UNIVERSITÉ جامعة MUNDIAPOLIS
HONORIS UNITED UNIVERSITIES

Rapport projet final à rendre :

Mise en œuvre d'une infrastructure cloud de supervision centralisée sous AWS : Déploiement de Zabbix conteneurisé pour le monitoring d'un parc hybride (Linux & Windows)

Module : Ingénierie Des Infrastructures Cloud
Année universitaire 2025/2026

Ecole d'Ingénieurs
Cycle Ingénieur : « Génie Informatique »
2A CI INF GA

Réalisé par :

ADIB Taha Mohamed

Encadré par :

Prof Azeddine KHIAT

I. Introduction Générale :

I.I. Contexte et Enjeux de la Supervision Informatique :

Dans un écosystème technologique en constante évolution, la disponibilité et la performance des infrastructures informatiques sont devenues des piliers critiques pour toute organisation. La complexité croissante des réseaux, mêlant serveurs locaux, environnements virtuels et instances Cloud, impose la mise en œuvre de solutions de monitoring proactives. La supervision ne se limite plus à vérifier si une machine est "allumée" ou "éteinte" ; elle consiste désormais à anticiper les pannes, à optimiser l'utilisation des ressources et à garantir la continuité de service (SLA).
Le présent projet s'inscrit dans cette démarche d'excellence opérationnelle. Il vise à déployer une plateforme de supervision centralisée capable de monitorer un parc hétérogène composé de serveurs Linux et de stations de travail Windows, le tout hébergé sur une infrastructure Cloud de pointe.

I.II. Présentation de la Solution : Zabbix 6.0 LTS :

Pour répondre à ces besoins, le choix s'est porté sur Zabbix, une solution de surveillance "Enterprise-class" reconnue pour sa flexibilité et sa robustesse. Contrairement à d'autres outils, Zabbix offre une gestion native des agents, une capacité d'auto-découverte et une interface web intuitive permettant une visualisation granulaire des données.
L'architecture de Zabbix repose sur plusieurs composants clés que nous avons implémentés :

- Zabbix Server : Le cœur du système qui analyse les données et génère les alertes.
- Zabbix Web Interface : L'interface d'administration PHP/Apache accessible via navigateur.
- Zabbix Database : Le stockage persistant des configurations et des données historiques, ici sous MySQL.
- Zabbix Agents : Les services installés sur les machines cibles (Linux et Windows) pour collecter les métriques locales.

I.III. Problématique et Objectifs du Projet :

Le défi majeur de ce projet réside dans l'instabilité inhérente aux environnements Cloud dynamiques. Lors de nos premières phases de test, nous avons constaté que le redémarrage des instances AWS entraînait un changement des adresses IP publiques, rompant ainsi la liaison entre le serveur et ses agents.

Les objectifs principaux de ce travail sont donc :

L'architecture Haute Disponibilité : Déployer le serveur Zabbix via **Docker-Compose** pour garantir une isolation des services et une portabilité maximale.
La Fixation du Réseau : Mettre en œuvre des **Adresses IP Élastiques (EIP)** sur AWS pour assurer la pérennité des communications entre le serveur et les clients.
La Supervision Hybride : Configurer et valider la remontée de données en temps réel depuis une instance Ubuntu Linux et une instance Windows Server.
La Sécurisation des Flux : Paramétrer les groupes de sécurité (Security Groups) AWS pour n'autoriser que les flux strictement nécessaires (ports 80, 10050, 10051).

I.IV. Méthodologie de Réalisation :

Le rapport détaille chaque étape du cycle de vie du projet :

Phase 1 : Préparation de l'infrastructure sur AWS (EC2 & Elastic IP).
Phase 2 : Déploiement du stack Docker Zabbix.
Phase 3 : Installation et configuration des agents sur les hôtes distants.
Phase 4 : Validation, administration web et résolution des incidents (Troubleshooting).

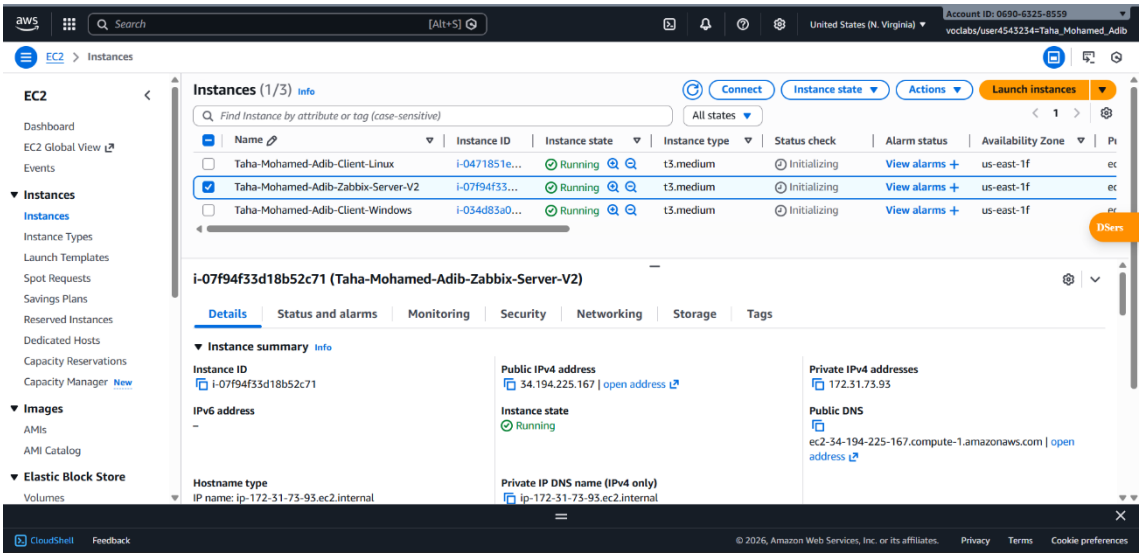
II. Architecture et Infrastructure Cloud (AWS) :

