

Welcome to Just Enough Chef!



Chef

Based on: Chef Fundamentals OPS150-04.01 (3 Day Course)
Created and Sponsored by Opscode, Inc.
Adapted by Brad Knowles

Logistics

- Start and finish time
- Breaks
- Lunch
- Restrooms
- Refreshments
- Parking

Instructor

- Name: Brad Knowles
- Company: ihiji, Inc.
- Experience: System Administrator/Engineer/Consultant: 20+ years
 - Chef user: since 2011-09-14
- Contact
 - E-mail: bknowles@ihiji.com
 - Twitter: [@bradknowles](https://twitter.com/bradknowles)
 - github: [bknowles](https://github.com/bknowles)

Students

- Name
- Company / Group
- Experience w/ Configuration Management or Chef itself
- Objective for course

Course Objectives

- Understand Chef's tools and architecture
- Obtain, create and modify cookbooks
- Work with the Chef Server API
- Understand common patterns used by Opscode
- Understand Opscode's products

Chef is a complex system and we could spend 2 weeks on it and not cover everything. This course aims to teach students the fundamentals and most important topics, terminology and common hurdles.

Topics Not Covered

We can't cover everything. Specifically, this course does not cover:

- Direct comparison to other tools.
- In depth details on advanced topics.
- Discussion of application deployment strategies.
- Setup/management of Open Source Chef Server.

Learning Chef

This course is to facilitate learning Chef.

We will take frequent breaks and do some hands on exercises.

Please keep questions on topic for the section. Some questions will cover subject material that will be covered later.

Extensive or detailed questions can be answered at the end of the relevant section, day or at the end of the course.

Agenda

- Introduction
- Getting Started
- Anatomy of a Chef Run
- Just Enough Ruby for Chef
- Cookbooks, Recipes and Resources

Agenda

- Troubleshooting
- Chef Node
- Roles
- More Cookbooks
- Multiple Nodes and Search
- Additional Topics
- Chef Development and Further Resources

Course Artifacts

At the end of the class you will have:

- Opscode account and Opscode Hosted Chef organization (Chef Server)
- Workstation setup to work with Opscode Hosted Chef
- Repository that can be used to get started managing infrastructure as code with Chef

About Best Practices

This course encompasses what are considered "best practices".

Many things in Chef have multiple approaches (TIMTOWTDI). We will focus on one, but may mention others for awareness.

Best practices themselves are subjective.

Chef is flexible and you can make it do almost anything you want.

About the course materials

These training materials are dual-licensed by Opscode.

- Creative Commons Attribution-ShareAlike (CC BY-SA) for slides, guides and notes.
- Apache License, Version 2.0 for supporting code and significant example code on slides.

Introduction to Chef

- Configuration Management
- System Integration
- Core Principles

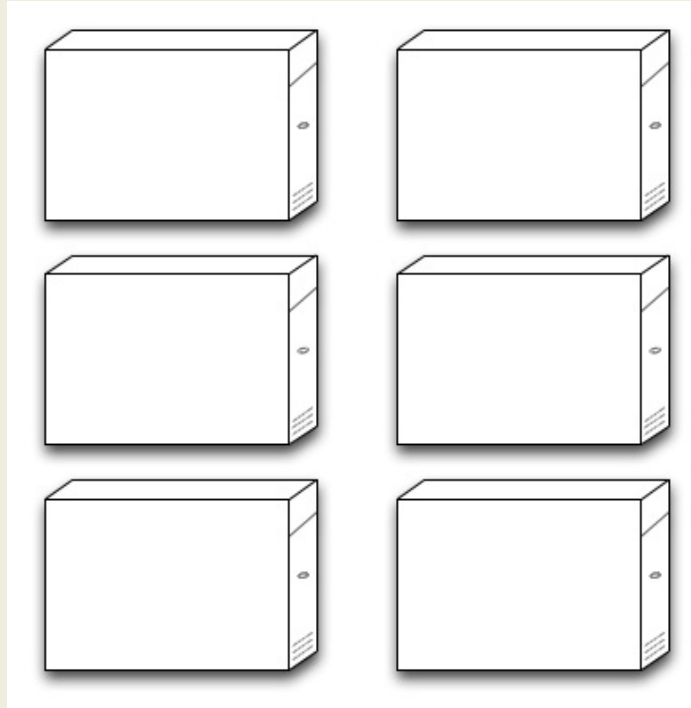
Configuration Management

"Keep track of all the stuff you do to take a system from 'bare metal' to 'doing its job'." - Adam Jacob, Web Operations (O'Reilly, 2010)

- Hand-written log
- Wiki notes, copy/paste
- Scripting, ssh-in-a-for-loop
- Automation frameworks

Bare Metal

We have computers sitting in a rack somewhere.



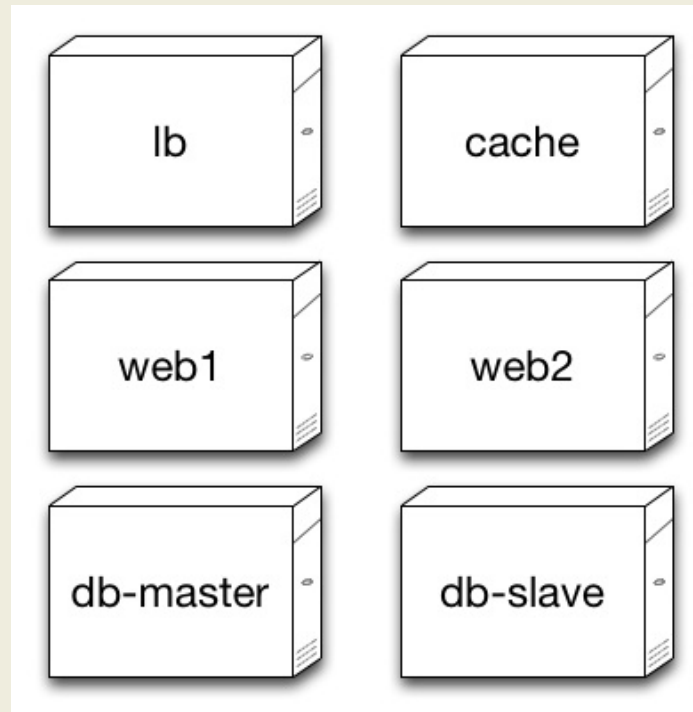
Bare Metal...Cloud?

Or, we have an idea of what computers we need running in a cloud somewhere.



Doing their Job

All the configuration management has been done, now they're doing their jobs.



System Integration

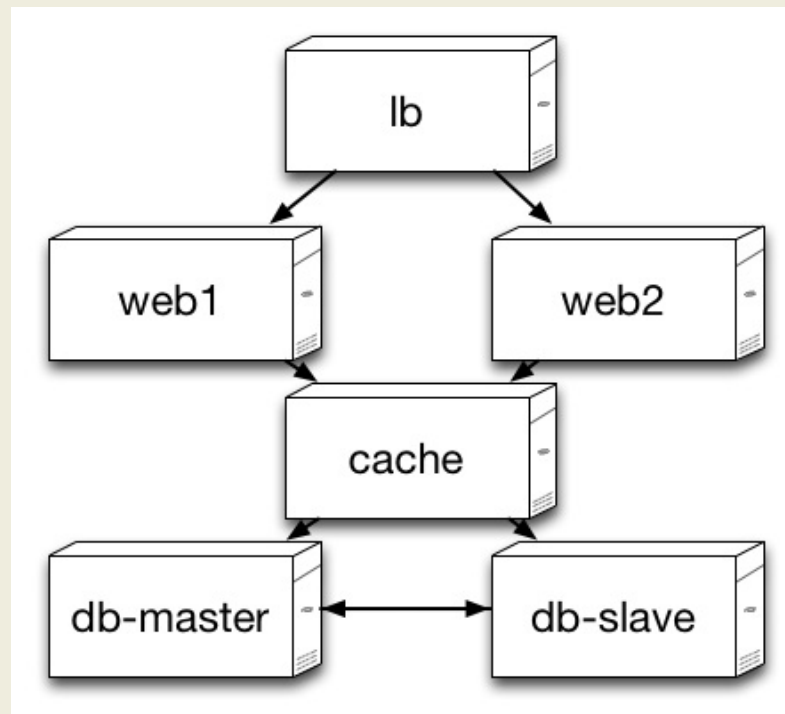
Systems simply running with the right software don't provide value to the business.

They need to be integrated together.

- Load balancers connect to the right web servers.
- Web servers hit the database or the cache layer.
- Database servers are clustered for availability.

System Integration

In a typical architecture, this is complex. Six systems talk to each other, and two of those are a complex subsystem of their own - HA database.



