

Lab 02 - Cybersecurity 3

Installation Wazuh

1. Mise en place

Pour mettre en place la stack, je décide de passer par un `docker-compose.yml` unifié, avec un réseau partagé en mode "pont" pour des sorties internet s'il le faut.

En plus de la stack Wazuh (Indexer, Manager, Dashboard) et du Ubuntu, je tente un truc sympa avec un Docker... Windows !

Comme je suis déjà sur Windows et que Docker tourne sur WSL2, je dois modifier quelques options dans WSL pour autoriser la virtualisation imbriquée, pousser la config du Docker Engine jusqu'à 16GB de RAM (c'est que c'est gourmand Wazuh) mais ça passe.

Il a fallu également des fichiers de génération de certificats. Pour m'aider, j'ai utilisé l'IA Kimi K2.7 Code :

`certs_gen.sh`

```
# Create working directory
mkdir -p config/wazuh_indexer_ssl_certs
cd config

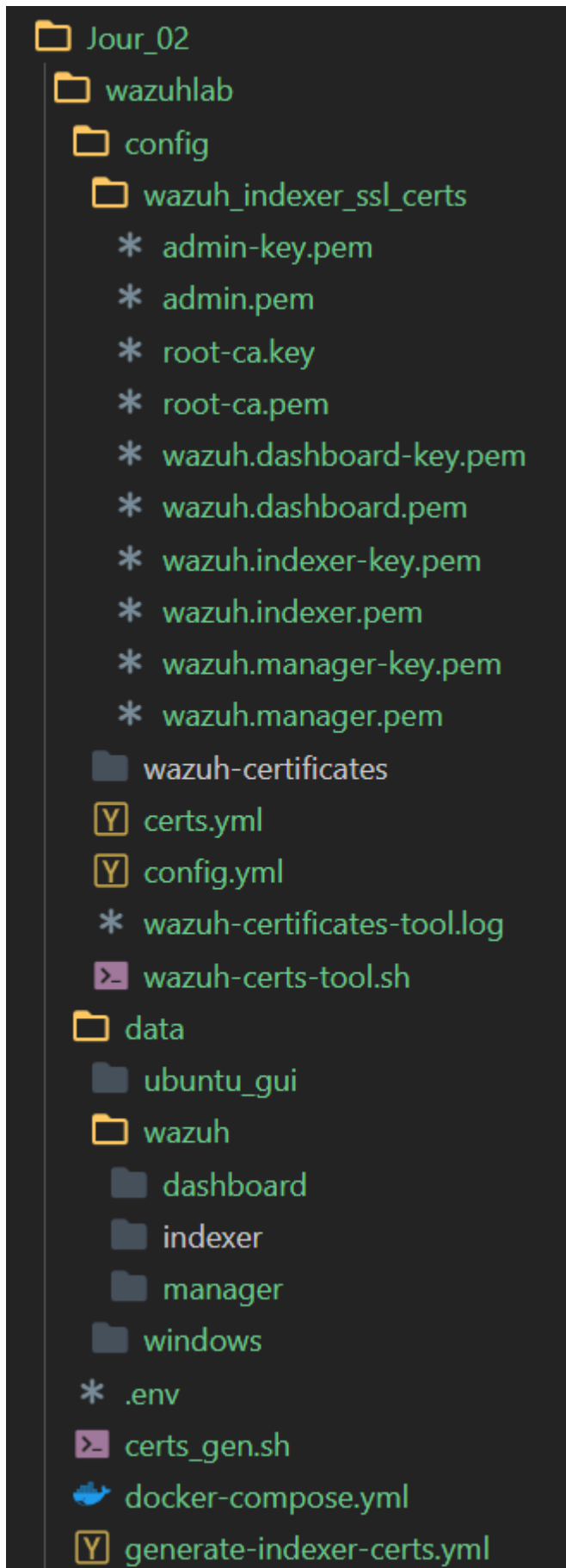
# Download the certificate tool
curl -sO https://packages.wazuh.com/4.9/wazuh-certs-tool.sh

# Create the configuration file
cat > config.yml <<EOF
nodes:
  indexer:
    - name: wazuh.indexer
      ip: wazuh.indexer
  server:
    - name: wazuh.manager
      ip: wazuh.manager
  dashboard:
    - name: wazuh.dashboard
      ip: wazuh.dashboard
EOF

# Generate certificates
chmod +x wazuh-certs-tool.sh
./wazuh-certs-tool.sh -A

# Move the generated certificates to the expected path
mv wazuh-certificates/* wazuh_indexer_ssl_certs/
```

Ca fonctionne :



En fait...

