Connect

DXC TECHNOLOGY | Red Hat

Red Hat Summit

Connect

Oronzo Santamato

Manager Cloud Architecture - DXC Technology

Mirko Spezie

Cloud Architect - DXC Technology

DXC TECHNOLOGY | Red Hat

Red Hat
Ansible Automation
Platform

DXC Technology in pillole

+ 16.3B$ FY22 revenue

+ 130K employee

+ 70 country

~ 6K customers

Italy Region facts

+ 450M€ FY22 revenue

~ 40K CIs

+ 2K employee

+ 350 new hire (last 18 months)

+ 1K certifications (last 18 months)

DELIVERING EXCELLENCE FOR OUR CUSTOMERS AND COLLEGUES

DXC TECHNOLOGY **IN ITALIA** È ANCHE

Xchanging inspiring innovation · Luxoft A DXC Technology Company · LOGISTICA DIGITALE · FDS A DXC Technology Company

Siti aziendali in Italia

Le sei offerte DXC differenziate

Global Business Services

- **Analytics & Engineering**
- **Applications**
- **Insurance Software & BPS**

In GBS, la nostra offerta:

- Supporta le scelte delle organizzazioni con i dati, l'automazione e l'ingegneria più avanzata
- Aiuta i clienti a ridisegnare il business con applicazioni innovative
- Accelera il business con una vasta gamma di servizi software e di processi aziendali dedicati ad assicurazioni e banche

Global Infrastructure Services

- **Security**
- **Cloud Infrastructure & ITO**
- **Modern Workplace**

In GIS, aiutiamo i clienti a:

- Integrare al meglio i sistemi di sicurezza in tutta l'azienda
- Semplificare e ottimizzare l'IT on-premise e realizzare un ambiente cloud sicuro e ad alte prestazioni per raggiungere risultati aziendali concreti
- Costruire con successo un ambiente di lavoro unico

DXC TECHNOLOGY | Red Hat

Red Hat
Ansible Automation
Platform

Ansible: Un viaggio nell'automazione

- Introduzione ad Ansible
- Organizzazione del codice e dell'ambiente di lavoro
- Molecule Testing
- Processo di sviluppo
- Autenticazione e Sicurezza
- Gestire il ciclo di vita dei servers

DXC TECHNOLOGY | Red Hat

Introduzione ad Ansible

DXC TECHNOLOGY | Red Hat

Red Hat
Ansible Automation
Platform

Ansible - Introduzione

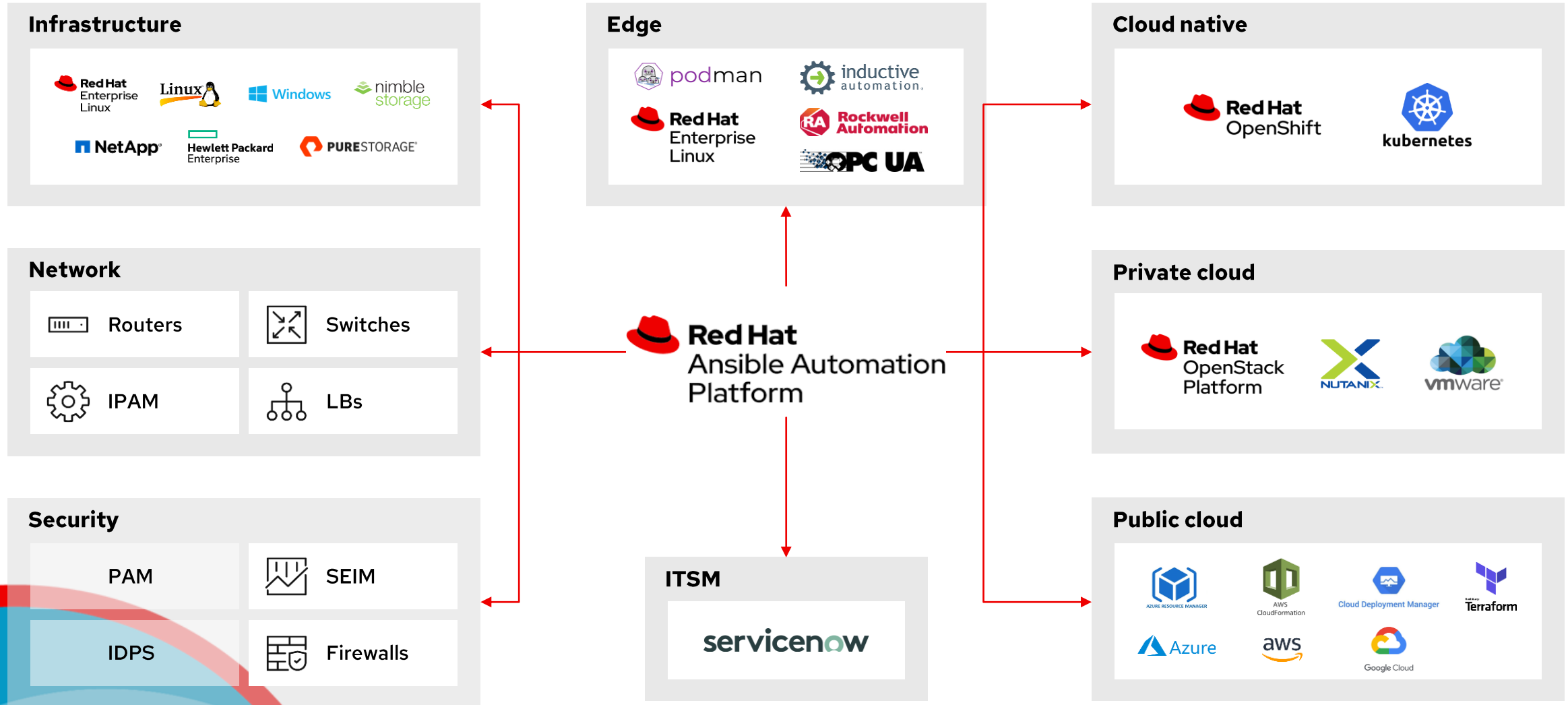
Ansible è uno **strumento di automazione IT** che automatizza il provisioning, la configurazione, la distribuzione delle applicazioni, l'orchestrazione e molti altri processi IT manuali e che permette di descrivere lo stato desiderato di una macchina (o altre componenti) in maniera dichiarativa

```
- hosts: myself
  tasks:
    - name: setup
      set_audio:
        level: 6
    - name: Hello
      say_hi:
        to: all
```

DXC TECHNOLOGY | Red Hat

Red Hat Ansible Automation Platform

Ansible Automation Platform



DXC TECHNOLOGY | Red Hat

Red Hat
Ansible Automation
Platform

Ansible - Playbook

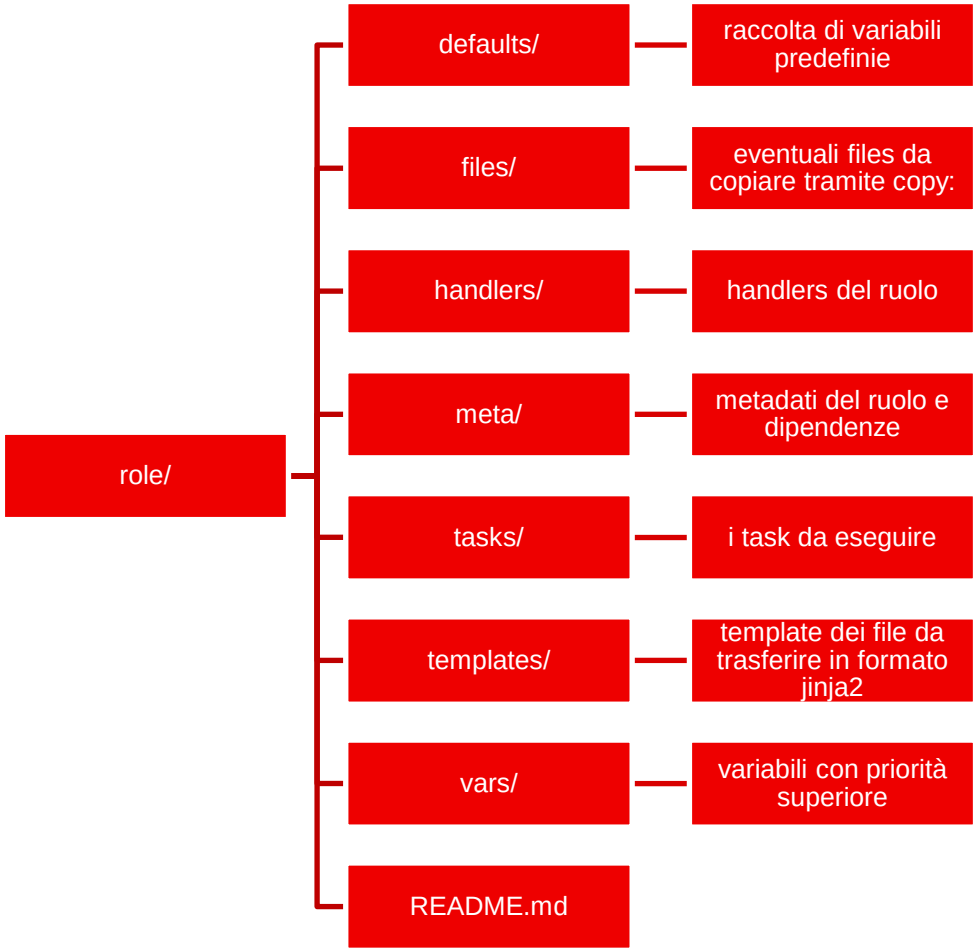
I Playbook permettono di descrivere una serie di azioni che si susseguono fino ad arrivare allo stato desiderato. Ogni **task** deve terminare con esito positivo per poter passare al successivo.

```yaml
- hosts: webservers
  tasks:
    - name: setup
      package:
        name: nginx
        state: present
    - name: Start nginx
      service:
        name: nginx
        state: started
        enabled: true
```

