OPSCODE

Understanding LWRP

Speaker:

Joshua Timberman Technical Evangelist

- joshua@opscode.com
- @jtimberman
- www.opscode.com

remote_file

link

cookbook_file

service

ruby_block

template

execute

Chef manages Resources on Nodes

bash

package

git

log

deploy

user

http_request

Tuesday, November 2, 2010

Resources are an abstraction we feed data into. When you write recipes in Chef, you create resources of things you want to configure. You describe the state of some part of the system. We're going to talk about how this works in depth.

```
gem_package "bluepill"
```


Tuesday, November 2, 2010

A very simple resource, simply tells Chef to install the bluepill package as a RubyGem.

```
package "bluepill" do
  provider Chef::Provider::Package::Rubygems
end

package "bluepill" do
  provider gem_package
end
```


Tuesday, November 2, 2010

Likewise, these are both the equivalent of the previous gem_package, we're just specifically telling the package resource to use a particular provider. We'll see why this matters later.

```
template "/etc/bluepill/dnscache.pill" do
  source "dnscache.pill.erb"
  mode 0644
end
```

Another somewhat simple resource, specify a file to be written somewhere as a template.

```
bluepill_service "dnscache" do
  action [:enable,:load,:start]
end
```


Tuesday, November 2, 2010

Here's a custom resource that specifies actions to run. We'll talk in a bit how this works.

Resources take *action* through Providers

Tuesday, November 2, 2010

The abstraction over the commands or API calls that will configure the resource to be in the state you have defined.

```
service "dnscache" do
  provider bluepill_service
  action [:enable, :load, :start]
end
```

Chef's Recipe DSL creates resources

Recipes evaluated in two phases

Compile Phase
Execution Phase

Tuesday, November 2, 2010

Going to leave out some of the details. The rest is in the code. :)

Compile Phase

Resources added to ResourceCollection

Tuesday, November 2, 2010

When recipes are processed for Resources during the compile phase, they are added to the resource collection. These can include..

Chef::ResourceCollection

Resources and definitions
Hash of indexed resources
Definitions are replaced

Tuesday, November 2, 2010

Chef “recognizes” resources and definitions through method_missing. These are created as a hash of numerically indexed resources. Definitions are replaced entirely by the resources they contain.

```
@resources_by_name={"file[/tmp/something]"=>0}
```


Tuesday, November 2, 2010

essential what the Resource Collection looks like.

Chef::RunContext

Loads cookbook components

- Libraries
- Providers
- Resources
- Attributes
- Definitions
- Recipes

Tuesday, November 2, 2010

The order is important because of where you can put things, and when things are available.

Execution Phase

Tuesday, November 2, 2010

Once everything is loaded up and the recipes have been parsed for the resources they contain, the runner starts the execution phase.

Chef::Runner converges the node

runs the provider actions specified by the resource

Chef::Resource calls run_action

Tuesday, November 2, 2010

run_action calls the method for that particular action in the provider

Provider is chosen by Chef::Platform

Tuesday, November 2, 2010

This is how platform specific providers are chosen like apt/yum, useradd, etc. It also contains logic to get the provider for the resource..

Chef::Platform finds the provider

Specifically named parameter
Platform assigned providers
Matching the resource name

Tuesday, November 2, 2010

Specifically named providers are like we saw earlier where the resource itself had the provider parameter.
Platform assigned are the ones like apt/yum for packages.
Matching the resource name is how LWRPs get chosen.

Cookbook LWRP

Tuesday, November 2, 2010

LWRPs, or resources and providers written in this special DSL, reside in cookbooks.

Ruby DSL for writing Resources & Providers

Tuesday, November 2, 2010

All LWRPs are, a Ruby DSL for writing resources and providers. This DSL is designed to be easier to write and understand than the fully resource/provider code in the code included by the Chef libraries.

