Support pédagogique Ansible avancé

Le programme de la formation

- **Ansible**
  - Présentation de Ansible
  - Les concepts clés et le versionning
  - Quelques éléments de vocabulaire
  - Les ressources
- Setup SSH
  - Accès SSH sur les serveurs distants

Le programme de la formation

- **Setup Ansible**
  - Installation d’Ansible
  - Les fichiers Inventaire
  - Les inventaires dynamiques
  - Fichier de configuration
- Ansible en mode ad hoc
  - La documentation avec ansible-doc
  - Cas pratique avec l'installation de Apache
  - YAML - Yet Another Markup Language

Le programme de la formation

- **Ansible Playbooks**
  - Vérification de la syntaxe des playbooks avec ansible-lint
  - Gestion des blocs
- Les variables et leurs hiérarchies
- Le chiffrement des fichiers
- Le moteur de template Jinja
- Les Templates Jinja
- Exécuter des tâches à plusieurs niveaux

Le programme de la formation

- Exécution conditionnée
- Les tags
- Les handlers
- Les Rôles
- Les collections
- Utilisation de Docker
- Création de module
- Développement de plugin
- Optimisation des performances
- Installation de MediaWiki
- Visite de l'interface AWX

Déroulement de la formation

Durée : 2 jours

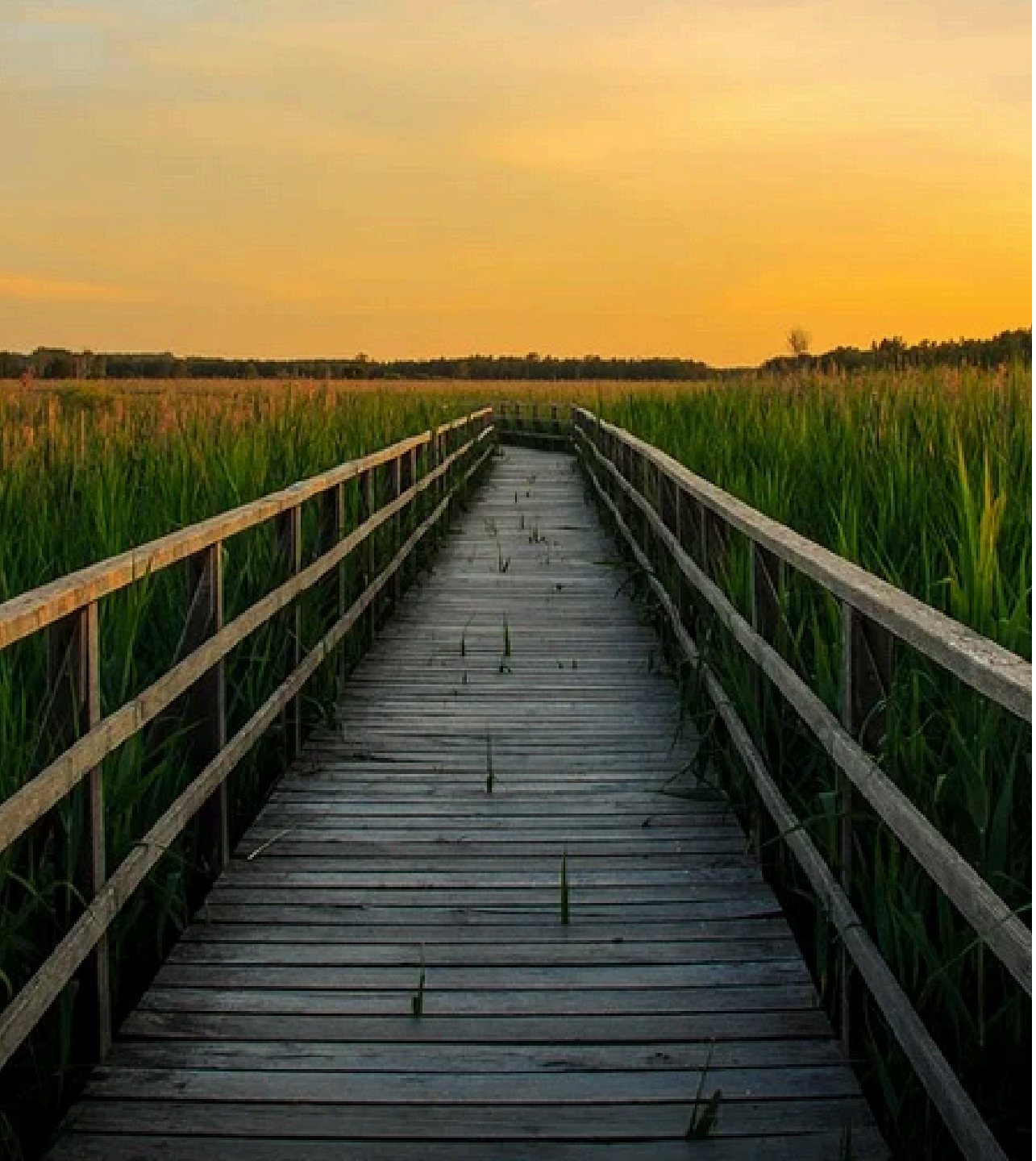
--

Horaires : 09H00 - 12H00 / 14H00 -17H00

--

Pauses : 10H30 / 15h30

Faisons CONNAISSANCE



Présentation de Ansible

Présentation de Ansible

Michael DeHaan

- Créateur de Ansible en **2012**
- Support et sponsor Ansible par **AnsibleWorks Inc.** en **2013**
- Achat par RedHat en **2015**
- puis **IBM** en **2018**
- **5,400+** Community Contributors
- **1,600+** Ansible Modules
- Version **10.0.1** (juin 2024)
- Licence publique générale **GNU** version **3** ou ultérieure

- Une version majeure approximativement tous les deux mois
- **Agentless** (OpenSSH)
- **Idempotence** : une opération a le même effet qu'on l'applique une ou plusieurs fois
- Format de données en `Json`
- Langage de programmation objet en `Python`
- Format de représentation de données en `YAML`
- Intègre des modules pour des composants matériels (notamment pour piloter du matériel réseau)

Ansible est un outil d'**automatisation** informatique. Il peut configurer des systèmes, déployer des logiciels et orchestrer des tâches informatiques avancées telles que des déploiements continus.

- **Contexte et Développement**:
  - **Michael DeHaan** a commencé à développer Ansible après avoir travaillé sur d'autres projets open source comme **Cobbler**.
  - **Frustré** par la **complexité** des outils existants comme **Chef** et **Puppet**, il voulait créer une solution plus **simple** et **rapide** pour l'**automatisation** des configurations et des déploiements.
  - L'idée d'Ansible était de créer un **outil basé sur SSH** sans nécessiter d'**agents de gestion**, simplifiant ainsi grandement le processus

- **Philosophie et Design**:
  - Ansible a été **conçu** pour être **facile** à utiliser et **accessible**.
  - La **simplicité** était un objectif clé dès le début, ce qui a permis à Ansible de se **démarquer** rapidement dans la communauté de l'**open source**.
  - **DeHaan** a mis un accent particulier sur la **documentation**, estimant que les utilisateurs devaient pouvoir **réussir à utiliser l'outil en moins de 30 minutes**

Pourquoi Ansible ?

- **Éliminer** les répétitions et simplifier les flux de travail
- **Gérer** et **maintenir** la configuration du système
- **Déployer** en continu des logiciels complexes
- **Effectuer** des mises à jour continues sans interruption

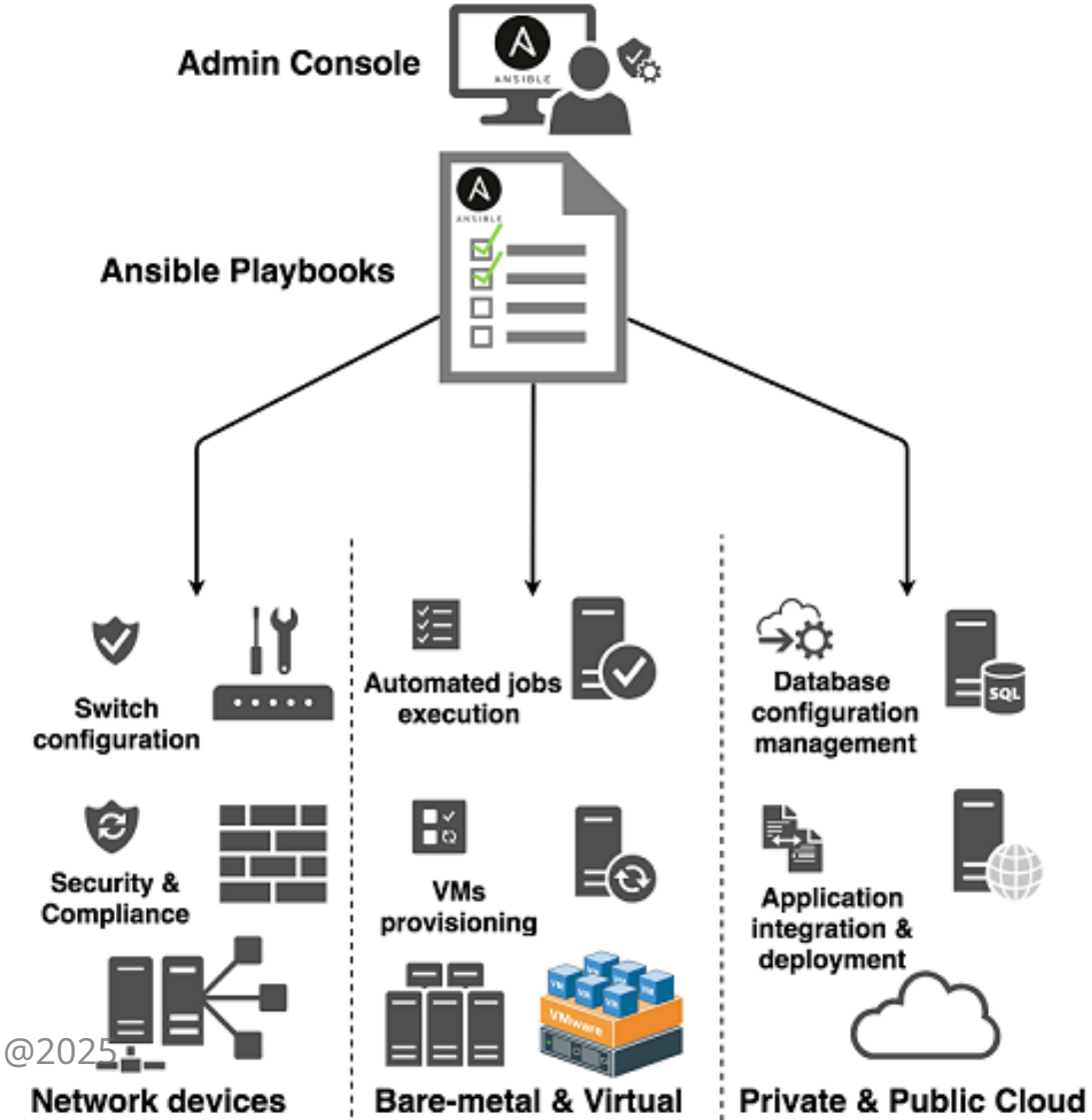
Principes d'Ansible

- **Architecture sans agent** : Faible maintenance, évite l'installation de logiciels supplémentaires dans toute l'infrastructure informatique.
- **Simplicité** : Syntaxe YAML simple, Les playbooks d'automatisation utilisent une syntaxe YAML pour un code qui se lit comme une documentation.
- **Décentralisé** : Utilise SSH et les identifiants existants pour accéder aux machines distantes.

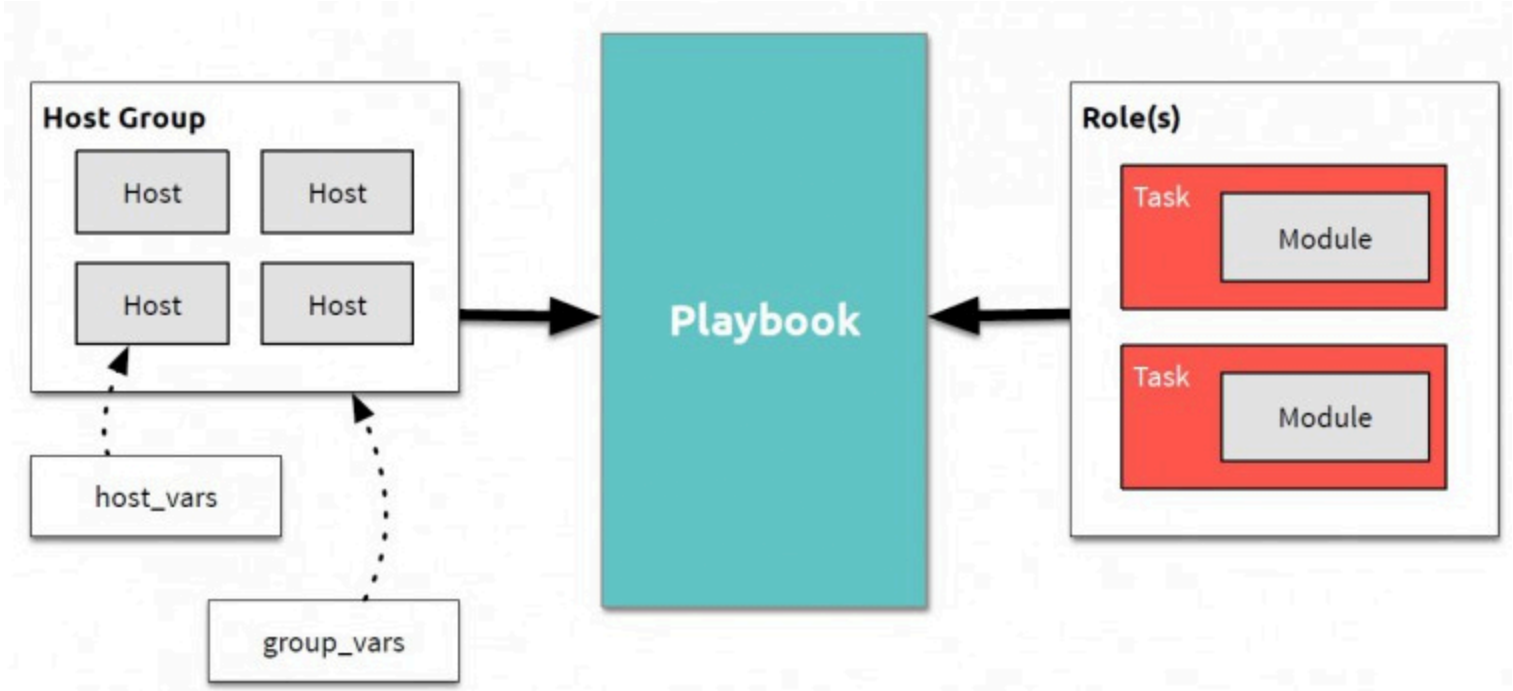
Principes d'Ansible

- **Scalabilité et flexibilité** : Conception modulaire, prend en charge un large éventail de systèmes d'exploitation, de plateformes cloud et de dispositifs réseau, facilitant ainsi l'évolutivité des systèmes que vous automatisez.
- **Idempotence et prévisibilité** : État cohérent, lorsque le système est dans l'état décrit par votre playbook, Ansible ne change rien, même si le playbook est exécuté plusieurs fois.