"Je ne dirais pas que c'est un échec... ça n'a pas marché."

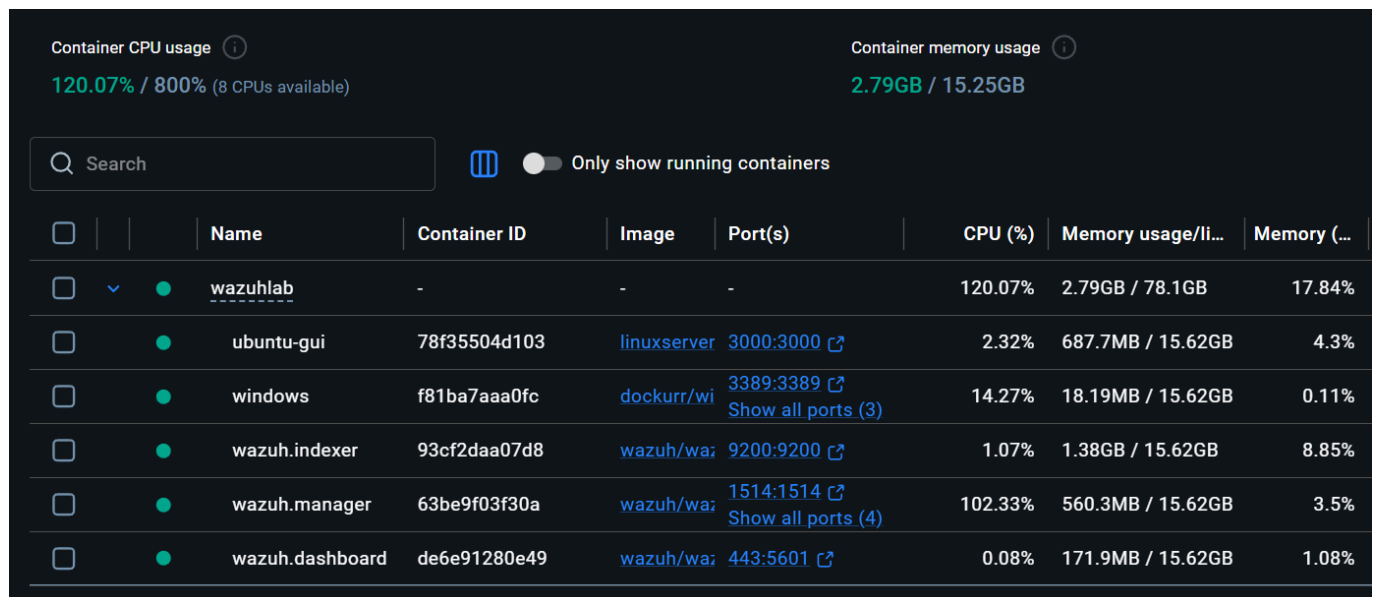
Le docker-compose n'a pas abouti, et à 14h j'ai migré : Wazuh est maintenant déployé sur VirtualBox, et le docker-compose.yml ne donne que du Ubuntu.