DXC TECHNOLOGY | Red Hat

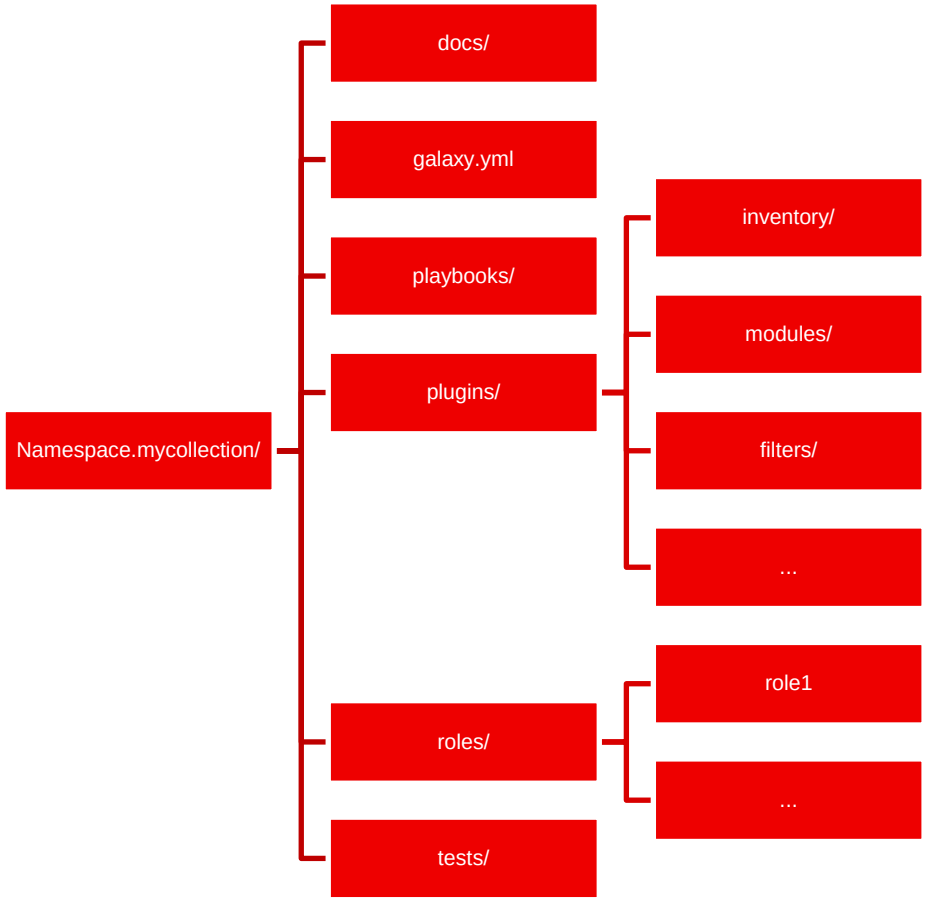
Ansible - Roles

I Ruoli vengono utilizzati per raggruppare una serie di tasks, variabili, templates e files in un contenitore.



Ansible - Collections

Le collections sono un formato di distribuzione e raggruppamento di playbook, ruoli, moduli e plugins.



Red Hat
Ansible Automation
Platform

Ansible –Variabili e precedenza

Le variabili permettono di modificare il comportamento di Ansible e usare valori specifici per determinati gruppi di host.

Attraverso 22 livelli di **precedenza** è possibile alterare il valore in differenti parti del codice.

1. command line values (for example, `-u my_user`, these are not variables)
2. role defaults (defined in role/defaults/main.yml)
3. inventory file or script group vars
4. inventory group_vars/all
5. playbook group_vars/all
6. inventory group_vars/*
7. playbook group_vars/*
8. inventory file or script host vars
9. inventory host_vars/*
10. playbook host_vars/*
11. host facts / cached set_facts
12. play vars
13. play vars_prompt
14. play vars_files
15. role vars (defined in role/vars/main.yml)
16. block vars (only for tasks in block)
17. task vars (only for the task)
18. include_vars
19. set_facts / registered vars
20. role (and include_role) params
21. include params
22. extra vars (for example, `-e "user=my_user"`)(always win precedence)

https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_variables.html#understanding-variable-precedence

DXC TECHNOLOGY | Red Hat

Red Hat
Ansible Automation
Platform

Ansible – Facts

Attraverso le facts vengono recuperate informazioni sul sistema remoto e possono essere usate per rendere il comportamento di Ansibile «intelligente»

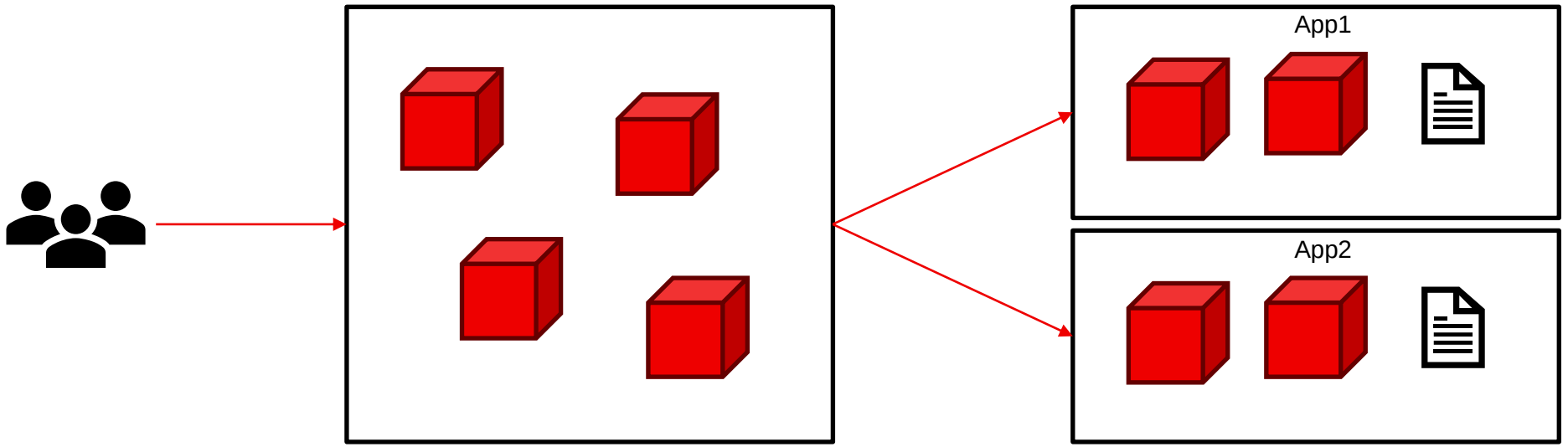
```
- hosts: localhost
  vars:
    ansible_connection: local
  tasks:
    - name: debug var
      debug:
        msg:
          fqdn: "{{ ansible_fqdn }}"
          hostname: "{{ ansible_hostname }}"
          distribution: "{{ ansible_distribution }}"
```