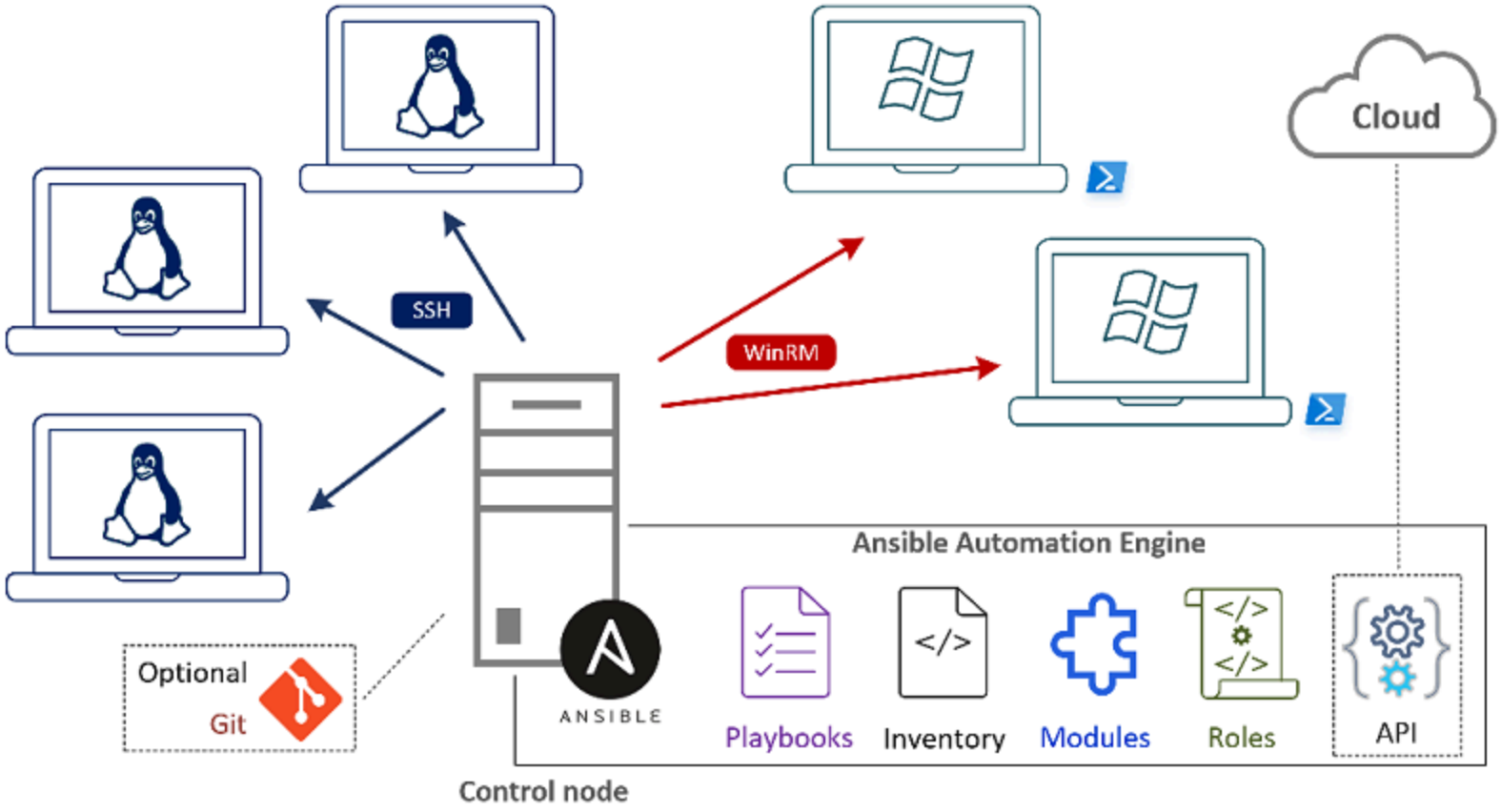
Les concepts clés



Les concepts clés



Les concepts clés



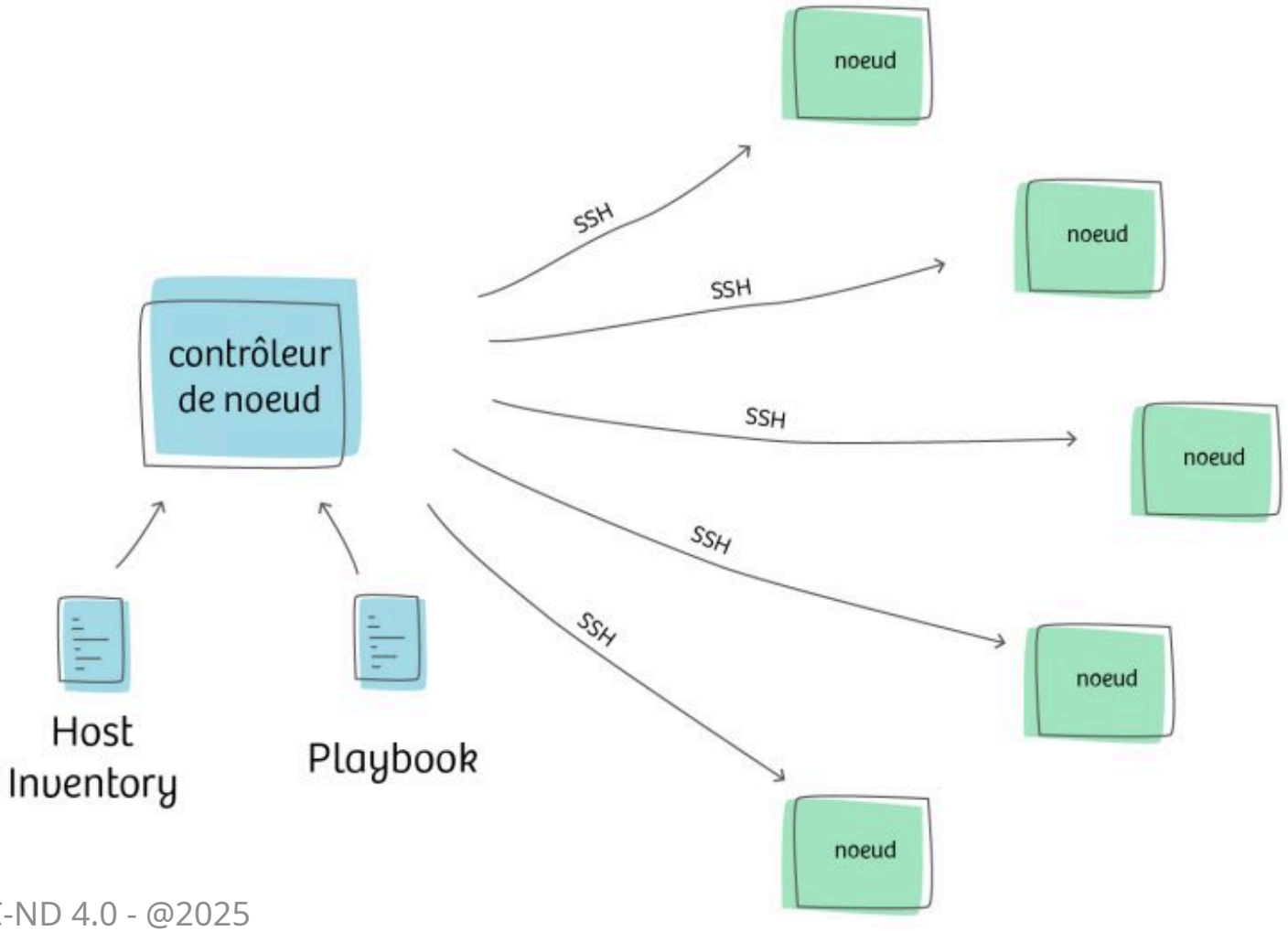
Quelques éléments de vocabulaire

- **Inventaire** : liste des hôtes/serveurs gérés par Ansible, regroupés dans un fichier (`inventory`) ou générés dynamiquement.
- **Task** : une action unique, généralement l'exécution d'un **module** avec des paramètres.
- **Module** : unité de travail d'Ansible, c'est ce qui est réellement exécuté sur les hôtes (ex : `apt` , `copy` , `ping` ...).
- **Playbook** : fichier YAML décrivant une suite de **plays**, qui eux contiennent des **tasks** à exécuter sur un ou plusieurs hôtes.
- **Rôle** : structure réutilisable regroupant tasks, variables, templates, handlers... pour organiser le code par fonction.

Quelques éléments de vocabulaire

- **Collection** : paquet distribuant rôles, modules, plugins… sous un même namespace ; facilite le partage de contenu.
- **Template** : fichier dynamique écrit en **Jinja2**, permettant de générer des fichiers de configuration adaptés à chaque hôte.
- **Plugin** : extension du comportement d'Ansible (connexion, callback, filtrage, etc.) sous forme de code Python.
- **Fact** : données collectées automatiquement sur les hôtes (nom d'hôte, OS, IP, CPU, etc.), utilisables comme variables.

Comment Ansible fonctionne ?



Comment Ansible fonctionne ?

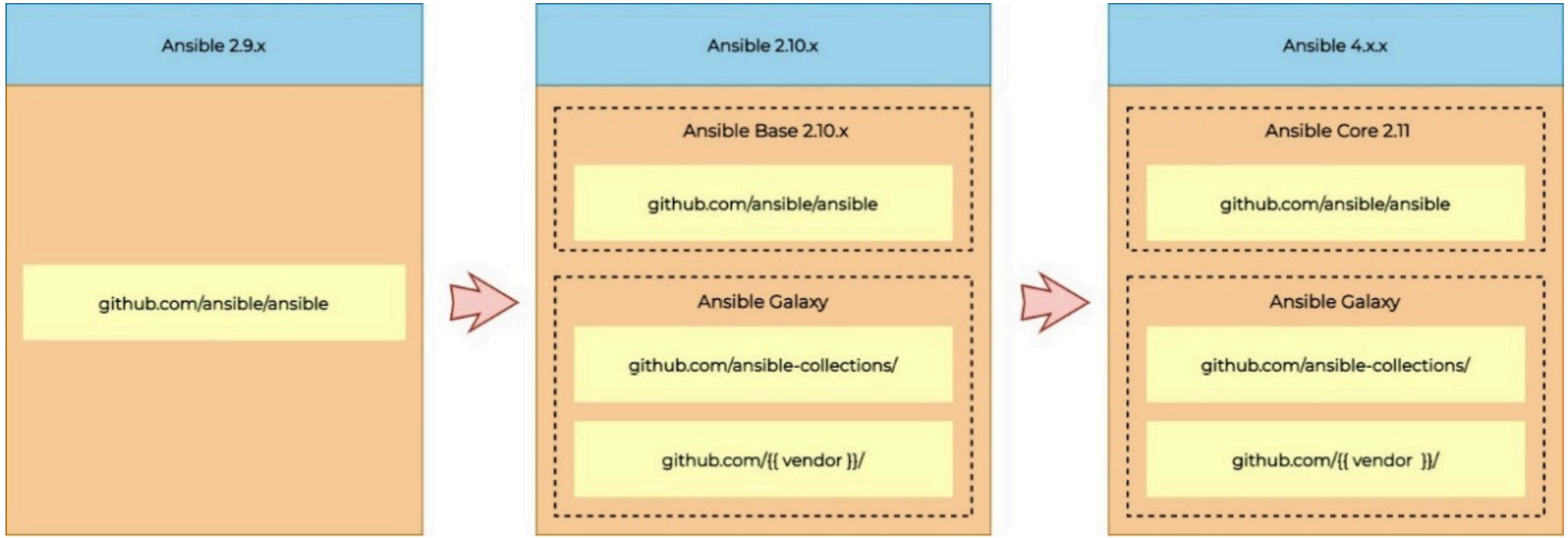
- **Étape 1** : Ansible lit la liste des serveurs gérés à partir du fichier d'inventaire
- **Étape 2** : Ansible se connecte aux serveurs distants à l'aide de SSH avec l'utilisateur courant
- **Étape 3** : Ansible pousse notre module vers les serveurs distants (~/.ansible) et exécute la tâche sur le serveur à l'aide de Python et supprime ensuite le module utilisé une fois la tâche terminée.

📍 Composant	💻 Manager (Contrôleur)	🖥️ Cibles Linux	🪟 Cibles Windows
OS	Linux/macOS (ou WSL sous Windows)	Linux / Unix-like	Windows 10/11, Server 2016+
Python	✅ Requis (Python 3)	✅ Requis (souvent préinstallé)	❌ Pas requis
SSH	✅ Client SSH requis	✅ Serveur SSH actif	🔁 Pas utilisé (par défaut)
WinRM (Windows)	❌ Non requis	❌ Non requis	✅ Requis (configuré pour accepter les connexions distantes)
Ansible installé	✅ Oui (`pip`, `apt`, `brew`)	❌ Non	❌ Non
Agent Ansible	❌ Aucun agent requis	❌ Aucun agent requis	❌ Aucun agent requis
Droits d'accès	Accès SSH avec clé ou mot de passe	Accès SSH et sudo si besoin	Compte administrateur local ou domaine
Connexion réseau	Accès SSH ou WinRM	Port 22 ouvert	Port WinRM (5985 HTTP / 5986 HTTPS) ouvert

Comment Ansible fonctionne sur des serveurs Linux ?

- Ansible utilise des lots pour l'exécution des tâches, qui sont contrôlés par un paramètre appelé **forks**
- La valeur par défaut pour **forks** est **5**, ce qui signifie qu'Ansible exécute une tâche sur les cinq premiers serveurs en parallèle et attend que la tâche se termine, puis sur les cinq serveurs suivants, et ainsi de suite (les forks peuvent être modifiées)
- Nous avons besoin de **1 Go** pour **10 forks** + **2 Go** réservés pour un Ansible Controller
- Donc, si le nombre de forks=20, alors il nous faut **2 Go forks** + **2 Go** pour un Ansible Controller, soit un total de **4 Go**

Restructuration du projet Ansible



Les collections

- Disponible depuis la **version 2.10 d'Ansible**
- Les **collections** facilite la **distribution** et le **partage** des contenus Ansible sous forme de "package".
- Les **collections** vont permettre une **installation cohérente** incluant les modules, rôles, plugins, et les playbooks Ansible.

Restructuration du projet Ansible

- **Avant la version 2.9.x** (jusqu'en fin 2019) :
  Le paquet `ansible` contenait tout : les modules, plugins, rôles, ainsi que le moteur d'exécution. C'était un monobloc.
- **À partir de la version 2.10+** (2020) :
  Ansible a été restructuré. Le cœur du moteur a été renommé **`ansible-base`**, tandis que les modules et plugins ont été déplacés dans des **collections** (publiées par la communauté ou des éditeurs).
- **À partir de la version 2.11+** (mai 2021) :
  `ansible-base` devient **`ansible-core`**. C'est le nouveau nom du moteur minimal, sans collections intégrées (à minima buildin).

Restructuration du projet Ansible

- **Avec les versions `Ansible 3.x` et `Ansible 4.x`** (2021–2022) :
  Ces versions correspondent au **paquet `ansible` de la communauté**, qui regroupe :
  - `ansible-core`
  - Un ensemble de **collections officielles** (community + vendor)

📦 En résumé :

- `ansible-core` = moteur + commandes de base
- `ansible` (community package) = `ansible-core` + collections

Install Ansible

Les packages d'Ansible sont distribués de deux manières :

- **ansible-core** : contient uniquement le code d'exécution Ansible (comme la commande ansible-playbook) et certaines fonctionnalités intégrées (Ansible.Builtin). Suite de la version Ansible 2.9, devient ansible-base 2.10 (2020), et ansible-core 2.15 (en novembre 2023)
- **ansible (community)** : un package beaucoup plus volumineux, qui ajoute une sélection de collections. Ansible 2.10 avec ansible-base 2.10 (2020), puis Ansible 4.x avec ansible-core 2.11 (2021), Ansible 11.x avec ansible-core 2.18 (avril 2025)

Install Ansible

Plusieurs façons d'installer Ansible :

- Le gestionnaire de paquets python : `pip` ou `pipx`

```
$ python3 -m pip install --user ansible
```

```
$ pipx install ansible-core==2.12.3
```

- Le package distribution (Rocky, Debian, Fedora, ...)

```
$ sudo dnf install ansible
```

> Lien vers le site officiel pour installer Ansible de différentes façons

Les ressources

- **Ansible Documentation** : couvre toutes les options Ansible en profondeur.
  - https://docs.ansible.com/ansible/
- **Ansible Glossary** : si il y a un terme que vous ne semblez pas comprendre, consultez le glossaire.
  - https://docs.ansible.com/ansible/glossary.html
- **Ansible Mailing List** - groupe de discussion Google
  - https://groups.google.com/forum/!forum/ansible-project
- **AnsibleFest** : conférence annuelle de la communauté utilisateurs
  - https://www.ansible.com/ansiblefest

