Chef and Ruby Intermediate Cheat Sheet

Page 1: Chef Basics and Resource Management

Essential Chef Commands

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

# Cookbook operations
knife cookbook list
knife cookbook upload COOKBOOK_NAME
knife cookbook download COOKBOOK_NAME
knife cookbook create COOKBOOK_NAME

# Bootstrap and SSH
knife bootstrap SERVER_IP -x USERNAME -P PASSWORD --sudo
knife ssh "name:*" "sudo chef-client" -x USERNAME -P PASSWORD

# Search operations
knife search node "platform:ubuntu"
knife search node "role:web_server"
knife search node "chef_environment:production"
```

Core Chef Resources

```
# Package management
package 'nginx' do
  action :install
  version '1.18.0'
end

# Service management
service 'nginx' do
  action [:enable, :start]
  supports restart: true, reload: true
```

```
end

# File operations
file '/etc/nginx/nginx.conf' do
  content lazy { IO.read('/path/to/template') }
  owner 'root'
  group 'root'
  mode '0644'
  action :create
  notifies :restart, 'service[nginx]', :delayed
end

# Template rendering
template '/etc/myapp/config.yml' do
  source 'config.yml.erb'
  variables(
    database_host: node['myapp']['db_host'],
    database_port: node['myapp']['db_port']
  )
  owner 'myapp'
  group 'myapp'
  mode '0600'
end

# Directory creation
directory '/opt/myapp' do
  owner 'myapp'
  group 'myapp'
  mode '0755'
  recursive true
  action :create
end

# Execute commands
execute 'update-package-cache' do
  command 'apt-get update'
  user 'root'
  only_if { node['platform'] == 'ubuntu' }
end

# User management
user 'myapp' do
  uid 1001
  gid 'myapp'
```

```
  home '/opt/myapp'
  shell '/bin/bash'
  action :create
end

group 'myapp' do
  gid 1001
  members ['myapp', 'deploy']
  action :create
end
```

Resource Notifications and Guards

```
# Notifications
template '/etc/nginx/sites-available/myapp' do
  source 'nginx-site.erb'
  notifies :reload, 'service[nginx]', :immediately
  notifies :run, 'execute[test-config]', :before
end

# Guards
package 'git' do
  action :install
  not_if { File.exist?('/usr/bin/git') }
end

execute 'compile-app' do
  command 'make install'
  cwd '/opt/myapp/src'
  only_if { File.exist?('/opt/myapp/src/Makefile') }
  only_if { node['myapp']['compile_from_source'] }
end
```

Page 2: Chef Attributes, Roles, and Environments

Attribute Precedence (Low to High)

1. Default attributes in cookbooks
2. Environment default attributes
3. Role default attributes
4. Attributes applied via Policyfile

5. Normal attributes (set via recipes)
6. Role override attributes
7. Environment override attributes
8. Automatic attributes (ohai)

Working with Attributes

```
# Setting attributes in recipes
node.default['myapp']['version'] = '2.1.0'
node.override['myapp']['port'] = 8080
node.normal['myapp']['installed'] = true

# Accessing attributes
app_version = node['myapp']['version']
db_password = node['database']['users']['app']['password']

# Conditional attributes
case node['platform']
when 'ubuntu', 'debian'
  node.default['myapp']['package_name'] = 'myapp-deb'
when 'centos', 'redhat'
  node.default['myapp']['package_name'] = 'myapp-rpm'
end

# Deep merge attributes
node.default['myapp'].merge!({
  'database' => {
    'host' => '10.0.1.100',
    'port' => 5432
  },
  'cache' => {
    'redis_url' => 'redis://10.0.1.101:6379'
  }
})
```

Role Definitions

```
# roles/web_server.rb
name 'web_server'
description 'Web server role'
run_list 'recipe[nginx]', 'recipe[myapp::web]'
default_attributes(
  'nginx' => {
    'worker_processes' => 4,
```

```
    'keepalive_timeout' => 65
  },
  'myapp' => {
    'web_port' => 3000,
    'worker_count' => 2
  }
)
override_attributes(
  'myapp' => {
    'environment' => 'production'
  }
)
```

Environment Definitions

```
# environments/production.rb
name 'production'
description 'Production environment'
cookbook_versions(
  'myapp' => '= 2.1.0',
  'nginx' => '~> 1.8.0'
)
default_attributes(
  'myapp' => {
    'database_pool_size' => 20,
    'cache_ttl' => 3600
  }
)
override_attributes(
  'nginx' => {
    'worker_processes' => 8
  }
)
```

Data Bags

```
# Create data bag
knife data bag create users

# Create data bag item
knife data bag create users alice --secret-file /path/to/secret

# Access data bags in recipes
```

```
users = data_bag('users')
alice_data = data_bag_item('users', 'alice')

# Encrypted data bags
secret = Chef::EncryptedDataBagItem.load_secret('/path/to/secret')
alice_encrypted = Chef::EncryptedDataBagItem.load('users', 'alice', secret)
```

Page 3: Ruby Fundamentals for Chef

Ruby Basics

```
# Variables and constants
app_name = 'myapp'
APP_VERSION = '2.1.0'

# Arrays and hashes
servers = ['web1', 'web2', 'web3']
config = {
  'database' => {
    'host' => 'db1.example.com',
    'port' => 5432
  },
  'cache' => {
    'servers' => ['cache1', 'cache2']
  }
}

# String interpolation
log_file = "/var/log/#{app_name}/application.log"
connection_string =
"postgresql://user:pass@#{config['database']['host']}:#{config['database']['port']}/#{app_name}"

# Symbols vs strings
# Symbols are immutable and memory efficient
config = {
  database: {
    host: 'localhost',
    port: 5432
  }
}
```

Control Structures

```
# Conditionals
if node['platform'] == 'ubuntu'
  package_manager = 'apt'
elsif node['platform'] == 'centos'
  package_manager = 'yum'
else
  package_manager = 'unknown'
end

# Case statements
service_command = case node['platform']
                  when 'ubuntu', 'debian'
                    'systemctl'
                  when 'centos', 'rhel'
                    'service'
                  else
                    'unknown'
                  end

# Loops
servers.each do |server|
  log "Processing server: #{server}"
end

config['cache']['servers'].each_with_index do |server, index|
  template "/etc/cache/server#{index}.conf" do
    source 'cache-server.erb'
    variables(server_name: server)
  end
end

# Times loop
3.times do |i|
  directory "/opt/app/worker#{i}" do
    action :create
  end
end
```

Methods and Blocks

```
# Method definition
def install_package(package_name, version = nil)
```

```
  package package_name do
    action :install
    version version if version
  end
end

# Call method
install_package('nginx', '1.18.0')
install_package('curl')

# Blocks with yield
def configure_service(service_name)
  service service_name do
    action [:enable, :start]
  end

  yield if block_given?

  service service_name do
    action :restart
  end
end

# Using the method with a block
configure_service('nginx') do
  template '/etc/nginx/nginx.conf' do
    source 'nginx.conf.erb'
  end
end
```

Error Handling

```
begin
  file_content = File.read('/etc/myapp/config.yml')
  config_data = YAML.load(file_content)
rescue Errno::ENOENT
  Chef::Log.warn("Config file not found, using defaults")
  config_data = default_config
rescue YAML::ParserError => e
  Chef::Log.error("Invalid YAML in config file: #{e.message}")
  raise
ensure
  Chef::Log.info("Configuration loading completed")
end
```

