# Ansible

**Ansible** est un outil open source conçu pour automatiser la gestion et la configuration des systèmes informatiques. Il permet de déployer des applications, configurer des serveurs, et orchestrer des tâches répétitives sans intervention manuelle.

Ansible facilite ainsi le travail des administrateurs systèmes en rendant les opérations plus rapides, fiables, et reproductibles.

# Présentation


# Qu’est-ce qu’Ansible ?

**Ansible** est un outil open source conçu pour automatiser la gestion et la configuration des systèmes informatiques. Il permet de déployer des applications, configurer des serveurs, et orchestrer des tâches répétitives sans intervention manuelle.

Ansible facilite ainsi le travail des administrateurs systèmes en rendant les opérations plus rapides, fiables, et reproductibles.

# À quoi sert Ansible ?

Ansible sert à automatiser les déploiements et la configuration de serveurs, qu’ils soient physiques, virtuels ou dans le cloud. Il permet aussi de gérer des mises à jour, installer des logiciels, ou orchestrer des processus complexes.

L’objectif est d’éviter les erreurs humaines, de gagner du temps, et d’assurer une cohérence sur tous les environnements.

# Comment fonctionne Ansible ?

Ansible fonctionne sans agent : il se connecte à distance aux machines via SSH pour exécuter des commandes. Les actions sont définies dans des fichiers appelés **playbooks**, écrits en YAML, qui décrivent les tâches à réaliser.

Ces playbooks sont lisibles et compréhensibles, ce qui facilite la maintenance et le partage des configurations.

# Pourquoi utiliser Ansible ?

Ansible est apprécié pour sa simplicité d'utilisation et son approche sans agent, qui évite l'installation de logiciels supplémentaires sur les machines gérées. Il est puissant, flexible, et peut gérer des infrastructures de toutes tailles.

Il favorise l'automatisation progressive, ce qui aide les équipes à standardiser leurs environnements.

# Cas d'usage courants

Ansible est utilisé pour :

- Installer et configurer des serveurs automatiquement,
- Déployer des applications ou des mises à jour,
- Gérer la configuration réseau,
- Orchestrer des tâches complexes multi-serveurs,
- Automatiser les sauvegardes et la sécurité.

# En résumé

**Ansible** est un outil simple et efficace pour automatiser la gestion de votre infrastructure. Il améliore la productivité, réduit les erreurs, et facilite la gestion à grande échelle.

# Installation


## Mise à jour

Dans un premier temps, il faut mettre à jour le système et les paquets :

```
apt update && apt upgrade -y
```

## Installation

Les mises à jour faites, on peut passer à l'installation à proprement parlé.

### Prérequis

Commencer par ajouter les paquets prérequis:

```
apt install -y software-properties-common
```

### Depots

On va à présent ajouter les dépots Ansible :

```
echo "deb http://ppa.launchpad.net/ansible/ansible/ubuntu focal main" | tee
/etc/apt/sources.list.d/ansible.list
```

ainsi que de la clé GPG du dépot :

```
apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 93C4A3FD7BB9C367
```

## Installation

Nous y voila enfin, l'installation du paquet voulu !

```
apt update && apt install ansible -y
```

# Vérification

Il faut vérifier la version installée :

```
ansible --version
```

Le serveur Ansible est installé et fonctionnel !

Dans une prochaine page, nous verrons comment le configurer et l'utiliser.

# Playbooks

# Playbook - copie d'un script

## Avant propos

Cette procédure aura pour exemple concret, la copie d'un script, ainsi que de la mise en place d'une tâche cron qui éxecute ce script.

## Echange de clés SSH

Dans un premier temps, pour se simplifier la vie, nous allons faire un échange de clé ssh, pour ne pas avoir à entrer identifiant et mot de passe chaque fois.

### Générer une clé

On va commencer par générer une clé SSH pour notre serveur Ansible.

```
ssh-keygen -t rsa -b 4096 -f ~/.ssh/id_rsa -C "root@Ansible"
```

### Copier la clé

On peut maintenant copier la clé sur les serveurs à gérer.

```
ssh-copy-id -i ~/.ssh/id_rsa.pub <user>@<IP>
```

Avec cette commande vient la demande d'acceptation de la fingerprint, et on tape ensuite le mot de passe pour valider la copie.

