

Rendu Projet d'études
PROJET PRO M2 DO



2022-2023

MSI 5-23 DO A

Sébastien Lafont

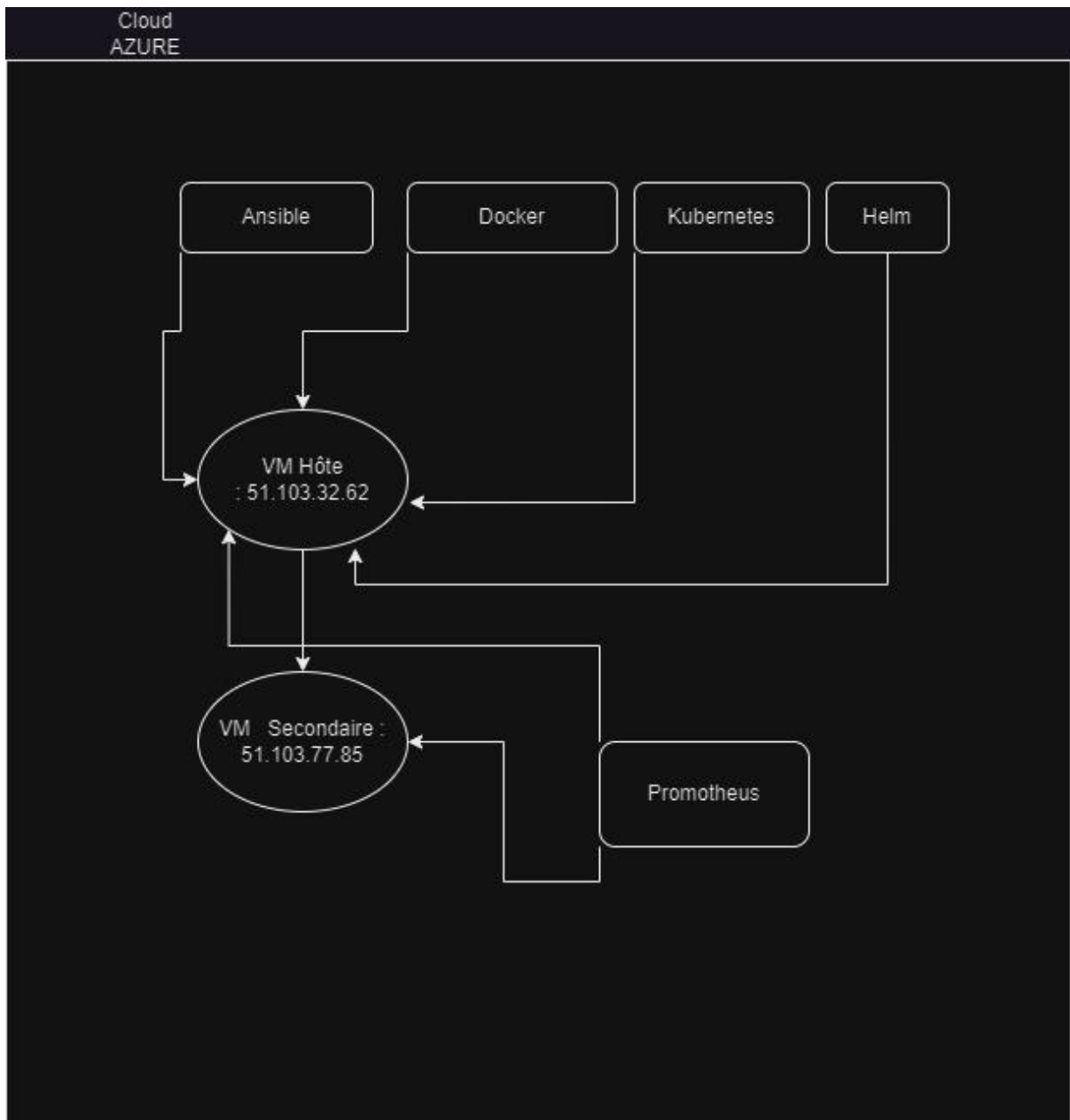
Ali-Eren Kartal

Bonjour,

Dans ce document, nous détaillerons notre solution.

