BTS SIO — Option SISR

DOSSIER TECHNIQUE E6

Supervision proactive et gestion unifiée du parc (MCO)

Candidat : SIDICINA Nathan

Session : 2026

Option : Solutions d’Infrastructure, Systèmes et Réseaux (SISR)

Environnement : maquette de validation en pré-production (VMware ESXi)

SOMMAIRE

I. Contexte et enjeux du projet
II. Architecture et choix techniques
III. Mise en œuvre — Phase 1 : l'inventaire automatisé (GLPI-Agent)
IV. Mise en œuvre — Phase 2 : la supervision temps réel (Zabbix)
V. Mise en œuvre — Phase 3 : l'interconnexion Zabbix-GLPI
VI. Validation et recette
VII. Maintien en condition opérationnelle (MCO)
VIII. Bilan et analyse critique
IX. Annexes

I. Contexte et enjeux du projet

1.1 L'environnement de l'entreprise

L'entreprise STEP a connu une croissance rapide ces dernières années. Le parc informatique s'est élargi et la gestion manuelle est devenue un vrai problème. L'inventaire était tenu sur des fichiers Excel, souvent incomplets ou obsolètes, et les interventions ne se déclenchaient que lorsqu'un utilisateur signalait un problème, parfois plusieurs heures après l'incident.

La direction a demandé une solution pour fiabiliser le suivi du parc et détecter les pannes avant qu'elles n'impactent la production.

1.2 Le besoin

Deux besoins distincts ont été identifiés :

- Un inventaire fiable et automatisé : plus de saisie manuelle, chaque équipement doit être recensé automatiquement avec ses caractéristiques réelles (OS, CPU, RAM, adresse réseau).
- Une supervision proactive : être alerté en temps réel dès qu'un seuil critique est dépassé, sans attendre l'appel de l'utilisateur.

Pour répondre à ces deux besoins, j'ai choisi d'utiliser deux outils open source et de les interconnecter : GLPI pour la gestion du parc et le helpdesk, Zabbix pour la supervision.

1.3 Compétences mobilisées

Ce projet couvre plusieurs compétences du référentiel SISR : gestion du patrimoine informatique (inventaire automatisé, CMDB), supervision et disponibilité des services, traitement des incidents (workflow ITSM), sécurisation des échanges inter-applicatifs via API REST, et automatisation des tâches d'administration.

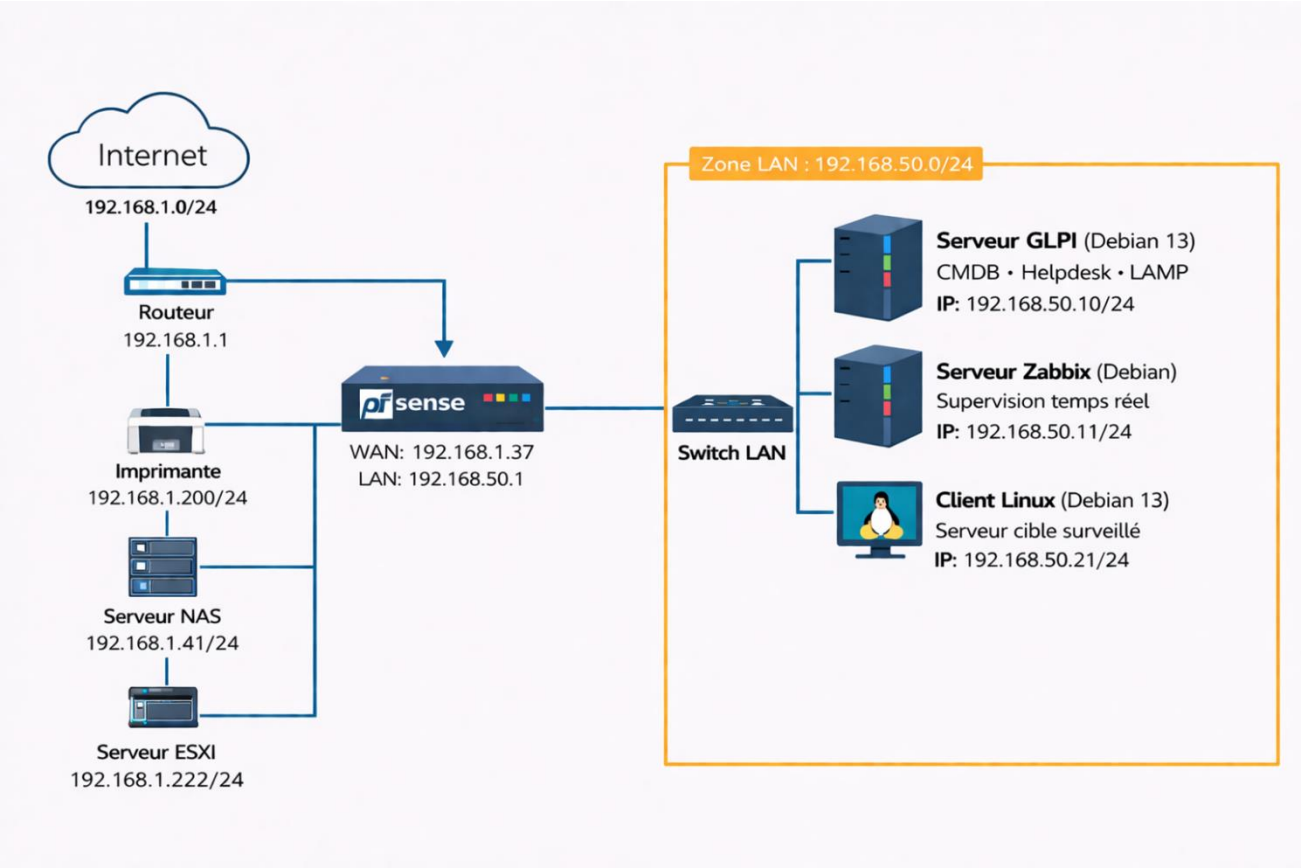
II. Architecture et choix techniques

2.1 Environnement de validation

Pour ne pas toucher à la production, j'ai reproduit l'infrastructure sur une maquette virtualisée sous VMware ESXi. Cela m'a permis de tester chaque étape sans risque.

La maquette comprend :

- Serveur GLPI (Debian 13) : héberge la base de données d'inventaire (CMDB) et le portail helpdesk. Adresse : 192.168.50.10/24.
- Serveur Zabbix (Debian) : collecte les métriques en temps réel et gère les alertes. Adresse : 192.168.50.11/24.
- Client Linux surveillé (Debian 13) : simule un serveur de production critique. Adresse : 192.168.50.21/24.
- Pare-feu : sépare la zone WAN (192.168.1.0) du réseau LAN interne (192.168.50.0/24).



2.2 Pourquoi ces deux outils ?

J'ai choisi GLPI et Zabbix pour plusieurs raisons. Les deux sont open source, donc sans coût de licence, ce qui correspond au contexte d'une PME. GLPI est très utilisé en France pour la gestion de parc et dispose d'une API REST native qui permet l'intégration avec des outils tiers. Zabbix offre un système de webhook natif qui peut appeler cette API directement, sans plugin supplémentaire.

Par rapport à Nagios par exemple, Zabbix est plus simple à configurer pour ce type d'intégration et ne nécessite pas de plugins payants. PRTG est propriétaire et limité en version gratuite.

2.3 Choix technique : agent actif ou SNMP ?

Pour remonter les informations des équipements, j'ai adopté une approche hybride :

- Sur les serveurs Linux : agent Zabbix en mode actif. C'est l'agent qui initie la connexion vers le serveur Zabbix, ce qui évite d'ouvrir des ports entrants sur les machines surveillées. Le mode actif permet aussi une remontée fine des métriques système (CPU, RAM, disque, état des services).
- Sur les équipements réseau (switchs) : protocole SNMP v3. Contrairement au SNMP v2c qui transmet la community string en clair, le SNMP v3 ajoute l'authentification et le chiffrement des trames.

III. Mise en œuvre — Phase 1 : l'inventaire automatisé

3.1 Objectif

Remplacer les fichiers Excel par une CMDB automatiquement mise à jour. Dès qu'un agent est installé sur une machine, ses caractéristiques doivent remonter dans GLPI sans aucune saisie manuelle.

3.2 Installation de l'agent GLPI sur la machine cible

Étape 1 — Connexion et préparation

On se connecte en SSH sur la machine cible (Debian 13) et on passe en root :

```
glpi@debian13:~$ su Mot de passe :
```

```
client@debian-srv:~$ su -
Mot de passe :
root@debian-srv:~#  apt update
Atteint :1 http://deb.debian.org/debian bookworm InRelease
Réception de :2 http://security.debian.org/debian-security bookworm-security InR
elease [48,0 kB]
Réception de :3 http://deb.debian.org/debian bookworm-updates InRelease [55,4 kB
]
Réception de :4 http://security.debian.org/debian-security bookworm-security/mai
n Sources [184 kB]
Réception de :5 http://security.debian.org/debian-security bookworm-security/mai
n amd64 Packages [294 kB]
Réception de :6 http://security.debian.org/debian-security bookworm-security/mai
n Translation-en [178 kB]
760 ko réceptionnés en 1s (560 ko/s)
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
Lecture des informations d'état... Fait
28 paquets peuvent être mis à jour. Exécutez « apt list --upgradable » pour les
voir.
root@debian-srv:~#
```

Étape 2 — Lancement de l'installation

On lance le script d'installation de l'agent GLPI v1.15 en lui indiquant l'adresse du serveur GLPI :

```
perl glpi-agent-1.15-linux-installer.pl --
server=http://192.168.50.10/glpi/front/inventory.php --install --runnow
```

Le paramètre --server indique où envoyer l'inventaire. Le paramètre --runnow force un premier scan immédiat sans attendre le prochain cycle.

```
root@debian-srv:~# wget https://github.com/glpi-project/glpi-agent/releases/down
load/1.16/glpi-agent-1.16-linux-installer.pl
```

```
root@debian-srv:~# perl glpi-agent-1.16-linux-installer.pl --server=http://192.1
68.50.10/front/inventory.php --install --runnow
Installing glpi-agent v1.16...
```

