Ansible

Capitolo 1 – Cos'è Ansible

1.1 Definizione

Ansible è uno strumento di automazione IT open source utilizzato per:

- Configuration Management
- Provisioning di server
- Application Deployment
- Orchestrazione di infrastrutture

È sviluppato da Red Hat ed è oggi uno degli strumenti più utilizzati in ambito sistemistico e DevOps.

A differenza di altre soluzioni (come Puppet o Chef), Ansible è **agentless**: non richiede l'installazione di alcun agente sui nodi gestiti.

1.2 A cosa serve in ambito Sysadmin

In un contesto sistemistico, Ansible permette di:

- Installare e configurare pacchetti su più server contemporaneamente
- Standardizzare configurazioni
- Ridurre errori manuali
- Automatizzare attività ripetitive
- Garantire coerenza tra ambienti (dev, staging, produzione)

Esempio pratico:

Senza Ansible:

- Accesso SSH su 10 server
- Installazione manuale nginx
- Modifica manuale dei file di configurazione
- Riavvio servizio

Con Ansible:

- Un singolo comando
- Configurazione identica su tutti i nodi
- Procedura ripetibile e versionabile

1.3 Architettura di base

Ansible utilizza un'architettura molto semplice.

Control Node

Macchina su cui è installato Ansible e da cui partono i comandi.

Managed Nodes

Server gestiti da Ansible tramite SSH.

Inventory

File che definisce l'elenco dei server da gestire.

Playbook

File YAML che descrive lo stato desiderato dei sistemi.

1.4 Come funziona

Il flusso operativo è il seguente:

1. Ansible legge l'inventory
2. Si connette ai nodi via SSH
3. Esegue i moduli richiesti
4. Riporta l'esito dell'operazione

Ansible lavora in modalità dichiarativa:
non si descrive "come fare", ma "quale deve essere lo stato finale".

Esempio concettuale:

- Stato desiderato: nginx installato
- Se non è installato → viene installato
- Se è già installato → non fa nulla

Questo comportamento si chiama **idempotenza**.

1.5 Perché è scelto in ambienti professionali

Ansible è diffuso in ambienti enterprise per diversi motivi:

- Nessun agente da mantenere
- Semplicità di apprendimento
- Utilizzo di YAML leggibile
- Integrazione con Git
- Ampia libreria di moduli
- Forte integrazione con ecosistema Red Hat

È utilizzato sia per piccole infrastrutture che per ambienti complessi multi-tier.

1.6 Casi d’uso tipici

- Setup iniziale di nuovi server
- Configurazione web server (nginx, apache)
- Gestione utenti e permessi
- Installazione stack LAMP / LEMP
- Hardening di base
- Deploy applicativi
- Aggiornamenti massivi

1.7 Quando NON usare Ansible

Ansible non è pensato per:

- Esecuzioni in tempo reale con latenza millisecondi
- Configurazioni estremamente dinamiche su singolo nodo
- Sostituire sistemi di monitoring

È uno strumento di automazione e gestione configurazione, non un sistema di controllo runtime continuo.

Conclusione

Ansible è uno strumento centrale nel toolkit di un sistemista moderno.

Permette di trasformare operazioni manuali in procedure automatizzate, ripetibili e versionabili, riducendo errori e aumentando la coerenza dell'infrastruttura.

Nel prossimo capitolo entreremo nella configurazione iniziale operativa dopo l’installazione.

Capitolo 2 – Architettura e Componenti di Ansible

Dopo aver compreso cos’è Ansible, è fondamentale capire come è strutturato e quali sono i suoi componenti principali.

Una corretta comprensione dell’architettura è essenziale per utilizzarlo in modo professionale in ambienti server.

2.1 Modello Architetturale

Ansible utilizza un modello **push-based**.

Questo significa che:

- Le configurazioni vengono inviate dal nodo di controllo
- I server remoti non eseguono processi agent in background
- Le operazioni vengono eseguite solo quando richiesto

Questo approccio riduce complessità, consumo risorse e superficie di attacco.

2.2 Control Node

Il **Control Node** è la macchina su cui è installato Ansible.

Caratteristiche:

- Da qui vengono eseguiti comandi e playbook
- Deve avere accesso SSH ai nodi gestiti
- Contiene inventory, playbook, ruoli e configurazioni

Requisiti:

- Linux o macOS (Windows supportato tramite WSL)

- Python installato
- Accesso SSH verso i nodi remoti

In ambienti enterprise il Control Node può essere:

- Una macchina amministrativa
- Un server dedicato all’automazione
- Un runner CI/CD

2.3 Managed Nodes

I **Managed Nodes** sono i server che vengono configurati da Ansible.

