



Luca Cabibbo
Architettura
dei Sistemi
Software

Strumenti per la gestione di ambienti virtuali

dispensa asw870
ottobre 2024

*Why You Should Never Use The Phrase:
'But It Works On My Machine'.
Kevin Wanke*



- Riferimenti

- ❑ Luca Cabibbo. **Architettura del Software: Strutture e Qualità**. Edizioni Efestò, 2021.
 - Capitolo 36, **Gestione di ambienti**
- ❑ Oracle VM VirtualBox
<https://www.virtualbox.org/>
- ❑ Vagrant by HashiCorp
<https://www.vagrantup.com/>
- ❑ Puppet by PuppetLabs
<https://puppet.com/>
- ❑ Hashimoto, M. **Vagrant: Up and Running**. O'Reilly, 2013.



- Obiettivi e argomenti

□ Obiettivi

- presentare ed esemplificare alcuni strumenti per la creazione e la gestione di ambienti di esecuzioni distribuiti virtuali

□ Argomenti

- introduzione
- VirtualBox
- Vagrant
- provisioning con Vagrant
- provisioning con Vagrant e Puppet
- discussione



* Introduzione

- Questa dispensa presenta brevemente – e solo a titolo di esempio – alcuni strumenti per la creazione e la gestione di ambienti di esecuzione, virtuali e distribuiti, nel proprio personal computer
 - l'intenzione è descrivere degli strumenti di supporto allo sviluppo e al test di semplici applicazioni distribuite – e non tanto per la gestione degli ambienti di produzione per applicazioni distribuite complesse
 - in particolare, gli strumenti descritti sono
 - VirtualBox
 - Vagrant
 - Puppet Apply
 - altri strumenti per la gestione di ambienti virtuali consentono di fare delle cose analoghe – ed anche molto di più



* VirtualBox

- ❑ **Oracle VM VirtualBox** (www.virtualbox.org) è un prodotto software di virtualizzazione per sistemi x86
 - è un hypervisor di tipo 2
 - per OS host Windows, Linux e MacOS
 - per OS guest Windows e Linux
 - un progetto open source controllato dalla Oracle
 - con licenza GNU – per uso personale o di valutazione
 - è anche possibile acquistare una licenza commerciale – per un uso commerciale o enterprise

- ❑ Un uso comune di VirtualBox è quello delle VM pre-costruite (virtual appliance) per sviluppatori
 - è possibile sperimentare uno stack software complesso installando solo VirtualBox e scaricando una singola virtual appliance pre-definita



VirtualBox

- ❑ In questa dispensa utilizziamo VirtualBox – ma non direttamente
 - con VirtualBox, le VM possono essere create mediante una GUI oppure mediante una interfaccia CLI dalla linea di comando (VBoxManage)
 - noi però non utilizzeremo direttamente né la GUI né la CLI – ma piuttosto useremo Vagrant come intermediario per interagire con VirtualBox
 - Vagrant utilizzerà VirtualBox come “provider di virtualizzazione” – ovvero, utilizzerà la CLI di VirtualBox per noi – traducendo i propri comandi e la propria sintassi in comandi della CLI del provider di virtualizzazione utilizzato
 - a tal fine, VirtualBox deve comunque essere installato sul nostro computer



* Vagrant

- ❑ **Vagrant** (www.vagrantup.com) è un prodotto software della HashiCorp per la creazione e configurazione di ambienti di sviluppo virtuali – riproducibili e portabili
 - Vagrant non è un prodotto di virtualizzazione autonomo – piuttosto, consente di creare e gestire VM e ambienti virtuali in sistemi di virtualizzazione come VirtualBox, Hyper-V e Docker, o anche VMware o AWS – agendo come client delle API o CLI offerte da questi sistemi
 - inoltre, Vagrant supporta strumenti per la gestione delle configurazioni come Puppet, Chef e Ansible
 - disponibile per Windows, Linux e MacOS
 - nel seguito, ipotizziamo che nel nostro computer (il sistema host), oltre a VirtualBox come hypervisor, sia installato anche Vagrant



