

The background features a dark navy blue field with abstract geometric shapes. On the left, there are overlapping triangles in a vibrant blue and a lighter, semi-transparent blue. Diagonal bands of a slightly lighter navy blue run across the bottom left and top right corners.

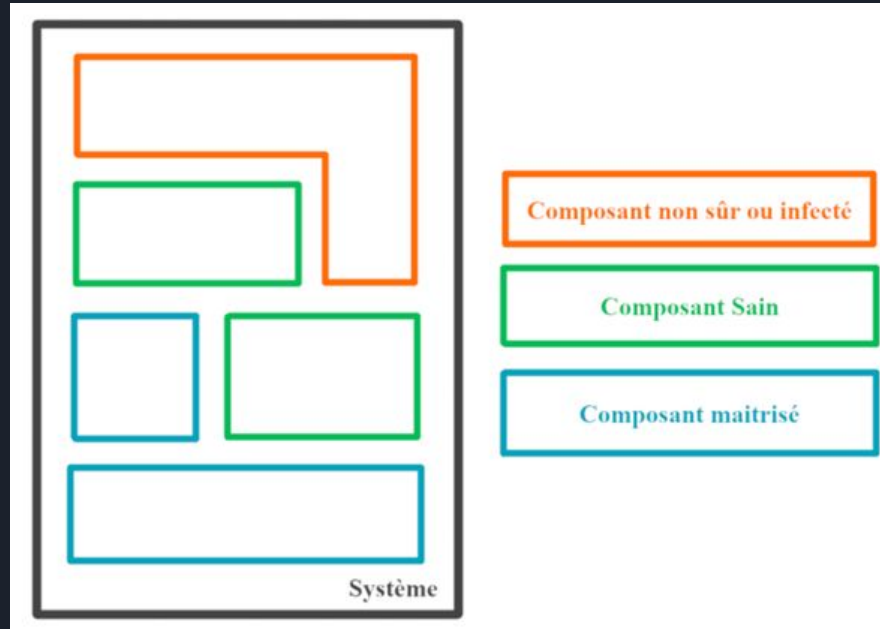
Conteneurisation



Le système standard

Au sein d'un système unique, un certain nombre de composants sont amenés à travailler en parallèle. Ceux-ci peuvent être maîtrisés (développés en interne), sains (provenance d'une source de confiance) ou non sûrs (ce qui implique qu'il existe une possibilité que celui-ci soit malveillant ou infecté).

Par système, on entend toute interaction entre des composants d'une même architecture, ces interactions peuvent subvenir sur une même machine ou entre plusieurs par le réseau (standard dans une architecture logicielle dite 3-tiers).



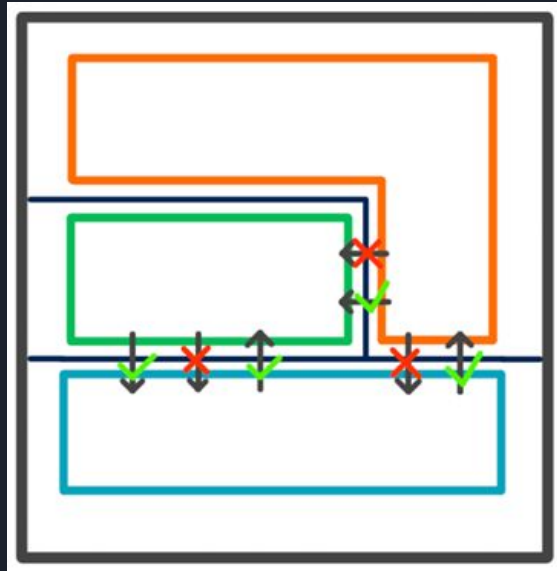


Quels sont les problématiques

- **La portabilité** : il est nécessaire de fournir en plus de la livraison logicielle, une liste très précise de binaires et de librairies avec des versions exactes pour garantir que l'environnement de production sera équivalent à celui de développement.
- **L'adhérence système** : sur un système exploité par plusieurs composants distincts, il est compliqué voire impossible d'identifier les librairies et configurations réellement nécessaires à la mise en production d'un composant.
- **La sécurité** : il nous faut une surface d'attaque et une surface de friction

Le système conteneurisé

Conteneuriser les composant a pour objectif de créer des murs entre les composants afin de les cloisonner et ne permettre que les comportements et interaction autorisés par chaque composant.