Requisiti minimi:

- SSH attivo
- Python installato (preferibilmente Python 3)
- Utente con privilegi sudo

Importante:
Non è richiesto alcun agente Ansible installato sul nodo remoto.

Questo è uno dei principali vantaggi rispetto ad altri strumenti di configuration management.

2.4 Inventory

L’**Inventory** è il file che definisce quali server devono essere gestiti.

Può essere:

- Statico (file ini o yaml)
- Dinamico (generato tramite script o integrazione cloud)

Esempio concettuale:

- Gruppo web
- Gruppo database
- Variabili specifiche per gruppo o host

L’inventory è il punto di partenza di qualsiasi esecuzione.

2.5 Moduli

I **Moduli** sono le unità operative di Ansible.

Sono piccoli programmi che:

- Installano pacchetti
- Gestiscono servizi
- Creano utenti
- Copiano file
- Modificano configurazioni

Quando eseguiamo un comando Ansible:

1. Il modulo viene copiato temporaneamente sul nodo remoto
2. Viene eseguito
3. Restituisce un output JSON
4. Viene rimosso

Questo processo è trasparente per l'amministratore.

2.6 Playbook

I **Playbook** sono file YAML che descrivono lo stato desiderato dell'infrastruttura.

Definiscono:

- Su quali host operare
- Quali task eseguire
- In quale ordine

Un playbook può:

- Installare pacchetti
- Configurare servizi
- Applicare template
- Riavviare demoni

Rappresentano il cuore dell'automazione con Ansible.

2.7 Idempotenza

Concetto fondamentale in ambito sistemistico.

Un’operazione è idempotente quando:

- Può essere eseguita più volte
- Produce sempre lo stesso risultato finale
- Non genera modifiche se lo stato è già corretto

Esempio:

- Se nginx è già installato → nessuna azione
- Se non è installato → viene installato

Questo garantisce:

- Sicurezza nelle riesecuzioni
- Stabilità in produzione
- Coerenza infrastrutturale

2.8 Flusso di Esecuzione

Quando si lancia un playbook:

1. Ansible carica il file di configurazione
2. Legge l’inventory
3. Stabilisce connessioni SSH
4. Esegue i task in ordine
5. Riporta lo stato finale (ok, changed, failed)

L’output fornisce:

- Numero host raggiunti
- Task modificati
- Eventuali errori

Conclusione

Ansible si basa su un’architettura semplice ma estremamente potente:

- Nessun agente
- Comunicazione SSH
- Stato dichiarativo
- Modularità

Comprendere questi elementi è fondamentale prima di passare alla configurazione operativa.

Nel prossimo capitolo entreremo nella configurazione iniziale dell’ambiente di lavoro: inventory, ansible.cfg e struttura progetto.

Capitolo 3 – Configurazione Iniziale dell’Ambiente di Lavoro

Dopo aver compreso architettura e componenti, il passo successivo è impostare correttamente l’ambiente operativo. In ambito sysadmin è fondamentale non lavorare in modo disordinato. Una struttura chiara permette scalabilità, manutenzione e integrazione con Git.

3.1 Non lavorare in /etc/ansible

Ansible può utilizzare:

- /etc/ansible/ansible.cfg
- /etc/ansible/hosts

Tuttavia, in ambienti professionali è sconsigliato lavorare direttamente lì.

Motivi:

- Difficoltà di versionamento
- Configurazione globale poco flessibile
- Problemi in ambienti multi-progetto

Best practice: creare una directory progetto dedicata.

3.2 Creazione Struttura Base Progetto

Esempio struttura iniziale:

```
mkdir -p ~/ansible/{inventory,playbooks,roles,group_vars,host_vars}
cd ~/ansible
```

Struttura consigliata:

```
ansible/
├── ansible.cfg
├── inventory/
│   ├── production
│   ├── staging
│   └── dev
├── playbooks/
├── roles/
├── group_vars/
└── host_vars/
```

Descrizione:

- ansible.cfg → configurazione locale del progetto
- inventory/ → separazione ambienti
- playbooks/ → file operativi
- roles/ → componenti modulari
- group_vars/ → variabili per gruppi
- host_vars/ → variabili per host specifici

Questa struttura è scalabile e adatta a infrastrutture reali.

