Perfect! I've created a comprehensive 10-page Basic Chef and Ruby cheat sheet that covers everything a beginner needs to know. Here's what each page covers:

Page 1: Chef Fundamentals and Getting Started

- What Chef is and key concepts
- Installation and first steps
- Basic cookbook structure
- Your first recipe and how to run it

Page 2: Basic Ruby for Chef

- Variables, strings, numbers, booleans
- Arrays and hashes
- Control structures (if/else, case, loops)
- Methods and basic Ruby patterns

Page 3: Essential Chef Resources

- Package management
- Service management
- File and directory operations
- Templates for dynamic configuration
- Execute commands and user management

Page 4: Working with Attributes

- Understanding attribute types and precedence
- Setting and using attributes
- Platform-specific configurations
- Using attributes in templates

Page 5: Templates and Files

- Creating dynamic configuration files
- Template syntax and variables
- Working with static files
- Best practices and conditional templates

Page 6: Cookbook Organization and Structure

- Standard cookbook layout
- Recipe organization patterns
- Metadata and documentation
- Library files for reusable code

Page 7: Environments, Roles, and Run Lists

- Understanding environments for different stages
- Creating and using roles
- Managing run lists
- Bootstrap with environments and roles

Page 8: Basic Testing with Test Kitchen

- Introduction to Test Kitchen
- Configuration and basic commands
- Writing tests with InSpec
- Debugging failed tests

Page 9: Troubleshooting and Debugging

- Common Chef client issues
- Recipe debugging techniques
- Attribute debugging
- Common error messages and solutions

Page 10: Best Practices and Next Steps

- Code organization and error handling
- Security and performance best practices
- Version control and documentation
- Learning path from beginner to advanced

This cheat sheet provides:

- **Practical, copy-paste examples**
- **Step-by-step instructions**
- **Real-world patterns**
- **Troubleshooting help**
- **Clear progression from basic to more complex topics**

It's designed specifically for beginners who are just starting with Chef and need to understand both the Chef concepts and the Ruby language basics required to write effective cookbooks.

Chef and Ruby Basic Cheat Sheet

Page 1: Chef Fundamentals and Getting Started

What is Chef?

Chef is an automation platform that transforms infrastructure into code. It helps you:

- Automate server configuration
- Manage applications and services
- Ensure consistency across environments
- Scale infrastructure efficiently

Key Chef Concepts

- **Node**: A server managed by Chef
- **Recipe**: Code that defines how to configure part of a system
- **Cookbook**: Collection of recipes and supporting files
- **Resource**: Building block that represents a piece of system state
- **Attribute**: Data about a node (like configuration settings)
- **Run List**: Ordered list of recipes to execute on a node

Chef Installation

```
# Install Chef Workstation (includes all tools)
# Download from: https://downloads.chef.io/tools/workstation

# Verify installation
chef --version
knife --version
cookstyle --version

# Generate Chef repository
chef generate repo my-chef-repo
cd my-chef-repo
```

First Steps with Chef

```
# Generate your first cookbook
chef generate cookbook cookbooks/my-app

# Generate a recipe
chef generate recipe cookbooks/my-app web-server

# Generate a template
chef generate template cookbooks/my-app nginx.conf

# Generate an attribute file
chef generate attribute cookbooks/my-app default
```

Basic Cookbook Structure

```
cookbooks/my-app/
├── attributes/
│   └── default.rb         # Default attributes
├── recipes/
│   └── default.rb         # Main recipe
├── templates/
│   └── nginx.conf.erb      # Configuration templates
├── files/
│   └── app-config.json     # Static files
├── metadata.rb            # Cookbook metadata
└── README.md              # Documentation
```

Your First Recipe

```
# cookbooks/my-app/recipes/default.rb

# Install a package
package 'nginx' do
  action :install
end

# Start and enable a service
service 'nginx' do
  action [:enable, :start]
end

# Create a simple file
file '/var/www/html/index.html' do
  content '<h1>Hello from Chef!</h1>'
  owner 'root'
  group 'root'
  mode '0644'
end

# Log a message
log 'Chef setup complete!' do
  level :info
end
```

Running Your First Recipe

```
# Test locally with Chef Solo
sudo chef-client --local-mode --runlist 'recipe[my-app]'

# Or use chef-apply for single recipes
sudo chef-apply -l info recipe_file.rb
```

Basic Knife Commands

```
# Node management
knife node list
knife node show NODE_NAME

# Cookbook management
knife cookbook list
knife cookbook show COOKBOOK_NAME

# Upload cookbook to Chef Server
knife cookbook upload my-app

# Bootstrap a new node
knife bootstrap SERVER_IP -x USERNAME -P PASSWORD --sudo -N NODE_NAME
```

Page 2: Basic Ruby for Chef

Ruby Basics You Need for Chef

```
# Variables - store data
app_name = 'my-application'
port_number = 3000
is_production = true

# Strings - text data
greeting = "Hello World"
app_path = "/opt/#{app_name}"        # String interpolation
config_file = '/etc/myapp/config.yml'

# Numbers
port = 8080
timeout = 30
max_connections = 100
```

```
# Booleans - true/false values
debug_mode = false
ssl_enabled = true
```

Arrays - Lists of Items

```
# Create arrays
servers = ['web1', 'web2', 'web3']
packages = %w[git curl wget vim]     # Quick way to create string array

# Access array items
first_server = servers[0]           # Gets 'web1'
last_server = servers[-1]           # Gets 'web3'

# Add items to array
servers << 'web4'                  # Add to end
servers.push('web5')                # Another way to add

# Loop through arrays
packages.each do |package_name|
  package package_name do
    action :install
  end
end
```

Hashes - Key-Value Pairs

```
# Create hashes
server_config = {
  'hostname' => 'web1.example.com',
  'port' => 80,
  'ssl_enabled' => true
}

# Modern Ruby syntax (symbols as keys)
app_settings = {
  database_host: 'db.example.com',
  database_port: 5432,
  cache_enabled: true
}

# Access hash values
```

```
hostname = server_config['hostname']
db_host = app_settings[:database_host]

# Add new values
server_config['backup_enabled'] = true
app_settings[:log_level] = 'info'
```