II.I. Choix de la plateforme Cloud : Amazon Web Services (AWS) :

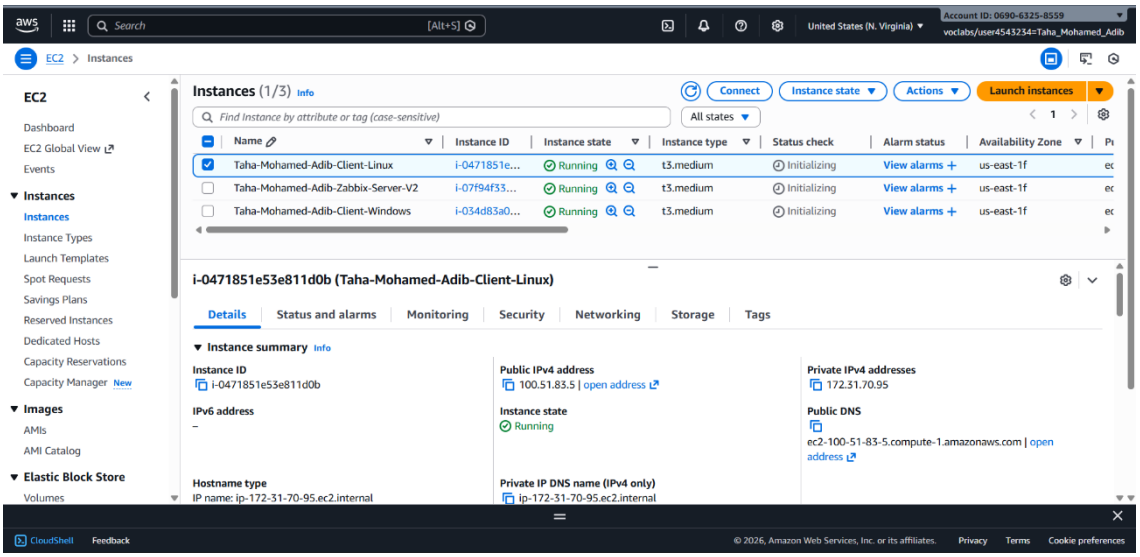
Dans cette section, expliquez pourquoi AWS a été choisi (leader du marché, fiabilité, scalabilité). Détaillez les ressources utilisées :

Les Instances EC2 :

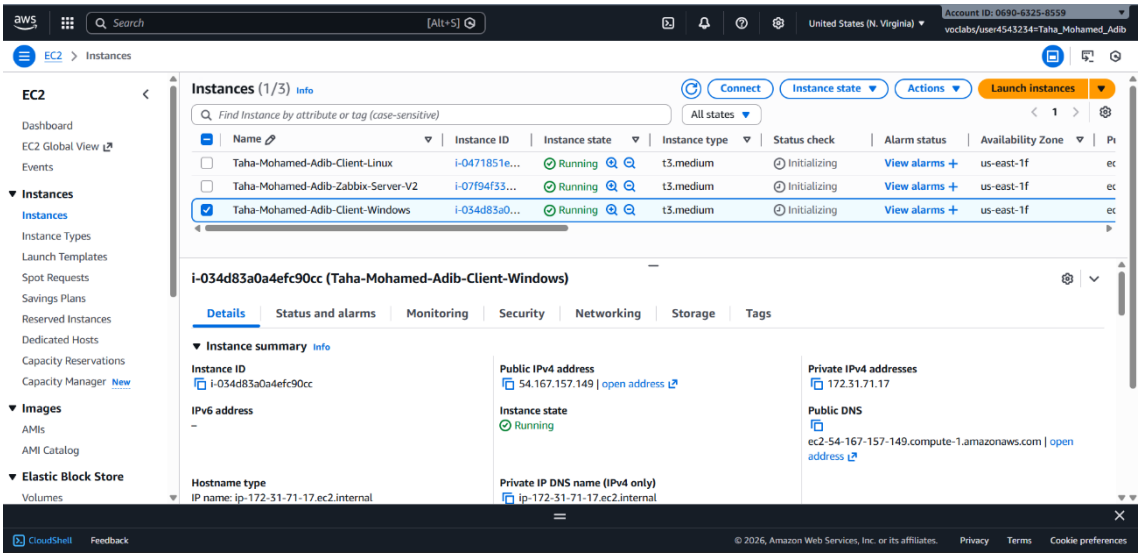
Serveur Zabbix : Instance sous Ubuntu 24.04/22.04 avec assez de RAM pour Docker.



