Atomic Upgrading of Distributed Systems

HotSWUp'08
Sander van der Burg, Eelco Dolstra, Merijn de Jonge
November 3, 2009
Introduction

• Nix deployment system
• Disnix deployment system
• Atomic upgrading
Distributed systems

- Distributed systems
  - Consists of multiple autonomous computers
  - Components work together to reach a common goal
  - Appear to a user as one logical system
  - Components have intra-dependencies and inter-dependencies
Dependency relationships
Upgrading distributed systems

• Upgrading distributed systems is complex:
  • Usually a semi automatic process
  • Requires up-to-date documentation
  • Requires people with skills
  • Tedious and error prone
  • Is not an atomic operation
Upgrading distributed systems

- Upgrading distributed systems should be a simple process
  - Automatic process
  - Configurations should be captured in models
  - Upgrading should be an atomic operation
Nix Deployment System

• Is a package manager, like RPM
• Builds packages from Nix expressions
• Stores components in isolation in a Nix store
• Nix profiles which abstracts over store components
• Upgrading is atomic
• Deals with single systems
Nix expression

rec {
  HelloService = derivation {
    name = "HelloService-1.0";
    src = fetchurl {
      url = http://nixos.org/.../HelloService.tar.gz;
      md5 = "de3187eac06baf5f0506c06935a1fd29";
    };
    buildInputs = [ant jdk axis2];
    builder = ./builder.sh;
  };

  HelloWorldService = derivation { ... };
  stdenv = ...
  firefox = import ...
  ... # other package definitions
}
Disnix Deployment System

- Extends the Nix deployment system with support for distributed systems
- Provides remote access to the Nix stores and Nix profiles through a webservice interface
- Introduces three model types to model a distributed system
- Uses a variant of the two-phase commit algorithm to allow distributed atomic commits
Disnix overview
Services model

```nix
rec {
    pkgs = import ./pkgs.nix;

    HelloService = {
        pkg = pkgs.HelloService;
        dependsOn = [];
    };
    HelloWorldService = {
        pkg = pkgs.HelloWorldService;
        dependsOn = [ HelloService ];
    };
}
```
{  
  itchy = {
    hostname = "itchy";
    targetEPR = http://itchy/.../DisnixService;
  };
  scratchy = {
    hostname = "scratchy";
    targetEPR = http://scratchy/.../DisnixService;
  };
}
Distribution model

{services, infrastructure}:

[
    { service = services.HelloService;
      target = infrastructure.itchy; } 
    { service = services.HelloWorldService;
      target = infrastructure.scratchy; } 
]

<?xml version="1.0" encoding="utf-8"?>
<expr>
  <list>
    <attrs>
      <attr name="service">
        <string value="/nix/store/bw7dnw...-HelloService-0.1"/>
      </attr>
      <attr name="target">
        <string value="http://itchy:8080/axis2/services/DisnixService"/>
      </attr>
    </attrs>
    <attrs>
      <attr name="service">
        <string value="/nix/store/2490znhi8...-HelloWorldService-0.1"/>
      </attr>
      <attr name="target">
        <string value="http://scratchy:8080/axis2/services/DisnixService"/>
      </attr>
    </attrs>
  </list>
</expr>
Atomic commits

• Commit-request phase:
  – Build all services on the coordinator machine
  – Transfer the services and intra-dependencies to the cohort machines through the webservice interface

• Request phase, on each cohort:
  – Deactivate the old components
  – Uninstall old components in profile
  – Install new component in profile
  – Activate new the components
  – All connections to the services are blocked
Concluding remarks

• We have demonstrated that we can extend the Nix approach of upgrading single systems to distributed systems
  • Upgrading can be done from a declarative specification
  • Upgrading is an atomic operation
• Distribution of services is still a static process
  • We are developing a dynamic approach based on quality of service models
• [http://www.nixos.org](http://www.nixos.org)