UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escuela Técnica Superior de Ingeniería Informática

Universitat Politècnica de València

Aplicación de herramientas *DevOps* en entornos de Desarrollo Web

Proyecto Final de Carrera

ITIG

Autor: Simón Muñoz Marín

Director: Andrés Terrasa Barrena

28/02/2015

Resumen

A lo largo del proyecto presentaremos la problemática a las que se enfrentan gran parte de las empresas de desarrollo web, con la gestión de los distintos entornos dónde se codifican y ejecutan sus aplicaciones. Para solucionarlo, introduciremos el movimiento *DevOps* y cómo herramientas creadas en torno al mismo como *VirtualBox*, *Vagrant* y *Ansible*, nos permiten redefinir la forma de trabajar de estas empresas y aumentar su productividad en el camino.

Palabras clave: devops, virtualbox, vagrant, ansible, web, servidores, nginx, php-fpm, mysql, drupal

Tabla de contenidos

1 Introducción

Históricamente la programación web, y en general, la programación basada en arquitecturas cliente-servidor, ha sufrido severos problemas de integración por las diferencias entre los distintos entornos donde desarrolladores programaban, y aquellos en los que finalmente una aplicación era desplegada por administradores de sistemas.

Versiones de librerías distintas, paquetes no disponibles en un entorno pero en el otro sí, distintas configuraciones de permisos... son sólo una parte de un innumerable conjunto de problemas que se producen cuando desarrolladores y operaciones trabajan de forma independiente.

Esta pelea constante no sólo genera frustración en los dos grupos, sino que llevados a la cuenta de resultados, también afecta de forma notable la productividad de cualquier empresa que se dedique al desarrollo de sistemas informáticos.

A raíz de la identificación de este problema, surgió entre los profesionales a ambos lados del espectro un movimiento por tratar de romper las barreras que separaban a los departamentos de desarrollo y operaciones, el movimiento *DevOps*[i].

Si nos atenemos a la definición de la Wikipedia:

> *DevOps (a portmanteau of "development" and "operations") is a concept dealing with, among other things: software development, operations, and services. It emphasizes communication, collaboration, and integration between software developers and information technology (IT) operations personnel.[1][2] DevOps is a response to the interdependence of software development and IT operations. It aims to help an organization rapidly produce software products and services.*[ii]

En la búsqueda de puntos de encuentro, durante los últimos años se han desarrollado nuevas herramientas que han cambiado la forma en la que los administradores de sistemas y desarrolladores colaboran para producir mejor software.

El proyecto de final de carrera que pretendemos desarrollar se centrará en cómo aprovechar varias de estas nuevas herramientas, como Ansible[iii] y Vagrant[iv] para mejorar la eficiencia de un departamento de desarrollo web en una empresa tradicional de desarrollo de software.

1.1 Problemas detectados en una empresa de desarrollo

En la mayor parte de empresas orientadas al desarrollo de aplicaciones web distinguiremos dos departamentos claramente diferenciados, uno dónde los programadores las desarrollan, y otro donde los administradores de sistemas se encargan de mantener los sistemas sobre los que se ejecutan y desplegarlas.

Así lo hace por ejemplo la empresa en la que trabaja el proyectando, Aureka Internet SL, una firma valenciana de desarrollo de aplicaciones web sobre Linux. A fecha de hoy, esta empresa basa sus proyectos en Ubuntu 12.04 LTS[v], una distribución ideada para su uso en servidores, y con soporte de paquetes garantizado durante cinco años por parte de Canonical[vi], la entidad detrás de Ubuntu.

Una de las primeras decisiones que el responsable de informática de la compañía tomó hace cinco años fue, tratando de prevenir problemas de compatibilidad entre los entornos de desarrollo y producción, que los programadores utilizaran la misma distribución de Linux que había desplegada en los servidores de producción.

Esta estrategia funcionó relativamente bien mientras la empresa se ciñó a un tipo de proyectos concreto, los cuales requerían siempre de la misma configuración base, pero pronto surgieron algunos inconvenientes no deseados.

1.1.1 Gestión de dependencias

Y es que, con el paso del tiempo, los desarrollos de las distintas aplicaciones requerían paquetes especiales que necesitaban de una configuración manual por parte del administrador de sistemas. Configuración que a su vez debía ser replicada en los ordenadores locales de los desarrolladores implicados, con el consiguiente coste en tiempo y dinero.

No sólo eso, sino que además, esto provocaba errores de inconsistencia entre aquellos programadores que cambiaban de proyecto y que, al hacer un *pull* del código al volver al proyecto original meses después y tratar de ejecutar la aplicación, lo más probable es que les fallase y perdieran un tiempo muy valioso tratando de descubrir el origen del problema.

Pero cambiar la configuración base de los equipos de los desarrolladores también tenía otro problema, y es que en el largo plazo, se tendía a asumir como paquetes por defecto a algunos que se habían instalado manualmente hacía tanto que ya nadie lo recordaba. Así por ejemplo, se comenzaba un desarrollo nuevo asumiendo que estaba compilado el módulo mod_geoip[vii] de nginx[viii], cuando éste sólo estaba desplegado en uno de los servidores online para una aplicación determinada.

1.1.2 Coste de incorporación de un nuevo miembro al equipo

Otro problema al que se enfrenta la compañía es el tiempo que se pierde cada vez que se incorpora un nuevo miembro al equipo de desarrollo, principalmente debido a lo complicado de la creación de un entorno de un proyecto web. Por poner un ejemplo, un programador de un programa compilado únicamente suele necesitar el compilador de turno en la máquina destino, pero un desarrollador web necesita que estén funcionando diversos sistemas independientes para poder siquiera ejecutar su aplicación, como son un servidor web, un servidor de base de datos, o un servidor de correo entre otros.

Así pues, con cada nuevo empleado, se dilapida un valiosísimo tiempo por parte de sus compañeros para explicarle todos los pasos para configurar los servicios que necesita. Una semana perdida antes de que escriba su primera línea de código productivo no es nada fuera de lo común.

1.1.3 Configuraciones subóptimas

Relacionado con lo anterior, hemos detectado que los servicios instalados en los ordenadores locales de los desarrolladores, suelen estar mal configurados de acuerdo a sus características. Esto es así porque los paquetes base que se añaden cuando instalas, por ejemplo, un servidor de base de datos como MySQL, traen una configuración por defecto pensada para ser desplegada en producción, y no en un ordenador de desarrollo con unas características generalmente mucho más reducidas. Esto hace que el desarrollo y la depuración de fallos por ejemplo, sea mucho más pesado de lo que en realidad debiera ser.

1.1.4 Dependencia del administrador de sistemas

Por último, otro problema habitual, no solo en esta empresa, sino en cualquier otra en la que trabaje un administrador de sistemas, es que generalmente su labor se percibe como una caja negra generalmente inaccesible. Esto es así porque, si bien los programadores utilizan herramientas de gestión de versiones que permiten de forma muy sencilla trazar el historial completo de cualquier cambio en el código, no encontramos lo mismo en el quehacer diario de un *sysadmin*.

Generalmente los administradores de sistemas guardarán un registro de cambios realizados en los sistemas, que puede, o no, estar compartido con el equipo, pero del que además depende su utilidad, del nivel de detalle que ponga en documentar sus cambios, tarea que puede ser muy subjetiva de una persona a otra.

Esta falta de procesos o herramientas de control, puede llegar en algunos casos extremos a convertir a las empresas en “rehenes” de sus propios administradores.

1.2 Objetivos del PFC

Hemos identificado cuatro áreas en los que la empresa podría ganar un extra de productividad si consiguiéramos solucionarlos. Los podríamos resumir en:

- Mejorar la gestión de dependencias entre aplicaciones y la configuración de sistemas
- Reducir el alto coste de introducir un nuevo miembro en el equipo
- Optimizar las configuraciones específicas de cada entorno
- Implantar procesos que permitan hacer más visible el trabajo del administrador de sistemas

En todos los problemas, la figura del administrador de sistemas es clave, y es que sus conocimientos se aplican para cada una de las soluciones. Sin embargo, si las observamos con detenimiento, son tareas de carácter repetitivo en cada proyecto, y por lo tanto candidatas ideales a ser automatizadas.

Nuestro principal objetivo con este PFC es proponer un nuevo flujo de trabajo alternativo al actual, utilizando herramientas del ámbito *DevOps* de código libre, y que permitan solventar los problemas detectados aumentando la productividad de la compañía.

El éxito del proyecto se medirá en si somos capaces de proveer un entorno de desarrollo completo a un desarrollador local, de forma que pueda ponerse a programar de forma rápida, en un entorno multiplataforma, y configurado de acuerdo a las necesidades de su proyecto.

Ese entorno local deberá ser capaz de:

Lanzar y configurar una máquina virtual sobre la que cargar nuestro código

Cada desarrollo que abordemos tendrá asociado un archivo de configuración dado, el cual definirá las características de una máquina virtual que crearemos en base al mismo, y que servirá de plataforma sobre la que se ejecutará.

Instalar los requisitos necesarios (servicios y configuraciones) en esa máquina virtual

La aplicación de ejemplo que utilizaremos para validar nuestro proyecto será un gestor de contenidos web PHP, Drupal. Nuestro entorno debería ser capaz de instalar automáticamente, y configurar, los paquetes necesarios para su correcto funcionamiento, como por ejemplo son:

- Un servidor web (nginx[ix])
- Un servidor PHP (php5-fpm[x])
- Un servidor de base de datos (MySQL[xi])

Configurar e inicializar la aplicación para poder trabajar sobre ella

Finalmente, nuestro entorno debe ser capaz de, una vez instalados los paquetes necesarios para ejecutar nuestro desarrollo, configurarlo e inicializarlo de forma que un programador pudiera acceder a él a través del navegador en el ordenador anfitrión de la máquina virtual.

1.3 Plan de trabajo

Una vez realizada la introducción a este trabajo de fin de carrera, planteamos que el resto del mismo se divida en tres grandes bloques.

1.3.1 Iniciación a herramientas *DevOps*

Un primer bloque dónde introduciremos al lector en las distintas herramientas de código libre del ámbito *DevOps* que utilizaremos para lograr nuestros objetivos, como son VirtualBox[xii], Vagrant[xiii], y Ansible[xiv].

Utilizaremos VirtualBox como gestor y motor de creación de máquinas virtuales. Introduciremos su uso básico y crearemos una máquina virtual de forma completamente manual siguiendo las instrucciones de su asistente.

Una vez creada nuestra primera máquina virtual, introduciremos Vagrant, un programa de línea de comandos que nos permitirá automatizar y definir mediante código la creación de máquinas virtuales. De esta forma, podremos añadir a nuestro proyecto un archivo que nos permitirá, con tan sólo una orden, levantar una máquina virtual completa, obviando de esa forma tener que pasar por el asistente de VirtualBox.

En último lugar introduciremos Ansible, un gestor de aprovisionamiento, que nos permitirá automatizar la tarea de gestión y configuración de los distintos paquetes y servicios necesarios para que nuestra aplicación pueda ejecutarse con éxito. Del mismo modo que Vagrant nos permitía añadir un archivo con la configuración de nuestra máquina virtual, Ansible también nos permitirá registrar en nuestro repositorio los archivos necesarios para la configuración de la misma.

1.3.2 Creación de una infraestructura virtualizada

Tras introducir las tres herramientas básicas que nos permitirán completar nuestro proyecto, abordaremos un segundo bloque en el que pondremos en práctica dichas herramientas. Comenzaremos introduciendo primero GitHub, el servicio web de gestión de repositorios Git por excelencia, el cual nos permitirá registrar los distintos cambios conforme vayamos avanzando en nuestro proyecto.

Continuaremos utilizando Vagrant introduciendo el archivo *Vagrantfile*, que definirá los parámetros lógicos de la máquina virtual que utilizaremos. También veremos como utilizar *Vagrant Cloud* para importar y levantar una máquina Ubuntu 12 LTS con tan sólo una instrucción.

Una vez creada nuestra máquina virtual, nos adentraremos en su aprovisionamiento con Ansible, e iremos paso a paso instalando y configurando los distintos servicios necesarios para ejecutar una aplicación web php en la máquina virtual, tales como son un servidor web (nginx), un servidor de base de datos (MySQL) y un intérprete PHP (PHP5-FPM), para pasar en último lugar a configurar una instancia del gestor de contenidos (Drupal) que utilizaremos como ejemplo en el último bloque del proyecto.

1.3.3 Prueba y conclusiones

En el tercer bloque, y en último lugar, veremos un caso práctico posible de uso en una empresa de desarrollo web, y de esa forma entender cómo estas nuevas prácticas pueden aumentar la productividad de cualquier equipo de programadores.

Simularemos una situación dónde un conjunto de programadores recibe la notificación de que una de las aplicaciones que instaló a un cliente años atrás, necesita una actualización de seguridad. A lo largo del capítulo, observaremos las ventajas de haber definido la infraestructura junto con el código fuente de la misma.

Tras el ejemplo, añadiremos nuestras conclusiones sobre el proyecto, añadiendo una crítica al futuro de los administradores de sistemas y sugiriendo posibles ampliaciones al mismo.

2 Herramientas DevOps

Para la realización del proyecto vamos a utilizar una serie de herramientas *DevOps* que a continuación introduciremos de forma progresiva.

2.1 VirtualBox

Sin duda, una de las piezas clave del desarrollo del movimiento DevOps e incluso de lo que hoy llamaríamos *Cloud*, o servidores en la nube (Amazon AWS[xv], Google Compute Engine[xvi], Heroku[xvii]), ha sido la virtualización.

La virtualización nos permite, sea cual sea el sistema operativo anfitrión, lanzar una máquina virtual totalmente independiente que utilizará parte de los recursos del primero para funcionar. Simplificando y en nuestro caso concreto, la virtualización nos permitirá cargar el sistema operativo Ubuntu 12.04 LTS, desde un ordenador local ejecutando Windows u OSX.

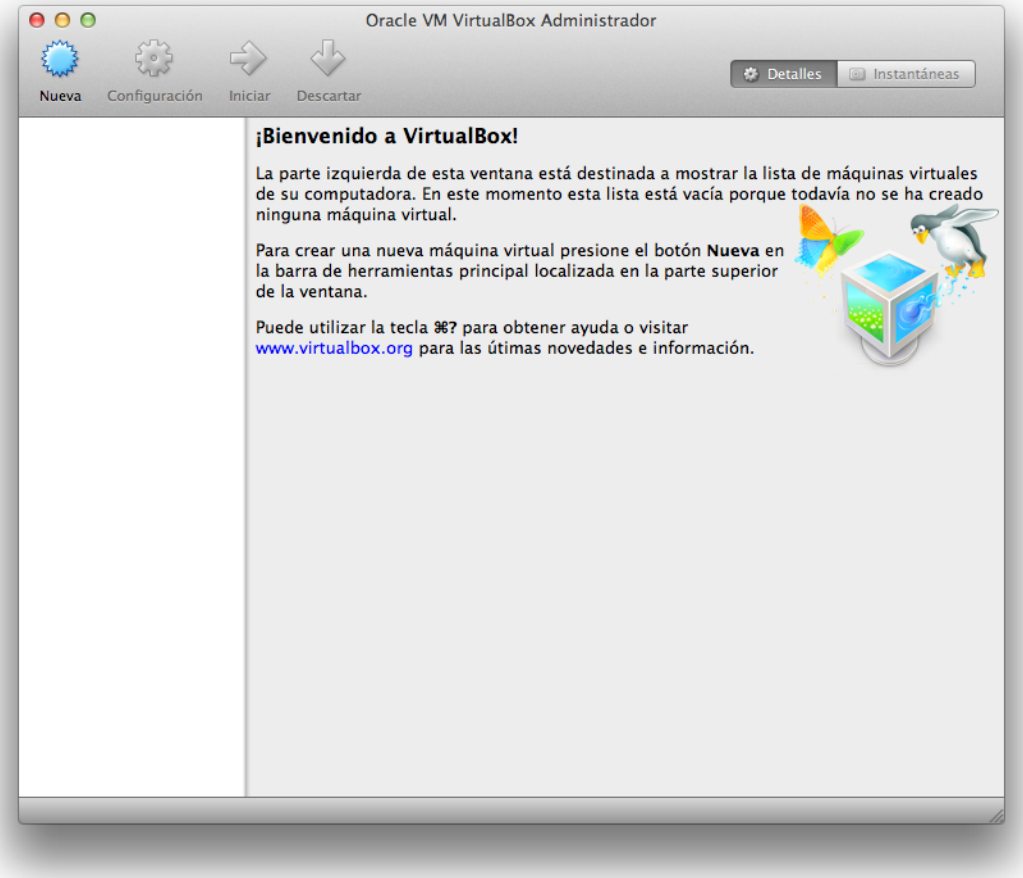
Existen muchas alternativas propietarias para la gestión de máquinas virtuales, como pueden ser VMWare[xviii] o Parallels[xix], pero nos centraremos en VirtualBox[xx], la alternativa de referencia de código libre creada por innotek GmbH, adquirida por Sun Microsystems en 2008, y propiedad actual de Oracle tras la absorción de Sun por ésta.[xxi]

Nada mejor que verlo en funcionamiento, así que acto seguido crearemos una máquina virtual VirtualBox de forma manual, del mismo modo que tendría que realizarlo nuestro administrador de sistemas si quisiera ponerla a disposición de los miembros del equipo.

Lo hacemos sólo a modo de referencia, ya que la próxima herramienta que utilicemos, Vagrant, nos permitirá crearlas programáticamente desde la línea de comandos evitando así tener que lidiar con procesos manuales.

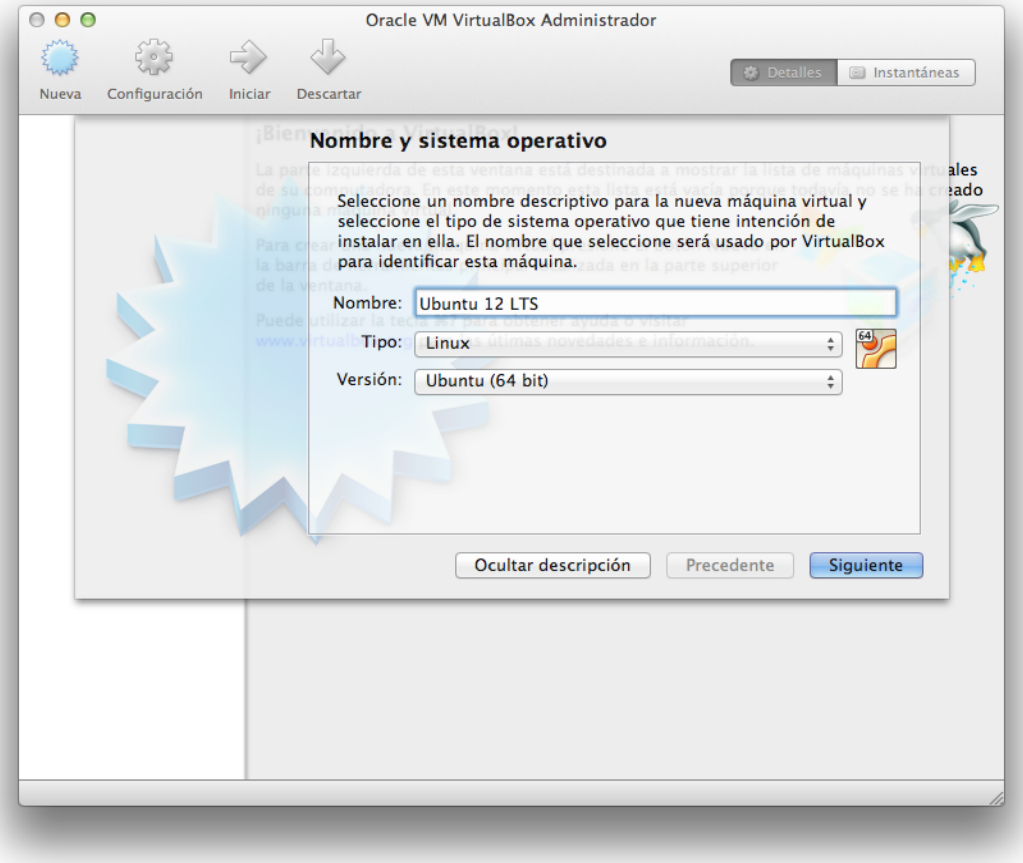
2.1.1 Creación de una máquina virtual con VirtualBox

Crear una nueva máquina virtual a través del interfaz de VirtualBox es relativamente sencillo. Tras abrir VirtualBox por primera vez veremos una pantalla como la que sigue, en la que tan sólo tendremos una opción que es pulsar sobre el botón “Nueva” en la parte superior izquierda.



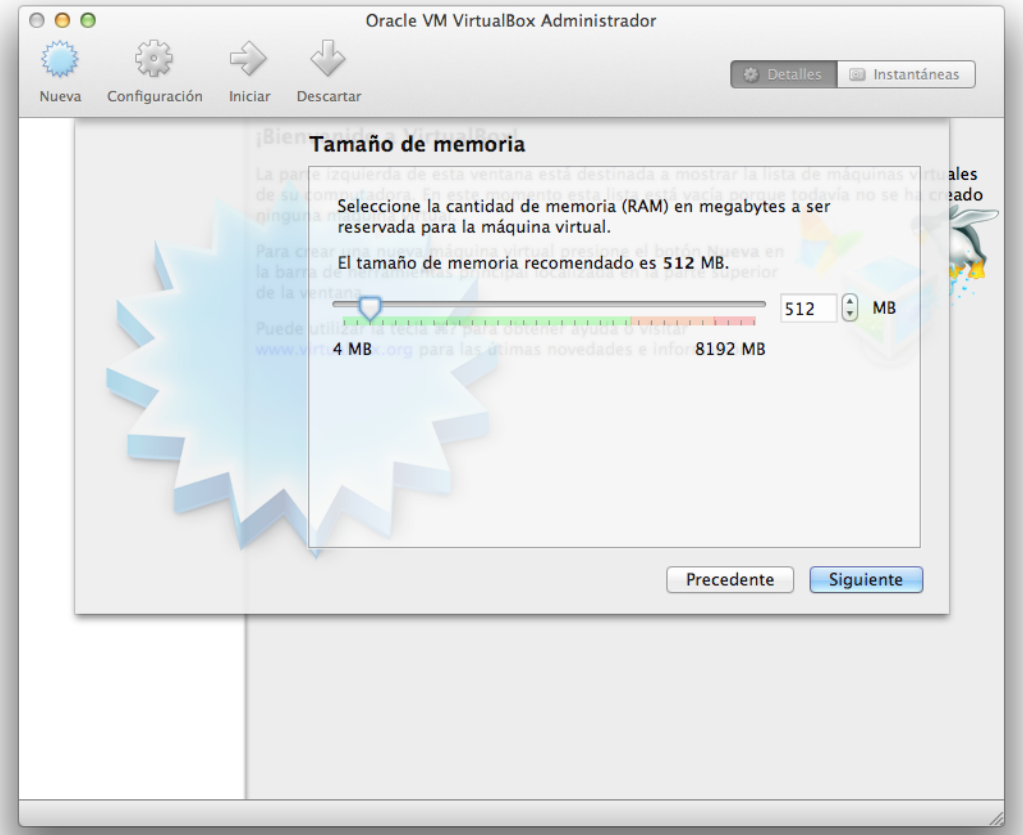
2.1.2 Detalles de la máquina virtual

VirtualBox nos pedirá entonces que definamos algunas características del sistema operativo que vamos a instalar en la máquina virtual, en este caso, una distribución de Ubuntu de 64 Bits que llamaremos 'Ubuntu 12 LTS'.



2.1.3 Memoria RAM

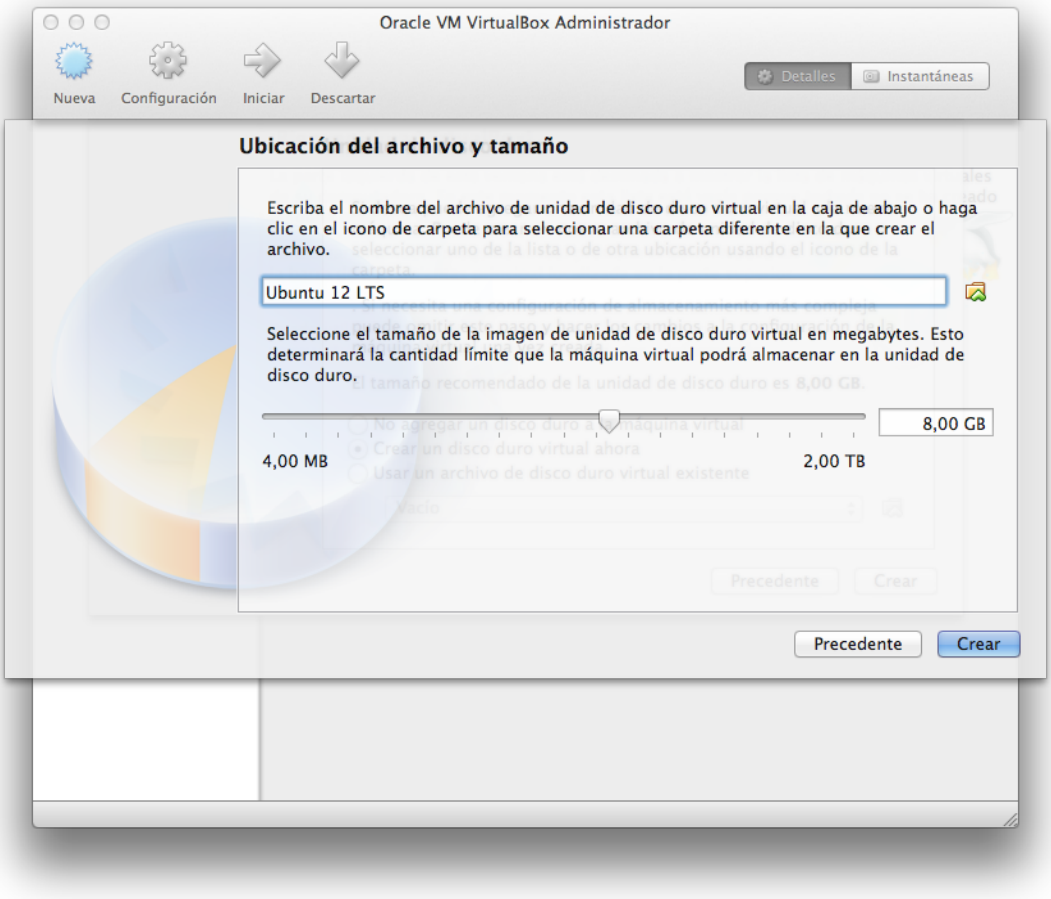
A continuación, VirtualBox nos pedirá definir el tamaño de memoria RAM que le queremos otorgar a nuestra máquina virtual. Nos ofrece por defecto otorgarle 512MB, con un máximo de 8G, que coincide con la memoria total del ordenador anfitrión. Dejaremos la opción por defecto, 512MB, y avanzamos al siguiente paso.



2.1.4 Disco Duro

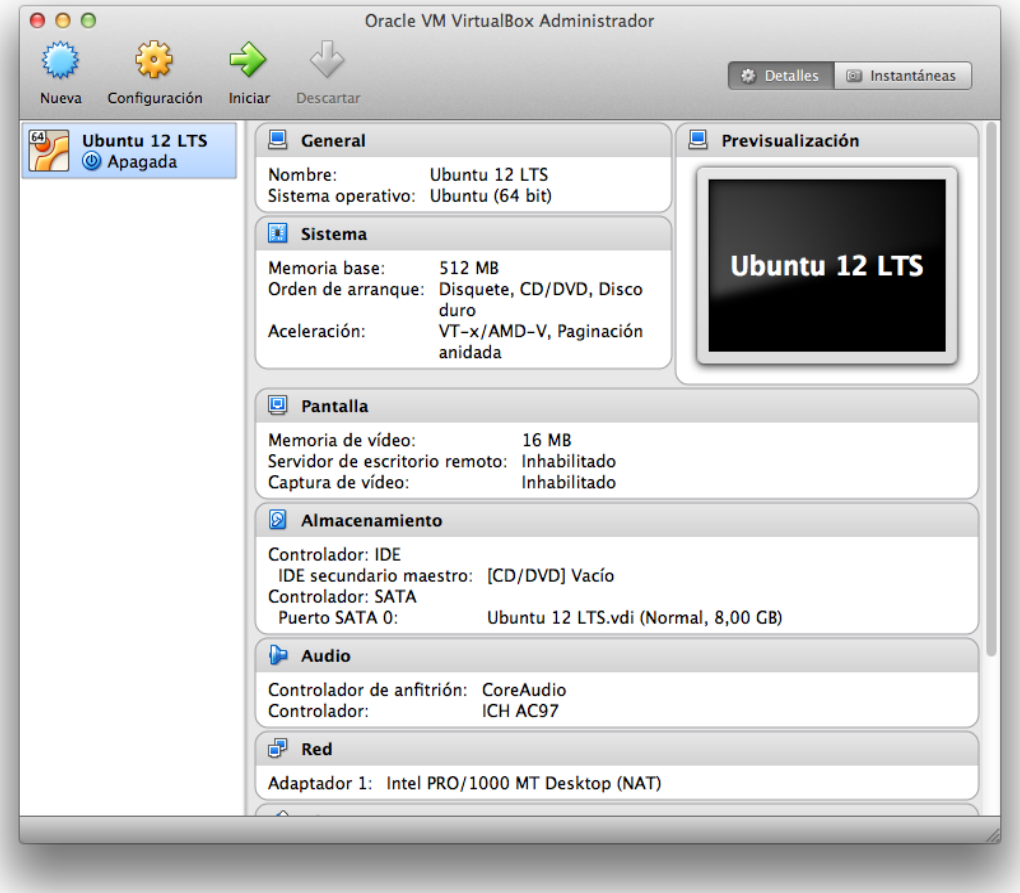
El siguiente paso será crear un disco duro virtual que contendrá los archivos de nuestra máquina virtual. El aspecto clave de este punto es escoger un tamaño correcto para el sistema operativo que vayamos a instalar en nuestra máquina virtual, pues escoger un tamaño más pequeño que el necesario nos obligaría a comenzar el proceso desde cero en el caso de que la instalación del sistema operativo a instalar requiriese más espacio del asignado.

Para el caso que nos ocupa, en el que tan sólo vamos a instalar el sistema operativo con motivo de reproducir el proceso de instalación, los 8GB que VirtualBox nos ofrece por defecto serán más que suficiente.