Control Structures

```
# If statements
if node['platform'] == 'ubuntu'
  package 'apache2'
elsif node['platform'] == 'centos'
  package 'httpd'
else
  log 'Unsupported platform'
end

# Case statements (better for multiple conditions)
web_server = case node['platform']
             when 'ubuntu', 'debian'
               'apache2'
             when 'centos', 'rhel'
               'httpd'
             else
               'nginx'
             end

# Simple loops
3.times do |i|
  directory "/opt/app#{i}" do
    action :create
  end
end

# Loop with condition
servers.each do |server|
  if server.include?('web')
    log "Configuring web server: #{server}"
  end
end
```

Methods - Reusable Code Blocks

```
# Define a method
def install_web_server(package_name)
  package package_name do
    action :install
  end

  service package_name do
    action [:enable, :start]
  end
end

# Use the method
install_web_server('nginx')

# Method with default parameter
def create_user(username, home_dir = "/home/#{username}")
  user username do
    home home_dir
    action :create
  end
end

# Call with and without optional parameter
create_user('alice')                  # Uses default home
create_user('bob', '/opt/bob')          # Uses custom home
```

Comments and Documentation

```
# Single line comment
package 'nginx'  # Install web server

=begin
Multi-line comment
This is useful for longer explanations
about what the code does
=end

# Good practice: explain WHY, not just WHAT
# Install nginx because we need a reverse proxy for our app
package 'nginx' do
  action :install
end
```

Page 3: Essential Chef Resources

Package Resource - Installing Software

```
# Basic package installation
package 'git' do
  action :install
end

# Install specific version
package 'nginx' do
  version '1.18.0'
  action :install
end

# Install multiple packages
%w[curl wget vim htop].each do |pkg|
  package pkg do
    action :install
  end
end

# Remove a package
package 'apache2' do
  action :remove
end

# Upgrade a package
package 'openssl' do
  action :upgrade
end
```

Service Resource - Managing Services

```
# Start and enable a service
service 'nginx' do
  action [:enable, :start]
end

# Stop and disable a service
service 'apache2' do
  action [:stop, :disable]
end
```

```
# Restart a service
service 'mysql' do
  action :restart
end

# Reload configuration without full restart
service 'nginx' do
  action :reload
end

# Service with supports options
service 'postgresql' do
  supports restart: true, reload: true, status: true
  action [:enable, :start]
end
```

File Resource - Managing Files

```
# Create a simple file
file '/tmp/hello.txt' do
  content 'Hello World!'
  owner 'root'
  group 'root'
  mode '0644'
  action :create
end

# Create file with multi-line content
file '/etc/motd' do
  content <<~EOF
    Welcome to #{node['hostname']}
    This server is managed by Chef
    Please follow company policies
  EOF
  mode '0644'
end

# Delete a file
file '/tmp/old-file.txt' do
  action :delete
end

# Create file only if it doesn't exist
file '/opt/app/first-run.flag' do
```

```
  content 'Application initialized'
  action :create_if_missing
end
```

Directory Resource - Managing Directories

```
# Create a directory
directory '/opt/myapp' do
  owner 'myapp'
  group 'myapp'
  mode '0755'
  action :create
end

# Create directory tree (recursive)
directory '/opt/myapp/logs/archive' do
  recursive true
  owner 'myapp'
  group 'myapp'
  mode '0755'
end

# Remove a directory
directory '/tmp/old-stuff' do
  recursive true
  action :delete
end
```

Template Resource - Dynamic Configuration Files

```
# Create configuration from template
template '/etc/nginx/sites-available/myapp' do
  source 'nginx-site.erb'
  owner 'root'
  group 'root'
  mode '0644'
  variables(
    server_name: 'myapp.example.com',
    port: 3000,
    root_path: '/opt/myapp/public'
  )
end
```

```
# Template with node attributes
template '/etc/myapp/config.yml' do
  source 'app-config.yml.erb'
  owner 'myapp'
  group 'myapp'
  mode '0640'
  variables(
    database_host: node['myapp']['db_host'],
    cache_servers: node['myapp']['cache_servers']
  )
end
```

Execute Resource - Running Commands

```
# Run a simple command
execute 'update-package-cache' do
  command 'apt-get update'
  user 'root'
end

# Run command with conditions
execute 'install-app-dependencies' do
  command 'bundle install'
  cwd '/opt/myapp'
  user 'myapp'
  only_if { File.exist?('/opt/myapp/Gemfile') }
end

# Run command only once
execute 'initialize-database' do
  command 'rake db:setup'
  cwd '/opt/myapp'
  user 'myapp'
  creates '/opt/myapp/db/production.sqlite3'
end
```

User and Group Resources

```
# Create a group
group 'myapp' do
  action :create
end
```

```
# Create a user
user 'myapp' do
  uid 1001
  gid 'myapp'
  home '/opt/myapp'
  shell '/bin/bash'
  action :create
end

# Create system user (for services)
user 'nginx' do
  system true
  shell '/bin/false'
  home '/var/lib/nginx'
  action :create
end
```

Page 4: Working with Attributes

What are Attributes?

Attributes are data about nodes - like configuration settings, system information, and application parameters. They help make your cookbooks flexible and reusable.

Attribute Types and Precedence

```
# From lowest to highest precedence:
# 1. default    - cookbook defaults
# 2. normal     - set during chef run
# 3. override   - role/environment overrides
# 4. automatic  - collected by Ohai (system info)
```

Setting Default Attributes

```
# cookbooks/myapp/attributes/default.rb

# Application settings
default['myapp']['version'] = '1.0.0'
default['myapp']['port'] = 3000
default['myapp']['environment'] = 'production'

# Database settings
```

```
default['myapp']['database']['host'] = 'localhost'
default['myapp']['database']['port'] = 5432
default['myapp']['database']['name'] = 'myapp_production'

# Web server settings
default['myapp']['web_server']['worker_processes'] = 2
default['myapp']['web_server']['timeout'] = 30
```

Using Attributes in Recipes

```
# cookbooks/myapp/recipes/default.rb

# Access attributes
app_version = node['myapp']['version']
app_port = node['myapp']['port']

# Use in resources
package 'myapp' do
  version node['myapp']['version']
  action :install
end

template '/etc/myapp/config.yml' do
  source 'config.yml.erb'
  variables(
    port: node['myapp']['port'],
    database_host: node['myapp']['database']['host'],
    database_port: node['myapp']['database']['port']
  )
end

# Set attributes during chef run
node.normal['myapp']['installed'] = true
node.normal['myapp']['installed_at'] = Time.now.to_s
```