Progetti per VM o ambienti

- ❑ Vagrant fornisce delle astrazioni per la configurazione e la gestione di VM o di interi ambienti virtuali – mediante un approccio di tipo *infrastructure-as-code*
 - in pratica, la specifica di una VM o di un ambiente richiede la definizione di un *progetto* Vagrant
 - ogni progetto va definito in una cartella distinta (del nostro computer, ovvero del sistema host)
 - in particolare, questa cartella deve contenere un file di testo, di nome **Vagrantfile**, che ha lo scopo di descrivere le diverse VM richieste nel progetto, come configurarle e come farne il provisioning
 - la sintassi usata nel **Vagrantfile** è un DSL basato su Ruby
 - oltre al **Vagrantfile**, la cartella di un progetto può contenere anche altri file e cartelle
 - ad es., script bash, template e moduli Puppet



Comandi per l'utente

- Le interazioni dell'utente con Vagrant avvengono, nel sistema host, mediante una CLI (interfaccia della linea di comando) – utilizzando il comando **vagrant** (con i suoi sotto-comandi), che va eseguito nell'ambito della cartella del progetto per una VM o un ambiente
 - **vagrant up** crea, configura e avvia l'ambiente virtuale descritto dal **Vagrantfile** – è il singolo comando più importante di Vagrant
 - **vagrant ssh [<vm>]** consente di collegarsi a una VM dell'ambiente tramite SSH
 - **vagrant halt** arresta le VM dell'ambiente virtuale
 - un ambiente arrestato può poi essere avviato di nuovo con **vagrant up**
 - **vagrant destroy -f** distrugge l'ambiente virtuale



Struttura del Vagrantfile

- Un **Vagrantfile** è composto da un insieme di sezioni di configurazione, racchiuse da un elemento **Vagrant.configure**

```
# Vagrantfile
Vagrant.configure(2) do |config|
  ... configurations ...
end
```

- “2” indica la versione della sintassi da utilizzare
- **config** (tra barre verticali | e |) è il nome di un elemento da configurare
- **#** indica un commento di linea



Box

- Le impostazioni **config.vm** consentono di configurare le VM gestite da Vagrant
 - **config.vm.box** specifica l'immagine di VM da utilizzare nella creazione della VM
 - di solito selezionata da un repository di box per Vagrant
 - ad es., **hashicorp/precise64** (Ubuntu 12.04 LTS a 64 bit), **bento/ubuntu-24.04** (Ubuntu 24.04 LTS Server a 64 bit) o **centos/8** (basata sulla versione più recente di CentOS 8)

```
# Vagrantfile
Vagrant.configure(2) do |config|

  # Ubuntu 24.04 LTS
  config.vm.box = "bento/ubuntu-24.04"

end
```

questo è un
Vagrantfile
minimale ma
“eseguibile”



Box

- I **box** sono il formato delle immagini usato da Vagrant
 - il modo più semplice per trovare un box è usare il repository pubblico di box per Vagrant – <https://app.vagrantup.com/>
 - è anche possibile usare dei box personalizzati in un repository “privato” – o anche condividere pubblicamente i propri box
 - i box sono soggetti a versionamento – e possono essere aggiornati facilmente
 - un box di base (**base box**) contiene un OS pre-installato e pre-configurato
 - i box pubblici, per convenzione, definiscono un utente con credenziali **vagrant/vagrant**, abilitato per SSH
 - un box, oltre all'OS, può contenere anche altro software pre-installato e pre-configurato – ad es., Puppet o Chef



Impostazioni di rete (e affini)

- Le impostazioni **config.vm.network** consentono di configurare le connessioni di rete per le VM – ecco alcuni esempi
 - **config.vm.network "private network", type: "dhcp"** specifica che la VM va collegata a un rete virtuale privata, con l'assegnazione dell'indirizzo IP tramite DHCP (di solito gestito dal provider di virtualizzazione)
 - **config.vm.network "private network", ip: "10.11.1.191"** specifica un indirizzo IP statico
 - **config.vm.hostname "server"** specifica l'hostname della VM
 - le impostazioni di rete vengono automaticamente riconfigurate ogni volta che l'ambiente viene avviato o riavviato con **vagrant up**



Port forwarding

- Il **port forwarding** consente di accedere a una porta di una VM guest tramite una porta del sistema host
 - **config.vm.network "forwarded_port", guest: 80, host: 8080** specifica che la porta 80 della VM guest sia mappata sulla porta 8080 del sistema host – e dunque possa essere acceduta tramite di essa
 - ad es., il web server in esecuzione nella VM può essere acceduto dall'host all'URL **http://localhost:8080**
 - per default, la porta 22 (SSH) del guest è reindirizzata sulla porta 2222 dell'host
 - per default, una porta reindirizzata può essere acceduta in rete da chiunque possa accedere al computer host
 - ci sono opzioni per limitare l'accesso alle porte reindirizzate
 - l'opzione **auto_correct: true** consente di risolvere automaticamente conflitti tra porte reindirizzate da più VM



