# 用Capistrano 3自动化部署Web应用

如果你部署过Web应用，你应该知道，很多时候并不是把源代码/war包放上服务器就了事的。你得修改配置，修改数据库结构，重启服务器，重启后台进程……人是很容易犯错的，步骤一多，就会记不住。如果你部署的是大型Web应用，几百台服务器各自负责不同的工作，那部署一次就能让你哭出来。有什么办法解决这个问题？自动化部署！计算机能够勤勤恳恳孜孜不倦地重复劳动，那为什么我们不把部署的任务交给它呢？于是乎，今天我就来讲讲怎么用Capistrano 3来实现自动化部署。

1. 先决条件
2. Ruby >= 1.9（Capistrano的运行环境，并不要求你的Web项目是Ruby的）
3. SSH Server（服务器端），采用公钥验证
4. SSH Client（开发机端）
5. SCM（Source Controll Management，版本控制，推荐使用Git）
6. 如果使用Git，建议申请一个git.oschina.net或github.com的帐号
7. 安装Capistrano 3并将项目纳入Capistrano管理

打开控制台（UNIX/Linux的Terminal、Windows用户可使用Git Bash），输入：

$ gem install -v ‘~> 3.3’ capistrano

$ cd $project\_root

$ cap install

其中$project\_root是你的项目的根。

注：如果你从来没用过UNIX/Linux，每行开头的那个$是不用你打的。它是命令行提示符，相当于Windows里的C:\>。中间的$xxx是变量，类似于Windows里的%xxx%。

最后一条命令会帮你在项目的根下创建下述文件/目录结构：

$project\_root/

|

|--Capfile

|

|--config/

| |

| |--deploy.rb

| |

| |--deploy/

| |

| |--production.rb

| |--staging.rb

|

|--lib/

|

|--capistrano/

|

|--tasks/

Capfile是一个胶水文件，用来导入各种第三方库。这些第三方库往往已经定义好了各种任务，如数据库更新/回滚、远程控制台、等等等等。很多库也把任务整合进了部署/回滚流程（见下节）。

config/deploy.rb可以看成是主配置文件，所有在部署/回滚中需要自动执行的自定义任务都可以挂在这里。虽然它其实是Ruby脚本，但是它提供了非常简洁的DSL，使得不懂Ruby的程序员也可以看得懂、写得出部署脚本。

config/deploy/\*.rb是针对每个不同环境（见下节）的配置，这里主要配置一些服务器端的IP、登录用户名之类的东西。由于这些文件会被提交至SCM，所以非常不建议把敏感信息（如密码、SSL私钥等）写在里面。敏感写在哪里下面会说。也正是由于这样的理由，建议SSH使用公钥验证，而非用户名+密码的验证方式。

lib/capistrano/tasks文件夹给你存放自定义任务的脚本。这些脚本必须以.rake结尾。

1. 快速上手

我们来看看如何以最少的配置、最快的速度部署项目。

1. 进入项目根目录

$ cd /path/to/my\_app

2. 把项目加入SCM（如果已经做过了，这一步可以省略。这里用Git，你也可以用SVN）

$ git init

$ git add --all .

$ git commit -m ‘Initial commit’

$ git remote add origin <remote repo url>

$ git push origin master

1. 把项目Capistrano化（如果已经做过了，直接跳过）

$ cap install

1. 修改config/deploy.rb里的这几行：

set :application, 'my\_app' # my\_app是你的项目名称，这决定部署的目录

set :scm, :git # 如果你用SVN，把:git改成:svn就行

set :repo\_url, '<remote repo url>' # 你的remote仓库的url

set :deploy\_to, '/var/www/my\_app' # 默认部署到 /var/www/my\_app目录

1. 修改config/deploy/staging.rb：

注释掉下面这几行：

# role :app, [%w{admin@example.com}](mailto:%w{admin@example.com})

# role :web, [%w{admin@example.com}](mailto:%w{admin@example.com})

# role :db, [%w{admin@example.com}](mailto:%w{admin@example.com})

根据实际服务器的IP/域名和用户名修改下面这一行：

server 'example.com', user: 'admin', roles: %w{web app db}, my\_variable: :my\_value

