Learning Ansible with Rocky (Italian version)

A book from the Documentation Team

Version : 2025/09/28

Rocky Documentation Team

Table of contents

1. Licence

RockyLinux offers Linux courseware for trainers or people wishing to learn how to administer a Linux system on their own.

RockyLinux materials are published under Creative Commons-BY-SA. This means you are free to share and transform the material, while respecting the author's rights.

BY : **Attribution**. You must cite the name of the original author.

SA : **Share Alike**.

- Creative Commons-BY-SA licence : https://creativecommons.org/licenses/by-sa/4.0/

The documents and their sources are freely downloadable from:

- https://docs.rockylinux.org
- https://github.com/rocky-linux/documentation

Our media sources are hosted at github.com. You'll find the source code repository where the version of this document was created.

From these sources, you can generate your own personalized training material using mkdocs. You will find instructions for generating your document here.

How can I contribute to the documentation project?

You'll find all the information you need to join us on our git project home page.

We wish you all a pleasant reading and hope you enjoy the content.

2. Imparare Ansible con Rocky

Ansible è un semplice, ma potente, motore di automazione per Linux. Questo tutorial ti guiderà attraverso i concetti dell'uso di Ansible per automatizzare le tue attività IT in un modo che sia (si spera) divertente e istruttivo. L'utilizzo degli esercizi in questi capitoli ti aiuterà a ottenere un buon livello di comfort con Ansible nelle applicazioni del mondo reale.

3. Nozioni di base su Ansible

In questo capitolo imparerai come lavorare con Ansible.

Obiettivi: In questo capitolo imparerai come:

✔ Implementare Ansible;
✔ Applicare modifiche alla configurazione su un server;
✔ Creare i primi playbook Ansible;

🏁 **ansible**, **moduli**, **playbook**

Conoscenza: ★ ★ ★
Complessità: ★ ★

Tempo di lettura: 30 minuti

Ansible centralizza e automatizza i compiti di amministrazione. È:

- **agentless** (non richiede implementazioni specifiche sui client),
- **idempotent** (stesso effetto ogni volta che viene eseguito).

Utilizza il protocollo **SSH** per configurare in remoto i client Linux o il protocollo **WinRM** per funzionare con i client Windows. Se nessuno di questi protocolli è disponibile, è sempre possibile per Ansible utilizzare un'API, che rende Ansible un vero coltellino svizzero per la configurazione di server, postazioni di lavoro, servizi docker, attrezzature di rete, ecc. (Quasi tutto in realtà).

> ⚠ **Attenzione**
>
> L'apertura di flussi SSH o WinRM a tutti i client dal server Ansible lo rende un elemento critico dell'architettura che deve essere attentamente monitorato.

Poiché Ansible è principalmente basato su push, non manterrà lo stato dei suoi server tra le sue esecuzioni. Al contrario, eseguirà nuovi controlli di stato ogni volta che viene eseguito. Si dice che sia senza stato.

Ti aiuterà con:

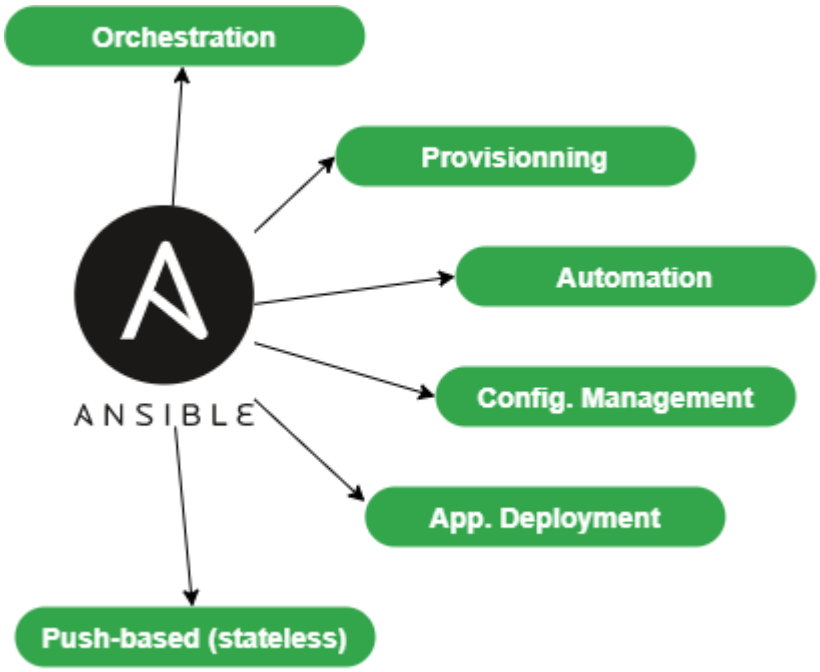
- fornitura (distribuzione di una nuova VM),
- distribuzione di applicazioni
- gestione della configurazione,
- automazione
- orchestrazione (quando è in uso più di 1 destinazione).

Nota

Ansible è stato originariamente scritto da Michael DeHaan, il fondatore di altri strumenti come Cobbler.

La prima versione è stata la 0.0.1, rilasciata il 9 marzo 2012.

Il 17 ottobre 2015, AnsibleWorks (la società dietro Ansible) è stata acquisita da Red Hat per 150 milioni di dollari.



Per offrire un'interfaccia grafica al tuo uso quotidiano di Ansible, puoi installare alcuni strumenti come Ansible Tower (RedHat), che non è gratuito, la sua controparte opensource Awx, o possono anche essere utilizzati altri progetti come Jenkins e l'eccellente Rundeck.

In astratto

Per seguire questa formazione, avrai bisogno di almeno 2 server con Rocky8:

- il primo sarà la **macchina di gestione**, Ansible sarà installato su di esso.
- il secondo sarà il server da configurare e gestire (un altro Linux diverso da Rocky Linux andrà altrettanto bene).

Negli esempi seguenti, la stazione di amministrazione ha l'indirizzo IP 172.16.1.10, la stazione gestita 172.16.1.11. Sta a voi adattare gli esempi in base al vostro piano di indirizzamento IP.

3.1 Il vocabolario Ansible

- La **macchina di gestione**: la macchina su cui è installato Ansible. Dal momento che Ansible è **agentless**, nessun software viene distribuito sui server gestiti.
- I **nodi gestiti**: i dispositivi di destinazione che Ansible gestisce sono anche chiamati "host" Possono essere server, dispositivi di rete o qualsiasi altro computer.
- L'**inventory**: un file contenente informazioni sui server gestiti.
- I **tasks**: un task è un blocco che definisce una procedura da eseguire (ad esempio, creare un utente o un gruppo, installare un pacchetto software, ecc.).

- Un **module**: un modulo astrae un compito. Esistono molti moduli forniti da Ansible.
- I **playbook**: un semplice file in formato yaml che definisce i server di destinazione e i compiti da eseguire.
- Un **role**: un role consente di organizzare i playbook e tutti gli altri file necessari (modelli, script, ecc.) per facilitare la condivisione e il riutilizzo del codice.
- Una **collection**: una collezione comprende un insieme logico di playbook, ruoli, moduli e plugin.
- I **facts**: sono variabili globali che contengono informazioni sul sistema (nome della macchina, versione del sistema, interfaccia di rete e configurazione, ecc.).
- Gli **handler**: sono utilizzati per causare l'arresto o il riavvio di un servizio in caso di modifica.

3.2 Installazione sul server di gestione

Ansible è disponibile nel repository *EPEL*, ma a volte può essere datato per la versione corrente e si desidera lavorare con una versione più recente.

Prenderemo quindi in considerazione due tipi di installazione:

- quella basata sui repository EPEL
- una basata sul gestore di pacchetti python `pip`

L'*EPEL* è necessario per entrambe le versioni, quindi potete procedere all'installazione:

- Installazione di EPEL:

```
sudo dnf install epel-release
```

3.2.1 Installazione da EPEL

Se installiamo Ansible da *EPEL*, possiamo fare quanto segue:

```
sudo dnf install ansible
```

Successivamente, verificare l'installazione:

```
$ ansible --version
ansible [core 2.14.2]
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/home/rocky/.ansible/plugins/modules', '/
usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3.11/site-packages/ansible
ansible collection location = /home/rocky/.ansible/collections:/usr/share/
ansible/collections
  executable location = /usr/bin/ansible
  python version = 3.11.2 (main, Jun 22 2023, 04:35:24) [GCC 8.5.0 20210514
(Red Hat 8.5.0-18)] (/usr/bin/python3.11)
  jinja version = 3.1.2
  libyaml = True

$ python3 --version
Python 3.6.8
```

Si noti che ansible viene fornito con una propria versione di python, diversa da quella di sistema (qui 3.11.2 contro 3.6.8). È necessario considerarlo quando si installano con pip i moduli python necessari per l'installazione (ad esempio `pip3.11 install PyVMomi`).

3.2.2 Installazione da python pip

Poiché vogliamo usare una versione più recente di Ansible, la installeremo da `python3-pip`:

> **Nota**
>
> Rimuovere Ansible se è stato installato in precedenza da *EPEL*.

In questa fase, possiamo scegliere di installare ansible con la versione di python che desideriamo.

```
sudo dnf install python38 python38-pip python38-wheel python3-argcomplete rust
cargo curl
```

> **Nota**
>
> `python3-argcomplete' è fornito da *EPEL*. Installare epel-release se non è ancora stato fatto. Questo pacchetto vi aiuterà a completare i comandi di Ansible.

Ora possiamo installare Ansible:

```
pip3.8 install --user ansible
activate-global-python-argcomplete --user
```

Controllare la versione di Ansible:

```
$ ansible --version
ansible [core 2.13.11]
  config file = None
  configured module search path = ['/home/rocky/.ansible/plugins/modules', '/
usr/share/ansible/plugins/modules']
  ansible python module location = /home/rocky/.local/lib/python3.8/site-
packages/ansible
  ansible collection location = /home/rocky/.ansible/collections:/usr/share/
ansible/collections
  executable location = /home/rocky/.local/bin/ansible
  python version = 3.8.16 (default, Jun 25 2023, 05:53:51) [GCC 8.5.0 20210514
(Red Hat 8.5.0-18)]
  jinja version = 3.1.2
  libyaml = True
```

> **Nota**
>
> La versione installata manualmente nel nostro caso è più vecchia della versione pacchettizzata da RPM perché abbiamo usato una versione precedente di python. Questa osservazione varia con il tempo, l'età della distribuzione e la versione di python.

3.3 File di configurazione

La configurazione del server si trova in `/etc/ansible`.

I file di configurazione principali sono due:

- Il file di configurazione principale `ansible.cfg` dove risiedono i comandi, i moduli, i plugin e la configurazione ssh;
- Il file di inventario della gestione dei computer client `hosts` in cui sono dichiarati i client e i gruppi di client.

Il file di configurazione viene creato automaticamente se Ansible è stato installato con il suo pacchetto RPM. Con un'installazione `pip`, questo file non esiste. Dovremo crearlo a mano con il comando `ansible-config`:

```
$ ansible-config -h
usage: ansible-config [-h] [--version] [-v] {list,dump,view,init} ...

Visualizzare la configurazione di ansibile.

argomenti di posizione:
  {list,dump,view,init}
    list Stampa tutte le opzioni di configurazione
    dump Configurazione Dump
    view Visualizza i file di configurazione
    init Crea configurazione iniziale
```

Esempio:

```
ansible-config init --disabled > /etc/ansible/ansible.cfg
```

L'opzione `--disabled` consente di commentare l'insieme delle opzioni anteponendo ad esse il prefisso `;`.

Nota

Si può anche scegliere di incorporare la configurazione di Ansible nel proprio repository di codice, con Ansible che carica i file di configurazione che trova nel seguente ordine (elaborando il primo file che incontra e ignorando gli altri):

- se la variabile d'ambiente `$ANSIBLE_CONFIG` è impostata, carica il file specificato.
- `ansible.cfg` se esiste nella directory corrente.
- `~/.ansible.cfg` se esiste (nella home directory dell'utente).

Se non viene trovato nessuno di questi tre file, viene caricato il file predefinito.

3.3.1 Il file di inventario `/etc/ansible/hosts`

Poiché Ansible dovrà lavorare con tutte le macchine configurate, è essenziale fornirgli uno (o più) file di inventario ben strutturati che corrispondano perfettamente alla vostra organizzazione.

A volte è necessario riflettere attentamente su come costruire questo file.

Andare al file di inventario predefinito, che si trova in `/etc/ansible/hosts`. Vengono forniti alcuni esempi commentati:

```
# This is the default ansible 'hosts' file.
#
# It should live in /etc/ansible/hosts
#
#   - Comments begin with the '#' character
#   - Blank lines are ignored
#   - Groups of hosts are delimited by [header] elements
#   - You can enter hostnames or ip addresses
#   - A hostname/ip can be a member of multiple groups

# Ex 1: Ungrouped hosts, specify before any group headers:

## green.example.com
## blue.example.com
## 192.168.100.1
## 192.168.100.10

# Ex 2: A collection of hosts belonging to the 'webservers' group:

## [webservers]
## alpha.example.org
## beta.example.org
## 192.168.1.100
## 192.168.1.110

# If you have multiple hosts following a pattern, you can specify
# them like this:

## www[001:006].example.com

# Ex 3: A collection of database servers in the 'dbservers' group:

## [dbservers]
##
## db01.intranet.mydomain.net
## db02.intranet.mydomain.net
## 10.25.1.56
## 10.25.1.57

# Here's another example of host ranges, this time there are no
# leading 0s:

## db-[99:101]-node.example.com
```

Come si può notare, il file fornito come esempio utilizza il formato INI, ben noto agli amministratori di sistema. Si noti che è possibile scegliere un altro formato di file (come yaml, per esempio), ma per i primi test il formato INI si adatta bene ai nostri prossimi esempi.

L'inventario può essere generato automaticamente in produzione, soprattutto se si dispone di un ambiente di virtualizzazione come VMware VSphere o di un ambiente cloud (Aws, OpenStack o altro).

- Creare un gruppo di host in `/etc/ansible/hosts`:

Come avrete notato, i gruppi sono dichiarati tra parentesi quadre. Poi vengono gli elementi che appartengono ai gruppi. Si può creare, ad esempio, un gruppo `rocky8` inserendo il seguente blocco nel file:

```
[rocky8]
172.16.1.10
172.16.1.11
```

I gruppi possono essere utilizzati all'interno di altri gruppi. In questo caso, occorre specificare che il gruppo padre è composto da sottogruppi con l'attributo `:children`, come in questo caso:

```
[linux:children]
rocky8
debian9

[ansible:children]
ansible_management
ansible_clients

[ansible_management]
172.16.1.10

[ansible_clients]
172.16.1.10
```

Non ci dilungheremo oltre sull'inventario, ma se siete interessati, valutate la possibilità di consultare questo link.

Ora che il nostro server di gestione è installato e il nostro inventario è pronto, è il momento di eseguire i nostri primi comandi `ansible`.

3.4 utilizzo della riga di comando `ansibile`

Il comando `ansible` lancia un task su uno o più host di destinazione.

```
ansible <host-pattern> [-m module_name] [-a args] [options]
```

Esempi:

> **Attenzione**
>
> Dal momento che non abbiamo ancora configurato l'autenticazione sui nostri 2 server di test, non tutti gli esempi seguenti funzioneranno. Sono forniti come esempi per facilitare la comprensione e saranno completamente funzionanti più avanti in questo capitolo.

- Elenca gli host appartenenti al gruppo rocky8:

```
ansible rocky8 --list-hosts
```

- Eseguire il ping di un gruppo di host con il modulo `ping`:

```
ansible rocky8 -m ping
```

- Visualizzare i fatti di un gruppo di host con il modulo `setup`:

```
ansible rocky8 -m setup
```

- Eseguire un comando su un gruppo di host invocando il modulo `command` con degli argomenti:

```
ansible rocky8 -m command -a 'uptime'
```

- Eseguire un comando con privilegi di amministratore:

```
ansible ansible_clients --become -m command -a 'reboot'
```

- Eseguire un comando utilizzando un file di inventario personalizzato:

```
ansible rocky8 -i ./local-inventory -m command -a 'date'
```

> **Nota**
>
> Come in questo esempio, a volte è più semplice separare la dichiarazione dei dispositivi gestiti in diversi file (ad esempio per progetto cloud) e fornire ad Ansible il percorso di questi file, piuttosto che mantenere un lungo file di inventario.

Opzione	Informazione
`-a 'arguments'`	Gli argomenti da passare al modulo.
`-b -K`	Richiede una password ed esegue il comando con privilegi superiori.
`--user=username`	Utilizza questo utente per connettersi all'host di destinazione invece dell'utente corrente.
`--become-user=username`	Esegue l'operazione come questo utente (predefinito: `root`).
`-C`	Simulazione. Non apporta alcuna modifica all'obiettivo, ma lo prova per vedere cosa dovrebbe essere modificato.
`-m module`	Esegue il modulo chiamato

3.4.1 Preparazione del client

Sia sulla macchina di gestione che sui client, creeremo un utente `ansible` dedicato alle operazioni eseguite da Ansible. Questo utente dovrà utilizzare i diritti di sudo, quindi dovrà essere aggiunto al gruppo `wheel`.

