**GESTIONE DI GRUPPI DI MACCHINE VIRTUALI**

*In questo capitolo viene illustrato l’utilizzo di Vagrant nell’ambito del lavoro di tesi. Saranno spiegati anche Chef e Puppet, linguaggi di provisioning utilizzati per configurare agevolmente ed efficacemente delle Vagrant Box.*

***INTRODUZIONE***

Nell’ambito del lavoro di tesi, un obiettivo da raggiungere era quello di riuscire a simulare, in un ambiente virtualizzato, le tecnologie affrontate nel corso di Architetture Software. Attualmente vengono messi in commercio personal computer con un quantitativo di Ram sempre maggiore. Il mercato attuale, ad un costo non troppo elevato, offre computer dai 4GB agli 8GB di RAM. Oltre che per eseguire le moderne applicazioni, un tale quantitativo di RAM rende l’utilizzo di Virtual Machine su personal computer molto più facile e fattibile. Questo ha aperto alcune opzioni interessanti nel ciclo di sviluppo di qualsiasi applicazione o servizio in quanto ora è possibile avere una macchina di test/sviluppo su ogni personal computer. Un amministratore di sistema vorrebbe che queste macchine virtuali siano il più simile possibile all’ambiente di produzione e **Vagrant** rappresenta un buon modo per gestire e distribuire macchine virtuali agli sviluppatori.

**COS’E’ VAGRANT**

Vagrant è un software libero ed open-source per la creazione e la configurazione di ambienti di sviluppo virtuali, fornisce un ambiente di lavoro facile da configurare, riproducibile, portatile costruito con tecnologie standard nel settore e controllato da un unico flusso di lavoro coerente per aiutare a massimizzare la produttività e la flessibilità di un singolo sviluppatore o di un team di programmatori.

Per ottemperare ai suoi scopi, “*Vagrant sta sulle spalle di giganti*”. Le macchine sono fornite tramite VirtualBox, VMware, AWS, o un qualsiasi altro provider. Strumenti standard di provisioning e configurazione come script in shell, Chef o Puppet, possono essere utilizzati per installare e configurare il software sulla macchina automaticamente.

In sintesi Vagrant è quindi un sistema per riprodurre e rendere portabili degli ambienti di sviluppo. Viene scaricata una macchina virtuale e, grazie al software di virtualizzazione (per esempio Virtual Box) si ha un server di testing privato e configurato secondo le specifiche, magari uguale a quello di produzione. La cosa formidabile è che grazie a **Vagrant** questo ambiente può essere riprodotto facilmente sulle macchine del team di sviluppo.

**PERCHE’ VAGRANT**

Utilizzare Vagrant permette di fornire a tutti gli sviluppatori del team una macchina virtuale con la configurazione sempre allineata, in quanto creata a partire da file che sono sotto controllo di versione assieme al progetto in fase di sviluppo. Gli stessi file possono essere utilizzati anche per il server di produzione, grantendo quindi un perfetto allineamento tra i due ambienti.

Vagrant comporta ripercussioni positive per lo sviluppatore, designer e per l’ingegnere di sistema.

Lo **sviluppatore** non si dovrà più preoccupare di errori del software derivanti da differenti ambienti di esecuzione. Vagrant infatti permette di isolare le dipendenze dalla loro configurazione in un unico ambiente coerente senza porre limitazioni e vincoli agli strumenti normalmente usati da un programmatore. Una volta che la configurazione dell’ambiente in cui si intende rilasciare l’applicazione è stata definita in un file, chiamato Vagrantfile, e completata è sufficiente avviare Vagrant per avere una macchina virtuale che ha tutti gli strumenti settati ed installati pronti per essere utilizzati. E’ possibile condividere il Vagrantfile tra gli sviluppatori dello stesso team per assicurare che ognuno lavori sulla stessa tipologia di macchina virtuale. Ogni membro del team può continuare ad usare il suo sistema operativo e strumenti di produzione preferiti creando comunque del codice che funzioni per la macchina di produzione che dovrà ospitare l’applicazione finale.

