Ansible

Ottavo laboratorio Tecnologie open-source

Table of Contents

Introduzione ad Ansible

Ansible é un tool per la configurazione ed il coordinamento di software su differenti macchine, in particolare si installa su di una macchina che avrà la funzione di server centrale che comunicherà con altre macchine chiamate nodi. La peculiarità di Ansible e strumenti di questo genere é che non bisogna installare una versione client, infatti é sufficiente che i nodi abbiamo sshd e python, nessun altro daemon o agente é richiesto, in più viaggiando su interfaccia ssh gode di una discreta sicurezza.

Prerequisiti

- Vagrant
- VirtualBox (max ver:6.0)

Preparazione

- Posizionarsi in una cartella a piacere.
- Eseguire nella powershell (Windows) o shell (linux) il comando
  - `vagrant init ubuntu/trusty64`
- Far partire la macchina virtuale appena creata con il comando
  - `vagrant up`

IMPORTANT	In caso di errore assicurarsi che la versione di virtualbox sia ⇐6.0 e che tutti i moduli siano attivi

- Connettersi alla macchina virtuale tramite il comando
  - `vagrant ssh`
- inserire la seguente password quando e se richiesto
  - `vagrant`

Installazione Ansible

- Dalla shell della macchina virtuale digitiamo i seguenti due comandi

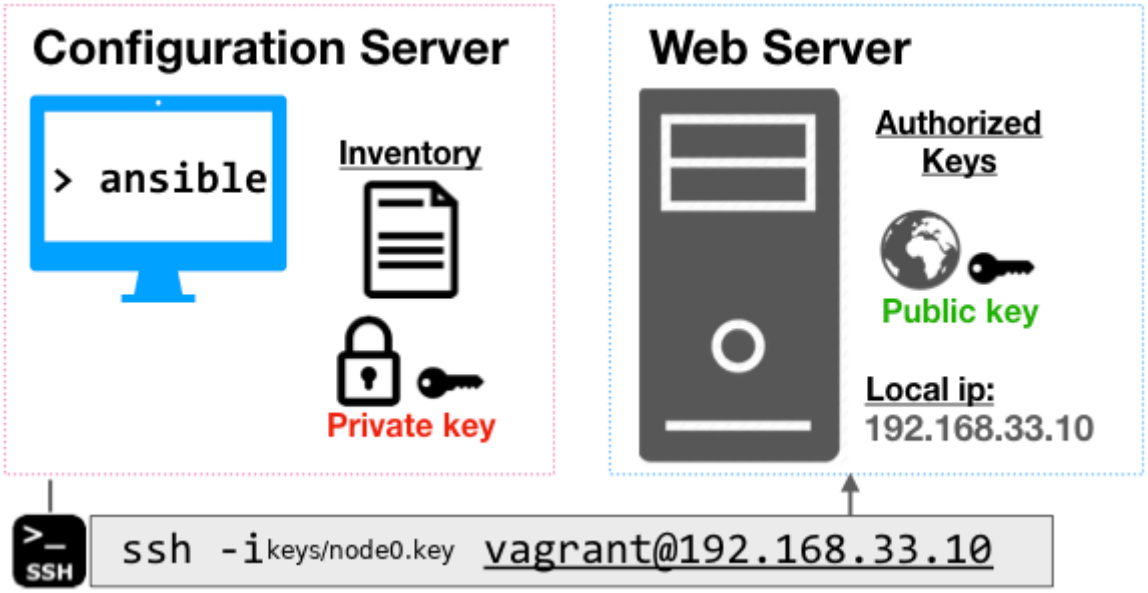
```
$ sudo apt-get update
$ sudo apt-get -y install git make vim python-dev python-pip libffi-dev libssl-dev
libxml2-dev libxslt1-dev libjpeg8-dev zlib1g-dev
```

- Quindi se tutto é andato bene digitare
  - `pip install ansible`

IMPORTANT	Se il precedente comando dovesse fallire é possibile installare ansible tramite il package manager della distribuzione in uso (Ubuntu) tramite il seguente comando: `sudo apt-get install ansible`

- Possiamo ora testare ansible col comando
  - `ansible all -m ping`

Macchina node



Ora che Ansible é stato installato sulla macchina server, non ci resta che reperire alcune informazioni dalla parte client, che in genere si tratterà di un webserver hostato su AWS o da qualche altra parte, quindi per completare la configurazione non faremo altro che creare un'altra macchina virtuale.

