Ruby on Rails Short Course: Configure and Deploy

William Sobel

UC Berkeley RAD Lab

Outline of the day

1. Web apps, MVC, SQL, Hello World
2. Just enough Ruby
3. Basic Rails

Lunch break

4. Advanced model relations
5. AJAX & intro to testing
6. Configure and Deploy

Informal discussion: RoR and pedagogy

Section 6

- Configuration
  - What about convention over configuration!
- Deployment
  - Capistrano
- Mongrel Application Server
- Routing
- REST - Representational State Transfer

Configuration

- The first and only configuration file you'll need to modify to get started
  - **`config/database.yml`**
- Tell the system a little about your database
  - Database connector
  - User name
  - Password

Database Configuration

config/database.yml

```
# ….
development:
  adapter: mysql
  database: pub_development
  username: root
  password:
  socket: /tmp/mysql.sock

# Warning: The database defined as 'test' will be erased and
# re-generated from your development database when you run 'rake'.
# Do not set this db to the same as development or production.
test:
  adapter: mysql
  database: pub_test
  username: root
  password:
  socket: /tmp/mysql.sock

production:
  adapter: mysql
  database: pub_production
  username: root
  password:
  socket: /tmp/mysql.sock
```

That's It

- Now your ready to start writing you app.
- Some additional configuration
  - Sessions
  - Logging
  - Email
  - Directories
  - Plugins
  - Mime-types
  - etc…

Sessions

- **`config/environment.rb`**
- To store you session data in the database, uncomment the following lines:

```
# Use the database for sessions instead of the file system
# (create the session table with 'rake create_sessions_table')
config.action_controller.session_store = :active_record_store
```

- Use rake to create the session table:

```
rake create_sessions_table
```

- Now your sessions are stored in the database

Logging

- Advanced logging with Hodel 3000

```
# Force all environments to use the same logger level
# (by default production uses :info, the others :debug)
config.log_level = :debug
require 'hodel_logger'
config.logger = Hodel3000CompliantLogger.new(config.log_path)
```

- Very detailed logger that provides easy to parse log formats for analysis and replay of events

Email

- config/environments/production.rb

```
config.action_mailer.raise_delivery_errors = true
config.action_mailer.delivery_method = :smtp

config.action_mailer.server_settings = {
  :address => "mail.mysite.com",
  :port => 25,
  :domain => "mysite.com",
  :authentication => :login,
  :user_name => "support@mysite.com",
  :password => 'password',
}
```

- config/environments/test.rb

```
config.action_mailer.delivery_method = :test
```

Capistrano

- Deployment made easy
  - DSL for running shell scripts on multiple machines in parallel
  - Rule based deployment with dependencies
  - Very Extensible
- Basic Recipes for Rails
- Gem based recipes available for hosting providers
- Initial deploy in 5 minutes
- 30 second deploys after that

Basics

- In the world of Rails, there are three roles for servers. A server can have more than one role
  - The web server
  - The application server
  - The database
- Like everything else, many conventions
- Easy to get started

```
cap --apply-to <path> AppName
```

- Puts up a system maintenance page when upgrading

In The Box

- You're required to supply the following:
  - An application name
  - A target directory root for deployment
  - A domain for the web site
  - A user name to deploy as
  - A repository to check out the code
    - Can be svn, CVS, or Perforce (more adapters have been written)
  - The rails environment
- For a single machine deploy, this is all you need to configure

Many Built in Tasks

- setup - Setup up the host machines for initial deploy
- deploy - Deploy this application either initial or upgrade
  - Checks out the code to a release directory
  - Creates a sym link to the directory called current
  - Restarts the server
- deploy_with_migrations - Same as above with database schema migrations
- rollback - Revert to previous deploy
- diff_from_last_deploy - Dumps out a list of changes since you last deployed
- update_code - Update the code without doing a full deploy
- cleanup - Remove old deploy directories, leaves the last 5

Conventions

- The following directory structure is recommended:
- AppName/shared
  - These are files that will be preserved between deploys
    - log - Log files
    - pid - Process ids for daemons
    - system - The system maintenance page
    - media and application resources like images
- AppName/releases
  - Release directories (for ever deploy a new directory is created)
- AppName/repos
  - Contains the scm repository
- AppName/current -> ../releases/<release>

Getting Fancy

- You can add before and after actions to any one of the deploy steps

```
task :after_update_code, :except => { :no_release => true } do
   run <<-CMD
     cd #{release_path}/app/views/subscription_mailer &&
     ln -nfs ../common .
   CMD
 end
```