Les ressources

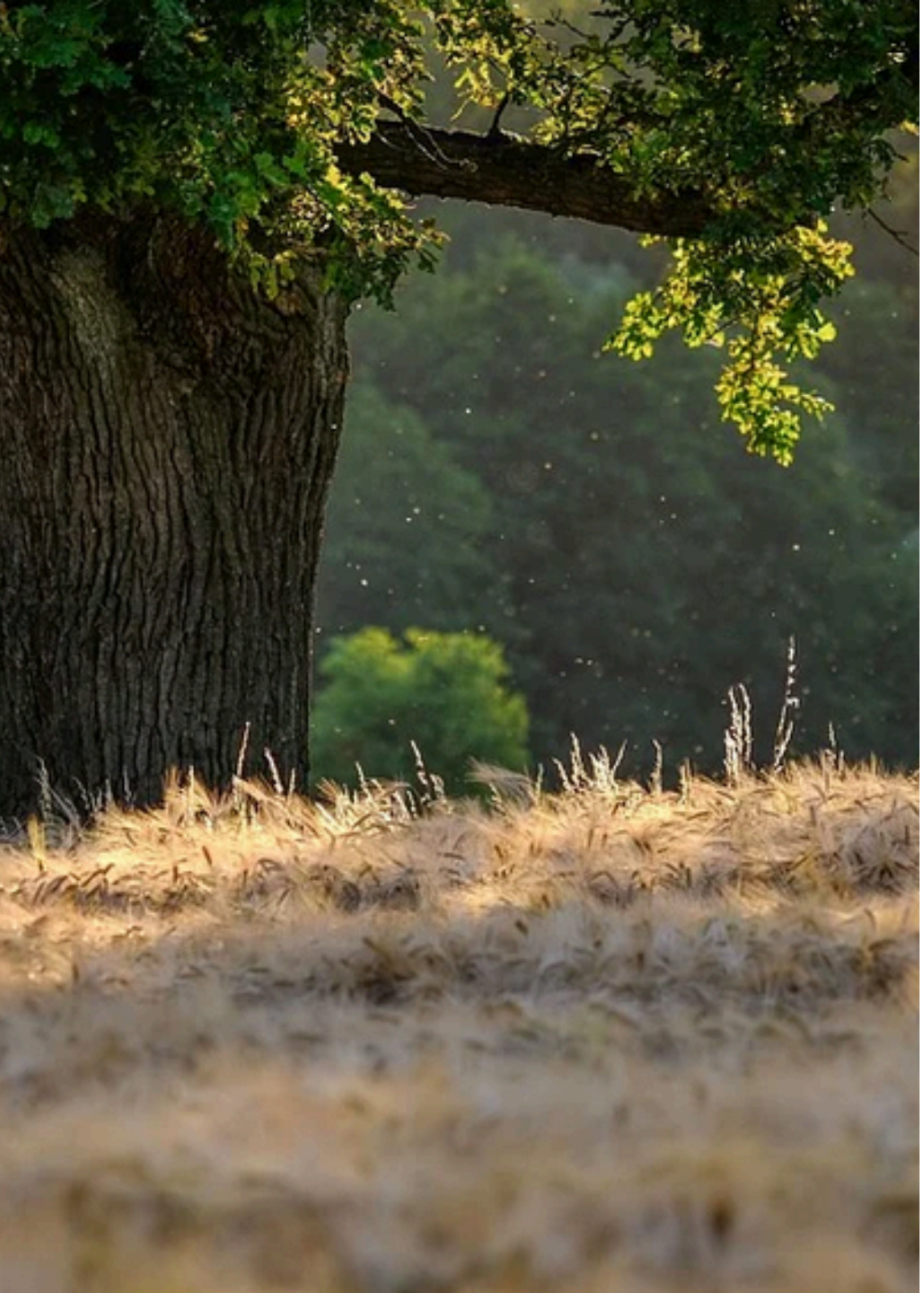
- **Ansible on GitHub** : le référentiel de code Ansible officiel.
  - https://github.com/ansible/ansible
- **Ansible Example Playbooks on GitHub** : de nombreux exemples de configurations de serveur communes.
  - https://github.com/ansible/ansible-examples
- **Getting Started with Ansible** : un guide simple sur la communauté et les ressources d'Ansible
  - https://www.ansible.com/get-started
- **Ansible Blog** : le Blog Ansible
  - https://www.ansible.com/blog

Plusieurs interfaces Web permettent de faire fonctionner Ansible

- **Automation controller** (anciennement Ansible Tower) est un logiciel commercialisé par Red Hat
- **AWX** est le logiciel libre parrainé par Red Hat dont certaines versions sont utilisées pour concevoir Automation controller
- **Semaphore UI** est un logiciel libre offrant une solution extérieure à Red Hat

DEMO

> Provisionner les LAB avec Ansible



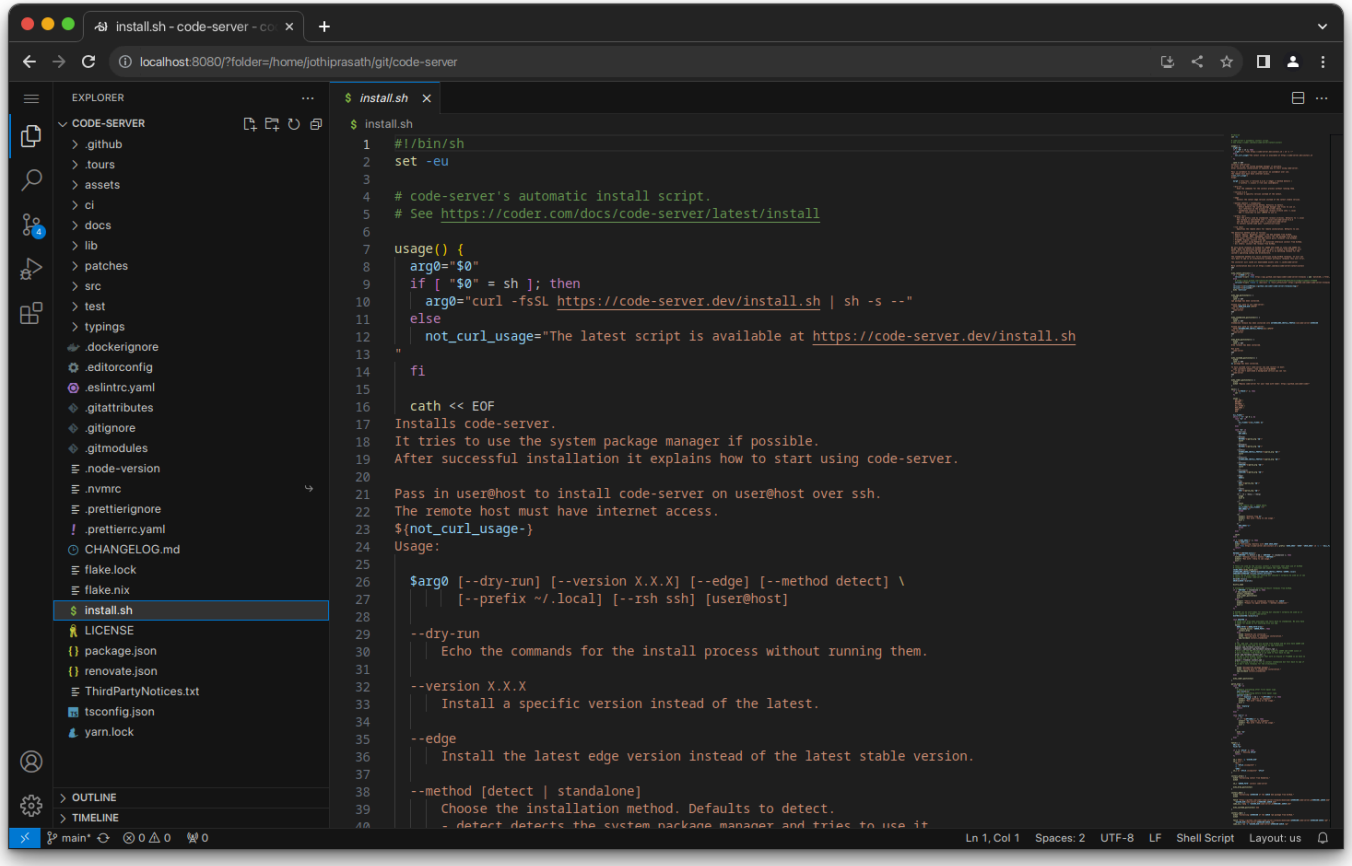
Environnement de travail

Pour les travaux pratiques

> Un LAB par personne

- `X.practice-k8s.cloud` est le control node manager (Rocky)
- `X-1.practice-k8s.cloud` est le node 1 (Rocky) - Apache2
- `X-2.practice-k8s.cloud` est le node 2 (Rocky) - Mysql/ Docker

VS Code sur le navigateur



> https://github.com/coder/code-server

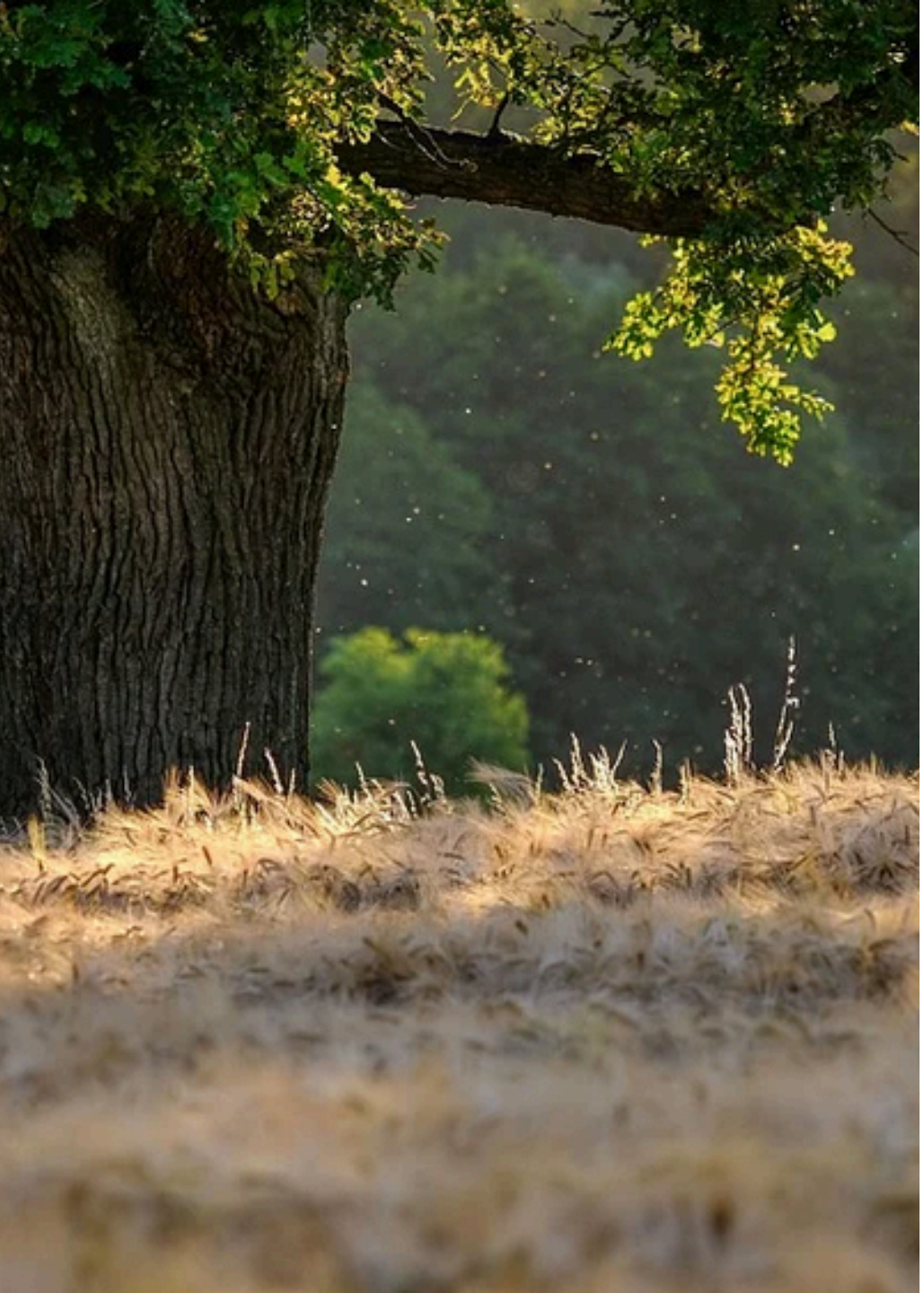
Préparer l'environnement de travail

> Moi

- Créer une paire de clés SSH sur le `node manager`
- Pousser la clés publique sur les `X-X.practice-k8s.cloud`

> Vous

- Installation de **Ansible** sur le **`node manger`** `X.practice-k8s.cloud`
- Créer un utilisateur **`ansible`** sur les `X-X.practice-k8s.cloud`
- Donner à l'utilisateur **`ansible`** les droits `sudo` sur `X-X.practice-k8s.cloud`
- Désactiver le SeLinux sur les `X-X.practice-k8s.cloud`



Configuration du control node manager

Configuration du control manager

> Un prérequis : le control doit avoir accés aux nodes avec SSH (à faire une seule fois).

A vous de tester sur le control node manager !

```
# su - rocky -c "ssh X-1.practice-k8s.cloud"
# su - rocky -c "ssh X-2.practice-k8s.cloud"
```

Installation d’Ansible

- **Rocky Linux** est un système d'exploitation d'entreprise open-source conçu pour être 100% bogue pour bogue compatible avec Red Hat Enterprise Linux®. Elle est en cours de développement intensif par la communauté.

```
$ cat /etc/rocky-release
Rocky Linux release 9.2 (Blue Onyx)
```

- Installation via les **packages systèmes**
- Installation via `pip` de Python (`virtualenv`)
- Installation via les **sources** (Archives ou Git)

> Désactive le selinx dans /etc/selinux/config : SELINUX=disabled

Installation Ansible sur Rocky (node manager)

> Deux étapes sont nécessaires :

- Installation de pip :

```
$ dnf install -y pip
```

- Installation de Ansible community

```
$ python3 -m pip install --user ansible dnspython
```

A vous de jouer !

Installation Ansible sur Rocky

- La version **Ansible** (core)

```
$ ansible --version
ansible [core 2.15.11]
```

- La version **Ansible** (community)

```
$ ansible-community --version
Ansible community version 8.7.0
```

> Release_and_maintenance

Installation Ansible sur Rocky

- Version de **python3**

```
$ python3 -V
Python 3.9.18
```

- Dans le cas où la version de `Python < 3.9`

```
$ alternatives --config python3
```

- Python 3.9 > Ansible Code 2.15 > Ansible 8.7.0 (End Of Life en Nov 2024)
    - Nous sommes limités par la version 3.9 de python. Pour installer une version plus récente de Ansible, il faudrait python > 3.9

> ansible-core-support-matrix

Voici les commandes Ansible

Commande	Description	Exemple d'utilisation
`ansible`	Exécute des tâches Ad-hoc sur un ou plusieurs hôtes sans playbook.	`ansible all -m ping` — Vérifie l'accessibilité de tous les hôtes
`ansible-config`	Gère et vérifie la configuration d'Ansible (fichier `ansible.cfg`).	`ansible-config dump --only-changed` — Montre les paramètres modifiés
`ansible-console`	Ouvre une console interactive pour exécuter des commandes Ansible en direct.	Lancer `ansible-console` pour tester des commandes en temps réel sur plusieurs hôtes
`ansible-galaxy`	Gère les rôles et collections d'Ansible, notamment les téléchargements et publications sur Galaxy.	`ansible-galaxy install geerlingguy.nginx` — Installe le rôle `nginx` de `geerlingguy`

Commande	Description	Exemple d'utilisation
`ansible-playbook`	Exécute des playbooks pour orchestrer des tâches complexes sur plusieurs hôtes.	`ansible-playbook site.yml` — Exécute le playbook `site.yml`
`ansible-test`	Exécute des tests sur les collections Ansible pour vérifier leur compatibilité et intégrité.	`ansible-test units` — Lance les tests unitaires d'une collection
`ansible-community`	Afficher la version du package Ansible de la communauté	
`ansible-connection`	Fournit des informations sur les types de connexion d'Ansible (`ssh`, `local`, `winrm`).	`ansible-connection ssh` — Affiche les configurations pour les connexions SSH

Commande	Description	Exemple d'utilisation
`ansible-doc`	Affiche la documentation des modules et plugins Ansible pour connaître leurs options et exemples.	`ansible-doc -s ansible.builtin.shell` — Montre la doc du module `shell`
`ansible-inventory`	Gère et manipule les inventaires Ansible pour vérifier les hôtes et groupes définis.	`ansible-inventory --list` — Affiche la liste des hôtes et groupes définis dans l'inventaire
`ansible-pull`	Permet aux hôtes de "tirer" leur configuration depuis un dépôt Git.	`ansible-pull -U https://github.com/user/repo.git` — Exécute les playbooks depuis le dépôt Git spécifié
`ansible-vault`	Chiffre et déchiffre des fichiers pour sécuriser les informations sensibles dans les playbooks.	`ansible-vault encrypt secrets.yml` — Chiffre le fichier `secrets.yml`

Introduction à Ansible Navigator

🧭 Qu'est-ce que `ansible-navigator` ?