1. 部署

$ cap staging deploy

这一步可能会出现一些权限问题，请确保服务器端的部署目录的上级目录存在，并且你配置的用户对该目录有完全权限，并且它的上级、上上级……目录至少具有读和执行权限。通常/var/www是没有写权限的，你可以把它的权限改成drwxrwsrwt（3777）。改完权限后再次尝试这一步即可。

1. 远端的文件夹结构

用Capistrano部署后，在服务器端会生成这样一个文件夹结构：

$deploy\_path/

|

|--revisions.log

|

|--releases/

| |--xxxxxxxxx/

| |--yyyyyyyyy/

|

|--shared/

|

|--repo/

|

|--current -> releases/xxxxxxxxx/

简要介绍一下几个文件/文件夹的作用:

1. releases

这个文件夹保存了你部署上去的历史版本。每个版本的文件夹的名称都是一个时间戳。

1. shared

这个文件夹保存了所有各版本共享的文件/文件夹，尤其是linked\_files、linked\_dirs指定的文件/文件夹（见后）

1. repo

这是一个git仓库（如果你用git作为SCM的话）

1. current

这是一个symlink（Windows用户可把它想象成快捷方式）,它指向当前运行的版本。你可以把它看成服务器端的项目的根目录。

1. revisions.log

这里记录了你所有的部署记录。

1. Capistrano 3基本概念

Capistrano 3每次部署都会让服务器去SCM上拉取最新版本（注意不是本地版本，更不是本地尚未提交至SCM的文件），你可以通过下述指令开始部署过程：

$ cd $project\_root

$ cap $env deploy

其中$env是你要部署的环境，通常会有production和staging两个环境，偶尔也可能需要部署development环境。

注意：如果你是Rails程序员，不要把Capistrano的env和Rails的env搞混了。Capistrano的env只包含服务器的IP/域名、登录用户名及任意多个环境变量。

Rails的env指的是Rails的运行环境，包含数据库连接配置、SMTP服务器配置、是否缓存页面、是否serve静态文件等。

你可以在Capistrano的staging环境下部署Rails的production环境（大家都这么做），也可以在Capistrano的production环境下部署Rails的development环境（没人这么做，后果自负）。你要做的只是在Capistrano的对应配置文件里设置RAILS\_ENV环境变量而已。

Production环境指正式的生产环境，也就是你的目标用户真正访问的环境。

Staging环境指的是一套和生产环境相仿的环境，用来做压力测试等。

Development环境是开发环境，通常不需要通过Capistrano部署。

每次部署Capistrano都会经历这样一个状态变迁的过程：

deploy:starting（部署中）

deploy:started（部署已开始）

deploy:updating（正在更新服务器端文件）

deploy:updated（服务器端文件更新完毕）

deploy:publishing（服务器端正在切换当前版本）

deploy:published（服务器端当前版本已切换至最新版本）

deploy:finishing（正在处理一些收尾工作）

deploy:finished（一切就绪）

所有这些状态的前后都可以添加自定义的任务，例如在updated之后执行css/js的编译/混淆，published之后重启app服务器和后台任务等。所有这些任务，包括自定义任务，只要有一个失败，则部署失败。注意：部署失败后并不会自动回滚！Capistrano默认会在服务器端保留5个部署的版本，以便回滚。

你可以通过下面这条命令开始回滚流程：

$ cap $env rollback

这条指令会让服务器端的APP回滚一个版本。回滚也有一个状态变迁的过程：

deploy:starting（部署尚未开始）

deploy:started（部署已开始）

deploy:reverting（正在更新服务器端文件）

deploy:reverted（服务器端文件更新完毕）

deploy:publishing（服务器端正在切换当前版本）

deploy:published（服务器端当前版本已切换至最新版本）

deploy:finishing\_rollback（正在处理一些收尾工作）

deploy:finished（一切就绪）

和部署一样，所有这些状态的前后也可以添加自定义任务，例如在reverted之后停掉服务器，回滚数据库结构，在published之后把服务器再启动起来等。

值得一提的是，所有Capistrano脚本都“跑在本地”，这意味着你没有必要每次修改Capistrano的文件都往SCM上提交一个新版本。

1. Capistrano中的角色（Role）

Capistrano有一个“角色（Role）”的概念。一个角色某台服务器代表在整个Web App里的起到的作用。例如有些服务器是用来跑Web App逻辑的，则可以把它们列入app角色；有些服务器是用来跑数据库的，则可以把它们放进db角色；有些服务器是用来提供静态文件的，则可以把它们归入assets角色……。对于小型项目，一台服务器可能扮演了所有角色，而对于大型项目，一个角色就有多台服务器。你可以自定义角色，如lb角色代表负载均衡（Load Balancer）。事实上，Capistrano没有给你预定义任何角色。