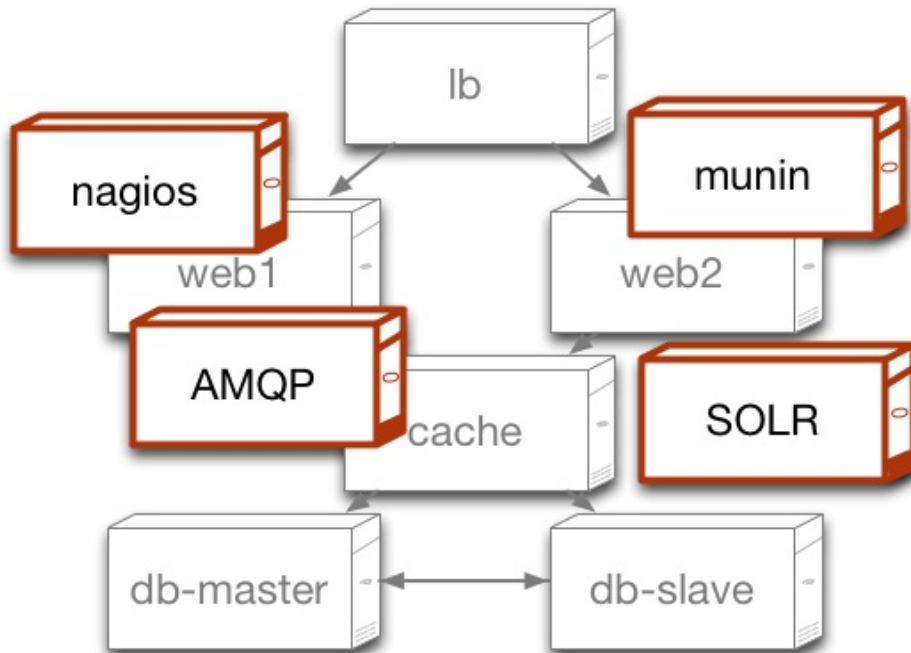
System Integration

In modern infrastructures, applications are not simply three-tier architectures anymore. Other components are added. Additional services are required to scale or add end-user features. We already have caching here, but wait, there's more:

- Message queues
- Search engines
- Third party services (e.g., billing, uptime/status, analytics)

Don't forget monitoring and trending!

Complexity Grows



Billing

Status Blog

Site Analytics

External Uptime

Introducing Chef



Chef

Chef Can Help

Chef is designed to help manage this kind of complexity. You may have met already!

- Configuration management tool
- Systems integration framework
- API for infrastructure management

Chef: The Tool

Chef is a tool for configuration management.

- Declarative: What, not how
- Idempotent: Only take action if required
- Convergent: Takes care of itself

Declarative Resources

You configure systems with Chef by writing self-documenting code. This code comprises lists of *Resources* that configure the system to do its job.

Chef manages system resources with a declarative interface that abstracts the details.

```
package "bash" do
  action :install
end
```

Idempotent Actions

Chef Resources have *Providers* that take idempotent action to configure the resource, but only if it needs to change.

```
INFO: Processing package[apache2] action install (apache2::  
DEBUG: package[apache2] checking package status for apache2  
DEBUG: package[apache2] current version is 2.2.20-1ubuntu1.1  
DEBUG: package[apache2] candidate version is 2.2.20-1ubuntu1.1  
DEBUG: package[apache2] is already installed - nothing to do
```

Convergent Nodes

Chef runs on the system, configuring the *Node*. The node is the unit of authority about itself.

In Chef, a single run should completely configure the system. If it does not, it is a bug (in your code, on the system, or in Chef itself).

Chef: The Framework

Chef provides a framework for system integration.

- Resources are written in Chef Recipes, a Ruby domain-specific language (DSL).
- Recipe helpers such as `search` allow dynamic data usage.
- Chef provides a library of primitives that can be used for other purposes.

Recipe Ruby DSL

Ruby is a 3rd generation interpreted programming language. Ruby has features that make it easy to create domain specific languages. This lends itself quite nicely to configuration management.

In Chef, Ruby gets out of the way, but it is still there when you need it.

Chef *Recipes* are a pure Ruby domain specific language. They are collected in *Cookbooks* along with associated components like config files or libraries.

Recipe Helpers

Chef provides a number of recipe helpers to obtain and manipulate data to use in Resources.

Search is used to discover information like IP addresses about other systems.

Arbitrary data about the infrastructure can be stored in *Data Bags* and accessed in recipes.

Library and Primitives

Chef can be used as a library within other applications. It speaks JSON and the server has a RESTful API accessed over HTTP(S).

Cookbooks can extend Chef with new libraries, including new resources and helpers to interact with 3rd party services.

Chef's included tools have plugin systems you can use to extend them.

Chef: The API

The Chef Server provides a network accessible API to stored data.

- Information about configured nodes
- Configuration policy in Cookbooks
- Descriptions of what policy to apply

Node Data

Chef gathers information about the node it is running on and saves this data to the Chef Server.

Node data is generated as a JSON key/value structure.

The JSON data is indexed for search by the Chef Server.

Configuration Policy

Policy about the nodes is written in recipes, which are stored in *Cookbooks*.

Cookbooks are uploaded to the Chef Server and distributed to the nodes that should be configured.

Cookbooks have versions and dependencies, both of which affect what code gets executed on particular nodes.

Applying the Policy

Tying it all together are `roles` which are descriptions of the nodes.

A `webserver` role contains the list of cookbooks and node-specific information required to fulfill serving HTTP traffic.

Chef inspects the node's role to determine what it should be to do its job.

Chef Summary: Configuration Management

- Declare configuration policy with resources
- Collect resources into recipes
- Package recipes and supporting code in cookbooks
- Apply cookbooks on nodes by specific roles
- Run Chef to configure nodes for their role

Chef Summary: Systems Integration

- Discovery through search
- 3rd generation programming language
- Fully expressive toolbox and primitives
- One run completely configures a single system

Questions

- What is configuration management?
- What is system integration?
- What are declarative resources?
- How are Chef resources idempotent?
- What language are Recipes written in?
- How are recipes distributed to nodes?
- Student questions?

Getting Started

Section Objectives

- Install Ruby and Chef
- Get familiar with the tools that come with Chef
- Set up connectivity to a Chef Server
- Create an initial Chef Repository

Supported Platforms

Opscode supports Chef Clients on the following platforms:

- Ubuntu (10.04, 10.10, 11.04, 11.10)
- Debian (5.0, 6.0)
- RHEL & CentOS (5.x, 6.x)
- Fedora 10+
- Mac OS X (10.4, 10.5, 10.6)
- Windows 7
- Windows Server 2003 R2, 2008 R2

Additional Platforms

Chef Client is known to also run on the following platforms:

- Ubuntu (6.06, 8.04-9.10)
- Gentoo (11.1,11.2)
- FreeBSD (7.1-9.0+)
- OpenBSD (4.4+)
- OpenSolaris (2008.11) / OpenIndiana / SmartOS
- Solaris 5.10 (u6)
- SuSE (11.x)
- Windows XP, Vista

Ruby and Chef

Chef is written in Ruby, an interpreted object oriented programming language.

Much of the code you'll work with for Chef is Ruby, albeit domain-specific language(s) suited for the task.

In order to install and use Chef, we need to have Ruby installed and available. It is important to understand some nuances about Ruby installation, first.

Ruby Versions

Ruby is an interpreted language and has several interpreter VMs available. The most common is "MRI" or the "Matz Ruby Interpreter"

The latest stable version of MRI is [Ruby 1.9.3](#).

Chef requires at least Ruby 1.8.7.

Platform Packages

Ruby and parts of the Ruby ecosystem have a reputation for backwards incompatibility. As such, not all platforms have the latest version available as the default package for "ruby".

Most Linux/Unix variants package version 1.8.7; it is an API-compatible transition version between 1.8 and 1.9.

Some make 1.9 version(s) available, but few use it as the default version.

RubyGems

Ruby software and libraries are usually published as Gems, using the Ruby packaging system *RubyGems*. Gems are published at <http://rubygems.org/>

Prior to Ruby 1.9, RubyGems was separately installed. It has been added to Ruby 1.9 in the standard library, at RubyGems version 1.3.7.

Chef requires at least RubyGems 1.3.7.

Installing Ruby and RubyGems

Chef is distributed primarily as a RubyGem, therefore Ruby and RubyGems need to be installed. Assuming the version requirements are met, we can simply install the Chef gem:

```
% sudo gem install chef
```

Were it that simple...

For some platforms, getting the required versions of Ruby and RubyGems for Chef is not trivial.

The current version of Ruby not being packaged by default has lead to other solutions, primarily centered around installing Ruby from source, to emerge.

Compiling from source is time consuming, and many (most?) system administrators prefer to install all software using packages.

Introducing Omnibus

Opscode needed a better way to distribute Chef so it is easier for customers and community members to install.

We also need a consistent installation method that our support team can troubleshoot.

Omnibus Build System

Opscode created the package build system "omnibus" to generate full-stack binary installers for Chef in a variety of formats.

- Native packages (rpm, deb)
- Self-extracting executable Tar.gz
- Microsoft Software Installer (MSI)

The idea is that all the software required above the standard C library for the OS is included. In Chef's case this is autoconf, openssl, zlib, Ruby, RubyGems and more.

Installing Chef Full

Omnibus is the build tool. The installation packages are called "chef-full".

Instructions for installation are at:

- <http://opscode.com/chef/install>

Supported Platforms

Installers are created for the following platforms. This may be different than the list of supported platforms, earlier.

- Debian, Ubuntu
- RHEL, CentOS, Scientific Linux, Oracle Enterprise Linux
- Fedora, Amazon Linux
- Mac OS X
- OpenIndiana
- Windows (Server 2003, 2008)

Linux/Unix Installation

Installation on Linux and Unix is done with a shell script:

```
% curl -L http://opscode.com/chef/install.sh | sudo bash
```

The script detects the platform of the system to determine what installation file to download.

Windows Installation

Installation on Windows is done by downloading the MSI and installing it.

- <http://opscode.com/chef/install.msi>

Server versions are directly tested, but the MSI is known to install and work fine on desktop versions of Windows such as Vista and 7.

What You Get: Linux/Unix

Chef binaries are in `/opt/opscode/bin`. The package installation symlinks them in `/usr/bin` so they are in the default `$PATH`.

Other binaries are in `/opt/opscode/embedded/bin`. This includes `ruby`, `gem`, and binaries for other included software like `autoconf`, `openssl` and more.

What You Get: Windows

Chef binaries are in `C:\opscode\bin`. The other binaries are in `C:\opscode\embedded\bin`.