Vagrant, ad un **ingegnere di sistema**, garantisce un ambiente coerente sul quale rilasciare applicazioni o script per la gestione dell’infrastruttura. E’ possibile configurare una macchina virtuale usando shell script, oppure mediante Chef cookbooks o moduli Puppet sfruttando la virtualizzazione offerta da strumenti altamente performanti quali VirtualBox o VMware.

Vagrant comporta benefici anche per i **designer**. Tale strumento imposterà automaticamente tutte le risorse necessarie per avviare una webapp ed il designer si potrà occupare solo del design. Una volta che uno sviluppatore ha configurato Vagrant, non è più necessario preoccuparsi di come mandare in esecuzione una applicazione. Sarà Vagrant stesso che, una volta vviato, provvederà a mandare in esecuzione l’applicazione. Questo permette ad un designer di modificare la parte grafica di un sistema senza dover continuamente interagire con gli sviluppatori. Il designer può quindi modificare la parte di codice di sua competenza e vederne direttamente gli effetti con una interazione minima, se non nulla, con gli altri component del team di sviluppo.

**Requisiti - Hypervisor**

Per iniziare ad usare Vagrant bisogna, per prima cosa, installare un hypervisor. Quelli supportati sono molti ma il predefinito, ed è quello che è stato usato nella tesi, è **VirtualBox**. Successivamente è necessario installare anche **Vagrant**. A questo punto è tutto praticamente pronto per avvivare una prima macchina virtuale.

**Concetti base**

**Vagrant Box**

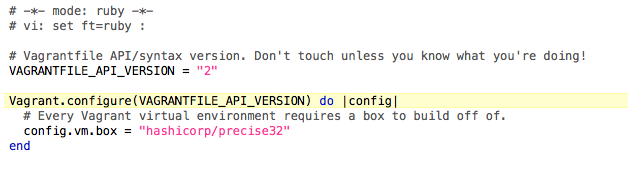
Il primo concetto fondamentale da capire per quanto riguarda **Vagrant** è quello di Box. Una **Box** è un vero e proprio contenitore che contiene il sistema operativo nonché i software di base ed ogni loro configurazione. Una box è l’immagine di base utilizzata per creare un ambiente virtuale con Vagrant. E’ pensata per essere un file portabile che può essere utilizzato da altri, su qualsiasi piattaforma su cui Vagrant possa funzionare per far partire un ambiente virtuale. Esistono già diverse box con delle configurazioni di base e sono rese disponibili dai server di Vagant. Per aggiungere a Vagrant una specifica box il comando è il seguente.

$ vagrant box add hashicorp/precise32

In questo modo dal Vagrant Cloud verrà scaricata una box chiamata “hashicorp/precise32”. Tale box è quella che, prevalentemente, è stata usata nella tesi ed è una semplice macchina virtuale a 32 bit con Ubuntu 12.04 con il supporto a Chef e Puppet pre-installato. I box forniscono solo l’immagine di base per Vagrant. Nel momento in cui si esegue **Vagrant up**, comando per avviare la creazione ed il boot di una macchinavirtuale, il box viene copiato in modo che possa essere poi modificato per quella specifica macchina virtuale. Pertanto, è possibile rimuovere o aggiornare il box dopo che una macchina virtuale è stata creata.

**Vagrantfile**

Il concetto più importante di vagrant è il **Vagrantfile** perché permette di descrivere il tipo di macchina richiesta per il progetto nonché come configurare ed effettuare il provision di tale macchina. Questo file è nominato in questo modo perché Vagrantfile è proprio il nome che il file contenente una configurazione Vagrant deve avere. Tale file è scritto mediante Ruby ma non è necessario conoscere in modo approfondito tale linguaggio di programmazione per configurare una o più macchine virtuali. Di seguito un esempio minimale di configurazione mediante Vagrantfile.



Escludendo le righe che iniziano con #, rappresentanti dei commenti, il frammento di codice precedente permette di configurare una macchina Ubuntu con supporto per Chef e Puppet. Il “2” nella prima riga effettiva di codice specifica la versione dell’oggetto config mediante il quale è possibile dettare le regole di configurazione della macchina all’interno della sezione do – end.

