

Rails 3

The Deep Dive

Jeremy McAnally

Tweet at @jm

<http://jeremymcanally.com>

Jeremy McAnally

Written lots of codes

<http://github.com/jm>

Jeremy McAnally

Written some books

<http://humblelittlerubybook.com>

Jeremy McAnally

Overcame flupus
[no url. it's a new disease]

Jeremy McAnally

Rails 3 Upgrade Handbook

<http://railsupgradehandbook.com>

Rails 3

The Deep Dive

Rack

Everyone's favorite web server interface

Rack

- Rack is a general web server interface

Apache/Passenger

Mongrel

Thin

Unicorn

Rack

Rails

Sinatra

Rack

Ramaze

```
# basic.rb
class Hello
  def call(env)
    [
      200,
      {"Content-Type" => "text/html"},
      "Hello, RailsConf!"
    ]
  end
end
```

```
# config.ru  
require 'basic'
```

```
run Hello.new
```

```
# config.ru
run Proc.new do |e|
  [
    200,
    { "Content-Type" => "text/html" },
    "Hello, RailsConf!"
  ]
end
```

Rack

- Rack is a general web server interface
- Useful for building small, tight applications and services

```
class LoadAverage
  def call(env)
    [
      200,
      {"Content-Type" => "text/html"},
      load_averages
    ]
  end

  def load_averages
    `uptime`.split("averages: ")[1]
  end
end
```

Rack

- Rack is a general web server interface
- Useful for building small, tight applications and services
- Bare metal is awesome
 - ...but who doesn't love a framework?

```
get "/" do
  "Howdy from Sinatra!"
end
```

```
get "/other" do
  "That other URL..."
end
```

```
require 'rubygems'  
require 'sinatra'
```

```
disable :run  
set :env, :production
```

```
require 'sinatra_example'  
run Sinatra.application
```

Labs: Rack apps

- Build a Sinatra app to serve personal info about you (e.g., “/name” is your name, “/vcf” is your vcf file, etc.)
- Build a rack app to make a “downforeveryoneorjustme.com” clone
- Build a Rack app to serve up your followed tweet stream except substitute the word “diapers” every time someone says “Rails” or “flotilla”

```
require 'net/http'
class Down
  def call(env)
    [
      200,
      {"Content-Type" => "text/html"},
      down?(env[ 'REQUEST_PATH' ])
    ]
  end

  def down?(url)
    url.slice!(0)
    response = "you"
    begin
      Net::HTTP.get_response(url, "/")
    rescue
      response = "everyone"
    end

    response
  end
end
end
```