2.1.5 Nuestra primera máquina virtual

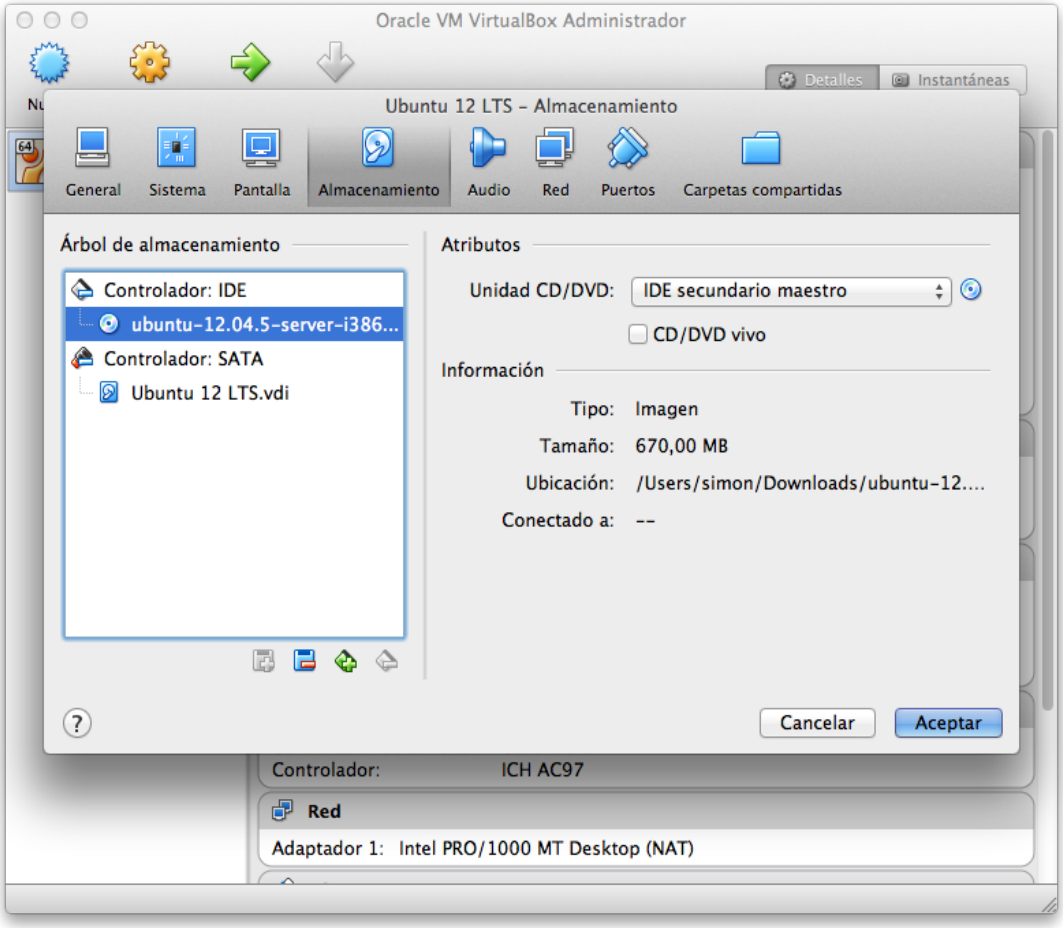
Una vez completado el asistente, VirtualBox nos devuelve a la pantalla inicial, con una nueva máquina virtual ya creada. Una máquina virtual recién creada sería equivalente a un ordenador al que no le hemos instalado ningún sistema operativo todavía. El siguiente paso, por tanto, será instalar Ubuntu.



2.1.6 Cargando un sistema operativo

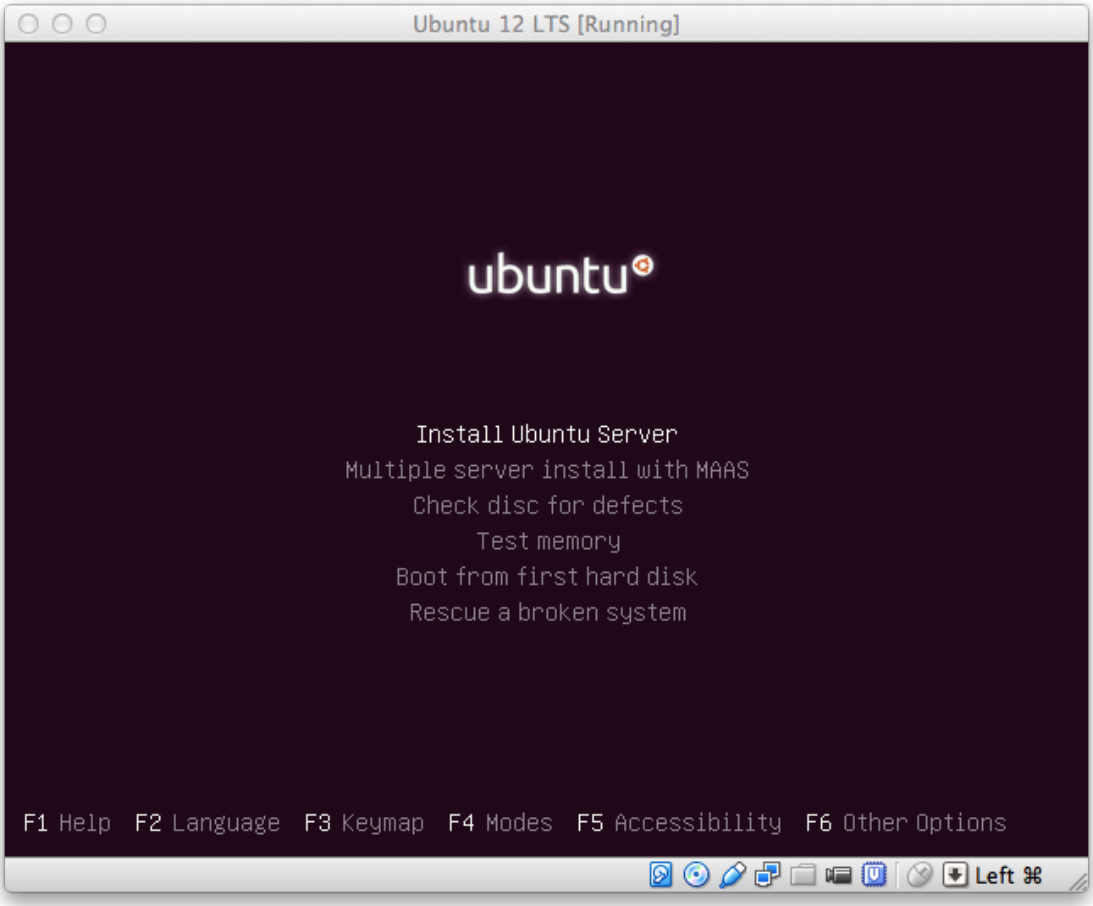
Siguiendo con el ejemplo del ordenador, VirtualBox nos provee controladores de dispositivos como son USB o CD/DVD. Obviamente se trata de controladores virtuales que pueden enlazar a dispositivos físicos del anfitrión.

Así por ejemplo, para simular que hemos introducido en nuestra máquina virtual un CD de instalación de Ubuntu, podemos descargar un archivo de imagen ISO desde Internet y cargarlo a través del menú Configuración > Almacenamiento. En esta ocasión hemos descargado la imagen desde la propia página de Releases de Ubuntu[xxii].



2.1.7 Instalando un sistema operativo

Una vez "introducido el CD" en nuestra máquina virtual, tan sólo necesitamos activarla como haríamos con cualquier otro ordenador para que comience a leerlo y a ejecutar la instalación.



2.1.8 Inconvenientes de VirtualBox

Una vez completado el proceso de instalación, tendremos una máquina virtual de Ubuntu VirtualBox dentro de un ordenador anfitrión cualquiera (Windows o Mac por ejemplo).

Llegados a ese punto, nuestro administrador de sistemas podría poner a disposición de cualquier miembro del equipo esta máquina virtual. Sin embargo, esta solución presenta algunos problemas.

Y es que, en primer lugar, la distribución entre los miembros del equipo es complicada. Al fin y al cabo, la máquina virtual mínima que hemos creado puede llegar a pesar hasta los 8GB del tamaño máximo del disco duro que le hemos asignado.

Por otra parte, la creación de la máquina virtual vuelve a convertirse en una tarea reservada al administrador de sistemas, y cuya documentación al detalle depende de nuevo de su buena voluntad.

En tercer y último lugar, asumiendo que utilizáramos esta máquina virtual para instalar las dependencias de la aplicación, un cambio en las mismas exigiría modificar la máquina virtual y volver a distribuirla, lo que es más que probable que creara problemas de versiones entre distintos miembros del equipo. Por ejemplo, dos desarrolladores programando sobre versiones distintas de la máquina virtual.

En definitiva, VirtualBox nos provee de una buena herramienta para crear nuestro entorno, pero nos hace falta algo más que nos permita tener más flexibilidad e independizar al *sysadmin* de tener que realizar este proceso manual. Es en este punto donde Vagrant entra en escena.

2.2 Vagrant

Según la definición de la Wikipedia:

> *Vagrant is free and open-source software for creating and configuring virtual development environments. It can be considered a wrapper around virtualization software such as VirtualBox and configuration management software such as Chef, Salt and Puppet.*[xxiii]

En otras palabras, Vagrant[xxiv] es un interfaz que nos permitirá crear máquinas virtuales programáticamente desde la línea de comandos, habilidad la cual será clave para poder automatizar y versionar el código de nuestras máquinas.

Para entender cómo funciona Vagrant probablemente lo mejor sea verlo en funcionamiento, así que vamos a mostrar un pequeño ejemplo, desde una ordenador con Mac OSX al que hemos instalado Vagrant siguiendo las instrucciones de su página web[xxv].

Veamos cómo decirle a Vagrant que cree una máquina virtual con Ubuntu en tan sólo dos comandos.

2.2.1 Inicializando una máquina virtual

En primer lugar, vamos a decirle a Vagrant que ejecute el proceso *init* invocando una máquina virtual de tipo hashicorp/precise64, dónde Hashicorp[xxvi] es el creador de la máquina, en este caso la misma empresa que desarrollar Vagrant, y *precise64* es el alias con el que Canonical identifica una distribución Ubuntu 12 LTS.

```
$ vagrant init hashicorp/precise64
A `Vagrantfile` has been placed in this directory. You are now
ready to `vagrant up` your first virtual environment! Please read
the comments in the Vagrantfile as well as documentation on
`vagrantup.com` for more information on using Vagrant.
```

Como podemos observar a través del output de nuestra orden, tras el init, Vagrant habrá creado un fichero llamado *Vagrantfile* en el directorio desde el que hayamos ejecutado la orden. Si hacemos un listado del directorio lo podremos ver:

```
$ ls -la .
total 24
drwxr-xr-x   5 simon  staff   170 11 oct 16:16 .
drwxr-x---  12 simon  staff   408  7 oct 18:50 ..
drwxr-xr-x  13 simon  staff   442 11 oct 16:18 .git
-rw-r--r--   1 simon  staff     8  7 oct 18:50 README.md
-rw-r--r--   1 simon  staff  4829 11 oct 16:16 Vagrantfile
```

2.2.2 El fichero *Vagrantfile*

Este fichero es el que Vagrant utilizará para gestionar la máquina virtual que lanzaremos posteriormente. Echemos un vistazo a su contenido:

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

# Vagrantfile API/syntax version. Don't touch unless you know what you're
doing!
VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  # All Vagrant configuration is done here. The most common configuration
  # options are documented and commented below. For a complete reference,
  # please see the online documentation at vagrantup.com.

  # Every Vagrant virtual environment requires a box to build off of.
  config.vm.box = "hashicorp/precise64"

  # Disable automatic box update checking. If you disable this, then
  # boxes will only be checked for updates when the user runs
  # `vagrant box outdated`. This is not recommended.
  # config.vm.box_check_update = false

  # Create a forwarded port mapping which allows access to a specific port
  # within the machine from a port on the host machine. In the example below,
  # accessing "localhost:8080" will access port 80 on the guest machine.
  # config.vm.network "forwarded_port", guest: 80, host: 8080

  # Create a private network, which allows host-only access to the machine
  # using a specific IP.
  # config.vm.network "private_network", ip: "192.168.33.10"

  # Create a public network, which generally matched to bridged network.
  # Bridged networks make the machine appear as another physical device on
  # your network.
  # config.vm.network "public_network"

  # If true, then any SSH connections made will enable agent forwarding.
  # Default value: false
  # config.ssh.forward_agent = true

  # Share an additional folder to the guest VM. The first argument is
  # the path on the host to the actual folder. The second argument is
  # the path on the guest to mount the folder. And the optional third
  # argument is a set of non-required options.
  # config.vm.synced_folder "../data", "/vagrant_data"

  # Provider-specific configuration so you can fine-tune various
  # backing providers for Vagrant. These expose provider-specific options.
  # Example for VirtualBox:
  #
  # config.vm.provider "virtualbox" do |vb|
```


```
  #   # Don't boot with headless mode

  #   vb.gui = true

  #
  #   # Use VBoxManage to customize the VM. For example to change memory:
  #   vb.customize ["modifyvm", :id, "--memory", "1024"]
  # end
  #
  # View the documentation for the provider you're using for more
  # information on available options.

  # Enable provisioning with CFEngine. CFEngine Community packages are
  # automatically installed. For example, configure the host as a
  # policy server and optionally a policy file to run:
  #
  # config.vm.provision "cfengine" do |cf|
  #   cf.am_policy_hub = true
  #   # cf.run_file = "motd.cf"
  # end
  #
  # You can also configure and bootstrap a client to an existing
  # policy server:
  #
  # config.vm.provision "cfengine" do |cf|
  #   cf.policy_server_address = "10.0.2.15"
  # end

  # Enable provisioning with Puppet stand alone.  Puppet manifests
  # are contained in a directory path relative to this Vagrantfile.
  # You will need to create the manifests directory and a manifest in
  # the file default.pp in the manifests_path directory.
  #
  # config.vm.provision "puppet" do |puppet|
  #   puppet.manifests_path = "manifests"
  #   puppet.manifest_file  = "site.pp"
  # end

  # Enable provisioning with chef solo, specifying a cookbooks path, roles
  # path, and data_bags path (all relative to this Vagrantfile), and adding
  # some recipes and/or roles.
  #
  # config.vm.provision "chef_solo" do |chef|
  #   chef.cookbooks_path = "../my-recipes/cookbooks"
  #   chef.roles_path = "../my-recipes/roles"
  #   chef.data_bags_path = "../my-recipes/data_bags"
  #   chef.add_recipe "mysql"
  #   chef.add_role "web"
  #
  #   # You may also specify custom JSON attributes:
  #   chef.json = { :mysql_password => "foo" }
  # end

  # Enable provisioning with chef server, specifying the chef server URL,
  # and the path to the validation key (relative to this Vagrantfile).
  #
```

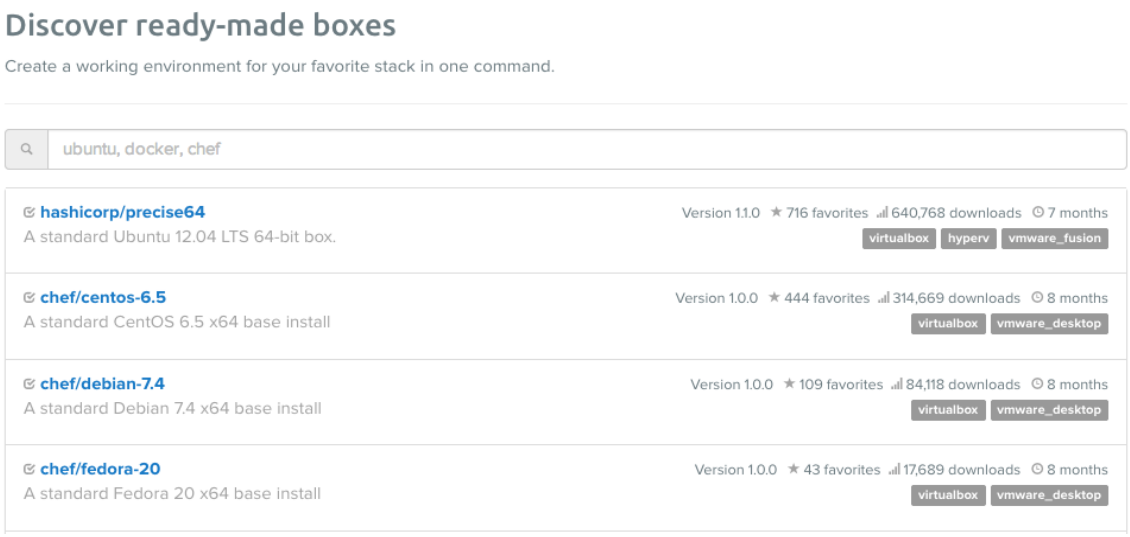
```
  # The Opscode Platform uses HTTPS. Substitute your organization for
  # ORGNAME in the URL and validation key.
  #

  # If you have your own Chef Server, use the appropriate URL, which may be

  # HTTP instead of HTTPS depending on your configuration. Also change the
  # validation key to validation.pem.
  #
  # config.vm.provision "chef_client" do |chef|
  #   chef.chef_server_url = "https://api.opscode.com/organizations/ORGNAME"
  #   chef.validation_key_path = "ORGNAME-validator.pem"
  # end
  #
  # If you're using the Opscode platform, your validator client is
  # ORGNAME-validator, replacing ORGNAME with your organization name.
  #
  # If you have your own Chef Server, the default validation client name is
  # chef-validator, unless you changed the configuration.
  #
  #   chef.validation_client_name = "ORGNAME-validator"
end
```

Podemos apreciar que se trata de un archivo de configuración de texto, escrito en Ruby[xxvii], y que la mayoría del documento está comentado para que lo utilicemos a modo de plantilla, para definir aspectos como el tipo de máquina a utilizar, datos de red, directorios compartidos y el gestor de aprovisionamiento.

La línea que nos interesa es la que sigue, heredera directa de la invocación de *`vagrant init`* que hemos invocado:

```
# Every Vagrant virtual environment requires a box to build off of.
  config.vm.box = "hashicorp/precise64"
```

Es en esta línea donde le indicamos a Vagrant qué sistema operativo vamos a cargar en la máquina virtual que más tarde lanzaremos.
Observamos aquí una de las principales ventajas de Vagrant, y es que podemos acceder a un servicio llamado Vagrant Cloud[xxviii] con cientos de máquinas virtuales ya preparadas para ser descargadas automáticamente y lanzadas con una única instrucción, con lo que nos ahorramos el tedioso proceso de instalación de cada sistema operativo.



Las máquinas virtuales que encontramos en Vagrant Cloud son subidas por empresas (Hashicorp, Canonical), pero también por individuos. Así por ejemplo, podríamos crear una máquina virtual con un servicio que hubiéramos desarrollado y ponerla a disposición de cualquier usuario. Sin embargo, hay que tener en cuenta que las máquinas serán públicas, por lo que no conviene subirlas si contienen configuraciones privadas. En el caso de que quisiéramos guardar un repositorio de máquinas virtuales privadas, Hashicorp ofrece un servicio de pago para ello.

2.2.3 Definiendo parámetros de Hardware de la máquina

Como hemos dicho, el archivo Vagrantfile nos sirve para configurar diversos aspectos de nuestra máquina virtual, incluso a nivel de Hardware. Así por ejemplo, y a modo de referencia, pues no será necesario para el desarrollo de nuestro proyecto, podríamos añadir las siguientes líneas para especificar la memoria RAM o el número de procesadores que destinaremos a la virtualización:

```
config.vm.provider :virtualbox do |vb|
   # Use VBoxManage to customize the VM. For example to change memory:
  vb.customize ["modifyvm", :id, "--memory", "2048"]
  vb.customize ["modifyvm", :id, "--cpus", 4]
end
```

2.2.4 Levantando la máquina

Una vez inicializada nuestra máquina, la segunda y última instrucción que necesitamos para lanzarla es *vagrant up*.

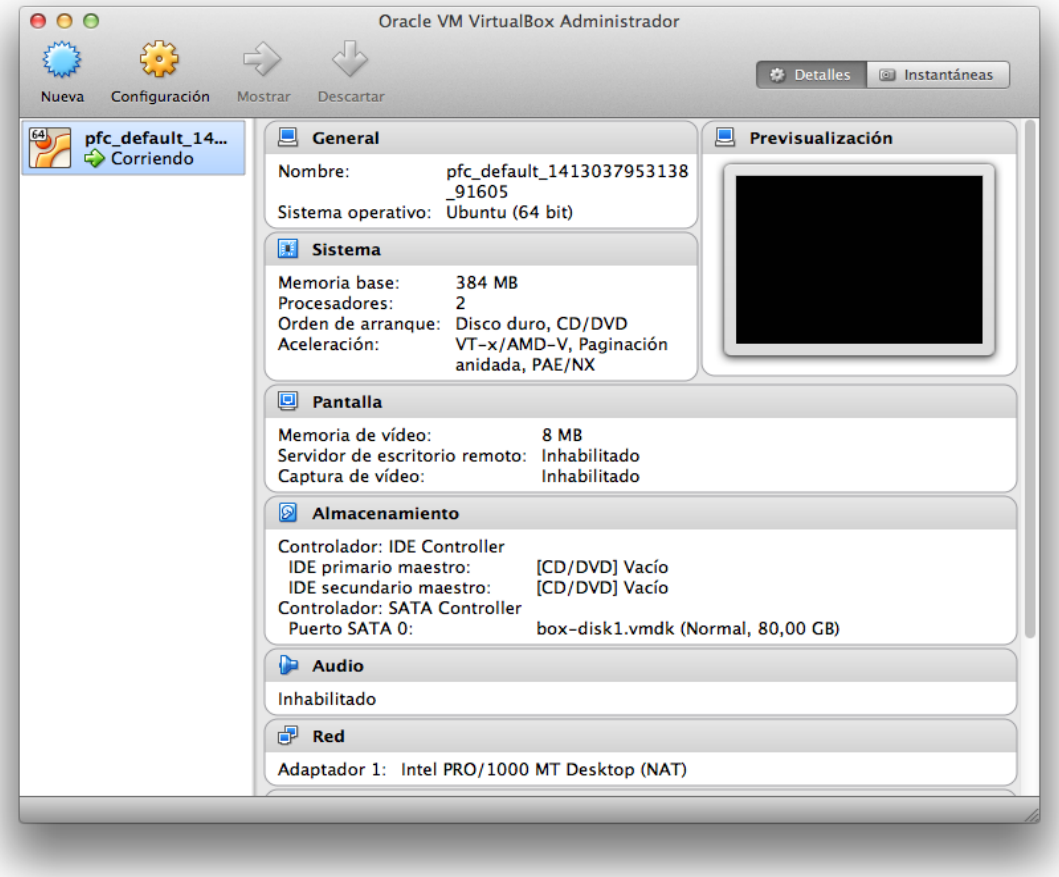
```
$ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
```

```
==> default: Box 'hashicorp/precise64' could not be found. Attempting to find
and install...
```

```
    default: Box Provider: virtualbox
```

```
    default: Box Version: >= 0
==> default: Loading metadata for box 'hashicorp/precise64'
    default: URL: https://vagrantcloud.com/hashicorp/precise64
==> default: Adding box 'hashicorp/precise64' (v1.1.0) for provider:
virtualbox
    default: Downloading:
https://vagrantcloud.com/hashicorp/precise64/version/2/provider/virtualbox.bo
x
==> default: Successfully added box 'hashicorp/precise64' (v1.1.0) for
'virtualbox'!
==> default: Importing base box 'hashicorp/precise64'...
==> default: Matching MAC address for NAT networking...
==> default: Checking if box 'hashicorp/precise64' is up to date...
==> default: Setting the name of the VM: tuto_default_1396617997922_90950
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
==> default: Forwarding ports...
    default: 22 => 2222 (adapter 1)
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
    default: SSH address: 127.0.0.1:2222
    default: SSH username: vagrant
    default: SSH auth method: private key
==> default: Machine booted and ready!
==> default: Checking for guest additions in VM...
    default: The guest additions on this VM do not match the installed
version of
    default: VirtualBox! In most cases this is fine, but in rare cases it can
    default: prevent things such as shared folders from working properly. If
you see
    default: shared folder errors, please make sure the guest additions
within the
    default: virtual machine match the version of VirtualBox you have
installed on
    default: your host and reload your VM.
    default:
    default: Guest Additions Version: 4.2.0
    default: VirtualBox Version: 4.3
==> default: Mounting shared folders...
    default: /vagrant => /home/simon/dev/pfc
```

Si abrimos el interfaz de VirtualBox podremos observar como se ha creado una nueva máquina virtual y aparece en estado de ejecución:



Incluso podemos entrar por *ssh* con la instrucción *`vagrant ssh`*:

```
$ vagrant ssh
Welcome to Ubuntu 12.04 LTS (GNU/Linux 3.2.0-23-generic x86_64)

Welcome to your Vagrant-built virtual machine.
Last login: Fri Sep 14 06:23:18 2012 from 10.0.2.2
```

2.2.5 Dos soluciones y un problema

Recapitulando, Vagrant nos ha permite descargar y lanzar una máquina virtual, con tan sólo dos comandos:

```
$ vagrant init hashicorp/precise64
$ vagrant up
```

Si bien estas dos órdenes, serán los que más utilicemos para gestionar nuestras máquinas virtuales, Vagrant dispone de otra serie de comandos los cuales podremos observar ejecutando *`vagrant help`*:

```
$ vagrant help

Usage: vagrant [options] <command> [<args>]

    -v, --version                    Print the version and exit.
    -h, --help                       Print this help.

Common commands:
     box          manages boxes: installation, removal, etc.
     connect      connect to a remotely shared Vagrant environment
     destroy      stops and deletes all traces of the vagrant machine
     halt         stops the vagrant machine
     help         shows the help for a subcommand
     init         initializes a new environment creating a Vagrantfile
     login        log in to Vagrant Cloud
     package      packages a running vagrant environment into a box
     plugin       manages plugins: install, uninstall, update, etc.
     provision    provisions the vagrant machine
     reload       restarts vagrant machine
     resume       resume a suspended vagrant machine
     share        share your environment with anyone in the world
     ssh          connects to machine via SSH
     ssh-config   outputs OpenSSH conf to connect to the machine
     status       outputs status of the vagrant machine
     suspend      suspends the machine
     up           starts and provisions the vagrant environment

For help on any individual command run `vagrant COMMAND -h`

Additional subcommands are available, but are either more advanced
or not commonly used. To see all subcommands, run the command
`vagrant list-commands`.
```

Si recordamos las conclusiones sobre VirtualBox del punto anterior del PFC, teníamos tres problemas fundamentales a la hora de utilizarlo. Veamos a continuación como Vagrant nos ayuda a solventarlos.

1. Problema de distribución de las máquinas virtuales
2. Problema de documentación del proceso
3. Problema de actualización de máquinas

En primer lugar, al ejecutar *`vagrant init`*, se ha creado un archivo *Vagrantfile* con la configuración de nuestra máquina virtual. Este archivo contiene la configuración de la misma, y el hecho de que sea un archivo de texto plano lo convierte en un candidato ideal para incorporarse al repositorio de nuestra aplicación.

De esta forma, el administrador de sistemas en el caso de que necesitase modificar un parámetro de configuración de la máquina virtual como la memoria *ram* o el número de *cpus*, tan sólo debería modificar este archivo y registrarlo en el sistema de control de versiones en el que gestionemos nuestro desarrollo, el cual estará por defecto, a disposición de cualquier miembro del equipo.

El problema de la distribución se soluciona en el momento en el que no hay una máquina ya creada que el administrador de sistemas tiene que pasar a cada desarrollador, sino que la máquina virtual se crea en el ordenador de cada programador, en base al archivo *Vagrantfile* obtenido del repositorio de la aplicación. Un usuario recién llegado, tan sólo debería descargar el código del proyecto y ejecutar `vagrant up` para levantar la máquina virtual asociada en su equipo.

El último problema, el de la actualización de máquinas lo tenemos parcialmente resuelto en el caso de parámetros físicos como hemos visto anteriormente. Decimos parcialmente, porque la configuración física es sólo una parte de lo que necesita una aplicación web para funcionar, siendo igualmente importantes los servicios que se ejecuten en la máquina virtual, como un servidor web o un motor de base de datos. Y es para eso que necesitaremos de la próxima herramienta, Ansible.

2.3 Ansible

Como hemos comentado, durante años, sino décadas, los servidores han sido controlados de forma casi absoluta por los administradores de sistemas, quienes en muchos casos atesoraban sus conocimientos de tal forma que se convertían en insustituibles para la empresa.

Un *sysadmin* gestionaba los servidores de los que era responsable, en el mejor de los casos, a través de uno o varios scripts propios que ejecutados en un orden concreto dejaban un servidor en el estado deseado. No era difícil ver también cómo después de ejecutar uno de estos scripts, llegaba el turno de aplicar ajustes de configuración manuales que no quedaban documentados más que en la cabeza del administrador.

Los tiempos cambiaron con la llegada de la nube y las aplicaciones que se ejecutaban sobre servidores virtuales, y es que si anteriormente un administrador de sistemas solía tener control directo, e incluso acceso físico a las máquinas que configuraba, con la nube todo adquirió unas proporciones que ya no eran manejables de la forma tradicional en la que nuestros *sysadmins* estaban acostumbrados a trabajar.

Es en ese entorno donde surgen los primeros sistemas de aprovisionamiento de servidores, como Puppet[xxix] y Chef[xxx], los cuales nacen con el objetivo de permitirnos configurar y controlar cientos de máquinas remotas a partir de una única configuración local.

El funcionamiento era relativamente sencillo. En lugar de configurar cada máquina individualmente, el administrador de sistemas tan sólo debía instalar un agente en los servidores a gestionar, el cual se comunicaba a su vez con un servidor central o master, quién le indicaba al agente las acciones a realizar sobre la máquina. Así por ejemplo, nuestro *sysadmin* le podía decir a master:

> "Instala apache en TODOS los servidores del datacenter UPV-12"

Y nuestro sistema de aprovisionamiento se encargaría de realizarlo automáticamente, obteniendo algunas ventajas en el camino:

- Definición de la arquitectura de la aplicación en código. Nuestro sistema de aprovisionamiento necesita saber qué servidores y de qué tipo la soportan. Esto obliga a que haya uno o varios archivos en el repositorio que la defina. En otras palabras, obtenemos documentación automática y versionada de la arquitectura de la misma a disposición de todos los miembros del equipo.

- Los sistemas de aprovisionamiento actuales son idempotentes[xxxi]. Esto quiere decir que su resultado tras ejecutarse una y otra vez siempre será el mismo independientemente del estado de la máquina destino. Esto no es tarea sencilla de conseguir con los tradicionales scripts de configuración creados por cada *sysadmin*.

- Los sistemas de aprovisionamiento están preparados para trabajar con plantillas y variables que definan grupos de servidores (distintos *datacenters*, distintos tipos de servidores como web, base de datos, *proxy*), lo que facilita la gestión de distintas máquinas a partir de una misma configuración.

- Son capaces de ejecutarse paralelamente y simultáneamente sobre varias máquinas (decenas o incluso cientos de ellas).

Creemos que queda clara la aportación de los sistemas de aprovisionamiento al mundo de la gestión y configuración de sistemas remotos, y como hemos comentado anteriormente, hasta hace poco teníamos principalmente dos alternativas a la hora de decantarnos por uno de estos sistemas, Chef y Puppet.

Si bien son alternativas perfectamente válidas y muy utilizadas en la actualidad, tienen a nuestro juicio un problema de base y es la arquitectura de agente - master. Y es que continúa siendo un trabajo manual del administrador de sistemas tener que instalar los agentes en cada máquina destino y configurarlos para que se comuniquen con nuestro master.

Esta configuración manual consiste básicamente en abrir un túnel *ssh* con la máquina destino para instalar el agente. Pero, si por defecto el primer paso es abrir una conexión *ssh*, ¿para qué necesitamos instalar un agente? ¿No podríamos simplemente utilizar esa conexión para efectuar los cambios de configuración necesarios?

Aquí es dónde entra Ansible[xxxii] en juego. Según la definición de la Wikipedia:

> *Ansible is an open-source software platform for configuring and managing computers. It combines multi-node software deployment, ad hoc task execution, and configuration management. It manages nodes over SSH and does not require any additional remote software (except Python 2.4 or later) to be installed on them. Modules work over JSON and standard output and can be written in any programming language. The system uses YAML to express reusable descriptions of systems.*

Básicamente Ansible es un sistema de aprovisionamiento *agentless*, o no basado en agente. Esto permite eliminar de la ecuación la instalación y gestión de los agentes en las máquinas remotas junto a la dependencia de una máquina master, facilitando enormemente el mantenimiento de la arquitectura.

Ansible será pues, la herramienta que utilizaremos para configurar las máquinas virtuales VirtualBox que creemos vía Vagrant. Veamos cómo.

2.3.1 Primeros pasos con Ansible

Instalar Ansible dependerá del equipo en el que nos encontremos, pero como norma general es un proceso bastante sencillo, siempre que trabajemos en entornos Linux/Unix, ya que Ansible conectará con los sistemas a gestionar a través de *ssh*.

Es por esto que a día de hoy, Ansible no dispone de una distribución Windows, si bien se puede ejecutar desde una máquina virtual Linux cargada desde el sistema operativo de Microsoft. Sí es posible, desde la versión 1.7 de Ansible, desplegar y configurar máquinas Windows[xxxiii] a través de Ansible.

Por ejemplo, instalar ansible[xxxiv] en OSX utilizando Brew[xxxv] es tan sencillo como ejecutar:

```
$ brew update
$ brew install ansible
```

Una vez instalado, ya podemos comenzar a utilizarlo, pero para ello, necesitaremos dedicar unos minutos a su configuración en nuestro proyecto.

La única configuración que Ansible necesita para funcionar a nivel básico es la creación de un archivo *hosts* el cual define las máquinas objetivo a controlar. Este archivo puede estar en cualquier sitio dentro de la estructura de nuestro proyecto. En nuestro caso, crearemos un directorio *ansible* dónde alojarlo, y lo editaremos para añadir el siguiente contenido:

```
# file: hosts

[pfc-vm]
192.168.33.10
```

Básicamente añadimos tan sólo un host con la dirección *ip 192.168.33.30* bajo el tag *[pfc-vm]*. Podemos observar que la dirección es una dirección de red local que en principio no hemos asignado a nuestra máquina virtual, por lo que también tenemos que editar nuestro *Vagrantfile* para que asigne esta ip fija cada vez que lance la máquina virtual.

Tan sencillo como editar *Vagrantfile* y añadir:

```
config.vm.network :private_network, ip: "192.168.33.30"
```

Necesitaremos recargar la máquina virtual con *vagrant reload* para comprobar que la dirección ip se aplica correctamente:

```
$ vagrant reload
==> default: Attempting graceful shutdown of VM...
==> default: Checking if box 'hashicorp/precise64' is up to date...
==> default: Clearing any previously set forwarded ports...
```

```
==> default: Clearing any previously set network interfaces...
```

```
==> default: Preparing network interfaces based on configuration...
```

```
    default: Adapter 1: nat
    default: Adapter 2: hostonly
==> default: Forwarding ports...
    default: 22 => 2222 (adapter 1)
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
    default: SSH address: 127.0.0.1:2222
    default: SSH username: vagrant
    default: SSH auth method: private key
    default: Warning: Connection timeout. Retrying...
==> default: Machine booted and ready!
GuestAdditions 4.3.14 running --- OK.
==> default: Checking for guest additions in VM...
==> default: Configuring and enabling network interfaces...
==> default: Mounting shared folders...
    default: /vagrant => /Users/simon/dev/pfc
==> default: VM already provisioned. Run `vagrant provision` or use `--
provision` to force it
```

Podemos comprobar que la ip es la correcta, entrando en la máquina y viendo su configuración de red:

```
$ vagrant ssh
Welcome to Ubuntu 12.04 LTS (GNU/Linux 3.2.0-23-generic x86_64)

Welcome to your Vagrant-built virtual machine.
Last login: Fri Sep 14 06:23:18 2012 from 10.0.2.2
vagrant@precise64:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:88:0c:a6
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe88:ca6/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:445 errors:0 dropped:0 overruns:0 frame:0
          TX packets:314 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:50114 (50.1 KB)  TX bytes:40153 (40.1 KB)

eth1      Link encap:Ethernet  HWaddr 08:00:27:d7:88:10
          inet addr:192.168.33.30  Bcast:192.168.33.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fed7:8810/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:5 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:542 (542.0 B)  TX bytes:468 (468.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
```

```
            RX packets:0 errors:0 dropped:0 overruns:0 frame:0
            TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:0
```

```
            RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

2.3.2 Ejecutando comandos vía ansible

Una vez configurada nuestra máquina virtual, vamos a tratar de hacer un ping[xxxvi] de Ansible para comprobar que podemos acceder a ella. Un ping no es más que un intento de conexión por parte de nuestro gestor de aprovisionamiento:

```
$ ansible -i ansible/hosts all -m ping
The authenticity of host '192.168.33.30 (192.168.33.30)' can't be
established.
RSA key fingerprint is 50:db:75:ba:11:2f:43:c9:ab:14:40:6d:7f:a1:ee:e3.
Are you sure you want to continue connecting (yes/no)? yes
192.168.33.30 | FAILED => SSH encountered an unknown error during the
connection. We recommend you re-run the command using -vvvv, which will
enable SSH debugging output to help diagnose the issue
```

Nos encontramos con un error común a la hora de conectar con nuestro primer servidor gestionado por Ansible, y es que por defecto, utilizará conexiones *ssh* identificadas por el método de clave pública / privada. Esto quiere decir que para poder acceder a la máquina virtual con Ansible, la clave pública de nuestro usuario debe estar autorizada en la máquina virtual.

Afortunadamente podemos apoyarnos en Vagrant, y es que esta herramienta ya autoriza a su usuario con su propio par de claves, por lo que podemos apropiarnos de su clave privada para lanzar el ping:

```
$ ansible -i ansible/hosts -u vagrant --private-
key=~/.vagrant.d/insecure_private_key all -m ping
192.168.33.30 | success >> {
    "changed": false,
    "ping": "pong"
}
```

Como podemos observar, es el mismo comando que anteriormente donde únicamente le hemos añadido dos parámetros, *-u vagrant* para decirle a Ansible que establezca la conexión con ese usuario, indicándole a su vez una clave privada con el parámetro *--private-key*.

Es llamativo, y debe hacernos pensar, el nombre de la clave, *insecure_private_key*, y es que es la clave privada que por defecto asigna utiliza Vagrant para acceder a

cualquier máquina virtual generada con *`vagrant up`*. Se hace así para facilitar su uso a desarrolladores pero puede ser un riesgo de seguridad en ciertos casos. Si fuera el caso, tan sólo tendríamos que borrar la clave pública del archivo */home/vagrant/.ssh/authorized_keys* de la máquina virtual, pero sólo tras haber autorizado nuestra propia cuenta de usuario o corremos el riesgo de perder la posibilidad de acceder a la máquina.

Una vez hemos testado la conexión, Ansible nos proporciona un interfaz anfitrión / máquina virtual capaz de ejecutar cualquier comando que pudiéramos lanzar entrando por *ssh* a la máquina. Podremos por ejemplo, imprimir un "hola mundo" en el terminal:

```
$ ansible -i ansible/hosts --private-key=~/.vagrant.d/insecure_private_key -u
vagrant all -a "/bin/echo hola mundo"
192.168.33.30 | success | rc=0 >>
hola mundo
```

Lo cual, no es que sea demasiado útil, pero imaginemos que necesitamos hacer algo tan crítico reiniciar un sistema remoto:

```
$ ansible -i ansible/hosts --private-key=~/.vagrant.d/insecure_private_key -u
vagrant all -a "sudo /sbin/reboot"
sudo password:
192.168.33.30 | success | rc=0 >>
```

En estos dos ejemplos hemos utilizado un parámetro nuevo, '-a', el cual nos sirve para indicarle a Ansible que vamos a ejecutar un comando[xxxvii] en lugar de llamar a un módulo propio de Ansible, como el módulo ping en el ejemplo anterior.

Ahora imaginemos que en nuestro fichero hosts de Ansible, no tenemos tan sólo un host definido, sino cientos de ellos. Con una sola instrucción en nuestro terminal podríamos, por ejemplo, ejecutar cualquier comando o serie de comandos sobre todas las máquinas de uno o varios *datacenters*. Quizás, incluso de toda nuestra infraestructura fuera del tamaño que fuera.

Hasta el momento hemos aprendido como ejecutar comandos individuales en uno o varios servidores utilizando Ansible. A continuación, introduciremos el concepto de los playbooks[xxxviii], que nos permitirán definir conjuntos de acciones a realizar sobre una máquina objetivo.

2.3.3 Playbooks

Los *playbooks*, o libros de jugadas, se componen de conjuntos de acciones o *plays*, que se ejecutan de forma secuencial sobre una máquina destino. El objetivo de una jugada es reproducir en uno o varios hosts destino, un conjunto de tareas, *tasks*, que lo dejarán

en el estado deseado. Una tarea, no es nada más que una llamada a un módulo de Ansible.

En el índice de módulos de Ansible[xxxix], encontramos módulos para casi cualquier acción relacionada con la gestión de un servidor, como pueden ser:

- copy[xl] - copia de archivos a hosts remotos
- apt[xli] - para la gestión de paquetes .deb
- service[xlii] - para la gestión de servicios

Un playbook se codificará en formato YAML[xliii], un formato de datos serializados especialmente diseñado para ser entendible por el ser humano, lo que facilita tanto la definición de tareas, como su lectura posterior.

Un *playbook*, por tanto, no será más que un conjunto de tareas definidas en un lenguaje *human friendly* que definirán una jugada. Probablemente será más sencillo verlo con un ejemplo:

```
---
- hosts: pfc-vm
  tasks:
  - name: Installing nginx
    apt: name=nginx state=latest
  - name: ensure nginx is running
    service: name=nginx state=started
```

En este ejemplo, nuestro *playbook* actuaría sobre los hosts de tipo *pfc-vm*, instalando el servidor web nginx en su última versión e iniciándolo posteriormente.

Para ejecutar este *playbook* sobre nuestra máquina virtual, tan sólo deberíamos crear un archivo al que llamaremos por ejemplo *playbook-nginx.yml*, y ejecutarlo con el comando *`ansible-playbook`*:

```
$ ansible-playbook playbook-nginx.yml -i hosts -u vagrant --private-
key=~/.vagrant.d/insecure_private_key -s

PLAY [pfc-vm]
*********************************************************************

GATHERING FACTS
*********************************************************************
ok: [192.168.33.30]

TASK: [ensure nginx is at the latest version]
*********************************
changed: [192.168.33.30]

TASK: [ensure nginx is running]
***********************************************
```

```
changed: [192.168.33.30]

PLAY                                                                    RECAP
********************************************************************
192.168.33.30              : ok=3    changed=2    unreachable=0    failed=0
```

El output de nuestro comando no tiene mayor misterio. En primer lugar vemos como hay una fase llamada de recogida de información, *gathering facts*, en la cual Ansible recopila datos sobre el host sobre el cual se va a ejecutar. Posteriormente, pasamos a las tareas de nuestro *playbook*, las cuales se ejecutan secuencialmente, instalando nginx y lanzándolo en nuestra máquina virtual.

Comprobemos que realmente es así entrando por *ssh*:

```
$ vagrant ssh
Welcome to Ubuntu 12.04 LTS (GNU/Linux 3.2.0-23-generic x86_64)

Welcome to your Vagrant-built virtual machine.
Last login: Sun Oct 26 10:58:57 2014 from 192.168.33.1

vagrant@precise64:~$ nginx -v
nginx version: nginx/1.1.19

vagrant@precise64:~$ service nginx status
 * nginx is running
```

Perfecto, ya hemos escrito y probado nuestro primer libro de jugadas, el cual será la base sobre la que posteriormente diseñemos nuestra arquitectura, que al estar definida de esta forma, podremos añadir fácilmente al sistema de control de versiones de cualquier proyecto.

2.3.4 Ansible como método de aprovisionamiento en Vagrant

En la introducción de este capítulo hemos observado algunas de las ventajas que los sistemas de aprovisionamiento, si bien no hemos entrado en la definición concreta del término. Citando la sección de la Wikipedia sobre aprovisionamiento de servidores[xliv]:

> *Server provisioning is a set of actions to prepare a server with appropriate systems, data and software, and make it ready for network operation. Typical tasks when provisioning a server are: select a server from a pool of available servers, load the appropriate software (operating system, device drivers, middleware, and applications), appropriately customize and configure the system and the software to create or change a boot image for this server, and then change its parameters, such as IP address, IP Gateway to find associated network and storage resources (sometimes separated as resource provisioning) to audit the system. By auditing the system, you ensure OVAL compliance with limit vulnerability, ensure compliance, or*

> *install patches. After these actions, you restart the system and load the new software. This makes the system ready for operation. Typically an internet service provider (ISP) or Network Operations Center will perform these tasks to a well-defined set of parameters, for example, a boot image that the organization has approved and which uses software it has license to. Many instances of such a boot image create a virtual dedicated host.*

Así pues, el objetivo de un sistema de aprovisionamiento como Ansible o cualquier otro será permitirnos configurar uno o varios sistemas objetivo de acuerdo a los requisitos de una organización.

En nuestro caso, utilizaremos Ansible para configurar la máquina virtual sobre la que ejecutaremos nuestra aplicación, y es que Vagrant está preparado para trabajar por defecto con varios de estos sistemas, Ansible incluído.

Esto nos permitirá poder especificar en nuestro fichero *Vagrantfile* la configuración necesaria para que Ansible se ejecute a través de Vagrant, como por ejemplo, al levantar la máquina virtual con *`vagrant up`*.

Veamos una configuración típica:

```
config.vm.provision "ansible" do |ansible|
  ansible.playbook = "ansible/playbook-nginx.yml"
  ansible.inventory_path = "ansible/hosts"
  ansible.limit = "pfc-vm"
  ansible.sudo = true
end
```

Dónde:

- *ansible.playbook* define el *playbook* que cargará Vagrant
- *ansible.inventory_path* indica el *path* dónde se encuentra nuestro archivo de *hosts*
- *ansible.limit* indica el nombre del host sobre el que queremos ejecutar el aprovisionamiento

Una vez configurado nuestro archivo Vagrantfile, ejecutar el aprovisionamiento será tan sencillo como ejecutar *`vagrant provision`*:

```
$ vagrant provision
==> default: Running provisioner: ansible...

PLAY                                                              [pfc-vm]
*********************************************************************

GATHERING                                                           FACTS
*********************************************************************
```

```
ok: [192.168.33.30]

TASK: [ensure nginx is at the latest version]
*********************************
ok: [192.168.33.30]

TASK: [ensure nginx is running]
*********************************************
```

```
ok: [192.168.33.30]
```

```
PLAY RECAP
*********************************************************************
192.168.33.30              : ok=3    changed=0    unreachable=0    failed=0
```

De esta forma, nos evitamos la molestia de tener que invocar el comando *ansible-playbook* facilitando así la gestión de nuestros entornos virtuales.

Por último, si comparamos el output de `vagrant provision` sobre el output de la primera invocación a `ansible-playbook`, podemos observar que en esta ocasión no se han efectuado cambios en el sistema (changed=0). Esto es así porque Ansible detecta que nginx ya estaba instalado y en ejecución, por lo que simplemente se salta esos pasos. Es por esto que, como señalábamos anteriormente, Ansible es idempotente. Distintas ejecuciones de `ansible-playbook` o `vagrant provision` dejarán el estado del host en el mismo estado sea cual sea la situación de partida.

2.3.5 Ansible, o la solución a nuestros problemas

A lo largo del proyecto hemos ido detectando diversos problemas que impedían nuestro objetivo de proporcionar un entorno virtual adecuado para desarrollar una aplicación, de forma automática e independiente, a la arquitectura de la máquina del desarrollador.

En primer lugar introducimos VirtualBox, el cual nos proveía de la capacidad de crear máquinas virtuales a través de un interfaz, pero su gestión y distribución traía varios problemas asociados.

Vagrant por su parte, nos permitía solventar la mayoría de problemas detectados con VirtualBox, como es la configuración automática de los parámetros físicos de la máquina virtual, la definición de en código de sus parámetros básicos, o la distribución de las mismas, si bien no nos permitía solucionar el último de los problemas.

Ese problema no era sino conseguir gestionar las dependencias de paquetes y configuración de nuestra aplicación, es decir, sus requisitos de software para que se pudiera ejecutar sobre nuestra máquina virtual.

Justo este problema el que Ansible nos permitirá solventar. Y es que, pese a haber tan sólo introducido el concepto de *playbook*, ya podemos anticipar la potencia que nos da para poder definir las dependencias completas de cualquier desarrollo vía código.

Algo que veremos en la siguiente fase del proyecto, en la que uno de nuestros objetivos será crear los *playbooks* necesarios para instalar y ejecutar una aplicación web PHP, que requiera a su vez de una conexión a base de datos.

3 Creando una infraestructura para instalar una aplicación PHP

En la primera fase del proyecto, hemos introducido las herramientas que utilizaremos para desarrollar la segunda etapa que ahora comenzamos, la cual dedicaremos a poner en práctica las mismas.

Nuestro objetivo será automatizar vía código la descarga, instalación y configuración de un proyecto PHP en un entorno virtualizado. La idea es que un desarrollador cualquiera, pudiera lanzar uno de estos entornos con tan sólo ejecutar un comando, `vagrant up`.

La aplicación que hemos escogido como ejemplo es Drupal[xlv], uno de los gestores de contenido o CMS[xlvi] más populares, cuyos requisitos son los habituales en este tipo de aplicaciones:

- Un servidor web sobre el que ejecutarse (nginx)
- Un servidor de base de datos (MySQL)
- Un intérprete de PHP (PHP5)

A lo largo de esta fase, utilizaremos un repositorio en GitHub[xlvii] para realizar el seguimiento de los distintos pasos que completemos. Éste se convertirá en el repositorio del código que podrán compartir los distintos miembros del equipo.

3.1 Creando un repositorio en GitHub

Hemos dicho que nuestra intención es utilizar GitHub como repositorio del código de nuestro proyecto. GitHub es un servicio web para la gestión de repositorios Git[xlviii], creado en 2007 por Tom Preston-Werner, Chris Wanstrath, y PJ Hyett.

GitHub destaca en dos funciones principales. Por un parte, sirve de plataforma de *hosting* para repositorios Git, lo que evita tener que configurar una máquina local para que actúe como servidor. Por otra, incorpora un interfaz web sobre el repositorio, con múltiples opciones de visualización, control de cambios, etc., que potencian en gran medida las posibilidades de colaboración sobre un proyecto.

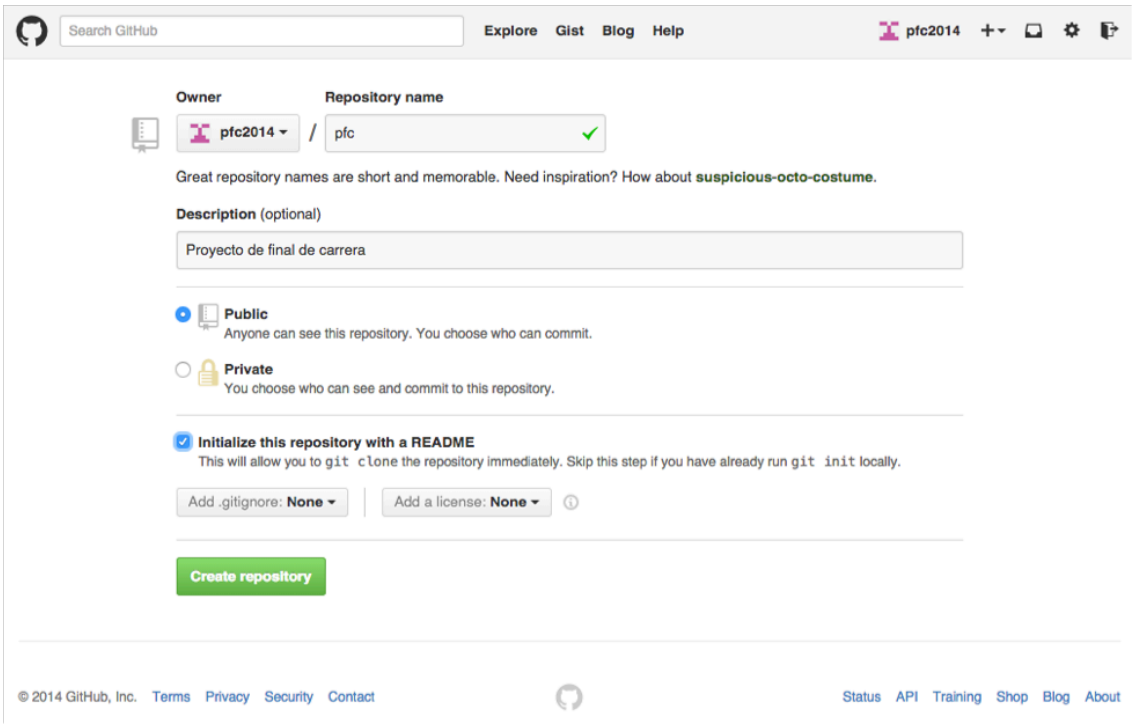
Una de las características más interesantes de GitHub y que nos sirve para introducir el flujo de trabajo más usual en equipos de desarrollo de todo el mundo, es la posibilidad de solicitar Pull Requests[xlix]. Un *pull request* es una petición / notificación al resto de miembros del equipo de que queremos introducir cambios en el código de un repositorio.

Así por ejemplo, un flujo de trabajo usual con un repositorio en GitHub incluiría estos pasos:

1. Un desarrollador se baja una copia del repositorio con la idea de añadir una funcionalidad o corregir un fallo.
2. El desarrollador crea una nueva rama del repositorio, que no es sino una copia del código sobre la que trabajar sin afectar al código principal.
3. Cuando el desarrollador termina su trabajo, sube su rama a GitHub haciendo *push* y solicita un *pull request* sobre la rama principal (*master*).
4. Los responsables del proyecto reciben una notificación avisando de la petición de fusionar esa nueva rama sobre *master*. Revisan los cambios, discuten con el desarrollador en caso de problemas, y cuando dan el visto nuevo fusionan el código, introduciendo los cambios propuestos por el desarrollador.

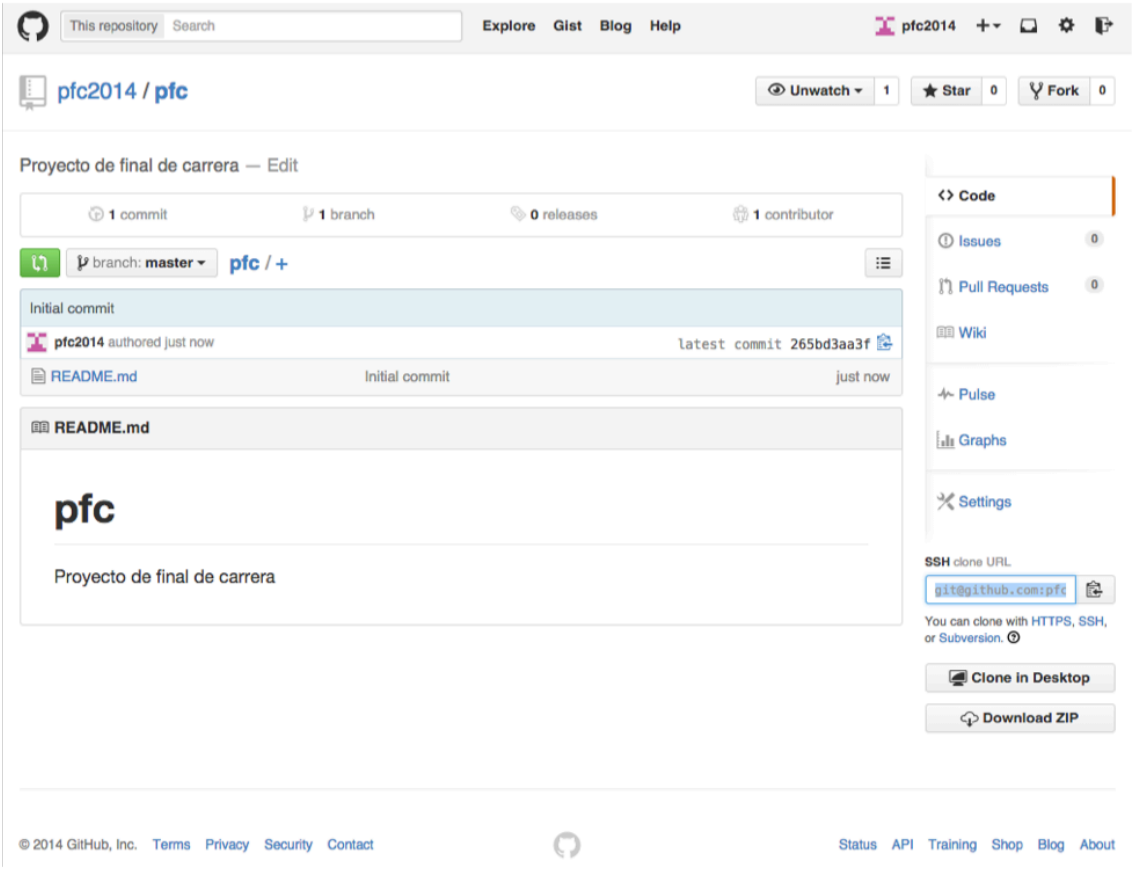
No es extraño que GitHub se haya convertido, de facto, en el servicio que alberga mayor número de proyectos de software libre de la red, como por ejemplo algunas de las herramientas que vamos a utilizar como Vagrant[l], Ansible[li], o proyectos tan de moda últimamente como Bitcoin[lii]. También ayuda el hecho de que GitHub sea gratuito para cualquier proyecto de código libre. En realidad es así mientras el proyecto utilice repositorios públicos, característica habitual de los proyectos *open source*.

Crear un repositorio en GitHub es tan sencillo como registrar una nueva cuenta en su web, y dar de alta el mismo el siguiente interfaz:



Seleccionar la opción *Initialize this repository with a README* es aconsejable, pues nos evitará tener que completar un paso adicional desde nuestro ordenador local para crear el repositorio en GitHub.

Una vez introducido el nombre del repositorio, y marcada la opción de inicializar el repositorio, podemos pulsar el botón de crear el cual nos llevará a la siguiente pantalla:



Este será el interfaz de nuestro repositorio que, como vemos, de momento tan sólo alberga un archivo, *README.md*, generado automáticamente y cuyo contenido podemos ver en la misma página. Esto es así porque GitHub muestra por defecto el contenido de este fichero a modo de portada en cada proyecto.

Una vez creado el repositorio, para poder trabajar sobre él, necesitamos hacer *checkout* del mismo a nuestro ordenador local. Nada tan sencillo como copiar la *Clone URL* que podemos observar en la columna derecha de la portada del repositorio. En este caso, *git@github.com:pfc2014/pfc.git*.

Ejecutando `git clone` de esa url en nuestro terminal, descargaremos el repositorio el cual quedará automáticamente configurado para que podamos ejecutar comandos git como `git commit` y `git push` a GitHub sin ningún problema:

```
# simon at MacBook-Pro-de-Simon-2.local in ~/dev on git:master x [11:14:28]
$ git clone git@github.com:pfc2014/pfc.git
Cloning into 'pfc'...
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (3/3), done.
Checking connectivity... done.

# simon at MacBook-Pro-de-Simon-2.local in ~/dev on git:master x [11:14:34]
$ cd pfc
```

```
#  simon  at  MacBook-Pro-de-Simon-2.local  in  ~/dev/pfc  on  git:master  o
[11:14:42]
$ ls -la
total 8
drwxr-xr-x   4 simon  staff  136  1 nov 11:14 .
drwxr-x---  15 simon  staff  510  1 nov 11:14 ..
drwxr-xr-x  13 simon  staff  442  1 nov 11:14 .git
-rw-r--r--   1 simon  staff   38  1 nov 11:14 README.md
```

Como vemos, la orden *git clone* ha descargado el repositorio recién creado a un directorio *pfc* en nuestro ordenador local. Hemos hecho un listado de archivos ocultos, y vemos que también git ha creado un directorio *.git*. Este no es más que el directorio donde guarda su configuración propia, como por ejemplo, la dirección del repositorio de origen.

Tan sólo un comentario más antes de pasar al siguiente punto, y es que nos hemos saltado un paso intencionalmente, y ha sido la configuración de GitHub para que utilice una clave pública/privada en nuestras comunicaciones, lo cual nos ha permitido ejecutar la orden git clone sin necesidad de introducir el usuario y contraseña de GitHub. Este proceso incluiría:

- Generar un par de claves pública / privadas en nuestro ordenador / cuenta de usuario (en el caso de que no las tuviéramos ya creadas en el directorio).
- Autorizar nuestra clave privada copiando la clave pública recién generada en GitHub.

El proceso detallado paso a paso lo podemos encontrar en las instrucciones[liii] que GitHub pone a nuestra disposición.

3.2 Creando nuestra máquina virtual con Vagrant

Una vez creado el repositorio en el cual guardaremos la evolución de nuestro proyecto, el siguiente paso será utilizar Vagrant para inicializar el archivo *Vagrantfile* que servirá como base la creación de la máquina virtual asociada a este proyecto.

Como ya hicimos en la fase anterior del proyecto, tan sólo necesitaremos decirle a Vagrant que se inicie con el tipo de máquina que nos servirá como base, en este caso *hashicorp/precise64*:

```
$ vagrant init hashicorp/precise64
A `Vagrantfile` has been placed in this directory. You are now
ready to `vagrant up` your first virtual environment! Please read
the comments in the Vagrantfile as well as documentation on
`vagrantup.com` for more information on using Vagrant.
```

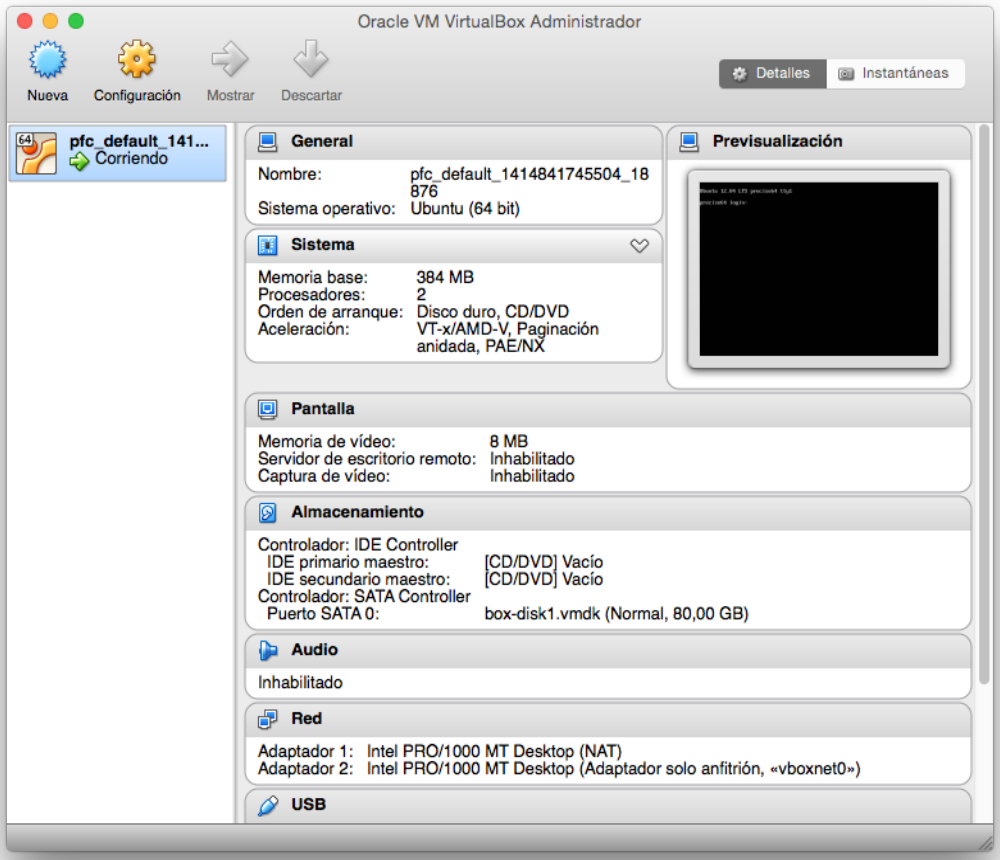
Por claridad, vamos a editar nuestro fichero *Vagrantfile* y a dejar únicamente las líneas de configuración que nos interesen (recordemos que el fichero generado por defecto es una plantilla con múltiples líneas de configuración comentadas). Nuestro *Vagrantfile* quedaría de la siguiente manera:

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|

  config.vm.box = "hashicorp/precise64"
  config.vm.network "private_network", ip: "192.168.33.10"

end
```

Además de borrar la mayor parte de los comentarios, el único cambio adicional que hemos hecho ha sido añadir la línea *config.vm.network*, que define la ip que utilizará nuestra máquina virtual.

Tras este paso, podemos ejecutar *vagrant up* para lanzar la máquina virtual y comprobar que se está ejecutando bien entrando por *ssh* o bien a través del interfaz de VirtualBox.