2. Fichier auth.log

Avec WSL2, adaptation du script en mono-ligne :

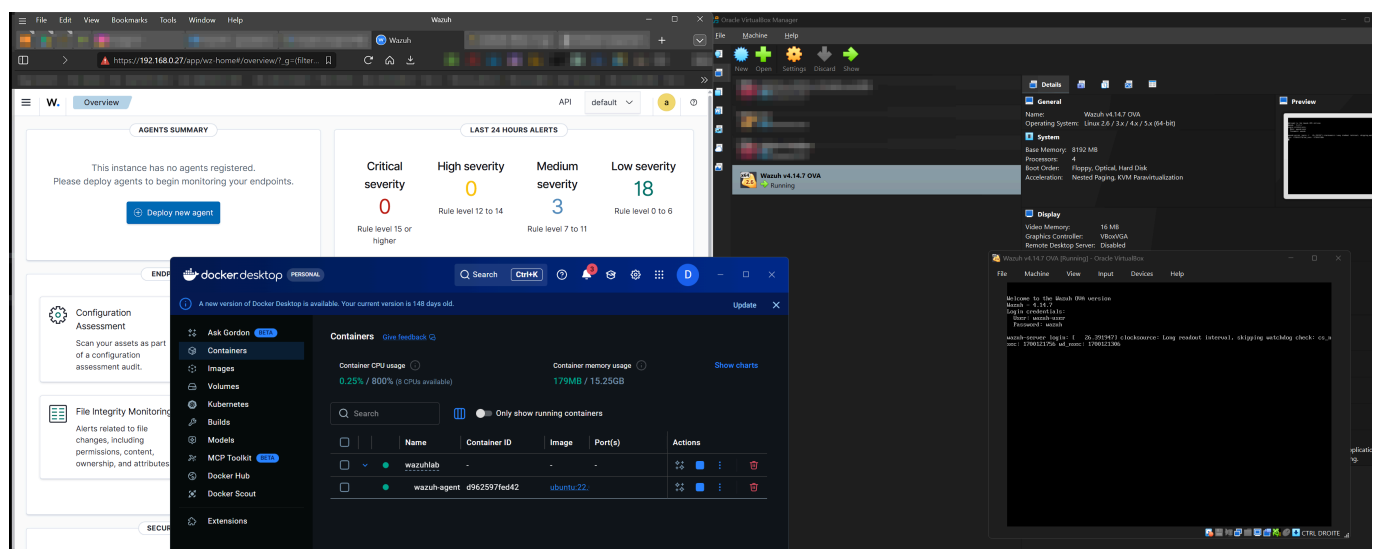
```
bash -c '> auth.log && for i in {1..40}; do echo "Jul 21 12:00:$(printf "%02d" $i) ubuntu sshd[12345]: Failed password for invalid user user$i from 192.168.0.$(($i%5+1)) port $(($5000+i)) ssh2" >> auth.log; done && echo "Jul 21 12:01:00 ubuntu sshd[12345]: Accepted password for validuser from 192.168.0.1 port 5000 ssh2" >> auth.log'
```