3.3 Creazione del File ansible.cfg Locale

Creare un file `ansible.cfg` nella root del progetto:

```
[defaults]
inventory = ./inventory/production
remote_user = ansible
host_key_checking = False
retry_files_enabled = False
```

```
roles_path = ./roles
forks = 10
timeout = 30

[privilege_escalation]
become = True
become_method = sudo
become_ask_pass = False
```

Spiegazione Parametri Principali

- inventory
  Percorso dell’inventory di default.
- remote_user
  Utente SSH utilizzato per la connessione.
- host_key_checking
  Se disabilitato evita prompt interattivi (valutare in produzione).
- retry_files_enabled
  Disabilita la creazione di file .retry.
- roles_path
  Percorso locale dei ruoli.
- forks
  Numero di host gestiti in parallelo.
- become
  Abilita automaticamente l’escalation privilegi.

3.4 Ordine di Precedenza del File di Configurazione

Ansible legge il file di configurazione in questo ordine:

1. Variabile ambiente ANSIBLE_CONFIG
2. ansible.cfg nella directory corrente
3. ~/.ansible.cfg
4. /etc/ansible/ansible.cfg

Questo significa che il file locale nel progetto ha priorità rispetto a quello globale.

Verifica configurazione attiva:

```
ansible-config view
```

3.5 Configurazione SSH Corretta

Prerequisito fondamentale: accesso SSH senza password.

Generazione chiave (se non esiste):

```
ssh-keygen -t ed25519
```

Copia chiave sui nodi remoti:

```
ssh-copy-id ansible@server
```

Test connessione:

```
ssh ansible@server
```

Best practice:

- Non usare root diretto
- Creare utente dedicato (es. ansible)
- Consentire sudo controllato

3.6 Verifica Connessione con Ansible

Prima di scrivere playbook, verificare la connettività.

Esempio:

```
ansible all -m ping
```

Output atteso:

- SUCCESS
- Nessun errore SSH
- Nessun errore Python

Se compare errore Python, verificare presenza di python3 sul nodo remoto.

3.7 Preparazione per Ambienti Multipli

Separare ambienti evita errori critici.

Esempio:

```
inventory/
├── production
├── staging
└── dev
```

Esecuzione su staging:

```
ansible-playbook -i inventory/staging playbooks/site.yml
```

Non mescolare mai produzione e ambienti di test nello stesso inventory.

Conclusione

Una corretta configurazione iniziale è fondamentale per:

- Scalabilità
- Sicurezza
- Manutenibilità
- Integrazione con controllo versione

Nel prossimo capitolo entreremo nel dettaglio dell’Inventory: sintassi, gruppi, variabili e struttura professionale.

Capitolo 4 – Inventory: Struttura e Gestione Professionale

L’inventory è il punto di partenza di qualsiasi operazione con Ansible.

Definisce:

- Quali host devono essere gestiti
- Come connettersi agli host
- Come raggrupparli logicamente
- Quali variabili applicare

Una gestione corretta dell’inventory è fondamentale in ambienti sysadmin.

4.1 Inventory Statico (Formato INI)

È il formato più semplice e immediato.

Esempio file `inventory/production`:

```
[web]
web01 ansible_host=192.168.10.10
web02 ansible_host=192.168.10.11

[db]
db01 ansible_host=192.168.10.20

[all:vars]
ansible_user=ansible
ansible_python_interpreter=/usr/bin/python3
```

Spiegazione

- `[web]` → gruppo logico di server
- `web01` → alias host
- `ansible_host` → IP reale o FQDN
- `[all:vars]` → variabili applicate a tutti gli host

Test di connettività:

```
ansible all -m ping
```

4.2 Parametri Host Specifici

È possibile definire parametri direttamente sull'host:

```
web01 ansible_host=192.168.10.10 ansible_port=2222 ansible_user=deploy
```

Parametri comuni:

- ansible_host
- ansible_user
- ansible_port
- ansible_ssh_private_key_file
- ansible_python_interpreter

Questo permette di gestire ambienti eterogenei.

4.3 Inventory in YAML

Alternativa più leggibile in ambienti complessi.

Esempio:

```
all:
  vars:
    ansible_user: ansible
    ansible_python_interpreter: /usr/bin/python3
```

```
  children:
    web:
      hosts:
        web01:
          ansible_host: 192.168.10.10
        web02:
          ansible_host: 192.168.10.11
    db:
      hosts:
        db01:
          ansible_host: 192.168.10.20
```

Vantaggi:

- Struttura gerarchica chiara
- Migliore gestione di ambienti complessi
- Più coerente con il resto dell’ecosistema YAML di Ansible

4.4 Gruppi e Sottogruppi

È possibile creare gruppi annidati.