Étape 3 — Vérification de l'installation

Une fois l'installation terminée, on vérifie que le service est bien actif :

```
systemctl status glpi-agent
```

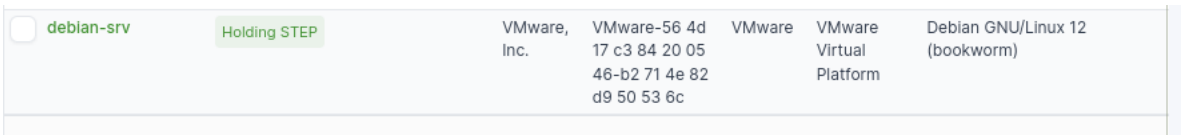
```
root@debian-srv:~#
root@debian-srv:~# systemctl status glpi-agent
● glpi-agent.service - GLPI agent
     Loaded: loaded (/lib/systemd/system/glpi-agent.service; enabled; preset: e>
     Active: active (running) since Mon 2026-03-23 19:36:16 CET; 50s ago
       Docs: man:glpi-agent
   Main PID: 6114 (glpi-agent: wai)
      Tasks: 1 (limit: 2235)
     Memory: 70.6M
        CPU: 440ms
     CGroup: /system.slice/glpi-agent.service
             └─6114 "glpi-agent: waiting"

mars 23 19:36:16 debian-srv glpi-agent[6114]: [info] GLPI Agent starting
mars 23 19:36:16 debian-srv glpi-agent[6114]: [info] [http server] HTTPD servic>
mars 23 19:36:16 debian-srv glpi-agent[6114]: [info] target server0: next run: >
mars 23 19:36:17 debian-srv systemd[1]: glpi-agent.service: Sent signal SIGUSR1>
mars 23 19:36:17 debian-srv glpi-agent[6114]: [info] GLPI Agent requested to ru>
mars 23 19:36:17 debian-srv glpi-agent[6114]: [info] target server0: server htt>
mars 23 19:36:17 debian-srv glpi-agent[6114]: [info] sending prolog request to >
mars 23 19:36:17 debian-srv glpi-agent[6114]: [error] [http client] communicati>
mars 23 19:36:17 debian-srv glpi-agent[6114]: [error] No supported answer from >
mars 23 19:36:17 debian-srv glpi-agent[6114]: [info] target server0: next run: >
lines 1-21/21 (END)
```

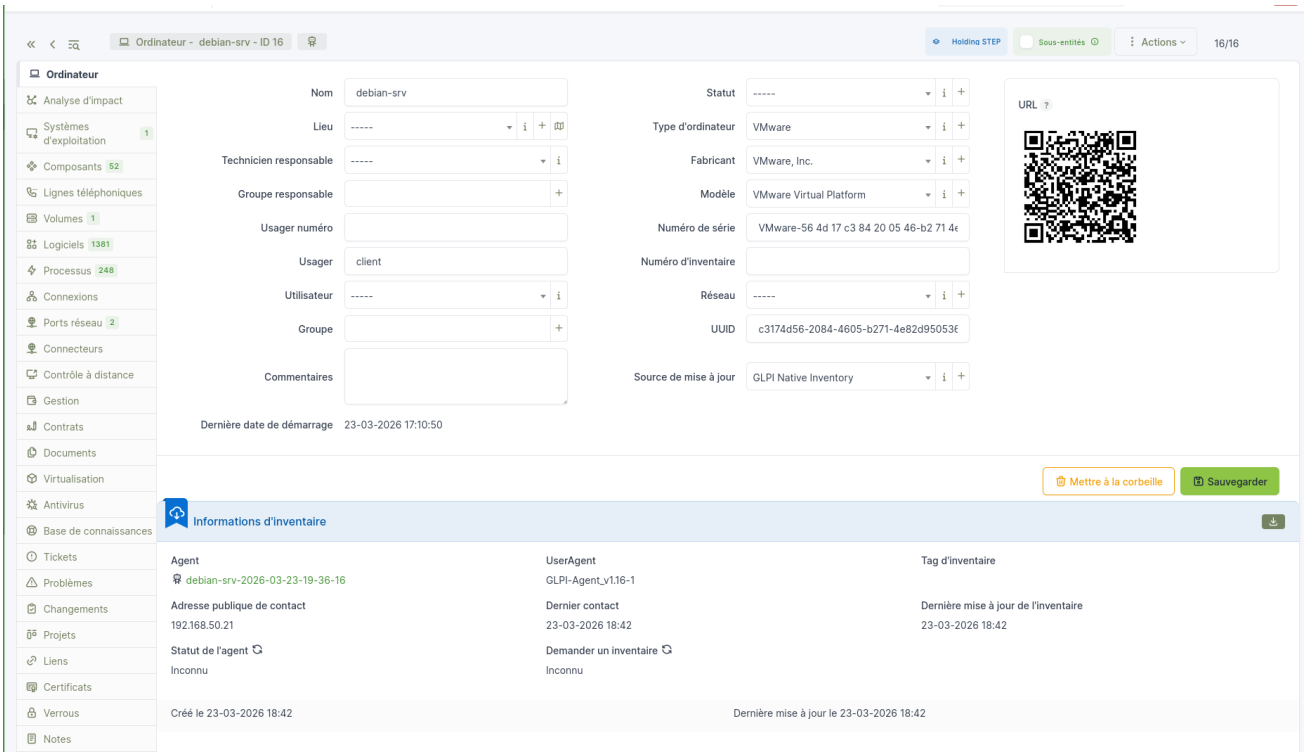
Le message « Asking service to run inventory now as requested » confirme que le premier inventaire a bien été envoyé au serveur GLPI.

```
root@debian-srv:~# perl glpi-agent-1.16-linux-installer.pl --server=http://192.1
68.50.10/glpi/front/inventory.php --install --runnow
Installing glpi-agent v1.16...
Applying configuration...
Enabling glpi-agent service...
Asking service to run inventory now as requested...
root@debian-srv:~#
```

3.3 Résultat dans GLPI — la CMDB à jour

On se connecte à l'interface GLPI et on vérifie que la machine est bien apparue dans l'inventaire, avec toutes ses caractéristiques : système d'exploitation (Debian GNU/Linux 13), version du noyau, fabricant (VMware), processeur, adresse MAC. Aucune saisie manuelle n'a été nécessaire.





IV. Mise en œuvre — Phase 2 : la supervision temps réel

4.1 Objectif

Avoir un tableau de bord en temps réel sur l'état de l'infrastructure, et déclencher des alertes automatisées quand un seuil critique est dépassé, avant que cela n'impacte la production.

4.2 Installation de l'agent Zabbix sur la machine surveillée

Étape 1 — Installation des paquets

Sur la machine cible (Debian 13), on met à jour les dépôts et on installe l'agent :

```
apt update apt install zabbix-agent -y
```

```
Réception de :2 http://deb.debian.org/debian bookworm/main amd64 zabbix-agent am
d64 1:6.0.14+dfsg-1+b1 [675 kB]
706 ko réceptionnés en 1s (1 222 ko/s)
Sélection du paquet libmodbus5:amd64 précédemment désélectionné.
(Lecture de la base de données... 121590 fichiers et répertoires déjà installés.
)
Préparation du dépaquetage de .../libmodbus5_3.1.6-2.1_amd64.deb ...
Dépaquetage de libmodbus5:amd64 (3.1.6-2.1) ...
Sélection du paquet zabbix-agent précédemment désélectionné.
Préparation du dépaquetage de .../zabbix-agent_1%3a6.0.14+dfsg-1+b1_amd64.deb ..
.
Dépaquetage de zabbix-agent (1:6.0.14+dfsg-1+b1) ...
Paramétrage de libmodbus5:amd64 (3.1.6-2.1) ...
Paramétrage de zabbix-agent (1:6.0.14+dfsg-1+b1) ...

Creating config file /etc/zabbix/zabbix_agentd.conf with new version
Created symlink /etc/systemd/system/multi-user.target.wants/zabbix-agent.service
 → /lib/systemd/system/zabbix-agent.service.
Traitement des actions différées (« triggers ») pour man-db (2.11.2-2) ...
Traitement des actions différées (« triggers ») pour libc-bin (2.36-9+deb12u13)
...
root@debian-srv:~#
```

```
root@debian-srv:~# apt install zabbix-agent -y
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
Lecture des informations d'état... Fait
zabbix-agent est déjà la version la plus récente (1:6.0.14+dfsg-1+b1).
0 mis à jour, 0 nouvellement installés, 0 à enlever et 28 non mis à jour.
root@debian-srv:~#
```

Étape 2 — Configuration du fichier zabbix_agentd.conf

On édite le fichier de configuration pour indiquer l'adresse du serveur Zabbix et le nom de la machine :

```
nano /etc/zabbix/zabbix_agentd.conf
```

```
root@debian-srv:~# nano /etc/zabbix/zabbix_agentd.conf
```

Les trois lignes à modifier ou ajouter :

```
Server=192.168.50.11 ServerActive=192.168.50.11 Hostname=debian13
```

Server définit l'adresse autorisée à interroger l'agent. ServerActive définit l'adresse vers laquelle l'agent pousse ses données en mode actif. Hostname est le nom que la machine utilisera pour s'identifier auprès de Zabbix.

```
Server=192.168.50.11
```

```
ServerActive=192.168.50.11
```

```
Hostname=debian13
```

On redémarre le service pour appliquer :

