Deploying and Monitoring

Ruby on Rails

A practical guide

Mathias Meyer and Jonathan Weiss, 02.09.2008
Peritor GmbH
Who are we?

Jonathan Weiss

- Consultant for Peritor GmbH in Berlin
- Specialized in Rails, Scaling, Deployment, and Code Review
- Webistrano - Rails deployment tool
- FreeBSD Rubygems and Ruby on Rails maintainer

http://www.peritor.com
http://blog.innerewut.de
Who are we?

Mathias Meyer

- Independent Contractor
- Specialized in Rails, Performance/Database Tuning, Deployment, and Refactoring
- Macistrano – Webistrano-Client for Mac OS X (and soon the iPhone)

http://www.paperplanes.de
Agenda

Infrastructure | Deployment | Practical Session | Monitoring | Q & A
Infrastructure
Simple Rails Setup

One Rails instance handles all requests
Rails is single-threaded: There is only one concurrent request
Rails Setup

Client

App Server  App Server

App Server  App Server

App Server  App Server

DB
Rails Setup
Typical Rails Setup

• A load-balancer distributes the incoming requests
• Some load-balancers will deliver static requests themselves
• Several Rails instances handle all requests
• Number of concurrent requests equals number of Rails instances
Rails Setup Options
Deployment Questions

mod_proxy_balancer?
Mongrel?
Proxy?
Nginx?
HA-Proxy?
Ebb?
Swiftiply?
Thin?
Rails Application Server?
Lighttpd?
Apache?
Pound?
FastCGI?
Load-balancer?
mod_rails?
Reverse Proxy?
Phusion Passenger?
Pen?
What we are going to cover today

Rails Application Server
- FastCGI
- Mongrel
- mod_rails / Phussion Passenger
- JRuby + Glassfish & Co.

Proxy/Web Server
- Apache2
- Nginx
- Lighttpd
- HA-Proxy
FastCGI
FastCGI

- Protocol to communicate with long-running CGI applications
- Usage of either mod_fcgid with Apache 1.3 or mod_fcgid with Lighttpd
- Proxy local and remote FastCGI instances
- Oldest way of deploying Rails
- Deprecated and unstable
- Hard to debug (FastCGI protocol)
FastCGI

Client

HTTP

Apache 1.3

mod_fcg

FCGI

FCGI Rails
remote

FCGI Rails
remote

FCGI Rails
local
FastCGI

Client

HTTP

Apache 1.3
mod_fcg

FCGI

FCGI
Rails
remote

FCGI
Rails
remote

FCGI
Rails
local

or

Client

HTTP

Lighttpd
mod_fcg

FCGI

FCGI
Rails
remote

FCGI
Rails
remote

FCGI
Rails
local
Mongrel
Mongrel

- Developed by Zed Shaw as an alternative to FastCGI
- Complete HTTP-Server that can load arbitrary Ruby-servlets
- Built-in Rails support

$ sudo gem install mongrel
$ cd my/rails/project
$ mongrel_rails start -d -e production -p 80
Mongrel
Mongrel

Client

HTTP

HTTP Balancer

HTTP

Mongrel Rails
remote

Mongrel Rails
remote

Mongrel Rails
local

Apache 2.2 – mod_proxy_balancer
Lighttpd
Nginx
HA-Proxy
Pound
Mongrel Cluster

Utility to manage several Mongrel instances

$ sudo gem install mongrel_cluster

$ cat /srv/www/www.example.com/config/mongrel.conf
---
cwd: /srv/www/www.example.com/current
port: 8000
pid_file: /srv/www/www.example.com/shared/log/mongrel.pid
servers: 10
environment: production
Mongrel Cluster

Control Mongrels

$ mongrel_rails cluster::start -C /srv/www/www.example.com/config/mongrel.conf
$ mongrel_rails cluster::stop -C /srv/www/www.example.com/config/mongrel.conf
$ mongrel_rails cluster::restart -C /srv/www/www.example.com/config/mongrel.conf
Mongrel and Apache 2.2

Define Mongrel Cluster in Apache
Simple Mongrel and Apache 2.2

Redirect all traffic to the Mongel cluster

```xml
<VirtualHost 10.0.10.1:80>
    ServerAdmin webmaster@www.example.com
    ServerName www.example.com

    ErrorLog /var/log/apache2/www.example.com/error.log
    CustomLog /var/log/apache2/www.example.com/access.log combined

    ProxyPass / balancer://rails_cluster/
    ProxyPassReverse / balancer://rails_cluster/
</VirtualHost>
```
A more complex example

