2011-05-16

Introduction to Chef

Igal Koshevoy, Pragmaticraft
Business-Technology Consultant
igal@pragmaticraft.com
@igalko

Who am I?

- Sysadmin turned programmer turned business-tech consultant.
- Nearly 20 years of experience using and writing automation tools. Variety of self-made & in-house tools, plus AutomateIt, Cfengine, Puppet, Chef.
- Please ask questions -- I can forget that this isn't obvious day-to-day stuff to everyone. :)

This talk is about:

- What is Chef and why use it?
- How to use Chef's features?
- How to use `chef-solo` deployment tools and 3rd party code?

What is Chef?

- Open source product from Opscode: http://www.opscode.com/
- Tool for automating the setup and maintenance of your computers.
- Language for describing a configuration, the intended state of a computer
- Tools for applying configurations to computers, bringing them to the intended state.

Why Chef?

- Used by many companies for large projects: 37signals, Engine Yard, IGN Entertainment, RightScale, Scribed, WebTrends, etc.
- Lots of open sourced configuration code you can easily reuse
- Easy things are easy, hard things are do-able
- Good reference documentation
- Nice configuration language, which supports high-level programming using Ruby (can also be considered a drawback)
- Nice structure for organizing configuration data
- Nice system for separating your custom attributes describing your site from other's generic, reusable code
- Simple, but effective dependency model
- Less verbose than many alternatives
- Choice of client-server or standalone (`chef-solo`) operation
- Powerful network interactions using shared database (databag)

Why not Chef?

- Something else may make more sense to you. :)
- In 0.10, its hard to tell what it did because its logging is either is too verbose or not verbose enough
- As of 0.10, still no databag support in `chef-solo`, must write code differently for attributes and databags
- No reporting or aggregation of what was done
- No dry-run/preview/noop mode
- No built-in exception handling, must add it yourself
- Not packaged by as many operating system distributions as some more established alternatives
- Supports fewer platforms
- Requires modern Ruby and RubyGems

Descriptions and reuse

UNIX shell commands describe actions. Unless wrapped in "`if`" statements, they can only be run once on a computer, e.g. "`mkdir`" will fail because directory already exists:

```
mkdir /tmp/mydir
chown root:root /tmp/mydir
chmod 0755 /tmp/mydir
```

Chef recipes describe resources and their intended state. These can be reapplied to a computer because they'll only be changed if needed, e.g. directory will only be created if needed:

```
directory "/tmp/mydir" do
  owner "root"
  group "root"
  mode 0755
  action :create
end
```

Cookbooks

...are collections of related configuration data. They're really just directories with this structure, although they may have more or less files:

```
mycookbook/
    attributes/
        default.rb
    templates/
        default/
            mytemplate.erb
    recipes/
        default.rb
        myrecipe.rb
    definitions/
        default.rb
```

Nodes

...are computers that you apply Chef configurations to. These are either managed using `chef-client` for the client-server mode, or `chef-solo` in standalone mode. E.g.: for `chef-solo`:

```
# nodes/mynode.json
{
  // Apply these cookbook recipes, roles, etc.
  "run_list": [
    "recipe[ntp]", "role[web_server]",
       "role[db_server]"
  ],
  // Attributes overriding those in cookbooks
  "ubuntu": {
    "security_url":
      "http://ubuntu.osuosl.org/ubuntu/",
    "archive_url": "http://ubuntu.osuosl.org/ubuntu/"
  }
}
```

Resources

...are cross-platform abstractions representing things you configure on a node. E.g. files, directories, packages, etc.

```
# A package named "apache2" should be installed
# using the default package manager
package "apache2" do
  action :install
end
```

Providers

...are platform-specific implementations of the things resources abstract. E.g. APT provider for the package resource:

```
# Specify provider
package "apache2" do
  action :install
  provider Chef::Provider::Package::Apt
end

# Or use shortcut
apt_package "apache2" do
  action :install
end
```

Package resource

...manages packages: APT, DEB, YUM, RPM, Gem, etc.

```
package "mypackage" do
  # Or :upgrade, :remove, :upgrade, :purge
  action :install # Optional, defaults to this
  version "1.23" # Optional, defaults to latest
  response_file "myresponsefile" # Optional
  source "myfile" # Optional if has location
end
```

Directory resource

...manages directories.

```
directory "/tmp/mydirectory" do
  # Or :delete
  action :create # Optional, defaults to this
  owner "root" # Optional
  group "root" # Optional
  mode 0755 # Optional
end
```