Platform-Specific Attributes

```
# cookbooks/myapp/attributes/default.rb

# Default values
default['myapp']['package_name'] = 'myapp'
default['myapp']['service_name'] = 'myapp'
```

```
# Platform-specific overrides
case node['platform']
when 'ubuntu', 'debian'
  default['myapp']['package_name'] = 'myapp-deb'
  default['myapp']['config_path'] = '/etc/myapp'
when 'centos', 'rhel'
  default['myapp']['package_name'] = 'myapp-rpm'
  default['myapp']['config_path'] = '/etc/sysconfig/myapp'
end
```

Using Attributes in Templates

```
<!-- templates/default/config.yml.erb -->
# MyApp Configuration
application:
  name: <%= @app_name %>
  version: <%= node['myapp']['version'] %>
  port: <%= @port %>
  environment: <%= node['myapp']['environment'] %>

database:
  host: <%= @database_host %>
  port: <%= @database_port %>
  name: <%= @database_name %>

web_server:
  workers: <%= node['myapp']['web_server']['worker_processes'] %>
  timeout: <%= node['myapp']['web_server']['timeout'] %>
```

Automatic Attributes (Ohai)

```
# System information automatically collected
log "Hostname: #{node['hostname']}"
log "Platform: #{node['platform']} #{node['platform_version']}"
log "CPU Count: #{node['cpu']['total']}"
log "Memory: #{node['memory']['total']}"
log "IP Address: #{node['ipaddress']}"

# Use automatic attributes for decisions
if node['memory']['total'].to_i < 2000000  # Less than 2GB
  node.default['myapp']['worker_processes'] = 1
else
  node.default['myapp']['worker_processes'] = node['cpu']['total']
```

```
end
```

Attribute Precedence in Action

```
# In attributes/default.rb
default['myapp']['port'] = 3000

# In a recipe
node.normal['myapp']['port'] = 8080  # This wins over default

# In role (covered later)
override['myapp']['port'] = 9000     # This wins over normal

# Final value will be 9000 due to precedence
log "App will run on port: #{node['myapp']['port']}"
```

Page 5: Templates and Files

Understanding Templates

Templates are ERB (Embedded Ruby) files that generate configuration files dynamically using node data and variables.

Basic Template Usage

```
# In recipe: cookbooks/myapp/recipes/web.rb
template '/etc/nginx/sites-available/myapp' do
  source 'nginx-site.erb'         # Located in templates/default/
  owner 'root'
  group 'root'
  mode '0644'
  variables(
    server_name: 'myapp.example.com',
    port: node['myapp']['port'],
    document_root: '/opt/myapp/public'
  )
  notifies :reload, 'service[nginx]', :delayed
end
```

Template File Structure

```
<!-- templates/default/nginx-site.erb -->
server {
    listen 80;
    server_name <%= @server_name %>;

    location / {
        proxy_pass http://localhost:<%= @port %>;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }

    root <%= @document_root %>;

    # Generated by Chef on <%= Time.now %>
    # Node: <%= node['fqdn'] %>
}
```

Advanced Template Examples

```
<!-- templates/default/database.yml.erb -->
<% if node['myapp']['environment'] == 'production' %>
production:
  adapter: postgresql
  host: <%= @database_host %>
  port: <%= @database_port %>
  database: <%= @database_name %>
  username: <%= @database_user %>
  password: <%= @database_password %>
  pool: 20
  timeout: 5000
<% else %>
development:
  adapter: sqlite3
  database: db/development.sqlite3
  pool: 5
  timeout: 5000
<% end %>
```

Loops in Templates

```
<!-- templates/default/servers.conf.erb -->
# Upstream servers
upstream app_servers {
```

```
<% @servers.each do |server| %>
    server <%= server['ip'] %>:<%= server['port'] %> weight=<%= server['weight'] || 1 %>;
<% end %>
}

# Server list
<% @servers.each_with_index do |server, index| %>
# Server <%= index + 1 %>
server_<%= index %>_host=<%= server['ip'] %>
server_<%= index %>_port=<%= server['port'] %>
<% end %>
```

Using Templates with Complex Data

```
# In recipe
servers = [
  { 'ip' => '10.0.1.10', 'port' => 3000, 'weight' => 3 },
  { 'ip' => '10.0.1.11', 'port' => 3000, 'weight' => 2 },
  { 'ip' => '10.0.1.12', 'port' => 3000, 'weight' => 1 }
]

template '/etc/haproxy/haproxy.cfg' do
  source 'haproxy.cfg.erb'
  variables(
    servers: servers,
    stats_enabled: node['haproxy']['stats_enabled'],
    admin_user: node['haproxy']['stats_user']
  )
  notifies :restart, 'service[haproxy]', :delayed
end
```

Working with Static Files

```
# Copy static files (no variables)
cookbook_file '/opt/myapp/config/settings.json' do
  source 'settings.json'          # From files/default/settings.json
  owner 'myapp'
  group 'myapp'
  mode '0644'
end

# Copy binary files
cookbook_file '/usr/local/bin/myapp-tool' do
```

```
  source 'myapp-tool'
  owner 'root'
  group 'root'
  mode '0755'                    # Executable
end

# Copy with different name
cookbook_file '/etc/ssl/certs/myapp.crt' do
  source 'certificates/production.crt'
  owner 'root'
  group 'root'
  mode '0644'
end
```

Template Best Practices

```
# Use descriptive variable names
template '/etc/myapp/config.yml' do
  source 'application-config.yml.erb'
  variables(
    application_name: node['myapp']['name'],
    listen_port: node['myapp']['port'],
    database_connection_string:
"postgres://#{node['myapp']['db_host']}:#{node['myapp']['db_port']}/#{node['myapp']['db_name']}",
    feature_flags: node['myapp']['features'],
    log_level: node['myapp']['log_level']
  )
end

# Handle missing attributes gracefully
template '/etc/myapp/optional-config.yml' do
  source 'optional-config.yml.erb'
  variables(
    redis_enabled: node['myapp']['redis']['enabled'] || false,
    redis_host: node['myapp']['redis']['host'] || 'localhost',
    cache_ttl: node['myapp']['cache']['ttl'] || 3600
  )
end
```