Questo utente servirà:

- Sul lato della stazione di amministrazione: per eseguire comandi `ansible` e SSH ai client gestiti.
- Sulle stazioni gestite (qui il server che funge da stazione di amministrazione funge anche da client, quindi è gestito da solo) per eseguire i comandi lanciati dalla stazione di amministrazione: deve quindi avere i diritti sudo.

Su entrambe le macchine, creare un utente `ansible`, dedicato ad ansible:

```
sudo useradd ansible
sudo usermod -aG wheel ansible
```

Impostare una password per questo utente:

```
sudo passwd ansible
```

Modificare la configurazione di sudoers per consentire ai membri del gruppo `wheel` di eseguire sudo senza password:

```
sudo visudo
```

Il nostro obiettivo è commentare l'opzione predefinita e decommentare l'opzione NOPASSWD, in modo che queste righe abbiano l'aspetto seguente:

```
## Allows people in group wheel to run all commands
# %wheel  ALL=(ALL)       ALL

## Same thing without a password
%wheel        ALL=(ALL)       NOPASSWD: ALL
```

Attenzione

Se si riceve il seguente messaggio di errore quando si inseriscono i comandi di Ansible, probabilmente significa che si è dimenticato questo passaggio su uno dei client: `"msg": " Missing sudo password"`

Quando si utilizza la gestione da questo momento in poi, si inizia a lavorare con questo nuovo utente:

```
sudo su - ansible
```

3.4.2 Verifica con il modulo ping

Per impostazione predefinita, il login con password non è consentito da Ansible.

Togliere il commento alla seguente riga dalla sezione `[defaults]` del file di configurazione `/etc/ansible/ansible.cfg` e impostarla su True:

```
ask_pass      = True
```

Eseguire un `ping` su ogni server del gruppo rocky8:

```
# ansible rocky8 -m ping
SSH password:
172.16.1.10 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
172.16.1.11 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Nota

Viene richiesta la password `ansible` dei server remoti, il che rappresenta un problema di sicurezza...

Suggerimento

Se si ottiene questo errore `"msg": "to use the 'ssh' connection type with passwords, you must install the sshpass program"`, è sufficiente installare `sshpass` sulla stazione di gestione:

```
$ sudo dnf install sshpass
```

In astratto

Ora puoi testare i comandi che non hanno funzionato in precedenza in questo capitolo.

3.5 Autenticazione con chiave

L'autenticazione tramite password sarà sostituita da un'autenticazione a chiave privata/pubblica molto più sicura.

3.5.1 Creazione di una chiave SSH

La doppia chiave sarà generata con il comando `ssh-keygen` sulla stazione di gestione dall'utente `ansible`:

```
[ansible]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ansible/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/ansible/.ssh/id_rsa.
Your public key has been saved in /home/ansible/.ssh/id_rsa.pub.
```

```
The key fingerprint is:
SHA256:Oa1d2hYzzdO0e/K10XPad25TA1nrSVRPIuS4fnmKr9g
ansible@localhost.localdomain
The key's randomart image is:
+---[RSA 3072]----+
|           .o . +|
|           o . =.|
|          . . + +|
|         o . = =.|
|        S o = B.o|
|         = + = =+|
|        . + = o+B|
|         o + o *@|
|        . Eoo .+B|
+----[SHA256]-----+
```

La chiave pubblica può essere copiata sui server:

```
# ssh-copy-id ansible@172.16.1.10
# ssh-copy-id ansible@172.16.1.11
```

Ricommentare la seguente riga dalla sezione `[defaults]` del file di configurazione `/etc/ansible/ansible.cfg` per impedire l'autenticazione tramite password:

```
#ask_pass      = True
```

3.5.2 Test di autenticazione con chiave privata

Per il prossimo test, viene utilizzato il modulo `shell`, che consente l'esecuzione di comandi remoti:

```
# ansible rocky8 -m shell -a "uptime"
172.16.1.10 | SUCCESS | rc=0 >>
 12:36:18 up 57 min,  1 user,  load average: 0.00, 0.00, 0.00

172.16.1.11 | SUCCESS | rc=0 >>
 12:37:07 up 57 min,  1 user,  load average: 0.00, 0.00, 0.00
```

Non è richiesta alcuna password, l'autenticazione a chiave privata/pubblica funziona!

> **Nota**
>
> Nell'ambiente di produzione, si dovrebbero ora rimuovere le password `ansible` precedentemente impostate per rafforzare la sicurezza (poiché ora non è necessaria una password di autenticazione).

3.6 Usare Ansible

Ansible può essere utilizzato dalla shell o tramite playbook.

3.6.1 I moduli

L'elenco dei moduli classificati per categoria può essere trovato qui. Ansible ne offre più di 750!

I moduli sono ora raggruppati in raccolte di moduli, il cui elenco può essere trovato qui.

Le collezioni sono un formato di distribuzione per i contenuti di Ansible che possono includere playbook, ruoli, moduli e plugin.

Un modulo viene invocato con l'opzione `-m` del comando `ansible`:

```
ansible <host-pattern> [-m module_name] [-a args] [options]
```

C'è un modulo per quasi tutte le necessità! Si consiglia quindi, invece di utilizzare il modulo shell, di cercare un modulo adatto alle esigenze.

Ogni categoria di esigenze ha un proprio modulo. Ecco un elenco non esaustivo:

Tipo	Esempi
Gestione Sistema	`user` (gestione utenti), `group` (gestione gruppi), ecc.
Gestione del software	`dnf`, `yum`, `apt`, `pip`, `npm`
Gestione file	`copy`, `fetch`, `lineinfile`, `template`, `archive`
Gestione database	`mysql`, `postgresql`, `redis`
Gestione cloud	`amazon S3`, `cloudstack`, `openstack`
Gestione cluster	`consul`, `zookeeper`
Invia comandi	`shell`, `script`, `expect`
Downloads	`get_url`
Gestione sorgenti	`git`, `gitlab`

Esempio di installazione software

Il modulo `dnf` consente di installare il software sui client di destinazione:

```
# ansible rocky8 --become -m dnf -a name="httpd"
172.16.1.10 | SUCCESS => {
    "changed": true,
    "msg": "",
    "rc": 0,
    "results": [
      ...
      \n\nComplete!\n"
    ]
}
172.16.1.11 | SUCCESS => {
    "changed": true,
    "msg": "",
    "rc": 0,
    "results": [
      ...
    \n\nComplete!\n"
    ]
}
```

Essendo il software installato un servizio, è necessario avviarlo con il modulo `systemd`:

```
# ansible rocky8 --become  -m systemd -a "name=httpd state=started"
172.16.1.10 | SUCCESS => {
    "changed": true,
    "name": "httpd",
    "state": "started"
}
172.16.1.11 | SUCCESS => {
    "changed": true,
    "name": "httpd",
    "state": "started"
}
```

Suggerimento

Provate a lanciare gli ultimi due comandi due volte. Si noterà che la prima volta Ansible intraprenderà delle azioni per raggiungere lo stato impostato dal comando. La seconda volta, non farà nulla perché avrà rilevato che lo stato è già stato raggiunto!

3.6.2 Esercizi

Per scoprire di più su Ansible e per abituarsi a consultare la documentazione di Ansible, ecco alcuni esercizi da fare prima di proseguire:

- Creare i gruppi Parigi, Tokio, NewYork
- Creare l'utente `supervisor`
- Cambiare l'utente per avere un uid di 10000
- Cambia l'utente in modo che appartenga al gruppo Parigi
- Installare il software ad albero
- Ferma il servizio di crond
- Creare un file vuoto con i permessi `0644`
- Aggiorna la distribuzione del client
- Riavvia il tuo client

> **Attenzione**
>
> Non utilizzare il modulo shell. Cercate nella documentazione i moduli appropriati!

modulo `setup`: introduzione ai fatti

I fatti di sistema sono variabili recuperate da Ansible tramite il suo modulo `setup`.

Date un'occhiata ai diversi fatti dei vostri client per avere un'idea della quantità di informazioni che possono essere facilmente recuperate con un semplice comando.

Vedremo più tardi come utilizzare i fatti nei nostri playbook e come creare i nostri fatti.

```
# ansible ansible_clients -m setup | less
192.168.1.11 | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "192.168.1.11"
        ],
        "ansible_all_ipv6_addresses": [
            "2001:861:3dc3:fcf0:a00:27ff:fef7:28be",
            "fe80::a00:27ff:fef7:28be"
```

```
        ],
        "ansible_apparmor": {
            "status": "disabled"
        },
        "ansible_architecture": "x86_64",
        "ansible_bios_date": "12/01/2006",
        "ansible_bios_vendor": "innotek GmbH",
        "ansible_bios_version": "VirtualBox",
        "ansible_board_asset_tag": "NA",
        "ansible_board_name": "VirtualBox",
        "ansible_board_serial": "NA",
        "ansible_board_vendor": "Oracle Corporation",
        ...
```

Ora che abbiamo visto come configurare un server remoto con Ansible dalla riga di comando, saremo in grado di introdurre la nozione di playbook. I playbook sono un altro modo di utilizzare Ansible, non molto più complesso, ma che renderà più facile il riutilizzo del codice.

3.7 Playbooks

I playbook di Ansible descrivono una politica da applicare ai sistemi remoti, per forzarne la configurazione. I playbook sono scritti in un formato di testo facilmente comprensibile che raggruppa un insieme di attività: il formato `yaml`.

> **Nota**
>
> Ulteriori informazioni su yaml qui

```
ansible-playbook <file.yml> ... [options]
```

Le opzioni sono identiche al comando `ansible`.

Il comando restituisce i seguenti codici di errore:

Codice	Errore
`0`	OK o nessun host corrispondente
`1`	Errore
`2`	Uno o più host hanno fallito
`3`	Uno o più host sono irraggiungibili
`4`	Analizzare errore
`5`	Opzioni errate o incomplete
`99`	Esecuzione interrotta dall'utente
`250`	Errore inaspettato

Nota

Si noti che `ansible` restituirà Ok quando nessun host corrisponde al target, il che potrebbe trarre in inganno!

3.7.1 Esempio di playbook Apache e MySQL

Il seguente playbook ci permetterà di installare Apache e MariaDB sui nostri server di destinazione.

Crea un file `test.yml` con il seguente contenuto:

```
---
- hosts: rocky8 <1>
  become: true <2>
  become_user: root

  tasks:

    - name: ensure apache is at the latest version
      dnf: name=httpd,php,php-mysqli state=latest

    - name: ensure httpd is started
      systemd: name=httpd state=started

    - name: ensure mariadb is at the latest version
      dnf: name=mariadb-server state=latest

    - name: ensure mariadb is started
```

```
      systemd: name=mariadb state=started
...
```

- <1> Il gruppo o il server in questione devono esistere nell'inventario
- <2> Una volta connesso, l'utente diventa `root` (tramite `sudo` per impostazione predefinita)

L'esecuzione del playbook avviene con il comando `ansible-playbook`:

```
$ ansible-playbook test.yml

PLAY [rocky8] ******************************************************************

TASK [setup] *******************************************************************
ok: [172.16.1.10]
ok: [172.16.1.11]

TASK [ensure apache is at the latest version] **********************************
ok: [172.16.1.10]
ok: [172.16.1.11]

TASK [ensure httpd is started] *************************************************
changed: [172.16.1.10]
changed: [172.16.1.11]

TASK [ensure mariadb is at the latest version]
**********************************
changed: [172.16.1.10]
changed: [172.16.1.11]

TASK [ensure mariadb is started]
*******************************************
changed: [172.16.1.10]
changed: [172.16.1.11]

PLAY RECAP
*********************************************************************
172.16.1.10              : ok=5    changed=3    unreachable=0    failed=0
172.16.1.11              : ok=5    changed=3    unreachable=0    failed=0
```

Per una maggiore leggibilità, è consigliabile scrivere i playbook in formato yaml completo. Nell'esempio precedente, gli argomenti sono indicati sulla stessa riga del

modulo, con il valore dell'argomento che segue il suo nome separato da un `=`. Guardate lo stesso playbook in yaml completo:

```
---
- hosts: rocky8
  become: true
  become_user: root

  tasks:

    - name: ensure apache is at the latest version
      dnf:
        name: httpd,php,php-mysqli
        state: latest

    - name: ensure httpd is started
      systemd:
        name: httpd
        state: started

    - name: ensure mariadb is at the latest version
      dnf:
        name: mariadb-server
        state: latest

    - name: ensure mariadb is started
      systemd:
        name: mariadb
        state: started
...
```

Suggerimento

`dnf` è uno dei moduli che permette di fornire un elenco come argomento.

Nota sulle collezioni: Ansible ora fornisce moduli sotto forma di collezioni. Alcuni moduli sono forniti di default nella collezione `ansible.builtin`, altri devono essere installati manualmente tramite il:

```
ansible-galaxy collection install [collectionname]
```

dove [collectionname] è il nome dell'insieme (le parentesi quadre servono a evidenziare la necessità di sostituirlo con il nome effettivo dell'insieme e NON fanno parte del comando).

L'esempio precedente dovrebbe essere scritto come segue:

```
---
- hosts: rocky8
  become: true
  become_user: root

  tasks:

    - name: ensure apache is at the latest version
      ansible.builtin.dnf:
        name: httpd,php,php-mysqli
        state: latest

    - name: ensure httpd is started
      ansible.builtin.systemd:
        name: httpd
        state: started

    - name: ensure mariadb is at the latest version
      ansible.builtin.dnf:
        name: mariadb-server
        state: latest

    - name: ensure mariadb is started
      ansible.builtin.systemd:
        name: mariadb
        state: started
...
```

Un playbook non è limitato a un obiettivo:

```
---
- hosts: webservers
  become: true
  become_user: root

  tasks:

    - name: ensure apache is at the latest version
      ansible.builtin.dnf:
        name: httpd,php,php-mysqli
        state: latest

    - name: ensure httpd is started
      ansible.builtin.systemd:
        name: httpd
```

```
        state: started

- hosts: databases
  become: true
  become_user: root

    - name: ensure mariadb is at the latest version
      ansible.builtin.dnf:
        name: mariadb-server
        state: latest

    - name: ensure mariadb is started
      ansible.builtin.systemd:
        name: mariadb
        state: started
...
```

Puoi controllare la sintassi del tuo playbook:

```
ansible-playbook --syntax-check play.yml
```

È anche possibile utilizzare un "linter" per yaml:

```
dnf install -y yamllint
```

quindi controllare la sintassi yaml dei tuoi playbook:

```
$ yamllint test.yml
test.yml
  8:1       error    syntax error: could not find expected ':' (syntax)
```

3.8 Risultati degli esercizi

- Creare i gruppi Parigi, Tokio, NewYork
- Creare l'utente `supervisor`
- Cambiare l'utente per avere un uid di 10000
- Cambiare l'utente in modo che appartenga al gruppo Parigi
- Installare l'albero del software
- Fermare il servizio crond

- Creare un file vuoto con i permessi `0644`
- Aggiornare la distribuzione del client
- Riavviare il tuo client

```
ansible ansible_clients --become -m group -a "name=Paris"
ansible ansible_clients --become -m group -a "name=Tokio"
ansible ansible_clients --become -m group -a "name=NewYork"
ansible ansible_clients --become -m user -a "name=Supervisor"
ansible ansible_clients --become -m user -a "name=Supervisor uid=10000"
ansible ansible_clients --become -m user -a "name=Supervisor uid=10000
groups=Paris"
ansible ansible_clients --become -m dnf -a "name=tree"
ansible ansible_clients --become -m systemd -a "name=crond state=stopped"
ansible ansible_clients --become -m copy -a
"content='' dest=/tmp/test force=no mode=0644"
ansible ansible_clients --become -m dnf -a "name=* state=latest"
ansible ansible_clients --become -m reboot
```

4. Ansibile Intermedio

In questo capitolo si continuerà a imparare a lavorare con Ansible.

Obiettivi: in questo capitolo imparerete come:

✔ lavorare con le variabili;
✔ usare i cicli;
✔ gestire i cambiamenti di stato e reagire a loro;
✔ gestire le attività asincrone.

🏁 **ansible**, **moduli**, **playbook**

Conoscenza: ★★★
Complessità: ★★

Tempo di lettura: 30 minuti

Nel capitolo precedente si è appreso come installare Ansible, utilizzarlo dalla riga di comando e scrivere playbook per promuovere la riutilizzabilità del codice.

In questo capitolo, possiamo iniziare a scoprire nozioni più avanzate su come utilizzare Ansible e alcuni task interessanti che utilizzerete regolarmente.

4.1 Le variabili

> **Nota**
>
> Maggiori informazioni possono essere trovate qui.

Sotto Ansible, ci sono diversi tipi di variabili primitive:

- stringhe,
- interi,
- booleani.

Queste variabili possono essere organizzate come:

- dizionari,
- elenchi.

Una variabile può essere definita in diversi luoghi, come un playbook, un ruolo o la riga di comando.

Per esempio, da un playbook:

```
---
- hosts: apache1
  vars:
    port_http: 80
    service:
      debian: apache2
      rhel: httpd
```

o dalla riga di comando:

```
ansible-playbook deploy-http.yml --extra-vars "service=httpd"
```

Una volta definita, una variabile può essere utilizzata chiamandola tra due parentesi graffe:

- `{{ port_http }}` per un valore semplice,
- `{{ service['rhel'] }}` o `{{ service.rhel }}` per un dizionario.

Per esempio:

```
- name: make sure apache is started
  ansible.builtin.systemd:
    name: "{{ service['rhel'] }}"
    state: started
```

Naturalmente, è anche possibile accedere alle variabili globali (i **fatti**) di Ansible (tipo di OS, indirizzi IP, nome VM, ecc.).

4.1.1 Variabili di esternalizzazione

Le variabili possono essere incluse in un file esterno al playbook, in questo caso questo file deve essere definito nel playbook con la direttiva `vars_files`:

```
---
- hosts: apache1
  vars_files:
    - myvariables.yml
```

Il file `myvariables.yml`:

```
---
port_http: 80
ansible.builtin.systemd::
  debian: apache2
  rhel: httpd
```

Può anche essere aggiunto dinamicamente con l'uso del modulo `include_vars`:

```
- name: Include secrets.
  ansible.builtin.include_vars:
    file: vault.yml
```

4.1.2 Visualizzare una variabile

Per visualizzare una variabile, è necessario attivare il modulo `di debug` come segue:

```
- ansible.builtin.debug:
    var: service['debian']
```

È anche possibile utilizzare la variabile all'interno di un testo:

```
- ansible.builtin.debug:
    msg: "Print a variable in a message : {{ service['debian'] }}"
```

4.1.3 Salva il ritorno di un'attività

Per salvare il risultato di un compito e per essere in grado di accedervi più tardi, devi usare la parola chiave `register` all'interno del compito stesso.

Uso di una variabile memorizzata:

```
- name: /home content
  shell: ls /home
  register: homes

- name: Print the first directory name
  ansible.builtin.debug:
    var: homes.stdout_lines[0]

- name: Print the first directory name
  ansible.builtin.debug:
    var: homes.stdout_lines[1]
```

Nota

La variabile `homes.stdout_lines' è un elenco di variabili di tipo stringa, un modo per organizzare le variabili che non avevamo ancora incontrato.

Le stringhe che compongono la variabile memorizzata possono essere consultate tramite il valore `stdout` (che ti permette di fare cose come `homes.stdout.find("core") != -1`), per sfruttarli usando un ciclo (vedi `loop`), o semplicemente dai loro indici come visto nell'esempio precedente.

4.1.4 Esercizi:

- Scrivere un playbook, `play-vars.yml,` usando variabili globali che stampino il nome della distribuzione e la versione principale del target.
- Scrivi un playbook usando il seguente dizionario per visualizzare i servizi che verranno installati:

```
service:
  web:
    name: apache
    rpm: httpd
  db:
```

```
    name: mariadb
    rpm: mariadb-server
```

Il tipo predefinito dovrebbe essere "web".

- Sovrascrivi la variabile `type` usando la riga di comando
- Esternalizza le variabili in un file `vars.yml`

4.2 Gestione dei cicli

Un ciclo consente di iterare un'operazione su un elenco, un hash o un dizionario, ad esempio.

> **Nota**
>
> Ulteriori informazioni possono essere trovate qui.

Un semplice esempio di utilizzo, la creazione di 4 utenti:

```
- name: add users
  user:
    name: "{{ item }}"
    state: present
    groups: "users"
  loop:
     - antoine
     - patrick
     - steven
     - xavier
```

Ad ogni iterazione del ciclo, il valore della lista utilizzata viene memorizzato nella variabile `item`, accessibile nel codice del ciclo.

Naturalmente, un elenco può essere definito in un file esterno:

```
users:
  - antoine
  - patrick
  - steven
  - xavier
```

ed essere utilizzato all'interno del task in questo modo (dopo aver incluso il file vars):

```
- name: add users
  user:
    name: "{{ item }}"
    state: present
    groups: "users"
  loop: "{{ users }}"
```

Possiamo utilizzare l'esempio visto durante lo studio delle variabili memorizzate per migliorarlo. Uso di una variabile memorizzata:

```
- name: /home content
  shell: ls /home
  register: homes

- name: Print the directories name
  ansible.builtin.debug:
    msg: "Directory => {{ item }}"
  loop: "{{ homes.stdout_lines }}"
```

Un dizionario può anche essere usato in un ciclo.

In questo caso, è necessario trasformare il dizionario in un elemento con un filtro **jinja** (jinja è il motore di template utilizzato da Ansible): `| dict2items`.

Nel ciclo, è possibile utilizzare `item.key`, che corrisponde alla chiave del dizionario, e `item.value`, che corrisponde ai valori della chiave.

Vediamo questo attraverso un esempio concreto, mostrando la gestione degli utenti del sistema:

```
---
- hosts: rocky8
  become: true
  become_user: root
  vars:
    users:
      antoine:
        group: users
        state: present
      steven:
```

```
        group: users
        state: absent

  tasks:

  - name: Manage users
    user:
      name: "{{ item.key }}"
      group: "{{ item.value.group }}"
      state: "{{ item.value.state }}"
    loop: "{{ users | dict2items }}"
```

Nota

I loop possono essere utilizzati per molte cose. Quando l'uso di Ansible vi spingerà a utilizzarli in modo più complesso, scoprirete le possibilità che offrono.

4.2.1 Esercizi:

- Visualizza il contenuto della variabile `service` dell'esercizio precedente utilizzando un loop.

Nota

Dovrai trasformare la tua variabile `service`, che è un dizionario, in una lista con l'aiuto del filtro jinja `list` in questo modo:

```
{{ service.values() | list }}
```

4.3 Condizionali

Nota

Ulteriori informazioni possono essere trovate qui.

L'istruzione `when` è molto utile in molti casi, ad esempio per non eseguire determinate azioni su certi tipi di server, se un file o un utente non esiste, ecc.

Nota

Dietro l'istruzione `when`, le variabili non hanno bisogno di doppie parentesi (sono infatti espressioni Jinja2...).

```
- name: "Reboot only Debian servers"
  reboot:
  when: ansible_os_family == "Debian"
```

Le condizioni possono essere raggruppate tra parentesi:

```
- name: "Reboot only CentOS version 6 and Debian version 7"
  reboot:
  when: (ansible_distribution == "CentOS" and
ansible_distribution_major_version == "6") or
        (ansible_distribution == "Debian" and
ansible_distribution_major_version == "7")
```

Le condizioni corrispondenti a una logica AND possono essere fornite sotto forma di elenco:

```
- name: "Reboot only CentOS version 6"
  reboot:
  when:
    - ansible_distribution == "CentOS"
    - ansible_distribution_major_version == "6"
```

Puoi testare il valore di un booleano e verificare che sia vero:

```
- name: check if directory exists
  stat:
    path: /home/ansible
  register: directory

- ansible.builtin.debug:
    var: directory

- ansible.builtin.debug:
    msg: The directory exists
  when:
    - directory.stat.exists
    - directory.stat.isdir
```

Puoi anche verificare che non sia vero:

```
when:
  - file.stat.exists
  - not file.stat.isdir
```

Probabilmente dovrai verificare che esiste una variabile per evitare errori di esecuzione:

```
when: myboolean is defined and myboolean
```

4.3.1 Esercizi:

- Stampa il valore di `service.web` solo quando `type` è uguale a `web`.

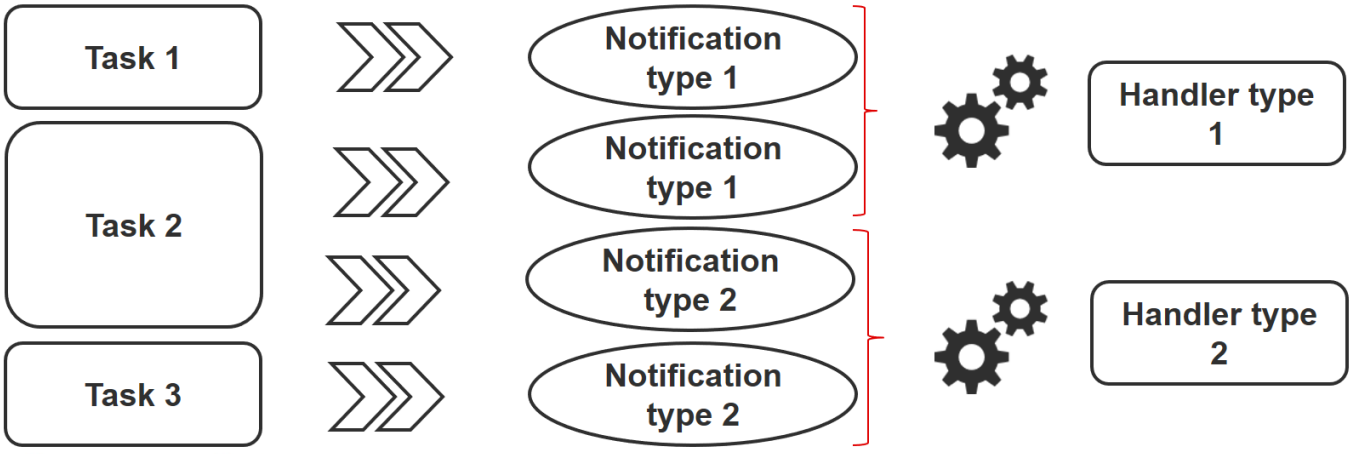
4.4 Gestione delle modifiche: gli `handlers`

Nota

Ulteriori informazioni possono essere trovate qui.

Quando si verificano le modifiche, i gestori sono autorizzati a lanciare operazioni, come il riavvio di un servizio.

Un modulo, essendo idempotente, un playbook può rilevare che c'è stato un cambiamento significativo su un sistema remoto e quindi attivare un'operazione in reazione a questo cambiamento. Una notifica viene inviata alla fine di un blocco di attività di playbook e l'operazione di reazione verrà attivata una sola volta, anche se più attività inviano la stessa notifica.



Ad esempio, diverse attività possono indicare che il servizio `httpd` deve essere riavviato a causa di un cambiamento nei suoi file di configurazione. Tuttavia, il servizio verrà riavviato solo una volta per evitare più avvii inutili.

```
- name: template configuration file
  template:
    src: template-site.j2
    dest: /etc/httpd/sites-availables/test-site.conf
  notify:
     - restart memcached
     - restart httpd
```

Un gestore è una sorta di compito referenziato da un nome globale unico:

- Uno o più notificatori lo attivano.
- Non inizia immediatamente, ma attende fino a quando tutte le attività sono complete.

Esempio di gestori:

```
handlers:

  - name: restart memcached
    systemd:
      name: memcached
      state: restarted

  - name: restart httpd
    systemd:
      name: httpd
      state: restarted
```

Dala versione 2.2 di Ansible, i gestori possono anche ascoltare direttamente:

```
handlers:

  - name: restart memcached
    systemd:
      name: memcached
      state: restarted
    listen: "web services restart"

  - name: restart apache
    systemd:
      name: apache
      state: restarted
    listen: "web services restart"
```

```
tasks:
    - name: restart everything
      command: echo "this task will restart the web services"
      notify: "web services restart"
```

4.5 Attività asincrone

> **Nota**
>
> Ulteriori informazioni possono essere trovate qui.

Per impostazione predefinita, le connessioni SSH agli host rimangono aperte durante l'esecuzione di varie attività del playbook su tutti i nodi.

Ciò può causare alcuni problemi, in particolare:

- se il tempo di esecuzione dell'attività è più lungo del timeout della connessione SSH
- se la connessione viene interrotta durante l'azione (riavvio del server, ad esempio)

In questo caso, si dovrà passare alla modalità asincrona e specificare un tempo massimo di esecuzione e la frequenza (per impostazione predefinita, 10s) con cui si controllerà lo stato dell'host.

Specificando un valore di misura di 0, Ansible eseguirà l'attività e continuerà senza preoccuparsi del risultato.

Ecco un esempio che utilizza attività asincrone, che consente di riavviare un server e attendere che la porta 22 sia nuovamente raggiungibile:

```
# Wait 2s and launch the reboot
- name: Reboot system
  shell: sleep 2 && shutdown -r now "Ansible reboot triggered"
  async: 1
  poll: 0
  ignore_errors: true
  become: true
  changed_when: False

  # Wait the server is available
  - name: Waiting for server to restart (10 mins max)
    wait_for:
```

```
      host: "{{ inventory_hostname }}"
      port: 22
      delay: 30
      state: started
      timeout: 600
    delegate_to: localhost
```

Puoi anche decidere di lanciare un'attività di lunga durata e dimenticarla (avvia e dimentica) perché l'esecuzione non ha importanza nel playbook.

4.6 Risultati delle esercitazioni

- Scrivere un playbook, `play-vars.yml, ' usando variabili globali, che stampi il nome della distribuzione del target e la versione principale.

```
---
- hosts: ansible_clients

  tasks:

    - name: Print globales variables
      debug:
        msg: "The distribution is {{ ansible_distribution }} version
{{ ansible_distribution_major_version }}"
```

```
$ ansible-playbook play-vars.yml

PLAY [ansible_clients]
**********************************************************************
**

TASK [Gathering Facts]
**********************************************************************
**
ok: [192.168.1.11]

TASK [Print globales variables]
**********************************************************************
ok: [192.168.1.11] => {
    "msg": "The distribution is Rocky version 8"
}

PLAY RECAP
**********************************************************************
*************
```

```
192.168.1.11               : ok=2    changed=0    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

- Scrivi un playbook usando il seguente dizionario per visualizzare i servizi che verranno installati:

```
service:
  web:
    name: apache
    rpm: httpd
  db:
    name: mariadb
    rpm: mariadb-server
```

Il tipo predefinito dovrebbe essere "web".

```
---
- hosts: ansible_clients
  vars:
    type: web
    service:
      web:
        name: apache
        rpm: httpd
      db:
        name: mariadb
        rpm: mariadb-server

  tasks:

    - name: Print a specific entry of a dictionary
      debug:
        msg: "The {{ service[type]['name'] }} will be installed with the
packages {{ service[type].rpm }}"
```

```
$ ansible-playbook display-dict.yml

PLAY [ansible_clients]
*********************************************************************************
**

TASK [Gathering Facts]
*********************************************************************************
**
ok: [192.168.1.11]
```

```
TASK [Print a specific entry of a dictionnaire]
*****************************************************
ok: [192.168.1.11] => {
    "msg": "The apache will be installed with the packages httpd"
}

PLAY RECAP
*********************************************************************************
**************
192.168.1.11               : ok=2    changed=0    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

- Sovrascrivi la variabile `type` usando la riga di comando:

```
ansible-playbook --extra-vars "type=db" display-dict.yml

PLAY [ansible_clients]
*********************************************************************************
**

TASK [Gathering Facts]
*********************************************************************************
**
ok: [192.168.1.11]

TASK [Print a specific entry of a dictionary]
*****************************************************
ok: [192.168.1.11] => {
    "msg": "The mariadb will be installed with the packages mariadb-server"
}

PLAY RECAP
*********************************************************************************
**************
192.168.1.11               : ok=2    changed=0    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

- Esternalizza le variabili in un file `vars.yml`

```
type: web
service:
  web:
    name: apache
    rpm: httpd
  db:
```

```
    name: mariadb
    rpm: mariadb-server
```

```
---
- hosts: ansible_clients
  vars_files:
    - vars.yml

  tasks:

    - name: Print a specific entry of a dictionary
      debug:
        msg: "The {{ service[type]['name'] }} will be installed with the
packages {{ service[type].rpm }}"
```

- Visualizza il contenuto della variabile `service` dell'esercizio precedente utilizzando un ciclo.