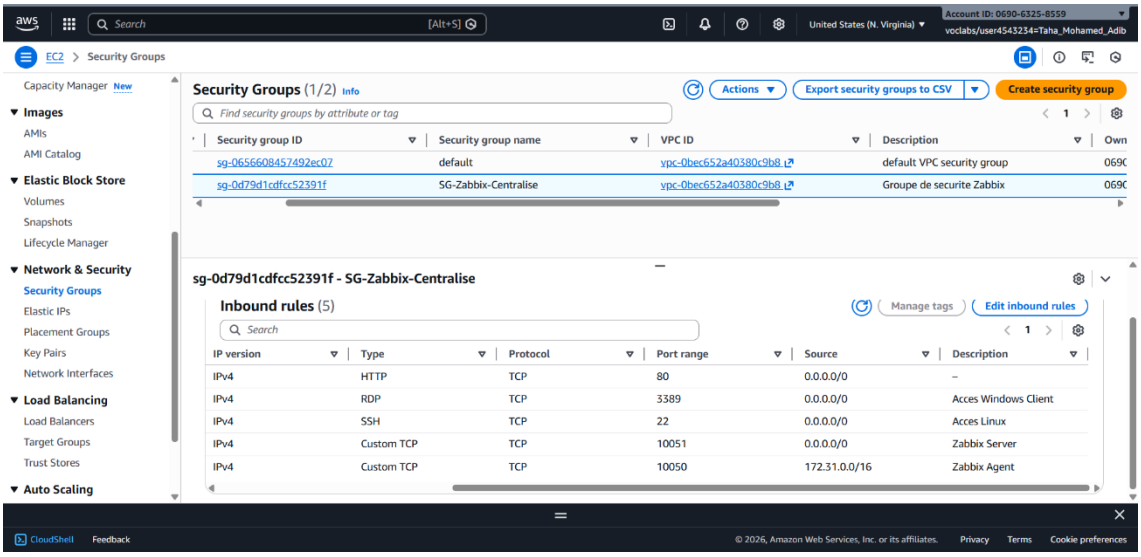
Client Linux : Instance Ubuntu légère.



Client Windows : Instance Windows Server avec interface graphique.

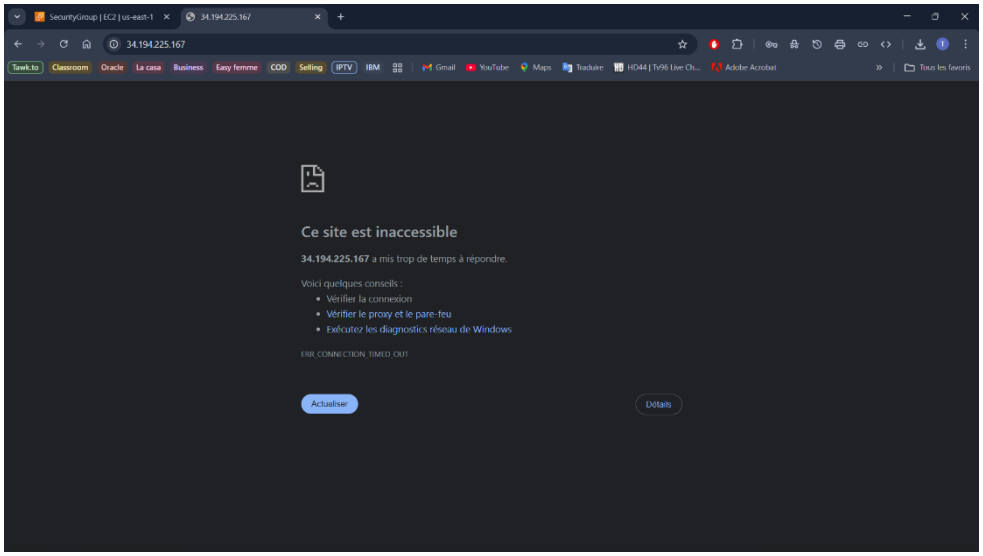


Le Network Border Group :



II.II. La problématique des adresses IP dynamiques :

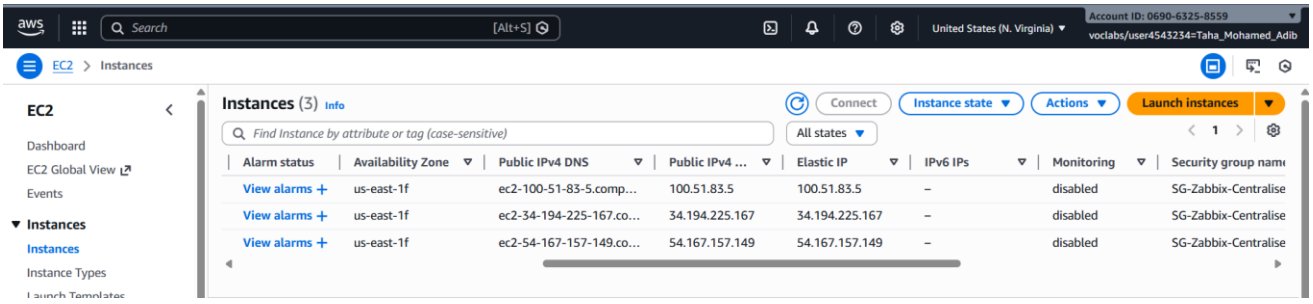
C'est un point crucial de votre projet. Expliquez qu'une instance EC2 classique change d'adresse IP publique à chaque arrêt/redémarrage. Dans un système de supervision, si l'IP du serveur change, tous les agents perdent la connexion.