These directories are both added to the system `%PATH%`.

Chef Full Ruby

The Ruby included in the Chef Full stack installation package is intended for Chef's use.

If Ruby is required for other tools or applications on the system, Opscode recommends installing it with a recipe.

If you wish to install Chef extensions such as knife plugins, report handlers or gems used in cookbooks, use the full-stack included Ruby.

Chef Toolbox

Chef comes with several tools of its own.

- ohai
- chef-client and chef-solo
- knife
- shuf

Other Tools

In working with infrastructure, we have several tools in the toolbox.

Non-Chef tools:

- Shell (Bash, Zsh, Powershell, Cmd.exe)
- Text editors (Emacs, Vim, Notepad++)
- Version control systems (Git, Subversion, Perforce)
- Ruby programming language

Each of the tools bundled with the Chef Full package share some common traits.

- Built-in help
- Configuration

Built-in Help

Each command has a `--help` or `-h` command-line option that displays options and contextual help output.

Each command also has a corresponding Unix `man(1)` page, which is included in the installed Chef library.

Built-in Help

As Windows does not have a `man(1)` help system, HTML pages are generated. These are located in:

- `C:\opscode\chef\embedded\lib\ruby\ gems\1.9.1\gems\chef-VERSION\distro\common\html`

Where "VERSION" is the version of Chef.

A future release will have helpers to make these easier to access.

Configuration

Each tool that Chef comes with has its own configuration file.

Configuration files populate values in the `Chef::Config` object.

Chef comes with sane default values for all configuration options.

Context of the configuration file to the appropriate tool is important.

Chef Configuration

`Chef::Config` uses a simple domain specific language where the setting and its value are specified.

```
log_level :info
log_location STDOUT
chef_server_url "https://api.opscode.com/organizations/opstrain"
```

Ohai is a standalone library written by Opscode that is installed with Chef as a dependency.

Ohai uses plugins to profile the local system when Chef runs to gather information.

When Chef runs, this data gets stored on the Chef Server.

Ohai Configuration

Ohai is usually configured via the chef-client configuration file (`/etc/chef/client.rb`). It has no other configuration file.

The ohai configuration must be modified with `Ohai::Config`. It is a Ruby hash-like object that uses symbols for key names.

```
Ohai::Config[:disabled_plugins] << 'passwd'
```

Knife is the "swiss army knife" of infrastructure management tools.

- manage the local Chef repository
- interact with the Chef Server API
- interact with cloud computing providers' APIs
- extend with custom plugins/libraries

Knife Sub-commands

Knife plugins are used as sub-commands. General format of knife sub-commands:

```
knife COMMAND verb noun (options)
```

This is consistent for Chef API, but some differences across other uses.

Knife Command Examples

```
knife node show NODENAME
```

```
knife cookbook upload fail2ban
```

```
knife role edit webserver
```

Knife Contextual Help

```
knife --help
```

```
knife sub-command --help
```

```
knife sub-command verb --help
```

Knife Man Pages

Knife has built-in man pages.

```
knife help
```

```
knife help list
```

```
knife help knife
```

```
knife help TOPIC
```

```
knife help node
```

Knife Configuration

The default configuration file for Knife is `.chef/knife.rb`; knife looks for it automatically, similar to `git`:

```
$PWD/.chef/knife.rb  
$PWD/"../.chef/knife.rb  
~/.chef/knife.rb
```

Opscode Hosted Chef provides a pregenerated `knife.rb` you can use.

Knife Configuration Options

Knife configuration uses `Chef::Config`.

Knife also has its own specific configuration for various plugins to use. These are in `Chef::Config[:knife]`, which is a hash of configuration options.

Knife Configuration

To work with the Chef Server API, Knife must be configured with:

- The Chef Server's URL (`chef_server_url`).
- The user to authenticate to the API (`node_name`).
- The private key for the authenticating user (`client_key`).

Knife Configuration

Minimal `knife.rb` configured to use a particular Opscode Hosted Chef organization and Opscode user.

```
node_name      "opscode-trainer"  
client_key     "opscode-trainer.pem"  
chef_server_url "https://api.opscode.com/organizations/opstrain"
```

Knife User

The configuration value `node_name` in the `knife.rb` refers to an Opscode user.

Users are global to the entire Opscode service.

Users may be associated with one or more organizations.

Knife Configuration

We work with a local Chef Repository that stores the various files, including cookbooks, that should be sent to the Chef Server. Knife is written to automatically use these locations.

One directory it does need to be told is the path where cookbooks live.

```
cookbook_path "./cookbooks"
```

This configuration file is Ruby, not bash, so we need to be careful with the path usage. The default `knife.rb` provided with a Hosted Chef account through the webui will have this already set for you.

Knife Configuration

The cookbook path in the pre-generated Knife configuration file uses a relative location based on the location of the `knife.rb` file.

```
current_dir = File.dirname(__FILE__)  
cookbook_path ["#{current_dir}/../cookbooks"]
```

Knife Command-line Options

Knife has a base set of command-line options that correspond to general options in the `knife.rb`.

Each sub-command may have specific command-line options that are different.

Knife Command-line Options

The following command-line options correspond to the config file settings seen above:

Command-line Option	knife.rb	
-----	-----	
-s, --server-url URL	chef_server_url	
-k, --key KEY	client_key	
-u, --user USER	node_name	

Knife Command-line Options

Other common command-line options used with knife.

Command-line Option	Purpose
-----	-----
-c, --config CONFIG	Configuration file to use.
-V, --verbose	Verbose output, can be specified twice.
-F, --format	Output format, can be json, yaml, text

Chef Client & Chef Solo

The programs `chef-client` and `chef-solo` load the Chef library and make it available to apply configuration management with Chef.

Both programs know how to configure the system given the appropriate recipes found in cookbooks.

Chef Client

chef-client talks to a Chef Server API endpoint, authenticating with an RSA key pair. It retrieves data and code from the server to configure the node per the defined policy.

List of recipes can be predefined, assigned to a node on the Chef Server, and retrieved when chef-client runs.

The default configuration file is `/etc/chef/client.rb`.

Chef Client Configuration

Like Knife, `chef-client` must be configured with the proper authentication information to connect to the Chef Server.

Unlike Knife, the `node_name` is not a user, but the actual system's name for itself. Unless otherwise specified in `/etc/chef/client.rb`, the `node_name` is the value detected by Ohai as the `fqdn` (fully qualified domain name).

Chef Client Configuration

The minimal configuration for `chef-client` in `/etc/chef/client.rb` to talk to the Chef Server:

```
chef_server_url  "https://api.opscode.com/organizations/opstrain"  
validation_client_name "opstrain-validator"
```

All other options will use default values, which are meant to be sane defaults.

Chef Client Configuration

Other common configuration options (default values are used below):

```
log_level          :info
log_location       STDOUT
verbose_logging    true
file_cache_path    "/var/chef/cache"
file_backup_path   "/var/chef/backup"
json_attribs       nil
```

Client Command-line Options

The following command-line options correspond to the specified config file settings.

Command-line Option	client.rb	
-----	-----	
-S, --server URL	chef_server_url	
-k, --key KEY	client_key	
-u, --user USER	node_name	

Client Command-line Options

Other common command-line options used with `chef-client`:

Command-line Option	client.rb	
-----	-----	
-l, --log_level LEVEL	log_level	
-L, --logfile LOGFILE	log_location	
-j JSON_ATTRIBUTES	json_attribs	
-N, --node-name NAME	node_name	

Client Command-line Options

The following options control how the `chef-client` process behaves.

Command-line Option	Purpose
-----	-----
<code>-d, --daemonize</code>	Daemonize the process
<code>-i, --interval INT</code>	Run every INT seconds
<code>-s, --splay SECONDS</code>	Random splay added to interval

Full example client.rb

```
log_level          :info
log_location       STDOUT
chef_server_url     "https://api.opscode.com/organizations/ORGNAME"
validation_client_name "ORGNAME-validator"
# Using default node name
```

Command-line Options Override

Options passed on the command-line override values in the configuration file.

```
chef-client -l debug
```

```
chef-client -N node-name
```

```
chef-client -S https://api.opscode.com/organizations/OTHER
```

```
chef-client --help
```

Chef Server

The Chef Server is a centralized publishing system for infrastructure data and code.

