



# **Ansible pour professionnels**

## **Linux / Unix**



Le support du cours «Ansible pour professionnel Linux/Unix » est non contractuel ; il ne doit pas être redistribué et/ou reproduit en partie ou en totalité sans permission explicite et écrite de la société Adlere.

Red Hat, le logo Red Hat, OpenShift et Ansible sont des marques déposées ou commerciales de Red Hat, Inc ou ses filiales aux États-Unis et dans d'autre pays. Linux® est une marque déposée de Linus Torvalds aux États-Unis et dans d'autre pays.

UNIX® est une marque déposée par « The Open Group » aux Etats-Unis et dans d'autres pays.  
Wiindows® est une marque déposée de Microsoft Corporation aux États-Unis et dans d'autre pays.

Les autres marques citées sont déposées par leurs propriétaires respectifs.



# Structure des playbooks





# Playbook de base - cas 1

## Mode local uniquement

mon-playbook.yml

```
---
- name: Mon playbook
  hosts: localhost
  connection: local
  gather_facts: no

  tasks:
  - name: Affiche un message
    ansible.builtin.debug:
      msg: "Message"
```

```
ansible-playbook mon-playbook.yml
```

- Pour une exécution / développement en local, tester des tâches, syntaxes, opérations sur des variables, ...
- Se mettre dans un répertoire de travail dédié, enrichir inventaire et `ansible.cfg` au fur et à mesure des besoins

Mots-clefs: [https://docs.ansible.com/ansible/latest/reference\\_appendices/playbooks\\_keywords.html](https://docs.ansible.com/ansible/latest/reference_appendices/playbooks_keywords.html)





# Playbook de base - cas 2

Avec adressage de cibles

5

```
ansible-playbook -i inventaire [-l limite] mon-playbook.yml
```

1/ inventaire

```
[web]
webserver      ansible-host=192.168.1.1
```

2/ ansible.cfg

```
[defaults]
inventory = ./inventory
remote_user = automation
ask_pass = false
host_key_checking = false

[privilege_escalation]
become = true
become_method = sudo
become_user = root
become_ask_pass = false
```

3/ mon-playbook.yml

```
---
- name: Mon playbook
  hosts: all

  tasks:

    # Ici commence vraiment le playbook commentaire

    - name: Installation apache
      ansible.builtin.dnf:
        name: httpd ≥ 2.4
        state: present
        become: true
        become_user: automation
```

- de multiples endroits où définir les paramètres de connexion ou d'escalade de privilège



- Il est recommandé qu'un projet Ansible suive une structure précise
- Il existe des répertoire conventionnels, où des objets sont recherchés en premier :
  - fichiers à recopier dans `files/`
  - templates jinja2 dans `templates/`
  - fichiers de variables dans `vars/`
  - sinon, dans le répertoire en cours (ie de la tâche)
- Best Practice : toujours documenter dans un fichier `README.md` comment utiliser un playbook (fonctionnement, variables, exemples, ...)

```
mon_projet/
├── ansible.cfg
├── inventory/
│   ├── production.ini
│   └── staging.ini
├── group_vars/
│   └── dev
│       └── vars.yml
│   ├── groupe1.yml
│   └── groupe2.yml
├── handlers/
│   └── services.yml
├── host_vars/
│   ├── hostname1.yml
│   └── hostname2.yml
├── playbooks/
│   ├── site.yml
│   ├── webservers.yml
│   └── dbservers.yml
├── files/
│   └── appStart.sh
├── templates/
│   └── config.j2
├── vars/
└── README.md
```

# Fichier de configuration  
# Répertoire de fichiers d'inventaire  
# Inventaire systèmes de production  
# Inventaire systèmes hors-production  
# Répertoires des variables de groupe

# Variables pour 'groupe1'  
# Variables pour 'groupe2'

# Handlers pour les services  
# Répertoire de variables par système  
# Variables pour 'systeme1'  
# Variables pour 'systeme2'

# Répertoire de playbooks  
# Playbook principal  
# Playbook pour le lot webservers  
# Playbook le lot database  
# Fichiers à recopier

# Fichiers pour template Jinja2

# Fichiers de variables  
# Documentation

```
# Pour toutes les taches
---
- hosts: all
  become: yes
  become_user: sysadm
  tasks:
    - name: Affiche un message
      ansible.builtin.debug:
        msg: Bonjour monde !

# Sur une tâche en particulier
---
- hosts: all
  tasks:
    - name: Affiche un message
      become: yes
      ansible.builtin.debug:
        msg: Bonjour monde !
```