```
# Inline rescue
database_host = node['database']['host'] rescue 'localhost'
```

Page 4: Advanced Chef Patterns and Ruby Classes

Custom Resources (HWRP - Heavy Weight Resource Provider)

```
# resources/myapp_service.rb
provides :myapp_service
unified_mode true

property :service_name, String, name_property: true
property :port, Integer, default: 3000
property :user, String, default: 'myapp'
property :config_template, String, default: 'service.conf.erb'

action :create do
  # Create user
  user new_resource.user do
    system true
    home "/opt/#{new_resource.service_name}"
    action :create
  end

  # Create service directory
  directory "/opt/#{new_resource.service_name}" do
    owner new_resource.user
    group new_resource.user
    mode '0755'
    action :create
  end

  # Deploy configuration
  template "/etc/#{new_resource.service_name}.conf" do
    source new_resource.config_template
    owner new_resource.user
    group new_resource.user
    mode '0644'
    variables(
      service_name: new_resource.service_name,
      port: new_resource.port,
```

```
      user: new_resource.user
    )
    notifies :restart, "service[#{new_resource.service_name}]", :delayed
  end

  # Create systemd service
  template "/etc/systemd/system/#{new_resource.service_name}.service" do
    source 'systemd.service.erb'
    owner 'root'
    group 'root'
    mode '0644'
    variables(
      service_name: new_resource.service_name,
      user: new_resource.user,
      working_directory: "/opt/#{new_resource.service_name}"
    )
    notifies :run, 'execute[systemctl-daemon-reload]', :immediately
  end

  execute 'systemctl-daemon-reload' do
    command 'systemctl daemon-reload'
    action :nothing
  end

  service new_resource.service_name do
    action [:enable, :start]
  end
end

action :remove do
  service new_resource.service_name do
    action [:stop, :disable]
  end

  file "/etc/systemd/system/#{new_resource.service_name}.service" do
    action :delete
  end
end
```

Using Custom Resources

```
# In a recipe
myapp_service 'api-server' do
  port 8080
```

```
  user 'api'
  config_template 'api-service.conf.erb'
  action :create
end

myapp_service 'worker-service' do
  port 8081
  user 'worker'
  action :create
end
```

Ruby Classes and Modules

```
# libraries/myapp_helpers.rb
module MyApp
  module Helpers
    def database_connection_string
      host = node['myapp']['database']['host']
      port = node['myapp']['database']['port']
      database = node['myapp']['database']['name']
      username = node['myapp']['database']['username']
      password = node['myapp']['database']['password']

      "postgresql://#{username}:#{password}@#{host}:#{port}/#{database}"
    end

    def app_servers
      search(:node, "role:app_server AND chef_environment:#{node.chef_environment}")
    end

    def generate_secret_key
      require 'securerandom'
      SecureRandom.hex(32)
    end
  end
end

# Include in Recipe class
class Chef::Recipe
  include MyApp::Helpers
end

# Include in Resource class
class Chef::Resource
```

```
  include MyApp::Helpers
end
```

Configuration Management Class

```
# libraries/config_manager.rb
class ConfigManager
  attr_reader :node, :config_data

  def initialize(node)
    @node = node
    @config_data = load_base_config
  end

  def load_base_config
    {
      'application' => {
        'name' => node['myapp']['name'] || 'myapp',
        'version' => node['myapp']['version'] || '1.0.0',
        'environment' => node.chef_environment
      },
      'server' => {
        'port' => node['myapp']['port'] || 3000,
        'workers' => node['myapp']['workers'] || node['cpu']['total']
      }
    }
  end

  def merge_environment_config!
    env_config = case node.chef_environment
                 when 'production'
                   production_config
                 when 'staging'
                   staging_config
                 else
                   development_config
                 end

    deep_merge!(config_data, env_config)
  end

  def to_yaml
    require 'yaml'
    YAML.dump(config_data)
```

```
end

def to_json
  require 'json'
  JSON.pretty_generate(config_data)
end

private

def production_config
  {
    'server' => {
      'workers' => node['cpu']['total'] * 2,
      'timeout' => 30
    },
    'database' => {
      'pool_size' => 20,
      'timeout' => 5000
    }
  }
end

def staging_config
  {
    'server' => {
      'workers' => 2,
      'timeout' => 60
    },
    'database' => {
      'pool_size' => 5,
      'timeout' => 10000
    }
  }
end

def development_config
  {
    'server' => {
      'workers' => 1,
      'timeout' => 120
    },
    'database' => {
      'pool_size' => 2,
      'timeout' => 15000
```

```
    }
   }
  end

  def deep_merge!(target_hash, source_hash)
    source_hash.each do |key, value|
      if target_hash[key].is_a?(Hash) && value.is_a?(Hash)
        deep_merge!(target_hash[key], value)
      else
        target_hash[key] = value
      end
    end
    target_hash
  end
end
```

Using the Configuration Class

```
# In a recipe
config_manager = ConfigManager.new(node)
config_manager.merge_environment_config!

file '/opt/myapp/config/application.yml' do
  content config_manager.to_yaml
  owner 'myapp'
  group 'myapp'
  mode '0640'
  action :create
end
```

Page 5: Testing, Debugging, and Best Practices

ChefSpec Testing

```
# spec/unit/recipes/default_spec.rb
require 'spec_helper'

describe 'myapp::default' do
  let(:chef_run) do
    ChefSpec::SoloRunner.new(platform: 'ubuntu', version: '20.04') do |node|
      node.normal['myapp']['version'] = '2.1.0'
      node.normal['myapp']['port'] = 3000
```

```
    end.converge(described_recipe)
  end

  it 'installs nginx package' do
    expect(chef_run).to install_package('nginx')
  end

  it 'creates myapp user' do
    expect(chef_run).to create_user('myapp').with(
      uid: 1001,
      home: '/opt/myapp'
    )
  end

  it 'renders application config template' do
    expect(chef_run).to create_template('/opt/myapp/config.yml').with(
      source: 'config.yml.erb',
      owner: 'myapp',
      group: 'myapp',
      mode: '0640'
    )
  end

  it 'notifies nginx service to restart' do
    template = chef_run.template('/etc/nginx/sites-available/myapp')
    expect(template).to notify('service[nginx]').to(:restart).delayed
  end

  context 'when on CentOS' do
    let(:chef_run) do
      ChefSpec::SoloRunner.new(platform: 'centos', version: '7').converge(described_recipe)
    end

    it 'uses yum package manager' do
      expect(chef_run).to install_yum_package('nginx')
    end
  end
end
```

Test Kitchen Configuration

```
# .kitchen.yml
driver:
  name: vagrant
```

```
provisioner:
  name: chef_zero
  product_name: chef
  chef_license: accept-silent

verifier:
  name: inspec

platforms:
  - name: ubuntu-20.04
  - name: centos-7

suites:
  - name: default
    run_list:
      - recipe[myapp::default]
    attributes:
      myapp:
        version: 2.1.0
        port: 3000
    verifier:
      inspec_tests:
        - test/integration/default
```

InSpec Testing

