Ruby on Rails 101 – Chapter 8

In the chapter, we are going to translate our rails app into multi-lingual.

And then we will discuss several deployment approaches.

1. Multilingual
2. Adding chinese
3. Switching locale
4. Pow local server
5. Mobile devices testing via local network
6. Deployment database-config
7. Introducing git
8. Lab – “Try Git”
9. Apply git to our project
10. Push git to server repo
11. Deploying options

Multilingual

All the languages files are in the `config/locales/` folder.

There should be a sample file named `en.yml` there. All the language files use the YAML format.

Take the "Home" wording as example. We can put the following code in the `en.yml` file and then access it again with the `translate` method, or `t` as alias.

```
en:
  home: "Home"
```

Using the locale in view:

```
<li><%= link_to translate('home'), root_path %></li>
```

Or we can have better organization by grouping the locales. This ensures we do not create conflicts with the same keys. Now in the `en.yml` file.

```
en:
  nav:
    home: "Home"
```

And the view becomes:

```
<li><%= link_to translate('nav.home'), root_path %>
</li>
```

By grouping the locales, it ensures we do not create conflicts with the same keys.

And typing the `translate()` method every time can be annoying. That's why rails comes with an alias `t()`:

```
<li><%= link_to t('nav.home'), root_path %></li>
```

Common Locales Setting

By default, the locale files comes with empty setting (Ignoring that hello world example). If you need some common local settings such as date and time. You can find it in this repo.

For date/time locals, please use `localize` method, or `l`, instead of the `translate`.

Adding chinese

How about adding the Chinese supports?

1. Create a new file named `zh-TW.yml` under the folder `config/locales/`.
2. Put the following content into the file.

```
zh-TW:
  nav:
    home: "首頁"
```

3. Setting the default local to Chinese.
    1. Create a new file named `locale.rb` under the `config/initializers/` folder.
    2. Put the following ruby code in the file. I18n.default_locale = "zh-TW"
    3. Restart the server.

Note: “zh-TW” is for Traditional Chinese. And “zh-CN” is for Simplified Chinese.

Before moving advance, let’s complete the navigation locale:

```
zh-TW:
  nav:
    home: "首頁"
    login: "登入"
    logout: "登出"
    signup: "成為用戶"
```

And make sure we updated the view to use the locale setting. In the `application.html.erb` file:

```
<nav>
  <ul>
    <li><%= link_to t('nav.home'), root_path %></li>
    <% if user_signed_in? %>
      <li><%= link_to "#{t('nav.logout')} (#
{current_user.email})", destroy_user_session_path,
```

```
:method => :delete %></li>
    <% else %>
      <li><%= link_to t('nav.login'),
new_user_session_path  %></li>
      <li><%= link_to t('nav.signup'),
new_user_registration_path  %></li>
    <% end %>
  </ul>
</nav>
```

Switching locale

How about switching the locale?

Getting locale option from params.

In the `application_controller.rb` file:

```
  class ApplicationController <
ActionController::Base
    protect_from_forgery

    before_filter :set_locale

    def set_locale
      I18n.locale = params[:locale] ||
I18n.default_locale
    end
  end
```

And now, we can access different with the params query:
`http://0.0.0.0:3000/?locale=en`.

Automatically appending the locale params to URL.

Rails comes with a `default_url_options` that is the default query string and values that will be attached in the URL.

We can override the `default_url_options`. Add the following method in the `application_controller.rb` file.

```
def default_url_options
  { locale: I18n.locale }
end
```