Les directives 'become\_\*' peuvent se positionner au niveau du play ou de la tâche

Best Practice : ne procéder à de l'escalade de privilège que lorsque nécessaire, idéalement au niveau de la tâche

```
# Lancement de commande sous un utilisateur spécifique
---
- hosts: all
  tasks:
    - name: Affiche un message
      become: yes
      become_user: dbadmin
      ansible.builtin.debug:
        msg: Bonjour monde !
```



```
---
- hosts: all
  gather_facts: no
  tasks:
    - name: Exécute une commande spécifique
      ansible.builtin.command:
        cmd: /usr/bin/my_cmd
        changed_when: false
```

```
---
- hosts: all
  gather_facts: no

  tasks:
    - ansible.builtin.command:
        cmd: /usr/bin/false
        register: output
        failed_when: output.rc != 1
```

Directive	Usage
run_once:	run_once: true En général, combiné à delegate_to:
delegate_to:	delegate_to: host05 La tâche sera exécutée sur la machine désignée
changed_when:	Défini la condition pour déterminer qu'une tâche a généré un changement. Sur les tâches 'command' ne faisant que des opérations de récupération d'information, mettre: changed_when: false
failed_when:	Défini la condition pour déterminer qu'une tâche a généré une erreur
ignore_errors:	true pour ne pas s'arrêter en cas d'erreur





# Valeurs par défaut de variables

- une variable non définie génère une erreur d'exécution
  - "The task includes an option with an undefined variable. "
- il est recommandé de fixer une valeur par défaut aux variables
  - | default('VALEUR')
- on peut spécifier qu'une variable est optionnelle
  - | default(omit)

```
---
- hosts: all
  gather_facts: false
  vars:
    user: "psmith"

  tasks:
    - ansible.builtin.user:
      name: "{{ user }}"
      comment: "{{ gecos | default('Human account') }}"
```

```
- name: Création de fichiers, mode optionnel
  ansible.builtin.file:
    dest: "{{ item.path }}"
    state: touch
    mode: "{{ item.mode | default(omit) }}"
  loop:
    - path: /tmp/foo
    - path: /tmp/bar
    - path: /tmp/baz
    mode: "0444"
```



```
$ ansible-playbook main.yml

PLAY [localhost] *****

TASK [Arrête service] *****
changed: [localhost]

TASK [Vérifie service toujours UP]
*****fatal: [localhost]:
FAILED! => {"changed": true, "cmd": ["/opt/appli/check.sh"], "delta": "0:00:00.017622", "end": "2024-04-06 08:19:30.026910", "msg": "non-zero return
code", "rc": 1, "start": "2024-04-06 08:19:30.009288", "stderr": "", "stderr_lines": [], "stdout": "", "stdout_lines": []}
...ignoring

TASK [Récupère données] *****
ok: [localhost]

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

Tâche exécutée, mais n'a généré aucun changement

Tâche non exécutée (probablement à cause d'une condition)

Tâche exécutée, a effectué un changement

Une erreur a été générée

Si nécessaire : mot-clef 'ignore\_errors: true'



```
---
- name: variable playbook test
  hosts: localhost
  connection: local

  tasks:
  - name: Install httpd
    ansible.builtin.dnf:
      name: httpd
      state: latest
      when: ansible_facts['distribution'] == 'RedHat'

  - name: Install apache
    ansible.builtin.apt:
      name: apache2
      state: latest
      when: ansible_facts['distribution'] == 'Debian' or
            ansible_facts['distribution'] == 'Ubuntu'
```

'when' pour conditionner l'exécution d'une tâche  
l'expression doit renvoyer 'true' pour que la tâche soit exécutée

L'exécution d'une tâche est conditionnée par la valeur de la variable.

Pas de 'if - then - else' en natif, il faut reproduire la logique en trouvant les bonnes conditions.

On peut tester que la variable existe déjà :

- `when: (is_appserver is defined) and is_appserver`

[https://docs.ansible.com/ansible/latest/playbook\\_guide/playbooks\\_conditionals.html](https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_conditionals.html)

Symbole	Utilisation
<code>==</code>	Comparaison d'égalité
<code>!=</code>	Comparaison de différence
<code>&gt;</code>	Strictement supérieur à
<code>≥</code>	Supérieur ou égal à
<code>&lt;</code>	Strictement inférieur à
<code>≤</code>	Inférieur ou égal à
<code>&lt;VALUE&gt; in &lt;VARIABLE&gt;</code>	Recherche d'une valeur dans un ensemble