- Interface **interactive en TUI (Terminal User Interface)** pour Ansible (juin 2021).
- Remplace partiellement `ansible-playbook`, avec des **fonctionnalités de navigation avancées**.
- Permet d'exécuter, explorer et déboguer les **collections, playbooks, rôles**, inventaires, etc.

🔧 Pourquoi l'utiliser ?

- Compatible avec **Ansible Execution Environments (EE)**.
- Navigation intuitive des tâches, variables, hôtes, et résultats.
- Centralise l'expérience de déploiement Ansible avec **Podman/Docker** et `ansible-runner`.

Ansible Navigator - Commandes et options avancées

🛠️ Commandes utiles

```
ansible-navigator run site.yml
ansible-navigator inventory --list
ansible-navigator collections
```

⚙️ Fichiers de configuration

- Par défaut : `ansible-navigator.yml`
- Exemple de configuration :

```yaml
mode: stdout
execution-environment:
  enabled: true
  image: quay.io/ansible/ansible-core-ee
```

Ansible Navigator

🎯 Options expertes

- `--ee` : active l'execution environment
- `--pull-policy` : always, missing, never
- `--container-options` : options supplémentaires pour Podman/Docker

Ansible Navigator - Intégration avancée et debug

🔍 Exploration détaillée d’un playbook

- `ansible-navigator run playbook.yml` donne accès :
  - à chaque tâche
  - aux variables par hôte
  - à la structure des rôles
  - aux erreurs détaillées avec tracebacks

🧪 Débogage & Replay

- Rejouer avec `--mode stdout` pour logs simples.
- `--log-level debug` : niveau de verbosité très élevé.
- Exploration des résultats avec `:0`, `:back`, `:vars`, `:tasks`.

Ansible Navigator

💡 Astuce d'expert

Utilisez les **EE personnalisés** pour embarquer vos propres collections, dépendances Python, ou binaires spécifiques.
Cela garantit **portabilité, reproductibilité et isolation** de l’environnement d'exécution.

Création d'un utilisateur sur les serveurs distants (nodes)

- Nous allons maintenant créer un utilisateur `user-ansible` avec les droits `sudo` sur `X-X.practice-k8s.cloud`.
- Pour cela, nous allons utiliser `Ansible` !
- Pour utiliser `Ansible`, nous avons besoin de definir un `inventaire`

Les fichiers `Inventaire`

Les fichiers Ansible

> Avant d'aller plus loin, nous allons cloner un depot avec les fichiers practice Ansible.

```
$ git clone https://github.com/AlexandreDomont/ansible-practice.git
```

```
$ cd ansible-practice/
```

- Ensuite, "`Open Folder` `ansible-practice` dans l'arborescence VS

Inventaire static

- Créer un fichier `inventory.ini` dans le répertoire `ansible` sur le control manager

```
[labs]
1-1.practice-k8s.cloud
1-2.practice-k8s.cloud

[all:vars] # Prend le dessus sur tout !
ansible_ssh_user=rocky
ansible_ssh_private_key_file=/home/rocky/.ssh/id_rsa
```

- Liste des serveurs dans le fichier *inventory.ini*

```
$ ansible all --list-hosts -i inventory.ini
  hosts (2):
      1-1.practice-k8s.cloud
      1-2.practice-k8s.cloud
```

Les inventaires dynamiques

> Il est possible de récupérer des listes de serveurs depuis des outils déjà présent existants grâce à des plugins.

- Outils de supervision (Nagios, Zabbix...)
- Infrastructure virtualisation (VMware, Proxmox...)
- Cloud (AWS, Azure ou autre)
- Conteneurs de type Docker (Kubernetes ou Docker Swarm)

> Les scripts d'inventaire dynamique sont disponibles sur :

```
$ ansible-doc -t inventory -l
```

Qu’est-ce qu’un inventaire dynamique AWS ?

☁️ Inventaire dynamique dans Ansible

- Permet d’interroger des services cloud (ex : AWS EC2) pour lister automatiquement les hôtes.
- Utilise un **plugin d’inventaire dynamique** intégré (`amazon.aws.aws_ec2`).
- Fini les fichiers d’inventaire statiques : les machines sont détectées dynamiquement à chaque exécution.

🔧 Prérequis :

- Ansible >= 2.10
- Collection `amazon.aws`
- Variables d’authentification AWS (via env vars ou fichier config)

Exemple de configuration pour AWS EC2

📁 Arborescence typique

```
inventory/
├── aws_ec2.yml
└── group_vars/
    └── all.yml
```

📄 `inventory/aws_ec2.yml` :

```yaml
plugin: amazon.aws.aws_ec2
regions:
  - eu-west-1
filters:
  tag:Environment: production
keyed_groups:
  - key: tags.Name
hostnames:
  - private-ip-address
```

Exécution & utilisation

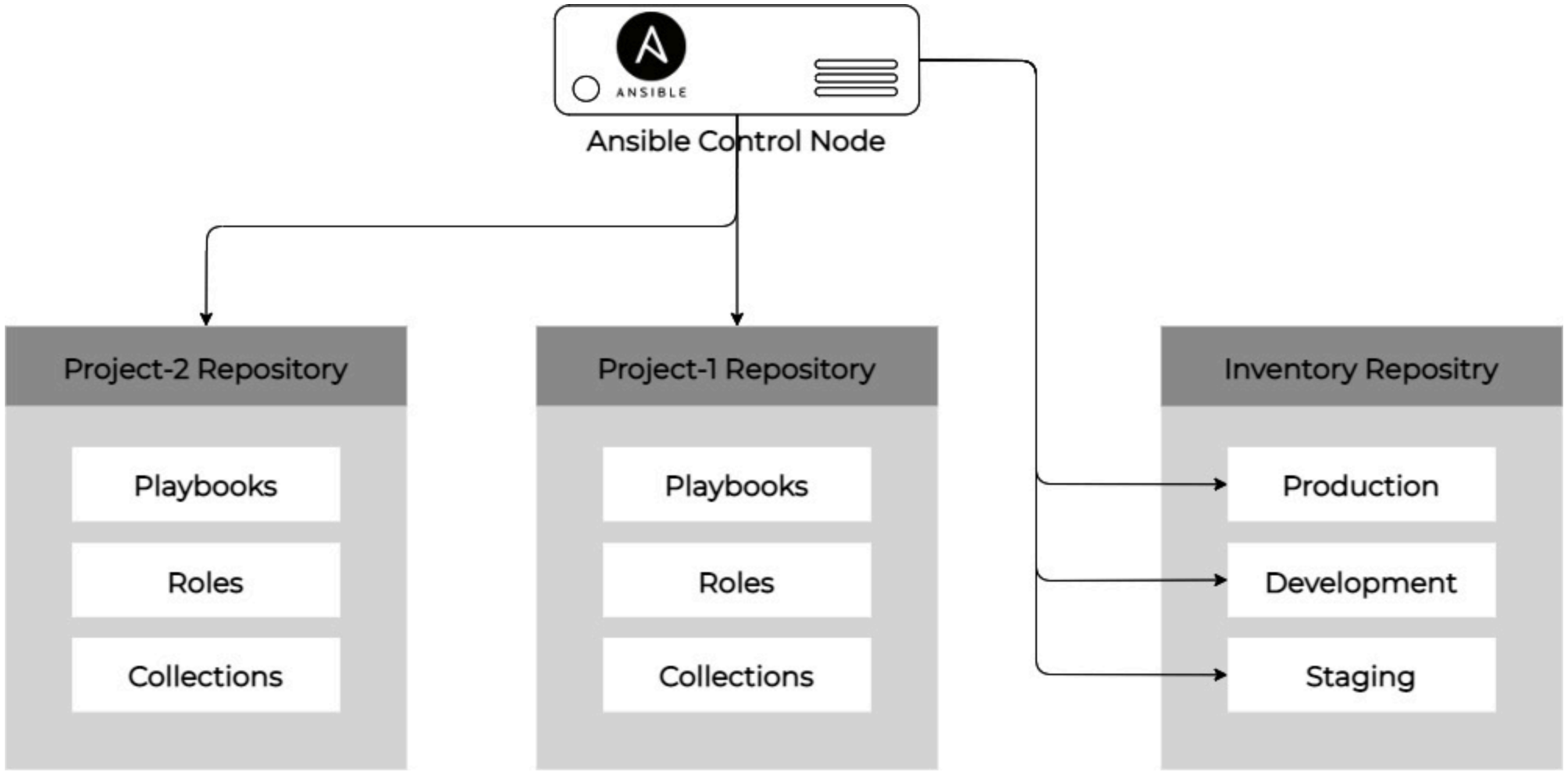
▶️ Commande d’exécution

```
ansible-inventory -i inventory/aws_ec2.yml --graph
ansible -i inventory/aws_ec2.yml all -m ping
```

🧪 Exemple de playbook

```yaml
- name: Mise à jour des serveurs EC2 de prod
  hosts: tag_Name_webserver
  become: yes
  tasks:
    - name: Update apt cache
      apt:
        update_cache: yes
```

Organisation des inventaires



Ansible en mode ad hoc

Setting up an Ansible environment

Ansible en mode ad hoc

> Avant toute chose, il faut s'assurer que python est installé sur les nodes pour que ansible puisse utiliser tous ses modules.

- On installe ***python3*** sur ***1-1.practice-k8s.cloud*** en utilisant le module ***raw***

```
$ ansible -i inventory.ini 1-1.practice-k8s.cloud \
          -m raw -a "dnf install -y python3" -become
```

> Le faire sur les deux : X-Y.practice-k8s.cloud

Ansible en mode ad hoc

On lance la commande Ansible : **ping** ! sur tous les nodes de l'inventaire

```
$ ansible -i inventory.ini -m ping all

1-2.practice-k8s.cloud | UNREACHABLE! => {
"changed": false,
"msg": "Failed to connect to the host via ssh: ssh: Could not
resolve hostname 1-2.practice-k8s.cloud: Name or service not known",
"unreachable": true
}
1-1.practice-k8s.cloud | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
```

- `-i` : **i**nventaire et `-m` : **m**odule

Ansible en mode ad hoc

Les Modules

- Un module est un type de plugin (une extension à Ansible)
- Fait partie des `collections` depuis la version 2.10
- Sous la forme `fully-qualified collection names` (FQCN)
  - Exemple : `ansible.builtin.command`
- La liste des modules disponibles : https://docs.ansible.com/ansible/latest/collections/index_module.html
- Un `module` définit le comportement d'une action (copier un fichier par exemple)

Ansible en mode ad hoc

> Utilisation du module `setup` pour interroger le nom de l'utilisateur connecté via Ansible sur le serveur `1-1.practice-k8s.cloud`

```
$ ansible -i inventory.ini -m setup -a 'filter=ansible_user_id' all --limit 1-1.practice-k8s.cloud

1-1.practice-k8s.cloud | SUCCESS => {
    "ansible_facts": {
        "ansible_user_id": "root",
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false
}
```

> Le code du module `setup` : /root/.local/lib/python3.9/site-packages/ansible/modules/setup.py (dans notre cas avec install pip)

Ansible en mode ad hoc

> Option : pour eviter le warning "discovered_interpreter_python": "/usr/bin/python3.8", on ajoute l'option **interpreter_python** au fichier de configuration **ansible.cfg**

```
$ cat ./ansible.cfg

[defaults]
interpreter_python = /usr/bin/python3
```

Cas pratique avec l'installation de Apache

- Installation du package `Apache` (`-m dnf` et l'option `name=httpd`) sur **1-1.practice-k8s.cloud**

```
$ ansible all -m dnf -a "name=httpd" --limit 1-1.practice-k8s.cloud

1-1.practice-k8s.cloud | CHANGED => {
[...]
    "changed": true,
    "msg": "",
    "rc": 0,
    "results": [
        "Installed: apr-1.7.0-11.el9.x86_64",
        "Installed: apr-util-bdb-1.6.1-20.el9_2.1.x86_64",
        "Installed: apr-util-1.6.1-20.el9_2.1.x86_64",
        "Installed: mod_lua-2.4.53-11.el9_2.5.x86_64",
[...]
```

Cas pratique avec l'installation de Apache

- Contrôle si `Apache` est installé sur `1-1.practice-k8s.cloud`

```
$ ansible 1-1.practice-k8s.cloud -a "systemctl status httpd"

1-1.practice-k8s.cloud | FAILED | rc=3 >>
○ httpd.service - The Apache HTTP Server
     Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled; preset: disabled)
     Active: inactive (dead)
```

- Par defaut Ansible utilise le module `command`

```
$ ansible 1-1.practice-k8s.cloud -a "systemctl status httpd" -vvv | grep "Using module file"

Using module file /usr/lib/python3.11/site-packages/ansible/modules/command.py
```

Cas pratique avec l'installation de Apache

- Une autre façon de le verifier avec `ansible-config` :

```
$ ansible-config dump | grep command
DEFAULT_MODULE_NAME(default) = command
```

- Activation et démarrage du service `Apache` (`state=started` et `enabled=yes`)

```
$ ansible 1-1.practice-k8s.cloud -m service -a "name=httpd state=started enabled=yes"
```

Cas pratique avec l'installation de Apache

- Contrôle du résultat sur le serveur distant

```
$ ansible 1-1.practice-k8s.cloud -a "systemctl status httpd"

1-1.practice-k8s.cloud | CHANGED | rc=0 >>
● httpd.service - The Apache HTTP Server
    Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; preset: disabled)
    Active: active (running)
```

> verifier sur http://X-1.practice-k8s.cloud/

Cas pratique avec l'installation de Apache

> Une fois le serveur Apache installé, nous allons copier un fichier à la racine du site en utilisant cette fois le module `copy`

- Copie d'un fichier (`www.html`) sur `1-1.practice-k8s.cloud` , avec les options du module `copy`
  - l'emplacement du fichier source : `src=www.html`
  - le répertoire destination : `dest=/var/www/html`
  - le propriétaire du fichier : `owner=apache`
  - le groupe du fichier : `group=apache`
  - le mode de protection du fichier : `mode=644`

Cas pratique avec l'installation de Apache

- Créer un fichier `www.html` sur le control manager

```
$ echo "Ansible <> Bonjour !" > www.html
```

- Lancement la copie en mode ad hoc

```
$ ansible 1-1.practice-k8s.cloud -m copy -a "src=www.html \
  owner=apache group=apache dest=/var/www/html"
```

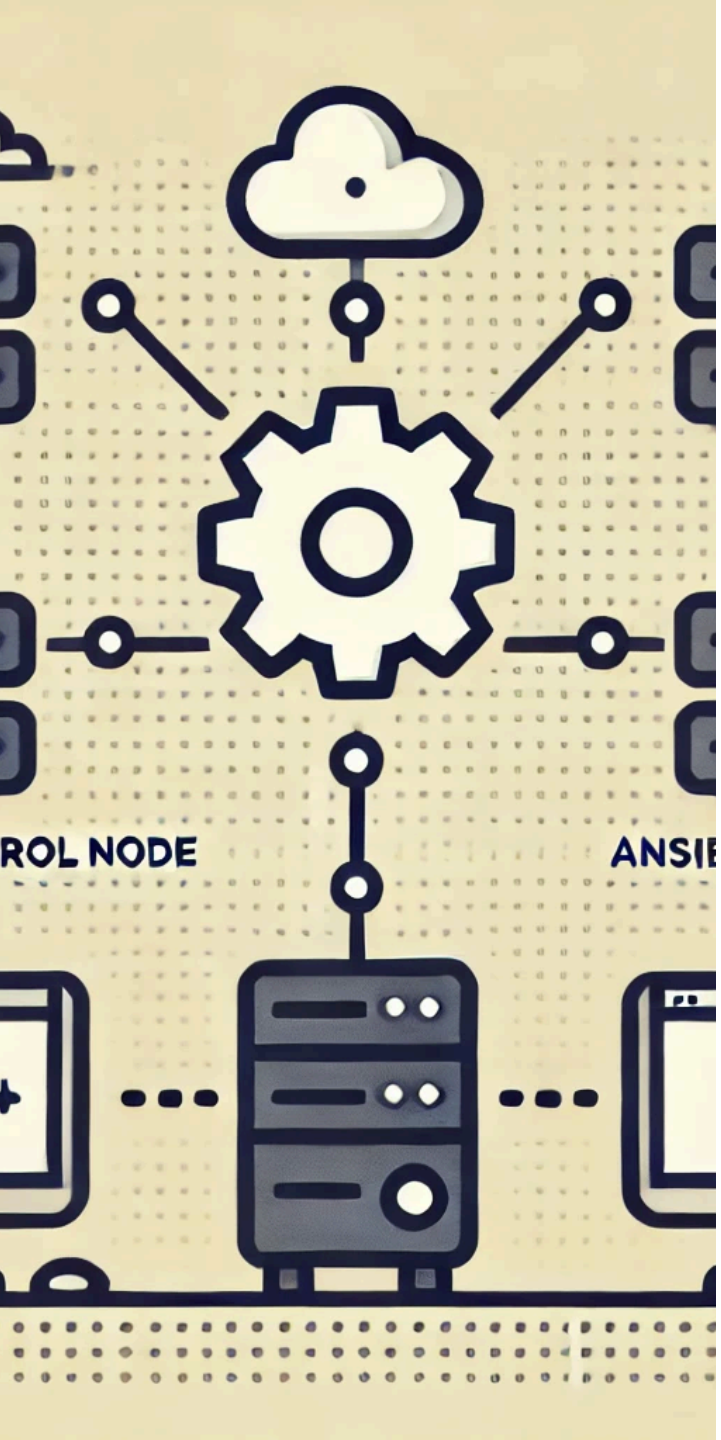
```
 1-1.practice-k8s.cloud | CHANGED
```

