Zero to Production

Erlang Factory

San Francisco

March 11, 2016

Susan Potter @ Lookout

twitter: @SusanPotter

github: mbbx6spp

InfraEng @ Lookout

```
# finger infraeng
Login: infraeng
Name: Infra Eng @ Lookout
Shell: /run/current-system/sw/bin/bash
Last login Mon Mar 11 14:10 (PST) on pts/10

* Multiple services in prod
* 200-300 hosts monitored already
* Internal Nix channel
* Internal binary cache
* One repository per service
* Repository is source of truth
* We are hiring! Come talk to me. :)
```

% whoami

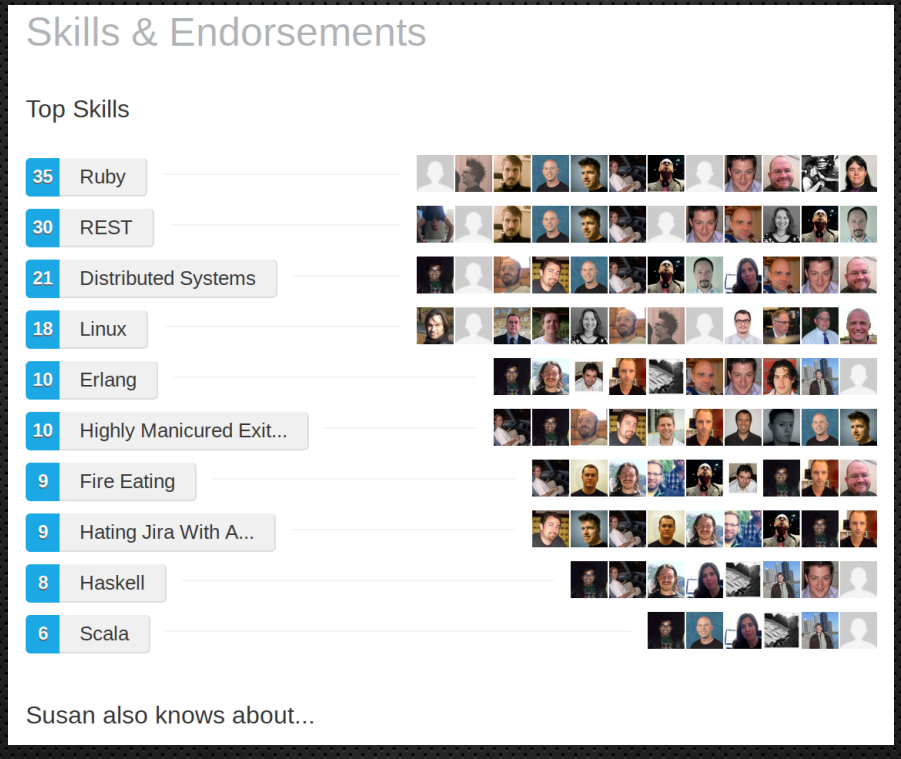

Figure: From backend dev to infrastructure engineering

Reliability

"Those who want really reliable software will discover that they must find means of avoiding the majority of bugs to start with, and as a result the programming process will become cheaper." – EWD340

Reduce Costs & Frustration

"If you want more effective programmers, you will discover that they should not waste their time debugging, they should not introduce the bugs to start with."– EWD340

Why care now?

1. Economic factors
   large distributed deployments

Why care now?

1. Economic factors
   large distributed deployments
2. Human factors
   high churn/turnover, low quality of ops life

Why care now?

1. Economic factors
   large distributed deployments
2. Human factors
   high churn/turnover, low quality of ops life
3. Technological factors
   programmable infrastructure & FP no longer just for academics

More Services

1. Currently 20-30 services
2. More services ready each month
3. Expect 50+ by end of year
4. Various stacks/runtimes

More Environments

1. Ephemeral (integration testing)
2. Product lines (consumer vs enterprise)
3. Performance
4. Partners

More Persistence

Agenda

1. The Problem
2. The Principle
3. Introduce Nix* Ecosystem
4. How Nix Solves Our Problems
5. Lessons Learned

Problem: Software Delivery

Environment provisioning not repeatable in practice

Problem: Software Delivery

Continuous integration builds break with app dependency changes

Problem: Software Delivery

Deploys have unexpected consequences that '–dry-run/–why-run' cannot catch

Requirements: Optimize for . . .