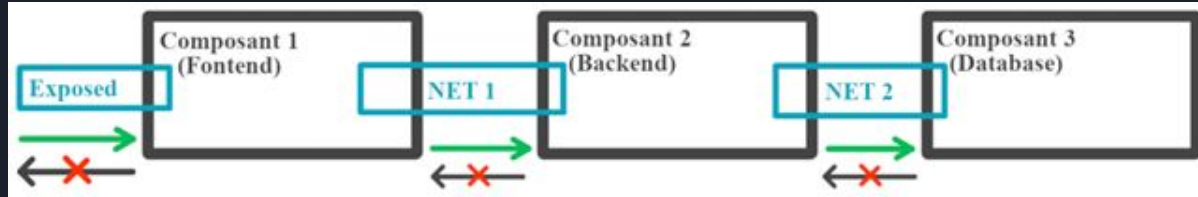


Les principe du kernel linux

Dans la pratique, le principe de conteneurisation se base sur des principes basiques de namespace du kernel Linux. Les ressources cloisonnées sont les suivantes :

- PID : Le conteneur n'accède qu'à son propre processus et n'a donc pas de vision sur les autres processus existants sur la machine ou dans les autres conteneurs.
- MNT : Le conteneur n'accède qu'à ses propres points de montages
- NET : Le conteneur n'accède qu'à ses propres interfaces réseau
- IPC : Le conteneur possède des canaux IPC (InterProcess Communication) dédiés.
- UTS : Le conteneur possède son propre hostname et son propre nom de domaine NIS.

Il existe cependant des moyens, comme option de démarrage d'un conteneur, de partager un ou plusieurs namespaces avec l'hôte ou un autre conteneur.



C'est notamment quelque chose qui est utilisé pour instaurer un principe 3-tiers sur une seule machine hôte (partage d'espaces réseau).

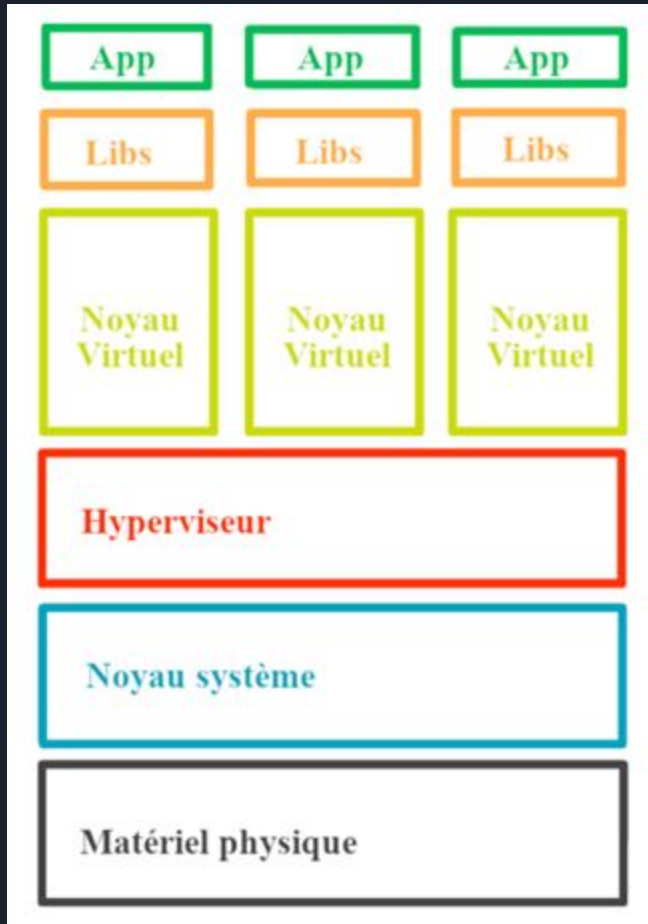


La conteneurisation standard

La solution de base mise en œuvre pour faire face aux problématiques citées est la suivante : la virtualisation.

Cette solution permet bien de supprimer les problématiques de sécurité liées au partage de ressources mais en amène de nouvelles :

- Consommation de ressource excessive ;
- Aucune flexibilité dans le déploiement ;
- Multiplication des besoins d'administration ;
- Création d'hétérogénéité dans les machines finales de l'infrastructure ;
- Création d'instabilité dans les déploiements ;
- Conservation des problématiques d'adhérence au système.