Config.vm.box specifica quale box deve essere avviata.

A questo punto il comando da lanciare da riga di comando per avviare tale macchina virtuale è **vagrant up**. Dopo aver lanciato tale comando Vagrant provvederà a fare il boot della macchina con le caratteristiche specificate. Terminato il boot, per essere operativi all’interno della macchina è sufficiente effettuare un **vagrant ssh**.

**Cartelle sincronizzate**

Questo concetto è molto utile quando si lavora con le box. Le cartelle sincronizzate consentono a Vagrant di sincronizzare una cartella sul computer host sulla macchina guest, permettendo di continuare a lavorare sul file del progetto sul computer host, ma utilizzano le risorse nella macchina guest per compilare o eseguire il progetto. Per impostazione predefinita, Vagrant condividerà la directory del progetto (la directory con il Vagrantfile) nella directory /vagrant.

config.vm.synced\_folder "src/", "/srv/website"

Il comando appena riportato permette di sincronizzare una cartella. Il primo parametro è il path di una cartella sulla macchina host. Il secondo parametro è un path assoluto ed indica in qualche cartella sulla macchina virtuale deve essere sincronizzata la cartella della macchina host. Se tale cartella non esiste nella macchina virtuale verrà creata.

Le cartelle sincronizzate, ovviamente, sono state molto utilizzate nell’ambito del lavoro di tesi. Un programma, distribuito sotto forma di runnable jar file, può essere collocato nella cartella della macchina host che è sincronizzata, come da configurazione, con una directory della macchina guest. Operando in questo modo i file saranno immediatamente disponibili anche nella macchina virtuale pronti per essere usati.

**Provisioning**

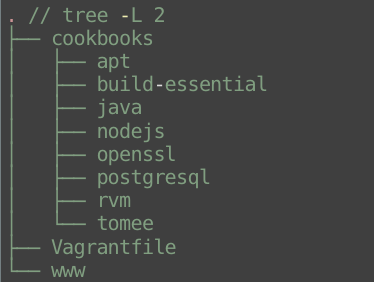
L’avvio di una macchina virtuale vuota non è molto utile, per questo motivo Vagrant supporta il provisioning delle macchine virtuali attraverso l’uso di provisioners. Vagrant infatti non è solo un sistema per il rapido avvio delle macchine virtuali ma, come già detto, permette di controllarne le impostazioni nel tempo insieme al software installato. Pertanto è possibile gestire la configurazione automatica attraverso una struttura di provisioning abbastanza flessibile che supporta normali shell script (Bash), Puppet, Chef e Ansible. Al termine delle operazioni di configurazione, indipendentemente dallo strumento osato, il risultato sarà quello di una macchina virtuale con tutto lo stack software che è di interesse avere in quello specifico ambiente. Nel lavoro di testi è stato usato con grande frequenza Chef ma talvolta è stato fatto ricorso a Puppet.

**Chef**

Come riporta la documentazione ufficiale [Chef di Opscode](http://www.opscode.com/chef/) è un sistema che permette di trasformare un server, sia esso fisico o virtuale, in codice. Chef permette di automatizzare il modo di creare, gestire e rilasciare l’infrastruttura. L’infrastruttura stessa diventa pertanto testabile, ripetibile e persino soggetta a controllo di versione proprio come accade con il codice di programmazione.

Chef si basa su definizioni riutilizzabili, dette recipes (ricette), per automatizzare le attività di configurazione dell’infrastruttura. Esempi di recipes sono le istruzioni per configurare un server web, database o un meccanismo di load balancing. Insieme le ricette descrivono in cosa consiste l’infrastruttura e come ogni parte della stessa deve essere rilasciata, configurata e gestita. Le ricette a loro volta sono costituite da risorse (resources). Ogni risorsa descrive un frammento dell’infrastruttura come un file, un template o un package che deve essere installato.

Spesso Chef viene utilizzato in modalità **client server**, dove un client (chiamato nodo) accede ad un repository (server) per recuperare le "ricette" da eseguire sulla macchina. Nel nostro caso useremo **Chef-Solo** che ci esonera dal dover avere un repository remoto, ma di contro dobbiamo avere tutte le "ricette" in una cartella. La cartella che deve contenere tali ricette è chiamata **cookbooks**. Per usare Chef è quindi necessario dare la seguente struttura alla cartella.