- Redirect dynamic requests
- Serve static content
- Support cached pages
- Support maintenance page
- Enable client-side caching of images, stylesheets, and JavaScript
- Compress output if supported
Mongrel

- Very robust
- Strict HTTP parser
- Easy to debug (HTTP!)
- Defacto standard deployment with Apache 2.2 and mod_proxy_balancer
- Can be a bit difficult to setup (mongrel_cluster, ports, Apache)
- Not so easy on mass/virtual hosting
mod_rails
mod_rails a.k.a Phusion Passenger

- Fairly new module for Apache 2.2
- Allows Apache to control Rails instances
- Apache starts and stops application instances depending on the application load
- Very easy to setup
- Able to run any RACK-compatible Ruby application (Merb & Co.)
- Only manages Rails on one host - no remote instances
- Combine with HTTP-Proxy / balancing solution
Install Phusion Passenger

Install Apache module

```
$ sudo gem install passenger
$ sudo passenger-install-apache2-module
```

Load and activate in Apache

```
LoadModule passenger_module /usr/local/libexec/apache22/mod_passenger.so
RailsSpawnServer /usr/local/bin/passenger-spawn-server
RailsRuby /usr/local/bin/ruby

<VirtualHost *:80>
  ServerName www.example.com
  DocumentRoot /srv/www/www.example.com/current/public
</VirtualHost>
```
Customized Phusion Passenger

Control Rails instance number

```bash
LoadModule passenger_module /usr/local/libexec/apache22/mod_passenger.so
RailsSpawnServer /usr/local/bin/passenger-spawn-server
RailsRuby /usr/local/bin/ruby

PassengerMaxPoolSize 10
PassengerMaxInstancesPerApp 5
PassengerPoolIdleTime 300

<VirtualHost *:80>
  ServerName www.example.com
  DocumentRoot /srv/www/www.example.com/current/public

  RailsEnv production
</VirtualHost>
```
Control Phusion Passenger

Restart after deployment:

touch /srv/www/www.example.com/current/tmp/restart.txt
Phusion Passenger

Client

HTTP

Apache2 + mod_rails

Rails

Rails

Rails

Rails

local

local

One machine
Phusion Passenger

One machine

Client

HTTP

Apache2 + mod_rails

Rails

Rails

Rails

Rails

local

local

Multiple machines

Client

HTTP

HTTP Balancer

Apache2 + mod_rails

Rails

Rails

Rails

Rails

local

local

local

local
Phusion Passenger

• Fairly new but ready for production
• Makes setup easier – on the single machine level
• Multiple servers still require load balancer
• Suitable for mass-hosting
• The upcoming standard way of deploying Rails
JRuby
JRuby

- Ruby Runtime on the Java Virtual Machine
- Implemented in Java and Ruby
- Compiles Ruby into Java-bytecode
- Integrates with Java code and libraries
- Java’s promises of native threads and JIT
- Allows for Ruby/Rails applications to be packaged as WAR files
- WAR files deployable on any J2EE-container: Glassfish, JBoss, Tomcat, Jetty, …
JRuby on Rails