```
# test/integration/default/myapp_test.rb
describe package('nginx') do
  it { should be_installed }
end

describe service('nginx') do
  it { should be_running }
  it { should be_enabled }
end

describe user('myapp') do
  it { should exist }
  its('uid') { should eq 1001 }
  its('home') { should eq '/opt/myapp' }
end

describe file('/opt/myapp/config.yml') do
```

```
  it { should exist }
  its('owner') { should eq 'myapp' }
  its('group') { should eq 'myapp' }
  its('mode') { should cmp '0640' }
end

describe port(3000) do
  it { should be_listening }
  its('protocols') { should include 'tcp' }
end

describe command('curl -f http://localhost:3000/health') do
  its('exit_status') { should eq 0 }
  its('stdout') { should match /OK/ }
end
```

Debugging Techniques

```
# Logging levels
Chef::Log.debug("Debug information: #{variable}")
Chef::Log.info("Application starting on port #{port}")
Chef::Log.warn("Configuration file not found, using defaults")
Chef::Log.error("Database connection failed: #{error.message}")
Chef::Log.fatal("Critical error occurred")

# Breakpoints for debugging
require 'pry'; binding.pry

# Node inspection
log "Node platform: #{node['platform']}" do
  level :info
end

log "All node attributes:" do
  message node.to_hash
  level :debug
end

# Resource inspection
ruby_block 'debug_resources' do
  block do
    Chef::Log.info("Current run_list: #{node.run_list}")
    Chef::Log.info("Expanded run_list: #{node.expanded_run_list}")
  end
```

```
end
```

Best Practices

Recipe Organization

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

# recipes/install.rb - Installation logic
package 'myapp' do
  version node['myapp']['version']
  action :install
end

# recipes/configure.rb - Configuration logic
template '/etc/myapp/config.yml' do
  source 'config.yml.erb'
  variables lazy {
    {
      database_url: database_connection_string,
      cache_servers: app_servers.map(&:name)
    }
  }
end

# recipes/service.rb - Service management
service 'myapp' do
  action [:enable, :start]
  subscribes :restart, 'template[/etc/myapp/config.yml]', :delayed
end
```

Attribute Management

```
# Good: Use appropriate precedence levels
# attributes/default.rb
default['myapp']['version'] = '2.1.0'
default['myapp']['port'] = 3000
default['myapp']['workers'] = node['cpu']['total']

# Use computed attributes carefully
```

```
default['myapp']['memory_per_worker'] = lazy {
  total_memory = node['memory']['total'].to_i
  worker_count = node['myapp']['workers']
  (total_memory / worker_count * 0.8).to_i
}
```

Resource Efficiency

```
# Good: Use lazy evaluation for dynamic content
template '/etc/myapp/database.yml' do
  source 'database.yml.erb'
  variables lazy {
    {
      servers: search(:node, 'role:database'),
      password: data_bag_item('secrets', 'database')['password']
    }
  }
end

# Good: Use subscribes instead of notifies when possible
service 'myapp' do
  action [:enable, :start]
  subscribes :restart, 'template[/etc/myapp/config.yml]'
  subscribes :reload, 'template[/etc/myapp/logging.conf]'
end
```

Common Troubleshooting Commands

```
# Chef client debugging
chef-client -W -l debug
chef-client --why-run

# Node debugging
knife node show NODE_NAME -a platform
knife node show NODE_NAME -a run_list

# Recipe testing
foodcritic cookbooks/myapp
rubocop cookbooks/myapp

# Test Kitchen
kitchen test
kitchen converge
kitchen login
```

```
kitchen destroy
```

Page 6: Advanced Ruby Patterns and Metaprogramming

Ruby Metaprogramming for Chef

```
# Dynamic method definition
class ChefHelper
  PLATFORMS = %w[ubuntu centos rhel debian].freeze

  # Create platform-specific methods dynamically
  PLATFORMS.each do |platform|
    define_method "#{platform}?" do
      node['platform'] == platform
    end

    define_method "#{platform}_package_manager" do
      case platform
      when 'ubuntu', 'debian'
        'apt'
      when 'centos', 'rhel'
        'yum'
      end
    end
  end
end

# Usage in recipes
helper = ChefHelper.new
if helper.ubuntu?
  package_manager = helper.ubuntu_package_manager
end
```

Method Missing and Dynamic Dispatch

```
# libraries/service_manager.rb
class ServiceManager
  def initialize(node)
    @node = node
    @services = {}
  end
```

```
  def method_missing(method_name, *args, &block)
    if method_name.to_s.end_with?('_service')
      service_name = method_name.to_s.sub('_service', '')
      define_service(service_name, *args, &block)
    else
      super
    end
  end

  def respond_to_missing?(method_name, include_private = false)
    method_name.to_s.end_with?('_service') || super
  end

  private

  def define_service(name, options = {})
    @services[name] = {
      port: options[:port] || 3000,
      user: options[:user] || name,
      config: options[:config] || {}
    }

    create_service_resources(name, @services[name])
  end

  def create_service_resources(name, config)
    # This would be called from within a recipe context
    user config[:user] do
      system true
      home "/opt/#{name}"
    end

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

# Usage in recipe
service_manager = ServiceManager.new(node)
service_manager.api_service(port: 8080, user: 'api')
service_manager.worker_service(port: 8081, user: 'worker')
```

Ruby Modules and Mixins

```
# libraries/deployment_helpers.rb
module DeploymentHelpers
  def git_deploy(repo_url, deploy_path, revision = 'master')
    git deploy_path do
      repository repo_url
      revision revision
      action :sync
      user deployment_user
      group deployment_group
    end
  end

  def create_deployment_structure(app_name)
    %w[releases shared current].each do |dir|
      directory "/opt/#{app_name}/#{dir}" do
        owner deployment_user
        group deployment_group
        mode '0755'
        recursive true
      end
    end

    %w[log tmp pids].each do |shared_dir|
      directory "/opt/#{app_name}/shared/#{shared_dir}" do
        owner deployment_user
        group deployment_group
        mode '0755'
        recursive true
      end
    end
  end

  def symlink_shared_files(app_name, files)
    files.each do |file|
      link "/opt/#{app_name}/current/#{file}" do
        to "/opt/#{app_name}/shared/#{file}"
        owner deployment_user
        group deployment_group
      end
    end
  end

  private
```

```
  def deployment_user
    node['deployment']['user'] || 'deploy'
  end

  def deployment_group
    node['deployment']['group'] || 'deploy'
  end
end

# Include in recipe
Chef::Recipe.send(:include, DeploymentHelpers)

# Usage
create_deployment_structure('myapp')
git_deploy('https://github.com/mycompany/myapp.git', '/opt/myapp/releases/v1.2.3', 'v1.2.3')
symlink_shared_files('myapp', %w[config/database.yml log tmp/pids])
```