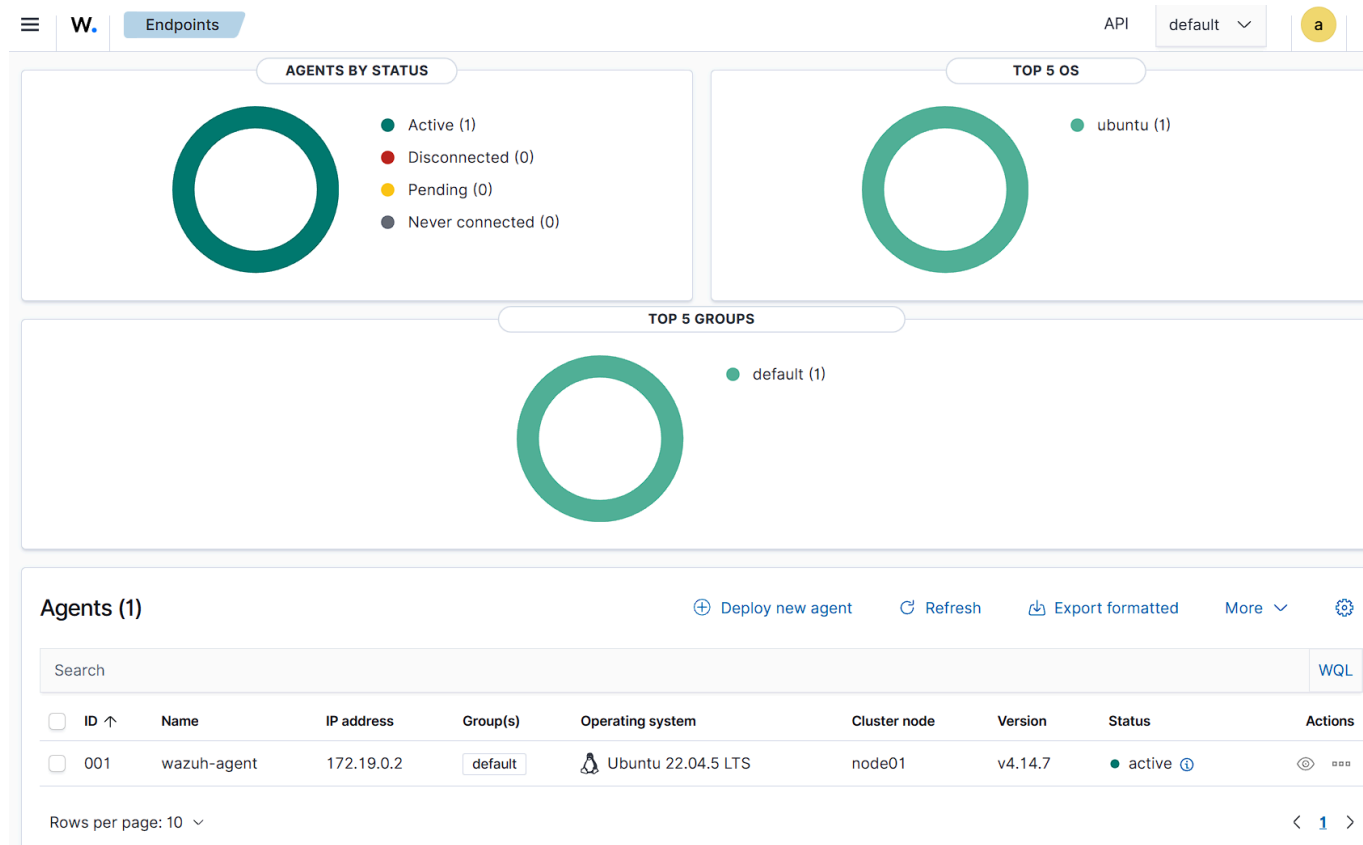
3. Démarrage



	Name	Container ID	Image	Port(s)	CPU (%)	Memory usage/li...	Memory (...)
☐	wazuhlab	-	-	-	120.07%	2.79GB / 78.1GB	17.84%
☐	ubuntu-gui	78f35504d103	linuxserver	3000:3000	2.32%	687.7MB / 15.62GB	4.3%
☐	windows	f81ba7aaa0fc	dockurr/wi	3389:3389 Show all ports (3)	14.27%	18.19MB / 15.62GB	0.11%
☐	wazuh.indexer	93cf2daa07d8	wazuh/wa	9200:9200	1.07%	1.38GB / 15.62GB	8.85%
☐	wazuh.manager	63be9f03f30a	wazuh/wa	1514:1514 Show all ports (4)	102.33%	560.3MB / 15.62GB	3.5%
☐	wazuh.dashboard	de6e91280e49	wazuh/wa	443:5601	0.08%	171.9MB / 15.62GB	1.08%

Avec la refonte de l'infra





4. Mettre en place la règle

La nouvelle règle est réécrite selon les derniers standards de Wazuh en XML :

```
<group name="local,syslog,sshd,">
  <rule id="100001" level="10" frequency="5" timeframe="60">
    <if_matched_sid>5710</if_matched_sid>
    <srcip>192.168.0.0/24</srcip>
    <description>Plus de 5 tentatives SSH echouees en 1 minute</description>
  </rule>
</group>
```

```
GNU nano 8.3 /var/ossec/etc/rules/local_rules.xml
<group name="local,syslog,sshd,">
  <rule id="100001" level="10" frequency="5" timeframe="60">
    <if_matched_sid>5710</if_matched_sid>
    <srcip>192.168.0.0/24</srcip>
    <description>Plus de 5 tentatives SSH echouees en 1 minute</description>
  </rule>
</group>
```