Rails Application

+ 

JRuby

warble

WAR File

deploy

Glassfish J2EE container
JRuby on Glassfish

Client

HTTP

Glassfish J2EE container

JRuby  
local

JRuby  
local

One machine
JRuby on Glassfish

One machine

Multiple machines
Setup JRuby on Glassfish

1. Download JRuby and Glassfish

```bash
~/work ➜ java -Xmx256M -jar ~/Downloads/glassfish-installer-v2ur2-b04-darwin.jar
~/work ➜ cd glassfish
~/work/glassfish ➜ chmod a+x lib/ant/bin/*
~/work/glassfish ➜ lib/ant/bin/ant -f setup.xml
~/work/glassfish ➜ bin/asadmin start-domain
~/work/glassfish ➜ cd ~/work/testapp
~/work/testapp ➜ jruby -S gem install warbler
~/work/testapp ➜ jruby -S gem install activerecord-jdbcmysql-adapter
~/work/testapp ➜ vi config/database.yml
~/work/testapp ➜ warble
~/work/testapp ➜ ./glassfish/bin/asadmin deploy --contextroot / testapp.war
```
Warble Configuration

```ruby
Warbler::Config.new do |config|
  # Application directories to be included in the webapp.
  config.dirs = %w(app config lib log vendor tmp)

  # Additional files/directories to include, above those in config.dirs
  # config.includes = FileList["db"]

  # config.gems += ["activerecord-jdbcmysql-adapter", "jruby-openssl"]
  config.gems << "tzinfo"

  # Include gem dependencies not mentioned specifically
  config.gem_dependencies = true

  # Name of the war file (without the .war) -- defaults to the basename
  # of RAILS_ROOT
  config.war_name = "mywar"

  # Value of RAILS_ENV for the webapp -- default as shown below
  # config.webxml.rails.env = ENV['RAILS_ENV'] || 'production'

  # Control the pool of Rails runtimes. Leaving unspecified means
  # the pool will grow as needed to service requests. It is recommended
  # that you fix these values when running a production server!
  config.webxml.jruby.min.runtimes = 2
  config.webxml.jruby.max.runtimes = 4
end
```

Define min/max Rails runtimes
Rails Setup
Proxy Options
Proxy Requirements

- Hide cluster backend from the user
- Load-balancer backend instances
- Recognize down hosts
- Fair scheduler
  - (Deliver static content)
Apache 2.2

- Apache 2.2 introduced mod_proxy_balancer
- mod_proxy_balancer can speak to multiple backends and balance requests
- Apache can acts as a pure proxy or can also serve static files
Apache 2.2

- Apache 2.2 introduced mod_proxy_balancer
- mod_proxy_balancer can speak to multiple backends and balance requests
- Apache can acts as a pure proxy or can also serve static files

```xml
<Proxy balancer://rails_cluster>
    # local
    BalancerMember http://127.0.0.1:8000 loadfactor=1 max=1 acquire=1
    BalancerMember http://127.0.0.1:8001 loadfactor=1 max=1 acquire=1
    BalancerMember http://127.0.0.1:8002 loadfactor=1 max=1 acquire=1
    BalancerMember http://127.0.0.1:8003 loadfactor=1 max=1 acquire=1

    # remote 1
    BalancerMember http://10.0.1.10:8000 loadfactor=1 max=1 acquire=1
    BalancerMember http://10.0.1.10:8001 loadfactor=1 max=1 acquire=1
    BalancerMember http://10.0.1.10:8002 loadfactor=1 max=1 acquire=1
    BalancerMember http://10.0.1.10:8003 loadfactor=1 max=1 acquire=1

    # remote 2
    BalancerMember http://10.0.1.11:8000 loadfactor=1 max=1 acquire=1
    BalancerMember http://10.0.1.11:8001 loadfactor=1 max=1 acquire=1
    BalancerMember http://10.0.1.11:8002 loadfactor=1 max=1 acquire=1
    BalancerMember http://10.0.1.11:8003 loadfactor=1 max=1 acquire=1
</Proxy>

<VirtualHost 10.0.10.1:80>
    ServerAdmin webmaster@www.example.com
    ServerName www.example.com
    ErrorLog /var/log/apache2/www.example.com/error.log
    CustomLog /var/log/apache2/www.example.com/access.log combined
    ProxyPass / balancer://rails_cluster/
    ProxyPassReverse / balancer://rails_cluster/
</VirtualHost>
```
Apache 2.2

• Apache 2.2 introduced mod_proxy_balancer
• mod_proxy_balancer can speak to multiple backends and balance requests
• Apache can act as a pure proxy or can also serve static files

<Proxy balancer://rails_cluster>

# local
BalancerMember http://127.0.0.1:8000 loadfactor=1 max=1 acquire=1
BalancerMember http://127.0.0.1:8001 loadfactor=1 max=1 acquire=1
BalancerMember http://127.0.0.1:8002 loadfactor=1 max=1 acquire=1
BalancerMember http://127.0.0.1:8003 loadfactor=1 max=1 acquire=1

# remote 1
BalancerMember http://10.0.1.10:8000 loadfactor=1 max=1 acquire=1
BalancerMember http://10.0.1.10:8001 loadfactor=1 max=1 acquire=1
BalancerMember http://10.0.1.10:8002 loadfactor=1 max=1 acquire=1
BalancerMember http://10.0.1.10:8003 loadfactor=1 max=1 acquire=1