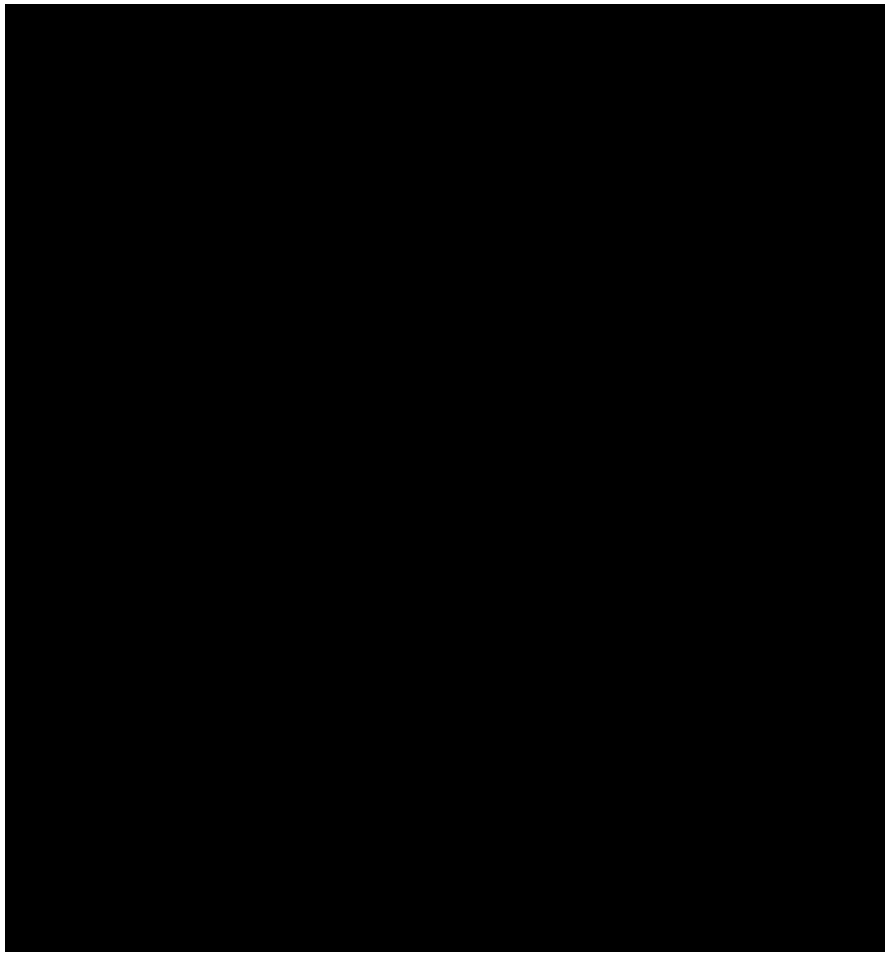
- This task will run after the code has been updated and create a symbolic link to another directory
- A task can run a script anywhere

```
task :restart_daemon, :roles => :app do
   run <<-CMD
     cd #{current_path}/daemon &&
     ./daemon stop && ./daemon start
   CMD
 end
```

- This task will run on the application machine(s)

Standard Deploy

- Apache or Lighttpd
  - Reverse proxy
  - Static content
- Mongrel
  - HTTP handler
  - Application Server
  - Rails application
- Database
  - MySQL
  - Postgsql
  - Oracle
  - MsSQL
  - Sybase
  - …



Not Only Ruby

- Capistrano can be used to deploy any web application
- Has been used to deploy Java application

Mongrel

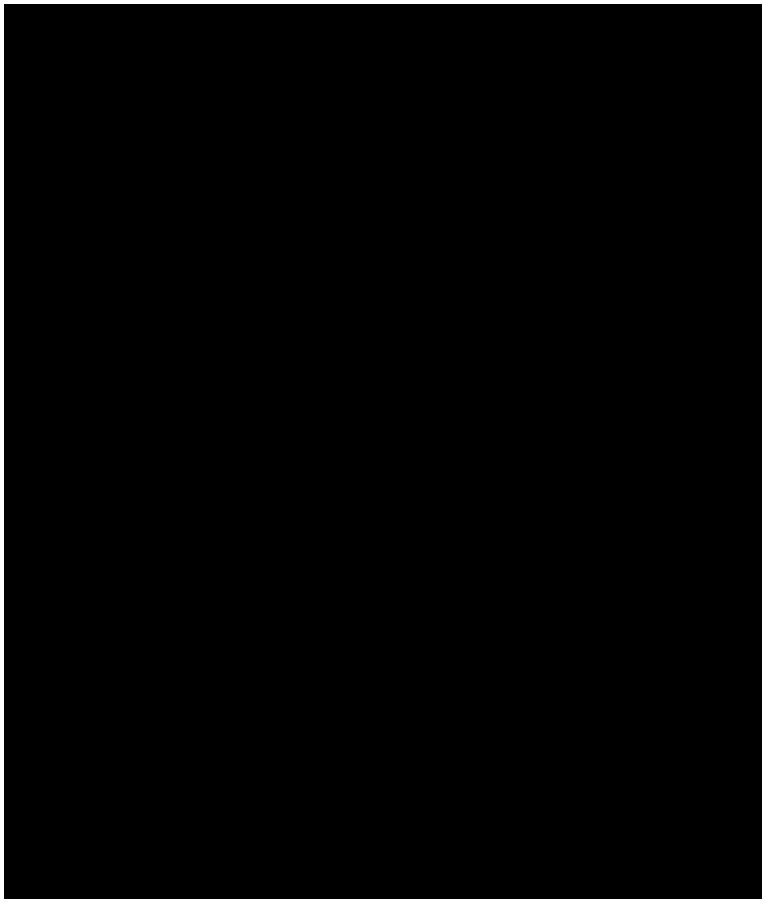
- Mongrel provides a high performance application server
- HTTP server written in C
- Lightweight container
- Can be clustered, about 70 MB per server
- Easy to install and test
- Joyent has scaled to 4,000 requests per second
  - Uses BigIP L7 load balancer
  - 48 Mongrels