Advanced String and Array Manipulation

```
# String manipulation for configuration
def parse_memory_string(memory_str)
  case memory_str.downcase
  when /(\d+)gb?$/
    $1.to_i * 1024 * 1024 * 1024
  when /(\d+)mb?$/
    $1.to_i * 1024 * 1024
  when /(\d+)kb?$/
    $1.to_i * 1024
  else
    memory_str.to_i
  end
end

# Array methods for server management
servers = node['myapp']['servers']

# Group servers by role
servers_by_role = servers.group_by { |server| server['role'] }

# Select healthy servers
healthy_servers = servers.select { |server| server['status'] == 'healthy' }

# Map to connection strings
```

```
connection_strings = servers.map do |server|
  "#{server['host']}:#{server['port']}"
end

# Reduce for configuration aggregation
total_memory = servers.reduce(0) do |sum, server|
  sum + parse_memory_string(server['memory'])
end

# Chain operations
web_servers = node['cluster']['servers']
              .select { |s| s['role'] == 'web' }
              .reject { |s| s['maintenance_mode'] }
              .sort_by { |s| s['priority'] }
              .map { |s| "#{s['hostname']}:#{s['port']}" }
```

Regular Expressions in Chef

```
# Validate configuration values
def valid_email?(email)
  email =~ /\A[\w+\-.]+@[a-z\d\-]+(\.[a-z\d\-]+)*\.[a-z]+\z/i
end

def valid_ip?(ip)
  ip =~ /\A(?:[0-9]{1,3}\.){3}[0-9]{1,3}\z/
end

# Parse log files
log_entries = File.readlines('/var/log/myapp.log').map do |line|
  if line =~ /(\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}) \[(\w+)\] (.+)/
    {
      timestamp: $1,
      level: $2,
      message: $3
    }
  end
end.compact

# Extract version numbers
version_string = "myapp-2.1.3-beta.1"
if version_string =~ /(\d+)\.(\d+)\.(\d+)(?:-(.+))?/
  major, minor, patch, prerelease = $1.to_i, $2.to_i, $3.to_i, $4
end
```

Page 7: Chef Server, Knife Plugins, and Advanced Workflows

Chef Server Management

```
# Organization management
chef-server-ctl org-create myorg "My Organization" --association_user admin
chef-server-ctl org-delete myorg
chef-server-ctl org-list

# User management
chef-server-ctl user-create username firstname lastname email password
chef-server-ctl user-delete username
chef-server-ctl org-user-add myorg username

# Backup and restore
chef-server-ctl backup
chef-server-ctl restore /path/to/backup

# SSL certificate management
chef-server-ctl install --certificate-path /path/to/cert.pem --private-key-path /path/to/key.pem
```

Advanced Knife Configuration

```
# ~/.chef/config.rb or knife.rb
current_dir = File.dirname(__FILE__)
log_level                :info
log_location             STDOUT
node_name                'your-username'
client_key               "#{current_dir}/your-username.pem"
chef_server_url          'https://chef-server.example.com/organizations/myorg'
cookbook_path            ["#{current_dir}/../cookbooks"]
environment_path         "#{current_dir}/../environments"
role_path                "#{current_dir}/../roles"
data_bag_path            "#{current_dir}/../data_bags"

# Proxy settings
http_proxy               'http://proxy.example.com:8080'
https_proxy              'http://proxy.example.com:8080'
no_proxy                 'localhost,127.0.0.1,.example.com'
```

```
# SSL settings
ssl_verify_mode          :verify_peer
ssl_ca_file              "#{current_dir}/ca_bundle.crt"

# Knife plugins configuration
knife[:bootstrap_template] = 'custom-bootstrap.erb'
knife[:ssh_user] = 'ubuntu'
knife[:ssh_identity_file] = '~/.ssh/id_rsa'
knife[:use_sudo] = true
```

Custom Knife Plugins

```
# lib/chef/knife/server_provision.rb
require 'chef/knife'

class Chef
  class Knife
    class ServerProvision < Knife
      banner 'knife server provision SERVER_NAME (options)'

      option :instance_type,
        short: '-t TYPE',
        long: '--instance-type TYPE',
        description: 'EC2 instance type',
        default: 't3.medium'

      option :environment,
        short: '-E ENV',
        long: '--environment ENV',
        description: 'Chef environment',
        default: 'production'

      option :run_list,
        short: '-r RUN_LIST',
        long: '--run-list RUN_LIST',
        description: 'Comma separated list of roles/recipes',
        proc: lambda { |o| o.split(/[\s,]+/) }

      def run
        server_name = name_args.first

        if server_name.nil?
          ui.error('You must specify a server name')
          exit 1
```

```
  end

  ui.info("Provisioning server: #{server_name}")

  # Create EC2 instance
  instance = create_ec2_instance(server_name)

  # Bootstrap with Chef
  bootstrap_server(instance.public_ip_address, server_name)

  ui.info("Server #{server_name} provisioned successfully!")
end

private

def create_ec2_instance(name)
  # AWS SDK integration
  require 'aws-sdk-ec2'

  ec2 = Aws::EC2::Resource.new(region: 'us-east-1')

  instance = ec2.create_instances({
    image_id: 'ami-0abcdef1234567890',
    min_count: 1,
    max_count: 1,
    instance_type: config[:instance_type],
    key_name: 'my-key-pair',
    tag_specifications: [{
      resource_type: 'instance',
      tags: [
        { key: 'Name', value: name },
        { key: 'Environment', value: config[:environment] }
      ]
    }]
  }).first

  # Wait for instance to be running
  instance.wait_until_running
  instance
end

def bootstrap_server(ip_address, node_name)
  bootstrap = Chef::Knife::Bootstrap.new
  bootstrap.name_args = [ip_address]
```

```
      bootstrap.config[:ssh_user] = 'ubuntu'
      bootstrap.config[:ssh_identity_file] = '~/.ssh/id_rsa'
      bootstrap.config[:use_sudo] = true
      bootstrap.config[:node_name] = node_name
      bootstrap.config[:environment] = config[:environment]
      bootstrap.config[:run_list] = config[:run_list] if config[:run_list]
      bootstrap.run
    end
  end
 end
end
```

Policyfiles (Modern Chef Workflow)

```
# Policyfile.rb
name 'myapp'
default_source :supermarket

# Version constraints
cookbook 'myapp', path: './cookbooks/myapp'
cookbook 'nginx', '~> 10.0'
cookbook 'postgresql', '= 7.1.0'

# Named run lists
named_run_list 'web', 'myapp::web', 'nginx'
named_run_list 'database', 'postgresql::server'
named_run_list 'worker', 'myapp::worker'

# Default attributes
default['myapp']['version'] = '2.1.0'
default['nginx']['worker_processes'] = 4

# Environment-specific attributes
default['myapp']['database_host'] = 'db.production.example.com'
default['myapp']['redis_url'] = 'redis://cache.production.example.com:6379'

# Policyfile commands
chef install                # Generate Policyfile.lock.json
chef update                 # Update cookbook versions
chef push production        # Upload policy to Chef Server
chef push-archive production    # Create and upload archive

# Node management with policies
knife node policy set NODE_NAME POLICY_GROUP POLICY_NAME
```

Cookbook Dependency Management

```
# metadata.rb advanced patterns
name 'myapp'
maintainer 'DevOps Team'
maintainer_email 'devops@example.com'
license 'Apache-2.0'
description 'Installs and configures MyApp'
version '2.1.0'
chef_version '>= 16.0'

# Platform support
supports 'ubuntu', '>= 18.04'
supports 'centos', '>= 7.0'
supports 'rhel', '>= 7.0'

# Cookbook dependencies
depends 'nginx', '~> 10.0'
depends 'postgresql', '= 7.1.0'
depends 'redisio', '>= 2.7.0'

# Optional dependencies
suggests 'monitoring', '~> 1.0'
suggests 'backup', '~> 2.0'

# Cookbook attributes documentation
attribute 'myapp/version',
  display_name: 'MyApp Version',
  description: 'Version of MyApp to install',
  type: 'string',
  default: '2.1.0'

attribute 'myapp/database/host',
  display_name: 'Database Host',
  description: 'Hostname of the database server',
  type: 'string',
  required: 'recommended'
```