```
---
- hosts: localhost
  connection: local
  gather_facts: no
  vars:
    var1:
      - 42
      - 24

  tasks:
    - ansible.builtin.debug:
        msg: "Valeur 42 détectée !"
        when: 42 in var1
```

Plusieurs conditions peuvent être regroupées par des parenthèses

```
tasks:
- name: Shutdown de systèmes CentOS 8 et Debian 12
  ansible.builtin.command: /sbin/shutdown -t now
  when: (ansible_facts['distribution'] == "CentOS" and ansible_facts['distribution_major_version'] == "8") or
        (ansible_facts['distribution'] == "Debian" and ansible_facts['distribution_major_version'] == "12")
```

Plusieurs conditions liées par un 'et' logique peuvent être combinées dans une liste :

```
tasks:
- name: Shutdown de systèmes CentOS 8
  ansible.builtin.command: /sbin/shutdown -t now
  when:
    - ansible_facts['distribution'] == "CentOS"
    - ansible_facts['distribution_major_version'] == "8"
```



# Condition en se basant sur le précédent résultat

```
---
- name: variable playbook test
  hosts: localhost

  tasks:
  - name: Ensure httpd package is present
    ansible.builtin.dnf:
      name: httpd
      state: latest
      register: httpd_results

  - name: Restart httpd
    ansible.builtin.service:
      name: httpd
      state: restarted
    when: httpd_results.changed
```

La sortie, ou plutôt le résultat de l'installation d'un package est stocké dans une variable (dictionnaire) `httpd_results`

On peut prendre ensuite une action, si la variable indique qu'un changement a eu lieu.





# Capturer le résultat d'une commande

15

```
---
- hosts: localhost
  connection: local
  gather_facts: no

  tasks:
    - ansible.builtin.command:
      cmd: uptime
      register: output

    - ansible.builtin.debug:
      var: output
```

Afficher `output['stdout']` pour avoir juste la sortie de la commande.

Noter :

- xxx.failed (true / false)
- xxx.changed (true / false)
- rc (return code), spécifique à 'command:'
- stderr\_lines ou stdout\_lines

```
PLAY [localhost]
*****
TASK [ansible.builtin.command]
*****
changed: [localhost]

TASK [ansible.builtin.debug]
*****
ok: [localhost] => {
  "output": {
    "changed": true,
    "cmd": [
      "uptime"
    ],
    "delta": "0:00:00.022944",
    "end": "2024-01-15 08:13:17.500079",
    "failed": false,
    "msg": "",
    "rc": 0,
    "start": "2024-01-15 08:13:17.477135",
    "stderr": "",
    "stderr_lines": [],
    "stdout": " 08:13:17 up 119 days, 28 min,  1 user,  load average: 0.27, 0.08, 0.02",
    "stdout_lines": [
      " 08:13:17 up 119 days, 28 min,  1 user,  load average: 0.27, 0.08, 0.02"
    ]
  }
}
```

Enregistrement  
d'une  
notification

Bloc de handlers

```
---
- name: variable playbook test
  hosts: localhost

  tasks:
    - name: Ensure httpd package is present
      ansible.builtin.dnf:
        name: httpd
        state: latest
      notify: restart_httpd

  handlers:
    - name: restart_httpd
      ansible.builtin.service:
        name: httpd
        state: restarted
```

Enchaînement des handlers :

```
notify :
  - restart httpd
  - restart mysqld
```

On peut importer ou inclure  
des handlers

```
handlers:
  - import_tasks: handlers.yml
```

```
tasks:
- name: Ensure httpd package is present
  ansible.builtin.dnf:
    name: httpd
    state: latest
    notify: restart_httpd

- name: Standardized index.html file
  ansible.builtin.copy:
    content: "This is my index.html file for {{ ansible_host }}"
    dest: /var/www/html/index.html
    notify: restart_httpd
```

Si une des tâches génère un évènement **change**, le handler sera exécuté une seule **fois**.