5. Relever les anomalies

Première "attaque" depuis l'agent Ubuntu :

```

for i in {1..100}; do
    RAND_IP="192.168.0.$((RANDOM % 254 + 1))"
    RAND_USER="user_$((RANDOM % 20 + 1))"
    RAND_PORT=$((RANDOM % 5000 + 10000))
    TIME=$(date +%b %d %H:%M:%S")

    echo "$TIME ubuntu sshd[$$]: Failed password for invalid user $RAND_USER from
$RAND_IP port $RAND_PORT ssh2" >> auth.log
    sleep 0.05
done

```

Avec d'autres règles également, on peut tester des "attaques" différentes :

```

for i in {1..100}; do
    RAND_IP="192.168.0.$((RANDOM % 254 + 1))"
    RAND_PORT=$((RANDOM % 5000 + 10000))
    TIME=$(date +%b %d %H:%M:%S")

    echo "$TIME ubuntu sshd[1234]: Failed password for root from $RAND_IP port
$RAND_PORT ssh2" >> auth.log
    sleep 0.05
done

```

```

for i in {1..100}; do
    TIME=$(date +%b %d %H:%M:%S")

    echo "$TIME ubuntu sudo:  attacker : 3 incorrect password attempts ; TTY=pts/1
; PWD=/tmp ; USER=root ; COMMAND=/bin/bash" >> auth.log
    sleep 0.05
done