# remote 2
BalancerMember http://10.0.1.11:8000 loadfactor=1 max=1 acquire=1
BalancerMember http://10.0.1.11:8001 loadfactor=1 max=1 acquire=1
BalancerMember http://10.0.1.11:8002 loadfactor=1 max=1 acquire=1
BalancerMember http://10.0.1.11:8003 loadfactor=1 max=1 acquire=1

</Proxy>
Apache 2.2

Pro

• Stable, robust, and mature
• Many people know how to work with Apache
• Integrates well with other modules (SVN, DAV, Auth, …)

Con

• Apache can be complicated to configure
• The stock Apache is quite resource-hungry compared to pure proxy solutions
• Nginx - popular Russian webserver with good proxy support
• Can load-balance multiple backends and deliver static content
• Quite popular with Mongrel as the Rails backend
Nginx Configuration

Simple proxy example

Get complete version here:

http://brainspl.at/nginx.conf.txt
Nginx

Pro

- Stable, robust, and fast
- Uses fewer resources (CPU and RAM) than Apache for proxy-mode and static files
- Simpler configuration file
- Can directly talk to memcached - SSI

Con

- More documentation would be nice
- No equivalent for many Apache modules
Lighttpd

- Lightweight and fast webserver
- Balancing proxy support
- Good FastCGI support
- Used to be popular – until Mongrel came around
Lighttpd Configuration

Simple proxy example

```perl
server.modules = ( "mod_rewrite", "mod_redirect", "mod_access", "mod_accesslog", "mod_compress", "mod_proxy")

$HTTP.host? == "www.example.com" {

    server.document-root = "/my/path/to/app/public/current"

    proxy.balance = "fair"
    proxy.server = ( "/" => (
        ( "host" => "127.0.0.1", "port" => 8001 ),
        ( "host" => "127.0.0.1", "port" => 8002 ),
        ( "host" => "127.0.0.1", "port" => 8003 ),
        ( "host" => "127.0.0.1", "port" => 8004 )
    ))

}
```
**Lighttpd**

**Pro**
- Fast and lightweight
- Uses fewer resources (CPU and RAM) than Apache for proxy-mode and static files
- Simpler configuration file

**Con**
- Unstable for some people
- Slow development cycle
- More documentation would be nice
- Configuration file can be too simple (virtual host aliasing)
- No equivalent for many Apache modules
• HAProxy – reliable, high performance TCP/HTTP load balancer
• Proxying and content inspection
• No content serving, just a proxy
• Mature proxy module (fair scheduler)
• ACL support

See also similar Pound and Pen
HAProxy

Simple proxy example

```bash
listen app_a_proxy 0.0.0.0:80
# - equal weights on all servers
# - check health of app. server every 20 seconds
server a1 127.0.0.1:8000 weight 1 check inter 20000
server a1 127.0.0.1:8001 weight 1 check inter 20000
```
HAProxy

**Pro**
- Mature, stable, robust, and fast
- TCP and HTTP balancing

**Con**
- Few Rails examples
- Usually not needed in a Rails setup
Recommended Setups
Small Site

**Recommendation**

Apache 2.2 with mod_rails / Phusion Passenger
Medium Site

Recommendation
- Apache 2.2 as the frontend proxy
- Use Mongrel or mod_rails as the backend
- Deliver static files with Apache
Medium Site

Recommendation

- Apache 2.2 as the frontend proxy
- Use Mongrel or mod_rails as the backend
- Deliver static files with Apache
Large Rails Setup

Recommendation

- Redundant load-balancer
- Redundant proxy / web
- Mongrel / mod_rails
Heavy Static Files

Recommendation

• Deliver static assets through separate web server farm
• Mongrel or mod_rails
Java Shop

Recommendation

- Deliver a Rails-WAR file and you are done
- Integrate with existing Java landscape and infrastructure
Remarks
Ruby Enterprise Edition

- Copy-On-Write patches to Ruby 1.8
- Saves memory when spawning several Rails instances
- Used by Phusion Passenger if available
Thin, Ebb, Evented Mongrel & Co.

- Alternatives to Mongrel
- Claim to be faster, lighter, and what have you
- Rendering “Hello World” is usually not your bottleneck

Stick with stable and robust Mongrel
In The Future, Watch

Fuzed
- Erlang based load balancing
- Dynamic registration

JRuby
- Faster Runtime
- Java Integration

Phusion Passenger
- Enterprise Ruby
- Performance
- Usability
Deployment
Deployment Options

FTP
Deployment Options

SCP
Deployment Options
Deployment Options
What does Capistrano do?
Capistrano Deployment Cycle

- Check SCM
- Login
- Checkout
- Update current link
- Restart
Requirements

SSH!