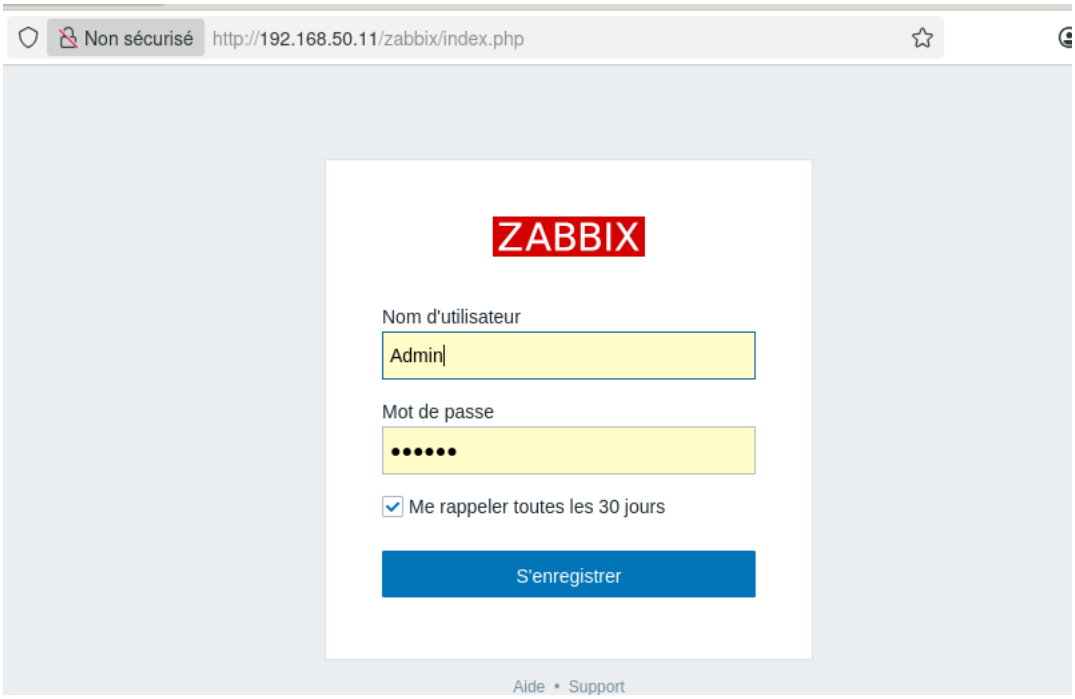
```
systemctl restart zabbix-agent
```

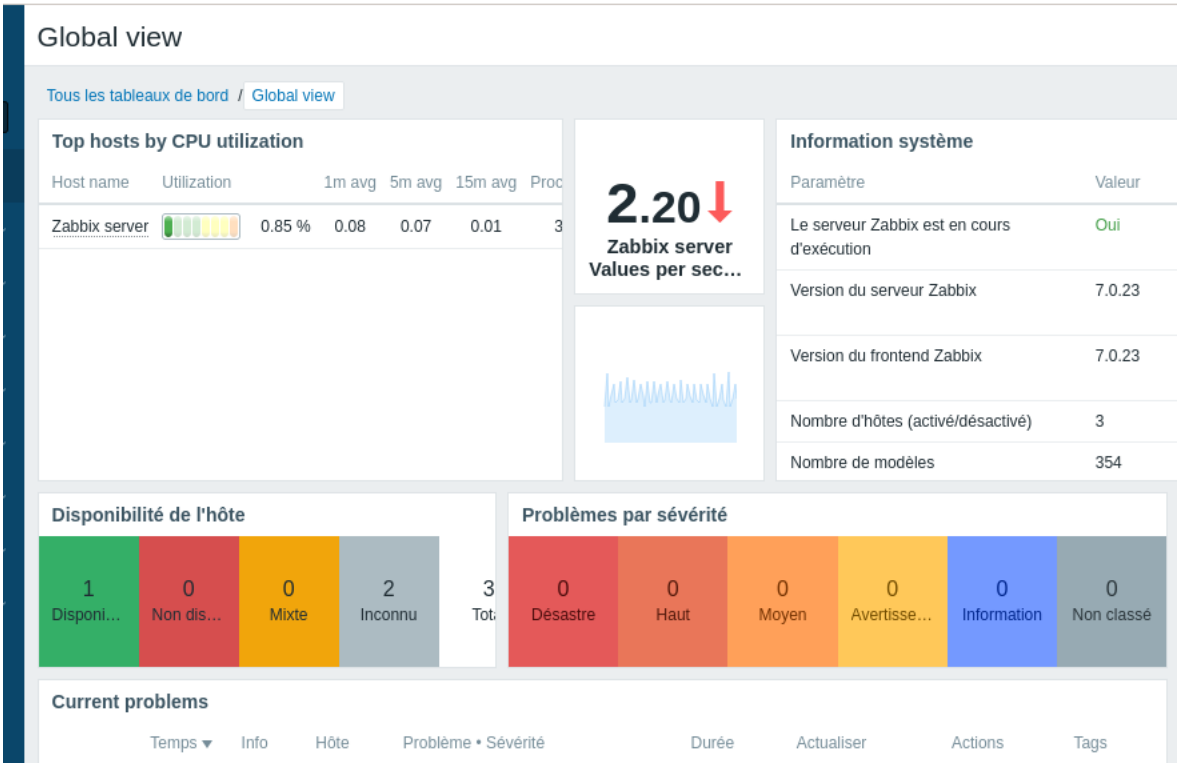
```
root@debian-srv:~# systemctl restart zabbix-agent
```

4.3 Ajout de l'hôte dans l'interface Zabbix

Étape 1 — Connexion à Zabbix

On ouvre un navigateur et on accède à l'interface Zabbix via http://IP_ZABBIX_SERVER/zabbix. On se connecte avec les identifiants admin.



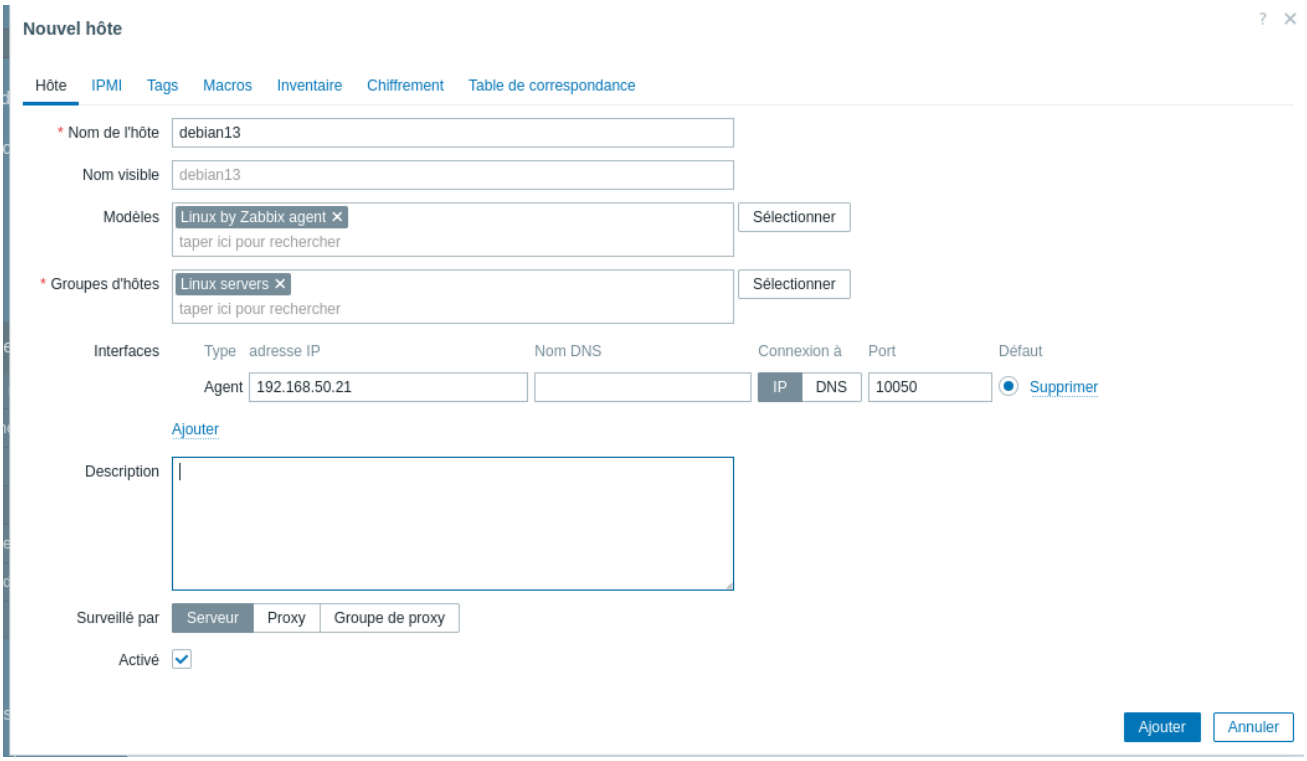


Étape 2 — Création de l'hôte

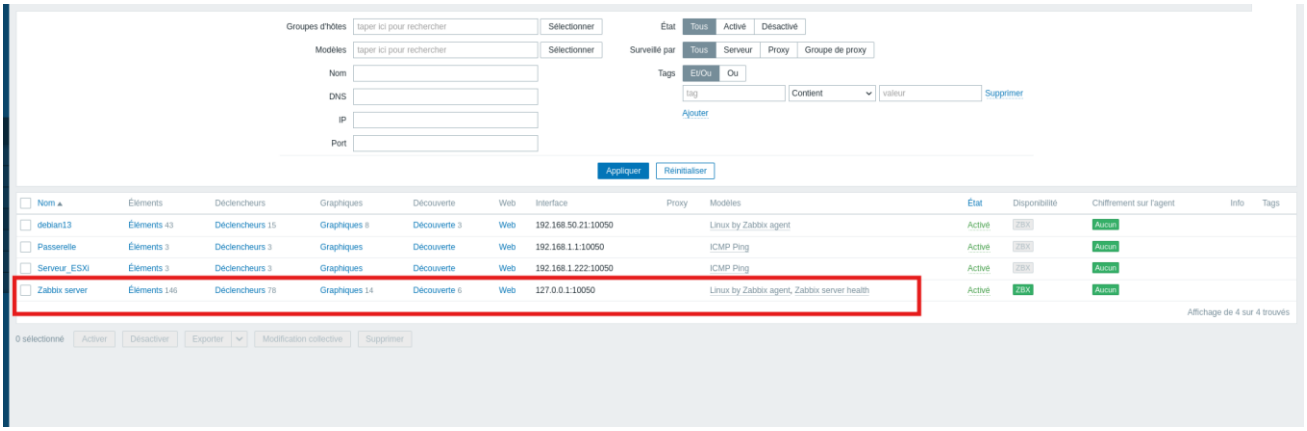
On va dans : Data collection > Hosts > Create Host.