Advanced Search Patterns

```
# Complex search queries
web_servers = search(:node,
```

```
  "role:web_server AND chef_environment:#{node.chef_environment} AND platform:ubuntu"
)

# Search with partial results for large environments
search(:node, "role:app_server",
  start: 0,
  rows: 100
) do |server|
  # Process each server as it's found
  log "Processing server: #{server.name}"
end

# Search data bags with filters
secrets = search(:secrets, "environment:#{node.chef_environment}")
  .select { |secret| secret['application'] == 'myapp' }
  .first

# Search with sort and specific attributes
sorted_servers = search(:node, "role:database",
  sort: "name ASC",
  filter_result: {
    name: ['name'],
    ip: ['ipaddress'],
    memory: ['memory', 'total']
  }
)
```

Page 8: Error Handling, Logging, and Monitoring

Comprehensive Error Handling

```
# Custom exception classes
module MyApp
  class ConfigurationError < StandardError; end
  class ServiceUnavailableError < StandardError; end
  class DeploymentError < StandardError; end
end

# Recipe with error handling
begin
  # Validate required attributes
```

```
  raise MyApp::ConfigurationError, "Database host not specified" unless
node['myapp']['database']['host']

  # Test service availability
  ruby_block 'test_database_connection' do
    block do
      require 'net/http'
      require 'timeout'

      begin
        Timeout::timeout(10) do
          http = Net::HTTP.new(node['myapp']['database']['host'], node['myapp']['database']['port'])
          response = http.request_head('/')
        end
      rescue Timeout::Error, Errno::ECONNREFUSED => e
        raise MyApp::ServiceUnavailableError, "Database unreachable: #{e.message}"
      end
    end
  end

  # Main deployment logic
  include_recipe 'myapp::deploy'

rescue MyApp::ConfigurationError => e
  Chef::Log.fatal("Configuration error: #{e.message}")
  ruby_block 'configuration_failure_notification' do
    block do
      # Send notification to monitoring system
      send_alert("Chef run failed on #{node['fqdn']}: #{e.message}")
    end
  end
  raise # Re-raise to fail the Chef run

rescue MyApp::ServiceUnavailableError => e
  Chef::Log.error("Service unavailable: #{e.message}")
  # Set node in maintenance mode
  node.normal['myapp']['maintenance_mode'] = true
  node.save unless Chef::Config[:solo]

rescue MyApp::DeploymentError => e
  Chef::Log.error("Deployment failed: #{e.message}")
  # Rollback logic
  include_recipe 'myapp::rollback'
```

```
rescue StandardError => e
  Chef::Log.fatal("Unexpected error: #{e.message}")
  Chef::Log.fatal("Backtrace: #{e.backtrace.join("\n")}")

  # Emergency cleanup
  service 'myapp' do
    action :stop
    ignore_failure true
  end

  raise
ensure
  # Always execute cleanup
  ruby_block 'cleanup_temp_files' do
    block do
      Dir.glob('/tmp/myapp-*').each { |f| File.delete(f) }
    end
    only_if { Dir.exist?('/tmp') }
  end
end
```

Advanced Logging Strategies

```
# Custom logger class
class ChefLogger
  def initialize(component)
    @component = component
    @start_time = Time.now
  end

  def info(message)
    Chef::Log.info("[#{@component}] #{message}")
  end

  def debug(message)
    Chef::Log.debug("[#{@component}] #{message}")
  end

  def error(message)
    Chef::Log.error("[#{@component}] #{message}")
  end

  def timing(operation)
    start = Time.now
```

```
    yield
    duration = Time.now - start
    info("#{operation} completed in #{duration.round(2)} seconds")
  end

  def with_context(context)
    old_component = @component
    @component = "#{old_component}::#{context}"
    yield
  ensure
    @component = old_component
  end
end

# Usage in recipes
logger = ChefLogger.new('myapp::deploy')

logger.timing('database_migration') do
  execute 'run_migrations' do
    command 'bundle exec rake db:migrate'
    cwd '/opt/myapp'
    user 'myapp'
  end
end

logger.with_context('config_generation') do
  logger.info("Generating configuration for environment: #{node.chef_environment}")

  template '/opt/myapp/config/application.yml' do
    source 'application.yml.erb'
    variables(
      environment: node.chef_environment,
      database_url: database_connection_string
    )
  end
end
```

Structured Logging with JSON

```
# libraries/json_logger.rb
require 'json'
require 'syslog'

class JSONLogger
```

```
  def initialize(program_name = 'chef-client')
    @program_name = program_name
    @syslog = Syslog.open(program_name, Syslog::LOG_PID, Syslog::LOG_LOCAL0)
  end

  def log(level, message, metadata = {})
    log_entry = {
      timestamp: Time.now.utc.iso8601,
      level: level.to_s.upcase,
      message: message,
      program: @program_name,
      node: node_info,
      chef_run_id: Chef::Config[:chef_guid]
    }.merge(metadata)

    json_message = JSON.generate(log_entry)

    case level
    when :debug
      @syslog.debug(json_message)
    when :info
      @syslog.info(json_message)
    when :warn
      @syslog.warning(json_message)
    when :error
      @syslog.err(json_message)
    when :fatal
      @syslog.crit(json_message)
    end
  end

  private

  def node_info
    {
      fqdn: node['fqdn'],
      platform: node['platform'],
      platform_version: node['platform_version'],
      chef_environment: node.chef_environment,
      run_list: node.run_list.to_s
    }
  end
end
```

```
# Usage
json_logger = JSONLogger.new('myapp')

json_logger.log(:info, 'Starting application deployment', {
  application: 'myapp',
  version: node['myapp']['version'],
  deployment_id: SecureRandom.uuid
})
```

Monitoring Integration

```
# libraries/monitoring.rb
class MonitoringClient
  def initialize
    @statsd_host = node['monitoring']['statsd_host'] || 'localhost'
    @statsd_port = node['monitoring']['statsd_port'] || 8125
  end

  def increment(metric, tags = {})
    send_metric("#{metric}:1|c", tags)
  end

  def gauge(metric, value, tags = {})
    send_metric("#{metric}:#{value}|g", tags)
  end

  def timing(metric, duration, tags = {})
    send_metric("#{metric}:#{duration}|ms", tags)
  end

  def time(metric, tags = {})
    start_time = Time.now
    yield
    duration = ((Time.now - start_time) * 1000).to_i
    timing(metric, duration, tags)
  end

  private

  def send_metric(metric_string, tags)
    require 'socket'

    tagged_metric = if tags.any?
                      "#{metric_string}|##{tags.map { |k, v| "#{k}:#{v}" }.join(',')}"
```

```
                else
                  metric_string
                end

    socket = UDPSocket.new
    socket.send(tagged_metric, 0, @statsd_host, @statsd_port)
    socket.close
  rescue StandardError => e
    Chef::Log.warn("Failed to send metric: #{e.message}")
  end
end

# Usage in recipes
monitoring = MonitoringClient.new

monitoring.time('chef.recipe.execution',
  recipe: 'myapp::deploy',
  environment: node.chef_environment
) do
  # Recipe logic here
  include_recipe 'myapp::install'
  include_recipe 'myapp::configure'
end

monitoring.gauge('chef.node.cpu_count', node['cpu']['total'])
monitoring.increment('chef.run.success',
  node: node['fqdn'],
  environment: node.chef_environment
)
```