II.III. Mise en œuvre des Adresses IP Élastiques (EIP) :

Détaillez ici la solution technique pour fixer les adresses :

Définition : Une IP Élastique est une adresse IPv4 statique conçue pour le cloud computing dynamique.
Procédure d'allocation : Expliquez comment vous êtes allé dans la console EC2 > Network & Security > Elastic IPs.
Procédure d'association : Expliquez comment vous avez lié chaque IP à son instance spécifique.
Résultat : Listez vos trois IP finales (Serveur : 34.194.225.167, etc.).



II.IV. Sécurisation des flux : Les Security Groups :

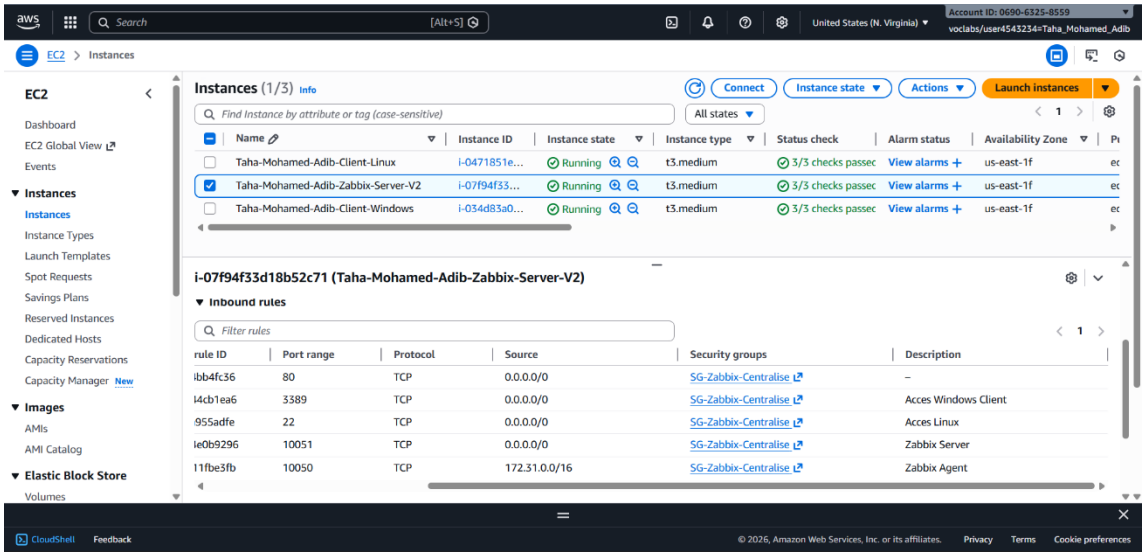
Le Security Group agit comme un pare-feu virtuel. Pour ce projet, vous avez dû configurer des règles entrantes (Inbound Rules) spécifiques :

HTTP (Port 80) : Pour accéder à l'interface web de Zabbix depuis votre domicile.

Zabbix Server (Port 10051) : Pour que les agents puissent envoyer leurs données au serveur.

Zabbix Agent (Port 10050) : Ouvert sur les clients pour que le serveur puisse les interroger.

SSH (22) & RDP (3389) : Pour l'administration à distance des machines.



III. III. Déploiement du Serveur Zabbix via la Technologie Docker :

III.I. Justification de l'approche par conteneurisation:

Le choix de la technologie Docker pour le déploiement du serveur Zabbix n'est pas fortuit. Dans un environnement Cloud comme AWS, l'utilisation de conteneurs offre une flexibilité indispensable.

Isolation des processus : Contrairement à une installation native où les dépendances (PHP, Apache, MySQL) peuvent entrer en conflit avec d'autres services, Docker isole chaque brique logicielle.

Persistance et portabilité : Grâce au fichier de configuration docker-compose.yaml, l'intégralité du serveur peut être déplacée ou réinstallée sur une nouvelle instance en quelques minutes sans perte de données.

Maintenance simplifiée : Les mises à jour de version (par exemple de Zabbix 6.0 à une version supérieure) se limitent à changer une ligne dans le fichier de configuration et à relancer les conteneurs.

III.II. Architecture Micro-services de la Solution :

Pour ce projet, nous avons implémenté une architecture basée sur trois conteneurs distincts qui interagissent de manière transparente au sein d'un réseau virtuel Docker dédié :

Conteneur Base de Données (zabbix-db-mysql) : Ce service utilise l'image MySQL 8.0 pour assurer le stockage de toutes les données de configuration, les utilisateurs, et surtout l'historique des métriques récoltées sur les agents Windows et Linux.

