Introduzione ad Ansible

Simone Piccardi

Truelite Srl

3 ottobre 2023

Cos'è Ansible

- un sistema di comunicazioni istantanee basato sulla teoria temporale di Shevek che ha permesso di creare l'Ekumene
- usato da Ender per controllare le navi spaziali della flotta terrestre nella guerra contro gli scorpioni
- ...
- ehmm, mi sa che queste funzionalità non le hanno ancora implementate

Cos'è Ansible ... oggi

Per ora è soltanto un sistema *configuration management*:

- permette di eseguire le operazioni consentono di portare un sistema in uno stato voluto
- consente di gestire in maniera centralizzata insiemi di macchine e di servizi diversi
- usato come alternativa a sistemi come puppet, chef, ecc.

Per cosa si può usare ?

Per cose molto più *Terra-Terra* come:

- gestione delle configurazioni
- automazione di operazioni
- deploy di applicativi
- orchestrazione e provisioning

Principali caratteristiche/1

- prerequisiti minimali sui sistemi gestiti, richiede solo una connessione SSH e python, non serve nessun agent
- consente di gestire tutto da una singola macchina, usando SSH per effettuare sui sistemi remoti le operazioni volute eseguendovi opportuni script in python (i *moduli*)
- usa YAML come linguaggio dichiarativo dello stato che si vuole ottenere, indicando i passi necessari a raggiungerlo; leggibile e auto-documentante

Principali caratteristiche/2

- architettura *push*: le operazioni vengono effettuate solo quando richieste esplicitamente sulla macchina da cui lo usa, (ma supporta anche una modalità *pull*)
- *idempotente*: le operazioni devono essere eseguite o fallire, ripeterne l'esecuzione non deve cambiare nulla nel sistema remoto (gran parte dei *moduli* lo garantiscono)

Pro

I vantaggi:

- prerequisiti minimali
- linguaggio dichiarativo e leggibile
- "facile" da usare e da estendere
- flessibile ed estremamente potente

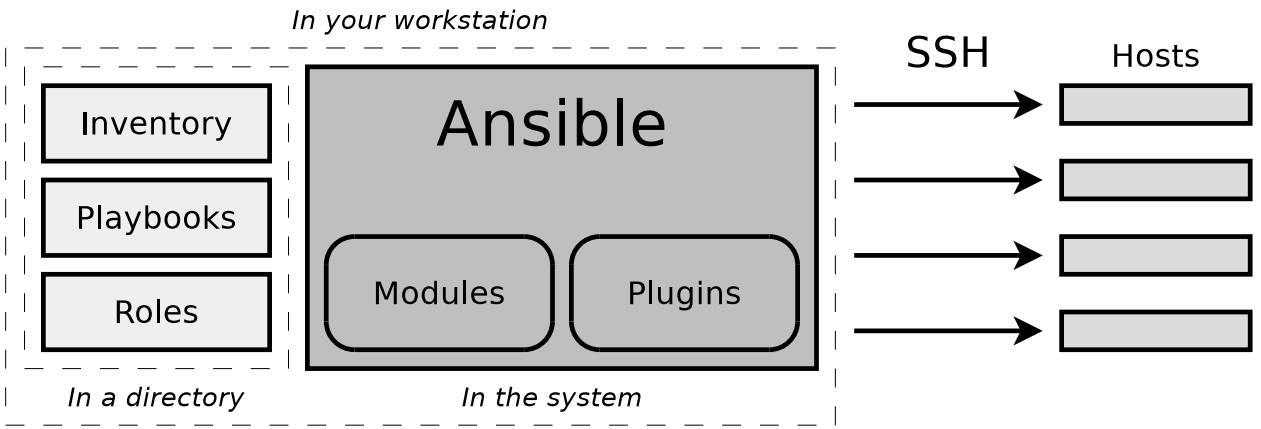
Contro

Gli svantaggi:

- è un pochino lento ...
- beh, diciamo che il pochino si può anche omettere
- ...
- però con mitogen migliora parecchio