```
TASK [Ensure httpd package is present] *****
ok: [web2]
ok: [web1]          aucun changement

TASK [Standardized index.html file] *****
changed: [web2]
changed: [web1]     changement

NOTIFIED: [restart_httpd] ** *****
changed: [web2]
changed: [web1]     handler exécuté une seule fois
```

```
tasks:
- name: Ensure httpd package is present
  ansible.builtin.dnf:
    name: httpd
    state: latest
    notify: restart_httpd

- name: Standardized index.html file
  ansible.builtin.copy:
    content: "This is my index.html file for {{ ansible_host }}"
    dest: /var/www/html/index.html
    notify: restart_httpd
```

Si les deux tâches génèrent un évènement **change**, le handler sera exécuté une seule **fois**.

```
TASK [Ensure httpd package is present] *****
changed: [web2]
changed: [web1]      changement

TASK [Standardized index.html file] *****
changed: [web2]
changed: [web1]      changement

NOTIFIED: [restart_httpd] *****
changed: [web2]
changed: [web1]      handler exécuté une seule fois
```

```
tasks:
- name: Ensure httpd package is present
  ansible.builtin.dnf:
    name: httpd
    state: latest
  notify: restart_httpd

- name: Standardized index.html file
  ansible.builtin.copy:
    content: "This is my index.html file for {{ ansible_host }}"
    dest: /var/www/html/index.html
  notify: restart_httpd
```

Si aucun **changement** n'est généré, le handler ne sera **PAS** exécuté.

```
TASK [Ensure httpd package is present] *****
ok: [web2]
ok: [web1]      pas de changement
```

```
TASK [Standardized index.html file] *****
ok: [web2]
ok: [web1]      pas de changement
```

```
PLAY RECAP *****
web2      : ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
web1      : ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

**handler n'est pas exécuté**



```
---
- name: Ajout d'utilisateurs
  hosts: node1
  become: yes

  tasks:
    - name: Ajout dev_user
      ansible.builtin.user:
        name: dev_user
        state: present

    - name: Ajout qa_user
      ansible.builtin.user:
        name: qa_user
        state: present

    - name: Ajout prod_user
      ansible.builtin.user:
        name: prod_user
        state: present
```

Depuis ansible 2.5, 'loop' est la méthode recommandée par rapport à 'with\_\*

```
---
- name: Ajout d'utilisateurs
  hosts: node1
  become: yes

  tasks:
    - name: Ajout d'utilisateurs
      ansible.builtin.user:
        name: "{{ item }}"
        state: present
      loop:
        - dev_user
        - qa_user
        - prod_user
```





## Installation multiple de packages

Rappel : inutile de faire des boucles avec dnf / package / apt, qui supportent de travailler avec des listes de nom

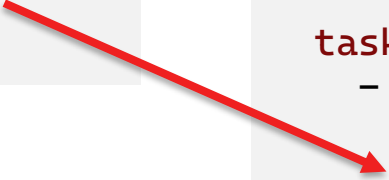
Fichier:packages.yml

```
---
rh_centos_packages:
- git
- make
- tmux
- tree
- rsync
```

Fichier: install\_software.yml

```
---
- name: Variables test
  hosts: localhost
  connection: local
  gather_facts: true

  tasks:
    - name: Install packages for RHEL or CentOS
      ansible.builtin.dnf:
        name: "{{ rh_centos_packages }}"
        state: present
        when: (ansible_facts['distribution'] == 'RedHat') or
              (ansible_facts['distribution'] == 'CentOS')
```



```
ansible-playbook -e@packages.yml -i inventory install_software.yml
```



# Boucle avec incrémentation



- 'with\_sequence' permet de générer une séquence de nombres (attention, type string)
- on peut spécifier une valeur de départ, d'arrivée, et d'intervalle

```
---
- hosts: localhost
  connection: local
  gather_facts: true
  vars:
    num_messages: 10

  tasks:
    - ansible.builtin.debug:
        msg: "This is a debug message: {{ item }}"
        with_sequence: 'count={{ num_messages }}
```

```
- name: Compte à rebours
  ansible.builtin.debug:
    msg: "{{item}} secondes avant mise à feu"
    with_sequence: start=10 end=0 stride=-1
```

```
- name: Avec une variable
  ansible.builtin.debug:
    msg: "{{ item }}"
    with_sequence: start=1 end="{{ end_at }}"
  vars:
    - end_at: 10
```

[https://docs.ansible.com/ansible/latest/collections/ansible/builtin/sequence\\_lookup.html](https://docs.ansible.com/ansible/latest/collections/ansible/builtin/sequence_lookup.html)