```

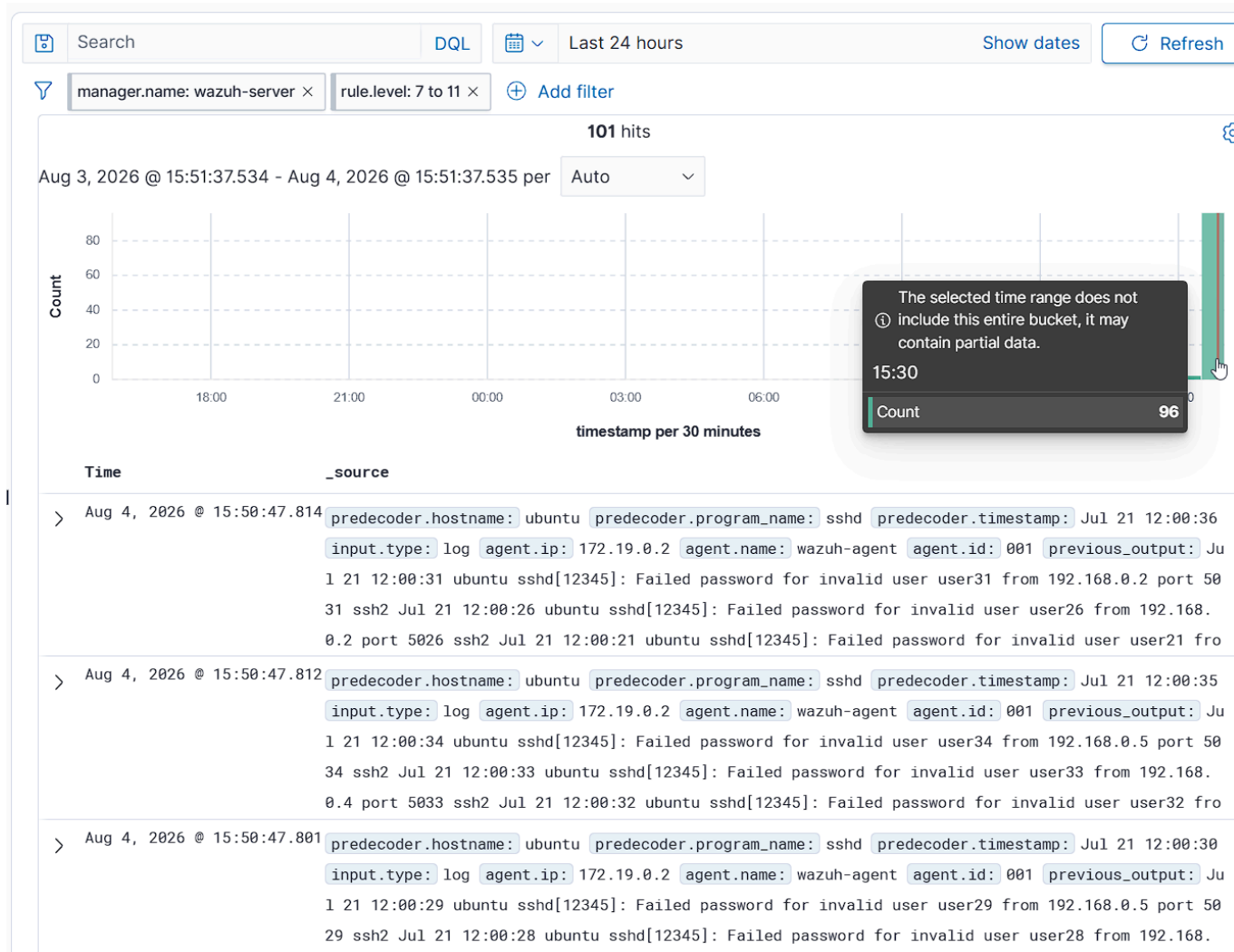
```

for i in {1..100}; do
    RAND_IP="192.168.0.$((RANDOM % 254 + 1))"
    RAND_PORT=$((RANDOM % 5000 + 10000))
    TIME=$(date +%b %d %H:%M:%S")