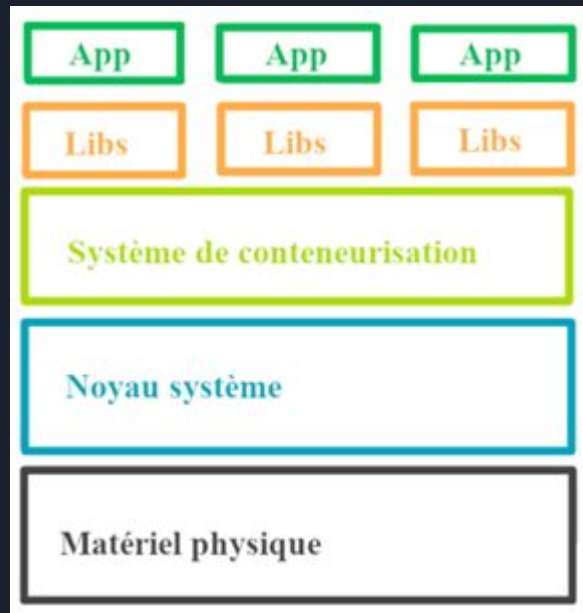
Ici, un exemple de structure avec des machines virtuelles.

Pour compléter et répondre à ces nouvelles problématiques, la conteneurisation fonctionne sur le principe de partage du noyau hôte.

Il est donc possible d'allumer et d'éteindre quasi instantanément un environnement applicatif étant donné que la base de fonctionnement (le noyau) est déjà allumée.

Le système historique de conteneurisation sur Linux est LXC (Linux Containers).

Le système le plus largement déployé à ce jour est Docker. Docker est initialement une surcouche à LXC. Le concept reste le même que décrit précédemment, chaque conteneur est créé manuellement et les liaisons inter-conteneurs également.





Orchestration de conteneur

Une fois que les problématiques de compatibilité avec les environnements de production sont levées, il est question des sujets d'automatisation. En effet, le fait de créer l'ensemble des briques nécessaires à un applicatif (front, back, db, networks, mnt, ...) manuellement est sujet à erreurs pouvant tant entamer le côté opérationnel que la sécurité du système.

L'orchestrateur de premier niveau sous Docker est Compose. L'objectif de Compose est de, sur la base d'un fichier de description, automatiser la création des éléments décrits (Les interactions réseau, les images à déployer, les points de montage, les ports exposés, ...).



```
version: "3.9"
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - ./code
    environment:
      FLASK_ENV: development
  redis:
    image: "redis:alpine"
```

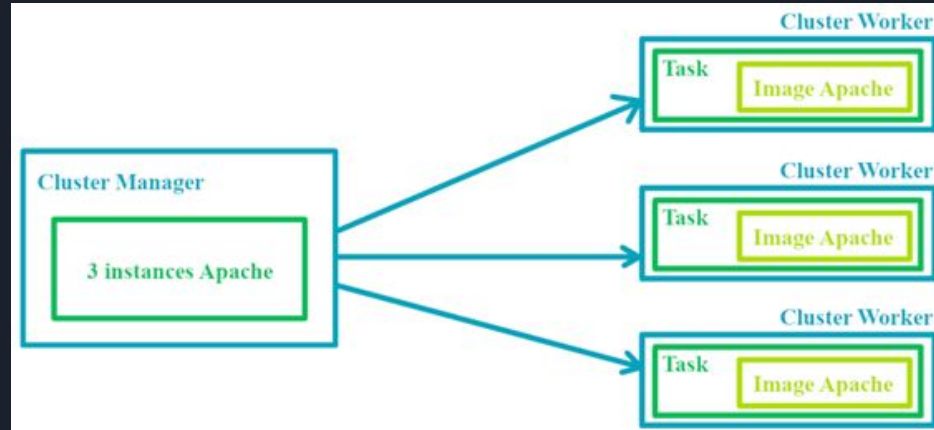
L'objectif de Compose est de fournir un fichier livrable capable de construire les images et les lancer en s'interfaçant directement avec les commandes standards de Docker.


L'objectif étant la capacité de déployer des solutions complètes ou chaque composant est conforme à ce qui est prévu pour son implémentation.

Orchestration de clusters

Maintenant que tout se déploie proprement et automatiquement, il nous faut trouver une solution pour gérer automatiquement le dimensionnement et la mise à l'échelle de nos composants.

Pour ce faire, nous allons nous tourner vers un orchestrateur de clusters. L'idée général de ce type d'orchestration est d'avoir un manager et n workers effectuant une tâche identifiée.