User resource

...manages users.

```
user "myusername" do
  # Or :remove, :modify, :manage, :lock, :unlock
  action :create # Optional, defaults to this
  uid 1234 # Optional
  gid 1234 # Optional
  home "/home/myusername" # Optional
  shell "/bin/bash" # Optional
  # Optional, use MD5 `makepasswd`
  password "$1$JJsvHslV$szsCjVEroftprNn4JHtDi."
end
```

Execute resource

...executes code. The `not_if` block tells Chef to only execute the command if needed. Chef runs the recipe's code to define resources, and actually applies them much later – these two `puts` statements will happen at very different times.

```
puts "Seen during definition time!"
execute "my_exec_resource_name" do
  puts "Seen during apply time!"

  # Arbitrary UNIX shell code in command:
  command "cd /tmp && mkdir /tmp/hello"

  # Only run command if this is false (no directory)
  not_if do # There's also a `only_if`.
    # Custom Ruby code to check state,
    # returns true if directory exists
    FileTest.directory? "/tmp/hello"
  end
end
```

Remote file resource

...downloads a remote file if needed.

```
remote_file "/tmp/myfile" do
  source "http://foo.bar/myfile"
  action :create # or :create_if_missing
  checksum "08da002l" # Optional, SHA256 of the file
  owner "root" # Optional
  group "root" # Optional
  mode 0755 # Optional
  only_if do # Optional
    ...
  end
end
```

Template resource

...generates files from an ERB template.

Resource in `mycookbook/recipes/default.rb`

```
template "/tmp/myoutput" do
  source "t.erb"
  variables :entity => "World"
end
```

Template in `mycookbook/templates/default/t.erb`

```
Hello <%= @entity %>!
```

Produces `/tmp/myoutput` with content:

```
Hello World!
```

Many other built-in resources

- cron
- deploy
- file
- git & subversion
- group
- http
- ifconfig
- link
- log
- mdadm
- mount
- remote_directory
- route
- script
- service

Dependencies, implicit

...these execute in linear order, "`/tmp/a`" then "`/tmp/b`":

```
directory "/tmp/a"

directory "/tmp/b"
```

Dependencies, explicit

...these execute in reverse order, "/tmp/b" then "/tmp/a":

```
directory "/tmp/a" do
  action :nothing # Don't create yet!
end

directory "/tmp/b" do
  action :create
  # Create the other directory when done
  notifies :create, "directory[/tmp/a]"
end
```

Dependencies, delayed

...these restart the service "apache2" after everything is done and only once:

```
directory "/tmp/a" do
  notifies :restart, "service[apache2]", :delayed
end

directory "/tmp/b" do
  action :create
  notifies :restart, "service[apache2]", :delayed
end
```

Dependencies, cookbook recipes

...these make sure that `apache2` (the `apach2` cookbook's default recipe) and `php5::php5-cgi` (the `php5` cookbook's `php-cgi` recipe) are applied before "`/tmp/a`". These other recipes will only be applied once, even if required by other recipes, roles, or your node's `run_list`:

```
require_recipe "apache"
require_recipe "php5::php5-cgi"

directory "/tmp/a"
```

Attributes

...describe variables inside generic code that can be overridden externally. This is fantastic for reusing generic cookbooks. E.g.

Default attributes in `mycookbook/attributes/default.rb`

```
node[:mycookbook][:dir] = "/tmp/mydir"
```

Recipe using these in `mycookbook/recipes/default.rb`

```
directory node[:mycookbook][:dir] # Looks up value
```