Conteneur Serveur (zabbix-server-mysql) : Il s'agit du composant central qui traite les données reçues, exécute les scripts de vérification et déclenche les alertes en fonction des seuils configurés.

Conteneur Interface Web (zabbix-web-apache-mysql) : Ce module fournit le portail d'administration graphique via un serveur Apache. Il est exposé sur le port 80 de l'instance AWS pour permettre une gestion intuitive via un navigateur web.

III.III. Analyse Technique de la Configuration YAML :

La pierre angulaire du déploiement repose sur le fichier ***docker-compose.yaml*** créé dans le répertoire ***~/zabbix-docker***. Ce fichier définit les paramètres cruciaux pour la stabilité du système :

Variables de liaison : Les paramètres ***DB_SERVER_HOST***, ***MYSQL_USER*** et ***MYSQL_PASSWORD*** permettent d'établir une connexion sécurisée et automatique entre le serveur Zabbix et sa base de données.

Mappage des Ports Stratégiques :

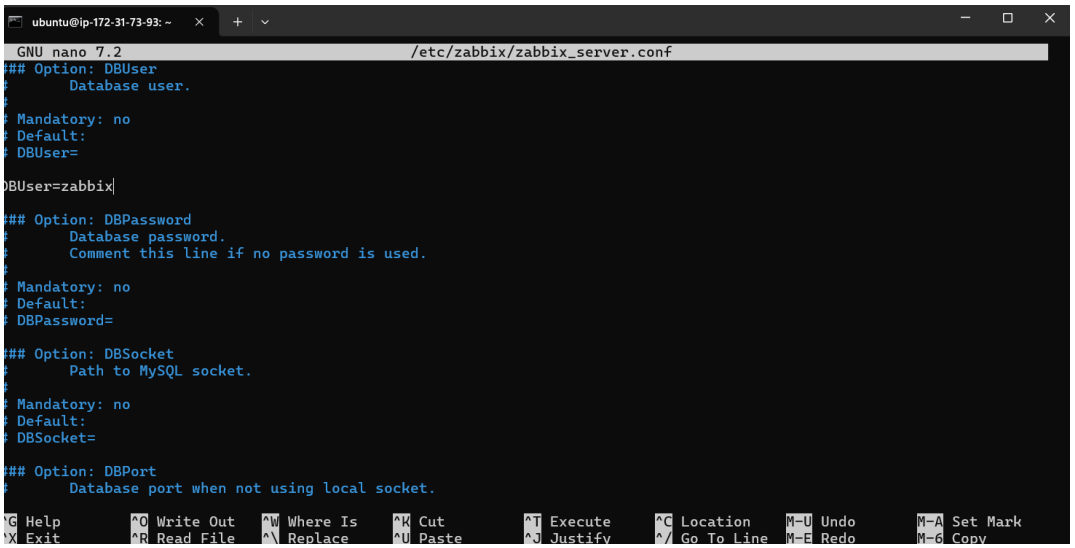
Port 80:8080 : Redirige le trafic HTTP externe vers le service Apache interne.
Port 10051:10051 : Permet au serveur d'écouter les rapports envoyés par les agents en mode actif.

III.IV. Déploiement et Vérification Opérationnelle :

Le déploiement effectif sur l'instance ***34.194.225.167*** a été réalisé après l'installation du moteur Docker et du plugin Docker-Compose.

Lancement du Stack : La commande ***sudo docker-compose up -d*** a été exécutée, déclenchant le téléchargement (Pull) des images officielles Zabbix et leur démarrage en arrière-plan (detached mode).

Vérification du statut : La commande ***sudo docker ps*** permet de confirmer que les trois conteneurs sont non seulement actifs, mais aussi dans un état "Healthy" (en bonne santé), garantissant que l'interface web est prête à recevoir les connexions



IV. Configuration et Déploiement des Agents de Supervision :

IV.I. Rôle et Fonctionnement de l'Agent Zabbix :

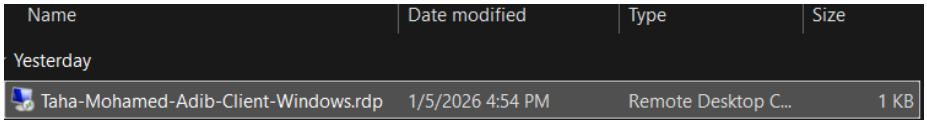
L'agent Zabbix est un processus léger installé sur les machines surveillées pour collecter des données locales (charge CPU, utilisation mémoire, statistiques réseau, état des services) et les transmettre au serveur central. Dans ce projet, nous avons configuré deux types de remontées :

Vérifications Passives : Le serveur Zabbix contacte l'agent sur le port **10050** pour demander une donnée spécifique.

Vérifications Actives : L'agent envoie de lui-même ses données au serveur sur le port **10051**, ce qui permet une supervision plus réactive et réduit la charge de travail du serveur.

IV.II. Déploiement sur l'Instance Windows Server :

L'installation sur l'hôte Windows, identifié dans la console AWS par l'instance **ID i-034d83a0a4efc90cc**, a été réalisée pour permettre une surveillance granulaire du système d'exploitation.