Shell

SCM
What doesn’t Capistrano do?

• Plan your initial server setup
• Configure basic services
Basic Ingredients

- The cap command
- Variables
- Roles
- Tasks
- Namespaces
Basic Ingredients - cap

Your one-stop deployment shop

$ cap deploy
Basic Ingredients - Variables

- Configure basic project information
- Override Capistrano’s default assumptions
- Once set, variables are available globally
- Defined using the set method

```
set :user, "ruth"
set :home, "~/home/#{user}""
Basic Ingredients - Roles

• Define types of servers
• Default roles
  • :www
  • :app
  • :db
• All can point to the same server
• But all three must be defined

```ruby
role :app, "app1.example.com"
role :www, "web1.example.com"
role :db, "db1.example.com", :primary => true
```

• At least one database server needs to be primary
Basic Ingredients - Roles

Define custom roles as you please

```
role :solr, "search.example.com"
role :activemessaging, "poller.example.com"
```

Can be reused when defining tasks
Basic Ingredients - Tasks

- Define an atomic set of actions for Capistrano
- Can be called from the command line
- Or other tasks
Basic Ingredients - Tasks

To find all the tasks available in your project, use

```bash
$ cap -T
cap deploy # Deploys your project.
cap deploy:check # Test deployment dependencies.
cap deploy:cleanup # Clean up old releases.
cap deploy:cold # Deploys and starts a 'cold' application.
cap deploy:migrate # Run the migrate rake task.
cap deploy:migrations # Deploy and run pending migrations.
cap deploy:pending # Displays the commits since your last deploy.
cap deploy:pending:diff # Displays the 'diff' since your last deploy.
cap deploy:restart # Restarts your application.
cap deploy:rollback # Rolls back to a previous version and restarts.
cap deploy:rollback_code # Rolls back to the previously deployed version.
cap deploy:setup # Prepares one or more servers for deployment.
```
Basic Ingredients - Namespaces

Group tasks together logically

```
cap deploy:restart
cap solr:reindex
```

Namespaces and tasks are separated with “:”
Get Your Capistrano On

$ gem install capistrano
$ cd ~/development/rails_petstore/
$ capify .
Get Your Capistrano On

Capfile, the place to include more recipes

```ruby
load 'deploy' if respond_to?(:namespace) # cap2 differentiator
Dir['vendor/plugins/*/recipes/*.rb'].each { |plugin| load(plugin) }
load 'config/deploy'
```
Get Your Capistrano On

config/deploy.rb, application specific configuration

```
set :application, "set your application name here"
set :repository,  "set your repository location here"

# If you aren't deploying to /u/apps/#{application} on the target
# servers (which is the default), you can specify the actual location
# via the :deploy_to variable:
# set :deploy_to, "/var/www/#{application}"

# If you aren't using Subversion to manage your source code, specify
# your SCM below:
# set :scm, :subversion

role :app,  "your app-server here"
role :web,  "your web-server here"
role :db,   "your db-server here", :primary => true
```
Capistrano’s Defaults

- Your SCM is Subversion
- Deployment directory is /u/apps/#{application_name}
- User for SCM and SSH is the currently logged-in user
- Commands are run with sudo
Get Your Capistrano On

```ruby
set :application, "rails_petstore"
set :repository, "git://github.com/mattmatt/rails_petstore.git"
set :deploy_to, "'/var/www/#{application}'
set :use_sudo, false

set :scm, :git

role :app, "rails.petstore.com"
role :web, "www.petstore.com"
role :db, "db.petstore.com", :primary => true
```
Get Your Capistrano On

- Capistrano expects a directory structure
- Can be created with cap deploy:setup
The Deployment Lifecycle
The Deployment Lifecycle

Check the prerequisites:

```
$ cap deploy:check
  * executing 'deploy:check'
  * executing "test -d /home/cftuser/simplelog/releases"
    servers: ["192.168.1.55"]
  Password:
    [192.168.1.55] executing command
    command finished
  * executing "test -w /home/cftuser/simplelog"
    servers: ["192.168.1.55"]
    [192.168.1.55] executing command
    command finished
  * executing "test -w /home/cftuser/simplelog/releases"
    servers: ["192.168.1.55"]
    [192.168.1.55] executing command
    command finished
  * executing "which git"
    servers: ["192.168.1.55"]
    [192.168.1.55] executing command
    command finished
  You appear to have all necessary dependencies installed
```
Set up your application for the first time

