Receta Capistrano detallada para realizar deployment de tu aplicación NodeJs

Ésta es una guía extensa en la que te explicamos como hacer deployment de una aplicación NodeJS usando Capistrano. Te explicamos todos los pasos necesarios para preparar el entorno y crearte tu propia receta o recipe Capistrano. Para una versión resumida, lee la guía Receta Capistrano para NodeJs.

La guía tiene como objeto que un usuario pueda realizar deployments de su aplicación con un desarrollo en su equipo local. Si has llegado aquí a través de una búsqueda simple en google o en otro buscador debes tener en cuenta que esta guía está pensada para trabajar en un plan de alojamiento con NodeJs en guebs.com. Algunos de los recursos presentes en esta guía sirven para cualquier aplicación en cualquier sitio, pero otros están pensados específicamente para el servicio de NodeJs en guebs.com. Si decides usarlos, asegúrate de comprender lo que hace cada uno de ellos.

Es fundamental acceder via ssh a tu cuenta de hosting

> *Como acceder vía SSH a tu cuenta*

Paso Previo

- Es necesario que crees la aplicación NodeJs en el panel de hosting. Hecho esto, entonces debes acceder a tu plan de alojamiento y realizar lo siguiente

```
cd ruby; rmdir miapp ; ln -s current miapp
```

Con esto dejaremos el directorio preparado para el deployment con capistrano

Trabajando con git

En primer lugar debes instalar git en tu equipo de desarrollo para que esté

disponible. Dependiendo de tu sistema operativo deberás usar algo «**yum install git**» o bien «**dnf install git**» para las versiones actuales de Fedora. En entornos debian «**apt-get install git** » como super usuario o bien con el gestor de software de tu distribución.

En el equipo donde estés desarrollando tu proyecto, típicamente en un entorno local, debes instalar capistrano y las herramientas de capistrano-npm, que es tan sencillo como ejecutar el siguiente comando:

```
gem install capistrano
gem install capistrano-npm
```

Ahora en nuestro equipo local debemos crear el repositorio local. Es importante tener en cuenta que los archivos con datos críticos no deben ser commitados al repositorio local, por lo que haremos uso de las capacidades de git para ignorar algunos archivos.

Para ello en raíz de la aplicación crearemos un archivo llamado **.gitignore**, que contendrá la ruta relativa al archivo database.yml

```
echo config.js >> .gitignore
```

Para inicicializar el repositorio realizamos lo siguiente en la raíz de la aplicación:

```
git init .

git add *

git config user.name "Tu nombre"

git config user.email "raul@midominio.com"

git commit -m "Commit inicial"
```

En remoto, en la cuenta de hosting, debes también crear un repositorio que será una réplica del que tienes en local. Cuando termines de trabajar en tu equipo local, realizarás un ‘push’ al remoto, lo que posibilitará que capistrano haga lo necesario. Para ello realizamos lo mismo que en local, pero con el añadido de que al crear el repositorio añadimos la opción –bare

```
mkdir nombrerepo

git init  --bare .

git config user.name "Tu nombre"

git config user.email "raul@midominio.com"
```

Antes de proseguir es necesario que tu usuario local del equipo pueda auntenticarse sin introducir un password contra el servicio ssh del plan de alojamiento aquí en guebs.com. Para ello debes configurar la autenticación de clave pública.

Como acceder vía SSH mediante llave pública

En este punto debemos establecer una relación entre el repositorio local y el repositorio remoto. El repositorio remoto debe ser añadido en tu equipo local en tu repositorio local para que se pueda hacer un «push» posteriormente. El modo de añadirlo es como repostiorio remoto de tu repositorio local

https://git-scm.com/book/en/v2/Git-Basics-Working-with-Remotes

Debemos ejecutar el siguiente comando por lo tanto en el equipo de desarrollo:

```
git remote add identificador ssh://usuario_cpanel@tu_dominio:333/home/usuario_cpanel/nombrerepo
```

«Identificador» es un nombre corto que identifica al repositorio internamente. tu_dominio debe resolver correctamente, de lo contrario no funcionará correctamente.

Cuando ejecutes «**git remote -v**» en tu repositorio local deberías ver este nuevo remote que has añadido. Por ejemplo si hubiéramos añadido como identificador 'produccion' veríamos lo siguiente:

```
produccion    ssh://usuario_cpanel@tudominio:333/nombrerepo (fetch)
```

```
produccion    ssh://usuario_cpanel@tudominio:333/nombrerepo (push)
```

Ahora es el momento de realizar un «push» de tu proyecto local al repositorio remoto de tu plan de alojamiento. Si el identificador que has asignado fue «produccion», entonces el comando es como sigue:

```
git push produccion master
```

Ahora debemos ir a la raíz de nuestra aplicación en el equipo local y ejecutar el siguiente comando que instalará los archivos necesarios para que capistrano maneje la aplicación. Debe ser ejecutado en la raíz de la aplicación

```
cap install
```

Hoy en día capistrano pregunta al usuario si quiere usar harrow.io. El objeto de esta guía no es este servicio por lo que decimos que no en este momento.

