Configuration management with Chef

Edd Dumbill edd@oreilly.com

OSCON 2009

About me

- Created Expectnation, event software that runs O'Reilly Conferences
- Co-chair of OSCON
- Perennial tinkerer and author (most recently "Learning Rails")

Today's tutorial

- Overview of Chef
- Learn by example
- Common usage patterns
- Moving on

Meta

- Please rate this talk and leave comments

- If you're twittering
  - I'm **@edd**
  - Hashtag is **#oscon**
- Asking questions

About you

Overview

Configuration management

- Creating and maintaining consistency
- Installing, updating, reporting
- Rich history in open source tools
  - *cfengine* through to *Puppet*

Today's needs

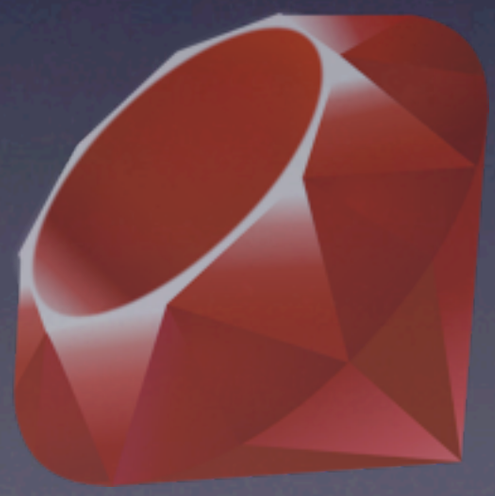
- Developers are becoming ops people
- Web architectures and cloud computing
- Agile sysadmin should complement agile development

Developers want

- Don’t Repeat Yourself
- Revision control

Chef

- Client-server architecture
- Embraces modern web technologies
- Written in Ruby