Rack

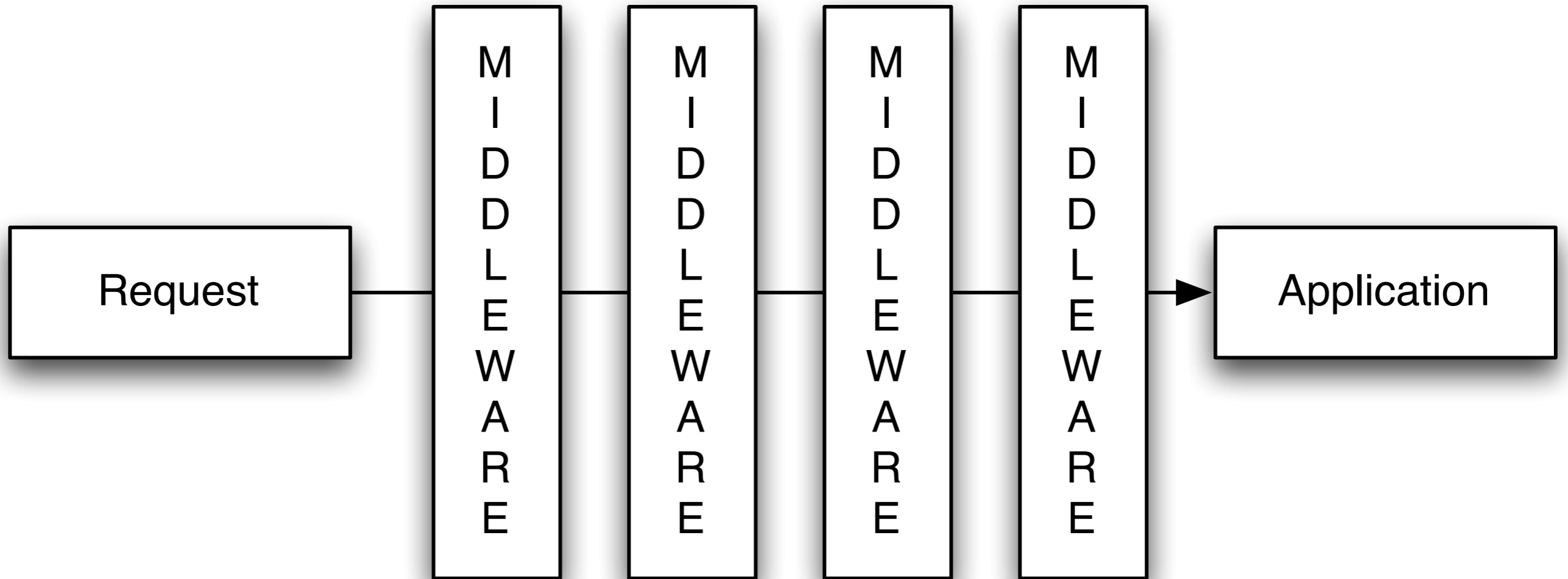
- Rails 2.3 integrated Rack more deeply

- ActionController::Dispatcher.new is a Rack endpoint
- Parameter parser is a middleware
- Controllers are Rack endpoints
- The router is a Rack app
- Custom middlewares can be used in the application

```
config.middleware.use Rack::Cache
```