Schema panoramico

Uno schema semplificato dell'architettura di funzionamento di Ansible:



L'architettura di funzionamento

- i server remoti su cui si andrà ad operare vengono inseriti come *host* in un inventario (*inventory*), identificati con il nome che verrà usato per la connessione SSH
- le operazioni da eseguire vengono elencate in un *playbook* come sequenza di *task*, cui in genere corrisponde l'esecuzione di un *modulo*

L'architettura di funzionamento/2

- in parallelo ogni *modulo* indicato da un *task* viene copiato, eseguito e poi rimosso su ogni *host* a cui si applica il *playbook*
- i *task* vengono in eseguiti in sequenza e si passa al successivo solo quando il precedente è stato completato su tutti gli *host* a cui si è applicato il *playbook*

L'architettura di funzionamento/3

- Ansible deve essere installato solo sulla macchina da cui si intende operare (in genere la propria workstation)
- tutti i file (*playbook*, *inventari*, ecc.) si mantengono in una directory all'interno della quale si lancia il programma
- è buona pratica tenere questa directory sotto controllo di versione

Nomenclatura

- *modules*: i programmi usati da Ansible per eseguire operazioni specifiche sulle macchine remote, come copiare o creare file, gestire utenti, pacchetti, servizi, ecc. ecc. ecc.
- *plugin*: estensioni di funzionalità che consentono elaborazioni di vario tipo (operazioni sulle variabili, diversi metodi di connessione, lettura di dati da file di vari formati, ecc.)
- *task*: le unità di esecuzione di un modulo, richiedono l'indicazione dello stesso e dei relativi parametri e consentono esecuzioni condizionali e cicli, registrazioni di dati, ecc.
- *handlers*: dei *task* speciali eseguiti al completamento di un *playbook* una sola volta e solo se richiesti (in genere usati per riavviare servizi in caso di cambi di configurazione)

Nomenclatura/2

- *host*: identificativo di un server remoto, in genere si usa il suo FQDN, ma si possono usare nomi qualunque, sono i principali componenti di un inventario
- *variables*: variabili usate per parametrizzare valori, possono essere definite negli inventari, nei playbook, sulla riga di comando, nei ruoli e vengono usate normalmente nei parametri dei moduli e nella definizione di altre variabili
- *facts*: un insieme di variabili attinenti le caratteristiche dell'*host* su cui si andrà ad operare, raccolte automaticamente all'inizio dell'esecuzione di un *playbook* dal modulo `setup`

Nomenclatura/3

- *inventory*: file in cui si elencano gli *host* su cui si vuole operare, si possono definire gruppi di *host* e impostare variabili associate sia a singoli *host* che a gruppi
- *playbook*: un file YAML che contiene un elenco di *task* da eseguire in parallelo su un gruppo di *host*
- *role*: un insieme di *variabili*, *task*, *handlers*, ecc. per eseguire una operazione che può essere riusato dai *playbook*

Ansible e YAML

YAML viene usato da Ansible per la gran parte della sua infrastruttura:

- è uno dei possibili formati in cui scrivere gli inventari
- è il formato in cui si scrivono il playbook
- è il formato in cui si scrivono i ruoli

Cos'è YAML

Acronimo di *YAML Ain't Markup Language*:

- un linguaggio che consente di serializzare delle strutture di dati
- una rappresentazione alternativa per il JSON
- semplice e leggibile (il contrario ad esempio di XML)

Sintassi di YAML

La struttura prevede:

- tutto quello che segue un # viene ignorato
- il contenuto inizia con ---
- consente di indicare stringhe, numeri, liste e dizionari
- usa l'indentazione per strutturare i contenuti

Sintassi di YAML/2

I dati elementari:

- i valori numerici si indicano direttamente, ad esempio: `10.1`, `25000`, ...
- lo stesso vale per le stringhe, ad esempio: `prima stringa`, `seconda stringa`, ...
- ma si dichiarano esplicitamente con `'` e `"`, ad esempio: `"terza stringa"`, `"120000"`, `'prova'`, ...