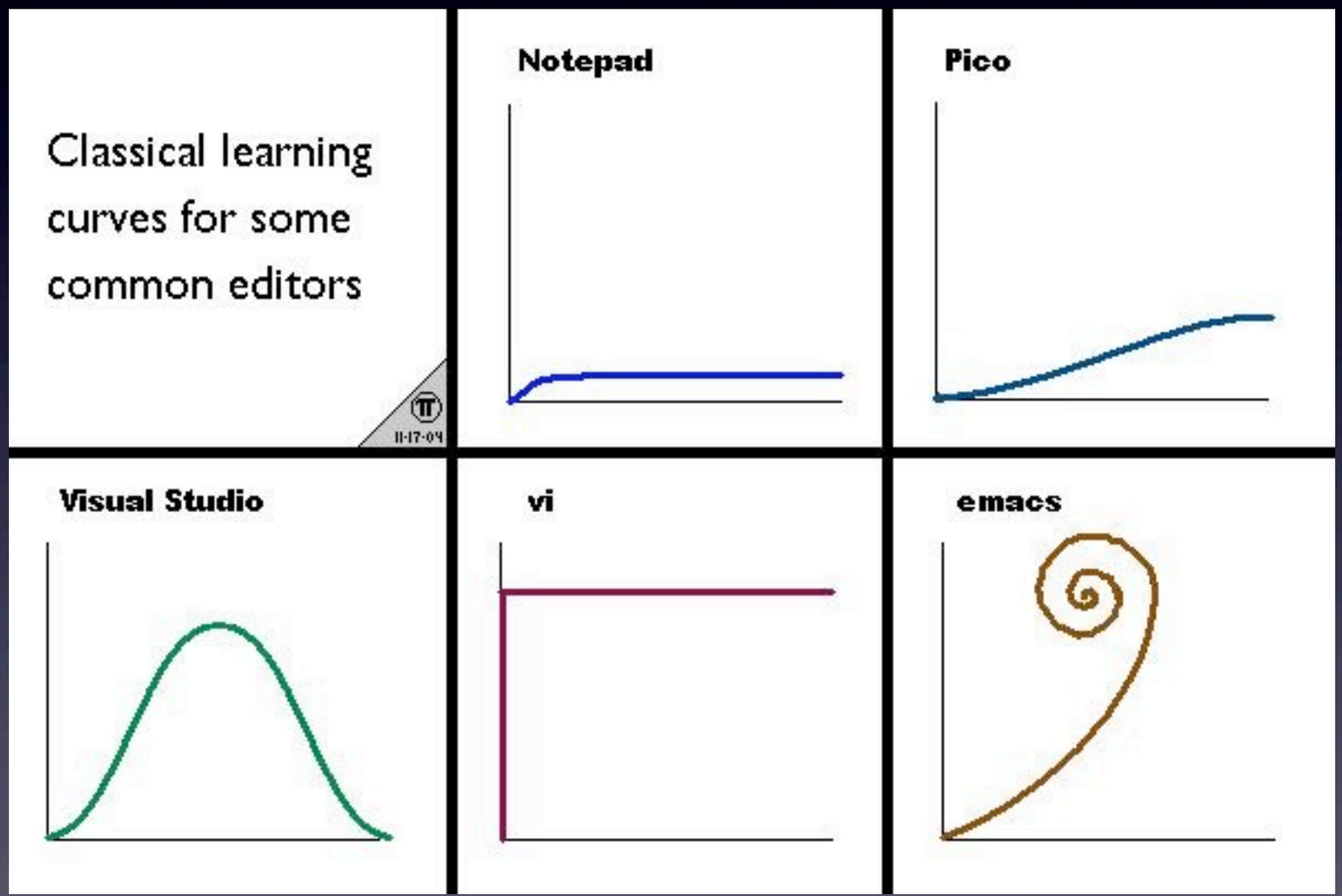
Chef

- Cleeent-serfer ercheetectoore-a
- Imbreces mudern veb technulugeeees
- Vreettee in Rooby
- Bork bork bork

Chef

- Has revision control at its core
- Doesn't make you learn a new language
- Comes from a culture of testability and predictability