# Arboréscence

Le plus simple pour utiliser Ansible est d'avoir une aboréscence simple :

```
/Ansible/
├── inventory
├── playbook.yml
└── files/
    └── docker_update.sh
```

Le répertoire Ansible est à la racine du système.
Le fichier inventory, permet de lister l'ensemble des machines gérées, ainsi que de les placer dans divers groupes.
Le fichier playbook.yml contient les tâches à effectuer.
Le répertoire files permet de contenir l'ensemble des ressources utiles : scripts etc

# Fichiers

## Inventory

```
[VM_Docker]
# Groupe d’hôtes appelé "VM_Docker" : liste des machines sur lesquelles Ansible va agir

Supervisor      ansible_host=<IP>
Vaultwarden     ansible_host=<IP>
RustDesk        ansible_host=<IP>
Servarr         ansible_host=<IP>
Immich          ansible_host=<IP>
NTFY            ansible_host=<IP>
Mealie          ansible_host=<IP>
Firefly         ansible_host=<IP>
FreshRSS        ansible_host=<IP>

[all:vars]
# Variables globales appliquées à tous les hôtes de tous les groupes

ansible_user=<user>
```

```
# Utilisateur SSH utilisé par Ansible pour se connecter aux machines (remplace <user> par le nom d’utilisateur réel)

ansible_become=true
# Active l’élévation de privilèges (exécution des commandes avec sudo par exemple)
# Utile si l’utilisateur SSH n’est pas root, mais doit exécuter des commandes administratives
```

- **[VM_Docker]** : groupe contenant la liste des machines où déployer le script.
- **ansible_host** : adresse IP de chaque machine.
- **[all:vars]** : variables communes à tous les hôtes.
- **ansible_become** : Demande à Ansible d’élever les privilèges, c’est-à-dire d’utiliser sudo.

## playbook.yml

```
---
- name: "Copier le script docker_update.sh et planifier son exécution"
  # Nom global du playbook, indique l'objectif général : copier le script puis planifier son exécution automatique
  hosts: VM_Docker
  # Indique que les tâches seront exécutées sur tous les hôtes du groupe VM_Docker
  tasks:
  # Liste des tâches à exécuter dans l'ordre

    - name: "S' assurer que le dossier /Docker existe"
      # Tâche 1 : garantir la présence du dossier /Docker sur chaque machine cible
      file:
        path: /Docker
        # Chemin du dossier à vérifier ou créer
        state: directory
        # On veut un dossier (et non un fichier)
        owner: root
        # Propriétaire du dossier
        group: root
        # Groupe propriétaire du dossier
        mode: '0755'
        # Permissions : rwxr-xr-x (propriétaire lecture-écriture-exécution, groupe et autres lecture-exécution)

    - name: "Copier docker_update.sh uniquement s’il n’existe pas"
      # Tâche 2 : copier le script shell vers /Docker/docker_update.sh si ce fichier n'existe pas déjà
```

```
      copy:
        src: files/docker_update.sh
        # Chemin source local du script à copier (dans le dossier files de ton projet Ansible)
        dest: /Docker/docker_update.sh
        # Destination sur la machine distante
        owner: root
        group: root
        mode: '0755'
        # Le script sera exécutable
        force: no
        # Ne pas écraser le fichier s'il est déjà présent (évite d'écraser une version
modifiée)

    - name: Ajouter un cronjob pour lancer le script chaque jour à 3h
      # Tâche 3 : ajouter une tâche cron qui lance le script tous les jours à 3h du matin
      cron:
        name: "Mise à jour Docker quotidienne"
        # Nom lisible de la tâche cron (pour repérer la tâche plus facilement)
        user: root
        # La tâche sera lancée avec l'utilisateur root
        minute: "0"
        # Minute de lancement (00)
        hour: "3"
        # Heure de lancement (3h du matin)
        job: "/Docker/docker_update.sh"
        # Commande exécutée : lancement du script docker_update.sh
```