Sintassi di YAML/3

Le liste sono sequenza ordinate di dati:

- si possono esprimere in forma compatta come:
  ```
  [ 10.5, "pincopallino", prova ]
  ```
- o in forma estesa come:
  ```
  - 10.5
  - "pincopallino"
  - prova
  ```

Sintassi di YAML/4

I dizionari sono coppie chiave/valore:

- si possono esprimere in forma compatta come:
  ```
  {num: 100, str: "pippo", list: [10, "abc", 1.4]}
  ```
- o in forma estesa come:
  ```
  num: 100
  str: pippo
  list: [10, "abc", 1.4]
  ```

Sintassi di YAML/5

Si possono indicare strutture complesse con l'indentazione:

```
num: 100
str: pippo
list:
  - 10
  - "abc"
dict:
  key: val
  abc: d
```

equivalente al dizionario:

```
{num:10, str: pippo, list: [10, "abc"], dict: { key: val, abc: d}}
```

Ancora su YAML

Alla fine è più semplice a scriverlo che a spiegarlo, ma occorre fare attenzione:

- l'indentazione indica la struttura, uno spazio cambia tutto
- l'uso di tabulazioni è **vietato**, si devono usare solo spazi
- si ricordi di proteggere le stringhe che iniziano con {

La configurazione

Il comportamento del programma viene controllato da un file di configurazione che consente di impostare parametri come il file da usare come inventario di default, le directory locali dove cercare ruoli, plugin ecc.

Il file usa il formato INI, la sezione `[default]` è la principale, altre consentono di impostare aspetti meno rilevanti come la colorazione dei messaggi, le modalità di collegamento agli host, ecc.

La configurazione/2

In ordine di priorità vengono consultati i file seguenti, ed usato il primo che viene trovato:

- il file indicato dalla variabile d'ambiente `ANSIBLE_CONFIG`
- il file `ansible.cfg` nella directory corrente*
- il file `/.ansible.cfg`
- il file `/etc/ansible/ansible.cfg`

* in genere si usa questo

Un esempio di configurazione

Un esempio del contenuto di `ansible.cfg`:

```
[defaults]
inventory=hosts
pipelining = True
nocows = 1
roles_path = roles
filter_plugins = filter_plugins
strategy = mitogen_linear
```

Prerequisiti di funzionamento

In sostanza uno solo, avere accesso SSH agli *host*, ma:

- il default assume l'autenticazione a chiavi, l'uso di una password deve essere richiesto esplicitamente con l'opzione `-k`
- il default esegue il collegamento con l'username corrente, in genere serve indicare l'utente remoto (con `-u` o nel playbook)
- se non si usa *root* deve essere installato `sudo` e ne va richiesto l'uso (con `-b` o nel playbook)

Ancora sui prerequisiti

Ansible richiede **sempre** un inventario di *host* su cui lavorare, il default è usare nell'ordine:

- quanto indicato come argomento dell'opzione `-i`
- quanto indicato nel file di configurazione alla voce `inventory`
- il file `/etc/ansible/hosts`

Gli inventari

Si ma in pratica, cos'è questo inventario?

- nella sua forma più comune è un file INI con l'elenco degli *host* uno per riga
- eventuali sezioni consentono di definire un gruppo elencandovi gli *host* che ne fan parte
- si possono associare variabili ad un *host* scrivendole di seguito separate da spazi nella forma `variabile=valore`

Un esempio

Un esempio di inventario con due gruppi:

```
anarres.truelite.it
[sede]
proxmox1.truelite.lan
gw.truelite.lan
[housing]
proxmox2.truelite.it
proxmox3.truelite.it
www.truelite.it
mail.truelite.it
```

Gli inventari/2

Ci sono comunque diverse diverse alternative ...

