Erlang On NixOS

Managing And Releasing Erlang Systems In The Cloud With A Fully Declarative ~~Package Manager~~ System

Eric Merritt

March 11, 2016

Declarative Supervision

```
init(_Args) ->
    SupFlags = #{strategy => one_for_one,
                 intensity => 1,
                  period => 5},
    ChildSpecs = [#{id => ch3,
                    start => {ch3, start_link, []},
                    restart => permanent,
                    shutdown => brutal_kill,
                    type => worker,
                    modules => [cg3]}],
    {ok, {SupFlags, ChildSpecs}}.
```

Declarative Releases

```
{application, 'gremlin',
 [{description, "Don't feed them after midnight"},
  {vsn, "0.1.0"},
  {registered, []},
  {mod, {'gremlin_app', []}},
  {applications,
   [kernel,
    stdlib,
    elli,
    jsx
   ]}
 ]}.
```

Declarative Releases

```
{relx, [{release, {thorndyke, "0.0.1"},
         [thorndyke]},

        {include_erts, false},

        {extended_start_script, false}]}.
```

????

Imperative Deployment

1. Create Tarballs/deb/rpm/fpm
2. Deploy Ansible/Salt/Puppet/Chef
3. Bake your images, or upgrade the boxes

Side Effect Hell


NixOS

What NixOS is

- Purely Functional Package Manager
- Linux Distribution based on the Nix Package Manager

Problems it solves

Directly Relevant

- Managed Via Files in a Git Repository
- Full dependency information
- Supports multiple versions of a package installed at the same time

Indirectly Relevant

- Upgrades are atomic
- Rollbacks
- Anyone can install packages

How it solves those problems

- Purely functional language to describe how to build packages and their dependencies
- Build results only depend on declared inputs.
- Packages never change after they have been built.

Store all packages in isolation from each other

/nix/store/2l55rylrv22....-renderproto-0.11.1.tar.bz2.

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

1. sources
2. libraries
3. compilers
4. etc ....

Declarative System

```
{ config, lib, pkgs, ... }:
{
  users.extraUsers.thorndyke = rec {
    description = "Thorndyke system user";
    home = "/home/thorndyke";
    createHome = true;
    shell = "${pkgs.bash}/bin/bash";
  };
  services.thorndyke = {
    enable = true;
    user = "thorndyke";
  };
}
```

Service Definition

```
{config, pkgs, lib, ...}:
......
      config = mkIf cfg.enable {
        systemd.services.thorndyke = {
          description =
            "Start the thorndyke user under \${cfg.u
          after = [ "network.target" ];
          wantedBy = [ "multi-user.target" ];
          serviceConfig.ExecStart =
            ''/var/setuid-wrappers/sudo -u \${cfg.u
            \${pkgs.thorndyke}/var/sunlight/thorndy
        };
      };
}
```

Package Definition

```
{ stdenv, erlangPackages, bash,
  nettools, erlang }:
erlangPackages.buildRebar3 {
  name = "thorndyke-0.0.1";
  src = ...;

  buildInputs = [ bash nettools erlang ];
  erlangDeps = with erlangPackages; [ elli jsx uri ];
  installPhase = ''
    runHook preInstall
    target="$out/var/thorndyke"
    erlang="${erlang}"
    make PREFIX=\$target install
    substituteAllInPlace \$target/thorndyke/bin/thornd
    runHook postInstall
  '';
}
```

Non Hex Packages

```
{stdenv, fetchFromGitHub, buildRebar3 }:
let
  pkg = self: buildRebar3 rec {
    name = "elli";
    version = "1.0.4";

    src = fetchFromGitHub {
        owner = "knutin";
        repo = "elli";
        rev = "a15f838b4223caf7faa616cbadac4b250215d2f
        sha256 = "1ybf1p7bqbl4cg469qdx6rwdl4r1p7h4hiysh
    };
};
in stdenv.lib.fix pkg
```

How We Did It

Build Support

- reads rebar.config and environment to build the dep structure
- symlinks in packages from the nix dependency environment
- rewrites rebar.config where it needs to

Package Support

- Pull down every package from Hex.pm
- Generate nix expressions for packages in dependency order using the same dependency algo as rebar3

References

- http://nixos.org
- https://github.com/erlang-nix/rebar3-nix-bootstrap
- https://github.com/erlang-nix/hex2nix