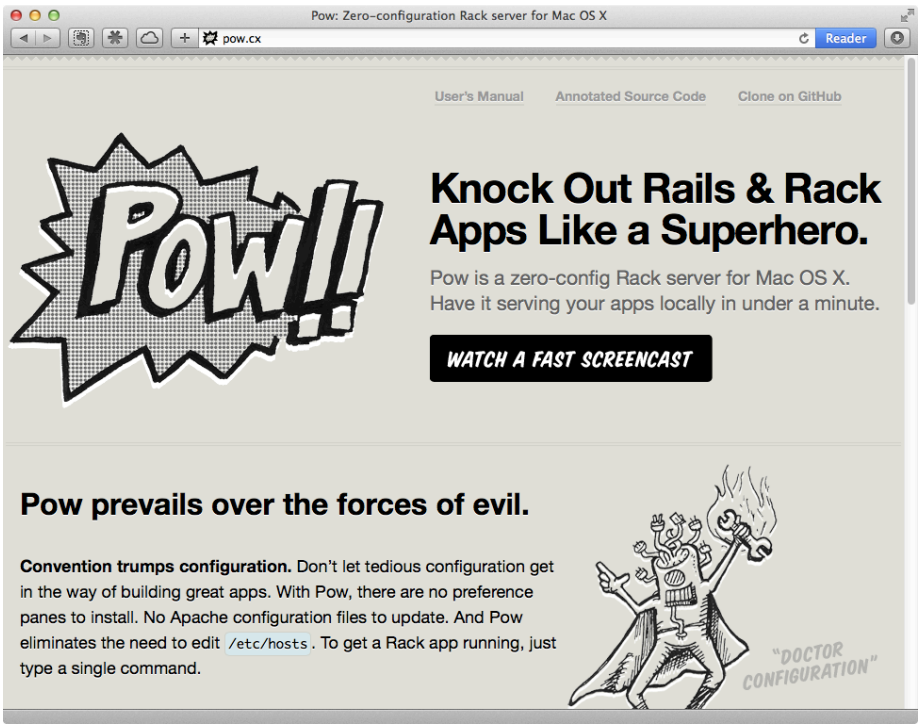
This only applies to URL which is generated by rails, such as root_path. Manually typing the URL will not get the `default_url_options` attached.

And than the UI to switch. Since we want the language available everywhere, put it in the `applicotian.html.erb` file.

```
<a href='?locale=zh-TW'>中</a> | <a href='?
locale=en'>EN</a>
```

Pow local server

In development, we have been using `http://0.0.0.0:3000`. Actually there is a better local development server called Pow. Pow is developed by 37signals which contributes a lot to the rails community.



It features auto loading the server and allow accessing the local rails app with something like `http://yourapp.dev/`.

To install pow, execute the following command.

```
$ curl get.pow.cx | sh
```

Pow is very easy to use. But `powder` makes it even easier. It makes setting up the `.dev` domain with just one line within the rails app directory.

To install the `powder` gem, execute:

```
$ gem install powder
```

Note: We are running `gem install` instead of adding `powder` in the Gemfile because this is a tool-chain on our system. The rails app does not depends on this gem.

After having both `pow` and `powder` installed, we can change to the rails app project directory and execute the following command:

```
$ powder link
Your application is now available at
http://gallery.dev/
```

And from now on, we have a nice URL and we can have multiple rails app running in local without worrying about the port conflicts.

Restarting Pow server

But now we need a way to restart the server sometimes. When a new request comes, Pow cheches a file in `tmp/restart.txt`. If the file is `touched` recently, it will restart the server.

So, in order to restart the pow server, we use the following command. (Assuming we are at the root directory of the rails project.)

```
$ touch tmp/restart.txt
```