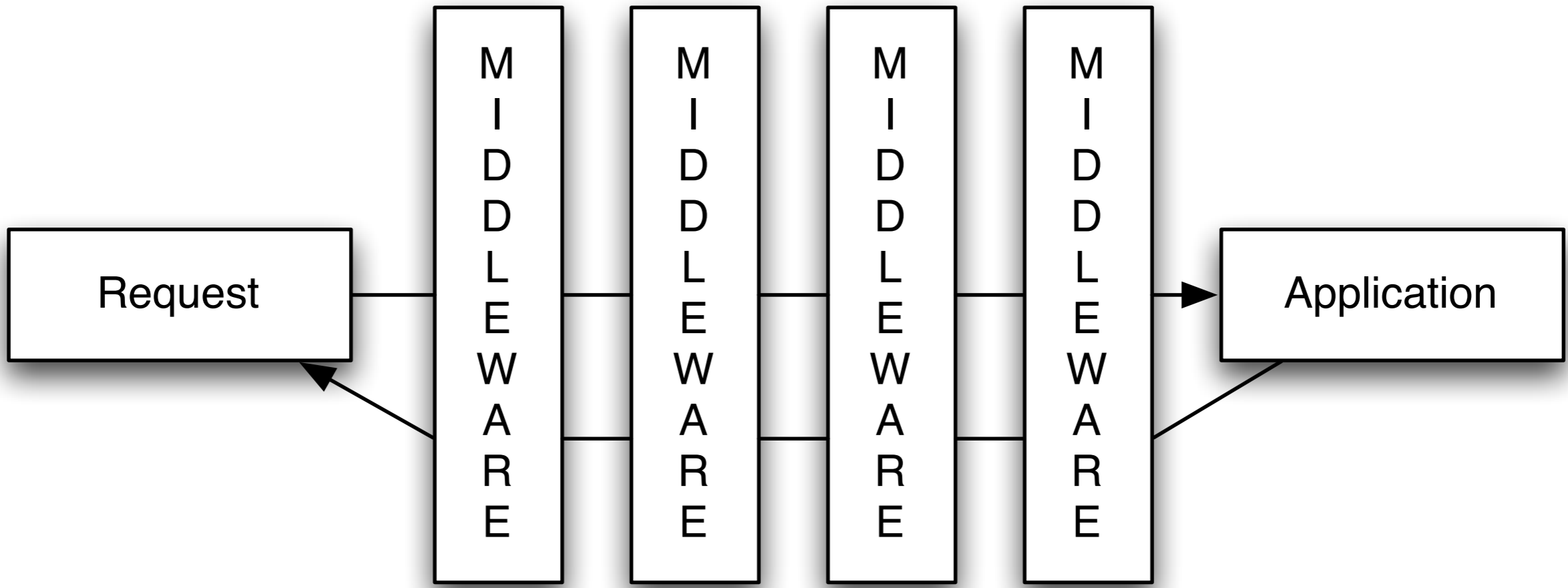
Rack

- Middlewares can be thought of as “filters” for requests



```
class NoOpMiddleware
  def initialize(app)
    @app = app
  end

  def call(env)
    @app.call(env)
  end
end
```



```
require 'net/http'
class Down
  def call(env)
    [
      200,
      {"Content-Type" => "text/html"},
      down?(env[ 'REQUEST_PATH' ])
    ]
  end

  def down?(url)
    url.slice!(0)
    response = "you"
    begin
      Net::HTTP.get_response(url, "/")
    rescue
      response = "everyone"
    end

    response
  end
end
end
```

```
class SlasherMiddleware
  def initialize(app)
    @app = app
  end

  def call(env)
    env[ 'REQUEST_PATH' ].slice!(0)
    @app.call(env)
  end
end
```

```
require 'net/http'
class Down
  def call(env)
    [
      200,
      {"Content-Type" => "text/html"},
      down?(env[ 'REQUEST_PATH' ])
    ]
  end

  def down?(url)
    response = "you"
    begin
      Net::HTTP.get_response(url, "/")
    rescue
      response = "everyone"
    end

    response
  end
end
end
```

```
require 'json'
require 'net/http'
class Twitter
  def call(env)
    [
      200,
      {"Content-Type" => "text/html"},
      get_tweets
    ]
  end

  def get_tweets
    j = Net::HTTP.get("api.twitter.com",
                     "/1/statuses/public_timeline.json")
    tweets = JSON.parse(j)

    tweets.map do |t|
      "<p><b>#{t['user']['screen_name']}</b>
      #{t['text']}</p>"
    end.join
  end
end
end
```

```
class Diaperator
  def initialize(app)
    @app = app
  end

  def call(env)
    status, headers, body = @app.call(env)
    [
      status,
      headers,
      [body.first.gsub(/(the|flotilla)/,
                      'DIAPERS')]
    ]
  end
end
```

```
require 'twitter'  
require 'diaperator'
```

```
use Diaperator  
run Twitter.new
```

Rack

- Middlewares can be thought of as “filters” for requests
- Middlewares can replace a large chunk of functionality in Rails (and Rack) applications

```
# http://github.com/josh/rack-openid
```

```
MyApp = lambda { |env|  
  if resp = env["rack.openid.response"]  
    case resp.status  
    when :success  
      ...  
    when :failure  
      ...  
  else  
    [  
      401,  
      {"WWW-Authenticate" => 'OpenID identifier="http://  
example.com/"'},  
      []  
    ]  
  end  
}
```

```
use Rack::OpenID
```

```
run MyApp
```

Labs: Middleware

- Create a middleware to timestamp every response
- Build a middleware for a simple application that returns the proper format requested (i.e., “whatever.xml” returns XML). You can return an array from the application and have a middleware render it for you...

Rack

- Rails 2.3 integrated Rack more deeply
- Rails 3 makes Rack an integral component

- Central Rack object is now Application

```
# config/application.rb
module RackApplication
  class Application < Rails::Application
    # ...
  end
end
```

- Central Rack object is now Application
- Action Dispatch extends Rack

- Exception handling
- Parameter parsing
- Session/cookie management
- Environment management
- ...more.

```
use ActionController::Static
use Rack::Lock
use Rack::Runtime
use Rails::Rack::Logger
use ActionController::ShowExceptions
use ActionController::RemoteIp
use Rack::Sendfile
use ActionController::Callbacks
use ActionController::Cookies
use ActionController::Session::CookieStore
use ActionController::Flash
use ActionController::ParamsParser
use Rack::MethodOverride
use ActionController::Head
use ActiveRecord::ConnectionAdapters::ConnectionManager
use ActiveRecord::QueryCache
run RackApplication::Application.routes
```

- Central Rack object is now Application
- Action Dispatch extends Rack
- Controllers' actions are actually mini applications

```
> HomeController.action(:index)
=> #<Proc:0x00000103149728@actionpack-3.0.0.beta3>
```

Rack

- Rails 2.3 integrated Rack more deeply
- Rails 3 makes Rack an integral component
- And you can totally rip this apart

```
config.middleware.use Rack::Cache
config.middleware.delete(Rack::Cache)
config.middleware.swap Rack::Cache, A::Cache
config.middleware.insert_before Rack::Cache,
                                Warm::Cache
```

Rack

- You can use your own middlewares really easily

```
class TimestampMiddleware
  def initialize(app)
    @app = app
  end

  def call(env)
    status, headers, @response = @app.call(env)
    [
      status,
      headers,
      self
    ]
  end

  def each(&block)
    block.call("<!-- #{Time.now} -->\n")
    @response.each(&block)
  end
end
```

```
module RackApplication
  class Application < Rails::Application
    config.encoding = "utf-8"
    config.middleware.use "TimestampMiddleware"

    config.filter_parameters += [:password]
  end
end
```

```
<!-- 2010-06-07 09:04:08 -0400 -->
```

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
  <title>RackApplication</title>
```

```
  <script src="/javascripts/prototype.js?1275910111  
javascript"></script>
```

```
<script src="/javascripts/effects.js?1275910110" t  
javascript"></script>
```

Rack

- You can use your own middlewares really easily
- Many new plugins use middlewares extensively

- Devise provides authentications
- CarrierWave handles uploads
- ...I'm sure there are more. :)

Rack

- You can mount your Rack application using the router

```
Hello = lambda { |env|  
  [  
    200,  
    {"Content-Type" => "text/html"},  
    ["Hello!"]  
  ]  
}
```

