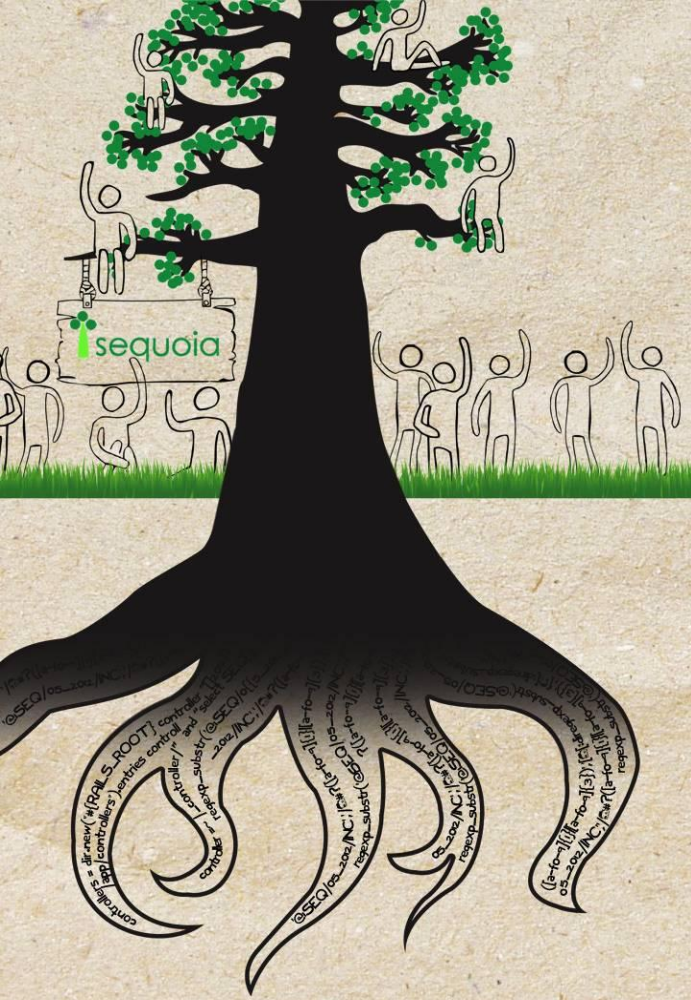

DevOps: Puppet

Stephen Mouring
Director of Software Engineering

Sequoia Holdings

Introductions

Introduction

- Software Developer
- Director of Software Engineering for Sequoia Holdings Inc.
- Full Stack Development
  - Recent focus on big data and the cloud
- Current project is transitioning to DevOps model
- Utilizing Puppet for Infrastructure Configuration Management

Introduction: Sequoia

- Employee Owned
- Broad competencies
- AWS and BigData specialty
- IC customer branching out into the commercial space
  - Amazon Partner
  - Cloudera Partner
- Currently involved in AWS transition efforts across four agencies

Overview

Overview

- What is DevOps?
- What is Puppet?
- Puppet Concepts
- Demo
- AWS/C2S Specifics

What is DevOps?

Intersection of Software **Dev**elopment and System **Op**eration**s**

Methodology

Not A

Technology

Leverages the best in multiple disciplines

Automation

Repeatability

preserving institutional knowledge

Frameworks

that generalize specialized knowledge

What is Puppet?

Server configuration **automation**

Server configuration **enforcement**

User specifies desired server state...

Puppet ensures state is applied...

… and remains applied

Example

Raspberry Pi configuration

Why?

Manual steps do not **scale**

especially for recurring tasks like patching!

Manual steps are **error prone**

Automation enables **new behavior**

such as AWS autoscaling

Declarative

Not

Imperative

(Which means you tell it **what** *not how…*)

State specified using Puppet language

Can either run standalone or client/server

Puppet Concepts

Suffers from disorganized documentation...

… and (deliberately?) indirect marketing

Concept: **Resource**

Configuration building block

```
<TYPE> { '<NAME>':
     <ATTRIBUTE>,
     <ATTRIBUTE>,
     ...
}
```

```
File { ‘/tmp/hello.txt’:
     owner => centos,
     group => users,
     mode => 755,
     content => 'Puppet was here.',
}
```

Concept: **Resource Type**

Define Resource attributes and implementation

```
<TYPE> { '<NAME>':
     <ATTRIBUTE>,
     <ATTRIBUTE>,
     ...
}
```

Several built in:
File, Package, Service, User, Group ...

You can also write your own

Resource Types are a lens for viewing system configuration

DEMO:

Examining Resource Types

Concept: **Class**

A Class is a **group** of related **resources**

```
class <NAME> {
    <RESOURCE>
    <RESOURCE>
}
```

```
class tomcat {
    User { 'tomcat':
        group => 'uesrs'
    }
    Package { 'apache-tomcat-7':
        ensure => present
    }
    ...
}
```

A Class should have a single purpose

Classes are **defined**

Then they must be **declared**

```
include <NAME>
```

```
include tomcat
```

Defining creates a class

Declaring indicates a class should be applied

Declaring a Class applies all contained resources

(at the **same time**!)

unless you specify otherwise

Concept: **Node**

A Node represents a particular server

Node contains Resources and Classes

```
node ‘<NAME>’ {
     <RESOURCE>,
     ...

     <CLASS DECLARATION>,
     ...
}
```

```
node 'example01.example.com' {
    User { 'foo':
        home => '/home/foo'
    }

    include tomcat
}
```

Node determine which configuration is applied to which servers

The Node name matches a server’s hostname

(‘default’ and wildcards also available)

Concept Review

Concept: **Manifest**

A file that contains:

- Resources
- Classes (definitions / declarations)
- Nodes

Ends in `.pp`

(we will build on this in a moment)

Puppet Architecture

Standalone

Client / Server

(But they called it Master/Agent)

Because Puppet Master was too good a phrase to not use...

In Standalone mode you explicitly invoke Puppet with a Manifest

In Master/Agent mode you have a **site manifest**

An Agent “checks in” with the Master

The Master computes the “catalog” that applies to the Agent

Concept: **Modules**

A Module is a way of distributing and organizing a manifest and support files

A directory structure and convention

<MODULE DIRECTORY>

- *manifests*
- *files*
- *templates*
- *lib*
- *facts.d*
- *examples*
- *spec*

The `modules` directory must contain an `init.pp` manifest

`init.pp` must contain a class with the same name as the module

To use a module, you:

```
include <module_name>
```

This doesn’t include the “module”

It includes a class…

… which puppet finds using the module directories

You can readily bundle files and other resources in modules

Concept: **Environments**

You can create multiple site manifests / sets of modules

When a node registers, it can be assigned an environment

Concept Review

DEMO:

Poking around file structure

Puppet Setup

Standalone

Invoke a manifest explicitly

(with an optional module path)

Pros:

Simple

Cons:

Hard to scale, computation on the client

Master/Agent

Install Puppet Master on a server

Puppet Master has site manifest

(or several if you have multiple environments)

Install Puppet Agent on servers

Puppet Agent gets a node name

(and optionally an environment)

This is based on DNS by default

Each Agent must be registered with the Master and approved

Master can be configured to auto approve

(enter tools like Foreman and IPA)

Puppet Agent then polls for configuration changes on Master

Pros:

Computation on Puppet Master, central copy of configuration

Cons:

Complex, must manage DNS

DEMO:
Puppet In Action

AWS Considerations

AMI Management / Autoscaling

Security Compliance

Control / Team Independence

Questions?

smouring@sequoiainc.com