On renseigne les informations de la machine à surveiller :

- Host name : debian13
- Templates : Linux by Zabbix agent (ce template contient déjà plus de 200 items préconfigurés pour un serveur Linux)
- Host groups : Linux servers
- Interfaces : Agent — 192.168.50.21 — port 10050



On enregistre. Après quelques secondes, l'hôte apparaît dans la liste avec une disponibilité verte, ce qui confirme que le serveur Zabbix communique bien avec l'agent.



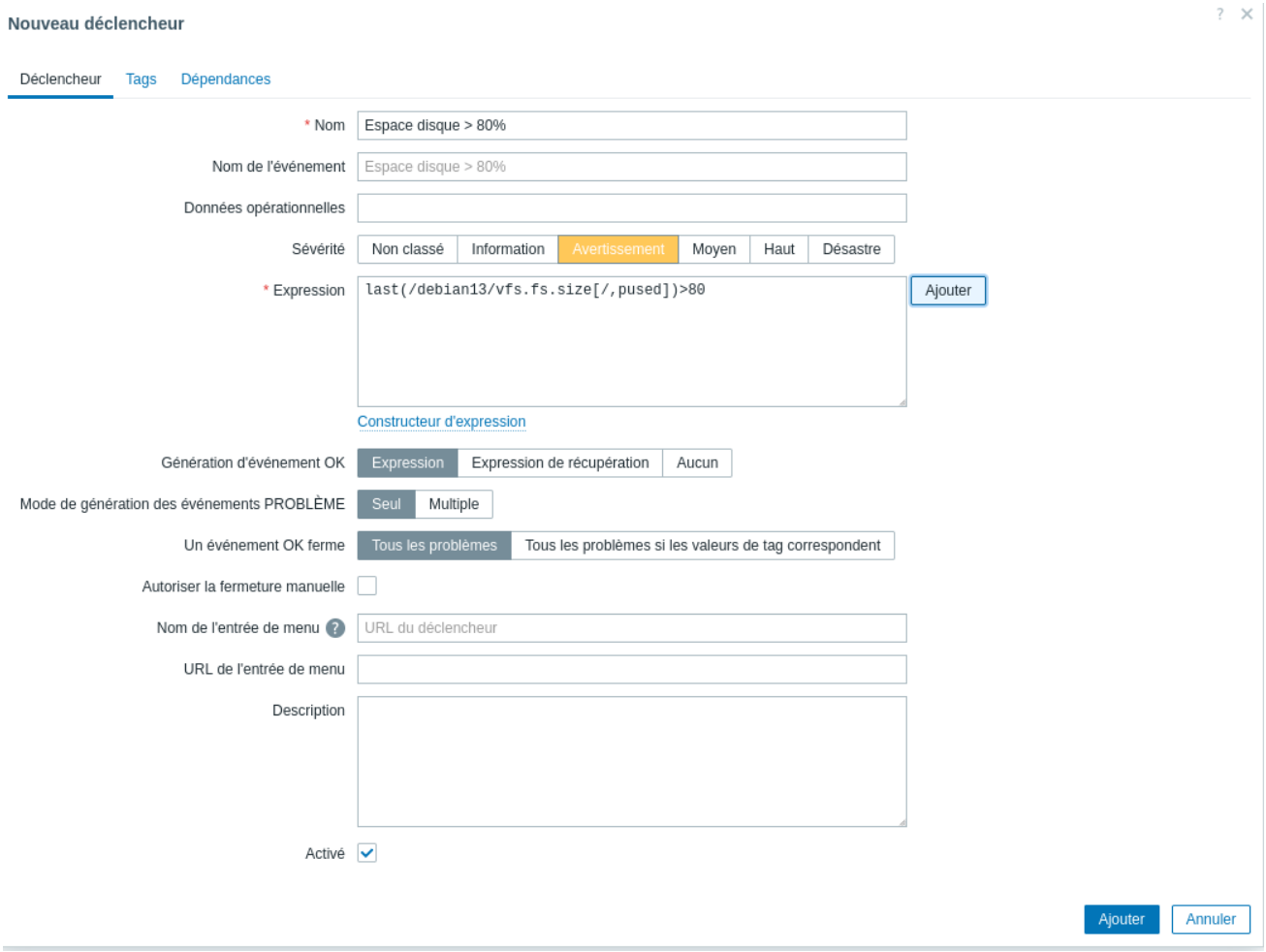
4.4 Définition des seuils d'alerte (Triggers)

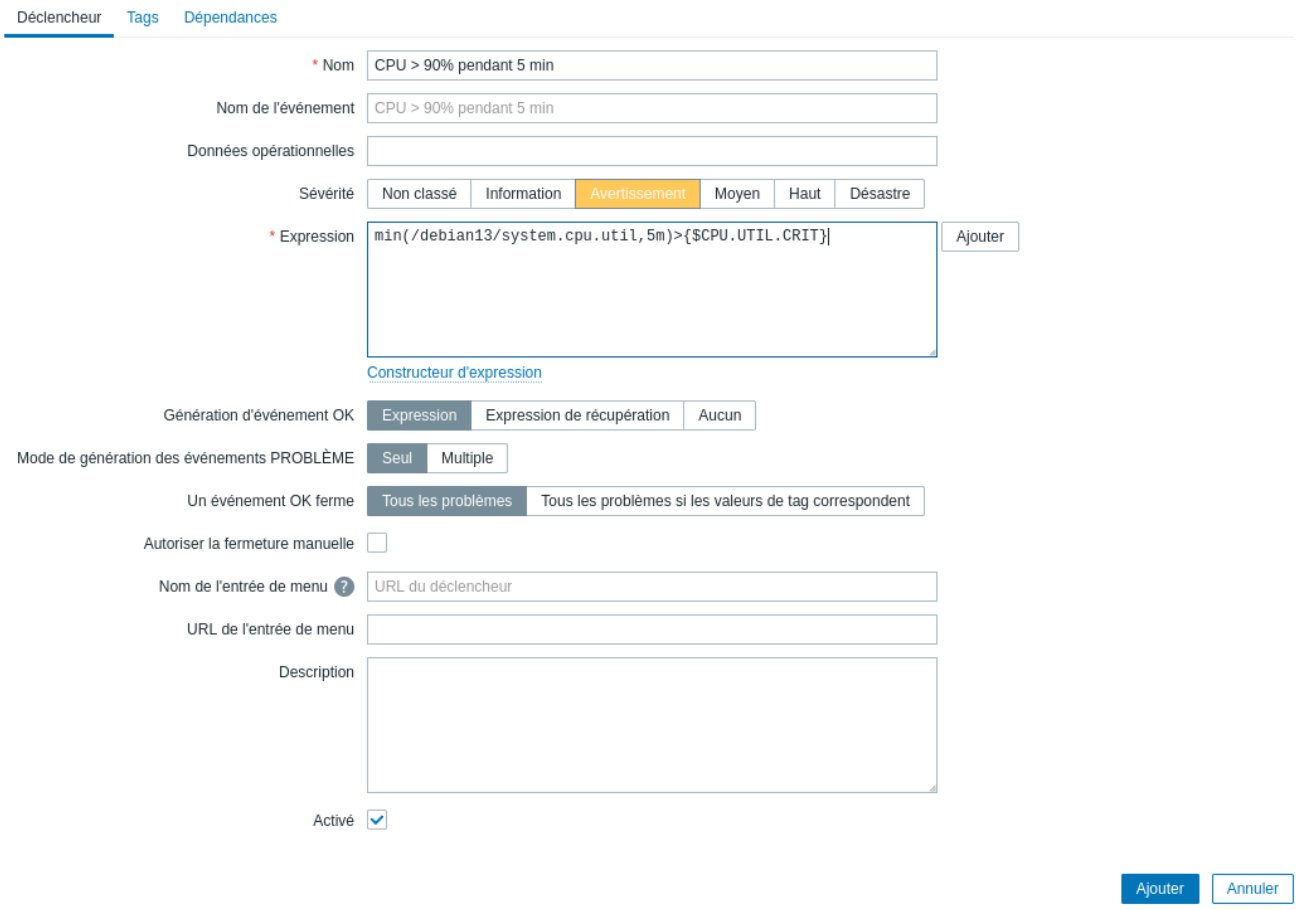
Un trigger est une condition logique qui déclenche une alerte quand elle est vraie. J'ai défini trois niveaux d'alerte en fonction de la gravité :