- Inizializziamo una nuova macchina virtuale in una sottocartella nel nostro pc fisico che, per comodità, chiameremo "node" aprendo una nuova scheda powershell (Windows) o shell (Linux) nella cartella appena creata
  - `vagrant init ubuntu/trusty64`

NOTE	Recarci in una sottocartella é essenziale in quanto non é possibile avere più Vagrantfile (e quindi macchine virtuali) per cartella

- Modifichiamo il Vagrantfile della cartella "node" in modo da eliminare lo sharp (#) dalla linea contente
  - `config.vm.network "private_network", ip: "192.168.33.10"`

NOTE	Passaggio utile per stabilire una connessione fra le due macchine virtuali

- Eseguire il comando per far partire la macchina virtuale
  - `vagrant up`
- Troviamo la locazione della chiave ssh generata con il precedente comando eseguendo il

comando

- `vagrant ssh-config`

NOTE	Appuntarsi la chiave o l'intero file in quanto ci servirà per dopo

Configurare il server

La struttura di un comando ansible é relativamente semplice

```
ansible all -s -m apt -i inventory -a "name=gparted state=installed"
```

- host (all): su quale nodo o gruppi eseguire l'azione espressa dal modulo

NOTE	ogni host appartiene sempre a due gruppi: al gruppo 'all' ed al gruppo definito dall'utente oppure al gruppo 'all' ed 'ungrouped'

- opzione -s: il comando verrà eseguito con i diritti di amministratore
- opzione -m: specifica il modulo da utilizzare
- opzione -i: specifica il file di inventory, cioé quello in cui sono definiti tutti gli host ed i gruppi
- opzione -a: argomenti del modulo scelto se necessari

Per definire quindi il file di inventory, per stabilire una connessione fra la macchina virtuale server e quella node, abbiamo a disposizione diverse opzioni

- creare un file (chiamato ad esempio `inventory`) con dentro il seguente contenuto
  - `node0 ansible_ssh_host=192.168.33.10 ansible_ssh_user=vagrant ansible_ssh_private_key_file=./keys/node0.key`
  - creare il file `node0.key` dentro la cartella `keys/` ed eseguire il comando `chmod 500 keys/node0.key` per fornire le giuste autorizzazioni

NOTE	La "ansible_ssh_private_key_file" é proprio quella appuntata in precedenza, non resta che incollarla nel percorso definito dalla variabile stessa

- modificare il file `/etc/ansible/hosts` aggiungendo la seguente riga nella sezione "Ex. 1"
  - `192.168.33.10`
  - copiare il file della chiave appuntato in precedenza ed incollarlo nella cartella `.ssh/`, quindi rinominarlo in `id_rsa`
- modificare il file `/etc/ansible/hosts` aggiungendo la seguente riga nella sezione "Ex. 1"
  - `192.168.33.10 ansible_ssh_user=vagrant ansible_ssh_private_key_file=./keys/node0.key`
  - creare il file `node0.key` dentro la cartella `/etc/ansible/hosts/keys/` ed eseguire il comando `chmod 500 keys/node0.key` per fornire le giuste autorizzazioni

Non ci resta che testare la connessione con il comando

- `ansible node0 -m ping -i inventory -vvvv` se avete creato il file con node0

oppure

- `ansible 192.168.33.10 -m ping -vvvv` se avete usato gli altri metodi

> **NOTE** `-vvvv` rende l'esecuzione verbosa

Web server di prova

Proviamo ora ad installare un webserver sulla macchina nodo.