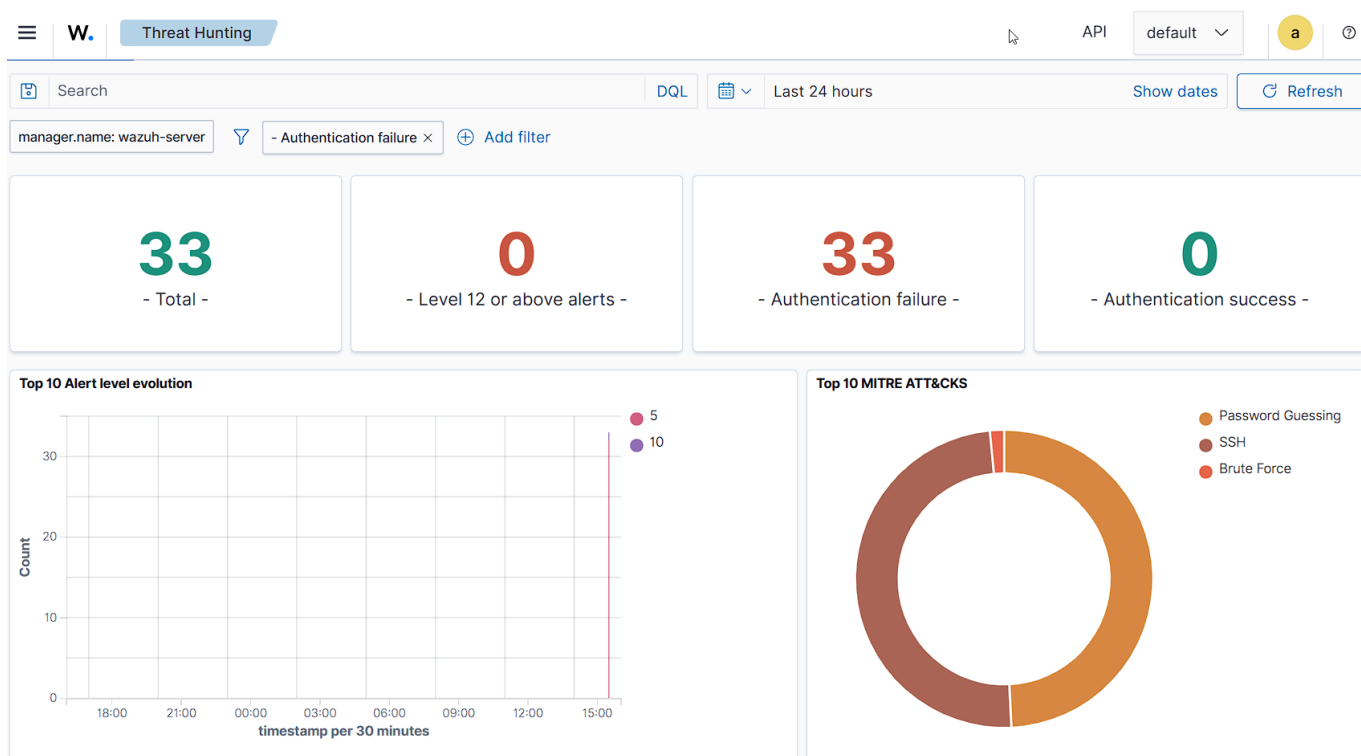
    echo "$TIME ubuntu sshd[1234]: Accepted password for root from $RAND_IP port
$RAND_PORT ssh2" >> auth.log
    sleep 0.05
done