Esempio:

```
[web]
web01
web02

[frontend]
web01

[backend]
web02

[production:children]
web
db
```

Questo permette di:

- Applicare playbook a gruppi specifici
- Creare logiche di infrastruttura multilivello

Esempio esecuzione su gruppo specifico:

```
ansible web -m command -a "uptime"
```

4.5 File group_vars e host_vars

Separare le variabili dall’inventory è una best practice.

Struttura:

```
group_vars/
├── web.yml
├── db.yml

host_vars/
├── web01.yml
```

Esempio `group_vars/web.yml`:

```
http_port: 80
max_clients: 200
```

Esempio `host_vars/web01.yml`:

```
http_port: 8080
```

Regola importante:

Le variabili host specifiche hanno priorità su quelle di gruppo.

4.6 Separazione Ambienti

Non utilizzare mai un unico inventory per tutto.

Struttura consigliata:

```
inventory/
├── production
├── staging
└── dev
```

Esecuzione su ambiente specifico:

```
ansible-playbook -i inventory/staging playbooks/site.yml
```

Questo evita errori critici su produzione.

4.7 Best Practice Sysadmin

- Non inserire password in chiaro nell’inventory
- Utilizzare ansible-vault per credenziali
- Separare ambienti in file distinti
- Usare alias host leggibili
- Definire sempre ansible_python_interpreter
- Versionare l’inventory con Git (con attenzione ai segreti)

4.8 Verifica Inventory

Comandi utili:

Visualizzare host:

```
ansible-inventory -i inventory/production --list
```

Visualizzare struttura grafica:

```
ansible-inventory -i inventory/production --graph
```

Questi strumenti aiutano nel troubleshooting.

Conclusione

L’inventory non è solo un elenco di server.
È la rappresentazione logica della tua infrastruttura.

Una progettazione corretta permette:

- Scalabilità
- Chiarezza operativa
- Riduzione errori
- Maggiore controllo sugli ambienti

Nel prossimo capitolo entreremo nei Comandi Ad-Hoc e nell’operatività quotidiana da sysadmin.

Capitolo 5 – Comandi Ad-Hoc e Operatività Quotidiana

I comandi ad-hoc permettono di eseguire operazioni rapide sui nodi gestiti senza scrivere un playbook.

Sono utili per:

- Verifiche veloci
- Interventi urgenti
- Troubleshooting
- Operazioni una tantum

Non sostituiscono i playbook in ambienti strutturati, ma sono uno strumento operativo fondamentale per un sysadmin.

5.1 Sintassi Base

Struttura generale:

```
ansible <gruppo_host> -m <modulo> -a "<argomenti>"
```

Esempio:

```
ansible all -m ping
```

Componenti:

- `<gruppo_host>` → gruppo definito nell'inventory
- `-m` → modulo da utilizzare
- `-a` → argomenti del modulo

5.2 Verifica Connettività

Test base su tutti i server:

```
ansible all -m ping
```

Se tutto è configurato correttamente, l'output sarà:

- SUCCESS
- Nessun errore SSH
- Nessun errore Python

5.3 Eseguire Comandi Remoti

Modulo command

Esegue un comando senza passare dalla shell.

```
ansible all -m command -a "uptime"
```

Caratteristiche:

- Non interpreta pipe
- Non interpreta redirect
- Non interpreta variabili shell

È più sicuro rispetto a `shell`.

Modulo shell

Usa la shell del sistema remoto.

```
ansible all -m shell -a "df -h | grep /dev/sda1"
```

Usare solo quando necessario.

Best practice: preferire sempre moduli nativi.

5.4 Gestione Pacchetti

Installare un pacchetto su gruppo web:

```
ansible web -m apt -a "name=nginx state=present update_cache=yes"
```

Rimuovere un pacchetto:

```
ansible web -m apt -a "name=nginx state=absent"
```

Versione generica (cross-distribution):

```
ansible all -m package -a "name=htop state=present"
```

5.5 Gestione Servizi

Avviare un servizio:

```
ansible web -m service -a "name=nginx state=started"
```

Riavviare:

```
ansible web -m service -a "name=nginx state=restarted"
```

Abilitare all’avvio:

```
ansible web -m service -a "name=nginx enabled=yes"
```

5.6 Gestione Utenti

Creare un utente:

```
ansible all -m user -a "name=deploy shell=/bin/bash groups=sudo append=yes"
```

Rimuovere un utente:

```
ansible all -m user -a "name=deploy state=absent remove=yes"
```

5.7 Copiare File

Copiare file locale su remoto:

```
ansible web -m copy -a "src=/tmp/test.txt dest=/tmp/test.txt owner=root group=root mode=0644"
```

5.8 Uso di Privilege Escalation

Se necessario eseguire come root:

```
ansible all -m apt -a "name=vim state=present" --become
```

Se `become` è già abilitato in ansible.cfg, non serve specificarlo.

