

Using Capistrano for Automated Deployments

ENDAX MANUALS

Content is freely available for non commercial, non Profit, educational purposes provided the content is not modified and the author / publisher are given full credit for their work. Permission to reproduce for any other use in any format is strictly forbidden without the copyright holders permission.

©2008 ENDAX

Capistrano

Capistrano is a very useful and easy to setup deployment tool for Rails. There no legitimate reason why you shouldn't be using it regardless of the size of your project.

Also keep in mind that Capistrano is very similar to Rake in that you can create your own tasks, write in Ruby, etc.

One final note is that Capistrano requires the use of SSH which should already be installed in your distribution.

Installing

Installing it is simply:

```
gem install -y capistrano
```

Once installed, and assuming this is the first use of Capistrano in your project you will need to create your Capfile in the root directory and create a second deploy.rb script in your config directory. This is accomplished with:

```
cd /project/root  
capify .
```

Deployment Configuration

Before getting started, there are a couple requirements that we have to meet. First, Capistrano requires the bash shell in order to run properly. Depending on your distribution, new users may or may not get assigned the bash shell when logging in. Lets go ahead and create a new user called deployadm:

```
useradd -m deployadm
```

and set a password:

```
passwd deployadm
```

Now ensure that the default shell is bash by editing the passwd file:

```
vi /etc/passwd
```

Change the following line (likely at the bottom of the file):

```
deployadm:x:1003:1001::/home/deployadm:/bin/sh
```

to

```
deployadm:x:1003:1001::/home/deployadm:/bin/bash
```

And really a requirement of this configuration, we should add the `deployadm` to the `/etc/sudoers` file so that user can restart the stack:

```
visudo
```

Now add the following to the bottom of the file:

```
deployadm ALL=(root) /etc/init.d/mongrel_cluster,  
/etc/init.d/nginx
```

This will allow the `deployadm` to only start these scripts. Now you are ready to edit the `config/deploy.rb` file:

```
#  
# Capistrano 2.0 Configuration  
  
set :application, "myapp"  
set :repository, "svn://you@someurl/repo/trunk"  
set :keep_releases, 3  
set :checkout, "export"  
  
# Optionals – path on the server you are deploying  
# to, and the SSH username  
set :deploy_to, "/var/www/#{application}"  
set :user, "you"  
  
# Roles – someurl.com is the server you are  
# deploying to  
role :app, "someurl.com"  
role :web, "someurl.com"  
role :db, "someurl.com", :primary => true  
  
# Persist Configuration by using symbolic links  
task :after_update_code,  
  :roles => :app,  
  :except => {:no_symlink => true} do  
  run <<-CMD
```

```

    cd #{release_path} &&
    ln -nfs #{shared_path}/config/database.yml
        #{release_path}/config/database.yml &&
    ln -nfs #{shared_path}/config/mongrel_cluster.yml
        #{release_path}/config/mongrel_cluster.yml
  CMD
end

```

Like Rake it is also possible to create your own custom tasks, in fact it is necessary when using `mongrel_cluster` (please see the note `Mongrel Cluster Revisited` in the following section on how to create the `mongrel_cluster` startup script and path configuration) and Nginx (or Apache2). Add this to the bottom of your `deploy.rb` file:

```

# Custom start / stop / restart
# need to override the default deploy:restart, etc.
namespace :deploy do

  # Mongrel and Capistrano 2
  namespace :mongrel do
    [ :stop, :start, :restart ].each do |t|
      desc "#{t.to_s.capitalize} Mongrel.."
      task t, :roles => :app do
        sudo "/etc/init.d/mongrel_cluster #{t.to_s}"
      end
    end
  end

  # Nginx and Capistrano 2
  namespace :nginx do
    [ :stop, :start, :restart ].each do |t|
      desc "#{t.to_s.capitalize} Nginx.."
      task t, :roles => :app do
        sudo "/etc/init.d/nginx #{t.to_s}"
      end
    end
  end

  desc "Custom restart task"
  task :restart,
    :roles => :app,
    :except => { :no_release => true } do
    deploy.mongrel.restart
    deploy.nginx.restart
  end

  desc "Custom start task"
  task :start, :roles => :app do
    deploy.mongrel.start
    deploy.nginx.start
  end
end

```

```

desc "Custom stop task"
task :stop, :roles => :app do
  deploy.mongrel.stop
  deploy.nginx.stop
end

end

```

This script is specific to Nginx, but changing to Apache is trivial; simply change the references to `nginx` to `apache2`.

Capistrano works by deploying your code based to a `releases` directory, one for each deployment and then uses a symbolic link to point to the current release.