En el archivo **Capfile** debemos añadir las siguientes líneas:

```
require 'capistrano/npm'
```

Hecho esto en el archivo **config/deploy.rb** debemos configurar una serie de parámetros

```
set :application, 'miapp'

set :repo_url, 'git@example.com:me/my_repo.git'

set :deploy_to, '/home/usuario_cpanel/nodejs'
```

- En **application** establecemos el nombre de la aplicación. Debe ser el nombre que hemos establecido al crear la aplicación en el panel de hosting.
- En **repo_url** el repositorio que usaremos. Usaremos el repositorio que hemos creado previamente y que está en la raíz de nuestro plan de alojamiento. Si estuviéramos usando un repo remoto, entonces deberíamos rellenarlo como la directiva ejemplo anterior. Pero esta guía está pensada para tener un repositorio en el propio plan de alojamiento así que hacemos uso de file:// para definir el repositorio

```
set :repo_url, 'file:///home/usuario_cpanel/repodir'
```

- En **deploy_to** introducimos el directorio en que despositaremos la aplicación y sus releases. Si tu plan de alojamiento va a tener una sola aplicación, entonces puedes elegir nodejs/ como destino de tu aplicación. Todos los archivos residirán entonces dentro de la carpeta nodejs y al final del proceso nodejs/miapp apuntará a nodejs/current. Si, en cambio, tu plan de alojamiento va a tener varias aplicaciones, es necesario que elijas otra carpeta de tu plan de alojamiento. Esta guía está pensada para planes de alojamiento con una sola aplicación.

Ahora debemos configurar el archivo **config/deploy/production.rb** . En este archivo introduciremos las credenciales y direcciones con las que queremos que capistrano publique. En este caso que estamos describiendo, necesitamos el usuario principal del plan de alojamiento y el nombre de dominio. Además introduciremos la opción adicional «port», que nos servirá para decirle a capistrano que queremos usar el puerto 333 en lugar del estándar 22.

Cómo obtener las credenciales para CPanel desde el panel de hosting

Hemos de crear la siguiente línea sustituyendo donde corresponda

```
server 'tudominio.com', user: 'usuario_cpanel', roles: %w{app db web}, port: 333
```

En este archivo también necesitamos configurar el directorio temporal en que se trabajará en remoto. Para ello introducimos la siguiente directiva

```
set :tmp_dir, '/home/usuario_cpanel/tmp'
```

Y el nombre de la aplicación que hemos dado al declararla en el panel de hosting. (Sí, también hemos de añadirla en este archivo)

```
set :application, 'miapp'
```

En el entorno de guebs.com, sin embargo este error seguirá apareciendo: «mkdir: cannot create directory `/home/usuario_cpanel/public_html’: File exists» Es un error que se puede ignorar. También el que indica » stdin: is not a tty»

Gestión de archivos fuera del deployment.

Hay algunos archivos que es necesario mantener sólo en el servidor de producción , es decir, en el plan de alojamiento. Estos archivos serán linkados de tal modo por capistrano que esperará que existan en el directorio shared.

Por ejemplo, típicamente especificaremos que el archivo **config.js** (en el caso de ghost) sea uno de estos archivos, por lo que no deberá existir en el repositorio y deberá existir en **shared/config.js**.

También podemos hacerlo con aquellos directorios que estén en la misma situación. Para ello disponemos de las opciones **linked_dirs** y **linked_files**. Por ejemplo para añadir el archivo config.js, en el archivo deploy/production.rb debemos escribir lo siguiente:

```
append :linked_files, "config.js", "otro_archivo"
```

Reiniciar la aplicación

Para que cada vez que se realice un deployment de la aplicación ésta se reinicie, debemos crear el archivo **lib/capistrano/tasks/touch.rake** con el siguiente contenido:

```
namespace :deploy do
desc "Reinicia aplicacion"
 task :restart do
  on roles(:all) do
    execute "mkdir -p #{ current_path }/tmp/ ; touch #{
current_path }/tmp/restart.txt"
  end
 end
end
```

Esta tarea puede ser llamada individualmente en cualquier momento con este comando, además de que se ejecutará automáticamente en cada deploy.

```
cap production deploy:restart
```

Deploy final

Gracias a las directivas que hemos ido añadiendo, la aplicación se instalará en el lugar que hemos indicado y como hemos indicado. Además tratará de realizar un «npm install» para instalar los módulos en el plan de alojamiento. Pero para que la ejecución sea correcta es necesario que se activen los compiladores :

> *Activar el acceso a compiladores*

Posteriormente sólo necesitamos hacer deployments normales.

```
cap production deploy
```

Actualizada a día 22 de junio de 2016 para capistrano > 3.x