- Stores node, role and user-entered data
- Data is indexed for search
- Stores cookbooks
- Provides an API for management and discovery

Chef Server API Implementations

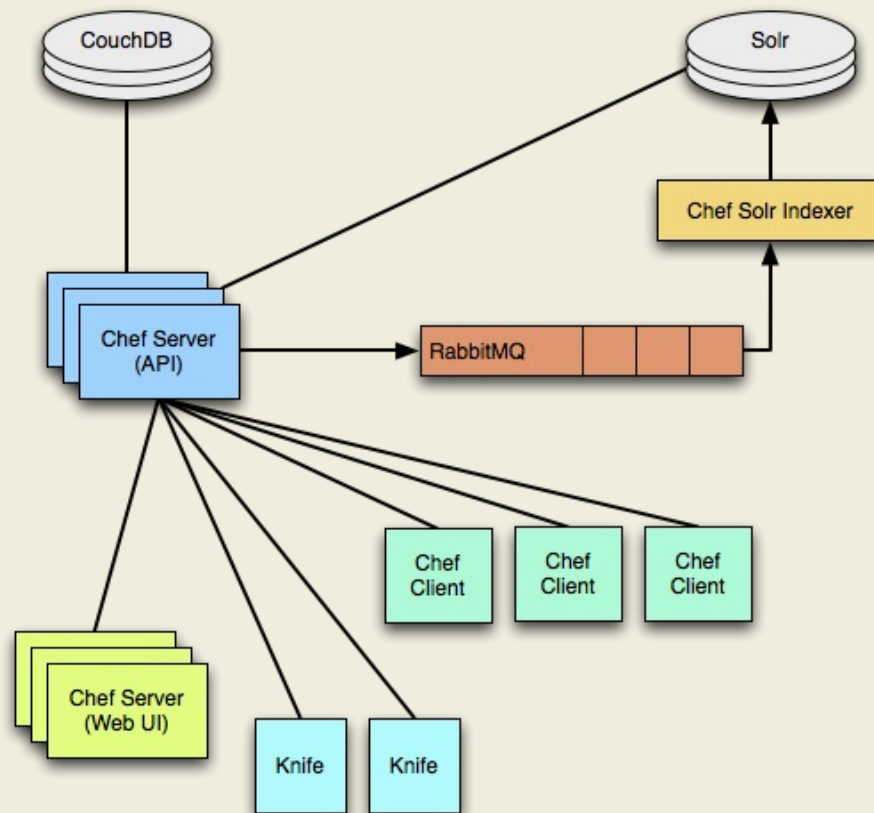
Opscode Hosted Chef

Opscode Private Chef

Open Source Chef Server

Commis (Python)

Chef Server Components



Chef Server API Services

API Server (HTTP & JSON, Authentication)

Data storage (JSON documents, Cookbooks)

Message queue (Search indexing, other services)

Search Engine (Full text search)

Web management console (API client)

Chef Server Security

All communication is initiated by API clients, and never from the Chef Server directly.

Communication is over HTTP. Opscode Hosted Chef and Opscode Private Chef use HTTPS.

All API requests are authenticated using digital signatures.

All API requests for Opscode Hosted Chef and Opscode Private Chef are authorized with role-based access controls.

Custom data ("data bags") can be encrypted with user-supplied keys.

Chef Server

We will use Opscode Hosted Chef as the Chef Server for this course.

- It is free up to 5 nodes.
- It is internet-accessible.
- No additional software to install/configure.

Sign up for Hosted Chef

Go to <http://opscode.com>

Hosted Chef:



Instant Access

Get instant access to a highly available, dynamically scalable, fully managed and supported automation environment - powered by the experts at Opscode.

[Learn more](#)  [Sign Up](#) 

Free Up to 5 Nodes

Plans & Pricing

	Launch	Standard	Premium
<i>Monthly Fees</i>	\$100	\$300	\$700
<i>Nodes</i>	20	50	100
<i>Users</i>	10	20	50
<i>Standard Support</i>	Included	Included	Included
<i>Onsite Training</i>	Not available	Not available	Available

Buy Now>

Buy Now>

Buy Now>

Need more nodes? Contact us about Hosted Chef for the Enterprise >

Hosted Chef is free for 5 nodes or less!

Questions about Hosted Chef pricing and support? Check the FAQ

Free Trial>

Sign up for Hosted Chef

Plans & Pricing

	Launch	Standard	Premium
<i>Monthly Fees</i>	\$100	\$300	\$700
<i>Nodes</i>	20	50	100
<i>Users</i>	10	20	50
<i>Standard Support</i>	Included	Included	Included
<i>Onsite Training</i>	Not available	Not available	Available

Buy Now>

Buy Now>

Buy Now>

Need more nodes? Contact us about Hosted Chef for the Enterprise >

Hosted Chef is free for 5 nodes or less!

Questions about Hosted Chef pricing and support? Check the [FAQ](#)



Free Trial>

Organization short name:

- Alphanumeric
- Hyphen and underscore

About you

1 2 3

First Name *	Middle Name	Last Name *
Username *		Email Address *
Password *		Confirm Password *
Organization Name *		Organization Short Name *
Phone Number *		Company *
Country * 	State * 	
<input type="checkbox"/> I agree to the Terms of Service , Opscode Platform Customer Agreement , and Opscode Service Level Agreement .		

Verify your email address.

Done! Next Steps

1 2 **3**

Verify your email address >

Please check your email for a message from Opscode for verification instructions

Read the Getting Started Guide >

Setup command-line access and your first cookbook

Experienced with Chef? >

Manage your org with the Operations Console

Need Help? >

If you have questions or you're stuck, we're here to help

Next Steps

Select "Experienced with Chef" to go to the Management Console.

Your address has been verified. Welcome!

What's Next?

Read the Getting Started Guide

setup command-line access and your first cookbook



Experienced with Chef?

manage your org with the Operations Console



Need Help?

if you have questions or you're stuck, we're here to help



Download Organization Assets

Login: <https://manage.opscode.com>

Logged in as: [opstrain42](#) | [Organizations](#) | [Logout](#)



All Joined Organizations

[opstrain_42](#)

Select | [Manage account](#) | [Regenerate validation key](#) | [Generate knife config](#)



All Joined Organizations

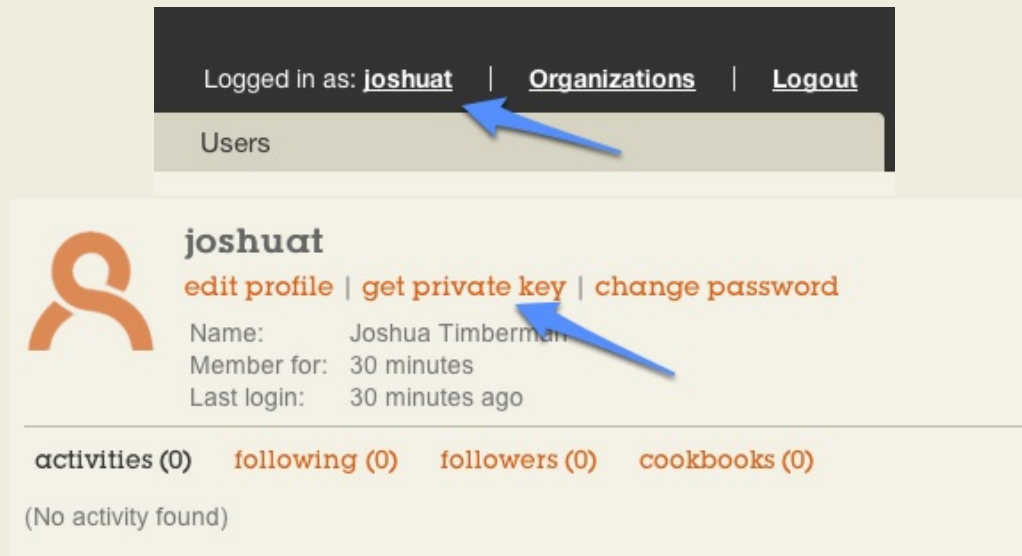
[opstrain_42](#)

Select | [Manage account](#) | [Regenerate validation key](#) | [Generate knife config](#)

Download the validation key and knife config to somewhere such as ~/Downloads.

Download User Private key


Select your username at the top of the management console to access your profile page.



The screenshot shows a management console interface. At the top, a dark header bar contains the text "Logged in as: [joshuat](#) | [Organizations](#) | [Logout](#)". Below this, a light gray bar displays the word "Users". A blue arrow points from the "joshuat" link in the header to the "joshuat" profile card below. The profile card features an orange user icon, the username "joshuat", and three links: "edit profile", "get private key", and "change password". A second blue arrow points from the "get private key" link to the right. Below the links, the user's details are listed: "Name: Joshua Timberman", "Member for: 30 minutes", and "Last login: 30 minutes ago". At the bottom of the card, there are four links: "activities (0)", "following (0)", "followers (0)", and "cookbooks (0)". Below these links, the text "(No activity found)" is displayed.

Logged in as: [joshuat](#) | [Organizations](#) | [Logout](#)

Users

 **joshuat**
[edit profile](#) | [get private key](#) | [change password](#)

Name: Joshua Timberman
Member for: 30 minutes
Last login: 30 minutes ago

[activities \(0\)](#) [following \(0\)](#) [followers \(0\)](#) [cookbooks \(0\)](#)

(No activity found)

Sign-up Results

- Opscode Hosted Chef Login
- Opscode Hosted Chef Organization
- User private key
- Validation or Organization key
- Knife Configuration file

Chef Repository

Very simply, the Chef Repository is a version controlled directory that contains cookbooks and other components relevant to Chef.

It contains your "Infrastructure as Code".

Knife already knows how to interact with many parts of the repository.

We'll look at each part of the repository in greater detail when we get to the relevant section.