Nous passerons en revue les différentes solutions employées.
Ainsi que les procédés pour les installer et les configurer.

Nous vous présenterons tout d'abord notre infrastructure.



Voici notre schéma: Vous pouvez constater que nous avons 2 machines virtuelles.

Il s'agit de 2 machines tournant sous Ubuntu en version 22.04:

- La machine principale 51.103.32.62 nommée " projet " (VM Hôte)
- et la machine secondaire 51.103.77.85 nommée " projet-test "(VM Secondaire).

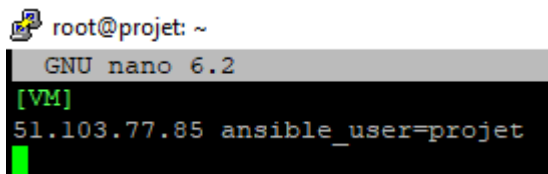
La machine principale, héberge Ansible.

Ceci nous permet donc de déployer des playbooks (fichier de déploiements, d'applications sur une machine cible), et d'installer, dans notre cas : Docker, Helm et Kubernetes.

Ansible nécessite plusieurs prérequis, afin de pouvoir exécuter les playbooks :

- Il faut d'abord avoir un fichier inventaire : inventory.yml ou alors inventory.ini.

Dans notre cas, nous avons opté pour le fichier inventory.ini.



```
root@projet: ~
GNU nano 6.2
[VM]
51.103.77.85 ansible_user=projet
```

Ce fichier nous permet de déclarer notre VM secondaire.

- Nous devons aussi créer une clé SSH Publique, et la copier sur la machine distante. Car **Ansible** utilise SSH pour se connecter en toute sécurité.

On peut la créer en utilisant SSH-KEYGEN, puis on la copie sur l'autre machine à l'aide de la commande `ssh-copy-id -i ~/.ssh/id_rsa.pub projet@51.103.77.85`.

Suite à cela, nous pouvons déployer les playbooks : avec la commande "ansible-playbook -i inventory.ini install docker.yml".

- **Ansible-playbook** indique que nous voulons faire un déploiement, **-i** indique le fichier inventaire, suivi du nom de notre fichier **YAML**.

Tout d'abord, nous choisissons de nommer celle-ci [VM].

Puis l'on renseigne l'adresse ip, ainsi que le compte avec lequel **Ansible** se connectera à la machine.

Nous passons ensuite au playbook d'installation de Docker : cf le fichier ci dessous.

root@projet: ~

GNU nano 6.2

--

- name: Docker Installation

hosts: VM

remote_user: projet

become: yes

become_user: root

tasks:

- name: Update apt cache

apt:

update_cache: yes

tags: [docker]

- name: Install apt-transport-https package

apt:

name: apt-transport-https

state: present

tags: [docker]

- name: Install ca-certificates package

apt:

name: ca-certificates

state: present

tags: [docker]

- name: Install curl package

apt:

name: curl

state: present

tags: [docker]

- name: Install software-properties-common package

apt:

name: software-properties-common

state: present

tags: [docker]

- name: Add Docker GPG key

apt_key:

url: https://download.docker.com/linux/ubuntu/gpg

state: present

tags: [docker]

- name: Add Docker repository

apt_repository:

repo: deb [arch=amd64] https://download.docker.com/linux/ubuntu {{ ansible_lsb.codename }} stable

state: present

tags: [docker]

- name: Install Docker

apt:

name: docker.io

state: present

tags: [docker]

- name: Start Docker service

service:

name: docker

state: started

enabled: yes

tags: [docker]

Nous devons renseigner notre machine dans la ligne hosts, sur laquelle le playbook s'exécutera, ainsi que l'utilisateur pour se connecter sur la machine.

Nous prenons le choix de devenir ROOT, pour ne pas être empêchés par des droits insuffisants.

Nous mettons à jour le gestionnaire apt, et installons des paquets nous permettant de télécharger sur des sources HTTPS.