5.9 Limitare l’Esecuzione

Eseguire solo su un host specifico:

```
ansible web01 -m command -a "hostname"
```

Limitare tramite opzione `--limit`:

```
ansible all -m ping --limit web
```

5.10 Esecuzione Parallela e Fork

Ansible esegue operazioni in parallelo.

Numero default: 5 fork.

Modificabile:

```
ansible all -m ping -f 20
```

Oppure in ansible.cfg:

```
forks = 20
```

5.11 Output e Debug

Aumentare verbosità:

```
ansible all -m ping -vvv
```

Livelli disponibili:

- -v
- -vv
- -vvv
- -vvvv

Utile per troubleshooting SSH o problemi di autenticazione.

5.12 Quando Usare i Comandi Ad-Hoc

Utilizzare quando:

- Serve un controllo rapido
- È richiesta un’azione urgente
- Non è necessario mantenere traccia strutturata

Non utilizzare quando:

- L’operazione deve essere ripetibile
- Serve versionamento
- È una configurazione strutturale

In questi casi è corretto scrivere un playbook.

Conclusione

I comandi ad-hoc sono uno strumento operativo quotidiano per il sysadmin.

Permettono interventi rapidi e controllati, ma non sostituiscono l’automazione strutturata.

Nel prossimo capitolo entreremo nel cuore di Ansible: i Playbook.

Capitolo 6 – Primo Playbook Reale: Setup Base Server

Dopo aver visto i comandi ad-hoc, è il momento di passare ai **playbook**, il cuore dell'automazione con Ansible.

Un playbook permette di definire lo **stato desiderato** dei server in modo dichiarativo e ripetibile.

6.1 Struttura di un Playbook

Un playbook è un file YAML composto da:

- `hosts` → gruppo di host target
- `become` → se abilitare l'escalation privilegi
- `tasks` → elenco di attività da eseguire

Esempio concettuale:

```
- name: Configurazione base server
  hosts: all
  become: true
  tasks:
    - name: Aggiornare cache pacchetti
      apt:
        update_cache: yes
```

6.2 Playbook Base

File: `playbooks/base.yml`

```
- name: Configurazione base server
  hosts: all
  become: true

  tasks:
    - name: Aggiornamento cache apt
      apt:
        update_cache: yes
      when: ansible_os_family == "Debian"

    - name: Installazione pacchetti base
      package:
        name:
          - vim
          - curl
          - htop
          - git
        state: present

    - name: Creazione utente deploy
      user:
        name: deploy
        groups: sudo
        append: yes
        shell: /bin/bash
```

Spiegazione

- `when` → condizione per eseguire il task
- `package` → modulo cross-distribution per installare pacchetti
- `user` → modulo per creare utenti con gruppo e shell

6.3 Esecuzione del Playbook

Esecuzione base:

```
ansible-playbook playbooks/base.yml
```

Modalità di **check** (simulazione senza modifiche):

```
ansible-playbook playbooks/base.yml --check
```

Modalità **verbose**:

```
ansible-playbook playbooks/base.yml -vvv
```

6.4 Best Practice per Playbook

- Non usare shell se esiste modulo nativo
- Testare con `--check` prima di eseguire in produzione
- Versionare i playbook con Git
- Separare ambienti (production, staging, dev)
- Usare variabili per parametri ripetuti
- Creare ruoli per componenti specifici (nginx, mysql, common)

6.5 Prossimi Passi

Dopo aver creato e testato il primo playbook:

- Strutturare progetti con ruoli
- Gestire variabili e segreti con `group_vars`, `host_vars` e `ansible-vault`
- Configurare playbook per deploy applicativi reali
- Implementare hardening base dei server
- Creare playbook modulari e riutilizzabili

Conclusione

Il primo playbook rappresenta il primo passo concreto verso l'automazione.

Da questo momento è possibile standardizzare setup, gestire pacchetti, utenti e configurazioni in modo ripetibile e sicuro.

Nei capitoli successivi approfondiremo la **gestione di ruoli, variabili avanzate e segreti**, e la **strutturazione professionale di progetti Ansible**.