Chef Repository

Example Chef Repository directory tree:

```
chef-repo
```

```
|— .chef
```

```
|— cookbooks
```

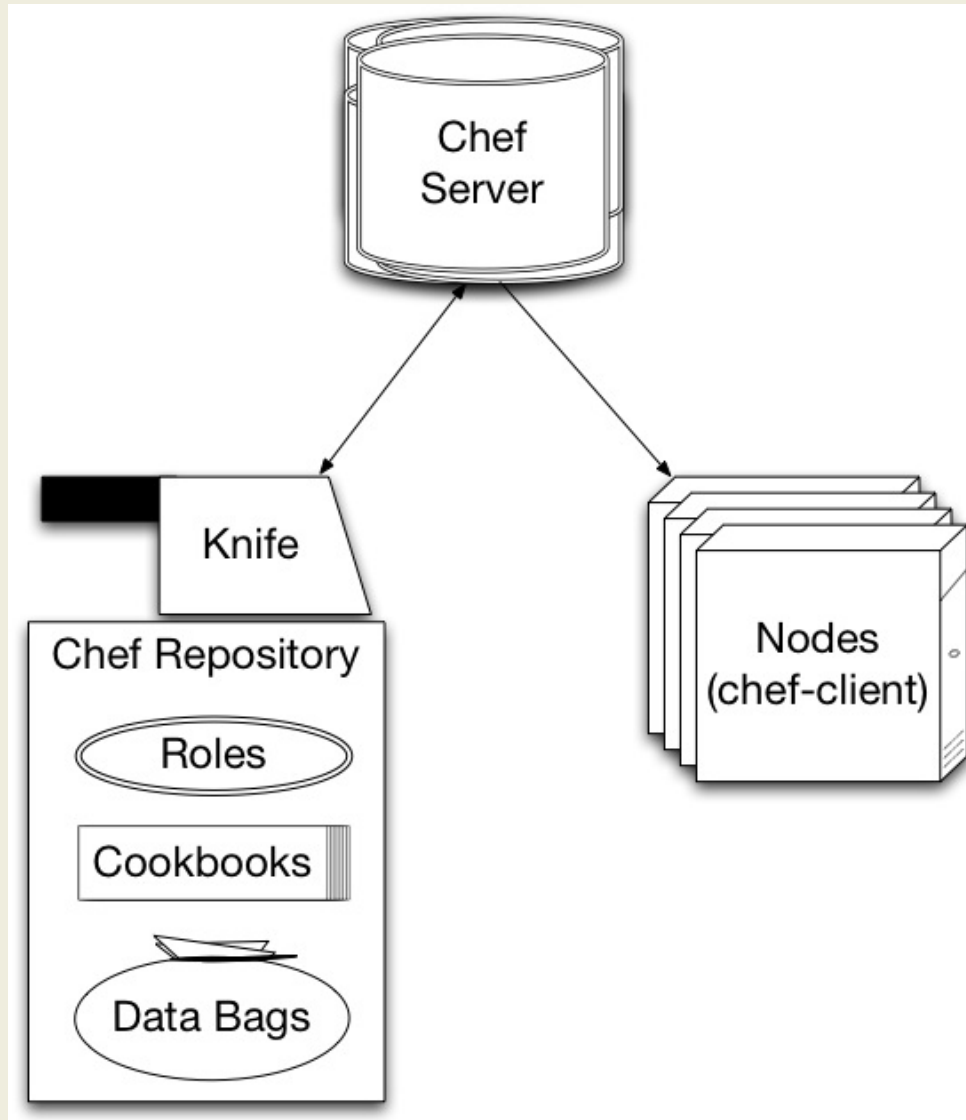
```
|— data_bags
```

```
|— environments
```

```
|— README.md
```

```
└─ roles
```

Working with Chef



Summary

- Ruby and Chef Installation
- Tools and commands that come with Chef
- Connectivity to the Chef Server
- Components of a Chef Repository

Questions

- In what language is Chef written?
- How is Chef distributed by Opscode?
- What are commands / tools that come with Chef?
- What do Chef's commands have in common?
- What are two implementations of the Chef Server API?
- What configuration is required to connect to a Chef Server?
- Student questions?

Additional Resources

- <http://wiki.opscode.com/display/chef/Resources>
- <http://wiki.opscode.com/display/chef/Recipes>
- <http://wiki.opscode.com/display/chef/Chef+Repository>
- <http://wiki.opscode.com/display/chef/Chef+Configuration+Settings>
- <http://wiki.opscode.com/display/chef/Server+API>
- <http://community.opscode.com/cookbooks>

Getting Started

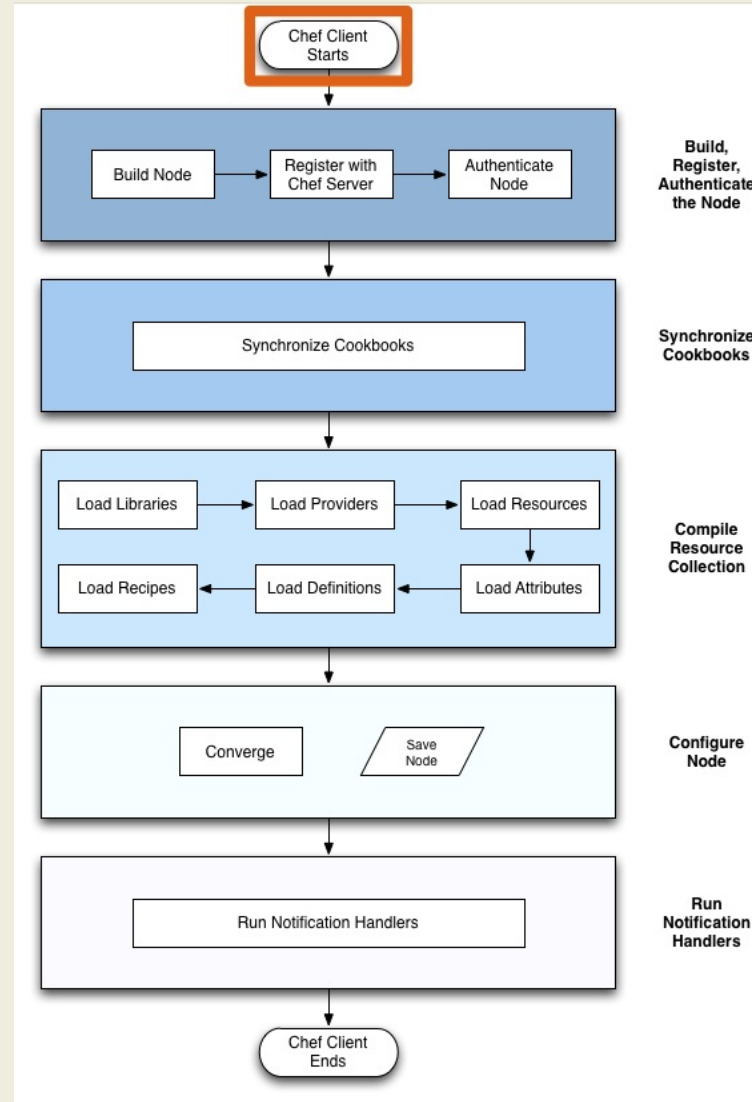
- Install Ruby and Chef
- Get familiar with the tools that come with Chef
- Set up connectivity to a Chef Server
- Create an initial Chef Repository

Anatomy of a Chef Run

Section Objectives:

- Chef API Clients
- Chef Nodes
- Node convergence phases
- Notification handler types

Anatomy of a Chef Run



Anatomy of a Chef Run

- Build the node
- Synchronize cookbooks
- Compile resource collection
- Configure the node
- Run notification handlers

Build the Node

Before anything else happens, the system is profiled with Ohai.

Chef will use the detected fully qualified domain name (fqdn) for the node's name unless it was specified:

- `node_name` in `/etc/chef/client.rb`
- `chef-client -N`

Node names should be unique, they are used for the managed node and the API client.

API Clients

API Clients authenticate with the Chef Server.

Chef uses Signed Header Authentication across all API requests.

The API requests are authenticated using the `node_name` for the API client. The timestamp is recorded in the headers to prevent replay attacks.

API Authentication

Does `/etc/chef/client.pem` exist?

- Use it to sign requests

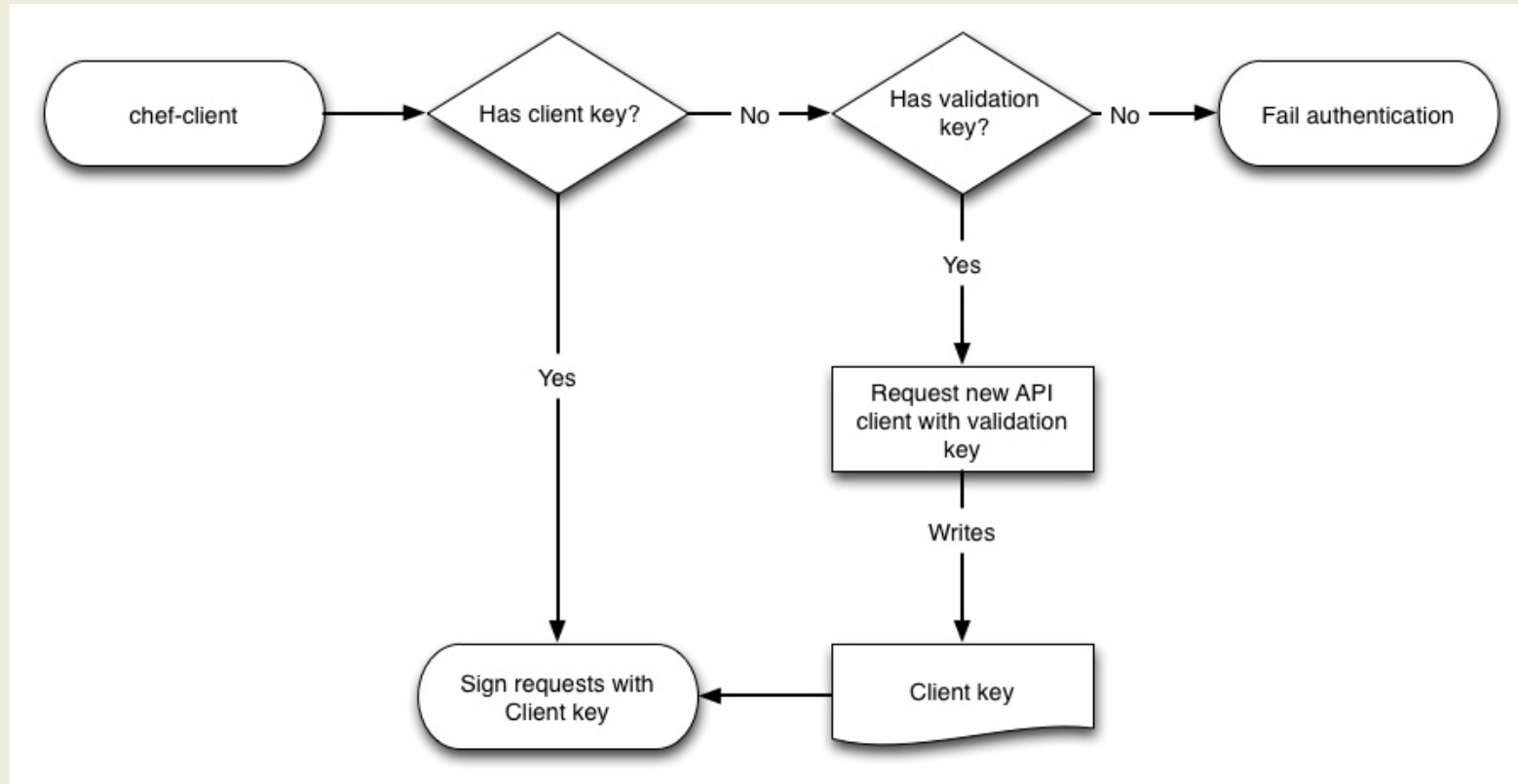
Does `/etc/chef/validation.pem` exist?