Health Checks and Self-Healing

```
# recipes/health_check.rb
ruby_block 'application_health_check' do
  block do
    require 'net/http'
    require 'json'

    health_endpoint = "http://localhost:#{node['myapp']['port']}/health"

    begin
      uri = URI(health_endpoint)
      response = Net::HTTP.get_response(uri)
```

```
      if response.code == '200'
        health_data = JSON.parse(response.body)

        # Update node attributes with health status
        node.normal['myapp']['health'] = {
          'status' => 'healthy',
          'last_check' => Time.now.iso8601,
          'response_time' => health_data['response_time'],
          'database_status' => health_data['database']
        }

        Chef::Log.info("Health check passed")
      else
        handle_unhealthy_service(response.code, response.body)
      end

    rescue StandardError => e
      handle_unhealthy_service('connection_error', e.message)
    end
  end

  # Only run if service should be running
  only_if { node['myapp']['enabled'] }
end

def handle_unhealthy_service(error_code, error_message)
  Chef::Log.error("Health check failed: #{error_code} - #{error_message}")

  node.normal['myapp']['health'] = {
    'status' => 'unhealthy',
    'last_check' => Time.now.iso8601,
    'error' => "#{error_code}: #{error_message}"
  }

  # Attempt self-healing
  attempts = node['myapp']['restart_attempts'] || 0
  max_attempts = node['myapp']['max_restart_attempts'] || 3

  if attempts < max_attempts
    Chef::Log.info("Attempting to restart service (attempt #{attempts + 1}/#{max_attempts})")

    service 'myapp' do
      action :restart
    end
```

```
    node.normal['myapp']['restart_attempts'] = attempts + 1
  else
    Chef::Log.error("Max restart attempts reached, manual intervention required")
    node.normal['myapp']['maintenance_mode'] = true
  end
end
```

Page 9: Performance Optimization and Scaling Patterns

Lazy Evaluation and Performance

```
# Expensive operations with lazy evaluation
template '/etc/myapp/database.yml' do
  source 'database.yml.erb'
  variables lazy {
    # This block only executes when the template is actually rendered
    {
      database_servers: search(:node, "role:database AND
chef_environment:#{node.chef_environment}"),
      password: encrypted_data_bag_item('secrets', 'database', secret_key)['password'],
      connection_pool_size: calculate_optimal_pool_size
    }
  }
  action :create
end

# Lazy attribute computation
default['myapp']['memory_settings'] = lazy {
  total_memory_kb = node['memory']['total'].to_i
  total_memory_mb = total_memory_kb / 1024

  {
    'heap_size' => "#{(total_memory_mb * 0.6).to_i}m",
    'perm_size' => "#{(total_memory_mb * 0.1).to_i}m",
    'stack_size' => '512k'
  }
}
```

Caching Strategies

```
# libraries/cache_manager.rb
```

```
class CacheManager
  def initialize
    @cache = {}
    @cache_ttl = {}
    @default_ttl = 300 # 5 minutes
  end

  def get(key, ttl = @default_ttl)
    if cached?(key)
      @cache[key]
    else
      value = yield
      set(key, value, ttl)
      value
    end
  end

  def set(key, value, ttl = @default_ttl)
    @cache[key] = value
    @cache_ttl[key] = Time.now + ttl
  end

  def cached?(key)
    @cache.key?(key) && @cache_ttl[key] > Time.now
  end

  def clear
    @cache.clear
    @cache_ttl.clear
  end
end

# Global cache instance
CHEF_CACHE = CacheManager.new

# Usage in recipes and libraries
def expensive_database_query
  CHEF_CACHE.get('database_servers', 600) do
    search(:node, "role:database AND chef_environment:#{node.chef_environment}")
  end
end

def load_encrypted_secrets
  CHEF_CACHE.get("secrets_#{node.chef_environment}", 1800) do
```

```
    secret_key =
Chef::EncryptedDataBagItem.load_secret('/etc/chef/encrypted_data_bag_secret')
    Chef::EncryptedDataBagItem.load('secrets', node.chef_environment, secret_key)
  end
end
```

Batch Operations and Resource Optimization

```
# Batch file operations
files_to_create = [
  { path: '/opt/myapp/config/app.conf', template: 'app.conf.erb' },
  { path: '/opt/myapp/config/db.conf', template: 'db.conf.erb' },
  { path: '/opt/myapp/config/cache.conf', template: 'cache.conf.erb' }
]

# Create all files with shared variables
shared_variables = {
  environment: node.chef_environment,
  app_version: node['myapp']['version'],
  timestamp: Time.now.iso8601
}

files_to_create.each do |file_config|
  template file_config[:path] do
    source file_config[:template]
    variables shared_variables
    owner 'myapp'
    group 'myapp'
    mode '0644'
    # Only notify once after all files are created
    notifies :restart, 'service[myapp]', :delayed if file_config == files_to_create.last
  end
end

# Batch package installation
packages = %w[curl wget git vim htop nginx postgresql-client redis-tools]

# Install all packages in one resource
package packages do
  action :install
end

# Or with version specifications
package_list = {
```

```
  'nginx' => '1.18.0',
  'postgresql-client' => '12+214',
  'redis-tools' => '6.2.6'
}

package_list.each do |pkg, version|
  package pkg do
    version version
    action :install
  end
end
```

Parallel Execution Patterns

```
# Parallel service management using threads
services = %w[nginx postgresql redis-server memcached]

# Create thread pool for parallel operations
require 'thread'

thread_pool = []
mutex = Mutex.new

services.each do |service_name|
  thread_pool << Thread.new do
    begin
      # Each service check runs in parallel
      status = `systemctl is-active #{service_name}`.strip

      mutex.synchronize do
        node.normal['services'][service_name] = {
          'status' => status,
          'checked_at' => Time.now.iso8601
        }
      end

      Chef::Log.info("Service #{service_name} status: #{status}")
    rescue StandardError => e
      Chef::Log.error("Failed to check #{service_name}: #{e.message}")
    end
  end
end

# Wait for all threads to complete
```

```
thread_pool.each(&:join)
```

Conditional Resource Execution

```
# Smart resource execution based on state
ruby_block 'conditional_deployment' do
  block do
    current_version = node['myapp']['current_version']
    target_version = node['myapp']['target_version']

    if current_version != target_version
      Chef::Log.info("Deploying from #{current_version} to #{target_version}")

      # Only run deployment resources if version changed
      run_context.include_recipe 'myapp::deploy'

      # Update version tracking
      node.normal['myapp']['current_version'] = target_version
      node.normal['myapp']['deployed_at'] = Time.now.iso8601
    else
      Chef::Log.info("Version #{current_version} already deployed, skipping")
    end
  end
end

# Resource guards with complex logic
execute 'compile_application' do
  command 'make clean && make install'
  cwd '/opt/myapp/src'
  user 'myapp'

  # Multiple conditions using not_if
  not_if { File.exist?('/opt/myapp/bin/myapp') }
  not_if { node['myapp']['skip_compilation'] }
  not_if do
    # Check if binary is newer than source
    binary_mtime = File.mtime('/opt/myapp/bin/myapp') rescue Time.at(0)
    source_mtime = Dir.glob('/opt/myapp/src/**/*.c').map { |f| File.mtime(f) }.max rescue
Time.at(1)
    binary_mtime > source_mtime
  end
end
```