Nous ajoutons aussi la clé GPG de docker, ce qui nous permet de vérifier l'authenticité de ce que nous téléchargeons.

Enfin, évidemment nous installons Docker et démarrons le service sur la machine distante.

Docker est installé ainsi que toutes les dépendances.

A propos de **Kubernetes** : dont le fichier ci dessous

```

--
- hosts: VM
  remote_user: projet
  become: yes
  become_method: sudo
  become_user: root
  gather_facts: yes
  connection: ssh

tasks:
  - name: Create containerd config file
    file:
      path: "/etc/modules-load.d/containerd.conf"
      state: "touch"

  - name: Add conf for containerd
    blockinfile:
      path: "/etc/modules-load.d/containerd.conf"
      block: |
        overlay
        br_netfilter

  - name: modprobe
    shell: |
      sudo modprobe overlay
      sudo modprobe br_netfilter

  - name: Set system configurations for Kubernetes networking
    file:
      path: "/etc/sysctl.d/99-kubernetes-cri.conf"
      state: "touch"

  - name: Add conf for containerd
    blockinfile:
      path: "/etc/sysctl.d/99-kubernetes-cri.conf"
      block: |
        net.bridge.bridge-nf-call-iptables = 1
        net.ipv4.ip_forward = 1
        net.bridge.bridge-nf-call-ip6tables = 1

  - name: Apply new settings
    command: sudo sysctl --system

  - name: install containerd
    shell: |
      sudo apt-get update && sudo apt-get install -y containerd
      sudo mkdir -p /etc/containerd
      sudo containerd config default | sudo tee /etc/containerd/config.toml
      sudo systemctl restart containerd

  - name: disable swap
    shell: |
      sudo swapoff -a
      sudo sed -i 's/ swap / s/^(.*)$/#\1/g' /etc/fstab

  - name: install and configure dependencies
    shell: |
      sudo apt-get update && sudo apt-get install -y apt-transport-https curl
      curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -

  - name: Create kubernetes repo file
    file:
      path: "/etc/apt/sources.list.d/kubernetes.list"
      state: "touch"

  - name: Add K8s Source
    blockinfile:
      path: "/etc/apt/sources.list.d/kubernetes.list"
      block: |
        deb https://apt.kubernetes.io/ kubernetes-xenial main

  - name: install kubernetes
    shell: |
      sudo apt-get update
      sudo apt-get install -y kubelet=1.20.1-00 kubeadm=1.20.1-00 kubectl=1.20.1-00
      sudo apt-mark hold kubelet kubeadm kubectl

```

Le fichier est assez similaire au Docker, car il s'agit du même langage (YAML Yet Another Market Language), qui est utilisé par Ansible pour les playbooks.

Nous devons également spécifier la même machine, le compte est passé en ROOT.

Cette fois, nous créons un fichier de configuration vide `containerD`, auquel on ajoute les valeurs "overlay" et "br_netfilter", puis on charge les modules des noyaux.

Ensuite nous configurons le fichier de configuration réseau, vide mais on le peuple ensuite avec "net.bridge.bridge-nf-call-iptables = 1", "net.ipv4.ip_forward = 1" et "net.bridge.bridge-nf-call-ip6tables = 1". On applique ensuite les nouvelles configuration système.

On désactive la swap, ce qui est un prérequis fondamental de kubernetes. On installe la clé GPG de ce dernier.

Puis on installe les différents logiciels Kube, tels que kubelet, kubectl, kubeadm.

Désormais Kubernetes est installé sur le serveur.