Conditional Templates

```
<!-- templates/default/app-config.yml.erb -->
# Application Configuration
```

```
app_name: <%= @application_name %>
environment: <%= node['myapp']['environment'] %>

<% if @redis_enabled %>
# Redis Configuration
redis:
  host: <%= @redis_host %>
  port: <%= @redis_port || 6379 %>
  timeout: <%= @redis_timeout || 5 %>
<% end %>

<% if node['myapp']['environment'] == 'development' %>
# Development-only settings
debug: true
log_level: debug
<% else %>
# Production settings
debug: false
log_level: <%= @log_level || 'info' %>
<% end %>

<% unless @feature_flags.nil? || @feature_flags.empty? %>
# Feature Flags
features:
<% @feature_flags.each do |flag, enabled| %>
  <%= flag %>: <%= enabled %>
<% end %>
<% end %>
```

Page 6: Cookbook Organization and Structure

Standard Cookbook Structure

```
cookbooks/myapp/
├── attributes/
│   ├── default.rb          # Default attributes
│   └── database.rb          # Database-specific attributes
├── recipes/
│   ├── default.rb          # Main recipe (entry point)
│   ├── install.rb          # Installation logic
│   ├── configure.rb          # Configuration logic
│   └── service.rb          # Service management
│
```

```
├── templates/
│   └── default/
│       ├── app-config.yml.erb  # Application config template
│       └── nginx-site.erb      # Nginx configuration
├── files/
│   └── default/
│       ├── app-script.sh       # Shell scripts
│       └── settings.json       # Static config files
├── libraries/
│   └── helpers.rb              # Ruby helper methods
├── metadata.rb                 # Cookbook metadata
├── README.md                   # Documentation
└── CHANGELOG.md                # Version history
```

Cookbook Metadata

```
# metadata.rb
name 'myapp'
maintainer 'Your Name'
maintainer_email 'you@example.com'
license 'Apache-2.0'
description 'Installs and configures MyApp'
long_description 'This cookbook handles the complete setup of MyApp including database, web server, and application configuration.'
version '1.0.0'
chef_version '>= 15.0'

# Supported platforms
supports 'ubuntu', '>= 16.04'
supports 'centos', '>= 7.0'

# Dependencies
depends 'nginx', '~> 10.0'
depends 'postgresql', '~> 7.1'

# Issues and source URLs
issues_url 'https://github.com/yourorg/myapp-cookbook/issues'
source_url 'https://github.com/yourorg/myapp-cookbook'
```

Recipe Organization Patterns

```
# recipes/default.rb - Main entry point
include_recipe 'myapp::install'
include_recipe 'myapp::configure'
include_recipe 'myapp::service'

log 'MyApp installation completed successfully!' do
  level :info
end

# recipes/install.rb - Installation logic
# Create application user
user 'myapp' do
  system true
  home '/opt/myapp'
  action :create
end

# Create application directories
directory '/opt/myapp' do
  owner 'myapp'
  group 'myapp'
  mode '0755'
  action :create
end

# Install application package
package 'myapp' do
  version node['myapp']['version']
  action :install
end

# recipes/configure.rb - Configuration logic
# Generate main configuration file
template '/etc/myapp/config.yml' do
  source 'app-config.yml.erb'
  owner 'myapp'
  group 'myapp'
  mode '0640'
  variables(
    database_host: node['myapp']['database']['host'],
    database_port: node['myapp']['database']['port'],
    port: node['myapp']['port']
  )
  notifies :restart, 'service[myapp]', :delayed
end
```

```
# Set up log directory
directory '/var/log/myapp' do
  owner 'myapp'
  group 'myapp'
  mode '0755'
  action :create
end

# recipes/service.rb - Service management
service 'myapp' do
  supports restart: true, reload: true, status: true
  action [:enable, :start]
end
```

Attribute Organization

```
# attributes/default.rb - Main attributes
default['myapp']['version'] = '1.0.0'
default['myapp']['port'] = 3000
default['myapp']['user'] = 'myapp'
default['myapp']['group'] = 'myapp'
default['myapp']['log_level'] = 'info'

# attributes/database.rb - Database-specific attributes
default['myapp']['database']['host'] = 'localhost'
default['myapp']['database']['port'] = 5432
default['myapp']['database']['name'] = 'myapp_production'
default['myapp']['database']['pool_size'] = 5
default['myapp']['database']['timeout'] = 5000
```

Library Files for Reusable Code

```
# libraries/helpers.rb
module MyApp
  module Helpers
    # Helper method to build database connection string
    def database_url
      host = node['myapp']['database']['host']
      port = node['myapp']['database']['port']
      name = node['myapp']['database']['name']
      user = node['myapp']['database']['user']
      password = node['myapp']['database']['password']
```

```ruby
      "postgresql://#{user}:#{password}@#{host}:#{port}/#{name}"
    end

    # Helper to check if app is installed
    def app_installed?
      File.exist?('/opt/myapp/bin/myapp')
    end

    # Helper to get platform-specific package name
    def app_package_name
      case node['platform']
      when 'ubuntu', 'debian'
        'myapp-deb'
      when 'centos', 'rhel'
        'myapp-rpm'
      else
        'myapp'
      end
    end
  end
end

# Make helpers available in recipes
Chef::Recipe.send(:include, MyApp::Helpers)
Chef::Resource.send(:include, MyApp::Helpers)
```

Using Helpers in Recipes

```ruby
# recipes/database.rb
# Install database client package
package app_package_name do
  action :install
end

# Only configure database if app is installed
if app_installed?
  template '/etc/myapp/database.yml' do
    source 'database.yml.erb'
    variables(
      database_url: database_url
    )
  end
end
```

Creating Multiple Cookbook Variants

```ruby
# For different environments, create separate attribute files

# attributes/development.rb
default['myapp']['database']['host'] = 'localhost'
default['myapp']['database']['name'] = 'myapp_development'
default['myapp']['log_level'] = 'debug'
default['myapp']['workers'] = 1

# attributes/production.rb
default['myapp']['database']['host'] = 'db.production.example.com'
default['myapp']['database']['name'] = 'myapp_production'
default['myapp']['log_level'] = 'warn'
default['myapp']['workers'] = 4
```