ruby-on-rails-101

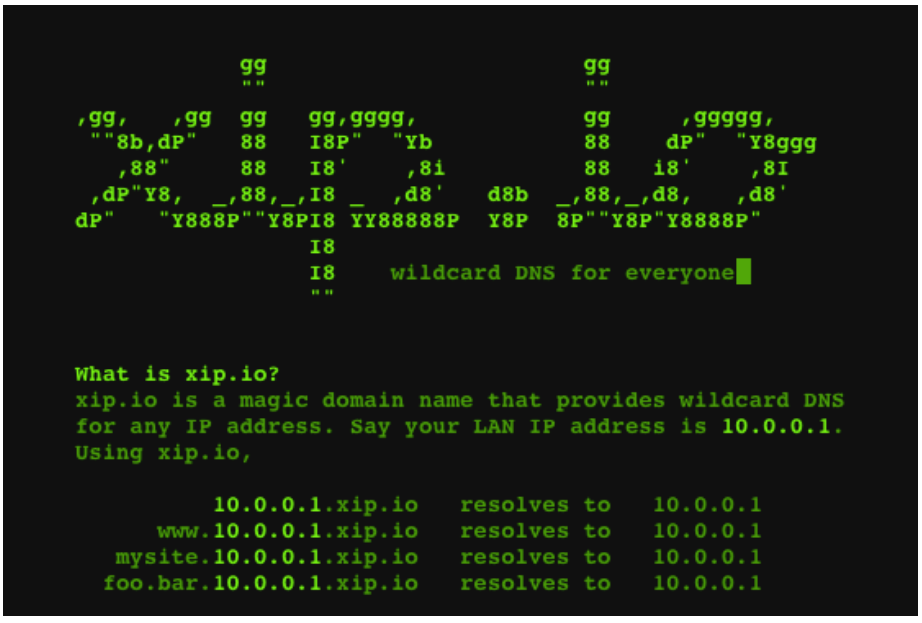
Mobile devices testing via local network

In many scenarios we want to test our rails app in mobile devices, tablets or smartphones.

If we are using the rails build-in `rails server`, we can access the app on mobile device under the same local network with the `<local_ip>:3000`.

The `yourapp.dev`, however, is only available in the development machine because it modifies the `/etc/hosts` file. We can't access it from mobile devices.

Pow comes with a custom DNS server called `xip.io` – also from 37signals. It is made to make pow-powered rails app accessible through local network.



It comes from pow without any configuration needs. Just open the mobile browser in the device and type in the URL in the following format.

```
http://<pow site name>.<dev machine IP>.xip.io
```

For instance, the develop machine is at 10.0.0.3 and the rails app is registered as “gallery” in pow.

In local, we access the app through `http://gallery.dev`.

In mobile device, we use `http://gallery.10.0.0.3.xip.io`.

Again, you can have multiple devices testing multiple rails app at the same time.

Deployment database config

We set the database connection configuration in the `database.yml` file. This is convenient in development environment. But in production, we would better put the database connection and secret out of the database configuration file. We can do that by putting the value in environment variable and use `ENV` to access them.

```
production:
  adapter: postgresql
  encoding: unicode
  pool: 5
  database: <%= ENV['APP_DATABASE'] %>
  username: <%= ENV['APP_USERNAME'] %>
  password: <%= ENV['APP_PASSWORD'] %>
  host: <%= ENV['APP_HOST'] %>
  port: <%= ENV['APP_PORT'] %>
```

Introducing git

Now we have the files ready. What are the options to put the rails app on server?