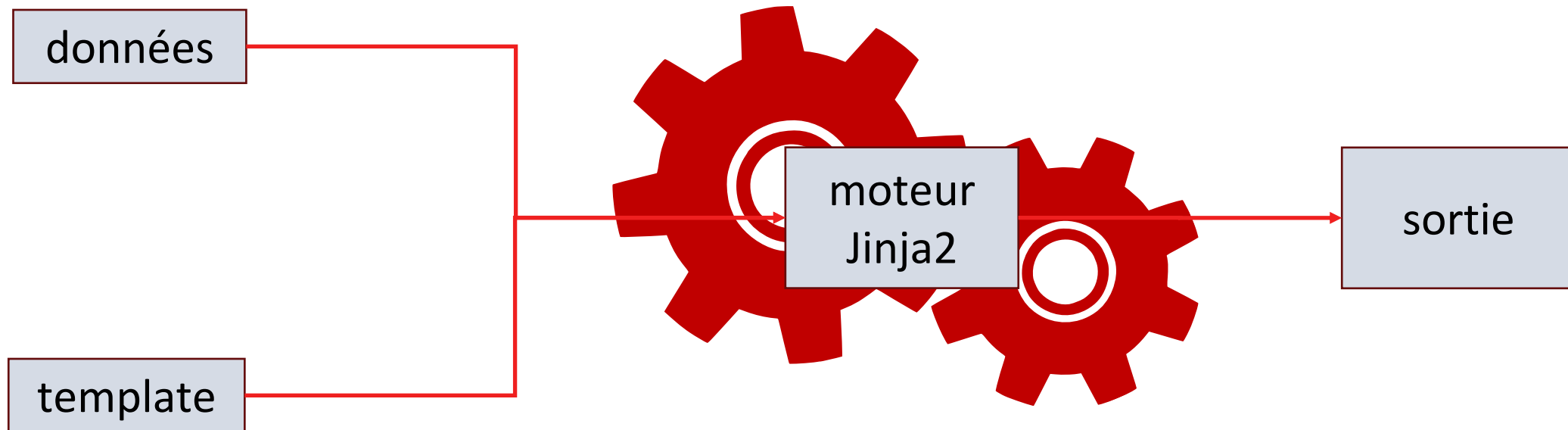
- directive `until` pour ré-essayer une tâche jusqu'à ce qu'une condition soit remplie

```
- name: Boucle sur une tâche
  ansible.builtin.shell: /usr/bin/commande
  register: result
  until: result.stdout.find("all systems go") != -1
  retries: 5
  delay: 10
```

- `-vv` au lancement pour afficher dans le résultat final de la tâche les différents résultats obtenus

```
fatal: [localhost]: FAILED! => {"attempts": 5, "changed": true, "cmd": "uptime",
"delta": "0:00:00.027618", "end": "2024-02-03 14:30:14.167866", "msg": "", "rc": 0,
"start": "2024-02-03 14:30:14.140248", "stderr": "", "stderr_lines": [], "stdout":
" 14:30:14 up 138 days,  6:45,  1 user,  load average: 0.35, 0.16, 0.05",
"stdout_lines": [" 14:30:14 up 138 days,  6:45,  1 user,  load average: 0.35, 0.16,
0.05"]}
```

<https://ttl255.com/jinja2-tutorial-part-1-introduction-and-variable-substitution/>  
<https://ttl255.com/jinja2-tutorial-part-2-loops-and-conditionals/>



<https://jinja.palletsprojects.com/en/3.1.x/>

datas.yml

```
---
packages:
  - git
  - tmux
  - tree

os_name: Red Hat
```

jinja2-demo.yml

```
---
- name: Test jinja2
  hosts: localhost
  connection: local

  tasks:
    - name: Genere le fichier
      ansible.builtin.template:
        src: modele.j2
        dest: /tmp/output.txt
```

modele.j2

```
{{ ansible_managed }}
Voici la liste des packages :

{% for my_item in packages %}
  {{ loop.index }} {{ my_item }}
{% endfor %}

OS destination :  {{ os_name }}

Ansible fact:  {{ ansible_facts['distribution'] }}
```

ansible.cfg

```
[defaults]
ansible_managed = Gere par ANSIBLE
```

```
ansible-playbook -e@datas.yml -i inventory jinja2.yml
```