Cas pratique avec l'installation de Apache

- Le résultat visible sur le serveur Apache

```
$ curl 1-1.practice-k8s.cloud/www.html -i
```

```
HTTP/1.1 200 OK
Date: Thu, 02 Nov 2023 16:53:06 GMT
Server: Apache/2.4.53 (Rocky Linux)
Last-Modified: Thu, 02 Nov 2023 16:51:05 GMT
ETag: "14-6092e319ec3c3"
Accept-Ranges: bytes
Content-Length: 20
Content-Type: text/html; charset=UTF-8

Ansible <> Bonjour !
```



Exercice

Tester une URL avec Ansible pour vérifier un code HTTP 200

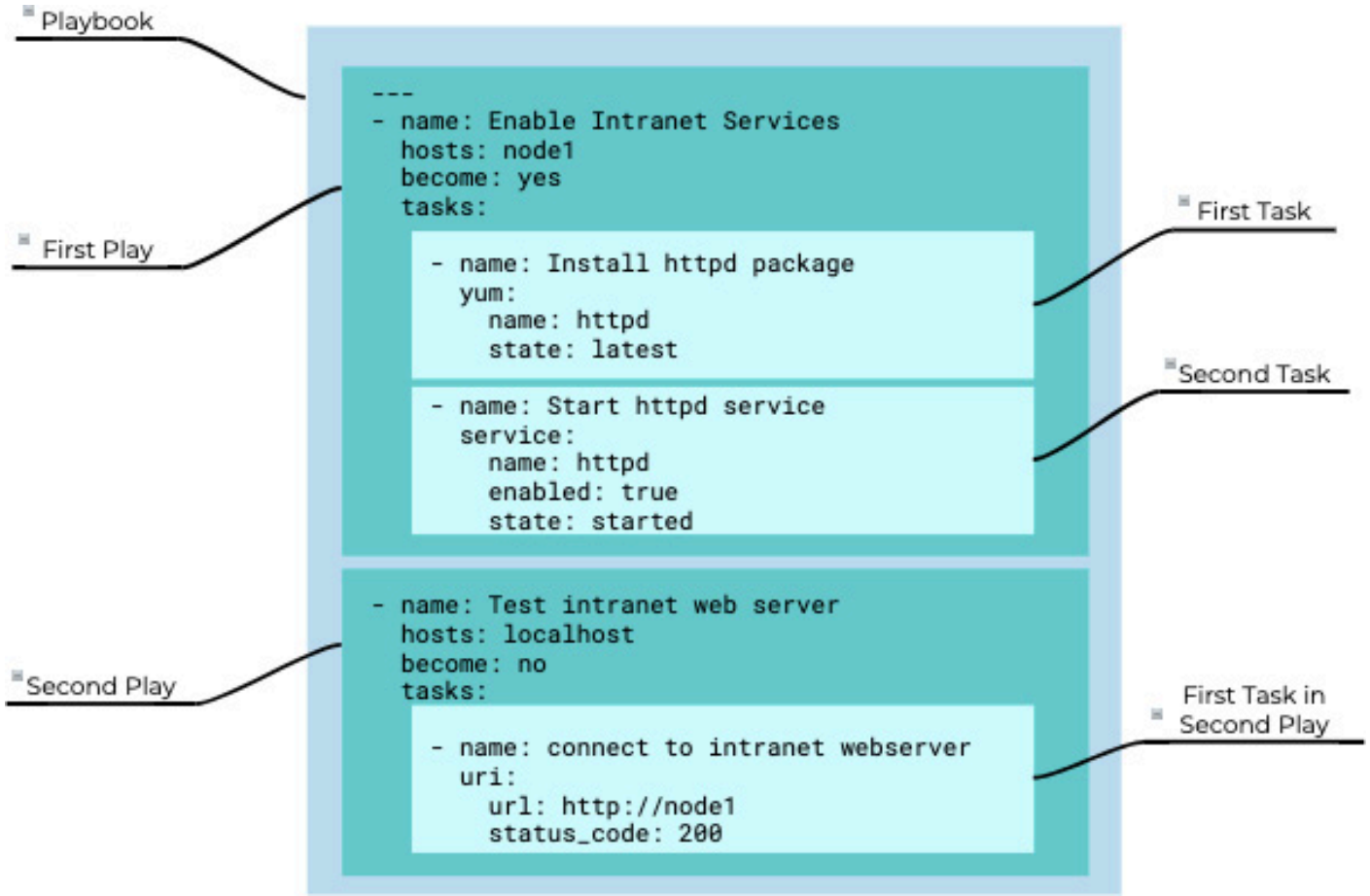
- Utilise le module uri sur l'hôte : X-1.practice-k8s.cloud/www.html
- En mode adhoc (ligne de commande)

Les Playbooks

Les Playbooks

> Un playbook est un fichier au format `YAML`. Ce dernier va donner une liste d'instructions. Ces instructions sont passées à Ansible dans l'ordre de leur déclaration. L'avantage par rapport au mode ad hoc est que tout est écrit dans un fichier, y compris l'enchaînement des opérations.

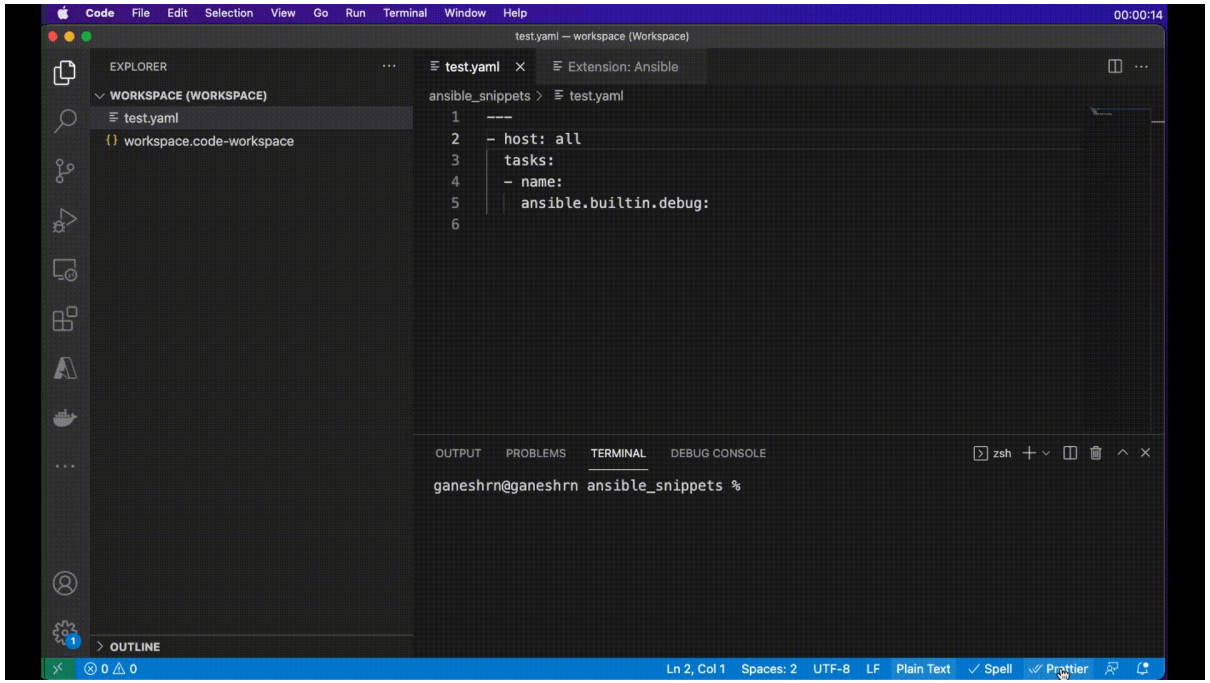
Les Playbooks



Playbook Keywords

Playbook YAML

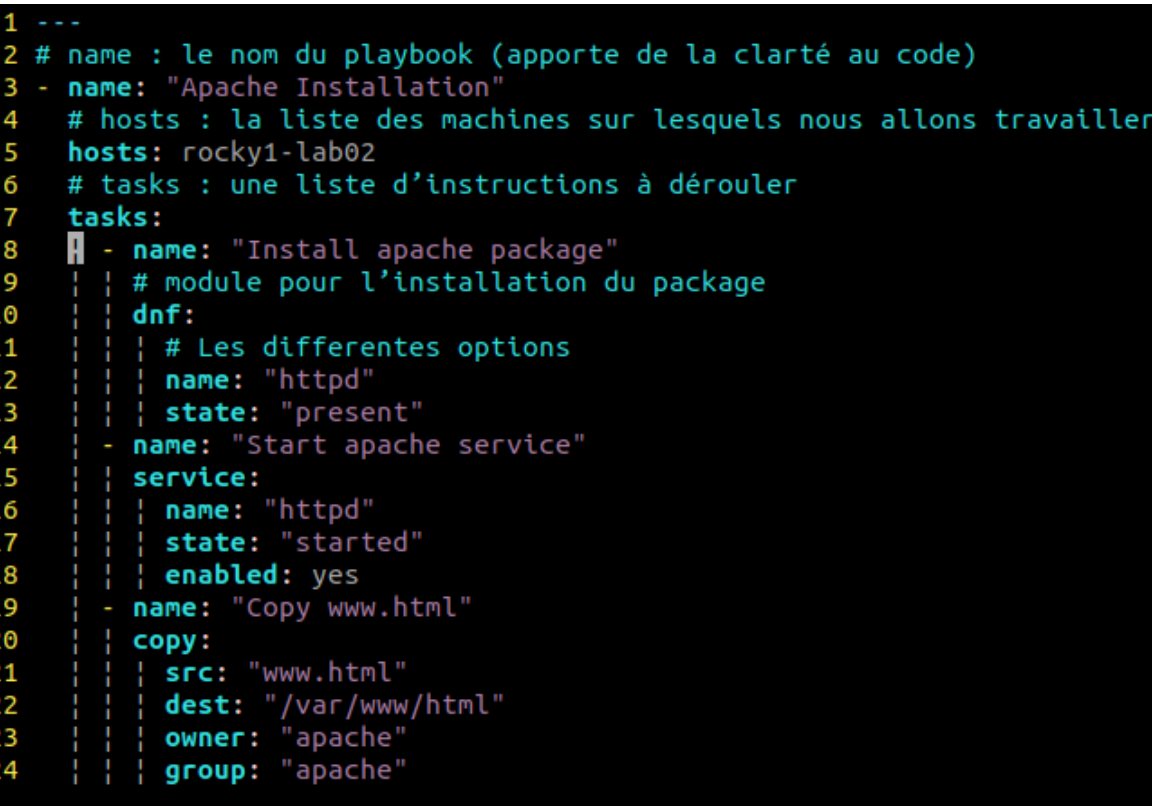
```yaml
- name: Exemple de navigation dans un dictionnaire avec une boucle
  hosts: localhost
  vars:
    projet:
      nom: "Gestion de site web"
      date_debut: "2024-09-01"
      budget: 50000
    equipe:
      - nom: "Alice"
        role: "Chef de projet"
        age: 35
      - nom: "Bob"
        role: "Développeur"
        age: 28
      - nom: "Claire"
        role: "Designer"
        age: 32

  tasks:
    - name: Afficher les informations du projet
      debug:
        msg:
          - "Nom du projet : {{ projet.nom }}"
          - "Date de début : {{ projet.date_debut }}"
          - "Budget : {{ projet.budget }}"

    - name: Parcourir et afficher les membres de l'équipe avec une boucle
      debug:
        msg: "Nom : {{ item.nom }}, Rôle : {{ item.role }}, Âge : {{ item.age }}"
      loop: "{{ equipe }}"

    - name: Afficher le troisième membre de l'équipe
      debug:
        msg: "Le troisième membre de l'équipe est {{ equipe[2].nom }}. Son rôle est {{ equipe[2].role }} et il/elle a {{ equipe[2].age }} ans."
      when: equipe | length >= 3
```



Les Playbooks

> Pour ecrire plus facilement des Playbook :

- VS Code : https://marketplace.visualstudio.com/items?itemName=redhat.ansible



```
---
# name : le nom du playbook (apporte de la clarté au code)
- name: "Apache Installation"
  # hosts : la liste des machines sur lesquels nous allons travailler
  hosts: rocky1-lab02
  # tasks : une liste d'instructions à dérouler
  tasks:
  - name: "Install apache package"
    # module pour l'installation du package
    dnf:
      # Les differentes options
      name: "httpd"
      state: "present"
  - name: "Start apache service"
    service:
      name: "httpd"
      state: "started"
      enabled: yes
  - name: "Copy www.html"
    copy:
      src: "www.html"
      dest: "/var/www/html"
      owner: "apache"
      group: "apache"
```

Pour VIM

- Ajoute au **vimrc** local la configuration pour le positionnement des TAB

```
$ vi ~/.vimrc

 autocmd FileType yaml
setlocal et ts=2 ai sw=2 nu sts=0
```

- Pour ajouter une ligne d'**indentation** : indentLine
- `$ dnf -y install vim-enhanced`

Créer un utilisateur dédié avec un playbook Ansible

🧪 Exercice : Créer un utilisateur avec clé SSH et droits sudo
🎯 Objectif

Créer un playbook Ansible qui :

- Crée un utilisateur nommé **user-ansible**
- Copie une clé SSH publique dans son fichier **authorized_keys**
- Configure les droits **sudo** sans mot de passe pour ce nouvel utilisateur
- Vérifie via SSH que l'utilisateur peut exécuter **sudo -l**

Créer un utilisateur dédié avec un playbook Ansible

📝 Consignes

Contexte :

- Vous travaillez avec un hôte distant défini dans inventory.ini
- Le playbook sera exécuté depuis un poste où l'utilisateur rocky a déjà une clé SSH (~/.ssh/id_rsa)
- Le playbook doit être idempotent, clair, et bien organisé

Créer un utilisateur dédié avec un playbook Ansible

🔧 **Étapes à suivre**

1. Définir un playbook **setup_user.yml** avec les tâches suivantes :
    - Créer l'utilisateur **user-ansible**
    - Copier la clé publique de rocky dans **~user-ansible/.ssh/authorized_keys**
    - Lui donner les droits **sudo** sans mot de passe avec community.general.sudoers
2. Utiliser la commande **lookup('file', ...)** pour lire la clé publique depuis le node manager

Créer un utilisateur dédié avec un playbook Ansible

3. Ajouter une tâche pour tester la connexion :
    - Utiliser une commande SSH depuis localhost pour se connecter en tant que user-ansible et exécuter sudo -l
    - Afficher le résultat avec un module debug
4. Utiliser pause: si nécessaire pour laisser le temps à la configuration de se propager

Configuration Ansible

Pour les options de configuration de `Ansible` (par ordre de priorité) :

- **$ANSIBLE_CONFIG** : Chemin du fichier de configuration dans une variable d'environnement
- **./ansible.cfg** : Fichier de configuration dans le répertoire courant
- **~/.ansible.cfg** : Fichier de configuration dans le répertoire personnel.
- **/etc/ansible/ansible.cfg** : Fichier de configuration par défaut

