# AWS S3를 이용한 Image Upload

## 도입

지금까지 웹에 표시할 이미지들을 `app/assets/images`에 저장했습니다.

그런데 사용자들이 업로드하는 이미지도 서버에 저장해야할까요?

비싼 서버 대신 dropbox처럼 저렴한 저장공간을 빌릴 수는 없을까요?

이 예제는 이미지 업로드만을 다루지만 모든 형식의 파일 업로드에 적용 가능합니다.

## 목표

1. rails scaffold를 이용해서 5초만에 CRUD를 완성한다.
2. figaro gem을 이용해서 환경변수를 관리한다.
3. Amazon IAM User를 이용해서 안전하게 AWS 계정을 관리한다.
4. carrierwave, fog-aws,  AWS S3를 이용해서 게시판에 Image upload를  추가한다.
5. minimagick을 이용해서 이미지의 크기를 변경한 뒤 업로드 해본다.

# Scaffold

```
$ rails generate scaffold Post title:string content:text
```

model, view, controller, route, test를 resources 형식에 맞게 모두 만들어준다.

기차는 rails 위에서 가장 빠르다. Rails가 미리 제공하는 기능들을 적극 활용하면 개발이 매우 쉬워진다.

- root를 추가하자

```
#config/route.rb
Rails.application.routes.draw do
  root 'posts#index'
  resources :posts
end
```

- migrate한 뒤 서버를 켜 보라. CRUD가 정말 뼈대만 완성된 모습을 볼 수 있다.

```
$ rake db:migrate
```

# 이미지 업로드 하기

## 1. 젬파일 추가

```
#Gemfile
gem 'carrierwave'
gem 'mini_magick'
gem 'fog-aws'
gem 'figaro'
```

## 2. ImageMagick 설치하기

- 이미지를 변환하기 위한 툴이다.
- mini_magick을 사용하기 위해서는 imagemagick을 먼저 설치해야 한다.
- 공식사이트: https://www.imagemagick.org/script/index.php
- 설치방법
  - Mac OS

    ```
    $ brew install imagemagick
    ```

  - Ubuntu Linux

    ```
    $ sudo apt-get install imagemagick
    ```

## 3. image_uploader 만들기

```
$ rails generate uploader image
```

`app/uploaders/image_uploader.rb` 파일이 생성되었습니다. 이름에 주의하세요!

```
class ImageUploader < CarrierWave::Uploader::Base
```

## 4. model에 string column 추가

```
$ rails g migration add_image_to_posts image:string
$ rake db:migrate
```

> migration file이란?
>
> 쉽게 database를 조작하는 명령 파일이라고 생각하시면 됩니다.
>
> - 모두 db/migrate에 있습니다.
> - `rake db:migrate` 는 이전의 migrate 명령을 실행할 때 없었던 파일들만 골라서 migrate합니다. 이 때까지 새로 migrate파일을 수정하면 `rake db:drop` 을 했던 이유가 여기 있습니다.

더 자세한 내용은 공식문서를 참고하세요! http://guides.rubyonrails.org/active_record_migrations.html

## 5. model에 uploader를 얹기(mount)

```
#app/models/post.rb
class Post < ActiveRecord::Base
  mount_uploader :image, ImageUploader  #uploader class이름과 동일하게
end
```

## 6. CRUD를 수정

먼저 form에서 `:image` input field를 만들어주고 `form_with` 를 `form_for` 형식으로 바꿔줍니다.

Post model에 moute ImageUploader가 있는 경우, form_for @post는 `enctype="multipart/form-data"` 를 자동으로 넣어줍니다.

일반 `form_tag` 로 파일 업로드를 할 경우 `multipart: true` 가 반드시 들어가야 합니다.

`app/views/posts/_form.html.erb`

```
<%= form_for(@post) do |f| %>
  <% if @post.errors.any? %>
    <div id="error_explanation">
      <h2><%= pluralize(@post.errors.count, "error") %> prohibited this post
from being saved:</h2>

      <ul>
      <% @post.errors.full_messages.each do |message| %>
        <li><%= message %></li>
      <% end %>
      </ul>
    </div>
  <% end %>

  <div class="field">
    <%= f.label :title %><br>
    <%= f.text_field :title %>
  </div>
  <div class="field">
    <%= f.label :content %><br>
    <%= f.text_area :content %>
  </div>
  <div class="field">
    <%= f.label :image %><br>
    <%= f.file_field :image, accept: 'image/png,image/gif,image/jpeg' %>
  </div>
  <div class="actions">
    <%= f.submit %>
  </div>
<% end %>
```

PostsController가 `params[:post][:images]`를 받을 수 있도록 controller 최하단의 `post_params` method를 수정합니다.

`app/controllers/posts_controller.rb`

```
class PostsController < ApplicationController
  private
    def post_params
      params.require(:post).permit(:title, :content, :image)
    end
end
```

이제 view에서 이미지가 보이도록 해주겠습니다.