Long Running Actions

- Use BackgroundDrb to handle long running processes
- Start a Drb daemon
- Send asynchronous requests to DrbDaemon
- Poll until finished from client

BackgroundDRB

- Rails application uses MiddleMan object as proxy for worker
- Server daemon starts sub processes to handle work requests
- Dynamic status reporting on progress



Routing

- Routing is a set a rules that determine what to do with a url
- A route is a very flexible pattern that can match various URL paths and parts
- Routes can also generate URLs for generating HTML links
- Routes now support REST

Simple Routes

- What's in the box

```
ActionController::Routing::Routes.draw do |map|
  # The priority is based upon order of creation: first created -> highest priority.

  # Sample of regular route:
  # map.connect 'products/:id', :controller => 'catalog', :action => 'view'
  # Keep in mind you can assign values other than :controller and :action

  # Sample of named route:
  # map.purchase 'products/:id/purchase', :controller => 'catalog', :action => 'purchase'
  # This route can be invoked with purchase_url(:id => product.id)

  # You can have the root of your site routed by hooking up ''
  # -- just remember to delete public/index.html.
  # map.connect '', :controller => "welcome"

  # Allow downloading Web Service WSDL as a file with an extension
  # instead of a file named 'wsdl'
  map.connect ':controller/service.wsdl', :action => 'wsdl'

  # Install the default route as the lowest priority.
  map.connect ':controller/:action/:id.:format'
  map.connect ':controller/:action/:id'
end
```

Advanced Routing

- The :<name> section of the route will match a portion of a path
  - It's possible to place pattern restrictions on any portion of the route
  - Any part of the route not specified can be defaulted

```
# This route matches any url in the following format:
#   http://www.xxx.com/account/balance/123041.html
map.connect ':controller/:action/:id.:format'

# If you leave off the format, you'll match this route:
#   http://www.xxx.com/account/balance/123041
map.connect ':controller/:action/:id'

# How about:
#   http://www.xxx.com/trx/123041/2007.01.01/2007.06.31
map.connect 'trx/:account/:start/:end',
            :controller => 'transactions',
            :action => 'list',
            :requirements => { :start => /(19|20)\d{2,2}\.\d{2,2}\.\d{2,2}/,
                               :end => /(19|20)\d{2,2}\.\d{2,2}\.\d{2,2}/,
                               :account /\d+/ }
```

URL Generation

- A route is bi-directional. A route that can be parsed can also be generated

```
url_for(:controller => 'account', :action => 'show', :id => 123141)
=> http://www.xxx.com/account/show/123141
```

- If we're rendering a controller action, it's even easier. The params are used to default the new URL

```
url_for(:action => 'show')
=> http://www.xxx.com/account/show/123141
```

Named Routes

- Named routes are short hand when generating URLs

```
…
  map.index 'account/', :controller => 'account', :action => 'index'
  map.trx 'trx/:account/:start/:end',
               :controller => 'transactions',
               :action => 'list',
               :requirements => { :start => /(19|20)\d{2,2}\.\d{2,2}\.\d{2,2}/,
                                  :end => /(19|20)\d{2,2}\.\d{2,2}\.\d{2,2}/,
                                  :account /\d+/ }
…

index_url
=> http://www.xxx.com/account/

trx_url(:account => account, :start_date => start_date, :end_date => end_date)
=> http://www.xxx.com/trx/123432/2007.01.01/2007.06.31

<%= link_to 'Index', index_url) %>
```

RESTful Routing

- Maps web resources to URLs

```
ActionController::Routing::Routes.draw do |map|
 map.resources :postings
end

GET http://www.xxx.com/postings
=> List of postings (index)

POST http://www.xxx.com/postings
=> Creates a new posting (create)

GET http://www.xxx.com/postings/1
=> Shows the posting with id 1 (show)

PUT http://www.xxx.com/postings/1
=> Updates posting with id 1 (update)

DELETE http://www.xxx.com/postings/1
=> Deletes posting with id 1 (delete)
```

Importance of REST

- Simple application integration
  - Lightweight web services
  - Use another web application as your model
- Simple
  - Provide a simple controller pattern
  - Automatic named routes
- Can be nested multiple layers deep
  - Traverse the object graph by nesting the resources
- Automatically responds with formatted results
  - use respond_to and provide html to browsers and XML to other applications

Questions