> Pour lister la configuration actuelle d'Ansible, tu peux utiliser la commande suivante :

```
$ ansible-config dump
```

Configuration Ansible

Créer le fichier de `configuration` dans le repertoire de travail : `ansible.cfg`

```
[defaults]
inventory = inventory.ini
remote_user = user-ansible ## <= ajuster
host_key_checking = false

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

> Attention de commenter `ansible_ssh_user` dans l'inventory

```
[all:vars] # Prend le dessus sur tout !
# ansible_ssh_user=rocky
```

Les Playbooks

> Installation et activation du service Apache avec un Playbook sur 1-1.practice-k8s.cloud

- Le fichier de configuration, `install-apache.yml`

```
---
 name : le nom du playbook (apporte de la clarté au code)
- name: "Apache Installation"
   hosts : la liste des machines sur lesquels nous allons travailler
  hosts: 1-1.practice-k8s.cloud
   tasks : une liste d'instructions à dérouler
  tasks:
    - name: "Install apache package"
       module pour l'installation du package
      dnf:
         Les differentes options
        name: "httpd"
        state: "present"
    - name: "Start apache service"
      service:
        name: "httpd"
        state: "started"
        enabled: yes
    - name: "Copy www.html"
      copy:
        src: "www.html"
        dest: "/var/www/html"
        owner: "apache"
        group: "apache"
```

Les Playbooks

- La commande `ansible-playbook` **SANS** le `ansible.cfg`

```
ansible-playbook -b -K -i inventory.ini install-apache.yml
```

  - l'inventaire `-i inventaire`
  - le playbook à lancer `install-apache.yml`
  - l'option `--become` ou `-b`
  - l'option `--ask-become-pass` ou `-K`

- Lancement du playbook **AVEC** le `ansible.cfg`

```
ansible-playbook install-apache.yml
```

Les Playbooks

- Résultat de la sortie (callback)

```
PLAY [Apache Installation] ***************************************************************************

TASK [Gathering Facts] *******************************************************************************
ok: [1-1.practice-k8s.cloud]

TASK [Install apache package] ************************************************************************
ok: [1-1.practice-k8s.cloud]

TASK [Start apache service] **************************************************************************
ok: [1-1.practice-k8s.cloud]

TASK [Copy www.html] *********************************************************************************
ok: [1-1.practice-k8s.cloud]

PLAY RECAP *******************************************************************************************
1-1.practice-k8s.cloud     : ok=4    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

> Tous les actions sont validées (`ok=4`) puisque elles ont deja été réalisées précédement et que Ansible gére l'idempotence !

Vérification des playbooks avec ansible-lint

> **ansible-lint** est un outil complementaire à Ansible. Il est maintenu par RedHat et fonctionne avec des régles de conformités qui permettent de contrôler les fichiers Ansible.
> https://ansible.readthedocs.io/projects/lint/

- Installer l'outil **ansible-lint**

```
$ python3 -m pip install ansible-lint
```

- Pour connaitre toutes les régles

```
$ ansible-lint --list-rules
```

Vérification des playbooks avec ansible-lint

- Lance la verification du fichier `install-apache.yml`

```
$ ansible-lint install-apache.yml
```

```
                           Rule Violation Summary
 count tag                 profile    rule associated tags
     1 yaml[trailing-spaces] basic      formatting, yaml
     1 yaml[truthy]          basic      formatting, yaml
     1 risky-file-permissions safety    unpredictability, experimental (warning)
     3 fqcn[action-core]     production formatting

Failed after min profile: 5 failure(s), 1 warning(s) on 1 files.
```

```
fqcn[action-core]: Use FQCN for builtin module actions (dnf).
install-apache.yml:8 Use `ansible.builtin.dnf` or `ansible.legacy.dnf` instead.
```

Vérification des playbooks avec ansible-lint

- Ouvre le fichier à modifier avec l'option `list` pour faire apparaître les espaces

```
$ vi +11 "+set list nu" install-apache.yml
```

```
[...]
  tasks:$
  8     - name: "Install apache package"$
  9        module pour l’installation du package$
 10       dnf:$
 11          Les differentes options $
 12         name: "httpd"$
 [...]
```

> Il faudrait ecrire **ansible.builtin.dnf:** vs **dnf:**

Vérification des playbooks avec --syntax-check

> A noter aussi le check de la commande ansible :

- `--syntax-check` : Cette option est utilisée avec la commande ansible-playbook.
- Lorsque vous exécutez `ansible-playbook --syntax-check`, Ansible analyse vos fichiers de playbook YAML et les vérifie pour détecter d'éventuelles erreurs de syntaxe.
- `--syntax-check` est une vérification simple des erreurs de syntaxe YAML (Ansible ne lit pas de fichier de règles, il utilise directement son propre analyseur YAML intégré)
- `ansible-lint` fournit une analyse plus complète, des problèmes potentiels et de la conformité aux styles

Options : `--diff` et `--check`

🧪 Ces options permettent de tester vos playbooks **en toute sécurité**.

✅ `--check` : Mode simulation

- Lance le playbook en **mode dry-run**
- Aucun changement appliqué
- Pratique pour voir *ce qui serait fait*

```
ansible-playbook playbook.yml --check
```

🔍 `--diff` : Voir les différences

- Affiche les **différences entre l'état actuel et l'état souhaité**
- Utile avec certains modules (`template` , `copy` , etc.)

```
ansible-playbook playbook.yml --diff
```

💡 Combinez : `--syntax-check --check --diff` pour une revue complète sans danger !

Les Variables

Les Variables

> Les variables sont indispensables pour definir le contexte et l'environnement d'execution. Elles peuvent prendre differentes formes et se placer à differents endroits, tout en respectant une héirachie stricte.

Qu'est-ce qu'une variable ?

- Élément clé pour rendre les playbooks dynamiques et réutilisables
- Syntaxe : `{{ variable_name }}`
- Définies dans différents contextes avec des niveaux de priorité

Types de variables

1. Variables définies dans les fichiers

- `group_vars/` et `host_vars/`
- `vars:` dans un playbook
- `defaults/main.yml` ou `vars/main.yml` dans un rôle

2. Variables en ligne de commande

- Passées avec `--extra-vars` ou `-e`

```
ansible-playbook playbook.yml -e "env=prod"
```

Types de variables

3. Variables d'inventaire

- Définies directement dans les fichiers d’inventaire (`ini` , `yaml` , ou plugins dynamiques)

4. Variables de faits (facts)

- Collectées automatiquement par `setup` (ou `gather_facts: true`)
- Exemple :

```
ansible_facts['distribution']
```

Autres types de variables

5. Variables d’environnement

- Accès via `lookup('env', 'NOM')`

6. Variables d'enregistrement

- Résultats d'une tâche :

```
- name: Tester un ping
  ping:
  register: resultat_ping
```

Types de variables

7. Variables conditionnelles ou temporaires

- Définies avec `set_fact` ou via des boucles

```
- set_fact:
    chemin: "/tmp/{{ inventory_hostname }}"
```

8. Variables de rôle

- `defaults/main.yml` (priorité faible)
- `vars/main.yml` (priorité plus élevée)

9. Variables magiques

- Variables spéciales générées automatiquement par Ansible

Variable	Description
`inventory_hostname`	Nom de l'hôte actuel dans l'inventaire
`group_names`	Liste des groupes dont l'hôte fait partie
`hostvars`	Dictionnaire de toutes les variables d’un hôte donné
`ansible_play_hosts`	Liste des hôtes ciblés dans le play courant
`playbook_dir`	Répertoire du playbook en cours
`role_path`	Chemin du rôle en cours d'exécution
`ansible_version`	Version actuelle d’Ansible
`omit`	Permet d’omettre une valeur d’argument si vide

Hiérarchie des variables (ordre de priorité)

De la plus faible à la plus forte :

1. `defaults/main.yml` dans un rôle
2. Variables dans les fichiers d'inventaire
3. `vars` dans un playbook
4. Variables dans `host_vars` / `group_vars`
5. Variables enregistrées (`register`)
6. `set_fact`
7. Variables passées en ligne de commande (`-e`)

Bonnes pratiques avec les variables

✅ Utiliser `defaults` pour les valeurs par défaut dans les rôles
✅ Regrouper les variables dans `group_vars/` et `host_vars/` pour plus de clarté
✅ Documenter les variables (description, type, valeur attendue)
✅ Préférer `set_fact` pour les valeurs calculées dynamiquement
✅ Utiliser des noms explicites et préfixés pour éviter les conflits
✅ Tirer parti de la commande `ansible -m setup` pour explorer les facts

❌ Éviter les noms de variables trop génériques (`value`, `temp`, `data`)
❌ Ne pas mélanger les sources de vérité sans contrôle (risque de conflits)

⚠️ Ne pas mélanger les sources de vérité sans contrôle

En Ansible, une **source de vérité** est un endroit où une donnée de configuration est définie :

- Fichiers `group_vars/` ou `host_vars/`
- Variables dans les playbooks ou rôles
- Variables d'environnement
- Lignes de commande (`-e`)
- Inventaires multiples
- Collections tierces

🚨 Risques si on les mélange sans contrôle

- ❌ Conflits entre valeurs (même variable définie à plusieurs endroits)
- ❌ Difficultés de débogage
- ❌ Comportement imprévisible (quelle valeur est prise ?)
- ❌ Déploiement incohérent ou cassé

✅ Bonnes pratiques

- Définir **une hiérarchie claire**
- Éviter la redondance de variables
- Documenter où sont définies les valeurs importantes
- Connaître l'ordre de **priorité des variables** dans Ansible :

```
CLI vars > play vars > role vars > inventory vars > defaults
```

💡 Une seule source de vérité = une infrastructure plus fiable !

Quelques exemples : Variables Inventaire

- Soit directement dans le fichier **inventaire**

```
[lamp]
web apache_url=rec.wiki.localdomain mysql_user_password=MyPassWord!
```

Quelques exemples de variables

- Soit dans des répertoires `group_vars` et `host_vars`, avec un fichier par serveur ou groupe de serveurs au format `YAML` ou `JSON`

```
$ tree inventories/
inventories/
└── stg
    ├── group_vars
    │   └── web.yml
    ├── hosts
    └── host_vars
        ├── node1.yml
        └── node2.yml
```

Quelques exemples de variables

```
$ cat inventories/stg/hosts

[web]
node1 ansible_host=192.168.33.10 web_server_port=8080
node2 ansible_host=192.168.33.11
node3 ansible_host=192.168.33.12 web_server_port=8083

[all:vars]
ansible_ssh_private_key_file=/home/ansible/.ssh/id_eddsa
```

Quelques exemples de variables

```
inventories/stg/group_vars/web.yml
web_server_port: 80
```

```
inventories/stg/host_vars/node1.yml
web_server_port: 8081
```

```
inventories/stg/host_vars/node2.yml
web_server_port: 8082
default_web_page_content: "Welcome to node2"
```

Quelques exemples de variables

```
$ ansible-inventory --list -i inventories/stg/
```

```
$ ansible-inventory --list -yaml --vars -i inventories/stg/
```

Quelques exemples de variables

- Définition d'une variable dans un fichier

```
$ echo varfile: file > fichier-variables.yml
```

- Soit en mode ad hoc avec l'option `-e`

```
$ ansible -e variable=valeur -e @fichier-variables.yml \
      -m debug -a var=variable,varfile localhost

      localhost | SUCCESS => {
      "variable,varfile": "('valeur', 'file')"
      }
```

Quelques exemples de variables

- Un exemple avec des variables dans le `playbook`

```
- name: Test variables
  hosts: localhost

  vars:
   prio: playbook_vars

  vars_files:
    - prio.yml

  vars_prompt:
    - name: "prio"
      prompt: "A vous :"
      private: no

  tasks:
    - name: Show my variable
      debug:
        msg: "{{ prio }}"
```

Quelques exemples de variables

> variables_example.yaml

```
- name: create a user using a variable
  hosts: all
  vars:
    user: lisa
  tasks:
    - name: create a user {{ user }} on host {{ ansible_hostname }}
      ansible.builtin.debug:
        msg: "{{ user }} and its data type is : {{ user | ansible.builtin.type_debug}} "
```

```
 ansible-playbook variables_example.yaml
```

> `ansible_hostname` est une variable `fact` !
>
> `ansible.builtin.type_debug` est un plugin de type filter

Les Variables / set_fact

- `set_fact` est un module utilisé pour definir une variable **dynamique** dans le contexte d'un playbook

```
- name: get snapshot id
  shell: >
    aws ec2 describe-snapshots --filters
    Name=tag:Name,Values=my-snapshot
    | jq --raw-output ".Snapshots[].SnapshotId"
  register: snap_result

- set_fact: snap={{ snap_result.stdout }}

- name: delete old snapshot
  command: aws ec2 delete-snapshot --snapshot-id "{{ snap }}"
```

Les Variables / set_fact

- Un autre exemple `set_fact` qui fait evoluer la valeur d'une variable dans une tache

```
- hosts: localhost
  vars:
    x: 5
  gather_facts: false
  tasks:
    - ansible.builtin.debug:
        msg:
          - "x value is: {{x}}"
    - ansible.builtin.set_fact:
        x: 8
        y: "ansible playbooks"
    - ansible.builtin.debug:
        msg:
          - "x value is: {{x}}"
          - "y value is: {{y}}"
```

Les Variables / Magic

`Les Variables magiques` /magic-variables sont réservées pour les variables systems :

- **hostvars**
- **groups**
- **group_names**
- **inventory_hostname**
- **...**

> Ces variables ne peuvent pas être définies directement par l'utilisateur

Les Variables / Magic

> magic-variable.yml

```
- name: playbook directory demo
  hosts: all
  gather_facts: false
  tasks:
    - name: print current playbook directory
      ansible.builtin.debug:
        var: playbook_dir
```

```
  $ ansible-playbook magic-variable.yml

  ok: [1-2.practice-k8s.cloud] => {
  "playbook_dir": "/root/ansible"
  }
```

Les Variables / Register

Le `registre` est un moyen de capturer le résultat de l'exécution d'une tâche et de le stocker dans une variable.

> register.yml

```
- name: register
  hosts: all
  gather_facts: no
  tasks:
  - name: starting httpd
    service: name=httpd state=started enabled=yes

  - name: httpd status
    command: service httpd status
    register: httpd_status

  - name: httpd status output
    debug:
      var: httpd_status
```

Les Variables / Register / When

> command-output-test.yml

```
- name: test command output
  hosts: all
  tasks:
    - name: Test for VG existance
      command: vgs
      register: vgout
      ignore_errors: True

    - name: Show variable value
      debug:
        var: vgout

    - name: print
      debug:
        msg: vg does not exist
      when: "'No such' in vgout.msg"
```

Les Variables / Register / When

```
$ ansible-playbook command-output-test.yml

TASK [Show variable value]
ok: [1-1.practice-k8s.cloud] => {
    "vgout": {
        "changed": false,
        "cmd": "vgs",
        "failed": true,
        "msg": "[Errno 2] No such file or directory: b'vgs'",
        "rc": 2,
        "stderr": "",
        "stderr_lines": [],
        "stdout": "",
        "stdout_lines": []
    }
}

TASK [print]
ok: [1-1.practice-k8s.cloud] => {
    "msg": "vg does not exist"
}
```

Les Variables / Register / When

> Un autre exemple avec un filtre

```
- name: test register
  hosts: all
  tasks:
    - shell: cat /etc/passwd
      register: passwd_contents
    - debug:
        msg: "{{ passwd_contents }}"
    - debug:
        msg: passwd contains user lisa
      when: passwd_contents.stdout.find('lisa') != -1
```

Hiérarchie et priorité des variables

- Il y a d'autres variables qui peuvent être trouvés dans les `Roles` ou dans les variable d'environnement `facts` et à bien d'autres endroits encore....la documentation officiel en donne un aperçu :
    - understanding-variable-precedence (*les dernières variables de la liste remplacent toutes les autres variables*)
- Théoriquement, il faudrait utiliser les variables au niveau des groupes dans le répertoire `group_vars`. Le fichier `host` ne devrait contenir que des déclarations de serveurs.

