# AWX

# Démarrage - Première connexion

# ? 1. Première connexion

Accède à AWX via ton service :

kubectl get svc -n awx

□□ URL = `http://<node-ip>:30080` (dans ton cas)

Login :

- user : `admin`
- password :

kubectl get secret -n awx awx-admin-password -o jsonpath="{.data.password}" | base64 -d

---

# ?? 2. Configuration minimale (à faire tout de suite)

Dans l'UI AWX :

## ? Settings ? System

- Time zone → `Europe/Paris`
- (optionnel) définir un base URL si tu exposes proprement

---

# ? 3. Créer une Credential SSH

C’est la base de tout.

## ? Resources ? Credentials ? Add

Type : **Machine**

Remplis :

- Name : `ssh-homelab`
- Username : ton user (`root` ou autre)
- Private key : ta clé SSH

□□ Exemple :

cat ~/.ssh/id_rsa

Colle la clé privée.

---

# ?? 4. Créer un Inventory

## ? Resources ? Inventories ? Add

- Name : `homelab`

Puis :

## ? Hosts ? Add

Exemple :

name: node1
variables:
ansible_host: 10.151.151.10
ansible_user: root

□□ Tu peux aussi ajouter un groupe :

- `k8s`
- `vm`
- `nas`

# ? 5. Créer un Project (Git)

AWX bosse avec Git, pas avec des fichiers locaux.

## ? Resources ? Projects ? Add

- Name : `homelab-infra`
- Source Control Type : Git
- URL : ton repo (GitHub / Gitea / GitLab)

□□ Si privé :
→ ajoute une credential Git (token ou clé SSH)

---

# ?? 6. Créer un Job Template

C’est là que tout s’exécute.

## ? Resources ? Templates ? Add ? Job Template

Remplis :

- Name : `Ping test`
- Inventory : `homelab`
- Project : `homelab-infra`
- Playbook : `ping.yml`
- Credentials : `ssh-homelab`

---

# ? 7. Exemple de playbook

Dans ton repo Git :

# ping.yml
- hosts: all
gather_facts: false
tasks:

- name: Test ping
ping:

---

# ?? 8. Lancer un job

Clique sur □□

□□ Si tout est bon :

- hosts OK
- SSH OK
- repo OK

➡ tu vois les résultats en live

---

# ? 9. Bonus utiles (important en homelab)

## ?? Variables globales (Inventory ? Variables)

ansible_python_interpreter: /usr/bin/python3

---

## ?? Tester la connectivité

Ajoute un playbook :

- hosts: all
tasks:
- command: hostname

---

## ?? Ajouter sudo

Dans Credential :

- Privilege escalation : ✔
- Method : sudo

---

# ? 10. Bonnes pratiques dès le début

□□ Tu es déjà en GitOps avec ArgoCD, donc :

- ✔ versionne :
  - inventories (via SCM)
  - projets
  - playbooks
- □ évite :
  - config manuelle non trackée

---

# ? Ce que tu viens de construire

Avec ça tu as :

- AWX = orchestrateur
- Git = source de vérité
- SSH = accès machines
- Kubernetes = infra

□□ Tu peux maintenant :

- déployer des apps
- configurer tes nodes
- gérer ton cluster

# Création d'un Token API AWX et Commandes Utiles

L'API REST d'AWX permet d'automatiser et de piloter l'intégralité de votre infrastructure sans passer par l'interface web. Pour interagir avec cette API en toute sécurité, il est nécessaire de générer un **Personal Access Token (PAT)**.

---

# 1. Créer un Token d'Accès dans AWX

La génération du token se fait une seule fois depuis l'interface web d'AWX :

1. Connectez-vous à l'interface web d'AWX avec votre compte administrateur.
2. Dans le menu latéral de gauche, allez dans **Access** > **Users**.
3. Cliquez sur votre nom d'utilisateur (ex: *admin*).
4. Allez dans l'onglet **Tokens** en haut de la page.
5. Cliquez sur le bouton **Add** (Ajouter).
6. Remplissez le formulaire :
   - **Description :** Donnez un nom clair (ex: *Token Script Bash Homelab*).
   - **Scope (Portée) :** Sélectionnez **Write** (si vous avez besoin de lancer des jobs) ou **All** .
7. Cliquez sur **Save**.

> ⚠️**ATTENTION :** Le Token généré (une longue chaîne de caractères) ne s'affichera **qu'une seule fois**. Copiez-le immédiatement et conservez-le dans un endroit sûr (comme un gestionnaire de mots de passe ou Vaultwarden).

---

# 2. Comment utiliser le Token

Pour chaque requête API (souvent via `curl`), vous devez passer ce token dans les en-têtes (Headers) HTTP pour prouver votre identité.

```
# Structure de base de l'authentification
curl -H "Authorization: Bearer VOTRE_TOKEN_ICI" https://awx.domaine.lan/api/v2/...
```

---

# 3. Antisèche (Cheat Sheet) des requêtes API utiles

Voici quelques commandes Bash prêtes à l'emploi. Pensez à adapter `$AWX_HOST` et `$TOKEN` avec vos valeurs.