It's worth noting that Capistrano also has a number of tasks build in [like Rake] for Rails, listed as follows:

<code>cap deploy</code>	Deploys your project.
<code>cap deploy:check</code>	Test deployment dependencies.
<code>cap deploy:cleanup</code>	Clean up old releases.
<code>cap deploy:cold</code>	Deploys and starts a 'cold' application.
<code>cap deploy:migrate</code>	Run the migrate rake task.
<code>cap deploy:migrations</code>	Deploy and run pending migrations.
<code>cap deploy:pending</code>	Displays the commits since your last deploy.
<code>cap deploy:pending:diff</code>	Displays the 'diff' since your last deploy.
<code>cap deploy:restart</code>	Restarts your application.
<code>cap deploy:rollback</code>	Rolls back to a previous version and restarts.
<code>cap deploy:rollback_code</code>	Rolls back to the previously deployed version.
<code>cap deploy:setup</code>	Prepares one or more servers for deployment.
<code>cap deploy:start</code>	Start the application servers.
<code>cap deploy:stop</code>	Stop the application servers.
<code>cap deploy:symlink</code>	Updates the symlink to the most recently deployed.
<code>cap deploy:update</code>	Copies your project and updates the symlink.

cap deploy:update_code	Copies your project to the remote servers.
cap deploy:upload	Copy files to the currently deployed version.
cap deploy:web:disable	Present a maintenance page to visitors.
cap deploy:web:enable	Makes the application web-accessible again.
cap invoke	Invoke a single command on the remote servers.
cap shell	Begin an interactive Capistrano session.

Additional information about a particular task can be read by using the `-e` option:

```
cap -e TASK
```

One final note about passwords is that Capistrano assumes all of your passwords are the same, both the machine username and password and the svn username and password (technically they can be set in the script, but that is very dangerous since even ssh+svn transfers are unsecure). This of course maybe a problem and the preferred method of authentication is using public keys.

Briefly, this can be accomplished by creating a private/public key on your machine:

```
cd ~
ssh-keygen -t dsa
```

Ensure that your `.ssh` directory is secure:

```
chmod 600 ~/.ssh
```

Then secure copy the public key to the server you will deploy to:

```
scp ~/id_dsa.pub you@deploymentserver.com:~
```

Log into that server and append to the know `authorized_keys` file:

```
ssh you@deploymentserver.com
cat id_dsa.pub >> .ssh/authorized_keys
exit
```

And lastly delete the public key on your local server:

```
rm id_dsa.pub
```

Using

Now you are ready to get started on your application. The following will log into your server as create a number of directories:

```
cd /your/project/dir
cap deploy:setup
```

The following directories are created as a result:

```
#{deploy_to}/
#{deploy_to}/releases
#{deploy_to}/shared
#{deploy_to}/shared/log
#{deploy_to}/shared/system
#{deploy_to}/shared/pids
```

Where `#{deploy_to}` is which PATH you specified in your `deploy.rb` file.

In the above deployment file, I've included an `after_update_code` task that copies configuration files from a config directory. Simple create this directory in your `#{deploy_to}/shared` path called `config` and copy your configuration files to this directory. When you deploy your application these files are kept in place and the symbolic links are updated, i.e. to avoid having to update your configuration every time you deploy. This of course could be included in the `deploy:setup` by writing a custom extension of it.

Now that you're ready to deploy for the first time, be sure you use the cold deploy method. This simply starts your server for the first time:

```
cap deploy:cold
```

Each subsequent deployment use:

```
cap deploy:migrations
```

If something goes terribly wrong with your application (not the deployment, this will roll back on its own if there is an error), rollback to

the previous version by using:

```
cap deploy:rollback
```

Fix the problem and try again.

Mongrel Cluster Revisited

As suggested earlier, at this point we will return to configuration of Mongrel Cluster in regard to Capistrano deployments.

Create a directory in:

```
mkdir /etc/mongrel_cluster
```

Make sure that you have copied the `mongrel_cluster.yml` file over to the `shared/config` directory as suggested above. Once there create a symbolic link to it:

```
ln -s  
/var/www/myapp/shared/config/mongrel_cluster.yml \  
/etc/mongrel_cluster/myapp.yml
```

Also lets make sure that the script is available in the `init.d` directory:

```
cp \  
/var/lib/gems/1.8/gems/mongrel_cluster-  
1.0.5/resources/mongrel_cluster \  
/etc/init.d/
```

Make it executable:

```
chmod +x /etc/init.d/mongrel_cluster
```

And finally, install as a System 5 startup script:

```
update-rc.d -f mongrel_cluster defaults
```