```
$ cap deploy:setup
* executing 'deploy:setup'
* executing "umask 02 && mkdir -p /home/ruth/rails_petstore /home/ruth/rails_petstore/releases
 /home/ruth/rails_petstore/shared /home/ruth/rails_petstore/shared/system
 /home/ruth/rails_petstore/shared/log /home/ruth/rails_petstore/shared/pids"

servers: ["192.168.1.55"]
Password: [192.168.1.55] executing command
command finished
```
The Deployment Lifecycle

The initial deployment

1. Checks the revision from the local machine
2. Checks out the code on the remote machines
3. Sets a link called current pointing to the latest release
4. Runs the migrations
5. Fires up application servers

$ cap deploy:cold
The Deployment Lifecycle
The Deployment Lifecycle

Subsequent deployments

```bash
$ cap deploy
```

1. Checks the revision from the local machine
2. Checks out the code on the remote machines
3. Updates current link
4. Restarts application servers
The Deployment Lifecycle

rails_petstore

shared

releases

current

20080825095945

20080825120336

20080827142052
Common Capistrano Tasks

Deploy and run migrations

$ cap deploy:migrate

Run only the migrations

$ cap deploy:migrations

Restart application servers

$ cap deploy:restart

Rollback to the previous release

$ cap deploy:rollback
Have Your Shell, and Eat It Too

```bash
$ cap shell
* executing 'shell'

Welcome to the interactive Capistrano shell! This is an experimental feature, and is liable to change in future releases. Type 'help' for a summary of how to use the shell.

cap> ls
[establishing connection(s) to 192.168.2.106]
Password:
** [out :: 192.168.2.106] rails_petstore
** [out :: 192.168.2.106] simplelog
cap>
```
Invoking any Command

```bash
$ cap invoke COMMAND="ls -l"
* executing 'invoke'
* executing "ls -l"
  servers: ["192.168.2.106"]
Password:
  [192.168.2.106] executing command
** [out :: 192.168.2.106] total 2
** [out :: 192.168.2.106] drwxrwxr-x 4 cftuser cftuser 1024 Aug 24 23:54 rails_petstore
** [out :: 192.168.2.106] drwxrwxr-x 4 cftuser cftuser 1024 Aug 23 22:15 simplelog
command finished
```
Deployment Strategies

![No FTP symbol]
Deployment Strategies

Direct checkout (from scratch) on the servers

```ruby
set :deploy_via, :checkout
set :deploy_via, :export
```
Deployment Strategies

Keep a cached copy of the current SCM head

```ruby
set :deploy_via, :remote_cache
set :repository_cache, "/var/www/checkout/rails_petstore"
```
Deployment Strategies

Check out locally and transfer

```ruby
set :deploy_via, :copy
set :copy_strategy, :checkout
set :copy_remote_dir, "~/var/tmp/rails_petstore"
set :copy_dir, "~/tmp/rails_petstore"
```
Give it a little Spin

Capistrano expects a script called spin in script/process

```
#/bin/sh
/var/www/rails_petstore/script/process/spawner -p 11000 -i 3
```

For Passenger

```
#/bin/sh
touch /var/www/rails_petstore/tmp/restore.txt
```
Customizing Capistrano
Write your own Tasks

task :gem_list do
  run "gem list"
end
Write your own Tasks

desc "Starts the poller service"
task :start, :roles => :poller, :except => { :no_release => true } do
  "long_running", "short_running"[].each do |group|
    invoke_command "cd #{deploy_to}/current &&
      DISABLE_MASOCHISM=1 TARGET_ENV='#{site_target}'
      RAILS_ENV=#{rails_env}
      ruby script/poller start -- process-group=#{group} ; echo '0' ",
    :via => run_method
  end
end
Namespace your Tasks

```ruby
namespace :poller do
  desc "Starts the poller service"
  task :start, :roles => :poller, :except => { :no_release => true } do
    ...
  end

  desc " Stops the poller service"
  task :stop, :roles => :poller, :except => { :no_release => true } do
    ...
  end

  desc "Restarts the poller service"
  task :restart, :roles => :poller, :except => { :no_release => true } do
    end
end
```
Callbacks

Execute a task before another runs