- Installiamo nginx
  - `ansible 192.168.33.10 -s -m apt -i inventory -a 'pkg=nginx state=installed update_cache=true'`
- Avviamo il webserver sul nodo
  - `ansible all -s -m shell -i inventory -a 'nginx'`
- Apriamo il browser del computer e visitiamo la pagina `http://192.168.33.10`

Se il tutto ha funzionato possiamo rimuovere nginx e le sue dipendenze con i comandi

```
$ ansible 192.168.33.10 -s -m apt -i inventory -a 'pkg=nginx state=absent
update_cache=true'
$ ansible 192.168.33.10 -s -m shell -i inventory -a 'sudo apt-get -y autoremove'
```

File Inventory

Unico scopo é quello di definire gli host che deve gestire Ansible, raggruppandoli ed eventualmente definendoci delle variabili. Possono essere creati gruppi di gruppi (con la keyword `children`) ed un host può essere membro di più gruppi. Possono essere scritti in formato

- INI

```
192.168.33.10

[gruppo1]
192.xxx.xxx.xxx http_port=80 maxRequestsPerChild=808

[gruppo2]
192.xxx.xxx.xxx
192.[205:211].yyy.yyy

[gruppo3:children]
gruppo1
gruppo2
```

- YAML

```
all:
  hosts:
    mail.example.com:
  children:
    webservers:
      hosts:
        foo.example.com:
        bar.example.com:
    dbservers:
      hosts:
        one.example.com:
        two.example.com:
        three.example.com:
    east:
      hosts:
        foo.example.com:
        one.example.com:
        two.example.com:
    west:
      hosts:
        bar.example.com:
        three.example.com:
    prod:
      hosts:
        foo.example.com:
        one.example.com:
        two.example.com:
    test:
      hosts:
        bar.example.com:
        three.example.com:
```

NOTE con `192.[205:211].yyy.yyy` si intendono la lista di indirizzi `192.205.yyy.yyy` `192.206.yyy.yyy` `192.207.yyy.yyy` `192.208.yyy.yyy` ⋯. Funziona anche con una serie alfabetica del tipo `[a:f]` e la stessa sintassi si usa anche nel formato YAML.

É possibile definire degli alias

- INI

```
prova ansible_port:8080 ansible_host:192.168.33.10
```

- YAML

```
all:
  hosts:
    prova:
      ansible_port:8080
      ansible_host:192.168.33.10
```

Ed é anche possibile definire delle variabili per un intero gruppo

- INI

```
[prova]
host1
host2

[prova:vars]
ntp_server=ntp.prova.example.com
proxy=proxy.prova.example.com
```

- YAML

```
prova:
  hosts:
    host1:
    host2:
  vars:
    ntp_server: ntp.prova.example.com
    proxy: proxy.prova.example.com
```

YAML

Vari modi per strutturare il file

YAML come lista (sequenza ordinata)

- In linea

```
[birth, taxes, death]
```

- Su più linee

```
- birth
- taxes
- death
```

NOTE Lo spazio dopo il trattino é regola di linguaggio quindi va messo *sempre

YAML come hash (coppia chiave-valore)

- In linea

```
- {item: shirt, colour: red, size: 42}
- {item: shirt, colour: yellow, size: 44}
```

NOTE Lo spazio dopo i due punti (:) é regola di linguaggio quindi va messo **sempre**

- Su più linee

```
- item: shirt
  colour: red
  size: 42
  description: |
      this is a very long multi-line text field which is all
      one value Space after colon required
- item: shirt
  colour: red
  size: 42
  description: |
      this is a very long multi-line text field which is all
      one value Space after colon required
```

NOTE L'indentazione é regola di linguaggio quindi va rispettata **sempre**

Playbook

I playbook non sono altro che una lista di host o gruppi in formato YAML ai quali vengono assegnati attività o ruoli, come nell'esempio seguente

```
- hosts:
    - pc1.example.com
    - pc3.example.com
  tasks:
    - name: install Apache
      action: apt pkg=apache2 state=present
    - name: ensure Apache is running
      action: service name=apache2 state=running
- hosts: dns_servers
  roles:
    - dns_server
    - ntp
```

NOTE	L'indentazione é regola di linguaggio quindi va rispettata **sempre**

I Ruoli non sono altro che gruppi di attività concernenti lo stesso ambito.