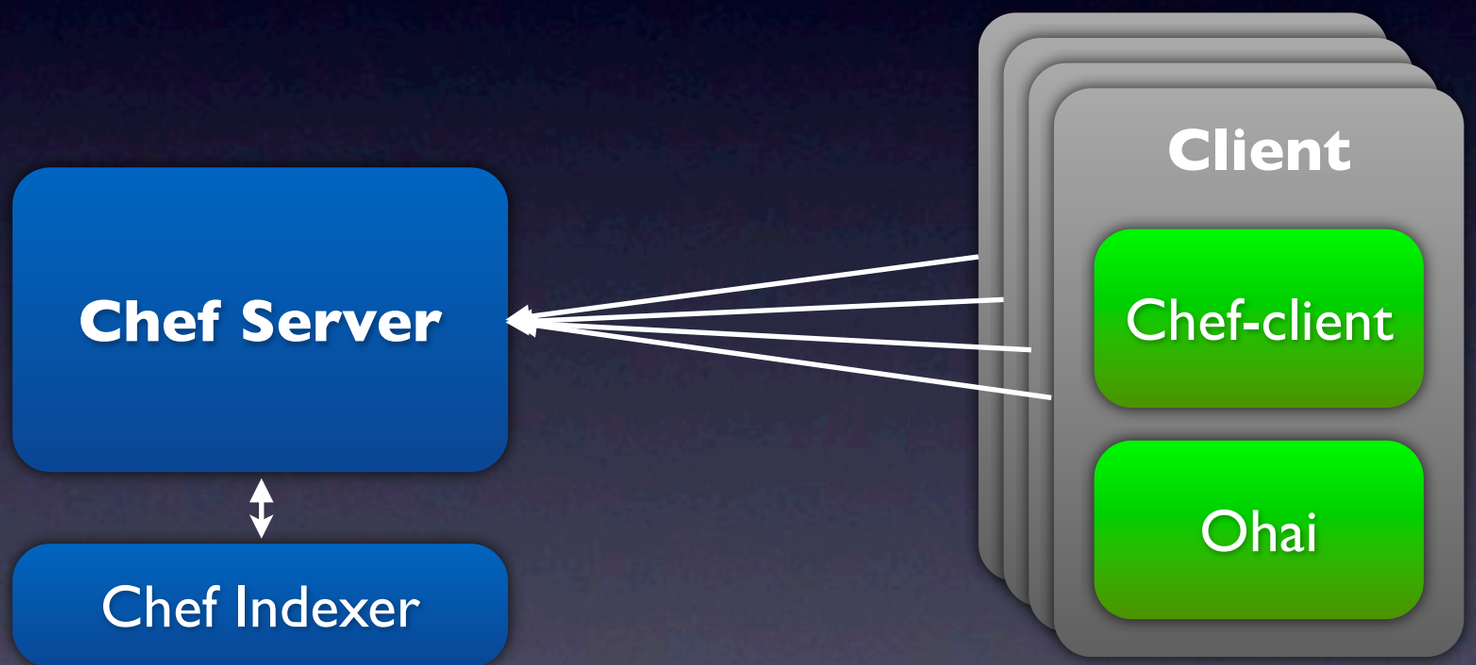
Chef vs Puppet



Chef vs Puppet

- Because we needed another open source war
- Objective differences
- Subjective differences
- Chef has had chance to learn from several years of Puppet

Architecture



Getting started

Assemble your victims

- Use VMs for testing environment
- Ubuntu 8.10 or newer is the sweet spot
- VirtualBox is a free virtualization tool
- Identify a server and one or more clients

Prerequisites

- Two stage install: basics & bootstrap
- Minimal prerequisites: Ruby & RubyGems
- Install via Gems: *ohai* and *chef*
- Bootstrap differs for server and client
- Packages coming soon

Server

- Apache + Passenger
- Provides administrative Web UI
- Users identified by OpenID
- Recipes defined by your *chef repository*

Client

- Invocation of *chef-client*
  - One-time
  - As a daemon

```
chef-client -i 3600 -s 600
```

Chef repository

- Contains configuration and cookbooks
- Clone the Opscode template to start
- Copy your configuration

First look at the server

First client run

Node attributes

- Explore with Web UI
- OS attributes provided by *ohai*
- Other attributes are configured by the installed cookbooks
- Attributes are mutable

Making a cookbook

- Cookbook is the unit of reuse in Chef
- Unsurprisingly, it contains recipes
- Generate one with

```
rake new_cookbook COOKBOOK=hello_world
```

Inside the cookbook

- *metadata.rb* — cookbook metadata
- *attributes* — variables
- *recipes* — list of instructions (“resources”)
- *files* — files used by resources
- *templates* — ERB templates

Inside the cookbook

- *roles* — collections of recipes and attributes
- *definitions* — macros of resources
- *libraries* — Ruby to extend Chef DSL

Metadata

- Functionality similar to metadata in package management
- Human readable docs
- Dependency declarations

Define an attribute

- Simple attribute
  attributes/my_name.rb

```
my_name “John Henry”
```

A simple recipe

- *recipes/default.rb*

```
template “/tmp/hello_world.txt” do
  source “hello_world.txt.erb”
  variables :my_name => node[:my_name]
  mode 00664
  action :create
end
```

The template

- *templates/default/hello_world.txt.erb*

```
Hello, <%= @my_name %>, how are you
today?
```

Running the recipe

- Add the recipe to the node's recipe list
- Invoke *chef-client*
- Default *chef-client* setup has client invoked periodically

When *chef-client* runs

- Node authenticates with server
- Libraries, attributes, definitions & recipes are synchronized
- Libraries, attributes, definitions & recipes compiled
- Node state is converged
- **Everything happens on the node**