Les Variables / arithmetic Operations

```
- hosts: localhost
  vars:
    a: 90
    b: 40
  gather_facts: false
  tasks:
    - ansible.builtin.debug:
        msg:
          - "Inputs are:  a = {{a}} and b = {{b}}"
          - "a + b = {{a+b}}"
          - "a - b = {{a-b}}"
          - "a * b = {{a*b}}"
          - "a / b = {{a/b}}"
          - "a // b = {{a//b}}"  Quotient
          - "a ** 2 = {{a**2}}"  Exponent
```

Les Variables / filters et Methods

> upper.yml

```
- hosts: localhost
  vars:
    a: "ansible"
  gather_facts: false
  tasks:
    - ansible.builtin.debug:
        msg:
          - "a value is : {{a}}"
          - "a in upper: {{a|upper}}"  # Jinja Filter
          - "a in upper: {{a.upper()}}" # Python Metods (filter plugin ansible)
          - "{{'hello'|upper}}"
```

```
$ ansible-playbook upper.yml
```

> https://jinja.palletsprojects.com/en/3.1.x/templates/builtin-filters

Les Variables / tests

```
- hosts: localhost
  gather_facts: false
  vars:
    x: 40
    my_name: 'ansible'
    my_path: '/home/ansadmin/apbooks/displayMsg.yml'
    my_link_path: '/home/ansadmin/apbooks'
  tasks:
    - ansible.builtin.debug:
        msg:
          - "x is defined:   {{ x is defined }}"
          - "y is defined:   {{ y is defined }}"
          - "z is undefined: {{ z is undefined }}"
          - "my_name is lower: {{my_name is lower}}"
          - "my_name is upper: {{my_name is upper}}"
          - "my_name is string: {{my_name is string}}"
          - "x is divisibleby 2: {{x is divisibleby 2}}"
          - "x is even: {{ x is even }}"
          - "x is odd: {{ x is odd }}"
          - "x is number: {{ x is number}}"
          - "The given path is: {{my_path}}"
          - "my_path is file:   {{my_path is file}}"
          - "my_path is directory: {{my_path is directory}}"
          - "my_path is exists:   {{my_path is exists}}"
          - "my_link_path is link  {{my_link_path is link }}"
          - "hello is lower: {{'hello' is lower}}"
```

Les Variables / lists

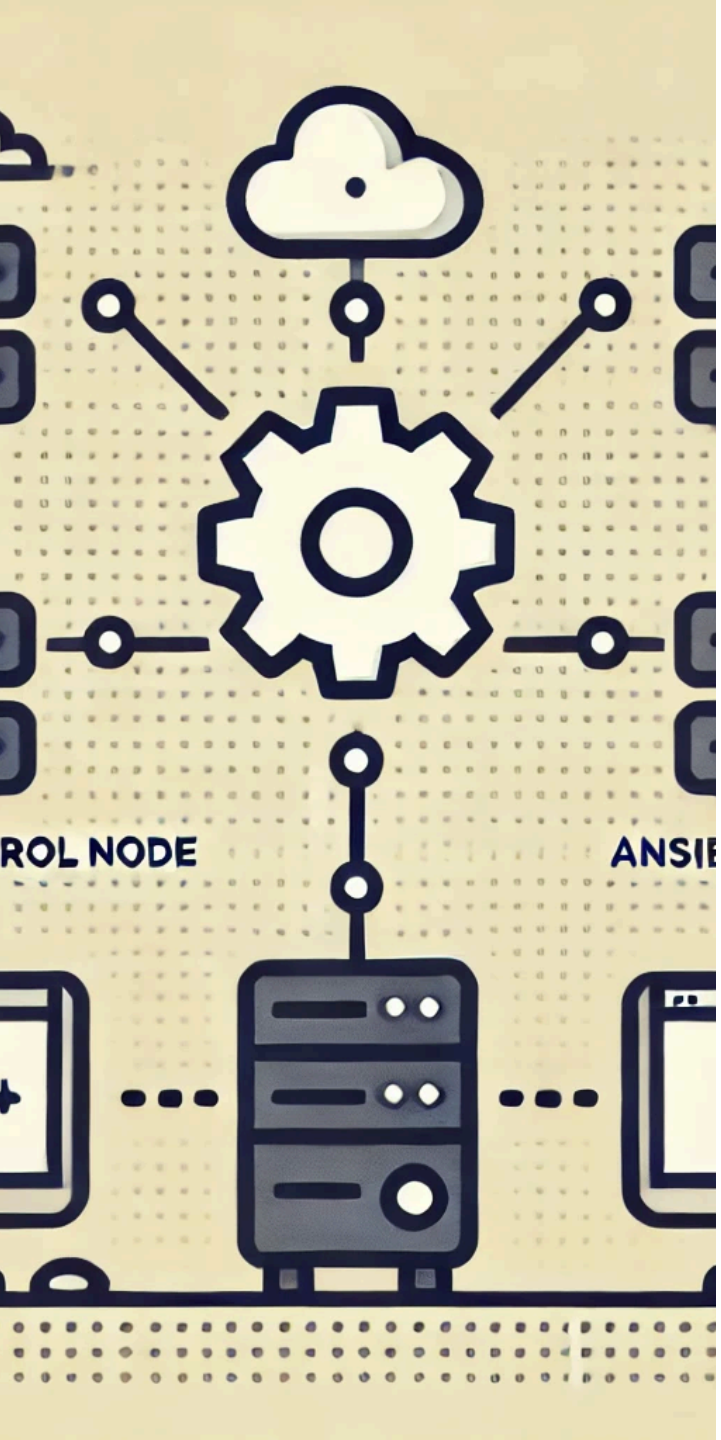
```
- hosts: localhost
  vars:
    mylist: [5,8,3,7,4]
    mynewlist: [ 67,89]
    mergedlist: "{{mylist + mynewlist }}"
  gather_facts: false
  tasks:
    - ansible.builtin.debug:
        msg:
          - "mylist is: {{mylist}}"
          - "no of elements : {{mylist | length}}"
          - "min num: {{mylist | min}}"
          - "max num: {{mylist | max}}"
          - "first num: {{mylist[0]}}"
          - "second num: {{mylist[1]}}"
          - "first num: {{mylist[-5]}}"
          - "second num: {{mylist[-4]}}"
          - "fist num with filter: {{mylist | first}}"
          - "last num with filter: {{mylist | last}}"
          - "mergedlist is :{{mergedlist}}"
    - ansible.builtin.set_fact:
        newmergedlist: "{{mylist + mynewlist }}"
        joineddata: "{{mylist | join('-')}}"
    - ansible.builtin.debug:
        msg:
          - "joined data is: {{joineddata}}"
```

Variables / command

- A quoi ressemble la commande ansible-playbook pour exécuter le playbook suivant :

```
- name: Play to execute any command
  hosts: "{{targets}}"
  gather_facts: false
  tasks:
    - name: Executing given command
      ansible.builtin.command:
        cmd: "{{inputCmd}}"
      register: cmdOutput
    - name: Displaying result
      ansible.builtin.debug:
        msg: "{{cmdOutput.stdout  if cmdOutput.stdout | length != 0 else  cmdOutput.stderr }}"
```

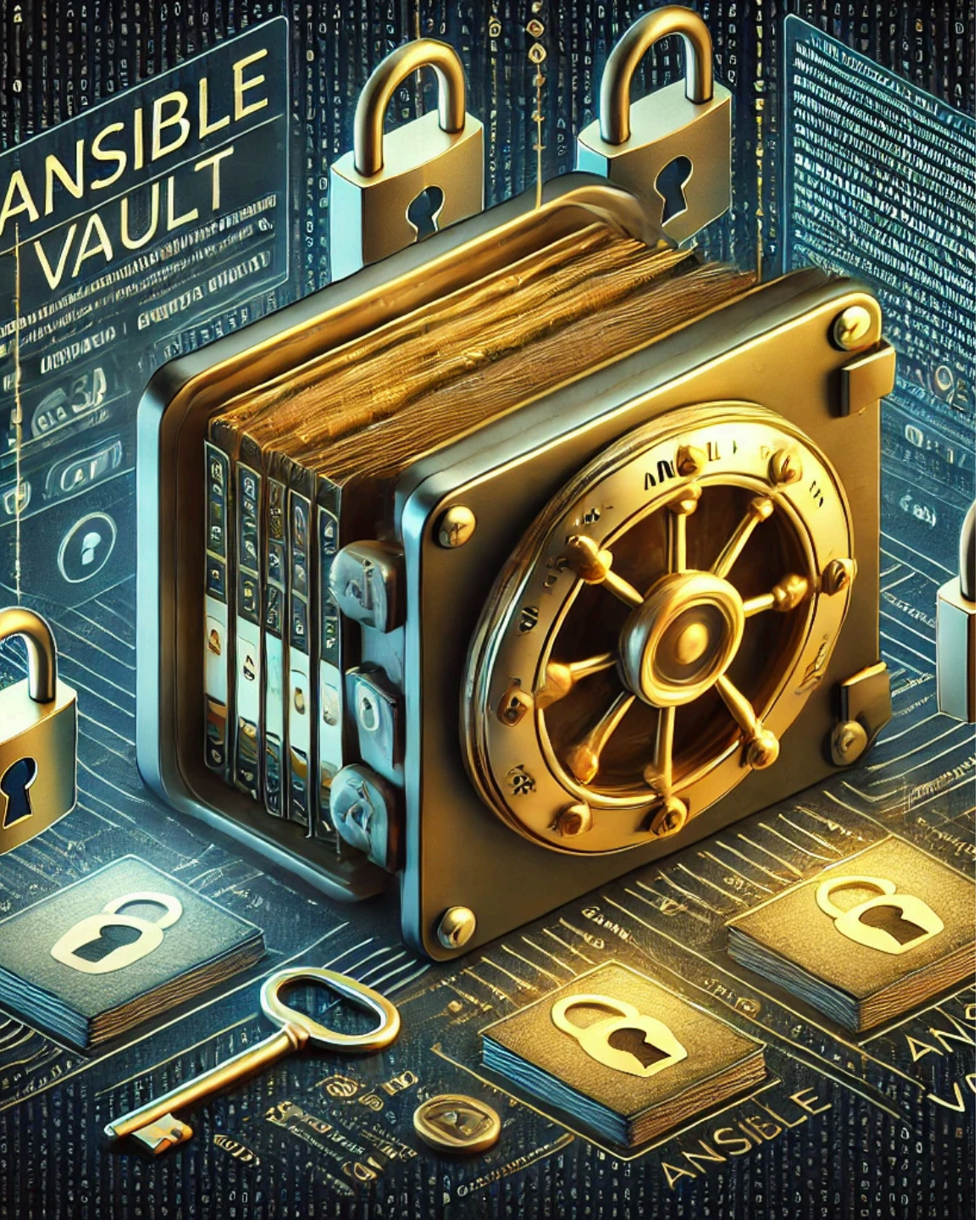
- Expliquer la ligne : `msg: "{{cmdOutput.stdout  if cmdOutput.stdout | length != 0 else cmdOutput.stderr }}"`



Exercice

Tester une URL dans un `playbook` pour vérifier un code HTTP 200

- Utilise le module `uri` sur l'hôte : X-1.practice-k8s.cloud/www.html
- Utiliser `register` : Enregistre la réponse de la requête dans la variable `result`.
- Utiliser `assert` : Vérifie si le code de statut HTTP est 200. Si ce n'est pas le cas, il échoue avec un message d'erreur personnalisé.



Le chiffrement des fichiers

Le chiffrement des fichiers

> Certaines informations sensibles sont stockées en clair dans les fichiers de configurations (mot de passe, identifiants...) et ces fichiers sont parfois partagés ou accessibles trop facilement. Il est donc nécessaire de les protéger.

- On crée et chiffre le fichier `secrets.yml`

```
$ rm -rf secrets.yml && ansible-vault create secrets.yml
New Vault password:

user: alex
pass: pass
```

Le chiffrement des fichiers

- Le contenu du fichier après le chiffrement

```
$ cat secrets.yml
$ANSIBLE_VAULT;1.1;AES256
61613037326133376338336562313764643935666
```

- Le contenu des variables une fois déchiffré

```
$ ansible-vault view secrets.yml
Vault password:
user: alex
pass: pass
```

Le chiffrement des fichiers

> Utilisation d'un fichier pour stocker le mot de passe de chiffrement

- Le fichier qui va contenir le mot de passe

```
$ echo alexandre > vault.key
```

- On créer un fichier `intranet-pass.yml`

```
user: alex
pass: pass
```

- Chiffrement d'un fichier avec le mot de passe contenu dans le fichier `vault.key`

```
$ ansible-vault encrypt intranet-pass.yml --vault-password-file vault.key
Encryption successful
```

Le chiffrement des fichiers

- Lecture du fichier chiffré contenant les variables

```
$ ansible -m debug -a var=user,pass -e @intranet-pass.yml \
        --vault-password-file vault.key localhost
```

```
localhost | SUCCESS => {
    "user,pass": "('alex', 'pass')"
}
```

> Attention de ne pas mettre le fichier mot de passe dans l'arborescence Git !

Le chiffrement des fichiers

- La variable `vault_password_file` peut être définie directement dans le fichier de configuration de Ansible

```
vault_password_file = /path/to/vault_password_file
```

- Création d'un fichier `ansible.cfg` dans le répertoire de travail avec le chemin du fichier password

```
$ cat ./ansible.cfg
```

```
[defaults]
...
vault_password_file = /root/ansible/ansible-practice/vault.key
...
```

Le chiffrement des fichiers

- Les commandes Ansible peuvent être utilisées directement sans option de chiffrement

```
$ ansible -m debug -a var=user,pass -e @intranet-pass.yml localhost
```

```
localhost | SUCCESS => {
    "user,pass": "('alex', 'pass')"
}
```

- Il est également possible d'exporter dans l'environnement courant, le nom du fichier mot de passe

```
$ export DEFAULT_VAULT_PASSWORD_FILE=vault.key
```

Le chiffrement des fichiers

- Modifier le fichier chiffré

```
$ ansible-vault edit intranet-pass.yml
```

- Changement du mot de passe de chiffrement (si `vault_password_file` n'est pas défini dans le fichier de configuration ansible)

```
$ ansible-vault rekey intranet-pass.yml
```

```
Vault password:
New Vault password:
Confirm New Vault password:
Rekey successful
```

Le chiffrement d'une chaine

Chiffrement d’un champ

> Depuis la version 2.3 de Ansible, il est possible de chiffrer uniquement des champs dans un fichier, ce qui est plutôt pratique pour gérer le contenu des fichiers.