La cartella di un progetto Vagrant avrà quindi, oltre al Vagrantfile, una directory chiamata cookbooks. All’interno di cookbooks avremo i vari cookbook come ad esempio quello di java oppure di glassfish. Ogni cookbook può essere composto da diverse recipes. In glassfish ad esempio esistono 4 tipi di recipes: **default** (effettua il download e l’estrazione del file binario di glassfish e crea un utente ed un gruppo), **attribute\_driven\_domain** (configura 0 o più domini di glassfish usando l’attributo glassfish/domains), **attribute\_driven\_mq** (configura 0 o più broker GlassFish OpenMQ usando l’attributo openmq/instances) e **search\_driven\_domain** (configura 0 o più domini di GlassFish usando search per generare la configurazione). Ogni recipe è personalizzabile mediante le **resources**.

Vista la struttura di una configurazione Chef è necessario capire come possa essere usata in coabitazione con Vagrant.



Per prima cosa è necessario indicare che come strumento di provisioning si usa Chef e successivamente indicare le recipes che si intende adottare per la configurazione.

Il comando **client.vm.provision :chef\_solo** sta ad indicare che si usa Chef-Solo mentre **java::default** indica che all’interno del cookbook java, presente all’interno della cartella cookbooks, va utilizzata la recipe default. L’elenco di tutti i recipe è chiamato “**run-list**”. Lo stesso ragionamento si applica al caso di glassfish. Come detto, è anche possibile andare a personalizzare ogni recipes e questo è possibile farlo mediante JSON.

**Puppet**

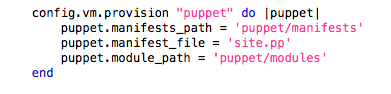
Come Chef, anche Puppet è uno strumento per automatizzare la gestione delle infrastrutture. Il suo scopo è quello di automatizzare le attività ripetitive per quanto riguarda il setup di ambienti di produzione e distribuzione di applicazioni mediante la definizione di un flusso di attività e di regole che devono essere applicate per ottenere la configurazione desiderata. Puppet è uno strumento molto efficace ma, rispetto a Chef, è più complicato. Nell’ambito del lavoro di tesi, Puppet è stato usato per configurare una vagrant box con Glassfish.

Puppet funziona con un proprio linguaggio di configurazione che è scritto in file chiamati “**manifests**” che hanno estensione .pp . Per mantenere il tutto nel modo più ordinato possibile è opportuno creare una cartella chiamata **puppet** nella directory principale del progetto in questione. All’interno della cartella puppet è necessario creare una directory chiamata **manifests** dove andremo a posizionare i file manifest. All’interno dei file di manifest sono definite delle classi che specificano delle risorse.



Nel frammento di codice precedente sono definite due classi: system-update ed apache. E’ possibile anche dichiarare delle variabili, si veda ad esempio $syspackage che, in questo caso, è un array di package che necessitano di essere installati. L’ordine con il quale sono definite le classi non ha nessuna influenza sul modo con il quale puppet esegue i comandi. Le dipendenze possono far variare l’ordine di esecuzione dei task in modo dinamico.

Per utilizzare Vagrant, nel Vagrantfile, è necessario inserire un frammento di configurazione simile al seguente.



Mediante tali semplici righe di configurazione indicheremo che verrà usato Puppet quale strumento di provisioning. Al fine di migliorare l’attività di provisoning, è consigliato distribuire la configurazione in più file, chiamati **Puppet Modules**. Per tale scopo è necessario creare, all’interno della cartella puppet, una directory di nome **modules** che dovrà contenere i vari moduli. Per modulo si intendono directory e file organizzati secondo una specifica struttura. I file di manifest presenti all’interno di un modulo devono rispettare alcuni vincoli sul loro nome.