Attributes & resources

Attributes

- May be simply defined, e.g.
  `my_name “John Henry”`
- Allow overriding, e.g.
  `my_name “John Henry” unless attribute? (“my_name”)`
- List values are regular arrays
  `[“foo”, “bar”, “whizz”]`

Attribute hashes

- Logical groupings of configuration information, e.g. Apache settings, network interface properties
- Class used is *Mash* (from extlib)
  - so you can use :foo or ‘foo’ as a key

Advanced attributes

- Methods: *attribute?()* & *recipe?()*

- Access to recipes array

```
recipes << “hello_world” unless
recipe?(“hello_world”)
```

Resources

- The steps that make up a recipe

```
package “git-core” do
  action :install
end
```

- Resources are implemented via Providers

Package

```
package "tar" do
  version "1.16.1-1"
  action :install
end
```

- Action can be install, upgrade, remove, purge
- Version is optional

Ruby gems

- Install gems with *package* too

```
package “capistrano” do
  provider
Chef::Provider::Package::Rubygems
end
```

- Easier:

```
gem_package “capistrano”
```

- Can use *source* attribute for gem source

Remote files

- Copying remote files is easy

```
remote_file “/tmp/foo.png” do
  source “foo.png”
  owner “root”
  group “root”
  mode 0444
  action :create
end
```

- Where does the file live?

Search path

- Files and templates are searched for in the following order: FQDN, platform-version, platform, default

- For Ubuntu 9.04:

```
myhost.example.com
ubuntu-9.04
ubuntu
default
```

More remote file fun

- File source can be a URL

```
source “http://warez.com/thing.tgz”
```

- Provide SHA256 hash to prevent needless downloading from chef-server each time

```
checksum “08da0021”
```

Links

- Symbolic or hard links

```
link “/usr/bin/randomthing1.8” do
  to “/usr/bin/randomthing”
end
```

- Use `link_type :hard` or `:symbolic` (default)

File

- Control existence and attributes of a file, not its contents

```
file “/tmp/whatever” do
  owner “root”
  group “root”
  mode “0644”
  action :create
end
```

- Other actions are `touch`, `delete`

Other FS resources

- `directory` — analog of the File resource
- `remote_directory` — recursive remote copy

Service

- Control system services from */etc/init.d* and friends
- We can en/disable, start, stop & restart

```
service “my_daemon” do
  supports :restart => true
  action [ :enable, :start ]
end
```

Other resources

- User
- Group
- Cron
- Route
- Mount

Execute

- Execute arbitrary command

```
command “mysql-stuff” do
  execute “/usr/bin/mysql </tmp/
foo.sql”
  creates “/tmp/outfile.sql”
  environment {‘FOO’ => “bar”}
  action :run
end
```

Script

- bash, perl, python, ruby, csh

```
bash “install_foo” do
  user “root”
  cwd “/tmp”
  code <<-EOC
    wget http://example.org/foo.tgz
    tar xvf foo.tgz && cd foo
    ./configure && make install
  EOC
end
```

HTTP Request

- Useful for connecting to existing services

```
http_request “say_hello” do
  url “http://myserv.local/check_in”
  message :node => node[:fqdn]
  action :post
end
```

- Posts a JSON payload
- GET by default

Resource tricks

Notifies

- Chain actions

```
template “/etc/my_daemon/my.cnf” do
  source “my.cnf.erb”
  notifies :restart,
  resources(:service => “my_daemon”)
end
```

- By default, notification postponed until end of run, add `:immediately` as final argument to override

Action :nothing

- If you want a resource to run only on a notify, specify `action :nothing`

```
execute "index-gem-repository" do
  command "gem generate_index -d /srv/
gems"
  action :nothing
end
```

Conditional resources

- Use *only_if* and *not_if* to control resource execution
- Takes either shell commands or Ruby blocks, e.g.

```
only_if do
  IO.read(“/tmp/foo”).chomp == ‘bar’
end
```