Nota

Dovrai trasformare la tua variabile `service`, che è un dizionario, in una lista con l'aiuto del filtro jinja `list` in questo modo:

```
{{ service | dict2items }}
```

```
{{ service.values() | list }}
```

Con `dict2items`:

```
---
- hosts: ansible_clients
  vars_files:
    - vars.yml

  tasks:

    - name: Print a dictionary variable with a loop
      debug:
        msg: "{{item.key }} | The {{ item.value.name }} will be installed with
the packages {{ item.value.rpm }}"
      loop: "{{ service | dict2items }}"
```

```
$ ansible-playbook display-dict.yml

PLAY [ansible_clients]
*********************************************************************************
```

```
**

TASK [Gathering Facts]
*********************************************************************************
**
ok: [192.168.1.11]

TASK [Print a dictionary variable with a loop]
*************************************************
ok: [192.168.1.11] => (item={'key': 'web', 'value': {'name': 'apache', 'rpm':
'httpd'}}) => {
    "msg": "web | The apache will be installed with the packages httpd"
}
ok: [192.168.1.11] => (item={'key': 'db', 'value': {'name': 'mariadb', 'rpm':
'mariadb-server'}}) => {
    "msg": "db | The mariadb will be installed with the packages mariadb-
server"
}

PLAY RECAP
*********************************************************************************
**************
192.168.1.11               : ok=2    changed=0    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

Con `list`:

```
---
- hosts: ansible_clients
  vars_files:
    - vars.yml

  tasks:

    - name: Print a dictionary variable with a loop
      debug:
        msg: "The {{ item.name }} will be installed with the packages
{{ item.rpm }}"
      loop: "{{ service.values() | list}}"
~
```

```
$ ansible-playbook display-dict.yml

PLAY [ansible_clients]
*********************************************************************************
**
```

```
TASK [Gathering Facts]
*********************************************************************************
**
ok: [192.168.1.11]

TASK [Print a dictionary variable with a loop]
*************************************************
ok: [192.168.1.11] => (item={'name': 'apache', 'rpm': 'httpd'}) => {
    "msg": "The apache will be installed with the packages httpd"
}
ok: [192.168.1.11] => (item={'name': 'mariadb', 'rpm': 'mariadb-server'}) => {
    "msg": "The mariadb will be installed with the packages mariadb-server"
}

PLAY RECAP
*********************************************************************************
**************
192.168.1.11               : ok=2    changed=0    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

- Stampa il valore di `service.web` solo quando `type` è uguale a `web`.

```
---
- hosts: ansible_clients
  vars_files:
    - vars.yml

  tasks:

    - name: Print a dictionary variable
      debug:
        msg: "The {{ service.web.name }} will be installed with the packages
{{ service.web.rpm }}"
      when: type == "web"


    - name: Print a dictionary variable
      debug:
        msg: "The {{ service.db.name }} will be installed with the packages
{{ service.db.rpm }}"
      when: type == "db"
```

```
$ ansible-playbook display-dict.yml

PLAY [ansible_clients]
```

```
*********************************************************************************
**

TASK [Gathering Facts]
*********************************************************************************
**
ok: [192.168.1.11]

TASK [Print a dictionary variable]
*********************************************************************
ok: [192.168.1.11] => {
    "msg": "The apache will be installed with the packages httpd"
}

TASK [Print a dictionary variable]
*********************************************************************
skipping: [192.168.1.11]

PLAY RECAP
*********************************************************************************
**************
192.168.1.11               : ok=2    changed=0    unreachable=0    failed=0
skipped=1    rescued=0    ignored=0

$ ansible-playbook --extra-vars "type=db" display-dict.yml

PLAY [ansible_clients]
*********************************************************************************
**

TASK [Gathering Facts]
*********************************************************************************
**
ok: [192.168.1.11]

TASK [Print a dictionary variable]
*********************************************************************
skipping: [192.168.1.11]

TASK [Print a dictionary variable]
*********************************************************************
ok: [192.168.1.11] => {
    "msg": "The mariadb will be installed with the packages mariadb-server"
}

PLAY RECAP
*********************************************************************************
**************
```

```
192.168.1.11                : ok=2    changed=0    unreachable=0    failed=0
skipped=1    rescued=0    ignored=0
```

5. Ansible - Gestione dei file

In questo capitolo imparerai come gestire i file con Ansable.

Obiettivi: In questo capitolo imparerai come:

✔ modificare il contenuto del file;
✔ caricare i file ai server di destinazione;
✔ recuperare i file dai server di destinazione.

🏁 **ansible**, **moduli**, **files**

Conoscenza: ⭐⭐
Complessità: ⭐

Tempo di lettura: 20 minuti

A seconda delle vostre esigenze, dovrete utilizzare diversi moduli Ansible per modificare i file di configurazione del sistema.

5.1 modulo `ini_file`

Quando si desidera modificare un file INI (sezione tra doppie `[]` e `key=value`), il modo più semplice è usare il modulo `ini_file`.

> **Nota**
>
> Ulteriori informazioni possono essere trovate qui.

Il modulo richiede:

- Il valore della sezione
- Il nome dell'opzione
- Il nuovo valore

Esempio di utilizzo:

```
- name: change value on inifile
  community.general.ini_file:
    dest: /path/to/file.ini
    section: SECTIONNAME
    option: OPTIONNAME
    value: NEWVALUE
```

5.2 modulo `lineinfile`

Per garantire che una riga sia presente in un file, o quando una singola riga in un file debba essere aggiunta o modificata, usa il modulo `linefile`.

> **Nota**
>
> Maggiori informazioni possono essere trovate qui.

In questo caso, la riga da modificare in un file verrà trovata usando un regexp.

Ad esempio, per garantire che la linea che inizia con `SELINUX=` nel file `/etc/selinux/config` contenga il valore `enforcing`:

```
- ansible.builtin.lineinfile:
    path: /etc/selinux/config
    regexp: '^SELINUX='
    line: 'SELINUX=enforcing'
```

5.3 modulo `copy`

Quando un file deve essere copiato dal server Ansible in uno o più host, è meglio utilizzare il modulo `copy`.

> **Nota**
>
> Maggiori informazioni possono essere trovate qui.

Qui stiamo copiando `myflile.conf` da una posizione all'altra:

```
- ansible.builtin.copy:
    src: /data/ansible/sources/myfile.conf
    dest: /etc/myfile.conf
    owner: root
```

```
    group: root
    mode: 0644
```

5.4 modulo `fetch`

Quando un file deve essere copiato da un server remoto al server locale, è meglio utilizzare il modulo `fetch`.

> **Nota**
>
> Ulteriori informazioni possono essere trovate qui.

Questo modulo fa il contrario del modulo `copy`:

```
- ansible.builtin.fetch:
    src: /etc/myfile.conf
    dest: /data/ansible/backup/myfile-{{ inventory_hostname }}.conf
    flat: yes
```

5.5 modulo `template`

Ansible e il suo modulo `template` utilizzano il sistema di template **Jinja2** (http://jinja.pocoo.org/docs/) per generare i file sugli host di destinazione.

> **Nota**
>
> Ulteriori informazioni possono essere trovate qui.

Per esempio:

```
- ansible.builtin.template:
    src: /data/ansible/templates/monfichier.j2
    dest: /etc/myfile.conf
    owner: root
    group: root
    mode: 0644
```

È possibile aggiungere una fase di convalida se il servizio di destinazione lo permette (ad esempio apache con il comando `apachectl -t`):

```
- template:
    src: /data/ansible/templates/vhost.j2
    dest: /etc/httpd/sites-available/vhost.conf
    owner: root
    group: root
    mode: 0644
    validate: '/usr/sbin/apachectl -t'
```

5.6 modulo `get_url`

Per caricare file da un sito web o ftp a uno o più host, utilizzare il modulo `get_url`:

```
- get_url:
    url: http://site.com/archive.zip
    dest: /tmp/archive.zip
    mode: 0640
    checksum:
sha256:f772bd36185515581aa9a2e4b38fb97940ff28764900ba708e68286121770e9a
```

Fornendo un checksum del file, il file non verrà nuovamente scaricato se è già presente nella posizione di destinazione e il suo checksum corrisponde al valore fornito.

6. Galassia Ansibile: Collezioni e Ruoli

In questo capitolo imparerai come usare, installare e gestire ruoli e collezioni Ansible.

Obiettivi: In questo capitolo imparerai come:

✓ installare e gestire collezioni;
✓ installare e gestire i ruoli.

🏁 **ansible**, **ansible-galaxy**, **ruoli**, **collezioni**

Conoscenza: ⭐⭐
Complessità: ⭐⭐⭐

Tempo di lettura: 40 minuti

La Galassia Ansible fornisce Ruoli Ansible e Collezioni della Comunità Ansible.

Gli elementi forniti possono essere referenziati nei playbook e pronti da usare

6.1 comando `ansible-galaxy`

Il comando `ansible-galaxy` gestisce ruoli e collezioni utilizzando galaxy.ansible.com.

- Per gestire i ruoli:

```
ansible-galaxy role [import|init|install|login|remove|...]
```

Sotto comandi	Funzionalità
`install`	installare un ruolo.
`remove`	rimuovere uno o più ruoli.
`list`	mostrare il nome e la versione dei ruoli installati.
`info`	visualizzare informazioni su un ruolo.
`init`	generare uno scheletro di un nuovo ruolo.
`import`	importare un ruolo dal sito web della galassia. Richiede un login.

- Per gestire le collezioni:

```
ansible-galaxy collection [import|init|install|login|remove|...]
```

Sotto comandi	Funzionalità
`init`	generare uno scheletro di una nuova collezione.
`install`	installare una collezione.
`list`	visualizzare il nome e la versione delle collezioni installate.

6.2 Ruoli Ansible

Un ruolo Ansible è un'unità che promuove la riutilizzabilità dei playbook.

> **Nota**
>
> Ulteriori informazioni possono essere trovate qui

6.2.1 Installazione di ruoli utili

Al fine di evidenziare l'interesse ad utilizzare i ruoli, vi suggerisco di utilizzare il ruolo `alemorvan/patchmanagement`, che vi permetterà di eseguire un sacco di attività (pre-aggiornamento o post-aggiornamento, per esempio) durante il processo di aggiornamento, in poche righe di codice.

Puoi controllare il codice nel repo github del ruolo qui.

- Installare il ruolo. Questo richiede un solo comando:

```
ansible-galaxy role install alemorvan.patchmanagement
```

- Creare un playbook per includere il ruolo:

```
- name: Start a Patch Management
  hosts: ansible_clients
  vars:
    pm_before_update_tasks_file: custom_tasks/pm_before_update_tasks_file.yml
    pm_after_update_tasks_file: custom_tasks/pm_after_update_tasks_file.yml

  tasks:
    - name: "Include patchmanagement"
      include_role:
        name: "alemorvan.patchmanagement"
```

Con questo ruolo, puoi aggiungere i tuoi compiti per tutto il tuo inventario o solo per il nodo selezionato.

Creiamo delle attività che saranno eseguite prima e dopo il processo di aggiornamento:

- Crea la cartella `custom_tasks`:

```
mkdir custom_tasks
```

- Crea `custom_tasks/pm_before_update_tasks_file.yml` (puoi liberamente cambiare il nome e il contenuto di questo file)

```
---
- name: sample task before the update process
  debug:
    msg: "This is a sample tasks, feel free to add your own test task"
```

- Crea `custom_tasks/pm_after_update_tasks_file.yml` (puoi liberamente cambiare il nome e il contenuto di questo file)

```
---
- name: sample task after the update process
  debug:
    msg: "This is a sample tasks, feel free to add your own test task"
```

E lancia il tuo primo Gestione Patch:

```
ansible-playbook patchmanagement.yml

PLAY [Start a Patch Management]
*********************************************************************

TASK [Gathering Facts]
*********************************************************************************
***
ok: [192.168.1.11]

TASK [Include patchmanagement]
*********************************************************************

TASK [alemorvan.patchmanagement : MAIN | Linux Patch Management Job]
***********************************
ok: [192.168.1.11] => {
    "msg": "Start 192 patch management"
}

...

TASK [alemorvan.patchmanagement : sample task before the update process]
*******************************
ok: [192.168.1.11] => {
    "msg": "This is a sample tasks, feel free to add your own test task"
}

...

TASK [alemorvan.patchmanagement : MAIN | We can now patch]
*********************************************
included: /home/ansible/.ansible/roles/alemorvan.patchmanagement/tasks/
patch.yml for 192.168.1.11

TASK [alemorvan.patchmanagement : PATCH | Tasks depends on distribution]
*******************************
ok: [192.168.1.11] => {
    "ansible_distribution": "Rocky"
}

TASK [alemorvan.patchmanagement : PATCH | Include tasks for CentOS & RedHat
tasks] **********************
included: /home/ansible/.ansible/roles/alemorvan.patchmanagement/tasks/
linux_tasks/redhat_centos.yml for 192.168.1.11

TASK [alemorvan.patchmanagement : RHEL CENTOS | yum clean all]
```

```
*****************************************
changed: [192.168.1.11]

TASK [alemorvan.patchmanagement : RHEL CENTOS | Ensure yum-utils is installed]
*************************
ok: [192.168.1.11]

TASK [alemorvan.patchmanagement : RHEL CENTOS | Remove old kernels]
***********************************
skipping: [192.168.1.11]

TASK [alemorvan.patchmanagement : RHEL CENTOS | Update rpm package with yum]
****************************
ok: [192.168.1.11]

TASK [alemorvan.patchmanagement : PATCH | Inlude tasks for Debian & Ubuntu
tasks] ***********************
skipping: [192.168.1.11]

TASK [alemorvan.patchmanagement : MAIN | We can now reboot]
*******************************************
included: /home/ansible/.ansible/roles/alemorvan.patchmanagement/tasks/
reboot.yml for 192.168.1.11

TASK [alemorvan.patchmanagement : REBOOT | Reboot triggered]
*****************************************
ok: [192.168.1.11]

TASK [alemorvan.patchmanagement : REBOOT | Ensure we are not in rescue mode]
****************************
ok: [192.168.1.11]

...

TASK [alemorvan.patchmanagement : FACTS | Insert fact file]
*******************************************
ok: [192.168.1.11]

TASK [alemorvan.patchmanagement : FACTS | Save date of last PM]
*****************************************
ok: [192.168.1.11]

...

TASK [alemorvan.patchmanagement : sample task after the update process]
*********************************
ok: [192.168.1.11] => {
    "msg": "This is a sample tasks, feel free to add your own test task"
}
```

```
PLAY RECAP
*********************************************************************************
***************
192.168.1.11               : ok=31   changed=1    unreachable=0    failed=0
skipped=4    rescued=0    ignored=0
```

Piuttosto facile per un processo così complesso, vero?

Questo è solo un esempio di ciò che può essere fatto utilizzando i ruoli resi disponibili dalla comunità. Dai un'occhiata a galaxy.ansible.com per scoprire i ruoli che potrebbero essere utili per te!

Puoi anche creare i tuoi ruoli per le tue esigenze e pubblicarli su Internet se te la senti. Questo è quanto affronteremo brevemente nel prossimo capitolo.

6.2.2 Introduzione allo sviluppo del ruolo

Uno scheletro di ruolo, che serve come punto di partenza per lo sviluppo di ruolo personalizzato, può essere generato dal comando `ansible-galaxy`:

```
$ ansible-galaxy role init rocky8
- Role rocky8 was created successfully
```

Il comando genererà la seguente struttura ad albero per contenere il ruolo `rocky8`:

```
tree rocky8/
rocky8/
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── README.md
├── tasks
│   └── main.yml
├── templates
├── tests
│   ├── inventory
│   └── test.yml
└── vars
```

```
    └── main.yml

8 directories, 8 files
```

I ruoli consentono di eliminare la necessità di includere i file. Non c'è bisogno di specificare i percorsi dei file o `includere` direttive nei playbook. Devi solo specificare un compito, e Ansible si occupa delle inclusioni.

La struttura di un ruolo è abbastanza evidente da capire.

Le variabili sono semplicemente memorizzate in `vars/main.yml` se le variabili non devono essere sovrascritte, oppure in `default/main.yml` se vuoi lasciare la possibilità di sovrascrivere il contenuto variabile dall'esterno del tuo ruolo.

I gestori, i file e i modelli necessari per il tuo codice sono memorizzati rispettivamente in `handlers/main.yml`, `files` e `templates`.

Tutto ciò che rimane è definire il codice per le attività del tuo ruolo in `tasks/main.yml`.

Una volta che tutto questo funziona bene, è possibile utilizzare questo ruolo nei tuoi playbook. Sarete in grado di utilizzare il vostro ruolo senza preoccuparvi dell'aspetto tecnico dei suoi compiti, durante la personalizzazione del suo funzionamento con variabili.

6.2.3 Lavoro pratico: creare un primo ruolo semplice

Implementiamo questo con un ruolo "go anywhere" che creerà un utente predefinito e installerà pacchetti software. Questo ruolo può essere applicato sistematicamente a tutti i tuoi server.