DXC TECHNOLOGY | Red Hat

Organizzazione del codice

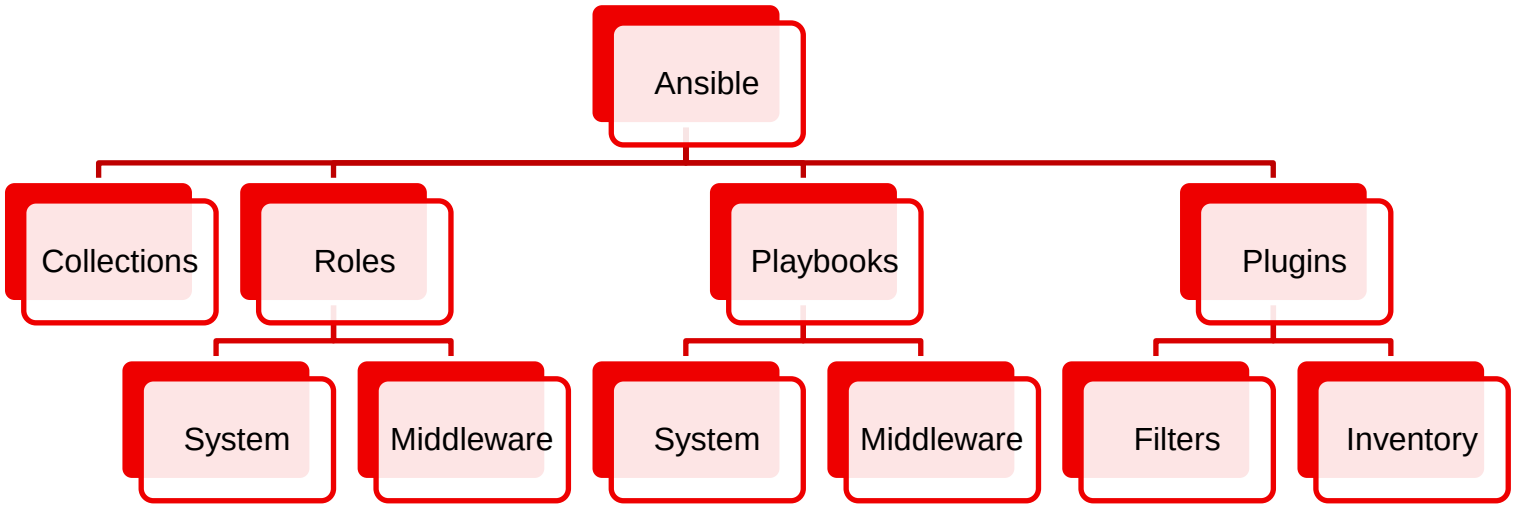
DXC TECHNOLOGY | Red Hat

Il catalogo di automazione

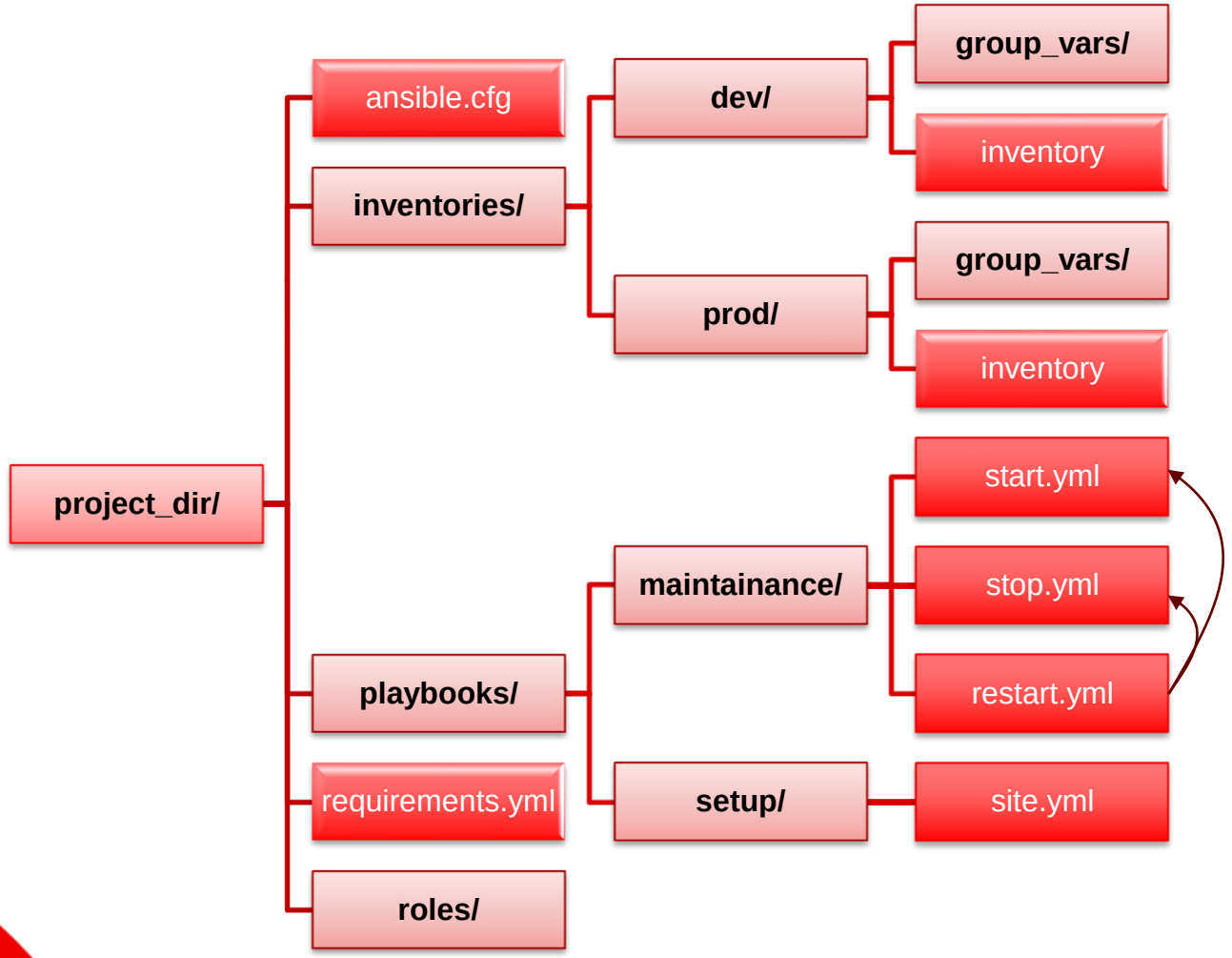


Git Repository

L'organizzazione del codice permette di suddividere le funzionalità e consente uno sviluppo più efficace del codice.



Ansible Project



Red Hat Ansible Automation Platform

Inventory – ini vs yml

```
[frontend]
ws1 ansible_host=10.192.10.1
ws2 ansible_host=10.192.10.2

[backend]
as1 ansible_host=10.192.11.1
as2 ansible_host=10.192.11.2

[frontend:vars]
users='[{"user": "feuser", "uid": "1011"}, {"user":
"feuser2", "uid": "1012"}]'

[all:vars]
ansible_user=ansible
```

```
frontend:
  hosts:
    ws1:
      ansible_host: 10.192.10.1
    ws2:
      ansible_host: 10.192.10.2
  vars:
    users:
      - name: feuser
        uid: 1011
      - name: feuser2
        uid: 1012
backend:
  hosts:
    as1:
       ansible_host: 10.192.11.1
    as2:
      ansible_host: 10.192.11.2
all:
  vars:
    ansible_user: ansible
```

DXC TECHNOLOGY | Red Hat

Requirements.yml

Il file di requirements ci permette di installare **ruoli** e **collections** attraverso galaxy e repository git pubbliche e private

```
# requirements.yml
collections:
  - community.crypto
  - azure.azcollection

roles:
  - name: myrole
    version: 1.0.0
    src: git+https://github.com/example/ansible-role-myrole.git
```