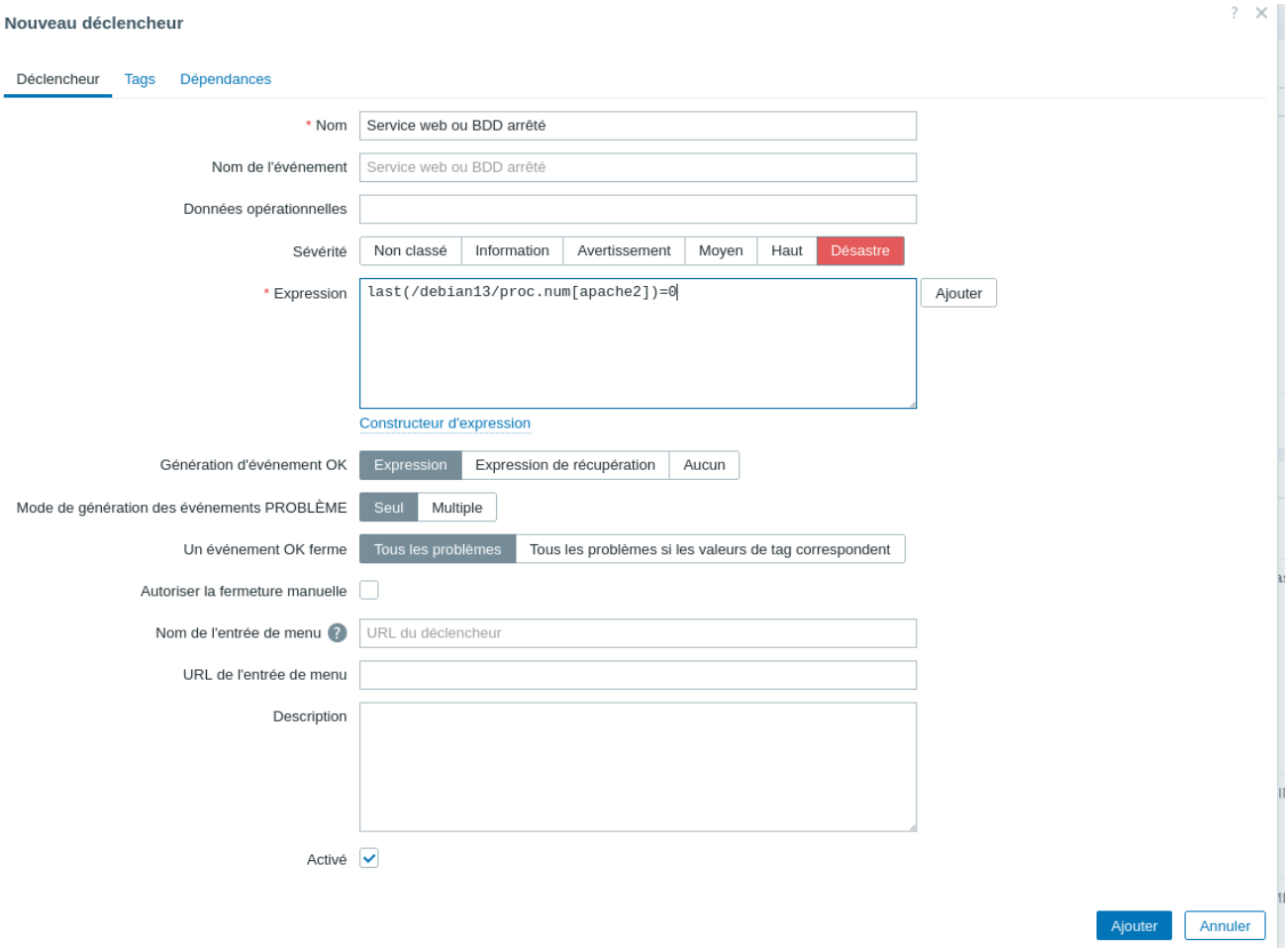
Niveau	Condition	Expression Zabbix	Action
Warning	Espace disque > 80%	last(/debian13/vfs.fs.size[/,pused])>80	Email d'alerte
Average	CPU > 90% pendant 5 min	min(/debian13/system.cpu.util,5m)>{$CPU.UTIL.CRIT}	Ticket GLPI créé automatiquement
Disaster	Service web ou BDD arrêté	last(/debian13/proc.num[apache2])=0	Alerte immédiate + ticket prioritaire

La durée de 5 minutes sur le trigger CPU évite les faux positifs liés aux pics de charge passagers. Sans cette durée, un simple pic de quelques secondes déclencherait une alerte.

Pour créer un trigger, on va dans : Data collection > Hosts > debian13 > Triggers > Create trigger.




Déclencheur
Tags
Dépendances
* Nom
CPU > 90% pendant 5 min
Nom de l'événement
CPU > 90% pendant 5 min
Données opérationnelles
Sévérité
Non classé
Information
Avertissement
Moyen
Haut
Désastre
* Expression
min(/debian13/system.cpu.util,5m)>{$CPU.UTIL.CRIT}
Ajouter
Constructeur d'expression
Génération d'événement OK
Expression
Expression de récupération
Aucun
Mode de génération des événements PROBLÈME
Seul
Multiple
Un événement OK ferme
Tous les problèmes
Tous les problèmes si les valeurs de tag correspondent
Autoriser la fermeture manuelle
Nom de l'entrée de menu
URL du déclencheur
URL de l'entrée de menu
Description
Activé
Ajouter
Annuler


Nouveau déclencheur
Déclencheur
Tags
Dépendances
* Nom
Service web ou BDD arrêté
Nom de l'événement
Service web ou BDD arrêté
Données opérationnelles
Sévérité
Non classé
Information
Avertissement
Moyen
Haut
Désastre
* Expression
last(/debian13/proc.num[apache2])=0
Ajouter
Constructeur d'expression
Génération d'événement OK
Expression
Expression de récupération
Aucun
Mode de génération des événements PROBLÈME
Seul
Multiple
Un événement OK ferme
Tous les problèmes
Tous les problèmes si les valeurs de tag correspondent
Autoriser la fermeture manuelle
Nom de l'entrée de menu
URL du déclencheur
URL de l'entrée de menu
Description
Activé
Ajouter
Annuler

V. Mise en œuvre — Phase 3 : l'interconnexion Zabbix-GLPI

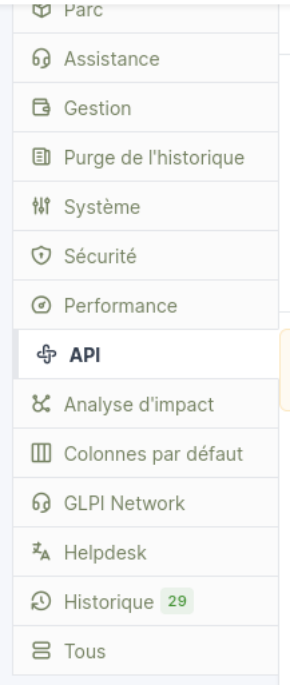
5.1 Objectif

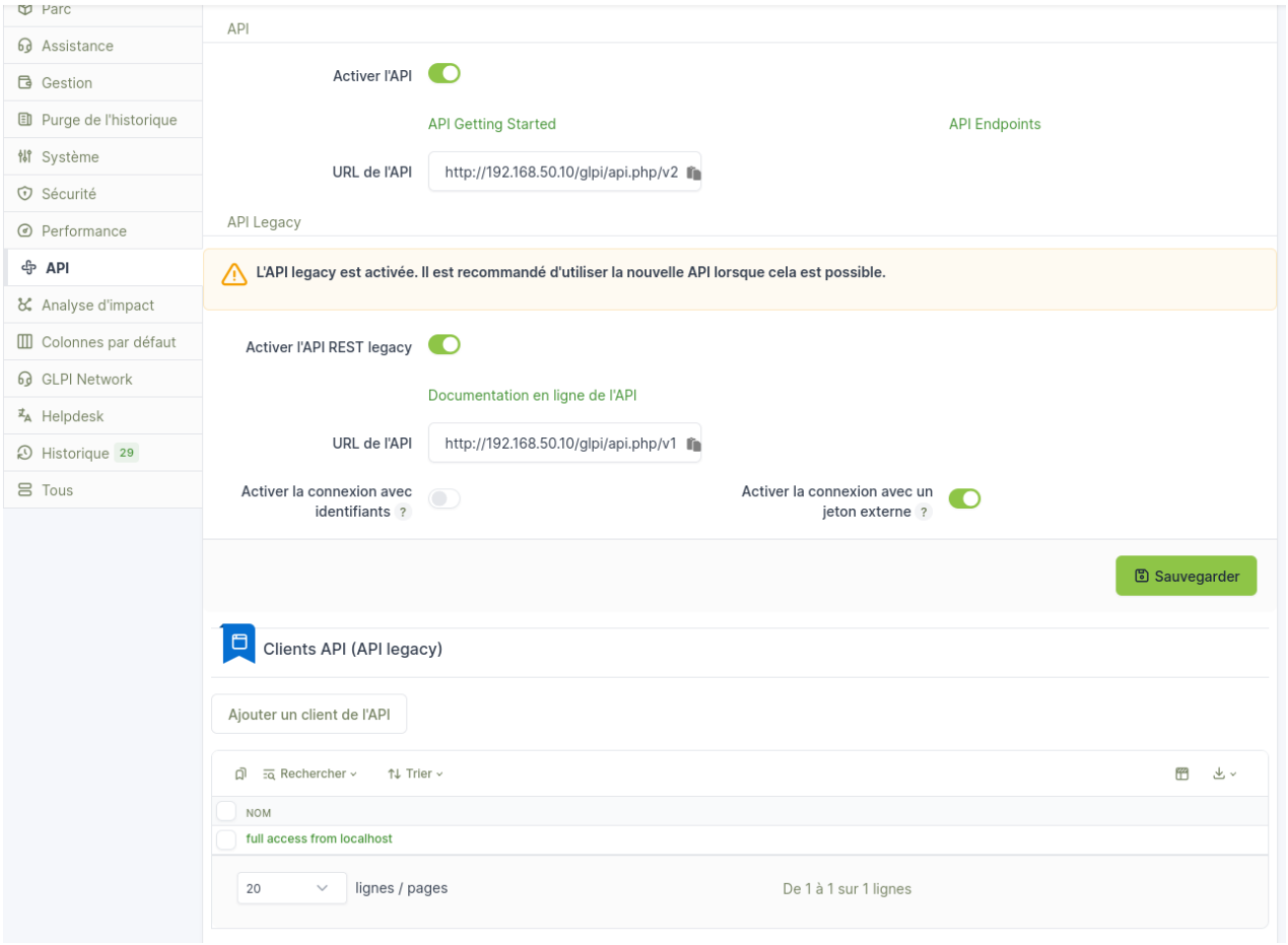
C'est la partie la plus importante du projet : relier les deux outils pour que Zabbix crée automatiquement un ticket dans GLPI quand il détecte un incident. Plus besoin d'intervention humaine pour ouvrir un ticket, tout se fait en quelques secondes.

5.2 Activation de l'API REST dans GLPI

Étape 1 — Activer l'API

Dans GLPI, on va dans : Configuration > Générale > API. On active l'API REST et on copie l'URL du point d'entrée qui sera utilisée par Zabbix.