- si può richiedere l'uso di un plugin per ottenerlo, da indicare in un file `.yml`
- si può usare un programma che lo generi come *output* indicando un file eseguibile
- si può usare una directory e saranno usate tutte le fonti in essa contenute

Gli inventari/2

L'uso di una directory è il più flessibile:

- vi si possono mettere più inventari e saranno accorpati
- si possono definire nelle sottodirectory `host_vars` e `group_vars` variabili per host e i gruppi
- fra gli inventari usabili ci sono altre sottodirectory

In sostanza è un buon modo per documentare la propria infrastruttura.

Gli inventari/3

Ma io voglio lavorare solo su quella macchina ... che me ne faccio di tutto questo ?

- magari ci documenti la tua infrastruttura ...
- si può sempre passare a `-i` una semplice lista di *host* separata da virgole
- per cui sulla singola macchina puoi comunque operarci con: `-i my.host.name.me,*`

* si la virgola ci vuole, dev'essere una lista!

Come si usa Ansible

I due comandi principali forniti da Ansible sono:

- `ansible`, consente di eseguire un singolo *modulo* nella cosiddetta modalità *ad-hoc*, la sintassi è:
  `ansible -u user pattern -i invent -m mod -a params`
- `ansible-playbook`, esegue un *playbook*, la sintassi è:
  `ansible-playbook playbook -i invent -e var=val`

La modalità *ad-hoc*

Utile per operazioni semplici, specie se da applicare ad un insieme di macchine, per il comando `ansible`:

- sono obbligatori un inventario (con `-i`) ed un *pattern* (`all` per tutti) per indicare su quali *host* operare
- si deve indicare con `-m` quale modulo usare, se non si specifica nulla viene usato `command`
- se il modulo richiede argomenti van specificati con `-a`

La modalità *ad-hoc*

Qualche esempio

L'uso di un *playbook*

Un *playbook* consente di eseguire sequenze di operazioni su un gruppo di macchine:

- è sempre obbligatorio avere un inventario a cui applicarlo
- lo si può trasformare in uno script eseguibile direttamente iniziando con:
  `#!/usr/bin/env ansible-playbook`

Un esempio elementare di *playbook*

Un *playbook* elementare per installare il pacchetto ncdu:

```
#!/usr/bin/env ansible-playbook
---
name: An example play on a playbook
hosts: all
vars:
  package: ncdu
tasks:
- name install package {{package}}
  apt:
    name: '{{package}}'
    state: present
```

La struttura di un *playbook*

Un playbook è un file YAML che esprime una lista di "*plays*", ciascuno dei quali è un dizionario che contiene le voci:

- `name` con una sua descrizione testuale
- `hosts` col *pattern* degli *host* cui si applica
- `vars` con un dizionario di definizioni di variabili
- `tasks` una lista di *task* da eseguire
- `handlers` una lista di azioni di finalizzazione
- e molto, molto altro ...

Le variabili

All'interno della voce `vars` si definiscono le variabili:

- si indicano strutturate come un dizionario
- ogni chiave del dizionario indica il nome di una variabile
- si può metter tutto in dei file e leggerli con `vars_files`
- si ottiene il valore dalla variabile con `{{nomevar}}`*

*quindi va protetta adeguatamente in caso di ambiguità nel riuso

Un esempio di variabili

Come definire un file di configurazioni personalizzate per PHP:

```
vars:
  ...
  php_manager: fpm   # mettere apache2 se si usa mod_php
  php_ver:
    bullseye: 7.4
    buster: 7.3
    stretch: 7.0
  php_dir: /etc/php/{{php_ver[ansible_distribution_release]}}
  php_conf: "{{php_dir}}/{{php_manager}}/conf.d/99-local.ini"
  ...
```

I *task*

Ciascun *task* di un *playbook* è un dizionario che contiene le voci:

- `name` con una descrizione (opzionale)
- il nome del *modulo* che sarà eseguito con un dizionario dei valori dei parametri relativi parametri
- ed altre voci per il controllo delle operazioni (condizionali, cicli, registrazione di risultati, notifiche)

vengono eseguiti nell'ordine in cui sono elencati nella sezione `tasks`

I *task*/2

Oltre a un modulo da eseguire un *task* può indicare come voce:

- `when` con una condizione di esecuzione
- `loop` con una lista di valori su cui eseguire un ciclo
- `register` per registrare l'esito in una variabile
- `notify` per notificare un `handler`
- ed altro ancora ...