I Tag hanno più o meno la stessa funzione, ma permettono una granularità maggiore in quanto applicabili ai singoli ruoli o task di un playbook, cosí da eseguire solo quelli richiesti tramite l'opzione `-t <tag>` del comando `ansible-playbook`

É possibile definire anche delle condizioni

```
- action: apt pkg=apache2 state=present
  when: ansible_os_family=='Debian'
```

o dei cicli

```
- action: apt pkg={{item}} state=present
  with_items:
    - openssh
    - server
    - acpid
    - rsync
    - telnet
```

Facts

Sono variabili che contengono le informazioni raccolte dai sistemi target. Per conoscerne il contenuto bisogna utilizzare il module `setup`, con output del tipo

```
host1 | success >> {
  "ansible_facts": {
    "ansible_distribution": "Ubuntu",
    "ansible_distribution_version": "12.04",
    "ansible_domain": "ws.nsrc.org",
    "ansible_eth0": {
      "ipv4": {
        "address": "10.10.0.241",
        "netmask": "255.255.255.0",
        "network": "10.10.0.0"
        }, ⋯ etc
```

Queste informazioni vengono raccolte ogni volta che si avvia una connessione, nel caso in cui non si vogliano raccogliere basta aggiungere `gather_facts: no` al playbook.

Buone pratiche

Disposizione

É buona prassi mantenere una disposizione dei file di questo tipo

```
production                # inventory file for production servers
staging                   # inventory file for staging environment

group_vars/
   group1.yml             # here we assign variables to particular groups
   group2.yml
host_vars/
   hostname1.yml          # here we assign variables to particular systems
   hostname2.yml

library/                  # if any custom modules, put them here (optional)
module_utils/             # if any custom module_utils to support modules, put them
here (optional)
filter_plugins/           # if any custom filter plugins, put them here (optional)

site.yml                  # master playbook
webservers.yml            # playbook for webserver tier
dbservers.yml             # playbook for dbserver tier

roles/
    common/               # this hierarchy represents a "role"
        tasks/            #
            main.yml      #  <-- tasks file can include smaller files if warranted
        handlers/         #
            main.yml      #  <-- handlers file
        templates/        #  <-- files for use with the template resource
            ntp.conf.j2   #  <------- templates end in .j2
        files/            #
            bar.txt       #  <-- files for use with the copy resource
            foo.sh        #  <-- script files for use with the script resource
        vars/             #
            main.yml      #  <-- variables associated with this role
        defaults/         #
            main.yml      #  <-- default lower priority variables for this role
        meta/             #
            main.yml      #  <-- role dependencies
        library/          # roles can also include custom modules
        module_utils/     # roles can also include custom module_utils
        lookup_plugins/   # or other types of plugins, like lookup in this case

    webtier/              # same kind of structure as "common" was above, done for the
webtier role
    monitoring/           # ""
    fooapp/               # ""
```

É comunque possibile usare la seguente struttura nel caso in cui le variabili dei gruppi e degli host perdono di significato al di fuori del proprio ambiente

```
inventories/
   production/
      hosts               # inventory file for production servers
      group_vars/
         group1.yml       # here we assign variables to particular groups
         group2.yml
      host_vars/
         hostname1.yml    # here we assign variables to particular systems
         hostname2.yml

   staging/
      hosts               # inventory file for staging environment
      group_vars/
         group1.yml       # here we assign variables to particular groups
         group2.yml
      host_vars/
         stagehost1.yml   # here we assign variables to particular systems
         stagehost2.yml

library/
module_utils/
filter_plugins/

site.yml
webservers.yml
dbservers.yml

roles/
    common/
    webtier/
    monitoring/
    fooapp/
```

Se invece si é costretti o nasce la necessità di utilizzare un inventory statico allora é buona regola formare diversi gruppi con diversi scopi, come ad esempio la locazione e la funzione, dato che é possibile formare gruppi di gruppi, come nell'esempio seguente

```
# file: production

[atlanta_webservers]
www-atl-1.example.com
www-atl-2.example.com

[boston_webservers]
www-bos-1.example.com
www-bos-2.example.com

[atlanta_dbservers]
db-atl-1.example.com
db-atl-2.example.com

[boston_dbservers]
db-bos-1.example.com

# webservers in all geos
[webservers:children]
atlanta_webservers
boston_webservers

# dbservers in all geos
[dbservers:children]
atlanta_dbservers
boston_dbservers

# everything in the atlanta geo
[atlanta:children]
atlanta_webservers
atlanta_dbservers

# everything in the boston geo
[boston:children]
boston_webservers
boston_dbservers
```