```

Détection comme risque "faible" :



Dans le panneau "Threat Hunting", plus prédictif, un détail apparaît :

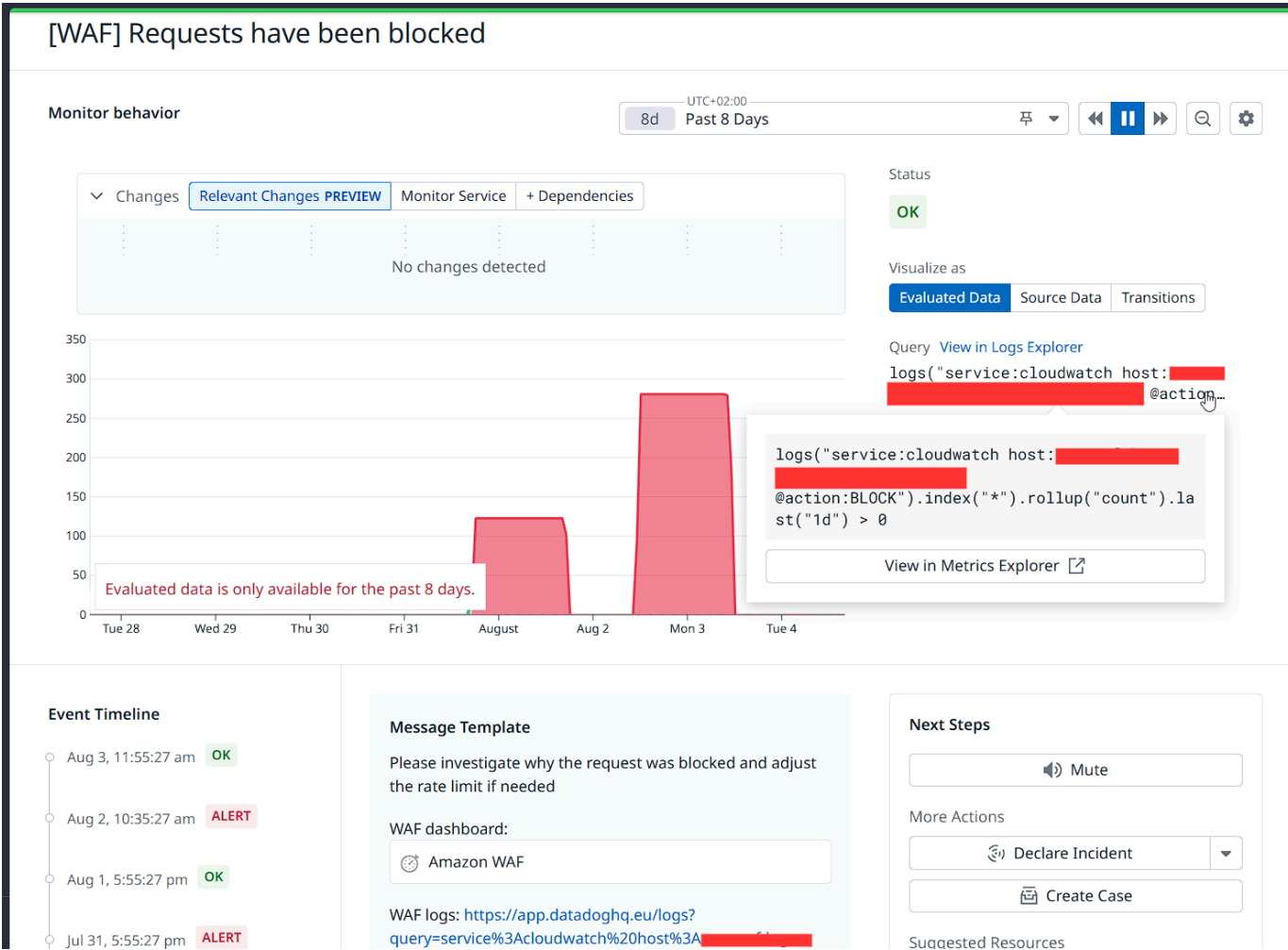


Annexe : Travaux et Moniteurs réels sur Datadog

Pour montrer un peu à quoi ressemble de la prédiction froide sur les logs suspects, des exemples de VRAIS moniteurs déployés sur Datadog, outil français de gestion centralisée des journaux, avec outils de détections divers :

The screenshot displays the Datadog Monitors interface. The left sidebar contains navigation links for various Datadog features. The main content area shows a list of monitors under the 'List' tab, filtered by the user 'Gauvain BOICHÉ'. The filters on the left include Status (Triggered, Alert, Warn, No Data, OK), Muted (True, Muted elapsed, Muted left, False), Notification Recipient, Draft Status (Published, Draft), and Priority (P1, P2, P3). The monitor list table has columns for Priority, Status, Muted Left, Name, and Tags. The monitors listed include alerts for EKS cluster names, GPU usage, RabbitMQ account names, Lambda account names, API account names, EBS account names, Karpenter ServiceLinkRoleCreation, Keda jobs objects, and Kube deployment names.

PRIORITY	STATUS	MUTED LEFT	NAME	TAGS
P2	ALERT		[EKS] {{kubernetes_cluster_name.name}} : {{kubernetes_cluster_name.name}}	team:infra
P3	ALERT	∞	DRAFT [EKS] GPU in {{kubernetes_cluster_name.name}}	team:infra
P2	WARN		[RabbitMQ] {{aws_account_name.name}} : {{aws_account_name.name}}	team:infra
P2	NO DATA		[EKS] {{kubernetes_cluster_name.name}} : {{pod_name.name}}	team:infra
P2	OK		[LAMBDA] {{aws_account_name.name}} : {{function_name.name}}	team:infra
P2	OK		[API] {{aws_account_name.name}} : {{api_name.name}}	team:infra
P3	NO DATA	∞	[EBS] {{aws_account_name.name}} : Burst buffer	team:infra
P3	OK		[EKS] Karpenter has a "ServiceLinkRoleCreation"	team:infra
-	OK	∞	DRAFT [EKS] Keda jobs objects growing un	
P2	OK		[EKS] {{aws_account_name.name}} : {{cluster_name.name}}	team:infra
-	OK	∞	DRAFT [EKS] {{cluster_name.name}} : {{pod_name.name}}	
P2	WARN	∞	DRAFT [EKS] {{kubernetes_cluster_name.name}} :	team:infra
P2	OK		[EKS] {{kubernetes_cluster_name.name}} : {{pod_name.name}}	team:infra
P3	ALERT	∞	DRAFT [EKS] {{kubernetes_deployment.name}} is	team:infra
-	ALERT	∞	DRAFT [EKS] {{service.name}} has keda job	



Sans compter le déploiement de WAF dans des comptes AWS par Terraform depuis des modules personnalisés. Que je n'ai pas fait, mais que j'ai amélioré avec des lifecycle, etc.