Un esempio di *task*

Un *task* che configura SSH per l'accesso a root:

```
- name: Riconfigura SSH per accesso generico a root
  copy:
    dest: '/etc/ssh/sshd_config.d/rootaccess.conf'
    content: |
      PermitRootLogin yes
  when: ansible_distribution_major_version|int >= 11
  notify:
  - reload ssh
```

Gli *handler*

Sono *task* di finalizzazione eseguiti dopo tutti quelli ordinari:

- devono essere richiesti da un *task* ordinario con `notify` *`handler name`*
- vengono eseguiti una sola volta anche se notificati più volte
- sono eseguiti nell'ordine in cui sono elencati nella sezione `handlers`
- sono identificati dal nome indicato dalla voce `name`
- o da quello della voce `listen` usabile per raggrupparli

Un esempio per la sezione handlers

Definizione di alcuni *handler* per servizi comuni:

```
handlers:
  - name: restart ssh
    service:
      name: ssh
      state: reloaded
    listen: "reload all"
  - name: reload apache
    service:
      name: apache2
      state: reloaded
    listen: "reload all"
```

Il moduli di Ansible

La potenza e flessibilità di Ansible deriva dai moduli disponibili:

- ne esiste una quantità impressionante: `https://docs.ansible.com/ansible/2.9/modules/modules_by_category.html`
- possono eseguire le operazioni più varie
- normalmente assicurano l'idempotenza dell'operazione

Il modulo `file`

Gestisce le proprietà di un file, indicato con la voce `path`:

- imposta proprietario (voce `user`) e gruppo proprietario (voce `group`
- imposta i permessi (voce `mode`)
- crea o rimuove directory, link simbolici e file (voce `state`)

Un esempio del modulo `file`

Crea una directory con permessi e proprietari adeguati:

```
- name: crea la directory per vacation
  file:
    path: "{{vacation_dir}}"
    state: directory
    mode: 0750
    owner: root
    group: vacation
```

Il modulo copy

Copia un file a su un host remoto:

- indica la destinazione con la voce `dest`
- indica il file da copiare con la voce `src`
- alternativamente copia quanto indicato con `content`
- imposta ownership e permessi come `file`
- ecc. ecc.

Un esempio del modulo copy

Installa il file `vacation.pl` dagli esempi di Postfixadmin:

```
- name: Installa lo script di vacation dagli esempi di Postfixadmin
  copy:
    src:  "{{vacation_source}}"
    dest: "{{vacation_script}}"
    remote_src: yes
    group: vacation
    mode: 0750
    force: no
```

Il modulo `template`

Crea un file a su un host remoto a partire da un template:

- indica la destinazione con la voce `dest`
- indica il file del template con `src`
- imposta ownership e permessi come `file`
- ecc. ecc.

Un esempio di template

I template sono file processati usando Jinja2, un contenuto possibile:

```
$db_type = '{{pfadm_dbtype}}';
$db_username = '{{pfadm_dbuser}}';
$db_password = '{{pfadm_dbpass}}';
$db_name = '{{pfadm_dbname}}';
$vacation_domain = '{{vacation_domain}}';
{% if vacation_logging is defined %}
$logfile = '{{vacation_log_file}}';
$log_to_file = 1;
{% endif %}
```

Un esempio del modulo `template`

Installa una configurazione processando un template:

```
- name: Installa il file di configuratione per lo script di vacation
  template:
    dest: /etc/postfixadmin/vacation.conf
    src: vacation.conf
    mode: 0640
    group: vacation
  when: pfadm_dbpass is defined
```

Il modulo `blockinfile`

Inserisce un blocco di testo in un file indicato con `dest`:

- indica il blocco di testo da inserire con `block`
- indica un marcatore di delimitazione con `marker`
- indica dove metterlo con `insertafter` e `insertbefore`

Un esempio del modulo `blockinfile`

Configura il `master.cf` di Postfix per vacation:

```
blockinfile:
  dest: /etc/postfix/master.cf
  insertafter: EOF
  marker: "# {mark} ANSIBLE MANAGED BLOCK vacation transport"
  block: |
    # for vacation.pl from Postfixadmin
    vacation    unix  -       n       n       -       -       pipe
      flags=Rq user=vacation argv={{vacation_script}} -f ${sender} -- ${recipient}
notify:
- restart postfix
```

Il modulo `lineinfile`

Modifica una riga in un file indicato con `path`:

- indica la riga da inserire con `line`
- permette di rimuovere/sostituire le righe corrispondenti a `regexp`
- indica dove operare con `insertafter` e `insertbefore`

Un esempio del modulo lineinfile

Imposta configurazioni in /etc/postfixadmin/config.local.php:

```
- name: Imposta configurazioni generiche per Postfixadmin
  lineinfile:
    path: /etc/postfixadmin/config.local.php
    regexp: "[$]CONF[[]'{{item.key}}']"
    line: "$CONF['{{item.key}}'] = '{{item.value}}';"
  with_items:
  - { key: 'default_language', value: 'it' }
  - { key: 'min_password_length', value: '8' }
  - { key: 'vacation', value: 'YES' }
  - { key: 'vacation_domain', value: '{{vacation_domain}}' }
  - { key: 'footer_text', value: '{{footer_text}}' }
  - { key: 'footer_link', value: '{{footer_link}}' }
```

Il modulo apt

Gestisce i pacchetti con APT:

- indica i pacchetti su cui operare con `name`
- indica che fare coi pacchetti con `state`
- effettua altre operazioni generiche come `clean`, `update`, `upgrade`, ecc.

Un esempio del modulo apt

Installa una serie di pacchetti:

```
- name: Installa una serie di dipendenze
  apt:
    name: "{{item}}"
    state: present
  loop:
    - sudo,default-mysql-server,python3-pymysql
    - apache2,php-fpm
    - php-gd,php-mysql,php-curl,php-mbstring,php-imagick,php-zip,php-xml
```

Il modulo user

Gestisce gli utenti di sistema:

- indica l'utente su cui operare con `name`
- indica se deve esser presente o meno con `state`
- ne imposta le proprietà con `shell`, `uid`, `groups`, `home`, `password`*, ecc.

* la password va fornita già opportunamente cifrata

Un esempio del modulo user

Crea un utente vacation senza home e capacità di login:

```
- name: Crea un utente per vacation
  user:
    name: vacation
    uid: 65501
    group: vacation
    home: /nonexistent
    shell: /sbin/nologin
    createhome: no
```

Il modulo `command`

Esegue un programma sull'host remoto, ha una sintassi peculiare:

- il comando da eseguire si indica direttamente come valore del modulo
- esegue un programma, non una riga di shell
- allo stesso livello la voce `args` è un dizionario con le voci:
  - `chdir`: esegue il comando nella directory indicata
  - `creates`: indica un file che se c'è non esegue il comando
  - `removes`: il rovescio di `creates`

Un esempio del modulo command

Attiva la configurazione di PHP-FPM per Apache

```
- name: Abilita la configurazione di PHP-FPM per Apache
  command: /usr/sbin/a2enconf {{php_fpm_conf}}
  args:
    creates: /etc/apache2/conf-enabled/{{php_fpm_conf}}.conf
  notify:
  - reload apache
```

Il modulo `service`

Gestisce un servizio in modo generico (sia per SysV che Systemd):

- indica il servizio con la voce `name`
- indica se deve essere attivato all'avvio con `enable`
- indica lo stato voluto, con `state` che può essere, `started`, `stopped`, `restarted`, `reloaded`