- Request a new API client key
- Or fail

Was a new client key generated?

- Use it to sign requests

API Authentication Process



Users are Special API Clients

With Opscode Hosted Chef, people authenticate as *Users*, which are special API clients since they're global.

Users are associated with an *Organization*.

What a user can do is determined by the access control lists in the organization and by granting of certain group-based privileges.

Node Objects

After the client has authenticated with the Server, Chef retrieves the node object from the server.

Node objects represent a set of data called attributes and a list of configuration to apply called a run list.

Node Object

`Chef::Node` is the node object. It looks and almost behaves like a hash, except when it doesn't.

Nodes have attributes at varying priority levels (automatic, default, normal, override).

Nodes have a run list.

Nodes have an environment.

Node Object: JSON

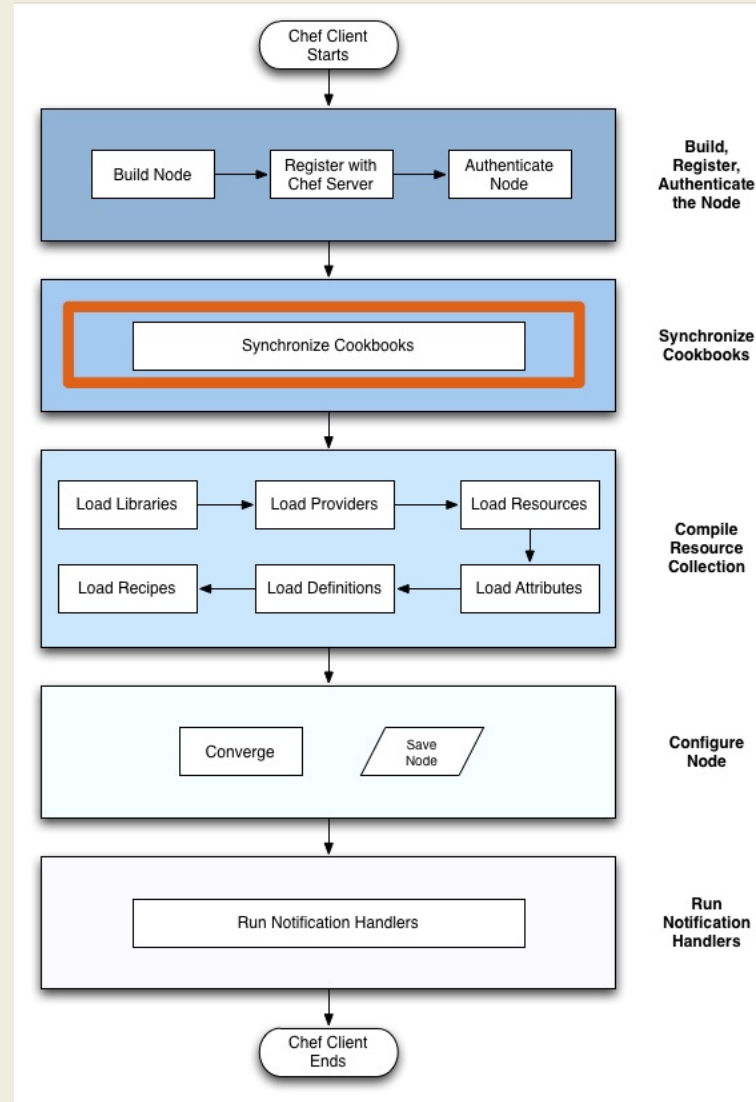
```
{  
  "name": "www1.example.com",  
  "json_class": "Chef::Node",  
  "chef_type": "node",  
  "chef_environment": "_default",  
  "automatic": { ... },  
  "default": { ... },  
  "normal": { ... },  
  "override": { ... },  
  "run_list": [ ... ]  
}
```

Node Run List Expands

The run list can contain recipes and roles. Roles can contain recipes and also other roles.

Chef expands the node's run list down to the recipes. The roles and recipes get set to node attributes.

Anatomy of a Chef Run



Synchronize Cookbooks

Chef downloads from the Chef Server all the cookbooks that appear as recipes in the node's expanded run list.

Chef also downloads all cookbooks that are listed as dependencies which might not appear in the run list.

If the node's `chef_environment` specifies cookbook versions, the Chef downloads the version specified. Otherwise the latest available version is downloaded.

Cookbook Metadata

If a recipe from another cookbook is included in a recipe, it isn't automatically downloaded.

Some cookbooks don't actually have recipes, and instead provide helper code, libraries or other assets we want to use.

To ensure the node has components from other cookbooks used in recipes, we declare explicit dependencies in cookbook metadata.

For example, if we want to re-use a template from the `apache2` cookbook in the `webserver` cookbook, declare a dependency on the `apache2` cookbook.

```
# in webserver/metadata.rb...  
  
depends "apache2"
```

Cookbook Cache

Cookbooks are stored on the local system in the directory configured by `"file_cache_path"`. The default is `/var/chef/cache` unless changed in `/etc/chef/client.rb`.

Cookbooks that have not changed are not downloaded again, the cached copy will be used.

Chef Run

This starts when you see:

```
INFO: Starting Chef Run for NODE_NAME
```

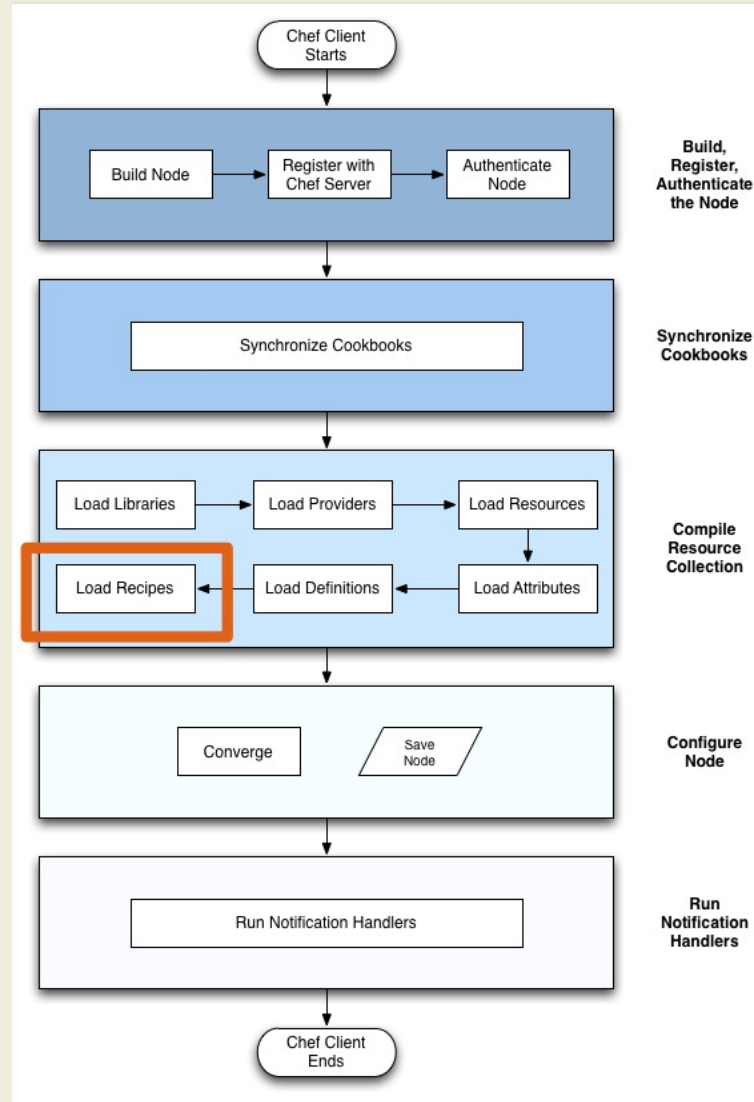
The run context is created with the node and the cookbook collection.

Load Cookbooks

Once the cookbooks are synchronized to the local system, their components are loaded in the following order:

- Libraries
- Providers
- Resources
- Attributes
- Definitions
- Recipes (in the order specified)

Anatomy of a Chef Run



Cookbook Files and Templates

Cookbook static assets (files) and dynamic assets (templates) are not retrieved or loaded at this time.

They are retrieved from the server and rendered when needed by a resource in a recipe.

Node Convergence

Convergence is when the configuration management system brings the node into compliance with policy.

In other words, the node is configured based on the roles and recipes in its run list.

Convergence in Chef happens in two phases.

- Compile
- Execute

Convergence: Compile

Chef recipes are written in Ruby. During the compile phase, the Chef Recipe DSL is processed for Chef *Resources* to be configured.