Capitolo 7 – Ruoli, Variabili e Gestione Segreti

Dopo aver imparato i playbook base, è fondamentale organizzare i progetti in modo modulare e sicuro.

7.1 Introduzione ai Ruoli

I **ruoli** permettono di raggruppare tasks, file, template e variabili in moduli riutilizzabili.

Struttura tipica di un ruolo `nginx`:

```
roles/
└── nginx/
    ├── tasks/
    │   └── main.yml
    ├── templates/
    │   └── nginx.conf.j2
    ├── files/
    ├── vars/
    │   └── main.yml
    ├── defaults/
    │   └── main.yml
    └── handlers/
        └── main.yml
```

- `tasks/main.yml` → definisce le azioni principali
- `handlers/main.yml` → definisce azioni come restart servizi
- `templates/` → file Jinja2 parametrizzati
- `vars/` → variabili specifiche di ruolo
- `defaults/` → variabili di default (può essere sovrascritte)

7.2 Creazione di un Ruolo

Esempio:

```
ansible-galaxy init nginx
```

Questo comando genera automaticamente la struttura del ruolo.

7.3 Utilizzo dei Ruoli in un Playbook

Esempio `playbooks/web.yml`:

```
- name: Configurazione Web Server
  hosts: web
  become: true
  roles:
    - nginx
```

Il ruolo viene eseguito con tutte le sue tasks, handlers, template e variabili.

7.4 Variabili

Le variabili possono essere definite in diversi livelli di precedenza:

1. Variabili host specifiche (`host_vars/host1.yml`)
2. Variabili di gruppo (`group_vars/web.yml`)
3. Variabili definite nel ruolo (`vars/main.yml`)
4. Variabili di default del ruolo (`defaults/main.yml`)
5. Variabili inline nel playbook

Esempio `group_vars/web.yml`:

```
http_port: 80
max_clients: 200
```

7.5 Gestione Segreti con Ansible Vault

Ansible Vault permette di cifrare variabili sensibili come password o chiavi.

Creazione file criptato:

```
ansible-vault create group_vars/web/secrets.yml
```

Esempio contenuto:

```
db_password: "SuperSegreta123"
```

Esecuzione playbook con Vault:

```
ansible-playbook playbooks/web.yml --ask-vault-pass
```

Modifica file criptato:

```
ansible-vault edit group_vars/web/secrets.yml
```

7.6 Best Practice Ruoli e Variabili

- Separare configurazioni per ruolo
- Non hardcodare password nei playbook
- Usare nomi di variabili chiari e coerenti
- Riutilizzare ruoli comuni per diversi progetti
- Versionare tutto il progetto con Git (escludendo segreti se necessario)

7.7 Conclusione

I ruoli, le variabili e la gestione dei segreti sono elementi fondamentali per strutturare progetti Ansible professionali.

Permettono:

- Modularità
- Scalabilità
- Sicurezza
- Manutenzione più semplice

Nei prossimi capitoli si può approfondire:

- Template Jinja2
- Deploy applicativi completi
- Hardening e sicurezza avanzata
- Logging e troubleshooting

Capitolo 8 – Deploy Applicativo e Template Jinja2

Dopo aver appreso ruoli, variabili e gestione dei segreti, il passo successivo è imparare a **deployare applicazioni** in modo automatizzato e a utilizzare **template Jinja2** per configurazioni dinamiche.

8.1 Cos’è un Template Jinja2

I template Jinja2 permettono di creare file di configurazione dinamici.

Caratteristiche principali:

- Sintassi semplice simile a Python
- Variabili sostituite al momento dell’esecuzione
- Supporto per cicli, condizioni e filtri
- Ideale per configurazioni come nginx, apache, systemd, ecc.

Esempio minimo di template `nginx.conf.j2`:

```
server {
    listen {{ http_port }};
    server_name {{ server_name }};

    root {{ web_root }};
    index index.html;
}
```

8.2 Creazione di un Template per il Web Server

Esempio ruolo `roles/nginx/templates/nginx.conf.j2`:

```
user www-data;
worker_processes auto;

pid /run/nginx.pid;

events {
    worker_connections 768;
}

http {
    server {
        listen {{ http_port }};
        server_name {{ server_name }};
        root {{ web_root }};
        index index.html index.htm;

        location / {
            try_files $uri $uri/ =404;
        }
    }
}
```

Variabili tipiche da definire in `group_vars/web.yml`:

```
http_port: 80
server_name: example.com
web_root: /var/www/html
```

