major source of problems!)

So a change to the configuration choices must be realised in many different places.

- Lots of work
- Danger of inconsistency
Example: port number

In /etc/init.d/httpd.conf

ServerName www.example.org:12443
Listen 12443
<VirtualHost _default_:12443>

In repoman.pl

my $url = "https://www.example.org:12443/"
print "... <a href='$url/repos/$repoName'> ...";
The solution: integrate build management, software deployment, and service deployment into a single formalism.

Namely, the Nix deployment system.

Nix was originally created for *software deployment*.

Nice properties:
- Automatic building of components and their dependencies
- Side-by-side deployment, rollbacks
- Prevention of undeclared dependencies, automatic determination of runtime dependencies
- Functional component description language

All these are also useful for *service deployment*.

Central idea of this paper: *treat services as components*.
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.
Nix 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
    (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 store

Unique paths for different versions

Eelco Dolstra, Martin Bravenboer, Eelco Visser {eelco, martin, visser}@cs.uu.nl
Service Configuration Management
Nix expressions

hello/default.nix

{stdenv, fetchurl, perl}:

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

Eelco Dolstra, Martin Bravenboer, Eelco Visser {eelco, martin, visser} @cs.uu.nl
Service Configuration Management
hello/default.nix

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

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

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

. $stdenv/setup

PATH=$\texttt{perl}/bin:$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
```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

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 = ...;
```

We’re just going to build service configuration files and control scripts as software components, i.e., as immutable objects in the Nix store.
Back to service deployment

services/svn.nix

{ stdenv, apacheHttpd, subversion }:
stdenv.mkDerivation {
    name = "svn-service";
    builder = ./builder.sh;  # Build script.
    control = ./control.in;  # Control script template.
    conf = ./httpd.conf.in;  # Apache configuration template.
    inherit apacheHttpd subversion;
}

Eelco Dolstra, Martin Bravenboer, Eelco Visser {eelco, martin, visser} @cs.uu.nl

Service Configuration Management
services/httpd.conf.in

... LoadModule dav_svn_module
    @subversion@/modules/mod_dav_svn.so
...
services/control.in

#!/@shell@/bin/sh
...
@apacheHttpd@/sbin/apachectl -k start
  -f @out@/httpd.conf
...

The builder just replaces `@subversion@`, `@apacheHttpd@`, etc., with the actual paths of the components in the Nix store (passed as arguments to the function).
Using the service

Building and starting

# upgrade-server svn ./svn.nix

Upgrading and restarting

Idem

Rollback

# /nix/var/nix/profiles/svn-service/bin/control stop
# nix-env -f /nix/var/nix/profiles/svn-service --rollback
# /nix/var/nix/profiles/svn-service/bin/control start
Modularity

- The previous service is *monolithic*: the Apache instance provides only the Subversion service.
- In general it should be possible to run multiple services ("subservices") within a web server, database server, etc.
- Therefore the previous service should be split into:
  - An Apache service
  - A list of *subservices* that plug into Apache
Example

Service Configuration Management

Eelco Dolstra, Martin Bravenboer, Eelco Visser {eelco, martin} @cs.uu.nl
subversionService = import ..:/subversion-service {
    httpPort = 80;
    reposDir = "/data/subversion";
};
twikiService = import ..:/twiki-service {
    twikisDir = "/data/twiki"
};
webServer = import ..:/apache-httpd {
    inherit (pkgs) stdenv apacheHttpd;
    hostName = "svn.cs.uu.nl";
    httpPort = 80;
    subServices = [subversionService twikiService];
};
Example — combined Subversion / TWiki server

```
subversionService = import ../subversion-service {
    httpPort = 80; # Oops!
    reposDir = "/data/subversion"; ...
};
twikiService = import ../twiki-service {
    twikisDir = "/data/twiki"; ...
};
webServer = import ../apache-httpd {
    inherit (pkgs) stdenv apacheHttpd;
    hostName = "svn.cs.uu.nl";
    httpPort = 80; # Oops!
    subServices = [subversionService twikiService];
};
```
The Nix expression language is functional, which makes it easy to define cross-cutting configuration choices once and propagate them to their realisation sites.

This also makes it easy to express variability in configurations.

- E.g., whether to build a test or production server
- Due to hashing any change to the configuration will result in the resulting components stored in a different location in the Nix store
Example

```haskell
{productionServer}:  # Variation point
let {
    port = if productionServer then 80 else 8080;
    webServer = import ./apache-httpd {
        inherit (pkgs) stdenv apacheHttpd;
        hostName = "svn.cs.uu.nl";
        httpPort = port;
        subServices = [subversionService twikiService];
    };
    subversionService = import ./subversion-service {
        httpPort = port;
        reposDir = "/data/subversion"; ... 
    };
    twikiService = import ./twiki-service {
        twikisDir = "/data/twiki"; ...
    };
}
```
Services frequently consist of subservices running on different machines / platforms
Distributed services

Nix already supports multi-platform distributed builds, semi-transparently:

derivation {
    name = "foo";
    builder = ./builder.sh;
    system = "i686-linux"; ... }

Attribute system denotes platform for component build action; if machine is not **i686-linux**, the build will be forwarded to a machine of the right type.

*Starting and stopping* is done by a *service runner* component that remotely starts/stops subservices on the machines identified by their **host** attributes.
Example

PostgreSQL server on FreeBSD

# Build a Postgres server on FreeBSD.
postgresService = import ./postgresql {
    inherit (pkgsFreeBSD) stdenv postgresql;
    host = "losser.labs.cs.uu.nl";  # Machine to run on.
    dataDir = "/var/postgres/jira-data";

    subServices = [jiraService];
    allowedHosts = [jettyService.host];  # Access control.
};
Jetty container on Linux

# Build a Jetty container on Linux.
jettyService = import ./jetty {
    inherit (pkgsLinux) stdenv jetty j2re;
    host = "itchy.labs.cs.uu.nl"; # Machine to run on.
}

# Include the JIRA web application at URI path.
subServices = [
    { path = "/jira"; war = jiraService; }
];

# Build a JIRA service.
jiraService = import ./jira/server-pkgs/jira/jira-war.nix {
    inherit (pkgsLinux) stdenv fetchurl ant postgresql_jdbc;
    databaseHost = postgresService.host; # Database to use.
};
# Compose the two services.

```python
serviceRunner = import ./runner {
    inherit (pkgsLinux) stdenv substituter;
    services = [postgresService jettyService];
};
```
Related work

- Package management tools (e.g., RPM)
  - Don’t do service configuration

- Build managers
  - Make: doesn’t deal with variability very well
  - Better build managers: Odin, Vesta; these have better variability support

- Cfengine (Burgess 1995)
  - Specification of destructive action to be performed to realise a desired target state
  - E.g., add lines $X$ to configuration file $Y$
  - Destructive model and lack of abstraction over paths makes it hard to support multiple instances of a service
  - Disconnect from software deployment
Nix’s properties for software deployment carry over to service deployment
- i.e., full dependencies (⇒ reproducibility), automatic builds, side-by-side deployment of variants, rollbacks
- Nix expression language is good for dealing with cross-cutting configuration choices
- Approach extends to distributed deployment

Links
- Homepage: http://www.cs.uu.nl/groups/ST/Trace/Nix
- Installation instructions for the Subversion service: https://svn.cs.uu.nl:12443/repos/trace/services/trunk/subversion/INSTALL