## docker_update.sh

```
#!/bin/bash
# Indique que le script doit être exécuté avec bash

BASE_DIR="/Docker"
# Répertoire racine où sont situés les dossiers des différents projets Docker

TOPIC="Docker_Updates"
# Nom du sujet (topic) pour la notification ntfy

NTFY_URL="https://ntfy.rakouns.bzh"
# URL du serveur ntfy pour envoyer les notifications
```

```
timestamp=$(date +"%Y-%m-%d %H:%M:%S")
# Date et heure actuelles, formatées pour le rapport

report="Rapport de mise à jour Docker - $timestamp\n\n"
# Initialisation de la variable rapport avec un titre et un saut de ligne

for dir in "$BASE_DIR"/*/; do
    # Boucle sur chaque sous-dossier du répertoire /Docker (un dossier par projet Docker)

    container_name=$(basename "$dir")
    # Extraction du nom du dossier (nom du conteneur/service Docker)

    # Trouver le bon fichier docker-compose
    if [[ -f "$dir/docker-compose.yaml" ]]; then
        compose_file="$dir/docker-compose.yaml"
    elif [[ -f "$dir/docker-compose.yml" ]]; then
        compose_file="$dir/docker-compose.yml"
    else
        # Ignorer les répertoires sans fichier docker-compose (pas un projet Docker valide)
        continue
    fi

    # Pull update : récupération des dernières images Docker du fichier compose
    pull_output=$(docker compose -f "$compose_file" pull 2>&1)
    pull_exit=$?
    # Capture la sortie et le code de retour de la commande "docker compose pull"

    # Up update : relance des conteneurs en mode détaché avec les nouvelles images
    up_output=$(docker compose -f "$compose_file" up -d 2>&1)
    up_exit=$?
    # Capture la sortie et le code de retour de la commande "docker compose up -d"

    # Résumé avec ou sans erreurs
    if [[ $pull_exit -eq 0 && $up_exit -eq 0 ]]; then
        # Si les deux commandes ont réussi, ajouter une coche verte au rapport
        report+="$container_name : \n"
    else
        # Sinon, coche rouge et détails des erreurs
```

```
        report+="$container_name : □\n"
        if [[ $pull_exit -ne 0 ]]; then
            # Ajouter les 5 dernières lignes de la sortie pull en cas d'erreur
            report+="  Erreur pull :\n$(echo "$pull_output" | tail -n 5)\n"
        fi
        if [[ $up_exit -ne 0 ]]; then
            # Ajouter les 5 dernières lignes de la sortie up en cas d'erreur
            report+="  Erreur up :\n$(echo "$up_output" | tail -n 5)\n"
        fi
    fi
done

# Envoi de la notification condensée
printf "$report" | curl -s -X POST \
    -H "Title: Rapport de mise à jour Docker" \
    -H "Priority: 5" \
    --data-binary @- \
    "$NTFY_URL/$TOPIC"
# Envoie le rapport via curl en POST à ntfy, avec un titre et une priorité,
# la donnée est envoyée depuis la variable report
```

# Execution

On va pouvoir maintenant lancer le playbook, sur le groupe souhaité.
Pour cela, rien de plus simple, on tape la commande suivante :

```
ansible-playbook playbook.yml -i inventory -l VM_Docker
```

On à alors un retour qui ressemble à ça :

```
PLAY ["Copier le script docker_update.sh et planifier son exécution"]
*********************************************************************************************
****************************************

TASK [Gathering Facts]
*********************************************************************************************
****************************************
ok: [RustDesk]
ok: [Immich]
```

```
ok: [Servarr]
ok: [Supervisor]
ok: [NTFY]
ok: [Mealie]
ok: [FreshRSS]
ok: [Firefly]
ok: [Vaultwarden]

TASK ["S'assurer que le dossier /Docker existe"]
*********************************************************************************************
*****************************************
ok: [Servarr]
ok: [Vaultwarden]
ok: [Immich]
ok: [RustDesk]
ok: [Supervisor]
ok: [Mealie]
ok: [NTFY]
ok: [Firefly]
ok: [FreshRSS]

TASK ["Copier docker_update.sh uniquement s’il n’existe pas"]
*********************************************************************************************
*****************************************
ok: [Servarr]
ok: [RustDesk]
ok: [Immich]
ok: [Supervisor]
ok: [NTFY]
ok: [Mealie]
ok: [FreshRSS]
ok: [Firefly]
changed: [Vaultwarden]

TASK ["Ajouter un cronjob pour lancer le script chaque jour à 3h"]
*********************************************************************************************
*****************************************
ok: [RustDesk]
ok: [Servarr]
ok: [Immich]
```

```
changed: [Vaultwarden]
ok: [Supervisor]
ok: [NTFY]
ok: [Mealie]
ok: [Firefly]
ok: [FreshRSS]

PLAY RECAP
*********************************************************************************************
*****************************************
Firefly                    : ok=4    changed=0    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0
FreshRSS                   : ok=4    changed=0    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0
Immich                     : ok=4    changed=0    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0
Mealie                     : ok=4    changed=0    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0
NTFY                       : ok=4    changed=0    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0
RustDesk                   : ok=4    changed=0    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0
Servarr                    : ok=4    changed=0    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0
Supervisor                 : ok=4    changed=0    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0
Vaultwarden                : ok=4    changed=2    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0
```

On peut vérifier que le script est lancé en se rendant sur une machine cible, et vérifier la présence de la tâche cron :

```
crontab -l
```

Ainsi que de la présence du script :

```
ls -la /Docker
```

Normalement, Ansible ne se trompe pas lorsqu'il affiche un status ok, autrement, le score de failed n'aurait ps été de 0.