Documentation Best Practices

~~~markdown
<!-- README.md -->
# MyApp Cookbook

This cookbook installs and configures MyApp.

## Requirements

- Chef 15.0 or higher
- Ubuntu 16.04+ or CentOS 7+

## Attributes

### General Settings
- `node['myapp']['version']` - Version to install (default: '1.0.0')
- `node['myapp']['port']` - Port for application (default: 3000)
- `node['myapp']['user']` - Application user (default: 'myapp')

### Database Settings
- `node['myapp']['database']['host']` - Database host (default: 'localhost')
- `node['myapp']['database']['port']` - Database port (default: 5432)

## Usage

Add to your node's run list:
```json
{
~~~

```
  "run_list": ["recipe[myapp]"]
}
```

Recipes

- `myapp::default` - Installs and configures MyApp
- `myapp::install` - Only installs MyApp
- `myapp::configure` - Only configures MyApp

## Page 7: Environments, Roles, and Run Lists

### Understanding Environments
Environments represent different stages of your application lifecycle (development, staging, production) and allow you to:
- Set environment-specific attributes
- Pin cookbook versions
- Manage different configurations

### Creating Environments
```ruby
# environments/development.rb
name 'development'
description 'Development environment'

# Cookbook version constraints
cookbook_versions({
  'myapp' => '= 1.0.0',
  'nginx' => '~> 10.0'
})

# Environment-specific attributes
default_attributes({
  'myapp' => {
    'database' => {
      'host' => 'dev-db.example.com',
      'name' => 'myapp_dev'
    },
    'log_level' => 'debug',
    'workers' => 1
  }
})

override_attributes({
  'myapp' => {
    'environment' => 'development'
  }
})

# environments/production.rb
name 'production'
description 'Production environment'

cookbook_versions({
  'myapp' => '= 1.0.0',
  'nginx' => '= 10.1.0'
})

default_attributes({
  'myapp' => {
    'database' => {
      'host' => 'prod-db.example.com',
      'name' => 'myapp_production'
    },
    'log_level' => 'warn',
    'workers' => 4
  }
})

override_attributes({
  'myapp' => {
    'environment' => 'production'
  }
})
```

Working with Environments

```
# Create environment on Chef Server
knife environment from file environments/production.rb

# List environments
knife environment list

# Show environment details
```

```
knife environment show production

# Set node's environment
knife node environment set NODE_NAME production
```

Understanding Roles

Roles define what a server does (web server, database server, etc.) and include:

- Run lists (what recipes to run)
- Attributes specific to that role

Creating Roles

```
# roles/database_server.rb
name 'database_server'
description 'Database server role'

run_list(
  'recipe[postgresql::server]',
  'recipe[myapp::database]'
)

default_attributes({
  'postgresql' => {
    'version' => '12',
    'config' => {
      'shared_buffers' => '256MB',
      'max_connections' => 100
    }
  },
  'myapp' => {
    'type' => 'database'
  }
})

# roles/app_server.rb
name 'app_server'
description 'Application server role'

run_list(
  'recipe[myapp::app]',
  'recipe[myapp::monitoring]'
)
```

```
default_attributes({
  'myapp' => {
    'port' => 3000,
    'workers' => 2,
    'type' => 'application'
  }
})
```

Working with Roles

```
# Create role on Chef Server
knife role from file roles/web_server.rb

# List roles
knife role list

# Show role details
knife role show web_server

# Add role to node's run list
knife node run_list add NODE_NAME 'role[web_server]'
```

Understanding Run Lists

Run lists define the order of recipes and roles that Chef executes on a node.

```
# Add recipe to run list
knife node run_list add NODE_NAME 'recipe[myapp]'

# Add role to run list
knife node run_list add NODE_NAME 'role[web_server]'

# Add specific recipe from cookbook
knife node run_list add NODE_NAME 'recipe[myapp::database]'

# Remove from run list
knife node run_list remove NODE_NAME 'recipe[myapp]'

# Set entire run list (replaces existing)
knife node run_list set NODE_NAME 'role[web_server],recipe[myapp::monitoring]'
```

Run List Examples

```
// Simple web server node
{
  "run_list": [
    "recipe[nginx]",
    "recipe[myapp::web]"
  ]
}

// Complex application server
{
  "run_list": [
    "role[base]",
    "recipe[myapp::app]",
    "recipe[monitoring::client]",
    "recipe[backup::client]"
  ]
}

// Database server with multiple roles
{
  "run_list": [
    "role[base]",
    "role[database_server]",
    "recipe[myapp::database_backup]"
  ]
}
```

Environment and Role Interaction

```
# How attributes combine:
# 1. Cookbook defaults
# 2. Environment defaults
# 3. Role defaults
# 4. Environment overrides
# 5. Role overrides
# 6. Node normal attributes

# Example: Final attribute value calculation
# Cookbook default: port = 3000
# Environment default: port = 8080
# Role default: port = 9000
# Result: port = 9000 (role wins over environment)
```

Managing Node Attributes

```
# View node attributes
knife node show NODE_NAME -a myapp

# Edit node attributes directly
knife node edit NODE_NAME

# Set specific attribute
knife exec -E "nodes.transform('name:NODE_NAME') { |n| n.normal['myapp']['port'] = 8080;
n.save }"
```

Bootstrap with Environment and Role

```
# Bootstrap node with environment and role
knife bootstrap SERVER_IP \
  -x ubuntu \
  -i ~/.ssh/id_rsa \
  --sudo \
  -N web-server-01 \
  -E production \
  -r 'role[web_server]'
```

Page 8: Basic Testing with Test Kitchen

What is Test Kitchen?

Test Kitchen is a testing framework that:

- Creates virtual machines for testing
- Runs your cookbooks in isolated environments
- Verifies your infrastructure code works correctly
- Supports multiple platforms and cloud providers

Kitchen Configuration

```
# .kitchen.yml
---
driver:
  name: vagrant                  # Use Vagrant for VMs
```