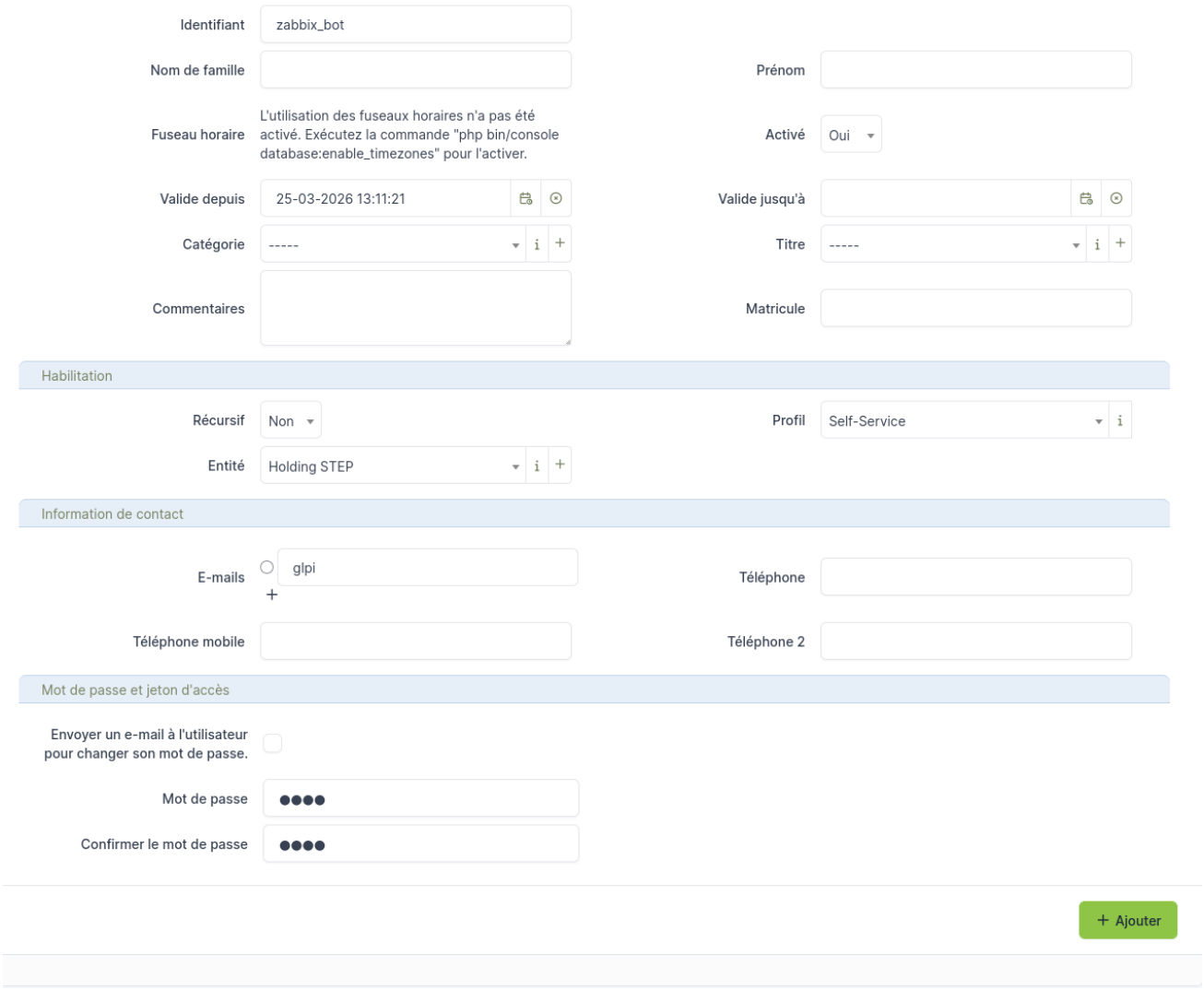




Étape 2 — Création de l'utilisateur technique

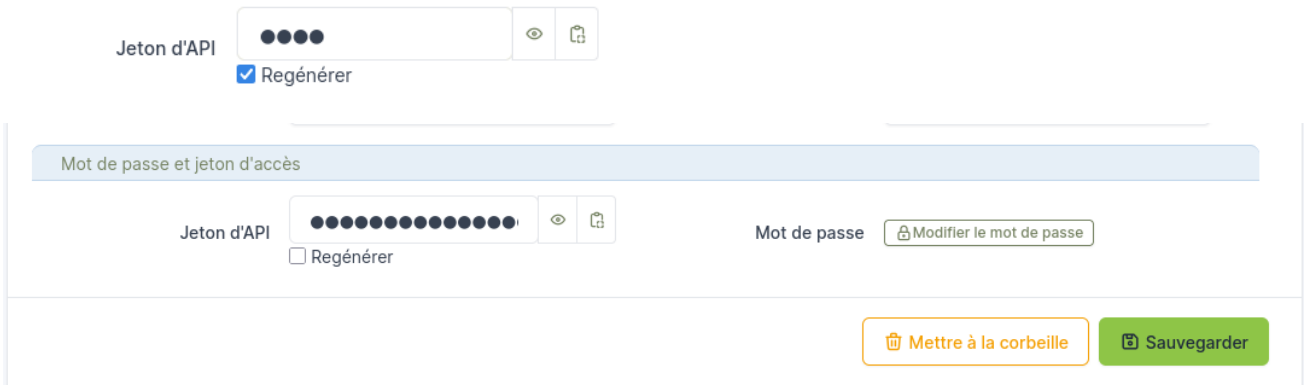
On crée un utilisateur dédié pour Zabbix dans GLPI. Cet utilisateur aura uniquement les droits nécessaires pour créer des tickets, pas plus. C'est le principe du moindre privilège : en cas de compromission du token, l'impact reste limité.

Identifiant choisi : zabbix_bot. Entité : Holding STEP.



Étape 3 — Génération du token API

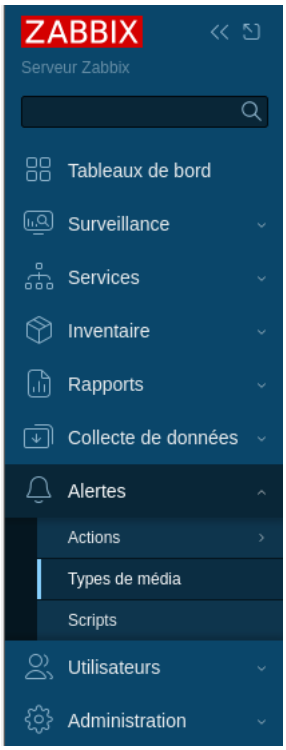
Dans la fiche de l'utilisateur zabbix_bot, on descend jusqu'à la section « Mot de passe et jeton d'accès ». On clique sur « Régénérer » puis on sauvegarde. On revient ensuite sur la fiche pour copier le token généré.



5.3 Configuration du webhook dans Zabbix

Étape 4 — Paramétrage du Media type GLPI

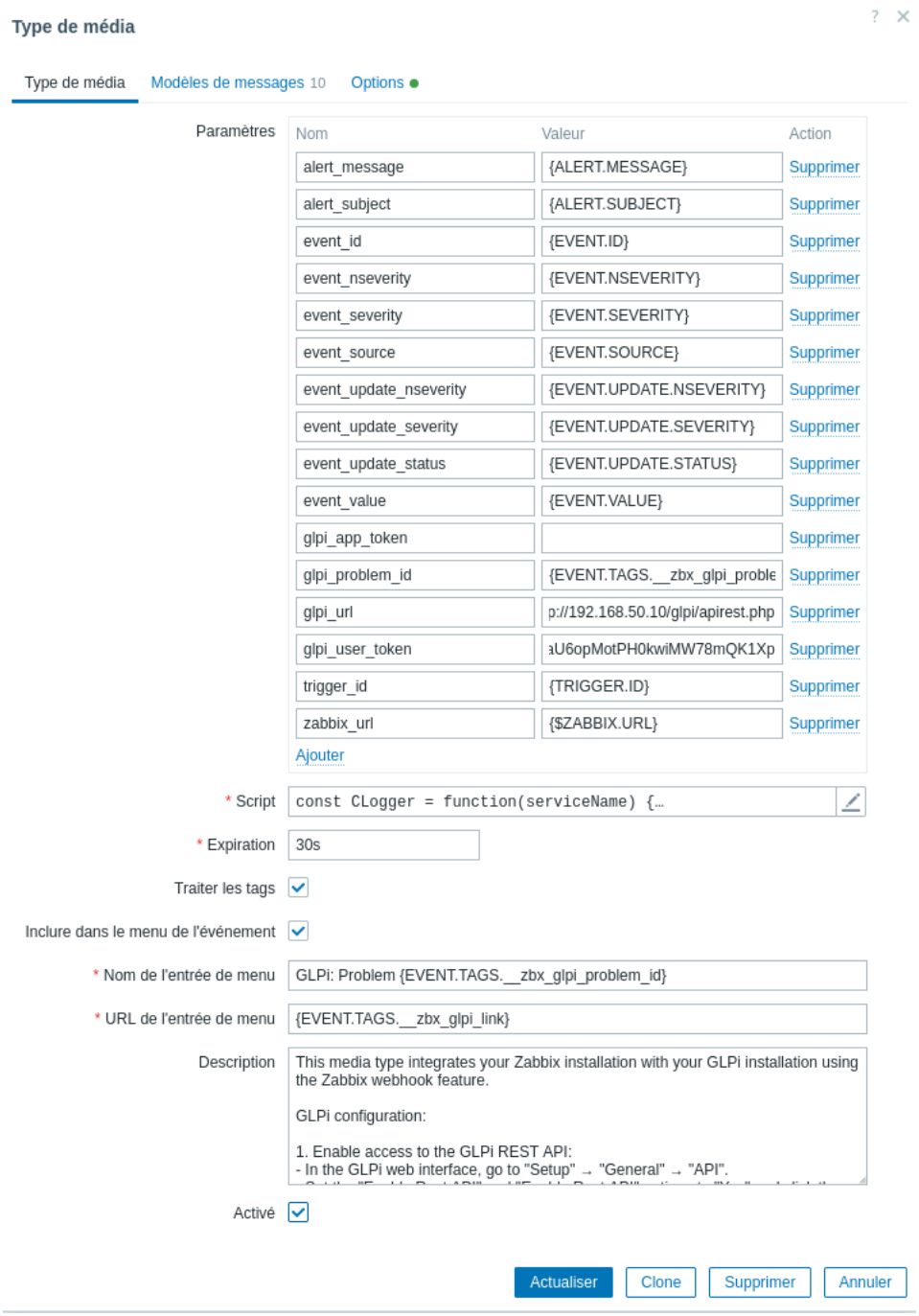
Dans Zabbix, le media type GLPI est intégré nativement. On va dans : Alertes > Media types.