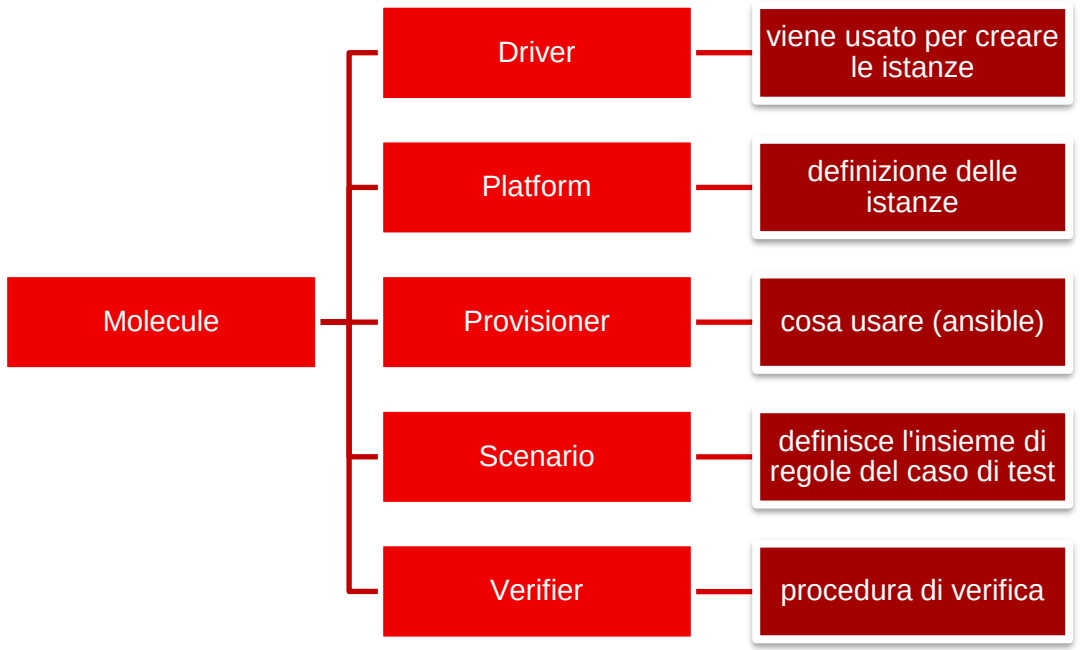
```
$ ansible-galaxy install -r requirements.yml
```

Molecule Testing

DXC TECHNOLOGY | Red Hat

Molecule - Introduzione

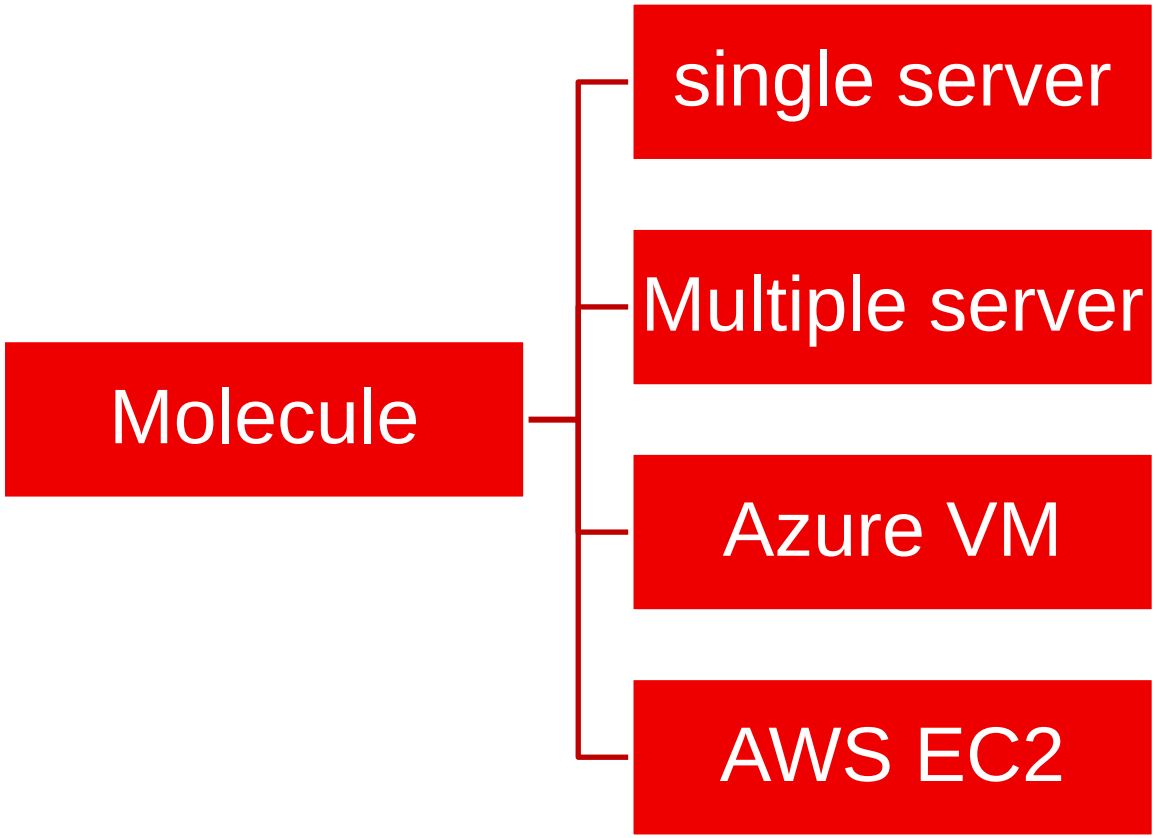
Molecule è un **framework di test** che attraverso una serie di **plugins** permette di creare risorse, lanciare Ansible e verificarne il comportamento.
Verificare il codice con Molecule, consente di eseguire codice sicuro in ambienti di esercizio.



Molecule - Scenari

Attraverso gli scenari si può testare il codice su differenti ambienti, Docker, Azure VM, AWS Ec2, ...
Ogni scenario lancia il ruolo con una determinata configurazione.

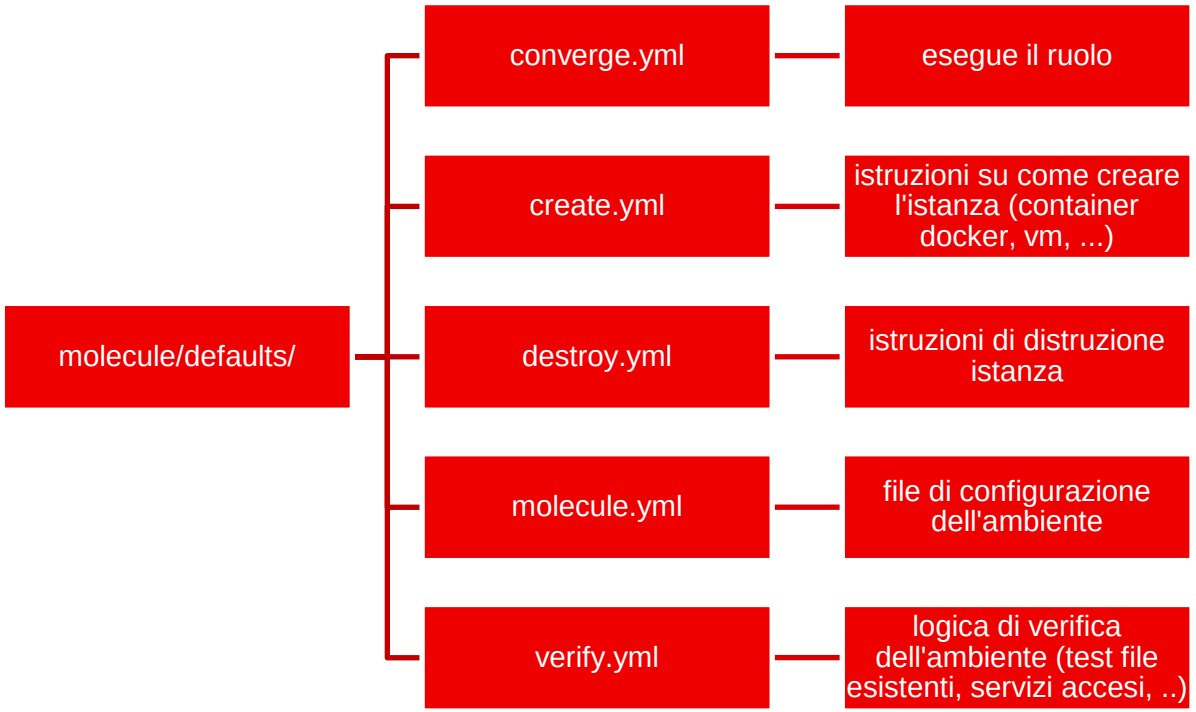
Gli scenari possono essere relativi alla funzionalità o all'ambiente su cui vengono eseguiti.