```
provisioner:
  name: chef_zero                  # Use Chef Zero (local Chef server)
  product_name: chef
  chef_license: accept-silent

verifier:
  name: inspec                     # Use InSpec for testing

platforms:
  - name: ubuntu-20.04             # Test on Ubuntu 20.04
  - name: centos-8                 # Test on CentOS 8

suites:
  - name: default                  # Test suite name
    run_list:
      - recipe[myapp::default]     # What to test
    attributes:
      myapp:
        version: '1.0.0'
        port: 3000
```

Basic Kitchen Commands

```
# List test instances
kitchen list

# Create test VMs
kitchen create

# Run Chef on test VMs
kitchen converge

# Run tests
kitchen verify

# Complete test cycle (create, converge, verify)
kitchen test

# Login to test VM for debugging
kitchen login default-ubuntu-2004

# Destroy test VMs
kitchen destroy
```

```
# Test specific platform
kitchen test default-ubuntu-2004
```

Writing Basic Tests with InSpec

```
# test/integration/default/default_test.rb

# Test that package is installed
describe package('nginx') do
  it { should be_installed }
end

# Test that service is running
describe service('nginx') do
  it { should be_running }
  it { should be_enabled }
end

# Test that port is listening
describe port(80) do
  it { should be_listening }
end

# Test that file exists with correct content
describe file('/var/www/html/index.html') do
  it { should exist }
  it { should be_file }
  its('content') { should match /Hello from Chef/ }
  its('owner') { should eq 'root' }
  its('mode') { should cmp '0644' }
end

# Test that directory exists
describe directory('/opt/myapp') do
  it { should exist }
  its('owner') { should eq 'myapp' }
  its('group') { should eq 'myapp' }
end

# Test that user exists
describe user('myapp') do
  it { should exist }
  its('home') { should eq '/opt/myapp' }
```

```
end
```

More InSpec Test Examples

```
# test/integration/default/web_server_test.rb

# Test configuration file
describe file('/etc/nginx/sites-available/myapp') do
  it { should exist }
  its('content') { should match /server_name myapp.example.com/ }
  its('content') { should match /proxy_pass http:\/\/localhost:3000/ }
end

# Test process is running
describe processes('nginx') do
  it { should exist }
  its('users') { should include 'www-data' }
end

# Test HTTP response
describe http('http://localhost') do
  its('status') { should cmp 200 }
  its('body') { should match /Hello from Chef/ }
end

# Test command output
describe command('nginx -t') do
  its('exit_status') { should eq 0 }
  its('stderr') { should match /syntax is ok/ }
end

# Test log file
describe file('/var/log/nginx/access.log') do
  it { should exist }
  its('owner') { should eq 'www-data' }
end
```

Testing Different Scenarios

```
# .kitchen.yml with multiple test suites
suites:
  - name: default
    run_list:
      - recipe[myapp::default]
    attributes:
      myapp:
        environment: development

  - name: production
    run_list:
      - recipe[myapp::default]
    attributes:
      myapp:
        environment: production
        workers: 4

  - name: database
    run_list:
      - recipe[myapp::database]
    attributes:
      postgresql:
        version: '12'
```

Debugging Failed Tests

```
# Keep VM running after failure for debugging
kitchen test --no-destroy

# Login to failed VM
kitchen login

# Check Chef logs
sudo cat /tmp/kitchen/cache/chef-stacktrace.out

# Run Chef manually for debugging
sudo chef-client -z -o myapp::default

# Check service status
sudo systemctl status nginx

# Check configuration files
cat /etc/nginx/sites-available/myapp

# Run tests manually
cd /tmp/verifier
inspec exec default_test.rb
```

Kitchen Workflow Best Practices

```
# Development workflow
kitchen create                    # Create VM once
kitchen converge                  # Test changes quickly
# Make cookbook changes
kitchen converge                  # Test again
kitchen verify                    # Run tests
kitchen destroy                   # Clean up when done

# Full testing before release
kitchen test                      # Complete test cycle
```

Page 9: Troubleshooting and Debugging

Common Chef Client Issues

Chef Client Won't Start

```
# Check Chef client status
sudo systemctl status chef-client

# Check Chef client logs
sudo journalctl -u chef-client -f

# Run Chef client manually for debugging
sudo chef-client -l debug

# Check Chef configuration
sudo cat /etc/chef/client.rb

# Verify connectivity to Chef Server
sudo knife ssl check -c /etc/chef/client.rb
```

SSL Certificate Issues

```
# Fetch SSL certificates from Chef Server
sudo knife ssl fetch -c /etc/chef/client.rb

# Check SSL configuration
sudo knife ssl check -c /etc/chef/client.rb

# Manual SSL verification
```

```
openssl s_client -connect chef-server.example.com:443 -verify_return_error
```

Node Registration Problems

```
# Check if node is registered
knife node show NODE_NAME

# Re-register node
sudo chef-client -N NODE_NAME

# Check client key
sudo ls -la /etc/chef/client.pem

# Recreate client key if needed
knife client delete NODE_NAME
knife client create NODE_NAME > /etc/chef/client.pem
```

Recipe Debugging Techniques

Using Log Messages

```
# Add debug logging to recipes
log "Starting MyApp installation" do
  level :info
end

log "Database host: #{node['myapp']['database']['host']}" do
  level :debug
end

# Log variable values
app_version = node['myapp']['version']
log "Installing MyApp version: #{app_version}"

# Log conditionals
if node['platform'] == 'ubuntu'
  log "Detected Ubuntu platform, using apt packages"
else
  log "Non-Ubuntu platform: #{node['platform']}"
end
```

Using Ruby Blocks for Debugging

```
# Debug node attributes
```

```
ruby_block 'debug_node_info' do
  block do
    Chef::Log.info("Node platform: #{node['platform']}")
    Chef::Log.info("Node IP: #{node['ipaddress']}")
    Chef::Log.info("Available memory: #{node['memory']['total']}")
    Chef::Log.info("CPU count: #{node['cpu']['total']}")
  end
  only_if { node['myapp']['debug_mode'] }
end

# Debug file contents
ruby_block 'check_config_file' do
  block do
    if File.exist?('/etc/myapp/config.yml')
      content = File.read('/etc/myapp/config.yml')
      Chef::Log.info("Config file content: #{content}")
    else
      Chef::Log.warn("Config file does not exist")
    end
  end
end
```

Testing Resource Conditions