Variabili

Creeremo un utente `rockstar` su tutti i nostri server. Poiché non vogliamo che questo utente sia sovrascritto, definiamolo nel `vars/main.yml`:

```
---
rocky8_default_group:
  name: rockstar
  gid: 1100
```

```
rocky8_default_user:
  name: rockstar
  uid: 1100
  group: rockstar
```

Ora possiamo usare queste variabili all'interno dei nostri `tasks/main.yml` senza alcuna inclusione.

```
---
- name: Create default group
  group:
    name: "{{ rocky8_default_group.name }}"
    gid: "{{ rocky8_default_group.gid }}"

- name: Create default user
  user:
    name: "{{ rocky8_default_user.name }}"
    uid: "{{ rocky8_default_user.uid }}"
    group: "{{ rocky8_default_user.group }}"
```

Per testare il tuo nuovo ruolo, creiamo un playbook `test-role.yml` nella stessa directory del tuo ruolo:

```
---
- name: Test my role
  hosts: localhost

  roles:

    - role: rocky8
      become: true
      become_user: root
```

e lancialo:

```
ansible-playbook test-role.yml

PLAY [Test my role]
*********************************************************************************
*****

TASK [Gathering Facts]
*********************************************************************************
**
```

```
ok: [localhost]

TASK [rocky8 : Create default group]
*********************************************************************
changed: [localhost]

TASK [rocky8 : Create default user]
*********************************************************************
changed: [localhost]

PLAY RECAP
*********************************************************************************
**************
localhost                  : ok=3    changed=1    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

Congratulazioni! Ora siete in grado di creare grandi cose con un playbook di solo poche righe.

Vediamo l'uso delle variabili predefinite.

Crea un elenco di pacchetti da installare per impostazione predefinita sui server e un elenco vuoto di pacchetti da disinstallare. Modifica i file `defaults/main.yml` e aggiungi questi due elenchi:

```
rocky8_default_packages:
  - tree
  - vim
rocky8_remove_packages: []
```

e usali nei tuoi `tasks/main.yml`:

```
- name: Install default packages (can be overridden)
  package:
    name: "{{ rocky8_default_packages }}"
    state: present

- name: "Uninstall default packages (can be overridden)
{{ rocky8_remove_packages }}"
  package:
    name: "{{ rocky8_remove_packages }}"
    state: absent
```

Verifica il tuo ruolo con l'aiuto del playbook precedentemente creato:

```
ansible-playbook test-role.yml

PLAY [Test my role]
*********************************************************************************
*****

TASK [Gathering Facts]
*********************************************************************************
**
ok: [localhost]

TASK [rocky8 : Create default group]
*******************************************************************
ok: [localhost]

TASK [rocky8 : Create default user]
*******************************************************************
ok: [localhost]

TASK [rocky8 : Install default packages (can be overridden)]
*****************************************
ok: [localhost]

TASK [rocky8 : Uninstall default packages (can be overridden) []]
*************************************
ok: [localhost]

PLAY RECAP
*********************************************************************************
**************
localhost                  : ok=5    changed=0    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

Ora puoi sovrascrivere i `rocky8_remove_packages` nel tuo playbook e disinstallare ad esempio `cockpit`:

```
---
- name: Test my role
  hosts: localhost
  vars:
    rocky8_remove_packages:
      - cockpit

  roles:
```

```
    - role: rocky8
      become: true
      become_user: root
```

```
ansible-playbook test-role.yml

PLAY [Test my role]
*********************************************************************************
*****

TASK [Gathering Facts]
*********************************************************************************
**
ok: [localhost]

TASK [rocky8 : Create default group]
*****************************************************************
ok: [localhost]

TASK [rocky8 : Create default user]
*****************************************************************
ok: [localhost]

TASK [rocky8 : Install default packages (can be overridden)]
*****************************************
ok: [localhost]

TASK [rocky8 : Uninstall default packages (can be overridden) ['cockpit']]
*****************************
changed: [localhost]

PLAY RECAP
*********************************************************************************
**************
localhost                  : ok=5    changed=1    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

Ovviamente, non c'è limite a quanto si può migliorare il proprio ruolo. Immaginate che per uno dei vostri server, avete bisogno di un pacchetto che è nell'elenco di quelli da disinstallare. Si potrebbe quindi, ad esempio, creare una nuova lista che può essere sovrascritta e quindi rimuovere dall'elenco dei pacchetti da disinstallare quelli nell'elenco dei pacchetti specifici da installare utilizzando il filtro jinja `difference()`.

```
- name: "Uninstall default packages (can be overridden)
{{ rocky8_remove_packages }}"
  package:
    name: "{{ rocky8_remove_packages |
difference(rocky8_specifics_packages) }}"
    state: absent
```

6.3 Collezioni Ansible

Le collezioni sono un formato di distribuzione per i contenuti Ansible che possono includere libri di gioco, ruoli, moduli e plugin.

> **Nota**
>
> Ulteriori informazioni possono essere trovate qui

Per installare o aggiornare una collezione:

```
ansible-galaxy collection install namespace.collection [--upgrade]
```

È quindi possibile utilizzare la collezione appena installata usando lo spazio dei nomi e il nome del modulo prima del nome o del ruolo:

```
- import_role:
    name: namespace.collection.rolename

- namespace.collection.modulename:
    option1: value
```

Puoi trovare un indice di raccolta qui.

Installiamo la collezione `community.general`:

```
ansible-galaxy collection install community.general
Starting galaxy collection install process
Process install dependency map
Starting collection install process
Downloading https://galaxy.ansible.com/download/community-general-3.3.2.tar.gz
to /home/ansible/.ansible/tmp/ansible-local-51384hsuhf3t5/tmpr_c9qrt1/
community-general-3.3.2-f4q9u4dg
Installing 'community.general:3.3.2' to '/home/ansible/.ansible/collections/
```

```
ansible_collections/community/general'
community.general:3.3.2 was installed successfully
```

Ora possiamo utilizzare il nuovo modulo disponibile `yum_versionlock`:

```
- name: Start a Patch Management
  hosts: ansible_clients
  become: true
  become_user: root
  tasks:

    - name: Ensure yum-versionlock is installed
      package:
        name: python3-dnf-plugin-versionlock
        state: present

    - name: Prevent kernel from being updated
      community.general.yum_versionlock:
        state: present
        name: kernel
      register: locks

    - name: Display locks
      debug:
        var: locks.meta.packages
```

```
ansible-playbook versionlock.yml

PLAY [Start a Patch Management]
*********************************************************************

TASK [Gathering Facts]
*************************************************************************************
***
ok: [192.168.1.11]

TASK [Ensure yum-versionlock is installed]
*************************************************************
changed: [192.168.1.11]

TASK [Prevent kernel from being updated]
*************************************************************
changed: [192.168.1.11]

TASK [Display locks]
*************************************************************************************
```

```
*****
ok: [192.168.1.11] => {
    "locks.meta.packages": [
        "kernel"
    ]
}

PLAY RECAP
*********************************************************************************
***************
192.168.1.11               : ok=4    changed=2    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

6.3.1 Creare la propria collezione

Come per i ruoli, puoi creare la tua collezione con l'aiuto del comando `ansible-galaxy`:

```
ansible-galaxy collection init rocky8.rockstarcollection
- Collection rocky8.rockstarcollection was created successfully
```

```
tree rocky8/rockstarcollection/
rocky8/rockstarcollection/
├── docs
├── galaxy.yml
├── plugins
│   └── README.md
├── README.md
└── roles
```

È quindi possibile memorizzare i propri plugin o ruoli all'interno di questa nuova collezione.

7. Distribuzione Ansible con Ansistrano

In questo capitolo imparerai come distribuire applicazioni con il ruolo Ansible Ansistrano.

Obiettivi: In questo capitolo imparerai come:

✔ Implementare Ansistrano;
✔ Configurare Ansistrano;
✔ Usare cartelle e file condivisi tra le versioni distribuite;
✔ Distribuire diverse versioni di un sito da git;
✔ Reagire tra le fasi di distribuzione.

🏁 **ansible**, **ansistrano**, **ruoli**, **distribuzioni**

Conoscenza: ⭐⭐
Complessità: ⭐⭐⭐

Tempo di lettura: 40 minuti

Ansistrano è un ruolo Ansible per distribuire facilmente applicazioni PHP, Python, ecc. Si basa sulla funzionalità di Capistrano.

7.1 Introduzione

Ansistrano richiede quanto segue:

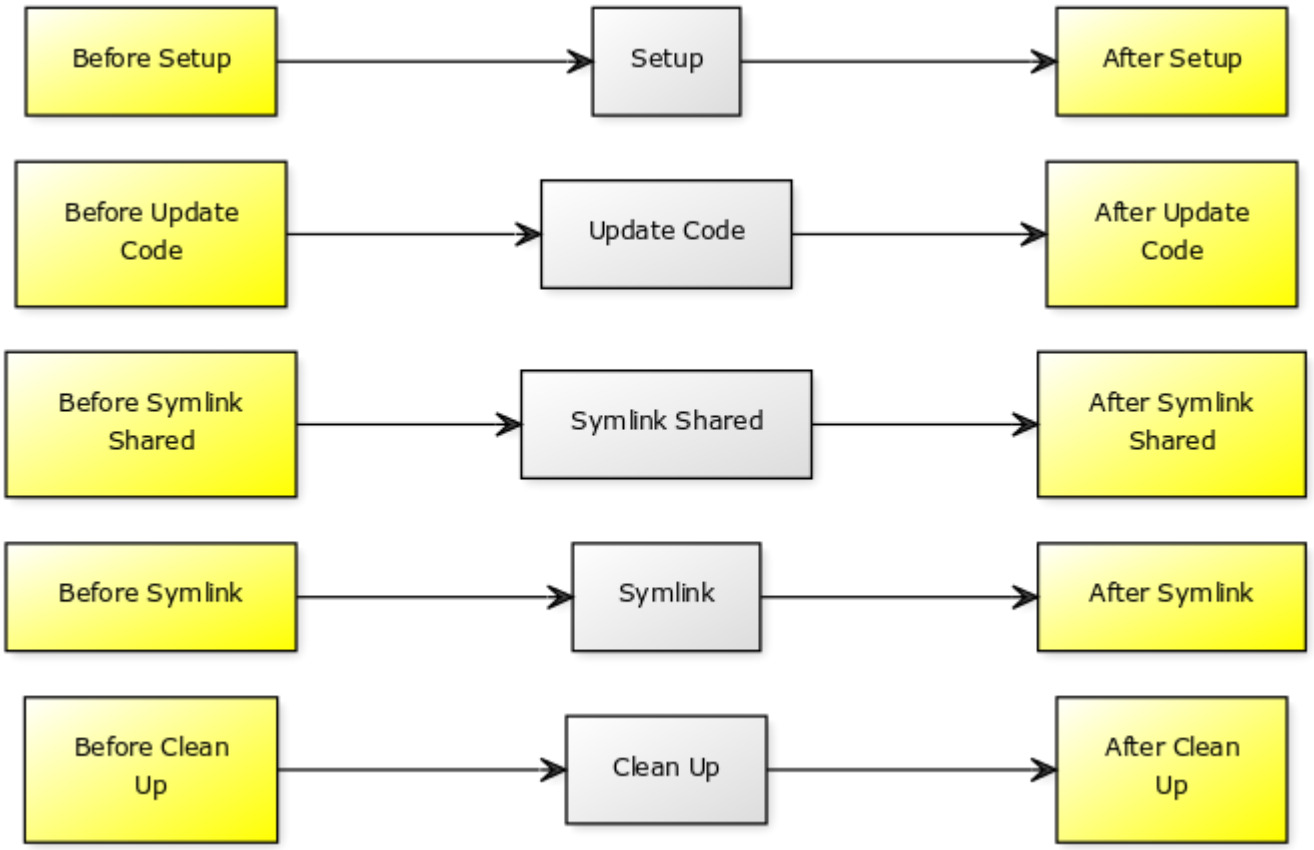
- Ansible sulla macchina di distribuzione,
- `rsync` o `git` sulla macchina client.

Può scaricare il codice sorgente da `rsync`, `git`, `scp`, `http`, `S3`, ...

> **Nota**
>
> Per il nostro esempio di distribuzione, utilizzeremo il protocollo `git`.

Ansistrano distribuisce applicazioni seguendo questi 5 passaggi:

- **Setup**: crea la struttura delle directory per ospitare le release;
- **Update Code**: scaricando la nuova release per gli obiettivi;
- **Symlink Shared** e **Symlink**: dopo aver distribuito la nuova release, il `link simbolico corrente` è modificato per puntare a questa nuova versione;
- **Clean Up**: per fare un po' di pulizia (rimuovi le vecchie versioni).



Lo scheletro di una distribuzione con Ansistrano assomiglia a questo:

```
/var/www/site/
├── current -> ./releases/20210718100000Z
├── releases
│   └── 20210718100000Z
│       ├── css -> ../../shared/css/
│       ├── img -> ../../shared/img/
│       └── REVISION
├── repo
└── shared
    ├── css/
    └── img/
```

Puoi trovare tutta la documentazione di Ansistrano sul suo repository Github.

7.2 Labs

Continuerai a lavorare sui tuoi 2 server:

Il server di gestione:

• Ansible è già installato. Dovrai installare il ruolo `ansistrano.deploy`.

Il server gestito:

• Sarà necessario installare Apache e distribuire il sito client.

7.2.1 Distribuzione del server Web

Per una maggiore efficienza, useremo il ruolo `geerlingguy.apache` per configurare il server:

```
$ ansible-galaxy role install geerlingguy.apache
Starting galaxy role install process
- downloading role 'apache', owned by geerlingguy
- downloading role from https://github.com/geerlingguy/ansible-role-apache/
archive/3.1.4.tar.gz
- extracting geerlingguy.apache to /home/ansible/.ansible/roles/
geerlingguy.apache
- geerlingguy.apache (3.1.4) was installed successfully
```

Probabilmente avremo bisogno di aprire alcune regole del firewall, quindi installeremo anche la collezione `ansible.posix` per lavorare con il suo modulo `firewalld`:

```
$ ansible-galaxy collection install ansible.posix
Starting galaxy collection install process
Process install dependency map
Starting collection install process
Downloading https://galaxy.ansible.com/download/ansible-posix-1.2.0.tar.gz to /
home/ansible/.ansible/tmp/ansible-local-519039bp65pwn/tmpsvuj1fw5/ansible-
posix-1.2.0-bhjbfdpw
Installing 'ansible.posix:1.2.0' to '/home/ansible/.ansible/collections/
ansible_collections/ansible/posix'
ansible.posix:1.2.0 was installed successfully
```

Una volta installato il ruolo e la collezione, possiamo creare la prima parte del nostro playbook, che sarà:

- Installare Apache,
- Creare una cartella di destinazione per il nostro `vhost`,
- Creare un `vhost` di default,
- Apri il firewall,
- Avviare o riavviare Apache.

Considerazioni tecniche:

- Distribuiremo il nostro sito nella cartella `/var/www/site/`.
- Come vedremo più tardi, `ansistrano` creerà un collegamento simbolico `corrente` alla cartella di rilascio corrente.
- Il codice sorgente da distribuire contiene una cartella `html` alla quale il vhost dovrebbe puntare. Il suo `DirectoryIndex` è `index.htm`.
- La distribuzione è fatta da `git`, il pacchetto sarà installato.

> **Nota**
>
> Il target del nostro vhost sarà: `/var/www/site/current/html`.

Il nostro playbook per configurare il server: `playbook-config-server.yml`

```
---
- hosts: ansible_clients
  become: yes
  become_user: root
  vars:
    dest: "/var/www/site/"
    apache_global_vhost_settings: |
      DirectoryIndex index.php index.htm
    apache_vhosts:
      - servername: "website"
    documentroot: "{{ dest }}current/html"

  tasks:

    - name: create directory for website
      file:
```

```
        path: /var/www/site/
        state: directory
        mode: 0755

    - name: install git
      package:
        name: git
        state: latest

    - name: permit traffic in default zone for http service
      ansible.posix.firewalld:
        service: http
        permanent: yes
        state: enabled
        immediate: yes

  roles:
    - { role: geerlingguy.apache }
```

Il playbook può essere applicato al server:

```
ansible-playbook playbook-config-server.yml
```

Nota l'esecuzione dei seguenti compiti:

```
TASK [geerlingguy.apache : Ensure Apache is installed on RHEL.]
****************
TASK [geerlingguy.apache : Configure Apache.]
*********************************
TASK [geerlingguy.apache : Add apache vhosts configuration.]
******************
TASK [geerlingguy.apache : Ensure Apache has selected state and enabled on
boot.] ***
TASK [permit traffic in default zone for http service]
*************************
RUNNING HANDLER [geerlingguy.apache : restart apache]
*************************
```

Il ruolo `geerlingguy.apache` rende il nostro lavoro molto più facile prendendosi cura dell'installazione e della configurazione di Apache.

Puoi controllare che tutto funzioni usando `curl`:

```
$ curl -I http://192.168.1.11
HTTP/1.1 404 Not Found
```

```
Date: Mon, 05 Jul 2021 23:30:02 GMT
Server: Apache/2.4.37 (rocky) OpenSSL/1.1.1g
Content-Type: text/html; charset=iso-8859-1
```

Nota

Non abbiamo ancora distribuito alcun codice, quindi è normale che `curl` restituisca un codice HTTP `404`. Ma possiamo già confermare che il servizio `httpd` sta funzionando e che il firewall è aperto.