On ouvre le media type GLPI et on renseigne les paramètres clés dans l'onglet Parameters. Les paramètres importants sont les suivants :

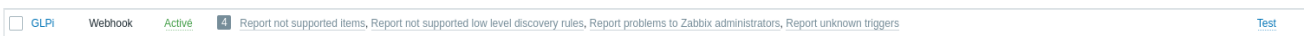
- alert_message : {ALERT.MESSAGE} — contenu de l'alerte transmis dans le ticket
- alert_subject : {ALERT.SUBJECT} — sujet du ticket créé dans GLPI
- event_id : {EVENT.ID} — identifiant unique de l'événement Zabbix
- event_severity : {EVENT.SEVERITY} — niveau de gravité (Warning, Average, Disaster)
- glpi_url : http://192.168.50.10/glpi/apirest.php — point d'entrée de l'API
- glpi_user_token : [jeton copié depuis la fiche utilisateur GLPI] — clé d'authentification
- trigger_id : {TRIGGER.ID} — permet de lier le ticket au trigger d'origine

Les autres paramètres (event_source, event_value, zabbix_url...) sont renseignés par des macros Zabbix et alimentent automatiquement les détails du ticket.



Dans l'onglet Script, on peut voir le code JavaScript natif qui effectue l'appel HTTP POST vers l'API GLPI. On n'a pas à le modifier.

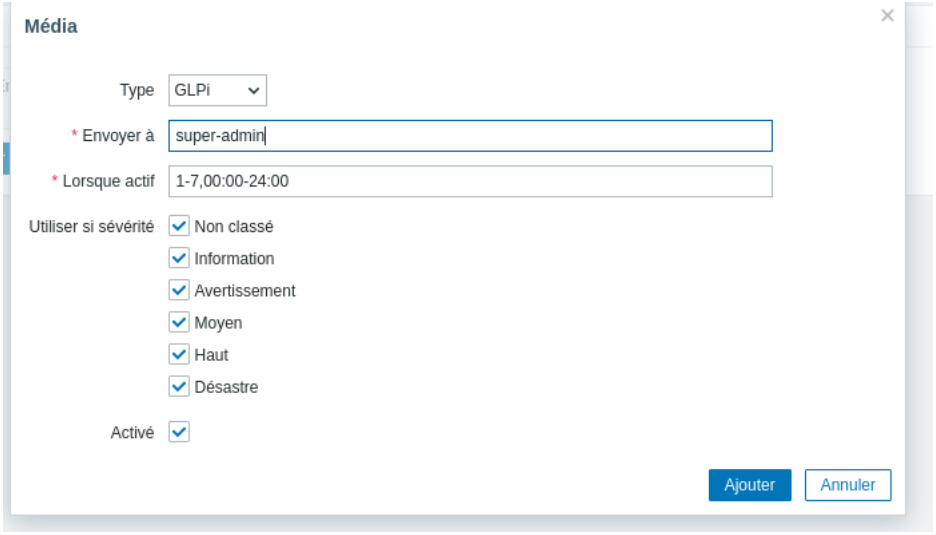
On enregistre. Le media type GLPI passe en statut Enabled.



Étape 5 — Association du canal de notification à l'utilisateur admin

Pour que Zabbix sache où envoyer les alertes, on associe le media type GLPI à l'utilisateur admin. On va dans : Users > Users > Admin > onglet Media > Add.

- Type : GLPi
- Send to : super-admin (le profil d'envoi)
- When active : 1-7,00:00-24:00 (toute la semaine, 24h/24)
- Use if severity : toutes les sévérités cochées (Not classified, Information, Warning, Average, High, Disaster)
- Enabled : coché

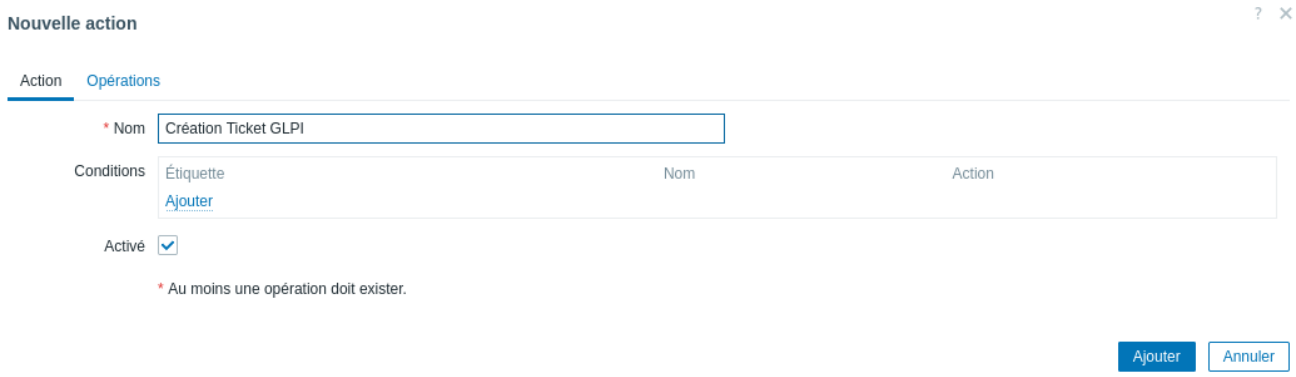


On clique sur Add. Le media apparait dans l'onglet Media de l'utilisateur admin.

Étape 6 — Création de l'action automatisée (Trigger action)

C'est l'étape finale qui relie tout. L'action définit la condition de déclenchement et l'opération à exécuter. On va dans : Alertes > Actions > Trigger actions > Create action.

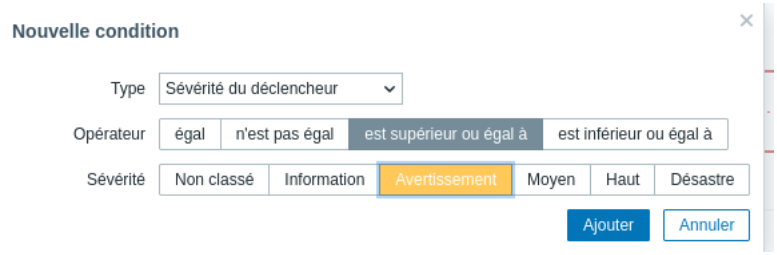
On nomme l'action « Création Ticket GLPI ».



Dans l'onglet Conditions, on ajoute la règle de déclenchement :

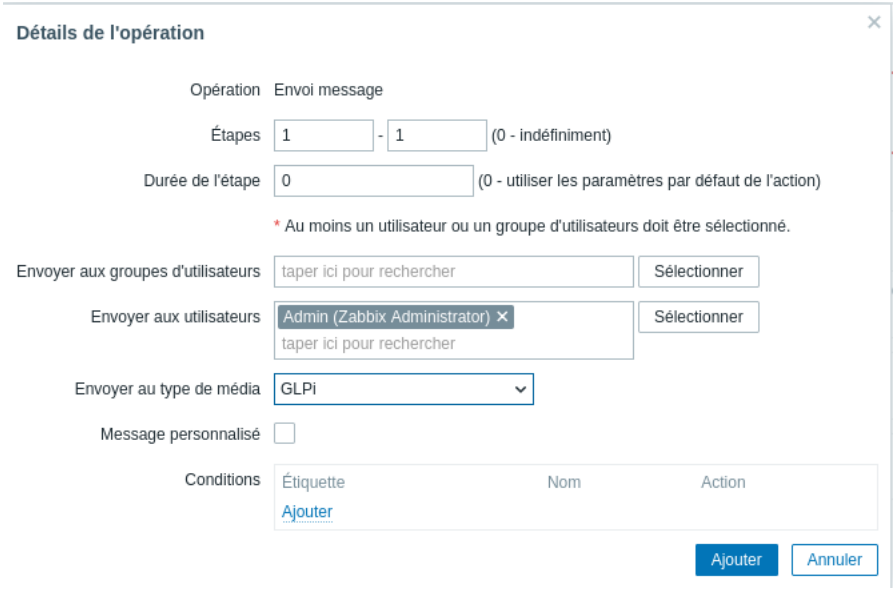
- Type : Trigger severity
- Operator : is greater than or equals
- Severity : Average (niveau sélectionné — orange)

Cela signifie que l'action se déclenchera pour toute alerte de niveau Average, High ou Disaster. Les alertes Warning passent par les triggers mais ne créent pas de ticket.

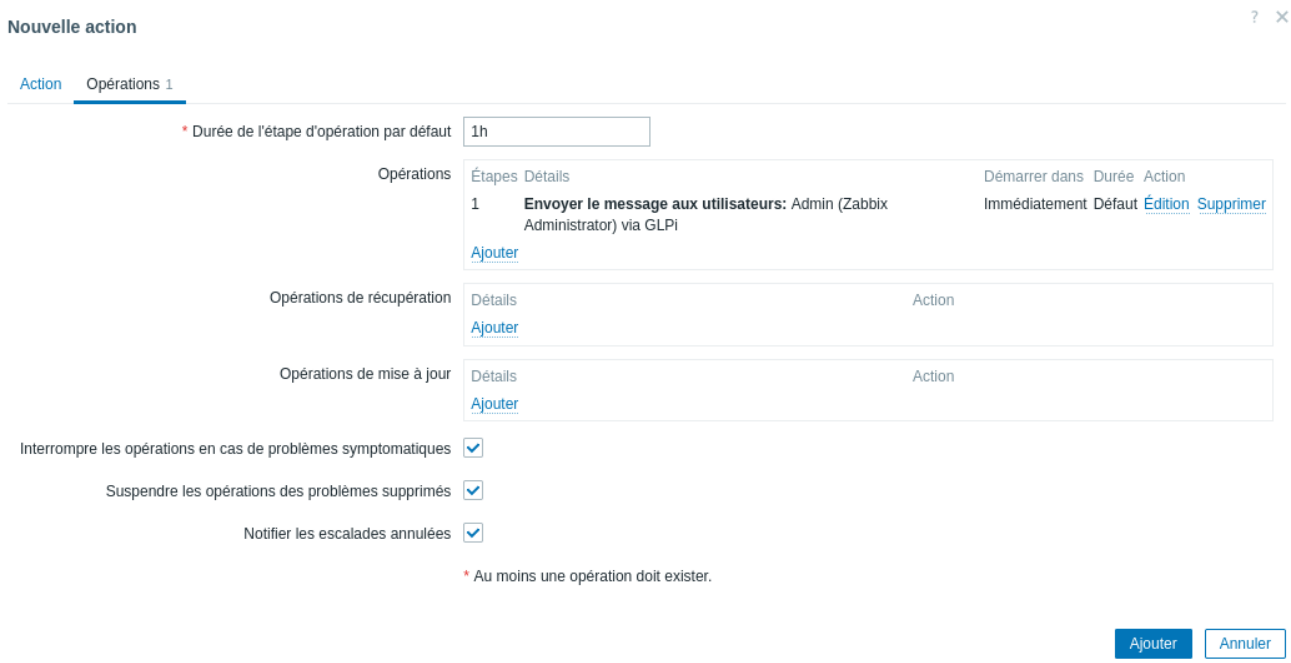


Dans l'onglet Operations, on définit ce que Zabbix doit faire quand la condition est vraie :

- Operation : Send message
- Steps : 1 à 1 (une seule notification, pas de rappels)
- Send to users : Admin (Zabbix Administrator)
- Send to media type : GLPi



On sauvegarde. L'action apparaît dans la liste avec la condition et l'opération.



