J2EE on Rails

Dwayne C. Litzenberger

PatientWay™ | Access Management Platform

PatientWay's Rails/J2EE Applications

- Current platform:
  - Ruby 1.8 and Rails 2.3
  - Rails 3 soon
- Developed the usual way
  - ./script/server
  - JRuby or ordinary ruby (MRI)
- Deployed to a J2EE container (.war file)
  - Glassfish on Windows or Linux
- One .war file per application
  - No site-specific builds
  - Auto-detects MySQL or MS-SQL

Overview

- rbenv and ruby-build
- rails myapp
- Bundler (Rails 2.3)
- Database configuration (for development)
- Automatic migrations
- Hard-coded secrets
- Database configuration (JNDI & JDBC in production)
- Glassfish (J2EE container)
- Build script
- Caveats

rbenv and ruby-build

rbenv and ruby-build

```
git clone https://github.com/sstephenson/rbenv ~/.rbenv
git clone https://github.com/sstephenson/ruby-build ~/.ruby-build

# Add to your shell's startup (~/.bashrc or ~/.bash_profile):
export PATH="$HOME/.rbenv/bin:$HOME/.ruby-build/bin:$PATH"
eval "$(rbenv init -)"
```

rbenv and ruby-build

```
rbenv-install jruby-1.6.5
rbenv-install 1.8.7-p352

alias rr='rbenv shell 1.8.7-p352'
alias rj='rbenv shell jruby-1.6.5'
```

Creating your Rails Application

Creating your Rails Application

```
rbenv shell jruby-1.6.5
gem install -v=2.3.14 rails
gem install bundler
rbenv rehash

rails myapp

cd myapp
echo jruby-1.6.5 > .rbenv-version
```

Command not found?

```
rbenv rehash

hash -r
```

.gitignore

```
/.bundle
/log
/tmp
/public/javascripts/cache_*
/public/stylesheets/cache_*
/db/*.sqlite3
/db/*_structure.sql
/db/schema.rb
/config/database.yml
/VERSION
/*.war
```

Bundler (Rails 2.3)

Copy boilerplate init code:

http://gembundler.com/rails23.html

$\Downarrow$

config/boot.rb
config/preinitializer.rb

config/boot.rb

```
# Insert the following code in config/boot.rb, right above the line `Rails.boot!`
class Rails::Boot
  def run
    load_initializer

    Rails::Initializer.class_eval do
      def load_gems
        @bundler_loaded ||= Bundler.require :default, Rails.env
      end
    end

    Rails::Initializer.run(:set_load_path)
  end
end
```

config/preinitializer.rb

```
begin
  require "rubygems"
  require "bundler"
rescue LoadError
  raise "Could not load the bundler gem. Install it with `gem install bundler`."
end

if Gem::Version.new(Bundler::VERSION) <= Gem::Version.new("0.9.24")
  raise RuntimeError, "Your bundler version is too old for Rails 2.3." +
   "Run `gem install bundler` to upgrade."
end

begin
  # Set up load paths for all bundled gems
  ENV["BUNDLE_GEMFILE"] = File.expand_path("../../Gemfile", __FILE__)
  Bundler.setup
rescue Bundler::GemNotFound
  raise RuntimeError, "Bundler couldn't find some gems." +
    "Did you run `bundle install`?"
end
```

Gemfile

```
source "http://rubygems.org"

gem "rails", "2.3.14"
gem "activerecord-jdbc-adapter", :platforms => :jruby

# We don't ship database drivers in the .WAR file; they are provided by the J2EE container.
group :development do
  platforms :jruby do
    gem "activerecord-jdbcsqlite3-adapter"
    gem "activerecord-jdbcmysql-adapter"
    gem "activerecord-jdbcmssql-adapter"
    gem "activerecord-jdbcderby-adapter"
    gem "warbler"
  end

  platforms :ruby do
    gem "mysql2", "~> 0.2.6"    # 0.3.x only works on Rails 3.1 and above
  end
end
```