Cartelle condivise

- Vagrant consente la condivisione e sincronizzazione di cartelle tra una VM guest e il sistema host
 - ad es., per editare dei file localmente nel sistema host – ma per compilarli ed eseguirli nella VM guest
 - per default, la cartella radice del progetto Vagrant (nell'host) viene condivisa nella VM guest come cartella **/vagrant**
 - **config.vm.synced_folder "host_folder", "/guest/folder"** consente di condividere una cartella dell'host con una cartella della VM guest
 - se il path dell'host è relativo, allora lo è rispetto alla cartella radice del progetto
 - il path del guest deve essere assoluto
 - anche le impostazioni relative alla condivisione di cartelle vengono automaticamente riconfigurate ogni volta che l'ambiente viene avviato o riavviato con **vagrant up**

15

Strumenti per la gestione di ambienti virtuali

Luca Cabibbo ASW



Impostazioni specifiche

- Vagrant consente anche di configurare delle impostazioni specifiche per il provider di virtualizzazione in uso
 - nel caso di VirtualBox, è possibile richiedere l'esecuzione di ogni comando **VBoxManage** (la CLI di VirtualBox) prima dell'avvio o riavvio della VM
 - c'è anche una sintassi semplificata per alcune impostazioni dei processori virtuali e della memoria

```
# VirtualBox custom settings
config.vm.provider "virtualbox" do |v|
  v.memory = 1024
  v.cpus = 2
  v.customize ["modifyvm", :id, "--cpuexecutioncap", "50"]
end
```

16

Strumenti per la gestione di ambienti virtuali

Luca Cabibbo ASW



Ambienti composti da più VM

- Vagrant consente anche di configurare ambienti virtuali composti da più VM
 - tramite le impostazioni **config.vm.define**

```
# Vagrantfile for a multi-machine environment
Vagrant.configure("2") do |config|

  config.vm.define "web" do |web|
    web.vm.box = "..."

    ...
  end

  config.vm.define "db" do |db|
    db.vm.box = "..."

    ...
  end

end
```



Ambienti composti da più VM

- Vagrant consente anche di configurare ambienti virtuali composti da più VM
 - **config.vm.define** consente di specificare delle configurazioni (**web** e **db**) annidate entro un'altra configurazione (**config**)
 - le impostazioni di **config** sono ereditate dalle configurazioni annidate – ma possono anche essere sovrascritte
 - la comunicazione tra le VM va configurata usando le impostazioni di rete
 - in particolare, è possibile creare una rete privata che collega le VM tra di loro e con il sistema host
 - il comando **vagrant up** consente di creare e avviare tutte le VM dell'ambiente – ma **vagrant up <vm>** consente di creare e avviare una sola VM dell'ambiente
 - **vagrant ssh <vm>** consente di collegarsi a una specifica VM dell'ambiente



Ambienti composti da più VM

- ❑ Vagrant, basandosi su Ruby, consente anche di configurare ambienti virtuali composti da più VM e parametrici, iterando sulle definizioni delle VM

```
# Vagrantfile for a multi-machine environment
Vagrant.configure("2") do |config|

  # creates 3 nodes
  (1..3).each do |i|

    config.vm.define "node-#{i}" do |node|
      # configure node-i
      node.vm.box = "..."
      ...
    end
  end

end

end
```



Vagrant up

- ❑ Ecco che cosa fa Vagrant (per ciascuna VM) quando viene eseguito il comando **vagrant up**
 - copia il box specificato in una cache locale dell'host
 - crea una nuova VM tramite il provider di virtualizzazione
 - prepara le schede di rete virtuali
 - configura il port forwarding della VM
 - avvia la VM
 - configura e abilita la rete
 - configura le cartelle condivise tra l'host e la VM guest
 - effettua il provisioning (del software) della VM