8.3 Copiare Template con il Modulo `template`

Playbook esempio: `playbooks/deploy_web.yml`

```
- name: Deploy Web Server
  hosts: web
  become: true

  roles:
    - nginx

  tasks:
    - name: Deploy configurazione nginx
      template:
        src: nginx.conf.j2
        dest: /etc/nginx/sites-available/default
        owner: root
        group: root
        mode: '0644'

    - name: Riavvia nginx se configurazione cambiata
      service:
        name: nginx
        state: restarted
      when: ansible_os_family == "Debian"
```

8.4 Gestione Deploy Applicativo Completo

Oltre al web server, un deploy tipico include:

- Creazione cartella applicazione
- Copia codice (modulo `git` o `copy`)
- Gestione dipendenze (es. `pip`, `npm`, `apt`)
- Configurazione variabili ambiente
- Avvio servizi (es. systemd o docker)

Esempio snippet per copia codice:

```
- name: Clona repository applicativo
  git:
```

```
    repo: 'https://github.com/tuo-progetto/app.git'
    dest: /var/www/html
    version: main
```

8.5 Gestione Segreti nel Deploy

Variabili sensibili (API key, password DB) mai hardcodate.

Esempio `group_vars/web/secrets.yml` cifrato con Vault:

```
db_password: "SuperSegreta123"
api_key: "XYZ987654321"
```

Utilizzo nel template:

```
DB_PASSWORD={{ db_password }}
API_KEY={{ api_key }}
```

Esecuzione playbook con Vault:

```
ansible-playbook playbooks/deploy_web.yml --ask-vault-pass
```

8.6 Best Practice per Deploy

- Creare ruoli per ogni componente (nginx, app, db)
- Usare template per tutte le configurazioni modificabili
- Separare variabili di ambiente da variabili di default
- Testare sempre con `--check` prima di eseguire in produzione
- Versionare codice e playbook su Git
- Automatizzare restart servizi solo se necessario (`notify` e `handlers`)

8.7 Conclusione

L’utilizzo di template Jinja2 e playbook modulare permette di:

- Standardizzare deploy
- Rendere configurazioni dinamiche e riutilizzabili
- Gestire segreti in sicurezza
- Automatizzare tutte le operazioni di setup applicativo

Nei prossimi capitoli si può approfondire:

- Hardening e sicurezza server
- Logging e troubleshooting avanzato
- Deploy multi-tier e orchestrazione

Capitolo 9 – Hardening e Sicurezza dei Server

Dopo aver imparato a deployare applicazioni, è fondamentale proteggere i server.

L'**hardening** consiste nell'applicare configurazioni e best practice per ridurre la superficie di attacco e aumentare la sicurezza.

Ansible è uno strumento ideale per automatizzare queste procedure.

9.1 Aggiornamenti di Sicurezza

Eseguire regolarmente aggiornamenti di sicurezza è fondamentale.

Esempio playbook:

```
- name: Aggiornamenti di sicurezza
  hosts: all
  become: true
  tasks:
    - name: Aggiornamento pacchetti Debian
      apt:
        upgrade: dist
      when: ansible_os_family == "Debian"

    - name: Aggiornamento pacchetti RedHat
      yum:
        name: "*"
        state: latest
      when: ansible_os_family == "RedHat"
```

9.2 Gestione Utenti e SSH

- Disabilitare login root diretto via SSH
- Creare utenti con privilegi limitati
- Forzare l’uso di chiavi SSH
- Disabilitare password login quando possibile

Esempio task per SSH:

```
- name: Disabilita root login SSH
  lineinfile:
    path: /etc/ssh/sshd_config
    regexp: '^PermitRootLogin'
    line: 'PermitRootLogin no'
  notify: Restart ssh

- name: Imposta chiavi SSH per utente deploy
  authorized_key:
    user: deploy
    state: present
    key: "{{ lookup('file', '~/.ssh/id_ed25519.pub') }}"
```

Handler per restart SSH:

```
- name: Restart ssh
  service:
    name: ssh
    state: restarted
```

9.3 Firewall e Accesso

Abilitare firewall con regole base:

```
- name: Configura UFW
  ufw:
    state: enabled
    rule: allow
```

```
    port: "{{ item }}"
  loop:
    - 22
    - 80
    - 443
```

- Port 22 → SSH
- Port 80 → HTTP
- Port 443 → HTTPS

9.4 Rimozione Pacchetti Non Necessari

Ridurre il rischio rimuovendo software inutile:

```
- name: Rimuovi pacchetti non necessari
  package:
    name: "{{ item }}"
    state: absent
  loop:
    - telnet
    - ftp
    - rsh-client
```