Memory and Resource Management

```
# Memory-efficient data processing
def process_large_dataset(file_path)
  processed_count = 0

  File.foreach(file_path) do |line|
    # Process one line at a time instead of loading entire file
    process_line(line.strip)
    processed_count += 1

    # Periodic garbage collection for large datasets
    GC.start if processed_count % 10000 == 0
  end

  Chef::Log.info("Processed #{processed_count} lines")
end

# Resource cleanup patterns
ruby_block 'cleanup_old_releases' do
  block do
    releases_dir = '/opt/myapp/releases'
    keep_releases = node['myapp']['keep_releases'] || 5

    if Dir.exist?(releases_dir)
      releases = Dir.entries(releases_dir)
                    .select { |d| d.match(/^\d{14}$/) }  # timestamp format
                    .sort
                    .reverse

      releases_to_remove = releases[keep_releases..-1] || []

      releases_to_remove.each do |release|
        release_path = File.join(releases_dir, release)
        FileUtils.rm_rf(release_path)
        Chef::Log.info("Removed old release: #{release}")
      end
    end
  end
end
```

Page 10: Security, Secrets Management, and Production Patterns

Secrets Management Best Practices

```
# Secure secret loading with fallbacks
def load_secret(secret_name, fallback = nil)
  secret_sources = [
    -> { load_from_vault(secret_name) },
    -> { load_from_encrypted_data_bag(secret_name) },
    -> { load_from_environment(secret_name) },
    -> { fallback }
  ]

  secret_sources.each do |source|
    begin
      secret = source.call
      return secret if secret && !secret.empty?
    rescue StandardError => e
      Chef::Log.debug("Secret source failed: #{e.message}")
    end
  end

  raise "Unable to load secret: #{secret_name}"
end

def load_from_vault(secret_name)
  # HashiCorp Vault integration
  require 'vault'

  Vault.address = node['vault']['address']
  Vault.token = node['vault']['token']

  secret = Vault.logical.read("secret/#{node.chef_environment}/#{secret_name}")
  secret.data[:value] if secret
end

def load_from_encrypted_data_bag(secret_name)
  secret_key = Chef::EncryptedDataBagItem.load_secret('/etc/chef/encrypted_data_bag_secret')
  bag_item = Chef::EncryptedDataBagItem.load('secrets', node.chef_environment, secret_key)
  bag_item[secret_name]
end

def load_from_environment(secret_name)
```

```
  ENV[secret_name.upcase]
end

# Usage in recipes
database_password = load_secret('database_password')
api_key = load_secret('external_api_key', 'default-dev-key')
```

File Permission and Ownership Management

```
# Secure file creation with proper permissions
def create_secure_file(path, content, owner = 'root', group = 'root', mode = '0600')
  # Create parent directories with secure permissions
  parent_dir = File.dirname(path)
  directory parent_dir do
    owner owner
    group group
    mode '0750'
    recursive true
  end

  # Create temporary file first
  temp_file = "#{path}.tmp.#{Process.pid}"

  file temp_file do
    content content
    owner owner
    group group
    mode mode
    sensitive true  # Prevents content from appearing in logs
  end

  # Atomic move to final location
  ruby_block "move_#{File.basename(path)}" do
    block do
      File.rename(temp_file, path)
    end
  end
end

# Secure configuration template
template '/etc/myapp/secrets.yml' do
  source 'secrets.yml.erb'
  owner 'myapp'
  group 'myapp'
```

```
  mode '0600'
  sensitive true
  variables lazy {
    {
      database_password: load_secret('database_password'),
      api_keys: load_secret('api_keys'),
      encryption_key: load_secret('encryption_key')
    }
  }
  notifies :restart, 'service[myapp]', :delayed
end
```

SSL/TLS Certificate Management

```
# SSL certificate deployment and management
def deploy_ssl_certificate(domain, cert_source = 'letsencrypt')
  cert_dir = "/etc/ssl/certs/#{domain}"
  key_dir = "/etc/ssl/private/#{domain}"

  # Create certificate directories
  [cert_dir, key_dir].each do |dir|
    directory dir do
      owner 'root'
      group 'ssl-cert'
      mode '0750'
      recursive true
    end
  end

  case cert_source
  when 'letsencrypt'
    deploy_letsencrypt_certificate(domain, cert_dir, key_dir)
  when 'vault'
    deploy_vault_certificate(domain, cert_dir, key_dir)
  when 'files'
    deploy_file_certificate(domain, cert_dir, key_dir)
  end

  # Set up certificate renewal
  cron 'renew_ssl_certificate' do
    minute '0'
    hour '3'
    day '*'
    month '*'
```

```
    weekday '1'  # Monday
    command "/usr/local/bin/renew-cert.sh #{domain}"
    user 'root'
  end
end

def deploy_letsencrypt_certificate(domain, cert_dir, key_dir)
  # Install certbot
  package 'certbot' do
    action :install
  end

  # Generate certificate
  execute "generate_letsencrypt_cert_#{domain}" do
    command "certbot certonly --standalone -d #{domain} --non-interactive --agree-tos --email
#{node['ssl']['admin_email']}"
    creates "/etc/letsencrypt/live/#{domain}/fullchain.pem"
  end

  # Link certificates to application directories
  link "#{cert_dir}/certificate.pem" do
    to "/etc/letsencrypt/live/#{domain}/fullchain.pem"
  end

  link "#{key_dir}/private.key" do
    to "/etc/letsencrypt/live/#{domain}/privkey.pem"
  end
end
```

Security Hardening Patterns

```
# System security hardening
def apply_security_hardening
  # Disable unnecessary services
  unnecessary_services = %w[telnet rsh rlogin finger]
  unnecessary_services.each do |svc|
    service svc do
      action [:stop, :disable]
      ignore_failure true
    end
  end

  # Configure secure SSH
  template '/etc/ssh/sshd_config' do
```

```
    source 'sshd_config.erb'
    owner 'root'
    group 'root'
    mode '0600'
    variables(
      port: node['ssh']['port'] || 22,
      permit_root_login: node['ssh']['permit_root_login'] || 'no',
      password_authentication: node['ssh']['password_authentication'] || 'no',
      allowed_users: node['ssh']['allowed_users'] || []
    )
    notifies :restart, 'service[ssh]', :delayed
  end

  # Configure firewall rules
  firewall_rules = node['security']['firewall_rules'] || []
  firewall_rules.each do |rule|
    execute "configure_firewall_#{rule['name']}" do
      command "ufw #{rule['action']} #{rule['port']}/#{rule['protocol']} from #{rule['source']}"
      not_if "ufw status | grep '#{rule['port']}/#{rule['protocol']}'"
    end
  end

  # Set up fail2ban
  package 'fail2ban' do
    action :install
  end

  template '/etc/fail2ban/jail.local' do
    source 'fail2ban.jail.erb'
    owner 'root'
    group 'root'
    mode '0644'
    notifies :restart, 'service[fail2ban]', :delayed
  end
end
```

