Intercity Chef Recipes

This Chef repository aims at being the easiest way set up and configure your own Rails server to host one or more Ruby on Rails applications using best practices from our community.

The configuration is heavily inspired by blog posts and chef recipes from 37signals and the Opscode Community Cookbooks.

Features

Takes care of automatic installation and configuration of the following software on a single server or multiple servers:

- nginx webserver
- Passenger or Unicorn for running Ruby on Rails
- Multiple apps on one server
- Database creation and password generation
- Easy SSL configuration
- Deployment with Capistrano
- Configure ENV variables
- Easy backup scheduling

Supported OSes

- Ubuntu 12.04 LTS
- Ubuntu 14.04 LTS

Databases

- MySQL
- PostgreSQL

Getting started

The following paragraphs will guide you to set up your own server to host Ruby on Rails applications.

1. Set up this repository

Clone the repository onto your own workstation. For example in your `~/Code` directory:

```
$ cd ~/Code
$ git clone git://github.com/intercity/chef-repo.git chef_repo
```

RootFile

Run bundle:

```
$ bundle install
```

ContinueFile

Run Librarian install:

```
$ librarian-chef install
```

ContinueFile

2. Install your server

Use the following command to install Chef on your server and prepare it to be installed by these cookbooks:

```
bundle exec knife solo prepare <your user>@<your host/ip>
```

DeclareVariables "your user", "your host/ip"
RootFile

This will create a file

```
nodes/<your host/ip>.json
```

Now copy the the contents from the `nodes/sample_host.json` from this repository into this new file. Replace the sample values between `< >` with the values for your server and applications.

Not realizable with LDS
Can't copy content yet

When this is done. Run the following command to start the full installation of your server:

```
bundle exec knife solo cook <your user>@<your host/ip>
```

DeclareVariables
RootFile

3. Deploy your application

You can deploy your applications with Capistrano.

Add the Capistrano gem to your Gemfile:

```
# your other gems..

gem 'capistrano', '~> 3.2.1'
gem 'capistrano-rails', '~> 1.1'
```

ChangeFileLines

And run bundle to install it:

```
bundle
```

RootFile

Now generate configuration files for Capistrano:

```
bundle exec cap install
```

ContinueFile

This command will generate the following files in your application:

```
Capfile
config/deploy.rb
config/deploy/production.rb
config/deploy/staging.rb
```

Edit the file `Capfile` and change it's contents to:

```
# Load DSL and Setup Up Stages
require 'capistrano/setup'

# Includes default deployment tasks
```

ChangeFileLines

```
require 'capistrano/deploy'

require 'capistrano/rails'

# Loads custom tasks from `lib/capistrano/tasks' if you have any defined.
Dir.glob('lib/capistrano/tasks/*.cap').each { |r| import r }
```

Then edit `config/deploy.rb` and change it to the sample below. Replace `>> your git repo_url <<` with the SSH clone URL of your repository:

DeclareVariables

ChangeFileLines

```
# config valid only for Capistrano 3.2.1
lock '3.2.1'

set :application, '>> your_application_name <<'
set :repo_url, '>> your git repo_url <<'

# Default branch is :master
# Uncomment the following line to have Capistrano ask which branch to deploy.
# ask :branch, proc { `git rev-parse --abbrev-ref HEAD`.chomp }

# Replace the sample value with the name of your application here:
set :deploy_to, '/u/apps/>> your_application_name <<_production'

# Use agent forwarding for SSH so you can deploy with the SSH key on your workstation.
set :ssh_options, {
  forward_agent: true
}

# Default value for :pty is false
set :pty, true

set :linked_files, %w{config/database.yml .rbenv-vars .ruby-version}
set :linked_dirs, %w{log tmp/pids tmp/cache tmp/sockets vendor/bundle public/system}

set :default_env, { path: "/opt/rbenv/shims:$PATH" }

set :keep_releases, 5

namespace :deploy do

  desc 'Restart application'
  task :restart do
    on roles(:app), in: :sequence, wait: 5 do
      execute :touch, release_path.join('tmp/restart.txt')
    end
  end

  after :publishing, :restart

end
```

Replace the contents of `config/deploy/production.rb` with

DeclareVariables
ChangeFileLines

```
server '>> your server address <<', user: 'deploy', roles: %w{web app db}
```

Replace `>> your server address <<` with the domain name or ip address of your server.

To verify that everything is set up correctly run:

RootFile

```
bundle exec cap production deploy:check
```

Finally to deploy, run:

RootFile

```
bundle exec cap production deploy
```

This will deploy your app and run your database migrations.

Congratulations! You've now deployed your application. Browse to your application in your webbrowser and everything should work!

Try these cookbooks with Vagrant

You can use Vagrant to experience how easy it is to install your servers with this repository.

First, install Vagrant from http://vagrantup.com. Then install the following two Vagrant plugins:

Make sure you have Vagrant version 1.6.5 or higher installed.

```
vagrant plugin install vagrant-librarian-chef
vagrant plugin install vagrant-omnibus
```

manual work
no version checks supported by LDS
insert cmd for manual version check in console

RootFile

Finally, start a Vagrant machine with a sample server configuration:

```
vagrant up mysql
```

RootFile

This will boot a local Ubuntu virtual machine and install it so you can deploy Ruby on Rails applications that use MySQL as the database.

You can check out the sample configuration in file `Vagrantfile`

When you run into problems:

These steps should let you **set up or test your own Rails infrastructure in 5 - 10 minutes**. If something doesn't work or you need more instructions:

Please! Open an issue or email hello@intercityup.com.

Testing with test-kitchen

CI testing

Test-kitchen is a tool where you can automatically provision a server with these cookbooks and run the tests for them. The configuration in `.kitchen.yml` works with DigitalOcean.

First you need to obtain a DigitalOcean access token here: https://cloud.digitalocean.com/settings/applications. Then you need to find IDs of the SSH keys you added to your account: https://cloud.digitalocean.com/ssh_keys. You can obtain these IDs with the following command:

manual work more feasible

```
$ curl -X GET https://api.digitalocean.com/v2/account/keys -H "Authorization: Bearer <YOUR_DIGITALOCEAN_ACCESS TC
```

DeclareVariables
RootFile

When you've obtained both your access token and your key IDs you can run the tests like this:

```
$ export DIGITALOCEAN_ACCESS_TOKEN=<YOUR DIGITALOCEAN ACCESS TOKEN>
$ export DIGITALOCEAN_SSH_KEY_IDS=<YOUR DIGITALOCEAN SSH KEY ID>
$ bin/kitchen test
```

DeclareVariables
RootFile

This command boots up a Droplet in your DigitalOcean account, provisions it with Chef, runs the tests and destroys the Droplet.

Testing while developing

If you want to keep the Droplet running and do testing while making changes you can use the `kitchen verify` command instead of the `kitchen test` command to verify your changes:

```
$ bin/kitchen verify
```

Resources and original authors

- Most of the cookbooks that are used in this repository are installed from the Opscode Community Cookbooks.
- The `rails` and `bluepill` configuration is based off the cookbooks by jsierles at https://github.com/jsierles/chef_cookbooks