Voici le playbook d'installation de **Helm**.

```
root@projet: ~  
GNU nano 6.2  
--  
- name: Install Helm  
  hosts: VM  
  remote_user: projet  
  become: true  
  
  tasks:  
    - name: Download Helm installation script  
      get_url:  
        url: https://get.helm.sh/helm-v3.6.3-linux-amd64.tar.gz  
        dest: /tmp/helm.tar.gz  
  
    - name: Extract Helm archive  
      unarchive:  
        src: /tmp/helm.tar.gz  
        dest: /tmp  
        remote_src: true  
        creates: /tmp/linux-amd64/helm  
  
    - name: Copy Helm binary to /usr/local/bin  
      copy:  
        src: /tmp/linux-amd64/helm  
        dest: /usr/local/bin/helm  
        mode: '0755'  
  
    - name: Verify Helm installation  
      command: helm version  
      register: helm_version_output  
      changed_when: false  
  
    - name: Display Helm version  
      debug:  
        var: helm_version_output.stdout_lines
```

Nous téléchargeons le script d'installation de Helm qui est contenu au sein d'une archive .tar.gz. Après extraction, on copie le contenu et on vérifie l'installation.

Voilà, nous avons maintenant toutes les dépendances installées.

Passons aux solutions à déployer.

Parlons d'abord de **Wazuh**.



Wazuh est une solution de gestion de la sécurité complète qui permet la surveillance, la détection des menaces et la conformité réglementaire. Elle est Open Source.

Fonctionnalités :

Wazuh offre des fonctionnalités avancées de collectes et d'analyse des journaux, de corrélation d'événements, de gestion des vulnérabilités et de visualisation des alertes.

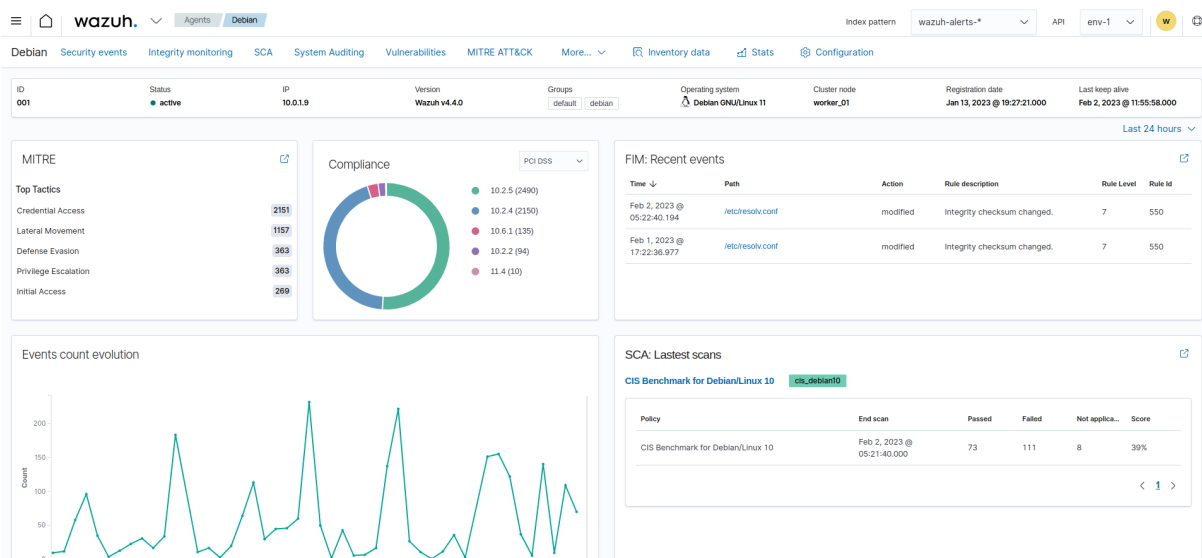
Wazuh fonctionne grâce à plusieurs composants:

Il y a tout d'abord l'indexeur, il s'agit d'un moteur d'analyse et de recherche textuelle, il indexe et stocke les alertes, générées par le wazuh serveur.

Ensuite le serveur analyse les données reçues par les agents, il peut gérer plusieurs agents et les configurer à distance.

Enfin il y a le dashboard, il s'agit là de l'interface graphique.

Celle ci permet de voir toutes les informations remontées.



Nous allons maintenant voir comment installer et configurer la solution Wazuh.