```
$ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Importing base box 'hashicorp/precise64'...
==> default: Matching MAC address for NAT networking...
==> default: Checking if box 'hashicorp/precise64' is up to date...
==> default: Setting the name of the VM: pfc_default_1414841745504_18876
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
    default: Adapter 2: hostonly
==> default: Forwarding ports...
    default: 22 => 2222 (adapter 1)
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
    default: SSH address: 127.0.0.1:2222
    default: SSH username: vagrant
    default: SSH auth method: private key
    default: Warning: Connection timeout. Retrying...
==> default: Machine booted and ready!
==> default: Checking for guest additions in VM...
    default: Guest Additions Version: 4.2.0
    default: VirtualBox Version: 4.3
==> default: Configuring and enabling network interfaces...
==> default: Mounting shared folders...
    default: /vagrant => /Users/simon/dev/pfc
```



Por el momento no necesitaremos ninguna otra configuración sobre nuestra máquina virtual, por lo que podemos pasar al siguiente paso, no sin antes guardar nuestra evolución en nuestro repositorio.

Para conseguirlo, en primer lugar ejecutaremos *`git status`* para comprobar qué archivos presentan cambios o son nuevos respecto al repositorio:

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    .vagrant/
    Vagrantfile
```

Lógicamente aparece listado el fichero que hemos creado, *Vagrantfile*, pero también un directorio oculto llamado *.vagrant*, y que guarda información relacionada con la máquina virtual que hemos lanzado. Es un directorio que no nos interesa enviar al repositorio, pues guarda información local a cada usuario, y deberemos excluirlo del

mismo. Para ello, git nos permite crear un archivo *.gitignore* dónde explicitaremos tales exclusiones.

Por el momento, tan sólo excluiremos el directorio *.vagrant*:

```
#  simon  at  MacBook-Pro-de-Simon-2.local  in  ~/dev/pfc  on  git:master  o
[12:53:45]
$ echo .vagrant > .gitignore

#  simon  at  MacBook-Pro-de-Simon-2.local  in  ~/dev/pfc  on  git:master  o
[12:53:45]
$ cat .gitignore
.vagrant
```

Si ejecutamos de nuevo `git status`, el directorio *.vagrant* debería haber desaparecido del listado:

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    .gitignore
    Vagrantfile
```

En este punto, tan sólo tenemos que añadir los archivos al repositorio local, hacer *commit* de nuestro cambio, y enviarlo al servidor de GitHub:

```
#  simon  at  MacBook-Pro-de-Simon-2.local  in  ~/dev/pfc  on  git:master  o
[12:53:45]
$ git commit -m “Creating Vagrantfile”

#  simon  at  MacBook-Pro-de-Simon-2.local  in  ~/dev/pfc  on  git:master  o
[12:53:45]
$ git push
Counting objects: 5, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 521 bytes | 0 bytes/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To git@github-pfc:pfc2014/pfc.git
   265bd3a..3b83f94  master -> master
```

Comprobar si los cambios se han subido a GitHub es tan sencillo como cargar la página del proyecto y ver si están allí:

3.3 Configurando nuestra máquina virtual con Ansible

Tras haber creado en el punto anterior la definición de nuestra máquina virtual y haberla añadido al repositorio, ha llegado el momento de utilizar Ansible para gestionar nuestras dependencias de software, que como dijimos con anterioridad consistirán en:

- Un servidor web, nginx en este caso.
- Un servidor de base de datos, MySQL.
- Un intérprete de php, PHP5-FPM.

En este punto abordaremos la instalación y configuración de cada uno de esos servicios con Ansible, pero en primer lugar vamos a añadir al directorio de nuestro proyecto la configuración básica para que Ansible funcione.

Así pues, repitamos los pasos que describimos en la fase anterior y creemos un directorio de nombre *ansible* en nuestro proyecto, dónde a su vez crearemos un archivo *hosts* que especificará las máquinas que Ansible debe configurar:

```
# simon at MacBook-Pro-de-Simon-2.local in ~/dev/pfc/ansible on git:master x
[14:32:08]
$ ll
total 8
drwxr-xr-x  3 simon  staff   102B  1 nov 14:28 .
drwxr-xr-x  8 simon  staff   272B  1 nov 14:27 ..
-rw-r--r--  1 simon  staff    23B  1 nov 14:28 hosts

# simon at MacBook-Pro-de-Simon-2.local in ~/dev/pfc/ansible on git:master x
[14:32:09]
$ cat hosts
[pfc-vm]
192.168.33.30
```

Antes de continuar, asegurémonos que Ansible puede ejecutar comandos sobre la máquina virtual, con la orden que ya vimos en la etapa anterior:

```
$ ansible -i ansible/hosts -u vagrant --private-
key=~/.vagrant.d/insecure_private_key all -m ping

192.168.33.30 | success >> {
    "changed": false,
```

```
    "ping": "pong"
}
```

Perfecto, Ansible es capaz de comunicarse y ejecutar comandos sobre la máquina virtual, con lo que ya podemos avanzar al siguiente punto dónde comenzaremos a instalar los servicios necesarios.

3.3.1 Instalando nginx vía Ansible

Hasta el momento, tan sólo hemos introducido el concepto de *playbook*, en el que vimos una tarea básica, que instalaba nginx de forma elemental sobre la máquina virtual.

Simplificando, podríamos utilizar el mismo *playbook* para ir añadiendo todas nuestras dependencias, pero lo más probable es que el resultado fuera un archivo muy extenso en el que además se mezclarían aplicaciones y configuraciones de distintos servicios.

Es por eso que vamos a introducir un concepto nuevo de Ansible, los Roles[liv], que no son sino una herramienta que nos permitirá organizar nuestros *playbooks* de forma más eficiente.

Los roles a nivel físico, no son más que una estructura de directorios predefinida:

```
roles/
   common/
     files/
     templates/
     tasks/
     handlers/
     vars/
     defaults/
     meta/
```

Ansible nos proporcionará a nivel lógico ciertos automatismos, que son lo que hace que los roles sean una herramienta adecuada para organizar nuestro código. Por ejemplo:

- Si *roles/x/tasks/main.yml* existe, las tareas en ese archivo se añadirán automáticamente a nuestra jugada.
- Si *roles/x/handlers/main.yml* existe, los *handlers* en ese archivo se añadirán automáticamente a nuestra jugada.
- Si *roles/x/vars/main.yml* existe, las variables en ese archivo se añadirán automáticamente a nuestra jugada.
- Si *roles/x/meta/main.yml* existe, las dependencias lisadas en ese archivo se añadirán automáticamente a nuestra jugada.
- *roles/x/files* nos permitirá alojar archivos que se podrán utilizar en nuestras tareas en el rol sin hacer referencia a su ruta completa.

- *roles/x/templates* nos permitirá alojar archivos de plantilla que se podrán utilizar en nuestras tareas en el rol sin hacer referencia a su ruta completa.

Para entender cómo funciona un rol, nada mejor que un ejemplo, así que aprovechemos que necesitamos configurar nginx, para crear nuestro primer rol en Ansible.

En primer lugar, vamos a crear la estructura de directorios mínima que necesitaremos para nuestro rol:

```
#  simon  at  MacBook-Pro-de-Simon-2.local  in  ~/dev/pfc  on  git:master  x
[14:45:24]
$ mkdir -p ansible/roles/pfc-nginx/{defaults,handlers,tasks,templates}

#  simon  at  MacBook-Pro-de-Simon-2.local  in  ~/dev/pfc  on  git:master  x
[14:47:53]
$ tree ansible
ansible
├── hosts
└── roles
    └── pfc-nginx
        ├── defaults
        ├── handlers
        ├── tasks
        └── templates
```

Hemos creado varios directorios, *defaults*, *tasks*, *handlers* y *templates*, que almacenarán distintos tipos de archivos que utilizaremos para configurar nuestra máquina destino. Comenzaremos a trabajar sobre las tareas, para posteriormente ir entrando en la utilidad de los otros directorios.

3.3.1.1 Creando nuestro primer fichero de tareas main.yml

Nuestro primer paso será crear un archivo en formato Yaml llamado *main.yml* en el directorio recién creado *tasks*. El nombre *main.yml* no es casual, ya que como hemos comentado, es el nombre del archivo que Ansible buscará para ejecutar dentro de cada carpeta de un rol.

Una vez creado el archivo, dotémoslo de contenido, añadiendo las primeras tareas para instalar nginx. Si bien en la anterior fase del proyecto instalamos la versión oficial de Ubuntu del paquete, en esta ocasión nuestro objetivo será instalar la versión oficial de los desarrolladores de nginx, la cual como norma general será más actual que la que cualquier distribución de Ubuntu traiga en sus fuentes de paquetes.

El contenido del archivo que *main.yml* quedaría tal que así:

```
# ansible/roles/pfc-nginx/tasks/main.yml
```

```
---
- name: Adding nginx official ppa repository
  apt_repository: >
    repo="ppa:nginx/stable"
    update_cache=yes

- name: Installing ngxinx-full
  apt: >
    name=nginx-full
    state=latest
```

En el código anterior podemos observar dos tareas, cada una utilizando un módulo de Ansible distinto. La primera, utiliza apt_repository[lv], cuya función principal es añadir un nuevo repositorio fuente a los orígenes por defecto del equipo a gestionar. En este caso, le indicamos al sistema que añada el repositorio ppa:nginx/stable[lvi], el oficial publicado por el equipo de nginx. El parámetro *update_cache* será necesario para que el sistema ejecute *apt-get update* tras haberlo añadido. De lo contrario, la máquina virtual no sería consciente de que se ha introducido una nueva referencia a un repositorio en el sistema.

Tras haber añadido el repositorio, la segunda tarea es la encargada de instalar nginx. En esta ocasión utilizamos el módulo apt[lvii], que es el encargado de gestionar la instalación de paquetes APT en arquitecturas Debian / Ubuntu.

Con tan sólo estas dos tareas, Ansible ya debería ser capaz de instalar nginx sobre nuestra máquina virtual. Pero para comprobarlo, necesitaremos una forma sencilla de testar nuestros roles según vayamos avanzando.

3.3.1.2 Configurando `vagrant provision`

Para probar nuestro recién añadido *playbook*, necesitamos configurar *Vagrantfile* de forma que podamos ejecutar *`vagrant provision`* en lugar de tener que invocar el pesado comando de Ansible que ya vimos en la primera fase del proyecto:

```
ansible-playbook playbook-nginx.yml -i hosts -u vagrant--private-
key=~/.vagrant.d/insecure_private_key -s
```

Como también vimos anteriormente, afortunadamente Vagrant está preparado para utilizar a Ansible como método de aprovisionamiento automático. Para lograrlo necesitaremos añadir las siguientes líneas a nuestro archivo *Vagrantfile*:

```
config.vm.provision "ansible" do |ansible|
    ansible.playbook = "ansible/playbook.yml"
    ansible.inventory_path = "ansible/hosts"
    ansible.limit = "pfc-vm"
```

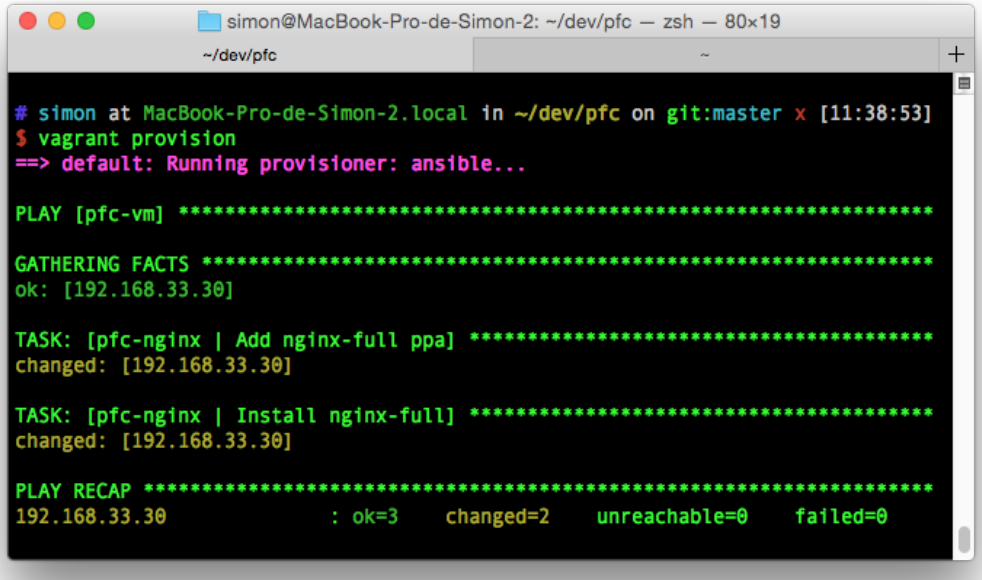
```
    ansible.sudo = true
  end
```

Si examinamos la configuración anterior, podemos ver que en el parámetro *ansible.playbook* hemos introducido un archivo que todavía no existe en nuestro proyecto *ansible/playbook.yml*, y que en la fase uno no era sino el *playbook* maestro que ejecutaba los distintos comandos de configuración.

Sin embargo, en esta ocasión hemos optado por separar los distintos *playbooks* en forma de roles, por lo que necesitaremos uno que, en lugar de contener directamente las tareas, contenga la invocación a los roles que vamos a aplicar sobre las distintas máquinas. Tan sencillo como crear el archivo *playbook.yml* con el siguiente contenido:

```
# playbook.yml
---

- hosts: [pfc-vm]
  sudo: yes
  roles:
    - pfc-nginx
```

Una vez configurado Vagrant para invocar a Ansible, probemos si cumple nuestras expectativas:



Por lo que parece, todo ha funcionado como esperábamos, y Ansible parece haber ejecutado las dos tareas tal sin problemas, por lo que nginx ya debería estar instalado en el host destino.

No nos confiemos en cualquier caso y hagamos *login* en nuestra máquina virtual para comprobarlo, y de paso, asegurémonos también que la versión de nginx es más reciente que la que instalamos en la primera fase a través de los canales base de Ubuntu (1.1.19).

```
$ vagrant ssh
Welcome to Ubuntu 12.04 LTS (GNU/Linux 3.2.0-23-generic x86_64)

Welcome to your Vagrant-built virtual machine.
Last login: Sun Nov  2 10:42:02 2014 from 10.0.2.2

vagrant@precise64:~$ nginx -v
nginx version: nginx/1.6.2
```

Y efectivamente, así es, con nuestro recién creado rol de Ansible y apoyándonos en Vagrant hemos instalado nginx en nuestra máquina virtual en su versión 1.6.2.

Pasemos ahora a añadir algunas tareas de forma que nuestro rol sea lo más completo posible.

3.3.1.3 Subiendo un archivo de configuración propio

Hasta el momento hemos conseguido instalar nginx a través del repositorio oficial de los desarrolladores. Se trata de un nginx funcional, pero cuya configuración base está orientada a servidores de producción. A continuación vamos a añadir una tarea que sustituya el archivo nginx.conf por una versión propia.

Analicemos primero la estructura del directorio */etc/nginx* recién instalado en nuestra máquina virtual::

```
vagrant@precise64:/etc/nginx$ tree
.
|-- conf.d
|-- fastcgi.conf
|-- fastcgi_params
|-- koi-utf
|-- koi-win
|-- mime.types
|-- nginx.conf
|-- proxy_params
|-- scgi_params
|-- sites-available
|   `-- default
|-- sites-enabled
|   `-- default -> /etc/nginx/sites-available/default
|-- snippets
|   |-- fastcgi-php.conf
|   `-- snakeoil.conf
|-- uwsgi_params
```

```
`-- win-utf
```

Podemos ver que el archivo que nos interesa se encuentra en el raíz de ese directorio. Echemos un vistazo a su contenido por defecto:

```
vagrant@precise64:/etc/nginx$ cat nginx.conf
user www-data;
worker_processes 4;
pid /run/nginx.pid;

events {
   worker_connections 768;
   # multi_accept on;
}

http {

   ##
   # Basic Settings
   ##

   sendfile on;
   tcp_nopush on;
   tcp_nodelay on;
   keepalive_timeout 65;
   types_hash_max_size 2048;
   # server_tokens off;

   # server_names_hash_bucket_size 64;
   # server_name_in_redirect off;

   include /etc/nginx/mime.types;
   default_type application/octet-stream;

   ##
   # Logging Settings
   ##

   access_log /var/log/nginx/access.log;
   error_log /var/log/nginx/error.log;

   ##
   # Gzip Settings
   ##

   gzip on;
   gzip_disable "msie6";

   # gzip_vary on;
   # gzip_proxied any;
   # gzip_comp_level 6;
   # gzip_buffers 16 8k;
   # gzip_http_version 1.1;
```


```
    # gzip_types text/plain text/css application/json application/javascript
text/xml application/xml application/xml+rss text/javascript;

    ##
    # Virtual Host Configs
    ##

    include /etc/nginx/conf.d/*.conf;
    include /etc/nginx/sites-enabled/*;
}


#mail {
#  # See sample authentication script at:
#  # http://wiki.nginx.org/ImapAuthenticateWithApachePhpScript
#
#  # auth_http localhost/auth.php;
#  # pop3_capabilities "TOP" "USER";
#  # imap_capabilities "IMAP4rev1" "UIDPLUS";
#
#  server {
#         listen     localhost:110;
#         protocol   pop3;
#         proxy      on;
#  }
#
#  server {
#         listen     localhost:143;
#         protocol   imap;
#         proxy      on;
#  }
#}
```

Como vemos, al igual que el fichero *Vagrantfile* que genera Vagrant, se trata de un archivo en texto plano con algunas líneas de configuración definidas y otras comentadas para que sirvan a modo de plantilla.

Tras un primer análisis, y teniendo en cuenta que estamos preparando un entorno de desarrollo, decidimos que no queremos que nginx comprima las páginas que sirva con gzip, lo cual nos lleva a tener que cambiar el parámetro *gzip on* por *gzip off*. Al mismo tiempo, vamos a aprovechar y limpiar el archivo de comentarios que no creemos necesarios.

Para poder hacer esto automáticamente, un administrador de sistemas necesitaría crear un script propio que, tras instalar el servidor web, editase el archivo *nginx.conf*. Conseguir un control de los cambios realizados en el host con ese script, sería tarea casi imposible. Ansible, nos proporciona una solución adecuada a este problema, su sistema de plantillas[lviii].

Ansible utiliza este sistema, basado en el lenguaje Jinja2[lix], y que nos permite incluso insertar variables en nuestros archivos de configuración estáticos, que podrá sustituir dinámicamente durante su ejecución. La forma de insertar una variable en un archivo

de configuración es tan sencilla como poner su nombre entre dobles llaves, como por ejemplo *{{ nginx_gzip_status }}*. Los archivos de plantilla se guardarán en el directorio *templates*, con extensión *.j2*.

Así quedaría por ejemplo nuestro fichero de configuración *nginx.conf.j2*:

```
$ cat ansible/roles/pfc-nginx/templates/nginx.conf.j2

user www-data;
worker_processes 4;
pid /run/nginx.pid;

events {
  worker_connections 768;
}

http {

  sendfile on;
  tcp_nopush on;
  tcp_nodelay on;
  keepalive_timeout 65;
  types_hash_max_size 2048;

  include /etc/nginx/mime.types;
  default_type application/octet-stream;

  access_log /var/log/nginx/access.log;
  error_log /var/log/nginx/error.log;

  gzip {{ nginx_gzip_status }};
  gzip_disable "msie6";

  # gzip_vary on;
  # gzip_proxied any;
  # gzip_comp_level 6;
  # gzip_buffers 16 8k;
  # gzip_http_version 1.1;
  #  gzip_types  text/plain  text/css  application/json  application/javascript
text/xml application/xml application/xml+rss text/javascript;

  include /etc/nginx/conf.d/*.conf;
  include /etc/nginx/sites-enabled/*;
}
```

Como vemos, hemos introducido nuestra primera variable, pero no la hemos definido en nuestro código. El siguiente paso será definirla dentro del rol, para lo que necesitaremos crear un nuevo archivo *main.yml* en el directorio defaults de nuestro rol, directorio el cual almacena las variables que, por defecto, cargará nuestro rol.

El contenido de ese archivo será:

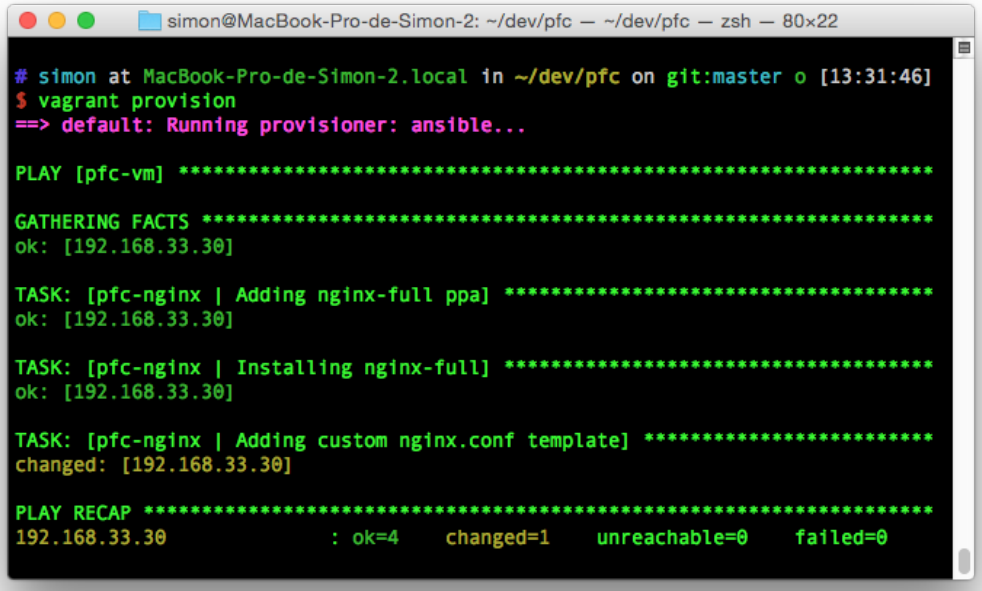
```
# file: ansible/roles/pfc-nginx/defaults/main.yml
---
nginx_gzip_status: "off"
```

Llegados a este punto, podemos preguntarnos ¿por qué necesitamos una plantilla, en lugar de simplemente guardar un archivo *nginx.conf* en local con la configuración directamente sobrescrita. En nuestro caso, realmente el resultado sería el mismo, pero hemos querido hacerlo así para introducir el concepto de variables, las cuales permitirían utilizar una misma plantilla para configurar distintos entornos (variable a *off* para desarrollo, a *on* para producción).

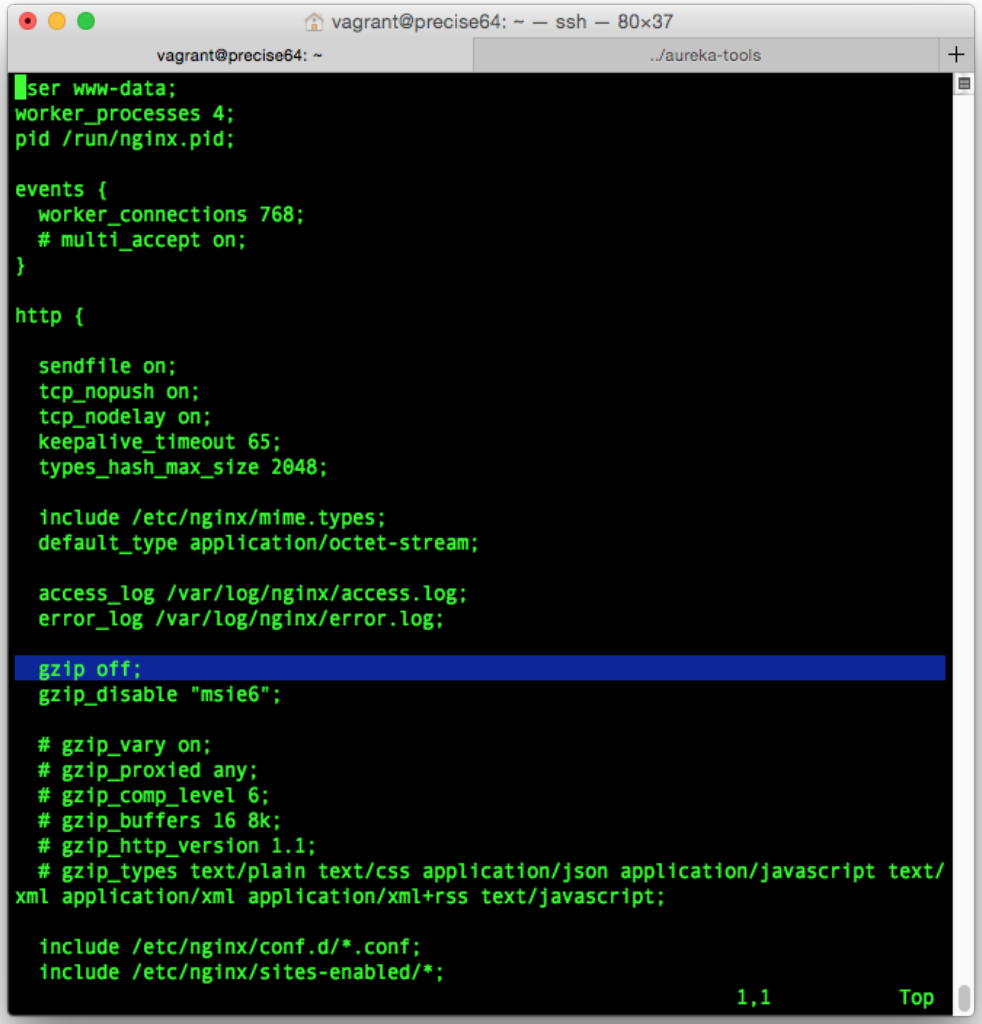
Hecho el comentario, y tras añadir el *template* y la definición de la misma, ya estamos listos para crear la tarea que sustituirá el archivo *nginx.conf* de nuestra máquina virtual por el nuevo utilizando el comando *template*:

```
- name: Adding custom nginx.conf template
  template: >
    src=nginx.conf.j2
    dest=/etc/nginx/nginx.conf
```

El parámetro *src* no necesita que le indiquemos la ruta, pues por defecto, el módulo buscará en la carpeta *templates* de nuestro rol. Ejecutemos *vagrant provision* de nuevo para comprobar si todo funciona:



Por el momento, Ansible parece que ha hecho su trabajo. Para comprobarlo, entremos en la máquina virtual y comprobemos que *nginx.conf* ha sido realmente modificado:

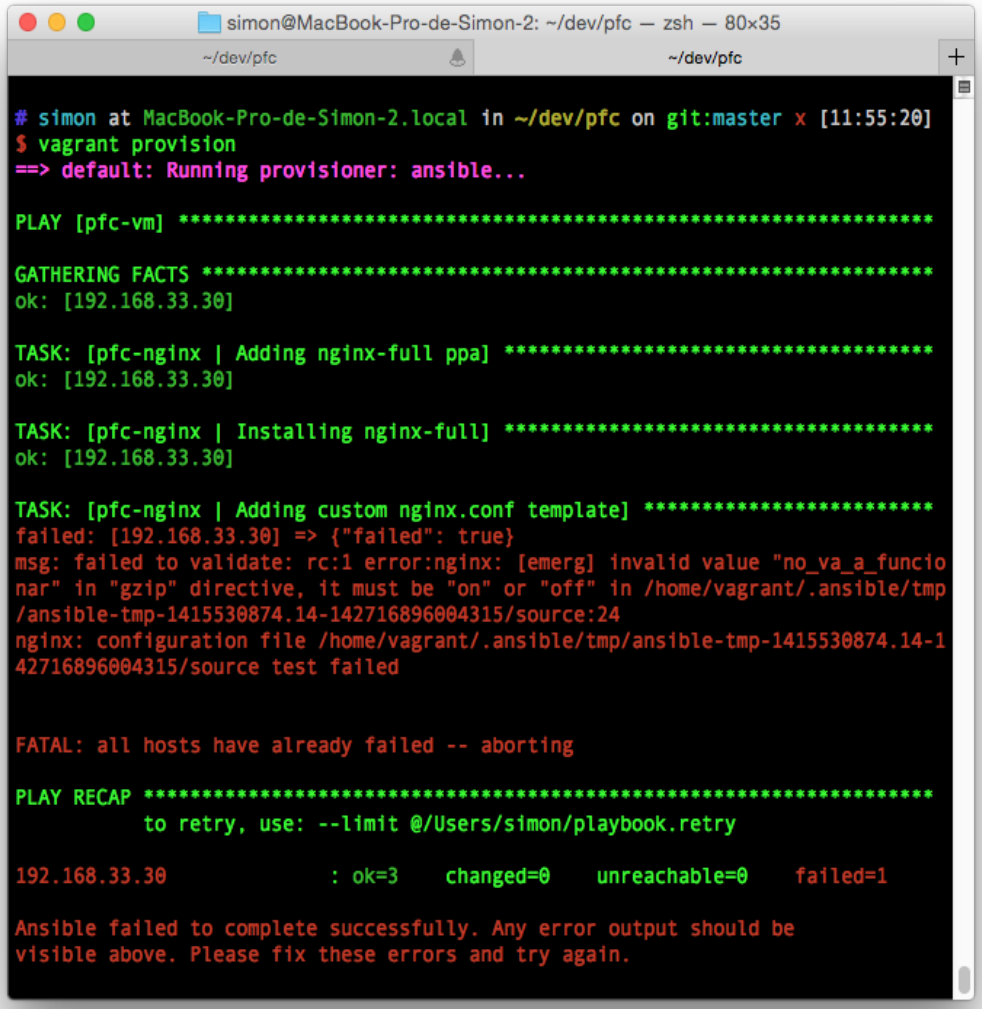


Afortunadamente, el resultado valida nuestro código, pero ¿qué sucedería si hubiésemos subido un archivo de configuración con un parámetro erróneo? En ese caso, nuestro nuevo archivo de configuración machacaría el antiguo y, si hubiéramos introducido algún error nuestro servicio quedaría inutilizado. Afortunadamente, podemos prevenirlo utilizando el parámetro *validate* en nuestra tarea:

```
- name: Adding custom nginx.conf template
  template: >
    src=nginx.conf.j2
    dest=/etc/nginx/nginx.conf
    validate="nginx -t -c %s"
```

Validate ejecuta el comando que le pasemos como paso previo a sobrescribir el archivo, siendo en este caso una invocación al comando *`nginx -t -c %s`* el cual se encarga de comprobar que la sintaxis del archivo de configuración que vamos a copiar es correcta (%s representa el archivo *src* de nuestra tarea). Probemos a introducir un error en el

mismo, cambiando la variable en *default/main.yml nginx_gzip_status* de *off* a *no_va_a_funcionar*:



Efectivamente, comprobamos que Ansible falla avisándonos de que hemos introducido un parámetro inapropiado en la configuración de nginx.

Hasta el momento hemos sido capaces de instalar un paquete y aplicar una configuración propia con control de fallos, pero en ningún punto hemos dado la orden de recargar esta configuración en el host destino, por lo que el servidor web en la máquina virtual sigue funcionando con la configuración original.

Para recargar el servicio, necesitaremos apoyarnos en un concepto nuevo, los *handlers*.

3.3.1.4 Handlers

Como comentábamos en el punto anterior, tras instalar un servicio y realizar un cambio en su configuración, necesitamos solicitar una acción *reload* para recargarla. Así pues

necesitaremos apoyarnos en dos nuevos conceptos de Ansible, las notificaciones y los *handlers* o manejadores.

Las notificaciones son simples llamadas en YAML que se añaden a las tareas que necesitan notificar de algún cambio a un manejador. Añadir una notificación para reiniciar nginx es tan sencillo como añadir una línea a la tarea que lo requiera:

```
# ansible/roles/pfc-nginx/tasks/main.yml
---

- name: Adding nginx-full ppa
  apt_repository: >
    repo="ppa:nginx/stable"
    update_cache=yes

- name: Installing nginx-full
  apt: >
    name=nginx-full
    state=latest

- name: Adding custom nginx.conf template
  template: >
    src=nginx.conf.j2
    dest=/etc/nginx/nginx.conf
  notify: reload nginx
```

Si tratamos de ejecutar ese código, Ansible nos devolverá un error, y es que toda llamada a *notify* va asociada a una tarea llamada *handler* que todavía no hemos definido.

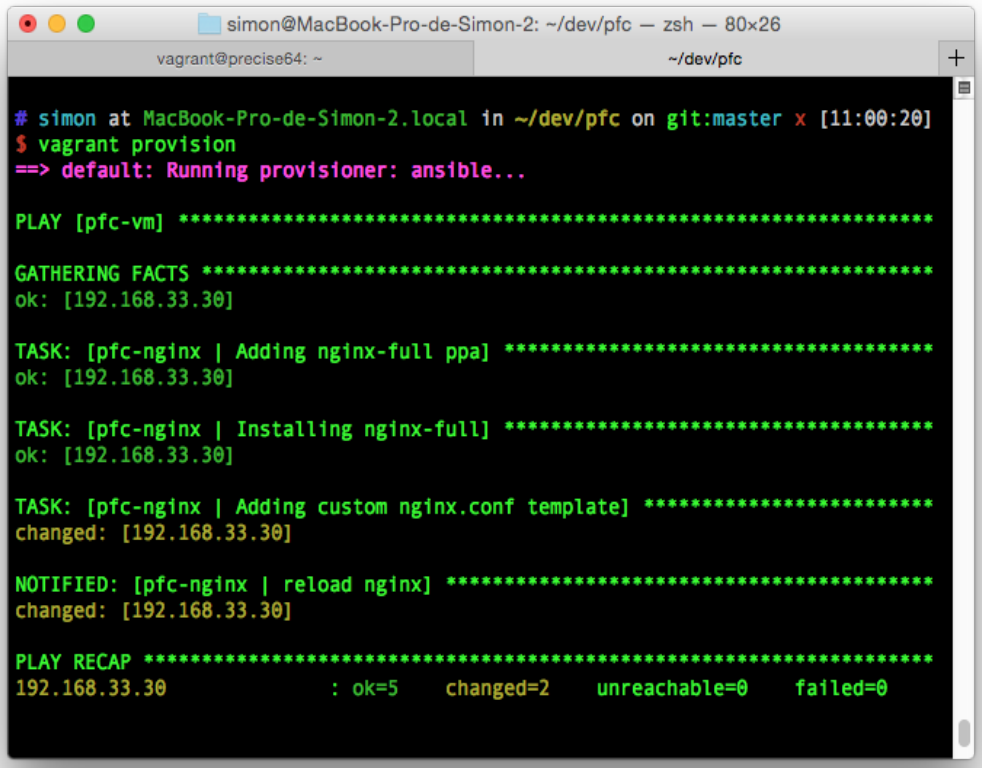
Los *handlers* son tareas o listas de tareas que únicamente se ejecutan si son invocadas por una notificación, con la característica de que sólo se ejecutan una vez independientemente de cuántas notificaciones sobre el mismo manejador se hayan solicitado . Es decir, que si dos notificaciones solicitan el reinicio de nginx, éste sólo se efectuará una vez

Los *handlers*, serán, por tanto, candidatos naturales para definir nuestras acciones sobre servicios, tales como *start*, *stop*, *reload*, *restart*, etc. Para añadir un nuevo manejador a nuestro rol, nada más sencillo que añadir un archivo *main.yml* a nuestra carpeta *handlers* con el siguiente contenido:

```
# ansible/roles/pfc-nginx/handlers/main.yml
---
- name: reload nginx
  service: name=nginx state=reloaded
```

Si todo ha ido bien, Ansible debería recargar el servicio de nginx en la máquina virtual únicamente en el caso de que hubiera un cambio de configuración. Comprobémoslo,

cambiando temporalmente la variable *nginx_gzip_status: off* a *nginx_gzip_status: on* en *defaults* para forzar la recarga del servicio:



Efectivamente, podemos apreciar como Ansible ha cambiado la configuración de nginx, efectuado una notificación y recargado el servicio.

3.3.1.5 Iniciando un servicio tras un reinicio

Prácticamente hemos terminado nuestro primer rol de Ansible para configurar nginx en cualquier host. Echando un vistazo a la estructura de directorios del proyecto, podemos ver cómo va tomando forma:

```
# simon at MacBook-Pro-de-Simon-2.local in ~/dev/pfc on git:master x
[11:43:25]
$ tree .
.
├── README.md
├── Vagrantfile
└── ansible
    ├── hosts
    ├── playbook.yml
    └── roles
        └── pfc-nginx
            ├── defaults
            │   └── main.yml
```

```
├── handlers
│   └── main.yml
├── tasks
│   └── main.yml
└── templates
    └── nginx.conf.j2

7 directories, 8 files
```

Antes de avanzar al siguiente rol, nos queda todavía un aspecto a configurar, y sería decirle a nuestra máquina destino que configure el servicio de nginx de forma que se arranque en cada reinicio de la misma. Lo conseguiremos añadiendo la siguiente tarea a *tasks/main.yml*:

```
- name: Ensure nginx is started at boot
  service: >
    name=nginx
    state=started
    enabled=yes
```

Básicamente, se trata de una invocación al módulo service[lx] de Ansible, indicándole el servicio nginx, el estado en el que queremos que se encuentre, *started*, y añadiendo el parámetro *enabled=yes* el cual se encargará de configurar el sistema para que se inicie tras un reinicio.

En este punto ya podemos dar por concluido nuestro rol de nginx, el cual podríamos ejecutar sobre cualquier host dejándolo en el estado en el que lo hemos definido. A continuación abordaremos la creación del rol para instalar nuestro motor de base de datos, MySQL.

3.3.2 Instalando MySQL vía Ansible

Tras haber creado nuestro primer rol de nginx en el punto anterior, llega el turno de crear el siguiente, el cual nos permitirá instalar el servidor de base de datos MySQL. Comenzamos por crear la carpeta que lo albergará, junto con los directorios que necesitaremos:

```
#  simon  at  MacBook-Pro-de-Simon-2.local  in  ~/dev/pfc  on  git:master  o
[12:25:18]
$ mkdir -p ansible/roles/pfc-mysql/{defaults,handlers,tasks,templates}
```

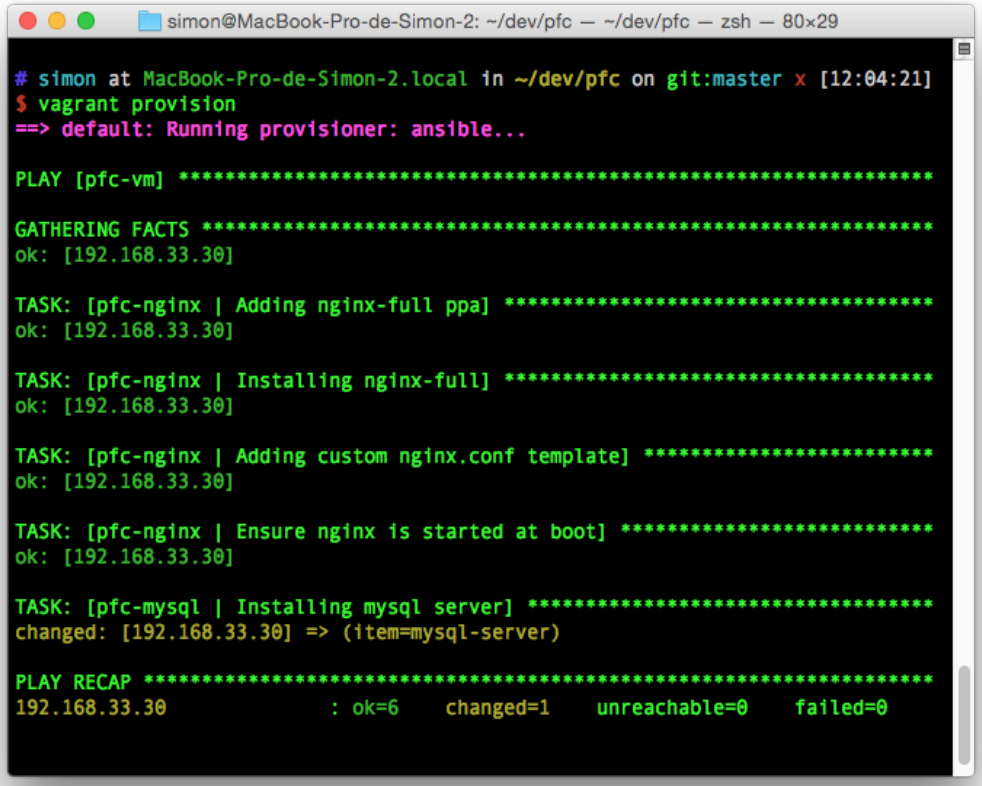
Una vez creada la estructura de directorios, comencemos por crear el archivo *main.yml* dentro de tasks, dónde definiremos la primera tarea para instalar MySQL:

```
# ansible/roles/pfc-mysql/tasks/main.yml
---

- name: Installing mysql server
  apt: >
    name=mysql-server
    state=latest
```

Pese a haberla añadido, si ejecutásemos *`vagrant provision`* esta nueva tarea no se iniciaría, y es que si bien hemos creado un nuevo rol, todavía no lo hemos añadido al host destino en nuestro *playbook*. Tan sencillo como editar el archivo *playbook.yml* en el directorio raíz de *ansible*:

```
# ansible/playbook.yml
---

- hosts: [pfc-vm]
  sudo: yes
  roles:
    - pfc-nginx
    - pfc-mysql
```

Ahora sí, *`vagrant provision`* debería ejecutar la tarea recién añadida al *playbook*:



Podemos comprobar que MySQL se ha instalado en nuestra máquina virtual entrando por *ssh* y tratando de conectar al servidor de base de datos:

```
# simon at MacBook-Pro-de-Simon-2.local in ~/dev/pfc on git:master x
[12:44:41]
$ vagrant ssh
Welcome to Ubuntu 12.04 LTS (GNU/Linux 3.2.0-23-generic x86_64)

vagrant@precise64:~$ mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 45
Server version: 5.5.40-0ubuntu0.12.04.1 (Ubuntu)

Copyright (c) 2000, 2014, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input
statement.

mysql>
```

Efectivamente, podemos ver como se ha instalado MySQL en su versión 5.5.40, suficiente para los requisitos de nuestro proyecto.

Antes de avanzar al siguiente apartado, dónde abordaremos la securización de MySQL, aprovechemos para añadir también la tarea que se encargará de iniciar el servidor de base de datos tras un reinicio de la máquina virtual, del mismo modo que ya vimos con nginx:

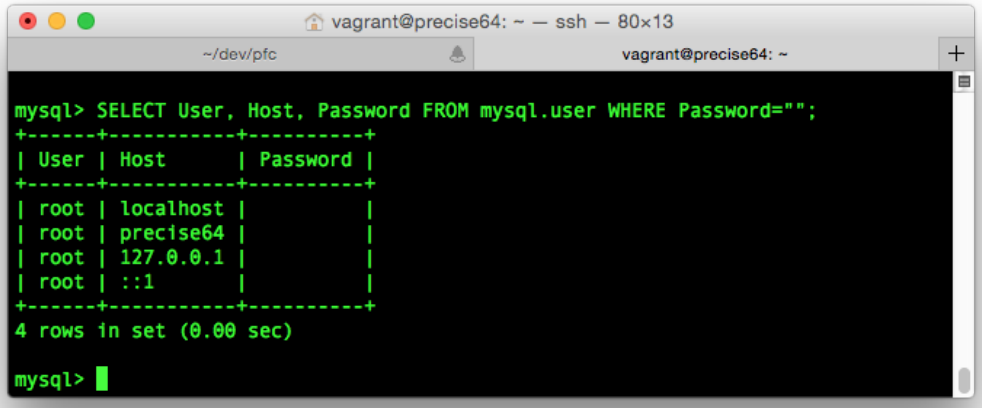
```
# ansible/roles/pfc-mysql/tasks/main.yml
---

- name: Installing mysql-server
  apt: >
    name=mysql-server
    state=latest

- name: Ensuring mysql is started at boot
  service: >
    name=mysql
    state=started
    enabled=yes
```

3.3.2.1 Securizando nuestra instalación de MySQL

Los ojos más avezados habrán detectado un fallo de seguridad en nuestra configuración de MySQL, y es que al hacer la prueba en la máquina virtual para ver si se había instalado correctamente, hemos entrado con el usuario administrador *(root)* sin introducir ninguna contraseña.

Si bien en un entorno de desarrollo virtualizado, la seguridad no es un aspecto crítico ya que estamos montado el entorno sobre nuestro propio equipo, pensemos que este rol también lo podríamos extender en el futuro para que configurase cualquier otro host en un entorno de pruebas o producción. Vamos, por lo tanto a configurar MySQL de forma segura a modo de ejemplo para futuras ampliaciones del rol.

El primer paso en esta securización será crear una contraseña para el usuario administrador del servidor de base de datos. Generalmente este proceso se realiza interactivamente durante la instalación de MySQL, pero como en esta ocasión es Ansible quién está realizando el proceso de forma desasistida, necesitamos averiguar exactamente las cuentas a securizar. Hagamos un listado de los usuarios sin contraseña de nuestro recién instalado motor de base de datos:



Como podemos apreciar, hay cuatro líneas en la tabla *mysql.user* de nuestra base de datos con contraseña vacía. Todas ellas asociadas a *root*, y que se diferencian en el host asociado. Necesitaremos una tarea que se encargue de crear un *password* para cada una de ellas:

```
- name: Setting the root password
  mysql_user:
    user: root
    host: "{{ item }}"
    password: "{{ mysql_root_password }}"
  with_items:
   - localhost
   - "{{ ansible_hostname }}"
   - 127.0.0.1
   - ::1
```

En esta tarea utilizamos un módulo nuevo de Ansible, mysql_user[lxi], el cual nos permite añadir o eliminar usuarios de una base de datos MySQL. También utilizamos unas cuantas variables, *item*, *mysql_root_password* y *ansible_hostname*, e introducimos un concepto nuevo de Ansible del que todavía no habíamos hablado, los bucles[lxii].

3.3.2.1.1 Uso de bucles en Ansible

Si analizamos el problema que intentamos resolver con esta tarea, observamos que el objetivo de la misma es dotar de una contraseña al usuario *root* de la base de datos, pero para hacerlo necesitamos alterar cuatro registros distintos de la misma. Si contemplamos cada dupla *root/host* como un usuario distinto, nuestra tarea se resume en ejecutar cuatro veces la llamada a la modificación de un usuario.

Así pues, en la primera parte de nuestra tarea encontramos la instrucción necesaria para asignar una contraseña a un usuario/host:

```
- name: Setting the root password
  mysql_user:
    user: root
    host: "{{ item }}"
    password: "{{ mysql_root_password }}"
```

En el caso de que quisiéramos por ejemplo asignar la contraseña *secret_password* al usuario *root* en nuestro *localhost*, la tarea quedaría de la siguiente forma:

```
- name: Setting the root password
  mysql_user:
    user: root
    host: localhost
    password: secret_password
```

Ahora bien, como tendríamos que repetir el proceso hasta cuatro veces, una por cada *host* identificado en la base de datos susceptible de ser cambiado, sería conveniente utilizar un bucle en lugar de escribir cuatro veces la tarea.

Para entender como funciona un bucle en Ansible es necesario que nos fijemos en la variable *{{ item }} y* en la segunda parte de la tarea, *with_items*:

```
- name: Setting the root password
  mysql_user:
    user: root
    host: "{{ item }}"
    password: "{{ mysql_root_password }}"
  with_items:
   - localhost
   - "{{ ansible_hostname }}"
   - 127.0.0.1
   - ::1
```

La palabra *with_items* es una de las palabras reservadas que define el comienzo de un bucle en Ansible. Básicamente le indica que debe repetir la primera parte de la tarea, tantas veces como *items* haya en la lista que le sigue. Además, Ansible se encargará de ir iterando la variable en cada pasada del bucle hasta que haya recorrido toda la lista.

De esa forma, podemos ver que nuestra tarea repite cuatro veces la modificación de un usuario MySQL, cada una de las veces con su *host* correspondiente.

3.3.2.1.2 Recogiendo información del host en Ansible

Habiendo introducido los bucles, todavía hay un par de variables en nuestra tarea sobre las que no hemos hablado. Por una parte *{{ mysql_root_password }}*, que no será sino la contraseña que queremos asignar al usuario, y que suele ser la misma

independientemente del host. Necesitamos asignarla, así que lo que haremos será crear el archivo *defaults/main.yml* definiéndola.

```
# simon at MacBook-Pro-de-Simon-2.local in ~/dev/pfc/ansible on git:master x
[21:16:28]
$ cat roles/pfc-mysql/defaults/main.yml
# ansible/roles/pfc-mysql/defaults/main.yml
---
mysql_root_password: "secret_password"
```

La otra variable que nos queda por abordar, *{{ ansible_hostname }}* es interesante, pues es una variable interna que se asigna en cada ejecución del comando, utilizando el módulo setup[lxiii], y que consiste en la recuperación por parte de Ansible de una serie de parámetros del host sobre el que se está ejecutando.

La información que el módulo setup devuelve es realmente extensa (ver Anexo 1). A continuación, veamos tan sólo un extracto básico del mismo:

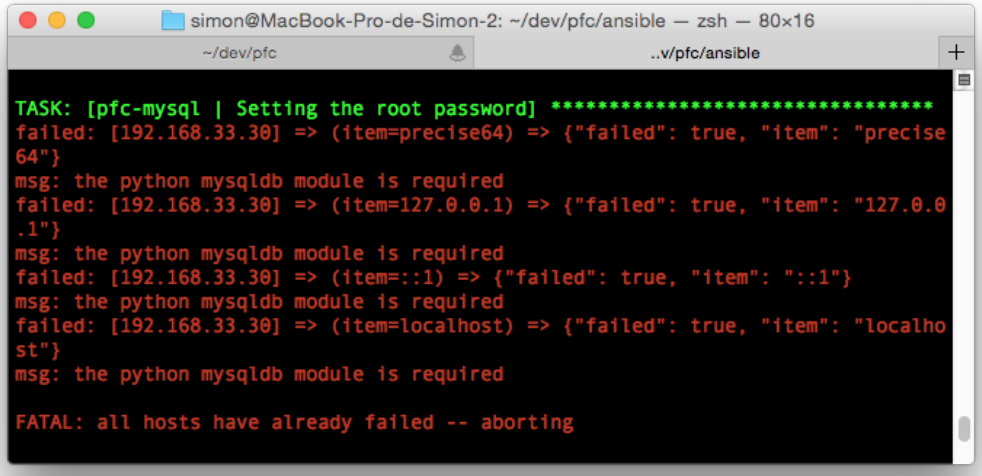
```
$ ansible -i hosts pfc-vm -u vagrant --private-
key=~/.vagrant.d/insecure_private_key -m setup

192.168.33.30 | success >> {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "10.0.2.15",
            "192.168.33.30"
        ],
        "ansible_all_ipv6_addresses": [
            "fe80::a00:27ff:fe88:ca6"
        ],
        "ansible_architecture": "x86_64",
        "ansible_bios_date": "12/01/2006",
        "ansible_bios_version": "VirtualBox",
        "ansible_cmdline": {
            "BOOT_IMAGE": "/vmlinuz-3.2.0-23-generic",
            "quiet": true,
            "ro": true,
            "root": "/dev/mapper/precise64-root"
   …
   …
   …

"ansible_hostname": "precise64"
```

Como anticipábamos, se trata de información sobre la arquitectura y configuración del host destino, tales como la dirección ip, *bios* y, avanzando en los resultados del comando, el *hostname* que necesitamos para ejecutar una de nuestras iteraciones del bucle, *precise64*.

¿Por qué recuperar el *hostname* a través de Ansible y no especificarlo directamente en la tarea? Porque de esa forma nuestro rol será independiente del nombre de host que la máquina destino tenga asignada. Así por ejemplo, *precise64* es el nombre del host que Hashicorp le ha asignado a esta máquina, pero la máquina virtual *hashicorp/trusty64* (Ubuntu 14.04), tendría un *hostname* diferente. Apoyándonos en el proceso de Ansible para recuperar información del host, logramos así que nuestra tarea sea genérica para cualquier nombre de *host*.

3.3.2.1.3 Ejecutando la actualización de contraseñas en MySQL

Una vez explicados los aspectos nuevos de nuestra tarea, vamos a ejecutarla y comprobar que efectivamente actualiza las contraseñas de los registros correspondientes:

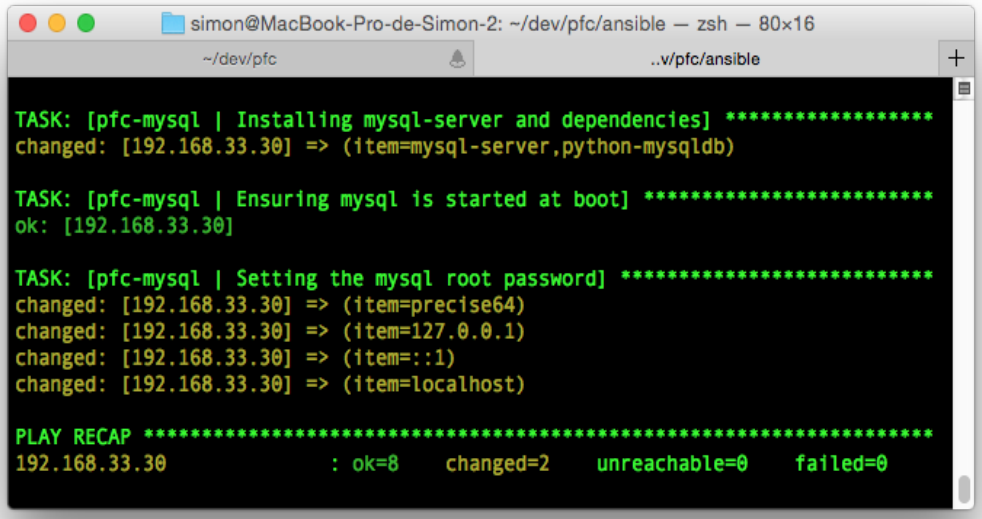


Nuestra tarea ha fallado debido a una dependencia no satisfecha, el paquete *python_mysqldb*, necesario para que Ansible pueda interactuar directamente con la base de datos en la máquina virtual.

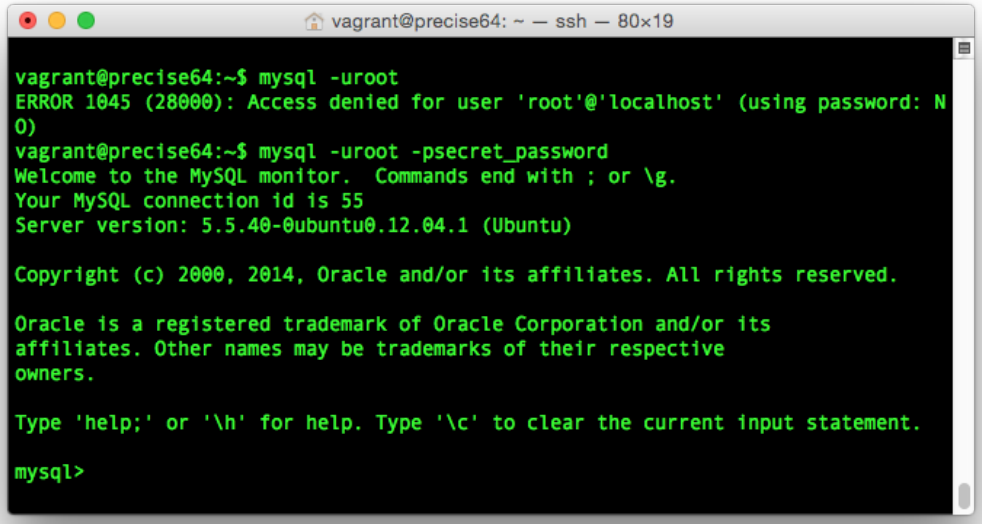
Apliquemos los conocimientos que tenemos sobre bucles y modifiquemos la primera tarea de nuestro rol, la instalación de MySQL, para añadir los paquetes necesarios para la ejecución de las distintas tareas:

```
- name: Installing mysql-server and dependencies
  apt: >
    name={{item}}
    state=latest
    with_items:
      - mysql-server
      - python-mysqldb
```

Con la nueva tarea en nuestro rol, ejecutemos de nuevo `vagrant provision`:



Ya limpio de errores, en principio hemos logrado nuestro objetivo. Entremos en la máquina virtual para comprobar que así ha sido:



Como vemos, la orden *`mysql -u root`* con la que anteriormente accedíamos directamente al servidor MySQL nos devuelve ahora acceso denegado. Por contra, si especificamos la contraseña con el parámetros *`-psecret_password`* sí podemos acceder sin problemas.

El siguiente paso será añadir una tarea que nos evite tener que escribir la contraseña en línea de comandos, facilitando así las tareas de los desarrolladores en el caso de que tuvieran que entrar en la base de datos de la máquina virtual.

3.3.2.1.4 Facilitando el acceso a MySQL al usuario root

En la medida de lo posible, el uso de la máquina virtual por parte de los desarrolladores debe ser lo más transparente posible, evitando por ejemplo, que tuvieran que indagar en el código de nuestra arquitectura para averiguar la contraseña de acceso a MySQL.

Afortunadamente MySQL nos provee de un método por el cual podemos asignar un archivo de configuración *.my.cnf* al directorio home de cualquier usuario, y del que MySQL es capaz de leer aspectos tales como el nombre de usuario y la contraseña que queremos utilizar en una conexión al servidor de base de datos.

Por ejemplo, un archivo que permitiría a un usuario *root* de nuestro sistema, conectarse sin especificar usuario y contraseña tendría este contenido:

```
$ cat /root/.my.cnf
[client]
user=root
password=secret_password
```

Por lo tanto, necesitaríamos crear una tarea en Ansible que nos permitiera subir un archivo con esa configuración, y que a su vez, nos permitiera utilizar una variable que sustituyera a *secret_password* dinámicamente, o lo que es lo mismo, necesitaremos utilizar el sistema de plantillas de Ansible que ya vimos cuando subimos una configuración propia para nginx.

Lo haremos en dos pasos. En primer lugar, creamos la plantilla, un archivo de nombre *.my.cnf.j2*, en el directorio *templates* de nuestro rol. El contenido de dicho archivo será el que acabamos de ver pero cambiando la contraseña por la variable que estamos utilizando en nuestro código:

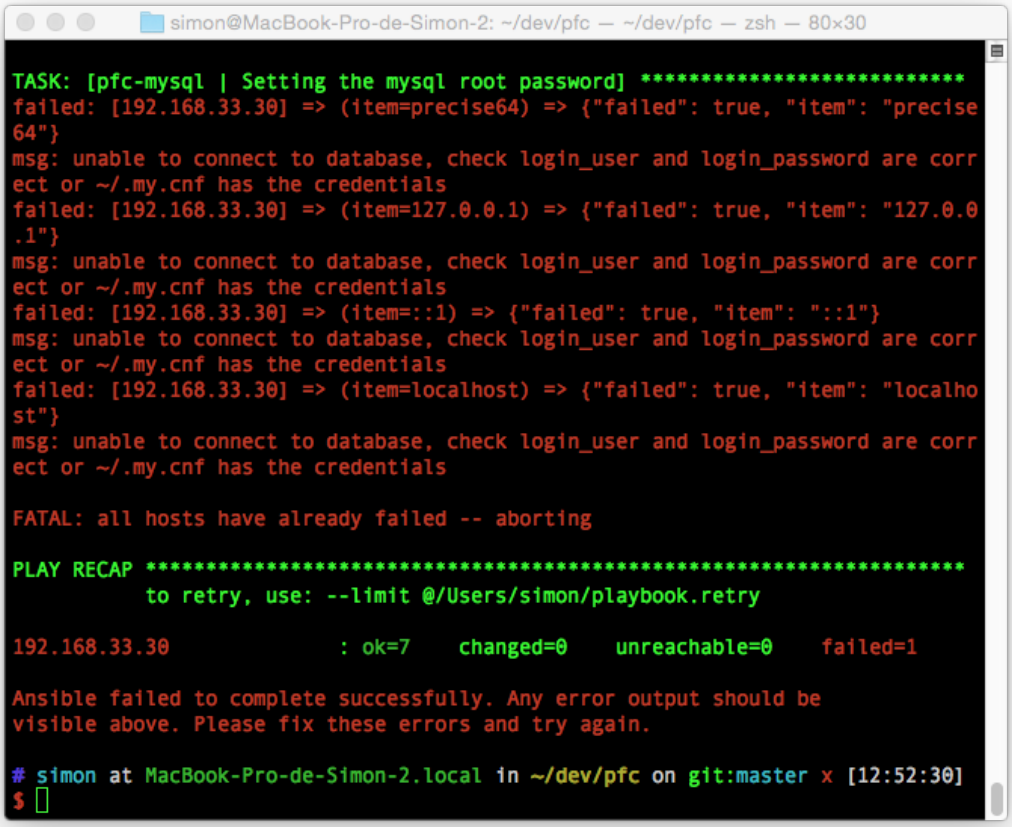
```
$ cat ansible/roles/pfc-mysql/templates/.my.cnf.j2
[client]
user=root
password={{ mysql_root_password }}
```

El siguiente y último paso será crear la tarea que subirá ese archivo al directorio *root* del host que estemos configurando con Ansible.