```
# Debug guard conditions
execute 'test_command' do
  command 'echo "This will run"'
  only_if do
    result = File.exist?('/opt/myapp/bin/myapp')
    Chef::Log.info("App binary exists: #{result}")
    result
  end
end

# Debug file permissions
file '/tmp/debug.txt' do
  content 'Debug file'
  owner 'root'
  group 'root'
  mode '0644'
  action :create
  verify do |file_path|
    stat = File.stat(file_path)
    Chef::Log.info("File owner: #{stat.uid}")
    Chef::Log.info("File group: #{stat.gid}")
    Chef::Log.info("File mode: #{sprintf('%o', stat.mode)}")
    true  # Return true to continue
  end
end
```

Attribute Debugging

```
# Debug attribute precedence
ruby_block 'debug_attributes' do
  block do
    Chef::Log.info("Default port: #{node.default['myapp']['port']}")
    Chef::Log.info("Normal port: #{node.normal['myapp']['port']}")
    Chef::Log.info("Override port: #{node.override['myapp']['port']}")
    Chef::Log.info("Final port value: #{node['myapp']['port']}")
  end
end

# Show all myapp attributes
ruby_block 'show_myapp_attributes' do
  block do
    Chef::Log.info("MyApp attributes: #{node['myapp'].to_hash}")
  end
end
```

Common Error Messages and Solutions

"No such file or directory"

```
# Problem: File doesn't exist
file '/nonexistent/path/config.yml' do
  content 'test'
end

# Solution: Create directory first
directory '/opt/myapp/config' do
  recursive true
  action :create
end

file '/opt/myapp/config/config.yml' do
  content 'test'
end
```

"Permission denied"

```
# Problem: Wrong user/permissions
execute 'app_command' do
  command '/opt/myapp/bin/start'
  user 'root'  # This might fail if app needs specific user
end

# Solution: Use correct user
execute 'app_command' do
  command '/opt/myapp/bin/start'
  user 'myapp'
  group 'myapp'
  cwd '/opt/myapp'
end
```

"Service won't start"

```
# Debug service issues
sudo systemctl status myapp
sudo journalctl -u myapp -f

# Check service file exists
ls -la /etc/systemd/system/myapp.service

# Reload systemd after changes
sudo systemctl daemon-reload
sudo systemctl start myapp
```

"Template variables undefined"

```
# Problem: Missing template variable
template '/etc/myapp/config.yml' do
  source 'config.yml.erb'
  variables(
    port: node['myapp']['port']
    # Missing database_host variable
  )
end

# Solution: Provide all needed variables
template '/etc/myapp/config.yml' do
  source 'config.yml.erb'
  variables(
    port: node['myapp']['port'],
    database_host: node['myapp']['database']['host'] || 'localhost'
  )
end
```

Using Chef Shell for Interactive Debugging

```
# Start Chef shell
sudo chef-shell -z

# In chef-shell, test recipes interactively
chef > recipe_mode
chef:recipe > package 'git'
chef:recipe > run_chef
```

Checking Resource Status

```
# Verify resource execution
ruby_block 'check_installation' do
  block do
    # Check if package was installed
    if system('which nginx > /dev/null 2>&1')
      Chef::Log.info("Nginx successfully installed")
    else
      Chef::Log.error("Nginx installation failed")
    end

    # Check if service is running
    if system('systemctl is-active nginx > /dev/null 2>&1')
      Chef::Log.info("Nginx service is running")
    else
      Chef::Log.warn("Nginx service is not running")
    end
  end
end
```

Dry Run Testing

```
# Test without making changes (why-run mode)
sudo chef-client --why-run

# Test specific recipe
sudo chef-client --why-run -o myapp::database
```

```
# Test with increased verbosity
sudo chef-client --why-run -l debug
```

Page 10: Best Practices and Next Steps

Cookbook Development Best Practices

Code Organization

```
# Good: Descriptive resource names
package 'install_nginx_web_server' do
  package_name 'nginx'
  action :install
end

service 'start_nginx_web_server' do
  service_name 'nginx'
  action [:enable, :start]
end

# Good: Use meaningful variable names
database_connection_host = node['myapp']['database']['host']
application_port = node['myapp']['port']
log_file_path = "/var/log/#{node['myapp']['name']}/application.log"

# Good: Group related resources
# Install and configure web server
package 'nginx' do
  action :install
end

template '/etc/nginx/nginx.conf' do
  source 'nginx.conf.erb'
  notifies :reload, 'service[nginx]', :delayed
end

service 'nginx' do
  action [:enable, :start]
end
```

Error Handling and Resilience

```
# Use ignore_failure for non-critical resources
package 'optional-package' do
  action :install
  ignore_failure true
end

# Use guards to prevent errors
execute 'database_migration' do
  command 'rake db:migrate'
  cwd '/opt/myapp'
  user 'myapp'
  only_if { File.exist?('/opt/myapp/Rakefile') }
  only_if { Dir.exist?('/opt/myapp/db/migrate') }
end

# Handle missing attributes gracefully
database_host = node['myapp']['database']['host'] || 'localhost'
database_port = node['myapp']['database']['port'] || 5432

template '/etc/myapp/database.yml' do
  source 'database.yml.erb'
  variables(
    host: database_host,
    port: database_port
  )
  only_if { database_host }
end
```

Resource Notifications

```
# Use delayed notifications for better performance
template '/etc/nginx/nginx.conf' do
  source 'nginx.conf.erb'
  notifies :reload, 'service[nginx]', :delayed  # Waits until end of run
end

template '/etc/nginx/sites-available/myapp' do
  source 'myapp-site.erb'
  notifies :reload, 'service[nginx]', :delayed  # Same service, one reload
end

# Use immediate notifications when needed
execute 'update_package_cache' do
  command 'apt-get update'
  notifies :install, 'package[nginx]', :immediately
end

# Use subscribes for reverse notifications
service 'nginx' do
  action [:enable, :start]
  subscribes :reload, 'template[/etc/nginx/nginx.conf]', :delayed
end
```

Security Best Practices

File Permissions