Cette tâche se résume par le lancement d'une image docker par un worker. Le manager choisit le nombre d'instance en fonction de la charge actuelle. Chacune des images instanciées dans le cluster est donc considérée comme étant jetable.

Les deux plus gros systèmes d'orchestration sont Docker Swarm et Kubernetes.



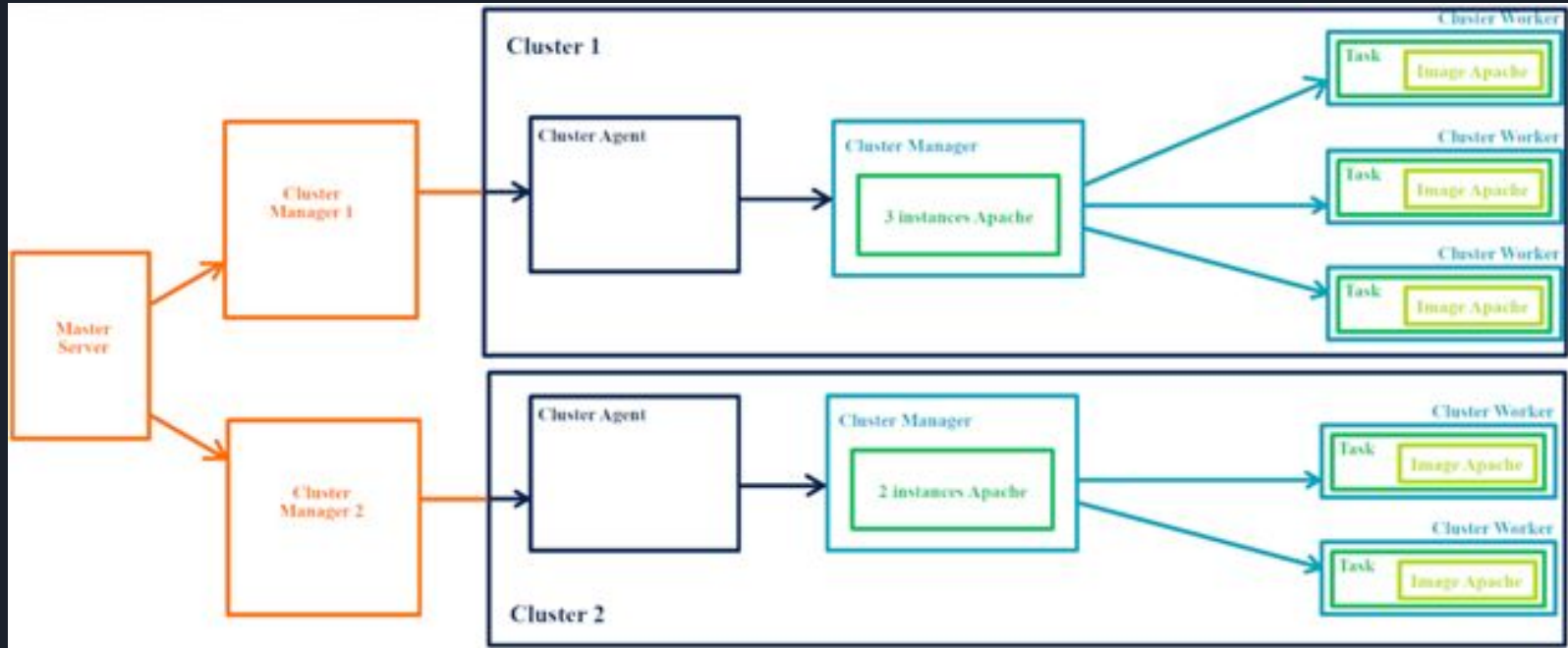
kubernetes




Orchestration d'orchestrateurs

Au-dessus des orchestrateurs de clusters, nous allons trouver des orchestrateurs d'orchestrateurs de clusters. L'idée finale étant de manager des clusters à très grande échelle depuis un point central.

Étant donné la complexité de l'architecture et la distance avec les systèmes, l'utilisation de ce type de solution implique le besoin de gérer une très grande quantité de conteneurs jetables dans une multitude d'environnements (on-prem, cloud1, cloud2, cloud3).





Les solutions de ce type sont très hétérogènes et ne sont pas vraiment équivalentes étant donné que la finalité de ces outils est plus de gérer des workflows.