- Création d'une chaîne chiffrée avec `encrypt_string`

```
$ ansible-vault encrypt_string 'sdq44sdOSKz!' \
--vault-password-file ./vault.key --encrypt-vault-id default
```

```
!vault |
          $ANSIBLE_VAULT;1.1;AES256
          343366336561636534363334326130313233366437 37[...]
Encryption successful
```

Le chiffrement d'une chaine

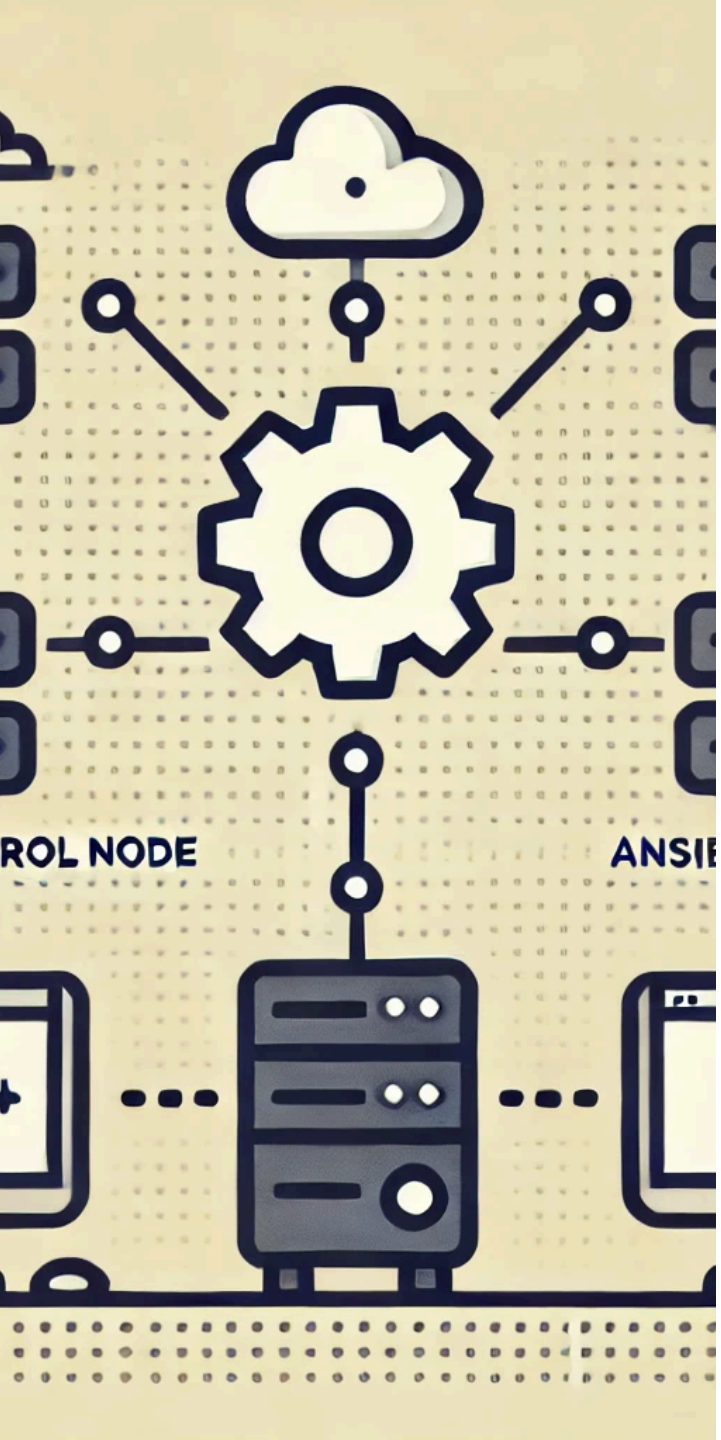
- Le fichier `partial-vault.yml` contenant le chiffrement du pass

```
user: "alex"
pass : !vault |
          $ANSIBLE_VAULT;1.1;AES256
          34336633656163653436333432613031323336643737[...]
```

- Affichage des variables user et pass

```
$ ansible -m debug -a var=user,pass localhost \
          -e @partial-vault.yml --vault-password-file ./vault.key
```

```
localhost | SUCCESS => {
    "user,pass": "(u'alex', u'sdq44sdOSKz!')"
}
```



Exercice

- Créer 2 fichiers chiffrés avec deux mots de passe différentes
  - "variable1: valeur_secrète_1" > secret1.yml
  - "variable2: valeur_secrète_2" > secret2.yml
- Les mots de passe sont dans
  - .ansible_vault_pass_secret1.txt
  - .ansible_vault_pass_secret2.txt

Exercice

- Créer le `playbook` qui permet d'afficher le `callback` suivant :

```
PLAY [Charger et afficher les variables depuis des fichiers chiffrés]

TASK [Afficher variable1]
ok: [localhost] => {
    "msg": "variable1 : valeur_secrète_1"
}

TASK [Afficher variable2]
ok: [localhost] => {
    "msg": "variable2 : valeur_secrète_2"
}

PLAY RECAP
localhost                  : ok=2
```

Le moteur de template Jinja

Introduction à Jinja2 dans Ansible

- Jinja2 est un moteur de templates utilisé par Ansible.
- Permet de générer dynamiquement des fichiers de configuration.
- Très utile pour personnaliser les fichiers selon l'environnement, l'inventaire, etc.

🔧 Utilisé dans :

- les fichiers `.j2`
- les tâches `template`
- les variables et conditions

Syntaxe de base Jinja2

- Délimiteurs :
  - `{{ variable }}` : expression
  - `{% instruction %}` : contrôle de flux
  - `{# commentaire #}` : commentaire

🟦 Exemple :

```
Bonjour {{ ansible_hostname }} !
{% if ansible_distribution == 'Ubuntu' %}
Ceci est une machine Ubuntu.
{% endif %}
```

Exemple avec le module template

📁 Fichier `template.j2` :

```
Nom de l'hôte : {{ inventory_hostname }}
```

📜 Tâche Ansible :

```
- name: Générer un fichier de configuration
  template:
    src: template.j2
    dest: /tmp/info.txt
```

Variables d'inventaire

🔍 Accès direct aux variables définies dans l’inventaire.

📂 `inventory.ini` :

```
[web]
web1 ansible_host=192.168.1.10 env=prod
```

📄 template.j2 :

```
Serveur : {{ inventory_hostname }}
Adresse IP : {{ ansible_host }}
Environnement : {{ env }}
```

Structures conditionnelles

✅ Permet d'ajouter de la logique dans vos templates.

📄 Exemple :

```
{% if ansible_os_family == 'Debian' %}
apt-get update
{% else %}
yum update
{% endif %}
```

Boucles for dans Jinja2

🔁 Utile pour générer des blocs répétés.

📄 Exemple :

```
{% for user in users %}
- {{ user.name }} ({{ user.role }})
{% endfor %}
```

📦 Variable :

```yaml
users:
  - name: alice
    role: admin
  - name: bob
    role: dev
```

Filtres dans Jinja2

🔧 Transforme des données en sortie.

📄 Exemples :

```
{{ name | upper }}
{{ path | replace("/home", "/data") }}
{{ items | length }}
```

Filtres les plus utilisés

- `default` : valeur par défaut
- `length` : taille d'une liste
- `join`, `split` : concaténation/séparation
- `replace`, `regex_replace` : substitution
- `to_nice_json`, `to_yaml` : formatage

🧪 Exemple :

```
{{ my_list | join(", ") }}
{{ var | default('valeur par défaut') }}
```

Exemple : Template systemd

`service.j2` :

```
[Unit]
Description={{ service_description }}

[Service]
ExecStart={{ exec_path }}

[Install]
WantedBy=multi-user.target
```

Itération sur un dictionnaire

📄 Exemple :

```
{% for key, value in config.items() %}
{{ key }} = {{ value }}
{% endfor %}
```

📦 Variable :

```yaml
config:
  host: localhost
  port: 3306
```

set_fact avec Jinja2

🛠️ Créer dynamiquement une variable.

```
- name: Définir un fait personnalisé
  set_fact:
    greeting: "Bonjour {{ ansible_hostname }}"
```

Exemple : /etc/hosts

`hosts.j2` :

```
{% for host in groups['all'] %}
{{ hostvars[host]['ansible_host'] }} {{ host }}
{% endfor %}
```

Test de valeurs booléennes

```
{% if use_ssl %}
SSL activé
{% else %}
SSL désactivé
{% endif %}
```

📦 Variables :

```
use_ssl: true
```

Masquer des secrets dans les templates

🔐 Utiliser `no_log` dans les tâches, ou Ansible Vault.

```
Mot de passe = {{ vault_password }}
```

🔒 Le contenu est chiffré avec `ansible-vault`.

lookup dans Jinja2

🔍 Permet de chercher des données externes.

```
{{ lookup('file', '/etc/hostname') }}
```

💡 Peut être combiné avec `template`, `set_fact`, etc.

Inclusion de templates

📄 Utiliser `import` ou `include`.

```
{% include 'header.j2' %}
Contenu principal
{% include 'footer.j2' %}
```

Gestion d'erreurs avec `default`

🚫 Évite les erreurs si une variable est absente.

```
{{ my_var | default('valeur par défaut') }}
```

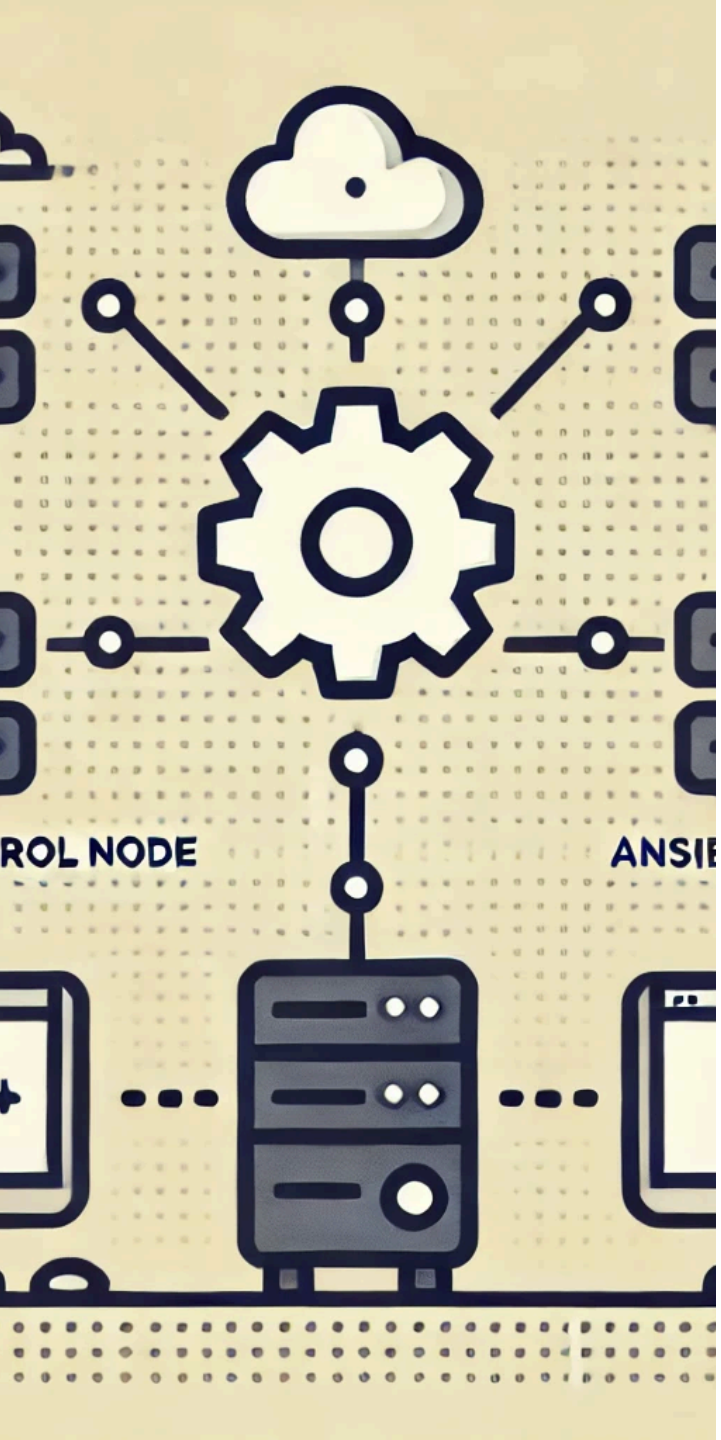
Exemple

```
{% if app_env == 'prod' %}
Serveur de production
{% elif app_env == 'staging' %}
Serveur de test
{% else %}
Serveur local
{% endif %}
```

Bonnes pratiques Jinja2

✅ Simplicité > Complexité
✅ Utiliser `default` partout
✅ Séparer logique & présentation

🎯 Objectif : des templates lisibles, sûrs et maintenables.



Exercice

Jinja2

Le moteur de template Jinja

- Un exemple de fichier **jinja.html.j2** avec la variable `inventory_hostname`

```
<html>
 <head>
        <title>Machine {{inventory_hostname}}</title>
 </head>
 <body>
        <p>Cette machine s’appelle {{inventory_hostname}}</p>
 </body>
</html>
```

Le moteur de template Jinja

- Le fichier de configuration utilisant le module `template` pour générer deux fichiers `HTML` contenant le nom des serveurs

```
$ cat jinja.yml

---
# Le moteur de template Jinja

- name: "Generate html file for each host"
  hosts: all
  connection: local
  tasks:
   - name: "html file generation"
     template:
      src: "jinja.html.j2"
      dest: "{{playbook_dir}}/{{inventory_hostname}}.html"
```

Le moteur de template Jinja

- Le fichier de configuration est joué sur l'inventaire créé précédemment

```
$ ansible-playbook jinja.yml
```

```
TASK [Gathering Facts] *********************************************************
ok: [1-1.practice-k8s.cloud]
ok: [1-2.practice-k8s.cloud]

TASK [html file generation] ****************************************************
ok: [1-2.practice-k8s.cloud]
ok: [1-1.practice-k8s.cloud]

PLAY RECAP *********************************************************************
1-1.practice-k8s.cloud     : ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
1-2.practice-k8s.cloud     : ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Le moteur de template Jinja

- Les deux fichiers `HTML` sont bien présents et contiennent respectivement la valeur des variables demandées (`inventory_hostname`)

```
$ grep title *.html
```

```
1-1.practice-k8s.cloud.html:    <title>Machine 1-1.practice-k8s.cloud</title>
1-2.practice-k8s.cloud.html:    <title>Machine 1-2.practice-k8s.cloud</title>
```

Les Templates Jinja / ansible_facts

> La notion de template va être illustrée en interrogeant les informations remontées avec le module `setup` (champ `gather_facts` et variable multi-valeurs `ansible_facts`)

Les `facts` fournissent un nombre impressionnant d'information sur les serveurs interrogés.

```
$ ansible  -m setup  localhost
```

> Le module `setup` retourne plus de 800 lignes !

> Attention, les nodes ont besoin de resoudre le nom du controler manager, cette condition peut ralentir le temps d'execussion du playbook

```
localhost | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "192.168.0.13",
            "192.168.122.1"
        ],
        "ansible_all_ipv6_addresses": [
            "fe80::3b02:2621:6bd0:8d4b"
        ],
        "ansible_apparmor": {
            "status": "enabled"
        },
        "ansible_architecture": "x86_64",
        "ansible_bios_date": "02/26/2018",
        "ansible_bios_version": "2.6.2",
        },
        "ansible_date_time": {
            "date": "2018-06-24",
            "day": "24",
```

Les Templates Jinja / ansible_facts

> old_facts.yaml `syntaxe obsolète`

```
tasks:
- name: show IP address
  debug:
    msg: >
      This host uses IP address {{ ansible_default_ipv4.address }}
```

> new_facts.yaml `syntaxe valide`

```
tasks:
- name: show IP address
  debug:
    msg: >
      This host uses IP address {{ ansible_facts['default_ipv4']['address'] }}
```

Les Templates Jinja

- Récupération des variables `ansible_all_ipv4_addresses` pour les afficher dans un template `Jinja`

```
$ cat template-net.html.j2
```

```
<html>
  <head>
    <title>Machine {{inventory_hostname}}</title>
  </head>
  <body>
    <p>Cette machine s’appelle {{inventory_hostname}}</p>
    <p>Ci-dessous la liste de ces adresses :</p>
    <ul>
     IP : {{ ansible_facts['default_ipv4']['address'] }}
    </ul>
  </body>
</html>
```

Les Templates Jinja

- Le fichier `jinja2.yml` qui fait appelle au fichier source `template-net.html.j2`

```
- name: "Generate html file for each host"
  hosts: all
  connection: local
  tasks:
   - name: "html file generation"
     template:
      src: "template-net.html.j2"
      dest: "{{playbook_dir}}/{{inventory_hostname}}.html"
```

- Le fichier de configuration est rejoué

```
$ ansible-playbook jinja2.yml
```

Les Templates Jinja

- Les deux fichiers `HTML` ont été modifiés pour intégrer les adresses `IP` dans le corps du fichier.

```
$ grep IP *.html
```

```
1-1.practice-k8s.cloud.html:     IP : 10.76.128.73
1-2.practice-k8s.cloud.html:     IP : 10.76.128.73
```

> On remarque ici que les adresses `IP` sont celles de `localhost`, pourquoi ?

Exécuter des tâches à plusieurs niveaux

Comment interroger un serveur distant et faire une action locale ?

> Un exemple pour illustrer ce concept : récupérer l'adresse `IP` d'un serveur et modifier le template `Jinja` en local.

- Le `jinja3.yml` de configuration modifié avec `connection: local`, qui a été déplacé dans la tâche pour qu'elle soit exécutée en local

```
- name: "Generate html file for each host"
  hosts: all
  tasks:
   - name: "html file generation"
     template:
      src: "template-net.html.j2"
      dest: "{{playbook_dir}}/{{inventory_hostname}}.html"
     connection: local
```

Comment interroger un serveur distant et faire une action locale ?

- Lancement du `playbook`

```
$ ansible-playbook jinja3.yml
```

```
TASK [Gathering Facts] *********************************************************
ok: [1-2.practice-k8s.cloud]
ok: [1-1.practice-k8s.cloud]

TASK [html file generation] ****************************************************
ok: [1-2.practice-k8s.cloud]
ok: [1-1.practice-k8s.cloud]

PLAY RECAP *********************************************************************
1-1.practice-k8s.cloud     : ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
1-2.practice-k8s.cloud     : ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

- Le fichier local a été mis à jour avec les informations des serveurs distants

```
$ grep IP *.html
```

```
1-1.practice-k8s.cloud.html:     IP : 10.75.40.101
1-2.practice-k8s.cloud.html:     IP : 10.76.132.143
```