```
# Secure configuration files
template '/etc/myapp/secrets.yml' do
  source 'secrets.yml.erb'
  owner 'myapp'
  group 'myapp'
  mode '0600'    # Only owner can read/write
  sensitive true # Don't show content in logs
end

# Executable scripts
cookbook_file '/usr/local/bin/myapp-backup' do
  source 'backup-script.sh'
  owner 'root'
  group 'root'
  mode '0755'    # Owner can execute, others can read
end

# Secure directories
directory '/opt/myapp/config' do
  owner 'myapp'
  group 'myapp'
  mode '0750'    # Owner full access, group read/execute, others none
end
```

User Management

```
# Create dedicated users for applications
user 'myapp' do
  system true        # System user (no login)
  shell '/bin/false' # No shell access
  home '/opt/myapp'
  action :create
end

# Add users to specific groups
group 'ssl-cert' do
  members ['myapp', 'nginx']
  action :create
end
```

Performance Best Practices

Efficient Resource Usage

```
# Install multiple packages at once
%w[git curl wget vim htop].each do |package_name|
  package package_name do
    action :install
  end
end

# Better: Use array for batch installation
package %w[git curl wget vim htop] do
  action :install
end

# Use lazy evaluation for expensive operations
template '/etc/myapp/config.yml' do
  source 'config.yml.erb'
  variables lazy {
    {
      servers: search(:node, "role:web_server"), # Only runs when needed
      timestamp: Time.now
    }
  }
end
```

Minimize Chef Runs

```
# Use guards to skip unnecessary work
execute 'compile_application' do
  command 'make && make install'
  cwd '/opt/myapp/src'
  not_if { File.exist?('/opt/myapp/bin/myapp') }
end
```

```
# Check if restart is actually needed
service 'myapp' do
  action :nothing  # Don't start automatically
end

template '/etc/myapp/config.yml' do
  source 'config.yml.erb'
  notifies :restart, 'service[myapp]', :delayed
  only_if do
    # Only create template if it would actually change
    new_content = render_template_content
    current_content = File.read('/etc/myapp/config.yml') rescue ''
    new_content != current_content
  end
end
```

Version Control and Collaboration

Git Workflow for Cookbooks

```
# Initialize cookbook repository
cd cookbooks/myapp
git init
git add .
git commit -m "Initial cookbook structure"

# Create feature branch
git checkout -b feature/add-database-support
# Make changes
git add .
git commit -m "Add database configuration recipe"
git push origin feature/add-database-support

# Tag releases
git tag -a v1.0.0 -m "Release version 1.0.0"
git push origin v1.0.0
```

Documentation Standards

```
<!-- cookbooks/myapp/README.md -->
# MyApp Cookbook

## Description
```

This cookbook installs and configures MyApp web application.

## Requirements
- Chef 15+
- Ubuntu 18.04+ or CentOS 7+

## Attributes
### Required
- `node['myapp']['database']['host']` - Database server hostname

### Optional
- `node['myapp']['port']` - Application port (default: 3000)
- `node['myapp']['workers']` - Number of workers (default: 2)

## Usage
```json
{
  "myapp": {
    "database": {
      "host": "db.example.com"
    },
    "port": 8080
  }
}
```

Testing

kitchen test

### Learning Path and Next Steps

#### Beginner to Intermediate
1. **Master the basics**: Resources, attributes, templates
2. **Learn testing**: Test Kitchen and InSpec
3. **Understand environments and roles**
4. **Practice troubleshooting**

#### Intermediate to Advanced
1. **Custom resources**: Create reusable components
2. **Library development**: Write helper methods
3. **Advanced testing**: ChefSpec unit tests
4. **CI/CD integration**: Automated testing and deployment

#### Recommended Resources
- **Official Documentation**: https://docs.chef.io/
- **Learn Chef Rally**: https://learn.chef.io/
- **Chef Community**: https://community.chef.io/
- **Supermarket**: https://supermarket.chef.io/ (community cookbooks)

#### Common Tools to Learn
- **Cookstyle**: Ruby and Chef style checking
- **Foodcritic**: Chef-specific linting
- **ChefSpec**: Unit testing for Chef
- **Berkshelf**: Cookbook dependency management

### Quick Command Reference
```bash
# Cookbook generation
chef generate cookbook COOKBOOK_NAME
chef generate recipe COOKBOOK_NAME RECIPE_NAME
chef generate template COOKBOOK_NAME TEMPLATE_NAME

# Local testing
sudo chef-client --local-mode --runlist 'recipe[COOKBOOK]'
kitchen test

# Chef Server operations
knife cookbook upload COOKBOOK_NAME
knife node list
knife bootstrap SERVER_IP -x USER -i KEY_FILE -N NODE_NAME

# Style and linting
cookstyle cookbooks/COOKBOOK_NAME
foodcritic cookbooks/COOKBOOK_NAME

# Testing
kitchen list
kitchen converge
kitchen verify
kitchen test

## Common Patterns Summary

# Basic resource pattern
RESOURCE_TYPE 'resource_name' do
  property1 'value1'
  property2 'value2'
  action :action_name
end

# Conditional execution
if condition
  # resources here
end

# Loops
array.each do |item|
  # resources using item
end

# Notifications
resource1 do
  # properties
  notifies :action, 'resource2', :timing
end

# Guards
resource do
  # properties
  only_if { condition }
  not_if { opposite_condition }
end
```

Quick Reference Card:

- **Official Docs**: https://docs.chef.io/
- **Learn Chef**: https://learn.chef.io/
- **Community**: https://community.chef.io/
- **Supermarket**: https://supermarket.chef.io/
- **GitHub**: https://github.com/chef

Remember:

- Start simple and build complexity gradually
- Test early and often with Test Kitchen
- Use version control for all cookbook development
- Read existing cookbooks to learn patterns
- Ask for help in the Chef community forums/web_server.rb name 'web_server' description 'Web server role'

What recipes this role should run

run_list('recipe[nginx]', 'recipe[myapp::web]')

Role-specific attributes

default_attributes({ 'nginx' => { 'worker_processes' => 2, 'keepalive_timeout' => 65 }, 'myapp' => { 'port' => 3000, 'type' => 'web' } })

override_attributes({ 'nginx' => { 'server_tokens' => 'off' } })

~~~
```ruby
# roles
~~~