Molecule - Scenario

Lo scenario contiene molteplici files che descrivono le azioni da compiere e l'ambiente su cui effettuare i tests

- **converge.yml**: logica di esecuzione del ruolo
- **create.yml**: definisce come creare l'ambiente su cui eseguire il ruolo
- **destroy.yml**: istruzioni di distruzione istanza
- **molecule.yml**: configurazione dello scenario molecule
- **verify.yml**: azioni di verifica



Molecule – Scenario docker

```
# molecule.yml
dependency:
  name: galaxy
driver:
  name: docker
platforms:
  - name: instance
    image: registry.access.redhat.com/ubi8/ubi-init:8.8
    tmpfs:
      - /run
      - /tmp
    volumes:
      - /sys/fs/cgroup:/sys/fs/cgroup:ro
    capabilities:
      - SYS_ADMIN
    command: "/usr/sbin/init"
    pre_build_image: true
provisioner:
  name: ansible
verifier:
  name: ansiblec
```

Red Hat Ansible Automation Platform

Molecule – Verify

```yaml
# verify.yml
- name: Verify
  hosts: all
  gather_facts: false
  tasks:
  - name: check nginx socket
    ansible.builtin.uri:
      url: http://localhost:80

  - name: Populate service facts
    ansible.builtin.service_facts:

  - name: check if service is running
    ansible.builtin.assert:
      that:
        - "'nginx.service' in ansible_facts.services"
        - "ansible_facts.services['nginx.service'].state == 'running'"
```

```
(venv) ➜  molecule verify
INFO     default scenario test matrix: verify
…
INFO     Running default > verify
INFO     Running Ansible Verifier
INFO     Sanity checks: 'docker'

PLAY [Verify] ******************************************************************

TASK [check nginx socket] ******************************************************
ok: [instance]

TASK [Populate service facts] **************************************************
ok: [instance]

TASK [check if service is running] *********************************************
ok: [instance] => {
    "changed": false,
    "msg": "All assertions passed"
}

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

INFO     Verifier completed successfully.
```

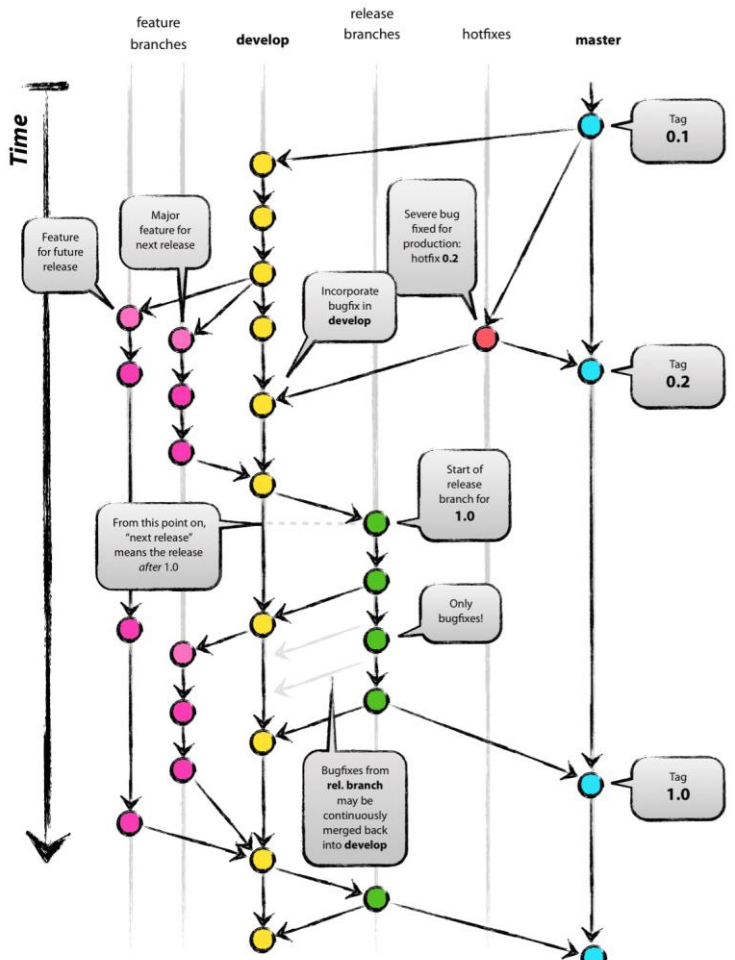
DXC TECHNOLOGY | Red Hat

Processo di Sviluppo

DXC TECHNOLOGY | Red Hat

Git: branching strategy

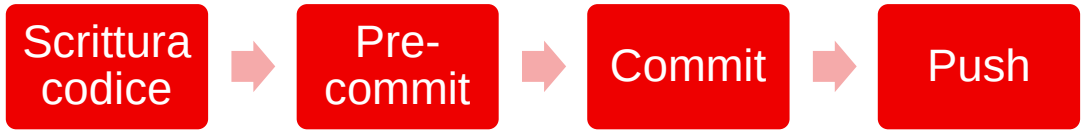
La branching strategy utilizzata per lo sviluppo è paragonabile a quella di qualsiasi altro software, a volte in maniera più o meno complessa a seconda del caso specifico.
L'esempio a sinistra riporta il classico GitFlow con i branch di release



https://nvie.com/posts/a-successful-git-branching-model/

Pre-commit

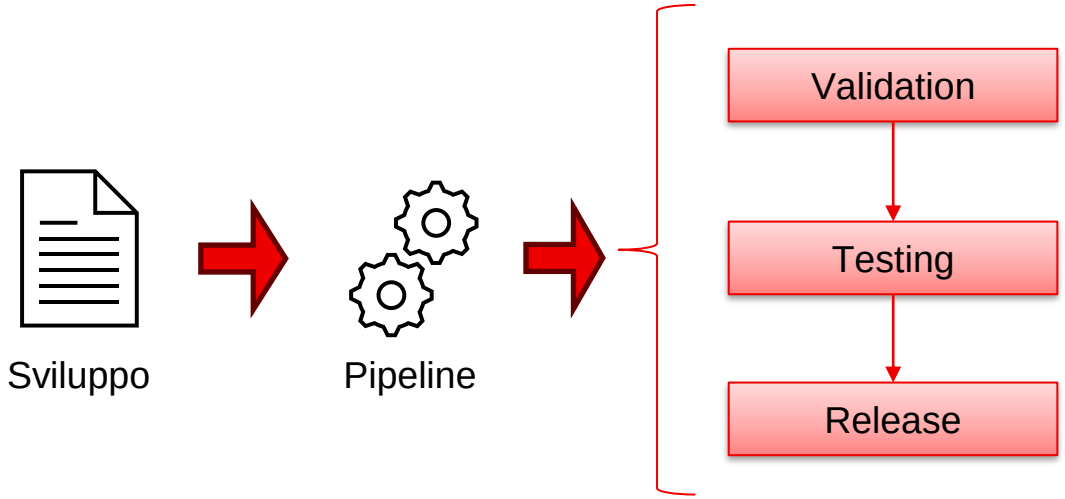
I pre-commit sono uno strumento efficace per lanciare una serie di azioni che verificano e correggono il codice prima di effettuare una git commit (usano githooks).




```yaml
---
repos:
  - repo: https://github.com/pre-commit/pre-commit-hooks
    rev: v2.4.0
    hooks:
      - id: end-of-file-fixer
      - id: trailing-whitespace
      - id: check-yaml
        files: .*\.(yaml|yml)$
  - repo: https://github.com/ansible/ansible-lint.git
    rev: v6.16.0
    hooks:
      - id: ansible-lint
        always_run: true
        verbose: true
        entry: ansible-lint --force-color -v .
```