1. Scalability solved by on-demand ”cloud”

Requirements: Optimize for . . .

1. Scalability solved by on-demand "cloud"
2. Reliability solved by . . .

So what yields reliability?

So what yields reliability?

Ability to reason about code.

What allows you to reason about code?

What allows you to reason about code?

Referential transparency (RT)!

RT Refresher

Functions have inputs (Erlang)

```
1 -module(myfuns).
2
3 % Two input arguments here
4 myadd(X, Y) -> X + Y.
5
6 % One input argument here
7 mylen(S) -> len(S).
```

Functions have inputs (Nix)

```nix
# stdenv, fetchurl, gcc, help2man are
# (package) inputs to this package
{ stdenv, fetchurl, gcc, help2man }:
let
  version = "2.1.1";
in stdenv.mkDerivation {
  inherit version;
  name = "hello-${version}";
  src = fetchurl { ... };
  # gcc and help2man are build deps
  buildInputs = [ gcc help2man ];
}
```

Functions return a result (Erlang)

```
1  Eshell V7.0  (abort with ^G)
2  1> c(myfuns).
3  {ok,myfuns}
4  2> myfuns:myadd(1,4).
5  5
6  3> myfuns:mylen("Hello, Erlang Factory.").
7  22
8  4> q().
9  ok
10 5>
```

Functions return a result (Nix)

```
$ nix-repl '<nixpkgs>'

Loading <nixpkgs>\ldots
Added 5876 variables.\

nix-repl> hello = import ./hello.nix { \
  inherit stdenv fetchurl gcc; \
}

nix-repl> hello
derivation /nix/store/...0am-hello-2.1.1.drv\
```

Functions return a result (Nix)

```
nix-repl> "${hello}"
"/nix/store/jg1l1...lsj-hello-2.1.1"

nix-repl> :q

$ nix-build hello.nix \
  --arg stdenv "(import␣<nixpkgs>␣{}).stdenv" \
  --arg fetchurl "(import␣<nixpkgs>␣{}).fetchurl" \
  --arg gcc "(import␣<nixpkgs>␣{}).gcc" \
  --arg help2man "(import␣<nixpkgs>␣{}).help2man"
/nix/store/jg1l1...lsj-hello-2.1.1
```

Only depend on inputs (Erlang)

```
1  \$ cat inputs.erl
2  -module(inputs).
3
4  mylol(X, Y) -> Z.
5  \$ erl
6  Eshell V7.0  (abort with ^G)
7  1> c(inputs).
8  inputs.erl:3: variable 'Z' is unbound
9  inputs.erl:3: Warning: function mylol/2 is unused
10 inputs.erl:3: Warning: variable 'X' is unused
11 inputs.erl:3: Warning: variable 'Y' is unused
12 error
13 2> q().
14 ok
```

Only depend on inputs

```
# Remove help2man from package input arguments
\$ cat hello.nix hello.nix.1
1c1
< { stdenv, fetchurl, gcc, help2man }:
---
> { stdenv, fetchurl, gcc }:

\$ nix-build hello.nix.1 \
  --arg stdenv "(import␣<nixpkgs>␣{}).stdenv" \
  --arg fetchurl "(import␣<nixpkgs>␣{}).fetchurl" \
  --arg gcc "(import␣<nixpkgs>␣{}).gcc"
error: undefined variable help2man at hello.nix:11:23
```

Only depend on inputs

```
# Remove help2man from buildInputs
\$ cat hello.nix hello.nix.2
11c11
<    buildInputs = [ gcc help2man ];
---
>    buildInputs = [ gcc ];

\$ nix-build hello.nix.2 \
  --arg stdenv "(import␣<nixpkgs>␣{}).stdenv" \
  --arg fetchurl "(import␣<nixpkgs>␣{}).fetchurl" \
  --arg gcc "(import␣<nixpkgs>␣{}).gcc" \
  --arg help2man "(import␣<nixpkgs>␣{}).help2man"
...
```