你可以在config/deploy/\*.rb中定义你的角色。定义角色的语法有两种：

role :app, [‘user1@192.168.1.100’, ‘user2@192.168.1.101‘, ...]

role :db, [‘user3@192.168.1.200’, ‘user4@192.168.1.201‘, ...]

这种语法适用于大型项目，一个角色就有很多服务器。

server ‘192.168.1.100’, user: ‘user1’, roles: [‘app’, ‘assets’, ‘db’]

这种语法适用于小型项目，一台服务器包揽多个角色。

角色定义完了怎么用？请看下节。

1. 自定义Capistrano任务

你可以在lib/capistrano/tasks下创建自定义任务。下面演示一个最简单的任务：在远端服务器上往stdout里打印Hello World，捕获stdout的的输出，并打印在本地的控制台上：

# lib/capistrano/tasks/greetings.rake

namespace :greetings do

task :hello do

on roles(:app, :db) do

execute(‘echo Hello World’)

end

end

end

要执行它，只需在命令行里输入：

$ cap $env greetings:hello

接下来简要介绍一下Capistrano任务所用到的语法（Rake语法）。

首先，一个rake任务可以有一个命名空间（namespace）和一个任务名称。在这个例子里，命名空间是”greetings”，任务名是”hello”。任务也可以没有命名空间，不过这通常不是个好主意。

接下来的几行

on roles(:app, :db) do

# ...

end

定义了这个任务需要哪些角色来执行（角色的用处来了吧？）。记住每个角色可以对应多台服务器，所以里面的具体工作会在扮演这些角色的所有服务器上执行！在这个例子里，所有角色是app或者db的服务器都会去执行任务规定的工作。

最后来看具体的工作：

execute(‘echo Hello World’)

execute函数的作用是在远端的命令行里执行给定的命令（这里是echo Hello World）。Execute会捕获stdout的输出，并以debug的log级别打印到本地控制台。

Capistrano提供了很多很好用的函数，其中很多是从ssh-kit这个库借来的。下面简要列举几个常用的：

1. on

on server do

# ...

end

在指定的服务器上运行do-end里的内容。on后面可以跟服务器的IP或域名，可以加SSH端口，例如

on [‘192.168.1.100:22’, ‘192.168.1.101:33’] do

# ...

end

也可以像之前的例子一样指定角色，这里就不多说了。

1. with

with var1: ‘val1’, var2: ‘val2’ do

# ...

end

在远端机器上添加临时的环境变量（这里是VAR1=val1 VAR2=val2，注意Capistrano会把变量名变成全大写），并执行do-end里的内容

1. within

within ‘/some/dir’ do（Windows用户可把它想象成快捷方式）

# ..

end

在远端的指定文件夹里执行do-end里的内容。通常会用来跑服务器端的rake任务，因为rake任务通常需要在项目的根目录来跑。

1. execute

execute ‘some shell command’

在远端运行shell命令

5. rake

rake ‘some:rake:task’

在远端运行rake任务。通常会和within一起使用。

6. current\_path

返回远端当前运行版本的路径（$deploy\_path/current/）,类型是Pathname

1. release\_path

返回当前部署中的版本在远端的路径（$deploy\_path/releases/xxxxxxxx/）,类型是Pathname

1. deploy\_path

返回部署目录（也就是你在config/deploy.rb里配置的deploy\_to那一项）,类型是Pathname

1. releases\_path

返回远端保存所有发布版本的目录（$deploy\_to/releases/）,类型是Pathname

Rake说白了就是个Ruby的库，在Rake脚本里，所有Ruby语言的语法都适用。

1. 把自定义的任务加入部署流程

Capistrano可以很轻松的把自定义的任务加入部署流程，在部署的过程中自动完成自定义任务。在流程中添加自定义任务是在config/deploy.rb中完成的。