```plaintext
before "deploy:restart", "deploy:web:disable"
```

Execute a task after another has finished

```plaintext
after "deploy:update_code", "poller:restart"
```

Callbacks are run in the order they’re defined
Useful Variables

current_path: /var/www/rails_petstore/current
release_path: /var/www/rails_petstore/releases/20080829104344
shared_path: /var/www/rails_petstore/shared
current_release: /var/www/rails_petstore/releases/20080829082827
previous_release: /var/www/rails_petstore/releases/20080828194529
Transactions

task :reindex do
  transaction do
    store_existing_index
    create_new_index
    replace_index
  end
end
Transactions

```ruby
# task: store_existing_index do
  # copy index
end

# task: create_new_index do
  # on rollback {
    # remove new index
  }
  # indexing...
end

# task: replace_index do
  # on rollback {
    # restore old index
  }
  # moving around indexes...
end
```
Transactions

task :replace_index do
  on_rollback {
    # restore old index
  }
  # moving around indexes...
end
Transactions

```ruby
task :reindex do
  transaction do
    store_existing_index
    create_new_index
    replace_index
  end
end
```

```ruby
task :store_existing_index do
  # copy index
end
```

```ruby
task :create_new_index do
  on_rollback {
    # remove new index
  }
  # indexing...
end
```

```ruby
task :replace_index do
  on_rollback {
    # restore old index
  }
  # moving around indexes...
end
```
The Rest

- Gem dependencies
- Support for deploying through gateway servers
- Server setup with deprec gem
- Lack of documentation
One Click Deploy
Webistrano

Deployment of stage: stable (of project Company Website)

Task: deploy:restart
Stage: stable of Project Company Website
Deployed by: jweis
Started (UTC): 2007-09-26 12:51
Completed (UTC): 2007-09-26 12:51

Comment
Update products and navigation

Log
** loading stage recipe '81'
** loading stage recipe '8e-22'
  * executing 'deploy:restart'

Status
successful

Projects
- Blog
- Company Website
- stable
- test
- HR-Management
- Rails Sample Project
- Time Tracking

Hosts
127.0.0.1
blog.innenwut.de
ec2-67-202-14-39.r-1.compv
localhost
webistrano.org
www.peritor.com

Recipes
- 2 recipes

Users
- 2 users

Back
Webistrano

- Web-UI to Capistrano
- Manages projects and their stages
- Alerting and Accounting
- Scriptable and extendable
- BSD License

http://labs.peritor.com/webistrano
Macistrano

- Mac-GUI to Webistrano
- Fire and monitor deployments from your desktop

http://github.com/mattmatt/macistrano
Pratical Session
Practical Capistrano
Get the slides

- Connect to “RailsConf Deployment” WLAN
- The slides are at http://10.0.0.1/
- The VMs are at 10.0.0.3 – 10.0.0.41
- Sample configurations files are in the GIT/SVN repositories
Monitoring
The two questions of monitoring
1. Is everything still running?
2. What are the trends?
Monit

- Process-level monitoring
- Checks PID-files, ports, and permissions
- Reacts by executing a script and/or alerting
Monit

MySQL

```bash
# mysql
check process mysql with pidfile /var/db/mysql/master.peritor.com.pid
   group database
   start program = "/usr/local/etc/rc.d/mysql-server start"
   stop program = "/usr/local/etc/rc.d/mysql-server stop"
   if failed host localhost port 3306 protocol mysql then restart
   if 5 restarts within 5 cycles then timeout
```
Monit

Apache

```bash
# apache
check process apache with pidfile /usr/local/apache/logs/httpd.pid
  start program = "/etc/init.d/httpd start"
  stop program = "/etc/init.d/httpd stop"
  if cpu > 60% for 2 cycles then alert
  if cpu > 80% for 5 cycles then restart
  if totalmem > 200.0 MB for 5 cycles then restart
  if children > 250 then restart
  if loadavg(5min) greater than 10 for 8 cycles then stop
  if failed host www.tildeslash.com port 80 protocol http
      and request "/monit/doc/next.php"
      then restart
  if failed port 443 type tcpssl protocol http
      with timeout 15 seconds
      then restart
  if 3 restarts within 5 cycles then timeout
depends on apache_bin
group server
```
Monit

Mongrel