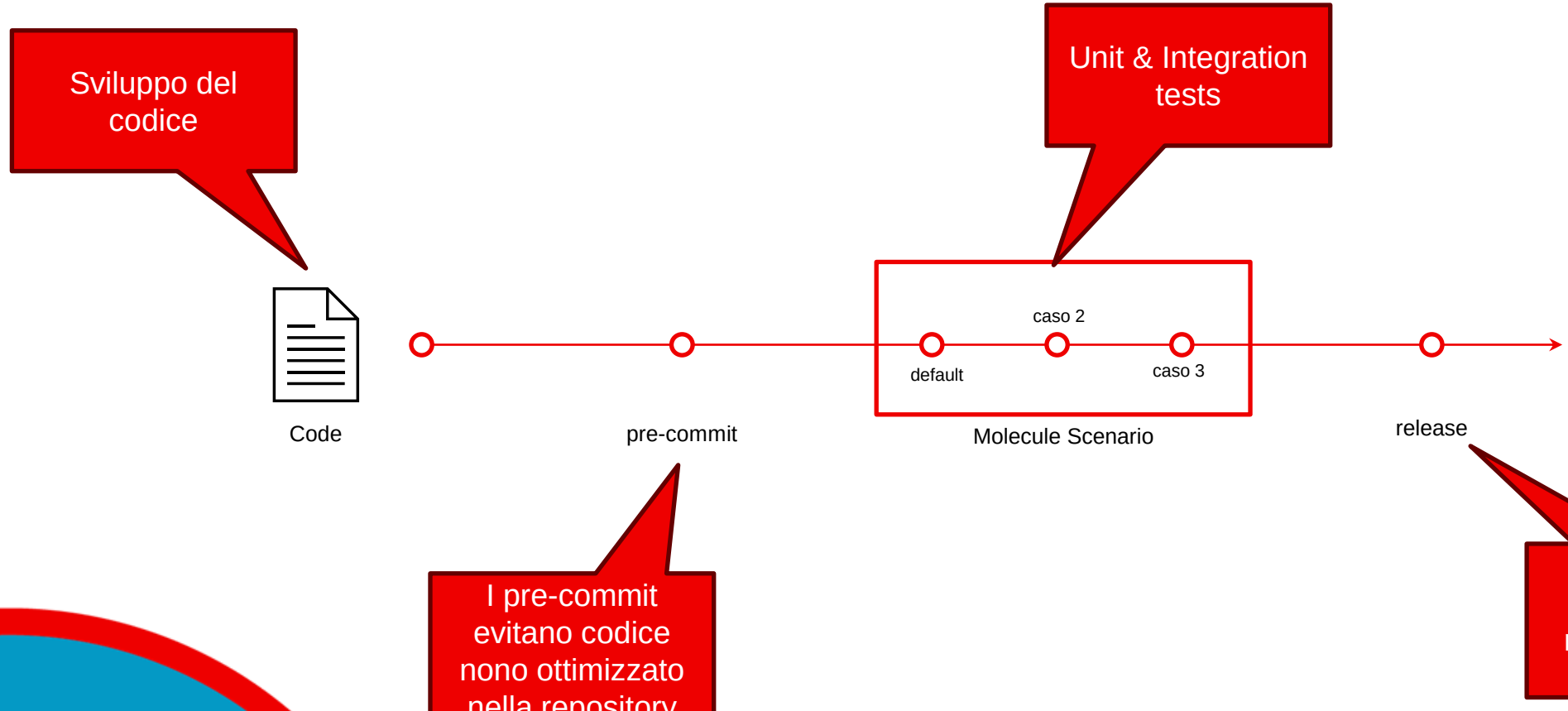
Continuous Testing

Con l'ausilio di piattaforme come Gitlab o Github, possiamo utilizzare meccanismi di «pipeline», ovvero flussi di lavoro che possono essere eseguiti al verificarsi di eventi come push e tags; si può lanciare una serie di azioni come ad esempio il linting, testing etc.
Tutto questo permette un controllo capillare delle versioni di ruoli, collection e playbooks che andiamo a rilasciare nel nostro perimetro di automazione.



Red Hat
Ansible Automation
Platform

Pipeline di Sviluppo



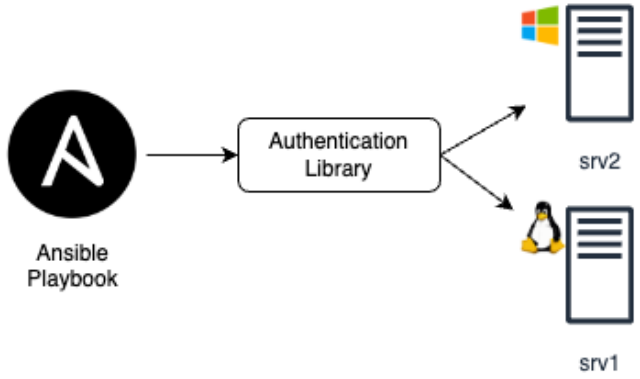
DXC TECHNOLOGY | Red Hat

Autenticazione e Sicurezza

DXC TECHNOLOGY | Red Hat

Autenticazione Trasparente

- La libreria di autenticazione, ovvero una raccolta di tasks, permette di usare lo standard di connessione per i sistemi operativi supportati.
- Una volta che viene eseguito il task è possibile caricare le facts attraverso il modulo `***setup***`




```
- hosts: all
  facts: no
  tasks:
    - name: "prepare | download common tasks"
      delegate_to: localhost
      run_once: true
      ansible.builtin.get_url:
       url: https://gitlab.example/ansible-system-auth/-/raw/v1.0.0/auth_linux.yml
       dest: ./auth.yml
       mode: "0644"

    - name: "prepare | include common tasks"
      ansible.builtin.include_tasks: ./auth.yml
```

Red Hat Ansible Automation Platform

Linux: Autenticazione

```yaml
- hosts: all
  gather_facts: false
  collections:
    - azure.azcollection
    - community.crypto
  tasks:
    - import_tasks: tasks/auth/auth_linux.yml
    - name: gather facts
      setup:

    - name: debug
      debug:
        msg: "hello"
```

```yaml
- name: block
  delegate_to: localhost
  block:
    - name: download keys from keyvault
      run_once: true
      azure_rm_keyvaultsecret_info:
        vault_uri: "https://ansidemokv.vault.azure.net"
        name: ssh-key
      register: _private_key

    - name: copy keys
      run_once: true
      copy:
        content: "{{ (_private_key.secrets | first ).secret }}"
        dest: "{{ playbook_dir }}/keys/id_rsa"

    - name: set fact
      set_fact:
        ansible_ssh_private_key_file: "{{ playbook_dir }}/keys/id_rsa"
```

DXC TECHNOLOGY | Red Hat

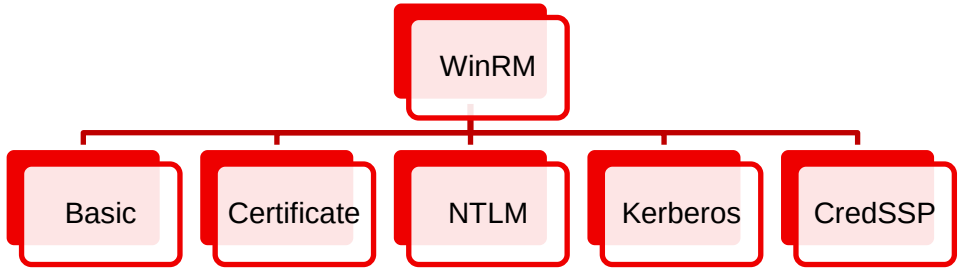
Linux: Sicurezza&Logs

- Creazione di un **utente dedicato** (ansible)
- Configurazione del **sudoers** file
- Restrizione degli accessi tramite **/etc/security/access.conf**
  (aggiungere a /etc/pam.d/sshd required pam_access.so)
  +:ansible:172.16.22.0/27
  -:ansible:ALL
- Usare una logging facility del **syslog** (rsyslog) (ex. local7.* -> /var/log/ansible.log)
  # ansible.cfg:
  [defaults] syslog_facility = LOG_LOCAL7

```
python3[124128]: ansible-setup Invoked with gather_subset=['all'] gather_timeout=10 filter=[] fact_path=/etc/ansible/facts.d
python3[124391]: ansible-ansible.legacy.apt Invoked with name=nginx state=present package=['nginx'] update_cache_retries=5
update_cache_retry_max_delay=12 cache_valid_time=0 purge=False force=False upgrade=no …
```

```
re.compile(r"^.*(ansible-\S+)\sInvoked with\s([\S(?:\s)+]+)")
```