Nous utiliserons Docker pour cette installation:

- pour installer Wazuh, nous devons cloner le git : `git clone https://github.com/wazuh/wazuh-docker.git -b v4.4.5`
- nous avons maintenant un dossier wazuh, nous entrons dans celui-ci afin de naviguer jusqu'au dossier single-nodes, pour executer la commande `docker-compose up -d`.

Celle ci exécute le fichier docker-compose.yml, l'argument -d permet de le faire en arrière plan, afin que l'on ne perde pas la main sur le terminal.

A la suite de cela, nous aurons les différents composants de wazuh up.

Ils sont publiés sur les ports suivants : 1514, 1515.

```
root@projet-test:~# cd wazuh-docker/
root@projet-test:~/wazuh-docker# ls
CHANGELOG.md LICENSE README.md VERSION build-docker-images indexer-certs-creator multi-node single-node
root@projet-test:~/wazuh-docker# cd single-node/
root@projet-test:~/wazuh-docker/single-node# ls
README.md config docker-compose.yml generate-indexer-certs.yml
root@projet-test:~/wazuh-docker/single-node#
```

```
root@projet-test:~/wazuh-docker/single-node
GNU nano 6.2 docker-compose.yml
# Wazuh App Copyright (C) 2017, Wazuh Inc. (License GPLv2)
version: '3.7'

services:
  wazuh.manager:
    image: wazuh/wazuh-manager:4.4.5
    hostname: wazuh.manager
    restart: always
    ports:
      - "1514:1514"
      - "1515:1515"
      - "514:514/udp"
      - "55000:55000"
    environment:
      - INDEXER_URL=https://wazuh.indexer:9200
      - INDEXER_USERNAME=admin
      - INDEXER_PASSWORD=SecretPassword
      - FILEBEAT_SSL_VERIFICATION_MODE=full
      - SSL_CERTIFICATE_AUTHORITIES=/etc/ssl/root-ca.pem
      - SSL_CERTIFICATE=/etc/ssl/filebeat.pem
      - SSL_KEY=/etc/ssl/filebeat.key
      - API_USERNAME=wazuh-wui
      - API_PASSWORD=My33cr37P450r.-
    volumes:
      - wazuh_api_configuration:/var/ossec/api/configuration
      - wazuh_etc:/var/ossec/etc
      - wazuh_logs:/var/ossec/logs
      - wazuh_queue:/var/ossec/queue
      - wazuh_var_multigroups:/var/ossec/var/multigroups
      - wazuh_integrations:/var/ossec/integrations
      - wazuh_active_responses:/var/ossec/active-response/bin
      - wazuh_agentless:/var/ossec/agentless
      - wazuh_wodles:/var/ossec/wodles
      - filebeat_etc:/etc/filebeat
      - filebeat_var:/var/lib/filebeat
      - ./config/wazuh_indexer_ssl_certs/root-ca-manager.pem:/etc/ssl/root-ca.pem
      - ./config/wazuh_indexer_ssl_certs/wazuh.manager.pem:/etc/ssl/filebeat.pem
      - ./config/wazuh_indexer_ssl_certs/wazuh.manager-key.pem:/etc/ssl/filebeat.key
      - ./config/wazuh_clusters/wazuh_manager.conf:/wazuh-config-mount/etc/ossec.conf

  wazuh.indexer:
    image: wazuh/wazuh-indexer:4.4.5
    hostname: wazuh.indexer
    restart: always
    ports:
      - "5084:5084"
    environment:
      - "OPENSEARCH_JAVA_OPTS=-Xms512m -Xmx512m"
    ulimits:
      memlock:
        soft: -1
        hard: -1
      nofile:
        soft: 65536
        hard: 65536
    volumes:
      - wazuh-indexer-data:/var/lib/wazuh-indexer
      - ./config/wazuh_indexer_ssl_certs/root-ca.pem:/usr/share/wazuh-indexer/certs/root-ca.pem
      - ./config/wazuh_indexer_ssl_certs/wazuh.indexer-key.pem:/usr/share/wazuh-indexer/certs/wazuh.indexer.key
      - ./config/wazuh_indexer_ssl_certs/wazuh.indexer.pem:/usr/share/wazuh-indexer/certs/wazuh.indexer.pem
      - ./config/wazuh_indexer_ssl_certs/admin.pem:/usr/share/wazuh-indexer/certs/admin.pem
      - ./config/wazuh_indexer_ssl_certs/admin-key.pem:/usr/share/wazuh-indexer/certs/admin-key.pem
      - ./config/wazuh_indexer/wazuh.indexer.yml:/usr/share/wazuh-indexer/opensearch-security/internal_users.yml
      - ./config/wazuh_indexer/internal_users.yml:/usr/share/wazuh-indexer/opensearch-security/internal_users.yml

  wazuh.dashboard:
    image: wazuh/wazuh-dashboard:4.4.5
    hostname: wazuh.dashboard
    restart: always
    ports:
      - 443:5601
    environment:
      - INDEXER_USERNAME=admin
      - INDEXER_PASSWORD=SecretPassword
      - WAZUH_API_URL=https://wazuh.manager
      - DASHBOARD_USERNAME=kibanaserver
      - DASHBOARD_PASSWORD=kibanaserver
      - API_USERNAME=wazuh-wui
      - API_PASSWORD=My33cr37P450r.-
    volumes:
      - ./config/wazuh_indexer_ssl_certs/wazuh.dashboard.pem:/usr/share/wazuh-dashboard/certs/wazuh-dashboard.pem
      - ./config/wazuh_indexer_ssl_certs/wazuh.dashboard-key.pem:/usr/share/wazuh-dashboard/certs/wazuh-dashboard-key.pem
```