5.4 Scénario de test de bout en bout

Pour valider que toute la chaîne fonctionne — de la détection dans Zabbix jusqu'à la création du ticket dans GLPI — on simule un incident réel.

Préparation : choix du scénario de test

Initialement, l'idée était d'arrêter le service Apache2 pour simuler une panne de service web. Mais on a identifié un problème : GLPI est hébergé sur la même machine (debian13) qu'Apache. Si on coupe Apache, GLPI s'éteint aussi et ne peut plus recevoir le ticket envoyé par Zabbix. Le test n'aurait pas fonctionné.

On opte donc pour l'alerte CPU : on provoque une surcharge CPU qui déclenche le trigger Average. GLPI reste actif et peut recevoir le ticket normalement.

Exécution : commande stress

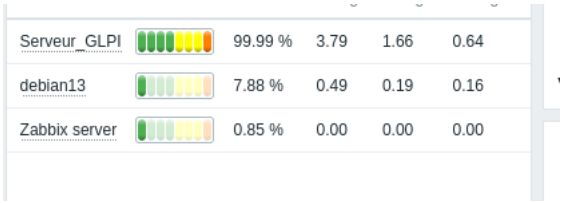
On installe l'outil stress sur la machine cible puis on lance une surcharge sur les 4 cœurs CPU pendant 10 minutes (600 secondes). Le trigger est configuré pour se déclencher après 5 minutes de CPU > 90%.

```
apt install stress -y stress --cpu 4 --timeout 600
```

```
^C
glpi@debian13:~$ stress --cpu 4 --timeout 600
stress: info: [3613] dispatching hogs: 4 cpu, 0 io, 0 vm, 0 hdd
```

Déroulement observé — les 4 étapes

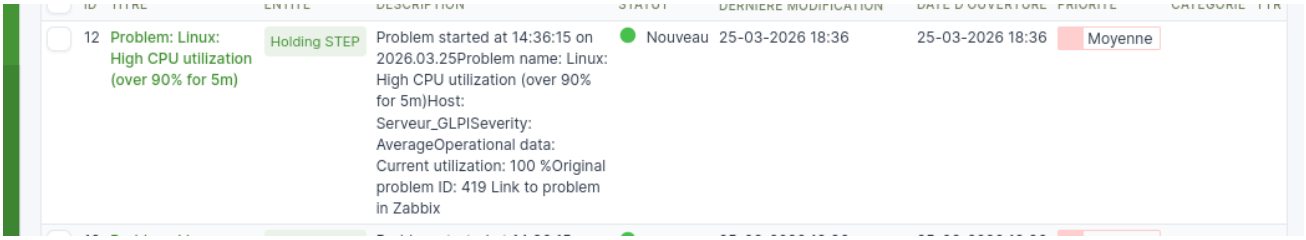
1. Détection : Zabbix détecte la montée de charge CPU sur la machine debian13. L'utilisation dépasse 90% pendant la durée configurée (5 min). Le dashboard affiche debian13 à près de 99% d'utilisation CPU.



2. Alerte : Le trigger Average passe au statut Problem. Il apparaît dans la section Current problems du dashboard avec sa sévérité et sa durée.

3. Action : Dès que le trigger est actif, Zabbix exécute l'action configurée. Il envoie instantanément une requête HTTP POST authentifiée vers l'API REST de GLPI, avec les détails de l'incident (nom du trigger, sévérité, timestamp, nom de l'hôte).

4. Résultat : Un ticket d'incident est créé automatiquement dans GLPI avec le statut « Nouveau ». Il contient les détails techniques de la panne. Aucune intervention humaine n'a été nécessaire.



> *Le délai entre le déclenchement du trigger Zabbix et la création du ticket dans GLPI est inférieur à 30 secondes. C'est l'objectif central du projet : éliminer le délai humain dans le traitement des incidents.*

VI. Validation et recette

Le tableau ci-dessous récapitule les tests effectués pour valider le passage d'une gestion manuelle à une gestion automatisée.

ID	Scénario testé	Résultat attendu	Résultat obtenu	Statut
T01	Remontée inventaire GLPI-Agent	Fiche auto-créée dans la CMDB	Fiche debian13 présente avec OS, CPU, RAM	OK
T02	Disponibilité ICMP (ping)	Zabbix détecte si machine en ligne	Détection confirmée	OK
T03	Communication API GLPI	Webhook s'authentifie via token	Réponse HTTP 200 reçue	OK
T04	Alerte seuil CPU (stress test)	Trigger Average déclenché après 5 min	Alerte visible dans Zabbix	OK
T05	Création automatique de ticket	Ticket « Nouveau » dans GLPI sans action humaine	Ticket créé automatiquement	OK

VII. Maintien en condition opérationnelle (MCO)

La solution ne s'arrête pas à la mise en place. Pour qu'elle reste fiable dans le temps, j'ai défini plusieurs procédures :

- Mises à jour de sécurité : planification mensuelle des mises à jour des serveurs Debian. Pour GLPI et Zabbix, on suit les versions LTS stables avant de mettre à jour.
- Sauvegardes automatisées : dump quotidien de la base MySQL de GLPI via mysqldump, vers un stockage externe. Un script bash planifié en cron gère l'automatisation.
- Gestion des faux positifs : les triggers ont une durée de persistance (5 min pour le CPU) pour éviter des alertes sur des pics passagers. On ajuste les seuils si nécessaire.
- Rotation des tokens API : renouvellement trimestriel du token de l'utilisateur zabbix_bot pour limiter le risque en cas de fuite.

VIII. Bilan et analyse critique

8.1 Ce que la solution apporte

Critère	Avant	Après	Verdict
Précision de l'inventaire	Faible — Excel obsolète	Fiable — remontée auto quotidienne	Amélioré
Délai de détection	Plusieurs heures (signalement user)	< 60 secondes (temps réel)	Très rapide
Traitement incident	Manuel — ticket à la main	Automatisé via API GLPI	Efficace
Vision globale	Aucune	Dashboard unifié Zabbix + GLPI	Clair

8.2 Limites identifiées

Ce projet a quelques limites à corriger avant une mise en production réelle :

- Co-localisation GLPI et Apache : dans la maquette, GLPI et le service à surveiller sont sur la même machine. En production, chaque service critique doit être sur un serveur séparé pour éviter ce point de défaillance unique.
- API en HTTP : dans la maquette, les appels entre Zabbix et GLPI passent en HTTP. En production, HTTPS avec un certificat valide est obligatoire pour sécuriser le transit du token.
- Déploiement manuel de l'agent : il a été installé à la main sur une machine. Pour couvrir tout un parc, un déploiement automatisé via Ansible ou GPO serait nécessaire.

8.3 Note sur le développement durable

Ce projet s'inscrit dans une démarche écoresponsable à plusieurs niveaux. La virtualisation sous VMware ESXi permet de faire tourner plusieurs serveurs sur une seule machine physique, ce qui réduit la consommation énergétique et limite le nombre d'équipements nécessaires. Le choix de solutions 100% open source (GLPI, Zabbix, Debian) élimine les dépendances aux licences propriétaires et évite l'obsolescence programmée liée aux cycles commerciaux des éditeurs. Enfin, l'automatisation du traitement des incidents réduit les déplacements inutiles des techniciens.

8.4 Conclusion

Ce projet m'a permis de mettre en place une vraie solution de supervision proactive, en interconnectant deux outils open source via une API REST. Le résultat concret : le délai de détection d'un incident est passé de plusieurs heures à moins de 5 minutes, et les tickets sont créés sans intervention humaine. La solution est fonctionnelle, documentée et maintenue dans le temps.

IX. Annexes

Annexe 1 : Workflow d'incident automatisé

Étape	Acteur	Action	Résultat
1	Sonde Zabbix	Détection du dépassement de seuil	Trigger activé
2	Moteur Zabbix	Vérification de la persistance (5 min)	Incident confirmé
3	Webhook Zabbix	Appel API REST GLPI avec token	Requête HTTP POST envoyée
4	API GLPI	Création du ticket catégorisé et assigné	Ticket visible dans le helpdesk
5	Technicien	Intervention guidée par les logs du ticket	Résolution et clôture

Annexe 2 : Plan de tests détaillé

ID	Description	Résultat attendu	Résultat obtenu	Conclusion
T01	Remontée inventaire GLPI-Agent	Fiche auto dans CMDB	Présente avec OS, CPU, RAM	Inventaire automatisé OK
T02	Disponibilité ICMP	Machine en ligne détectée	Confirmé	Monitoring de base OK
T03	Auth API — token GLPI	HTTP 200 — token accepté	Connexion validée	API opérationnelle
T04	Stress CPU — trigger Average	Alerte après 5 min	Trigger visible Zabbix	Détection OK
T05	Ticket auto dans GLPI	Ticket « Nouveau » sans action	Ticket créé automatiquement	Workflow complet OK