IV.II.I. Paramétrage de l'Agent via l'Interface Graphique (MSI):

L'installation a été effectuée à l'aide du package zabbix_agent-7.4.6-windows-amd64-openssl.msi. Lors de la phase de configuration des services, les paramètres critiques suivants ont été injectés pour assurer la liaison avec le serveur centralisé :

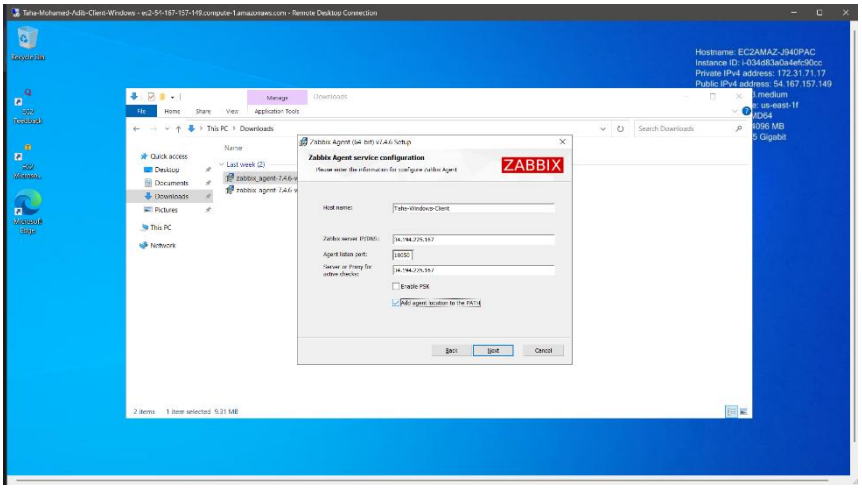
Host name : Taha-Windows-Client (Ce nom a été défini pour correspondre rigoureusement à l'entrée créée ultérieurement dans l'interface Web Zabbix).

Zabbix server IP/DNS : 34.194.225.167. Il s'agit de l'adresse IP publique fixe de notre serveur de supervision.

Agent listen port : 10050. Port standard sur lequel l'agent écoute les requêtes de type "passive checks" provenant du serveur.

Server or Proxy for active checks : 34.194.225.167. Cette configuration active la capacité de l'agent à pousser lui-même des données vers le serveur de manière proactive.

Options de déploiement : L'option "***Add agent location to the PATH***" a été activée pour permettre l'exécution des utilitaires Zabbix directement depuis n'importe quel terminal Windows (CMD ou PowerShell).



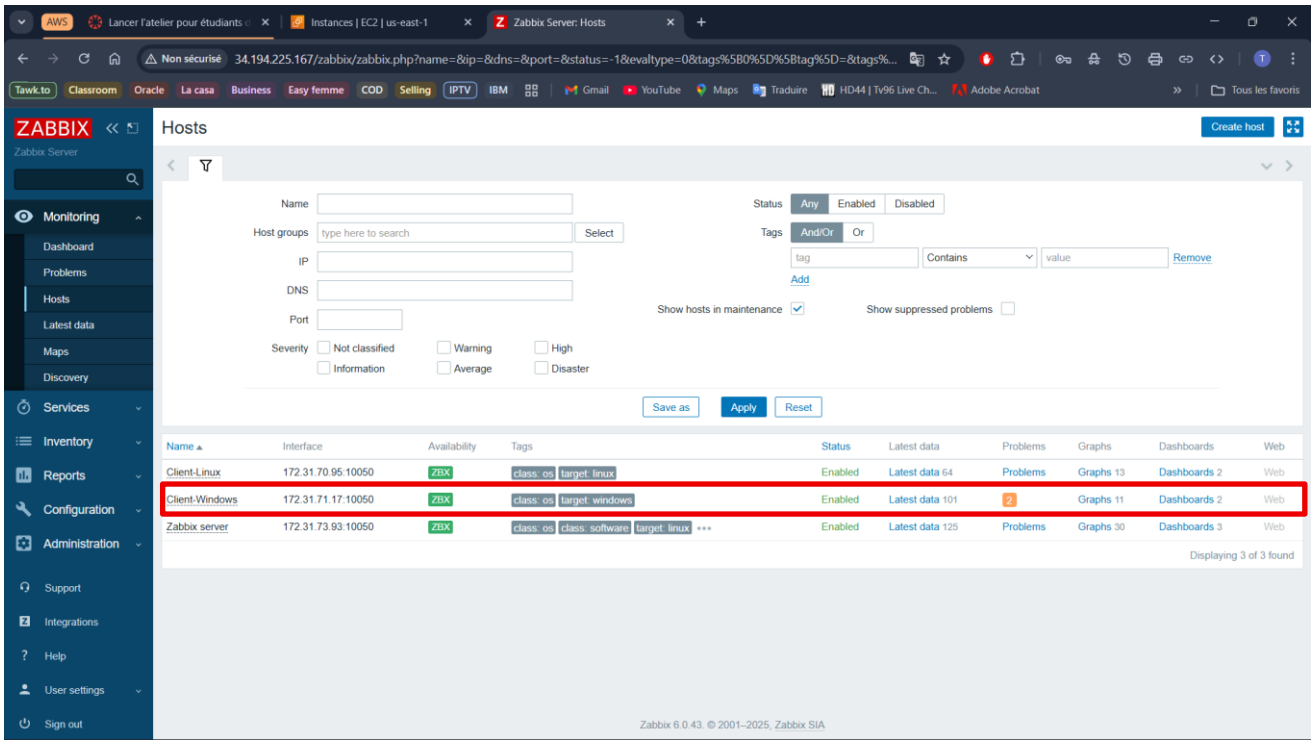
IV.II.II. Intégration et Validation dans l'Interface Web :

Une fois l'agent installé et les flux réseaux ouverts, l'hôte a été intégré dans le tableau de bord Zabbix.