Windows: Autenticazione



Option	Local Accounts	Active Directory Accounts	Credential Delegation	HTTP Encryption
Basic	Yes	No	No	No
Certificate	Yes	No	No	No
Kerberos	No	Yes	Yes	Yes
NTLM	Yes	Yes	No	Yes
CredSSP	Yes	Yes	Yes	Yes

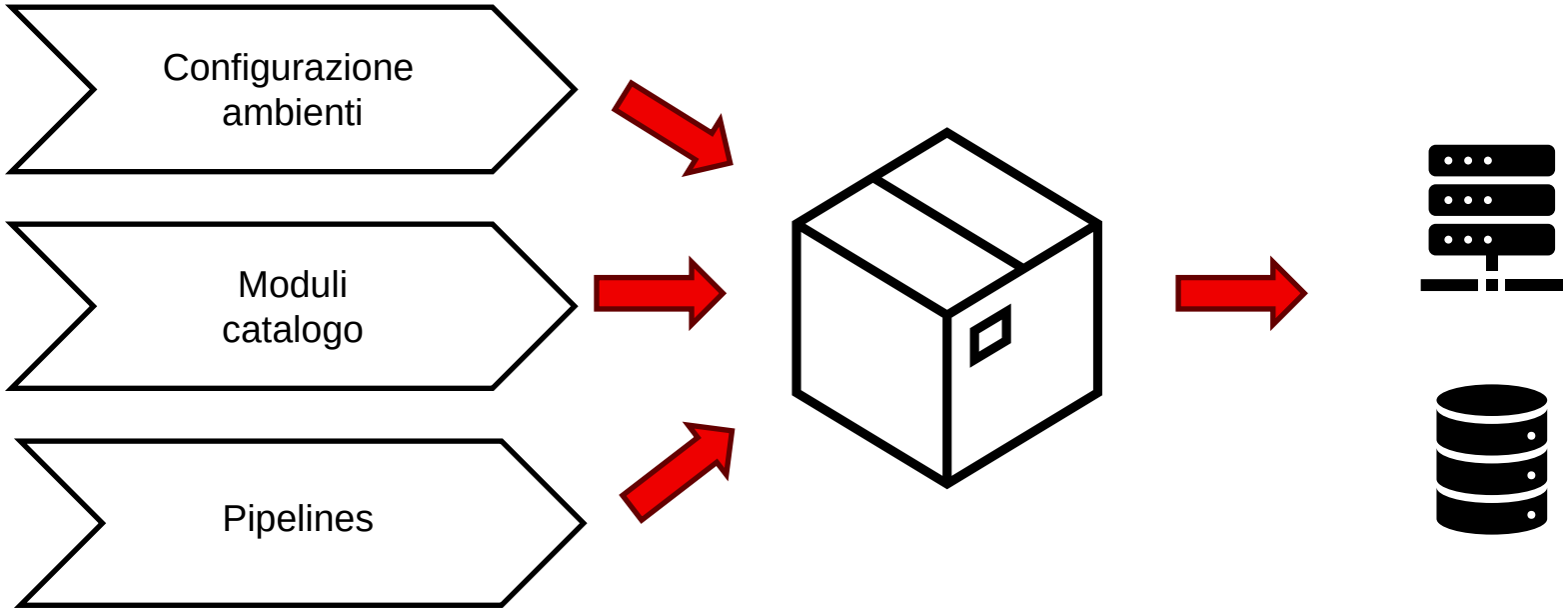
https://docs.ansible.com/ansible/latest/os_guide/windows_winrm.html#winrm-authentication-options