Par workflow on entend les processus automatiques d'une chaîne d'intégration et/ou de livraison continue.

Les solutions adaptées à ça sont généralement Rancher, APCERA, Docker Cloud, ...

La quasi-totalité des fournisseurs de cloud / matériel offrent une solution complémentaire (management de cluster) sur laquelle viennent s'interfacer les solutions au-dessus (AKS, Karbon, ...).





Besoins et solutions

Le premier niveau (conteneurisation simple => **Docker**) est le moyen le plus adapté pour opérer des tests unitaires dans un cycle de développement étant donné que cela permet d'opérer chaque test dans un nouvel environnement qui sera équivalent à la production.

Le deuxième niveau (conteneurisation orchestrée => **Compose**) est le moyen le plus adapté pour la mise en production. Elle permet de gérer l'intégralité des composants applicatifs en parallèle et de conserver le bon niveau de ségrégation et de sécurité.

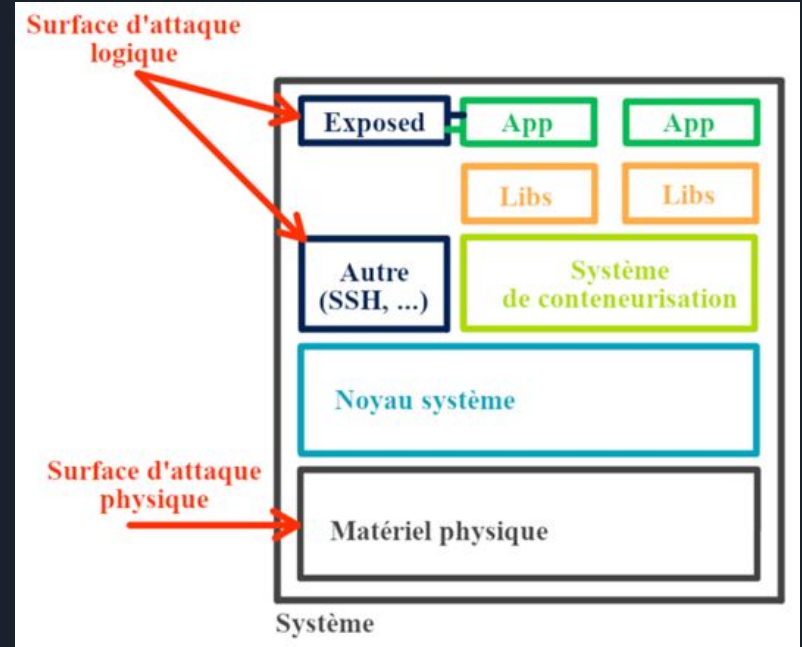
Le troisième niveau (conteneurisation en clusters => **Swarm, Kubernetes**) est adapté aux besoins de gestion de charge très complexe et de mise à jour de parc applicatif en continu. Ce niveau n'a de sens que pour les micro-services (ex : une recherche google) et pas aux services continus (ex : un openldap).


Le quatrième niveau (orchestration de conteneurisation en clusters => **Rancher**) est un complément au troisième niveau pour la gestion à très grande échelle dans de multiples environnements de micro-services applicatifs.

Surface d'attaque

Comme pour tout environnement, les prérequis de sécurité dépendent du nombre de couches entre la surface exposée et le noyau du système hôte.


Dans un système s'arrêtant au niveau 1 ou 2 de conteneurisation, la surface d'attaque s'arrête à la porte de la machine hôte.





Nous trouverons une surface d'attaque logique (par le réseau) et une surface d'attaque physique (accès physique au matériel). Ce document traitant uniquement de la partie conteneurisation, la partie surface d'attaque physique sera volontairement mise de côté.

Pour ce qui est de la partie logique, nous allons dans un premier temps trouver un service nécessaire à l'administration du système. Le durcissement de celui-ci fait partie des pratiques de base de durcissement d'un système.