```bash
# mongrel
check process mongrel-8000 with pidfile /home/user/rails/current/log/mongrel.8010.pid
   start program = "/usr/bin/mongrel_rails cluster::start -C /var/www/app/conf/mongrel.conf --only 8000"
   stop program = "/usr/bin/mongrel_rails cluster::stop -C /var/www/app/conf/mongrel.conf --only 8000"

if totalmem is greater than 60.0 MB for 5 cycles then restart # eating up memory?
if cpu is greater than 50% for 2 cycles then alert # send an email to admin
if cpu is greater than 80% for 3 cycles then restart # hung process?
if loadavg(Smin) greater than 10 for 8 cycles then restart # bad, bad, bad
if 3 restarts within 5 cycles then timeout # something is wrong, call the sys-admin
if failed port 8000 protocol http
   with timeout 10 seconds
   then restart
   group mongrel # check for response
```
Munin

- Host-level monitoring
- Master periodically asks nodes for local data
- Checks system resources and records historical data
- Allows to recognize trends and make predictions
- Alerting support
Munin

Memory usage - by month

<table>
<thead>
<tr>
<th>Category</th>
<th>Current</th>
<th>Minimum</th>
<th>Average</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>apps</td>
<td>5.84G</td>
<td>1.31G</td>
<td>3.10G</td>
<td>6.38G</td>
</tr>
<tr>
<td>swap_cache</td>
<td>13.48M</td>
<td>12.56M</td>
<td>1.07G</td>
<td>1.43G</td>
</tr>
<tr>
<td>vmalloc_used</td>
<td>8.45M</td>
<td>8.30M</td>
<td>8.45M</td>
<td>8.45M</td>
</tr>
<tr>
<td>slab_cache</td>
<td>329.80M</td>
<td>62.75M</td>
<td>492.49M</td>
<td>2.16G</td>
</tr>
<tr>
<td>cache</td>
<td>1.08G</td>
<td>312.17M</td>
<td>4.27G</td>
<td>6.32G</td>
</tr>
<tr>
<td>buffers</td>
<td>159.35M</td>
<td>4.81M</td>
<td>293.69M</td>
<td>3.11G</td>
</tr>
<tr>
<td>unused</td>
<td>379.84M</td>
<td>26.90M</td>
<td>595.32M</td>
<td>3.31G</td>
</tr>
<tr>
<td>swap</td>
<td>31.18M</td>
<td>31.18M</td>
<td>1.23G</td>
<td>1.82G</td>
</tr>
<tr>
<td>committed</td>
<td>5.95G</td>
<td>3.30G</td>
<td>4.98G</td>
<td>9.49G</td>
</tr>
<tr>
<td>mapped</td>
<td>19.12M</td>
<td>10.75M</td>
<td>17.18M</td>
<td>389.27M</td>
</tr>
<tr>
<td>active</td>
<td>6.35G</td>
<td>1.54G</td>
<td>4.89G</td>
<td>7.14G</td>
</tr>
<tr>
<td>inactive</td>
<td>728.02M</td>
<td>178.63M</td>
<td>1.81G</td>
<td>6.10G</td>
</tr>
</tbody>
</table>

Last update: Fri Aug 29 16:45:21 2008
Munin

MySQL queries - by week

- **select**: Cur: 69.77  Min: 0.00  Avg: 86.63  Max: 417.15
- **delete**: Cur: 46.50m  Min: 0.00  Avg: 542.24m  Max: 25.00
- **update**: Cur: 4.44  Min: 0.00  Avg: 11.26  Max: 103.16
- **insert**: Cur: 5.82  Min: 0.00  Avg: 6.80  Max: 55.18
- **cache_hits**: Cur: 60.94  Min: 0.00  Avg: 56.45  Max: 565.85
- **replace**: Cur: 0.00  Min: 0.00  Avg: 0.00  Max: 0.00
- **total**: Cur: 0.00  Min: 0.00  Avg: 161.25  Max: 875.11

Munin

Filesystem usage (in %) - by year

Other Tools

- Nagios
- Big Brother
- New Relic RPM
- FiveRuns
- JMX
Q & A
Peritor GmbH
Teutonenstraße 16
14129 Berlin
Telefon: +49 (0)30 69 20 09 84 0
Telefax: +49 (0)30 69 20 09 84 9
Internet: www.peritor.com
E-Mail: kontakt@peritor.com

©Peritor GmbH - Alle Rechte vorbehalten