7.2.2 Distribuzione del software

Ora che il nostro server è configurato, possiamo distribuire l'applicazione.

Per questo, useremo il ruolo `ansistrano.deploy` in un secondo playbook dedicato alla distribuzione delle applicazioni (per una maggiore leggibilità).

```
$ ansible-galaxy role install ansistrano.deploy
Starting galaxy role install process
- downloading role 'deploy', owned by ansistrano
- downloading role from https://github.com/ansistrano/deploy/archive/
3.10.0.tar.gz
- extracting ansistrano.deploy to /home/ansible/.ansible/roles/
ansistrano.deploy
- ansistrano.deploy (3.10.0) was installed successfully
```

Le fonti del software possono essere trovate nel repository github.

Creeremo un playbook `playbook-deploy.yml` per gestire la nostra distribuzione:

```
---
- hosts: ansible_clients
  become: yes
  become_user: root
  vars:
    dest: "/var/www/site/"
    ansistrano_deploy_via: "git"
    ansistrano_git_repo: https://github.com/alemorvan/demo-ansible.git
    ansistrano_deploy_to: "{{ dest }}"

  roles:
     - { role: ansistrano.deploy }
```

```
$ ansible-playbook playbook-deploy.yml

PLAY [ansible_clients]
*******************************************************

TASK [ansistrano.deploy : ANSISTRANO | Ensure deployment base path exists]
*****
TASK [ansistrano.deploy : ANSISTRANO | Ensure releases folder exists]
TASK [ansistrano.deploy : ANSISTRANO | Ensure shared elements folder exists]
TASK [ansistrano.deploy : ANSISTRANO | Ensure shared paths exists]
TASK [ansistrano.deploy : ANSISTRANO | Ensure basedir shared files exists]
TASK [ansistrano.deploy : ANSISTRANO | Get release version]
*******************
TASK [ansistrano.deploy : ANSISTRANO | Get release path]
TASK [ansistrano.deploy : ANSISTRANO | GIT | Register ansistrano_git_result
variable]
TASK [ansistrano.deploy : ANSISTRANO | GIT | Set git_real_repo_tree]
TASK [ansistrano.deploy : ANSISTRANO | GIT | Create release folder]
TASK [ansistrano.deploy : ANSISTRANO | GIT | Sync repo subtree[""] to release
path]
TASK [ansistrano.deploy : ANSISTRANO | Copy git released version into REVISION
file]
TASK [ansistrano.deploy : ANSISTRANO | Ensure shared paths targets are absent]
TASK [ansistrano.deploy : ANSISTRANO | Create softlinks for shared paths and
files]
TASK [ansistrano.deploy : ANSISTRANO | Ensure .rsync-filter is absent]
TASK [ansistrano.deploy : ANSISTRANO | Setup .rsync-filter with shared-folders]
TASK [ansistrano.deploy : ANSISTRANO | Get current folder]
TASK [ansistrano.deploy : ANSISTRANO | Remove current folder if it's a
directory]
TASK [ansistrano.deploy : ANSISTRANO | Change softlink to new release]
TASK [ansistrano.deploy : ANSISTRANO | Clean up releases]

PLAY RECAP
*********************************************************************************
*********************************************************************************
****************************************
192.168.1.11 : ok=25   changed=8    unreachable=0    failed=0    skipped=14
rescued=0    ignored=0
```

Tante cose fatte con sole 11 righe di codice!

```
$ curl http://192.168.1.11
<html>
<head>
<title>Demo Ansible</title>
</head>
```

```
<body>
<h1>Version Master</h1>
</body>
<html>
```

7.2.3 Controllo sul server

Ora puoi connetterti da ssh alla tua macchina client.

- Crea un `albero` nella directory `/var/www/site/`:

```
$ tree /var/www/site/
/var/www/site
├── current -> ./releases/20210722155312Z
├── releases
│   └── 20210722155312Z
│       ├── REVISION
│       └── html
│    └── index.htm
├── repo
│   └── html
│       └── index.htm
└── shared
```

Nota che:

- il `current` symlink alla release `./releases/20210722155312Z`
- la presenza di una directory `shared`
- la presenza dei git repos in `./repo/`
- Dal server Ansible riavviare la distribuzione **3** volte, quindi controllare il client.

```
$ tree /var/www/site/
var/www/site
├── current -> ./releases/20210722160048Z
├── releases
│   ├── 20210722155312Z
│   │   ├── REVISION
│   │   └── html
│   │       └── index.htm
│   ├── 20210722160032Z
│   │   ├── REVISION
│   │   └── html
```

```
│   │       └── index.htm
│   ├── 20210722160040Z
│   │   ├── REVISION
│   │   └── html
│   │       └── index.htm
│   └── 20210722160048Z
│       ├── REVISION
│       └── html
│    └── index.htm
├── repo
│   └── html
│       └── index.htm
└── shared
```

Nota che:

- `ansistrano` ha mantenuto le ultime 4 release,
- il link `current` è collegato ora all'ultima release

7.2.4 Limita il numero di release

La variabile `ansistrano_keep_releases` è usata per specificare il numero di rilasci da mantenere.

- Utilizzando la variabile `ansistrano_keep_releases`, mantieni solo 3 rilasci del progetto. Verifica.

```
---
- hosts: ansible_clients
  become: yes
  become_user: root
  vars:
    dest: "/var/www/site/"
    ansistrano_deploy_via: "git"
    ansistrano_git_repo: https://github.com/alemorvan/demo-ansible.git
    ansistrano_deploy_to: "{{ dest }}"
    ansistrano_keep_releases: 3

  roles:
     - { role: ansistrano.deploy }
```

```
---
$ ansible-playbook -i hosts playbook-deploy.yml
```

Sulla macchina client:

```
$ tree /var/www/site/
/var/www/site
├── current -> ./releases/20210722160318Z
├── releases
│   ├── 20210722160040Z
│   │   ├── REVISION
│   │   └── html
│   │       └── index.htm
│   ├── 20210722160048Z
│   │   ├── REVISION
│   │   └── html
│   │       └── index.htm
│   └── 20210722160318Z
│       ├── REVISION
│       └── html
│     └── index.htm
├── repo
│   └── html
│       └── index.htm
└── shared
```

7.2.5 Utilizzo di shared_path e shared_files

```
---
- hosts: ansible_clients
  become: yes
  become_user: root
  vars:
    dest: "/var/www/site/"
    ansistrano_deploy_via: "git"
    ansistrano_git_repo: https://github.com/alemorvan/demo-ansible.git
    ansistrano_deploy_to: "{{ dest }}"
    ansistrano_keep_releases: 3
    ansistrano_shared_paths:
      - "img"
      - "css"
    ansistrano_shared_files:
      - "logs"

  roles:
     - { role: ansistrano.deploy }
```

Sulla macchina client, crea il file `log` nella directory `shared`:

```
sudo touch /var/www/site/shared/logs
```

Quindi esegui il playbook:

```
TASK [ansistrano.deploy : ANSISTRANO | Ensure shared paths targets are absent]
*********************************************************
ok: [192.168.10.11] => (item=img)
ok: [192.168.10.11] => (item=css)
ok: [192.168.10.11] => (item=logs/log)

TASK [ansistrano.deploy : ANSISTRANO | Create softlinks for shared paths and
files] ************************************************
changed: [192.168.10.11] => (item=img)
changed: [192.168.10.11] => (item=css)
changed: [192.168.10.11] => (item=logs)
```

Sulla macchina client:

```
$  tree -F /var/www/site/
/var/www/site/
├── current -> ./releases/20210722160631Z/
├── releases/
│   ├── 20210722160048Z/
│   │   ├── REVISION
│   │   └── html/
│   │       └── index.htm
│   ├── 20210722160318Z/
│   │   ├── REVISION
│   │   └── html/
│   │       └── index.htm
│   └── 20210722160631Z/
│       ├── REVISION
│       ├── css -> ../../shared/css/
│       ├── html/
│       │   └── index.htm
│       ├── img -> ../../shared/img/
│       └── logs -> ../../shared/logs
├── repo/
│   └── html/
│       └── index.htm
└── shared/
    ├── css/
```

```
    ├── img/
    └── logs
```

Si prega di notare che l'ultima versione contiene 3 link: `css`, `img` e `log`

- da `/var/www/site/releases/css` alla directory `../../shared/css/`.
- da `/var/www/site/releases/img` alla directory `../../shared/img/`.
- da `/var/www/site/releases/logs` al file `../../shared/logs`.

Pertanto, i file contenuti in queste 2 cartelle e il file `log` sono sempre accessibili attraverso i seguenti percorsi:

- `/var/www/site/current/css/`,
- `/var/www/site/current/img/`,
- `/var/www/site/current/logs`,

ma soprattutto saranno mantenuti da una release all'altra.

7.2.6 Usa una sottodirectory del repository per la distribuzione

Nel nostro caso, il repository contiene una cartella `html`, che contiene i file del sito.

- Per evitare questo livello extra di directory, usa la variabile `ansistrano_git_repo_tree` specificando il percorso della sotto-directory da usare.

Non dimenticare di modificare la configurazione di Apache per tenere conto di questo cambiamento!

Modifica il playbook per la configurazione del server `playbook-config-server.yml`

```
---
- hosts: ansible_clients
  become: yes
  become_user: root
  vars:
    dest: "/var/www/site/"
    apache_global_vhost_settings: |
      DirectoryIndex index.php index.htm
    apache_vhosts:
      - servername: "website"
 documentroot: "{{ dest }}current/" # <1>
```

```
  tasks:

    - name: create directory for website
      file:
path: /var/www/site/
state: directory
mode: 0755

    - name: install git
      package:
name: git
state: latest

  roles:
    - { role: geerlingguy.apache }
```

<1> Modifica questa riga

Cambia il playbook per la distribuzione `playbook-deploy.yml`

```
---
- hosts: ansible_clients
  become: yes
  become_user: root
  vars:
    dest: "/var/www/site/"
    ansistrano_deploy_via: "git"
    ansistrano_git_repo: https://github.com/alemorvan/demo-ansible.git
    ansistrano_deploy_to: "{{ dest }}"
    ansistrano_keep_releases: 3
    ansistrano_shared_paths:
      - "img"
      - "css"
    ansistrano_shared_files:
      - "log"
    ansistrano_git_repo_tree: 'html' # <1>

  roles:
     - { role: ansistrano.deploy }
```

<1> Modifica questa riga

- Non dimenticare di eseguire entrambi i playbook
- Controlla la macchina cliente:

```
$  tree -F /var/www/site/
/var/www/site/
├── current -> ./releases/20210722161542Z/
├── releases/
│   ├── 20210722160318Z/
│   │   ├── REVISION
│   │   └── html/
│   │       └── index.htm
│   ├── 20210722160631Z/
│   │   ├── REVISION
│   │   ├── css -> ../../shared/css/
│   │   ├── html/
│   │   │   └── index.htm
│   │   ├── img -> ../../shared/img/
│   │   └── logs -> ../../shared/logs
│   └── 20210722161542Z/
│       ├── REVISION
│       ├── css -> ../../shared/css/
│       ├── img -> ../../shared/img/
│       ├── index.htm
│       └── logs -> ../../shared/logs
├── repo/
│   └── html/
│       └── index.htm
└── shared/
    ├── css/
    ├── img/
    └── logs
```

<1> Notare l'assenza di `html`

7.2.7 Gestione del ramo o dei tag git

La variabile `ansistrano_git_branch` è usata per specificare un `branch` o un `tag` da distribuire.

- Distribuisci il branch `releases/v1.1.0`:

```
---
- hosts: ansible_clients
  become: yes
  become_user: root
  vars:
    dest: "/var/www/site/"
    ansistrano_deploy_via: "git"
    ansistrano_git_repo: https://github.com/alemorvan/demo-ansible.git
    ansistrano_deploy_to: "{{ dest }}"
    ansistrano_keep_releases: 3
    ansistrano_shared_paths:
      - "img"
      - "css"
    ansistrano_shared_files:
      - "log"
    ansistrano_git_repo_tree: 'html'
    ansistrano_git_branch: 'releases/v1.1.0'

  roles:
     - { role: ansistrano.deploy }
```

Nota

Per divertirti, durante la distribuzione, puoi aggiornare il browser, per vedere in 'live' il cambiamento.

```
$ curl http://192.168.1.11
<html>
<head>
<title>Demo Ansible</title>
</head>
<body>
<h1>Version 1.0.1</h1>
</body>
<html>
```

- Distribuisci il tag `v2.0.0`:

```
---
- hosts: ansible_clients
  become: yes
  become_user: root
  vars:
    dest: "/var/www/site/"
    ansistrano_deploy_via: "git"
    ansistrano_git_repo: https://github.com/alemorvan/demo-ansible.git
    ansistrano_deploy_to: "{{ dest }}"
```

```
    ansistrano_keep_releases: 3
    ansistrano_shared_paths:
      - "img"
      - "css"
    ansistrano_shared_files:
      - "log"
    ansistrano_git_repo_tree: 'html'
    ansistrano_git_branch: 'v2.0.0'

  roles:
     - { role: ansistrano.deploy }
```

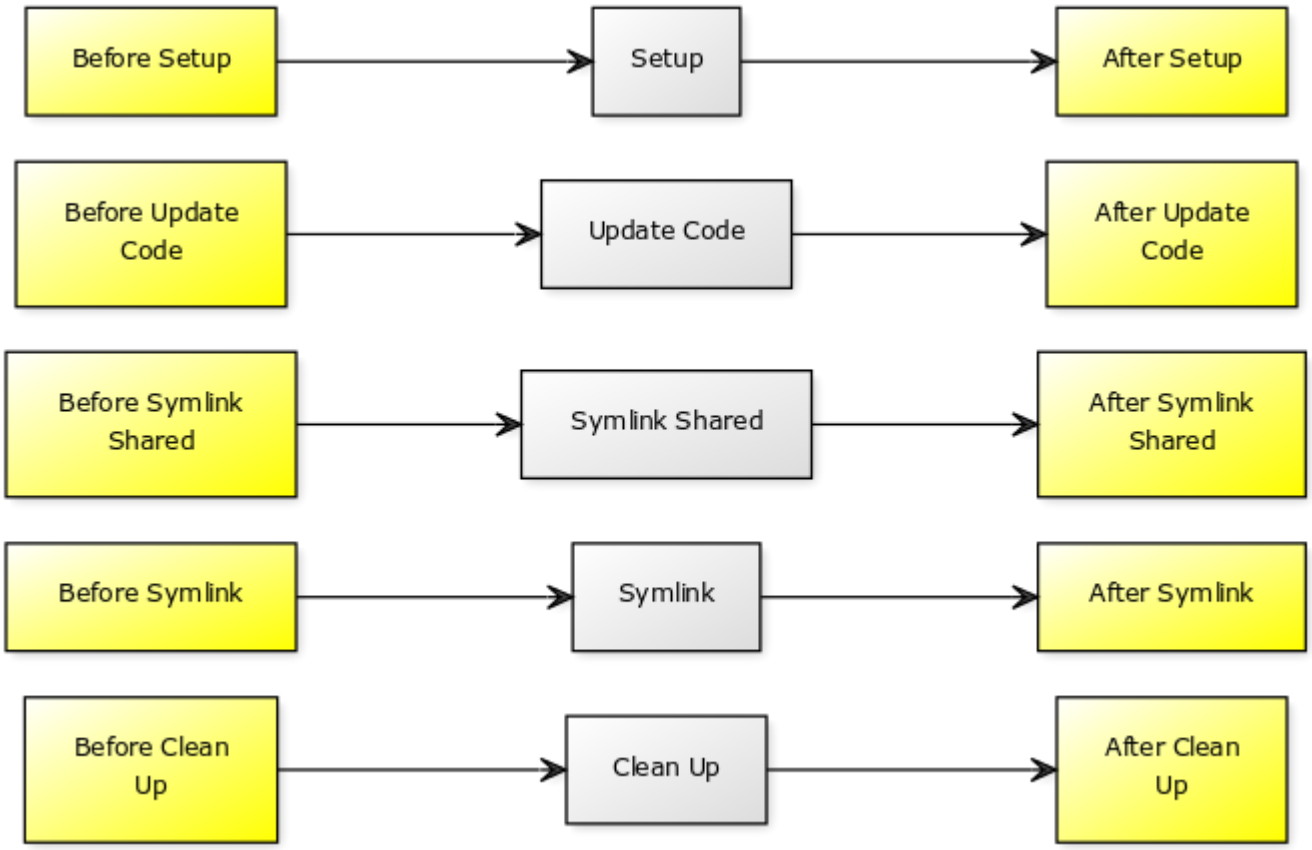
```
$ curl http://192.168.1.11
<html>
<head>
<title>Demo Ansible</title>
</head>
<body>
<h1>Version 2.0.0</h1>
</body>
<html>
```

7.2.8 Azioni tra le fasi di implementazione

Una distribuzione con Ansistrano rispetta le seguenti fasi:

- `Setup`
- `Update Code`
- `Symlink Shared`
- `Symlink`
- `Clean Up`

È possibile intervenire prima e dopo ciascuno di questi passi.