Configuration de l'interface : L'hôte **Client-Windows** a été configuré pour communiquer via son interface réseau privée ou publique.

Statut de disponibilité : La réussite de l'opération est confirmée par l'indicateur **ZBX** passant au **VERT** dans la colonne "**Availability**".

Statut de l'hôte : Le système affiche un statut "**Enabled**", signifiant que la collecte de données (items) et l'analyse des déclencheurs (triggers) sont actives.



V. Exploitation, Validation et Analyse des Résultats :

V.I. Accès à l'Interface d'Administration Centralisée :

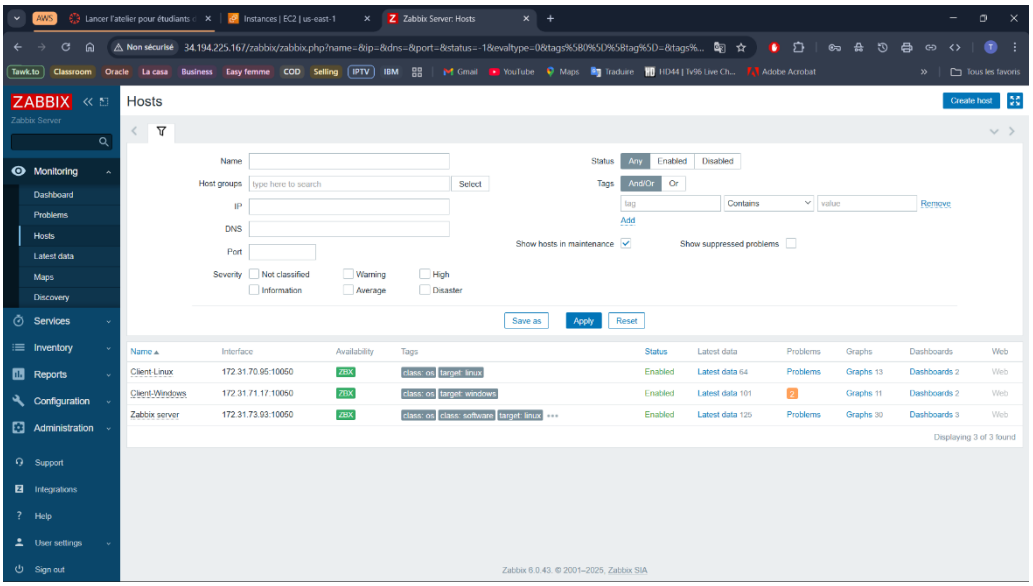
Le succès du déploiement se traduit par l'accessibilité de la console de gestion Web via l'adresse IP Élastique configurée (**34.194.225.167**). Cette interface PHP, propulsée par un serveur Apache conteneurisé, constitue le point unique de contrôle pour l'ensemble de l'infrastructure supervisée.

Authentification sécurisée : L'accès a été validé via le compte administrateur par défaut (**Admin** / **zabbix**).
Disponibilité du service : Malgré les tentatives infructueuses initiales dues aux pare-feu (Time-out), l'ouverture du port **80** sur AWS a permis de rendre l'interface opérationnelle.

V.II. Validation de la Connectivité des Hôtes (Voyants ZBX) :

La preuve ultime de la réussite du projet est visible dans la section Monitoring > Hosts. Le tableau de bord confirme que la communication bidirectionnelle est établie :

Client-Linux : Connecté via l'adresse IP interne du sous-réseau AWS (**172.31.70.95**), l'indicateur **ZBX** est affiché en **VERT**, confirmant que l'agent Ubuntu répond aux requêtes du serveur.
Client-Windows : Connecté via l'interface réseau **172.31.71.17**, l'agent Windows Server communique parfaitement avec le serveur central, comme en témoigne le voyant de disponibilité actif.
Zabbix Server : Le serveur s'auto-supervise également via son interface locale, garantissant l'intégrité du moteur de monitoring.



V.III. Collecte et Analyse des Métriques (Latest Data) :

Une fois les hôtes activés, le système a commencé à accumuler des données en temps réel (Latest Data) :

Quantité de données : Pour le **Client-Windows**, plus de 100 items (métriques) sont collectés simultanément, couvrant l'utilisation du **CPU**, de la **RAM** et de l'espace disque.

Supervision Linux : 64 métriques sont actives pour le client **Ubuntu**, permettant une surveillance fine des processus système.