Platform specifics

- Selective resource execution
  `only_if do platform?(“ubuntu”) end`
- Alter package name

```
package "libwww-perl" do
  case node[:platform]
  when "centos"
    name "perl-libwww-perl"
  end
  action :upgrade
end
```

Roles

What roles do

- Bundle recipes and attributes

```
name "webserver"
description "The base role for systems that
serve HTTP traffic"
recipes "apache2", "apache2::mod_ssl"
default_attributes "apache2" =>
{ "listen_ports"=> [ "80", "443" ] }
override_attributes "apache2" =>
{ "max_children"=> "50" }
```

What roles are for

- Convenient way of assigning bundles of functionality to servers
- Allow top-level configuration with minimal need to write new recipes

Creating roles

- Ad-hoc from the Web UI
- As Ruby or JSON from your chef repository

Opscode Cookbook

Opscode cookbooks

- http://github.com/opscode/cookbooks
- Integral part of the Chef project
- If you want it, it's probably already there
  - common configurations
  - smoothing over platform specifics

Using the cookbooks

- Keep your own stuff in *site-cookbooks*
- Use git to add cookbooks as a submodule

```
git submodule add
 git://github.com/opscode/cookbooks.git
 cookbooks
git submodule init
git submodule update
```

3rd party cookbooks

- The *cookbook_path* from the server config specifies precedence
- By default *site-cookbooks* overrides *cookbooks*
- You can adapt recipes simply by replacing the parts you wish

apache2 cookbook

- Attributes configure basic preferences (ports, timeout, keepalive)
- Default recipe sets up sane configuration
- *apache2::* namespace includes recipes for common modules

Overriding attributes

- If you control cookbook, easy enough to set a default
- Per-node customizations can be made in the UI
- To set new defaults, override selectively in *site-cookbooks*

apache2 definitions

- Macro for *a2ensite* & friends

```
apache_site “my_app”
  :enable => true
end
```

- *web_app* — wraps most of the common configuration for a web app (e.g. Rails)

mysql cookbook

- mysql::client, mysql::server
- EC2-aware

Rails cookbook

- Provides installation recipe and attributes for tuning
  - `rails[:version]`
  - `rails[:environment]`
  - `rails[:max_pool_size]`
- Provides *web_app* template you can copy

Chef and Rails

How Chef can help

- Configuration
- Deployment
- Configuration is the better trodden path

Example configuration

- Naive Chef recipe to get all the prequisites in place for an instance of Expectnation

Worked example

- Create and deploy a basic Rails app

chef-deploy

- A resource that implements Rails application deployment
- Models Capistrano's cached_deploy
- In rapid development, used at EngineYard
- http://github.com/ezmobius/chef-deploy

```
deploy "/data/#{app}" do
  repo "git://server/path/app.git"
  branch "HEAD"
  user "myuser"
  enable_submodules true
  migrate true
  migration_command "rake db:migrate"
  environment "production"
  shallow_clone true
  revision '5DE77F8ADC'
  restart_command “...”
  role “myrole”
  action :deploy
end
```

Callbacks

- Ruby scripts in your app's deploy/
- before_migrate, before_symlink, before_restart, after_restart
- Rails environment and 'role' passed as arguments to callback
- Could control this via
  `role node[:myapp][:role]`

Single source for gem dependencies

- Specify gems in *gems.yml* in your app's root

```
- :name: foo
  :version: "1.3"
- :name: bar
  :version: "2.0.1"
```

Deployment strategy

- Unlikely you want deploy to be attempted with the default chef-client behavior
- chef-deploy developed against a Chef Solo world view: explicit execution
- Use attribute to control deployment
- Work in progress

Gotchas

- Chef-deploy assumes shared *config/database.yml*
- Usual package/gem conflicts
- Don't install *rake* from packages! (but cookbooks are getting better at protecting you from this)

Chef Solo

Server-less operation

- Bundle up the cookbooks in a tarball
- Set attributes in a JSON file
- Good to go!