## playbook.yml

```
---
- name: Test jinja2
  hosts: all

  tasks:
    - name: Genere le fichier
      ansible.builtin.template:
        src: etc-hosts.j2
        dest: /tmp/output.txt
```

## /tmp/output

```
192.168.1.1 server01.example.com server01
192.168.1.2 server02.example.com server02
192.168.1.3 server03.example.com server03
```

## etc-hosts.j2

```
{% for host in groups['all'] %}
{{ hostvars[host]['ansible_facts']['default_ipv4']['address'] }} {{ hostvars[host]['ansible_facts']['fqdn'] }} {{
hostvars[host]['ansible_facts']['hostname'] }}
{% endfor %}
```

### Structure de contrôle

```
{% if kenny.sick %}
    Kenny is sick.
{% elif kenny.dead %}
    You killed Kenny!
{% else %}
    Kenny looks okay --- so far
{% endif %}
```

### Structure de contrôle avec filtre

```
{% if myvar | int is odd %}
    Variable impaire
{% elif myvar | int is even %}
    Variable paire
{% endif %}
```

### Générer du texte de remplissage

```
{{ lipsum(n=5, html=false, min=20, max=100) }}
```



```
tasks:
- name: Install, configure, and start Apache
  block:
    - name: Install httpd and memcached
      ansible.builtin.dnf:
        name:
          - httpd
          - memcached
        state: present

    - name: Apply the foo config template
      ansible.builtin.template:
        src: templates/src.j2
        dest: /etc/foo.conf

    - name: Start service bar and enable it
      ansible.builtin.service:
        name: bar
        state: started
        enabled: true
  when: ansible_facts['distribution'] == 'CentOS'
  become: true
  become_user: root
  ignore_errors: true
```

- Permet de regrouper des tâches en entité logique
- Toutes les tâches d'un bloc héritent des directives du bloc
- Celles-ci s'appliquent aux tâches, pas au bloc lui-même
- pas de boucles
- apparus en Ansible 2.0

## tasks:

- name: Attempt and graceful roll back demo

### block:

- name: Print a message  
  ansible.builtin.debug:  
    msg: 'I execute normally'
- name: Force a failure  
  ansible.builtin.command: /bin/false
- name: Never print this  
  ansible.builtin.debug:  
    msg: 'I never execute, due to the above task failing, :-'

('

### rescue:

- name: Print when errors  
  ansible.builtin.debug:  
    msg: 'I caught an error'
- name: Force a failure in middle of recovery! >:-)  
  ansible.builtin.command: /bin/false
- name: Never print this  
  ansible.builtin.debug:  
    msg: 'I also never execute :-('


### always:

- name: Always do this  
  ansible.builtin.debug:  
    msg: "This always executes"

- Permettent d'introduire des mécanismes de gestion des erreurs
- On peut forcer l'exécution des handlers avec :  
  meta: flush\_handlers

```
tasks:
  - name: Attempt and graceful roll back demo
    block:
      - name: Print a message
        ansible.builtin.debug:
          msg: 'I execute normally'
          changed_when: true
          notify: Run me even after an error

      - name: Force a failure
        ansible.builtin.command: /bin/false
    rescue:
      - name: Make sure all handlers run
        meta: flush_handlers
    handlers:
      - name: Run me even after an error
        ansible.builtin.debug:
          msg: 'This handler runs even on error'
```



- D'une façon générale, les IA savent produire des playbooks ou morceaux de playbooks exploitables
- La qualité des réponses est variable, on a déjà vu des références à des modules ou filtres inexistant
- **Attention à la confidentialité des données**

ChatGPT	OpenAI	<a href="https://chatgpt.com">https://chatgpt.com</a>
Claude AI	Anthropic	<a href="https://claude.ai">https://claude.ai</a>
Gemini	Google	<a href="https://gemini.google.com/">https://gemini.google.com/</a>
Le Chat	Mistral	<a href="https://chat.mistral.ai">https://chat.mistral.ai</a>
LLaMa	Meta/Facebook	<a href="https://www.meta.ai/">https://www.meta.ai/</a> - pas encore accessible en France
Bing Copilot	Microsoft	<a href="https://copilot.microsoft.com/">https://copilot.microsoft.com/</a>
Grok2	Twitter / X	Depuis l'application mobile, pour les abonnés X Premium/premium +
llama3-70b	nvidia	<a href="https://build.nvidia.com/meta/llama3-70b">https://build.nvidia.com/meta/llama3-70b</a>



Clipart d'illustration conçu avec Microsoft Designer®

« Globalement, elle [l'IA] me donne un début de solution mais je repasse toujours derrière »  
Mon collègue



# Exemples de prompt IA

30

Jeu 1 !