Nel Vagrantfile è necessario specificare che tutti i moduli necessari al provisioning del sistema si trovano nel path “puppet/modules”. Tale configurazione è specificata mediante il comando:

puppet.module\_path = “puppet/modules”

I moduli devono inoltre rispettare alcuni vincoli:

* Ogni modulo deve essere contenuto in una directory ed il nome della directory è il nome del modulo.
* Ogni directory deve avere una sotto-directory di nome manifest che contiene tutti i file con estensione .pp
* Il file manifest deve avere nome init.pp, deve essere contenuto in una directory di nome manifests, ed al suo interno ci deve essere la definizione di una singola classe che deve avere lo stesso nome del modulo.

Si supponga di avere un progetto Vagrant che fa uso di Puppet con la configurazione suddivisa in due moduli: system-update e apache. Un esempio di strutturazione di directory per un tale progetto potrebbe essere il seguente.

* Project Root ( ./ )
  + puppet
    - manifests
      * site.pp
    - modules
      * apache
        + manifests

init.pp

* + - * system-update
        + manifests

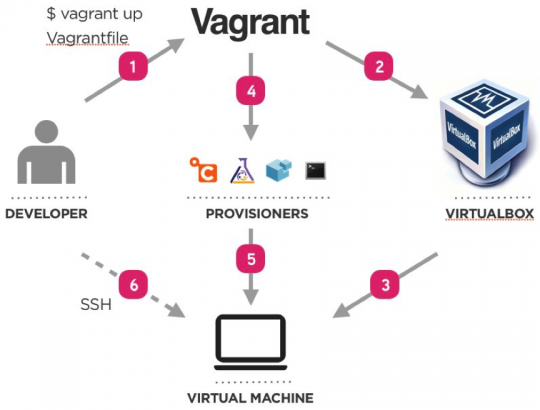
init.pp

* + Vagrantfile
  + (...)

Come indicato nella configurazione di Vagrant i vari moduli di Puppet si trovano all’interno della directory puppet/modules mentre il file di manifest principale, dal nome site.pp, è contenuto in puppet/manifest.

**Ciclo di sviluppo**

A questo punto, avendo passato brevemente in rassegna tutti gli strumenti che possono essere coinvolti nell’utilizzo di Vagrant, è utile fornire un quadro d’insieme per capire qual è il flusso di lavoro in un progetto che fa uso di tale tecnologia per gestire ambienti virtualizzati.



Lo sviluppatore interagisce con Vagrant utilizzando alcuni semplici comandi. Il principale comando usato è “vagrant up” (1) che ha lo scopo di effettuare il booting della macchina virtuale ed equivale al pulsante di accensione di una macchina fisica. Lo sviluppatore deve provvedere a fornire un file denominato “Vagrantfile” che contiene alcune direttive di base per creare una macchina virtuale con le caretteristiche necessarie ad ospitare l’applicazione che si intende sviluppare. A questo punto Vagrant interagirà con un hypervisor (solitamente VirtualBox) dandogli le istruzioni per la creazione di una macchina virtuale (2-3). (4) Vagrant può anche interagire con strumenti di provisioning (come Chef o Puppet) al fine di installare tutto il software richiesto per rendere eseguibile il progetto all’interno della macchina virtuale (5). Una volta che la macchina virtuale è stata creata, avviata e configurata, lo sviluppatore può accedervi (6) attraverso SSH. Questo metodo d’accesso ricorda quello verso un server remoto. Nel caso di Vagrant il server remoto risiede però in una macchina virtuale posizionata all’interno del personal computer dello sviluppatore.

*http://www.digitalforreallife.com/2012/11/boosting-teamwork-with-vagrant/*

<http://www.beinnova.it/vagrant-ambiente-di-sviluppo-ortatile/>

<http://brunobuccolo.com/vagrant-chef-and-opsworks/>

<http://www.lerrigatto.com/sviluppo-web-con-amazon-ec2/>

<http://www.beinnova.it/vagrant-chef-sol-cotto-e-configurato/>

http://www.example42.com/tutorials/PuppetTutorial/#slide-7

<http://www.digitalforreallife.com/2012/11/boosting-teamwork-with-vagrant/>

http://www.erikaheidi.com/blog/a-beginners-guide-to-vagrant-part-2-provisioning-and-puppet