During the processing of recipes:

- Ruby code is executed directly
- Recognized resources are added to the *Resource Collection*

For example:

```
pkg = "apache2"

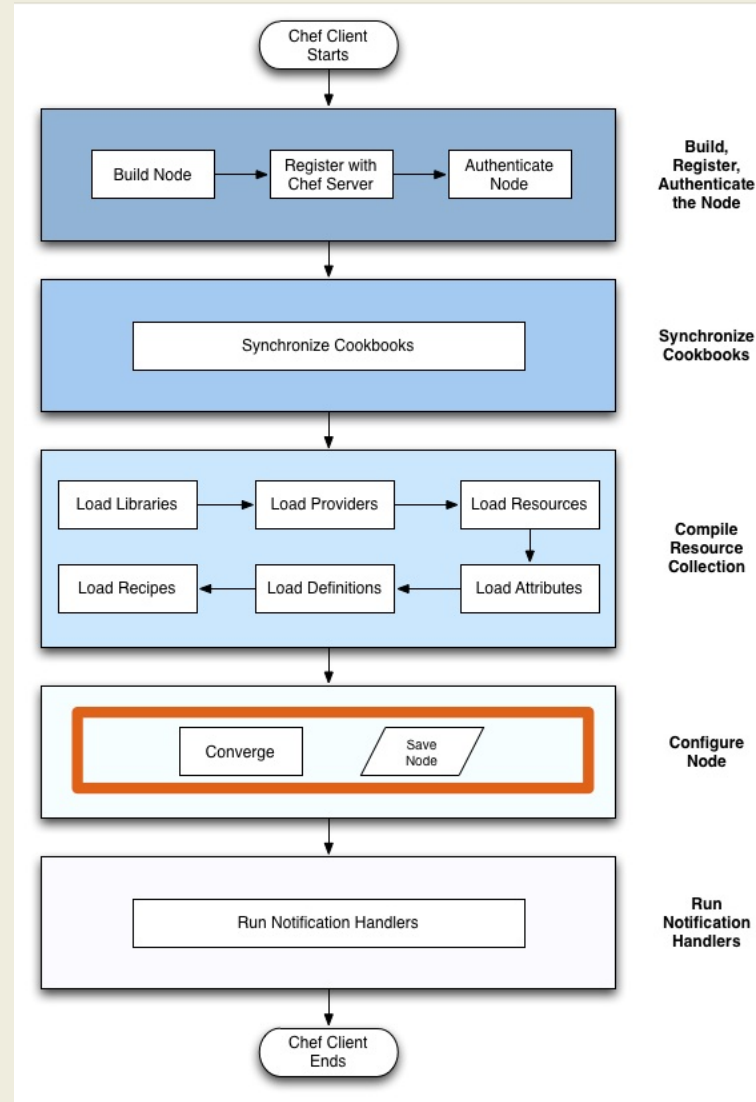
package pkg do
  action :install
end
```

Convergence: Execute

Chef walks the Resource Collection in order.

- Chef runs the specified actions for each resource
- Providers know how to perform the actions

Anatomy of a Chef Run



Report and Exception Handlers

At the end of the Chef Run, report and exception handlers are triggered.

- Report handlers run when Chef exits cleanly
- Exception handlers run when Chef exits abnormally with an unhandled exception.

Report Handlers

Normal, clean exit:

```
INFO: Chef Run complete in 42.72288 seconds
```

```
INFO: Running report handlers
```

```
INFO: Report handlers complete
```

Exception Handlers

Abnormal exit from unhandled exception:

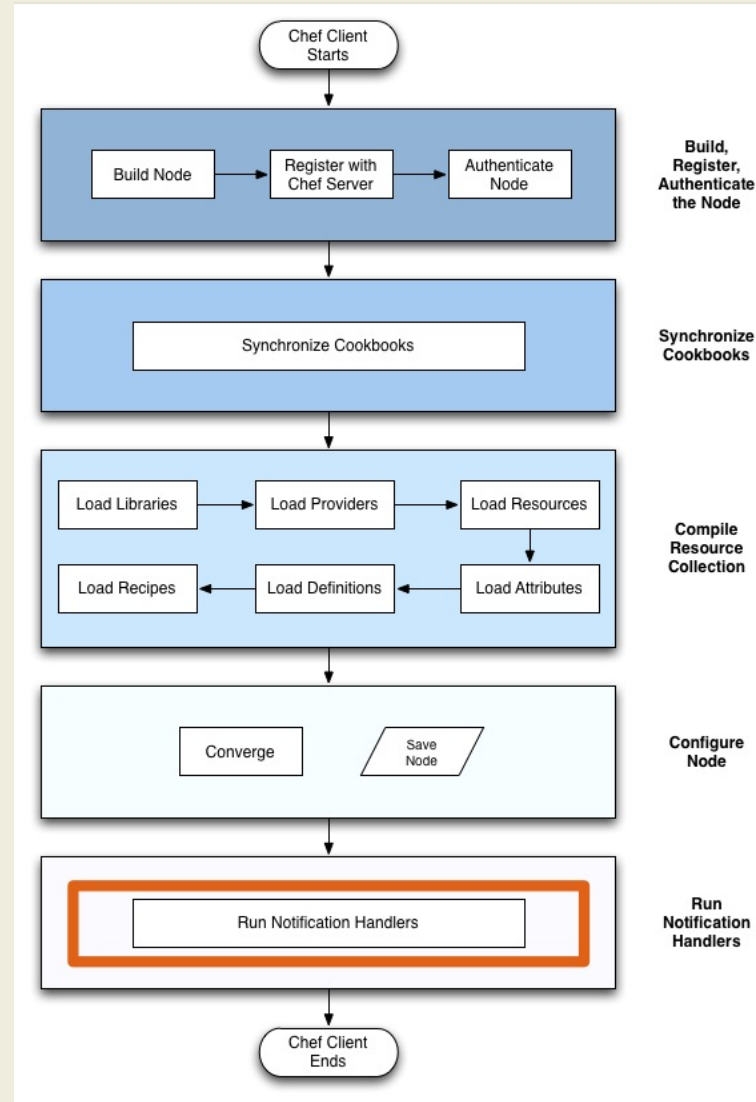
```
^CFATAL: SIGINT received, stopping
```

```
FATAL: SIGINT received, stopping
```

```
ERROR: Running exception handlers
```

```
ERROR: Exception handlers complete
```

Anatomy of a Chef Run



Summary

- Chef API Clients
- Chef Nodes
- Node convergence phases
- Notification handler types

Questions

- What is an API client?
- How does the API client get created automatically?
- How is an API client different from a node?
- What are the two main components of a node object?
- How does Chef determine what cookbooks to download?
- Where do cookbooks get downloaded?
- What are the two phases of node convergence and how do they differ?
- What is the difference between a report and exception handler?
- Student questions?

Additional Resources

- <http://wiki.opscode.com/display/chef/Anatomy+of+a+Chef+Run>
- <http://wiki.opscode.com/display/chef/Authentication>
- <http://wiki.opscode.com/display/chef/Chef+Client>
- <http://wiki.opscode.com/display/chef/Nodes>
- <http://wiki.opscode.com/display/chef/Attributes>
- <http://wiki.opscode.com/display/chef/Evaluate+and+Run+Resources>

Anatomy of a Chef Run

- Configure remote target to run `chef-client`
- Successful `chef-client` run with debug logging

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

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.

```
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 substitution than single quoted strings.

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

```
"This is a string."
```

```
'This is another string.'
```

Code Substitution

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

```
code_sub = "code substitution"  
"This is a string with #{code_sub}"
```

This is often done in Chef to use node attributes:

```
"#{node['my_package']['dir']}/my_package.conf"
```


Numbers

Ruby supports integers and floating point numbers.

```
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.

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

```
[ 80, 443, 8080 ]
```

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

```
%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.

```
{  
  "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.

```
:thing
```

Symbols have been used as hash keys instead of strings because they can be more memory efficient. However, they are not as easy for non-programmers to understand, so you generally want to avoid them where possible.

```
{  
  :site => "opscode.com",  
  :ports => [ 80, 443 ]  
}
```

True, False and Nil

In Ruby, only `nil` and `false` are false.

```
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.

```
if node['platform'] == "ubuntu"
  # do ubuntu things
end

unless node['platform'] == "ubuntu"
  # don't do ubuntu things
end
```

Case statements can be used as well.