Audit Logging and Compliance

```
# Audit logging setup
def configure_audit_logging
  # Install auditd
  package 'auditd' do
    action :install
  end
```

```
  # Configure audit rules
  template '/etc/audit/rules.d/myapp.rules' do
    source 'audit-rules.erb'
    owner 'root'
    group 'root'
    mode '0640'
    variables(
      monitored_files: [
        '/etc/myapp/',
        '/opt/myapp/config/',
        '/etc/ssl/private/'
      ],
      monitored_commands: [
        '/usr/bin/sudo',
        '/bin/su',
        '/usr/bin/ssh'
      ]
    )
    notifies :restart, 'service[auditd]', :delayed
  end

  # Set up log rotation for audit logs
  template '/etc/logrotate.d/audit' do
    source 'logrotate-audit.erb'
    owner 'root'
    group 'root'
    mode '0644'
    variables(
      retention_days: node['audit']['retention_days'] || 90,
      compress: node['audit']['compress'] || true
    )
  end
end

# Compliance reporting
ruby_block 'generate_compliance_report' do
  block do
    report = {
      timestamp: Time.now.iso8601,
      node: node['fqdn'],
      environment: node.chef_environment,
      checks: {}
    }
```

```
    # Check SSL certificate expiry
    ssl_certs = Dir.glob('/etc/ssl/certs/*/*.pem')
    ssl_certs.each do |cert_file|
      cert_info = `openssl x509 -in #{cert_file} -text -noout`
      if cert_info =~ /Not After : (.+)/
        expiry_date = Date.parse($1)
        days_until_expiry = (expiry_date - Date.today).to_i
        report[:checks]["ssl_cert_#{File.basename(cert_file)}"] = {
          status: days_until_expiry > 30 ? 'pass' : 'warn',
          days_until_expiry: days_until_expiry
        }
      end
    end

    # Check for security updates
    if node['platform'] == 'ubuntu'
      security_updates = `apt list --upgradable 2>/dev/null | grep -i security | wc -l`.strip.to_i
      report[:checks]['security_updates'] = {
        status: security_updates == 0 ? 'pass' : 'fail',
        pending_updates: security_updates
      }
    end

    # Write report
    File.write('/var/log/chef-compliance.json', JSON.pretty_generate(report))
    Chef::Log.info("Compliance report generated: #{report[:checks].count} checks")
  end
end
```

Production Deployment Patterns

```
# Blue-Green deployment pattern
def blue_green_deployment(app_name, new_version)
  current_color = node['deployment']['current_color'] || 'blue'
  next_color = current_color == 'blue' ? 'green' : 'blue'

  Chef::Log.info("Deploying #{app_name} v#{new_version} to #{next_color} environment")

  # Deploy to inactive environment
  deploy_to_environment(app_name, new_version, next_color)

  # Health check new deployment
  if health_check_passed?(next_color)
```

```
    # Switch traffic to new environment
    switch_load_balancer(next_color)

    # Update current color
    node.normal['deployment']['current_color'] = next_color
    node.normal['deployment']['previous_version'] = node['deployment']['current_version']
    node.normal['deployment']['current_version'] = new_version

    Chef::Log.info("Successfully deployed #{app_name} v#{new_version}")
  else
    Chef::Log.error("Health check failed for #{next_color} environment")
    raise "Deployment failed: health check failed"
  end
end

def deploy_to_environment(app_name, version, color)
  app_dir = "/opt/#{app_name}-#{color}"

  # Stop service if running
  service "#{app_name}-#{color}" do
    action :stop
    ignore_failure true
  end

  # Deploy new version
  git app_dir do
    repository node[app_name]['repository']
    revision "v#{version}"
    action :sync
  end

  # Install dependencies
  execute "bundle_install_#{color}" do
    command 'bundle install --deployment --without development test'
    cwd app_dir
  end

  # Start service
  service "#{app_name}-#{color}" do
    action :start
  end
end

def health_check_passed?(color)
```

```
  port = node['myapp']["#{color}_port"]

  30.times do |attempt|
    begin
      response = Net::HTTP.get_response(URI("http://localhost:#{port}/health"))
      return true if response.code == '200'
    rescue
      # Connection failed, retry
    end

    sleep 2
  end

  false
end
```

Monitoring and Alerting Integration

```
# Comprehensive monitoring setup
def setup_monitoring(application)
  # Install monitoring agents
  package 'datadog-agent' do
    action :install
  end

  # Configure application monitoring
  template '/etc/datadog-agent/conf.d/myapp.yaml' do
    source 'datadog-myapp.yaml.erb'
    variables(
      api_key: load_secret('datadog_api_key'),
      application: application,
      environment: node.chef_environment,
      metrics: {
        'response_time' => '/metrics/response_time',
        'error_rate' => '/metrics/errors',
        'throughput' => '/metrics/requests'
      }
    )
    notifies :restart, 'service[datadog-agent]', :delayed
  end

  # Set up custom metrics collection
  cron 'collect_custom_metrics' do
    minute '*/5'
```

```
    command '/usr/local/bin/collect-metrics.sh'
  end

  # Configure alerting
  setup_alerting_rules(application)
end

def setup_alerting_rules(application)
  alerting_rules = [
    {
      name: 'high_error_rate',
      condition: 'error_rate > 0.05',
      severity: 'critical'
    },
    {
      name: 'high_response_time',
      condition: 'response_time > 2000',
      severity: 'warning'
    },
    {
      name: 'low_disk_space',
      condition: 'disk_usage > 0.85',
      severity: 'warning'
    }
  ]

  template '/etc/monitoring/alerts.yml' do
    source 'monitoring-alerts.yml.erb'
    variables(
      application: application,
      environment: node.chef_environment,
      rules: alerting_rules,
      notification_channels: node['monitoring']['notification_channels']
    )
  end
end
```

Extended Quick Reference:

- **Chef Docs:** https://docs.chef.io
- **Ruby Docs:** https://ruby-doc.org
- **ChefSpec:** https://github.com/chefspec/chefspec
- **Test Kitchen:** https://kitchen.ci

- **InSpec:** https://www.inspec.io
- **Policyfiles:** https://docs.chef.io/policyfile.html
- **Vault Integration:** https://www.vaultproject.io/docs/auth/chef
- **Security Hardening:** https://dev-sec.io/
- **Chef Workstation:** https://docs.chef.io/workstation/
- **Habitat:** https://www.habitat.sh/docs/

Performance Tips:

- Use `lazy` evaluation for expensive operations
- Implement caching for search results and external API calls
- Batch similar operations together
- Use `sensitive true` for secrets in resources
- Leverage `only_if` and `not_if` guards effectively
- Consider using Policyfiles for better dependency management

Security Best Practices:

- Never store secrets in plain text
- Use encrypted data bags or external secret management
- Implement proper file permissions (principle of least privilege)
- Regularly rotate certificates and secrets
- Enable audit logging for compliance
- Use secure communication (SSL/TLS) everywhere