Un altro modo per far condividere le stesse variabili ad host dello stesso gruppo é creare file nella directory `group_vars`

```
# file: group_vars/all
ntp: ntp-boston.example.com
backup: backup-boston.example.com
```

Per una più facile gestione di esecuzione di playbook é anche possibile usare un playbook (ad esempio `site.yml`) che ne richiama altri, come nel seguente esempio

```
- import_playbook: webservers.yml
- import_playbook: dbservers.yml
```

Test vs Produzione

É sempre consigliato utilizzare prima della produzione un ambiente di test, per farlo ci basterà utilizzare diversi file di inventory, infatti ci basterà fare una gestione minima dei gruppi per trovare eventuali differenze fra i due ambienti.

Specificare sempre lo stato

Nei moduli in cui é presente lo stato é consigliato specificarlo sempre, soprattutto nei casi in cui gli stati possono essere molteplici.

es. `ansible host1 -m pkg -a "name=gparted state=present"`

Gruppi per scopo

Il concetto di creare dei gruppi sulla base dello scopo degli host é un concetto molto potente, quindi é consigliato farlo sempre.

Variazioni in base al Sistema Operativo

Nel caso in cui volessimo applicare delle variabili o compiti solo a determinati sistemi operativi é possibile creare **dinamicamente** dei gruppi nel seguente modo

```
- name: talk to all hosts just so we can learn about them
  hosts: all
  tasks:
    - name: Classify hosts depending on their OS distribution
      group_by:
        key: os_{{ ansible_facts['distribution'] }}  #variable

# now just on the CentOS hosts...

- hosts: os_CentOS
  gather_facts: False
  tasks:
    - # tasks that only happen on CentOS go here
```

Quindi secondo la disposizione precendete possiamo assegnare variabili nel seguente modo

```
---
# file: group_vars/all
asdf: 10

---
# file: group_vars/os_CentOS
asdf: 42
```

Se invece ci interessa conoscere il valore di una variabile a seconda del sistema opertaivo basta fare nel seguente modo

```
- hosts: all
  tasks:
    - name: Set OS distribution dependent variables
      include_vars: "os_{{ ansible_facts['distribution'] }}.yml"
    - debug:
        var: asdf
```

Keep it simple

Non cercare di applicare tutte queste buone pratiche assieme, se il progetto é piccolo conviene utilizzare solo un sottogruppo (ad esempio non utilizzare l'intera disposizione come definita prima), cosí come se non si ha confidenza con gli strumenti suggeriti.

Sistema di versionamento

Si parla di file di configurazione, quindi é utile tenere traccia dei cambiamenti dei file invetory e playbook con dei commit che descrivano il cambiamento e la motivazione di quest'ultimo.

Variabili contenenti dati sensibili e non

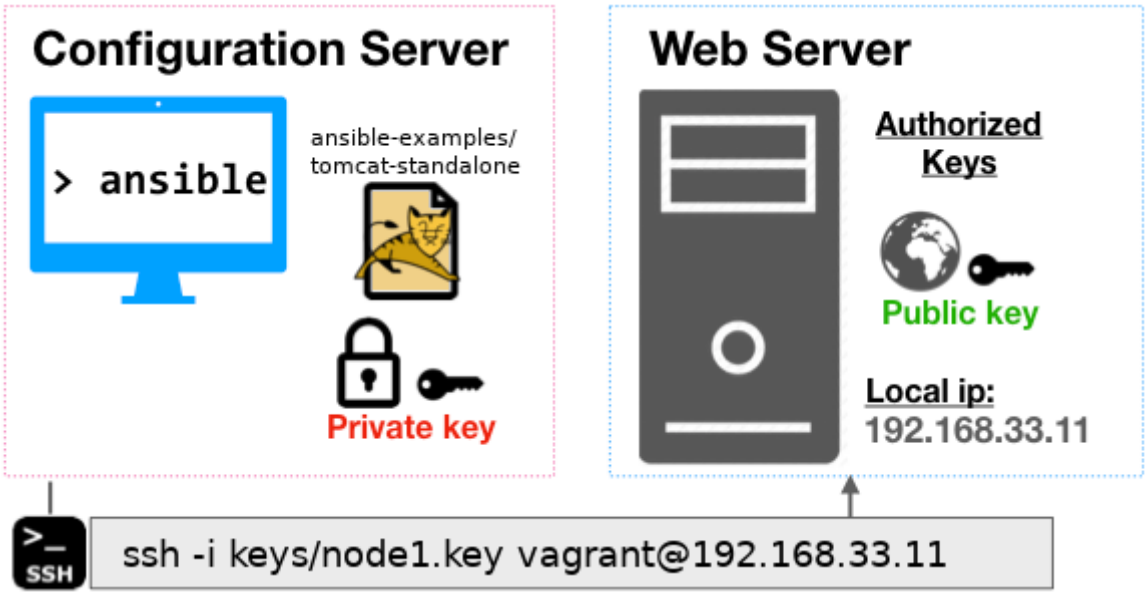
Come comportamento di default, durante la manutenzione, alcune variabili non vengono mostrate perchè contenenti dati sensibili (`vault`), quindi é utile inserire dentro la cartella `group_vars/` tutte le variabili necessarie (sensibili e non) e lasciare nel file avente prefisso `vault_` tutte quelle sensibili. Non ci resta che utilizzare jinja2 (non visto in questo laboratorio) per far puntare le variabili sensibili nel file `var` alle relative variabili sensibili del file `vault_`.