## A. Trouver l'ID d'un Job Template

Avant de lancer un playbook, vous avez besoin de connaître son ID. Cette commande liste tous les templates avec leur nom et leur ID.

```
curl -s -k -H "Authorization: Bearer $TOKEN" \
     -X GET "$AWX_HOST/api/v2/job_templates/" \
     | jq '.results[] | "ID: \(.id) - Nom: \(.name)"'
```

*Note : L'outil `jq` est utilisé ici pour formater la sortie JSON de manière lisible.*

## B. Lancer un Job Template immédiatement

Permet de déclencher un playbook à la demande (très utile pour l'intégrer dans d'autres scripts ou pipelines).

```
TEMPLATE_ID="42" # Remplacez par votre ID

curl -s -k -X POST "$AWX_HOST/api/v2/job_templates/$TEMPLATE_ID/launch/" \
     -H "Authorization: Bearer $TOKEN" \
     -H "Content-Type: application/json" \
     -d '{}'
```

## C. Lancer un Job Template avec une Limit spécifique

Idéal si vous voulez exécuter un playbook générique (comme "Setup SSH") sur une seule machine spécifique via l'API.

```
TEMPLATE_ID="42"
MACHINE_CIBLE="bookstack"
```

```
curl -s -k -X POST "$AWX_HOST/api/v2/job_templates/$TEMPLATE_ID/launch/" \
     -H "Authorization: Bearer $TOKEN" \
     -H "Content-Type: application/json" \
     -d '{
         "limit": "'"$MACHINE_CIBLE"'"
     }'
```

# D. Vérifier le statut du dernier Job exécuté

Permet de savoir si le job s'est bien passé (successful) ou s'il a échoué (failed).

```
TEMPLATE_ID="42"

curl -s -k -H "Authorization: Bearer $TOKEN" \
     -X GET "$AWX_HOST/api/v2/job_templates/$TEMPLATE_ID/jobs/?order_by=-id&page_size=1" \
     | jq '.results[0] | "Job ID: \(.id) - Statut: \(.status)"'
```

# Planifier un Job Template AWX via l'API

L'interface graphique d'AWX (Tower) peut parfois s'avérer capricieuse ou peu intuitive pour configurer des tâches planifiées complexes. Utiliser l'API REST via une simple commande `curl` permet de créer des planifications précises, rapides et reproductibles.

---

# 1. Le Script de Planification

Copiez ce code dans un terminal Linux (ou dans un script bash) pour créer la planification.
**Attention à bien modifier les variables avant de l'exécuter.**

```
#!/bin/bash

# ========================================
# Variables de configuration
# ========================================
AWX_HOST="https://awx.numericare.fr"
TOKEN="ton_token_api_ici"
TEMPLATE_ID="42"

# ========================================
# Appel API
# ========================================
curl -X POST "$AWX_HOST/api/v2/job_templates/$TEMPLATE_ID/schedules/" \
    -H "Authorization: Bearer $TOKEN" \
    -H "Content-Type: application/json" \
    -d '{
        "name": "Conversion Media Quotidienne",
        "description": "Exécution automatique 2 fois par jour",
        "rrule": "DTSTART:20240101T030000Z
RRULE:FREQ=DAILY;INTERVAL=1;BYHOUR=3,15;BYMINUTE=0",
        "extra_data": {},
        "inventory": null
    }'
```

# 2. Explication des Variables

Voici comment récupérer et adapter les valeurs nécessaires pour que la commande fonctionne avec votre environnement :

| Variable | Description & Comment la trouver |
|---|---|
| `AWX_HOST` | L'URL de base de votre serveur AWX. *Exemple : https://awx.mon-homelab.lan* |
| `TOKEN` | Votre jeton d'accès personnel. Pour le générer dans AWX : allez dans **Users** > Cliquez sur votre utilisateur > Onglet **Tokens** > **Add** (Cochez "Write" ou "All"). |
| `TEMPLATE_ID` | L'identifiant unique du playbook à lancer. Pour le trouver, ouvrez votre Job Template dans AWX et regardez l'URL : `/templates/job_template/42/details`. Ici, l'ID est 42. |

# 3. Comprendre la syntaxe `rrule` (La Planification)

Le paramètre le plus complexe de la requête JSON est le `rrule` (Recurrence Rule). Il utilise le standard iCalendar. Voici comment le décoder et le modifier :

> **Syntaxe utilisée :** `DTSTART:20240101T030000Z RRULE:FREQ=DAILY;INTERVAL=1;BYHOUR=3,15;BYMINUTE=0`

- **DTSTART:** Définit la date et l'heure de début de validité de la règle (au format UTC, indiqué par le `Z`). Mettre une date dans le passé fonctionne très bien pour que la règle soit active immédiatement.
- **FREQ=DAILY :** La fréquence de base. Peut être remplacé par `HOURLY` (Toutes les heures), `WEEKLY` (Toutes les semaines), ou `MONTHLY` (Tous les mois).
- **INTERVAL=1 :** S'applique tous les *X* jours (car FREQ=DAILY). Si on met 2, le script tournera un jour sur deux.
- **BYHOUR=3,15 :** Exécute la tâche à 03h00 et à 15h00 **(Heure UTC)**. Attention au décalage horaire selon votre fuseau !
- **BYMINUTE=0 :** S'assure que l'exécution se fait à l'heure pile (00 minute).

# Autres exemples de `rrule` utiles :

| | |
|---|---|
| **Toutes les 4 heures :** | `... RRULE:FREQ=HOURLY;INTERVAL=4` |
| **Tous les lundis à 2h du matin :** | `... RRULE:FREQ=WEEKLY;INTERVAL=1;BYDAY=MO;BYHOUR=2;BYMINUTE=0` |