假定我们有一个java的maven项目，每次部署时只上传java源码，我们想在服务器上的代码发布完成后通过最新的源码重新构建项目，把构建出来的东西放到tomcat的webapp文件夹下，并重启tomcat。假定我们已经实现了这些任务：

mvn:build（用maven构建java项目）

tomcat:deploy（把构建出来的东西放进tomcat）

tomcat:restart（重启tomcat）

我们只需要在config/deploy.rb的最底下加上这样几句就行：

after ‘deploy:published’, ‘mvn:build’

after ‘mvn:build’, ‘tomcat:deploy’

after ‘tomcat:deploy’, ‘tomcat:restart’

有after当然就有before啦。用法差不多，我就不说了。

1. config/deploy.rb里的其他配置

deploy.rb里还有很多东西可以配，包括使用的SCM（Git还是SVN），SCM的URL等。这些设置都很简单，你直接看这个文件里的注释就能了解的差不多了。实在不行找Google。你也可以通过这个文件设置一些远端的环境变量，不过这些环境变量都是临时的，部署完了就没了。而且这个文件同样不适合存放敏感信息。

比较重要的两个配置是linked\_files和linked\_dirs。通常情况下，Capistrano每次部署会更新所有的文件和文件夹，但是有时候有些东西我并不想让它每次都更新，而是永远都用最早部署上去的那个版本的。比较重要的是app server的pid文件。由于这个文件保存了app server的pid（进程ID），而这个pid又关系到app server的重启，如果每次部署都更新这个文件，将导致服务器无法重启。我们可以把这个文件加入linked\_files，或把它所在的文件夹整个加进linked\_dirs。Capistrano会把linked\_files和linked\_dirs里的配置的文件/文件夹放进shared/，并在每次部署的版本里用symlink的方式链接进项目里，这就保证了每次部署这些文件/文件夹都不会被更新，但是在应用里还是一样去访问，不用调整访问路径。

1. 关于Capistrano的环境变量

这个可以说是UNIX/Linux用户最头疼的问题了。有时候命名手动ssh登上服务器去拿环境变量能拿到，但是用Capistrano部署的时候就死活拿不到。如果那个环境变量很重要，甚至能导致服务起不起来，或部署失败。为什么呢？理由是，Capistrano默认采用的是non-interactive、non-login的环境，而通常SSH上去拿的是interactive环境，有时可以是login环境。这些环境加载的环境变量并不相同。Non-interactive、non-login环境几乎不会被用到，所以它的配置文件在哪里我也不知道。那怎么办？我的解决方案是：让SSH带上本地的环境变量去访问服务器！

具体做法是：

1. 打开服务器端的/etc/ssh/sshd\_config文件（注意文件名里有个d），修改AcceptEnv这一项，添加你想要从本地获取的环境变量名，通常是以项目名称做前缀的环境变量名。重启服务器端的ssh服务。
2. 打开本地的/etc/ssh/ssh\_config文件（注意文件名里没有d），修改SendEnv这一项，把你想带上的环境变量名都加进去。
3. 打开本地的包含环境变量的配置文件（通常是~/.bashrc，如果你用的是login shell，则~/.bash\_login或~/.bash\_profile也OK），把要传到服务器的环境变量都设进去。
4. 用Capistrano部署

这个方案的好处在于，所有的环境变量都只存在于开发机，就算生产环境的机器被攻破了，也不会泄露这些环境变量的值，除非黑客attach进你的Web App进程。这样我们就可以放心地在环境变量里保存敏感信息啦。

这个方案的缺点在于它只能在当前SSH的TTY及其子进程里使用带过去的环境变量。对于PHP这类每个请求一个进程，而且不是从你当前的SSH的TTY fork出来的进程，这些环境变量是访问不到的。另外，如果你有crontab触发的后台定时任务，则在这些任务里也是访问不到这些环境变量的。对于这种情况，解决方案是：配置文件改个名字，在敏感信息的位置放个占位符，把它放进linked\_files。然后通过SSH带环境变量的方法带上你的真实的敏感信息，在远端复制那个改过名的文件，用敏感信息替换掉占位符，并修改注意在远端修改该文件的权限。真实的配置文件也可以放进linked\_files。这种方式的缺点在于一旦你的服务器被攻破，你的敏感信息暴露无疑。