Link approfondimento

Lista completa dei moduli: https://docs.ansible.com/ansible/latest/modules/modules_by_category.html

Esempio di esecuzione di un playbook

l'obbiettivo dell'esercitazione è quello di installare il programma apache tomcat in un nodo con sistema operativo Centos 7.



Creazione nodo centos

Dal sistema host (il nostro pc) creare una cartella (p.es CM-NODE-CENTOS)

```
mkdir CM-NODE-CENTOS
```

Inizializziamo una nuova macchina virtuale nella cartella appena creata

```
cd CM-NODE-CENTOS
vagrant init centos/7
```

Modifichiamo il Vagrantfile presente nella cartella "CM-NODE-CENTOS" in modo da specificare che il nodo censtos sarà raggiungibile all'ip 192.168.33.11

```
config.vm.network "private_network", ip: "192.168.33.11"
```

Avviamo la macchina virtuale appena configurata

```
vagrant up
```

Recuperiamo il path della chiave privata che permette a vagrant di connettersi alla macchina virtuale:

```
vagrant ssh-config
```

Copiare il contenuto del file indicato dalla proprietà `IdentityFile`

Configurazione Configuration Server

Avviare e accedere al configuration Server (server dov'è installato ansible)

```
cd [PATH-CM-SERVER]
vagrant reload
vagrant up
vagrant ssh
```

Aggiungere la chiave privata del nodo centos nel file /home/vagrant/keys/node1.key

Clonare il progetto contenente il playbook di esempio per installare tomcat e accedere alla cartella contenente i playbooks per installare tomcat

```
git clone https://github.com/nicolabertazzo/ansible-examples.git
cd ansible-examples/tomcat-standalone
```

Modificare il file di inventory andando a specificare i riferimenti del nodo centos

```
vim hosts
```

Inserendo il contenuto:

```
[tomcat-servers]
node1 ansible_ssh_host=192.168.33.11 ansible_ssh_user=vagrant
ansible_ssh_private_key_file=/home/vagrant/keys/node1.key
```

Avviare l'esecuzione del playbook site.yml

```
ansible-playbook -i hosts site.yml
```

Ansible eseguirà il playbook site.yml nei nodi specificati nel file di inventory hosts.

- Nel playbook è specificato di eseguire i ruoli:
  - selinux
  - tomcat
- Nel gruppo `tomcat-servers`

- Nel file di inventory è stato specificato che il nodo con ip `192.168.33.11` appartiene al gruppo `tomcat-servers`
- Nella cartella roles sono presenti due cartelle che rappresentano i due ruoli utilizzati
- Ogni cartella roles ha una cartella tasks dove sono specificati i passi del ruolo (file main.yml)
- il file tasks/main.yml utilizza file e variabili presenti e specificate nelle cartelle presenti nel role e nel progetto (vedi copy, template e la variabile http_port su tomcat)