Gestire il ciclo di vita dei servers

DXC TECHNOLOGY | Red Hat

Red Hat
Ansible Automation
Platform

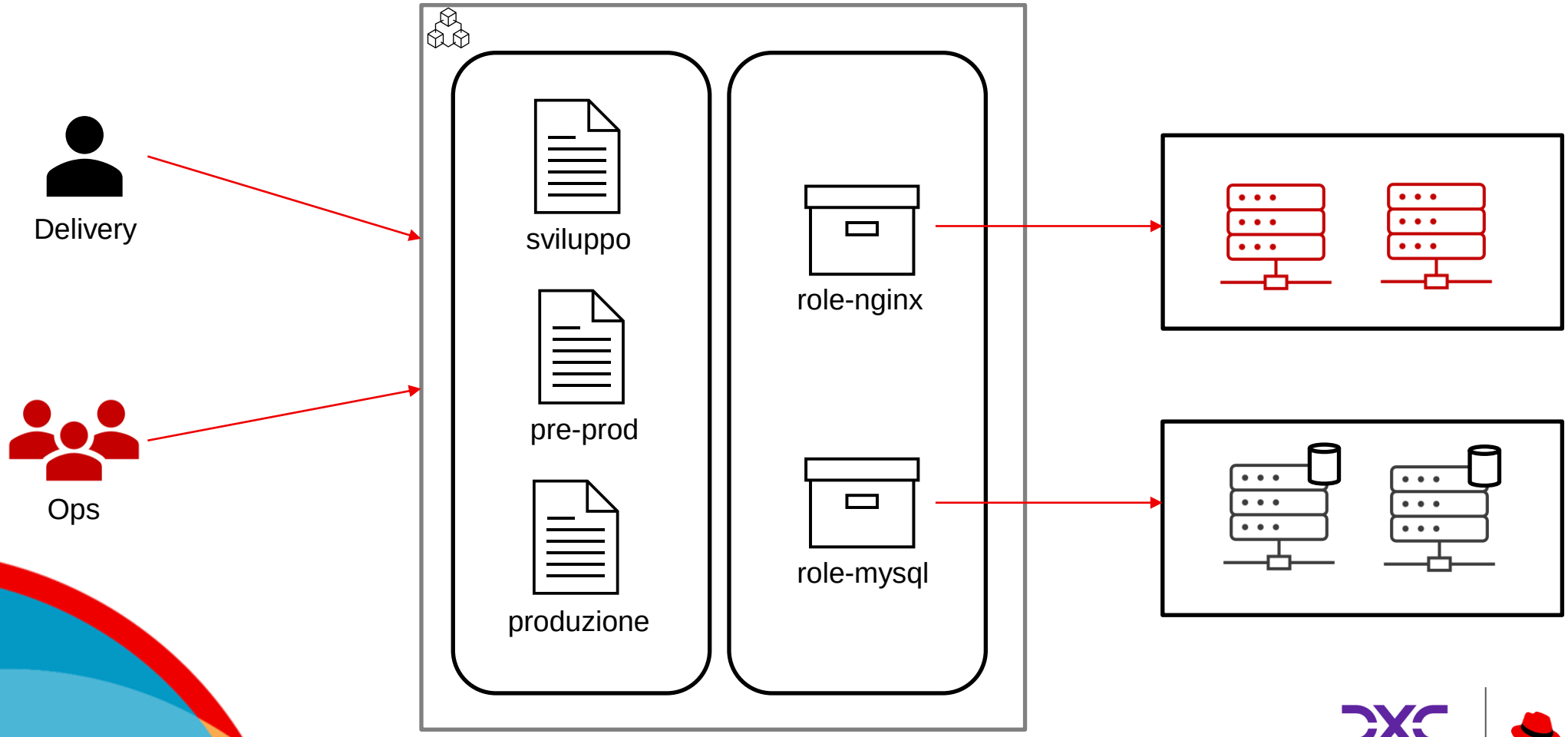
Dal delivery alla produzione



DXC TECHNOLOGY | Red Hat

Red Hat
Ansible Automation
Platform

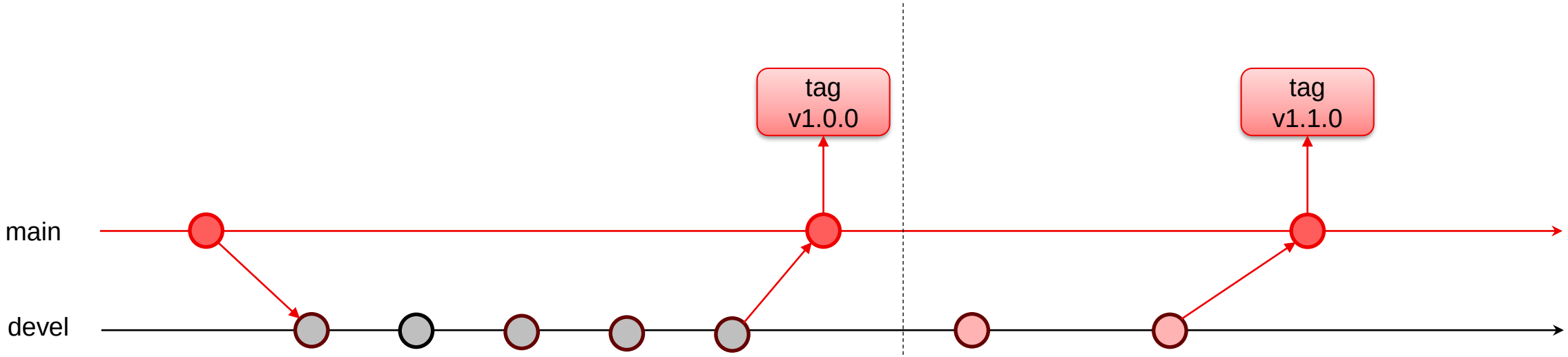
Dal delivery alla produzione



DXC TECHNOLOGY | Red Hat

Red Hat
Ansible Automation
Platform

Evoluzione infrastruttura



Setup infrastruttura

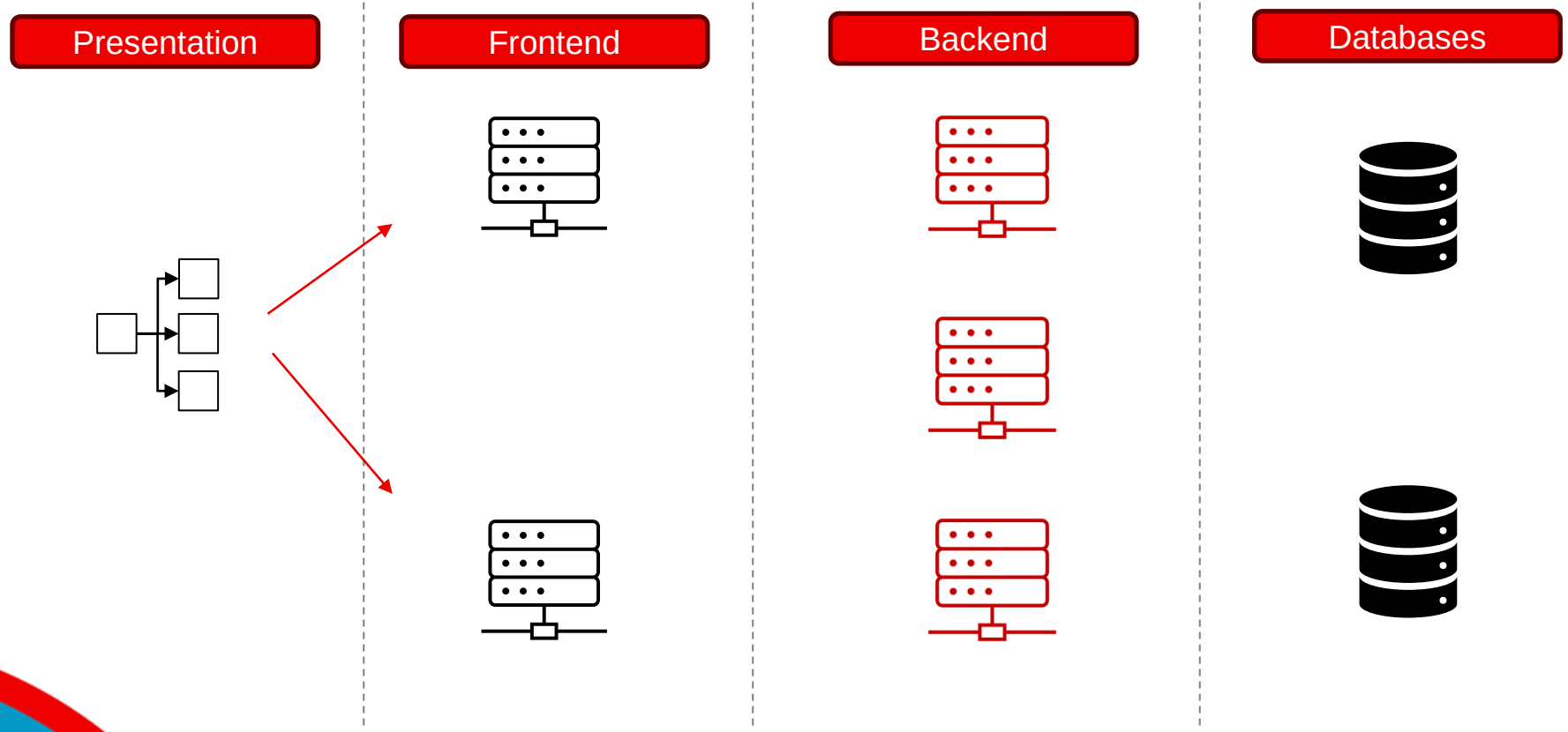
Manutenzione Evolutiva

DXC TECHNOLOGY

Red Hat

Red Hat
Ansible Automation
Platform

Gestione dei servers



DXC TECHNOLOGY | Red Hat

Rolling updates

Attraverso la keyword serial possiamo decidere su quanti server eseguire i comandi in parallelo.

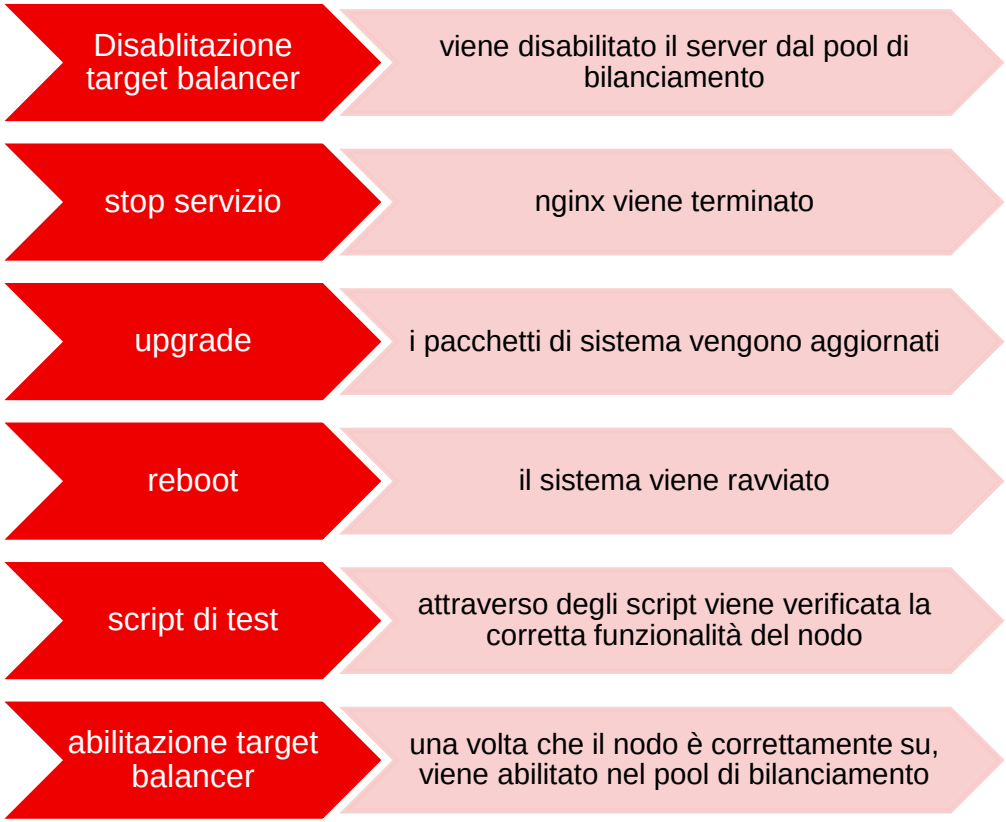
```yaml
- hosts: webservers
  serial: 1
  tasks:
    - name: stop nginx
      service:
        name: nginx
        state: stopped

    - name: upgrade nginx
      package:
        name: nginx
        state: latest

    - name: start nginx
      service:
        name: nginx
        state: started
```

Rolling updates – caso avanzato

Attraverso serial possiamo gestire l'upgrade di una componente isolandola, aggiornandola, verificandola ed infine riaccenderla.



Red Hat Summit

Connect

Q&A?

DXC TECHNOLOGY

Red Hat

Red Hat Summit

Connect

Thank you

DXC TECHNOLOGY

Red Hat