Un playbook può essere incluso attraverso le variabili fornite per questo scopo:

- `ansistrano_before_<task>_tasks_file`
- o `ansistrano_after_<task>_tasks_file`
- Esempio semplice: invia un'email (o qualsiasi cosa desideri come la notifica di Slack) all'inizio della distribuzione:

```
---
- hosts: ansible_clients
  become: yes
  become_user: root
  vars:
    dest: "/var/www/site/"
    ansistrano_deploy_via: "git"
    ansistrano_git_repo: https://github.com/alemorvan/demo-ansible.git
    ansistrano_deploy_to: "{{ dest }}"
    ansistrano_keep_releases: 3
    ansistrano_shared_paths:
      - "img"
      - "css"
    ansistrano_shared_files:
      - "logs"
```

```
    ansistrano_git_repo_tree: 'html'
    ansistrano_git_branch: 'v2.0.0'
    ansistrano_before_setup_tasks_file: "{{ playbook_dir }}/deploy/before-
setup-tasks.yml"

  roles:
     - { role: ansistrano.deploy }
```

Crea il file `deploy/before-setup-tasks.yml`:

```
---
- name: Send a mail
  mail:
    subject: Starting deployment on {{ ansible_hostname }}.
  delegate_to: localhost
```

```
TASK [ansistrano.deploy : include]
*********************************************************************************
******
included: /home/ansible/deploy/before-setup-tasks.yml for 192.168.10.11

TASK [ansistrano.deploy : Send a mail]
*********************************************************************************
******
ok: [192.168.10.11 -> localhost]
```

```
[root] # mailx
Heirloom Mail version 12.5 7/5/10.  Type ? for help.
"/var/spool/mail/root": 1 message 1 new
>N  1 root@localhost.local  Tue Aug 21 14:41  28/946   "Starting deployment on
localhost."
```

- Probabilmente dovrai riavviare alcuni servizi alla fine della distribuzione, per esempio per pulire la cache. Riavviamo Apache alla fine della distribuzione:

```
---
- hosts: ansible_clients
  become: yes
  become_user: root
  vars:
    dest: "/var/www/site/"
    ansistrano_deploy_via: "git"
    ansistrano_git_repo: https://github.com/alemorvan/demo-ansible.git
    ansistrano_deploy_to: "{{ dest }}"
```

```
    ansistrano_keep_releases: 3
    ansistrano_shared_paths:
      - "img"
      - "css"
    ansistrano_shared_files:
      - "logs"
    ansistrano_git_repo_tree: 'html'
    ansistrano_git_branch: 'v2.0.0'
    ansistrano_before_setup_tasks_file: "{{ playbook_dir }}/deploy/before-
setup-tasks.yml"
    ansistrano_after_symlink_tasks_file: "{{ playbook_dir }}/deploy/after-
symlink-tasks.yml"

  roles:
     - { role: ansistrano.deploy }
```

Crea il file `deploy/after-symlink-tasks.yml`:

```
---
- name: restart apache
  systemd:
    name: httpd
    state: restarted
```

```
TASK [ansistrano.deploy : include]
*********************************************************************
******
included: /home/ansible/deploy/after-symlink-tasks.yml for 192.168.10.11

TASK [ansistrano.deploy : restart apache]
*********************************************************************
*******
changed: [192.168.10.11]
```

Come avete visto durante questo capitolo, Ansible può migliorare notevolmente la vita dell'amministratore di sistema. Ruoli molto intelligenti come Ansistrano sono dei "must haves" che diventano rapidamente indispensabili.

L'utilizzo di Ansistrano garantisce il rispetto delle buone pratiche di diffusione, riduce i tempi necessari per mettere in produzione un sistema ed evita il rischio di potenziali errori umani. La macchina funziona velocemente, bene, e raramente commette errori!

8. Ansible - Infrastrutture su larga scala

In questo capitolo imparerai come ridimensionare il tuo sistema di gestione delle configurazioni.

Obiettivi: In questo capitolo imparerai come:

✓ Organizzare il tuo codice per un'infrastruttura di grandi dimensioni;
✓ Applicare tutto o parte della tua gestione di configurazione a un gruppo di nodi;

🏁 **ansible**, **config management**, **scale**

Conoscenza: ★★★
Complessità: ★★★★

Tempo di lettura: 30 minuti

Abbiamo visto nei capitoli precedenti come organizzare il nostro codice sotto forma di ruoli ma anche come utilizzare alcuni ruoli per la gestione degli aggiornamenti (patch management) o la distribuzione del codice.

Che dire della gestione della configurazione? Come gestire la configurazione di dieci, centinaia, o anche migliaia di macchine virtuali con Ansible?

L'avvento del cloud ha cambiato un po' i metodi tradizionali. La VM è configurata al momento della distribuzione. Se la sua configurazione non è più conforme, viene distrutta e sostituita da una nuova.

L'organizzazione del sistema di gestione della configurazione presentato in questo capitolo risponderà a questi due modi di consumare IT: "uso one-shot" o regolare "riconfigurazione" di una flotta.

Tuttavia, attenzione: l'utilizzo di Ansible per garantire la conformità di un pool di server richiede la modifica delle abitudini di lavoro. Non è più possibile modificare manualmente la configurazione di un service manager senza vedere queste modifiche sovrascritte alla prossima esecuzione di Ansible.

> **Nota**
>
> Quello che stiamo per impostare qui sotto non è il terreno preferito di Ansible. Tecnologie come Puppet o Salt faranno molto meglio. Ricordiamo che Ansible è un coltello svizzero dell'automazione ed è senza agenti, il che spiega le differenze nelle prestazioni.

> **Nota**
>
> Maggiori informazioni possono essere trovate qui

8.1 Archiviazione variabili

La prima cosa che dobbiamo discutere è la separazione tra i dati e il codice Ansible.

Poiché il codice diventa più grande e più complesso, sarà sempre più complicato modificare le variabili che contiene.

Per garantire la manutenzione del vostro sito, la cosa più importante è separare correttamente le variabili dal codice Ansible.

Non ne abbiamo ancora discusso qui, ma dovresti sapere che Ansible può caricare automaticamente le variabili che trova in cartelle specifiche a seconda del nome dell'inventario del nodo gestito, o i suoi gruppi membri.

La documentazione Ansible suggerisce di organizzare il nostro codice come sotto:

```
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
```

Se il nodo selezionato è `hostname1` di `group1`, le variabili contenute nei file `hostname1.yml` e `group1.yml` verranno caricate automaticamente. È un bel modo per memorizzare tutti i dati per tutti i tuoi ruoli nello stesso posto.

In questo modo, il file dell'inventario del tuo server diventa la sua carta d'identità. Contiene tutte le variabili che differiscono dalle variabili predefinite per il tuo server.

Dal punto di vista della centralizzazione delle variabili, diventa essenziale organizzare il nome delle sue variabili nei ruoli prefissandole, ad esempio, con il nome del ruolo. Si consiglia anche di utilizzare nomi di variabili flat piuttosto che dizionari.

Ad esempio, se si desidera rendere il valore `PermitRootLogin` nel file `sshd_config` una variabile, un buon nome della variabile potrebbe essere `sshd_config_permitrootlogin` (invece di `sshd.config.permitrootlogin` che potrebbe anche essere un buon nome di variabile).

8.2 Informazioni sui tag ansible

L'uso di tag Ansible ti permette di eseguire o saltare una parte delle attività nel tuo codice.

> **Nota**
>
> Maggiori informazioni possono essere trovate qui

Ad esempio, modifichiamo l'attività di creazione degli utenti:

```
- name: add users
  user:
    name: "{{ item }}"
    state: present
    groups: "users"
  loop:
     - antoine
     - patrick
     - steven
     - xavier
  tags: users
```

Ora puoi riprodurre solo le attività con il tag `users` con l'opzione `ansible-playbook --tags`:

```
ansible-playbook -i inventories/production/hosts --tags users site.yml
```

Puoi anche usare l'opzione `--skip-tags`.

8.3 Informazioni sul layout delle directory

Concentriamoci su una proposta per l'organizzazione di file e directory necessari per il corretto funzionamento di un CMS (Content Management System).

Il nostro punto di partenza sarà il file `site.yml`. Questo file è un po 'come il direttore d'orchestra del CMS in quanto includerà solo i ruoli necessari per i nodi di destinazione se necessario:

```
---
- name: "Config Management for {{ target }}"
  hosts: "{{ target }}"

  roles:

    - role: roles/functionality1

    - role: roles/functionality2
```

Naturalmente, questi ruoli devono essere creati sotto la directory `roles` allo stesso livello del file `site.yml`.

Mi piace gestire i miei vars globali all'interno di un `vars/global_vars.yml`, anche se potrei memorizzarli all'interno di un file situato in `inventories/production/group_vars/all.yml`

```
---
- name: "Config Management for {{ target }}"
  hosts: "{{ target }}"
  vars_files:
    - vars/global_vars.yml
  roles:

    - role: roles/functionality1

    - role: roles/functionality2
```

Mi piace inoltre mantenere la possibilità di disabilitare una funzionalità. Quindi includo i miei ruoli con una condizione e un valore predefinito come questo:

```
---
- name: "Config Management for {{ target }}"
  hosts: "{{ target }}"
  vars_files:
    - vars/global_vars.yml
  roles:

    - role: roles/functionality1
      when:
        - enable_functionality1|default(true)

    - role: roles/functionality2
      when:
        - enable_functionality2|default(false)
```

Non dimenticare di usare i tag:

```
- name: "Config Management for {{ target }}"
  hosts: "{{ target }}"
  vars_files:
    - vars/global_vars.yml
  roles:

    - role: roles/functionality1
      when:
        - enable_functionality1|default(true)
      tags:
        - functionality1

    - role: roles/functionality2
      when:
        - enable_functionality2|default(false)
      tags:
        - functionality2
```

Dovresti ottenere qualcosa di simile:

```
$ tree cms
cms
├── inventories
│   └── production
│       ├── group_vars
│       │   └── plateform.yml
│       ├── hosts
│       └── host_vars
│           ├── client1.yml
```

```
│           └── client2.yml
├── roles
│   ├── functionality1
│   │   ├── defaults
│   │   │   └── main.yml
│   │   └── tasks
│   │       └── main.yml
│   └── functionality2
│       ├── defaults
│       │   └── main.yml
│       └── tasks
│           └── main.yml
├── site.yml
└── vars
    └── global_vars.yml
```

Nota

Sei libero di sviluppare i tuoi ruoli all'interno di una collezione

8.4 Test

Avviamo il playbook ed eseguiamo alcuni test:

```
$ ansible-playbook -i inventories/production/hosts -e "target=client1" site.yml

PLAY [Config Management for client1]
*******************************************************************************

TASK [Gathering Facts]
*******************************************************************************
***********
ok: [client1]

TASK [roles/functionality1 : Task in functionality 1]
************************************************************
ok: [client1] => {
    "msg": "You are in functionality 1"
}

TASK [roles/functionality2 : Task in functionality 2]
************************************************************
skipping: [client1]

PLAY RECAP
*******************************************************************************
```

```
**********************
client1                    : ok=2    changed=0    unreachable=0    failed=0
skipped=1    rescued=0    ignored=0
```

Come puoi vedere, per impostazione predefinita, vengono giocate solo le attività del ruolo `functionality1`.

Attiviamo nell'inventario la `functionality2` per il nostro nodo mirato e riavviamo il playbook:

```
$ vim inventories/production/host_vars/client1.yml
---
enable_functionality2: true
```

```
$ ansible-playbook -i inventories/production/hosts -e "target=client1" site.yml

PLAY [Config Management for client1]
*********************************************************************************

TASK [Gathering Facts]
*********************************************************************************
***********
ok: [client1]

TASK [roles/functionality1 : Task in functionality 1]
*******************************************************
ok: [client1] => {
    "msg": "You are in functionality 1"
}

TASK [roles/functionality2 : Task in functionality 2]
*******************************************************
ok: [client1] => {
    "msg": "You are in functionality 2"
}

PLAY RECAP
*********************************************************************************
**********************
client1                    : ok=3    changed=0    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

Prova ad applicare solo `funzionalità2`:

```
$ ansible-playbook -i inventories/production/hosts -e "target=client1" --tags
functionality2 site.yml

PLAY [Config Management for client1]
*********************************************************************************

TASK [Gathering Facts]
*********************************************************************************
***********
ok: [client1]

TASK [roles/functionality2 : Task in functionality 2]
*****************************************************
ok: [client1] => {
    "msg": "You are in functionality 2"
}

PLAY RECAP
*********************************************************************************
**********************
client1                    : ok=2    changed=0    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

Eseguiamo l'intero l'inventario:

```
$ ansible-playbook -i inventories/production/hosts -e "target=plateform"
site.yml

PLAY [Config Management for plateform]
*********************************************************************

TASK [Gathering Facts]
*********************************************************************************
***********
ok: [client1]
ok: [client2]

TASK [roles/functionality1 : Task in functionality 1]
*****************************************************
ok: [client1] => {
    "msg": "You are in functionality 1"
}
ok: [client2] => {
    "msg": "You are in functionality 1"
}

TASK [roles/functionality2 : Task in functionality 2]
```

```
************************************************************
ok: [client1] => {
    "msg": "You are in functionality 2"
}
skipping: [client2]

PLAY RECAP
*********************************************************************************
***********************
client1                    : ok=3    changed=0    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
client2                    : ok=2    changed=0    unreachable=0    failed=0
skipped=1    rescued=0    ignored=0
```

Come puoi vedere, `functionality2` è riprodotto solo sul `client1`.

8.5 Vantaggi

Seguendo i consigli forniti nella documentazione Ansible otterrete rapidamente un:

- codice sorgente facilmente mantenibile anche se contiene un gran numero di ruoli
- un sistema di conformità relativamente veloce, ripetibile che è possibile applicare parzialmente o completamente
- può essere adattato caso per caso e dai server
- le specifiche del vostro sistema informativo sono separate dal codice, facilmente controllabili, e centralizzate nei file di inventario della vostra gestione della configurazione.

9. Ansible - Lavorare con filtri

In questo capitolo imparerai come trasformare i dati con i filtri jinja.

Obiettivi: In questo capitolo imparerai come:

✔ Trasformare le strutture dati come dizionari o liste;
✔ Trasformare variabili.

🏁 **ansible**, **jinja**, **filtri**

Conoscenza: ★ ★ ★
Complessità: ★ ★ ★ ★

Tempo di lettura: 20 minuti

Abbiamo già avuto la possibilità, durante i capitoli precedenti, di utilizzare i filtri jinja.

Questi filtri, scritti in python, ci permettono di manipolare e trasformare le nostre variabili ansible.

> **Nota**
>
> Maggiori informazioni possono essere trovate qui.

In tutto questo capitolo, useremo il seguente playbook per testare i diversi filtri presentati:

```
- name: Manipulating the data
  hosts: localhost
  gather_facts: false
  vars:
    zero: 0
    zero_string: "0"
    non_zero: 4
    true_booleen: True
    true_non_booleen: "True"
    false_boolean: False
```

```
    false_non_boolean: "False"
    whatever: "It's false!"
    user_name: antoine
    my_dictionary:
      key1: value1
      key2: value2
    my_simple_list:
      - value_list_1
      - value_list_2
      - value_list_3
    my_simple_list_2:
      - value_list_3
      - value_list_4
      - value_list_5
    my_list:
      - element: element1
        value: value1
      - element: element2
        value: value2

  tasks:
    - name: Print an integer
      debug:
        var: zero
```

Nota

Di seguito è riportato un elenco non esaustivo di filtri che si hanno più probabilità di incontrare o necessitare. Fortunatamente, ce ne sono molti altri. Potresti anche scrivere il tuo!

Il playbook sarà riprodotto come segue:

```
ansible-playbook play-filter.yml
```

9.1 Conversione dei dati

I dati possono essere convertiti da un tipo all'altro.

Per conoscere il tipo di dati (il tipo nel linguaggio python), è necessario utilizzare il filtro `type_debug`.