Only depend on inputs

```
these derivations will be built:
  /nix/store/19x32rhqx...mn80-hello-2.1.1.drv
building path(s) /nix/store/v38...2m58h-hello-2.1.1
unpacking sources
unpacking source archive /nix/store/...-hello-2.1.1.tar.gz
source root is hello-2.1.1
...
/nix/...-bash-.../bash: help2man: command not found
Makefile:282: recipe for target ‘hello.1’ failed
make[2]: *** [hello.1] Error 127
...
error: build of /nix/...n80-hello-2.1.1.drv failed
```

Return same result given same inputs

```
1 prop_ref_trans() ->
2   ?FORALL({X, Y}, {integer(), integer()},
3     begin
4       Z0 = myadd(X, Y),
5       Z1 = myadd(X, Y),
6       Z0 =:= Z1
7     end).
```

Return same result given same inputs

```
$ while true; do
  nix-build \
    --arg stdenv "(import␣<nixpkgs>␣{}).stdenv" \
    --arg fetchurl "(import␣<nixpkgs>␣{}).fetchurl" \
    --arg gcc "(import␣<nixpkgs>␣{}).gcc" \
    --arg help2man "(import␣<nixpkgs>␣{}).help2man" \
    hello.nix
done
/nix/store/jg1l1kw...sj-hello-2.1.1
/nix/store/jg1l1kw...sj-hello-2.1.1
...
/nix/store/jg1l1kw...sj-hello-2.1.1
^Cerror: interrupted by the user
```

The Big idea

Referential Transparency

Given same inputs, return same result. Always.

Questions so far?

Figure: Awake?

Mainstream Package Management

Based on shared + mutable state (filesystem)

Violates RT

Alternative Approaches

- shared + immutable
- private + mutable
- expensive coarse grained locks
- hybrid without the expense

Define all inputs

- Force clean build env (chroot)
- Requires explicit inputs
- Full dependency definition

Ensure RT

- Use private mutable space
- Different inputs, different result
- Symlink unique results (atomic op)

Nix Ecosystem

- Expression language: Nix
- Package management: Nix
- Channel: ‘¡nixpkgs¿’
- Operating System: NixOS
- Configuration ”modules”: NixOS modules
- Provisioning: NixOps
- Orchestration: Disnix
- CI: Hydra

Repeatable Dev Envs

```
$ nix-shell -p erlangR17_odbc
these paths will be fetched (37.65 MiB download, 112.65 MiB
  /nix/store/0jvs...3vd-unixODBC-2.3.2
  /nix/store/wf7w...6fp-erlang-17.5-odbc
fetching path /nix/store/0jvs...-unixODBC-2.3.2...
...
[nix-shell:~]$ erl
Erlang/OTP 17 [erts-6.4] [source] [64-bit] ...
Eshell V6.4  (abort with ^G)
1>
```

Repeatable Dev Envs

```
$ nix-shell -p erlangR18_javac
these paths will be fetched (38.04 MiB download, 113.91 MiB
  /nix/store/94a...b3xn-erlang-18.2
fetching path /nix/store/94a...b3xn-erlang-18.2...
...
[nix-shell:~]$ erl
Erlang/OTP 18 [erts-7.2] [source] [64-bit] ...

Eshell V7.2  (abort with ^G)

1>
```

Repeatable Dev Envs

```
$ declare pkghost="releases.nixos.org"
$ declare release_url="https://${pkghost}/nixos"
$ nix-channel --add \
  "${release_url}/16.03-beta/nixos-16.03.30.2068621" nixpkgs
$ nix-shell
these derivations will be built:
  /nix/store/267y...-elm-0.16.0.drv
these paths will be fetched (31.49 MiB download, 379.99 MiB
  /nix/store/0bkd...-scientific-0.3.4.4
  /nix/store/0d3y...-nodejs-4.3.1
...
building path(s) /nix/store/jjzr...-elm-0.16.0
created 6 symlinks in user environment
```