9.5 Logging e Audit

Abilitare logging e audit per monitorare accessi:

```
- name: Installazione auditd
  package:
    name: auditd
    state: present

- name: Avvia auditd
```

```
  service:
    name: auditd
    state: started
    enabled: yes
```

9.6 Sicurezza dei File di Configurazione

- Impostare permessi restrittivi su file sensibili
- Separare variabili segrete con Ansible Vault

Esempio:

```
- name: Imposta permessi su segreti
  file:
    path: /etc/app/secrets.yml
    owner: root
    group: root
    mode: '0600'
```

9.7 Best Practice Hardening

- Automatizzare tutto con Ansible
- Versionare playbook di sicurezza separatamente
- Testare ogni modifica in staging prima di prod
- Monitorare log di sistema e anomalie
- Aggiornare regolarmente le policy di sicurezza

9.8 Conclusione

Applicare hardening con Ansible permette di:

- Ridurre la superficie di attacco

- Automatizzare best practice di sicurezza
- Mantenere server coerenti e sicuri
- Avere procedure ripetibili e verificabili

Nei prossimi capitoli si può approfondire:

- Logging e monitoring avanzato
- Troubleshooting playbook

Capitolo 10 – Logging, Monitoraggio e Troubleshooting

La gestione dei server non si limita a deploy e configurazione: è fondamentale **monitorare lo stato** e poter **diagnosticare problemi** rapidamente.

Ansible offre strumenti integrati per log e debug, ma è utile combinarli con best practice sysadmin.

10.1 Verbosità e Output

Per ottenere informazioni dettagliate durante l'esecuzione:

```
ansible-playbook playbooks/site.yml -v       # livello base
ansible-playbook playbooks/site.yml -vv      # più dettagli
ansible-playbook playbooks/site.yml -vvv     # debug SSH e task
ansible-playbook playbooks/site.yml -vvvv    # dettagli estremi (output completo)
```

- Utilizzare i livelli più alti solo in fase di debug
- Per grandi infrastrutture, può generare molto output

10.2 Modulo Debug

Il modulo `debug` è fondamentale per verificare variabili o output intermedi.

Esempio:

```
- name: Mostra valore di una variabile
  debug:
    var: http_port
```

```
- name: Mostra messaggio personalizzato
  debug:
    msg: "Deploy completato su {{ inventory_hostname }}"
```

10.3 Check Mode

Permette di simulare l’esecuzione senza modificare i server.

```
ansible-playbook playbooks/base.yml --check
```

Utile per:

- Verificare impatto dei cambiamenti
- Evitare errori in produzione
- Validare logica dei playbook

10.4 Test di Syntax

Prima di eseguire un playbook, controllarne la sintassi:

```
ansible-playbook playbooks/base.yml --syntax-check
```

Permette di:

- Evitare errori di YAML
- Verificare correttezza struttura tasks e ruoli

10.5 Registrare Output

È possibile salvare l’output di un task in una variabile per uso successivo:

```
- name: Controlla spazio disco
  command: df -h /var
  register: disco
```

```
- name: Mostra spazio disco
  debug:
    var: disco.stdout
```

- `register` salva l'output del task
- Permette logica condizionale e troubleshooting

10.6 Gestione Errori

- Task possono essere ignorati con `ignore_errors: yes`
- Condizioni di fallback con `failed_when`
- Retry automatici con `retries` e `delay` (modulo `until`)

Esempio:

```
- name: Esegue comando con retry
  command: /usr/bin/check_service
  register: result
  retries: 3
  delay: 10
  until: result.rc == 0
```

10.7 Log Centralizzato

Per infrastrutture grandi, utile salvare output dei playbook:

```
ansible-playbook playbooks/site.yml | tee /var/log/ansible/site.log
```

- Consente audit e storico modifiche
- Utile per verificare esecuzioni su più server

10.8 Best Practice Troubleshooting

- Testare playbook in staging prima di prod
- Usare `--check` e `--diff` per evitare modifiche indesiderate
- Usare `debug` per variabili e output
- Registrare log per audit e rollback
- Isolare task problematici per esecuzione separata

10.9 Conclusione

Un corretto approccio a logging e troubleshooting permette di:

- Rilevare problemi rapidamente
- Migliorare sicurezza e affidabilità
- Automatizzare interventi di controllo
- Mantenere infrastruttura coerente