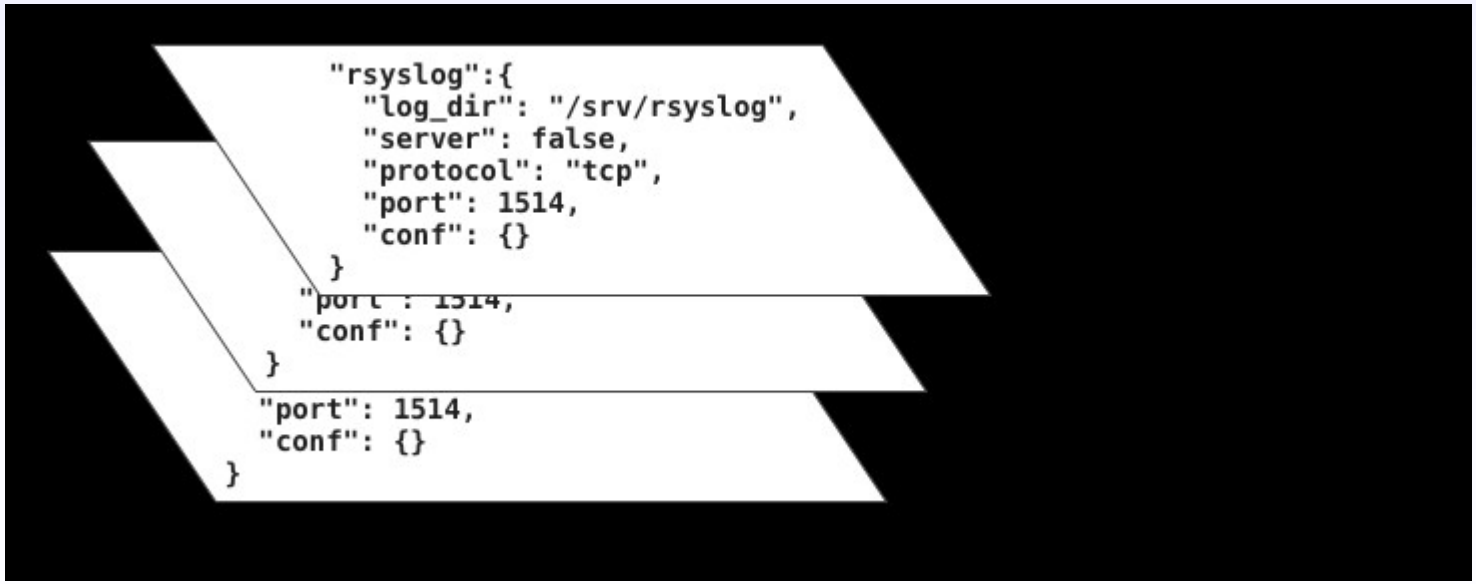
Override attributes for a node in `nodes/mynode.json`

```
{
  "run_list": [ "mycookbook" ],
  "mycookbook": {
    "dir": "myotherdir"
  }
}
```

Attribute precedence

Attributes	File	Role	Environment	Node or Recipe
Default	1	2	n/a	3
Normal	4	n/a	n/a	5
Override	6	7	8	9

Largest number has highest precedence



Definitions

...describe new resources by using existing resources. E.g.

Define in `mycookbook/definitions/default.rb`

```
define :hello, :greeting => "Hello" do
  unless params[:name]
    raise "hello: No entity specified!"
  end

  file "/tmp/hello" do
    # E.g. "Hello World!"
    content "#{params[:greeting]} #{params[:name]}!"
  end
end
```

Use in recipe `mycookbook/recipes/default.rb`

```
hello "World" do
  greeting "Greetings" # Produces "Greetings World"
end
```

LWRP

...are lightweight resources and providers. This is a simple Ruby DSL (domain specific language) for creating your own resources and providers.

```
# Resource: mycookbook/resources/database.rb
actions :create
attribute :name, :kind_of => String, :name_attribute => true
attribute :type, :kind_of => String
```

```
# Provider: mycookbook/providers/mysql.rb
action :create do
  execute "create database" do
    not_if "mysql -e 'show databases;' | " +
      "grep #{new_resource.name}"
    command "mysqladmin create #{new_resource.name}"
  end
end
```

```
# Use: mycookbook/recipes/default.rb
database "mydatabase" do
  action :create
  provider "mysql"
end
```

Roles

...describe sequences of other roles, cookbooks and recipes to apply. E.g.

```
# roles/ntp_client.json
{
  "name": "ntp_client",
  "chef_type": "role",
  "json_class": "Chef::Role",
  "run_list": [
    "recipe[ntp]"
  ],
  "override_attributes": {
    "ntp": {
      "servers": ["0.us.pool.ntp.org", "1.us.pool.ntp.org"]
    }
  }
}
```

Reusable cookbooks

Don't write your own! Reuse when possible:

- http://community.opscode.com/cookbooks
- https://github.com/opscode/cookbooks
- https://github.com/37signals/37s_cookbooks
- https://github.com/engineyard/ey-cloud-recipes

chef-solo deployers

Create directory structure on management workstation, where you describe your nodes, roles, cookbooks, etc. Then use the tool to deploy to remote nodes.

- pocketknife
  `https://github.com/igal/pocketknife`
- littlechef
  `https://github.com/tobami/littlechef`

Questions & Answers

Igal Koshevoy
Biz-Tech Consultant
igal@pragmaticraft.com
@igalko