Repeatable Dev Envs

```
$ cat shell.nix
{ pkgs ? import <nixpkgs> {}, ... }:
let
  inherit (pkgs) stdenv;
in stdenv.mkDerivation {
  name = "myerlprj-devenv";
  buildInputs = with pkgs; [
    gitFull           # Developer dependency
    erlangR18         # Erlang version to use
    hex2nix rebar3    # Erlang dev cycle tools
    postgresql        # RDBMS
    elmPackages.elm # for front-end compiler
  ];
  ...
```

Repeatable Dev Envs

```
  ...
  shellHook = ''
    export SERVICE_PORT=4444
    export DATABASE_PORT=5432
    export DATABASE_PATH=$PWD/data
    export LOG_PATH=$PWD/log
    if [ ! -d "${DATABASE_PATH}" ]; then
      initdb "${DATABASE_PATH}"
    fi
    pg_ctl -D "${DATABASE_PATH}" \
      -l "${LOG_PATH}" \
      -o --path="${DATABASE_PORT}" start
  '';
}
```

Consistent CI Deps

```
1 $ head -3 z/ci/verify
2 #!/usr/bin/env nix-shell
3 #!nix-shell -I nixpkgs=URL
4 #!nix-shell -p erlangR18 postgresql -i bash
```

Consistent CI Deps

```
...
set -eu

! test -d "${DATABASE_PATH}" && \
  initdb "${DATABASE_PATH}"
elm-make priv/elm/*
rebar3 clean compile dialyzer
pg_ctl -D "${DATABASE_PATH}" \
  -l "${LOG_PATH}" -o \
  --port="${DATABASE_PORT}" start
rebar3 ct
pg_ctl -D "${DATABASE_PATH}" stop
```

Consistent CI Deps

- Pin channel versions =¿ source + CI consistency
- Update CI build deps with app code
- No OOB ‘converge’-ing CI build hosts!

Predictable Deploys

- Diff dependency path tree
- Test node configuration in VM
- Test NixOS module logic
- Security auditing

Diff Dependencies

```
$ nix-store -qR /nix/store/*-myerlprj-*
/nix/store/8jhy2j7v0mpwybw13nd4fjlsfqc9xnlh-write-mirro
/nix/store/17h0mw5sipbvg70hdsn8i5mai461918c-move-docs.sh
...
/nix/store/p6gn7inwvm61phqw3whhlbl20n8c5dgb-git-2.7.1.drv
/nix/store/z2jvckzhy5322d9ir0xv2hbqp6yakayj-myerlprj-devenv
```

Machine Config

```nix
{ config, pkgs, ... }:
let
  inherit (pkgs) lib;
  ntpF = (idx: "${idx}.amazon.pool.ntp.org")
  domain = "example.com";
in {
  boot.cleanTmpDir = true;
  boot.kernel.sysctl = {
    "net.ipv4.tcp_keepalive_time" = 1500;
    # other sysctl key-values here...
  };
  networking.hostName = "nixallthethings.${domain}";
  networking.firewall.enable = true;
  services.ntp.servers = map ntpF (lib.range 0 3);
  services.zookeeper.enable = true;
  security.pki.certificateFiles = [./internal_ca.crt];
  time.timeZone = "UTC";
}
```

Test Machine Config (VM)

```
1 $ env NIXOS_CONFIG=$PRJROOT/priv/nix/config.nix \
2   nixos-rebuild build-vm
3 $ ./result/bin/run-hostname-vm
4 ...
5
6 $ env NIXOS_CONFIG=$PRJROOT/priv/nix/config.nix \
7   --target-host myerlprj-test-1.integ.bla \
8   nixos-rebuild build-vm
```

Module Integration Testing

```
$ grep -A8 elasticsearch.enable $PWD/priv/nix/config.nix
    elasticsearch.enable = true;
    elasticsearch.jre =
      mychannel.elasticsearch_2_2_0;
    elasticsearch.jre =
      mychannel.oraclejre8u74;
    elasticsearch.node.name =
      "elasticsearch-0.${domain}";
    elasticsearch.dataDir =
      [ "/data0" "/data1" "/data3" ];
```