`app/views/posts/show.html.erb`

```
<p id="notice"><%= notice %></p>

<p>
  <strong>Title:</strong>
  <%= @post.title %>
</p>

<p>
  <strong>Content:</strong>
  <%= @post.content %>
</p>

<p>
<%= image_tag("#{@post.image}") %>
</p>

<%= link_to 'Edit', edit_post_path(@post) %> |
<%= link_to 'Back', posts_path %>
```

이미지를 업로드 해봅시다. 잘 작동하죠? 이제 public/을 열어봅시다. 앗! 뭔가 이상하지 않나요?

## 7. figaro gem

figaro gem은 환경변수를 쉽게 관리할 수 있게 도와줍니다.

> **환경변수**란?
>
> 쉽게 컴퓨터가 동작하는 데 필요한 설정값으로 이해하면 됩니다. 비밀번호를 환경변수로 지정하면 나머지 파일에서는 ENV['password']와 같이 그 값을 참조함으로써 정보를 숨길 수 있습니다.

figaro gem을 사용하기 위해서는 terminal에 아래의 명령어를 추가적으로 입력해야 합니다.

```
$ bundle exec figaro install
```

이 명령은 자동으로 `config/application.yml` file을 만들고 이를 `.gitignore` 에 추가해줍니다. 애써 비밀번호를 숨겨놓은 파일을 다시 git repo에 올리면 안 되니까요.

`config/application.yml` 에 아래와 같이 입력한다.

```
AWS_ACCESS_KEY_ID: "YOUR_ACCESSK_KEY_ID"
AWS_SECRET_ACCESS_KEY: "YOUR_SECRET_KEY"
SECRET_KEY_BASE:
"40db552d52f3c84d732f099c54813c822455ace03134c1f4246c043778a756ccc293a72a53a5
9e154a1f32c9b7b5e93a16e7bfca5f103b0401cef97bdfff3122"
```

SECRET_KEY_BASE에는 terminal에 `rake secrets` 을 입력하면 나오는 긴 스트링을 복사해서 붙여넣으면 됩니다.

## 8. AWS IAM User

root 계정은 모든 권한을 가지고 있기 때문에 secret_key가 노출될 경우 손실이 막대합니다. 따라서 보안을 위해 s3만 이용할 수 있는 유저를 따로 만들어 봅시다.

### 9. AWS S3 bucket 만들기

파일을 저장할 s3 저장소(bucket)를 만들겠습니다.

http://docs.aws.amazon.com/AmazonS3/latest/gsg/CreatingABucket.html 을 참조하면 쉽게 만들 수 있습니다.

### 10. aws.rb 파일 추가

`fog-aws`를 사용하기 위해서는 `config/initializers`에 aws.rb 파일을 만들어야 합니다.

```
$ touch config/initializers/aws.rb
```

`config/initializers/aws.rb` 에 아래의 코드를 추가해 주세요.

```
CarrierWave.configure do |config|
  config.fog_provider = 'fog/aws'                        # required
  config.fog_credentials = {
    provider:              'AWS',                        #required
    aws_access_key_id:     ENV['AWS_ACCESS_KEY_ID'],     # required
    aws_secret_access_key: ENV['AWS_SECRET_ACCESS_KEY'], # required
    region:                'ap-northeast-2',             # optional,
defaults to 'us-east-1'
  }
  config.fog_directory  = 'your bucket name'             # required
  config.fog_public     = false                          # optional,
defaults to true
end
```

### 11.  storage :fog 로 설정 변경

`app/uploaders/image_uploader.rb` 파일을 아래와 같이 바꿔줍니다.

```
#storage :file
storage :fog
```

AWS S3에 해당 이미지 파일이 잘 저장되는지 확인합니다.

### 11. MiniMagick으로 이미지 크기 조절하기

```ruby
class ImageUploader < CarrierWave::Uploader::Base

  # Include RMagick or MiniMagick support:
  # include CarrierWave::RMagick
  # MiniMagick을 사용하기 위해 아래 주석을 해제합니다.
  include CarrierWave::MiniMagick

  # file을 쓰면 서버에 파일이 저장되지만 fog를 쓰면 s3에 파일이 저장됩니다.
  #storage :file
  storage :fog

  # aws.rb에서 지정된 s3 bucket내에서 파일의 분류를 설정하는 코드로 거의 변경할 일이 없습니다.
  def store_dir
    "uploads/#{model.class.to_s.underscore}/#{mounted_as}/#{model.id}"
  end

  # 이미지 사이즈가 너무 크지 않도록 잘라서 저장합니다.
  process resize_to_fit: [200, 300]

  # view file에서 post.image.thumb로 접근 가능한 이미지를 하나 더 저장합니다.
  version :thumb do
     process resize_to_fit: [50, 50]
  end

end
```

- `resize_to_fit: [width, height]` 는 원본 이미지를 비율에 맞게 크기조정을 합니다. (단, 가로 세로 중 적어도 하나는 지정된 width 혹은 height와 일치하게 합니다.)
- `resize_to_fill: [width, height]` 는 원본 이미지를 정확히 width * height의 크기로 잘라서 저장합니다.
- 조건에 따라 이미지 크기 변경하기 등은 carrierwave 문서 를 참조합시다.

---

### 참고한 페이지

https://github.com/carrierwaveuploader/carrierwave

https://hcn1519.github.io/articles/2016-02/carrierwave

## contributor

이현민