```
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.

```
"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`.

Blocks

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

```
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.

```
%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

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

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

```
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.

```
remote_file "#{Chef::Config[:file_cache_path]}/mystuff.tar.gz" do
  source "http://example.com/mystuff.tar.gz"
end
```

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`.

```
unless Chef::Config[:solo] # if we're not using solo...  
  results = search(:node, "role:webserver") # perform search  
end
```


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>

Cookbooks, Recipes, Resources

Section Objectives:

- Components of Chef cookbooks
- Create new cookbooks
- Write simple recipes
- Recognize and write Chef resources
- Run Chef with a cookbook on a node

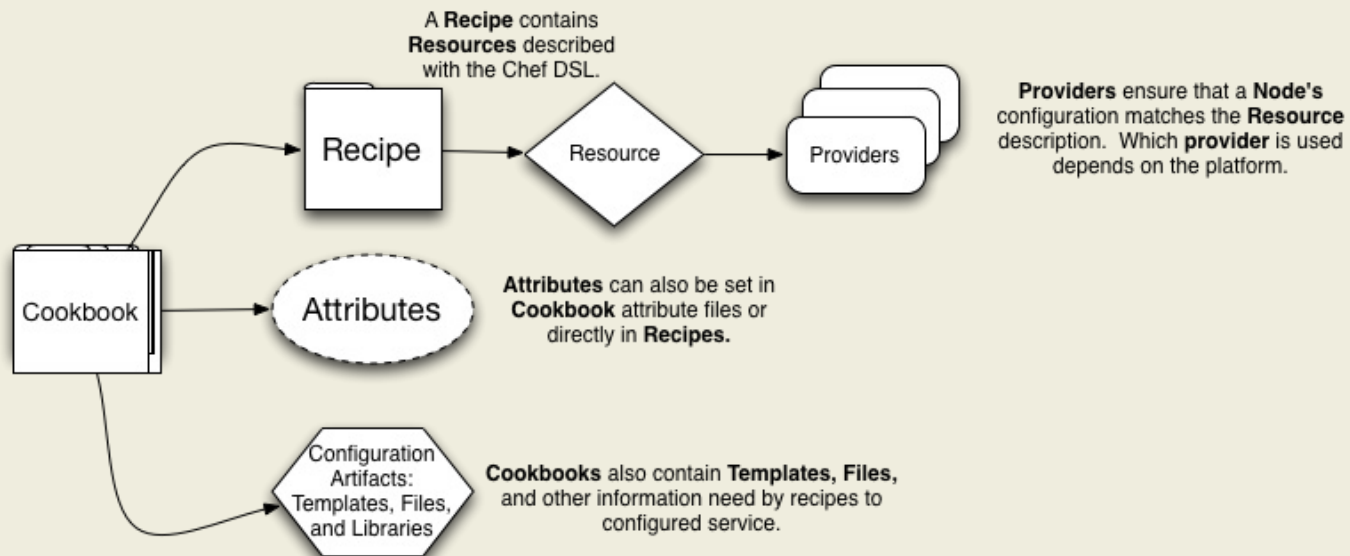
Cookbooks Overview

Cookbooks are the fundamental units of file distribution in Chef. They are "packages" for Chef recipes and other helper components.

They are designed to be sharable packages for managing infrastructure as code. Cookbooks can be shared within an organization, or with the Chef Community.

Nodes managed by Chef download cookbooks from the Chef Server to apply their configuration.

Cookbook Basics



Creating Cookbooks

Cookbooks are merely a structured set of directories in your Chef Repository.

Create a new cookbook in the `cookbooks` directory. The content of the recipe and `metadata.rb` don't matter right now, they just need to exist.

```
mkdir cookbooks/webserver  
mkdir cookbooks/webserver/recipes  
touch cookbooks/webserver/recipes/default.rb  
touch cookbooks/webserver/metadata.rb
```

Cookbook Components

The most commonly used cookbook components are:

- recipes
- metadata
- assets (files and templates)

Common Components: Metadata

Chef Cookbook *Metadata* is written using a Ruby domain-specific language.

Metadata serves two purposes.

- Documentation
- Dependency management

Metadata Documentation

Cookbook metadata contains documentation about the cookbook itself.

```
maintainer      "Opscode, Inc."  
maintainer_email "cookbooks@opscode.com"  
license         "apachev2"  
description     "Configures web servers"  
long_description "Configures web servers with a cool recipe"
```

Metadata Dependency Management

Metadata is also used for dependency management.

Cookbooks are like OS packages in that they can be interrelated. You can use parts of one cookbook in another.

Cookbook dependencies are assumed when using part(s) of one cookbook in another, such as including a recipe, or re-using a template.

Cookbook dependencies are explicitly defined in metadata with the `depends` field.

Metadata Version Management

Related to dependency management is version management. Cookbooks can be versioned, and dependencies on versions can be declared.

Cookbook versioning is enforced with dependencies if used, and in Chef Environments.

Specify a cookbook's version with the `version` field. It should be a quoted string in the form "X.Y.Z", e.g., "1.0.0" or "0.5.2".

Cookbook Dependencies

The metadata defines the additional cookbooks required that might not appear in the run list explicitly.

When cookbooks are uploaded, the Ruby code is parsed by Knife and translated to JSON when it is stored on the Chef Server.

This is a security feature, so the Chef Server does not execute user-defined Ruby code.

Common Components: Assets

One of the most common things to manage with Chef are configuration files on the node's filesystem.

- `/etc/mysql/my.cnf`
- `/var/www/.htaccess`
- `C:\Program Files\My Application\Configuration.ini`

Chef cookbooks can contain `files` and/or `templates` directories to contain the source files for these resources as we'll see later.

Common Components: Recipes

Recipes are the work unit in Chef. They contain lists of resources that should be configured on the node to put it in the desired state to fulfill its job.

Nodes have a run list, which is simply a list of the recipes that should be applied when Chef runs.

The node will download all the cookbooks that appear in its run list.

Chef Recipes

Recipes contain lists of resources.

Resources are declarative abstract interfaces to OS resources like packages, services, config files and users.

Information about the node itself is available via the `node` object.

Chef Recipes

Recipes are an internal Ruby domain-specific language (DSL).

- You need a 3rd generation programming language.
- You can't be limited by the language.

How Are Recipes Applied?

Nodes have a list of recipes they will run.

This run list can include recipes that also include other recipes.

These are applied to the node in the order listed.

Chef Recipes

Recipes are processed in the order they are written.

- Ruby code is evaluated.
- Ruby recognized as Resources are added to the resource collection.
- Chef walks the resource collection to configure the resources.
- Providers take action to configure the resource as it was declared.

Resources

Resources are the fundamental configuration object.

Chef manages system resources on the node so it can be configured to do its job.

- The apache2 package should be installed.
- The application user should be created.

Resources

Resources abstract the details of how to configure the system. The commands:

```
apt-get install apache2  
useradd application
```

Become Chef resources:

```
package "apache2"  
user "application"
```

Resources

Resources take idempotent actions through *Providers*.

Providers know how to determine the current state of the resource on the node.

Providers do not take action if the resource is in the declared state.

Resources Chef Can Configure

- directories, files, templates, remote files
- packages, services, users, groups
- scripts, commands, ruby code blocks
- subversion and git code repositories
- application deployment, HTTP requests
- network interfaces, filesystem mounts

Chef includes over 25 different kinds of resources.

Resource Components

- Resources have a type
- Resources have a name
- Resources take parameter attributes
- Resources specify the action to take

Example Resources

```
directory "/etc/thing" do
  owner "root"
  group "root"
end
```

```
template "/etc/thing/config.conf" do
  source "config.conf.erb"
  owner "root"
  group "root"
  mode 0644
  action :create
end
```

Sane Default for Resources

Each resource has a "name attribute."

This corresponds to a parameter attribute as the default value.

Parameter attributes all have default values internal to Chef. Specify your own to be explicit, or to change the default.

Resources also have a default action. The default value depends on the resource type.

Resource Attributes

Resources are data driven through their parameter attributes.

- Packages have versions
- Users have home directories, shells and numeric IDs.

This data can come from multiple sources, either by writing in the code itself or an external source.

Distributing Cookbooks

Cookbooks are packages of source code and they can be distributed all over the place.

- With version control
- With nodes to be configured
- With the Community

Version Control

Use a version control system for your Chef Repository where the cookbooks are stored.

Community best practice is Git.

However, it is beyond the scope of this course to discuss version control strategies in depth.

Storing a cookbook in version control does not make it available to Chef. It must be uploaded to the Chef Server. There are plugins to make this easier.

Nodes and Chef Server

In order for nodes to be configured with Chef, the cookbooks they need must be uploaded to the Chef Server.

```
knife cookbook upload COOKBOOK
```

Applying Cookbooks

To run a recipe on a node with Chef, add the recipe to the node's run list.

Recipes are stored in cookbook directories, and namespaced by the cookbook's directory name.

The cookbook and recipe names can contain alpha-numeric characters, including dash and underscore.

Recipes are stored in Ruby files, with the extension `.rb`. The `.rb` is omitted when adding a recipe to a node.

Add Recipe to a Node

When adding a recipe to a node, combine the cookbook name and the recipe name with `::`. If the `default` recipe is used, it is optional.

```
recipe[webserver]  
recipe[webserver::default]
```

Are equivalent. To use a different recipe, specify it by name:

```
recipe[webserver::different-recipe]
```


Add Recipe to a Node

Use knife to add a recipe to an existing node's run list on the Chef Server.

```
knife node run list add NODE 'recipe[webserver]'
```

Use quotes to prevent shell meta-character expansion.

Run Chef on the node and it will apply the recipe.

Add Recipe to a Node

If the node does not exist on the Chef Server already, the run list can be specified by passing a JSON file with `chef-client -j FILE.json`.

```
{  
  "run_list": [  
    "recipe[webserver]"  
  ]  
}
```

Chef Community Cookbooks

Opscode hosts the Chef Community site where Chef users share cookbooks:

- <http://community.opscode.com/>

Knife includes sub-commands for working with the site.

```
knife cookbook site --help
```

Summary

- Components of Chef cookbooks
- Create new cookbooks
- Write simple recipes
- Recognize and write Chef resources
- Run Chef with a cookbook on a node

Questions

- What are Chef cookbooks?
- What do cookbooks contain?
- Which part of a cookbook determines the version and how to handle dependencies?
- What two kinds of assets are distributed in cookbooks?
- How do recipes get applied to a node?
- What are the four components of a resource?
- How does Chef determine the order to configure resources?
- Student questions?

Additional Resources

- <http://wiki.opscode.com/display/chef/Cookbooks>
- <http://wiki.opscode.com/display/chef/Recipes>
- <http://wiki.opscode.com/display/chef/Resources>

Lab Exercise

Cookbooks, Recipes and Resources

- Create a new cookbook
- Write a simple recipe with two resources
- Run Chef with the cookbook on a node