Vagrant up

- ❑ Ecco che cosa fa Vagrant (per ciascuna VM) quando viene eseguito il comando **vagrant up**
 - copia il box specificato in una cache locale dell'host (*)
 - crea una nuova VM tramite il provider di virtualizzazione (*)
 - prepara le schede di rete virtuali (**)
 - configura il port forwarding della VM (**)
 - avvia la VM (**)
 - configura e abilita la rete (**)
 - configura le cartelle condivise tra l'host e la VM guest (**)
 - effettua il provisioning (del software) della VM (***)
- (*) = se non è stato già fatto in precedenza
- (**) = ogni volta
- (***) = se il provisioning non è stato già effettuato in precedenza oppure se ne è stata richiesta la ripetizione



* Provisioning con Vagrant

- ❑ Vagrant consente di effettuare il provisioning del software delle VM create – per installare software aggiuntivo e per modificare le configurazioni in modo automatizzato
 - in Vagrant, questa attività è chiamata semplicemente *provisioning*
 - il provisioning è utile perché i box predefiniti di solito non sono perfettamente adatti per l'uso che si intende fare delle VM
 - il provisioning manuale è possibile (con **vagrant ssh**) – ma è un anti-pattern
 - i vantaggi del provisioning automatizzato sono già stati descritti nel capitolo sulla Gestione di ambienti – il più importante è probabilmente la ripetibilità



Opzioni di provisioning

- Vagrant offre più opzioni di provisioning (*provisioner*)
 - mediante alcune impostazioni del **Vagrantfile**
 - mediante shell – con comandi di scripting oppure interi script, ad es., bash
 - di solito è il modo più semplice per iniziare
 - ma non è il modo migliore o il più potente – ad es., può essere difficile scrivere script idempotenti o che sostengono l'autonomia
 - mediante l'uso di strumenti per la gestione delle configurazioni
 - come Puppet, Chef o Ansible



Provisioning mediante shell

- Ogni sezione di provisioning inizia con **config.vm.provision** – e specifica il tipo di provisioner da utilizzare
 - ad es., il provisioning mediante **shell** (con diverse varianti)

```
# Vagrantfile
Vagrant.configure(2) do |config|

  # other configurations
  ...

  # provisioning
  config.vm.provision "shell", inline: "echo hello"

  # provisioning (alternative syntax)
  config.vm.provision "shell" do |s|
    s.inline = "echo hello"
  end
end
```



Provisioning di file

- Il provisioning di **file** consente di copiare un file o una cartella dal sistema host alla VM guest (con **scp**)

```
# file provisioning
config.vm.provision "file", source: "~/.gitconfig",
                    destination: ".gitconfig"
```

- diversamente dalle cartelle condivise (che sono sincronizzate in modo continuo), il provisioning di file viene effettuato solo al momento del provisioning



Quando avviene il provisioning

- Il provisioning viene eseguito di solito solo durante la **prima** esecuzione di **vagrant up** per una VM o ambiente – per creare e configurare la VM (o l'ambiente virtuale)
 - invece, durante le successive esecuzioni di **vagrant up**, alcune attività vengono ripetute (come la configurazione della rete e delle cartelle condivise) – ma il provisioning, di solito, non viene ripetuto
 - è possibile però specificare che una sezione di provisioning vada ripetuta in ogni esecuzione di **vagrant up** mediante la clausola **run: "always"**

```
# provisioning (run always)
config.vm.provision "shell", run: "always" do |s|
  s.inline = "echo hello"
end
```

- questo è utile, ad es., per mandare in esecuzione un'applicazione o un servizio ad ogni avvio di una VM



Quando avviene il provisioning

- Il provisioning viene eseguito di solito solo durante la **prima** esecuzione di **vagrant up** per una VM o ambiente – per creare e configurare la VM (o l'ambiente virtuale)
 - è anche possibile richiedere la ripetizione del provisioning di una VM (o di un ambiente)
 - con il comando **vagrant up --provision** su una VM arrestata
 - con il comando **vagrant provision** su una VM in esecuzione
 - è anche possibile effettuare o ripetere il provisioning in modo selettivo
 - ad es., sulla base del tipo del provisioner oppure assegnando dei nomi alle sezioni di provisioning e indicando quali sezioni ripetere



- Provisioning mediante shell

- Il provisioning mediante **shell** può essere di tipo **inline** oppure di tipo **path** (ovvero, basato sull'esecuzione di uno script esterno)

```
# provisioning with an external script
config.vm.provision "shell", path: "script.sh"

# provisioning (alternative syntax)
config.vm.provision "shell" do |s|
  s.path = "script.sh"
end
```