Esempio:

```
- name: Display the type of a variable
  debug:
    var: true_boolean|type_debug
```

che ci fornisce:

```
TASK [Display the type of a variable]
*********************************************************************
ok: [localhost] => {
    "true_boolean|type_debug": "bool"
}
```

È possibile trasformare un intero in una stringa:

```
- name: Transforming a variable type
  debug:
    var: zero|string
```

```
TASK [Transforming a variable type]
*********************************************************************
ok: [localhost] => {
    "zero|string": "0"
}
```

Trasforma una stringa in un intero:

```
- name: Transforming a variable type
  debug:
    var: zero_string|int
```

oppure una variabile in un booleano:

```
- name: Display an integer as a boolean
  debug:
    var: non_zero | bool

- name: Display a string as a boolean
  debug:
    var: true_non_boolean | bool

- name: Display a string as a boolean
  debug:
    var: false_non_boolean | bool
```

```
- name: Display a string as a boolean
  debug:
    var: whatever | bool
```

Una stringa di caratteri può essere trasformata in maiuscolo o minuscolo:

```
- name: Lowercase a string of characters
  debug:
    var: whatever | lower

- name: Uppercase a string of characters
  debug:
    var: whatever | upper
```

che ci fornisce:

```
TASK [Lowercase a string of characters]
*****************************************************
ok: [localhost] => {
    "whatever | lower": "it's false!"
}

TASK [Uppercase a string of characters]
*****************************************************
ok: [localhost] => {
    "whatever | upper": "IT'S FALSE!"
}
```

Il filtro `replace` consente di sostituire i caratteri con altri.

Qui rimuoviamo gli spazi o addirittura sostituiamo una parola:

```
- name: Replace a character in a string
  debug:
    var: whatever | replace(" ", "")

- name: Replace a word in a string
  debug:
    var: whatever | replace("false", "true")
```

che ci fornisce:

```
TASK [Replace a character in a string]
*****************************************************
ok: [localhost] => {
    "whatever | replace(\" \", \"\")": "It'sfalse!"
}

TASK [Replace a word in a string]
*****************************************************
ok: [localhost] => {
    "whatever | replace(\"false\", \"true\")": "It's true !"
}
```

Il filtro `split` divide una stringa in una lista in base a un carattere:

```
- name: Cutting a string of characters
  debug:
    var: whatever | split(" ", "")
```

```
TASK [Cutting a string of characters]
*****************************************************
ok: [localhost] => {
    "whatever | split(\" \")": [
        "It's",
        "false!"
    ]
}
```

9.2 Unire gli elementi di una lista

È frequente dover unire i diversi elementi in una singola stringa. Possiamo quindi specificare un carattere o una stringa da inserire tra ogni elemento.

```
- name: Joining elements of a list
  debug:
    var: my_simple_list|join(",")

- name: Joining elements of a list
  debug:
    var: my_simple_list|join(" | ")
```

che ci fornisce:

```
TASK [Joining elements of a list]
*********************************************************************
ok: [localhost] => {
    "my_simple_list|join(\",\")": "value_list_1,value_list_2,value_list_3"
}

TASK [Joining elements of a list]
*********************************************************************
ok: [localhost] => {
    "my_simple_list|join(\" | \")": "value_list_1 | value_list_2 |
value_list_3"
}
```

9.3 Trasformare dizionari in liste (e viceversa)

I filtri `dict2items` and `itemstodict`, un po 'più complessi da implementare, sono frequentemente utilizzati, soprattutto nei cicli.

Si noti che è possibile specificare il nome della chiave e del valore da usare nella trasformazione.

```
- name: Display a dictionary
  debug:
    var: my_dictionary

- name: Transforming a dictionary into a list
  debug:
    var: my_dictionary | dict2items

- name: Transforming a dictionary into a list
  debug:
    var: my_dictionary | dict2items(key_name='key', value_name='value')

- name: Transforming a list into a dictionary
  debug:
    var: my_list | items2dict(key_name='element', value_name='value')
```

```
TASK [Display a dictionary]
*********************************************************************
ok: [localhost] => {
    "my_dictionary": {
        "key1": "value1",
        "key2": "value2"
    }
```

```
}

TASK [Transforming a dictionary into a list]
************************************************************
ok: [localhost] => {
    "my_dictionary | dict2items": [
        {
            "key": "key1",
            "value": "value1"
        },
        {
            "key": "key2",
            "value": "value2"
        }
    ]
}

TASK [Transforming a dictionary into a list]
************************************************************
ok: [localhost] => {
    "my_dictionary | dict2items (key_name = 'key', value_name = 'value')": [
        {
            "key": "key1",
            "value": "value1"
        },
        {
            "key": "key2",
            "value": "value2"
        }
    ]
}

TASK [Transforming a list into a dictionary]
************************************************************
ok: [localhost] => {
    "my_list | items2dict(key_name='element', value_name='value')": {
        "element1": "value1",
        "element2": "value2"
    }
}
```

9.4 Lavorare con liste

È possibile unire o filtrare i dati da una o più liste:

```
- name: Merger of two lists
  debug:
    var: my_simple_list | union(my_simple_list_2)
```

```
ok: [localhost] => {
    "my_simple_list | union(my_simple_list_2)": [
        "value_list_1",
        "value_list_2",
        "value_list_3",
        "value_list_4",
        "value_list_5"
    ]
}
```

Per mantenere solo l'intersezione dei 2 elenchi (i valori presenti nelle 2 liste):

```
- name: Merger of two lists
  debug:
    var: my_simple_list | intersect(my_simple_list_2)
```

```
TASK [Merger of two lists]
*********************************************************************************
ok: [localhost] => {
    "my_simple_list | intersect(my_simple_list_2)": [
        "value_list_3"
    ]
}
```

O al contrario mantenere solo la differenza (i valori che non esistono nella seconda lista):

```
- name: Merger of two lists
  debug:
    var: my_simple_list | difference(my_simple_list_2)
```

```
TASK [Merger of two lists]
*********************************************************************************
ok: [localhost] => {
    "my_simple_list | difference(my_simple_list_2)": [
        "value_list_1",
        "value_list_2",
```

```
    ]
}
```

Se la tua lista contiene valori non univoci, è anche possibile filtrarli con il filtro `unique`.

```
- name: Unique value in a list
  debug:
    var: my_simple_list | unique
```

9.5 Trasformazione json/yaml

Potrebbe essere necessario importare i dati json (da un'API per esempio) o esportare i dati in yaml o json.

```
- name: Display a variable in yaml
  debug:
    var: my_list | to_nice_yaml(indent=4)

- name: Display a variable in json
  debug:
    var: my_list | to_nice_json(indent=4)
```

```
TASK [Display a variable in yaml]
**********************************************************************
ok: [localhost] => {
    "my_list | to_nice_yaml(indent=4)": "-   element: element1\n    value:
value1\n-   element: element2\n    value: value2\n"
}

TASK [Display a variable in json]
**********************************************************************
ok: [localhost] => {
    "my_list | to_nice_json(indent=4)": "[\n    {\n        \"element\":
\"element1\",\n        \"value\": \"value1\"\n    },\n    {\n
\"element\": \"element2\",\n        \"value\": \"value2\"\n    }\n]"
}
```

9.6 Valori predefiniti, variabili opzionali, proteggere le variabili

Se non fornisci valori predefiniti per le tue variabili, ti troverai rapidamente di fronte a errori nell'esecuzione dei tuoi playbook, o se non li proteggi.

Il valore di una variabile può essere sostituito con un'altra se non esiste con il filtro `default`:

```
- name: Default value
  debug:
    var: variablethatdoesnotexists | default(whatever)
```

```
TASK [Default value]
*********************************************************************************
*
ok: [localhost] => {
    "variablethatdoesnotexists | default(whatever)": "It's false!"
}
```

Nota la presenza dell'apostrofo `'` che dovrebbe essere protetto, per esempio, se usi il modulo `shell`:

```
- name: Default value
  debug:
    var: variablethatdoesnotexists | default(whatever| quote)
```

```
TASK [Default value]
*********************************************************************************
*
ok: [localhost] => {
    "variablethatdoesnotexists | default(whatever|quote)": "'It'\"'\"'s
false!'"
}
```

Infine una variabile facoltativa in un modulo può essere ignorata se non esiste con la parola chiave `omit` nel filtro `default`, che ti salverà da un errore al runtime.

```
- name: Add a new user
  ansible.builtin.user:
    name: "{{ user_name }}"
    comment: "{{ user_comment | default(omit) }}"
```

9.7 Associa un valore in base ad un altro (`ternary`)

A volte è necessario utilizzare una condizione per assegnare un valore a una variabile, in questo caso è comune passare attraverso un passaggio `set_fact`.

Questo può essere evitato utilizzando il filtro `ternary`:

```
- name: Default value
  debug:
    var: (user_name == 'antoine') | ternary('admin', 'normal_user')
```

```
TASK [Default value]
*********************************************************************************
*
ok: [localhost] => {
    "(user_name == 'antoine') | ternary('admin', 'normal_user')": "admin"
}
```

9.8 Altri filtri

- `{{ 10000 | random }}`: come indica il suo nome, dà un valore casuale.
- `{{ my_simple_list | first }}`: estrae il primo elemento della lista.
- `{{ my_simple_list | length }}`: dà la lunghezza (di una lista o di una stringa).
- `{{ ip_list | ansible.netcommon.ipv4 }}`: visualizza solo gli IP v4. Senza soffermarsi su questo, se necessario, ci sono molti filtri dedicati alla rete.
- `{{ user_password | password_hash('sha512') }}`: genera una password hash in sha512.

10. Ottimizzazioni del server di gestione

In questo capitolo, esamineremo le opzioni di configurazione che possono essere di interesse per ottimizzare il nostro server di gestione Ansible.

10.1 Il file di configurazione `ansible.cfg`

Alcune interessanti opzioni di configurazione da commentare:

- `forks`: per impostazione predefinita a 5, è il numero di processi che Ansible avvierà in parallelo per comunicare con gli host remoti. Più alto è questo numero, più clienti Ansible sarà in grado di gestire allo stesso tempo, e quindi accelerare l'elaborazione. Il valore che puoi impostare dipende dai limiti CPU/RAM del tuo server di gestione. Nota che il valore predefinito, `5`, è molto piccolo, la documentazione Ansible afferma che molti utenti la impostano a 50, anche 500 o più.
- `gathering`: questa variabile cambia la politica per la raccolta dei fatti. Per impostazione predefinita, il valore è `implicit`, il che implica che i fatti saranno raccolti sistematicamente. Il passaggio di questa variabile a `smart` consente di raccogliere i fatti solo quando non sono già stati acquisiti. Accoppiato con una cache di fatti (vedi sotto), questa opzione può aumentare notevolmente le prestazioni.
- `host_key_check`: Fai attenzione alla sicurezza del tuo server! Tuttavia, se si è in controllo del vostro ambiente, può essere interessante disattivare il controllo della chiave dei server remoti e risparmiare un po 'di tempo alla connessione. È inoltre possibile, sui server remoti, disabilitare l'utilizzo del DNS del server SSH (in `/etc/ssh/sshd_config`, opzione `UseDNS no`), questa opzione spreca tempo alla connessione ed è per la maggior parte del tempo, utilizzata solo nei registri di connessione.
- `ansible_managed`: Questa variabile, contenente `Ansible managed` per impostazione predefinita, è tipicamente utilizzata nei modelli di file che vengono distribuiti su server remoti. Consente di specificare ad un amministratore che il file viene gestito automaticamente e che qualsiasi modifica apportata ad esso verrà potenzialmente persa. Può essere interessante far arrivare agli amministratori un messaggio più completo. Fai attenzione, però, la modifica di questa variabile, potrebbe causare il riavvio dei demoni (tramite i gestori associati ai modelli).
- `ssh_args = -C -o ControlMaster=auto -o ControlPersist=300s -o PreferredAuthentications=publickey`: specifica le opzioni di connessione ssh. Disabilitando tutti i metodi di autenticazione diversi dalla chiave pubblica, puoi risparmiare molto tempo. È inoltre possibile aumentare il `ControlPersist` per migliorare le prestazioni (la documentazione suggerisce che un valore equivalente a 30 minuti può essere appropriato). La connessione a un client rimarrà aperta più a lungo e potrà essere riutilizzata quando ci si riconnette allo stesso server, il che rappresenta un notevole risparmio di tempo.

- `control_path_dir`: Specifica il percorso dei socket di connessione. Se questo percorso è troppo lungo, può causare problemi. Considera di cambiarlo in qualcosa di breve, come `/tmp/.cp`.
- `pipelining`: Impostare questo valore a `True` aumenta le prestazioni riducendo il numero di connessioni SSH necessarie quando si eseguono moduli remoti. Devi prima assicurarti che l'opzione `requiretty` sia disabilitata nelle opzioni `sudoers` (vedi documentazione).

10.2 Memorizzazione dei fatti

Raccogliere fatti è un processo che può richiedere un certo tempo. Può essere interessante disabilitare questa raccolta per i playbook che non ne hanno bisogno (tramite l'opzione `collect_facts`) o per mantenere questi fatti in memoria in una cache per un certo periodo di tempo (ad esempio 24H).

Questi fatti possono essere facilmente memorizzati in un database `redis`:

```
sudo yum install redis
sudo systemctl start redis
sudo systemctl enable redis
sudo pip3 install redis
```

Non dimenticate di modificare la configurazione ansibile:

```
fact_caching = redis
fact_caching_timeout = 86400
fact_caching_connection = localhost:6379:0
```

Per controllare il corretto funzionamento, è sufficiente richiedere il server `redis`:

```
redis-cli
127.0.0.1:6379> keys *
127.0.0.1:6379> get ansible_facts_SERVERNAME
```

10.3 Usare Vault

Le varie password e segreti non possono essere memorizzate in un testo in chiaro assieme al codice sorgente Ansible o localmente sul server di gestione o su un possibile gestore di codice sorgente.

Ansible propone di utilizzare un gestore di cifratura: `ansible-vault`.

Il principio è quello di crittografare una variabile o un intero file con il comando `ansible-vault`.

Ansible sarà in grado di decifrare questo file durante l'esecuzione recuperando la chiave di crittografia dal file (ad esempio) `/etc/ansible/ansible.cfg`. Quest'ultimo può anche essere uno script python o altro.

Modifica il file `/etc/ansible/ansible.cfg`:

```
#vault_password_file = /path/to/vault_password_file
vault_password_file = /etc/ansible/vault_pass
```

Memorizza la password in questo file `/etc/ansible/vault_pass` e assegna i diritti restrittivi necessari:

```
mysecretpassword
```

È quindi possibile crittografare i file con il comando:

```
ansible-vault encrypt myfile.yml
```

Un file crittografato `ansible-vault` può essere facilmente riconosciuto dall'intestazione:

```
$ANSIBLE_VAULT;1.1;AES256
353765323436633533306131336638346261363162343239643337353633333961366132663839
6
666432226163326135656638343839373838616533396666 0a34303266323334376263393631363
0
343732301245616637663061346562353862333239643362393366616534336630366333343666
1
643465663030626165 0a313364636261393931313739363931336664386536333766326264633
33
```

```
0
6334
```

Una volta cifrato, un file, esso può ancora essere modificato con il comando:

```
ansible-vault edit myfile.yml
```

È inoltre possibile esportare la tua password di archiviazione in qualsiasi password manager.

Ad esempio, per recuperare una password che dovrebbe essere memorizzata nel rundeck vault:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import urllib.request
import io
import ssl

def get_password():
    '''
    :return: Vault password
    :return_type: str
    '''
    ctx = ssl.create_default_context()
    ctx.check_hostname = False
    ctx.verify_mode = ssl.CERT_NONE

    url = 'https://rundeck.rockylinux.org/api/11/storage/keys/ansible/vault'
    req = urllib.request.Request(url, headers={
                         'Accept': '*/*',
                         'X-Rundeck-Auth-Token': '****token-rundeck****'
                         })
    response = urllib.request.urlopen(req, context=ctx)

    return response.read().decode('utf-8')

if __name__ == '__main__':
    print(get_password())
```

10.4 Lavorare con i server Windows

Sarà necessario installare sul server di gestione parecchi pacchetti:

• Attraverso il gestore dei pacchetti:

```
sudo dnf install python38-devel krb5-devel krb5-libs krb5-workstation
```

e configurare il file `/etc/krb5.conf` per specificare il corretto `realms`:

```
[realms]
ROCKYLINUX.ORG = {
    kdc = dc1.rockylinux.org
    kdc = dc2.rockylinux.org
}
[domain_realm]
  .rockylinux.org = ROCKYLINUX.ORG
```

• Tramite il gestore di pacchetti python:

```
pip3 install pywinrm
pip3 install pywinrm[credssp]
pip3 install kerberos requests-kerberos
```

10.5 Lavorare con i moduli IP

I moduli di rete di solito richiedono il modulo python `netaddr`:

```
sudo pip3 install netaddr
```

10.6 Generazione di un CMDB

Uno strumento, `ansible-cmdb` è stato sviluppato per generare un CMDB da ansibile.

```
pip3 install ansible-cmdb
```

I fatti devono essere esportati con il seguente comando:

```
ansible --become --become-user=root -o -m setup --tree /var/www/ansible/cmdb/
out/
```

Puoi quindi generare un file globale `json`:

```
ansible-cmdb -t json /var/www/ansible/cmdb/out/linux > /var/www/ansible/cmdb/
cmdb-linux.json
```

Se preferisci un'interfaccia web:

```
ansible-cmdb -t html_fancy_split /var/www/ansible/cmdb/out/
```