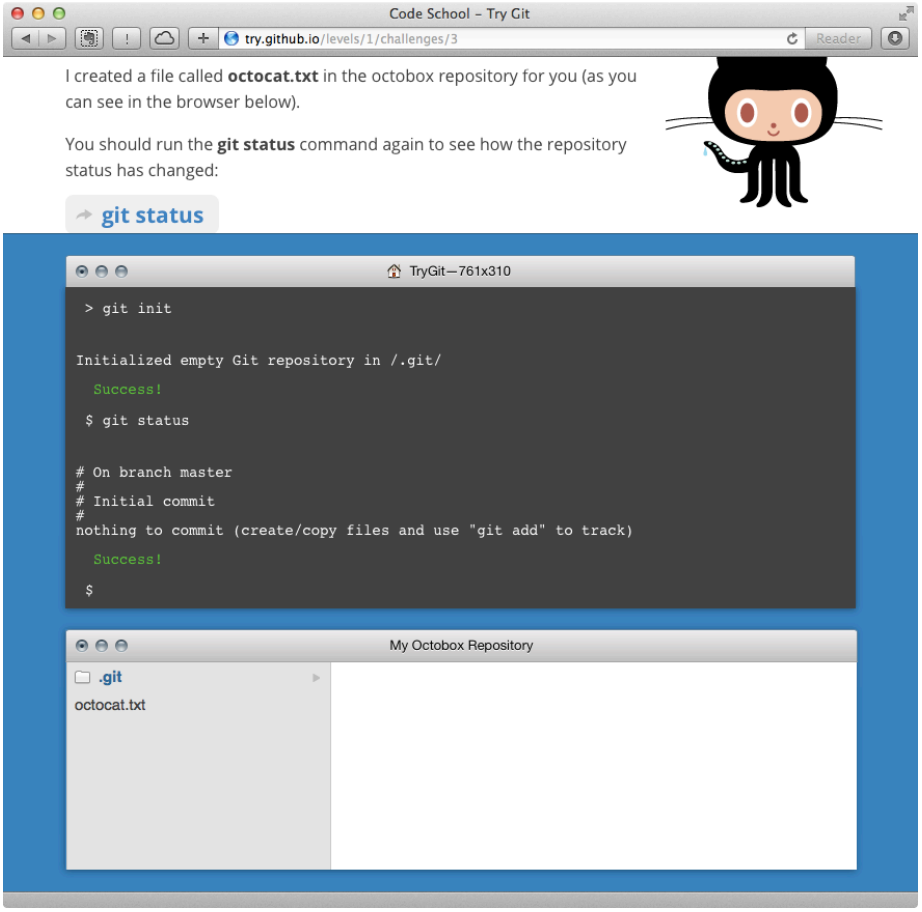
1. FTP upload?
2. Git repo

In rails community, we normally use the git approach.

What is Git?

Git is a source code management tool-chain with version control. It is created by Linus Torvalds, the man behind linux kernel.

Github's interactive course is a good place to get started learning git.



Lab – "Try Git"

Finish the Github "Try Git" course.

1. Open http://try.github.io in your browser.
2. Type the command from the grey box into shell prompt.
3. Check the files finder for the current working files.

Tips: You can click on the command in the grey box and the command will be typed into shell automatically.

Apply git to our project

So how we apply the git in our project?

1. In shell, change directory into the rails app project.
2. `git init` the local repo.
3. `git add .` to add all the files to the repo.
4. `git commit -m "init repo with all files"` to commit the just added changes. It must comes with a message.
5. `git add the_changed_files` after every time some files are changed (and you need that changes)
6. `git commit -m "message here"` to commit the changes every time you make.

Please note that this all happen in your local device. Git repos are distributed so every one own a local code base. There are several benefits on this:

1. Can work in offline.
2. No centralized server.
3. Fast code changing. (branch switching)

Push git to server repo

But you want a server? There are some git services.

1. Github
2. BitBucket

```
$ git remote add origin ssh://.......
```

Then every time when your local code are ready to push to the others:

```
$ git push origin master
```

Or use self hosted git service

1. You need a server, or VPS
2. SSH access
3. Setup Git repo

We have used `git init` to make the current folder a git managed repo, with all the git controlling files put inside a `.git` directory. When we are creating our now git repo for others to use, we just want those git controlling files by using the `--bare` option.

```
$ mk a-new-repo.git
$ cd a-new-repo.git
$ git init --bare
```

Git is something powerful yet not easy to learn. Here are more resources to help you get started.

1. Git Book
2. Git flow

Deploying options

So what server we can use to host rails app?

1. Heroku
2. Brightbox
3. AWS EC2
4. OpenShift
5. Dedicated/VPS Server

Heroku getting started guide

rails 3: https://devcenter.heroku.com/articles/getting-started-with-rails3

rails 4: https://devcenter.heroku.com/articles/getting-started-with-rails4

Addons: https://addons.heroku.com

Dedicated Server

There are quite a lot tutorials discussing how to deploy the rails app to linode. I found this one helpful to me.

https://gist.github.com/jpantuso/1031946

Although the steps are based on linode, it should be quite similar to deploy on the other self-managed server.