Gestion des problèmes : L'interface identifie en temps réel les alertes (Triggers). Par exemple, deux problèmes mineurs ont été détectés sur le client **Windows**, permettant une intervention proactive avant toute interruption de service.

V.IV. Synthèse de la Performance Réseau :

L'utilisation des adresses IP Élastiques a permis de stabiliser les flux sur les ports de supervision (**10050** et **10051**). Cette architecture garantit que, même après une maintenance ou un redémarrage des instances **AWS**, le système de supervision reste opérationnel sans intervention manuelle sur les fichiers de configuration des agents.

VI. VI. Résolution de Problèmes (Troubleshooting) et Analyse des Incidents :

Cette section documente les défis techniques rencontrés lors du cycle de vie du projet. La résolution d'incidents est une compétence critique démontrée ici par une approche méthodologique.

VI.I. Diagnostic de l'indisponibilité de l'interface Web (Timeout) :

L'un des obstacles majeurs a été l'impossibilité d'accéder à l'interface Zabbix via l'adresse http://34.194.225.167/zabbix au début du projet

.

Symptôme : Le navigateur affichait l'erreur **ERR_CONNECTION_TIMED_OUT** après une longue attente.

Analyse : Le diagnostic a porté sur trois couches : le service Docker, le pare-feu local (**UFW**) et les **Security Groups AWS**. Bien que les conteneurs soient "**Healthy**", la requête n'atteignait jamais le serveur.

Résolution : Le problème se situait au niveau de la couche réseau externe **d'AWS**. L'ajout d'une règle entrante (**Inbound Rule**) autorisant le protocole **HTTP** sur le port **80** pour la source **0.0.0.0/0** a été nécessaire pour lever le blocage.

VI.II. Instabilité liée au redémarrage des instances (IP Dynamiques) :

Initialement, la configuration reposait sur les adresses IP publiques assignées dynamiquement par AWS.

Problématique : Lors d'une interruption de service ou d'une maintenance, l'adresse IP du serveur changeait, rendant les fichiers de configuration des agents (**zabbix_agentd.conf**) obsolètes instantanément.

Solution pérenne : La mise en œuvre d'adresses **IP Élastiques** (**EIP**) a permis de fixer l'identité réseau de chaque machine. Cette étape a nécessité une réallocation complète dans la console **EC2** et une mise à jour des paramètres des agents Windows et Linux pour pointer vers la nouvelle IP fixe **34.194.225.167**.

VI.III. Conflits de dépendances système sur Ubuntu :

L'installation de l'agent sur le client Linux a révélé des incompatibilités de bibliothèques.

Incident : Lors de l'exécution de **apt install zabbix-agent**, le système renvoyait des erreurs de dépendances liées à **libcurl** et **libssl**.

Correction : Il a fallu purger les anciens dépôts, mettre à jour le cache **APT** avec le paquet **zabbix-release** officiel et forcer l'installation des versions compatibles avec l'architecture de l'instance.

VII. VII. Conclusion Générale et Perspectives :

VII.I. Synthèse de la réalisation :

Ce projet de mise en œuvre d'une plateforme de supervision **Zabbix** sur l'infrastructure **Cloud AWS** a permis de démontrer la faisabilité et l'efficacité de la surveillance d'un parc hybride. En combinant la puissance de la conteneurisation Docker et la flexibilité des services **AWS**, nous avons érigé une architecture capable de surveiller en temps réel des serveurs aux systèmes d'exploitation radicalement différents (Ubuntu Linux et Windows Server).

Les objectifs initiaux ont tous été atteints :

Centralisation : Un point unique de contrôle via une interface web fluide et réactive.

Fiabilité : Une connexion stable garantie par l'utilisation d'IP Élastiques.

Visibilité : Une collecte de données excédant 160 métriques cumulées, permettant une vision précise de la santé du parc informatique**.**

VII.II. Apports personnels et professionnels :

Sur le plan technique, ce travail a permis de consolider des compétences avancées en administration système (Linux/Windows), en réseaux (routage, pare-feu, gestion d'IP fixes) et en technologies DevOps (Docker-Compose). La gestion des incidents rencontrés a également renforcé une capacité d'analyse rigoureuse face aux contraintes de sécurité imposées par les environnements Cloud professionnels.

VII.III. Perspectives d'évolution :

Bien que la plateforme soit pleinement opérationnelle, elle constitue une base évolutive qui pourrait être enrichie par :

Le durcissement de la sécurité : Mise en place d'un certificat SSL (**HTTPS**) pour l'interface web et utilisation de tunnels chiffrés (**TLS**) pour les communications entre agents et serveur.

L'automatisation des alertes : Configuration d'un serveur de messagerie (**SMTP**) ou d'une passerelle **Webhook** pour recevoir des notifications critiques sur smartphone ou par email en cas de panne.

L'auto-scaling : Utiliser les métriques **Zabbix** pour déclencher automatiquement la création de nouvelles instances **AWS** en cas de surcharge des serveurs actuels.

En conclusion, ce projet valide la robustesse de **Zabbix** comme outil de supervision leader et souligne l'importance d'une préparation minutieuse des couches réseau dans tout déploiement Cloud d'envergure.