 - ad es., lo script **script.sh** potrebbe essere

```
#!/bin/bash
echo "hello"
```
- con l'opzione **path**, lo script deve risiedere sull'host – Vagrant lo copia nel guest e poi lo esegue
 - in alternativa, si può copiare uno script sul guest e poi avviarlo usando l'opzione **inline**



Esempio: Apache HTTP Server (modo 1)

- Come esempio, presentiamo tre diversi modi per effettuare il provisioning di una VM con Apache HTTP Server con Vagrant
 - il primo modo è il provisioning mediante uno script bash

```
Vagrant.configure("2") do |config|
  config.vm.box = "bento/ubuntu-24.04"
  config.vm.network "private_network", ip: "10.11.1.191",
                                     virtualbox____intnet: true
  config.vm.network "forwarded_port", guest: 80, host: 8081
  config.vm.provision :shell, path: "install-apache.sh"
end
```

- si noti come il servizio HTTP della VM viene reindirizzato sulla porta 8081 dell'host



Esempio: Apache HTTP Server (modo 1)

- Ecco lo script `install-apache.sh` per effettuare il provisioning di Apache HTTP

```
#!/bin/bash
apt-get update
apt-get install -y apache2
if ! [ -L /var/www/html ]; then
  rm -rf /var/www/html
  ln -fs /vagrant/www /var/www/html
fi
```

- Apache HTTP serve i file contenuti nella cartella `/var/www/html` – che è linkata alla cartella `/vagrant/www` della VM – che è a sua volta sincronizzata (implicitamente) con la cartella `www` del progetto Vagrant per l'ambiente
- provare con GET `http://localhost:8081`



- Installazione di applicazioni

- Gli strumenti discussi finora possono essere utilizzati
 - per l'installazione e la configurazione dello stack software di interesse in una VM – OS e middleware
 - ma anche per il deployment (installazione e configurazione) delle proprie applicazioni – o di componenti e servizi applicativi
 - anche questa attività può infatti essere di solito ricondotta alla copia dei file dell'applicazione e/o all'esecuzione di opportuni script – per installare, configurare e avviare l'applicazione – e/o all'installazione di un package con un package manager (se l'applicazione è stata assemblata in questo modo)



* Provisioning con Vagrant e Puppet

- Vagrant consente anche di effettuare il provisioning delle VM mediante l'uso di sistemi per la gestione delle configurazioni – tra cui *Puppet*
 - Puppet può essere utilizzato in due modalità
 - in modalità agent-server
 - richiede un *Puppet Server* che gestisce le configurazioni software di tutti gli ambienti
 - nel nodo da configurare deve essere installato un agente (*Puppet Agent*)
 - in una modalità autocontenuta (*Puppet Apply*) – in cui la configurazione di un nodo è gestita e applicata localmente nel nodo
 - qui esemplifichiamo un uso elementare di Puppet Apply



- Provisioning di base con Puppet

- La forma più semplice di provisioning in Vagrant con Puppet (senza i moduli, che sono esemplificati più avanti)

```
Vagrant.configure("2") do |config|  
  config.vm.box = "bento/ubuntu-24.04"  
  
  config.vm.network "private_network", ...come prima...  
  config.vm.network "forwarded_port", guest: 80, host: 8082  
  
  config.vm.provision :shell, path: "install-puppet.sh"  
  
  config.vm.provision "puppet"  
  
end
```

- Puppet (Puppet Apply) deve essere installato nel box
 - in alcuni box è preinstallato
 - qui lo installiamo con lo script `install-puppet.sh`
- Puppet userà il suo punto di ingresso di default – che è il manifesto `default.pp` nella cartella `manifests`



Esempio: Apache HTTP Server (modo 2)

- Ecco un secondo modo per effettuare il provisioning di una VM con Apache HTTP Server con Vagrant
 - utilizza il provisioning di base con Puppet
 - il file `manifests/default.pp` (mostrato dopo) descrive i package e i servizi da installare, eventuali comandi da eseguire, nonché altre informazioni di configurazione
 - provare con GET <http://localhost:8082>



Esempio: Apache HTTP Server (modo 2)