Bundler

bundle install

database.yml

- Rename as config/database.sample.yml
- Add /config/database.yml to .gitignore
- Add to config/environment.rb:

```
# Fall back to config/database.sample.yml
unless File.exist?(config.database_configuration_file)
  config.database_configuration_file =
    File.join(config.root_path, "config", "database.sample.yml")
end
```

database.yml.sample

- Add "encoding: UTF8" to each configuration
  (or else you suffer!)
- We'll discuss using JNDI in production a bit later...

table_name_prefix

- In config/environment.rb:

```
# Allow multiple applications to use the same database.
config.active_record.table_name_prefix = 'myapp_'
```

- Needs a patch to ActiveRecord to fix some bugs:

```
mkdir rakelib
curl -o rakelib/rails_bug_1210_schemadumper_monkeypatch.rake \
  https://raw.github.com/gist/1503457
```

Automatic migrations

```
# config/initializers/zz0110_migrate_database.rb
# The Rails environment can be loaded for a bunch of reasons.
# Set $server_mode to true if it's really running as a server.
$server_mode = (
  $servlet_context or
  caller.any?{ |c| c =~ %r<commands/server|script/server> } or
  !ENV['FORCE_SERVER_MODE'].blank?)

if $server_mode
  # Run the migrations
  previous_version = ActiveRecord::Migrator.current_version
  ActiveRecord::Migrator.migrate("db/migrate/")
  current_version = ActiveRecord::Migrator.current_version

  # Load seeds
  load File.expand_path("db/seeds.rb", Rails.root) if previous_version == 0
end
```

Hard-coded secrets

```
# config/initializers/cookie_verification_secret.rb
ActionController::Base.cookie_verifier_secret =
'cc250bb681b03c5360f80e017a6f89adfc017a494f5a6ac6ab3ef5dd
89b69bfd869130ebb124e8e6040a8abc958bef30c27e46b9dfb339
160b39b7540748730f';

# config/initializers/session_store.rb
ActionController::Base.session = {
  :key          => '_myapp_session',
  :secret       =>
'dc0842657166597 5b5a00958482302af5030e98e2af73c8d3d890
571a6755899850e74ef90da25c5b31daae1611919d608de7372dfc
52241fa993058e84880f1'
}
```

Hard-coded secrets

```
bundle exec ./script/generate model app_secret name:string value:string
```

Hard-coded secrets

```
# app/models/app_secret.rb
class AppSecret < ActiveRecord::Base
  validates_presence_of :name, :value
  validates_uniqueness_of :name

  def self.secret(name)
    random = ActiveSupport::SecureRandom.hex(64)
    if $server_mode
      find_or_create_by_name(name, :value => random).value
    else
      random
    end
  end
end
```

Hard-coded secrets

- Rename the secret initializers so that they run after zz0110_migrate_database:

```
git mv config/initializers/cookie_verification_secret.rb \
  config/initializers/zz0120_cookie_verification_secret.rb

git mv config/initializers/session_store.rb \
  config/initializers/zz0120_session_store.rb
```

- Use AppSecret.secret instead of hard-coded secrets.

Hard-coded secrets

- config/initializers/zz0120_cookie_verification_secret.rb

```
ActionController::Base.cookie_verifier_secret =
  AppSecret.secret("cookie_verifier_secret")
```

- config/initializers/zz0120_session_store.rb

```
ActionController::Base.session = {
  :key         => '_myapp_session',
  :secret      => AppSecret.secret("session_secret")
}
```

Warbler and the J2EE environment

Warbler and the J2EE Environment

- Servlet deployed as a .war file
  - Warbler builds .war files
- All configuration in the database
- Database connection via JNDI
- Single multi-threaded instance of the JRuby runtime

warble works like rake

```
bundle exec warble -T
```

```
warble config     # Generate a configuration file to customize your archive
warble war        # Create the project war file
```

config/warble.rb

```
bundle exec warble config

vim config/warble.rb
```

config/warble.rb

- config.dirs
  - Remove "log" and "tmp"
  - Add "db"
- config.includes
  - Add "VERSION"
- config.excludes
  - Add "config/database.yml"
  - Add "db/*.sqlite3"
  - Add "db/schema.rb"

config/warble.rb

- Make sure only one instance of JRuby runs.

```
config.webxml.jruby.min.runtimes = 1
config.webxml.jruby.max.runtimes = 1
```

