**Just Enough Ruby For Chef**

Section Objectives:

* Learn where Ruby is installed
* Understand basic Ruby data types
* Understand some of the common Ruby objects used in Chef
* Familiarity with the ways Chef uses Ruby for DSLs

.notes These course materials are Copyright © 2010-2012 Opscode, Inc. All rights reserved. This work is licensed under a Creative Commons Attribute Share Alike 3.0 United States License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/3.0/us>; or send a letter to Creative Commons, 171 2nd Street, Suite 300, San Francisco, California, 94105, USA.

**What is Ruby?**

Ruby is an object oriented programming language.

The most common implementation is MRI, Matz Ruby Interpreter, named after the language's inventor.

This section does not comprehensively cover Ruby. It will familiarize you with the syntax and idioms used by Chef.

**How does Chef Use Ruby?**

Chef uses Ruby to construct "Domain Specific Languages" that are used for managing infrastructure.

* Recipes
* Roles
* Metadata
* Plugins (knife, ohai)

**Where is Ruby Installed?**

When installing Chef with the full-stack installer, Ruby is installed as well. The binaries for Ruby (mainly ruby and irb) are included in the package.

On Unix/Linux systems:

* /opt/opscode/embedded/bin

On Windows systems:

* C:\opscode\embedded\bin

**Variables**

Ruby can assign local variables to various built-in Ruby data types, or from other expressions. Variable names start with a letter and can contain alphanumeric characters and underscore.

@@@ ruby

my\_number = 3

floating\_point = 3.14159

a\_string = "I like Chef!"

**Ruby Data Types**

* Strings
* Numbers
* Arrays
* Hashes
* Symbols

**Strings**

Strings are bytes of characters enclosed in quotes, either double or single. Strings in double quotes allow further substition than single quoted strings.

Strings are the most common Ruby data type used in Chef.

@@@ ruby

"This is a string."

'This is another string.'

**Code Substitution**

Within a string, code can be substituted with the #{} notation.

@@@ ruby

code\_sub = "code substitution"

"This is a string with #{code\_sub}"

This is often done in Chef to use node attributes:

@@@ ruby

"#{node['my\_package']['dir']}/my\_package.conf"

**Numbers**

Ruby supports integers an floating point numbers.

@@@ ruby

cpus = 2

mem\_in\_gb = 3.14159

**Arrays**

Ruby *Arrays* are lists of elements. They are ordered by insertion and each element can be any kind of Ruby object, including numbers, strings, other arrays, hashes and more.

Use square brackets to enclose arrays.

@@@ ruby

[ "apache", "mysql", "php" ]

[ 80, 443, 8080 ]

We can use the %w{} shortcut to write array of strings without the quotes and commas.

@@@ ruby

%w{ apache mysql php }

**Hashes**

Ruby *Hashes* are key/value pairs. The key can be a "string" or :symbol. The value can be any Ruby object, including numbers, strings, arrays, hashes and more. Specify values for each key with =>, and separate them with comma.

Use curly braces to enclose hash key/value pairs.

@@@ ruby

{

"site" => "opscode.com",

"ports" => [ 80, 443 ]

}

**Symbols**

Ruby has a special data type called symbols. They are specified by prefixing a string with a colon.

@@@ ruby

:thing

Symbols are commonly used as hash keys instead of strings because they are often more memory efficient.

@@@ ruby

{

:site => "opscode.com",

:ports => [ 80, 443 ]

}

**True, False and Nil**

In Ruby, only nil and false are false.

@@@ ruby

true # => true

false # => false

nil # => nil

0 # => 0 ( 0 is the integer 0, not false )

**Conditionals**

Ruby supports common types of logic conditionals.

* if/else
* unless
* case

**If/Else and Unless**

Just like other languages, if, else and unless statements test boolean values of true or false.

@@@ ruby

if node['platform'] == "ubuntu"

# do ubuntu things

end

unless node['platform'] == "ubuntu"

# don't do ubuntu things

end

**Case**

Case statements can be used as well.

@@@ ruby

case node['platform']

when "debian", "ubuntu"

package "apache2"

when "centos", "redhat"

package "httpd"

end

**Methods**

Ruby methods are called on an object with the dot-notation.

@@@ ruby

"I like Chef".gsub(/like/, "love") # => "I love Chef"

FileTest.exists?("/etc/passwd") # => true

1.even? # => false

2.even? # => true

Ruby has a special method available called method\_missing. It is called when a method is not found for the object. Most of the DSLs in Chef are written using method\_missing.

.notes We will see the actual DSLs in their relevant sections.

**Blocks**

Ruby blocks are code statements between braces or a do/end pair.

@@@ ruby

my\_array.each {|i| puts i}

my\_array.each do |i|

puts i

end

Common convention is to use braces for a single line, and do/end for multiple lines.

**Enumerables**

Array and Hash mix-in the Enumerable class. It contains a number of helper methods, such as .each or .map that are particularly useful in Chef.

@@@ ruby

%w{ apache mysql php }.each do |pkg|

package pkg do

action :upgrade

end

end

We do this often in Chef to handle creating the same kind of resource without having to type the resource multiple times.

**Where does Chef use Ruby?**

Chef uses Ruby for a number of Domain Specific Languages.

* Recipes
* Roles
* Cookbook Metadata
* Environments

.notes This is an overview, not a comprehensive section on these topics, they have their own corresponding sections.

**Chef Ruby Objects**

We use a number of Chef's Ruby objects within Recipes. The three most common objects are:

* Chef::Node, via node
* Chef::Config, a hash-like structure containing configuration.
* Chef::Log, send log messages

**Chef::Node**

The node object is available anywhere Ruby is used. Attributes are accessed like Ruby hash keys:

@@@ ruby

node['platform']

node['fqdn']

node['kernel']['release']

Some parts of the node object are accessed with method calls.

**Chef::Config**

The Chef::Config object is available within recipes so behavior can be modified depending on how Chef itself is configured.

Commonly, we use Chef::Config[:file\_cache\_path] as a "temporary" location to download files such as software tarballs.

@@@ ruby

remote\_file "#{Chef::Config[:file\_cache\_path]}/mystuff.tar.gz" do

source "http://example.com/mystuff.tar.gz"

end

**Chef::Config**

Since chef-solo behaves differently, it may be desirable to account for it in recipes, particularly those that use Chef Server-specific features such as search.

The value Chef::Config[:solo] will only be true if Chef was invoked with chef-solo.

@@@ ruby

unless Chef::Config[:solo] # if we're not using solo...

results = search(:node, "role:webserver") # perform search

end

**Chef::Log**

Log messages using Chef's logger can be displayed with Chef::Log. The different levels of log output are specified by calling the appropriate method.

Chef::Log.info("INFO level message")

Chef::Log.debug("DEBUG level message")

**Summary**

* Learn where Ruby is installed
* Understand basic Ruby data types
* Understand some of the common Ruby objects used in Chef
* Familiarity with the ways Chef uses Ruby for DSLs

**Additional Resources**

The primary site for Ruby is maintained by the Ruby Community:

* [http://ruby-lang.org](http://ruby-lang.org/)