Nous avons maintenant accès aux différentes fonctionnalités de wazuh.

Nous pouvons maintenant ajouter un agent depuis le manager, ce qui nous permettra de surveiller un poste utilisateur ou bien un serveur critique.

Autre solution à déployer : **Cortex & Thehive.**



TheHive et Cortex sont des outils complémentaires utilisés dans la gestion et l'analyse des incidents de sécurité.

TheHive est une plateforme de gestion des incidents avec des fonctionnalités de suivi, de corrélation, d'analyse des observables et de collaboration, tandis que Cortex est une plateforme d'analyse automatisée et de réponse aux incidents qui peut être intégrée à TheHive pour enrichir les observables et automatiser certaines actions de réponse. Ensemble, ils offrent une solution complète pour la gestion efficace des incidents de sécurité.

Pour cette solution, nous utilisons également docker au travers de docker compose.

Un seul fichier docker compose nous permet d'installer dans un conteneur à la fois cortex et the hive, et dans un conteneur séparé Elasticsearch.



elasticsearch

Elasticsearch est un moteur de recherche et d'analyse de données open-source, basé sur Apache Lucene.

Je crée donc un dossier vierge que je nomme cortex.

Dans celui-ci je fais un fichier .env qui contient ceci :

- job_directory=/tmp/cortex-jobs
- ainsi qu'un fichier docker-compose.yml

root@projet-test: ~/cortex

GNU nano 6.2

version: '3.8'

services:

elasticsearch:

image: 'elasticsearch:7.11.1'

container_name: elasticsearch

restart: unless-stopped

ports:

- '0.0.0.0:9201:9201'

environment:

- http.host=0.0.0.0

- discovery.type=single-node

- cluster.name=hive

- script.allowed_types= inline

- thread_pool.search.queue_size=100000

- thread_pool.write.queue_size=10000

- gateway.recover_after_nodes=1

- xpack.security.enabled=false

- bootstrap.memory_lock=true

- 'ES_JAVA_OPTS=-Xms256m -Xmx256m'

ulimits:

nofile:

soft: 65536

hard: 65536

volumes:

- ./elasticsearch_data:/usr/share/elasticsearch/data

- ./elasticsearch_logs:/usr/share/elasticsearch/logs

cortex:

image: 'thehiveproject/cortex:3.1.1-1'

container_name: cortex

restart: unless-stopped

volumes:

- '/var/run/docker.sock:/var/run/docker.sock'

- '\${job_directory}:\${job_directory}'

depends_on:

- elasticsearch

ports:

- '0.0.0.0:9001:9001'

thehive:

image: 'thehiveproject/thehive:3.5.1-1'

container_name: thehive

restart: unless-stopped

depends_on:

- elasticsearch

- cortex

ports:

- '0.0.0.0:9000:9000'

command: '--cortex-port 9001'

Nous avons donc 3 services installés à cet endroit :

```
root@projet-test:~# cd cortex/  
root@projet-test:~/cortex# ls  
docker-compose.yml  elasticsearch_data  elasticsearch_logs
```

elasticsearch sur le port 9201, Cortex sur le port 9001, The Hive 9000

Nous pouvons ainsi sécuriser les postes et serveurs en concordance avec Wazuh.

Grâce à cela nous pouvons détecter les différentes attaques/fichiers malveillants et virus sur les systèmes informatiques déployés.

Nous allons voir ensuite comment nous faisons remonter tout cela avec Prometheus.

Prometheus :



Prometheus est un système open-source de monitoring et d'alerte conçu pour surveiller les performances et l'état des applications et des infrastructures.

Il utilise son propre langage de requêtes PROMQL (Prometheus Query Language).

Nous utiliserons Docker une nouvelle fois.

Cette fois l'on crée un Dockerfile

```
root@projet-test: ~/prometheus
GNU nano 6.2
FROM prom/prometheus
ADD prometheus.yml /etc/prometheus/
```

puis l'on build l'image avec docker build
docker build -t my-prometheus .
docker run -p 9090:9090 my-prometheus
et l'on run l'image récemment buildé.

On se retrouve donc avec un dashboard prometheus ouvert sur le port 9090.

On pourra y retrouver toutes les alertes des différents serveur déployés, ce qui permet de veiller au bon fonctionnement de ceux-ci, indispensable à la bonne performance d'une infrastructure.

Il fonctionne de pair avec **Grafana**.



Grafana est une plateforme open-source de visualisation et de monitoring des données.

Il peut être connecté à un grand nombre de sources différentes, afin de récupérer le plus de données possibles.

Il est compatible avec nombres de plugins.

Pour la sauvegarde de notre solution, nous utilisons **Azure Backup Center**.



Microsoft Azure Backup

Azure Backup Center est une solution de gestion centralisée des sauvegardes dans Microsoft Azure.

Ce service Azure permet une gestion centralisée des sauvegardes.

Celui-ci étant directement intégré à Azure, il nous permet de facilement et efficacement sauvegarder nos données.

Pour bénéficier de ce service, nous devons le configurer sur notre compte (Azure étudiant) on choisit le groupe de ressource déjà existant, car il est inutile d'en recréer un, cela facilitera la sauvegarde de l'existant.

On choisit la région East US, car il s'agit d'une région différente de celle utilisée par nos machines virtuelles. Cela assure une redondance, en cas de défaillance dans la région France Central.

Traefik

Pour la dernière partie nous verrons Traefik.



Il s'agit d'un proxy/loadbalancer.

Nous l'utilisons car il est très utile dans notre cas, car nous utilisons des conteneurs **docker**. Il reçoit les requêtes internet et les redirige vers les services spécifiques.

```
version: '3'

services:
  reverse-proxy:
    # The official v2 Traefik docker image
    image: traefik:v2.10
    # Enables the web UI and tells Traefik to listen to docker
    command: --api.insecure=true --providers.docker
    ports:
      # The HTTP port
      - "80:80"
      # The Web UI (enabled by --api.insecure=true)
      - "8080:8080"
    volumes:
      # So that Traefik can listen to the Docker events
      - /var/run/docker.sock:/var/run/docker.sock
```

Nous utilisons encore une fois Docker.

Ce compte rendu vous permet de voir la mise en oeuvre de la solution choisie par nous pour répondre au problème énoncé.

Nos choix ont été guidés par nos expériences et recherches pour apporter de la performance et de la sécurité à nos solutions.