- JNDI data source name

```
config.webxml.jndi = 'jdbc/patientway-myapp'
```

config/environments/production.rb

```
# Enable threaded mode
config.threadsafe!

# Send logs to the J2EE container's log
config.logger = Logger.new(STDERR)
```

config/database.yml.sample

```
production:
  adapter: jdbc
  jndi: jdbc/patientway-myapp
  timeout: 5000
  pool: 32                # Glassfish's default maximum connection pool size
  wait_timeout: 30        # max seconds to wait while checking out a connection before raising a timeout error
  encoding: UTF8          # Without this, we depend on the JDBC connection pool to be configured for UTF-8.
```

config/environment.rb (optional stuff)

```
# Enable some (but not all) of the stuff enabled by config.threadsafe!,
# so that we catch related bugs in development and/or testing.
config.preload_frameworks = true
config.dependency_loading = false
config.eager_load_paths +=
  Dir.glob("vendor/plugins/*/app/{models,controllers,helpers,metal}")
```

Build the .war file!

```
bundle exec warble war
```

Our build.sh script

(run from a clean checkout)

```
#!/bin/bash
set -e    # Abort on errors

git describe --tags --always --dirty > VERSION

unset GEM_HOME GEM_PATH RBENV_VERSION
eval "$(rbenv init -)"

JRUBY="jruby -J-Xmx1024m -J-Djava.awt.headless=true"

$JRUBY -S bundle install
$JRUBY -S bundle exec rake db:test:purge db:migrate db:test:prepare test RAILS_ENV=test
$JRUBY -S bundle exec warble war
```

Before we deploy myapp.war

- We currently use Glassfish. YMMV.
- Create a JDBC connection pool
  - ServerName
  - DatabaseName
  - User
  - Password
- Create a JDBC resource (JNDI name) that points to it
  - jdbc/patientway-myapp

Deploy myapp.war !

Success!

http://localhost:8080/myapp/

Caveats

Caveats

- The usual Java pain:
  - No Unix filesystem semantics
    - Don't delete/rename a file while it's in use
  - No native code (although it might work sometimes)
  - No fork()
  - No cron
  - Slow startup time
  - Background threads must be stopped when the app is undeployed.

Caveats

- PermGen space leaked on each redeploy
  - -XX:MaxPermSize=384m helps somewhat
  - Might be a Glassfish-specific bug.
  - Just restart the container every few redeploys.
- Might use lots of heap
  - -Xmx1024m
- Might use lots of stack
  - -Xss512k

Managing background threads

- Gemfile:

```
gem "clean_thread", "~> 1.0"
```

- lib/blah_blah_thread.rb

```
class BlahBlahThread < CleanThread
  def main
    loop {
      check_finishing
      sleep 1.0
    }
  end
end
```

- config/initializers/zz0150_blah_blah_thread.rb

```
require 'blah_blah_thread'
if $server_mode
  $blah_blah_thread = BlahBlahThread.new
  at_exit { $blah_blah_thread.finish if $blah_blah_thread }
  $blah_blah_thread.start
end
```

Caveats

activerecord-jdbc-adapter

Caveats

- The Ruby on Rails community mostly does SaaS
  - e.g. hard-coded secrets
- Licensing
  - Check the licenses of your dependencies!
  - Occasionally, there are terms you don't want to comply with (e.g. GPL) or no terms at all
  - Explode the .war file and look at what you're actually shipping.

But, hey, at least you don't have to write Java code!

Thanks!

Dwayne C. Litzenberger

PatientWay™ | Access Management Platform