- peux-tu me faire un playbook ansible qui affiche "hello world" et qui s'exécute sur localhost ?

```
---
- name: Hello World Playbook
  hosts: localhost
  connection: local
  gather_facts: no

  tasks:
    - name: Afficher Hello World
      debug:
        msg: "Hello World"
```

```
- name: Hello World local
  hosts: localhost
  tasks:
    - name: Display Hello World
      debug:
        msg: "Hello world!"
```

```
---
- hosts: localhost
  tasks:
    - debug:
        msg: "Hello World"
```

```
- hosts: localhost
  gather_facts: false
  tasks:
    - name: Afficher "Hello World"
      debug:
        msg: "Hello World"
```

- Principale remarque porte sur le respect de best practices (FQCN, utilisation de 'name')
- Seul #1 rajoute le mot-clef 'connection: local', plutôt recommandé quand la cible est localhost
- gather\_facts à false ou no est un ajout utile ('false' recommandé vs 'no' pour des raisons de style)



# Exemples de prompt IA

31

## Jeu 2 !

- peux-tu me faire une tâche ansible qui affiche l'adresse IP du système géré ?

```
---
- name: Afficher l'adresse IP du système
  hosts: all
  gather_facts: yes

  tasks:
    - name: Afficher l'adresse IPv4 principale
      debug:
        msg: "L'adresse IPv4 principale de {{ inventory_hostname }} est {{ ansible_default_ipv4.address }}"

    - name: Afficher toutes les adresses IPv4
      debug:
        msg: "Toutes les adresses IPv4 de {{ inventory_hostname }} sont : {{ ansible_all_ipv4_addresses }}"

    - name: Afficher l'adresse IPv6 principale (si disponible)
      debug:
        msg: "L'adresse IPv6 principale de {{ inventory_hostname }} est {{ ansible_default_ipv6.address }}"
      when: ansible_default_ipv6.address is defined
```

```
- name: Afficher toutes les adresses IP
  hosts: localhost
  gather_facts: false

  tasks:
    - name: Récupérer les informations réseau
      ansible.netcommon.network_facts:
        register: network_facts

    - name: Afficher chaque adresse IP
      loop: "{{ network_facts.ansible_all_ipv4_addresses }}"
      debug:
        msg: "Adresse IP : {{ item }}"
```

```
---
- hosts: localhost
  tasks:
    - name: Get IP address
      command: ip addr show
      register: ip_output

    - name: Display IP address
      debug:
        msg: "{{ ip_output.stdout }}"
```

```
- hosts: all
  gather_facts: true
  tasks:
    - name: Afficher l'adresse IP du système
      debug:
        msg: "L'adresse IP est {{ ansible_default_ipv4.address }}"
```





**Merci**







<https://yaml-multiline.info/>

There are two types of formats that YAML supports for strings: *block scalar* and *flow scalar* formats. (*Scalars* are what YAML calls basic values like numbers or strings, as opposed to complex types like arrays or objects.) Block scalars have more control over how they are interpreted, whereas flow scalars have more limited escaping support.

## Block Scalars

A block scalar header has three parts:

**Block Style Indicator:** The *block style* indicates how newlines inside the block should behave. If you would like them to be kept as newlines, use the **literal** style, indicated by a pipe ( `|` ). If instead you want them to be replaced by spaces, use the **folded** style, indicated by a right angle bracket ( `>` ). (To get a newline using the folded style, leave a blank line by putting *two* newlines in. Lines with extra indentation are also not folded.)

**Block Chomping Indicator:** The *chomping indicator* controls what should happen with newlines at the *end* of the string. The default, **clip**, puts a single newline at the end of the string. To remove all newlines, **strip** them by putting a minus sign ( `-` ) after the style indicator. Both clip and strip ignore how many newlines are actually at the end of the block; to **keep** them all put a plus sign ( `+` ) after the style indicator.

**Indentation Indicator:** Ordinarily, the number of spaces you're using to indent a block will be automatically guessed from its first line. You may need a *block indentation indicator* if the first line of the block starts with extra spaces. In this case, simply put the number of spaces used for indentation (between 1 and 9) at the end of the header.