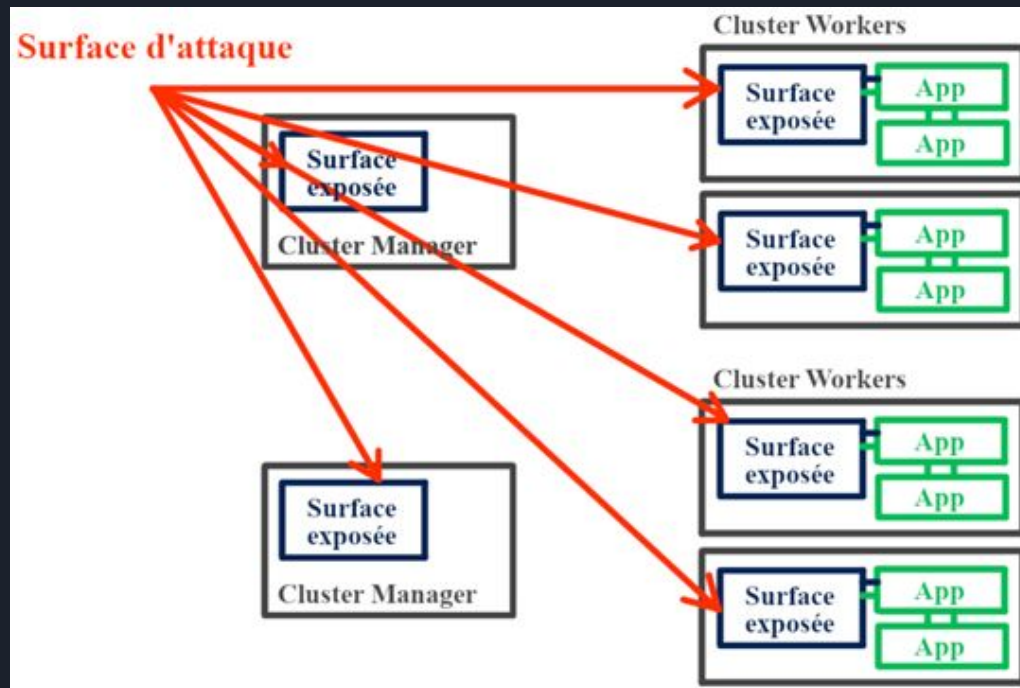



Cela implique comme prérequis que les bonnes pratiques de gestion des conteneurs soient respectées. Un grand nombre de documentation de référence s'applique pour cela (ANSSI, CIS, ...) :

Par exemple pour l'ANSSI :

- [linux_configuration-fr-v1.2.pdf \(ssi.gouv.fr\)](#)
- [guide_cloisonnement_systeme_anssi_pg_040_v1.pdf](#)
- [docker_fiche_technique.pdf \(ssi.gouv.fr\)](#)

Pour un système en clusters (conteneurisation de niveau 3 et 4), les problématiques de sécurité pour les workers se trouvent multipliées par la taille de l'infrastructure :





A ces besoins de durcissement, viennent s'ajouter les nouveaux besoins de sécurité (le système de cluster étant un système supplémentaire à manager).

Sur ces points, il n'existe pas de documentation en provenance de l'ANSSI, seuls sont traités les cas généraux de conteneurisation (niveau 1 et 2). Il existe cependant des benchmarks pour Kubernetes ainsi que des bonnes pratiques d'utilisation et de sécurisation proposées par les éditeurs.



Conflits d'administration

Les outils d'orchestration étant généralement à la main des opérationnels projet pour des raisons de maniabilité des solutions, cela implique qu'ils administrent en partie (et généralement pleinement) les systèmes associés (réseau, système, ...).

Ce phénomène implique que les règles de cloisonnement liés à la sécurisation, notamment réseau sont à la main complète d'opérationnel non formés et non avertis des sujets de sécurité sous-jacents.

Plus le niveau de conteneurisation est grand, plus le nombre de problématiques opérationnelles de sécurité grandis et donc les coûts de sécurisation et de validation de processus également (formation, ralentissement des modèles de développement / déploiement, audit, ...).



Outillage de sécurité

Les outils standard permettant d'opérer la sécurité sur une machine (antivirus, sonde,...) ne sont capables de s'interfacer qu'avec des systèmes hôte et pas à des conteneurs.

Chaque conteneur possédant sa propre racine, les outils de sécurité deviennent inefficaces (le conteneur étant la partie vulnérable et potentiellement compromise).

De plus, pour les cas de clusters hébergés (AKS, ...), les outils étant inexistants, la seule capacité de sécurité est le fait de s'en remettre à la conscience de sécurité des opérationnels projets de suivre les bonnes pratiques d'utilisation sécurisée dictées par le fournisseur (celles-ci sont sans grande surprise très rarement suivies).

La seule option restante est de faire de l'étude et de la corrélation sur l'ensemble des journaux. Cette option nécessite la présence d'un SoC (Security Operations center).