```
- name: Setting mysql passwordless root access
  template:
    src: root.my.cnf.j2
    dest: /root/.my.cnf
    owner: root
    group: root
    mode: 0600
```

En esta tarea *template* apreciamos algunos parámetros nuevos fáciles de ver y entender como son *owner*, *group* y *mode*. Los dos primeros no serán más que el propietario y el grupo de acceso que se debe asignar al archivo que se subirá con *template*. El tercero será la máscara de permisos que queremos sobre dicho archivo, y que al tratarse de un archivo sensible utilizaremos una máscara restrictiva dónde sólo el propietario (*root*) pueda leer y escribir el archivo.

Probemos a ejecutar nuestra tarea, dónde ya anticipo que nos llevaremos una sorpresa:



Y es que nuestro aprovisionamiento falla justo en un paso que ya habíamos completado, el de actualizar la contraseña de los usuarios *root* de la base de datos. Tiene lógica además, porque si bien hasta el momento el módulo *mysql_user* de Ansible podía conectarse a la base de datos con el usuario administrador sin contraseña, al haber creado una en el punto anterior, Ansible ya no es capaz de conectar con MySQL.

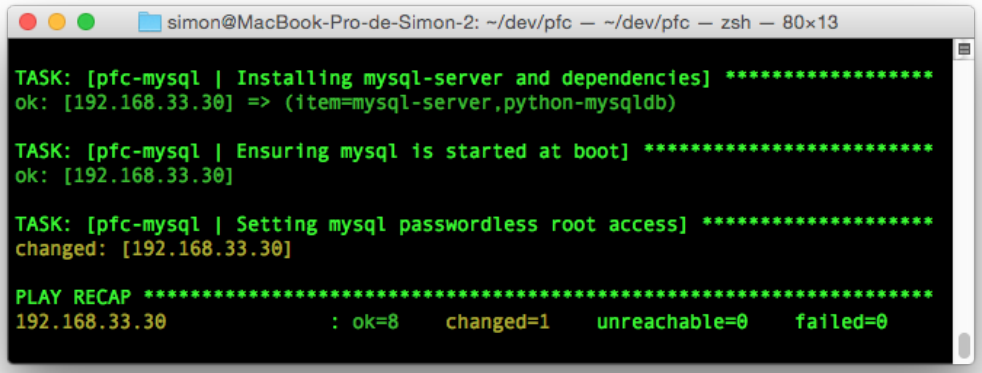
Para solucionarlo, necesitaremos precisamente la tarea que acabamos de crear, la cual permitiría al usuario *root* conectar sin contraseña. Pero, tenemos un problema, y es que no podemos llegar a la misma si la tarea anterior nos devuelve error. Para solucionarlo hay distintas alternativas:

- Podríamos mover la tarea encima de la anterior para devolverla después de nuevo detrás.
- Podríamos comentar la anterior y volver a activarla tras ejecutar el *template*.

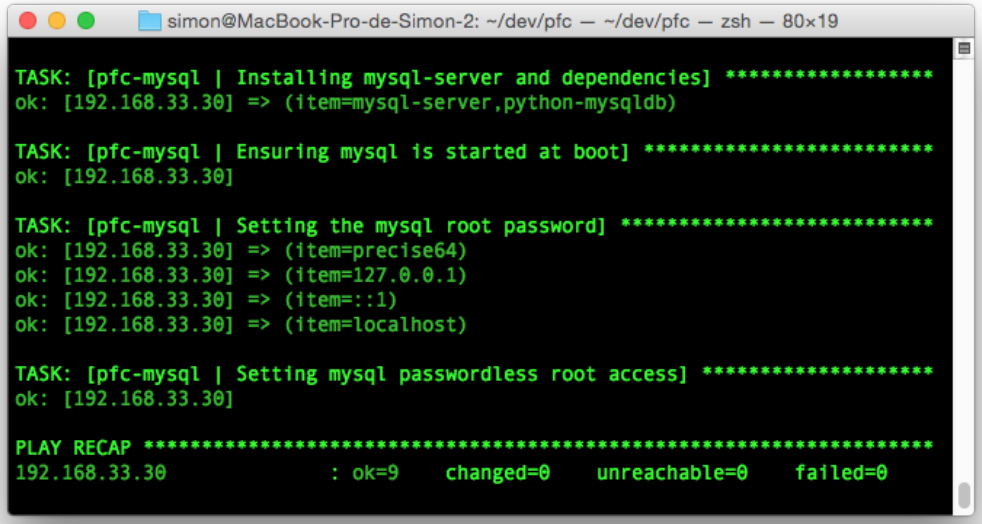
- Podríamos ejecutar un comando de Ansible para lanzar únicamente la tarea en cuestión.

Es importante dejar la tarea en su posición inicial, después de actualizar las contraseñas de *root* porque de lo contrario nos encontraríamos el mismo problema a la inversa, y es que, en la primera ejecución del rol, MySQL trataría de conectarse con una contraseña que todavía no ha sido definida en la base de datos. El orden correcto de ejecución debe ser, 1) actualizar las contraseñas, para justo después, 2) colocar el archivo *.my.cnf* en el directorio raíz de *root*. Si lo hacemos así desde el principio, el sistema se configurará sin problemas.

En este caso vamos a optar por simplemente comentar la tarea anterior y ejecutar Ansible para que lance nuestra nueva tarea sin errores:



Una vez configurado el sistema, podemos volver a dejar la tarea anterior activada y ver como nos hemos deshecho del error:



Con esto, terminamos la configuración en Ansible que nos permite instalar y configurar un servidor de MySQL en cualquier host Ubuntu / Debian destino con tan sólo ejecutar un comando.

Nuestro siguiente punto a abordar será la instalación del intérprete PHP en el host, requisito indispensable para que nuestra aplicación, no sin antes registrar la evolución del proyecto en nuestro repositorio en GitHub.

3.3.3 Instalando un intérprete PHP

Los primeros pasos para instalar un intérprete php no serán muy distintos a los que ya hemos visto al instalar nginx y MySQL. Como de costumbre, el primer paso será crear la estructura de directorios para nuestro rol:

```
#  simon  at  MacBook-Pro-de-Simon-2.local  in  ~/dev/pfc  on  git:master  o
[12:25:18]
$ mkdir -p ansible/roles/pfc-php-fpm/{defaults,handlers,tasks,templates}
```

Acto seguido, creamos y editamos el archivo *tasks/main.yml* para comenzar a añadir las tareas relacionadas con la instalación del intérprete. En nuestro caso, instalaremos PHP5-FPM[lxiv], una variante de PHP optimizada para alto rendimiento y que se suele asociar al uso con el servidor web nginx.

Como viene siendo habitual, utilizaremos el módulo de Ansible apt[lxv], para instalar los paquetes que necesitemos del intérprete. También utilizaremos un bucle de tipo *with_items* que pudimos ver en la sección de instalación de MySQL y que nos permitirá instalar varios paquetes con una única tarea:

```
# file: roles/pfc-php-fpm/defaults/main.yml
---
- name: Install php5-fpm
  apt: >
    name={{ item }}
    state=present
  with_items:
    - php5-fpm
    - php5-cli
    - php5-mysql
    - php5-gd
    - php5-curl
    - php5-mcrypt
```

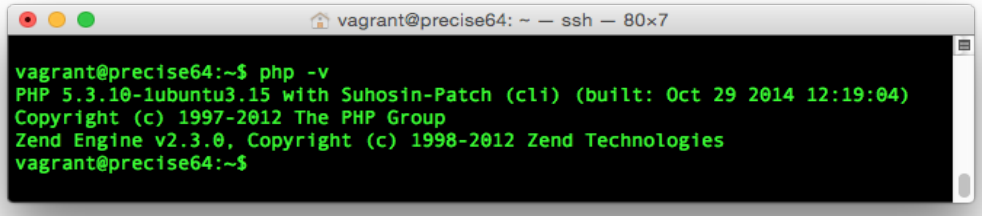
Como podemos apreciar, además de PHP5-FPM, hemos instalado otros paquetes de PHP que suelen ser requisitos habituales de aplicaciones web, como pueden ser *php5-mysql* que servirá de puente entre PHP y MySQL, *php5-gd* para manipulación de imágenes, *php5-curl* que permitirá a PHP utilizar el comando `curl` del host y *php5-mcrypt* para tareas de encriptado.

Antes de realizar la primera ejecución de `vagrant provision` recordemos que necesitaremos añadir el nuevo rol al *playbook* maestro.

```
# file: ansible/playbook.yml
---

- hosts: [pfc-vm]
  sudo: yes
  roles:
    - pfc-nginx
    - pfc-mysql
    - pfc-php-fpm
```

Una vez añadido, ya podemos lanzar el aprovisionamiento y comprobar que los distintos paquetes se instalan sin problemas:

A estas alturas ya no deberíamos tener dudas de que Ansible habrá hecho lo que le hemos pedido con diligencia, pero nunca está de más asegurar, así que entremos en la máquina virtual y comprobemos que PHP está efectivamente instalado:



Ansible sigue sin defraudarnos en el desempeño de sus tareas. Como anteriormente, añadamos una segunda tarea para iniciar el servicio PHP5-FPM al reiniciar:

```
- name: Ensuring php-fpm is started at boot
  service: >
    name=php5-fpm
```

```
    state=started
    enabled=yes
```

3.3.3.1 Configurando PHP para un entorno de desarrollo

La configuración por defecto de PHP5-FPM asume que estamos desplegándolo en un entorno de producción, por lo que trae desactivadas por defecto muchas opciones que son realmente útiles a la hora de codificar, como diversas opciones de muestreo de errores por ejemplo. En este punto desarrollaremos la adaptación de la configuración por defecto de PHP a una propia más adecuada para un entorno de desarrollo local.

El archivo de configuración de PHP, *php.ini*[lxvi], contiene ni más ni menos que 2000 líneas, por lo que nos centraremos tan sólo en aquellas configuraciones concretas que queramos ajustar en nuestro entorno de desarrollo, como las siguientes:

```
; Maximum amount of memory a script may consume (128MB)
; http://php.net/memory-limit
memory_limit = 128M

; Maximum size of POST data that PHP will accept.
; http://php.net/post-max-size
post_max_size = 8M

; Maximum allowed size for uploaded files.
; http://php.net/upload-max-filesize
upload_max_filesize = 2M

; display_errors
;   Default Value: On
;   Development Value: On
;   Production Value: Off

; display_startup_errors
;   Default Value: Off
;   Development Value: On
;   Production Value: Off

; error_reporting
;   Default Value: E_ALL & ~E_NOTICE
;   Development Value: E_ALL | E_STRICT
;   Production Value: E_ALL & ~E_DEPRECATED

; track_errors
;   Default Value: Off
;   Development Value: On
;   Production Value: Off

; When PHP displays or logs an error, it has the capability of inserting html
;  links to documentation related to that error. This directive controls
whether
```

```
; those HTML links appear in error messages or not. For performance and
security
```

```
; reasons, it's recommended you disable this on production servers.
; Note: This directive is hardcoded to Off for the CLI SAPI
; Default Value: On
; Development Value: On
; Production value: Off
; http://php.net/html-errors
html_errors = Off
```

Podemos comprobar como gran parte de ellas traen valores recomendados para producción, por lo que necesitamos una tarea que modifique ese archivo en el *host* poniendo los parámetros que consideremos oportunos.

Para hacerlo, vamos a utilizar un módulo nuevo de Ansible que no habíamos visto hasta ahora, ini_file[lxvii], y cuya función principal es precisamente la de editar archivos de tipo *ini* opción a opción y de esa forma evitar tener que crear una plantilla como hicimos en el caso de *nginx.conf*:

```
- name: Configuring php.ini
  ini_file: >
    dest=/etc/php5/fpm/php.ini
    section=PHP
    option="{{ item.option }}"
    value="{{ item.value }}"
    backup=yes
  with_items: php_ini_settings
  notify:
    - restart php-fpm
```

Dónde:

- *dest* será el archivo .ini a modificar**.**
- *section* será la sección dentro del archivo a modificar.
- *option* identificará a la opción a modificar.
- *value* será el valor a emplear en tal opción.
- *backup* especificará que se haga una copia de seguridad previa del archivo.

Como tenemos que modificar varias opciones simultáneamente, utilizaremos también un bucle *with_items*, al cual le pasamos una lista de elementos referenciados a una variable, *php_ini_settings*, en lugar de listar cada *item* en la propia tarea.

Este listado de *items* lo guardaremos en *defaults/main.yml*, y es curioso porque no es una lista simple, sino que se trata de nuestro primer diccionario[lxviii] en formato YAML, el cual define pares clave/valor:

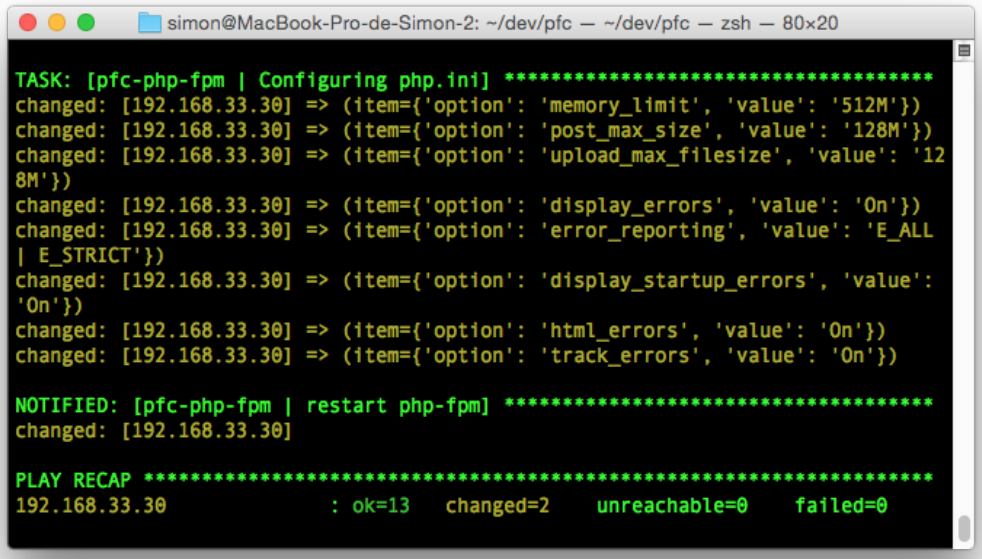
```
# file: roles/pfc-php-fpm/defaults/main.yml
---
php_ini_settings:
  - option: "memory_limit"
    value: "512M"
  - option: "post_max_size"
    value: "128M"
  - option: "upload_max_filesize"
    value: "128M"
  - option: "display_errors"
    value: "On"
  - option: "error_reporting"
    value: "E_ALL | E_STRICT"
  - option: "display_startup_errors"
    value: "On"
  - option: "html_errors"
    value: "On"
  - option: "track_errors"
    value: "On"
```

Esto es lo que nos habilitará para usar las variables en la forma *{{ item.option }}*, *{{ item.value }}* en la tarea que estamos codificando.

Por último, ya vimos en la configuración de nginx que cada vez que modificamos la configuración de un servicio, necesitamos reiniciarlo para que los cambios tomen efecto. De ahí la última parte de la tarea inicial, que efectúa una acción *notify* llamando a un *handler* para que reinicie el servicio. Manejador que no hemos definido, pero que ahora añadimos en el archivo *handlers/main.yml*:

```
# file: roles/pfc-php-fpm/handlers/main.yml
---
- name: restart php-fpm
  service: name=php5-fpm state=restarted
```

Hecho lo cual, no queda más que ejecutar el aprovisionamiento para comprobar si todo funciona como es debido:



Con la instalación de PHP5-FPM, hemos concluido el *setup* de los tres requisitos fundamentales a nivel de arquitectura: un servidor web, un motor de base de datos, y un intérprete de PHP.

No sólo eso, sino que todo nuestro entorno se levanta vía código con una sola instrucción, `vagrant provision`. El siguiente paso será descargar el código que queremos ejecutar sobre el sistema, e igual que hemos hecho con los requisitos, trataremos de automatizarlo con Ansible.

3.3.4 Instalando Drupal y configurándolo con Ansible

El último paso para lograr nuestro objetivo de poder levantar un entorno virtualizado completo con un sólo comando, será instalar y configurar la aplicación PHP sobre la que queramos desarrollar, en este caso Drupal.

Podríamos hacerlo en la máquina virtual recurriendo a Ansible, como hemos hecho con el resto de paquetes que hemos instalado, pero se alejaría de nuestro propósito, que no es sino poder compartir el código fuente entre varios desarrolladores en un entorno común. Decimos que se alejaría, porque si instalamos Drupal en el interior de la máquina virtual de cada desarrollador, éstos no podrían compartir la misma base de código con el resto del equipo. Cada instalación sería propia a cada máquina virtual independiente.

Nuestro objetivo será por tanto, conseguir que el código sobre el que trabajemos, esté tanto disponible en el repositorio compartido, como en la máquina virtual local que lancemos para trabajar sobre él. Afortunadamente, Vagrant está preparado para facilitarnos esta tarea como veremos a continuación.

3.3.4.1 Descargando Drupal

En el momento de escribir estas líneas, la última versión estable de Drupal es la 7.33, la cual encontramos en la página de descargas de Drupal.org[lxix], y será ésta la que utilizaremos a lo largo del proceso de instalación.

La descargamos, y la incorporamos al raíz del repositorio sobre el que venimos trabajando en nuestro ordenador local, en una carpeta de nombre *drupal*, quedando de esta forma nuestro directorio raíz del repositorio:

```
#  simon  at  MacBook-Pro-de-Simon-2.local  in  ~/dev/pfc  on  git:master  x
[22:08:53]
$ tree -L 2
.
├── README.md
├── Vagrantfile
├── ansible
│   ├── hosts
│   ├── playbook.yml
│   └── roles
└── drupal
    ├── CHANGELOG.txt
    ├── COPYRIGHT.txt
    ├── INSTALL.mysql.txt
    ├── INSTALL.pgsql.txt
    ├── INSTALL.sqlite.txt
    ├── INSTALL.txt
    ├── LICENSE.txt
    ├── MAINTAINERS.txt
    ├── README.txt
    ├── UPGRADE.txt
    ├── authorize.php
    ├── cron.php
    ├── includes
    ├── index.php
    ├── install.php
    ├── misc
    ├── modules
    ├── profiles
    ├── robots.txt
    ├── scripts
    ├── sites
    ├── themes
    ├── update.php
    ├── web.config
    └── xmlrpc.php
```

Una vez añadido el código de Drupal, tan sólo tendríamos que ejecutar tres comandos para añadirlo al repositorio compartido del proyecto:

```
$ git add .
```

```
$ git commit -m "adding drupal code"
$ git push
```

De esta forma tendríamos, por una parte el código de la aplicación en el directorio *drupal*, y por otra, el código para configurar la máquina virtual en el directorio *ansible* del mismo repositorio. El siguiente paso será utilizar Ansible para instalarlo en el sistema, para lo que crearemos un nuevo rol.

3.3.4.2 Configurando Drupal con Ansible

Como con los anteriores roles que hemos creado, los primeros pasos serán, crear la estructura de directorios del rol que vamos a crear para configurar, y añadir la invocación al rol en el archivo *playbook.yml*:

```
#  simon  at  MacBook-Pro-de-Simon-2.local  in  ~/dev/pfc  on  git:master  o
[12:58:16]
$ mkdir -p ansible/roles/pfc-drupal-conf/{defaults,handlers,tasks,templates}

#  simon  at  MacBook-Pro-de-Simon-2.local  in  ~/dev/pfc  on  git:master  o
[12:58:33]

$ cat ansible/playbook.yml
# ansible/playbook.yml
---

- hosts: [pfc-vm]
  sudo: yes
  roles:
    - pfc-nginx
    - pfc-mysql
    - pfc-php-fpm
    - pfc-drupal-conf
```

Una vez hecho esto, comencemos a definir las tareas del rol en *tasks/main.yml* La primera de ellas, será la creación de un usuario MySQL que será el que enlace Drupal con el servidor de base de datos:

```
# ansible/roles/pfc-drupal-conf/tasks/main.yml
---

- name: Creating a MySQL user for drupal
  mysql_user: >
    name={{ drupal_db_user }}
    host={{ item }}
    priv={{ drupal_db }}.*:ALL
    password={{ drupal_db_password }}
  with_items:
```

```
    - 127.0.0.1
    - ::1
    - localhost
```

Dónde *drupal_db_user*, *drupal_db* y *drupal_db_password* serán variables que definiremos en *defaults/main.yml* y veremos más adelante.

Una vez creado el usuario con el que Drupal interactuará con MySQL, realizaremos tres tareas de forma simultánea:

- La creación de la base de datos de Drupal.
- La creación de un archivo *settings.php* de configuración de Drupal.
- La ejecución de un script que cargue las primeras tablas de Drupal en la base de datos.

Las tres tareas las completaremos apoyándonos en Drush[lxx], una utilidad de línea de comandos que permite ejecutar diversas acciones sobre una instalación de Drupal, como por ejemplo, configurarla por primera vez.

Drush se distribuye en los canales oficiales de muchas distribuciones, entre ellas Ubuntu, por lo que instalarlo será tan sencillo como instalar cualquier otro paquete. Aprovechamos también para instalar el paquete *sendmail*, que nos permitirá enviar correos electrónicos y es necesario para ciertas funciones de Drush:

```
- name: Installing drush
  apt: >
     name={{ item }}
     state=latest
  with_items:
    - drush
    - sendmail
```

Una vez instalado, utilizaremos el comando "*drush si" (site install)*, el cual configurará nuestra instalación de acuerdo a los parámetros especificados:

```
- name: Configuring drupal settings.php with drush
  shell: >
    drush si standard -y
    --site-name="{{ drupal_site_name }}"
    --account-name={{ drupal_admin_name }}
    --account-pass={{ drupal_admin_pass }}
    --db-url=mysql://{{  drupal_db_user  }}:{{  drupal_db_pass  }}@localhost/{{
drupal_db }}
    chdir={{ drupal_path }}
    creates={{ drupal_path }}/sites/default/settings.php
```

Dónde:

- *site-name* será el nombre con el que identificar a nuestro sitio
- *account-name* será el nombre del usuario administrador *(uid=1)*
- *account-pass* será la contraseña del usuario administrador
- *db-url* será una línea de configuración de MySQL dónde especificaremos usuario, contraseña y nombre de la base de datos sobre la que realizar la instalación
- *chdir* indica que cambiemos al directorio base de drupal como paso previo a la tarea
- *creates* indica que esta tarea sólo se ejecute la primera vez (mientras no esté creado el archivo referenciado que justo esta tarea con `drush si` crea)

La información requerida por tales parámetros, las variables, las definimos en el archivo *defaults/main.yml*:

```
# ansible/roles/pfc-drupal.conf/defaults/main.yml
---
drupal_db: "our_drupal_db"
drupal_db_user: "drupal_user"
drupal_db_password: "drupal_pass"
drupal_site_name: "our_site_name"
drupal_admin_name: "admin"
drupal_admin_pass: "admin"
drupal_path: "/vagrant/drupal"
```

La más interesante de ellas es *drupal_path*, y es que referencia a un directorio */vagrant* en la máquina virtual del que todavía no hemos hablado.

3.3.4.3 El directorio compartido /vagrant

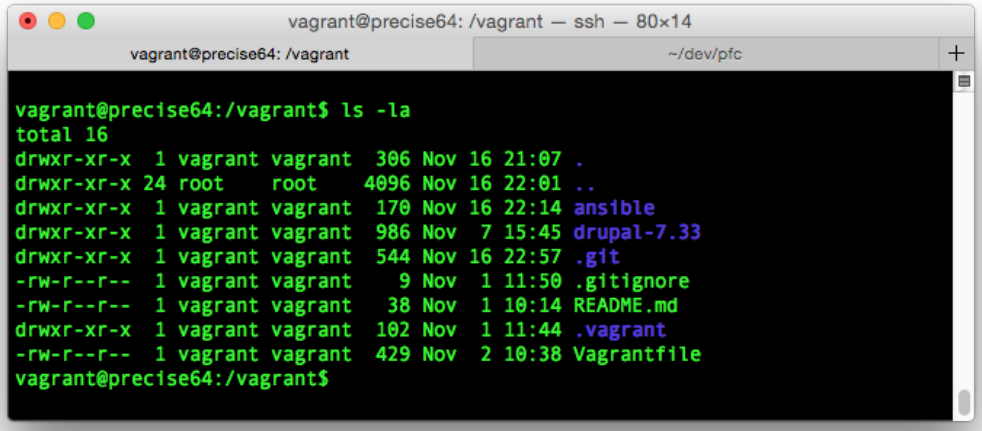
Si hacemos memoria, recordaremos que el código de Drupal reside en nuestro repositorio, si bien en ningún punto lo hemos trasladado a la máquina virtual, pues como hemos comentado, esto evitaría poder compartirlo con el resto del equipo de desarrollo.

Pero al mismo tiempo, si no tenemos de alguna forma el código de la aplicación en la máquina virtual, nunca podremos ejecutarlo sobre la arquitectura que hemos definido. Entonces, ¿para qué tanto trabajo?

Necesitamos, por tanto, un método que nos permita compartir el código de con la máquina virtual, de forma que, aquella parte que modifiquemos en nuestro ordenador local, se corresponda con el código en la máquina virtual, y viceversa. La palabra clave, será “compartir”.

Y es que Vagrant permite que compartamos directorios entre el anfitrión y la máquina virtual de distintas formas utilizando una característica denominada *Synced Folders*[lxxi]. No sólo eso, sino que además, el directorio que contiene el archivo Vagrantfile en el anfitrión, es decir, el directorio de nuestro proyecto, se encuentra compartido automáticamente y por defecto en la ruta */vagrant* dentro de la máquina virtual.

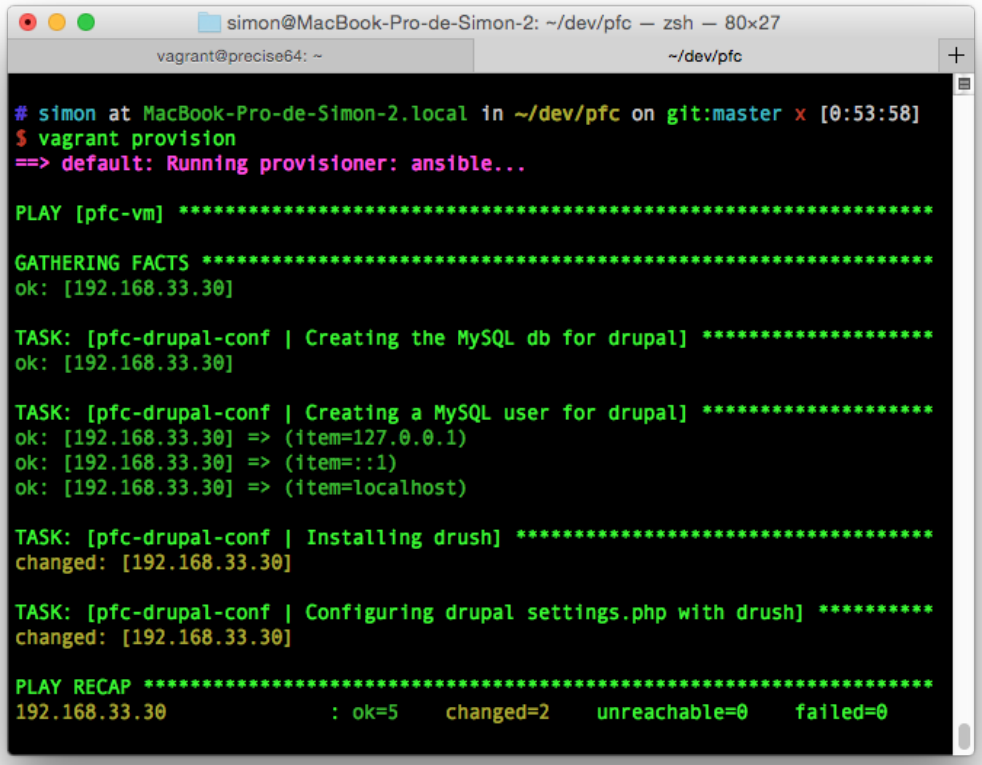
Comprobémoslo haciendo un listado de los archivos de dicho directorio en la máquina virtual:



Como vemos, sin haberlos copiado o movido en ningún momento, nuestra máquina virtual puede acceder a los archivos que componen nuestro proyecto. Esto es lo que permitirá a los desarrolladores trabajar sobre el código fuente en sus respectivas máquinas, abstrayéndolo a su vez de la máquina virtual.

Cada vez que un desarrollador termine una nueva funcionalidad, subiría el código al repositorio común, y el resto de desarrolladores tan sólo tendrían que hacer un `git pull` para cargarlo en sus respectivas máquinas virtuales.

Siguiendo dónde nos habíamos quedado, llega el momento de probar nuestras recién creadas tareas de configuración de Drupal:



En principio ya hemos configurado la base de datos y Drupal para que se comuniquen entre sí. Sin embargo, todavía nos falta una pieza por encajar, y es que debemos configurar nuestro servidor web para que sirva nuestro código.

3.3.4.4 Configurando nginx para servir Drupal

El último paso de configuración será avisar al servidor web de que hay un nuevo sitio que tiene que servir. Para eso, necesitaremos crear un archivo de configuración del sitio, *drupal-nginx.conf*, copiarlo, activarlo en la máquina virtual, y recargar el servidor web para que así sirva el nuevo sitio.

Definamos en primer lugar el archivo *templates/nginx-vhost.conf.j2*:

```
# simon at MacBook-Pro-de-Simon-2.local in ~/dev/pfc on git:master x
[1:15:45]
$ cat ansible/roles/pfc-drupal-conf/templates/nginx-vhost.conf.j2

server {

  listen    80;
  server_name dev-drupal.org;
  root {{ drupal_path }};
  index index.php;
```

```
  location ~ \..*/.*\.php$ {
    return 403;
  }

  location / {
    try_files $uri @rewrite;
  }

  location @rewrite {
    rewrite ^/(.*)$ /index.php?q=$1;
  }

  location ~ \.php$ {
    fastcgi_split_path_info ^(.+\.php)(/.+)$;
    include fastcgi_params;
    fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    fastcgi_pass 127.0.0.1:9000;
  }

}
```

La primera parte son elementos comunes de cualquier archivo de configuración de un servidor web como el puerto de escucha, el nombre del servidor o dónde se encuentra la carpeta raíz del sitio.

La segunda parte del mismo es más interesante, y corresponde a la configuración de PHP5-FPM con nginx, la cual podemos importar desde su propia documentación[lxxii].

Una vez creado el archivo, necesitaremos copiarlo a la máquina virtual, y como hemos visto que utilizamos variables en el mismo, echaremos mano de una tarea de tipo *template* que ya vimos con anterioridad:

```
- name: Add Apache virtualhost for Drupal.
  template: >
    src=drupal-vhost.conf.j2
    dest=/etc/nginx/sites-available/drupal-vhost.conf
    owner=root
    group=root
    mode=644
  notify: reload nginx
```

También necesitaremos activar el sitio en el servidor web, creando un enlace simbólico del archivo recién subido a su carpeta *sites-enabled*:

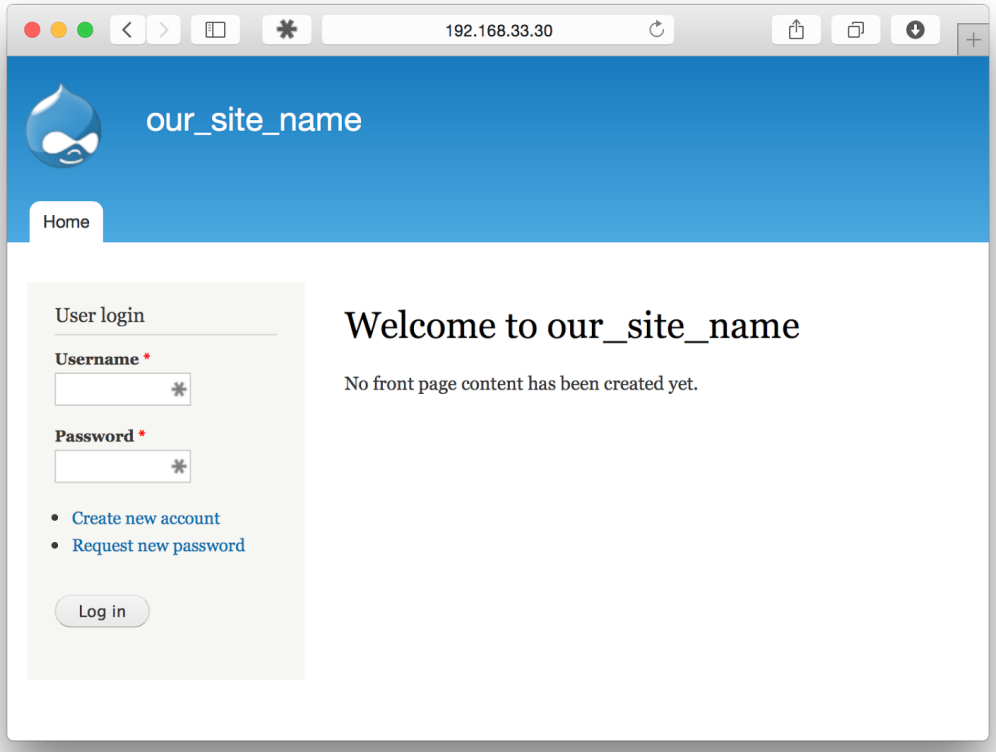
```
- name: Enabling nginx site and reload the webserver
  file: >
    src=/etc/nginx/sites-available/nginx-vhost.conf
    dest=/etc/nginx/sites-enabled/nginx-vhost.conf
```

```
    state=link
  notify: reload nginx
```

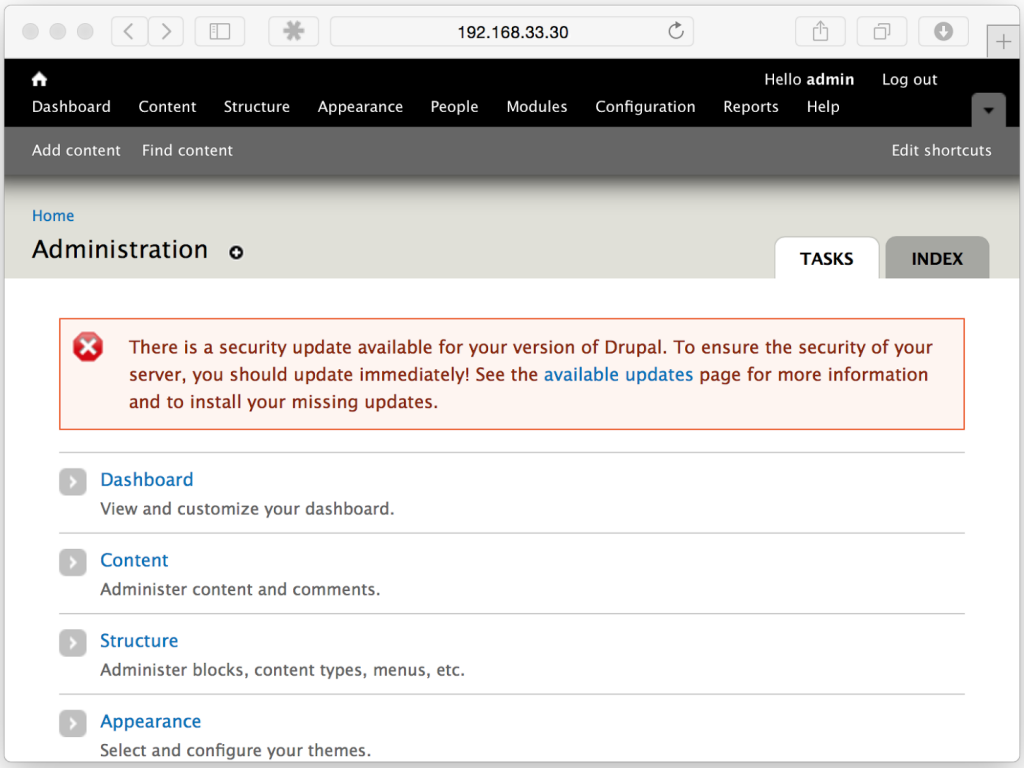
Y por último, tan sólo necesitaremos desactivar el sitio *default*, ya que nos puede molestar a la hora de navegar por nuestro proyecto:

```
- name: Disabling nginx default site
  file: >
    path=/etc/nginx/sites-enabled/default
    state=absent
  notify: reload nginx
```

Ejecutadas esas tareas con *`vagrant provision`*, el resultado no será otro sino la correcta configuración de nuestra aplicación en la máquina virtual, la cual ya debería ser accesible directamente a través de cualquier navegador del ordenador local del desarrollador, en la ip que definimos en un primer momento en el fichero *Vagrantfile* (192.168.33.30):



Como podemos observar, Drupal se ha levantado con los parámetros que configuramos en Ansible, como el nombre del sitio, *"our_site_name"*. Deberíamos poder entrar a la parte de administración con el usuario *"admin"* y contraseña *"admin"* que ya definimos:



Efectivamente, podemos entrar a la sección de administración de nuestra aplicación recién instalada en la máquina virtual. Y hete aquí, que Drupal nos avisa de un problema de seguridad en nuestro código, y es que en el tiempo que ha transcurrido mientras completábamos el proyecto, el equipo de seguridad de Drupal descubrió una nueva vulnerabilidad y lanzó una nueva versión, la 7.34, que lo solucionaba.

Lo cual nos da una excusa perfecta, para demostrar cómo un desarrollador de nuestra empresa de desarrollo puede solucionar la vulnerabilidad en una máquina virtual, y propagarla al resto de desarrolladores con suma facilidad, lo cual abordaremos en el próximo capítulo, en el cual podremos ver en un caso práctico, los beneficios que nos aporta haber invertido todo el trabajo que hemos hecho hasta ahora.

4 Prueba y conclusiones

Comenzábamos el proyecto que nos ocupa observando una serie de problemas comunes que encontrábamos en las empresas tradicionales de desarrollo web como eran la complicada gestión de dependencias en el ordenador de cada programador, la configuración óptima de los servicios necesarios para ejecutarlos, o incluso el alto coste que para una empresa de este tipo, conlleva incorporar un nuevo trabajador debido a estos costes de integración del mismo. Veíamos también como en todos estos problemas, la figura del administrador de sistemas era clave, creando a su vez un nuevo problema, que no es otro que la dependencia del mismo.

Nuestro objetivo principal consistía en proponer un flujo de trabajo alternativo al habitual, que permitiera solventar tales problemas, aumentando de esa forma la productividad de cualquier empresa u organización que los adoptase. Para lograrlo, debíamos alcanzar tres metas claves.

Debíamos ser capaces de:

- Lanzar y configurar una máquina virtual sobre el que ejecutar nuestro código
- Instalar los requisitos necesarios (servicios y configuraciones) en esa máquina virtual
- Configurar e inicializar Drupal para poder trabajar sobre su código

Dedicamos el segundo capítulo del proyecto a introducir distintas herramientas de código libre como VirtualBox, Vagrant y Ansible, las cuales utilizaríamos para conseguir ese anhelado nuevo flujo de trabajo.

Una vez introducidas las herramientas, en el tercer capítulo nos poníamos manos a la obra utilizándolas para definir la infraestructura necesaria para poder cargar una instalación de Drupal. En primer lugar introducíamos GitHub que utilizaríamos para alojar y compartir nuestro repositorio. A continuación utilizábamos Vagrant para crear una máquina virtual en VirtualBox mediante línea de comandos.

Creada la máquina virtual, instalábamos y configurábamos los servicios requeridos por nuestra aplicación, un servidor web (nginx), un motor de base de datos MySQL y intérprete de PHP (PHP5-FPM) con Ansible. Terminábamos el capítulo explicando como compartir el código fuente de nuestro desarrollo con la máquina virtual, realizando los pasos necesarios para dejar listo el código de forma que cualquier programador pudiera trabajar sobre ella.

Llegábamos así al final del capítulo haciendo *login* en nuestra instalación virtual de Drupal, y descubriendo que, mientras desarrollábamos el proyecto, los desarrolladores del popular gestor de contenido habían lanzado una actualización de seguridad sobre el código.

Será de este punto de dónde partamos en este cuarto y último capítulo, dónde probaremos la infraestructura creada y aplicaremos, en base a un caso práctico, el nuevo flujo de trabajo propuesto en este proyecto.

4.1 Caso práctico del nuevo flujo de trabajo propuesto

Como comentábamos líneas atrás, vamos a aprovechar la advertencia de seguridad observada tras la instalación de Drupal, para representar un caso de uso habitual en cualquier empresa de desarrollo de software web, la necesidad de parchear un proyecto antiguo. En este caso la razón es la necesidad de aplicar una actualización de seguridad, pero los motivos pueden ser variados, como solucionar un bug generado por una actualización de una librería en el servidor, o incluso una nueva funcionalidad demandada por el cliente.

El escenario base será el de una empresa de desarrollo con años de proyectos a sus espaldas, con un equipo de desarrolladores con una rotación elevada, lo que hace que en algunos proyectos antiguos no quede nadie del equipo original que los desarrolló.

En este escenario, cualquier incidencia como las que hemos comentado, puede suponer, y en muchos casos supone, un auténtico trauma para la compañía, pues el simple hecho de poder reproducir el entorno en el que se creó la aplicación en un primer momento puede conllevar decenas de horas enterradas por parte del equipo de desarrollo.

Sólo cuando el equipo ha sido capaz de reproducir el entorno en el que corría, comienza efectivamente el verdadero trabajo productivo de los desarrolladores, aplicar las modificaciones necesarias al código para solucionar la incidencia.

Frente a esta, lamentablemente a día de hoy la más usual de las formas de organización de las empresas de desarrollo web, veremos a continuación, como se resolvería la misma situación en el caso de que se hubiera adoptado el flujo de trabajo que proponemos.

4.1.1 Reproduciendo el entorno de la aplicación

Para validar la efectividad del nuevo flujo de desarrollo partiremos de cero. Imaginemos que uno de los miembros del equipo de desarrollo descubre el boletín[lxxiii] que anunciaba la vulnerabilidad de seguridad de Drupal, poniéndola acto seguido en conocimiento del resto del equipo.

El equipo evalúa el problema y decide que es necesario intervenir, aplicando la actualización requerida, pero se trata de un desarrollo de hace años, por lo que ningún programador tiene un entorno creado en su ordenador con la desarrollo a parchear funcional. Tendremos por tanto, en primer lugar, que reproducir el entorno base sobre el que actuar.

Afortunadamente para el equipo, cuando se desarrolló la aplicación a parchear, se hizo siguiendo el flujo de trabajo propuesto a lo largo de este proyecto, por lo que toda la infraestructura del mismo se haya reproducida en el repositorio junto al código fuente.

Siendo así, uno de los desarrolladores, el protagonista de nuestra historia, comunica al resto del equipo que tiene un hueco y se puede encargar de parchear el problema.

Partiendo como parte de cero, lo primero que necesita nuestro desarrollador será bajarse el repositorio con el código fuente del proyecto:

```
# simon at MacBook-Pro-de-Simon-2.local in ~/dev on git:master x [14:12:21]
$ git clone git@github.com:pfc2014/pfc.git
Cloning into 'pfc'...
remote: Counting objects: 1406, done.
remote: Compressing objects: 100% (12/12), done.
remote: Total 1406 (delta 4), reused 0 (delta 0)
Receiving objects: 100% (1406/1406), 3.29 MiB | 762.00 KiB/s, done.
Resolving deltas: 100% (221/221), done.
Checking connectivity... done.
```

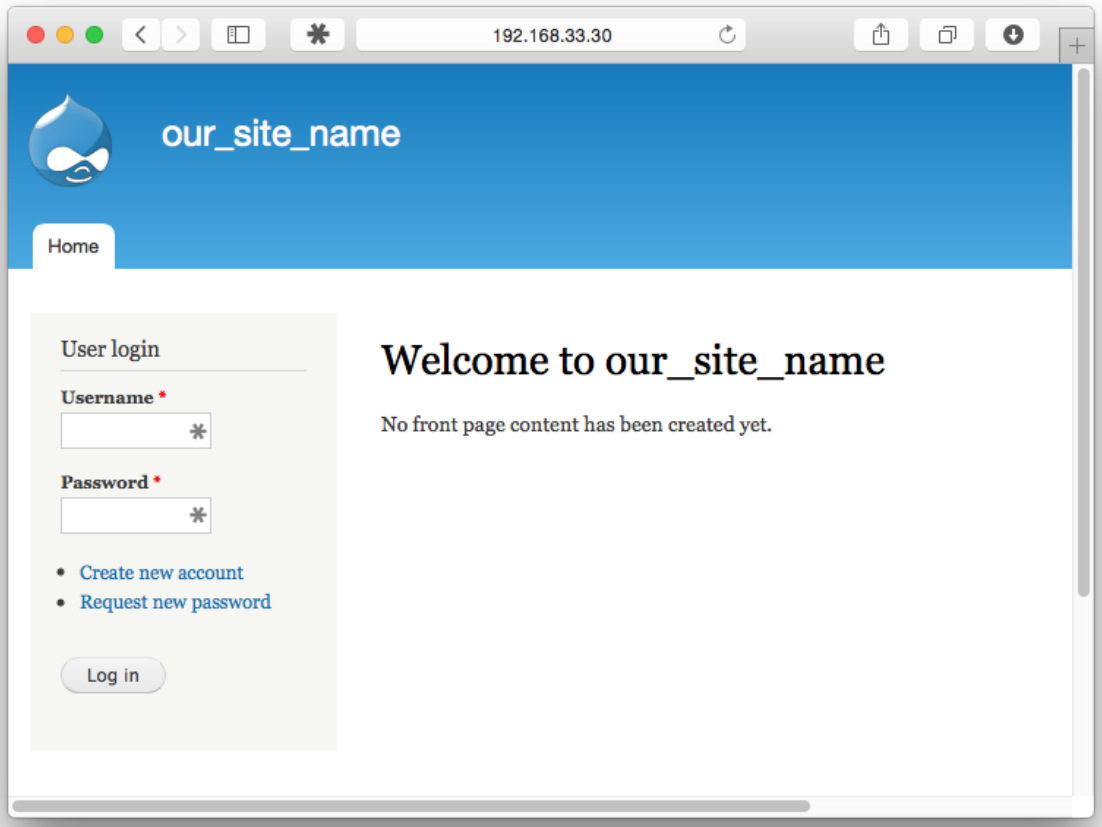
El siguiente paso, será entrar al directorio y levantar la máquina virtual automáticamente con *`vagrant up`*:

```
# simon at MacBook-Pro-de-Simon-2.local in ~/dev/pfc on git:master o
[14:13:49]
$ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Importing base box 'hashicorp/precise64'...
==> default: Matching MAC address for NAT networking...
…
…
…
NOTIFIED: [pfc-nginx | reload nginx]
****************************************
changed: [192.168.33.30]

NOTIFIED: [pfc-php-fpm | restart php-fpm]
**********************************
changed: [192.168.33.30]

PLAY RECAP
********************************************************************
192.168.33.30 : ok=20 changed=16 unreachable=0 failed=0

# simon at MacBook-Pro-de-Simon-2.local in ~/dev/pfc on git:master x
[14:20:19]
```

Hemos acortado voluntariamente el output del comando para hacerlo más legible. La salida completa se puede ver en el Anexo 2.

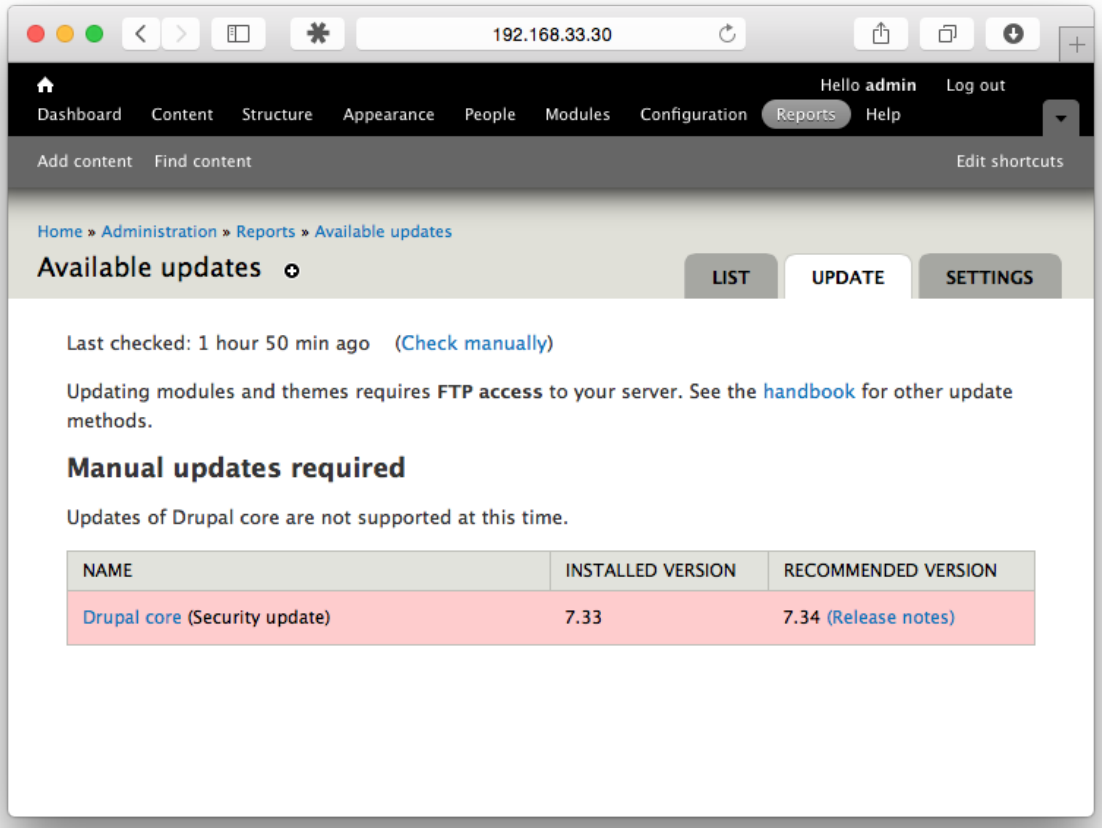
Si nos fijamos en el bloque de código anterior, podemos ver que hemos dejado intencionadamente la siguiente línea de información que nos ofrece nuestro terminal, en la cual podemos ver al final de la misma la hora a la que ha terminado el comando anterior, las 14:20. Comparando con la hora que nos muestra el primer comando del primer bloque, el de descarga del repositorio, las 14:12, podemos ver que todo el proceso ha tenido una duración de apenas ocho minutos. A partir de este momento comienza el verdadero trabajo productivo, aquel que aporta valor, por parte del programador.

Volviendo al caso práctico, comprobemos que todo se ha instalado correctamente en la máquina virtual de nuestro desarrollador. Nada más sencillo que probar si es accesible en la dirección 192.168.33.30 del navegador (la dirección asignada en el fichero *Vagrantfile* del repositorio).



4.1.2 Trabajo efectivo sobre el código

Nuestro desarrollador ya tiene Drupal levantado en su máquina virtual recién creada. El siguiente paso consiste en aplicar la actualización de seguridad demandada por el gestor de contenidos en su propio *backend* de administración.

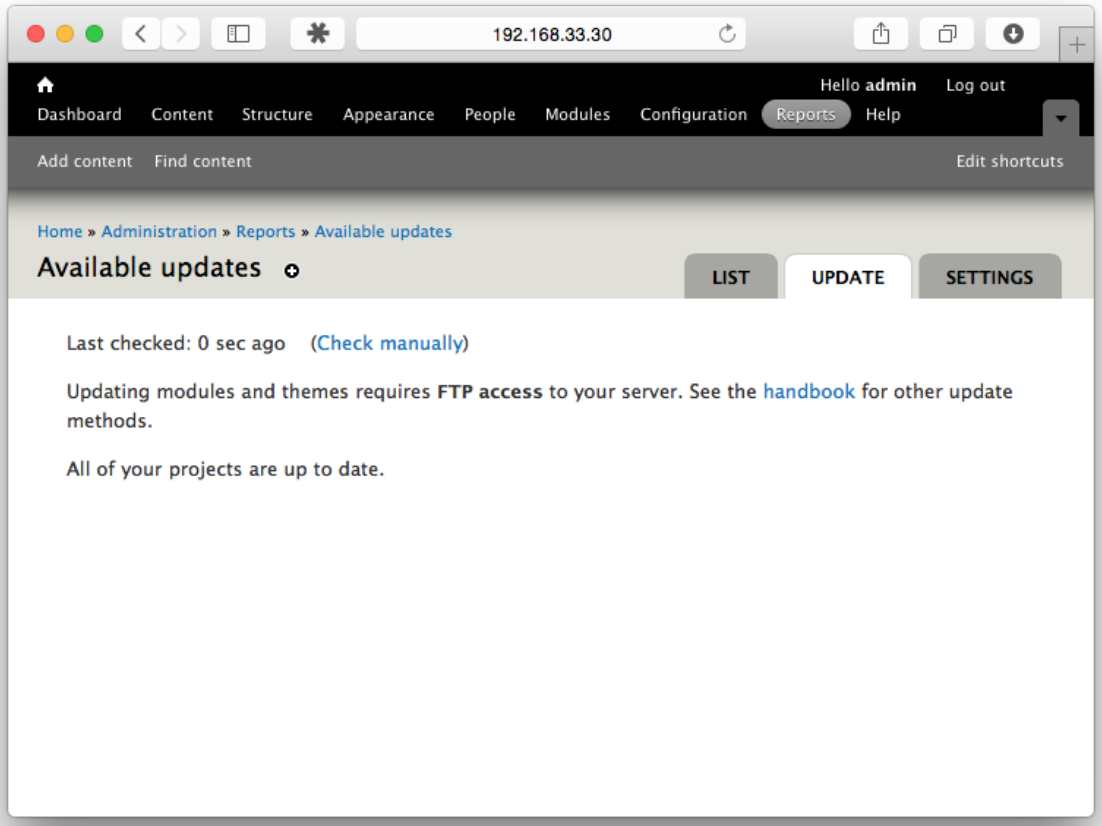


Para hacerlo, nuestro desarrollador deberá descargar el código de la nueva versión de Drupal, 7.34, y reemplazar el actual.

```
# simon at MacBook-Pro-de-Simon-2.local in ~/dev/pfc on git:master x
[14:22:15]
$ wget http://ftp.drupal.org/files/projects/drupal-7.34.tar.gz
--2014-11-29 16:12:17-- http://ftp.drupal.org/files/projects/drupal-
7.34.tar.gz
Resolving ftp.drupal.org... 140.211.166.134
Connecting to ftp.drupal.org|140.211.166.134|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3229858 (3.1M) [application/x-gzip]
Saving to: 'drupal-7.34.tar.gz'

100%[======================================>] 3,229,858 1.14MB/s in 2.7s

2014-11-29 16:12:20 (1.14 MB/s) - 'drupal-7.34.tar.gz' saved
[3229858/3229858]


# simon at MacBook-Pro-de-Simon-2.local in ~/dev/pfc on git:master x
[14:22:40]
$ tar xzf drupal-7.34.tar.gz
```

```
#  simon  at  MacBook-Pro-de-Simon-2.local  in  ~/dev/pfc  on  git:master  x
[14:22:51]

$ ls -la
total 6336
drwxr-xr-x  11 simon  staff   374B 29 nov 16:12 .
drwxr-x---  20 simon  staff   680B 29 nov 14:12 ..
drwxr-xr-x  13 simon  staff   442B 29 nov 16:12 .git
-rw-r--r--   1 simon  staff     9B 29 nov 14:12 .gitignore
drwxr-xr-x   3 simon  staff   102B 29 nov 14:13 .vagrant
-rw-r--r--   1 simon  staff    38B 29 nov 14:12 README.md
-rw-r--r--   1 simon  staff   429B 29 nov 14:12 Vagrantfile
drwxr-xr-x   5 simon  staff   170B 29 nov 14:12 ansible
drwxr-xr-x  29 simon  staff   986B 29 nov 14:12 drupal
drwxr-xr-x  29 simon  staff   986B 19 nov 21:38 drupal-7.34
-rw-r--r--   1 simon  staff   3,1M 19 nov 21:38 drupal-7.34.tar.gz

#  simon  at  MacBook-Pro-de-Simon-2.local  in  ~/dev/pfc  on  git:master  x
[14:23:53]
$ cp -rp drupal-7.34/* drupal/
```

Tras copiar el código desde la carpeta de la nueva versión a la nueva, ya deberíamos tener la actualización de seguridad aplicada. Refresquemos la página de actualizaciones para ver si es así:



Efectivamente, así es. Drupal ha dejado de quejarse y se encuentra actualizado a la última versión.

Lo único que le queda por hacer a nuestro usuario es propagar el cambio al resto del equipo, subiendo el código recién modificado al repositorio, no sin antes eliminar algunos de los archivos temporales que hemos creado en el proceso.

```
#  simon  at  MacBook-Pro-de-Simon-2.local  in  ~/dev/pfc  on  git:master  x
[14:25:20]
$ ls -l
total 6328
-rw-r--r--   1 simon  staff       38 29 nov 14:12 README.md
-rw-r--r--   1 simon  staff      429 29 nov 14:12 Vagrantfile
drwxr-xr-x   5 simon  staff      170 29 nov 14:12 ansible
drwxr-xr-x  29 simon  staff      986 29 nov 14:12 drupal
drwxr-xr-x  29 simon  staff      986 19 nov 21:38 drupal-7.34
-rw-r--r--   1 simon  staff  3229858 19 nov 21:38 drupal-7.34.tar.gz

#  simon  at  MacBook-Pro-de-Simon-2.local  in  ~/dev/pfc  on  git:master  x
[14:25:36]
$ rm -rf drupal-7*
```

Una vez dejado el directorio limpio, tan sólo nos queda subir el código al repositorio para ponerlo a disposición del resto de miembros del equipo:

```
#  simon  at  MacBook-Pro-de-Simon-2.local  in  ~/dev/pfc  on  git:master  x
[14:26:20]
$ git add .
$ git commit -m "applying security update drupal-7.34"
$ git push
```

En total, nuestro desarrollador ha invertido menos de 15 minutos en resolver un problema que con las técnicas tradicionales de desarrollo web hubiera podido llevar horas, sino días. La hasta ahora tediosa tarea de parchear una aplicación antigua, se ha convertido en algo que se puede realizar en el tiempo que te tomas un café.

4.2 Conclusiones

Comenzábamos este proyecto analizando la dualidad de funciones entre dos departamentos o roles dentro de una empresa de desarrollo orientado a web, y por extensión, también a cualquier arquitectura cliente-servidor. Por una parte, encontrábamos a los desarrolladores, aquellos que codifican las aplicaciones, y por otra, a los administradores de sistemas, quienes son los responsables de desplegar las aplicaciones desarrolladas por los primeros en las máquinas a su cargo.

Nos centrábamos especialmente en la función del administrador de sistemas, una posición que históricamente ha quedado al margen de la evolución de herramientas de desarrollo y compartición de código que tanto han progresado desde el lado de los programadores en los últimos tiempos, lo que a su vez aumentaba la división entre dichos roles.

Veíamos también los problemas que esta separación de funciones provocaba dentro de un equipo, y como una empresa podía ver afectada su productividad a varios niveles, ya fuera por el alto coste que suponía incorporar a un nuevo miembro al equipo, por la gestión de los distintos requerimientos de las aplicaciones, por la configuración de las mismas o la dependencia del administrador de sistemas en todos estos procesos.

Nuestro objetivo era plantear un flujo de desarrollo alternativo al tradicional que encontramos en muchas empresas de desarrollo web en España. Un flujo que permitiera sacar del ostracismo al administrador de sistemas e incorporarlo al flujo habitual de desarrollo de los programadores, cohesionando de esa forma al equipo y eliminando las ineficiencias que supone tener ambos roles separados.

A lo largo del proyecto hemos ido introduciendo diversas herramientas en busca de este objetivo. Primero VirtualBox, que nos permitía crear máquinas virtuales con sistemas operativos distintos a los que cada desarrollador tuviera en su propio ordenador, pudiendo emular de esa forma sistemas remotos. En segundo lugar Vagrant, que nos ha permitido automatizar la creación de máquinas virtuales, y especialmente, poder compartir el código que lo hacía. Y finalmente Ansible, un gestor de aprovisionamiento

que nos posibilitaba gestionar la instalación y configuración de dependencias de software de nuestras aplicaciones.

El uso de estas herramientas nos ha permitido diseñar un flujo de trabajo totalmente distinto y mucho más eficiente al tradicional, dónde históricamente:

- Desarrolladores y administradores de sistemas trabajaban por separado.
- Cada desarrollador era responsable de su entorno, dando lugar a inconsistencias entre ellos y con los sistemas remotos (el famoso síndrome "en mi ordenador funciona, no sé por qué en producción no").
- El tiempo destinado a levantar el entorno de un proyecto se contaba por horas o días.
- El administrador de sistemas se convertía en un cuello de botella ya que era el único implicado en detectar y solucionar los problemas en la infraestructura.

Frente a este modo de trabajar, y apoyándonos en las herramientas que hemos ido introduciendo durante el desarrollo del proyecto, hemos propuesto la creación de uno nuevo dónde:

- El administrador de sistemas está integrado dentro del equipo de desarrollo.
- La infraestructura queda definida en código dentro del repositorio de código fuente.
- Cualquier cambio en esa infraestructura queda reflejado en el sistema de control de versiones a disposición de todos los miembros del equipo.
- Cualquier desarrollador es capaz de levantar un entorno de desarrollo en pocos minutos.
- El coste de adaptación de un nuevo desarrollador a un proyecto se reduce drásticamente.

Hemos podido observar la eficiencia del flujo propuesto al contemplar el ejemplo de un desarrollador con la tarea de efectuar una actualización de seguridad sobre un proyecto pasado. Poder comenzar a trabajar sobre el mismo con el método de desarrollo tradicional hubiera supuesto, de media, varias horas invertidas tan sólo levantando y configurando el entorno necesario para que funcionase, pudiendo incluso llegar a ser imposible reproducirlo exactamente si por ejemplo, corriese sobre un sistema operativo obsoleto que un desarrollador no pudiera emular correctamente en su propio ordenador.

Frente a estos problemas, nuestra propuesta ha permitido al mismo desarrollador crear un entorno con todas sus dependencias y aplicar la actualización de seguridad en menos de veinte minutos. Todo ello gracias a haber integrado al administrador de sistemas en el equipo y haber registrado la infraestructura junto al código fuente en nuestro repositorio.

Es una conclusión lógica, por tanto, pensar que las empresas de desarrollo tradicional de software deberían ir abandonando con el tiempo estos modelos tradicionales de separación de funciones, por métodos más cohesivos como el que proponemos, aumentando así la productividad y eficiencia del conjunto.

Ponemos por tanto a disposición de cualquier empresa este proyecto, el cual podrán seguir paso a paso para ir incorporando esta nueva forma de desarrollar a sus procesos. El proyecto se ha desarrollado en forma de paso a paso, por lo que será trivial para cualquier equipo ir adoptando las distintas herramientas.

Ponemos también a disposición el código fuente del mismo, el cual adjuntaremos a este proyecto y que también se puede encontrar en GitHub[lxxiv] libremente, con sus respectivas instrucciones para reproducir la instanciación del entorno y configuración de Drupal automáticamente desarrollado a lo largo del proyecto.

4.3 Comentario al papel de los administradores de sistemas en el futuro

Una derivada que se puede extraer del proyecto es saber qué papel jugarán los administradores de sistemas en el futuro. Y es que, si la administración de sistemas se puede llevar a código, y la infraestructura *cloud* hace innecesario el aprovisionamiento físico de las máquinas, podíamos tender a pensar que cada vez harán falta menos administradores de sistemas.

Nuestra opinión es que no estamos ante la extinción de una categoría profesional, sino más bien ante una evolución, que además, pensamos mejorará la relación con el resto de la empresa. Y es que el administrador de sistemas seguirá siendo clave para definir la infraestructura necesaria ya sea en un servidor físico o en la máquina virtual de los desarrolladores.

Ser capaz de asociar esa infraestructura junto al código en el que trabajan los desarrolladores, fomentará la comunicación entre unos y otros. Al mismo tiempo la posibilidad de levantar entornos completos con tan sólo un par de comandos le liberará de un precioso tiempo que podrá ser destinado a otras labores como son la monitorización de sistemas clave.

De hecho, pensamos que una de las evoluciones lógicas del trabajo del administrador de sistemas será la reducción del peso destinado a configurar los equipos, hacía un trabajo de monitorización más exhaustivo de las aplicaciones instaladas sobre sus sistemas.

De nuevo, aquí observamos las ventajas de introducirle junto a los programadores, y es que si por ejemplo fruto de esta tarea de revisión descubre que, desde el último *deploy* a producción, una llamada al servidor de base de datos está causando problemas, puede acudir a GitHub directamente, comprobar qué cambios se han enviado, e incluso identificar él mismo el origen del problema para o bien comunicárselo al equipo o bien cambiar algún parámetro lógico en la infraestructura.

4.4 Posibles ampliaciones del proyecto

El proyecto que hemos desarrollado deja lugar a varias ampliaciones, siendo en nuestra opinión la más interesante, ampliar el ámbito de uso de Ansible para que no sirva únicamente para configurar una máquina virtual local, sino que se extienda con el objetivo de manejar los distintos servidores remotos dónde los desarrollos se encuentren instalados.

Así por ejemplo, podríamos utilizar una misma configuración de Ansible con distintos parámetros, apoyándonos en el uso de variables, según quisiéramos configurar una máquina virtual, un servidor de pruebas, uno de calidad o incluso los de producción. Si consiguiéramos hacerlo, se podrían lanzar actualizaciones simultáneas sobre todos los servidores que soportasen una aplicación determinada con tan sólo una instrucción.

Otra ampliación interesante sería estudiar el concepto de Ansible Vault[lxxv], una característica de Ansible que nos permite encriptar datos en nuestro repositorio. Por ejemplo, puede ser útil si no queremos añadir información en claro sobre las contraseñas de nuestros servicios en el repositorio de GitHub. El problema es que *Vault* requeriría de una clave de encriptación a introducir cada vez que ejecutásemos una invocación de Ansible, lo que a día de hoy lo convierte en bastante incómodo.

Poder hacer *deploy* de código directamente con Ansible sería otra de las ampliaciones a considerar. Igual que podemos descargar e instalar paquetes, Ansible debería ser capaz de descargar el código a instalar, gestionar distintas versiones del mismo, e incluso permitirnos hacer *rollbacks* de forma rápida en servidores remotos.

Por último, un aspecto interesante a estudiar sería Docker[lxxvi], la última tendencia tecnológica en cuanto a gestión de paquetes y servicios. Docker se apoya en una característica muy interesante de Linux denominada contenedores LXC[lxxvii]. Los contenedores LXC nos permiten virtualizar partes del sistema operativo sin necesidad de crear una máquina completa.

Así pues, si creamos un contenedor LXC dentro de una instalación de Ubuntu, éste se crea instantáneamente, conteniendo una virtualización completa del host y aislada del mismo. Dentro del contenedor encontraremos lo que a todas luces parecerá una máquina virtual de Ubuntu pero sin la sobrecarga que estas acarrean.

Docker no es sino un interfaz surgido hace un par de años como proyecto *open source*, y que permite gestionar contenedores LXC fácilmente, de forma parecida a como Vagrant nos permite gestionar máquinas virtuales. La ventaja de Docker sobre las máquinas virtuales, es que los contenedores se crean muy fácil y rápidamente, lo que los convierte en realmente portables.

Anexo A

Ejecución del módulo setup de Ansible para ver la cantidad de parámetros que recupera internamente sobre el host en cada ejecución.

```
$ ansible -i hosts pfc-vm -u vagrant --private-key=~/.vagrant.d/insecure_private_key -m setup

192.168.33.30 | success >> {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "10.0.2.15",
            "192.168.33.30"
        ],
        "ansible_all_ipv6_addresses": [
            "fe80::a00:27ff:fe88:ca6"
        ],
        "ansible_architecture": "x86_64",
        "ansible_bios_date": "12/01/2006",
        "ansible_bios_version": "VirtualBox",
        "ansible_cmdline": {
            "BOOT_IMAGE": "/vmlinuz-3.2.0-23-generic",
            "quiet": true,
            "ro": true,
            "root": "/dev/mapper/precise64-root"
        },
        "ansible_date_time": {
            "date": "2014-11-09",
            "day": "09",
            "epoch": "1415535745",
            "hour": "12",
            "iso8601": "2014-11-09T12:22:25Z",
            "iso8601_micro": "2014-11-09T12:22:25.844522Z",
            "minute": "22",
            "month": "11",
            "second": "25",
            "time": "12:22:25",
            "tz": "UTC",
            "tz_offset": "+0000",
            "weekday": "Sunday",
            "year": "2014"
        },
        "ansible_default_ipv4": {
            "address": "10.0.2.15",
            "alias": "eth0",
            "gateway": "10.0.2.2",
            "interface": "eth0",
            "macaddress": "08:00:27:88:0c:a6",
            "mtu": 1500,
            "netmask": "255.255.255.0",
            "network": "10.0.2.0",
            "type": "ether"
```


```
        },
        "ansible_default_ipv6": {},
        "ansible_devices": {
            "sda": {
                "holders": [],
                "host":  "SATA  controller:  Intel  Corporation  82801HM/HEM
(ICH8M/ICH8M-E) SATA Controller [AHCI mode] (rev 02)",
                "model": "VBOX HARDDISK",
                "partitions": {
                    "sda1": {
                        "sectors": "497664",
                        "sectorsize": 512,
                        "size": "243.00 MB",
                        "start": "2048"
                    },
                    "sda2": {
                        "sectors": "2",
                        "sectorsize": 512,
                        "size": "1.00 KB",
                        "start": "501758"
                    },
                    "sda5": {
                        "sectors": "167268352",
                        "sectorsize": 512,
                        "size": "79.76 GB",
                        "start": "501760"
                    }
                },
                "removable": "0",
                "rotational": "1",
                "scheduler_mode": "cfq",
                "sectors": "167772160",
                "sectorsize": "512",
                "size": "80.00 GB",
                "support_discard": "0",
                "vendor": "ATA"
            },
            "sr0": {
                "holders": [],
                "host": "IDE interface: Intel Corporation 82371AB/EB/MB PIIX4
IDE (rev 01)",
                "model": "CD-ROM",
                "partitions": {},
                "removable": "1",
                "rotational": "1",
                "scheduler_mode": "cfq",
                "sectors": "2097151",
                "sectorsize": "512",
                "size": "1024.00 MB",
                "support_discard": "0",
                "vendor": "VBOX"
            },
            "sr1": {
                "holders": [],
                "host": "IDE interface: Intel Corporation 82371AB/EB/MB PIIX4
IDE (rev 01)",
```

```
                "model": "CD-ROM",
                "partitions": {},
                "removable": "1",
                "rotational": "1",
                "scheduler_mode": "cfq",
                "sectors": "2097151",
                "sectorsize": "512",
                "size": "1024.00 MB",
                "support_discard": "0",
                "vendor": "VBOX"
            }
        },
        "ansible_distribution": "Ubuntu",
        "ansible_distribution_major_version": "12",
        "ansible_distribution_release": "precise",
        "ansible_distribution_version": "12.04",
        "ansible_domain": "",
        "ansible_env": {
            "HOME": "/home/vagrant",
            "LANG": "en_US.UTF-8",
            "LC_ALL": "en_US",
            "LC_CTYPE": "en_US.UTF-8",
            "LOGNAME": "vagrant",
            "MAIL": "/var/mail/vagrant",
            "PATH":
"/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games",
            "PWD": "/home/vagrant",
            "SHELL": "/bin/bash",
            "SHLVL": "1",
            "SSH_AUTH_SOCK": "/tmp/ssh-SmDqEU9482/agent.9482",
            "SSH_CLIENT": "192.168.33.1 62432 22",
            "SSH_CONNECTION": "192.168.33.1 62432 192.168.33.30 22",
            "SSH_TTY": "/dev/pts/0",
            "TERM": "xterm-256color",
            "USER": "vagrant",
            "_": "/bin/sh"
        },
        "ansible_eth0": {
            "active": true,
            "device": "eth0",
            "ipv4": {
                "address": "10.0.2.15",
                "netmask": "255.255.255.0",
                "network": "10.0.2.0"
            },
            "ipv6": [
                {
                    "address": "fe80::a00:27ff:fe88:ca6",
                    "prefix": "64",
                    "scope": "link"
                }
            ],
            "macaddress": "08:00:27:88:0c:a6",
            "module": "e1000",
            "mtu": 1500,
            "promisc": false,
```


```
        "type": "ether"
    },
    "ansible_eth1": {
        "active": true,
        "device": "eth1",
        "ipv4": {
            "address": "192.168.33.30",
            "netmask": "255.255.255.0",
            "network": "192.168.33.0"
        },
        "macaddress": "08:00:27:86:32:db",
        "module": "e1000",
        "mtu": 1500,
        "promisc": false,
        "type": "ether"
    },
    "ansible_form_factor": "Other",
    "ansible_fqdn": "precise64",
    "ansible_hostname": "precise64",
    "ansible_interfaces": [
        "lo",
        "eth1",
        "eth0"
    ],
    "ansible_kernel": "3.2.0-23-generic",
    "ansible_lo": {
        "active": true,
        "device": "lo",
        "ipv4": {
            "address": "127.0.0.1",
            "netmask": "255.0.0.0",
            "network": "127.0.0.0"
        },
        "ipv6": [
            {
                "address": "::1",
                "prefix": "128",
                "scope": "host"
            }
        ],
        "mtu": 16436,
        "promisc": false,
        "type": "loopback"
    },
    "ansible_lsb": {
        "codename": "precise",
        "description": "Ubuntu 12.04 LTS",
        "id": "Ubuntu",
        "major_release": "12",
        "release": "12.04"
    },
    "ansible_machine": "x86_64",
    "ansible_memfree_mb": 67,
    "ansible_memtotal_mb": 365,
    "ansible_mounts": [
        {
```

```
                "device": "/dev/mapper/precise64-root",
                "fstype": "ext4",
                "mount": "/",
                "options": "rw,errors=remount-ro",
                "size_available": 77831938048,
                "size_total": 84696281088
            },
            {
                "device": "/dev/sda1",
                "fstype": "ext2",
                "mount": "/boot",
                "options": "rw",
                "size_available": 200482816,
                "size_total": 238787584
            },
            {
                "device": "/vagrant",
                "fstype": "vboxsf",
                "mount": "/vagrant",
                "options": "uid=1000,gid=1000,rw",
                "size_available": 14869639168,
                "size_total": 120137519104
            }
        ],
        "ansible_nodename": "precise64",
        "ansible_os_family": "Debian",
        "ansible_pkg_mgr": "apt",
        "ansible_processor": [
            "Intel(R) Core(TM) i5-3210M CPU @ 2.50GHz",
            "Intel(R) Core(TM) i5-3210M CPU @ 2.50GHz"
        ],
        "ansible_processor_cores": 2,
        "ansible_processor_count": 1,
        "ansible_processor_threads_per_core": 1,
        "ansible_processor_vcpus": 2,
        "ansible_product_name": "VirtualBox",
        "ansible_product_serial": "NA",
        "ansible_product_uuid": "NA",
        "ansible_product_version": "1.2",
        "ansible_python_version": "2.7.3",
        "ansible_selinux": false,
        "ansible_ssh_host_key_dsa_public":
"AAAAB3NzaC1kc3MAAACBAJwR6q4VerUDe7bLXRL6ZPTXj5FY66he+WWlRSoQppwDLqrTG73Pa9qU
HMDFb1LXN1qgg0p0lyfqvm8ZeN+98rbT0JW6+Wqa7v0K+N82xf87fVkJcXAuU/A8OGR9eVMZmWsIO
pabZexd5CHYgLO3k4YpPSdxc6S4zJcOGwXVnmGHAAAAFQDHjsPg0rmkbquTJRdlEZBVJe9+3QAAAI
BjYIAiGvKhmJfzDjVfzlxRD1ET7ZhSoMDxU0KadwXQP1uBdlYVEteJQpUTEsA+7kFH7xhtZ/zbK2a
fEFHriAphTJmz8GqkIR5CJXh3dZspdk2MHCgxkXl5G/iVPLR9UShN+nsAVxfm0gffCqbqZu3Ridt3
JwTXQbiDfXO/a6T/eQAAAIEAlsW/i/dUuFbRVO2zaAKwL/CFWT19Al7+njszC5FCJ2deggmF/NIKJ
UbJwkRZkwL4PY1HYj2xqn7ImhPSyvdCd+IFdw73Pndnjv0luDc8i/a4JUEfna4rzXt1Y5c24J1pEo
KA05VicyCBD2z6TodRJEVEFSsa1s8s2p9x6LxwsDw=",
        "ansible_ssh_host_key_ecdsa_public":
"AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBFxsiWE3WImfJcjiWS5asOVo
Msn+0gFLU5AgPNs2ATokB7kw00IsB0YGrqClwYNauRRddkYMsi0icJSR60mYNSo=",
        "ansible_ssh_host_key_rsa_public":
"AAAAB3NzaC1yc2EAAAADAQABAAABAQDZt46W9slSN3Y6D2f931rijUPCEewhQWmBfGhybuF4qLft
fJMuyFcREZkG6UretVI8ZnQn/OMDgbf2DYMzKsRLnz7W5cGy1Mt1pWoG0iCgi2xHzLqOqPYo4mP9/
```


```
    hdZT6pANXapETT55yx8sHAYLAa9NK5Dtyv+QNQ2dUUb1wUTCqgYffLVDgoHvNNDwCwB6biJf6uopq
    fg2KXvAzcqSa6oaRChJOXjFlM08HebMwkMSzrOXjWbXhFsONy5JuDf3WztCtLMsFrVRHTdDwTh7uL
    2UQ8Qcky+kP6Wd7G8NlW5RxubYIFpAM0u2SsQIjYOxz+eOfQ8GE3WjvaIBqX05gat",
            "ansible_swapfree_mb": 766,
            "ansible_swaptotal_mb": 767,
            "ansible_system": "Linux",
            "ansible_system_vendor": "innotek GmbH",
            "ansible_user_id": "vagrant",
            "ansible_userspace_architecture": "x86_64",
            "ansible_userspace_bits": "64",
            "ansible_virtualization_role": "guest",
            "ansible_virtualization_type": "virtualbox",
            "module_setup": true
        },
        "changed": false
    }
```

Anexo B

```
# simon at MacBook-Pro-de-Simon-2.local in ~/dev/pfc on git:master o
[14:13:49]
$ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Importing base box 'hashicorp/precise64'...
==> default: Matching MAC address for NAT networking...
==> default: Checking if box 'hashicorp/precise64' is up to date...
==> default: Setting the name of the VM: pfc_default_1417266848141_77259
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
    default: Adapter 2: hostonly
==> default: Forwarding ports...
    default: 22 => 2222 (adapter 1)
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
    default: SSH address: 127.0.0.1:2222
    default: SSH username: vagrant
    default: SSH auth method: private key
    default: Warning: Connection timeout. Retrying...
==> default: Machine booted and ready!
==> default: Checking for guest additions in VM...
    default: The guest additions on this VM do not match the installed
version of
    default: VirtualBox! In most cases this is fine, but in rare cases it can
    default: prevent things such as shared folders from working properly. If
you see
    default: shared folder errors, please make sure the guest additions
within the
    default: virtual machine match the version of VirtualBox you have
installed on
    default: your host and reload your VM.
    default:
    default: Guest Additions Version: 4.2.0
    default: VirtualBox Version: 4.3
==> default: Configuring and enabling network interfaces...
==> default: Mounting shared folders...
    default: /vagrant => /Users/simon/dev/pfc
==> default: Running provisioner: ansible...

PLAY                                                              [pfc-vm]
*****************************************************************

GATHERING                                                          FACTS
*****************************************************************
ok: [192.168.33.30]

TASK:        [pfc-nginx        |        Adding        nginx-full        ppa]
***********************************
changed: [192.168.33.30]
```

```
TASK: [pfc-nginx | Installing nginx-full]
***********************************
changed: [192.168.33.30]

TASK: [pfc-nginx | Adding custom nginx.conf template]
*************************
changed: [192.168.33.30]

TASK: [pfc-nginx | Ensure nginx is started at boot]
**************************
ok: [192.168.33.30]

TASK: [pfc-mysql | Installing mysql-server and dependencies]
******************
changed: [192.168.33.30] => (item=mysql-server,python-mysqldb)

TASK: [pfc-mysql | Ensuring mysql is started at boot]
************************
ok: [192.168.33.30]

TASK: [pfc-mysql | Setting the mysql root password]
**************************
changed: [192.168.33.30] => (item=precise64)
changed: [192.168.33.30] => (item=127.0.0.1)
changed: [192.168.33.30] => (item=::1)
changed: [192.168.33.30] => (item=localhost)

TASK: [pfc-mysql | Setting mysql passwordless root access]
********************
changed: [192.168.33.30]

TASK: [pfc-php-fpm | Installing php5-fpm]
***********************************
changed: [192.168.33.30] => (item=php5-fpm,php5-cli,php5-mysql,php5-gd,php5-
curl,php5-mcrypt)

TASK: [pfc-php-fpm | Ensuring php-fpm is started at boot]
*********************
ok: [192.168.33.30]

TASK: [pfc-php-fpm | Configuring php.ini]
************************************
changed: [192.168.33.30] => (item={'option': 'memory_limit', 'value':
'512M'})
changed: [192.168.33.30] => (item={'option': 'post_max_size', 'value':
'128M'})
changed: [192.168.33.30] => (item={'option': 'upload_max_filesize', 'value':
'128M'})
changed: [192.168.33.30] => (item={'option': 'display_errors', 'value':
'On'})
changed: [192.168.33.30] => (item={'option': 'error_reporting', 'value':
'E_ALL | E_STRICT'})
changed: [192.168.33.30] => (item={'option': 'display_startup_errors',
'value': 'On'})
changed: [192.168.33.30] => (item={'option': 'html_errors', 'value': 'On'})
```

```
changed: [192.168.33.30] => (item={'option': 'track_errors', 'value': 'On'})

TASK: [pfc-drupal-conf | Creating a MySQL user for drupal]
********************
changed: [192.168.33.30] => (item=precise64)
changed: [192.168.33.30] => (item=127.0.0.1)
changed: [192.168.33.30] => (item=::1)
changed: [192.168.33.30] => (item=localhost)

TASK: [pfc-drupal-conf | Installing drush]
***********************************
changed: [192.168.33.30] => (item=drush,sendmail)

TASK: [pfc-drupal-conf | Configuring drupal settings.php with drush]
**********
changed: [192.168.33.30]

TASK: [pfc-drupal-conf | Adding nginx site conf file]
*************************
changed: [192.168.33.30]

TASK: [pfc-drupal-conf | Enabling nginx site and reload the webserver]
********
changed: [192.168.33.30]

TASK: [pfc-drupal-conf | Disabling nginx default site]
************************
changed: [192.168.33.30]

NOTIFIED: [pfc-nginx | reload nginx]
*****************************************
changed: [192.168.33.30]

NOTIFIED: [pfc-php-fpm | restart php-fpm]
***********************************
changed: [192.168.33.30]

PLAY RECAP
********************************************************************

192.168.33.30 : ok=20 changed=16 unreachable=0 failed=0
```


Referencias

[i] "Vagrant." 2013. 12 Oct. 2014 <https://www.vagrantup.com/>
[ii] "DevOps - Wikipedia, the free encyclopedia." 2010. 12 Oct. 2014 <http://en.wikipedia.org/wiki/DevOps>
[iii] "Vagrant." 2013. 12 Oct. 2014 <https://www.vagrantup.com/>
[iv] "Vagrant." 2013. 12 Oct. 2014 <https://www.vagrantup.com/>
[v] "Precise Pangolin release notes - Ubuntu Wiki." 2011. 12 Oct. 2014 <https://wiki.ubuntu.com/PrecisePangolin/ReleaseNotes>
[vi] "Canonical | The company behind Ubuntu." 26 Oct. 2014 <http://www.canonical.com/>
[vii] "mod_geoIP - Maxmind Developer Site." 2013. 29 Nov. 2014 <http://dev.maxmind.com/geoip/legacy/mod_geoip2/>
[viii] "Nginx." 2009. 12 Oct. 2014 <http://nginx.org/en/>
[ix] "nginx." 2009. 29 Nov. 2014 <http://nginx.org/en/>
[x] "PHP: Manejador de Procesos FastCGI (FPM) - Manual." 2010. 29 Nov. 2014 <http://php.net/manual/es/install.fpm.php>
[xi] "MySQL :: The world's most popular open source database." 29 Nov. 2014 <http://www.mysql.com/>
[xii] "Oracle VM VirtualBox." 2011. 29 Nov. 2014 <https://www.virtualbox.org/>
[xiii] "Vagrant." 2013. 29 Nov. 2014 <https://www.vagrantup.com/>
[xiv] "Ansible is Simple IT Automation." 29 Nov. 2014 <http://www.ansible.com/>
[xv] "Amazon Web Services (AWS) - Cloud Computing Services." 2005. 19 Oct. 2014 <http://aws.amazon.com/>
[xvi] "Compute Engine - Google Cloud Platform." 2012. 19 Oct. 2014 <https://cloud.google.com/compute/>
[xvii] "Heroku | Cloud Application Platform." 2010. 19 Oct. 2014 <https://www.heroku.com/>
[xviii] "VMware Virtualization for Desktop & Server, Application ..." 19 Oct. 2014 <http://www.vmware.com/>
[xix] "Parallels." 2002. 19 Oct. 2014 <http://www.parallels.com/>
[xx] "Oracle VM VirtualBox." 2011. 19 Oct. 2014 <https://www.virtualbox.org/>
[xxi] "VirtualBox - Wikipedia, the free encyclopedia." 2007. 11 Oct. 2014 <http://en.wikipedia.org/wiki/VirtualBox>
[xxii] "Ubuntu 12.04.5 LTS (Precise Pangolin) - Ubuntu Releases." 2012. 26 Oct. 2014 <http://releases.ubuntu.com/12.04/>
[xxiii] "Vagrant (software) - Wikipedia, the free encyclopedia." 2012. 11 Oct. 2014 <http://en.wikipedia.org/wiki/Vagrant_(software)>
[xxiv] "Vagrant." 2013. 26 Oct. 2014 <https://www.vagrantup.com/>
[xxv] "Download Vagrant - Vagrant." 2013. 12 Oct. 2014 <https://www.vagrantup.com/downloads.html>
[xxvi] "hashicorp - Vagrant Cloud." 2014. 12 Oct. 2014 <https://vagrantcloud.com/hashicorp>
[xxvii] "Lenguaje de Programación Ruby." 2013. 12 Oct. 2014 <https://www.ruby-lang.org/es/>
[xxviii] "Lenguaje de Programación Ruby." 2013. 12 Oct. 2014 <https://www.ruby-lang.org/es/>
[xxix] "Puppet Labs: IT Automation Software for System ..." 2010. 19 Oct. 2014 <http://puppetlabs.com/>
[xxx] "Chef | IT automation for speed and awesomeness | Chef." 2013. 19 Oct. 2014 <https://www.getchef.com/chef/>
[xxxi] "Idempotencia - Wikipedia, la enciclopedia libre." 2006. 26 Oct. 2014 <http://es.wikipedia.org/wiki/Idempotencia>
[xxxii] "Ansible (software) - Wikipedia, the free encyclopedia." 2012. 19 Oct. 2014 <http://en.wikipedia.org/wiki/Ansible_(software)>
[xxxiii] "Windows Support — Ansible Documentation." 2014. 5 Ene. 2015 < http://docs.ansible.com/intro_windows.html>
[xxxiv] "Installation — Ansible Documentation." 2014. 19 Oct. 2014 <http://docs.ansible.com/intro_installation.html>

[xxxv] "Homebrew — The missing package manager for OS X." 2013. 19 Oct. 2014 <http://brew.sh/>
[xxxvi] "ping - Try to connect to host and return pong on success ..." 2014. 26 Oct. 2014 <http://docs.ansible.com/ping_module.html>
[xxxvii] "Introduction To Ad-Hoc Commands — Ansible Documentation." 2014. 26 Oct. 2014 <http://docs.ansible.com/intro_adhoc.html>
[xxxviii] "Playbooks — Ansible Documentation." 2014. 26 Oct. 2014 <http://docs.ansible.com/playbooks.html>
[xxxix] "Module Index — Ansible Documentation." 2014. 26 Oct. 2014 <http://docs.ansible.com/modules_by_category.html>
[xl] "copy - Ansible Documentation." 2014. 26 Oct. 2014 <http://docs.ansible.com/copy_module.html>
[xli] "apt - Manages apt-packages — Ansible Documentation." 2014. 26 Oct. 2014 <http://docs.ansible.com/apt_module.html>
[xlii] "service - Manage services. — Ansible Documentation." 2014. 26 Oct. 2014 <http://docs.ansible.com/service_module.html>
[xliii] "The Official YAML Web Site." 2002. 26 Oct. 2014 <http://www.yaml.org/>
[xliv] "Server provisioning - Wikipedia, the free encyclopedia." 2006. 5 Ene. 2014 <http://en.wikipedia.org/wiki/Provisioning - Server_provisioning>
[xlv] "Drupal - Open Source CMS | Drupal.org." 2004. 1 Nov. 2014 <https://www.drupal.org/>
[xlvi] "Content management system - Wikipedia, the free ..." 2003. 1 Nov. 2014 <http://en.wikipedia.org/wiki/Content_management_system>
[xlvii] "GitHub · Build software better, together." 2008. 2 Nov. 2014 <https://github.com/>
[xlviii] "Git (software) - Wikipedia, the free encyclopedia." 2005. 1 Nov. 2014 <http://en.wikipedia.org/wiki/Git_(software)>
[xlix] "GitHub · Using pull requests." 2014. 5 Ene. 2014 <https://help.github.com/articles/using-pull-requests/>
[l] "mitchellh/vagrant - GitHub." 2014. 1 Nov. 2014 <https://github.com/mitchellh/vagrant/issues/4362>
[li] "ansible/ansible · GitHub." 2012. 1 Nov. 2014 <https://github.com/ansible/ansible>
[lii] "bitcoin/bitcoin · GitHub." 2010. 1 Nov. 2014 <https://github.com/bitcoin/bitcoin>
[liii] "Generating SSH keys - GitHub Help." 2012. 1 Nov. 2014 <https://help.github.com/articles/generating-ssh-keys/>
[liv] "Playbook Roles and Include Statements — Ansible ..." 2014. 1 Nov. 2014 <http://docs.ansible.com/playbooks_roles.html>
[lv] "apt_repository - Ansible Documentation." 2014. 2 Nov. 2014 <http://docs.ansible.com/apt_repository_module.html>
[lvi] "NGINX Stable : “Nginx” team - Launchpad." 2014. 2 Nov. 2014 <https://launchpad.net/~nginx/+archive/ubuntu/stable>
[lvii] "apt - Manages apt-packages — Ansible Documentation." 2014. 2 Nov. 2014 <http://docs.ansible.com/apt_module.html>
[lviii] "template - Ansible Documentation." 2014. 2 Nov. 2014 <http://docs.ansible.com/template_module.html>
[lix] "Welcome to Jinja2 — Jinja2 Documentation (2.8-dev)." 2014. 2 Nov. 2014 <http://jinja.pocoo.org/docs/dev/>
[lx] "service - Manage services. — Ansible Documentation." 2014. 9 Nov. 2014 <http://docs.ansible.com/service_module.html>
[lxi] "mysql_user - Ansible Documentation." 2014. 9 Nov. 2014 <http://docs.ansible.com/mysql_user_module.html>
[lxii] "Loops — Ansible Documentation." 2014. 9 Nov. 2014 <http://docs.ansible.com/playbooks_loops.html>
[lxiii] "setup - Gathers facts about remote hosts — Ansible ..." 2014. 9 Nov. 2014 <http://docs.ansible.com/setup_module.html>
[lxiv] "PHP: FastCGI Process Manager (FPM) - Manual." 2010. 15 Nov. 2014 <http://php.net/manual/en/install.fpm.php>
[lxv] "apt - Manages apt-packages — Ansible Documentation." 2014. 15 Nov. 2014 <http://docs.ansible.com/apt_module.html>

[lxvi] "php.ini." 2014. 15 Nov. 2014 <https://raw.githubusercontent.com/php/php-src/master/php.ini-production>
[lxvii] "ini_file - Tweak settings in INI files — Ansible Documentation." 2014. 15 Nov. 2014 <http://docs.ansible.com/ini_file_module.html>
[lxviii] "YAML Syntax — Ansible Documentation." 2014. 15 Nov. 2014 <http://docs.ansible.com/YAMLSyntax.html>
[lxix] "Download & Extend | Drupal.org." 2014. 16 Nov. 2014 <https://www.drupal.org/download>
[lxx] "drush-ops/drush · GitHub." 2012. 29 Nov. 2014 <https://github.com/drush-ops/drush>
[lxxi] "Synced Folders - Vagrant Documentation." 2013. 16 Nov. 2014 <https://docs.vagrantup.com/v2/synced-folders/>
[lxxii] "PHPFcgiExample - Nginx Community." 2009. 17 Nov. 2014 <http://wiki.nginx.org/PHPFcgiExample>
[lxxiii] "Drupal Core - Moderately Critical - Multiple Vulnerabilities ..." 2014. 29 Nov. 2014 <https://www.drupal.org/SA-CORE-2014-006>
[lxxiv] PFC 2014 / PFC. <https://github.com/pfc2014/pfc>
[lxxv] "Vault — Ansible Documentation." 2015. 2 Ene. 2015 <http://docs.ansible.com/playbooks_vault.html>
[lxxvi] "Docker" 2015. 2 Ene 2015 <https://www.docker.com>
[lxxvii] "LXC Containers" 2015. 2 Ene 2015 <http://en.wikipedia.org/wiki/LXC>