Deploying with solo

- Tar up your cookbooks
- Create a solo.rb

```
file_cache_path “/tmp/chef-solo”
cookbook_path “/tmp/chef-solo/
cookbooks”
```

Deploying with solo (2)

- Create your JSON, e.g.

```
{ “recipes”: “chef-server”,
  “myvar”: “foo” }
```

- Execute

```
chef-solo -c solo.rb -j chef.json
    -r http://path/to/tarball.tgz
```

- JSON path can be URL too

Why Chef Solo?

- When you don't or can't control access to the server
- When clients aren't in the same security zone
- When you care about installation rather than long-term maintenance

REST API

Chef's REST API

- Chef's REST API is pretty mature
- Reused a lot internally
- Best way to programmatically integrate
- Chef wiki carries API examples

What can you do with the API?

- Programmatic access to the server
- Add remove/recipes from nodes
- Interrogate and set attributes
- Perform searches

API authentication

- Register in the same way a node does

```
Chef::Config.from_file(
  “/etc/chef/server.rb”)
@rest = Chef::REST.new(
  Chef::Config[:registration_url])
@rest.register(user, password)
```

- Thereafter, authenticate

```
@rest.authenticate(user, password)
```

Manipulating nodes

```
node = @rest.get_rest("nodes/
foo_example_com")

puts node.recipes.inspect
node.recipes << "apache2"

puts node[:myattr].inspect
node[:myattr] = { :foo => "bar" }

@rest.put_rest("nodes/foo_example_com",
node)
```

Knife

- Basic command line interface to the server
- For now, get from http://gist.github.com/104080

Searching

Searching the server

- Powerful feature
- Not that mature yet
- Ferret indexes the Chef Server database
- Queries expressed in FQL

Access from recipes

- search(*INDEX, QUERY*)
- `search(:node, “*”)` reports every node in the DB
- Find the IP of every node running Apache

```
search(:node,
“recipe:apache2”).collect {|n|
n[‘ipaddress’]}
```

Access from REST API

- As implemented in the Web UI

```
@rest.get_rest(
  "search/node?q=recipe:apache2")
```

Development patterns

Git strategy

- Use submodules to bring in 3rd party cookbooks
- Develop against testbed, push to shared repository
- Server install rule does a *git pull*

VM testbed

- Use a VM tool that supports snapshotting
- VirtualBox is free
- VMware good, supported by Poolparty
- Use Avahi/Bonjour for convenience

Use roles

- Allow site-wide customization
- Bundling your configuration with choice of cookbooks
- Recipes can then implement control inflexion points using attributes

Refactor into definitions & attributes

- For maintainability, consider refactoring obvious components into definitions
- e.g. the directory creation stage of a Rails app (what *cap deploy:setup* does)

Chef & EC2

In OpsCode cookbooks

- ec2 cookbook
- EC2 awareness in, e.g. mysql recipes
- Bunch of handy EC2 attributes exposed

Chef AMIs

- Work in progress
- Preconfigured Ubuntu with chef-client and/or server
- Chef attributes sent as instance data
- Chef wiki has worked EC2 + Rails architecture

Poolparty

- Configure and deploy to the cloud
- Uses Chef
- http://poolpartyrb.com/

What Poolparty does

- Launches VM (EC2 or VMware), waits for IP and ssh
- Bootstrap: rsyncs dependencies and installs
- Configure: compile cookbooks, rsyncs, executes Chef Solo
- Verifies installation

Community resources

- Wiki is a great and ever-improving reference
  http://wiki.opscode.com/display/chef/Home
- IRC
  irc://irc.freenode.net/chef
- Mailing list

The future

- Chef is evolving rapidly
- Platform support improving through contributions
- Opscode-agent
  - nanite
  - selective resource execution

In conclusion

- Please rate this tutorial and leave comments http://bit.ly/chef-oscon
- Q&A
- Thank you!