```
RackApplication::Application.routes.draw do |map|  
  match "/hello", :to => Hello  
end
```

```
class Hello < Sinatra::Base
  get "/hello" do
    "Hello!"
  end
end
```

```
RackApplication::Application.routes.draw do |map|
  match "/hello", :to => Hello
end
```

Rack

- You can mount your Rack application using the router
- You can also URL map to other Rack apps

```
map "/" do
  run RackApplication::Application
end
```

```
map "/howdy" do
  run Greeting.new
end
```

```
map "/index" do
  run HomeController.action(:index)
end
```

Labs: Rack + Rails

- Convert one of the earlier labs into a Rails app and use its middleware
- Create a Rack app to display some system stats and mount it at “/system” in a Rails 3 app
- Create a Rack middleware to transparently translate Rails actions using the babelfish gem

Ruby tricks

Rails 3 turns tricks like Siegfried and Roy.

- Modularity vs. metaprogramming
- Method compilation
- `ActiveSupport::Concern`

- Provides included/extended callback chain
- Include/extend InstanceMethods/
ClassMethods automatically

Plugins

Because public API's are awesome

Plugins

- Many plugins in Rails 2 are hacks

- `alias_method_chain`
- Tons of `include/extend` hackery
- Stomping on each other
- A lot of duck punching

Plugins

- Many plugins in Rails 2 are hacks
- Now there's a "real" API

Railties

- Railties give you access to the load process and Rails environment without hackery

```
class FancyPlugin < Rails::Railtie
  railtie_name :fancy_plugin

  initializer :set_it_up do |app|
    # do stuff with app.config etc...
  end
end
```

```
class FancyPlugin < Rails::Railtie
  railtie_name :fancy_plugin

  rake_task do
    load "fancy_plugin/tasks.rake"
  end

  initializer :set_it_up do |app|
    # do stuff with app.config etc...
  end
end
```

Railties

- Railties give you access to the load process and Rails environment without hackery
- Only create one if necessary!

Labs: Plugin basics

- Create a plugin that provides a helper for displaying a message configured in the environment
- Create a plugin that provides a Rake task to zip the project and place it in a “release” folder in Rails.root

Engines

- Engines were previously a third party plugin, but now they're first class citizens

```
vendor/  
  plugins/  
    my_engine/  
      app/  
        . . .  
      config/  
        routes.rb  
        locales/  
          . . .  
      lib/  
        tasks/  
          . . .
```

```
module YourApp
  class Engine < Rails::Engine
    engine_name :your_engine
  end
end
```

Engines

- Engines were previously a third party plugin, but now they're first class citizens
- They're even more flexible than before now

```
paths.app = "app"
paths.app.controllers = "app/controllers"
paths.app.helpers = "app/helpers"
paths.app.models = "app/models"
paths.app.metal = "app/metal"
paths.app.views = "app/views"
paths.lib = "lib"
paths.lib.tasks = "lib/tasks"
paths.config = "config"
paths.config.initializers = "config/initializers"
paths.config.locales = "config/locales"
paths.config.routes = "config/routes.rb"
```

```
class YourEngine < Rails::Engine
  config.middleware.use MyMiddleware

  config.load_paths << "your/path"
end
```

Labs: Engines

- Craft an engine that uses one of the middlewares from previous labs
- Create an engine to display your tweets via the Twitter API (or a simple blog engine if you aren't cool enough for Twitter)

Generators

- Generators are now a whole different animal

```
class FileGenerator < Rails::Generator::Base
  def manifest
    record do |m|
      m.file "my_file.txt", ""
    end
  end
end
```

```
class FileGenerator < Rails::Generators::Base
  def create_file
    create_file "my_file.txt", ""
  end
end
```

To the internets!

Generators

- Generators are now a whole different animal
- The new hooks are ridiculously cool

```
config.generators do |g|
  g.template_engine :haml
  g.test_framework :rspec
  g.fixture_replacement :factory_girl
end
```

```
config.generators do |g|
  g.template_engine :haml
  g.test_framework :rspec
  g.fixture_replacement :factory_girl
end
```

```
generate test_unit:unit_test
generate rspec:unit_test → vendor/
generate shoulda:unit_test   plugins/
etc.                          shoulda/
etc.                          lib/
etc.                           generator
                               unit_te
```

.....

Labs: Generators

- Create a generator that creates mustache (<http://github.com/defunkt/mustache>) views for generated controllers
- Group up, find a popular plugin with a generator, and work on upgrading it

Modularity

Get all up in Rails' bizness

Modularity

- Rails 3 provides a few places to jack your own code into
- Makes it really easy to tear the stack apart and use only what you want

Thanks!

<http://railsupgradehandbook.com>