- il file manifests/default.pp

```
node 'default' {
  include apache
}

class apache {
  package { 'apache2':
    ensure => present
  }
  service { 'apache2':
    require => Package['apache2'],
    ensure => running,
    enable => true,
  }
  file { '/var/www/html':
    require => Package['apache2'],
    target => '/vagrant/www',
    ensure => link,
    force => true
  }
}
```

35

Strumenti per la gestione di ambienti virtuali

Luca Cabibbo ASW



- Uso di moduli Puppet predefiniti

- ▣ Puppet (come altri sistemi per la gestione delle configurazioni) offre un repository pubblico di moduli (configurazioni) predefiniti ma parametrici, riusabili e condivisibili – *Puppet Forge* (<https://forge.puppet.com/>)
 - i moduli semplificano la configurazione dei nodi
 - i moduli possono essere scaricati e utilizzati in modo personalizzato
 - in un progetto Vagrant
 - i moduli Puppet di interesse possono essere scaricati e salvati nella cartella `puppet/modules`
 - i manifesti Puppet (personalizzati per il progetto) possono essere nella cartella `puppet/manifests`
 - ad es., il modulo Puppet per Apache HTTP Server è `apache` – insieme a questo modulo è necessario utilizzare anche le sue dipendenze (`stdlib` e `concat`) – in versioni compatibili tra loro

36

Strumenti per la gestione di ambienti virtuali

Luca Cabibbo ASW



Esempio: Apache HTTP Server (modo 3)

- Ecco un terzo modo per effettuare il provisioning di una VM con Apache HTTP Server con Vagrant
 - utilizza il provisioning con i moduli di Puppet
 - la VM viene configurata utilizzando e personalizzando il modulo Puppet per Apache HTTP Server

```
Vagrant.configure("2") do |config|
```

```
  config.vm.box = "bento/ubuntu-24.04"
```

```
  config.vm.network "private_network", ...come prima...
```

```
  config.vm.network "forwarded_port", guest: 80, host: 8083
```

```
  config.vm.provision :shell, path: "install-puppet.sh"
```

```
  config.vm.provision :shell, path: "download-apache-puppet-modules.sh"
```

```
  config.vm.provision "puppet" do |puppet|
```

```
    puppet.manifests_path = "puppet/manifests"
```

```
    puppet.manifest_file = "init.pp"
```

```
    puppet.module_path = "puppet/modules"
```

```
  end
```

37

end

Strumenti per la gestione di ambienti virtuali

Luca Cabibbo ASW



Esempio: Apache HTTP Server (modo 3)

- Questo è il manifesto `init.pp` che applica il modulo `puppetlabs/apache`
 - si noti che la personalizzazione (che in questo caso è elementare) è comunque più semplice che non negli esempi precedenti

```
class { 'apache':  
  docroot => '/vagrant/www'  
}
```

- provare con GET <http://localhost:8083>

38

Strumenti per la gestione di ambienti virtuali

Luca Cabibbo ASW



* Discussione

- ❑ In pratica, un progetto Vagrant – per una singola VM o per un intero ambiente virtuale – può essere composto da un insieme auto-contenuto (ovvero localizzato in una singola cartella) di file di configurazione testuali (*infrastructure-as-code*)
 - l'ambiente può essere creato e configurato – in modo automatizzato – usando il singolo comando **vagrant up**
 - questi ambienti sono riproducibili e portabili
 - si può dire addio alla scusa “ma sul mio PC funziona” – poiché un progetto Vagrant porta sempre (o, almeno, in teoria 😊) a un ambiente di sviluppo identico
 - i progetti Vagrant possono anche essere condivisi facilmente – pubblicamente o nella propria organizzazione
 - “installare un'applicazione, un server o una piattaforma complessa è semplice come scaricare un'app sul tuo smartphone”



Discussione

- ❑ Anche altri strumenti per la gestione di ambienti offrono astrazioni e funzionalità analoghe a quelle di Vagrant – ed anche più ricche
 - gestione automatizzata di ambienti, sulla base di un approccio infrastructure-as-code
 - creazione di VM a partire da un insieme di informazioni di configurazione e da un'immagine di base
 - provisioning di VM basata su
 - copia di file e template
 - esecuzione di script
 - uso di ulteriori strumenti per la gestione delle configurazioni
 - ci sono anche strumenti per la gestione di ambienti mediante hypervisor di tipo 1 e nel cloud