Module Integration Testing

```
$ grep -A8 "node health" $PWD/priv/nix/modules/elasticsearch.nix
  subtest "elasticsearch node health", sub {
    $es0->waitForUnit("elasticsearch.service");
    $es1->waitForUnit("elasticsearch.service");
    $es0->succeed("${waitForTcpPort "es0" 9300 60}");
    $es1->succeed("${waitForTcpPort "es1" 9300 60}");
    $es0->succeed("${curl "es0" 9200 "/"}");
    $es1->succeed("${curl "es1" 9200 "/"}");
  }
```

Security Auditing

```
$ nix-store -qR /path/to/app/pkg | sort | uniq
/nix/store/002v...-libdc1394-2.2.3
/nix/store/04bw...-expat-2.1.0
/nix/store/04df...-haskell-x509-validation-ghc7.8.4-1.5.1-s
/nix/store/06p6...-packer-e3c2f01cb8d8f759c02bd3cfc9d27cc1a
...
/nix/store/zv9r...-perl-libwww-perl-6.05
/nix/store/zvgj...-pypy2.5-stevedore-0.15
/nix/store/zw00...-libpciaccess-0.13.3
/nix/store/zz78...-libdvbpsi-0.2.2
```

Security Auditing

```
$ nix-store -qR /run/current-system | grep openssl
/nix/store/x1zwzk4hrvj5fz...9hyn-openssl-1.0.1p
/nix/store/m4kzbwji9jkw71...lx92-openssl-1.0.1p
```

Tradeoffs

- Provisioning not solved
  Nixops expressiveness vs Terraform ‘coverage’
- Steep learning curve
  Docs great reference, but bad for n00bs!
- Lots of upfront setup
  Internal Nix channels vs nixpkgs fork curation

Benefits

- Repeatable dev envs
- Consistent CI
- Predictable deploys
- Real rollback

The Win!

Simplicity at its core

What Next?

- Nix AWS Provisioning
- Idris to Nix Backend
- NixBSD Anyone????
- Nix on BEAM???

Where to Next?

- Nix Manual:
  http://nixos.org/nix/manual
- NixOS Manual:
  http://nixos.org/nixos/manual
- Nix Cookbook:
  http://funops.co/nix-cookbook
- Nix Pills (by Lethalman)

Questions

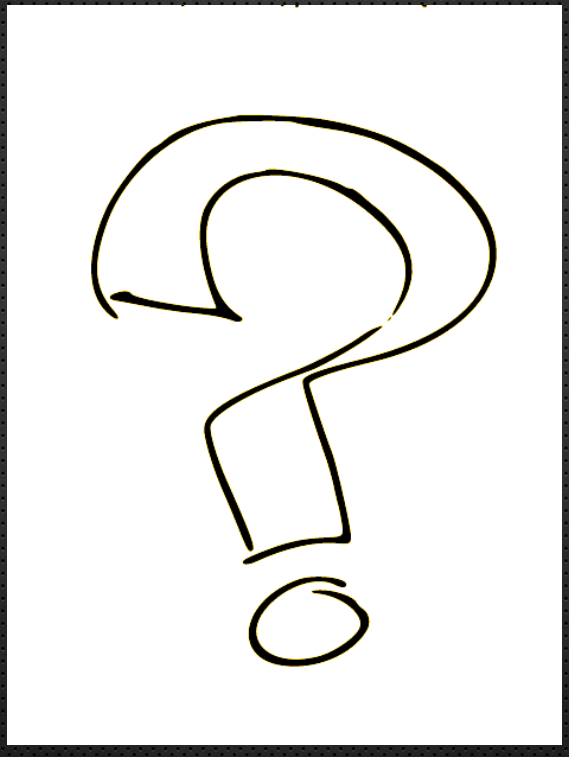

Figure: Heckle me @SusanPotter later too.