Resources have states

Resources have states

Resource DSL describe states

Two possible states

- Current (`@current_resource`)
- Desired (`@new_resource`)

Providers `load_current_resource`

Tuesday, November 2, 2010

The resource describes the state that some aspect of the system should be in by passing parameters that configure it for that state.
Current and desired are the two states that a resource can be in.
Providers load_current_resource determines what state the resource is in.

```ruby
def load_current_resource
  @bp = Chef::Resource::BluepillService.new(new_resource.name)
  @bp.service_name(new_resource.service_name)

  Chef::Log.debug("Checking status of service #{new_resource.service_name}")

  begin
    if run_command_with_systems_locale(:command => "#{node['bluepill']['bin']} status #
{new_resource.service_name}") == 0
      @bp.running(true)
    end
  rescue Chef::Exceptions::Exec
    @bp.running(false)
    nil
  end

  if ::File.exists?("#{node['bluepill']['conf_dir']}/#{new_resource.service_name}.pill")
    && ::File.symlink?("#{node['bluepill']['init_dir']}/#{new_resource.service_name}")
    @bp.enabled(true)
  else
    @bp.enabled(false)
  end
end
```

Tuesday, November 2, 2010

The code is not important. What is important that we're checking for some various states on the system by running commands or code and setting "state" parameters.

Resource DSL

```
actions :start, :stop, :enable, :disable, :load, :restart

attribute :service_name, :name_attribute => true
attribute :enabled, :default => false
attribute :running, :default => false
attribute :variables, :kind_of => Hash
attribute :supports, :default => { :restart => true, :status => true }
```

Actions correspond to ‘action’ methods in the provider.
Attributes correspond to methods for the resource. These are also called parameters and are not node attributes.

Provider DSL

```
action :start do
  unless @bp.running
    execute "/usr/bin/bluepill start #{new_resource.service_name}"
  end
end
```

Tuesday, November 2, 2010

The provider DSL at a minimum has action methods. The action methods handle doing whatever is needed to configure the resource to be in the declared state.
Earlier we set @bp_running to true or false depending on the current state, here we check.

```
actions :start, :stop, :enable, :disable, :load, :restart

attribute :service_name, :name_attribute => true
attribute :enabled, :default => false
attribute :running, :default => false
attribute :variables, :kind_of => Hash
attribute :supports, :default => { :restart => true, :status => true }
```

```
action :start do
  unless @bp.running
    execute "/usr/bin/bluepill start #{new_resource.service_name}"
  end
end
```

Lets break down the provider's methods

```
actions :start, :stop, :enable, :disable, :load, :restart

attribute :service_name, :name_attribute => true
attribute :enabled, :default => false
attribute :running, :default => false
attribute :variables, :kind_of => Hash
attribute :supports, :default => { :restart => true, :status => true }


def load_current_resource
  @bp = Chef::Resource::BluepillService.new(new_resource.name)
  @bp.service_name(new_resource.service_name)

  Chef::Log.debug("Checking status of service #{new_resource.service_name}")

  begin
    if run_command_with_systems_locale(:command => "#{node['bluepill']['bin']} status #
{new_resource.service_name}") == 0
      @bp.running(true)
    end
  rescue Chef::Exceptions::Exec
    @bp.running(false)
    nil
  end

  if ::File.exists?("#{node['bluepill']['conf_dir']}/#{new_resource.service_name}.pill")
    && ::File.symlink?("#{node['bluepill']['init_dir']}/#{new_resource.service_name}")
    @bp.enabled(true)
  else
    @bp.enabled(false)
  end
end
```

```
action :start d
  unless @bp.running
    execute "/usr/bin/bluepill start #{new_resource.service_name}"
  end
end
```

Chef Resources!

New resource parameters

```
action :start do
  unless @dbp.running
    execute "/usr/bin/bluepill start #{new_resource.service_name}"
  end
end
```


You can use chef resources inside LWRPs, too.

The goal: Replace definitions with LWRPs

LWRPs in Opscode Cookbooks

- aws_ebs_volume
- aws_elastic_ip
- bluepill_service
- daemontools_service
- dynect_rr
- mysql_database
- pacman_aur
- pacman_group
- samba_user

Tuesday, November 2, 2010

built from the name of the cookbook and the ruby file in the resources/providers directory. these are the aws, bluepill, daemontools, dynect, mysql, pacman and samba cookbooks respectively.